Exact Stochastic Computing Multiplication in Memristive Memory

Mohsen Riahi Alam and M. Hassan Najafi
University of Louisiana at Lafayette, Lafayette, LA 70504 USA

Nima TaheriNejad
Technische Universität Wien (TU Wien), Vienna, Austria

Editor's notes:
This article focuses on memristive nano-devices and their use for stochastic computing (SC). It demonstrates how to convert numbers between binary and stochastic domains and how to perform multiplications using in-memory computations by the memristive logic family "MAGIC." In contrast to earlier works on memristive SC, the authors do not harness the intrinsic stochasticity of memristive devices but rather create deterministic SNs using well-defined operations.
—Weikang Qian, Shanghai Jiao Tong University

Transferring data between memory and processing units in conventional computing systems is expensive in terms of energy and latency. It also constitutes the performance bottleneck, also known as Von-Neumann's bottleneck. Memristors offer a promising solution by tackling this challenge via in-memory computation (IMC), i.e., the ability to both store and process data within memory cells. One promising in-memory logic for IMC is memristor-aided logic (MAGIC) [2]. In MAGIC, NOR and NOT logical operations can be natively executed within memory and with a high degree of parallelism [3]. Thus, applications such as stochastic computing (SC) that execute the same instruction on multiple data in parallel can benefit greatly from MAGIC.

Multiplication is a common but complex operation used in many data-intensive applications such as digital signal processing and convolutional neural networks. In-memory methods for fixed-point binary multiplication using MAGIC have been previously investigated [3], [4]. These methods are faster and more energy efficient than conventional off-memory binary multipliers. However, memristive technology is not a fully mature technology yet, in particular, compared to complementary metal–oxide–semiconductor (CMOS) technology [5]. It suffers from considerable process variations and nonidealities that affect its performance. These nonidealities can lead to the introduction of faults and noise into the memristive memory and in-memory calculations. The inherent vulnerability of fixed-point binary methods to fault and noise (e.g., to bit flips) poses a challenge to the reliability of the system.

SC [6] is a reemerging computing paradigm that offers simple execution of complex arithmetic functions. The paradigm is more robust against fault and noise compared to conventional binary computing. Multiplication, as a complex operation in conventional binary designs, can be implemented using simple standard AND gates in SC [6]. Input data are converted from binary to independent (uncorrelated) bit streams and connected to the inputs of the AND gate. Logical 1's are produced at the output of the gate with a probability equal to the product of the input data.

Digital Object Identifier 10.1109/MDAT.2021.3051296
Date of publication: 13 January 2021; date of current version: 6 December 2021.

Copublished by the IEEE CEDA, IEEE CASS, IEEE SSCS, and TTTC

An important overhead of performing computation in the stochastic domain is the cost of converting data between binary and stochastic representation. Prior works have exploited the intrinsic nondeterministic properties of memristors to generate random stochastic bit streams in memory [7], [8]. The bit-stream generation and the computation performed, however, are both probabilistic and approximate. Often very long bit streams must be processed to produce acceptable results. These make the previous SC-based in-memory multipliers inefficient compared to their fixed-point binary counterparts. In this work, to the best of our knowledge, we develop the first exact SC-based in-memory multiplier. The proposed multiplier can perform fully accurate multiplication, replacing the conventional binary multiplier, when needed. To this end, we exploit the recent progress in SC: deterministic and accurate computation with stochastic bit streams [9]. The proposed multiplication method benefits from the complementary advantages of both SC and memristive IMC to enable energy-efficient and low-latency multiplication of data. In summary, the main contributions of this work are as follows:

- Performing deterministic and accurate bit-stream-based multiplication in memory. To this end, we propose using memristive crossbar memory arrays and MAGIC logic.
- Proposing an efficient in-memory method for generating deterministic bit streams from binary data, which takes advantage of the inherent properties of memristive memories.
- Improving the speed and reducing the memory usage as compared to the state-of-the-art (SoA) limited-precision in-memory binary multipliers.
- Reducing latency and energy consumption compared to the SoA accurate off-memory SC multiplication techniques.

Background

Deterministic computation with stochastic bit streams

In SC, data are represented by streams of 0's and 1's. Independent of the length and distribution of 1's, the ratio of the number of 1's to the length of the bit stream determines the data value. For example, bit streams 0100 and 11000000 both represent 0.25 in the stochastic domain. Compared to conventional binary radix, this form of representation is more noise-tolerant as all bits have equal weight. A single bit-flip, regardless of its position in the bit stream, introduces a least significant bit error.

Deterministic approaches of SC [9] were proposed recently to perform accurate computation with SC circuits. By properly structuring bit streams, these methods are able to produce exact (fully accurate) output. Clock-dividing bit streams, using bit streams with relatively prime lengths, rotation of bit streams, and using low-discrepancy (LD) bit streams are the primary deterministic methods. Compared to conventional SC, with these methods, the bit-stream length is reduced by a factor of approximately $(1/2^N)$ where N is the equivalent number of bits precision. The output bit stream produced by all these methods has the same length of $2^{i\times N}$ bits, when multiplying i N-bit precision data [9]. Due to the fast converging property of LD bit streams, we use the LD deterministic approach to process bit streams in memory. However, the proposed idea is applicable to all deterministic methods.

Figure 1 shows an example of multiplying two input values, 1/4 and 3/4, using the LD deterministic method. With the LD method, the inputs are converted to independent bit streams by using different LD distributions [9]. Here, we use the algorithm proposed in [10] to determine the bit selection order for converting each binary input to the bit-stream format. The bit selection orders of [10] are determined based on the distribution of numbers in different Sobol sequences. The output bit stream of the example in Figure 1 is a 16-bit bit stream representing 3/16, the exact result expected for multiplication of the two inputs. In general, when multiplying two N-bit precision data, the full-precision output bit stream has a total length of 2^{2N} bits. This corresponds to a

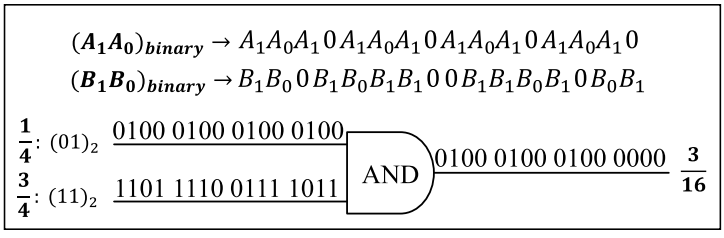

Figure 1. Example of accurate 2-bit precision multiplication using the LD deterministic method [9]. The inputs are converted to independent LD bit streams based on the bit-stream generation method proposed in [10].

total processing time of 2^{2N} clock cycles when producing one bit of the output bit stream at any cycle.

Comparator-based [6], [9], and multiplexer (MUX)-based [10] bit-stream generators are proposed in prior work to convert the data from binary to bit stream representation. The overhead cost of conversion and the latency of generating and processing bit streams make the conventional SC multiplier energy-inefficient compared to its binary counterpart. The large overhead of reading/storing data from/to memory further makes the conventional off-memory stochastic and binary multipliers inefficient compared to the emerging in-memory multipliers.

SC and memristors

Knag et al. [7] exploit the intrinsic non-deterministic properties of memristors to generate random stochastic bit streams in memory. They develop a hybrid system that consists of memristors integrated with CMOS-based stochastic circuits. Analog input data are converted to random bit streams by a stochastic group writing into the memristive memory. The computation is performed on the bit streams off-memory using CMOS logic and the output bit stream is written back to the memristive memory. In every write to the memristive memory, a new random bit stream is produced. The design in [7] eliminates the large overhead of off-memory stochastic bit-stream generation. Their bit-stream generation process, however, can be affected by variation and noise, and the computation is approximate.

A flow-based in-memory SC architecture is proposed in [8]. Their design exploits the flow of current through probabilistically switching memristive nanoswitches in high-density crossbars to perform stochastic computations. The data are represented using bit-vector stochastic streams of varying bit widths instead of traditional stochastic streams composed of individual bits. The crossbar computation performed in [8] is again approximate and probabilistic. The design cannot produce accurate results and must generate and process very long bit streams.

In this work, we propose a crossbar-compatible SC-based multiplier to perform deterministic and accurate multiplication in memory. We propose a new method to convert input binary data into deterministic bit streams and employ SC to multiply the data by ANDing the generated bit streams. Both the bit stream generation and the logical operation on the generated bit streams will be performed in memory.

Memristive IMC

Memristors are two-terminal electronic devices with variable resistance. This resistance depends on the amount and direction of the charge passed through the device in the past. For stateful IMC, we treat this resistance as the logical state, where the high and low resistances are considered, respectively, as logical zero and one. MAGIC [2] is a well-known stateful logic family proposed for IMC. It is fully compatible with the usual crossbar design and supports NOR, which can be used to implement any Boolean logic. Figure 2 shows how NOR logic operation can be executed within the memory in MAGIC by applying specific voltages [2] to the input(s) and output memristors. As shown in Figure 2 and the embedded truth tables, performing logical NOR on a negated version of two inputs (i.e., $\overline{\bar{A} + \bar{B}}$) is equivalent to performing logical AND on the original inputs (i.e., $A \cdot B$). We will exploit this logical property to implement AND operation in memory. Imani et al. [4] proposed a fixed-point MAGIC-based multiplication algorithm by serializing the addition of partial products in memory. An N-bit fixed-point multiplication with their method takes $15N^2 - 11N - 1$ cycles and $15N^2 - 9N - 1$ memristors. An improved method to perform fixed-point multiplication within memristive

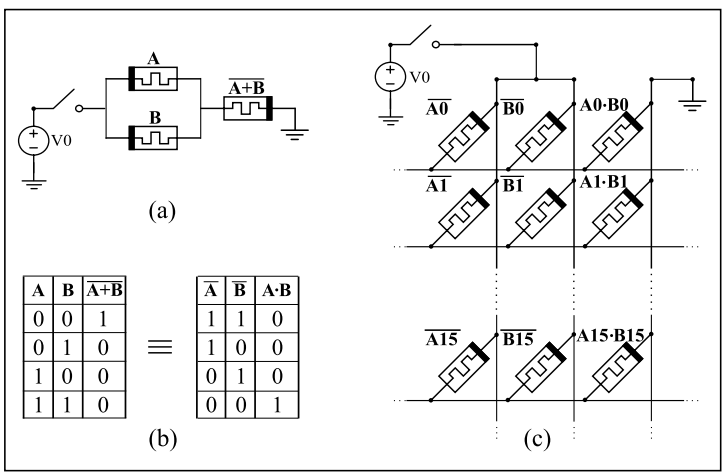

Figure 2. (a) Performing a MAGIC NOR operation within a memristive memory. For further detail on the execution of the NOR operation, the readers are referred to [2]. (b) NOR truth table. (c) Performing AND operation using MAGIC NOR within crossbar memristive memory array.

memory using MAGIC gates is proposed in [3]. To multiply two numbers they use the partial product multiplication algorithm and reuse the memristor cells during execution. A two-input full-precision multiplication (the output has twice the precision/length of the inputs) using this method needs $13N^2 - 14N + 6$ cycles and $20N - 5$ memristors. They also propose a limited-precision multiplication (the output has the same precision/length as the inputs) by generating and accumulating only the necessary partial products to produce the lower half (less significant bits) of the full-precision product. This improves latency by approximately 2×. The latency is reduced to $6.5N^2 - 7.5N - 2$ cycles while $19N - 19$ memristors are required. The limited-precision multiplication is especially useful for digital signal processing and fixed-point design of neural networks. More recently, Radakovits et al. [11] introduced a fast and low-cost full-precision in-memory multiplier, which performs two-input multiplication using $2N^2 + N + 2$ memristors in $\lceil \log_2 N \rceil (10N + 2) + 4N + 2$ cycles.

Proposed method

In this section, we discuss our proposed method of exact SC-based multiplication in memristive memory. We assume that the input data are already in memory in binary-radix format. We convert the data from binary to bit-stream representation in memory, process using stateful logic, and then convert the result back to binary format.

Binary to bit stream

Prior works exploited the probabilistic properties of memristors to generate random bit streams in memory [7], [8]. The bit streams generated by these methods suffer from random fluctuations and cannot produce accurate results. For accurate i-input multiplication, the input binary data must be converted to $i\ 2^{i \times N}$-bit independent bit streams [9]. With the LD deterministic method, the independence between bit streams is guaranteed by converting each input data based on a different LD sequence. We convert the data to LD bit streams by using the LD distributions proposed in [10].

Figure 3a shows the sub-computations of a 3-input 2-bit precision multiplication using the LD method. As can be seen, out of 64 operations only 27 operations can produce a non-zero output and contribute to the final result. This stems from the fact that the maximum value representable by a 2-bit precision data and the maximum result of multiplying three 2-bit data is 3/4 and 27/64, respectively. In the general case, in an i-input N-bit precision multiplication, $(2^N - 1)^i$ bitwise AND operations contribute to the output value. Our proposed in-memory multiplier only performs these operations. To achieve high-performance multiplication within memristive memory, we perform these bitwise operations in a parallel manner.

For multiplication, presented in the "Stochastic multiplication using MAGIC" section, we need the generated bit stream to be stored in a column (as opposed to a row). To this end, we use external CMOS switches to connect binary input memristors (e.g., A_j, B_j, C_j) to respective bit-stream memristors in different rows. A CMOS control circuitry controls the connection of switches. Since memristors are CMOS compatible and can be produced as back end of line (BEOL) [5], [11], these external switches can be placed below the memristor crossbar to avoid area overhead. Moreover, our synthesis results show that the overhead power and energy consumption of the control circuitry is negligible compared to the IMC operations of the multipliers themselves.

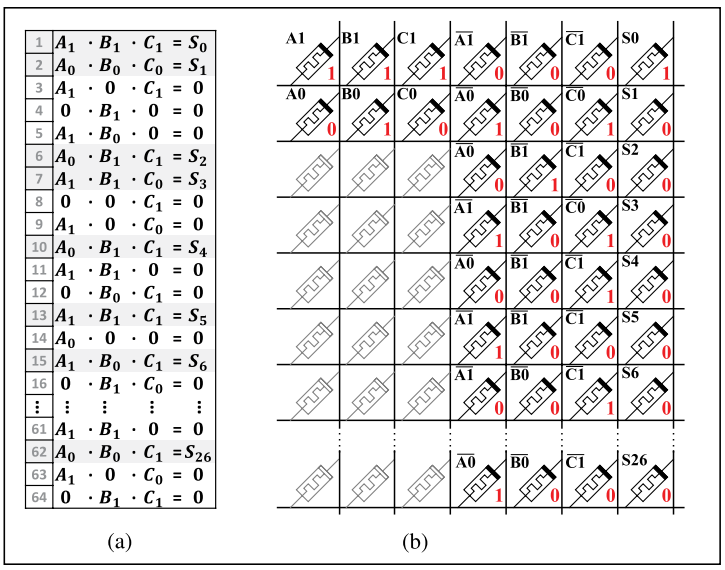

Figure 3. Example of multiplying three 2-bit precision data using the proposed method: (a) symbolic operations and (b) effective operations in memory. Inputs are *A* = 2/4, B = 3/4, and *C* = 2/4 in binary format, and the output is bit stream *S* representing 12/64. Only 27 out of 64 operations are performed in memory. The inputs are converted from binary to LD bit streams based on the LD distributions of [10].

To convert each input data, we first initialize $(2^N - 1)^i$ memristors in a column (e.g., the fourth column in Figure 3b), to a low resistance state (LRS) or logical value of one. For conversion, we apply V_0 to the negative terminal of the input binary memristors (e.g., A_j), which is connected to respective memristors in the bit-stream column. If A_j is storing a logical zero, i.e., it is in a high resistance state (HRS), it is virtually an open circuit. Thus, the connected memristors see no voltage and will not change their state. If A_j stores one, it is in LRS and acts as a virtual short circuit. Thus, all memristors connected to it see a V_0 across themselves. By selecting V_0 large enough, all respective memristors experience a state change from LRS to HRS. In other words, from logical one (their initial value) to logical zero. Therefore, at the end of the conversion operation, the bit-stream memristors corresponding to a binary input bit of one will have a logical value of zero, and vice versa (i.e., zero → one). We note that this representation is complementary to (i.e., it is the inverted version of) conventional bit-stream representation. However, this inversion—as we show later in this article—is to our advantage as it reduces the number of steps necessary to perform a multiplication.

Stochastic multiplication using MAGIC

We convert each N-bit binary data to a $(2^N - 1)^2$ bit bit stream for two-input exact (full precision) and to a $(2^N - 1)$ bit bit stream for limited-precision multiplication. The multiplication consists of a bitwise AND operation between the two operands. However, in MAGIC, which we have chosen for this work, the only operation compatible with crossbar memory is NOR. Therefore, we need to use an equivalency, namely

$$A \wedge B = \overline{\bar{A} \vee \bar{B}}. \quad (1)$$

As we see in (1), to perform AND in MAGIC, the input operands need to be inverted, followed by a NOR operation. Therefore, our proposed method has the advantage that by generating the bit streams already in their inverted form, as explained in the "Binary to bit stream" section, we save two steps (one for inversion of each operand). Hence, the proposed multiplication here consists of only one MAGIC NOR operation between the two bit-stream operands. To perform the multiplication, i.e., MAGIC NOR, the two operands need to be connected in a row as shown in Figure 2c. That is, for this operation, each corresponding bit of the two operands need to be in the same row, which is one of the reasons why bit streams are generated in columns (as opposed to rows). The proposed design can be extended to i-input multiplication by performing i-input MAGIC NOR on i bit-stream operands. Converting each operand needs one initialization and one execution cycle. The NOR operation also takes one initialization and one execution cycle. To decrease sneak paths, we perform these initializations in different cycles. This makes the total latency of i-input multiplication $2 \times (i + 1)$ cycles. Figure 3 shows an example of a 3-input 2-bit precision multiplication using the proposed method. We will show that this 3-input multiplication is executed in eight cycles.

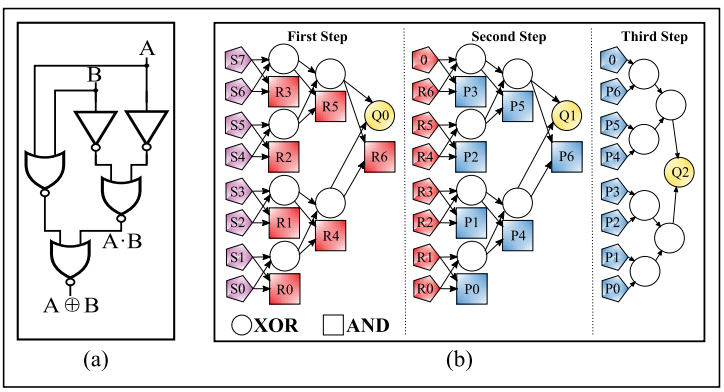

Figure 4. (a) XOR and AND operations using NOR gates. (b) Proposed algorithm for 8-bit bit stream (S7-S0) to 3-bit binary (Q2Q1Q0) conversion. Each square represents an AND operation and each circle represents an XOR operation.

Bit Stream to Binary

After performing multiplication using MAGIC, the output is in memory in the bit-stream format. The output bit stream can be preserved in memory in the current format for future bit-stream-based processing. However, if an output in binary format is desired, a final bit-stream-to-binary step is also needed. This can be done by counting the number of 1's in the bit stream by adding all the bits of the bit stream. We suggest two methods to convert the output bit stream to binary representation.

In-memory conversion

We propose a new algorithm for counting all the 1's of a bit stream in memory. Figure 4b depicts the proposed method for converting an 8-bit bit stream to a 3-bit binary data. The proposed algorithm consists of AND and XOR operations. As shown in Figure 4a, every pair of AND and XOR operations is implemented

with three NOR and two NOT MAGIC operations. We re-use memristors to minimize the number of required memristors in implementing this in-memory conversion. This algorithm can be easily extended to convert longer bit streams. It takes $4 \times (\log_2 L)^2$ cycles to count the number of 1's in a bit stream of length L. The two-input full-precision and the limited-precision multiplication require $0.5 \times (2^N - 1)^2 + N$ and $0.5 \times (2^N - 1) + N$ additional memristors, respectively, for in-memory conversion using this method.

Off-memory conversion

The output bit stream (e.g., bit stream S in Figure 3b) is read from the memory and its bits are summed using an off-memory combinational CMOS circuit. We described a sum function for adding L bits using Verilog HDL and let the synthesis tool find the best hardware design for summing those bits. The latency and hardware costs for conversion of output bit streams with this method are extracted from synthesis reports and used in the "Results and comparison" section for evaluation.

Results and comparison

Circuit-level simulations

For circuit-level evaluation of the proposed design, we implemented a 32×32 crossbar and necessary control signals in Cadence Virtuoso. For memristors, we used the Voltage-controlled ThrEshold Adaptive Memristor (VTEAM)[1] model. The values used for the parameters are $\{R_{on}, R_{off}, VT_{on}, VT_{off}, x_{on}, x_{off}, k_{on}, k_{off}, \alpha_{on}, \alpha_{off}\}$ = {1 kΩ, 300 kΩ, –1.5 V, 300 mV, 0 nm, 3 nm, –216.2 m/s, 0.091 m/s, 4, 4}.

Figure 5 shows the states of the memristors in the first two rows of the example shown in Figure 3b. At first, all memristors (except the binary memristors holding the input data) are in HRS. To convert each input we initialize the bit-stream memristors in the respective column to LRS using $V_{SET} = 2.08\,V$ (cycles 1, 3, and 5 for initializing bit streams of input A, B, and C, respectively). After initialization, we apply $V_0 = 1.48$V to binary memristors and GND to bit-stream memristors to generate the bit streams (cycles 2, 4, and 6). The output memristors are initialized in the next cycle and $V_0 = 1.08$V is applied to execute the NOR operations (cycles 7 and 8). Based on the LRS to HRS switching time of a memristor, 1ns was considered for time-length of each and every operation (i.e., voltage pulse-width is 1 ns).

[1] https://asic2.group/tools/memristor-models/

Comparison with in-memory binary multiplication

Table 1 compares the latency (number of processing cycles) and the area (number of memristors) of the proposed bit-stream-based multiplier with the prior in-memory fixed-point multiplication methods. As shown, the proposed multiplier is significantly faster than the prior in-memory binary methods by producing the output bit stream in only six cycles. In terms of the area too, the proposed method is more efficient (requires a smaller number of memristors) for $N < 5$ for the limited-precision case. Compared to the limited-precision design of [3] that produces the lower half (least significant bits), our method is more precise as it produces the higher half of the full-precision result. For larger Ns, other design considerations regarding the trade-off between memory and area should be taken into account. In general, for an i-input full-precision multiplication, $3 \times (2^N - 1)^i$ memristors are needed. If a binary output is desired, the additional latency and area of the bit-stream-to-binary step must also be considered.

The inherent fault tolerance of the proposed design can still be a winning proposition for larger Ns as the nonidealities of memristive technology can lead to the introduction of faults and noise into the memristive memory and in-memory calculations. The current accurate in-memory multiplication methods are all based on the conventional binary representation of data which makes them inherently more vulnerable to faults compared to the SC-based methods.

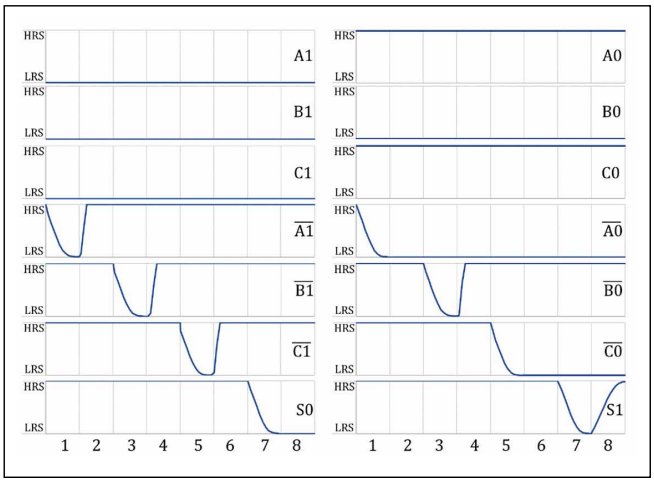

Figure 5. Simulation output of the first two rows of the crossbar in the example of Figure 3b.

Table 1. Latency and area of the two-input stateful *N*-bit precision in-memory multiplication.

Methods		Latency (Cycles)	Area (# of memristors)
Full Precision	Haj-Ali *et al.* [3]	$13N^2 - 14N + 6$	$20N - 5$
	Imani *et al.* [4]	$15N^2 - 11N - 1$	$15N^2 - 9N - 1$
	Radakovits *et al.* [11]	$\lceil \log_2 N \rceil (10N + 2) + 4N + 2$	$2N^2 + N + 2$
	This work	6	$3 \times (2^N - 1)^2$
Limited Precision	Haj-Ali *et al.* [3]	$6.5N^2 - 7.5N - 2$	$19N - 19$
	This work	6	$3 \times (2^N - 1)$

Table 2. Energy consumption results (in *pJ*): comparison of the proposed method and off-memory exact SC-based multiplication.

Design Method	Limited Precision							Full Precision						
	N=2	3	4	5	6	7	8	2	3	4	5	6	7	8
This work (no bit-stream-to-binary conversion)	0.026	0.061	0.13	0.27	0.55	1.12	2.24	0.08	0.43	1.98	8.47	35	142	573
This work (+ in-memory bit-stream-to-binary)	0.035	0.09	0.21	0.47	1.01	2.17	4.62	0.12	0.77	3.1	19	87	386	1687
This work (+ off-memory bit-stream-to-binary)	7	15	29	56	108	210	413	20	86	366	1,529	6,263	25,166	101,391
Off-Memory Exact SC-based Multiplication	38	40	44	53	76	124	234	54	76	133	694	3,092	13,919	62,541

We note that the power consumption of various IMC units heavily depends on the memristive technology used for the implementation (or the model representing it) and its respective necessary setup. Therefore, to have a fair comparison with prior work, they need to be implemented using the same technology or simulated using the same model and model parameters. Moreover, most related works in the literature do not report any power or energy consumption numbers at all. Due to these reasons, we could not compare our work with others in that regard.

Comparison with off-memory stochastic multiplication

For an off-memory SC-based multiplication of *N*-bit binary data, the data must be first read from the memory and be converted from binary to bit-stream representation. The clock division method [9] has the lowest hardware cost among the SoA deterministic methods of SC. We implemented the clock division circuit of [9] to convert the data and generate bit streams. Multiplication is performed by ANDing the generated bit streams. The output is converted back to binary format using a binary counter and is stored in memory. We described this off-memory design using Verilog HDL and synthesized it using the Synopsys Design Compiler v2018.06-SP2 with the 45-nm NCSU-FreePDK[2] gate library.

[2]https://www.eda.ncsu.edu/wiki/FreePDK

Table 2 compares the energy consumption of the proposed in-memory multiplier with that of the implemented off-memory SC multiplier for data precision of two to eight bits. For the cases that include off-memory processing, we assume the data is read from or written to a memristive memory. We use the per-bit energy consumption reported in [12] to calculate the total energy of the read and write operations. As shown in Table 2, for all different *N*s, the proposed in-memory design with in-memory bit-stream-to-binary conversion provides significantly lower energy consumption than the off-memory exact SC-based multiplier. For off-memory bit-stream-to-binary conversion, the size of the data read from the memory plays a crucial role. Our work is more energy efficient for small *N*s. However, for larger *N*s the traditional CMOS off-memory SC consumes less energy. The reason is the size of the data read from the memory, which grows exponentially in the case of in-memory multiplication off-memory conversion (bit streams are read), compared to the traditional off-memory SC computation (where binary data are read), giving the latter an edge.

THIS WORK PROPOSES the first in-memory architecture to execute exact multiplication based on SC. The multiplication results are as accurate as the results from fixed-point binary multiplication. The proposed method significantly reduces the energy consumption compared to the SoA off-memory exact SC-based

multiplier. Compared to prior in-memory fixed-point multiplication methods, the proposed design provides faster results. For smaller Ns, the area is comparable too. For larger Ns, the area is the price for the gained speed. The proposed limited-precision multiplication is particularly interesting for applications such as neural networks and certain signal processing algorithms since it is not only faster but also more precise and for the usually targeted Ns, area efficient. If outputs are desired in binary format, a bit-stream-to-binary conversion overhead should be considered too. We propose an efficient crossbar compatible method for this conversion. The inherent noise-tolerance of bit-stream processing makes the proposed design further advantageous for memristive-based computation compared to its binary counterparts. We leave the study of this aspect for future works. ■

Acknowledgments

This work was supported in part by the Louisiana Board of Regents Support Fund under Grant LEQSF(2020-23)-RD-A-26 and in part by the National Science Foundation under Grant 2019511. A preliminary version of this article appeared in [1].

References

[1] M. R. Alam, M. H. Najafi, and N. TaheriNejad, "Exact in-memory multiplication based on deterministic stochastic computing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

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

[3] A. Haj-Ali et al., "Efficient algorithms for in-memory fixed point multiplication using MAGIC," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.

[4] M. Imani, S. Gupta, and T. Rosing, "Ultra-efficient processing in-memory for data intensive applications," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–6.

[5] 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.

[6] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 8, pp. 1515–1531, Aug. 2018.

[7] P. Knag, W. Lu, and Z. Zhang, "A native stochastic computing architecture enabled by memristors," *IEEE Trans. Nanotechnol.*, vol. 13, no. 2, pp. 283–293, Mar. 2014.

[8] S. Raj, D. Chakraborty, and S. K. Jha, "In-memory flow-based stochastic computing on memristor crossbars using bit-vector stochastic streams," in *Proc. IEEE 17th Int. Conf. Nanotechnol. (IEEE-NANO)*, Jul. 2017, pp. 855–860.

[9] M. H. Najafi et al., "Performing stochastic computation deterministically," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 12, pp. 2925–2938, Dec. 2019.

[10] S. Asadi and M. H. Najafi, "Late breaking results: LDFSM: A low-cost bit-stream generator for low-discrepancy stochastic computing," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–2.

[11] D. Radakovits et al., "A memristive multiplier using semi-serial IMPLY-based adder," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 5, pp. 1495–1506, May 2020.

[12] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnol.*, vol. 8, no. 1, p. 13, 2013.

Mohsen Riahi Alam is currently a Research Scholar with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA, USA. His research interests include in-memory computation, stochastic computing, low power and energy-efficient VLSI design, and embedded systems. Alam has an MSc in computer architecture from the University of Tehran, Tehran, Iran.

M. Hassan Najafi is currently an Assistant Professor with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA, USA. His research interests include stochastic and approximate computing, unary processing, in-memory computing, and machine learning. Najafi has a PhD in electrical engineering from the University of Minnesota, Twin Cities, Minneapolis, MN, USA.

Nima TaheriNejad is currently a "Universitätassistant" with the TU Wien (formerly known as Vienna University of Technology as well), Vienna, Austria, where his areas of research interests includes self-awareness in resource-constrained cyber–physical (embedded) systems, computing in memory, memristor-based circuit and systems, and healthcare. TaheriNejad has a PhD in electrical and computer engineering from The University of British Columbia (UBC), Vancouver, BC, Canada. He is a member of ACM and the IEEE Circuits and Systems Society as well as the IEEE Engineering in Medicine and Biology Society.

■ Direct questions and comments about this article to Mohsen Riahi Alam, School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70504 USA; mohsen.riahi-alam@louisiana.edu.