Introduction to Dynamic Stochastic Computing

Siting Liu
Department of Electrical and Computer Engineering
McGill University
Montreal, QC H3A 0E9, Canada
Email: siting.liu@mail.mcgill.ca

Warren J. Gross
Department of Electrical and Computer Engineering
McGill University
Montreal, QC H3A 0E9, Canada
Email: warren.gross@mcgill.ca

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

***Abstract*—Stochastic computing (SC) is an old but reviving computing paradigm for its simple data path that can perform various arithmetic operations. It allows for low power implementation, which would otherwise be complex using the conventional positional binary coding. In SC, a number is encoded by a random bit stream of '0's and '1's with an equal weight for every bit. However, a long bit stream is usually required to achieve a high accuracy. This requirement inevitably incurs a long latency and high energy consumption in an SC system. In this article, we present a new type of stochastic computing that uses dynamically variable bit streams, which is, therefore, referred to as dynamic stochastic computing (DSC). In DSC, a random bit is used to encode a single value from a digital signal. A sequence of such random bits is referred to as a dynamic stochastic sequence. Using a stochastic integrator, DSC is well suited for implementing accumulation-based iterative algorithms such as numerical integration and gradient descent. The underlying mathematical models are formulated for functional analysis and error estimation. A DSC system features a higher energy efficiency than conventional computing using a fixed-point representation with a power consumption as low as conventional SC. It is potentially useful in a broad spectrum of applications including signal processing, numerical integration and machine learning.**

I. Introduction

As an unconventional computing paradigm, stochastic computing (SC) operates on numbers encoded by sequences of random '0's and '1's [1]. The probability of a '1' in the stochastic sequence is then used to encode a number. The datapath of a typical SC system consists of three major components: stochastic number generators (SNGs), stochastic circuits and probability estimators (PEs), as shown in Fig. 1. Complex computation can be performed by the stochastic circuits in relatively simple bit-wise operations, while the SNG array and PEs are used to convert numbers between the conventional binary representation and stochastic encoding. SC has been studied for decades for its relatively simple logic, low power consumption, an inherent resilience to bit-flip errors and biological plausibility.

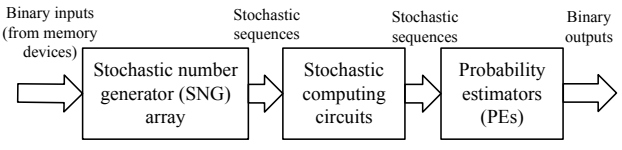
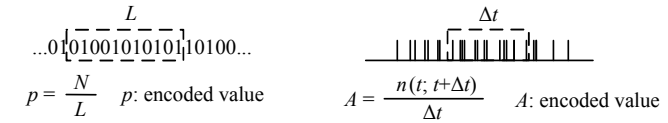


Fig. 1: Datapath of a stochastic computing system.



(a) Stochastic unipolar encoding. (b) Rate coding.

Fig. 2: (a) Stochastic encoding vs. (b) rate coding.

In a stochastic sequence, a number, x, can be simply encoded as the probability of a '1' or, approximately, as the ratio of the number of '1's in the total number of bits. An inverter then computes $1 - x$ for an input sequence encoding x. For two independent input sequences encoding x and y, an AND gate outputs a '1' when both the input bits are '1's, so the probability of a '1' in the output sequence is equal to the product of the probabilities of '1's of the two input stochastic sequences, i.e., xy, whereas in conventional arithmetic, a multiplier requires a complex circuit.

Since numbers are represented by redundant random bits, a bit-flip usually incurs a relatively small error in the final result, so SC is fault-tolerant and robust against moderate operational errors. However, if a bit flip occurs at a more significant bit in a conventional circuit, it can introduce a large error [2], [3].

In fact, a stochastic sequence works in a similar manner to the rate coding [4] in a spiking neuron, as shown in Fig. 2. With this coding, the frequency of spikes during a time interval carries the information in a neuron [5], similar to the stochastic encoding using the frequency of the occurrence of '1' in SC. Inspired by the neuronal coding, SC has been used to implement various machine learning algorithms [6]–[17].

To achieve a high accuracy, however, a long sequence is often required in SC due to stochastic fluctuations. A long sequence in turn implies a high latency and a high energy consumption. In addition, the costly SNGs and PEs can account for a major part of the circuits in an SC system [18].

To address this issue, we noticed that a stochastic sequence can carry the information of a varying signal with each bit encoding a value from the signal. This new type of sequences is referred to as a dynamic stochastic sequence (DSS). In

a DSS, the sequence length encoding each number is just 1 bit, so it can significantly improve the performance and energy efficiency of a stochastic circuit. In this article, we present dynamic stochastic computing (DSC) using DSS's and discuss the efficient implementation of iterative accumulation-based computations, including filtering, the Euler method for solving ordinary differential equations (ODEs) and the gradient descent (GD) algorithm in machine learning.

II. Background

Originally proposed in the 1960s, SC is intended to be a low-cost alternative to conventional computing. In SC, numbers are encoded by random binary bit streams or stochastic sequences. The probability of each bit being '1' is considered to be the probability encoded by a stochastic sequence and used in different mapping schemes to encode numbers within different ranges [1].

Assume that x is the number to be encoded and that p is the probability of a stochastic sequence. The simplest mapping is to let $x = p$, i.e., the probability of the stochastic sequence is used to represent a real number within [0,1]. This is referred to as the unipolar representation. In order to expand the range to include negative values, the bipolar representation takes a linear transformation of the unipolar representation by $x = (p - 0.5) \times 2$, so that the representation range is $[-1, 1]$.

Recently, a sign-magnitude representation was proposed to expand the unipolar representation range by adding an extra sign bit to a stochastic sequence [19]. The same range is obtained as the bipolar representation and the computed result is more accurate. However, it is still considered a linear mapping since it is a modified unipolar representation. A few examples of these representations are shown in Table I.

TABLE I: Examples of different SC representations

	Sequence	p	Sign	Value encoded
Unipolar	0110101110	0.6	–	0.6
Bipolar	0100110100	0.4	–	$2 \times 0.4 - 1 = -0.2$
Sign-mag.	0110101110	0.6	1	$(-1) \times 0.6 = -0.6$

A different mapping scheme may require different computing elements. For the rest of this article, the unipolar and sign-magnitude representations are adopted for its simple circuit implementation and high accuracy, unless stated otherwise.

A. Stochastic sequence generation

An SNG consists of a random number generator (RNG) and a comparator, as shown in Fig. 3. An RNG is usually implemented by a linear-feedback-shift register (LFSR). A maximum-length n-bit LFSR traverses all the integer numbers within $[1, 2^n - 1]$. The numbers generated by an LFSR are considered pseudorandom numbers because they are deterministic rather than truly random once the seed and the structure of the LFSR are determined. However, due to their statistical characteristics, deliberately generated pseudorandom numbers can be considered as uniformly distributed. If the n-bit number, used to encode a fractional number within $[0, 1]$, p, after the normalization by 2^n, is larger than an n-bit pseudorandom number, a '1' is generated; otherwise, '0' is the output. Then, the probability of generating a '1' is approximately p. The resulting stochastic sequence encodes p in the unipolar representation and x in the bipolar representation using the aforementioned linear mapping, $x = 2p - 1$. A sequence with probability p is generated similarly to encode x in this representation. A detailed description and the mathematical model for an LFSR-based SNG can be found in [20].

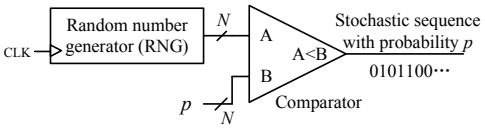


Fig. 3: An SNG.

In [21] and [22], two types of low-discrepancy (LD) sequences, the Halton and Sobol sequences, are introduced to SC. They are originally proposed to accelerate and improve the accuracy of Monte Carlo simulations by using a regular generation of quasirandom numbers, as shown in Fig. 4 [23]. Halton sequences can be generated by a binary-coded prime number-based counter (e.g., $(2120)_3$ is stored as $(10)_2(01)_2(10)_2(00)_2$ in the counter). The order of the digits in the counter is then reversed and the resulting number is converted to a binary number for generating a Halton sequence [21]. Sobol sequences can be generated by the circuit shown in Fig. 5. As per the Sobol generation algorithm [24], the index of the direction vector is first obtained by the least significant zero (LSZ) detection and index generation component. The direction vector array (DVA) is generated by using a primitive polynomial. At each clock cycle, one direction vector (DV), V_k, is fetched and XORed with the current Sobol number, R_i, to generate the next Sobol number, R_{i+1}. In SC, the use of the LD sequences improves the accuracy with a shorter length. However, a fairly long sequence is still required for a high accuracy.

B. Stochastic circuits

Stochastic circuits are the core of an SC system. The operation of a combinational circuit can be considered as

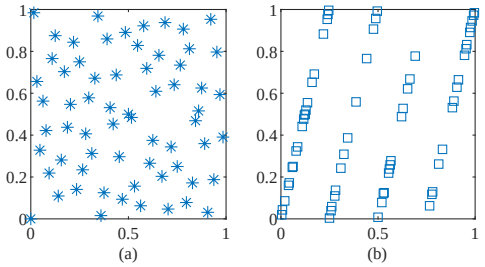


Fig. 4: Examples of two-dimensional (a) Sobol sequences, and (b) LFSR-generated sequences. The quasirandom numbers in (a) are more evenly distributed than the pseudorandom numbers in (b).

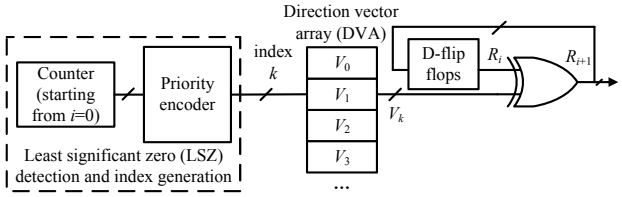


Fig. 5: A Sobol sequence generator [25]. It can be used as the RNG in an SNG to improve the accuracy.

a Monte Carlo simulation process [21]. For example, the function of a unipolar stochastic multiplier is equivalent to estimating the rectangular area determined by p_1p_2, where p_1 and p_2 are the probabilities encoded by the input sequences, by randomly dropping points into a unit square. The AND gate is used to decide whether the randomly generated point with coordinates (r_1, r_2) is within the rectangle. Only when both $r_1 < p_1$ and $r_2 < p_2$, is the random point located within the rectangle,, as shown in Fig. 6, and the output of the AND gate is '1'. The counter is then used to count the total number of points within the rectangle as an estimate of p_1p_2.

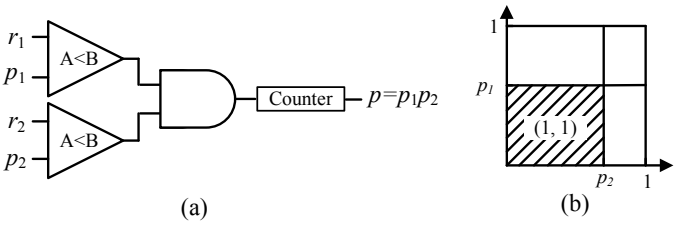


Fig. 6: (a) A unipolar stochastic multiplier and (b) its Monte Carlo model.

Fig. 7 shows several stochastic multipliers in different representations: a bipolar multiplier is implemented by an XNOR gate and the sign-magnitude stochastic multiplier is implemented by a unipolar multiplier and an XOR gate for computing the sign bit. Generic methods have also been proposed to synthesize combinational circuits for computing Bernstein polynomials [2] or multi-linear functions [26].

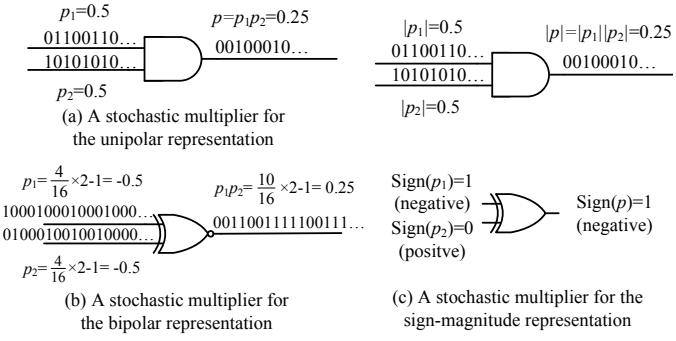


Fig. 7: Three stochastic multipliers for different representations. The input sequences should be uncorrelated for a higher accuracy.

There are primarily two types of sequential circuits, the FSM-based and stochastic integrator-based circuits. An FSM-based circuit can be modeled by a state transition diagram, as shown in Fig. 8. For an input bit of '0', the FSM moves to the left of its current state; otherwise, it moves to the right. Note that when the FSM is at the leftmost (or rightmost) state, it stays at the current state when receiving a '0' (or '1'). The FSM can be considered as a Markov chain if the input sequence is not autocorrelated or every bit in the sequence is generated independently with the same probability to be '1'. A steady hyper-state can be reached after several runs, where the probability that the FSM staying at each state converges [1]. By assigning a different output ('0' or '1') to each state, exponential, logarithmic and high-order polynomial functions can be implemented by the FSM-based circuits. A general synthesis method is discussed in [27].

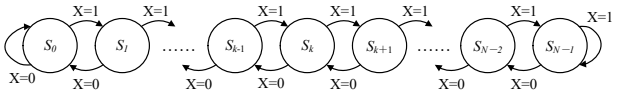


Fig. 8: State transition diagram for an FSM-based stochastic circuit.

Stochastic integrators are sequential SC elements that accumulate the difference between two input sequences [1]. The stochastic integrator consists of an RNG, an up/down counter and a comparator, as shown in Fig. 9(a). Let the output of the comparator be S_i (either '0' or '1') at clock cycle i, the input bits be A_i and B_i, and the n-bit integer stored in the counter be C_i. The RNG and the comparator work as an SNG, and the probability of generating a '1' is $c_i = C_i/2^n$, i.e., the n-bit counter stores a number in a fixed-point manner and all the bits belong to the fractional part. The counter updates its value C_i at each clock cycle by $C_{i+1} = C_i + A_i - B_i$, or, equivalently,

$$c_{i+1} = c_i + 1/2^n(A_i - B_i). \tag{1}$$

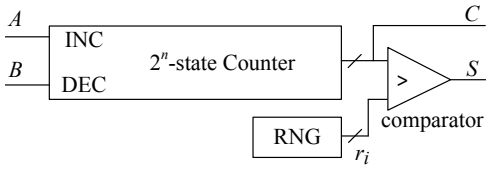
In k clock cycles, the stochastic integrator implements

$$c_k = c_0 + 1/2^n \sum_{i=0}^{k-1}(A_i - B_i), \tag{2}$$

where c_0 is the initial (fractional) value stored in the counter.

A stochastic integrator works in a similar manner to a spiking neuron, as shown in Fig. 9. The stochastic integrator "fires" randomly depending on the value stored in the counter, whereas the spiking neuron fires when the accumulated membrane potential, V_M, reaches a threshold, V_{th} [5]. The inputs to the neuron can be excitatory (ϵ_{ij}) or inhibitory (ι_{ij}), while the input sequences, A and B, of the stochastic integrator increase and decrease the value stored in the counter.

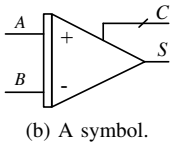
The integrator has been used in a stochastic divider, with the output stochastic sequence as the feedback signal [1], as shown in Figs. 10(a) and (b). The quotient is obtained when the value stored in the counter reaches an equilibrium state, i.e., the up/down counter has an equal probability to increase



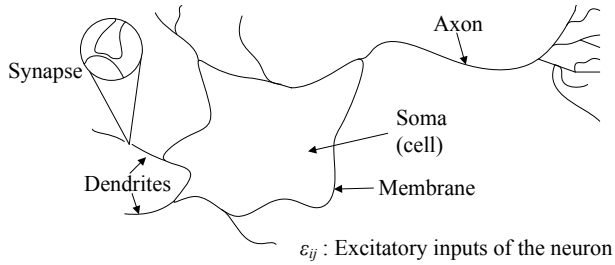
$C_{i+1} = C_i + A_i - B_i$

$\begin{cases} \text{if } C_i > r_i, S = 1 \\ \text{else} \quad , S = 0 \end{cases}$

(a) A stochastic integrator.



(b) A symbol.



$V_M(t+1) = V_M(t) + \Sigma\varepsilon_{ij} - \Sigma\iota_{ij}$

$\begin{cases} \text{if } V_M > V_{th}, \text{ reset and fire} \\ \text{else} \quad , \text{ do not fire} \end{cases}$

ε_{ij} : Excitatory inputs of the neuron
ι_{ij} : Inhibitory inputs of the neuron
V_M : Membrane potential
V_{th} : Threshold voltage

(c) An integrate-and-fire spiking neuron model.

Fig. 9: (a) A stochastic integrator, (b) its symbol and (c) a spiking neuron [5].

or decrease. However, it may take a long time to converge to the equilibrium state as discussed in [6], [25]. A stochastic integrator with a feedback loop has been used as an ADpative DIgital Element (ADDIE), shown in Fig. 10(c). The ADDIE can function as a probability follower, i.e., the probability of the output sequence tracks the change of the probability of the input sequence [1], [28].

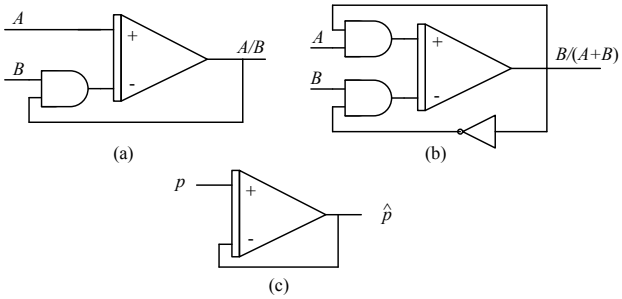


Fig. 10: Stochastic dividers compute: (a) A/B, (b) $B/(A+B)$. (c) shows an ADDIE.

C. Probability estimators

The computed probability of the output sequence can be found by using a counter. It is estimated as the quotient of the number of '1's divided by the sequence length. The ADDIE in Fig. 10(c) is an alternative component to estimate the probability by using a stochastic integrator reaching the equilibrium state [1].

D. Limitations

Although the SC circuits are much simpler than conventional arithmetic circuits, a large sequence length is required to achieve a high accuracy. This requirement leads to poor performance and low energy efficiency. Fig. 11 shows the root mean square error (RMSE) encoding a number in the unipolar representation using different types of "random" numbers. The pseudorandom numbers are generated by a 16-bit LFSR with randomly generated seeds. The RMSEs are obtained from 1,000 trials using different sequence lengths. As shown in Fig. 11, to achieve an RMSE of 1×10^{-2}, a sequence length of 2^{12} bits is required for the LFSR generated sequences. For the stochastic Sobol sequences, the length is significantly reduced to 2^6 or 64 bits to achieve a similar accuracy, although it is still not as compact as an equivalent 5-bit fixed-point number that results in a similar RMSE. Meanwhile, low precision (e.g. at an RMSE of 1×10^{-2} or larger) can be tolerated in many applications such as inference in a machine learning model [12]–[17], [29]–[33], digital signal processing [34]–[40] and image processing [41]–[44]. Hence, progressive precision (PP) can be employed and the Sobol sequence length can be as low as 8 bits in an application in a CNN inference engine [45].

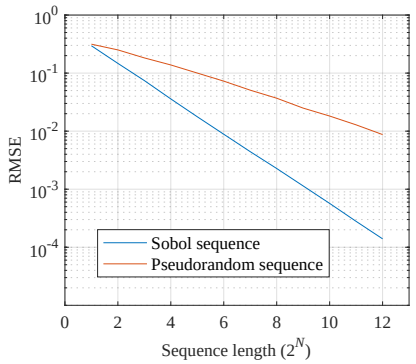


Fig. 11: RMSE of the encoded numbers with different types of stochastic sequences.

Despite the fact that some of these implementations achieve a lower energy consumption with PP compared with conventional circuits, the application of conventional SC (CSC) is still quite limited.

III. Dynamic Stochastic Computing

In a DSC system, as shown in Fig. 12, the stochastic sequences encode varying signals instead of static numbers. Consider a discrete-time digital signal $\{x_i\}$ within $[0, 1]$ ($i = 0, 1, 2, \dots$). A DSS encoding $\{x_i\}$ satisfies that $\mathbb{E}[X_i] = x_i$, where X_i is the ith bit in the DSS. Thus, every bit in the sequence can have a different probability to be 1.

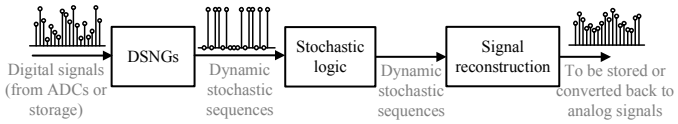


Fig. 12: A DSC system.

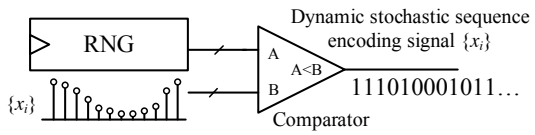


Fig. 13: A DSNG. $\{x_i\}$ is an input to the comparator at one sample per clock cycle.

A. DSS generation

A DSS can be generated by a dynamic SNG (DSNG), as shown in Fig. 13. It is similar to a conventional SNG except that the input is a varying signal instead of a static number. The RNG generates uniformly distributed random numbers within $[0, 1]$. Given a discrete signal, $\{x_i\}$, as shown in Fig. 13, it is compared with a random number one sample at a time. If x_i is larger than the random number, a '1' is generated; otherwise, the DSNG generates a '0'. Generating every bit in the DSS is a Bernoulli process (in the ideally random case), so the expectation getting a '1' is equal to the corresponding sample value from the discrete digital signal. The discrete signal can either be sampled from an analog signal or read from a memory device.

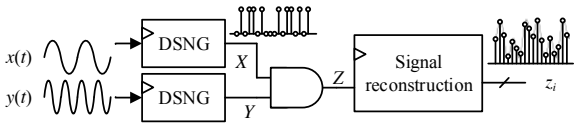
B. Data representations in DSC

Similar to CSC, all the aforementioned mapping schemes can be used to encode numbers within certain ranges in DSC, which subsequently requires the use of different computational elements. Specifically, a DSS can encode a signal within $[0, 1]$ in the unipolar representation or within $[-1, 1]$ in the bipolar or sign-magnitude representation. For the sign-magnitude representation, two sequences are used to encode a signal. One encodes the magnitude of the signal. Since the signal can have both positive and negative values, another sequence is used to represent the signs of the numbers.

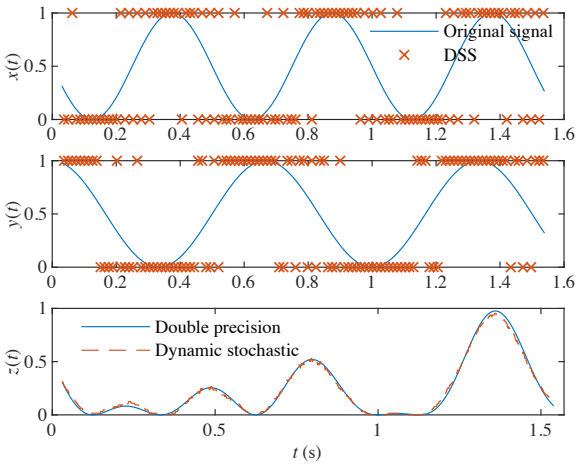
C. Stochastic circuits

The DSC circuits can be combinational or sequential.

1) Combinational circuits: A combinational circuit using DSS's as its inputs implements the composite function of the original stochastic function. For example, an AND gate, or a unipolar stochastic multiplier, computes $z_i = x_i y_i$ for each pair of bits in the input DSS's that independently encode x_i and y_i ($i = 0, 1, 2 \ldots$), respectively [46], due to $\mathbb{E}[Z_i] = \mathbb{E}[X_i Y_i] = \mathbb{E}[X_i]\mathbb{E}[Y_i] = x_i y_i$, where Z_i, X_i and Y_i are the ith bits in the output and input DSS's, respectively. Note that for a continuous signal, oversampling is required to produce the discrete signals, x_i and y_i, for accurate multiplication [46]. Fig. 14(a) shows the DSC circuit for a signal multiplication, and (b) shows the results of multiplying two sinusoidal signals



(a)



(b)

Fig. 14: A frequency mixer by using a stochastic multiplier with DSS's as the inputs: (a) circuit and (b) results.

with frequencies of 1.5 Hz and 2 Hz, both sampled at a rate of 2^{10} Hz. The sampled signals are converted to DSS's by two DSNGs with independent Sobol sequence generators [47] as the RNGs. The output DSS is reconstructed to a digital signal by a moving average [28]. The DSS can also be reconstructed by an ADDIE [1].

2) Sequential circuits: The stochastic integrator is an important component used in DSC for iterative accumulation. It implements integrations of the signal encoded by accumulating the stochastic bits in a DSS. Fig. 15 shows that a bipolar stochastic integrator can be used to perform numerical integration. In Fig. 15(a), the input digital signal is $\cos(\pi t/2)$, sampled at a sampling rate of 2^8 Hz and converted to a DSS by the DSNG. The other input of the stochastic integrator is set to a bipolar stochastic sequence encoding 0 (a stochastic bit stream with half of the bits being '1'). By accumulating the input bits, the stochastic integrator provides an unbiased estimate to the Riemann sum [48]. The integration is obtained by recording the changing values of the counter. Since the integrated results are directly provided by the counter, a reconstruction unit is not required.

As can be seen from Fig. 15(b), the results produced by the stochastic integrator is very close to the analytical results. Since the output DSS is generated by comparing the changing signal (as the integration results) with uniformly distributed random numbers, the RNG and the comparator inside the integrator works as a DSNG. As a result, the output sequence encodes the discretized stochastic integration solution for the analytical solution, $(2/\pi)\sin(\pi t/2)$. The output DSS is also shown in Fig. 15(b).

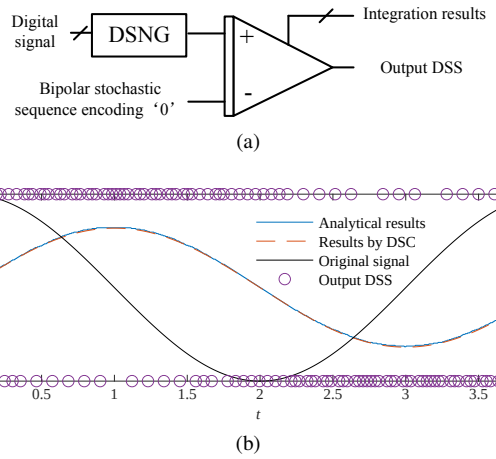


Fig. 15: A bipolar stochastic integrator for numerical integration: (a) circuit and (b) results. The output DSS is decimated for a clear view.

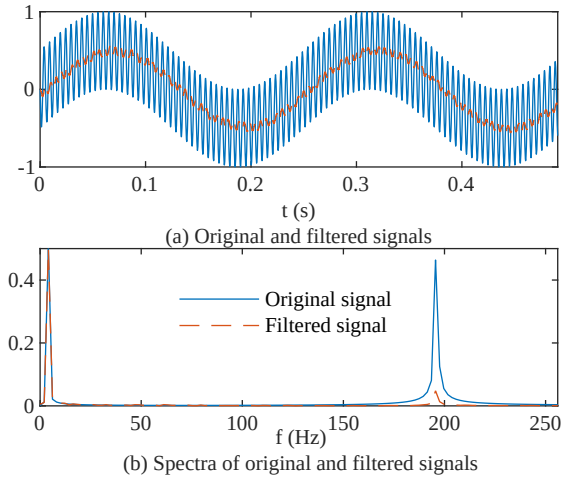


Fig. 17: The low-pass filtering results in (a) the time domain and (b) the frequency domain.

The stochastic integrator has been used in a stochastic low-density parity-check (LDPC) decoder to implement a tracking forecast memory in [49]. It solves the latch-up problem in the decoding process with a low hardware complexity.

For the FSM-based sequential circuits, however, the DSS cannot work well because the probability consistently changes and a steady hyper-state is hard to reach. The DSS could also violate the condition of the model being a Markov chain due to the autocorrelation of the encoded signal.

IV. Application of DSC

By using the stochastic integrators, DSC can implement a series of algorithms that involve iterative accumulations, such as an IIR filter, the Euler method and a GD algorithm.

A. IIR filtering

In [50], it is shown that an ADDIE with different parameters can be used as a low-pass IIR filter with different frequency responses, as shown in Fig. 16. Given the bit width of the stochastic integrator n, the transfer function of such a filter in the Z-domain is given by

$$H(Z) = \frac{1}{2^n Z + 1 - 2^n}, \tag{3}$$

for a stable first-order low-pass IIR filter [50]. An oversampled $\Delta - \Sigma$ modulated bit stream is then used as the input of the circuit. Since the generation of a $\Delta - \Sigma$ modulated bit stream

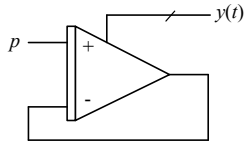


Fig. 16: ADDIE circuit produces an exponential function when the input is a sequence encoding a static number.

can also be considered as a Bernoulli process [51], the DSS can be used for IIR filtering as well.

Fig. 17 shows the filtering of mixed sinusoidal signals of 4 Hz and 196 Hz. The mixture is first sampled at a sampling rate of 65.6k Hz and the sampled digital signal is then converted to a DSS. The DSS is filtered by an 8-bit ADDIE. As can be seen, the high-frequency signals are almost filtered out while the low-frequency signals remain.

B. ODE solvers

The Euler method provides numerical solutions for ODEs by accumulating the derivative functions [52]. For an ODE $\mathrm{d}y/\mathrm{d}t = f(t, y(t))$, the estimated solution, $\hat{y}(t)$, is given by

$$\hat{y}(t_{i+1}) = y_{i+1} = y_n + hf(t_i, y_i) \tag{4}$$

with h as the step size and $t_{i+1} = t_i + h$. After k steps of accumulation,

$$y_k = y_0 + h\sum_{i=0}^{k-1} f(t_i, y_i). \tag{5}$$

To use a stochastic integrator, let the pair of input DSS's (e.g., A and B in (2)) encode the discrete derivative function, $\{f(t_i, y_i)\}$, such that $\mathbb{E}[A_i - B_i] = f(t_i, y_i)$. By doing so, the stochastic integrator implements (2) and approximately computes (5) with a step size $h = 1/2^n$. As a result, the bit width of the counter does not only decide the precision of the computation but also the time resolution. The integration in the Euler method is implemented by accumulating the stochastic bits in the DSS's instead of real values.

Using this method, an exponential function can be generated by a stochastic integrator. The ADDIE circuit, as shown in Fig. 16, solves

$$\frac{\mathbf{d}y(t)}{\mathbf{d}t} = p - y(t), \tag{6}$$

which is the differential equation for a leaky integrator model [53]. Thus, the ADDIE can be used to implement

neuronal dynamics in a leaky integrate-and-fire model. The analytical solution of (6) is given by

$$y(t) = p - (p - y_0)e^{-t}, \tag{7}$$

where y_0 is provided in the initial condition, i.e., $y(0) = y_0$. This model is commonly seen in natural processes such as neuronal dynamics [53]. Fig. 18 shows the results produced by the ADDIE when $y_0 = 1$ and $p = 0$.

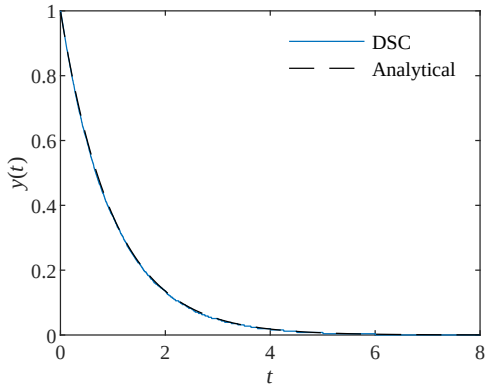


Fig. 18: ADDIE used as an ODE solver. The solution is an exponential function.

For a set of ODEs,

$$\begin{cases} \frac{\mathbf{d}y_1(t)}{\mathbf{d}t} = y_2(t) - 0.5, \\ \frac{\mathbf{d}y_2(t)}{\mathbf{d}t} = 0.5 - y_1(t), \end{cases} \tag{8}$$

with the initial values $y_1(0) = 0$ and $y_2(0) = 0.5$, a numerical solution can be found by the circuit in Fig. 19. The upper integrator computes the numerical estimate of $y_1(t)$ by taking the output DSS from the lower integrator and a conventional stochastic sequence encoding 0.5 as inputs. So the difference of the two input sequences approximately encodes $y_2(t) - 0.5$, the derivative of $y_1(t)$. Simultaneously, the output DSS of the upper stochastic integrator encodes the numerical solution of $y_1(t)$, which is used to solve the second differential equation in (8). Thus, the stochastic integrator can be viewed as an unbiased Euler solution estimator [54]. The results produced by the dynamic stochastic circuit is shown in Fig. 20. 8-bit up/down counters are used in both stochastic integrators and the RNG is implemented by a Sobol sequence generator [47]. Since the results are provided by the counter in the stochastic integrator, an explicit reconstruction unit, as shown in Fig. 12, is not required.

A partial differential equation, such as a Laplace's equation, can also be solved by an array of stochastic integrators with a finite-difference method.

C. Weight update for an adaptive filter

DSC can be applied to update the weights in an adaptive digital filter. An adaptive filter system consists of a linear filter and a weight update component, as shown in Fig. 21. The output of the linear filter at time i is given by

$$y_i = F(\mathbf{w}_i, \mathbf{x}_i) = \mathbf{w}_i \mathbf{x}_i = \sum_{j=0}^{M-1} w_i^j x_{i-M+j+1}, \tag{9}$$

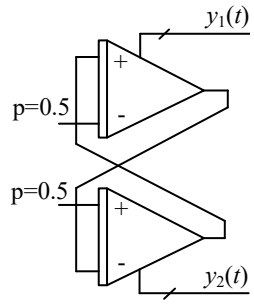


Fig. 19: A pair of stochastic integrators solving (8) [54].

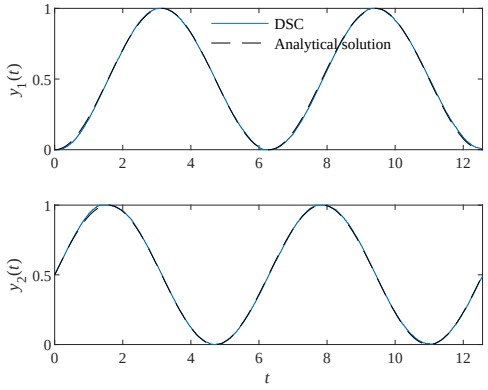


Fig. 20: Simulation results of the ODE solver in Fig. 19. They are compared with the analytical solutions.

where M is the length of the filter or the number of weights, $\mathbf{w}_i$ is a vector of M weights, $\mathbf{w}_i = [w_i^{(0)}, w_i^{(1)}, \ldots, w_i^{(M-1)}]$ and $\mathbf{x}_i$ is the input vector for the digital signal sampled at different time points, $\mathbf{x}_i = [x_{i-M+1}, x_{i-M+2}, \ldots, x_i]^{\mathbf{T}}$.

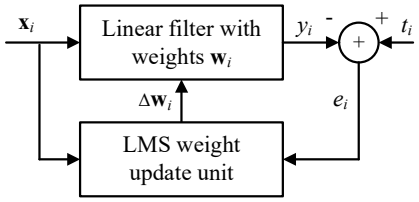


Fig. 21: An adaptive filter.

In the least-mean-square (LMS) algorithm, the cost function is the mean squared error between the filter output and the guidance signal, $\{t_i\}$. The weights are updated by

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \eta \mathbf{x}_i (t_i - y_i), \tag{10}$$

where η is the step size. The update is implemented by the circuit in Fig. 22. The two multipliers are used to compute $\mathbf{x}_i t_i$ and $\mathbf{x}_i y_i$, respectively. The integrator is then used to accumulate $\eta \mathbf{x}_i (t_i - y_i)$ over $i = 0, 1, \ldots$

The dynamic stochastic circuit is used to perform system identification for a high pass 103-tap FIR filter. 103 weights are trained by using randomly generated samples with the LMS algorithm. 14-bit stochastic integrators are used. After around 340k iterations of training, the misalignment of the

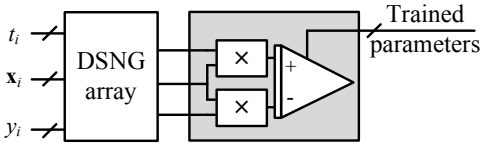


Fig. 22: A DSC circuit training a weight in an adaptive filter. The stochastic multipliers are denoted by rectangles with "×". Bipolar and sign-magnitude representations can be used for the stochastic multipliers and integrator.

weights in the adaptive filter converges to about -45 dB compared to the target system. The frequency responses of the target and the trained filter are shown in Fig. 23. As can be seen, the adaptive filter has a similar frequency response to the target system, especially in the passband, indicating an accurate identification result.

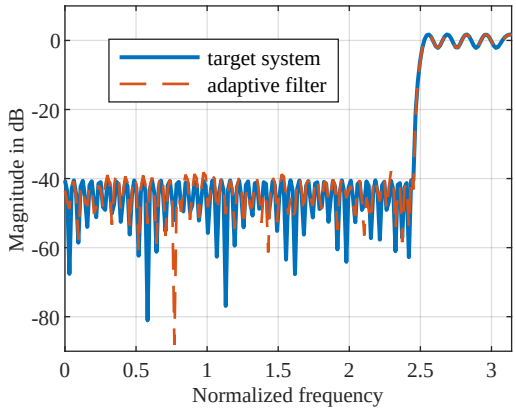


Fig. 23: Frequency responses of the target and trained filter using the sign-magnitude representation.

D. A gradient descent (GD) circuit

GD is a simple yet effective optimization algorithm. It finds the local minima by iteratively taking steps along the negatives of the gradients at current points. One-step GD is given by

$$w_{i+1} = w_i - \eta \nabla F(w_i), \tag{11}$$

where $F(w_i)$ is the function to be minimized at the ith step, $\nabla F(w_i)$ denotes the current gradient, and η is the step size. The step size is an indicator of how fast the model learns. A small step size leads to a slow convergence while a large one can lead to divergence.

Similar to the implementation of the Euler method, let a pair of differential DSS's encode the gradient $\{-\nabla F(w_i)\}$, the stochastic integrator can be used to compute (11) with a step size, $\eta = 1/2^n$, by using these DSS's as inputs. DSC can then be used to perform the optimization of the weights in a neural network (NN) by minimizing the cost function.

The training of an NN usually involves two phases. Take a fully connected NN as an example, shown in Fig. 24. In the forward propagation, the sum of product of the input vector $\mathbf{x}$ and the weight vector $\mathbf{w}$ is obtained as $v = \mathbf{wx}$. The output

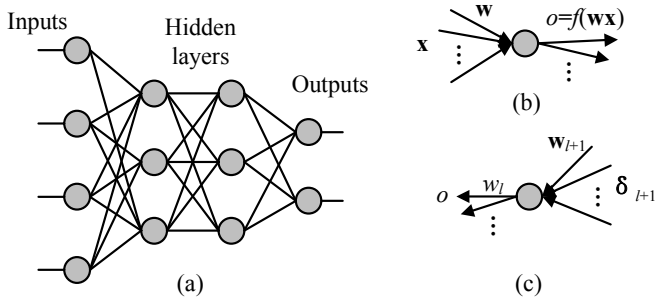


Fig. 24: (a) A fully connected NN, in which the neurons are organized layer by layer. (b) The computation during the forward propagation. (c) The local gradient for w_l is computed by using the backward propagation. The output o from previous neuron is required to compute the gradient.

of the neuron, o, is then computed by $o = f(v)$, where f is the activation function. During this phase, the weights do not change. In the backward propagation, the difference between the actual and the desired final outputs, i.e., the classification results, is calculated, and the local gradients are computed for each neuron based on the difference [55]. Take the neuron in Fig. 24(c) as an example, the local gradient is obtained by $\delta_l = f'(v)\mathbf{w}_{l+1}\boldsymbol{\delta}_{l+1}$, where $f'(\cdot)$ is the derivative of the activation function, $\mathbf{w}_{l+1}$ is the weight vector connecting layer l and $l+1$, and $\boldsymbol{\delta}_{l+1}$ is the local gradient vector of the neurons in layer $l+1$. Then, the gradient of a weight at layer l is computed by multiplying the local gradient with the recorded output of the designated neuron, i.e., $\nabla F(w_l) = -o\delta_l$.

In [56], it is found that the local gradient can be decomposed into two parts $\delta = \delta^+ - \delta^-$, where δ^+ is contributed by the desired output of the NN and δ^- by the actual output of the NN. So the gradient for weight w can be rewritten as $\nabla F(w) = -o(\delta^+ - \delta^-)$. Similar to the implementation of (10), the DSC circuit in Fig. 22 can be used to implement the GD algorithm to perform the online training (one sample at a time) of the fully connected NN with the input signals δ^+, o and δ^-. However, a backpropagation, i.e., $\delta = f'(v)\mathbf{w}_{l+1}\boldsymbol{\delta}_{l+1}$, is still computed by using a conventional method (e.g., a fixed-point arithmetic circuit) to obtain the local gradients. Otherwise, the accuracy loss would be too much for the algorithm to converge.

Fig. 25 shows the convergence of cross entropy (as a cost function) and testing accuracy for the training of a 784-128-128-10 NN with the MNIST handwritten digit dataset. As can be seen, the testing accuracy of the NN trained by the DSC circuit using the sign-magnitude representation is similar to the one produced by the fixed-point implementation (around 97%). However, the accuracy of the DSC implementation using the bipolar representation is relatively low.

The proposed circuitry can be useful for online learning where real-time interaction with the environment is required with an energy constraint [57]. It can also be used to train a machine learning model using private or security-critical data on mobile devices if data are sensitive and cannot be uploaded

to a cloud computer.

V. Error Reduction and Assessment

One major issue in SC is the loss of accuracy [58]. Independent RNGs have been used to avoid correlation and improve the accuracy of components such as stochastic multipliers. This method inevitably increases the hardware overhead. However, in a stochastic integrator, the sharing of RNGs for generating the two input sequences reduces the variance, thus reducing the error. This is shown as follows.

The error introduced by DSC at each step of computation is related to the variance of the difference of the two accumulated stochastic bits, $A_i - B_i$. When independent RNGs are used to generate A_i and B_i, the variance at a single step is given by

$$\begin{aligned}\text{Var}[A_i - B_i] &= \mathbb{E}[(A_i - B_i - \mathbb{E}(A_i - B_i))^2] \\ &= \mathbb{E}[(A_i - B_i)^2] - 2\mathbb{E}[A_i - B_i] \\ &\quad (P_{A,i} - P_{B,i}) + (P_{A,i} - P_{B,i})^2,\end{aligned} \tag{12}$$

where $P_{A,i}$ and $P_{B,i}$ are the probabilities of A_i and B_i being '1', respectively, i.e., $\mathbb{E}[A_i] = P_{A,i}$ and $\mathbb{E}[B_i] = P_{B,i}$. Since $(A_i - B_i)^2$ is 0 when A_i and B_i are equal or 1 when they are different, $\mathbb{E}[(A_i - B_i)^2]$ equals $P_{A,i}(1-P_{B,i}) + P_{B,i}(1-P_{A,i})$, which is the probability that A_i and B_i are different when A_i and B_i are independently generated[1]. (12) is then rewritten as

$$\begin{aligned}\text{Var}_{\text{ind}}[A_i - B_i] &= P_{A,i}(1 - P_{B,i}) + P_{B,i}(1 - P_{A,i}) \\ &\quad - (P_{A,i} - P_{B,i})^2 \\ &= P_{A,i}(1 - P_{A,i}) + P_{B,i}(1 - P_{B,i}).\end{aligned} \tag{13}$$

On the other hand, if the same RNG is used to generate A_i and B_i, the variance of $A_i - B_i$ can be derived from (12) as well. However, since only a shared random number is used to generate A_i and B_i, $\mathbb{E}[(A_i - B_i)^2]$ equals $|P_{A,i} - P_{B,i}|$, which gives the probability that A_i and B_i are different. Thus, (12) can be rewritten as

$$\begin{aligned}\text{Var}_{\text{share}}[A_i - B_i] &= |P_{A,i} - P_{B,i}| - (P_{A,i} - P_{B,i})^2 \\ &= |P_{A,i} - P_{B,i}|(1 - |P_{A,i} - P_{B,i}|).\end{aligned} \tag{14}$$

Since $\text{Var}_{\text{share}}[A_i - B_i] - \text{Var}_{\text{ind}}[A_i - B_i] = 2P_{B,i}P_{A,i} - 2\min(P_{A,i}, P_{B,i}) \leq 0$, we obtain $\text{Var}_{\text{share}}[A_i - B_i] \leq \text{Var}_{\text{ind}}[A_i - B_i]$ for any $i = 0, 1, 2, \ldots$ [54]. Therefore, sharing the use of RNGs to generate input stochastic bit streams improves the accuracy.

For the circuit in Fig. 22, though the inputs of the stochastic integrator are the outputs of the stochastic multipliers instead of being directly generated by a DSNG, the RNG sharing scheme still works in the generated DSS's encoding δ^+ and δ^-, i.e., it produces a smaller variance in the computed result [56]. However, an uncorrelated RNG must be used to generate the DSS encoding the recorded output of the neuron, o, because independency is still required in general for accurate stochastic multiplication.

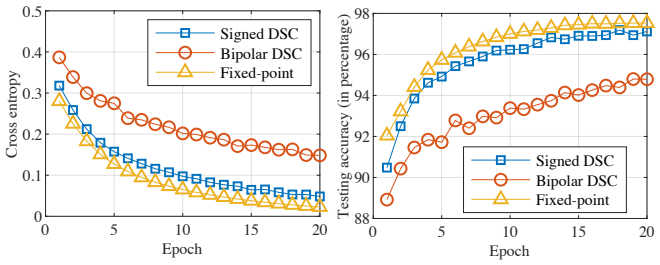
The accuracy of the accumulated result is also related to the bit-width of the counter in the stochastic integrator. The bit width does not only determine the resolution of the result but also the step size, especially in the ODE solver, adaptive filter and the gradient descent circuit. A larger bit width indicates a smaller resolution and a smaller step size for fine tuning at the cost of extra computation time. It is shown in [56] that the multi-step variance bound exponentially decreases with the bit width of the counter. However, for the IIR filter, the change of bit-width affects the characteristic of the filter.

For the ODE solver circuit, it is also found that the use of quasirandom numbers improves the accuracy compared to the use of pseudorandom numbers [54]. However, when the DSS is used to encode signals from the input or images such as in the adaptive filter and the gradient descent algorithm, quasirandom and pseudorandom numbers produce similar accuracy. The reason may lie in the fact that the values of the encoded signals in these two applications are independently selected from the training dataset. However, for the ODE solver, the signals encoded are samples from continuous functions, and adjacent samples have similar values. As a result, within a short segment of the DSS in this case, it approximately encodes the same value. It then works similarly as CSC, for which the Sobol sequence helps improving the accuracy. Fig. 26 compares the convergence of misalignment during the training of the adaptive filter using pseudorandom and quasirandom numbers. 16-bit stochastic integrators are used in this experiment. The misalignment is given by

$$\text{Misalignment} = \frac{\mathbb{E}[(w - \hat{w})^2]}{\mathbb{E}[w^2]}, \tag{15}$$

where w denotes the target value and $\hat{w}$ is the trained result. It can be seen that the convergence speed of using these two types of random numbers is similar and their misalignments both converge to around -52 dB.

[1] $\mathbb{E}[(A_i - B_i)^2] = \text{Prob}[(A_i - B_i)^2 = 1] \times 1 + \text{Prob}[(A_i - B_i)^2 = 0] \times 0$



(a) (b)

Fig. 25: (a) Convergence of cross entropy and (b) testing accuracy of MNIST hand-written digit recognition dataset. Cross entropy is the cost function to be minimized during the training of an NN. The weights of the NN are updated by the DSC circuits. The DSC circuits using the sign-magnitude and bipolar representations and the fixed-point implementation are considered for comparison.

TABLE II: Hardware evaluation of the gradient descent circuit array training a 784-128-128-10 neural network.

Metrics	DSC	Fixed-point	Ratio
Step size	2^{-10}	2^{-10}	-
Epochs	20	20	-
Min. time (ns)	1.6×10^6	4.7×10^6	1:3
EPO (fJ)	1.2×10^7	1.1×10^8	1:9.2
TPA (image/s/μm^2)	5.7×10^1	1.5	38:1
Aver. test Accu.	97.04%	97.49%	-

TABLE III: Hardware performance comparison of the ODE solvers for (8).

Metric	DSC	Fixed-point	Ratio
Step size	2^{-8}	2^{-8}	-
Iterations	3217	3217	-
Min. time (ns)	2573.59	8557.20	1:3.3
EPO (fJ)	201.21	466.00	1:2.3
TPA (word/μs/μm^2)	4.75	0.58	8.2:1
RMSE	4.7×10^{-3}	3.6×10^{-3}	-

VI. Hardware efficiency evaluation

Due to the extremely low sequence length used for encoding a value in DSC, the power consumption of DSC circuits is much lower than conventional arithmetic circuits. Moreover, since each bit in a DSS is used to encode a number, DSC is more energy-efficient than CSC. The gradient descent and ODE solver circuits are considered as examples to show the hardware efficiency. Implemented in VHDL, the circuits are synthesized by Synopsys Design Compiler with a 28-nm technology to obtain the power consumption, maximum frequency and hardware measurements. The minimum runtime, energy per operation (EPO), throughput per area (TPA) and accuracy are reported in Table II for the gradient descent circuits and in Table III for ODE solvers.

As can be seen, with a slightly degraded accuracy, the DSC circuits outperform conventional circuits using a fixed-point representation in speed, energy efficiency and hardware efficiency. For the gradient descent circuit, since the two RNGs can be shared among the 118,282 stochastic integrators (for 118,282 weights), the improvement in hardware efficiency is even more significant than the ODE solver, for which one RNG is shared between two stochastic integrators. However, since the sharing of RNGs does not significantly affect the critical path, the performance improvements are similar for these two circuits.

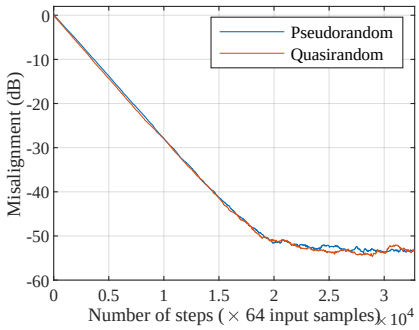


Fig. 26: Comparison of the convergence of misalignment using pseudorandom and quasirandom numbers.

VII. Conclusion

In this article, a new type of stochastic computing, dynamic stochastic computing (DSC), is introduced to encode a varying signal by using a dynamically variable binary bit stream, referred to as a dynamic stochastic sequence (DSS). A DSC circuit can efficiently implement an iterative accumulation-based algorithm. It is useful in implementing the Euler method, training a neural network and an adaptive filter, and IIR filtering. In those applications, integrations are implemented by stochastic integrators that accumulate the stochastic bits in a DSS. Since the computation is performed on single stochastic bits instead of conventional binary values or long stochastic sequences, a significant improvement in hardware efficiency is achieved with a high accuracy.

Acknowledgment

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Projects RES0025211 and RES0048688.

References

[1] B. R. Gaines, *Stochastic Computing Systems*. Boston, MA: Springer US, 1969, pp. 37–172.

[2] 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.

[3] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, 2011, pp. 154–161.

[4] S. C. Smithson, N. Onizawa, B. H. Meyer, W. J. Gross, and T. Hanyu, "Efficient CMOS invertible logic using stochastic computing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 6, pp. 2263–2274, June 2019.

[5] W. Maass and C. M. Bishop, *Pulsed neural networks*. MIT press, 2001.

[6] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A stochastic computational multi-layer perceptron with backward propagation," *IEEE Transactions on Computers*, vol. 67, no. 9, pp. 1273–1286, 2018.

[7] J. Zhao, J. Shawe-Taylor, and M. van Daalen, "Learning in stochastic bit stream neural networks," *Neural Networks*, vol. 9, no. 6, pp. 991–998, 1996.

[8] B. D. Brown and H. C. Card, "Stochastic neural computation. I. computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, 2001.

[9] D. Zhang and H. Li, "A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 2, pp. 551–561, 2008.

[10] Y. Ji, F. Ran, C. Ma, and D. J. Lilja, "A hardware implementation of a radial basis function neural network using stochastic logic," in *Proceedings of the Design, Automation & Test in Europe Conference (DATE)*, 2015, pp. 880–883.

[11] 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, 2016.

[12] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI implementation of deep neural network using integral stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2688–2699, 2017.

[13] 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," in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017, pp. 405–418.

[14] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze, "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 13–18.

[15] R. Hojabr, K. Givaki, S. Tayaranian, P. Esfahanian, A. Khonsari, D. Rahmati, and M. H. Najafi, "SkippyNN: An embedded stochastic-computing accelerator for convolutional neural networks," in *Proceedings of the 56th Annual Design Automation Conference (DAC)*, 2019, pp. 1–6.

[16] Y. Liu, L. Liu, F. Lombardi, and J. Han, "An energy-efficient and noise-tolerant recurrent neural network using stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 9, pp. 2213–2221, Sep. 2019.

[17] R. Cai, A. Ren, O. Chen, N. Liu, C. Ding, X. Qian, J. Han, W. Luo, N. Yoshikawa, and Y. Wang, "A stochastic-computing based deep learning framework using adiabatic quantum-flux-parametron superconducting technology," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: ACM, 2019, pp. 567–578.

[18] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 92:1–92:19, May 2013.

[19] 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 (DAC)*, 2018, pp. 1–6.

[20] P. K. Gupta and R. Kumaresan, "Binary multiplication with PN sequences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 4, pp. 603–606, 1988.

[21] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *Proceedings of the conference on Design, Automation & Test in Europe (DATE)*, 2014, pp. 1–4.

[22] S. Liu and J. Han, "Energy efficient stochastic computing with Sobol sequences," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 650–653.

[23] H. Niederreiter, *Random Number Generation and quasi-Monte Carlo Methods*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.

[24] P. Bratley and B. L. Fox, "Algorithm 659: Implementing Sobol's quasirandom sequence generator," *ACM Transactions on Mathematical Software (TOMS)*, vol. 14, no. 1, pp. 88–100, 1988.

[25] 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, July 2018.

[26] A. Alaghi and J. P. Hayes, "STRAUSS: Spectral transform use in stochastic circuit synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1770–1783, 2015.

[27] 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 *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 480–487.

[28] P. Mars and W. J. Poppelbaum, *Stochastic and deterministic averaging processors*. Peter Peregrinus Press, 1981, no. 1.

[29] 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 *IEEE 34th International Conference on Computer Design (ICCD)*, 2016, pp. 678–681.

[30] A. Ren, Z. Li, Y. Wang, Q. Qiu, and B. Yuan, "Designing reconfigurable large-scale deep learning systems using stochastic computing," in *IEEE International Conference on Rebooting Computing (ICRC)*, 2016, pp. 1–7.

[31] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Proceedings of the 53rd Annual Design Automation Conference (DAC)*, 2016, pp. 124:1–124:6.

[32] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017, pp. 1–6.

[33] 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, Sep. 2018.

[34] J. F. Keane and L. E. Atlas, "Impulses and stochastic arithmetic for signal processing," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2, 2001, pp. 1257–1260.

[35] R. Kuehnel, "Binomial logic: extending stochastic computing to high-bandwidth signals," in *Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers*, vol. 2, 2002, pp. 1089–1093.

[36] N. Yamamoto, H. Fujisaka, K. Haeiwa, and T. Kamio, "Nanoelectronic circuits for stochastic computing," in *6th IEEE Conference on Nanotechnology (IEEE-NANO)*, vol. 1, 2006, pp. 306–309.

[37] K. K. Parhi and Y. Liu, "Architectures for IIR digital filters using stochastic computing," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 373–376.

[38] B. Yuan, Y. Wang, and Z. Wang, "Area-efficient scaling-free DFT/FFT design using stochastic computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 12, pp. 1131–1135, 2016.

[39] R. Wang, J. Han, B. F. Cockburn, and D. G. Elliott, "Stochastic circuit design and performance evaluation of vector quantization for different error measures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, pp. 3169–3183, 2016.

[40] 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.

[41] T. Hammadou, M. Nilson, A. Bermak, and P. Ogunbona, "A 96/spl times/64 intelligent digital pixel array with extended binary stochastic arithmetic," in *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, 2003.

[42] 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 (DAC)*, 2013, pp. 136:1–136:6.

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

[44] M. H. Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani, "Time-encoded values for highly efficient stochastic circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1644–1657, 2017.

[45] S. R. Faraji, M. Hassan Najafi, B. Li, D. J. Lilja, and K. Bazargan, "Energy-efficient convolutional neural networks with deterministic bit-stream processing," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 1757–1762.

[46] S. Liu and J. Han, "Dynamic stochastic computing for digital signal processing applications," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[47] I. L. Dalal, D. Stefan, and J. Harwayne-Gidansky, "Low discrepancy sequences for Monte Carlo simulations on reconfigurable platforms," in *International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 2008, pp. 108–113.

[48] W. G. McCallum, D. Hughes-Hallett, A. M. Gleason, D. O. Lomen, D. Lovelock, J. Tecosky-Feldman, T. W. Tucker, D. Flath, T. Thrash, K. R. Rhea *et al.*, *Multivariable calculus*. Wiley, 1997, vol. 200, no. 1.

[49] S. S. Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross, "Majority-based tracking forecast memories for stochastic LDPC decoding," *IEEE Transactions on Signal Processing*, vol. 58, no. 9, pp. 4883–4896, 2010.

[50] N. Saraf, K. Bazargan, D. J. Lilja, and M. D. Riedel, "IIR filters using stochastic arithmetic," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014, pp. 1–6.

[51] F. Maloberti, "Non conventional signal processing by the use of sigma delta technique: a tutorial introduction," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 6, 1992, pp. 2645–2648.

[52] J. C. Butcher, *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*. USA: Wiley-Interscience, 1987.

[53] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[54] S. Liu and J. Han, "Hardware ODE solvers using stochastic circuits," in *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017, pp. 1–6.

[55] S. S. Haykin, *Neural networks and learning machines*. Pearson Upper Saddle River, NJ, USA:, 2009, vol. 3.

[56] 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, Nov 2018.

[57] R. Nishihara, P. Moritz, S. Wang, A. Tumanov, W. Paul, J. Schleier-Smith, R. Liaw, M. Niknami, M. I. Jordan, and I. Stoica, "Real-time machine learning: The missing pieces," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, ser. HotOS '17. ACM, 2017, pp. 106–110. [Online]. Available: http://doi.acm.org/10.1145/3102980.3102998

[58] J. P. Hayes, "Introduction to stochastic computing and its challenges," in *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, 2015, pp. 59:1–59:3.