From Multipliers to Integrators: a Survey of Stochastic Computing Primitives

Shanshan Liu, Josep L. Rosselló, Siting Liu, Xiaochen Tang, Joan Font-Rosselló, Christian F. Frasser, Weikang Qian, Jie Han, Pedro Reviriego, and Fabrizio Lombardi

***Abstract*—Stochastic Computing (SC) has the potential to dramatically improve important nanoscale circuit metrics, including area and power dissipation, for implementing complex digital computing systems, such as large neural networks, filters, or decoders, among others. This paper reviews the state-of-the-art design of important SC building blocks covering both arithmetic circuits, including multipliers, adders, and dividers, and finite state machines (FSMs) that are needed for numerical integration, accumulation, and activation functions in neural networks. For arithmetic circuits, we review newly proposed schemes, such as Delta Sigma Modulator-based dividers providing accurate and low latency computation, as well as design considerations by which the degree of correlation/decorrelation can be efficiently handled at the arithmetic circuit level. As for complex sequential circuits, we review classical stochastic FSM schemes as well as new designs using the recently-proposed dynamic SC to reduce the length of a stochastic sequence to obtain computation results. These stochastic circuits are compared to traditional implementations in terms of efficiency and delay for various levels of accuracy to illustrate the ranges of values for which SC provides significant performance benefits.**

***Index Terms*—Stochastic computing, multiplier, adder, divider, finite state machine, integrator, gradient descent**

Manuscript received April 7, 2023; revised November 25, 2023, and January 24, 2024. This work was partially supported by the FUN4DATE (PID2022-136684OB-C22), ENTRUDIT (TED2021-130118B-I00) and PID2020-120075RB-I00 projects funded by the Ministerio de Ciencia e Innovación (MCIN/AEI/10.13039/501100011033).

Shanshan Liu and Xiaochen Tang are with University of Electronic Science and Technology of China, Chengdu, Sichuan, China. Email: {ssliu,xctang}@uestc.edu.cn.

Josep L. Rosselló, Joan Font-Rosselló and Christian F. Frasser are with the Electronic Engineering Group, Universitat de les Illes Balears, Palma de Mallorca, 07121, Balears, Spain. Email: j.rossello,joan.font,christian.franco@uib.es. Josep L. Rosselló is also member of the Balearic Islands Health Research Institute (IdISBa), and the Institut d'Investigació en Inteligència Artificial de les Illes Balears (IAIB), Palma de Mallorca, Spain.

Siting Liu is with the School of Information Science and Technology, ShanghaiTech University and Shanghai Engineering Research Center of Energy Efficient and Custom AI IC, Shanghai 201210, China. Email: liust@shanghaitech.edu.cn.

Weikang Qian is with the University of Michigan-Shanghai Jiao Tong University Joint Institute, and the MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai 200240, China. Email: qianwk@sjtu.edu.cn.

Jie Han is with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada. Email: jhan8@ualberta.ca.

Pedro Reviriego is with Universidad Politécnica de Madrid, 28040 Madrid, Spain. Email: pedro.reviriego@upm.es.

Fabrizio Lombardi is with Northeastern University, Department of ECE, Boston, MA 02115, USA. Email: lombardi@ece.neu.edu.

I. Introduction

Stochastic Computing (SC) was originally proposed in the 1960s as an alternative to traditional computing paradigms with the aim of reducing hardware complexity [1]. There is a renewed interest in SC for nanoscale implementations of emerging systems that require many arithmetic operations with hardware constraints. For such applications, SC can substantially reduce the circuit area and power dissipation. Extensive applications of SC-based implementation include neural networks [2]–[5], digital filters [6], [7], image processing [8], [9], decoding of complex error correction codes [10], [11], or mixed-signal sensors [12], [13]. In addition to typical CMOS-based hardware, SC implementation techniques using emerging nanoscale devices, such as memristor and nanomagnetic logic, have also been developed [14]–[18].

To implement an SC system, the understanding of different options and trade-offs when designing building blocks such as adders, multipliers, dividers or finite state machines (FSM), is required; for example, multiplication is significantly simpler in SC (using only an AND or XNOR gate) than in traditional implementations. For stochastic adders or dividers, the design trade-offs must be considered for different systems, because their performance and hardware overhead can vary for different implementation mechanisms. Therefore, for system-level implementations, the design of primitives to build an SC system must be considered first.

A number of technical papers have reviewed SC from different perspectives, such as basic concepts and key applications [19], models [20], sequence management approaches [21], devices for SC implementation [22], and specific system designs for neural networks [23] and image processing [9]. However, there is no recent review of low-level implementations of SC building blocks that covers novel schemes introduced in the last few years. In this paper, we review the state-of-the-art design and implementation of the main building primitives used in SC and compare them with traditional implementations. The challenges and trade-offs when using these primitives to build larger systems are also discussed.

The rest of the paper is organized as follows. Section II presents a brief review of few relevant SC basics. Typical SC primitives including stochastic multipliers, adders, dividers, and FSM-based complex sequential circuits are reviewed in Sections III to VI, respectively. Section VII discusses some emerging alternative SC solutions to the classic SC schemes. Finally, the paper ends in Section VIII with the conclusion.

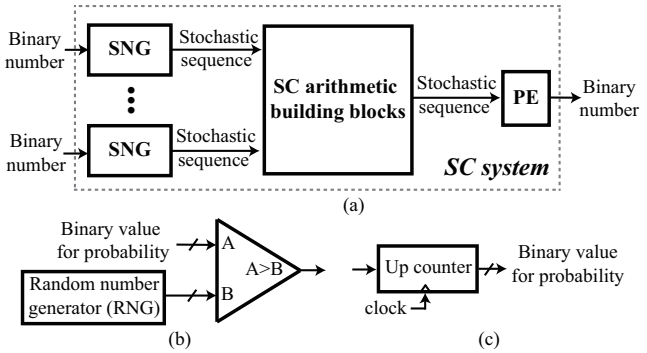

Fig. 1. A stochastic computing (SC) system: (a) the system; (b) a stochastic number generator (SNG); (c) a probability estimator (PE).

II. Basics of Stochastic Computing

A. Data Conversion

Unlike traditional computing paradigms using integer or fixed/floating point numbers, SC is performed on the stochastic encoding of numbers. As shown in Fig. 1, each real number in the binary format is converted/encoded to a stochastic sequence prior to computation and the opposite process is performed after completing computation. The stochastic sequence is a bit stream consisting of 1s and 0s which are binomial distributed. To represent a real number, the probability of each bit being 1 is utilized. Typically, there are two coding approaches for sequence representation, referred to as *unipolar* and *bipolar* codifications [1]. When using the *unipolar* codification, the occurrence probability of 1 (defined as p) in a sequence is used to represent the real number it encodes; while in the *bipolar* case, the real number being represented is $2p-1$. Therefore, the standard value range in SC with *unipolar* or *bipolar* coding is $[0,1]$ or $[-1,1]$, respectively. Alternatively, a $unipolar$ sequence can also be extended with an additional sign bit to represent $bipolar$ information; this is the so-called sign-magnitude representation [4].

To generate a stochastic sequence, a stochastic number generator (SNG) as shown in Fig. 1 is utilized for converting each real number. In the SNG, the probability p associated with the real number is set to the total number (in binary format) of 1s in the sequence to be generated, *i.e.*, $p \cdot N$ where N is the sequence length. Through comparing this binary number with the pseudorandom numbers generated by a random number generator (RNG), the bit stream is obtained after N clock cycles. Typically, the RNG is implemented by a Linear Feedback Shift Register (LFSR), but other techniques that improve the distribution of bits have also been developed as discussed in Section VII. Since there are always many numbers being computed in a system, a considerable number of SNGs is required for generating uncorrelated sequences, which becomes a major fraction of the hardware circuits (*e.g.*, the SNG circuitry can reach up to 80% of the overall area in some reported designs). Therefore, some design schemes have been investigated for implementing the SNGs, for example, sharing the RNG for sequences that do not participate in the same arithmetic operations [24]. Once the SC system completes all computations, the PE counts the number of 1s in the final sequence to convert it back to the binary value associated with p (and the real number).

B. Uncorrelation and Correlation

Correlation between different stochastic bit streams has become a crucial issue in SC. With the advent of this technique in the sixties, correlation was originally considered as one of the main drawbacks and limitations of SC and as such it was to be eliminated, or at least mitigated. Given that most of the errors produced by SC systems come from operating the bit streams with an undesired degree of correlation, the initial strategy was to address uncorrelation as a whole, by using an independent SNG for generating each sequence, hence resulting in a severe limitation for applications with many inputs.

To avoid the propagation of the correlation errors as the signal advances along the pipeline of a combinatorial or sequential circuit, one of the possible solutions is to regenerate the correlation-corrupt stochastic signals through D flip-flops (DFFs) or registers that work as isolators. The problem of this technique is to determine the exact location to put these registers, which is not obvious.

As the research community has moved deeper into the SC-based realm, it has been found that correlation, far from always being a problem to be avoided, could also be exploited. As defined in [25], the correlation between two sequences x and y is quantified using the SC correlation coefficient $SCC(x,y)$; $SCC=0$ corresponds to completely uncorrelated sequences originating from different SNGs, while $SCC=1$ or (-1) indicates a maximum positive (negative) correlation. Depending on the degree of correlation of the input bit streams, a logic gate behaves in a completely different manner (an example can be found in the supplementary material of this paper). The treatment of correlation in designing SC primitives is discussed in the next sections when reviewing each type of arithmetic blocks.

C. Advantages and Disadvantages

The use of stochastic sequences in arithmetic computation enables a low complexity of a hardware implementation; in addition, they also inherently tolerate bit-flip errors. However, SC requires a considerable number of clock cycles to complete the sequence-based computation; the limited computational range of SC and its low data precision also lead to a degradation in computational accuracy. These issues may not be acceptable in some high-performance systems, and thus a system-level design for SC approaches using the building blocks must be considered. This will be discussed in Section VII.

III. Stochastic Multiplication

The use of SC for implementing multiplication is the best illustration of the hardware-saving feature typical of SC. Gaines introduced the basic circuitry for this operation, which is given by AND and XNOR gates for the *unipolar* and the *bipolar* codifications, respectively [1]. Although different works in the literature have introduced some other circuits for multiplication [26]–[29], the AND and XNOR gates are the primary schemes to carry out this operation.

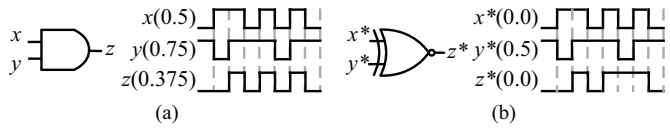

Fig. 2. Stochastic multipliers and the time diagram: (a) an AND gate for *unipolar* multiplication; (b) an XNOR gate for *bipolar* multiplication.

A. The AND Gate

The AND gate is used to implement stochastic multiplication when using the *unipolar* codification. The only constraint is that the inputs of the AND gate must be statistically uncorrelated, as shown in Fig. 2 (a), in which the input signals are $x(t)$ and $y(t)$, where t is the clock cycle and $1 \leq t \leq N$. Their associated probabilities of being in the logic high state are p_x and p_y, respectively. Since both are encoded as probabilities, the range of possible values is $[0, 1]$. Assuming independence between stochastic signals $x(t)$ and $y(t)$, we have:

$$\begin{aligned} p_z &= P(z(t) = 1) = P(x(t) = 1 \text{ and } y(t) = 1) \\ &= P(x(t) = 1)P(y(t) = 1) = p_x p_y. \end{aligned} \tag{1}$$

B. The XNOR Gate

An XNOR gate is needed to implement multiplication when the *bipolar* codification is used. Using the notation followed in Fig. 2 (b), the probability of having a logic high value at the output of an XNOR gate is given by (hereafter stochastic signals are denoted as lowercase letters without the time-dependent reference t for sake of clarity):

$$\begin{aligned} p_z &= P(x = 1)P(y = 1) + P(x = 0)P(y = 0) \\ &= p_x p_y + (1 - p_x)(1 - p_y). \end{aligned} \tag{2}$$

where we assumed independence between the two stochastic variables along with the fact that both events ($(x, y) = (1, 1)$ and $(x, y) = (0, 0)$) are mutually exclusive. By applying the variable change $p = \frac{p^*+1}{2}$ for every unipolar signal (z, x, and y) we obtain:

$$\begin{aligned} p_z &= \frac{p_z^* + 1}{2} \\ &= \frac{p_x^* + 1}{2} \cdot \frac{p_y^* + 1}{2} + (1 - \frac{p_x^* + 1}{2})(1 - \frac{p_y^* + 1}{2}) \\ &= \frac{p_x^* p_y^* + 1}{2}, \end{aligned} \tag{3}$$

which proves that the XNOR performs the product function $p_z^* = p_x^* p_y^*$.

C. Error in Stochastic Multipliers

The results provided by both *unipolar* and *bipolar* multipliers are subject to inherent variations in SC, both in the input signals of the gates and in the output provided by each gate. For the study of those variations, we first must consider that stochastic signals are binomial distributed. The binomial distribution, or binomial probability distribution, is a discrete probability distribution that counts the number of successes in a sequence of N independent trials, each with a fixed probability p of success. For situations in which the binomial distribution is applicable, only two outcomes are possible, one with a probability of occurrence p, and the other with a probability of $1-p$. This scenario is encountered in stochastic computing processes, where for N different time steps, each stochastic bit $z(t)$ has a probability p_z of being high and $1-p_z$ of being low. For this reason, the probability of having k_z high values in the stochastic signal $z(t)$ over a total of N time steps is given by the following mathematical expression:

$$P(k_z) = \binom{N}{k_z} \cdot p_z^{k_z} (1 - p_z)^{N-k_z}. \tag{4}$$

For this distribution, the expected value of the number of times when $z(t)$ is equal to 1 (that we denote as $\langle k_z \rangle$) for N samples of the signal $z(t)$ is $\langle k_z \rangle = N p_z$, that is related to the *unipolar* value codifying the stochastic signal $z(t)$ (the probability p_z).

In SC, to estimate the *unipolar* variable it is common to count the number of times a stochastic variable is equal to 1 during N cycles, such that $p_z \simeq k_z/N$ (the approximate symbol $\simeq$ is used because the number k_z is subject to variations intrinsic to the stochastic nature of the process). To derive an analytical expression for the error incurred in a specific computation, we employ the binomial distribution (4) to determine the variance associated with the ratio k_z/N used as an approximation for the *unipolar* variable p_z. This variance is related directly to the Mean Square Error (MSE):

$$\begin{aligned} MSE\left(\frac{k_z}{N}\right) = \frac{1}{M}\sum_i \left(\frac{k_z}{N} - p_z\right)^2 = \left\langle \left(\frac{k_z}{N} - \frac{\langle k_z \rangle}{N}\right)^2 \right\rangle = \\ = \frac{\langle (k_z - \langle k_z \rangle)^2 \rangle}{N^2} = \frac{VAR(k_z)}{N^2} = \frac{p_z(1 - p_z)}{N}, \end{aligned} \tag{5}$$

where the parameter M is referred to M different experiments of sequences of N bits for $z(t)$, while index i is referred to a specific sequence. We also consider that the variance of k_z for a binomial-distributed sequence is $VAR(k_z) = N p_z(1 - p_z)$.

The MSE for the *unipolar* multiplication can be obtained by considering that the signal $z(t)$ (and its unipolar value p_z) is generated by an AND gate with independent random inputs $x(t)$ and $y(t)$ so that $p_z = p_x p_y$. In this sense, we have:

$$MSE(p_x, p_y) = \frac{p_x p_y (1 - p_x p_y)}{N}. \tag{6}$$

Expression (6), can be averaged over all possible values of p_x and p_y to provide an input-independent MSE value:

$$MSE = \int_0^1 \int_0^1 MSE(p_x, p_y)\, dp_x dp_y = \frac{5}{36N}. \tag{7}$$

Similarly to the *unipolar* case, when using the *bipolar* codification, we estimate the p_z^* value as $p_z^* \simeq 2k_z/N - 1$. This is obviously subject to variations due to the intrinsic stochasticity of the process.

To estimate the MSE, as per $MSE = \langle (2k_z/N - 1 - \langle (2k_z/N - 1) \rangle)^2 \rangle$ we have:

$$MSE(p_z) = \frac{4\langle (k_z - \langle k_z \rangle)^2 \rangle}{N^2} = \frac{4p_z(1-p_z)}{N}. \quad (8)$$

Since the *bipolar* multiplier is implemented using an XNOR gate, probability p_z is $p_z = p_x p_y + (1-p_x)(1-p_y)$. Using the variable changes $p_x = (p_x^* + 1)/2$ and $p_y = (p_y^* + 1)/2$, and after few algebraic manipulations we have:

$$MSE(p_x^*, p_y^*) = \frac{4p_z(1-p_z)}{N} = \frac{1 - \left(p_x^* p_y^*\right)^2}{N}. \quad (9)$$

Then, the averaged MSE value over all the possible *bipolar* values is obtained as:

$$MSE = \frac{1}{4}\int_{-1}^{1}\int_{-1}^{1} MSE(p_x^*, p_y^*)\, dp_x^* dp_y^* = \frac{8}{9N}. \quad (10)$$

When considering comparing the precision of both product methodologies (*bipolar* and *unipolar*), it is useful to use the relative error, that is defined as the ratio between the MSE square root and the typical margin of variation of the coding ($R_e = \sqrt{MSE}/Margin$), where this margin is equal to 1 for *unipolar* (since p_z signals are defined between 0 and 1) and 2 for *bipolar* (since the range is now between -1 and $+1$). In Fig. 3 (a) we show the relative error vs. N for both numerical simulations (symbols) and the proposed analytical expressions (lines) obtained in (7) and (10). As can be appreciated, *bipolar* coding provides a larger relative error (being $\sqrt{8/5}$ times larger than the *unipolar*). This figure also shows numerical results of both products (circles for *unipolar* and triangles for *bipolar*) averaged over 1.68 millions of different simulations for each N between 4 and 256; a perfect match is shown between numerical simulations and analytical expressions.

All these findings are highly relevant when considering SC implementations of neural networks. Specifically, when any input is zero the MSE reaches its maximum value ($MSE(p_x^* \cdot p_y^*{==}0) = 1/N$) in the *bipolar* case and its minimum value ($MSE(p_x \cdot p_y{==}0) = 0$) in the *unipolar* case. This observation carries significance due to the common post-training scenario of neural networks, in which weight distributions tend to follow a Gaussian pattern with a mean value of zero. Consequently, a substantial number of product computations involve input signals that are either zero or very close to zero. In addition to the higher error associated with *bipolar* coding for near-zero signals, there is an additional concern. For *bipolar* zero values, the maximum power dissipation occurs since the switching activity peaks (with a 50% probability of being in the high or low state). Instead, a *unipolar* signal with a zero value corresponds to a bit without switching activity (and therefore zero power dissipation). Therefore, the recommendation leans towards the adoption of *unipolar* encoding when implementing neural networks, as discussed in earlier works [4], [30].

IV. Stochastic Addition

An accurate implementation of stochastic addition is still a challenge, and different circuits have been proposed to achieve this task. Fig. 4 shows three of them: a simple OR gate, a multiplexer, and an Accumulative Parallel Counter (APC).

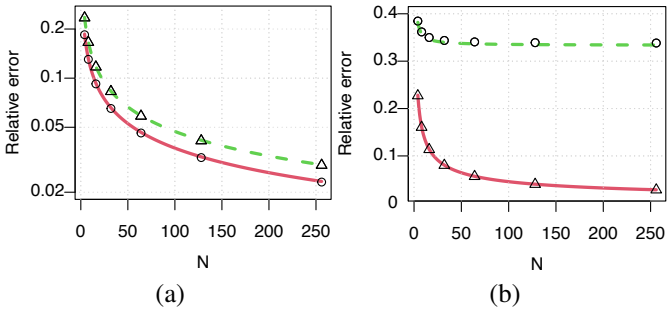

Fig. 3. The relative error's dependence on N is observed in the following cases: (a) for both *unipolar* (solid line) and *bipolar* (dashed line) multipliers, and (b) for additive functions such as the OR gate (dashed line) and the multiplexer utilized as a mean value estimator (solid line).

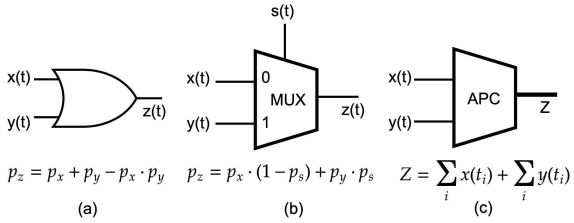

Fig. 4. Stochastic addition circuits: (a) an OR gate, where $p_x \cdot p_y$ must be close to zero in order to compute the addition accurately. (b) a multiplexer, where the accuracy is dependent on the number of inputs. (c) an Accumulative Parallel Counter (APC), where the accuracy is not degraded, and the output is represented in two's complement codification.

A. The OR Gate

The use of an OR gate as an adder (Fig. 4 (a)) is the smallest circuit in terms of hardware footprint. Assuming that the *unipolar* stochastic inputs x and y are uncorrelated, the probability of having a logic high at the output is:

$$\begin{aligned} p_z &= P(z=1) = P(x=1 \text{ or } y=1) \\ &= P(x=1) + P(y=1) - P(x=1)P(y=1) \\ &= p_x + p_y - p_x p_y. \end{aligned} \quad (11)$$

So, for a reasonable accuracy in the addition, both probabilities of p_x and p_y must be close to zero to neglect the $p_x p_y$ factor. Hence, this circuit is unable to perform the sum correctly, specially when numbers to be added are relatively large. This is the reason for its limited use as a stochastic addition circuit in most applications [31] unless a negative correlation can be guaranteed between its inputs [32], [33] or a splitting representation is used [34].

B. The Multiplexer

The use of a multiplexer (Fig. 4 (b)) is one of the most popular ways to estimate the stochastic addition [21], [31]. This type of circuitry presents a low cost in terms of area. However, its inaccuracy increases as the number of inputs grows. The multiplexer has a control signal (the selector) that must be uncorrelated with respect to the inputs. An example is shown in Fig. 4 (b), in this case the output is given by:

$$\begin{aligned} p_z &= P(z=1) = P(s=0)P(x=1) + P(s=1)P(y=1) \\ &= p_x \cdot (1-p_s) + p_y \cdot p_s \end{aligned} \quad (12)$$

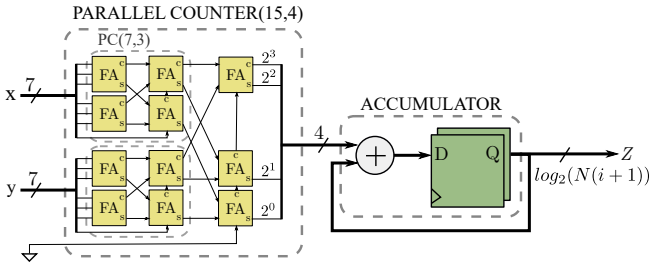

Fig. 5. A 15-input Accumulative Parallel Counter (APC).

The multiplexer performs the weighted sum of the inputs as per the selector. Moreover, to get a reasonable accuracy, the number of inputs must be relatively small, and a long period of evaluation is needed. Though this type of adder has been historically considered *correlation-insensitive*, a deeper analysis in its error sources has led to a new type of multiplexer adders called *Correlation-enhanced multiplexers* (CeMux). By applying new techniques such as precise sampling and full correlation to mitigate the error sources, CeMux outperforms the conventional weighted multiplexer adders in terms of area cost, accuracy and latency, even for large input sizes [35].

C. The Accumulative Parallel Counter

The APC (Fig. 4 (c)) counts the number of pulses in the inputs and accumulates the counted value for a period of time [36], producing, unlike the former addition circuits, a digital two's complement output. Depending on the number of input bit streams i, the APC output size is $k = \log_2(N(i+1))$. The entire circuit is composed of an accumulator to store the result of a Parallel Counter (PC) (Fig. 5).

Different designs have been proposed for the PC such as threshold gate-based counters [37], ROM designs [38], or a Full Adder (FA) network [39]. Fig. 5 shows a 15-input FA PC network, made of two 7-input PC blocks along with a classical 3-bit adder. The PC block uses a 7-line FA network tree that is able to provide a 3-bit result (7, 3) [37]. A generalization of this design to more inputs is shown in Fig. 5 for a (15, 4) block. The inputs of the (7, 3) PC are joined into a set of 3 lines (for each FA) while the other inputs are connected to a second layer. The number of inputs must be set to $i = 2^k - 1$. Therefore, the way of generalizing the design is to add as many (7, 3) blocks as required. For an arbitrary number of inputs, we can ground the unused inputs, as shown for the 15th input signal in Fig. 5. The number of FAs needed is related to the line reduction of the system, from 3 lines to 2. The number of FAs can be expressed as $i - k$.

As shown in Fig. 5, the first FA-tree layer outputs its result with a delay of one δ, where δ is the base unit delay of a single FA. The second layer output is therefore stable at 2δ. The delay for the least significant bit of the full block is $(k-1)\delta$. For the worst case, the delay of the most significant bit is

$$\delta_{max} = (2k - 3)\delta. \tag{13}$$

The APC solution is the most accurate of these three circuits, and it is correlation-insensitive; so, it is the preferable solution for most implementations. Also, it can be considered as a type of code converter from the SC domain to the classical 2's-complement domain. The main shortcoming is that it is the most area-consuming approach.

D. Error in Stochastic Adders

Similarly to the approach followed for stochastic multipliers, the average error associated with classical stochastic adders (OR gate and multiplexer) can be estimated. For the case of the OR gate, we can state that $MSE(p_x, p_y) = \langle (k_z/N - (p_x + p_y))^2 \rangle$. For the estimate of this expression, we first calculate $\langle (k_z/N - p_z)^2 \rangle$ by considering that for the OR gate, p_z can be expressed as $p_z = p_x + p_y - p_x p_y$:

$$\left\langle \left(\frac{k_z}{N} - p_z \right)^2 \right\rangle = \left\langle \left(\frac{k_z}{N} - (p_x + p_y) + p_x p_y \right)^2 \right\rangle =$$
$$= MSE(p_x, p_y) + 2 \left\langle \left(\frac{k_z}{N} - (p_x + p_y) \right) p_x p_y \right\rangle + \langle (p_x p_y)^2 \rangle. \tag{14}$$

Equation (14) can be used to isolate the MSE value by using $\langle (k_z/N - p_z)^2 \rangle = p_z(1 - p_z)/N$, and also considering that $\langle k_z \rangle = p_z N$. After some algebraic manipulations, we have:

$$MSE(p_x, p_y) = \frac{(p_x + p_y - p_x p_y) \cdot (p_x p_y + 1 - p_x - p_y)}{N} + \\ + p_x^2 p_y^2. \tag{15}$$

This equation shows that the MSE will never converge to a zero value, even if $N = \infty$. Finally, the averaged MSE value over all possible input values can be obtained as doned in (7), obtaining $MSE(OR) = 5/(36N) + 1/9$.

For the case of the multiplexer operating as a mean value estimator, we consider (12), in which we particularize for $p_s = 0.5$ (corresponding to the mean value of the inputs). Therefore, $MSE(p_x, p_y) = \langle (k_z/N - p_z)^2 \rangle$, where $p_z = 0.5p_x + 0.5p_y$, and thus:

$$MSE(p_x, p_y) = \frac{2(p_x + p_y) - (p_x + p_y)^2}{4N}. \tag{16}$$

Finally, the averaged MSE error is obtained similarly to (7), leading to the expression $MSE(MUX) = 5/(24N)$.

These results confirm that, on average, an OR gate provides a higher error than a multiplexer when implementing an additive function.

Fig. 3 (b) shows the averaged relative error vs. N obtained by both numerical simulations and the proposed analytical expressions. The solid line and triangles are related to the results obtained for the multiplexer when using the obtained analytical formula and when performing different software simulations respectively. At the same time, for the OR gate used as an adder, the dashed line and circles are related to analytical results and simulations, respectively. Hence, the multiplexer provides a higher precision than the OR gate.

V. Stochastic Division

The divider is another fundamental block in SC [40]. Differently from the previous operations, for which a stream of N bits are computed in N clock cycles, the division usually

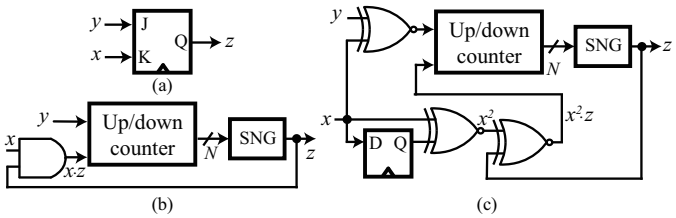

Fig. 6. The conventional stochastic dividers: (a) JK-FF; (b) ADDIE-based for $unipolar$ division; (c) ADDIE-based for $bipolar$ division.

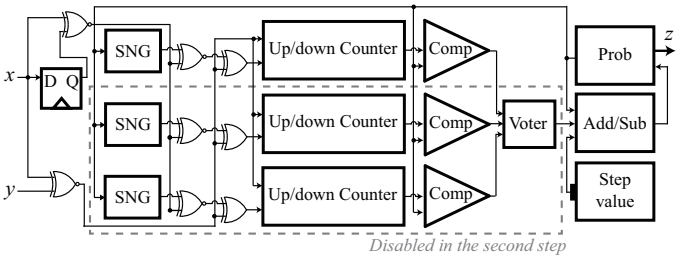

Fig. 7. The binary searching (BS) divider of [24] for $bipolar$ division. Since each module of the TMR blocks operates the same as the ADDIE-based divider, the $unipolar$ version of a BS divider can be derived as per the difference between Figs. 6 (b) and (c).

requires a larger number of cycles to achieve good accuracy. This additional latency can impact the throughput of the entire SC system and also introduce a complicated timing in the design. Therefore, significant effort has been made in recent years to design efficient stochastic dividers with both low latency and high accuracy. Some of these dividers focus on circuit designs with more efficient convergence mechanisms [24], [41]–[43], while others explore the use of correlation between input sequences [44]–[46].

A. Conventional Dividers

Initially, a JK flip-flop (FF) is considered to implement a stochastic divider [1]. As shown in Fig. 6 (a), when the JK-FF reaches a steady state, the probability of the output sequence z (calculated on the input sequences y and x) is obtained as $p_z = p_y/(p_y + p_x)$; therefore, such design can implement an approximate division if $p_y \ll p_x$.

A more accurate division is performed by employing the adaptive digital element (ADDIE) [40]. As shown in Fig. 6 (b), the ADDIE-based divider is formed by a feedback loop that keeps track of $y{\cdot}x$ by computing $x^2{\cdot}z$ for $bipolar$ computation; the quotient is then obtained when the loop converges. In the $unipolar$ case, the DFF and XNOR gates used to generate x^2 and y^x are not required (i.e., the loop keeps track of y by $x \cdot z$), and the other XNOR gates should be replaced with the AND gates (for $unipolar$ multiplication). Despite achieving higher accuracy, the ADDIE-based divider requires a large number of clock cycles to converge. For example, for a 1024-bit stream, convergence can take 46341 cycles [41]. This is due to the slow feedback loop that adjusts the quotient in small increments. In addition to latency, this conventional divider can be inaccurate when inputs are near the center of the stochastic computation range (*i.e.*, 0 for the *bipolar*); this occurs because its fundamental operation is multiplication, which in SC implementations has typically larger computational errors in that range as analyzed previously. To overcome these limitations, a number of alternative dividers have been proposed, as reviewed in the following subsections.

B. Exponential Search Dividers

A first group of alternative divider implementations [24], [41] uses the same feedback loop as in the ADDIE-based divider, but it performs an exponential rather than a linear adjustment to accelerate convergence. For example, [24] determines in the 1st iteration the half of the range that the quotient lies in, then in the 2nd iteration the half of that reduced range and so on, leading to a Binary Search (BS) on the range. In [41] this method is generalized to a Decimal Search (DS), so that at each iteration the divider determines a fraction, for example, the decimal that the quotient lies in the remaining computation range. This can further accelerate convergence.

A potential problem with this accelerated search is that an erroneous decision may be made at an iteration due to random fluctuation errors; this cannot be undone at a later iteration because subsequent iterations can only make smaller adjustments to the quotient, so it finally generates an inaccurate result. Moreover, such search method can only determine a specific value range of the quotient, despite a more refined range by the design of [41]. To avoid these issues, these dividers employ a two-step calculation. In the first step that specifies a correct value range, the dividers utilize three signals in parallel and take as the decision for an accurate adjustment the majority of them during each iteration; a triple modular redundancy (TMR) configuration is thus employed in the circuit as shown in Fig. 7. One of the TMR blocks is then utilized in the second step as a conventional divider to support finer adjustments. Note that each module of the TMR blocks operates the same as the conventional ADDIE-based divider, so the $unipolar$ version of a BS divider can easily be derived from Fig. 7 as per Section V-A and Figs. 6 (b) and (c).

With a faster convergence process, the exponential search dividers significantly reduce the required number of clock cycles to obtain an accurate quotient; however, this number is still considerable and the TMR configuration also incurs considerable hardware area and power dissipation.

C. Delta Sigma Dividers

A completely different approach to designing efficient stochastic dividers has been proposed in [42] and extended in [43]. It implements a convergence loop using a similar scheme as that in a Delta Sigma Modulator (DSM) [47]. Instead of adjusting the computed quotient based on each bit of the inputs (in the ADDIE and BS/DS dividers), the DSM-based dividers perform adjustment on a segment basis. This methodology permits a faster convergence and a higher accuracy.

Fig. 8 (a) shows a $bipolar$ divider design with a first-order DSM configuration (DSM-U1); it includes two blocks that continually estimate the dividend and divisor probabilities based on a real-time updated segment (*e.g.*, 4 bits) of the sequences as shown in Fig. 8 (b). This segment information

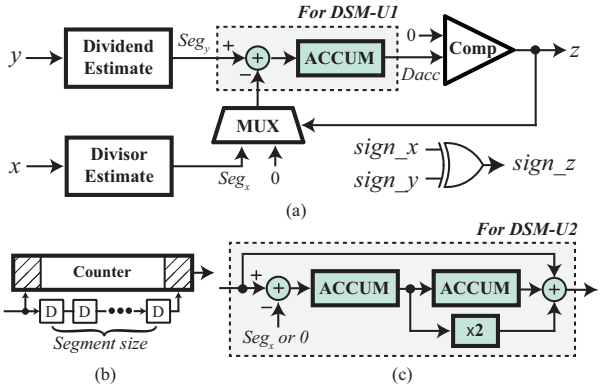

Fig. 8. The delta sigma divider of [43] for *bipolar* division: (a) DSM-U1 divider; (b) dividend/divisor estimate block; (c) accumulator block for DSM-U2 divider. The absolute value of quotient is calculated using the feedback loop that keeps $Seg_x \cdot z$ tracking Seg_y (the MUX output can be considered as $Seg_x \cdot z$ because it generates Seg_x when $z = 1$ and 0 when $z = 0$); the sign is obtained using an XOR gate as an example when the sequences are represented in the sign-magnitude format; it can be removed to perform *unipolar* division.

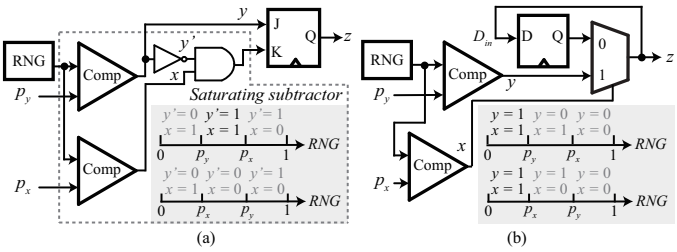

Fig. 9. The correlation-based dividers for *unipolar* division: (a) JK-FF-based design [45]; (b) DFF-based design of [46]. The *bipolar* version of these designs is unavailable in [45] but can be found or derived in [46] (which is not further discussed in this paper due to the page limitation).

(Seg_y and Seg_x, the counted number of 1s in the current segment of y and x, respectively) is used to establish a feedback loop with the DSM configuration. It is worth to note that despite the *bipolar* division, the feedback loop of DSM-based dividers is established to keep track of Seg_y by computing $Seg_x \cdot z$, which is conventionally implemented for the *unipolar* division. This is performed to avoid the so-called dead zone issue [47] in a typical DSM circuit, *i.e.*, when the input signal is within an extremely low amplitude range (the dead zone) in a DSM circuit, the feedback loop tends to incorrectly converge. Therefore, the *bipolar* divider design shown in Fig. 8 (a) only calculates the absolute value of the quotient using the *unipolar* version of a feedback loop to avoid the occurrence of a very small x^2; then, depending on the different SC representations, the sign of the quotient can be obtained using an XOR gate for the sign-magnitude sequences, or using a sign estimate block for standard *bipolar* sequences [42]. For *unipolar* division, these units used for obtaining the sign information can be directly removed.

In the DSM-based feedback loop, the difference between Seg_y and $Seg_x \cdot z$ is integrated in the accumulator block, and the integration result D_{acc} is used for quotient adjustment. If $D_{acc} \geq 0$ (so, $Seg_y \geq Seg_x \cdot z$), 0 is generated as the new bit of z, such that 0 (the smaller value between the two candidates in the MUX) is fed into the accumulator unit to avoid a further increase in the difference between Seg_y and $Seg_x \cdot z$. Otherwise, if $D_{acc} < 0$, Seg_x is fed into the accumulator to decrease the difference. Since the difference between input and output is better captured by checking each segment pair, the divider achieves a smaller number of clock cycles to converge compared to all aforementioned designs and its result is also more accurate. However, the first segment pair must be prepared prior to adjustment, which incurs additional cycles (*e.g.*, 2^4) to N. To address this issue, a second-order DSM configuration (Fig. 8 (c)) can be employed [43]. With this more complex difference-accumulation scheme, a smaller size of sequence segment can be utilized and a more refined adjustment is achieved. Finally, the design of [43] achieves full compatibility with other SC arithmetic blocks in terms of the number of clock cycles (*i.e.*, that is equal to N); moreover, its accuracy is higher than other divider designs, especially for input values centered around the SC range.

D. Correlation-based Dividers

The stochastic dividers reviewed in the previous subsections employ different convergence mechanisms using uncorrelated sequences. Alternatively, a type of stochastic dividers focuses on the mathematical variant of a division operation; the original JK-FF divider is an example of such dividers and some improved designs have been recently proposed by using the properties of correlated inputs.

For example, the divider in [45] considers the division as the format of $p_z = p_y/(p_y + (p_x - p_y)$; it implements the *unipolar* divider by integrating a JK-FF and saturating subtractor, as shown in Fig. 9 (a). When the same RNG is used to generate fully correlated inputs, the subtractor generates a sequence with probability of $max(p_x - p_y, 0)$, because it only depends on the case when $y' = 1$ and $x = 1$ for all possible values of p_y and p_x (as illustrated in Fig. 9 (a)). Another example utilizes a DFF to realize the division [46]. As shown in Fig. 9 (b), the circuit implements $p_z = p_x \cdot p(y = 1|x = 1) + (1 - p_x)D_{in}$; it can be reduced to p_y/p_x because $p(y = 1|x = 1) = min(p_y/p_x, 1)$.

Overall, these correlated dividers address the approximation feature of the original JK-FF divider by using correlated sequences and thus, they achieve a more accurate computation; moreover, they also require very low hardware overhead because only a few logic gates and one FF are required. The issue of these dividers is that the same RNG is required or the correlation between the input sequences needs to be manipulated; this may pose a challenge when combined with other (former/latter) blocks in a large SC system.

E. Comparison

To better understand the benefits and drawbacks of the different dividers, several performance metrics are summarized in Table I by taking N = 1024 as an example. In the evaluation, only uncorrelated divider designs are considered as correlated dividers cannot be easily combined with other standard SC blocks to build a system; moreover, the *bipolar* versions of

TABLE I
COMPARISON OF DIFFERENT DIVIDERS

Divider	MSE ($log_{10}()$)	Area (μm^2)	Latency (ns)	Power (μW)	# clock cycles	PLP
FP [48]	-	8248.8	100.0	10.0	50	1000
ADDIE [40]	-3.4	**635.6**	10380.3	**1.2**	46341	12456.4
BS [24]	-3.4	2770.6	1233.7	4.0	9214	4934.8
DS [41]	-3.4	13900.5	463.3	19.7	4300	9127.0
DSM-U1 [43]	-4.0	780.9	270.4	1.3	1040	**351.5**
DSM-U2 [43]	**-4.7**	1081.6	**256.0**	1.7	**1024**	435.2

these dividers are considered. The computational accuracy of different designs is evaluated in terms of the MSE for 10000 random input pairs with a uniform distribution. The hardware overhead metrics are evaluated by synthesizing the designs using an ASAP 7 nm technology library; different optimization constraints corresponding to each metric are set to obtain the best circuit area and power (with an operating frequency of 200 MHz), and the timing performance. A widely used Newton-Raphson-based floating-point (FP) divider [48] is also considered for comparison.

The results reported in Table I verify the effectiveness of stochastic dividers in terms of area overhead and power dissipation compared to a traditional (not-SC) design, as well as their issue of large computational latency. Consider all SC designs; the ADDIE-based divider is shown to have the lowest area and power, but the largest number of clock cycles and latency to complete a division. This occurs because the conventional design has the lowest circuit complexity, while its convergence process is slow. By providing the same accuracy, the BS and DS designs significantly reduce the number of clock cycles and latency, but they incur considerable area and power overhead due to the TMR configuration. The DSM-based dividers provide a very competitive performance; they achieve the best accuracy, latency and the number of clock cycles compared to the other designs, by introducing moderate area and power. Specifically, the DSM-U2 divider achieves a fully timing-compatible SC block design by requiring exactly 1024 clock cycles. Finally, as a comprehensive evaluation metric, the power latency product (PLP) results show that all DSM-based designs achieve better overall performance than the FP divider, despite the inherent issue of SC in latency.

Note that although correlation-based dividers are not compared, they can be a good candidate for implementing systems that allow correlated sequences. As per [45] and [46], these dividers also achieve a good accuracy (like the designs with quotient adjustment on a bit basis) with a low hardware cost.

VI. COMPLEX SEQUENTIAL STOCHASTIC CIRCUITS

In addition to the aforementioned stochastic circuits, more complex sequential circuits can implement functions with higher complexity, although a sequential circuit can be as simple as an FF. With only one FF, the circuit can be modeled by a 2-state FSM. When the model is generalized to contain a larger number of states, an FSM provides an efficient way to design complex stochastic circuits.

A. FSM-based Stochastic Circuits

Some early works on FSM-based stochastic circuits use the design shown in Fig. 10; it has a saturated counter followed by a combinational circuit [40], [49]. The counter is driven by a Boolean input x and outputs the current count value s in the range $\{0, 1, \ldots, n\}$; its state transition diagram (STD) is shown in Fig. 11. The combinational circuit further maps the value s to a Boolean output y through a function $y = f(s)$.

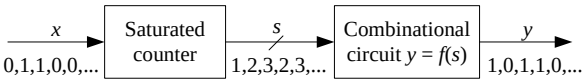

Fig. 10. The structure of a counter-based stochastic circuit.

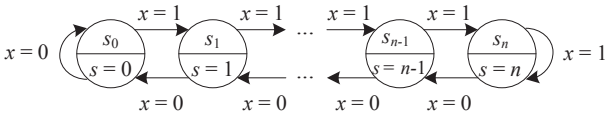

Fig. 11. The state transition diagram of a saturated counter. When the state of the counter is s_i, its output is i.

To realize an arithmetic function by SC, the input x is a stochastic bit stream encoding a value X. In this case, the state transition behavior of the counter can be modeled as a time-homogeneous Markov chain. A Markov chain has an equilibrium distribution $(\pi_0(X), \ldots, \pi_n(X))$, where $\pi_i(X)$ is the probability of the state i at equilibrium, which is a function of the input value X. Given that x is provided by a stochastic bit stream, the final output y is also a stochastic bit stream, encoding a value Y. Then, Y is an arithmetic function on X, *i.e.*, $Y = F(X)$. At equilibrium, the value of Y depends on the equilibrium distribution of the Markov chain, the function $f(s)$, and the encoding formats (*i.e.*, $unipolar$ or $bipolar$) of the input and output stochastic bit streams. For example, when the output stream uses the $unipolar$ encoding, we have

$$Y = P(y = 1) = P(f(s) = 1) = \sum_{i=0}^{n} f(i)\pi_i(X).$$

By properly choosing the function $f(s)$ and the encoding formats of the input and output stochastic bit streams, some useful arithmetic functions $F(X)$ can be realized by the design shown in Fig. 10 [40], [49]:

- **tanh function**: When the function
$$f(s) = \begin{cases} 0, & 0 \le s \le \frac{n}{2} - 1, \\ 1, & \frac{n}{2} \le s \le n - 1, \end{cases}$$
and both the input and output bit streams are in bipolar format, then $F(X) = \tanh\left(\frac{n}{2}X\right)$ [40].
- **Exponentiation function**: Let the function
$$f(s) = \begin{cases} 1, & 0 \le s \le n - G - 1, \\ 0, & n - G \le s \le n - 1, \end{cases}$$
where G is a positive integer such that $G \ll n$, and use the $bipolar$ and $unipolar$ encoding for the input and

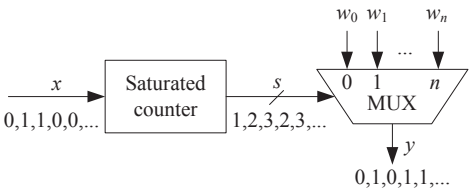

Fig. 12. A stochastic circuit based on a counter and a MUX.

output bit streams, respectively. Then, the function $F(X)$ is an exponentiation function in the following form [40]:

$$F(X) = \begin{cases} 1, & -1 \le X < 0, \\ e^{-2GX}, & 0 \le X \le 1. \end{cases}$$

- **Absolute value function**: When the function

$$f(s) = \begin{cases} 1, & s \text{ is an even number in } [0, \frac{n}{2} - 1], \text{or} \\ & s \text{ is an odd number in } [\frac{n}{2}, n-1], \\ 0, & s \text{ is an odd number in } [0, \frac{n}{2} - 1], \text{or} \\ & s \text{ is an even number in } [\frac{n}{2}, n-1], \end{cases}$$

and both the input and output bit streams are in *bipolar* format, then $F(X) = |X|$ [49].

To support the implementation of more functions, Li *et al.* have proposed the design shown in Fig. 12; this instantiates the combinational circuit in Fig. 10 as a MUX [50]. The data inputs $w_0, \ldots, w_n$ of the MUX are provided with stochastic bit streams of probabilities $W_0, \ldots, W_n$, respectively. In this case, it can be proved that the probability of the output bit y to be a 1 at equilibrium is given by

$$P(y = 1) = \sum_{i=0}^{n} W_i \pi_i(X) \tag{17}$$

Note that this design resembles the design in [51] that implements a Bernstein polynomial. It is reconfigurable through the probabilities $W_0, \ldots, W_n$.

However, it is found that the design in Fig. 12 cannot realize a wide range of functions because the linear combination of the equilibrium probability functions of the saturated counter has a limited expressive power. To improve this feature, later works replaced the counter in Fig. 12 by different FSMs [52], [53]. In [52], the counter is replaced by an FSM whose STD is a 2-dimensional mesh with an additional input. Therefore, it has an additional degree of configuration, supporting the implementation of a wider range of functions.

In [53], another FSM has been proposed to replace the counter. Its STD is shown in Fig. 13, in which the number near each arrow denotes the transition probability. The equilibrium probability distribution of this FSM can be proved as the binomial distribution, *i.e.*, $\pi_i(X) = \binom{n}{i}(1-X)^{n-i}X^i$, for $i = 0, \ldots, n$. Substituting this into (17), we can obtain the probability of the output y as

$$P(y = 1) = \sum_{i=0}^{n} W_i \binom{n}{i} (1 - X)^{n-i} X^i. \tag{18}$$

The above function is known as the Bernstein polynomial, which can approximate a wider range of functions than those supported by the design in Fig. 12.

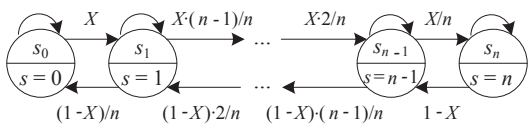

Fig. 13. The state transition diagram of the FSM of [53].

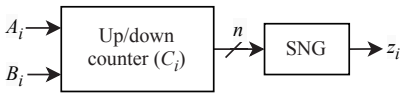

Fig. 14. A stochastic integrator.

B. Stochastic Integrator-based Stochastic Circuits

When the number of the states of a stochastic FSM is extended to infinity, the functionality of the sequential circuit can be modeled by a stochastic integrator instead of a Markov chain. A stochastic integrator consists of an up/down counter and an SNG as shown in Fig. 14. Note that the up/down counter is very similar to the saturated counter in Fig. 10, and the main difference is that it has two inputs instead of one. Ideally, the range of the counter is unlimited, and the value stored in the counter at clock cycle i, C_i, can be considered as the state of the state machine. For each clock cycle, this value is updated as follows

$$C_i = C_{i-1} + A_i - B_i, \tag{19}$$

where A_i and B_i are the input stochastic bits of 0 or 1. After k cycles of accumulation, the value is then $C_k = C_0 + \sum_{i=0}^{k-1} A_i - \sum_{i=0}^{k-1} B_i$.

Meanwhile, this value is compared with the random number generated by the SNG at each clock cycle to produce the output sequence z. Assume that the counter is N-bit wide, so of the same width as the random number, and the stochastic sequences all use the unipolar encoding. The expectation of the output sequence z can be derived as $\mathbb{E}[z_i] = C_i/2^n$. Consequently, the changing values stored in the counter can be interpreted as a digital signal, $\{c_i\} = \{C_i/2^n\}$, within $[0, 1]$. Since c_i can change from time to time, the value encoded by z_i also changes. This type of bit stream is referred to as a *dynamic stochastic sequence* (*DSS*), and the DSS $\{z_i\}$ encodes the signal $\{c_i\}$ [54]. The stochastic integrator performs numerical integration of the difference of the signals a and b, i.e.,

$$\mathbb{E}[z] = c = (\sum A_i - \sum B_i)/2^n \approx \int a - \int b, \tag{20}$$

where a and b are encoded by the input DSS's $\{A_i\}$ and $\{B_i\}$, respectively. By taking the derivative of this equation, we have

$$\frac{\mathrm{d}c}{\mathrm{d}t} \approx a - b, \tag{21}$$

which can be solved by the stochastic integrator with a step size of $1/2^n$. In [55], it is used to solve non-homogeneous ordinary differential equations (ODEs), systems of ODEs, and higher-order ODEs. A system of ODE is used to demonstrate the principle of operation as shown in Fig. 15.

In Fig. 15, "0.5" refers to a stochastic sequence with the probability of 0.5. For the left stochastic integrator, another

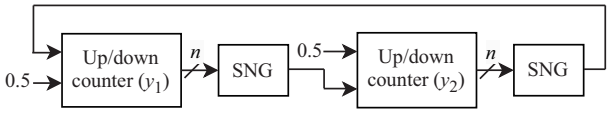

Fig. 15. An ODE solver utilizes stochastic integrators as basic element for solving a system of ODEs. The solution produced by the left stochastic integrator is denoted as y_1, while the one generated by the right one as y_2.

input is the DSS generated by the right stochastic integrator, encoding y_2. As per the formulation, the left stochastic integrator solves $\mathrm{d}y_1/\mathrm{d}t = y_2 - 0.5$. Similarly, the right stochastic integrator solves $\mathrm{d}y_2/\mathrm{d}t = 0.5 - y_1$. The stochastic integrators are cross-coupled to solve a system of ODEs. The analytical solution is a pair of sine waves, and the stochastic integrators are able to generate the results with a high accuracy.

Instead of waiting for processing the entire stochastic sequence to obtain the computation results as in a conventional SC, the counter directly provides the integration results at each clock cycle. The speed and energy efficiency is largely improved with a limited accuracy loss [55]. Table II compares a stochastic ODE solver with a binary one implementing the Runge-Kutta 2 method [56]. The clock cycle is calculated with respect to generating a full cycle of the sine wave. The hardware efficiency is evaluated using the same synthesis method as in Section V-D.

TABLE II
STOCHASTIC VS. BINARY ODE SOLVER

Circuits	**MSE** ($log_{10}()$)	**Area** (μm^2)	**Latency** (ns)	**Power** (μW)	**# clock cycles**
Stochastic	-2.3	**939.2**	**482.1**	**1.6**	1607
Binary	**-2.4**	3254.9	964.2	4.3	1607

A widely-used optimization algorithm in machine learning, gradient descent, can also be implemented by the stochastic integrator of (20). This unit provides an unbiased estimate of the optimization result through an iterative accumulation of the input DSS encoding the gradient [57]. The factor $1/2^n$ can be considered as the learning rate of the algorithm. The circuit in Fig. 16 shows the gradient descent circuit for neural network training. The inputs δ_+ and δ_- are the DSS's encoding the positive and negative local gradients, respectively. The input y is the DSS encoding the output of the previous layer in the neural network. The sign-magnitude encoding is used (one bit indicates the sign and another bit encodes the magnitude). The signed stochastic multiplier and integrator operate together to first obtain the gradient $(\delta_+ - \delta_-)y$, and then perform the accumulation. The optimized weight is then obtained by this accumulation. This design has succeeded in training a multilayer perceptron of size 784-128-128-10 with an accuracy of over 97% for the MNIST dataset, while providing around 90% energy and 75% latency savings compared to its conventional binary computing counterpart [57].

To further improve the training capability, gradient descent with momentum (GDM) can be implemented by cascading two stochastic integrators as shown in Fig. 17 [54]. To maintain a high accuracy, the first stochastic integrator uses a hybrid design, *i.e.*, g, a fixed-point number denoting the gradient, is

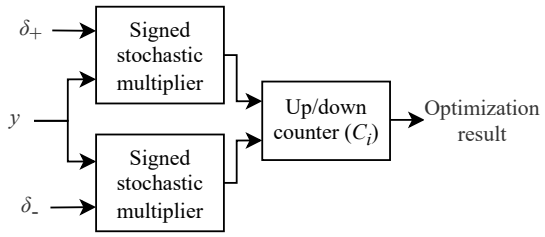

Fig. 16. A stochastic gradient descent circuit for neural network training [57].

added to the value stored in the counter, while the negative feedback is a DSS. It implements a moving average circuit, computing the velocity in the GDM algorithm. As a result, the output is a DSS encoding the velocity. After this computation, the second stochastic integrator performs an iterative accumulation of the velocity to obtain the optimized weight. With the second-order GDM algorithm, the circuit can train other complex neural networks such as VGG16, ResNet18 and MobileNet-V2 at an accuracy similar to the floating-point software implementation, as shown in Table III.

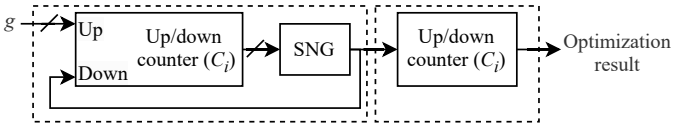

Fig. 17. A stochastic GDM circuit for neural network training. The first stochastic integrator computes the exponentially weighted moving average of the gradients, which is the velocity. The second stochastic integrator is used for velocity accumulation and estimates the optimized weight [54].

TABLE III
TEST ACCURACY: STOCHASTIC VS. FLOATING-POINT GDM USING DIFFERENT NEURAL NETWORKS [54]

Test Accuracy (%)	**Stochastic**	**Floating-point**
VGG16	90.23	90.55
ResNet18	91.36	91.85
MobileNet-V2	88.51	88.82

VII. EMERGING SC SOLUTIONS

As introduced previously, SC is a promising solution to implement computing systems with a very large number of arithmetic operations. However, even though the building blocks have in general a lower hardware complexity, the arrangement of a large number of these blocks poses some challenges when designing a complete SC system. The main challenges and some emerging SC solutions that can potentially address these challenges, are discussed next.

The first challenge is about computational accuracy. In general, ideal conditions for performing accurate computation in SC include the uniformly distributed bits in each stochastic sequence and uncorrelation among different sequences. However, these features degrade after a large number of computing stages, so finally leading to an accuracy loss. Even though correlation can be explored and manipulated to break the constraint of uncorrelation [25], [33], there are likely difficulties when building a large SC system.

A more general solution is to improve the randomness of bits. This can be achieved by generating sequences using low-discrepancy (LD) quasi-random numbers (*e.g.*, Sobol or Halton [58]). Specifically, such types of sequences can either increase accuracy when using the same sequence length as the traditional LFSR-based sequences, or they can achieve a reduction of sequence length when providing the same accuracy. An alternative solution to address the randomness issue is to use deterministic sequences [59], [60]. By arranging the position of bits following a specific rule, a completely accurate computation can be achieved (but with long sequences).

Another reason for the accuracy loss of an SC implementation is its limited computational range. Extended stochastic logic (ESL) units have been proposed for some blocks, such as multipliers and adders [61]. By utilizing two sequences and taking the ratio to represent a real number, ESL units can extend the *unipolar* coding range from [0, 1] to [0, $N/2$] and the *bipolar* range from [-1, 1] to ($-N/2$, $N/2$), respectively. The drawback is the increase in hardware overhead, because the size of ESL multipliers or adders is approximately doubled compared to a standard version. As a trade-off, ESL units can be employed for some computations and standard SC units for the remaining ones [24], [41]. In this case, a divider is required between the two parts to convert the outcome of an ESL unit (*i.e.*, two sequences) to a single sequence.

In addition to accuracy, computational latency is another challenge when designing SC systems. To guarantee a good accuracy, a considerable sequence length is usually required, which incurs a significant latency to complete the entire computation. This may also offset the advantage of SC in power dissipation, resulting in a potential large energy. Therefore, for some building blocks that often require an additional number of clock cycles compared to N (*e.g.*, a divider), a fully timing-compatible design that removes the further burden on timing (even by trading-off some other metrics), can be beneficial. Moreover, a hybrid implementation scheme that combines SC and traditional computing blocks can also avoid the requirement of using long sequences; this scheme is also helpful to address the accuracy issue [43]. The DSS using a dynamically variable binary bit stream [62] is also very efficient in terms of latency and energy for implementing integration that is often required.

Overall, the use of SC to implement larger computing systems always requires a comprehensive study of both the entire framework and the building primitives. Trade-offs among computational accuracy, latency, circuit area, and power/energy dissipation need to be conducted and further investigated to meet the specific requirements of different applications.

VIII. Conclusion

Stochastic Computing (SC) has been widely utilized as a paradigm for efficient and low-power design at the nanoscale. As the fundamental units of an SC system, the design of different primitives has been extensively investigated and reviewed, including the state-of-the-art designs of multipliers, adders, and dividers, as well as FSMs and integrators. Different options and design trade-offs have been described in detail for each primitive, and an in-depth analysis for the error in classic SC gates is also proposed. An interesting direction for future research is the study of the implementation of the SC primitives using alternative nanotechnology-based devices, structures, or circuits. The DSM-based dividers described in this paper are an example of the influence of a low-level analog design on SC. Similar ideas can potentially be developed from a nanotechnology perspective, leading to more efficient implementations of SC systems.

References

[1] B. R. Gaines, "Stochastic computing systems," *Advances in information systems science*, vol. 2, no. 2, pp. 37–172, 1969.

[2] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan, "Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 2016, pp. 678–681.

[3] Y. Liu, Y. Wang, F. Lombardi, and J. Han, "An energy-efficient online-learning stochastic computational deep belief network," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 454–465, 2018.

[4] A. Zhakatayev, S. Lee, H. Sim, and J. Lee, "Sign-magnitude sc: getting 10x accuracy for free in stochastic computing for deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[5] C. F. Frasser, P. Linares-Serrano, I. D. de los Ríos, A. Morán, E. S. Skibinsky-Gitlin, J. Font-Rosselló, V. Canals, M. Roca, T. Serrano-Gotarredona, and J. L. Rosselló, "Fully parallel stochastic computing hardware implementation of convolutional neural networks for edge computing applications," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2022.

[6] N. Onizawa, S. Koshita, and T. Hanyu, "Scaled iir filter based on stochastic computation," in *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2015, pp. 1–4.

[7] H. Ichihara, T. Sugino, S. Ishii, T. Iwagaki, and T. Inoue, "Compact and accurate digital filters based on stochastic computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 1, pp. 31–43, 2016.

[8] M. H. 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*, vol. 24, no. 2, pp. 808–812, 2015.

[9] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–6.

[10] B. Yuan and K. K. Parhi, "Belief propagation decoding of polar codes using stochastic computing," in *2016 IEEE International symposium on circuits and systems (ISCAS)*. IEEE, 2016, pp. 157–160.

[11] G. Sarkis, S. Hemati, S. Mannor, and W. J. Gross, "Stochastic decoding of ldpc codes over gf (q)," *IEEE transactions on communications*, vol. 61, no. 3, pp. 939–950, 2013.

[12] Z. Chen, H. Zhu, E. Ren, Z. Liu, K. Jia, L. Luo, X. Zhang, Q. Wei, F. Qiao, X. Liu *et al.*, "Processing near sensor architecture in mixed-signal domain with cmos image sensor of convolutional-kernel-readout method," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 2, pp. 389–400, 2019.

[13] D. Fick, G. Kim, A. Wang, D. Blaauw, and D. Sylvester, "Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2d edge detection and noise filtering," in *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*. IEEE, 2014, pp. 1–4.

[14] 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 International Conference on Nanotechnology (IEEE-NANO)*. IEEE, 2017, pp. 855–860.

[15] P. Knag, W. Lu, and Z. Zhang, "A native stochastic computing architecture enabled by memristors," *IEEE Transactions on Nanotechnology*, vol. 13, no. 2, pp. 283–293, 2014.

[16] S. Bhuin, A. K. Biswas, and L. Pileggi, "Strained mtjs with latch-based sensing for stochastic computing," in *2017 IEEE 17th International Conference on Nanotechnology (IEEE-NANO)*. IEEE, 2017, pp. 1027–1030.

[17] R. Perricone, Y. Liu, A. Dingler, X. S. Hu, and M. Niemier, "Design of stochastic computing circuits using nanomagnetic logic," *IEEE Transactions on Nanotechnology*, vol. 15, no. 2, pp. 179–187, 2015.

[18] H. Zhang, D. Zhu, W. Kang, Y. Zhang, and W. Zhao, "Stochastic computing implemented by skyrmionic logic devices," *Physical Review Applied*, vol. 13, no. 5, p. 054049, 2020.

[19] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded computing systems (TECS)*, vol. 12, no. 2s, pp. 1–19, 2013.

[20] E. Baker, P. Barbillon, A. Fadikar, R. B. Gramacy, R. Herbei, D. Higdon, J. Huang, L. R. Johnson, P. Ma, A. Mondal *et al.*, "Analyzing stochastic computer models: A review with opportunities," *Statistical Science*, vol. 37, no. 1, pp. 64–89, 2022.

[21] Z. Lin, G. Xie, S. Wang, J. Han, and Y. Zhang, "A review of deterministic approaches to stochastic computing," in *2021 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2021, pp. 1–6.

[22] M. Alawad and M. Lin, "Survey of stochastic-based computation paradigms," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 1, pp. 98–114, 2016.

[23] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A survey of stochastic computing neural networks for machine learning applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 7, pp. 2809–2824, 2020.

[24] ——, "A stochastic computational multi-layer perceptron with backward propagation," *IEEE Transactions on Computers*, vol. 67, no. 9, pp. 1273–1286, 2018.

[25] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Oct 2013, pp. 39–46.

[26] Z. Li, J. Li, A. Ren, R. Cai, C. Ding, X. Qian, J. Draper, B. Yuan, J. Tang, Q. Qiu *et al.*, "Heif: Highly efficient stochastic computing-based inference framework for deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, pp. 1543–1556, 2018.

[27] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[28] Y. Zhang, L. Xie, J. Han, and G. Xie, "Highly accurate and energy efficient stochastic multipliers," in *2022 IEEE 22nd International Conference on Nanotechnology (NANO)*, 2022, pp. 12–15.

[29] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. S. Miguel, "Ugemm: Unary computing architecture for gemm applications," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 377–390.

[30] J. Yu, K. Kim, J. Lee, and K. Choi, "Accurate and efficient stochastic computing hardware for convolutional neural networks," in *Proceedings - 35th IEEE International Conference on Computer Design, ICCD 2017*, 2017, p. 105 – 112.

[31] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, "SC-DCNN: Highly-scalable deep convolutional neural network using stochastic computing," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 405–418, 2017.

[32] S. Aygun and E. O. Gunes, "Utilization of contingency tables in stochastic computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 6, pp. 2942–2946, 2022.

[33] V. T. Lee, A. Alaghi, and L. Ceze, "Correlation manipulating circuits for stochastic computing," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 1417–1422.

[34] W. Romaszkan, T. Li, T. Melton, S. Pamarti, and P. Gupta, "Acoustic: Accelerating convolutional neural networks through or-unipolar skipped stochastic computing," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 768–773.

[35] T. J. Baker and J. P. Hayes, "Cemux: Maximizing the accuracy of stochastic mux adders and an application to filter design," *ACM Transactions on Design Automation of Electronic Systems*, vol. 27, no. 3, pp. 1–26, 2022.

[36] B. Parhami and C.-H. Yeh, "Accumulative parallel counters," in *Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, vol. 2. IEEE, 1995, pp. 966–970.

[37] E. Swartzlander, "A review of large parallel counter designs," in *IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2004, pp. 89–98.

[38] E. E. Swartzlander, "Parallel counters," *IEEE Transactions on Computers*, vol. C-22, no. 11, pp. 1021–1024, 1973.

[39] B. Parhami, "Efficient hamming weight comparators for binary vectors based on accumulative and up/down parallel counters," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 2, pp. 167–171, 2009.

[40] B. D. Brown and H. C. Card, "Stochastic neural computation i: Computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, p. 891–905, sep 2001.

[41] S. Liu, X. Tang, F. Niknia, P. Reviriego, W. Liu, A. Louri, and F. Lombardi, "Stochastic dividers for low latency neural networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 10, pp. 4102–4115, 2021.

[42] X. Tang, S. Liu, F. Niknia, P. Reviriego, Z. Wang, W. Tang, A. Louri, and F. Lombardi, "A delta sigma modulator-based stochastic divider," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 8, pp. 3272–3283, 2022.

[43] X. Tang, S. Liu, F. Niknia, Z. Wang, S. Liu, P. Reviriego, and F. Lombardi, "Delta sigma modulator-based dividers for accurate and low latency stochastic computing systems," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 13, no. 1, pp. 270–284, 2023.

[44] D. Wu and J. S. Miguel, "In-stream stochastic division and square root via correlation," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[45] S.-I. Chu, "New divider design for stochastic computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 1, pp. 147–151, 2020.

[46] T.-H. Chen and J. P. Hayes, "Design of division circuits for stochastic computing," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 116–121.

[47] R. Schreier, G. C. Temes *et al.*, *Understanding delta-sigma data converters*. IEEE press Piscataway, NJ, 2005, vol. 74.

[48] Y. Li *et al.*, *Computer principles and design in Verilog HDL*. John Wiley & Sons, 2015.

[49] P. Li, D. J. Lilja, W. Qian, M. D. Riedel, and K. Bazargan, "Logical computation on stochastic bit streams with linear finite-state machines," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1473–1485, 2014.

[50] P. Li, W. Qian, M. Riedel, K. Bazargan, and D. Lilja, "The synthesis of linear finite state machine-based stochastic computational elements," in *Asia and South Pacific Design Automation Conference*, 2012, pp. 757–762.

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

[52] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," in *International Conference on Computer-Aided Design*, 2012, pp. 480–487.

[53] N. Saraf and K. Bazargan, "Polynomial arithmetic using sequential stochastic logic," in *Great Lakes Symposium on VLSI*, 2016, pp. 245–250.

[54] S. Liu and W. J. Gross, "Gradient descent with momentum using dynamic stochastic computing," in *ICLR 2021 Hardware Aware Efficient Training Workshop*, 2021, pp. 1–5.

[55] S. Liu and J. Han, "Hardware ODE solvers using stochastic circuits," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[56] J. C. Butcher, *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. Wiley-Interscience, 1987.

[57] S. Liu, H. Jiang, L. Liu, and J. Han, "Gradient descent using stochastic circuits for efficient training of learning machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2530–2541, 2018.

[58] S. Liu and J. Han, "Toward energy-efficient stochastic circuits using parallel sobol sequences," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1326–1339, 2018.

[59] M. H. Najafi, D. J. Lilja, and M. Riedel, "Deterministic methods for stochastic computing using low-discrepancy sequences," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2018, pp. 1–8.

[60] Y. Kiran and M. Riedel, "A scalable, deterministic approach to stochastic computing," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, 2022, pp. 45–51.

[61] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rosselló, "A new stochastic computing methodology for efficient neural network implementation," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 3, pp. 551–564, 2015.

[62] S. Liu, W. J. Gross, and J. Han, "Introduction to dynamic stochastic computing," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 19–33, 2020.

SUPPLEMENTARY MATERIAL

A. Impact of Correlation on Logic Behavior

This subsection analyzes the impact of uncorrelation/correlation between sequences on the behavior of a logic gate in SC; specifically, the example of an XOR gate is considered next, but similar analytical expressions can also be found out for AND, OR or XNOR gates as a function of the correlation of their inputs [5].

As shown in Fig. S1, with correlated inputs ($R_x = R_y$) an XOR gate performs the absolute subtraction of the input probabilities; instead, when the inputs are uncorrelated ($R_x \neq R_y$) the XOR gate performs the function $(1-p_x)p_y + p_x(1-p_y)$.

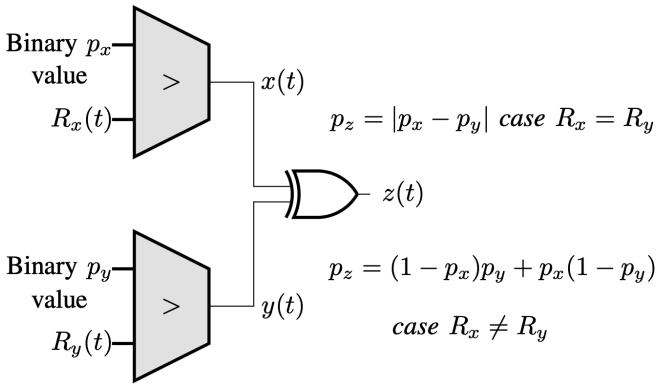

Fig. S1. Correlation impact on the behavior of an XOR gate in SC. Stochastic signals x and y are considered to be completely correlated when they share the same random number ($R_x = R_y$) in the SNG, producing the function $|p_x - p_y|$. By contrast, if $R_x \neq R_y$ the sequences are uncorrelated and the output function is different: $p_z = (1-p_x)p_y + p_x(1-p_y)$.

To obtain the above analytical expressions in *unipolar* format, the two diagrams shown in Fig. S2 are considered. The two $[0,1] \times [0,1]$ squares represent a surface domain with all possible R_x and R_y values (that fluctuate with time), given fixed binary p_x and p_y values. We assume that all these unsigned n-bit binary values are all defined in the interval between 0 ($0.000\ldots$) and 1 ($0.111\ldots$). When $x(t)$ and $y(t)$ are correlated, R_x, R_y, p_x and p_y share the same axis (Fig. S2 (a)). According to the comparison of p_x with R_x and p_y with R_y, as per the operations of the comparator of the SNG (Fig. S1 (a)), we obtain several rectangular areas (which are numerically related to their probabilities) with all possible combinations of the input value for random variable R_x (that is equal to R_y). The XOR outputs a value of $z = 1$ when the two inputs are different ($x = 1, y = 0$), while $z = 0$ otherwise. For the correlated case, this area, in green color, is equal to $p_z = max(p_x, p_y) - min(p_x, p_y)$ that yields $p_z = |p_x - p_y|$ by taking into consideration a sweep of all possible R_x and R_y values.

By contrast, when the two inputs are completely uncorrelated, as shown in Fig. S2 (b), we represent the p_x and R_x values on the X-axis and the p_y and R_y on the Y-axis, respectively. In this way, we obtain four different areas as a function of the pair (p_x, p_y) values. Since the XOR has an output of one when $x(t)$ and $y(t)$ are different (01, 10), the overall area in green color is proportional to the probability of having a one at the output z (probability p_z). This total area is the sum of the two green-shadowed rectangles, so $p_z = (1-p_x)p_y + p_x(1-p_y)$.

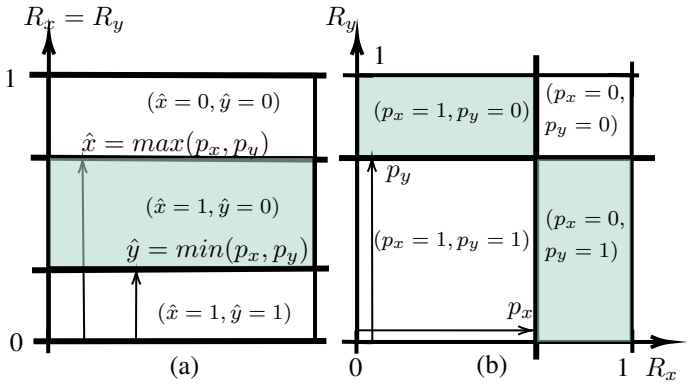

Fig. S2. Diagrams for the stochastic functions for an XOR gate: (a) A generic case in which $\hat{x}$ represent the maximum value of input signal while $\hat{y}$ is the minimum. The green-shadowed area is $max(p_x, p_y) - min(p_x, p_y)$ which turns out to be $|p_x - p_y|$ in general. (b) R_x and R_y are different and therefore represented on different axes. The diagram shows a stochastic case for certain couple of (p_x,p_y) values and the result after having been compared with R_x and R_y, respectively. The overall green-shadowed area is the addition of the two rectangular areas $(1-p_x)p_y$ and $p_x(1-p_y)$, producing a probability at the output of the XOR gate equal to $(1-p_x)p_y + p_x(1-p_y)$.

The previous analysis is correct when stochastic signals are either completely uncorrelated (since the random numbers used in the conversion are different), or completely correlated (when the same random number is used to generate both bit streams). Moreover, as negative correlations can be generated between stochastic signals (as when $R_y = -R_x$), for which the previous analysis is not applicable without few modifications.