Exact In-Memory Multiplication Based on Deterministic Stochastic Computing

Mohsen Riahi Alam*, M. Hassan Najafi*, and Nima TaheriNejad†
* University of Louisiana at Lafayette, Louisiana, USA
† Technische Universität Wien, Vienna, Austria
{mohsen.riahi-alam, najafi}@louisiana.edu, nima.taherinejad@tuwien.ac.at

***Abstract*—Memristors offer the ability to both store and process data in memory, eliminating the overhead of data transfer between memory and processing unit. For data-intensive applications, developing efficient in-memory computing methods is under investigation. Stochastic computing (SC), a paradigm offering simple execution of complex operations, has been used for reliable and efficient multiplication of data in-memory. Current SC-based in-memory methods are incapable of producing accurate results. This work, to the best of our knowledge, develops the first accurate SC-based in-memory multiplier. For logical operations, we use Memristor-Aided Logic (MAGIC), and to generate bit-streams, we propose a novel method, which takes advantage of the intrinsic properties of memristors. The proposed design improves the speed and reduces the memory usage and energy consumption compared to the State-of-the-Art (SoA) accurate in-memory fixed-point and off-memory SC multipliers.**

I. Introduction

Transferring data between memory and processing unit 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 [1]. 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 [1]. 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 that execute the same instruction on multiple data in parallel can benefit greatly from MAGIC.

Multiplication is a common but complex operation in many data intensive applications [4]. In-memory methods for fixed-point binary multiplication using MAGIC have been previously investigated [3], [5]. These methods are faster and more energy efficient than conventional off-memory binary multipliers. However, memristive technology is a not a fully mature technology yet, in particular compared to Complementary Metal-Oxide Semiconductor (CMOS) technology [6]. It suffers from considerable process variations and nonidealities which affects its performance [6], [7]. These nonidealities can lead to introduction of faults and noise into the memristive memory and in-memory calculations. The inherent vulnerability of binary methods to noise (i.e., to bit flips) which poses a challenge to the reliability of the system should not be forgotten either.

Stochastic Computing (SC) [8] is a re-emerging computing paradigm that offers simple execution of complex arithmetic functions. The paradigm is more robust against fault and noise compared to the conventional binary computing [9], [10]. Multiplication, as a complex operation in conventional binary designs, can be implemented using simple standard `AND` gates in SC [8], [11]. Input data is converted from binary to independent (uncorrelated) bit-streams and connected to the inputs of the `AND` gate. Logical 1s are produced at the output of the gate with a probability equal to the product of the two input data. Prior works have exploited the intrinsic non-deterministic properties of memristors to generate random stochastic bit-streams in memory [10], [12], [13]. 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 prior 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. To this end, we exploit the recent progress in deterministic and accurate SC computation with bit-streams [14]–[17]. The proposed multiplication method benefits from the complementary advantages of both SC and memristive IMC in producing exact results. 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 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 energy consumption compared to the SoA accurate off-memory SC-based multiplication techniques.

II. Background

A. Deterministic Computation with Stochastic Bit-Streams

In SC, data is represented by streams of 0s and 1s. Independent of the length, the ratio of the number of 1s to the length of the bit-stream determines the data value. For example, 0100 and 11000000 both represent 0.25 in stochastic domain. Compared to conventional binary radix, this form of representation is more noise-tolerant as all bits have equal weight [11], [18]. A single bit-flip, regardless of its position in the bit-stream, introduces a least significant bit error.

Deterministic approaches have been proposed recently to perform accurate computation using stochastic bit-streams. Clock dividing bit-streams [14], using bit-streams with relatively prime lengths [16], and rotation of bit-streams [14] are examples of these methods. Compared to conventional SC, with these deterministic methods, the bit-stream length is reduced by a factor of approximately ($1/2^N$) where N is the equivalent number of bits precision [14]. Due to simplicity of understanding the clock division method, we select this method to present the idea of this work. The proposed design, however, is applicable to all the aforementioned deterministic methods.

Fig. 1 shows an example of multiplying two input values, 1/4 and 3/4, using the clock division method. These input values can be precisely represented using two 4-bit streams (1000 and 1110, respectively). As can be seen, the first bit-stream (1000) is repeated four times while the second bit-stream (1110) is clock divided by four (each bit is repeated

978-1-7281-3320-1/20/$31.00 ©2020 IEEE

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on January 07,2021 at 10:16:13 UTC from IEEE Xplore. Restrictions apply.

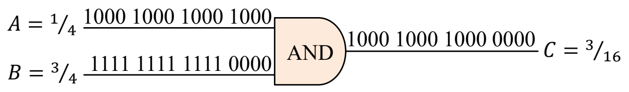


Fig. 1. Example of multiplication using clock division method.

four times). The output bit-stream, produced by ANDing the input bit-streams, is a 16-bit stream representing 3/16, the exact result expected for multiplication of the two inputs.

In general, for multiplication of any two input data using the clock division method, the first bit-stream is repeated for the length of the second bit-stream and the second bit-stream is clock divided by the length of the first one. The length of the output bit-stream will be the product of the length of the input bit-streams. Hence, when multiplying two N-bit precision data, each represented using a 2^N-bit stream, the output bit-stream has a total length of 2^{2N} bits.

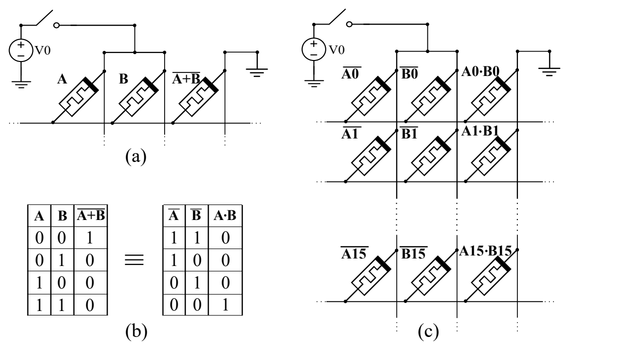
B. Stochastic Computing and Memristors

Knag *et al.* [10] exploit the intrinsic non-deterministic properties of memristors to generate random stochastic bit-streams in memory. They develop a hybrid system consisted of memristors integrated with CMOS-based stochastic circuits. Input data in analog format 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. The design in [10] 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 [13]. The design exploits the flow of current through probabilistically-switching memristive nanoswitches in high-density crossbars to perform stochastic computations. The data is represented using bit-vector stochastic streams of varying bit-widths instead of traditional stochastic streams composed of individual bits. The crossbar computation performed in [13] is again approximate and probabilistic. The design cannot produce accurate results and must process very long bit-streams.

In this work, we propose a crossbar-compatible SC-based design to perform efficient 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 bit-streams will be performed in memory.

C. Memristive In-Memory Computation

Memristors are two-terminal electronic devices with a variable resistance. This resistance depends on the amount and direction of the charge passed through the device in the past [19]. For IMC, we treat this resistance as the logical state, where the high and low resistance are considered, respectively, as logical zero and one. Material Implication (IMPLY) [20], [21] and Memristor Aided LoGIC (MAGIC) [2] are two well-known logic families proposed for IMC [22], [23]. MAGIC has certain advantages in that it is fully compatible with the usual crossbar design, requires a lower number of voltages, and supports NOR, which can be used to implement any Boolean logic. Fig. 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 the figure and the embedded truth tables, performing logical NOR on 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.



A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

≡

$\bar{A}$	$\bar{B}$	A·B
1	1	0
1	0	0
0	1	0
0	0	1

Fig. 2. (a) Performing a MAGIC NOR operation within a memristive memory. (b) NOR truth table (c) performing AND operation using MAGIC NOR.

An algorithm to execute an N-bit addition in memory using MAGIC was proposed in [24]. An N-bit addition with this method takes $12N+1$ cycles. Imani *et al.* [5] proposed a fixed-point MAGIC-based multiplication algorithm by serializing 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 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 full-precision multiplication (the output has twice the precision/length of the inputs) with this method takes $13N^2 - 14N + 6$ 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 the latency by approximately $2\times$. 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.

III. Proposed Method

In this section, we discuss our proposed method of exact SC-based multiplication in memristive memory. We assume that the input data is already in memory in binary-radix format.

A. Binary to Bit-Stream

To convert the data to deterministic and accurate bit-streams, we propose the topology shown in Fig. 3. In this topology, we first initialize 2^N memristors in a column (e.g., column E in Figs. 3a-b), to Low Resistance State (LRS) or logical value of '1'. For conversion, we apply V_0 to the positive terminal of the input binary memristors, A_i, which is connected to 2^i memristors in the bit-stream column, in this example, column E. If A_i is storing a logical '0', i.e., it is in High Resistance State (HRS), it is virtually open circuit and thus the 2^i connected memristors see no voltage and will not change their state. If A_i stores '1', 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 2^i memristors experience a state change from LRS to HRS. In other words, from logical '1' (their initial value) to logical '0'. Therefore, at the end of conversion operation, for each '1' ('0') in the binary input A_i, we will have 2^i logical '0's ('1's) in the generated bit-stream. 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 the paper– is to our advantage as it reduces the number of steps necessary to perform multiplication.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on January 07,2021 at 10:16:13 UTC from IEEE Xplore. Restrictions apply.

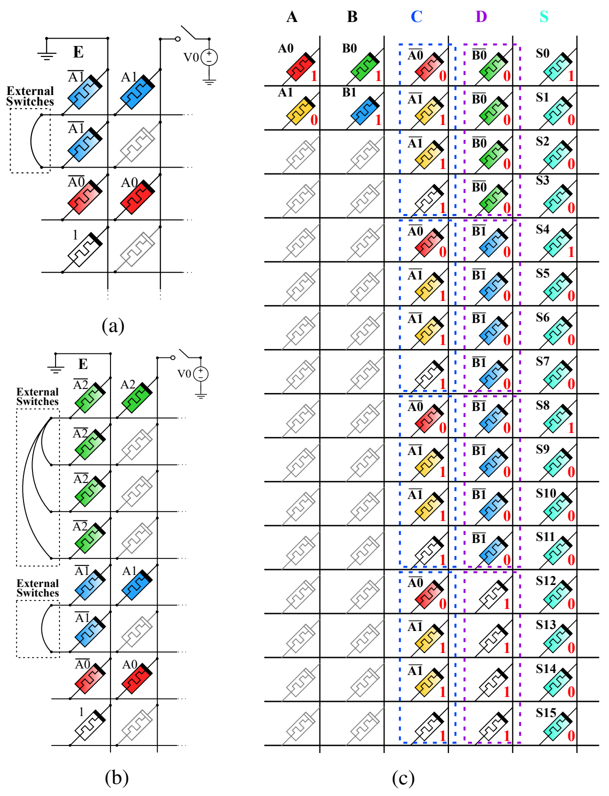


Fig. 3. (a) 2-bit precision (b) 3-bit precision binary to bit-stream conversion in memory. (c) Example of 2-input bit-stream-based multiplication using the proposed method. Inputs are $A = 1/4$ and $B = 3/4$ in binary format, and the output is bit-stream S representing $3/16$. The dashed boxes show the bit repetition pattern based on the clock division method.

For multiplication, presented in Section III-B, 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 input memristors to respective memristors in different rows. Figs. 3a-b depict examples of how a 2-bit and a 3-bit precision binary data is converted to bit-stream representation in memory. As shown in these examples, with the discussed method an N-bit binary data is converted to a $(2^N - 1)$-bit stream stored in $2^N - 1$ memristors. For an exact bit-stream-based multiplication of data, however, we need a 2^N-bit bit-stream representation for any N-bit binary data. An additional one bit, 2^N-th bit, is always 0 as the maximum value presentable by an N-bit binary data is $2^N - 1$. An additional memristor is therefore initialized to logical '1' (inverted version of '0') and used as the required 2^N-th bit.

Another reason for using external switches in the conversion is the fact that for SC multiplication, the input bit-streams need to be uncorrelated [11], [14]. That is, the distribution of '1's and '0's for each operand must be independent of the other operand. Using external switches allows us to connect the binary input memristors (e.g., A_i, B_i) to their corresponding bit-stream memristors in an uncorrelated fashion. This configuration (uncorrelated connection of switches) can be generated using a CMOS control circuitry. The deterministic method (e.g., clock division) determines the distribution and control algorithm/configuration. External switches and even more complex circuitry (such as sense amplifier and buffers) have been previously used for similar purposes [3], [24]. Since memristors are CMOS compatible and can be produced as Back End Of Line (BEOL) [25]–[27], the external switches used in the bit-stream generation can be placed below the memristor crossbar to avoid area overhead.

B. Stochastic Multiplication using MAGIC

To perform multiplication, each N-bit binary data must be converted to a 2^{2N} or 2^N bit-stream for exact (full) and limited precision multiplication, respectively. The multiplication is consisted of a bit-wise 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,

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

As we see in Equation (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 deterministic bit-streams already in their inverted form, as explained in Section III-A, we save two steps (one for inversion of each operand). Hence, the deterministic bit-stream 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 Fig. 2(c). 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). Fig. 3(c) shows an example of a 2-bit input full-precision multiplication using the proposed method. The bit-stream operands in this example are generated based on the clock division concept discussed in Section II-A and the binary-to-bit-stream method discussed in Section III-A.

C. Bit-Stream to Binary

After performing bit-stream-based multiplication using MAGIC, the output is in memory in bit-stream format. This bit-stream can be preserved in memory in 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 1s in the bit-stream through adding all the bits of the bit-stream. We suggest two methods for this step:

1) In-memory conversion: we propose a new algorithm for counting all the '1's of the bit-stream in memory. Fig. 4(b) depicts the proposed method for converting an 8-bit bit-stream to a 3-bit binary data. The proposed algorithm consists of XOR and AND operations. As shown in Fig. 4(a), the output of logical ANDing two input bits can be used in performing logical XOR on the same inputs. This lets us implement every pair of AND and XOR gates with three NOR and two NOT operations. We re-use memristors to minimize the number of required memristors in implementing this in-memory converter. 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. This means $4 \times (2N)^2$ cycles for a full-precision and $4 \times N^2$ cycles for a limited-precision output. The full-precision and the limited-precision multiplication require $0.5 \times 2^{2N} + N$ and $0.5 \times 2^N + N$ additional memristors, respectively, for in-memory conversion using this method.

2) Off-memory conversion: the output bit-stream (e.g., bit-stream S in Fig. 3(c)) can be read from the memory and its bits 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 Section IV for evaluation.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on January 07,2021 at 10:16:13 UTC from IEEE Xplore. Restrictions apply.

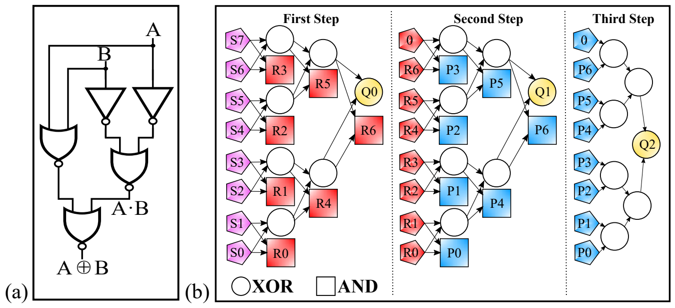


Fig. 4. (a) XOR and AND operations using NOR gates (b) Proposed algorithm for 8-bit bit-stream (S7-S0) to 3-bit binary (Q2Q1Q0) conversion. Each square presents an AND operation and each circle represents an XOR operation.

IV. Results and Comparison

A. Circuit-Level Simulations

For circuit-level evaluation of the proposed design, we implemented a 16×16 crossbar and necessary control signals in LTSpice. For memristors, we used the Voltage-controlled ThrEshold Adaptive Memristor (VTEAM) model [28] ported to LTspice [29], [30]. The values used for the parameters are $\{R_{on}, R_{off}, VT_{on}, VT_{off}, x_{on}, x_{off}, k_{on}, k_{off},\ \alpha_{on}, \alpha_{off}\}$= {1kΩ, 300kΩ, -40mV, 300mV, 0nm, 3nm, -100m/sec, 0.091m/sec, 4, 4}. At first, we initialized all memristors to LRS using $V_{SET} = 1.9V$. In each step of the operations, we apply $V_0 = 0.5V$ to respective memristors involved in the operation and disable all other memristors. Based on the LRS to HRS switching time of a memristor, 4ps was considered for each operation step. We evaluated all cases of 2-bit precision multiplication and verified the functionality of the design.

B. Comparison with In-Memory Binary Multiplication

Table I 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 methods by producing the output bit-stream in only three cycles. In terms of 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. 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 introduction of faults and noise into the memristive memory and in-memory calculations. The current in-memory fixed-point multiplication methods are all based on the conventional binary radix representation of data which makes them inherently more vulnerable to faults compared to the SC-based methods.

C. Comparison with Off-Memory Stochastic Multiplication

Clock division is the preferred method for exact bit-stream-based multiplication as it provides minimum area×delay [17].

TABLE I
LATENCY AND AREA OF IN-MEMORY MULTIPLICATION ALGORITHMS

Methods		Latency (Cycles)	Area (# of memristors)
Full Precision	Haj-Ali *et al.* [3]	$13N^2 - 14N + 6$	$20N - 5$
	Imani *et al.* [5]	$15N^2 - 11N - 1$	$15N^2 - 9N - 1$
	This work	3	3×2^{2N}
Limited Precision	Haj-Ali *et al.* [3]	$6.5N^2 - 7.5N - 2$	$19N - 19$
	This work	3	3×2^N

For an off-memory bit-stream-based multiplication of data, the data must be first read from the memory and be converted from binary to bit-stream representation. We implemented the clock division circuit discussed in [14] 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 implemented this off-memory design using Verilog HDL and synthesized using the Synopsys Design Compiler v2018.06-SP2 with the 45nm NCSU-FreePDK library.

Table II compares the energy consumption of the proposed exact in-memory multiplier with that of the implemented off-memory bit-stream-based multiplier. For the cases that include off-memory processing, we assume the data is read/written from/to a memristive memory. We used the per bit energy consumption reported in [31] to calculate the total energy of the read and write operations. As shown in Table II, for all different Ns, 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, however, size of the data read from the memory plays a crucial role. Our work is more energy efficient for small Ns, however, for larger Ns traditional CMOS off-memory SC consumes less energy. The reason is the size of the data read from the memory, which grows exponentially (bit-streams are read) in the case of in-memory multiplication off-memory conversion, compared to traditional off-memory SC computation (where binary data are read), giving the latter an edge.

V. Discussion and Conclusions

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 lower or 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, 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.

TABLE II
ENERGY CONSUMPTION COMPARISON (pJ) 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	N=2	3	4	5	6	7	8
This work (no bit-stream-to-binary conversion)	0.04	0.07	0.14	0.28	0.56	1.13	2.26	0.14	0.56	2.3	9.0	36	145	578
This work (+ in-memory bit-stream-to-binary)	0.05	0.11	0.23	0.49	1.03	2.20	4.64	0.23	1.03	3.7	21	90	393	1702
This work (+ off-memory bit-stream-to-binary)	9	17	31	58	110	212	416	31	110	416	1628	6462	25561	102184
Off-Memory Exact SC-based Multiplication	38	40	44	53	76	124	234	54	76	133	694	3092	13919	62541

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on January 07,2021 at 10:16:13 UTC from IEEE Xplore. Restrictions apply.

References

[1] Mohammed A. Zidan, John Paul Strachan, and Wei D. Lu. The future of electronics based on memristive systems. *Nature electronics*, 1:22–29, 2018.

[2] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser. MAGIC—memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, Nov 2014.

[3] A. Haj-Ali, R. Ben-Hur, N. Wald, and S. Kvatinsky. Efficient algorithms for in-memory fixed point multiplication using MAGIC. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, May 2018.

[4] D. Radakovits, N. Taherinejad, M. Cai, T. Delaroche, and S. Mirabbasi. A memristive multiplier using semi-serial imply-based adder. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1–12, 2020.

[5] Mohsen Imani, Saransh Gupta, and Tajana Rosing. Ultra-Efficient Processing In-Memory for Data Intensive Applications. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, New York, NY, USA, 2017. ACM.

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

[7] Nimrod Wald and Shahar Kvatinsky. Understanding the influence of device, circuit and environmental variations on real processing in memristive memory using memristor aided logic. *Microelectronics Journal*, 2019.

[8] B.R. Gaines. Stochastic computing systems. In *Advances in Information Systems Science*, pages 37–172. Springer US, 1969.

[9] Weikang Qian, Xin Li, M.D. Riedel, K. Bazargan, and D.J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *Computers, IEEE Trans. on*, 60(1):93–105, Jan 2011.

[10] P. Knag, W. Lu, and Z. Zhang. A Native Stochastic Computing Architecture Enabled by Memristors. *IEEE Trans. on Nanotechnology*, 13(2), March 2014.

[11] Armin Alaghi and John P. Hayes. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):92:1–92:19, 2013.

[12] S. Gaba, P. Knag, Z. Zhang, and W. Lu. Memristive devices for stochastic computing. In *2014 IEEE Intern. Symp. on Circuits and Systems (ISCAS)*, June 2014.

[13] S. Raj, D. Chakraborty, and S. K. Jha. In-memory flow-based stochastic computing on memristor crossbars using bit-vector stochastic streams. In *2017 IEEE 17th Intern. Conf. on Nanotechnology (IEEE-NANO)*, pages 855–860, July 2017.

[14] Devon Jenson and Marc Riedel. A Deterministic Approach to Stochastic Computation. In *the 35th ICCAD*, pages 102:1–102:8, New York, NY, USA, 2016.

[15] M. Hassan Najafi, David J. Lilja, and Marc Riedel. Deterministic Methods for Stochastic Computing using Low-Discrepancy Sequences. In *Proceedings of the 37th International Conference on Computer-Aided Design*, ICCAD '18, 2018.

[16] M. Hassan Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani. Time-Encoded Values for Highly Efficient Stochastic Circuits. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 25(5):1644–1657, May 2017.

[17] M. Hassan Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel. Performing Stochastic Computation Deterministically. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2925–2938, Dec 2019.

[18] M. Hassan Najafi and M. E. Salehi. A Fast Fault-Tolerant Architecture for Sauvola Local Image Thresholding Algorithm Using Stochastic Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(2):808–812, Feb 2016.

[19] Leon Chua. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 18(5):507–519, 1971.

[20] Eero Lehtonen and Mika Laiho. Stateful implication logic with memristors. In *Proceedings of the 2009 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 33–36. IEEE Computer Society, 2009.

[21] Julien Borghetti, Gregory S. Snider, Philip J. Kuekes, J. Joshua Yang, Duncan R. Stewart, and R. Stanley Williams. Memristive switches enable stateful logic operations via material implications. *Nature*, 464:873–876, 2010.

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

[23] S. Ganjeheizadeh Rohani, N. Taherinejad, and D. Radakovits. A semiparallel full-adder in imply logic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(1):297–301, Jan 2020.

[24] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky. Logic design within memristive memories using memristor-aided logic (MAGIC). *IEEE Transactions on Nanotechnology*, 15(4):635–650, July 2016.

[25] Neuro-bit. Bio inspired technologies llc, 2017.

[26] Knowm. Knowm inc., 2017.

[27] Peter Clarke. *TSMC to offer embedded ReRAM in 2019*. eeNews, 2017.

[28] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny. Vteam: A general model for voltage-controlled memristors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(8):786–790, Aug 2015.

[29] https://www.ict.tuwien.ac.at/staff/taherinejad/projects/memristor/files/vteam.sub, March 2018.

[30] N. TaheriNejad, T. Delaroche, D. Radakovits, and S. Mirabbasi. A semi-serial topology for compact and fast IMPLY-based memristive full adders. In *2019 IEEE New Circuits and Systems symposium (NewCAS)*, pages 1–5, 2019.

[31] J Joshua Yang, Dmitri B Strukov, and Duncan R Stewart. Memristive devices for computing. *Nature nanotechnology*, 8(1):13, 2013.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on January 07,2021 at 10:16:13 UTC from IEEE Xplore. Restrictions apply.