# Stochastic Computing in Beyond Von-Neumann Era: Processing Bit-Streams in Memristive Memory

Mohsen Riahi Alam , *Student Member, IEEE*, M. Hassan Najaf , *Member, IEEE*, Nima TaheriNejad , *Member, IEEE*, Mohsen Imani, *Member, IEEE*, and Raju Gottumukkala , *Member, IEEE*

***Abstract*—Stochastic Computing (SC) is an alternative computing paradigm that promises high robustness to noise and outstanding area- and power-eff ciency compared to traditional binary. It also enables the design of fully parallel and scalable computations. Despite its advantage, SC suffers from long latency and high energy consumption compared to conventional binary computing, especially with current CMOS technology. The cost of conversion between binary and stochastic representation takes a signif cant cost with CMOS circuits. In-Memory Computation (IMC) is introduced to accelerate Big Data applications by removing the data movement between memory and processing units, and by providing massive parallelism. In this work, we explore the efforts in employing IMC for fast and energy-eff cient SC system design. We specially focus on memristors as an emerging technology that promises eff - cient memory and computation beyond CMOS. We discuss the potentials and challenges for realizing eff cient SC systems in memory.**

***Index Terms*—Stochastic computing, in-memory computing, resistive RAM, emerging computing methods, fault tolerant systems.**

## I. Introduction

STOCHASTIC Computing (SC) [1], [2] is re-emerging as a promising alternative to the traditional binary computing. SC offers extremely simple execution of complex arithmetic operations (e.g., multiplication using bit-wise AND) and high tolerance to noise and variation in data and computation logic. The paradigm has been used for low-cost and noise-tolerant implementation of a wide range of applications from image [3] and signal processing to coding [4], sorting [5], and artif cial neural networks [6], to name a few. Recent work has shown that SC is not limited to approximate computations [2]. SC bit-streams and logic circuits can be structured to process data deterministically and produce completely accurate results. This recent advancement in the f eld has been one of the key contributors to their re-emergence as a promising unconventional computing paradigm.

Despite its advantages, SC faces certain limitations, especially in currently dominant CMOS technology. Data conversion between binary and stochastic representation is costly with CMOS logic, consuming more than 80% of the total system cost [7]. Energy consumption is another major challenge. Often long bit-streams must be processed for acceptable accuracy. Processing long bit-streams serially results in high latency, and high latency translates to high energy consumption, often higher than the energy consumption of binary counterparts. Parallel processing reduces the latency but increases the area and power consumption, which in the end results in high energy consumption again. Reading and writing stochastic bit-streams from and to memory further cost-prohibitive latency and energy, especially for today's big data applications. So bit-streams are f rst converted back to binary format before storing them to memory. Even though compared to Stochastic Computing (SC), binary data may be easier and more eff cient for the transfer between the memory and the processing unit, with the exponential growth of the data that needs to be processed, this data movement has been proven to be a major bottleneck [8]. To tackle this challenge, Processing in Memory (PIM) or In-Memory Computation (IMC) is introduced. IMC refers to processing data near its source, i.e., where it is stored: memory. This is in contrast to the traditional Von-Neumann architecture, in which processor and memory are two separate entities, located far apart, and the data needs to travel between the two. Thus, IMC accelerates applications by removing this data movement. It also provides massive parallelism. As mentioned before, size of the data and its movement is one of the major challenges that SC faces too and hence it can considerably benef t from IMC. IMC signif cantly increases the competitiveness of SC, rendering it a serious contender among unconventional computing paradigms.

This work explores the efforts in combining the complementary advantages of (memristive) IMC and SC in developing fast and energy-eff cient computing systems. We discuss the state-of-the-art approaches for converting data between binary and bit-stream representations and executing SC operations in memory. We then discuss the potentials and challenges for realizing eff cient SC systems in memory.

Manuscript received February 7, 2022; accepted March 20, 2022. Date of publication March 24, 2022; date of current version May 3, 2022. This work was supported in part by the National Science Foundation (NSF) under Grant 2127780 and Grant 2019511; in part by the Semiconductor Research Corporation (SRC) Task under Grant 2988.001; in part by the Department of the Navy, Off ce of Naval Research, under Grant N00014-21-1-2225 and Grant N00014-22-1-2067; in part by the Air Force Off ce of Scientif c Research; in part by the Louisiana Board of Regents Support Fund under Grant LEQSF(2020-23)-RD-A-26; and in part by Cisco. This brief was recommended by Associate Editor P. López-Martínez. *(Corresponding author: Mohsen Riahi Alam.)*

Mohsen Riahi Alam and M. Hassan Najaf are with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70503 USA (e-mail: mohsen.riahi-alam@louisiana.edu; najaf @louisiana.edu).

Raju Gottumukkala is with the Department of Mechanical Engineering, University of Louisiana at Lafayette, Lafayette, LA 70503 USA (e-mail: raju@louisiana.edu).

Nima TaheriNejad is with the Institute of Computer Technology, Technische Universität Wien, 1040 Vienna, Austria (e-mail: nima.taherinejad@tuwien.ac.at).

Mohsen Imani is with the Department of Computer Science, University of California at Irvine, Irvine, CA 92697 USA (e-mail: m.imani@uci.edu).

Color versions of one or more f gures in this article are available at https://doi.org/10.1109/TCSII.2022.3161995.

Digital Object Identif er 10.1109/TCSII.2022.3161995

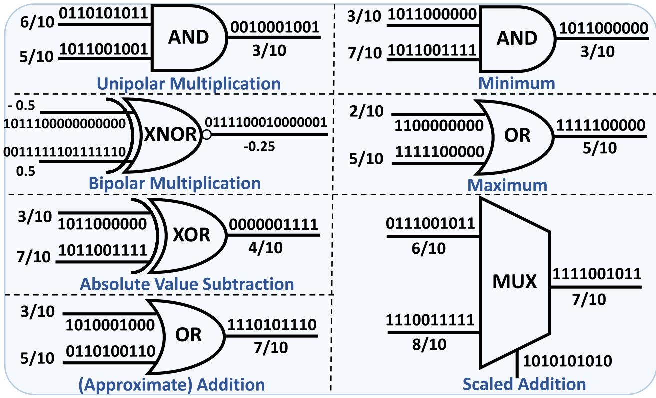


Fig. 1. Basic SC Operations: Multiplication, (approximate) addition, and scaled addition use independent bit-streams, whereas minimum, maximum, and absolute value subtraction use correlated bit-streams.

## II. Fundamentals

### A. Stochastic Computing

In SC, data are encoded with uniform[1] bit-streams with the value determined by the probability of observing a '1'. When representing a real-valued number, $x$, with a *unipolar* and a *bipolar* stochastic encoding, each bit has the probability of $x$ and $(x+1)/2$ of being '1', respectively. Unlike the positional binary representation, in a stochastic bit-stream all bits have the same weight. This provides tolerance to noise since a single bit flip results in only a least significant bit error. In a common form, which we call stochastic or random bit-stream, the '1's are distributed randomly in the bit-stream: e.g., 10100011. In a so-called *unary bit-stream*, first all '1's appear followed by all '0's or vice versa, e.g., 11110000. Both of these bit-streams represent the real value of 0.5. Two bit-streams are *correlated* if they have a high overlap between the positions of 1s. They are *uncorrelated* or independent if the '1's are distributed independently. Some SC operations such as multiplication require independent inputs. Some operations such as minimum (min) and maximum (max) value functions can be realized using a single logic gate with correlated inputs [5]. Min and max SC designs insensitive to correlation are also proposed [3] but they are more costly and not as accurate. Fig. 1 shows some of the basic SC operations. Conventionally, a random bit-stream is generated by comparing a random value $r$ from a random number generator to the target value $x$. A '1' is generated if $r \leq x$. The cost of generating bit-streams with this approach is relatively high, consuming more than 80% of the total hardware cost of a typical SC system [7].

### B. Memristive In-Memory Computing

Memristive technology is one of the promising technologies for IMC. Memristors support both storage [9]–[13] and logic [14]–[17]. Single-level and multi-level memristor cells are available. A single-level memristor cell has two resistance levels (low resistance state (LRS) representing logical '1' and high resistance state (HRS) representing logical '0') and can represent one bit of data per cell. A multi-level memristor cell has one or more middle resistance states between the LRS and HRS. These states can represent different logical values. By applying a stimulus (voltage or current) to memristors, it is possible to induce logical operation among memristors. One of the most efficient types of such an operation is *stateful* logic. In this type, the resistance of the input memristor prior to the operation represents the logical input value and the resistance of the output memristor after the operation represents the logical output [14]. Among different memristive-based IMC techniques, stateful logics such as IMPLY [15], MAGIC [16], FELIX [17], and SIXOR [18] are of the most efficient solutions. For these, no access to the world outside the array (e.g., read or write) is necessary. Such operations can be natively executed within memory array with a high degree of parallelism. So parallel architectures such as SC designs can benefit greatly from such IMC logic.

[1]"Uniform" here refers to the property of having bits with the same weight, which is a significant attribute of the stochastic representation [2].

## III. Stochastic Computing in Memory

Despite finding a large body of work in the literature in employing SC for low-cost and noise-tolerant implementation of different applications, only few works are dedicated to the in-memory implementation of SC designs. Knag *et al.* [19] developed a hybrid system consisting of memristors integrated with CMOS-based stochastic circuits. The bit-streams are generated in memory, but the computations are performed off-memory using CMOS stochastic circuits. Finally, the output bit-streams are written back to the memristive memory. Expanding the effort in [19], the authors in [20] exploited the well-known switching stochasticity of probabilistic Conductive Bridging RAM (CBRAM) devices to efficiently generate stochastic bit-streams in memory. They use the generated bit-streams to perform deep learning parameter optimization using a hybrid CMOS-memristor stochastic processor. A flow-based in-memory SC architecture is proposed in [21]. The design exploits the flow of current through probabilistically-switching memristive nano switches in high-density crossbars to perform SC. They represent data using bit-vector stochastic streams of varying bit-widths instead of traditional stochastic streams composed of individual bits. A physics-based probabilistic switching model for Resistive Random Access Memory (ReRAM) stochastic bit-stream generation is developed in [22]. Gupta *et al.* [23] developed SCRIMP, an architecture for SC acceleration with ReRAM in-memory processing. Riahi Alam *et al.* [24], [25] developed an exact (completely accurate) method for SC multiplication in memristive memory. To this end, they propose a method for deterministic and accurate binary to bit-stream conversion in memory. In-memory architectures for sorting unary bit-streams and median filtering of unary data are proposed in [26]. Sun *et al.* [27] employ unary coding, implemented with multi-level memristor cells, for weight representation in a ReRAM-based neural network (NN) design. They apply unary coding to tolerate the device resistance variations and design accurate ReRAM-based NN accelerators.

A summary of these works and their key features can be seen in Table I. This provides an overview of the literature, whereas, in the following subsections, we dive into more details of the literature and existing approaches to SC in memory. In particular, we look into how SC bit-streams are generated, converted to binary, and processed in memory.

### A. Bit-Stream Generation in Memory

In prior work, the intrinsic non-deterministic properties of memristors have been exploited to generate random bit-streams in memory. In [19], input data in an analog format are directly converted to random bit-streams by a stochastic group writing into the memristive memory. Stochastic bit-streams are generated by applying programming pulses with variable

TABLE I
AN OVERVIEW OF STOCHASTIC COMPUTING IN MEMORY LITERATURE

| Year | Design | Input | Bit-Stream | Logic | Memristor | Bit-Stream-to-Bin. | In-Mem. SC Arith. | Applications |
|---|---|---|---|---|---|---|---|---|
| 2014 | Knag et al. [19] | Analog | Random | CMOS | Single-Level | - | - | Gradient Descent Solver, K-means clustering |
| 2017 | Raj et al. [21] | Analog | Random | In-Memory | Single-Level | - | Add, Multiplication | - |
| 2019 | Zhao et al. [22] | Analog | Random | CMOS | Single-Level | - | - | Image Processing (Robert Edge detection) |
| 2020 | Gupta et al. [23] | Analog | Random | In-Memory | Single-Level | Off-Memory | Add, Mul, Exp, Log, .. | Image Proecessing (Sobel, Robert, ... ) |
| 2021 | Lammie et al. [20] | Analog | Random | CMOS | Single-Level | - | - | Deep Learning Parameter Optimization |
| 2021 | Riahi Alam et al. [24] | Binary | Deter. LD | In-Memory | Single-Level | In-Mem., Off-Mem. | Multiplication | - |
| 2021 | Riahi Alam et al. [26] | Unary | Deter. Unary | In-Memory | Single-Level | Off-Memory | Min, Max | Sorting, Median Filtering |
| 2021 | Sun et al. [27] | Binary | Deter. Unary | CMOS | Multi-Level | - | - | Neural Networks (ResNet18, Vgg16) |

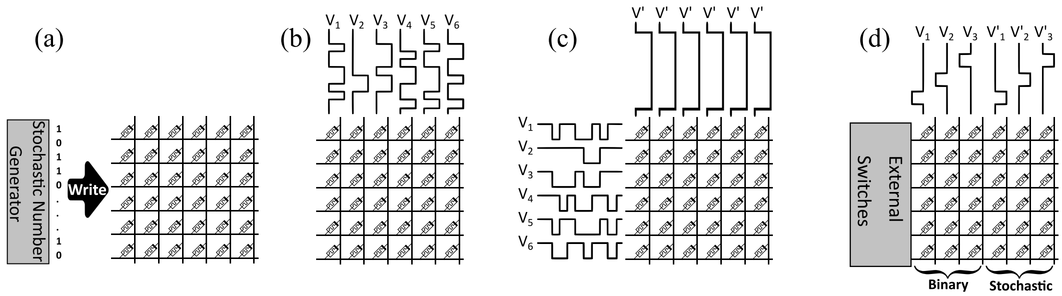


Fig. 2. Different Bit-Stream Generation Approaches for IMC: a) Off-Memory Generation, b) Group Write Proposed in [19], c) SCRIMP Row-Parallel Generation [23] d) Deterministic Method of [24].

pulse widths to memristor cells. In every write to the memristive memory, a new bit-stream statistically independent of previous bit-streams is generated. This approach is called a *native* approach for SC, as it eliminates the extra conversion steps between binary and bit-streams, accepts analog inputs directly, and takes advantage of the properties of memristors to provide randomness in the bit-streams. In comparison, the conventional SC requires analog-to-digital conversion to accept analog inputs, and the randomness must be created algorithmically using purely CMOS circuits [19]. SCRIMP [23] exploits the stochastic nature of ReRAM devices to propose a new stochastic bit-stream generation scheme. SCRIMP generates bit-streams in parallel over multiple rows. These in-memory methods eliminate the large overhead of off-memory CMOS-based bit-stream generation [7]. They, however, suffer from random f uctuations error. The bit-stream generation and hence the computations are all approximate and probabilistic.

Riahi Alam *et al.* [25] proposed an in-memory method to convert binary data into *deterministic* bit-streams. Assuming that the data are already in memristive memory in binary format, they connect the binary memristors in a column to bit-stream memristors in a different column. For an accurate conversion, an $n$-bit binary data stored in $n$ memristors are connected to $2^n$ memristors. For operations such as SC multiplication that independent bit-streams are needed, the control circuitry implements a different distribution for each bit-stream [24]. Their approach is able to generate fast-converging low-discrepancy (LD) bit-streams [2], [28]. LD bit-streams quickly and monotonically converge to the target value, producing acceptable results with much shorter bit-streams. The bit-streams generated with this method are free of random f uctuations error and can accurately represent input data. Fig. 2 compares different bit-stream generation approaches for IMC.

Sun *et al.* [27] propose a unary coding method with multi-level cell devices to decrease the deviation of the stored value in the presence of resistance variations. By utilizing multi-level memristor cells, they can store multiple bits on each memristor and effectively reduce the number of needed memristors compared to the case of using single-level cells.

## B. Bit-Stream to Binary Conversion

The output bit-streams from stochastic computations can be preserved in memory in the bit-stream format for a future bit-stream-based processing. However, if output in binary format is desired, a bit-stream-to-binary step is performed. This can be done by counting the number of 1s in the bit-stream by adding all the bits of the bit-stream. Reading the bit-streams from memory for summation using an off-memory CMOS circuit can be expensive, especially for long bit-streams. To avoid reading long bit-streams and off-memory conversion, Riahi Alam *et al.* in [25] propose an algorithm for counting all the '1's of the bit-stream in memory. The algorithm consists of `AND` and `XOR` operations. The method of [25] takes $4 \times (\log_2 L)^2$ cycles to count the number of '1's in a bit-stream of length $L$.

## C. Arithmetic Operations

*Multiplication:* Multiplication in SC involves bit-wise `AND` on unipolar and bit-wise `XNOR` on bipolar bit-streams. SCRIMP [23] executes SC multiplication by implementing an implication-based `AND` and `XNOR` logic in crossbar memory. Both `AND` and `XNOR` operations in their method take two cycles (using other fundamental in-memory logic operations).

Riahi Alam *et al.* [25] propose a crossbar-compatible SC-based design to perform accurate multiplication in memory. For accurate multiplication, the distribution of '1's and '0's for each operand must be independent of the other operand. They provide this independence by connecting the binary input memristors to their corresponding bit-stream memristors in an uncorrelated fashion based on the clock division method [2]. For a full-precision multiplication, bit-streams of $2^{2N}$ bits and for a limited precision multiplication bit-streams of $2^N$ bits are generated. To execute `AND` operation, MAGIC `NOR` is performed on inverted inputs. In an optimized implementation [24], the binary data are converted to independent LD bit-streams using the LD distribution proposed in [29]. The multiplication method of [24] reduces the number of `AND` operations by only executing the operations that can produce a non-zero output (that contributes to the f nal result). For 2-input full-precision multiplication, this reduces the size of bit-streams to $(2^n - 1)^2$ bits. Compared to the off-memory CMOS-based SC approach, they report 50× and 37× reductions in energy consumption for the 8-bit limited- and full-precision in-memory SC multiplication [24]. The multiplication method of [24] and [25] can be extended to *i*-input multiplication by performing *i*-input MAGIC NOR on *i* bit-stream operands. The total latency of *i*-input multiplication is $2 \times (i + 1)$ cycles. The result is completely accurate, free of random fluctuations and correlation errors.

*Addition:* An OR-based, a MUX-based, and a count-based stochastic additions are implemented in SCRIMP [23]. They generate OR of *n* bits in a single cycle. The operation is executed in parallel for the entire bit-stream and takes only one cycle. For MUX-based addition, they first stochastically select one of MUX inputs for each bit position. Then, the selected input bit is read using a memory sense amplifier and stored in an output register. The MUX-based addition takes one cycle to generate one output bit, taking *L* cycles for *L*-bit output bit-stream. They also propose a parallel count (PC)-based addition in memory. Every cycle, one input bit-stream is read out by the sense amplifier and sent to counters. This is done for *i* inputs sequentially, consuming *i* cycles. In the end, counters store the number of ones.

*Subtraction:* In the case of subtraction, the subtrahend can be first inverted using an in-memory NOT operation. Then, any addition technique can be used. Alternatively, subtraction can be realized by bit-wise XOR if the input bit-streams are highly correlated. In-memory XOR can be performed by three NOR and two NOT operations, as elaborated in [25]. It can also be implemented using FELIX [17] by executing single cycle OR and NAND in crossbar memory. To be faster, SIXOR [18] can be used, which implements XOR in a single cycle.

*Minimum and Maximum:* Bit-wise logical AND on two *correlated* bit-streams gives the min of the two bit-streams, and bit-wise logical OR, the max. Authors in [26] propose MAGIC-based in-memory designs for min and max operations on unary bit-streams. The approach, however, is applicable to any correlated bit-streams. The AND operation (min) is realized by first inverting the bit-streams through NOT and then performing bit-wise NOR on the inverted bit-streams in a total of three cycles. The OR operation (max) is achieved in two cycles by first bit-wise NOR on the input bit-streams and then NOT on the outputs of the NOR operations.

*Other Arithmetic Operations:* Trigonometric, logarithmic, and exponential functions are supported in SCRIMP using the Maclaurin Series expansion [30]. This expansion approximates the functions using a series of multiplications and additions.

## IV. POTENTIALS

*Low-Cost Bit-Stream Generation:* Taking advantage of the inherent properties of memristive memories to generate stochastic bit-streams in memory addresses a long-time key bottleneck in the cost-efficient design of SC systems. By accepting analog inputs, the extra conversion step between analog and digital binary can also be avoided [19].

*Robust to Soft-Error:* Memristive technology is an emerging technology still in evolution, facing practical challenges [31]. The fabrication process of memristors is not fully mature yet; ReRAMs suffer from endurance challenges, stochastic behavior, and resistance variations. Due to the changes to the physical characteristics of a memristor cell, faults such as resistance drift and retention failure are also observed in ReRAM [13], [32]. These faults increase the soft error rate in ReRAMs [32], [33]. The traditional reliability techniques for soft errors are placed in the memory access interface. Overcoming the soft errors is essential in IMC as they propagate within operations without the data ever being read to be recovered by the traditional techniques. The traditional binary encoding is inherently more vulnerable to soft-errors compared to uniform stochastic representation. SC representation and operations are inherently tolerant of soft-errors as any bit flip leads to only a least significant bit error. Improving the reliability of memristive IMC is an open challenge, and SC is proving itself as a promising solution for this issue [27].

*Massive Parallelism:* Memristive crossbar arrays provide massive bit-level parallelism. This is in particular suitable for SC systems with many bit-level operations. By applying the same voltage along bitlines/wordlines, we may induce a logical operation in all rows/columns of the crossbar at the same time. Further, crossbar arrays can be dynamically divided into multiple partitions to support simultaneous but different in-row (in-column) operations in the same row (column) [17], [26]. This allows performing various arithmetic operations on data with a very short latency (in only a few cycles), which otherwise need considerably more cycles to execute [24], [26].

## V. CHALLENGES

*Correlation Manipulation:* Prior works such as [19] and [23] take advantage of the inherent stochastic properties of memristive devices to generate random independent bit-streams. But not all SC designs operate on independent bit-streams. Single logic-gate design of SC operations such as min, max, and subtraction needs correlated bit-streams. Because none of the current in-memory bit-stream generation techniques can generate correlated bit-streams, the bit-streams must first be generated off-memory with CMOS-based techniques of generating correlated bit-streams and then be written. But this approach leads to significant latency and energy consumption, particularly when large bit-streams are to be written into memory. A more significant challenge is when the data are in memory in bit-stream format but are not correlated. They must first be converted to binary format in memory, read from memory, converted from binary to correlated bit-streams, and finally written back into memory. A different approach is to read the bit-streams from memory and make them correlated using CMOS-based correlation manipulation techniques [34], [35]. But this approach, in addition to the overheads of writing bit-streams into memory consumes a high latency and energy overhead for reading the bit-streams from memory.

A similar challenge exists when the bit-streams stored in memory are correlated (or their correlation status is unknown), but independent bit-streams are needed for the SC operation. Either in-memory correlation manipulation techniques must be developed to directly make the bit-streams independent in memory, or similar to what we discussed above, the bit-streams must be sent out of memory to manage their correlation with CMOS-based correlation manipulation techniques.

*Accuracy and Limitation of Memory Arrays:* Representing numbers with high precision and performing computation with high accuracy need long stochastic bit-streams. The length increases exponentially with precision. To execute accurate *n*-bit precision min and max operations bit-steams of $2^n$ bits [26] and to perform exact multiplication on two *n*-bit data, bit-streams of $(2^n - 1)^2$ bits [24] are needed. Limitation in terms of the number of memristors restricts the length of bit-streams and so the scalability of the in-memory SC designs.

In a fully parallel design approach, the size of the memory array defines an upper limit for the maximum length of bit-streams. In such a design, bit-streams with lengths longer than the number of rows (columns) can be supported by splitting each bit-stream into multiple shorter sub-bit-streams, storing each sub-bit-stream in a different column (row), and executing the operations in parallel on the sub-bit-streams. This approach processes the data with reduced latency as the primary objective. A different approach is to perform operations on the sub-bit-streams in a serial manner by re-using the memristors. This approach reduces the area (number of used memristors) at the cost of additional latency. In this case, after processing each pair of sub-bit-streams, the result is saved, and a new pair of sub-bit-streams is processed. Assuming that each bit-stream is split into $M$ sub-bit-streams, the number of processing cycles to process each pair of input data increases by a factor of $M$. Some additional cycles are also needed for data movement related to splitting the bit-stream. Combining the parallel and the serial approach is also possible for further trade-offs between area and delay. These approaches increase the range of supported data widths but incur a more complicated implementation and partition management [26].

Another solution to the long length issue is to use multi-level memristor cells [27], [36]. These cells have more than two resistance levels and can represent multi-bit data values per cell. In general, a memristor with $2^k$ resistance levels can represent data of $k$ bits per cell. Using this type of memristors can reduce the bit-stream length by a factor of $1/(2^k-1)$ [36].

*Sequential Circuits:* Sequential SC circuits including finite-state-machine-based approaches [37] also exist and are used to implement complex arithmetic functions. Among these we can name SC division, one of the four basic arithmetic operations, that is implemented in prior works with sequential CMOS-based circuits [35], [38]. These circuits are sequential in nature and it is not clear how to convert them to parallel operations for efficient execution within memory.

## REFERENCES


[1] 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.

[2] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing stochastic computation deterministically," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 12, pp. 2925–2938, Dec. 2019.

[3] P. Li, D. Lilja, W. Qian, K. Bazargan, and M. Riedel, "Computation on stochastic bit streams digital image processing case studies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 3, pp. 449–462, Mar. 2014.

[4] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," *Electron. Lett.*, vol. 39, no. 3, pp. 299–301, 2003.

[5] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan, "Low-cost sorting network circuits using unary processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 8, pp. 1471–1480, Aug. 2018.

[6] S. Lee, H. Sim, J. Choi, and J. Lee, "Successive log quantization for cost-efficient neural networks using stochastic computing," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

[7] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 93–105, Jan. 2011.

[8] M. A. Zidan, Y. Jeong, J. H. Shin, C. Du, Z. Zhang, and W. D. Lu, "Field-programmable crossbar array (FPCA) for reconfigurable computing," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 4, pp. 698–710, Oct.–Dec. 2018.

[9] H. Kim, M. P. Sah, C. Yang, and L. O. Chua, "Memristor-based multilevel memory," in *Proc. CNNA*, 2010, pp. 1–6.

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

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

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

[13] D. Radakovits and N. TaheriNejad, "Implementation and characterization of a memristive memory system," in *Proc. IEEE 32nd Can. Conf. Electr. Comput. Eng. (CCECE)*, 2019, pp. 1–4.

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

[15] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'Memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, pp. 873–876, Apr. 2010.

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

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

[18] 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.

[19] 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.

[20] C. Lammie, J. K. Eshraghian, W. D. Lu, and M. R. Azghadi, "Memristive stochastic computing for deep learning parameter optimization," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 5, pp. 1650–1654, May 2021.

[21] 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)*, 2017, pp. 855–860.

[22] Y. Zhao *et al.*, "A physics-based model of RRAM probabilistic switching for generating stable and accurate stochastic bit-streams," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, 2019, p. 32.4.1.

[23] S. Gupta *et al.*, "SCRIMP: A general stochastic computing architecture using ReRAM in-memory processing," in *Proc. Design Autom. Test Europe Conf. Exhibition (DATE)*, 2020, pp. 1598–1601.

[24] M. Riahi Alam, M. H. Najafi, and N. TaheriNejad, "Exact stochastic computing multiplication in memristive memory," *IEEE Design Test*, vol. 38, no. 6, pp. 36–43, Dec. 2021.

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

[26] M. Riahi Alam, M. H. Najafi, and N. TaheriNejad, "Sorting in memristive memory," *ACM J. Emerg. Technol. Comput. Syst.*, Jan. 2022, doi: 10.1145/3517181.

[27] Y. Sun *et al.*, "Unary coding and variation-aware optimal mapping scheme for reliable ReRAM-based neuromorphic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 12, pp. 2495–2507, Dec. 2021.

[28] S. Liu and J. Han, "Energy efficient stochastic computing with sobol sequences," in *Proc. DATE*, 2017, pp. 650–653.

[29] 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)*, 2020, pp. 1–2.

[30] K. K. Parhi and Y. Liu, "Computing arithmetic functions using stochastic logic by series expansion," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 44–59, Jan.–Mar. 2019.

[31] N. TaheriNejad and D. Radakovits, "From behavioral design of memristive circuits and systems to physical implementations," *IEEE Circuit Syst. Mag.*, vol. 19, no. 4, pp. 6–18, 4th Quart., 2019.

[32] D. Radakovits and N. Taherinejad, "Behavioral leakage and IntEr-cycle variability emulator model for ReRAMs (BELIEVER)," 2021, *arXiv:2103.04179*.

[33] S. Swami and K. Mohanram, "Reliable nonvolatile memories: Techniques and measures," *IEEE Design Test*, vol. 34, no. 3, pp. 31–41, Jun. 2017.

[34] V. T. Lee, A. Alaghi, and L. Ceze, "Correlation manipulating circuits for stochastic computing," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2018, pp. 1417–1422.

[35] S. Asadi, M. H. Najafi, and M. Imani, "CORLD: In-stream correlation manipulation for low-discrepancy stochastic computing," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–9.

[36] C. Ma, Y. Sun, W. Qian, Z. Meng, R. Yang, and L. Jiang, "Go unary: A novel synapse coding and mapping scheme for reliable ReRAM-based neuromorphic computing," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2020, pp. 1432–1437.

[37] M. H. Najafi, P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel, "A reconfigurable architecture with sequential logic-based stochastic computing," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, p. 57, 2017.

[38] T.-H. Chen and J. P. Hayes, "Design of division circuits for stochastic computing," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, 2016, pp. 116–121.