Received 2 December 2022; revised 23 March 2023; accepted 5 April 2023.
Date of publication 11 April 2023; date of current version 29 May 2023.

Digital Object Identifier 10.1109/JXCDC.2023.3266136

A Stochastic Computing Scheme of Embedding Random Bit Generation and Processing in Computational Random Access Memory (SC-CRAM)

BRANDON R. ZINK, YANG LV, MASOUD ZABIHI, HÜSREV CILASUN, SACHIN S. SAPATNEKAR (Fellow, IEEE), ULYA R. KARPUZCU (Member, IEEE), MARC D. RIEDEL (Senior Member, IEEE), and JIAN-PING WANG (Fellow, IEEE)

Electrical and Computer Engineering Department, University of Minnesota Twin Cities, Minneapolis, MN 55455 USA

CORRESPONDING AUTHOR: J.-P. WANG (jpwang@umn.edu)

This work was supported in part by the Center for Probabilistic Spin Logic for Low-Energy Boolean and Non-Boolean Computing (CAPSL), one of the Nanoelectronic Computing Research (nCORE) centers as task 2759.001; in part by the Semiconductor Research Corporation (SRC) Program through the National Science Foundation (NSF) under Grant 1739635; in part by the Spintronic Materials for Advanced Information Technologies (SMART), one of the seven centers of nCORE, a Semiconductor Research Corporation Program through the National Institute of Standards and Technology (NIST); in part by the SMART through the University of Minnesota (UMN) Materials Research Science Engineering Center (MRSEC) Program under Award DMR-1420013; in part by the Applications and Systems driven Center for Energy-Efficient Integrated NanoTechnologies (ASCENT), one of six centers of Joint University Microelectronics Program (JUMP), a Semiconductor Research Corporation Program through Microelectronics Advanced Research Corporation (MARCO) and Defense Advanced Research Projects Agency (DARPA); in part by DARPA through "Advanced Magnetic Tunnel Junctions (MTJs) for Computation in and Near Random Access Memory" under Grant HR001117S0056-FP-042; in part by the NIST; and in part by the DARPA Non-Volatile Logic Program, NSF Scalable Parallelism in the Extreme (SPX), under Grant 1725420.

This article has supplementary downloadable material available at https://doi.org/10.1109/JXCDC.2023.3266136, provided by the authors.

ABSTRACT Stochastic computing (SC) has emerged as a promising solution for performing complex functions on large amounts of data to meet future computing demands. However, the hardware needed to generate random bit-streams using conventional CMOS-based technologies drastically increases the area and delay cost. Area costs can be reduced using spintronics-based random number generators (RNGs), and however, this will not alleviate the delay costs since stochastic bit generation is still performed separately from the computation. In this article, we present an SC method of embedding stochastic bit generation and processing in a computational random access memory (CRAM) array, which we refer to as SC-CRAM. We demonstrate that SC-CRAM is a resilient and low-cost method for image processing, Bayesian inference systems, and Bayesian belief networks.

INDEX TERMS Bayesian systems, computational random access memory (CRAM), magnetic tunnel junction (MTJ), neuromorphic computing, stochastic computing (SC).

I. INTRODUCTION

Conventional computing schemes where data are encoded in deterministic binary bits have several challenges in meeting future demands in neuromorphic computing and other novel applications [1], [2], [3]. These unconventional computing schemes often require processing complex functions on large amounts of data and, therefore, one of the key challenges is the large circuit area and computation delay required for circuits based on conventional CMOS technology [4], [5]. In addition, the devices become highly sensitive to thermal noise as transistor size scaling trends toward nanometer-size dimensions [6]. Alternative methods of data encoding and alternative hardware implementations should be investigated to overcome these shortcomings.

Stochastic computing (SC) is a probabilistic scheme that has been studied intensively in recent years [7]. SC is an attractive method for several unconventional computing applications for two key reasons. One is that it is immune to the effects of thermal noise and random bit-flips, which allows for smaller device sizes [7], [8], [9], [10], [11]. The second is that complex functions that are common in numerous neuromorphic computing tasks, such as hyperbolic tangent, exponential functions, and square root, can be performed efficiently in SC using a small number of

logic gates [7], [9], [12], [13], [14]. Previous studies have demonstrated that SC-based circuits can achieve above 99% accuracy rates in image recognition tasks at noise injection rates (percentage of bit-flips) above 30%, whereas the same tasks processed in conventional computing schemes showed large error rates at noise injection rates below 10% [8].

One of the major shortcomings of SC is the hardware area costs required for stochastic bit-stream generation (SBG). CMOS-based random number generators (RNGs) are typically linear shift feedback registers (LSFRs) [15] or ring oscillators [16]. When these circuits are used in SC-based networks, the hardware for SBG can consume up to 80% of the total circuit area and 80% of the total energy consumption [8]. Therefore, despite the low circuit area required for computation in SC, there may be little to zero reductions in the total circuit area and total energy consumption. Furthermore, LSFRs and ring oscillators can only generate pseudorandom numbers rather than true-random numbers, and therefore, the circuit complexity needs to increase to improve the quality of randomness in the stochastic bit-streams. One attractive solution to overcome these shortcomings of SC is to replace CMOS-based RNGs with spintronics-based RNGs [17], [18], [19], [20], [21], [22]. This is because a single magnetic tunnel junction (MTJ) can be used as a true RNG (TRNG) with a tunable probability [23], [24], [25], [26]. While these solutions reduce the area and energy costs for SBG, the hardware and energy consumption for SBG still make up a majority of the total circuit area and total energy consumption. Furthermore, implementing spintronic-based RNGs in SC does not reduce the overall delay costs since SBG steps are still performed separately from computation steps. Recent experimental studies have shown that the overall computation delay can be reduced in spintronics-based TRNGs by using MTJs with superparamagnetic free layers [27], [28]. However, these devices are incompatible for methods of generating random bits with synchronized clock cycles.

In this study, we present an SC scheme that uses MTJ-based hardware where SBG and computation steps are completely embedded. Our method exploits the intrinsic stochasticity of MTJs for SBG and utilizes the computational random access memory (CRAM) array to efficiently implement various SC tasks. Previous studies have demonstrated that true in-memory computing can be achieved in the CRAM architecture, thus eliminating circuit area costs and energy consumption for transferring data between memory and computation circuits [29], [30], [31], [32]. However, performing neuromorphic applications that require processing large amounts of data in conventional CRAM (conv-CRAM) will still suffer large area and energy costs as well as noise sensitivity since the data are still encoded in binary bits. Our solution overcomes these shortcomings by implementing SC within the CRAM array, which we refer to as SC-CRAM. While conv-CRAM embeds memory and logic arrays, SC-CRAM embeds SBG and computation arrays. We demonstrate the effectiveness of SC-CRAM

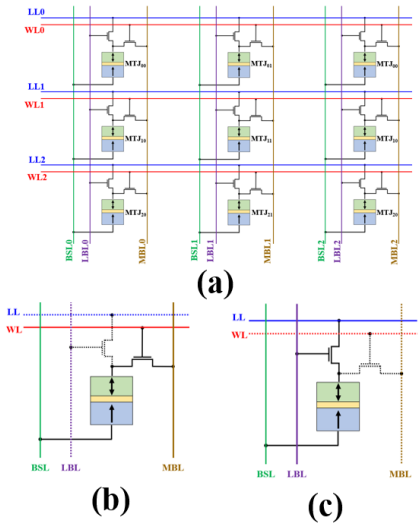

FIGURE 1. **(a) Circuit diagram of the CRAM array and illustration of (b) memory and (c) logic modes in CRAM.**

TABLE 1. **Output preset state and bias voltage criteria for logic operations in CRAM.**

Gate	Output Preset	Bias voltage criteria (in terms of output current)
NOT	0	$J_{OUT} > J_C$ if A = 0
BUFFER	1	otherwise $J_{OUT} < J_C$
AND	1	$J_{OUT} < J_C$ if [A,B] = [1,1],
NAND	0	otherwise $J_{OUT} > J_C$
OR	1	$J_{OUT} > J_C$ if [A,B] = [0,0],
NOR	0	otherwise $J_{OUT} < J_C$
MAJ3	1	$J_{OUT} < J_C$ if [A,B,S] = [1,1,1], [1,1,0], [1,0,1], or [0,1,1]
$\overline{MAJ3}$	0	otherwise $J_{OUT} > J_C$

* *Output preset state of '1' means that the output MTJ is initialized to the AP-state.*

in four neuromorphic applications: local image thresholding, Bayesian inference for object location, Bayesian belief network for heart disaster prediction, and kernel density estimation.

Previous studies have proposed methods of generating stochastic bit-streams using logic-in-memory hardware. Knag et al. [33] proposed a method where analog input data were converted to a series of programming pulses with variable pulsewidths and applied to an array of memristive memory cells. The method proposed by Gupta et al. [34] exploited the stochastic behavior of resistive random-access memory (ReRAM) devices to generate stochastic bit-streams in parallel over multiple rows. Riahi Alam et al. [35] took a slightly different approach where logic-in-memory architectures were used to convert binary data into deterministic bit-streams. The method presented in this article has one key advantage over previously proposed methods in that it allows for separate perturb and logic steps so that analog inputs can be converted directly into stochastic bit-streams. Furthermore, the method proposed in [35] requires additional control circuitry to implement different distributions for each bit-stream. This is not required in our design since the spintronic

devices are inherently stochastic, thus guaranteeing that the bit-streams will not be correlated.

The rest of this article is organized as follows. Background information on the CRAM architecture, SC basics, and MTJ-based TRNGs is presented in Section II. Section III provides an overview of SC-CRAM, including task scheduling, implementation of basic and complex functions, and its advantages over conv-CRAM. In Section IV, we elaborate on the four example applications that are simulated in SPICE. The output accuracy and performance evaluation of SC-CRAM, conv-CRAM, and CMOS-based SC are presented in Section V. Finally, this article is concluded in Section IV.

II. BACKGROUND

A. CRAM ARCHITECTURE

The general structure of the CRAM array is shown in Fig. 1(a). The configuration of the CRAM cell is similar to an STT-MRAM cell, which uses a 1-transistor 1-MTJ (1T1M) structure, except that the CRAM cell uses a 2T1M structure. This architecture allows for logic and memory paths, where voltages are applied to each cell independently during the memory read and write operations, whereas voltages are applied to multiple cells simultaneously during the logic operations. This enables true in-memory computing capability in CRAM. The MTJ in each bit cell is addressed using the memory word line (WL) and logic operations are performed in the CRAM cell by enabling the logic bitline (LBL).

The two operation modes in CRAM are the memory and logic modes, which are shown in Fig. 1(b) and (c), respectively. During the memory mode, WL is high and LBL is low. This enables data to be read from or written to the MTJ through the memory bitline (MBL). During the logic mode, WL is low and LBL is high, which allows MTJs in the same row to be connected through the logic line (LL). Logic operations with multiple inputs and one output are performed by applying the appropriate voltages (V_{BSL}) to the bit select lines (BSL) of the input cells. The current through the output MTJ (J_{OUT}) is dependent on the resistance states of the input MTs and switches the state of the output MTJ if $J_{\mathrm{OUT}} \geq J_C$. The criteria for V_{BSL} and the initial state of the output MTJ (output preset) for various logic operations are shown in Table 1.

B. STOCHASTIC COMPUTING

In SC, data are encoded as probabilities in streams of random bits. There are significant reductions in hardware area for processing circuits since multiplication, scaled addition, and scaled subtraction of two stochastic bit-streams can be performed using single logic gates [7], [9], [12], [13].

Fig. 2(c) shows the overall scheduling of steps for conventional SC schemes. The first step is the SBG, where the input data (either digital or analog representation) are converted to stochastic bit-streams. This is typically done using comparators with the input data and the output of an RNG with an output centered around 0.5 connected at the

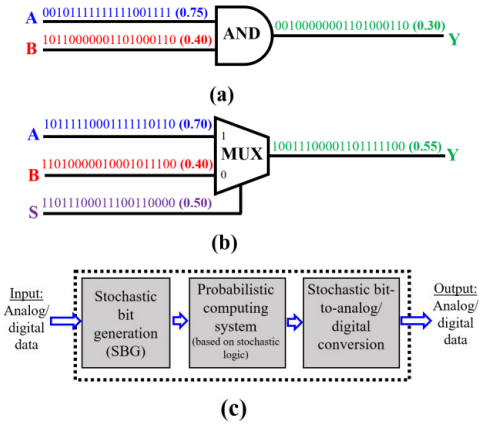

FIGURE 2. Illustration of (a) multiplication using AND logic and (b) scaled addition using MUX of stochastic bit-streams *A* and *B*. (c) Block diagram of the conventional stochastic computing scheme.

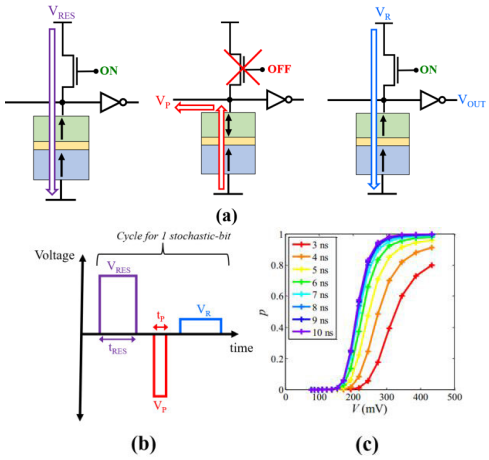

FIGURE 3. (a) Illustration of an MTJ-based tunable TRNG for SC; (b) scheduling of reset, perturb, and read voltage pulses (V_{RES}, V_P, and V_R); and (c) switching probability distribution versus V_P curves at various pulsewidths (extracted from [20]).

input terminals. At each clock cycle, a random number is generated and the comparator produces a "1" if the random number is greater than the input value; otherwise, it produces a "0." The total number of clock cycles is dependent on the desired number of bits in the bit-streams (N_B). Consider a stochastic bit-stream A generated from an input value of X_A. The numeric representation of A is the probability of a bit in A being "1," which is directly correlated with X_A. In this study, we represent bit-streams using unipolar formatting, meaning that they lie within a [0, 1] range.

Data computation is performed in the second step, where various arithmetic functions, determined by the desired application, are performed on the stochastic bit-streams. It is in this step where we see the advantages of SC best exemplified. For example, Fig. 2(a) shows how multiplication of bit-streams A and B can be performed using a single AND gate and Fig. 2(b) shows how a single MUX can perform scaled addition on A and B. For scaled addition, a third bit-stream, S, is applied at the selector terminal of the MUX and the output is expressed

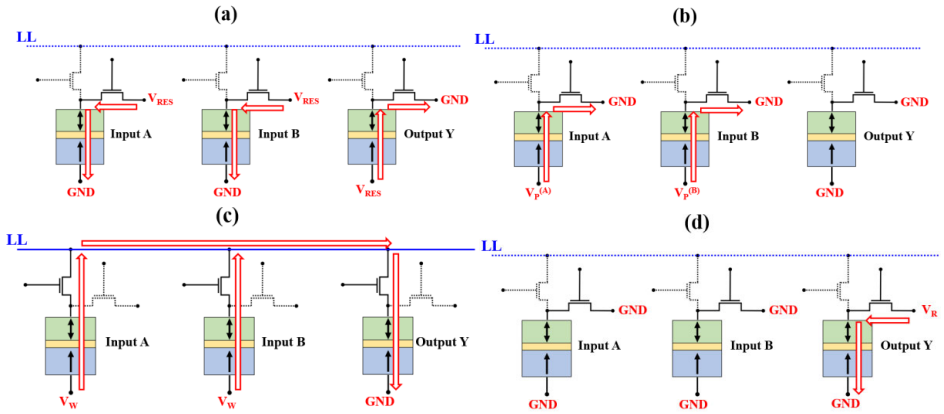

FIGURE 4. Current paths for (a) reset, (b) perturb, (c) logic, and (d) read steps for multiplication of bit-streams *A* and *B* using AND logic in SC-CRAM.

as $Y \approx S \times A + (1 - S) \times B$. For most applications, S is fixed at 0.5, therefore $Y \approx 0.5 \times (A + B)$. Note that scaled addition can also be done using a MAJ3 gate if one of the inputs is fixed at 0.5. More advanced arithmetic functions, such as scaled division, exponential, and hyperbolic tangent, can also be effectively implemented in SC [9], [14], some of which will be explained in more detail in Section III-B.

In the third and final steps, the numeric representations of the output bit-streams are determined. This is typically done using an N_B-bit digital counter. As with the input data, the output data can be represented in either analog or digital format. In this article, we will not focus on this third step since it is the same for our proposed method and conventional SC schemes.

C. MTJ-BASED TUNABLE TRNG

The spin-transfer torque (STT) and spin-orbit torque (SOT) switching mechanisms in MTJs are subject to random thermal fluctuations, thus introducing a probabilistic element. Extrinsic factors that determine the switching probability (P_{SW}) of the MTJ are the voltage pulse amplitude (V_P) and its duration (t_P), as shown in Fig. 3(c). The stochasticity inherent to MTJs make them promising for probabilistic computing applications such as elementary computing units in Boltzmann machines [36], [37], TRNGs [24], [25], [38], and most importantly for this article, generating stochastic bit-streams for SC [26], [17], [18]. Fig. 3(a) shows the general method of SBG using repeated cycles of synchronized reset, perturb, and read pulses, and Fig. 3(b) shows the scheduling of these pulses. Each cycle begins by setting the MTJ to the P-state with the reset pulse. The perturb pulse switches the MTJ to the AP-state probabilistically, where P_{SW} is determined by V_P and t_P. The resistance state of the MTJ is read at the end of each cycle using V_R, where V_R is set small enough so that it does not influence the state of the MTJ.

III. OVERVIEW OF SCCRAM

In this section, we describe the key aspects of our proposed method of implementing SC in the CRAM architecture. First, we explain the basic task scheduling of SC-CRAM, using an AND gate for multiplication as an example. Then, we describe the processes for performing more complicated arithmetic functions in SC-CRAM. Finally, we describe the preliminary advantages of SC-CRAM over conv-CRAM and CMOS-based SC.

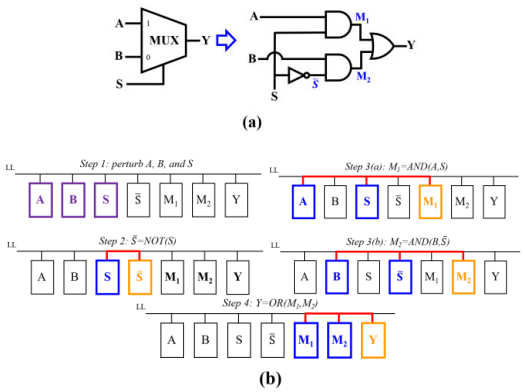

FIGURE 5. (a) Circuit diagram of a 2-to-1 MUX and (b) scheduling operations in SC-CRAM. For simplicity, reset steps and final read steps are not shown.

A. TASK SCHEDULING IN SC-CRAM

The SC-CRAM method proposed combines the processes for MTJ-based SBG described in Section II-C and logic operations in CRAM described in Section II-A. The SBG method in SC-CRAM is similar to the method shown in Fig. 3(a) and (b), except that each cycle includes an additional logic step between the perturb and read steps. Therefore, each cycle in SC-CRAM consists of synchronized reset, perturb, logic, and read pulses. It should be noted that the random bits are generated during the perturb and reset steps and computation is performed during the logic step, which is all done in the same cells. This makes SC-CRAM unique in that SBG and computation are embedded within the same circuits and the same cycles; therefore, external circuitry is not required for random number generation.

Consider the example shown in Fig. 4(a)–(d), where AND logic in CRAM is used to multiply inputs *A* and *B*. Fig. 4(a) shows the current paths during the reset step, where V_{RES}

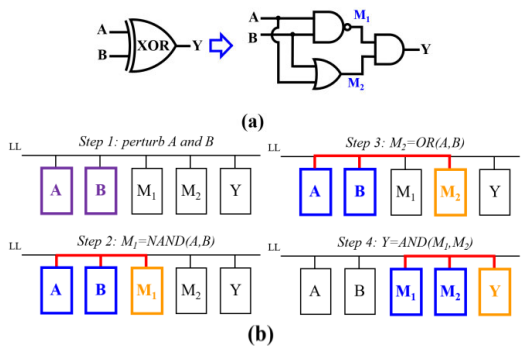

FIGURE 6. (a) Circuit diagram of an XOR gate and (b) scheduling operations in SC-CRAM. For simplicity, reset steps and final read steps are not shown.

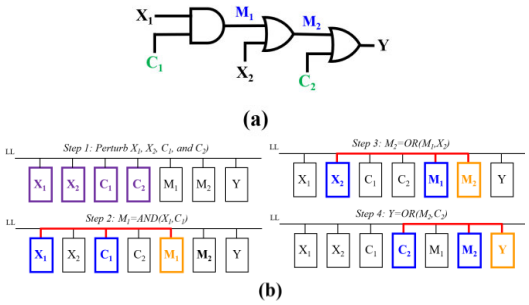

FIGURE 7. (a) Circuit diagram of the square root function in SC-CRAM where $Y \approx \sqrt{X}$, X_1 and X_2 are independent bit-streams with probabilities equal to X, and C_1 and C_2 are bit-streams with fixed probabilities of 0.67 and 0.18, respectively. (b) Scheduling operations in SC-CRAM. For simplicity, reset steps and final read steps are not shown.

initializes the input MTJs (A and B) to the P-state and the output MTJ (Y) to the AP-state. Note that Y is initialized to the AP-state to meet the output preset criteria for AND logic in CRAM (recall Table 1).

The perturb pulses, shown in Fig. 4(b), are applied along the MBL only for input cells A and B, which cause the MTJs to switch probabilistically, with probabilities dependent on $V_P^{(A)}$ and $V_P^{(B)}$. Note that the values for $V_P^{(A)}$ and $V_P^{(B)}$ can be determined in three ways depending on the desired task: 1) in machine learning applications, one of the input cells can perform a synaptic function, and therefore, V_P is dependent on the synaptic weight determined using learning algorithms; 2) other tasks may require multiplication of a fixed constant, and therefore, V_P is determined to produce a probability equal to the desired constant; and 3) the most likely scenario is that V_P is dependent on the input data. For example, in image processing applications, $V_P^{(A)}$ corresponds to intensity at pixel A. It should be noted that V_P only switches the MTJ from the P-state to the AP-state since the reset pulse switches the MTJ back to the P-state each cycle.

The logic step, shown in Fig. 4(c), replicates the logic operation performed in conv-CRAM (recall Section II-B). The voltage pulses applied at the BSL at A and B (V_W) follow the same criteria as V_{BSL} described in Table 1. During the perturb step, A and B switch probabilistically, whereas Y switches deterministically during the logic step.

The final state of Y is read at the end of each cycle with V_R, as shown in Fig. 4(d). As with the process shown in Fig. 3(a), V_R should be set so that it does not affect the state of Y. Alternatively, for tasks involving multiple computation stages, the steps at each computation stage can be overlapped so that the read steps at each stage prior to the final stage can perform the logic steps in the following stages. By adding a logic step in each cycle, SBG and computation are embedded within the same circuit.

B. ARITHMETIC FUNCTIONS IN SC-CRAM

The implementation of SC in CRAM allows for various arithmetic functions to be performed efficiently. Previous studies on computation with stochastic bit-streams have demonstrated that a wide variety of functions, such as exponential, linear gain function, and hyperbolic tangent, can be performed in SC schemes with a minimal number of logic gates [9], [14]. Theoretically, any function performed in CMOS-based SC can also be performed in SC-CRAM. However, in this study, we will only focus on the arithmetic functions needed in the three applications described in Section IV. These functions include multiplication (described in Section III-A), scaled addition, absolute valued subtraction, square root, and scaled division. Note that for our calculations, we assumed that each logic function was performed using NAND logic (see Supplementary Note 1), and however, this is not shown in Figs. 5–8.

Scaled addition can be done with either MAJ3 logic or an MUX [recall Fig. 2(b)]. In SC-CRAM, a MAJ3 gate consists of two input MTJs with variable probabilities (A and B), one input MTJ with a fixed probability of 0.5 (S), and one output MTJ (Y). Furthermore, MAJ3 logic in SC-CRAM has the same number of computation steps as AND logic. An MUX can also be implemented in SC-CRAM using the equivalent circuit diagram shown in Fig. 5(a). This circuit can be built in CRAM hardware with two input MTJs with variable probabilities (A and B), one input MTJ with any fixed probability (S), three intermediate MTJs ($\bar{S}$, M_1, and M_2), and one output MTJ (Y). Steps for implementing MUX logic in SC-CRAM consist of a perturb step on A, B, and S, one NOT logic step on $\bar{S}$, two overlapping AND logic steps on M_1 and M_2, and one OR logic step on Y, as shown in Fig. 5(b).

The advantage of using MAJ3 logic for scaled addition is that it requires fewer MTJs and fewer computation steps than MUX logic. However, S is limited to a value of 0.5 in MAJ3 logic, whereas S can be any value in MUX logic (recall Section II-C). Furthermore, when performing scaled addition on more than two inputs using MAJ3 logic, multiple layers of MAJ3 gates are required, which increases the number of MTJs and computation delay. Alternatively, MUX logic can handle more than one input without significantly increasing the number of computation steps. Therefore, for some applications, MUX logic is the preferred method for scaled addition.

Absolute valued subtraction of stochastic bit-streams A and B can be done using XOR logic. In CRAM, an XOR gate can

TABLE 2. Number of MTJs, computation steps, and subarray size for various arithmetic functions in SC-CRAM (the number of MTJs and computation steps are shown relative to Conv-CRAM).

Function	Gates used for SC-CRAM	Number of MTJs	Number of Computation steps	Sub-array (rows x columns) SC-CRAM	Sub-array (rows x columns) Conv-CRAM
Scaled Addition	MUX	0.080X	14.3X	1 x 7	1 x 88
	MAJ3	0.045X	10.7X	1 x 4	
Multiplication	AND	0.006X	5.1X	1 x 4	16 x 161
Absolute valued Subtraction	XOR	0.090X	22.5X	1 x 8	1 x 90
Scaled Division	JK flip-flop	0.013X	2.0X	1 x 13	8 x 130
Square Root	AND-OR-OR	0.002X	0.49X	1 x 10	32 x 1413

**Assume 8-bit resolution for values shown.*

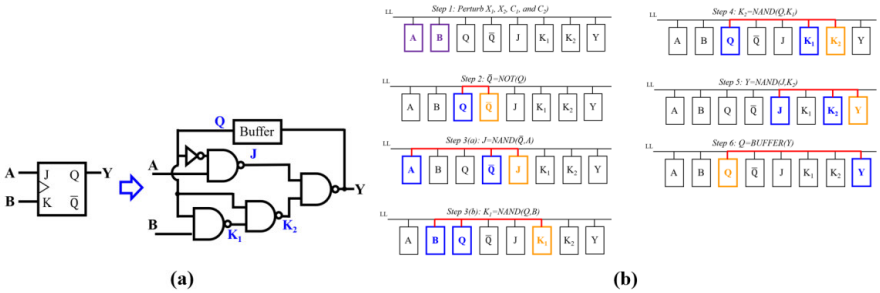

FIGURE 8. (a) Circuit diagram for a JK flip-flop to compute the function $Y \approx A/(A + B)$ and (b) scheduling operations in SC-CRAM. For simplicity, reset steps and final read steps are not shown.

be implemented using the equivalent circuit diagram shown in Fig. 6(a). It consists of two input MTJs (A and B), two intermediate MTJs (M_1 and M_2), and one output MTJ (Y). SC-CRAM steps for XOR logic consist of one perturb step on A and B, one NAND logic step on M_1, one OR logic step on M_2, and one AND logic step on Y, as shown in Fig. 6(b). It should be noted that XOR logic produces $Y \approx |A - B|$ only when bit-streams A and B have the maximum correlation. In most cases, this is undesirable because computation on stochastic bit-streams typically requires maximum decorrelation. However, the application described in Section IV-A is a special case and maximum correlation can be ensured using AND logic prior to XOR logic (process described in further detail in Section IV-A).

The square root of stochastic bit-stream X can be calculated using AND logic followed by two layers of OR logic, as shown in Fig. 7(a). The SC-CRAM circuit consists of two input MTJs (X_1 and X_2), two input MTJs with fixed probabilities of 0.67 and 0.18 (C_1 and C_2), two intermediate MTJs (M_1 and M_2), and one output MTJ (Y). SC-CRAM steps consist of one perturb step on X_1, X_2, C_1, and C_2, one AND logic step on M_1, and two consecutive OR logic steps on M_2 and Y, as shown in Fig. 7(b). Note that X_1 and X_2 both have probabilities equal to X, however, they are generated independently.

Scaled division of stochastic bit-streams A and B can be calculated using a JK flip-flop, which can be implemented in CRAM using the equivalent circuit diagram shown in Fig. 8(a). If A and B are applied to the J and K terminals, respectively, the output Y is expressed as $Y \approx A/(A + B)$. The SC-CRAM circuit consists of two input MTJs (A and B), five intermediate MTJs (Q, $\bar{Q}$, J, K_1, and K_2), and one output MTJ (Y). SC-CRAM steps consist of one perturb step on A and B, one NOT step on Q, one NAND operation to determine J, two consecutive NAND operations to determine K_2, and one final NAND operation to determine Y. In the final step, one BUFFER operation is performed to determine Q, as shown in Fig. 8(b). Note that Q is set to "0" for the first cycle.

C. ADVANTAGES OF SC-CRAM

Table 2 shows the number of MTJs (N_{MTJ}), the number of computation steps (N_C), and the subarray size for each arithmetic function described in Sections III-A and III-B in SC-CRAM. The values for N_{MTJ} and N_C are shown relative to conv-CRAM. Furthermore, calculations for each of these performance metrics compared computation on numbers with 8-bit resolution. For conv-CRAM, calculations were done on 8-bit digital numbers, and for SC-CRAM, calculations were performed on bit-streams with $N_B = 2^8 = 256$ bits. When calculating performance metrics in conv-CRAM, we assumed that addition and multiplication were performed using an 8-bit carry ripple adder (CRA) and an 8-bit Wallace tree multiplier (WTM), respectively. We also assumed that subtraction and division in conv-CRAM were performed using full subtractor (FS) circuits and nonrestoring array dividers (NAD), respectively. Each of these circuits comprises NAND-based full adders (FAs), details of which are described in Supplementary Notes 2–6. Finally, we assumed that square root in conv-CRAM was performed using three cycles of the Newton–Raphson (N–R) method, which was assumed in [10]. Details of the N–R method are described in Supplementary Note 7.

TABLE 3. Performance evaluation of SC-CRAM (the number of CRAM cells, computation steps, and energy consumption are shown relative to Conv-CRAM).

Application	Sub-array size (rows x columns)		Number of CRAM cells	Number of Computation steps	Energy consumption	Noise margin (feasible?)	
	SC-CRAM	Conv-CRAM				SC-CRAM	Conv-CRAM
Local image thresholding	128 x 128	2048 x 4096	0.048X	0.377X	12.9X	2% (Yes)	-196% (No)
Object location	1 x 16	16 x 1024	0.005X	1.016X	1.12X	10% (Yes)	9% (Yes)
Heart disaster prediction	8 x 32	128 x 1024	0.005X	0.662X	1.18X	9% (Yes)	1% (Yes)
Kernel density estimation	32 x 64	512 x 2048	0.022X	0.674X	5.37X	8% (Yes)	-117% (No)

**Assume 8-bit resolution for values shown.*

The values in Table 2 show that SC-CRAM uses significantly less MTJs for all functions than conv-CRAM. The reduction in N_{MTJ} in SC-CRAM is the most significant for multiplication and square root functions but the least significant for scaled addition and scaled subtraction functions. This is due to the large number of FAs needed for 8-bit WTM circuits and for the N–R method, which is not needed in SC-CRAM. However, N_C is larger in SC-CRAM than in conv-CRAM for all functions, except square root. This result is not unique to SC-CRAM and is observed when comparing any SC schemes to conventional deterministic schemes on binary numbers. The increase in N_C in SC-CRAM is most noticeable for scaled addition and absolute valued subtraction but least significant for multiplication and scaled division. Most notably, N_C is smaller for SC-CRAM than for conv-CRAM for square root. This is likely because we assumed that conv-CRAM repeated the N–R method three times to approximate the square root function, which increases N_C three times.

The subarray dimensions shown in Table 2 reveal that multiple subarray rows are required for multiplication, division, and square root in conv-CRAM, whereas only one subarray row is required for all functions in SC-CRAM. A low number of subarray rows are very important in applications where a large number of arithmetic functions are performed in parallel since subarrays with a large number of rows are susceptible to parasitic effects such as a large IR drop and low noise margins. As we will see in Section IV, the large number of subarray rows in conv-CRAM makes it an unfeasible solution for certain applications.

The calculations shown in Table 2 indicate several key features of SC-CRAM. One is that SC-CRAM is ideal for solving functions involving a large number of multiplication and scaled division functions (such as high-order polynomials and Maclaurin expansions of more complicated functions). Second is that both total circuit area and computation delay may decrease in SC-CRAM for more complex functions, as seen in the calculations for the square root function. While SC-CRAM is not well suited for tasks involving rapid, high-precision calculations, the results in Table 2 suggest that SC-CRAM is ideal for applications that involve rough approximations to detect statistical anomalies in large datasets.

IV. EVALUATION AND RESULTS

In this section, we describe the three applications that were performed in this study: local image thresholding for character recognition, Bayesian inference for object location, and Bayesian belief network for heart disaster prediction. These applications are described in greater detail in Supplementary Notes 8–11. We evaluated the performance of SC-CRAM for each of these applications. For each application, the execution time, circuit area, energy usage, and noise margin are compared with those of conv-CRAM. The parameters for the MTJ devices used for our analysis are described in Supplementary Note 12.

The performance of SC-CRAM and conv-CRAM was evaluated in terms of circuit area, computation delay, and total energy consumption (E_{TOT}). For each application described in Section IV, the circuit area was defined in terms of the number of CRAM cells used (N_{CELLS}) and the computation delay was defined in terms of the number of computation steps (N_C). The total energy consumption was calculated using (1), where R_{MTJ} is the resistance of the MTJ, V is the voltage amplitude, and t is the pulsewidth. E_{TOT} was determined by calculating the energy consumption for the reset, perturb, and logic steps, and adding them. Note that one difference in E_{TOT} in SC-CRAM and E_{TOT} in conv-CRAM is that E_{TOT} in conv-CRAM does not include the perturb steps. Furthermore, unlike conv-CRAM, the logic and reset steps are repeated 256 (8-bit resolution) times in SC-CRAM, which was accounted for in the E_{TOT} calculations

$$\text{Energy} = \frac{V^2 t}{R_{MTJ}}. \tag{1}$$

Table 3 shows all of the calculations for N_{CELLS}, N_C, and E_{TOT}. In addition, Table 3 shows the number of rows and the number of columns in the CRAM subarray. These data show that SC-CRAM uses approximately 20–180 times fewer CRAM cells than conv-CRAM for all four applications. Furthermore, SC-CRAM also has about 1/3–2/3 fewer computation steps than conv-CRAM for each application except the Bayesian inference system, where N_C is nearly equal between SC-CRAM and conv-CRAM. This shows that, for three of the four applications, the SC-CRAM circuit has less computation delay than the conv-CRAM circuit. This may seem counterintuitive since each step needs to be repeated 256 times in SC-CRAM, and however, the smaller N_C values

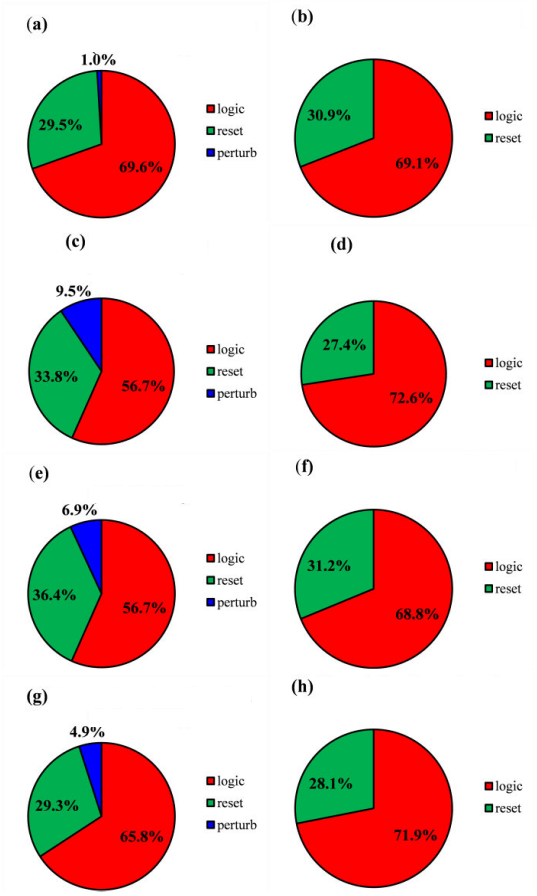

FIGURE 9. Breakdown of energy consumption in SC-CRAM and conv-CRAM for (a) and (b) local image thresholding, (c) and (d) object location via Bayesian inference, (e) and (f) heart disaster prediction via a Bayesian belief network, and (g) and (h) kernel density estimation.

in SC-CRAM can be attributed to two features of SC-CRAM. The first is the small number of cells required to perform both simple and complex arithmetic functions and the second is the ability to overlap the logic steps in some CRAM cells with the perturb and/or reset steps in other CRAM cells.

These data also show that the total energy consumption is approximately equal between SC-CRAM and conv-CRAM for the Bayesian inference system and the Bayesian belief network. However, E_{TOT} is larger for SC-CRAM for local image thresholding and kernel density approximation by 12.9× and 5.4×, respectively. It should be noted that conv-CRAM has nearly 40× less energy consumption than modern near-memory processing systems [31]; therefore, SC-CRAM is still a competitive solution since E_{TOT} in SC-CRAM is on the same order of magnitude as E_{TOT} in conv-CRAM.

The subarrays for SC-CRAM have significantly less rows than conv-CRAM, which leads to significantly less IR drop [39], leading to an increase in noise margin. The noise margin represents the size of the range of voltages where a given logic operation can be performed. For example, a noise margin of 10% means that there is a 10% difference between the minimum and maximum voltage that can be applied during the logic operation while maintaining an acceptable output accuracy. Any noise margin below zero means that there are no voltage values that exist that can reliably produce an accurate output (see Supplementary Note 13 for a more detailed description of noise margin). Table 3 shows that the noise margin is too low for local image thresholding and kernel density estimation to be feasible in conv-CRAM due to the large number of subarray rows. This shows that, despite the reduction in E_{TOT} in conv-CRAM, SC-CRAM is the preferred approach for local image thresholding and kernel density estimation because the larger noise margin in SC-CRAM makes it more reliable. The noise margin for the Bayesian inference system and the Bayesian belief network is large enough to be feasible in both SC-CRAM and conv-CRAM. However, the noise margin in SC-CRAM is still noticeably larger, and therefore, it is still the preferred method for these two applications.

The pie charts in Fig. 9 show the portion of E_{TOT} being consumed by the logic, reset, and perturb steps for SC-CRAM and conv-CRAM. Note that conv-CRAM does not use a perturb step, and therefore, only the logic and reset steps are considered. For conv-CRAM, the distribution of energy consumption between the reset and logic steps is nearly the same for all four applications, where approximately 25%–30% of E_{TOT} is consumed during the reset steps and 70%–75% of E_{TOT} is consumed during the logic steps. For SC-CRAM, the reset steps account for around 25%–30% of E_{TOT}. Most importantly, these charts show that adding the perturb step in SC-CRAM only accounts for a minority of the total energy consumption. It should be noted that the Bayesian inference system consumes the most portion of E_{TOT} for the perturb step, and however, it still only consumes 9.5% of E_{TOT}. Conversely, generating stochastic bit-streams with conventional CMOS-based methods could increase the total energy consumption by nearly 80% [8]. These results indicate that SC-CRAM may outperform CMOS-based SC methods in terms of E_{TOT}.

V. CONCLUSION

In this study, we introduced a method in which CRAM can be used to generate and compute stochastic bit-streams called SC-CRAM. In SC-CRAM, the SBG and computation steps are embedded within the same circuit block, meaning zero cost in the total circuit area and computation delay to generate stochastic bit-streams. We compared the performance metrics between SC-CRAM and conv-CRAM for local image thresholding, object location, heart disaster prediction, and kernel density estimation. We determined that SC-CRAM is very efficient at computing complicated functions, which reduces the total number of CRAM cells required to perform these four applications without sacrificing the total computation delay. Furthermore, the energy consumption was on the same order of magnitude for SC-CRAM and conv-CRAM except for local image thresholding. Finally, SC-CRAM had a significantly larger noise margin for local image thresholding and kernel density estimation, making it a much more feasible solution for these applications.

REFERENCES

[1] J. Grollier, D. Querlioz, and M. D. Stiles, "Spintronic nanodevices for bioinspired computing," *Proc. IEEE*, vol. 104, no. 10, pp. 2024–2039, Oct. 2016, doi: 10.1109/JPROC.2016.2597152.

[2] C. D. Schuman et al., "A survey of neuromorphic computing and neural networks in hardware," 2017, *arXiv:1705.06963*.

[3] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nature Comput. Sci.*, vol. 2, no. 1, pp. 10–19, Jan. 2022, doi: 10.1038/s43588-021-00184-y.

[4] G. Indiveri and S.-C. Liu, "Memory and information processing in neuromorphic systems," *Proc. IEEE*, vol. 103, no. 8, pp. 1379–1397, Aug. 2015, doi: 10.1109/JPROC.2015.2444094.

[5] D. Querlioz, O. Bichler, A. F. Vincent, and C. Gamrat, "Bioinspired programming of memory devices for implementing an inference engine," *Proc. IEEE*, vol. 103, no. 8, pp. 1398–1416, Aug. 2015, doi: 10.1109/JPROC.2015.2437616.

[6] K. Seshan, "Limits and hurdles to continued CMOS scaling," in *Handbook of Thin Film Deposition*, 4th ed. Norwich, NY, USA: William Andrew, 2018, pp. 19–41, doi: 10.1016/B978-0-12-812311-9.00002-5.

[7] 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, doi: 10.1109/TCAD.2017.2778107.

[8] 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, doi: 10.1109/TC.2010.202.

[9] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. 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, doi: 10.1109/TVLSI.2013.2247429.

[10] M. H. Najafi and M. E. Salehi, "A fast fault-tolerant architecture for Sauvola local image thresholding algorithm using stochastic computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 2, pp. 808–812, Feb. 2016, doi: 10.1109/TVLSI.2015.2415932.

[11] P. Li and D. J. Lilja, "A low power fault-tolerant architecture for the kernel density estimation based image segmentation algorithm," in *Proc. IEEE Int. Conf. Appl.-Specific Syst. Archit. Process*, Sep. 2011, pp. 161–168, doi: 10.1109/ASAP.2011.6043264.

[12] J. M. de Aguiar and S. P. Khatri, "Exploring the viability of stochastic computing," in *Proc. 33rd IEEE Int. Conf. Comput. Design (ICCD)*, Oct. 2015, pp. 391–394, doi: 10.1109/ICCD.2015.7357131.

[13] P. Li, W. Qian, D. J. Lilja, K. Bazargan, and M. D. Riedel, "Case studies of logical computation on stochastic bit streams," in *Integrated Circuit and System Design, Power and Timing Modeling, Optimization and Simulation* (Lecture Notes in Computer Science), vol. 7606. Berlin, Germany: Springer, 2013, pp. 235–244, doi: 10.1007/978-3-642-36157-9_24.

[14] Y. Liu and K. K. Parhi, "Computing hyperbolic tangent and sigmoid functions using stochastic logic," in *Proc. 50th Asilomar Conf. Signals, Syst. Comput.*, Nov. 2016, pp. 1580–1585, doi: 10.1109/ACSSC.2016.7869645.

[15] H. Hsiao, J. Anderson, and Y. Hara-Azumi, "Generating stochastic bitstreams," in *Stochastic Computing: Techniques and Applications*. Berlin, Germany: Springer, 2019, pp. 137–152, doi: 10.1007/978-3-030-03730-7_7.

[16] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2s, pp. 1–19, May 2013, doi: 10.1145/2465787.2465794.

[17] L. A. D. B. Naviner, H. Cai, Y. Wang, W. Zhao, and A. Ben Dhia, "Stochastic computation with spin torque transfer magnetic tunnel junction," in *Proc. IEEE 13th Int. New Circuits Syst. Conf. (NEWCAS)*, Jun. 2015, pp. 1–4, doi: 10.1109/NEWCAS.2015.7182031.

[18] N. Onizawa, D. Katagiri, W. J. Gross, and T. Hanyu, "Analog-to-stochastic converter using magnetic tunnel junction devices for vision chips," *IEEE Trans. Nanotechnol.*, vol. 15, no. 5, pp. 705–714, Sep. 2016, doi: 10.1109/TNANO.2015.2511151.

[19] X. Jia, J. Yang, Z. Wang, Y. Chen, H. H. Li, and W. Zhao, "Spintronics based stochastic computing for efficient Bayesian inference system," in *Proc. IEEE 23rd Asia South Pacific Design Automat. Conf. (ASP-DAC)*, Jan. 2018, pp. 580–585, doi: 10.1109/ASPDAC.2018.8297385.

[20] Y. Lv and J.-P. Wang, "A single magnetic-tunnel-junction stochastic computing unit," in *IEDM Tech. Dig.*, Dec. 2017, p. 36, doi: 10.1109/IEDM.2017.8268504.

[21] Y. Shao et al., "Implementation of artificial neural networks using magnetoresistive random-access memory-based stochastic computing units," *IEEE Magn. Lett.*, vol. 12, pp. 1–5, 2021, doi: 10.1109/LMAG.2021.3071084.

[22] E. Becle, G. Prenat, P. Talatchian, L. Anghel, and I.-L. Prejbeanu, "A fast, energy efficient and tunable magnetic tunnel junction based bitstream generator for stochastic computing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 8, pp. 3251–3259, Aug. 2022, doi: 10.1109/TCSI.2022.3173030.

[23] W. Ho Choi et al., "A magnetic tunnel junction based true random number generator with conditional perturb and real-time output probability tracking," in *IEDM Tech. Dig.*, Dec. 2014, p. 12, doi: 10.1109/IEDM.2014.7047039.

[24] S. Oosawa, T. Konishi, N. Onizawa, and T. Hanyu, "Design of an STT-MTJ based true random number generator using digitally controlled probability-locked loop," in *Proc. IEEE 13th Int. New Circuits Syst. Conf. (NEWCAS)*, Jun. 2015, pp. 3–6, doi: 10.1109/NEWCAS.2015.7182089.

[25] W. H. Choi, Y. Lv, H. Kim, J.-P. Wang, and C. H. Kim, "An 8-bit analog-to-digital converter based on the voltage-dependent switching probability of a magnetic tunnel junction," in *Proc. Symp. VLSI Technol. (VLSI Technology)*, Jun. 2015, pp. 162–163, doi: 10.1109/VLSIT.2015.7223662.

[26] C. Safranski, J. Kaiser, P. Trouilloud, P. Hashemi, G. Hu, and J. Z. Sun, "Demonstration of nanosecond operation in stochastic magnetic tunnel junctions," *Nano Lett.*, vol. 21, no. 5, pp. 2040–2045, Feb. 2021, doi: 10.1021/acs.nanolett.0c04652.

[27] K. Hayakawa et al., "Nanosecond random telegraph noise in in-plane magnetic tunnel junctions," *Phys. Rev. Lett.*, vol. 126, no. 11, Mar. 2021, doi: 10.1103/PhysRevLett.126.117202.

[28] M. W. Daniels, A. Madhavan, P. Talatchian, A. Mizrahi, and M. D. Stiles, "Energy-efficient stochastic computing with superparamagnetic tunnel junctions," *Phys. Rev. Appl.*, vol. 13, no. 3, Mar. 2020, Art. no. 034016, doi: 10.1103/PhysRevApplied.13.034016.

[29] Z. Chowdhury et al., "Efficient in-memory processing using spintronics," *IEEE Comput. Archit. Lett.*, vol. 17, no. 1, pp. 42–46, Jan./Jun. 2018, doi: 10.1109/LCA.2017.2751042.

[30] M. Zabihi et al., "Using spin-Hall MTJs to build an energy-efficient in-memory computation platform," in *Proc. 20th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2019, pp. 52–57, doi: 10.1109/ISQED.2019.8697377.

[31] M. Zabihi, Z. I. Chowdhury, Z. Zhao, U. R. Karpuzcu, J.-P. Wang, and S. S. Sapatnekar, "In-memory processing on the spintronic CRAM: From hardware design to application mapping," *IEEE Trans. Comput.*, vol. 68, no. 8, pp. 1159–1173, Aug. 2019, doi: 10.1109/TC.2018.2858251.

[32] H. Cilasun et al., "Spiking neural networks in spintronic computational RAM," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, pp. 1–21, Sep. 2021, doi: 10.1145/3475963.

[33] 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, doi: 10.1109/TNANO.2014.2300342.

[34] S. Gupta et al., "SCRIMP: A general stochastic computing architecture using ReRAM in-memory processing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 1598–1601, doi: 10.23919/DATE48585.2020.9116338.

[35] 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, doi: 10.1109/ISCAS45731.2020.9180743.

[36] K. Y. Camsari, B. M. Sutton, and S. Datta, "P-bits for probabilistic spin logic," *Appl. Phys. Rev.*, vol. 6, no. 1, Mar. 2019, Art. no. 011305, doi: 10.1063/1.5055860.

[37] O. Hassan, R. Faria, K. Y. Camsari, J. Z. Sun, and S. Datta, "Low-barrier magnet design for efficient hardware binary stochastic neurons," *IEEE Magn. Lett.*, vol. 10, pp. 1–5, 2019, doi: 10.1109/LMAG.2019.2910787.

[38] B. Parks, M. Bapna, J. Igbokwe, H. Almasi, W. Wang, and S. A. Majetich, "Superparamagnetic perpendicular magnetic tunnel junctions for true random number generators," *AIP Adv.*, vol. 8, no. 5, May 2018, Art. no. 055903, doi: 10.1063/1.5006422.

[39] M. Zabihi et al., "Analyzing the effects of interconnect parasitics in the STT CRAM in-memory computational platform," *IEEE J. Explor. Solid-State Comput. Devices Circuits*, vol. 6, no. 1, pp. 71–79, Jun. 2020, doi: 10.1109/JXCDC.2020.2985314.