Efficient Signed Arithmetic Multiplication on Memristor-based Crossbar

Zahedi, Mahdi; Shahroodi, Taha; Wong, Stephan; Hamdioui, Said

DOI
10.1109/ACCESS.2023.3263259

Publication date
2023

Document Version
Final published version

Published in
IEEE Access

Citation (APA)
Zahedi, M., Shahroodi, T., Wong, S., & Hamdioui, S. (2023). Efficient Signed Arithmetic Multiplication on Memristor-based Crossbar. *IEEE Access*, *11*, 33964-33978.
https://doi.org/10.1109/ACCESS.2023.3263259

IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 27 February 2023, accepted 25 March 2023, date of publication 29 March 2023, date of current version 7 April 2023.
Digital Object Identifier 10.1109/ACCESS.2023.3263259

RESEARCH ARTICLE

Efficient Signed Arithmetic Multiplication on Memristor-Based Crossbar

MAHDI ZAHEDI, TAHA SHAHROODI, STEPHAN WONG, (Senior Member, IEEE), AND SAID HAMDIOUI, (Senior Member, IEEE)
QCE Department, EEMCS Faculty, Delft University of Technology (TU Delft), 2628 CD Delft, The Netherlands
Corresponding author: Mahdi Zahedi (M.Z.Zahedi@tudelft.nl)

ABSTRACT The vast potential of memristor-based computation-in-memory (CIM) engines has mainly triggered the mapping of best-suited applications. Nevertheless, with additional support, existing applications can also benefit from CIM. In particular, this paper proposes an energy and area-efficient CIM-based methodology to perform arithmetic signed matrix multiplications. Our approach combines a) the mapping of the signed operands on the 1T1R crossbar, and b) the augmentation of the periphery with customized circuits to support the execution of shift and accumulate needed for the arithmetic operations. The operand mapping is performed without the need for sign extension; hence, reducing the required memory size. To demonstrate the superiority of our scheme as compared with the state-of-the-art, simulations are performed for different case studies including a neural network and two kernels which are taken from the Polybench/C benchmark suite. The results show that our approach achieves up to 8× energy-saving and 3× area-saving compared with other CIM-based prior works.

INDEX TERMS Memristor, computation-in-memory, signed computation.

I. INTRODUCTION

Matrix arithmetic operation is key in many applications such as image, graph, and language processing [1], [2], [3], [4], [5], [6]. It is the killer in terms of energy efficiency. This is mainly due to the inherent separation of data and processing in state-of-the-art computer architectures [7]. This separation reduces the energy efficiency as transmission to/from DRAM consumes several orders of magnitude higher energy than a single operation within a processor [8], [9], [10], [11]. This is further exacerbated when the amount of data explodes. As an alternative to traditional computing, designing and developing accelerators for matrix operations based on the notion of Computation-in-Memory (CIM) [12] and making use of emerging technologies, memristor devices [13], [14], [15], [16], [17], have attracted great attention. From the application perspective, mapping the operands to the memory unit, comprising emerging devices and their peripheries, is a critical step toward enabling energy-efficient CIM; it includes choosing the appropriate datatype structure and supporting signed numbers. These applications are often operating on signed numbers and require different data type sizes. Hence, without the support of signed numbers, the promise of CIM for these applications is greatly diminished. Therefore, it is essential for researchers to enable this feature for CIM-based design. Since memristor devices store the information as different (positive) conductance levels, additional considerations are required when the computation has to be performed on signed numbers. This enforces more complexity in the way the data is mapped to the crossbar as well as the way the computation should be performed. Therefore, a simple, yet energy-efficient, solutions are needed to perform operations on signed numbers.

Recent work has presented a few mapping solutions to be able to support data types and signed numbers on memristor-based crossbars. The two's (four's) complement representation is employed in [18]; however, a large overhead in terms of energy, performance, and area efficiency is imposed in this representation due to the *sign extension*. Considering that, ISAAC [19] biases the data intended to be programmed into the crossbar to bring them into the positive range while keeping the crossbar input as two's complement datatype.

The associate editor coordinating the review of this manuscript and approving it for publication was Baker Mohammad.

This requires keeping track of the number of elements that contribute to the result to be able to unbiased the output afterward. Hence, extra pre-and post-processing has to be considered. PRIME [20] and PipeLayer [21] map positive and negative weights into different crossbars, which requires incorporating more crossbars. Finally, storing the weights as differential conductance between memristor devices either located in one crossbar or in two crossbars has been suggested in [22] and [23]. Similarly, this approach has to employ more crossbar arrays, which in consequence, imposes more energy and area overhead. In addition, these works did not provide any detailed implementation required for extra processing in the periphery to add up intermediate values produced by the crossbar for Matrix-Matrix Multiplication (MMM). As the matrix and operand size incorporated in this operation increase, more overhead is imposed into the digital periphery due to the larger adders and more shift operators [24]. This implies the necessity of optimization in the periphery. To summarize, signed computation, compared to the unsigned computation, requires more complex weight *mapping*, consumes more area and memristor devices, and demands more *extra processing* in the periphery, which induces more energy consumption and latency to the system.

This work advances the state-of-the-art by proposing a novel mapping solution to support signed and unsigned MMM based on widely used two's complement representation. In this method, signed and unsigned operations are performed with minimum energy, latency, and area overhead by *eliminating* the costly sign extension. The proposed periphery architecture utilizes minimum-sized adders customized based on technology-driven restrictions. In short, this paper presents the following main contributions:

1) *A novel scheme to support widely used two's complement arithmetic operations*: This is accomplished by exploiting the behavior of the memristor crossbar without the costly sign extension. This eventuates to efficient support of signed integer/fix-point computations in terms of energy, latency, and area. The flexibility of the design allows applications to dynamically switch between signed and unsigned computation without changing the hardware or mapping of data.
2) *An energy and area efficient digital periphery architecture for MMM*: The periphery fulfills additional processing required for MMM using minimum-sized adders and registers. The proposed solution allows applications to dynamically change the data size without changing the hardware or the mapping scheme of data to the crossbar.
3) *Evaluation of the proposed solution using two memristor technologies (RRAM and PCM) and different benchmarks*: The design is validated and evaluated in terms of energy (for both computation and programming phases), execution time, and area while considering RRAM and PCM technologies. Compared to the baselines, we achieved up to 8× energy improvement and 3× area-saving.

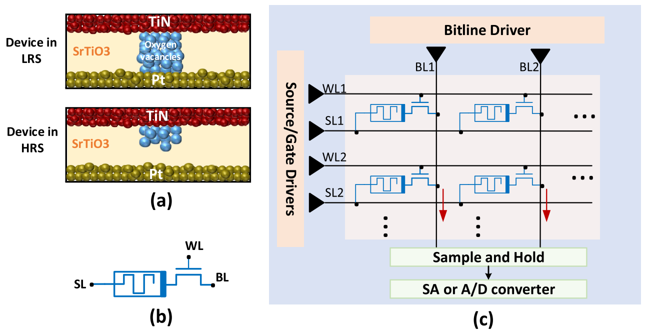


FIGURE 1. **(a) ReRAM memristor device behavior (b) 1T1R memristor cell (c) CIM tile encompassing crossbar and peripheries.**

The paper is organized as follows. Section II introduces the background of the memristor device. Section III explains the existing solutions for signed arithmetic multiplication on crossbars. Section IV proposes an efficient periphery architecture for *unsigned* arithmetic computation. Then, Section V explains the solution to flexibly and efficiently support *signed* and unsigned computations at the same time based on what is already proposed in Section IV. Finally, Section VI evaluates the design while Section VIII concludes the paper.

II. BACKGROUND

In this section, we briefly discuss the behavior of memristor devices, how and mainly which operations are supported by placing them in the crossbar structure, and the challenge arising on arithmetic computation due to the limited memristor resistance levels.

Contrary to charge-based memories, memristor devices are non-volatile, where the resistance levels represent data. The switching behavior of a bipolar memristor device is depicted in Figure 1(a). The memristor devices can alternate between resistance levels by the application of suitable voltage or current pulses during a writing or programming operation. Usually, the term SET is used when the transition occurs from Low Resistance Level (LRS) to High Resistance Level (HRS) while the term RESET is used for the opposite direction. In bipolar memristor devices, the device is set and reset by changing the polarity of the programming voltage (e.g., 2 V) [25]. However, for unipolar memristor devices, SET and RESET occur for the same voltage polarity but at different amplitudes [26]. The resistance levels can be interpreted as logic '0' and '1'. In order to read the device without disturbance, a small voltage (e.g., 0.2 V) (current) should be applied, and the current (voltage) through (across) the device should be sensed. Figure 1(b) shows a schematic representation of the 1T1R memristor-based structure. This is a fundamental block for constructing a CIM tile. Figure 1(c) portrays a CIM tile comprising a memristor crossbar array and peripheries where three drivers are employed to drive Word-Lines (WL), Source-Lines (SL), and Bit-Lines (BL). In the case of read/computational operations, the current or voltage is sampled and then converted to the digital

TABLE 1. List of some potential applications/algorithms/kernels that can be executed using memristor-based CIM [27].

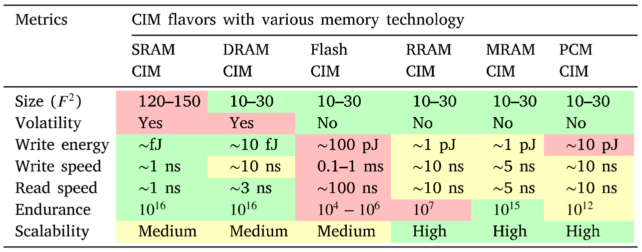
Domain		Applications/Algorithms/Kernels				
Network		Automata processor for network security [28]	Packet classification [29]	SAMCRA [30]	IP-routing	-
	operation	AND	Matching	VMM	Matching	-
Security		Security primitives (PUF-RNG) [31]	Encrypted communication [32]	AES [33]	Regular expression matching [34]	Salsa20/ChaCha20
	operation	Analogue properties	Analogue properties	Add-XOR-Multiply	Matching	Addition - XOR
Approximate computing		Image property calculation [35]	Approximate matrix multiplication	Quantum simulations [36]	-	-
	operation	Addition - Subtraction	VMM	VMM	-	-
Mathematics		Partial differential equation solver [37]	Eigenvector calculation [38]	Markov chain	Vector-cosine similarity	-
	operation	VMM	VMM	VMM	VMM	-
Signal and image processing		Compressed sensing [39]	Discrete Fourier Transform [40]	Convolutional Image filtering [41], [42]	Huffman encoding	GLCM feature extraction
	operation	VMM	VMM	VMM	Matching	Addition
Classification and Prediction		Recurrent neural network [43]	Convolutional neural network [19]	Hyperdimensional computing [44]	Reservoir computing	Auto-regressive models
	operation	VMM	VMM	AND - OR - VMM	Matching	VMM
Database		Query-06 of the TPC-H [45]	Pattern matching in databases [4]	Transitive closure	Bitmap indices	BitWeaving
	operation	Bitwise XOR	Matching	VMM	OR - AND - XOR	OR - AND - XOR
Bioinformatics		DNA sequencing or allignment [46]	Genome Base-calling [47]	Genome Profiling [48]	ECG Classification [49]	-
	operation	XOR - VMM	VMM	VMM	VMM	-
Graph processing		Graph Clustering [50]	Breadth first search [2]	VLSI routing	PageRank	All-pair-shortest-path
	operation	VMM - NOR - HD	VMM	VMM	VMM	VMM

domain by using a Sense Amplifier (SA) or Analog-to-Digital Converter (ADC).

Figure 2 compares three leading memristor technologies (RRAM, MRAM, and PCM) with other conventional memories. Memristors have great scalability, high density, near-zero standby power, and non-volatility. Besides, memristor-based analog computing can lead to high energy efficiency as it inherently allows for massive inter- and intra-crossbar parallelization. Examples of computational operations that can be executed on the crossbar are *addition* [50], *logical operations* [51], [52], and *MMM* [53], [54]. For the first two, operands are stored in the crossbar as vectors. For the latter, one operand (multiplier) is fed to the crossbar as input while the second operand (multiplicand) is stored inside the crossbar.Based on these supported operations, Table 1 provides a comprehensive list of potential applications that can be accelerated using memristor devices. Memristor devices can be typically programmed into two or a few resistance levels due to the technological constraints stemming from the device (e.g., variation) and peripheral circuits (e.g., ADC resolution). Therefore, with the provision that the datatype size required by an application is larger than what a single memristor cell can present, the data has to be distributed over several devices. Similarly, a limited resolution is supported by the input drivers of the crossbar too. Hence, special care on the execution flow of those operations has to be taken into account to produce a meaningful result.

III. RELATED WORK

As a primary step to enable CIM for many applications, we need to support signed representation for both (1) the data programmed into the crossbar –***array data***– and (2) the data provided as input to the crossbar –***input data***. However, to the best of our knowledge, there are only a few works that partially look into this problem and propose some solutions. In the following, we discuss the existing solutions on how they represent signed numbers for both *array data* and *input data*.



Metrics	CIM flavors with various memory technology					
	SRAM CIM	DRAM CIM	Flash CIM	RRAM CIM	MRAM CIM	PCM CIM
Size (F^2)	120–150	10–30	10–30	10–30	10–30	10–30
Volatility	Yes	Yes	No	No	No	No
Write energy	~fJ	~10 fJ	~100 pJ	~1 pJ	~1 pJ	~10 pJ
Write speed	~1 ns	~10 ns	0.1–1 ms	~10 ns	~5 ns	~10 ns
Read speed	~1 ns	~3 ns	~100 ns	~10 ns	~5 ns	~10 ns
Endurance	10^{16}	10^{16}	$10^4 - 10^6$	10^7	10^{15}	10^{12}
Scalability	Medium	Medium	Medium	High	High	High

FIGURE 2. Comparison of different memory technology [55].

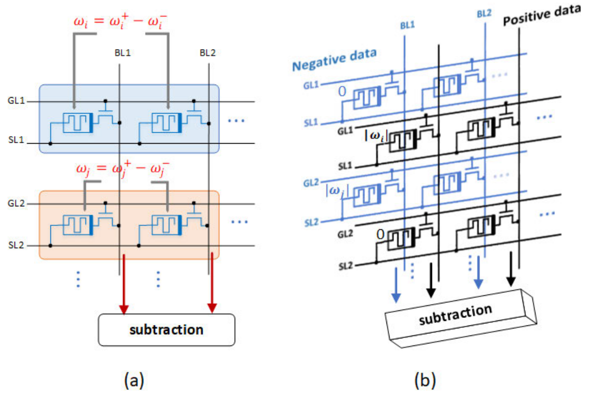


FIGURE 3. Supporting signed array data by (a) subtraction of cells' value (b) mapping negative and positive values to different crossbars.

A. PRIOR WORK ON SIGNED NUMBER REPRESENTATION OF ARRAY DATA

In order to support negative numbers, ISAAC [19] brings the array data into a positive range by using a bias value.

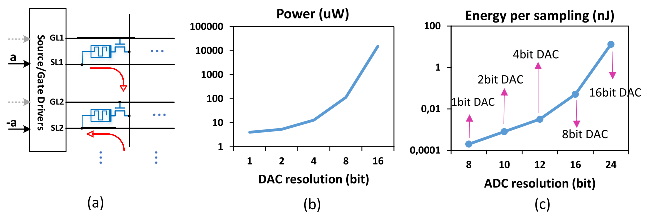


FIGURE 4. **(a) Bi-directional input current to support signed input data and its implication on power/energy consumption of (b) DACs and (c) ADCs in different resolutions (assuming 256 × 256 crossbar with 1-bit per cell).**

This imposes two types of extra processing. First, additional pre-processing if the data has to be programmed to the crossbar on the fly to bring them into the positive range. Second, mandatory post-processing to subtract the bias value several times equal to the number of array data that are added up. This has to be performed for every output element. Another approach represents a number as a difference of two or more cells [22], [23]. Figure 3(a) illustrates this approach assuming that the data is presented as a subtraction of two numbers, each residing in a single cell. PRIME [20] and PipeLayer [21] map positive and negative numbers into different crossbars (as unsigned numbers). Figure 3(b) depicts how a matrix with positive and negative numbers is mapped to two crossbars. In both cases, the result is obtained based on subtraction. Besides having more cells to represent a number, which leads to more energy/area consumption, more complexity is imposed on mapping data to the crossbars, especially when it is required to incorporate more cells for large datatype sizes. Another approach is using two's complement representation [18]. However, due to the necessity of sign extension, a big overhead is imposed in terms of area and energy (more details are in Section V). Finally, FloatPIM [56] uses the IEEE standard floating-point to store data as a floating-point number in the crossbar. Using a sign bit to indicate the sign of a number, as well as the way of computation in the crossbar [57], limits this approach to perform multiplication and addition between only two numbers. This is inherently different than what was depicted in Figure 1(c) to fulfill vector-matrix-multiplication (and, in turn, MMM) at once by exploiting Kirchhoff and Ohm's laws.

B. PRIOR WORK ON SIGNED NUMBER REPRESENTATION OF INPUT DATA

A few works have studied how the input data to the crossbar can be represented. ISAAC [19] provides 16-bit data to the crossbar in 16 cycles (one bit per cycle) in 2's complement format. As mentioned for array data, a big overhead is imposed due to the sign extension. Another approach is to reverse the direction of the current for negative input as depicted in Figure 4(a); this is only employed for subtract operation in [18]. Considering this approach, the analog input voltage levels that have to be supported by the Digital-to-Analog Converters (DACs) should be doubled. Consequently, more voltage/current levels should be distinguished by ADCs. Therefore, not only the system becomes more sensitive and error-prone, but also its power consumption/energy increase exponentially as the resolution of DACs and ADCs increase. This is shown in Figure 4(b) and 4(c), presenting the power and energy per sample for a DAC and an ADC, respectively, by increasing the resolution [58], [59]. It should be noted that the minimum required ADC resolution is based on the DAC resolution and the number of array data contributing to the analog addition (in this case, 256). As an example, if we have 2-bit DAC, the ADC resolution is calculated as $log2(256) + 2 = 10$ bits.

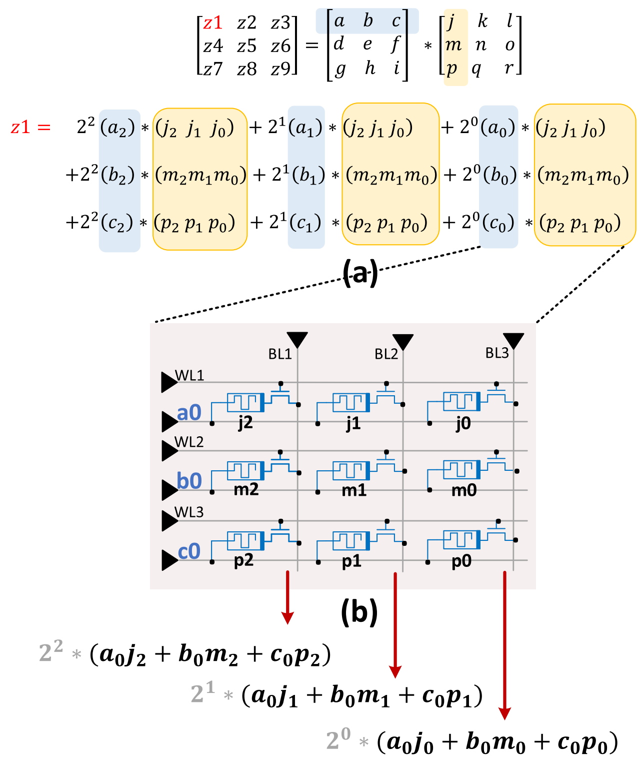
In conclusion, considering the limitations and overhead of existing solutions to support sign arithmetic computation in the crossbar, it is essential to reduce this overhead with an efficient crossbar periphery architecture. Any architectural solution for the digital periphery should take into account the following restrictions as well: 1) ADC resolution; this limits the maximum number of crossbar rows to be activated. 2) Limited resistance levels on a memristor device; if a number cannot be represented by these few levels in a single device, it has to be distributed over several devices. 3) Input drivers (DACs) resolution; due to the limitation on the number of voltage/current levels provided by the drivers to the crossbar, input data has to be split and fed to the crossbar in several steps. Considering the aforementioned limitations, more processing steps have to be performed on the output of the crossbar to achieve the final result for arithmetic operations.

IV. EFFICIENT ARITHMETIC COMPUTATION IN CIM TILE FOR UNSIGNED DATA REPRESENTATION

In this section, we propose an efficient structure for the digital periphery required next to a memristor crossbar to execute a complete unsigned addition and MMM operations. We first focus on unsigned data representation and computation. The presented structure forms the basis for supporting signed computations, which we will describe in Section V.

Our proposed architecture consists of three stages: 1) Analog Addition; this stage addresses the first limitation mentioned before (ADC resolution). 2) Sliding over multiple columns; the hardware designed for this stage addresses the second limitation (limited memristor resistance levels) 3) Sliding over input segments; this stage takes into account the third limitation (DACs resolution). These three stages are executed consecutively for several iterations in order to produce a final result for arithmetic operations. We explain each stage in the following subsections. The proposed design utilizes minimum size adders to lessen the energy and latency overhead of periphery circuits. Our approach is flexible and can be applied to a system with a different configuration (e.g., ADC/DAC resolution, resistance levels, datatype size).

Figure 5(a) depicts an example of a MMM operation where each element is assumed to have three bits. Considering this operation, the elements of the multiplicand matrix (array data) have to be programmed into the crossbar while the multiplier's elements (input data) are provided to the crossbar



z1 z2 z3 z4 z5 z6 z7 z8 z9 = a b c d e f g h i * j k l m n o p q r
z1 = 2^2 (a2) * (j2 j1 j0) + 2^1 (a1) * (j2 j1 j0) + 2^0 (a0) * (j2 j1 j0)
+2^2 (b2) * (m2 m1 m0) + 2^1 (b1) * (m2 m1 m0) + 2^0 (b0) * (m2 m1 m0)
+2^2 (c2) * (p2 p1 p0) + 2^1 (c1) * (p2 p1 p0) + 2^0 (c0) * (p2 p1 p0)
(a)
BL1
BL2
BL3
WL1
a0
j2
j1
j0
WL2
b0
m2
m1
m0
WL3
c0
p2
p1
p0
(b)
2^2 * (a0j2 + b0m2 + c0p2)
2^1 * (a0j1 + b0m1 + c0p1)
2^0 * (a0j0 + b0m0 + c0p0)

FIGURE 5. (a) An example of MMM operation where the elements of the array and input data are expanded (b) mapping of data to the crossbar considering limited resistance levels as well as limited input voltage/current levels. The coefficients colored gray are not part of the crossbar output.

as input. Figure 5(b) depicts how an element of the output matrix can be calculated and mapped to the crossbar, assuming 1) elements of the multiplicand matrix have to be distributed over three memristor cells (due to the restriction on the number of resistance levels – second limitation) and 2) the elements of multiplier matrix have to be sliced into three segments as well (due to the restriction on the number of voltage/current levels provided by DACs – third limitation). The segments are given to the DACs sequentially (e.g., the first segment is a_0, b_0, and c_0).

A. FIRST STAGE: ANALOG ADDITION

The first stage of computation is performed in an analog manner. As depicted in Figure 5(b), elements with the same significant bit are summed up. The analog addition is performed on the bit-lines as soon as the DACs are activated and proper voltage levels are provided to the crossbar inputs. In an ideal case, all the crossbar rows can be activated. Therefore, all the required elements can contribute to the analog addition. However, due to some limitations, this may not be realized.

Figure 6(a) illustrates a scenario where all the input drivers (DACs) cannot be activated at the same time. This can be due to either technology restrictions or limited ADC resolution (first limitation). Hence, the analog addition should be performed among smaller sets of rows, and the intermediate results need to be summed up together in the digital domain. In the example illustrated in this figure, the first two rows (green box) are activated in the first step. Afterward, the bit-lines are scanned by the ADC in sequence, and the intermediate results are stored in the registers dedicated for each column ($R1$, $R2$, $R3$). In the second step, the third row (blue box) gets activated. The results in this step have to be accumulated with the previous step. We present the required hardware to deal with this restriction in Figure 6(a). Based on which bit-line is read by the ADC, control signals are sent to the demultiplexer and multiplexer to load and store data from and into a proper register, respectively. This hardware is only required when this limitation exists. The size of the registers employed for each column should be equal to $log2(number\ of\ rows) + log2(resistance\ levels)$. The second term in the equation is added when the number of resistance levels is more than two.

B. SECOND STAGE: SLIDING OVER MULTIPLE COLUMNS

The intermediate results obtained from each column of the crossbar in the first stage just contain one bit-position of array data (it can be more if memristor devices can hold more than one bit). In order to achieve the final results related to one segment of the input data (e.g., $a_0, b_0,\ and\ c_0$), the intermediate results of the first stage obtained from different columns, have to be summed up (e.g., column 1, 2, and 3). This is due to deploying several memristor devices to represent a number (second limitation). In the following, we explain the traditional and proposed approach to perform the required processes in this stage.

As depicted in Figure 6(b), in each time step, the result of analog addition obtained from a column of the crossbar is stored in the *ADC register*. In the traditional approach, the data stored in this register has to be shifted to take into account the coefficients associated with each column (see Figure 5(b)). Then, the result is accumulated to the values obtained from other columns in the previous time steps. Considering that approach, the number of shift operations is variable and depends on to which column the value of the ADC register belongs. In addition, as more columns contribute to the result, the data size gets increased. Therefore, the adder which performs this accumulation should be sized for the worst-case scenario. This size is equal to $log2(number\ of\ rows) + log2(resistance\ levels) + collective\ columns$. The collective columns are the number of columns that represents a single value (e.g., in this example, 3). In short, this approach requires a variable number of shift operations and large-size adders, which in consequence, reduce the performance and energy efficiency of the crossbar periphery.

In contrast to the traditional approach, our proposed design does not require shift operation and minimizes the size of adders as well as registers that are placed in this stage. Considering Figure 6(b), the value of the ADC register has to be shifted and added up to the value of $R1_{temp}$ register where the intermediate result obtained from the previous columns is stored. In the proposed design, when performing addition

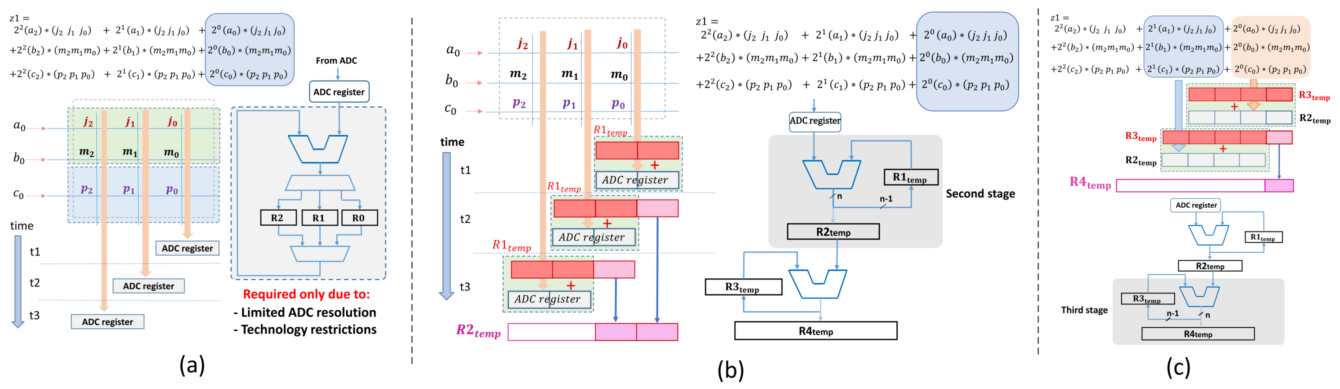


FIGURE 6. **(a) First stage: Activating rows in multi-steps and its required hardware when there is a limitation on ADC resolution or crossbar technology (b) Second stage: accumulation of partial result obtained from columns representing a number. The size of adders and registers are minimized in this stage (c) Third stage: performing addition between the higher partial results which are obtained from the input segments representing a number. The same technique applied in the second stage is used here to optimize the hardware.**

between these two registers, the least significant bit (or bits when more than one bit is stored in a single memristor cell) can be directly sent to the next level register ($R1_{temp}$) since it will not contribute to the later additions. As an example, in the first time step, the value of the ADC register is added to the value of $R1_{temp}$ (initialized to 0) and stored again in $R1_{temp}$. In the second time step, since the value of the ADC register, which currently stores the value of the second column, has to be shifted once to the left, the least significant bit of $R1_{temp}$ does not contribute to the addition. Therefore, it can be directly passed to the $R1_{temp}$ where we store the final result of this stage. As a consequence, the shift operation is implicitly implemented in the design without dedicating extra hardware to it. In addition, the size of $R1_{temp}$ register and the adder module are held as minimum as possible equal to $log2(number\ of\ rows) + log2(resistance\ levels)$. When the addition is performed for the last column (associated with the most significant bit(s) of multiplicand's elements), the entire content of $R1_{temp}$ register is copied to $R2_{temp}$ register, and the result of this stage is ready to be used for the next stage.

We perform the processing in this stage, assuming that the datatype size for array data (e.g.,j_0, J_1, and J_2) does not exceed the number of columns that share an ADC. Loosening this assumption leads to additional logic (extra stage) required to add up the intermediate results obtained from the columns representing one element of the multiplicand, but are shared among more than one ADC. To avoid complexity, this will not be discussed in detail.

C. THIRD STAGE: SLIDING OVER INPUT SEGMENTS

In this stage, the partial sum related to input segments has to be summed. Due to the input driver resolution (third limitation), the input data has to be segmented and provided to the crossbar in several iterations. Therefore, the design should be able to integrate the result of each iteration and produce the output of the MMM operation.

We illustrate this using an example in Figure 6(c) where the partial sum relating to the (a_0, b_0, c_0), (a_1, b_1, c_1), and (a_2, b_2, c_2) segments have to be integrated. The partial sum obtained from each input segment has to be shifted and added up to the value of $R3_{temp}$ register, where we store the result related to previous input segments. The design utilizes the same approach, deployed for the second stage, in order to avoid shift operations while minimizing the size of adders and registers. As an example, in the first iteration, the value of the first input segment (orange box) is available in $R2_{temp}$ register and will be added to the value of $R3_{temp}$ register, which is initialized to zero. The least significant bit of the result is directly stored in $R4_{temp}$ register, while the rest of the bits are again stored in $R3_{temp}$ register. Since the result of the second input segment (blue box) has to be shifted (theoretically) once to the left, the least significant bit of the result from the previous iteration is not contributing to the result of the current iteration. Therefore, we improve performance, area, and energy efficiency by omitting variable shift operators as well as minimizing adder and register sizes.

In summary, using the aforementioned organization leads to energy (and possibly performance) improvement specially when considering a large number of crossbars with hundreds of columns computing in parallel. Controlling this structure can be performed by designing a state machine. Alternatively, the entire system can also be controlled by employing an instruction set to exploit the maximum flexibility for the design (considering ADC resolution, cell levels, and datatype size).

V. A NOVEL SIGNED DATA REPRESENTATION AND COMPUTATION

In this section, we propose our optimized design to support two's complement representation in a CIM tile. The design uses the same hardware discussed for unsigned representation with some changes in the execution flow.

A. MOTIVATION

Despite other representations, the primitive arithmetic operations in two's complement representation are identical to

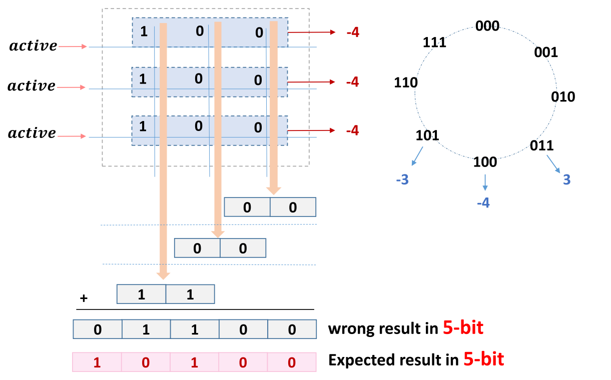

active
active
active
1 0 0
-4
0 0
0 0
+ 1 1
0 1 1 0 0 wrong result in 5-bit
1 0 1 0 0 Expected result in 5-bit
000
001
010
011
100
101
110
111
-3
-4
3

FIGURE 7. Example of sign extension required for two's complement representation when the size of the output is larger than the input data.

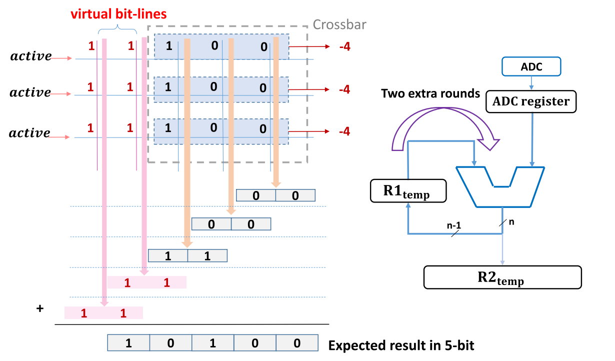


FIGURE 8. Low-cost implementation of signed addition by introducing virtual bit-lines.

unsigned numbers. This is particularly helpful when operating on many operands at once. However, considering the crossbar structure, the sign extension required for this representation imposes a big overhead on the system. Figure 7 shows an example where the range of data is from -4 up to +3 which can be represented in 3 bits. Assuming each memristor device can hold one bit, 3 devices can represent one number. As we show in this figure, by performing an addition among the numbers programmed to the crossbar, the result is different than expected. Although each number individually can be presented in 3 bits, more bits are required to present the output result since several numbers are summing up. Accordingly, input operands have to be sign-extended to be the same size as the output. In general and in the case of addition operation, the number of bits that have to be considered for sign extension is equal to $log2$ (*number of crossbar rows*). Clearly, these extra bits degrade energy, performance, and area efficiency. Therefore, it is essential to address this challenge and improve the efficiency of signed arithmetic computation in the crossbar.

The sign extension causes an overhead on both array data and input data. In Section V-B, we address the challenge of sign extension on array data using a simple arithmetic addition operation as an example. Subsequently, in Section V-C, we consider the MMM operation as a more complex operation where the overhead of sign extension exists on both array data and input data. Considering this operation, we extend our solution to address the overhead of sign extension on input data as well.

B. EFFICIENT SIGNED ADDITION BY INTRODUCING VIRTUAL BIT-LINES

To address the overhead of sign extension on array data, a new method is proposed based on what is named ***virtual bit-lines***. We illustrate our solution in Figure 8 using a simple arithmetic addition example where we want to obtain the result of addition between three signed numbers stored in the crossbar.

Instead of programming more memristor devices to capture the effect of sign extension, we propose a novel method to incorporate the impact of sign extension in the crossbar periphery. As demonstrated in Figure 8, the values obtained from the fourth and fifth columns, used to represent sign extension values, are the same as the third column. This is based on the fact that the values of the sign-extended bits are equal to the most significant bit before the sign extension. By recalling how unsigned computation is performed, in each time step, the value of one column is stored in the ADC register. In the third time step, the value of the third column is stored in the *ADC register* and has to be summed up with the intermediate result obtained from previous columns (columns 1 and 2) which are (partially) stored in the $R1_{temp}$ register. Since the values of the virtual bit-lines are equal to the value of the third column and this already exists in the *ADC register*, only the loop is required to be performed extra rounds. The loop has to be performed more times equal to the number of virtual bit-lines (in the worst-case $log2$(*number of crossbar rows*)). By doing this, 1) a fewer number of bit-lines have to be programmed (crossbar programming energy), 2) a fewer number of bit-lines are contributing to the computation (crossbar computation energy), and 3) a fewer number of bit-lines read by ADCs (ADC energy). This leads to a huge improvement in terms of area, performance, and energy efficiency.

C. EFFICIENT SIGNED MULTIPLICATION BY EMPLOYING VIRTUAL INPUT SEGMENTS AND VIRTUAL BIT-LINES

In the previous subsection, we discussed how to eliminate the overhead of sign extension on array data by only considering its contribution to the result in the periphery. In this subsection, we consider the execution model for a complete MMM operation where the overhead of sign extension exists both on array data and input data. We explain how this affects the number of virtual bit-lines and how we can optimize this operation further.

Figure 9(a) illustrates an example of the MMM operation where the elements of both multiplier and multiplicand are presented in two's complement format bound for three bits. The size of elements for the output matrix (S_{out}), as well as the number of sign extension bits for both multiplicand (E_{mpd}) and multiplier (E_{mpr}), are computed according to Equation 1, 2, and 3. Since the range of elements for the output matrix (8 bits) is larger compared to an addition operation (5 bits),

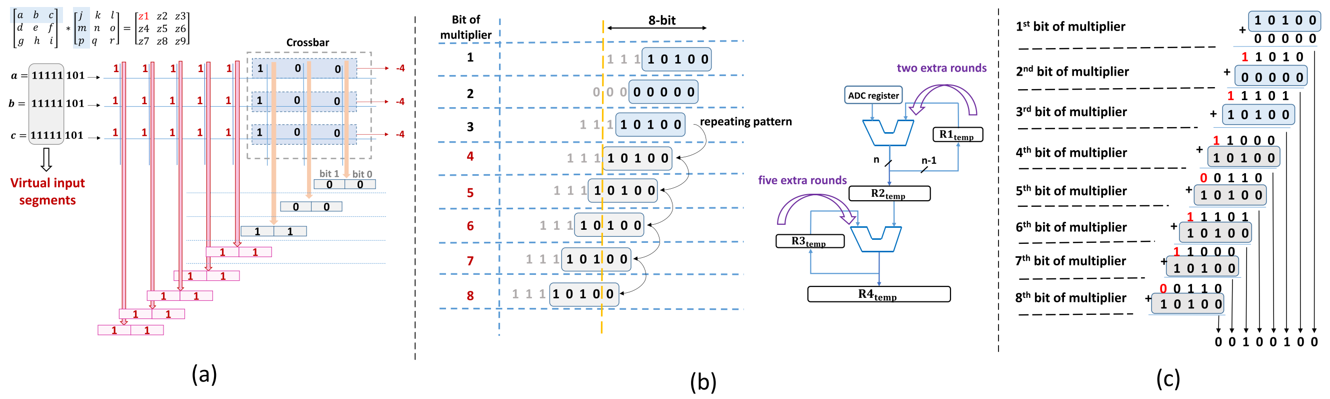


FIGURE 9. **(a) Introducing *virtual input segments* besides virtual bit-lines to reduce the overhead of signed MMM (b) applying small modifications on execution flow to support signed MMM while employing the same hardware used for unsigned computation (c) sign extending the output of second stage adder only for one-bit (red digit). This is enough for correct execution and also minimizes the size of adders/registers in the second stage.**

more overhead is imposed due to the sign extension even using the aforementioned technique. However, to reduce this overhead, we propose two optimizations as follows.

$$\begin{aligned} S_{out} &= (\text{multiplicand datatype size}) + \\ &\quad (\text{multiplier datatype size}) + \log_2(\text{active rows}) \quad (1)\\ E_{mpd} &= S_{out} - (\text{multiplicand datatype size}) \\ &= \text{number of virtual bit-lines} \quad (2)\\ E_{mpr} &= S_{out} - (\text{multiplier datatype size}) \\ &= \text{number of virtual input segments} \quad (3)\end{aligned}$$

❶ Similar to computation over unsigned numbers (Section IV), here, input segments including the sign-extended bits are given to the crossbar one at a time. In this example, since the output needs to be presented in 8 bits, we have to have 5 bits sign extension for each input operand to bring them into the same size as the output. The bits for the sign extension are indicated with red color in Figure 9(a). The result of each segment has to be summed up with the preceding segments to get the final value. Considering this example, Figure 9(b) shows the intermediate result obtained by applying each segment (here one bit at a time) to the crossbar. Due to the sign extension, the intermediate results of applying the fourth until the eighth bit of the multiplier are the same as the third bit. Similar to virtual bit-lines, we call these input segments as **virtual input segments**. Therefore, since the result by applying the virtual input segments already computed in the previous rounds and exists in the $R2_{temp}$ register, these segments are not given to the crossbar as input. Looking back on the proposed hardware for unsigned numbers, there are three stages, 1) analog addition, 2) sliding over multiple columns, and 3) sliding over input segments. Therefore, as depicted in Figure 9(b), the third stage, which is in charge of sliding over input segments, has to be performed five additional times (equal to the number of virtual input segments). Accordingly, this attains energy and performance improvement due to the fewer activations for the crossbar as well as periphery circuits.

❷ While the first optimization was focused on the input data and the virtual input segments, here we focus on the size of the adders and registers.

According to the aforementioned optimization techniques proposed in Section IV, the size of the adder associated with the third stage can be reduced down to the length of an intermediate result obtained for one input segment. However, due to the sign extension of the array data, one can think a larger adder size (e.g., 8 bits) is required since each input segment produces 8 bits as a result. By considering the example in Figure 9(b), it can be realized that from the 8-bit value resulting for each bit of input data, the last three (gray color digits) are the same as the fifth bit. This is due to the fact that giving an input bit (in general input segment) to the crossbar is like an add operation (discussed before) where the number of sign extended bits is proportional to only the number of active rows (crossbar rows in the worst-case) and the extra sign extended bits imposed by the MMM operation results to the bits with the same value. In this example, among five sign extended bits for the array data, **two bits** (or two virtual bit-lines) are due to the number of active rows (or the number of array data elements that are summed up), and the rest are imposed due to the input data in the MMM operation (see Equation 2). Hence, in order to get the first five bits of the result related to each bit of the multiplier elements, the second stage of the addition scheme requires performing only two extra rounds like what was mentioned for an add operation (more rounds lead to repeating the MSB). Consequently, we minimize 1) the number of extra rounds required in the second stage and 2) the length of output generated in the second stage which leads to downsized adders and registers in the third stage.

After obtaining the intermediate values from the second stage of the addiction scheme (e.g., 5 bits in this example), the third stage has to sum them up. Figure 9(c) depicts how to perform this summation while using the same adder size employed for unsigned operations (in this example 5-bit adder rather than 8-bit adder). As we can see, after each

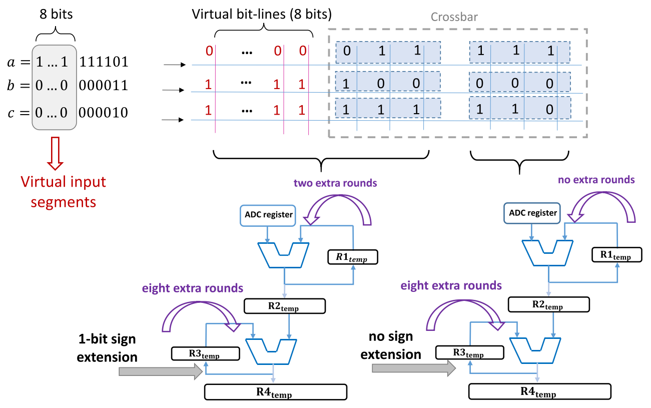


FIGURE 10. Asymmetric procedure for addition scheme when a number is split over several ADCs.

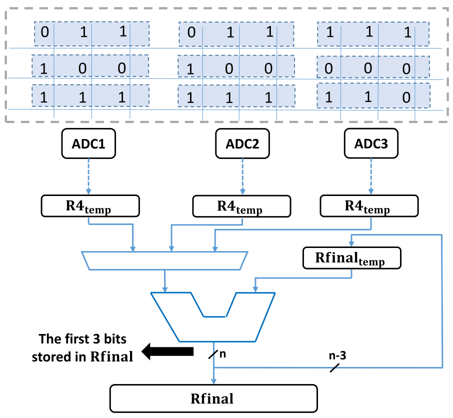


FIGURE 11. Extra stage to sum up the result obtained from addition scheme per ADC. This is illustrated for a simple addition operation, and we only require this hardware when a number is split into multiple ADCs.

addition, the LSB is directly stored in the $R4_{temp}$ register, and the rest with a one-bit sign extension will be stored in the $R3_{temp}$ register. This one-bit sign extension captures the contribution of sign-extended bits at the output of the second stage when we downsize it from 8 to 5 bits (gray digits in Figure 9(b)). Therefore, using this one-bit sign extension, we ensure the correct functionality of the third stage while reducing the size of the adder from 8-bit to 5-bit.

In order to generalize our discussion, the following equations are provided. The Equations 4 and 5 calculate the number of extra rounds required for stages 2 and 3. In addition, the size of the adders, as well as the registers for both stages, are shown in Equations 6 and 7 where $Adder_1$ and $Adder_2$ are the adders employed for stage 2 and 3, respectively. It is worth mentioning that the second term in Equation 6 is added when the number of resistance levels is more than two.

$$\text{Second Stage Extra Rounds} =$$
$$\log_2(\text{active rows}) \leq \text{number of virtual bitlines} \tag{4}$$
$$\text{Third Stage Extra Rounds} =$$
$$E_{mpr} = \text{number of virtual input segments} \tag{5}$$
$$\text{Size}(\text{Adder}_1) = \text{Size}(\text{ADC register}) = \text{Size}(\text{R1}_{temp})$$
$$= log2(\text{number of rows}) + log2(\text{resistance levels}) \tag{6}$$
$$\text{Size}(\text{Adder}_2) = \text{Size}(\text{R2}_{temp}) = \text{Size}(\text{R3}_{temp}) =$$
$$\text{multiplicand datatype size} + log2(\text{number of rows}) \tag{7}$$

D. ASYMMETRIC ADDITION SCHEME

Until now, we assumed that the array data datatype size is aligned or can be fit within the number of columns shared by an ADC. However, the proposed solution is valid even when the assumption does not hold. By increasing the number of ADCs per crossbar, the numbers which are stored in the crossbar might be distributed over several ADCs depending on the application and its datatype size. Figure 10 illustrates this scenario by employing a simple example where both input data and array data have 6 bits size and three columns shared by an ADC. As can be seen, while virtual bit-lines have only an implication on the addition unit operating on the most significant bits, the virtual input segments influence both units. Since three rows are considered, two extra rounds–$log2(number\ of\ rows)$– are imposed to the second stage of the addition unit on the left side. Besides, due to the eight segments of virtual input (here each segment is one bit), eight extra rounds have to be performed on the third stage of both units. It must be highlighted that since the virtual bit-lines are not associated with the addition unit on the right side, there is no 1-bit sign extension when storing the intermediate result in the $R3_{temp}$ register, while the addition unit on the left side requires this as explained before (see Figure 9(c)).

Considering the aforementioned scenario, when the final result associated with an ADC is achieved and stored in the $R4_{temp}$, they have to be summed up considering their positions. The same structure explained in section IV is employed here in order to minimize the size of the adder and register in this fourth stage. Figure 11 depicts an example where the values of the $R4_{temp}$ registers related to each addition unit are summed up in sequence. In each step, the first three bits of the adder's output are directly stored in the *Rfinal* register since the values in $R4_{temp}$ registers have three bits positional difference. Consequently, this reduces the size of the intermediate register as well as the adder which in turn improves performance and energy. It is worth mentioning that the processing in this stage can be carried out in parallel with the lower stages. Therefore, performing the computation sequentially in this stage does not induce performance overhead on the system since the previous stages require more time.

VI. EXPERIMENTAL SETUP

In this section, we explain how we evaluate energy, performance, and area as the main metrics in our simulation. The design is compared with two baselines on three different benchmarks which cover different scenarios when executing arithmetic operations on the crossbar.

TABLE 2. Value of parameters used for the analog components.

Component	Parameters	Spec	
clock	frequency	1GHz	
Memristive devices		ReRAM	PCM
	cell levels	2	2
	LRS	5k	20k
	HRS	1M	10M
	read voltage	0.2 V	0.2 V
	write voltage	2 V	1 V
	write current	100 μA	300 μA
	read time	10 ns	10 ns
	write time	100 ns	100 ns
Crossbar	structure	1T1R	
	num. columns	256	
	num. rows	256	
S&H	number	256	
	hold time	9.2 ms	
	latching energy	0.25 pJ	
	latency	0.6 ns	
DAC		read DAC	write DAC
	number	256	256
	power	3.9 μW	3.9 μW
ADC	power	2.6 mW	
	precision	8 bits	
	latency	1.2 GSps	

A. SIMULATION PLATFORM

The result is based on our SystemC simulator [53], [60] which is parameterized by incorporating power and timing characteristics for the memristor devices, digital, and analog circuits. First, we profile the application to decompose it into sets of MMM operations. In the next step, MMM operations are decomposed again into sets of in-memory instructions. This is done automatically by our in-memory compiler. The instructions give us programmability and controllability over memristor crossbars. The instructions are then executed by our in-memory simulators which mimic the behavior of the hardware including the crossbar and its analog as well as digital peripheries. The simulator reports the simulation time and energy numbers based on the activity factors of different components. The energy and latency numbers for each digital component (e.g., decoder, addition scheme) are obtained by a synthesis tool (Cadence Genus) and then exported into the simulator. The code for our compiler and simulator is available online [61]. All the values for the analog components of the CIM-tile are summarized in Table 2. The parameters for ReRAM and PCM technologies are taken from [62] and [39] validated for the actual device. For all the simulations, it is considered each memristor cell can hold one bit (two resistance levels).

1) PERFORMANCE

The simulator executes the in-memory instructions generated by our compiler to control the CIM tile, program the devices, and perform the computations. The instructions are decoded and executed in 1 GHz clock frequency. Assuming one bit stored in each cell, in order to activate all the crossbar rows at the same time, an 8-bit ADC at 32 nm is employed [59]. For all the simulations, each ADC is shared between 8 bit-lines, which directly influences the performance of the system.

2) AREA

The area of each ReRAM cell in the 1T1R crossbar structure is taken from [63]. The digital peripheries are synthesized in Cadence Genus targeting standard cell 15 nm Nangate library to obtain the area consumption. The number of required CIM tiles depends on the benchmarks.

3) ENERGY

Since all the information and control signals can be tracked by employing our simulator, a typical **activity factor** and performance number are extracted. These numbers are incorporated to achieve accurate energy consumption for the digital as well as analog components of the tile. The power consumption for ADC was obtained from [59]. Besides, the power consumption of 1-bit DAC is taken from [58]. However, in the simulations where more bits are required, the power consumption is taken from the equations provided in [58]. The power consumption related to the digital periphery is taken from the Cadence Genus report targeting standard cell 15 nm Nangate library.

B. BENCHMARKS

When we execute arithmetic matrix-matrix multiplication on the crossbar, there would be two scenarios in the perspective of the signed computation: 1) only one operand or 2) both operands (input and programming data) have to be expressed as a signed number. The benchmarks are selected to be able to assess the behavior of both scenarios. Considering the first scenario, a neural network is employed to classify the MNIST database. The network has two hidden layers with 80 and 60 neurons, respectively. The network trained, providing around %97 accuracy, and the weights are obtained using MATLAB. The programming weights are presented in 8-bit **signed** fix-point whereas the input data (pixels) is considered as 8-bit **unsigned** number. It is worth mentioning that the intermediate values generated by the hidden layers are considered binary numbers. For the second scenario, we take two kernels from the Polybench/C benchmark suite. The first kernel is linear-algebra "gemm" where the input matrix size is 1000 × 1200, and the programming data matrix size is 1200 × 1100. Similarly, the second kernel, "3m", has the problem size of 800 × 1000 and 1000 × 900 for the input and programming data, respectively. The datatype size for the aforementioned kernels is 8-bit, and despite the first benchmark, both input and programming data are presented as **signed** numbers. Since we only focus on the crossbar and its periphery circuit in this paper and not a full system simulation, considering more benchmarks will not add more insight to our study, and a similar pattern will be repeated.

C. BASELINES

The proposed design is compared against two baselines to quantify its efficiency in performing signed computations in terms of energy, performance, and area. The baselines are as follows:

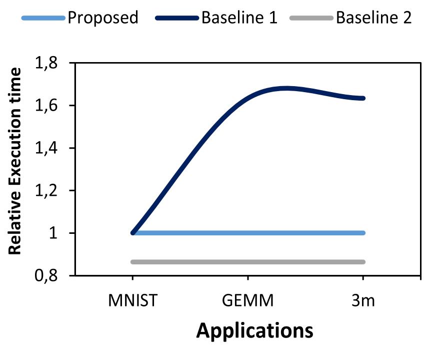


FIGURE 12. Relative execution time of the baselines normalized to the proposed design. No distinction between the two memristor technologies (According to Table 2, the two memristor technologies have similar latency numbers).

- Baseline1:
  In this baseline, both the inputs [19] and the programming data [18] are presented as a standard two's complement representation where the sign extension is required for both multiplier and multiplicand in the case of MMM operation. Furthermore, in order to have a fair comparison, we employ the same efficient structure proposed for the *addition unit* (see Section IV and [24]) rather than conventional *Shift-and-add* structure. Therefore, the result will completely focus on the way signed computation is performed. This *addition unit* with the proposed structure is solely evaluated in [24] against conventional *Shift-and-add* structure.
- Baseline2:
  In this approach, the positive and negative programming data are mapped into two different crossbars where the data on each crossbar is treated as unsigned value [20], [21]. Furthermore, in order to support signed input, the current is provided in two directions to represent negative and positive numbers (see Figure 4(a)). For all the simulations regarding this baseline, we consider 2-bit DACs to provide three voltage levels corresponding to logical -1, 0, and 1 values. Finally, the same *addition unit* considered for the first baseline is also employed here.

VII. RESULTS

In this section, we evaluate the proposed design in terms of execution time, area, and computation as well as programming energy. In the following, the result regarding each of the aforementioned aspects of the design is discussed to provide a good insight into its pros and cons.

A. EXECUTION TIME

Figure 12 depicts the relative execution time for the two baselines normalized with the proposed scheme. As can be seen in this figure, due to the extra rounds of addition required in the proposed scheme (see Figure 9), a small overhead is imposed on the execution time compared to Baseline 2. According to the description of Baseline 2, since the drivers have to generate three voltage levels, they may have more latency than the drivers employed for the proposed design as well as Baseline 1. This may have an adverse impact on the performance of Baseline 2 and reduce the gap between this scheme and the proposed design. Considering MNIST, Baseline 1 and the proposed scheme attain a similar performance since either 8 extra crossbar columns (Baseline 1) or 8 extra rounds in the *addition unit* (proposed design) has to be performed. This is due to the sign extension. However, if the digital clock frequency gets bigger than the ADC conversion rate, the overhead of extra rounds in the *addition unit* will be reduced. Regarding GEMM and 3m, since the inputs are also signed, the execution time for the Baseline 1 is increased due to the costly sign extension of input data. It is worth mentioning that the overhead of data communication to provide input data to the crossbar is considered for all the designs. Based on the architecture proposed in [53], in order to provide a clear separation of tile from outside and minimize the number of synchronizations, a buffer called *Row Data Buffer* with the width equal to the maximum supported datatype size, and the heights equal to the number of crossbar rows is placed next to the crossbar. This buffer provides input data for the crossbar. Filling each row of this buffer with the elements of the multiplier matrix takes one clock cycle.

As mentioned before, when we execute arithmetic matrix-matrix multiplication on the crossbar, there would be two scenarios in the perspective of the signed computation: 1) only one operand or 2) both operands (input and programming data) have to be expressed as a signed number. As can be seen in Figure 12, for the scenario where the inputs are unsigned (MNIST), the proposed design cannot outperform Baseline 2 in terms of performance. Baseline 2 maps positive and negative weights into two different crossbars. Hence, the required computations are performed in parallel in two crossbars and their peripheries. However, this computation is performed sequentially in the periphery of the proposed design. Although this could significantly improve energy and area consumption (we discuss it in the following), it brings marginal performance loss. It should be noted that this only happens in the scenario where the inputs are unsigned numbers.

B. AREA

Figure 13 depicts the area consumed by the three designs. Regarding the first baseline, since we stored the sign-extension bits in the crossbar, more crossbars have to be employed, and in consequence, more area is consumed. In the second baseline, we do not need this costly sign extension, but since positive and negative elements in a matrix have to be assigned to different crossbars, it consumes more area than the proposed design. Finally, the actual area number for each design depends on the problem size of the benchmarks. However, the ratio between these designs remains constant for different benchmarks. It should be noted that the area of digital peripheries is also considered in this figure for both the proposed design as well as the baselines. These numbers are based on the 15 nm Nangate library.

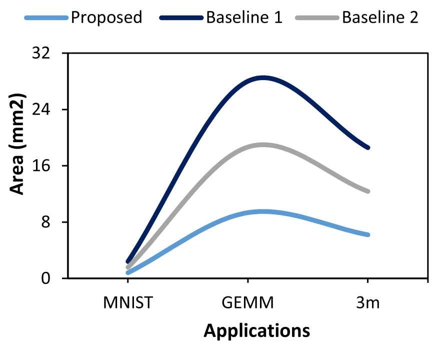


FIGURE 13. Area comparison of the Baselines with the proposed design for ReRAM crossbars taking into account the digital and analog components.

C. ENERGY

1) COMPUTATION ENERGY

Figure 14(a) demonstrates the computation energy for the aforementioned benchmarks considering the baselines and the proposed design. The results are presented for PCM and ReRAM technologies without considering the programming energy.

MNIST: For the MNIST benchmark, the experiment is performed for 10k input images, and as can be seen, more than 3× energy improvement is obtained compared to the baselines. In the following, we analyze the results obtained for each baseline.

- Regarding the first baseline and considering the 8-bit datatype size, the programming weights should be sign-extended to 24 bits (only the programming weights are signed, not inputs). This is because the inputs are 8 bits unsigned, and the total number of elements on each column of the crossbar is 256. Hence, the output size would be $24 = log2(256) + 8$. According to the output size, the weights should be sign extended to 24 bits. The implication is the necessity of more crossbar arrays together with their peripheries to encompass the weights. This leads to more energy consumption, as shown in Figure 14(a).
- Considering the second baseline, the number of crossbar arrays is doubled due to allocating positive and negative weights into different crossbars. Although the inputs for this application are not signed, the platform considered for this baseline has a higher precision of DACs and ADCs to support negative inputs, which imposes energy overhead on the system. It should be taken into account that the ADCs need to be sensitive enough in this baseline to detect finer current/voltage values in order to keep the accuracy as high as required. In conclusion, for this baseline, both the crossbar and peripheries contribute more to energy consumption than the proposed design.

Furthermore, we perform the analysis for two memristor technologies, ReRAM and PCM. According to Table 2, PCM has higher resistance values for both low (LRS) and high (HRS) states. Therefore, having the same ‘read voltage’, the energy consumed by this device is less than ReRAM during the computation. We can observe this in Figure 14(a), where we report the energy consumption for these two technologies.

GEMM and 3m: The problem sizes for these two kernels are larger than MNIST. In addition, the inputs and programming weights are 8-bit signed numbers. Therefore, considering the first baseline, both the inputs and programming weights should be sign-extended to 24 bits. Hence, not only more crossbars are required, but more input segments should be given to the crossbars as well. This leads to around 8× more energy consumption compared to the proposed design. Despite the first baseline, the approach used in the second baseline can deal with signed inputs with less overhead. Comparing the proposed design with the second baseline and similar to MNIST, the improvement is mainly obtained from employing fewer crossbars as well as lower DACs and ADCs precision.

2) PROGRAMMING ENERGY

Besides the high energy consumption of programming the memristive devices, the endurance of these devices is a critical feature [64], which persuades the users to program them as minimum as possible. Therefore, usually, it is simply considered that the crossbar is programmed once before the execution of the application without (or with minimum) reprogramming during the execution. However, it is necessary for the designer to have insight into the overhead of crossbar programming compared to the actual computation phase. Hence, Figure 14(b) provides the programming energy as well as the ratio of energy consumption for the computation over the programming phase.

Considering MNIST and after the classification of 10K input images, the result shows that the overhead of programming is getting negligible compared to the computation phase. However, this overhead is a bit higher for PCM devices due to the higher programming current. In addition, since the proposed design requires fewer crossbars, the energy overhead imposed by device programming is lessened compared to the baselines. This might also have positive implications on performance and aging which has not been considered in this paper. Since GEMM and 3m kernels have a larger problem size (require more crossbars) and relatively less computation, the overhead of programming is increased more. It should be noted that similar to MNIST, these two kernels can also be executed multiple times for different input matrices. Therefore, based on the system requirements and the application data flow, the designer should evaluate when and for how many times the devices can be reprogrammed.

D. LIMITATIONS AND CHALLENGES

The proposed design achives 8× and 3× energy and area efficiency. However, this comes with two main drawbacks. In the following, we provide a short explanation for each of them.

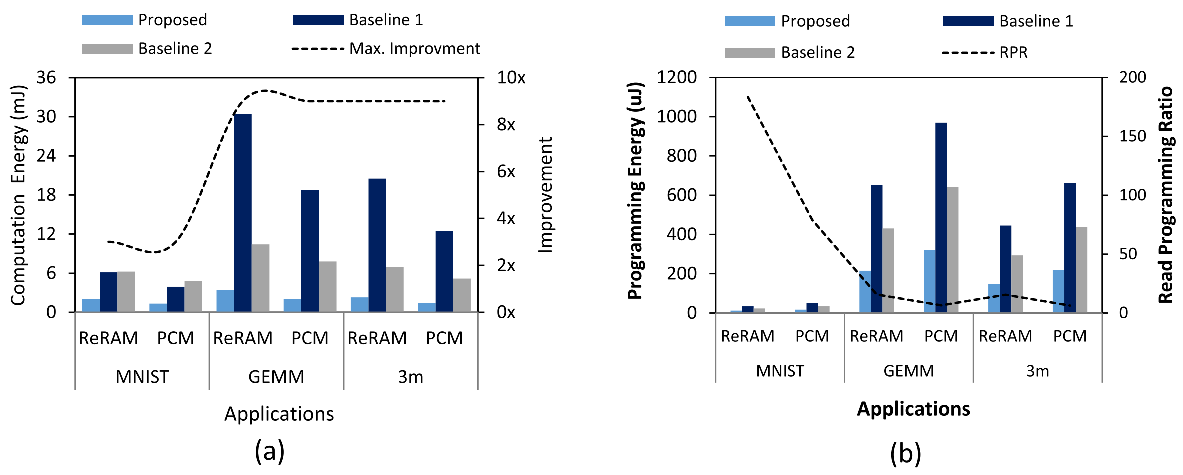


FIGURE 14. Energy consumption for (a) computation and (b) programming phase achieved for the two baselines and the proposed design. The energy comparison among different designs is performed for PCM and ReRAM technology.

1) Performance limitation:
   The proposed design has marginally lower performance compared to baseline 2 where positive and negative weights are mapped into two different crossbars. This is because the required computations are broken into two parts and performed in parallel. However, more sequential operations are performed in the peripheries of the proposed design. This performance loss highly depends on the clock frequency of digital circuits. If the clock frequency is high enough, the latency of the crossbar can hide the latency of digital peripheries.
2) Complexity of the controller:
   The proposed design can flexibility support different datatype sizes. This comes at the cost of a more complex controller. There are two approaches to control this design 1) implementing a state machine or 2) using instructions. The second approach relies more on the compiler and reduces the complexity of the hardware. However, more complex headway is still required compared to the baselines, with no flexibility in supporting different data sizes.

VIII. CONCLUSION

In summary, we proposed a new scheme based on two's complement representation to perform signed arithmetic matrix multiplication tailored for memristor-based crossbar array structure. The key insight of this novel scheme is that with the help of an innovative digital periphery design, the required sign extension can be avoided. The design provides flexibility to perform computation over different datatype sizes as well as switching between sign and unsigned computation at run time. The simulation results show that the proposed design achieves up to 8× energy-saving and 3× area-saving compared to the baselines. In our future work, we aim to employ the presented methods of supporting signed numbers in our system-level platform to figure out the contribution of energy/performance improvement in the entire system. Our current simulation platform will be integrated into Gem5, which will allow us to mimic the behavior of the full system with extensive analysis and more complex applications. Then, we can realize the contribution of the proposed scheme to the total energy and performance of the system.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[2] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating graph processing using ReRAM," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 531–543.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[4] I. Giannopoulos, A. Singh, M. L. Gallo, V. P. Jonnalagadda, S. Hamdioui, and A. Sebastian, "In-memory database query," *Adv. Intell. Syst.*, vol. 2, no. 12, 2020, Art. no. 2000141.

[5] H. Jin, C. Liu, H. Liu, R. Luo, J. Xu, F. Mao, and X. Liao, "ReHy: A ReRAM-based digital/analog hybrid PIM architecture for accelerating CNN training," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2872–2884, Nov. 2022.

[6] J. Sun, Y. Wang, P. Liu, S. Wen, and Y. Wang, "Memristor-based neural network circuit with multimode generalization and differentiation on Pavlov associative memory," *IEEE Trans. Cybern.*, early access, Sep. 21, 2022, doi: 10.1109/TCYB.2022.3200751.

[7] J. Yu, H. A. D. Nguyen, L. Xie, M. Taouil, and S. Hamdioui, "Memristive devices for computation-in-memory," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1646–1651.

[8] (2017). *Calculating Memory Power for DDR4 SDRAM*. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/technical-note/dram/tn4007_ddr4_power_calculation.pdf

[9] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proc. 8th ACM Int. Conf. Autonomic Comput.*, Jun. 2011, pp. 31–40, doi: 10.1145/1998582.1998590.

[10] S. Srikanth, L. Subramanian, S. Subramoney, T. M. Conte, and H. Wang, "Tackling memory access latency through DRAM row management," in *Proc. Int. Symp. Memory Syst.*, Oct. 2018, pp. 137–147.

[11] J. Doweck, W.-F. Kao, A. K.-Y. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, "Inside 6th-generation Intel core: New microarchitecture code-named Skylake," *IEEE Micro*, vol. 37, no. 2, pp. 52–62, Mar. 2017.

[12] S. Hamdioui, L. Xie, H. A. D. Nguyen, M. Taouil, K. Bertels, H. Corporaal, H. Jiao, F. Catthoor, D. Wouters, L. Eike, and J. van Lunteren, ''Memristor based computation-in-memory architecture for data-intensive applications,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib.* San Jose, CA, USA: EDA Consortium, 2015, pp. 1718–1725.

[13] M. K. Rahmani, M. Ismail, C. Mahata, and S. Kim, ''Effect of interlayer on resistive switching properties of SnO_2-based memristor for synaptic application,'' *Results Phys.*, vol. 18, Sep. 2020, Art. no. 103325.

[14] K. Kim, S. Park, S. M. Hu, J. Song, W. Lim, Y. Jeong, J. Kim, S. Lee, J. Y. Kwak, J. Park, J. K. Park, B.-K. Ju, D. S. Jeong, and I. Kim, ''Enhanced analog synaptic behavior of SiN_x/a-Si bilayer memristors through Ge implantation,'' *NPG Asia Mater.*, vol. 12, no. 1, pp. 1–13, Dec. 2020.

[15] C. Bengel, A. Siemon, F. Cüppers, S. Hoffmann-Eifert, A. Hardtdegen, M. von Witzleben, L. Hellmich, R. Waser, and S. Menzel, ''Variability-aware modeling of filamentary oxide-based bipolar resistive switching cells using SPICE level compact models,'' *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 12, pp. 4618–4630, Dec. 2020.

[16] C. Funck and S. Menzel, ''Comprehensive model of electron conduction in oxide-based memristive devices,'' *ACS Appl. Electron. Mater.*, vol. 3, no. 9, pp. 3674–3692, Sep. 2021.

[17] S. Kim, J. Park, T.-H. Kim, K. Hong, Y. Hwang, B.-G. Park, and H. Kim, ''4-bit multilevel operation in overshoot suppressed Al_2O_3/TiO_x resistive random-access memory crossbar array,'' *Adv. Intell. Syst.*, vol. 4, no. 9, Sep. 2022, Art. no. 2100273.

[18] D. Fujiki, S. Mahlke, and R. Das, ''In-memory data parallel processor,'' *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 1–14, Nov. 2018.

[19] A. Shafiee, ''ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,'' *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.

[20] P. Chi, ''Prime: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory,'' *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 27–39, Jun. 2016.

[21] L. Song, X. Qian, H. Li, and Y. Chen, ''PipeLayer: A pipelined ReRAM-based accelerator for deep learning,'' in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 541–552.

[22] F. Cai, ''A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations,'' *Nature Electron.*, vol. 2, no. 7, pp. 290–299, Jul. 2019.

[23] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, ''Fully hardware-implemented memristor convolutional neural network,'' *Nature*, vol. 577, no. 7792, pp. 641–646, 2020.

[24] M. Zahedi, M. Mayahinia, M. Abu Lebdeh, S. Wong, and S. Hamdioui, ''Efficient organization of digital periphery to support integer datatype for memristor-based CIM,'' in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2020, pp. 216–221.

[25] K. Fleck, U. Bottger, R. Waser, N. Aslam, S. Hoffmann-Eifert, and S. Menzel, ''Energy dissipation during pulsed switching of strontium-titanate based resistive switching memory devices,'' in *Proc. 46th Eur. Solid-State Device Res. Conf. (ESSDERC)*, Sep. 2016, pp. 160–163.

[26] D. J. Wouters, R. Waser, and M. Wuttig, ''Phase-change and redox-based resistive switching memories,'' *Proc. IEEE*, vol. 103, no. 8, pp. 1274–1288, Aug. 2015.

[27] M. Zahedi, T. Shahroodi, G. Custers, A. Singh, S. Wong, and S. Hamdioui, ''System design for computation-in-memory: From primitive to complex functions,'' in *Proc. IFIP/IEEE 30th Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2022, pp. 1–6.

[28] J. Yu, H. A. D. Nguyen, M. A. Lebdeh, M. Taouil, and S. Hamdioui, ''Time-division multiplexing automata processor,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 794–799.

[29] Y.-D. Kim, H.-S. Ahn, S. Kim, and D.-K. Jeong, ''A high-speed range-matching TCAM for storage-efficient packet classification,'' *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 6, pp. 1221–1230, Jun. 2009.

[30] P. Van Mieghem and F. A. Kuipers, ''Concepts of exact QoS routing algorithms,'' *IEEE/ACM Trans. Netw.*, vol. 12, no. 5, pp. 851–864, Oct. 2004.

[31] H. Nili, ''Hardware-intrinsic security primitives enabled by analogue state and nonlinear conductance variations in integrated memristors,'' *Nature Electron.*, vol. 1, no. 3, pp. 197–202, Mar. 2018.

[32] B. Cambou, D. Hély, and S. Assiri, ''Cryptography with analog scheme using memristors,'' *ACM J. Emerg. Technol. Comput. Syst.*, vol. 16, no. 4, pp. 1–30, Oct. 2020.

[33] M. Masoumi, ''Novel hybrid CMOS/memristor implementation of the AES algorithm robust against differential power analysis attack,'' *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 7, pp. 1314–1318, Jul. 2020.

[34] C. E. Graves, C. Li, X. Sheng, D. Miller, J. Ignowski, L. Kiyama, and J. P. Strachan, ''In-memory computing with memristor content addressable memories for pattern matching,'' *Adv. Mater.*, vol. 32, no. 37, Sep. 2020, Art. no. 2003437.

[35] S. Muthulakshmi, C. S. Dash, and S. R. S. Prabaharan, ''Memristor augmented approximate adders and subtractors for image processing applications: An approach,'' *AEU-Int. J. Electron. Commun.*, vol. 91, pp. 91–102, Jul. 2018.

[36] I.-A. Fyrigos, V. Ntinas, G. C. Sirakoulis, P. Dimitrakis, and I. Karafyllidis, ''Memristor hardware accelerator of quantum computations,'' in *Proc. 26th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2019, pp. 799–802.

[37] M. A. Zidan, ''A general memristor-based partial differential equation solver,'' *Nature Electron.*, vol. 1, pp. 411–420, Jul. 2018.

[38] Z. Sun, ''In-memory eigenvector computation in time O(1),'' *Adv. Intell. Syst.*, vol. 2, no. 8, 2020, Art. no. 2000042.

[39] M. Le Gallo, A. Sebastian, G. Cherubini, H. Giefers, and E. Eleftheriou, ''Compressed sensing with approximate message passing using in-memory computing,'' *IEEE Trans. Electron Devices*, vol. 65, no. 10, pp. 4304–4312, Oct. 2018.

[40] R. Cai, A. Ren, Y. Wang, and B. Yuan, ''Memristor-based discrete Fourier transform for improving performance and energy efficiency,'' in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2016, pp. 643–648.

[41] C. Li, ''Analogue signal and image processing with large memristor crossbars,'' *Nature Electron.*, vol. 1, no. 1, pp. 52–59, 2018.

[42] A. Yousefzadeh, J. Stuijt, M. Hijdra, H.-H. Liu, A. Gebregiorgis, A. Singh, S. Hamdioui, and F. Catthoor, ''Energy-efficient in-memory address calculation,'' *ACM Trans. Archit. Code Optim.*, vol. 19, no. 4, pp. 1–16, Dec. 2022.

[43] C. Li, ''Long short-term memory networks in memristor crossbar arrays,'' *Nature Mach. Intell.*, vol. 1, pp. 49–57, Jan. 2019.

[44] T. Shahroodi, M. Zahedi, C. Firtina, M. Alser, S. Wong, O. Mutlu, and S. Hamdioui, ''Demeter: A fast and energy-efficient food profiler using hyperdimensional computing in memory,'' *IEEE Access*, vol. 10, pp. 82493–82510, 2022.

[45] H. A. D. Nguyen, ''A computation-in-memory accelerator based on resistive devices,'' in *Proc. Int. Symp. Memory Syst.*, 2019, pp. 19–32.

[46] F. Zokaee, M. Zhang, and L. Jiang, ''FindeR: Accelerating FM-index-based exact pattern matching in genomic sequences through ReRAM technology,'' in *Proc. 28th Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, Sep. 2019, pp. 284–295.

[47] Q. Lou, ''Helix: Algorithm/architecture co-design for accelerating nanopore genome base-calling,'' in *Proc. ACM Int. Conf. Parallel Archit. Compilation Techn.*, 2020, pp. 293–304.

[48] T. Shahroodi, M. Zahedi, A. Singh, S. Wong, and S. Hamdioui, ''KrakenOnMem: A memristor-augmented HW/SW framework for taxonomic profiling,'' in *Proc. 36th ACM Int. Conf. Supercomput.*, Jun. 2022, pp. 1–14.

[49] S. Diware, S. Dash, A. Gebregiorgis, R. V. Joshi, C. Strydis, S. Hamdioui, and R. Bishnoi, ''Severity-based hierarchical ECG classification using neural networks,'' *IEEE Trans. Biomed. Circuits Syst.*, early access, Feb. 6, 2023, doi: 10.1109/TBCAS.2023.3242683.

[50] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, ''DUAL: Acceleration of clustering algorithms using digital-based processing in-memory,'' in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 356–371.

[51] N. TaheriNejad, ''SIXOR: Single-cycle in-memristor XOR,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 5, pp. 925–935, May 2021.

[52] L. Xie, H. A. D. Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. AlFailakawi, and S. Hamdioui, ''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.

[53] M. Zahedi, R. van Duijnen, S. Wong, and S. Hamdioui, ''Tile architecture and hardware implementation for computation-in-memory,'' in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2021, pp. 108–113.

[54] X. Yang, B. Taylor, A. Wu, Y. Chen, and L. O. Chua, ''Research progress on memristor: From synapses to computing systems,'' *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 5, pp. 1845–1857, May 2022.

[55] A. Gebregiorgis, A. Singh, A. Yousefzadeh, D. Wouters, R. Bishnoi, F. Catthoor, and S. Hamdioui, "Tutorial on memristor-based computing for smart edge applications," *Memories Mater., Devices, Circuits Syst.*, vol. 4, Jul. 2023, Art. no. 100025.

[56] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM: In-memory acceleration of deep neural network training with high precision," in *Proc. 46th Int. Symp. Comput. Archit.*, Jun. 2019, pp. 802–815.

[57] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC-memristor-aided logic," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 11, pp. 895–899, Nov. 2014.

[58] M. Saberi, R. Lotfi, K. Mafinezhad, and W. A. Serdijn, "Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation ADCs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 8, pp. 1736–1748, Aug. 2011.

[59] L. Kull, "A 3.1 mW 8b 1.2 GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32 nm digital SOI CMOS," *IEEE J. Solid-State Circuits*, vol. 48, no. 12, pp. 3049–3058, Dec. 2013.

[60] M. Zahedi, M. A. Lebdeh, C. Bengel, D. Wouters, S. Menzel, M. L. Gallo, A. Sebastian, S. Wong, and S. Hamdioui, "MNEMOSENE: Tile architecture and simulator for memristor-based computation-in-memory," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 3, pp. 1–24, Jul. 2022.

[61] *Memristor-CIM-Tile-Architecture-TUDeflt*. [Online]. Available: https://github.com/mahdi-zahedi/Memristor-CIM-tile-architecture-TUDeflt-

[62] A. Hardtdegen, C. L. Torre, F. Cuppers, S. Menzel, R. Waser, and S. Hoffmann-Eifert, "Improved switching stability and the effect of an internal series resistor in HfO_2/TiO_x bilayer ReRAM cells," *IEEE Trans. Electron Devices*, vol. 65, no. 8, pp. 3229–3236, Aug. 2018.

[63] O. Golonzka, "Non-volatile RRAM embedded into 22FFL FinFET technology," in *Proc. Symp. VLSI Technol.*, Jun. 2019, pp. T230–T231.

[64] S. Rashidi, M. Jalili, and H. Sarbazi-Azad, "A survey on PCM lifetime enhancement schemes," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 76:1–76:38, Aug. 2019, doi: 10.1145/3332257.

MAHDI ZAHEDI received the M.Sc. degree in electrical engineering from the University of Tehran, Iran, in 2018. He is currently pursuing the Ph.D. degree in electrical engineering with the Delft University of Technology (TU Delft), Delft, The Netherlands. His current research interests include computer architecture and system-level HW/SW co-design.

TAHA SHAHROODI received the B.Sc. degree in computer engineering from the Sharif University of Technology (SUT), Tehran, Iran, in 2018, and the M.Sc. degree in computer science from ETH Zürich, Zürich, Switzerland, in 2020. He is currently pursuing the Ph.D. degree with the Delft University of Technology (TU Delft), The Netherlands. His current research interests include bioinformatics, computer architecture, and hardware/software co-design.

STEPHAN WONG (Senior Member, IEEE) received the Ph.D. degree from the Delft University of Technology (TU Delft), The Netherlands, in December 2002. His Ph.D. thesis titled "Microcoded Reconfigurable Embedded Processor" describes the MOLEN polymorphic processor, organization, and (micro-) architecture. He is currently an Associate Professor with TU Delft. His research interests include reconfigurable computing, distributed collaborative computing, high-performance computing, embedded systems, and hardware/software co-design.

SAID HAMDIOUI (Senior Member, IEEE) received the M.S.E.E. and Ph.D. degrees (Hons.) from the Delft University of Technology (TU Delft), The Netherlands. Before joining TU Delft as a Professor, he was with Intel, Santa Clara, CA, USA, Philips Semiconductors Research and Development, Crolles, France, and Philips/NXP Semiconductors, Nijmegen, The Netherlands. He is currently a Chair Professor of dependable and emerging computer technologies and the Head of the Computer Engineering Laboratory (CE-Lab), TU Delft. He owns two patents, has published one book and contributed to the other two, and had coauthored more than 250 conference papers and journal articles. He has consulted for many semiconductor companies in the area of memory testing. His research interests include dependable CMOS nanocomputing (including reliability, testability, and hardware security) and emerging technologies and computing paradigms (including 3-D stacked ICs, memristors for logic and storage, and in-memory computing). He was a recipient of many international/national awards, such as the Best Tech Idea Award of The Netherlands, in 2021, the European Commission Components and Systems Innovation Award for the most innovative H2020 Project, the 2015 HiPEAC Technology Transfer Award, and the European Design Automation Association Outstanding Dissertation Award, in 2003. He received many best paper awards and nominations at leading international conferences, such as DATE 2020 and 2021, ITC 2021, ETS 2021, IVSLSI 2016, HPCS 2016, and ICCD 2015). He was an Associate Editor of many journals, including IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS and *JETTA*. He serves on the editorial board of *IEEE Design & Test* and *ACM Journal on Emerging Technologies in Computing Systems* (JETC).