IEEE Open Journal of Nanotechnology

Received 14 March 2024; revised 3 June 2024; accepted 4 June 2024. Date of publication 14 June 2024; date of current version 3 October 2024. The review of this article was arranged by Associate Editor G. Finocchio.

Digital Object Identifier 10.1109/OJNANO.2024.3414955

Pseudo-Random Number Generators for Stochastic Computing (SC): Design and Analysis

PILIN JUNSANGSRI [1] (Member, IEEE) AND FABRIZIO LOMBARDI [2] (Life Fellow, IEEE)

[1]School of Engineering, Wentworth Institute of Technology, Boston, MA 02115 USA
[2]Electrical and Computer Engineering Department, Northeastern University, Boston, MA 02115 USA

CORRESPONDING AUTHOR: PILIN JUNSANGSRI (email: junsangsrip@wit.edu).

ABSTRACT In most nanoscale stochastic computing designs, the Stochastic Number Generator (SNG) circuit is complex and occupies a significant area because each copy of a stochastic variable requires its own dedicated (and independent) stochastic number generator. This article introduces a novel approach for pseudo-random number generators (RNGs) to be used in SNGs. The proposed RNG design leverages the inherent randomness between each bit of data to generate larger sets of random numbers by concatenating the modules of the customized linear feedback shift registers. To efficiently generate random data, a plane of RNGs (comprising of multiple modules) is introduced. A sliding window approach is employed for reading data in both the horizontal and vertical directions; therefore, the sets of random numbers are generated by doubling the datasets and inverting the duplicated datasets. Flip-Flops are utilized to isolate the datasets and diminish correlation among them. This paper explores variations in parameters to evaluate their impact on the performance of the proposed design. A comparative analysis between the proposed design and existing SNG designs from technical literature is presented. The results show that the proposed nanoscale RNG design offers many advantages such as small area per RNG, low power operation, generated large datasets and higher accuracy.

INDEX TERMS Pseudo–random number generator (RNG), stochastic computing (SC), stochastic number generator (SNG).

I. INTRODUCTION

Stochastic computing (SC) is a computational technique that executes logic operations through the utilization of random bitstreams, making it particularly effective at nanoscales. In contrast to binary numbers, stochastic computing encodes data based on the probability of encountering 1's in bitstreams [1]. SC presents several advantages over traditional binary computation, including reduced hardware complexity, tolerance to errors, and low-power implementations of complex arithmetic functions, so very suitable for nanoscale environments. For instance, implementing a multiplier function in SC requires only a single AND gate, while a scaled adder function can be achieved using a multiplexer circuit. SC has been used in various applications such as image processing [2], digital filter [3], [4], and neural networks [5], [6], [7], [8].

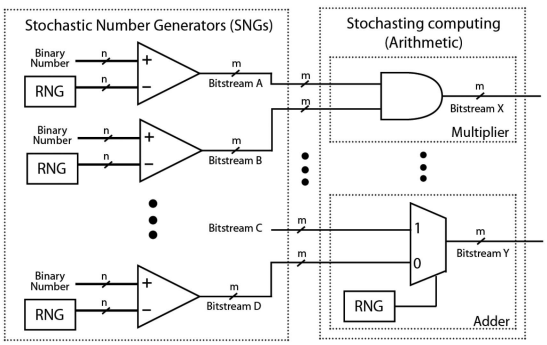


FIGURE 1. A SC multiplier circuit and a SC adder circuit.

Fig. 1 illustrates a schematic diagram of SC circuits, that can be categorized into two main components: the

stochastic number generators (SNGs) and the stochastic computing (arithmetic) section. The SNG generates a stochastic bitstream for each input/variable. During each clock cycle, a new n-bit random number (R) is compared to an n-bit binary number (B). If the random number (R) is less than or equal to the binary number, the output is set to 1; otherwise, it is 0. The sequence of output signal is referred to as a stochastic bitstream; its value can be determined by calculating the probability of encountering 1's in the bitstream. For instance, a number x represented as 0.375 or $3/8$ can be encoded by the sequence 0, 1, 0, 0, 1, 1, 0, 0, ..., where the frequency of 1's is equivalent to $3/8$. This encoded number is referred to as a stochastic number [9].

Once the stochastic bitstreams for all variables are generated, they are then fed to the stochastic arithmetic parts. In this example, SC circuits of a multiplier and an adder are considered. For the multiplier circuit, let the length of the stochastic stream be 8 bits. Bitstream A produces the sequence 1, 0, 0, 0, 0, 1, 0, 0, with a value of $1/4$ or 0.25. Bitstream B generates the sequence 0, 0, 1, 0, 1, 1, 0, 1, with a value of $1/2$ or 0.5. When these bitstreams are input into the AND gate, the resulting sequence is 0, 0, 0, 0, 0, 1, 0, 0, with a value of $1/8$ or 0.125. This value is equivalent to the product of bitstream A and bitstream B ($0.5 \times 0.25 = 0.125$).

The stochastic adder is implemented by using only a two-input multiplexer (MUX) in which the probability of the selecting signal is set at 0.5 [5], [10]. The output of a SC adder is determined by the following expression.

$$p_Y = \frac{p_C + p_D}{2} \tag{1}$$

where p_C and p_D are the stochastic values of bitstreams C and D respectively; p_Y is the stochastic value of the bitstream Y as output of the adder circuit (multiplexer); however, in (1), the value of the output of the adder circuit is reduced by half, so the output signal needs to be renormalized prior to the next step of the entire process [10].

As for the accuracy of a SC circuit, the individual bits in a bitstream sequence are random, therefore a long length is needed to increase the accuracy because the average of the sequence converges to the desired stochastic value [9].

In addition to the length of a stochastic bitstream, the correlation of stochastic bitstreams is another crucial factor influencing the accuracy of a stochastic circuit. Two correlation metrics are usually considered in an SC circuit: *autocorrelation* and *cross-correlation*. Autocorrelation assesses the independence of bits within a single stochastic number, indicating how effectively isolation can decorrelate the stochastic number from copies of itself. Cross-correlation gauges the level of independence between different stochastic numbers [11]. Correlation values fall within the range of -1 to 1, where 0 means no correlation or the maximum degree of independence [11], and ± 1 indicates maximum correlation. A correlation close to 1 suggests that an increase in a variable significantly influences an increase in another variable. Conversely, negative correlation signifies an inverse relationship between the two variables.

To mitigate the correlation between bitstreams and enhance the accuracy of an SC circuit [11], it is important to assign a dedicated and independent stochastic number generator (SNG) to each copy of a stochastic variable (or input). This approach ensures that the generation of stochastic numbers for different variables remains independent. However, hardware implementations for SNGs are typically complex; it has been reported that, in several stochastic designs, the SNGs occupy more than 80% of the total area of the stochastic circuit [12], [13].

This paper is an extension of a previous manuscript [14] proposing nanoscale pseudo-Random Number Generators (PRNGs) that can be used in Stochastic Number Generators (SNGs). Due to the challenge in implementing a true random number generator in which a physical noise source (such as voltage, and temperature) is then converted to a digital value, a deterministic "pseudo-random" circuit, that produces random-like number sequences, is often used as the random number source [11]. The proposed pseudo-RNGs, referred to as RNGs for simplicity, are constructed from multiple modules of a customized linear feedback shift register (LFSR); each module comprises a set of LFSRs in which data from each bit of the LFSR is transmitted to the other LFSRs. In the proposed design, a plane of RNGs array is accessed using a sliding window approach in both the horizontal and vertical directions. Due to low correlation observed when reading data in both the forward and reverse directions [1], the generated random data are effectively duplicated and read in the reverse. Next, an isolation method is employed to reduce correlation between datasets. For *isolation*, flip-flops (or isolators) are inserted in a circuit to reduce the correlation between each bit of data [15].

In this paper, the parameters of the proposed design are initially varied to study the impact on the generated random data; then, the proposed design is compared with other RNGs found in technical literature. The results of this paper show that the proposed design offers many advantages over other designs. When evaluating RNG planes of equal size, the proposed design stands out by generating datasets significantly larger than its counterparts. Each generated dataset has a very large size with low correlation, leading to accurate results. Furthermore, in terms of hardware, the circuit of the proposed design per dataset is small and consumes less power compared to other designs.

II. PROPOSED NANOSCALE DESIGN

This paper introduces a nanoscale design for pseudo-random number generator (RNGs) for SC. Rather than focusing on improving a single RNG, the proposed design exploits the advantage of randomness between each bit of data in the LFSRs to generate larger sets of random numbers. The architecture of the proposed RNG design comprises multiple modules of customized LFSRs. Each module consists of a group of

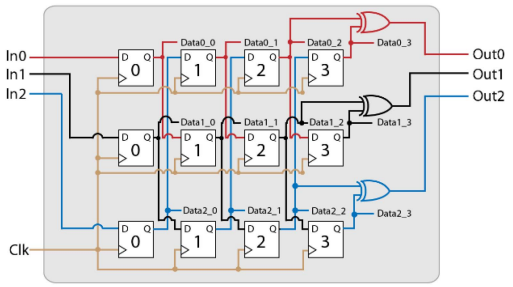


FIGURE 2. Circuit diagram for the proposed sample module; a module consists of 3 rows and each row has 4-bit data.

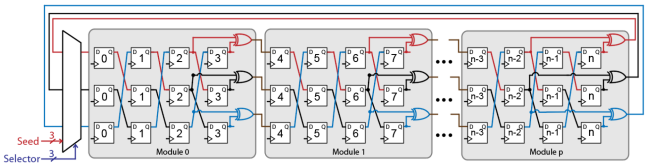


FIGURE 3. A row of proposed RNGs in which p modules are connected. Each module is made from three 4-bits customized LFSR.

LFSRs in which data is shifted between them. To illustrate the principles of the proposed design, an example is provided using a module consisting of three 4-bits customized LFSRs.

Fig. 2 depicts the circuit diagram of the proposed sample module, constructed with three 4-bit custom Linear Feedback Shift Register (LFSRs). The proposed module has three input signals (In0, In1, and In2), one input clock signal, three output signals (Out0, Out1, Out2), and a total of twelve output data bits. Each bit of data is stored and processed by twelve D-FF, with four D-FFs allocated per row and three rows in a module.

In each module, every row generates 4 bits of random data, and each bit is shifted to the next bit in the other rows. This deliberate shifting of each bit between the Random Number Generators (RNGs) effectively diminishes the autocorrelation among the generated random numbers. To further reduce correlation between data in the current and the next modules, three XOR gates are employed—one per row. The output signals of these XOR gates serve as input data for the subsequent module. For output, data in the proposed design is concurrently read, and these signals are processed to generate random numbers.

Fig. 3 depicts the connection of the proposed RNGs modules in a row of the RNG plane. The modules are connected in series. Output signals from each module serve as input data for the next module, except for the last one, where its output is looped back to the first module.

A multiplexer selects the input for the first column modules from either the last module's data or the seed input. Prior to utilizing the proposed Linear Feedback Shift Register (LFSR) plane, seed data is fed to the initial random numbers in which this data could be pre-selected or originating from a physical noise source (such as voltage, or temperature) as a true random number generator. Like the LFSR, the condition that must be avoided when selecting the seed data, is when every

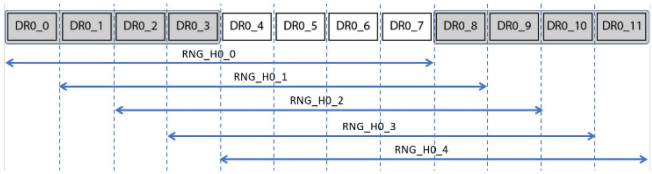


FIGURE 4. Sliding window to improve the utilization of the proposed RNG plane by considering only the horizontal direction.

bit of data in a row of the RNG plane is 0 (i.e., zero XOR zero is equal to zero); If every data in a row of the RNG is zero, the proposed RNG plane always generates a zero number. Once seeded, the modules' data is read in parallel, processed, and generates random numbers in normal operation.

N-bit random numbers are formed by concatenating adjacent n-bit data. At each clock cycle, the output data from the proposed modules is read. Due to low correlation between each bit, random numbers are generated using a sliding window approach, reading data from both horizontal and vertical directions. Fig. 4 illustrates the use of the sliding window to enhance the efficiency of the proposed RNG plane, focusing on the horizontal direction. The random numbers are obtained by concatenating data for a specific read bit, note that there are 8 bits in this case.

Next, permutation is used to increase the number of generated random variables from the existing random numbers. Leveraging the low cross-correlation observed when reading data (random numbers) in both forward and reverse directions, additional sets of random numbers are generated by reading the data in the inverse direction. For the proposed LFSR plane with n rows and m columns, the number of datasets to generate k-bit random numbers is given by.

$$\begin{aligned} N &= (horizontal + vertical) \times 2 \\ N &= [n \times (m - k + 1) + m \times (n - k + 1)] \times 2 \\ N &= 4mn + 2\,(m + n)\,(1 - k) \end{aligned} \tag{2}$$

where N represents the number of generated k-bit random number datasets using the proposed design, the term "horizontal" denotes the number of datasets that a RNG plane could generate from n rows and m columns when considering only the horizontal direction. The term "vertical" represents the number of datasets that an RNG plane size n × m (n rows and m columns) could generate when considering only the vertical direction.

The sliding window method increases the number of generated random number datasets in an RNG plane; however, a drawback of this method is the potential for high cross-correlation between specific pairs of datasets, especially when two datasets share the most significant bit (MSB). To mitigate this, an *isolation* method is employed. In this approach, flip-flops (FFs) are inserted to delay the data in each dataset. When a pair of datasets with high cross-correlation is delayed at different times, the cross-correlation between these datasets is reduced.

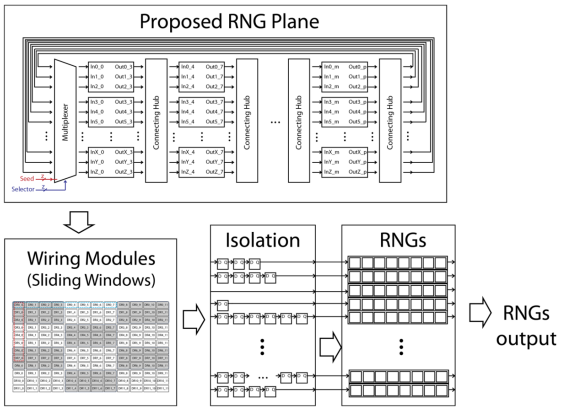


FIGURE 5. Organization of the proposed RNG plane.

In the proposed design, the number of flip flops for isolating data in each dataset is pre-determined and fixed. To determine the number of flip-flops needed to introduce the delay to each dataset, a process involves adding a flip-flop to the datasets with the highest cross-correlation until the cross-correlation between all datasets fall below a specified threshold. Once the datasets are delayed by this specific duration using flip-flops, the random numbers are derived by reading the data in each dataset.

Fig. 5 illustrates the organization of the proposed RNG plane. After the data in the RNG plane is read by using a sliding window, the datasets undergo the isolation method to introduce delays to each dataset at different times. The data from each dataset after the isolation method serves as the random number dataset.

III. SIMULATION AND DISCUSSION

In this section, the simulation results of the proposed scheme are discussed. The default parameters of the proposed module are configured as follows: each module comprises 3 rows of customized LFSRs, with each row generating 4 bits of random data. The XOR inputs for the last module in each row are positioned at bits 1 and 3 of its module, whereas the XOR inputs of other modules are selected from bits 2 and 3 of their modules.

In this simulation, an RNG plane is constructed with 16 proposed modules, there are 4 modules per row and 4 rows in total. The total bit of this RNG plane is 192 bits of data, with 16 bits per row and 12 rows in the plane, as illustrated in Fig. 6. The simulation focuses on 8-bit random numbers, generated by concatenating data from multiple modules aligned in the same direction. The simulation is implemented using Python, and the Genus Synthesis Solution tool at 32 nm CMOS technology to evaluate the delay, power dissipation, and area of the RNG circuits in the nanoscales.

By using the proposed method, 376 datasets are generated from 192 bits of data, with each dataset having a length exceeding 120 million random and with no repetition. The average autocorrelation and cross correlation of every dataset

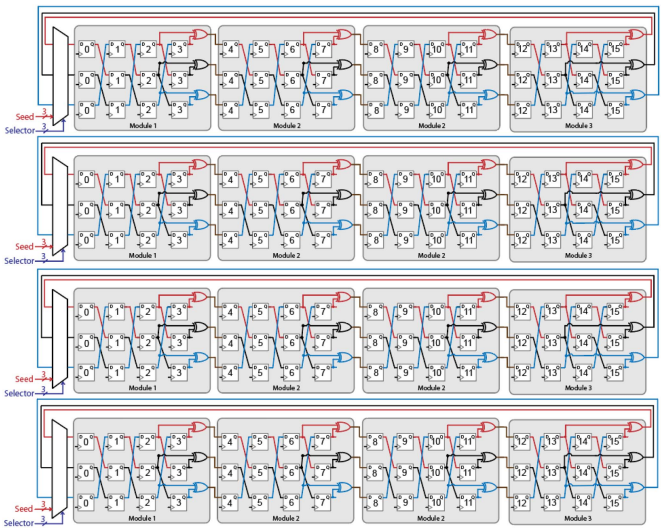


FIGURE 6. The simulated proposed RNG plane.

TABLE 1. Delay, Power Dissipation, and Area of the Plane of the Proposed RNG Array With 16 Proposed Modules in Fig. 6

Module	Delay (ps)	Power Dissipation (μW)	Area (μm^2)
Single Module	109	122.895	105.216
RNG plane	207	1,988.82	1 711.66
Isolation	109	23,907.09	19,876.11
Total	207	25,895.91	21,587.77

are 2.766×10^{-6} and 4.67×10^{-4} respectively – indicating very small values.

When assessing delay, power dissipation, and area, the following configuration is considered:

- A single module composed of three 4-bit RNGs (Fig. 2).
- An RNG plane comprised of 16 proposed modules: with 4 modules per row and 4 rows in an RNG plane (Fig. 6).
- For the isolation module, few FFs are added to each dataset until the cross correlation between datasets falls below 0.1. To achieve this goal, a total of 376 FFs are needed and the highest number of FFs to delay a dataset is 35, i.e., after the first 35 clock cycles, the proposed design operates normally.

Table 1 presents a summary of the delay, power dissipation, and area of the plane of the proposed RNG array.

A. PROPOSED CUSTOMIZED LFSR MODULE

This section investigates the impact of parameters on the proposed LFSR module, with respect to the size of the generated random numbers. To explicitly show the effect of each parameter, this section considers only one row of the RNG plane; therefore, each row of the RNG plane consists of two interconnect proposed modules where the outputs of the second module are fed back to the input of the first module. Each module consists of 3 rows of LFSRs, with each row generating 4 bits of random data. For the last module in each row, XOR inputs are positioned at bits 1 and 3, while for other modules, the XOR inputs are selected at bits 2 and 3. In this section,

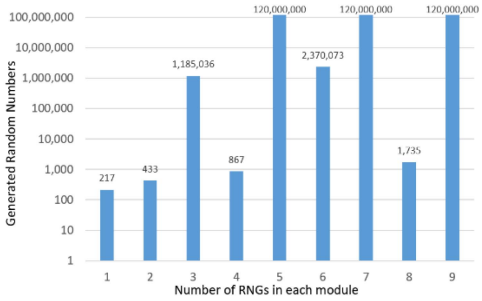


FIGURE 7. Size of generated 8-bit random numbers in each dataset. Each module is simulated 120,000,000 times.

TABLE 2. Length of Generated 8-bit Random Numbers From Each RNG; Each RNG is Made From Two Consecutive Proposed Modules

Number of bits in each row in the proposed module	Size of dataset of generated 8-bit random numbers
4	1,185,036
5	44,434,004
6	More than 120×10^6

the output generates a set of 8-bit random numbers, with data being read only from the first row (row 0) of the RNG plane.

1) NUMBER OF ROWS (RNGS) IN EACH MODULE

When varying the number of rows in each LFSR module, the size of each dataset is changed. Fig. 7 shows the size of the generated random numbers in each dataset when varying the number of rows in each module of the proposed design. Like a linear feedback shift register, the increase in the number of rows in a module does not always generate a large set of random numbers. The results in Fig. 7 show that the size of the generated random numbers in a dataset by using the proposed design is low when the number of rows (RNGs) in each module is a power of 2 or an even number. For example, when there are 32 rows of RNGs per module, the size of the generated 8-bit random numbers in a dataset is only 6943. However, when the number of rows in each module is odd or a prime number (for example such as 5, 7, 9), the number of generated 8-bit random numbers per dataset is increased to more than 120,000,000. For ease of presentation and as an example, in this paper, each module consists of only 3 RNGs (rows).

2) NUMBER OF BITS IN EACH ROW OF THE PROPOSED MODULE

The number of bits in each row of the proposed module also impacts the size of the generated random numbers in each dataset. As shown in Table 2, by increasing the number of bits in each module, the size of the generated random numbers in a dataset increases too. For example, when each row in a module consists of 6 bits of data and only 2 modules are connected in series, the proposed RNG design generates more than 120 million random numbers in a dataset (by reading data only from bits 0 to 7 in the first row).

TABLE 3. Length of Generated 8-bit Random Numbers Per Dataset When Varing the Number of Modules in Each Row

Number of modules in each row in the proposed module	Size of dataset of generated 8-bit random numbers
2	1,185,036
3	83
4	More than 120 million
5	59,593,841
6	1,048,574
7	664,019
8	More than 120 million
9	196,601

TABLE 4. Number of Generated 8-bit Random Number in a Datasets When Reading Data in Row 1 From Bit 0 to Bit 7; Each Row Consists of Four Modules in Series

		Positions of XOR inputs of first module					
		01	**02**	**03**	**12**	**13**	**23**
Positions of XOR inputs of other modules	01	$\mathbf{1\times10^7}$	$\mathbf{1\times10^7}$	35,804	$\mathbf{1\times10^7}$	$\mathbf{1\times10^7}$	15,809
	02	42,704	419	$\mathbf{1\times10^7}$	$\mathbf{1\times10^7}$	$\mathbf{1\times10^7}$	4,784,054
	03	1,392,554	$\mathbf{1\times10^7}$	60	$\mathbf{1\times10^7}$	$\mathbf{1\times10^7}$	$\mathbf{1\times10^7}$
	12	298,934	$\mathbf{1\times10^7}$	$\mathbf{1\times10^7}$	419	$\mathbf{1\times10^7}$	7,340,024
	13	4,094	$\mathbf{1\times10^7}$	$\mathbf{1\times10^7}$	$\mathbf{1\times10^7}$	120	$\mathbf{1\times10^7}$
	23	332,009	69,614	$\mathbf{1\times10^7}$	$\mathbf{1\times10^7}$	$\mathbf{1\times10^7}$	1,259

Each case is simulated 10 000 000 times.

3) NUMBER OF MODULES IN EACH ROW

Next, the number of modules connected in series in each row of the RNG plane is considered. In this simulation, a default module (4 bits per row and 3 rows of RNG as shown in Fig. 2) is considered. By varying the number of modules in each row, the size of the first 8-bit random number dataset is found by reading data horizontally (so only from bits 0 to 7 of a RNG plane).

In Table 3, an increase of the number of proposed modules in each row does not guarantee an increase in the size of the generated random numbers in a dataset; however, the size of the generated random number dataset is very large when the total bit in each row is a power of 2; so since each module has 4 bits of data per row, when 2, 4, and 8 modules are connected, the total numbers of bits in each row of the RNG plane are 8, 16, and 32 respectively, i.e., the number of generated 8-bit random numbers in a dataset is therefore large.

4) POSITION OF INPUTS FOR THE XOR GATE

Next, the impact of the position of the input for the XOR gates on the size of the generated random number dataset is considered. The positions of the XOR inputs are given as an index of each module (where the first index starts at zero). Each simulation is performed 10,000,000 times; in this simulation, each row of the RNG plane consists of 4 modules in series.

The results in Table 4 show that there are several cases in which the proposed RNG plane generates more than 10 million 8-bit random numbers per dataset; hereafter in this paper, the positions of the XOR inputs for the first module

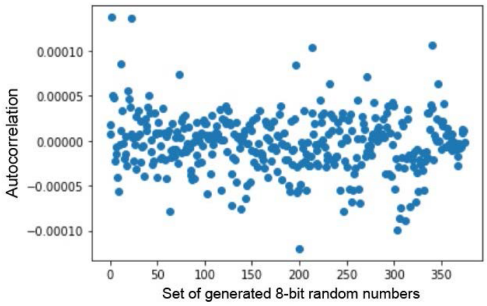


FIGURE 8. Scatter plot of autocorrelation of each 8-bits random number dataset by using the proposed method when reading data in both directions; horizontal and vertical, using sliding windows and permuting its output.

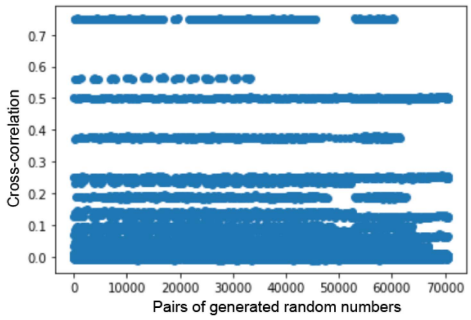


FIGURE 9. Scatter plot of cross-correlation of pairs of 8-bits random number datasets by using the proposed method when reading data in both directions; horizontal and vertical, using sliding windows and permuting its output.

are selected at bits 1 and 3 from the last module, while the XOR inputs of the other modules are selected from bits 2 and 3 of its previous module.

B. PLANE OF THE PROPOSED RNGS

Previously, each random number is generated by reading only the first 8 bits in each row of the RNG plane; a method is proposed next to improve the utilization of the proposed RNG plane by using a so-called sliding window. As presented in Figs. 4 and 5, data in an RNG plane can be reused to generate more datasets of random numbers by reading in both directions (horizontal and vertical) using the sliding window and permuting the data by reading it in the reverse direction.

For simulation, the RNG plane of Fig. 6 consists of 16 proposed modules; each row consists of 4 proposed modules connected horizontally. An RNG plane with 4 rows is used as an example. This RNG plane has 192 bits of data in total; so, 16 bits per row and there are 12 rows in an RNG plane. By reading data in both directions with the sliding window and permuting by reversing it, 376 sets of 8-bit random numbers are generated from the proposed RNG plane. The following features are attained.

- For the horizontal direction, 9 sets of random numbers are generated in each row. So, 108 sets of random numbers are generated horizontally.
- For the vertical direction, 5 sets of random numbers are generated per column. So, 80 sets of random numbers are generated vertically.

The total number of generated sets is 188; moreover, when using permutation, the number of sets of generated 8-bit random numbers is doubled, i.e., 376 sets; this number can also be found by using (2).

The proposed RNG plane is simulated 100,000 times to generate 376 datasets of 8-bit random numbers (each set has 100,000 random numbers). The size of the generated (8-bit) random numbers for each dataset and the correlation (between a dataset and other datasets in the same RNG plane) are assessed next.

For the size of the dataset, every set of generated random numbers (376 sets) generates more than 100,000 random numbers at a very low autocorrelation; the average autocorrelation of every generated set is only -2.766×10^{-6}. For cross-correlation, there are 376 sets of generated 8-bit random numbers, therefore 70,500 pairs of cross-correlation must be considered.

TABLE 5. Cross-Correlation Between Sets of Generated 8-bit Random Numbers

Level of cross-correlation	Range of cross-correlation [low, high]	Average cross-correlation in this level	Number of pair	Percentage (%)
Very high	(0.7, 1]	0.7501	240	0.34
High	(0.55, 0.7]	0.5627	48	0.07
Medium high	(0.45, 0.55]	0.5004	320	0.45
Medium	(0.3, 0.45]	0.3751	436	0.62
Medium low	(0.2, 0.3]	0.2457	368	0.52
Low	(0.1, 0.2]	0.1651	892	1.27
Very low	(0.05, 0.1]	0.0819	1208	1.71
Extremely low	***[0, 0.05]***	***0.0017***	***66988***	***95.02***

Figs. 8 and 9 show the scatter plot of the autocorrelation of the datasets and the cross-correlation between pairs of datasets generated by using the proposed methods. As shown in Figs. 8 and 9, the autocorrelations of the datasets are close; the cross-correlation between the pair of generated datasets can be categorized into multiple sub-categories. In this paper, cross-correlations between each pair are separated as shown in Table 5.

In Table 5, 95.02% of the cross-correlations are extremely low (less than 0.05); the average cross correlation is only 0.0017, so well suitable for stochastic computing.

Next, the relationship between each dataset is considered to find pairs of datasets that have high cross-correlation. A very high cross-correlation between datasets hereafter (i.e., a cross-correlation greater than 0.7) occurs when both datasets share their most significant bit (MSB) or the second most significant bit. Fig. 10 shows a sample of datasets that have high cross-correlation: three datasets (dataset 0, dataset1, and dataset 2) share the MSB at row 0 column 7. Note that dataset 0 is read from row 0 column 7 to the left, dataset 1 is read row 0 column 7 to the right, and dataset 2 is read from row 0 column 7 to the bottom. Cross-correlation between these datasets is very high, i.e., approximately 0.75.

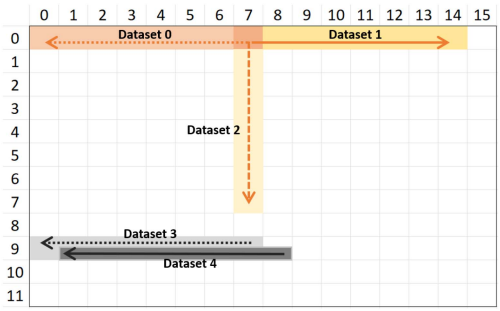


FIGURE 10. Sample datasets in the proposed RNG plane that have very high cross-correlation.

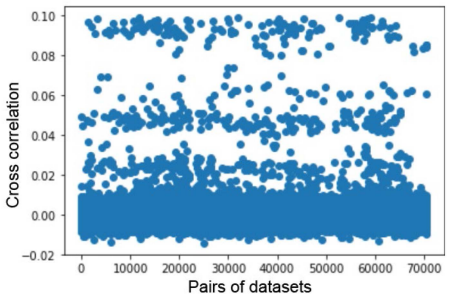


FIGURE 11. Scatter plot of cross-correlation of every pair of datasets of the proposed design with isolation; FFs are inserted before datasets.

A high cross-correlation is approximately 0.7, it occurs between datasets that are located next to each other and read in the same direction; for example, in Fig. 10, cross-correlation between dataset 3 (in which its MSB is at row 9 column 7), and dataset 4 (in which its MSB is at row 9 column 8) and both are read to the left, then the cross-correlation between these two datasets is approximately 0.5, because the MSB of dataset 3 is the second most significant bit of dataset 4, so the value in dataset 3 impacts the value in dataset 4.

The cross-correlation between datasets is reduced when shared bits between each set has less significance e.g., cross-correlation between a dataset read from row 0 column 7 to the left and a dataset read from row 0 column 4 to the right direction, is only 0.37.

1) ISOLATION METHOD

Due to high cross-correlation between some pairs of generated datasets, an isolation method is used to shift data in a dataset for a clock cycle. This is accomplished by adding a flip-flop after reading the data. This method can significantly reduce the cross-correlation; for example, cross-correlation between a pair of datasets with a high cross-correlation (0.75) is significantly reduced to 0.004 when shifting data in one of the datasets by a clock cycle.

Next, the number of FFs to isolate datasets in the proposed RNG plane is established; in the simulation, a FF is added to a dataset that has the highest total cross-correlation. This simulation is run till the cross-correlation of every dataset is less than the threshold value (set to 0.1 in this case) and the number of FFs for each dataset are found.

Fig. 11 shows the scatter plot of the cross-correlation between datasets after using the isolation method in which their cross-correlation threshold is set to 0.1. After using the isolation method, the average cross-correlation between these pairs of datasets is only 0.000467 and a total of 376 FFs are needed. The largest number of FFs to be inserted in a dataset is 35 (corresponding to the delay in clock cycles) i.e., after the first 35 clock cycles, the proposed design operates in a normal behavior. This occurs at dataset 65 in which data from row 7 column 9 is read in the left direction. 33 datasets do not need an FF to delay a value, i.e., data can directly be read from such dataset. The number of the inserted FFs tend to be high for a dataset whose MSB is located at the middle of the RNG plane.

TABLE 6. Mean Square Error in SC Multiplier When Stochastic Bitstreams are Generated Using the Proposed Design

Stream size	Number of bits in generated RNGs			
	8 bits	16 bits	32 bits	64 bits
8	1.81×10^{-2}	1.82×10^{-2}	1.82×10^{-2}	1.81×10^{-2}
16	9.22×10^{-3}	9.20×10^{-3}	9.19×10^{-3}	9.12×10^{-3}
32	4.58×10^{-3}	4.73×10^{-3}	4.58×10^{-3}	4.64×10^{-3}
64	2.35×10^{-3}	2.37×10^{-3}	2.29×10^{-3}	2.33×10^{-3}
128	1.18×10^{-3}	1.15×10^{-3}	1.17×10^{-3}	1.15×10^{-3}
256	5.74×10^{-4}	5.85×10^{-4}	5.71×10^{-4}	5.82×10^{-4}
512	3.17×10^{-4}	2.93×10^{-4}	2.85×10^{-4}	3.07×10^{-4}
1024	1.74×10^{-4}	1.48×10^{-4}	1.21×10^{-4}	1.53×10^{-4}

TABLE 7. Mean Square Error in SC Adder When Stochastic Bitstreams are Generated Using the Proposed Design

Stream size	Number of bits in generated RNGs			
	8 bits	16 bits	32 bits	64 bits
8	2.66×10^{-2}	2.67×10^{-2}	2.65×10^{-2}	2.66×10^{-2}
16	1.33×10^{-2}	1.33×10^{-2}	1.34×10^{-2}	1.33×10^{-2}
32	6.77×10^{-3}	6.68×10^{-3}	6.67×10^{-3}	6.66×10^{-3}
64	3.32×10^{-3}	3.408×10^{-3}	3.41×10^{-3}	3.36×10^{-3}
128	1.69×10^{-3}	1.68×10^{-3}	1.68×10^{-3}	1.64×10^{-3}
256	8.60×10^{-4}	8.85×10^{-4}	8.40×10^{-4}	8.59×10^{-4}
512	4.32×10^{-4}	4.03×10^{-4}	4.05×10^{-4}	4.09×10^{-4}
1024	2.19×10^{-4}	2.19×10^{-4}	1.96×10^{-4}	2.18×10^{-4}

C. ACCURACY

Next, the generated random numbers from the proposed design are used to generate stochastic bitstreams for each input. These bitstreams are fed into two stochastic circuits: a multiplier and adder. The value of the result is compared with the expected value. The Mean Square Error (MSE) is used as a metric to assess the difference between the expected and computed results using SC. The values of each input are selected randomly and sets of 1,000,000 random numbers are generated using the proposed design.

Tables 6 and 7 present the Mean Square Errors (MSEs) of the SC multiplier and adder, utilizing the proposed design for random number generation. The error in the SC circuit diminishes when increasing the bitstream length. However,

TABLE 8. Comparison Between the Proposed RNG Design and Other Designs When it is Simulated for 120×10^6 Times

Metric	Array of Random number generator (RNGs)			
	Proposed design	*LFSR*	*SBoNG [11]*	*Permuted LFSR [1]*
Number of datasets	376	24	192	192
Size of dataset	120×10^6	65,534	65,534	65,534
Average Auto correlation	2.766×10^{-6}	1.532×10^{-4}	1.561×10^{-5}	1.555×10^{-5}
Average Cross correlation	4.67×10^{-4}	5.685×10^{-5}	9.326×10^{-2}	1.317×10^{-1}
Delay/RNG (ps)	207	159	159	159
Power/RNG (μW)	68.872	130.02	96.77	71.887
Area/RNG (μm^2)	57.41	111.95	86.68	60.25
Mean Square Error of SC Multiplier ($\times10^{-4}$)	23.457	39.532	110.67	56.443
Mean Square Error of SC Adder ($\times10^{-4}$)	33.204	45.288	93.791	61.879

the number of bits in the generated random numbers has low impact to the accuracy of these stochastic circuits.

IV. COMPARISON

This section presents a comparison between the proposed and other random number generators found in technical literature e.g., the Linear Feedback Shift Register (LFSR), SBoNG [11] that generates low-correlated SNGs from a LFSR by using the S-box circuit, and the permuted LFSR [1] that generates more random numbers by permuted data from a single LFSR with no additional circuit. A 12×16 array of RNGs that consists of 192 bits of RNGs arranged with 16 bits per row and 12 rows in an RNG plane is considered. For the proposed design, a RNG is made of 16 default proposed modules as shown in Fig. 6. Various aspects are evaluated including the number of generated random numbers, delay, power dissipation, area, correlations, and accuracy of the stochastic circuit.

A. NUMBER OF GENERATED RANDOM NUMBER PER DATASETS

As shown in Table 8, the proposed design generates a very large number of RNGs per dataset, more than 120×10^6 data, while other designs generate only 65,534 random data. The SBoNG design [11] exhibits variability in the number of random data per dataset. Specific conditions must be met for SBoNG to generate large amounts of random data; however, these conditions cannot always be met. For the permuted LFSR [1], the size of generated random values in each dataset is the same as for the LFSR; in this case, a 16-bit LFSR is used in each row, and each 8-bit dataset generates a total of 65,634 data.

B. NUMBER OF DATASETS

For the RNG planes with a size of 12×16, the proposed design leverages the sliding window in both the vertical and horizontal directions, along with data reversal, to generate 376 sets of RNGs. The other designs generate a smaller number of RNGs than the proposed design. For the permuted LFSR [1], the number of generated datasets is limited to the number of bits per row. This limitation occurs from the increase in cross-correlation between datasets e.g., when datasets share their most significant bit (MSB), its cross-correlation is increased by 0.5. Hence, the permuted LFSR method [1] generates up to 16 datasets per row (with 16 bits in each row). The total generated dataset of [1] is limited to 192. For the SBoNG [11], each row also generates 16 datasets by rotating data to the right to increase the number of datasets. The total number of generated datasets for SBoNG [11] is limited to 192 datasets for 16 bits per row of data. As for the LFSR, it generates 2 datasets of 8-bit random numbers per row, and with 12 rows in this array, the total number of datasets that the LFSR generates is 24.

C. CORRELATION

In the simulation results presented in Table 8, the size of the RNG dataset is constrained by either the maximum number of generated random numbers in a dataset or 100,000, whichever is lower. As shown in Table 8, the proposed design generates datasets with very low autocorrelation. The cross-correlation of the proposed design is higher than for the LFSR, due to sharing bits when reading data through the sliding windows. However, after using the isolation method, the cross-correlation of the proposed design is significantly reduced, and it is better than the cross-correlation of the SBoNG method [11] and the permuted LFSR [1]. The cross-correlations of the permuted LFSR [1] are high because n datasets are generated from a LFSR with n bits in each row. Sharing with the second MSB increases the cross-correlation between datasets.

D. DELAY, AREA, AND POWER

In Table 8, the delay, power dissipation, and area are normalized per the number of generated datasets. The results show that both power dissipation and area per RNG of the proposed design are the lowest, followed by the permuted LFSR [1], SBoNG [11], and LFSR respectively. The main advantage of the proposed design is its ability to reuse data in both directions of an RNG plane to generate a larger number of datasets. Both power dissipation and area per RNG of the proposed design are very low.

For the LFSR, the values of the power dissipation and area per RNG are the largest (worst) because the number of generated datasets is small. SBoNG [11] generates larger datasets compared to LFSR however, it needs additional combinational circuits and the LFSR. The power dissipation and area of SBoNG [11] is high. The permuted LFSR [1] reuses data in the LFSR; nevertheless, the size of the permuted LFSR is limited by the number of bits in each row. Area and power dissipation per RNG of the permuted LFSR [1] are lower than SBoNG [11]; however, their values are still larger than the proposed design.

The only disadvantage of the proposed design is the delay. In addition to the initial setup of the proposed RNG plane, which operates at the first 35 clock cycles, the delay of the proposed design is high, because the complexity of the proposed

TABLE 9. Delay and Power Dissipation, of Stochastic Multipler and Stocastic Adder Circuits

Metric	Component	Array of Random number generator (RNGs)			
		Proposed design	*LFSR*	*SBoNG [11]*	*Permuted LFSR [1]*
Delay (ps)	RNG	207	159	159	159
	SC Multiplier	379			
	SC adder	434			
Power (μW)	RNG	68.872	130.02	96.77	71.887
	SC Multiplier	144.944	267.24	200.74	150.974
	SC adder	147.7	269.996	203.50	153.73
Power Delay Product (PDP) (fs×W)	RNG	14.26	20.67	15.39	11.43
	SC Multiplier	54.93	101.28	76.08	57.22
	SC adder	64.10	117.18	88.32	66.72

TABLE 10. Power Dissipation, and Area Per RNG

Design	Delay	Power	Area	Product Correlation	DPACP
Proposed	1	**0.53**	**0.51**	$\mathbf{6.29\times10^{-4}}$	$\mathbf{1.71\times10^{-4}}$
LFSR	**0.77**	1	1	4.23×10^{-3}	3.26×10^{-3}
SBoNG [11]	**0.77**	0.74	0.77	0.71	0.315
Permuted LFSR[1]	**0.77**	0.55	0.54	1	0.229

RNG plane is higher than the other designs with a complexity like LFSR.

Next, delay and power dissipation of stochastic applications (e.g., the SC multiplier and the SC adder) are evaluated when using various RNGs. Due to the simplicity of stochastic arithmetic, e.g., a multiplier circuit can be implemented by using an AND gate, a scaled adder circuit can be implemented by using a multiplexer circuit, therefore power dissipation of these circuits is lower than for stochastic number generators (SNGs); the power dissipation of a SC multiplier circuit is only 7.2 μW, while the power dissipation of a SC adder circuit is 9.956 μW. Hence, the power dissipations of the SC arithmetic hardware have a smaller impact to the SC system, because the power dissipation of the SNGs dominates.

For the delay of SC applications when using various RNG designs, the circuits of Fig. 1 are used; in these circuits, 2 comparator circuits are needed to generate the stochastic bitstreams; then these values are processed by to stochastic arithmetic hardware.

Table 9 presents the simulation results; in this simulation, the Genus synthesis tools at the 32 nm technology node are used at the nominal supply voltage of 1.05 V. The simulation results show that for both applications and the RNG only, the proposed scheme has a larger delay compared to other designs. However, the delay for the SC multiplier is 379 ps while the delay of the SC adder circuit is 434 ps; hence, the total delay of the SC applications dominates the delay of the entire SC circuit. As shown in Table 9, the worst-case delay doesn't occur from the SNGs, changing the RNG designs has no impact on the delay of the SC applications. Simulation also shows that the total power is significantly reduced when using the proposed scheme for both applications considered. As for the power-delay product (PDP) of the stochastic number

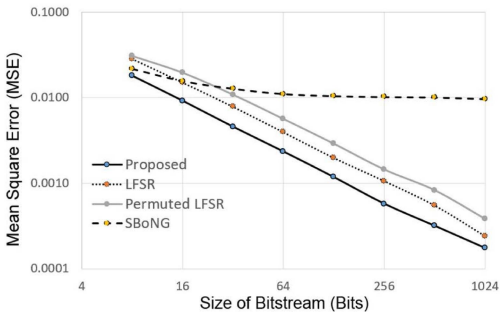


FIGURE 12. Mean square error (MSE) vs. size of bitstream in multiplier circuit; each bitstream is generated from 8 bits random number by different designs.

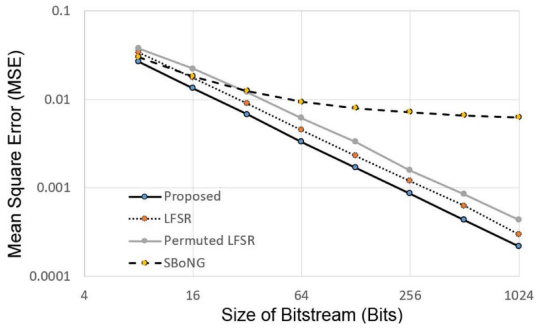


FIGURE 13. Mean square error (MSE) vs. size of bitstream in adder circuit; each bitstream is generated from 8 bits random number by different designs.

generators (SNGs), its value for the proposed design is better than for LFSR and SBoNG [11] but slightly worse than for the permuted LFSR [1] (due to its large delay). However, when the SNGs are used in the two considered SC applications, the delay of the stochastic applications dominates the delay of the SNGs; so for the SC multiplier and SC adder, the PDP of the proposed RNG design is better than all other RNG based designs, hence showing its validity and effectiveness for such widely used applications.

E. ACCURACY

Next, the accuracy comparison of stochastic circuits by using various random number generators is considered. Different types of random number generators are used to generate stochastic bitstreams as inputs for the SC multiplier and adder. The results of the SC multiplier and SC adder circuits are compared with the expected results; the Mean Square Errors (MSE) is used to evaluate errors from different RNGs. As in a prior section, the values of each input are selected randomly, and its value remains the same for the length of the bitstream times prior to updating its value.

To increase the length, the size of the random numbers in each dataset is increased by setting the number of bits in each row in a RNG plane to 32; by considering only the first 1,000,000 random numbers in each dataset, the results are shown in Figs. 12 and 13.

TABLE 11. Percentage *Improvement* of RNG Design vs. LFSRs When Considering a Plane of RNGs Size 12×16 bits

Metric	Percentage improvement from LFSR		
	Proposed	SBoNG [9]	Permuted LFSR [1]
Number of datasets	**1,466%**	700%	700%
Size of dataset	**183,001%**	0%	0%
Delay/RNG	-30.19%	**0%**	**0%**
Power/RNG	**47.03%**	25.57%	44.71%
Area/RNG	**48.71%**	22.57%	46.18%
Autocorrelation	**98.19%**	89.81%	89.85%
Cross-correlation	-721.46%	-163,945%	-231,632%
Mean Square Error (Multiplier circuit)	**40.62%**	-433.78%	-38.55%
Mean Square Error (Adder circuit)	**26.79%**	-247.14%	-34.24%

Figs. 12 and 13 present the mean square errors of the SC multiplier and adder circuits when bitstreams are generated from various RNG designs. As expected, the increase of the bitstream size enhances accuracy i.e., a reduction in errors. Compared to other RNG designs, the proposed RNG design has the lowest error, showing that it is very accurate.

F. PRODUCT METRIC

To evaluate each RNG design, a product metric (DPACP) is given in (3)

$$DPACP = delay \times power \times area \times correlation \quad (3)$$

where *power* is the average power dissipation per dataset, and *area* is the area per dataset. *Correlation* is the product of the cross-correlation and the auto correlation, i.e.,

$$correlation = cross\,correlation \times auto\,correlation$$

These parameters are normalized to equalize their significance such that a good RNG design has a low DPACP as combined metric for these 4 figures of merit.

Table 10 shows the DPACP of each considered design; the DPACP of the proposed RNG design is the lowest, followed by LFSR, the permuted LFSR [1], and SBoNG [11] respectively. The proposed design is better than these designs in terms of power dissipation, area, and correlation; however, its delay is higher. While both SBoNG [11] and permuted LFSR [1] are good in terms of power dissipation and area per RNG, the correlation is rather high, so the accuracy of these designs is low. For the LFSR, even though it has a high-power dissipation and area per RNG, its correlation is still low, so the LFSR is better than SBoNG [11] and the permuted LFSR [1].

Next, the percentage differences in each metric when comparing various RNG designs versus LFSR, are considered. A positive (negative) value shows an improvement (degradation) of an RNG design compared to LFSR. As shown in Table 11, even though the delay of the proposed RNG design is higher than LFSR and other RNG designs by 30.19%, the proposed RNG design is better than LFSRs and other RNG designs in all other figures of merit. Moreover, the mean square error (MSE) of the proposed RNG design is less (so more accurate) than these designs for the SC adder/multiplier circuits.

V. CONCLUSION

In a Stochastic Computing (SC) design, the Stochastic Number Generator (SNG) incur in a substantially large area, exceeding 80% of the overall circuit. This paper introduces a novel approach to random number generators (RNGs). The proposed RNG design leverages the inherent randomness between bits to generate larger sets of random numbers. The proposed design enhances RNG utilization through a sliding window and reversing technique, producing multiple RNGs from a shared RNG plane. To mitigate the potential high correlation from the sliding window (in which sharing of the significant bits between datasets is required), an isolation method is employed. Flip-Flops (FFs) are added at each RNG until cross-correlation of every dataset reaches an acceptable level.

In this paper, parameters in the proposed design are varied to study the impact of each parameter in the design; the results in this paper show that to generate large datasets of random numbers from the proposed design, the following criteria must be met.

- An increase in the number of rows in each module tends to increase the random numbers in each dataset. However, to generate a large dataset, the number of rows in each module must not be a power of 2 or an even number.
- An increase in the number of bits in each row of the proposed module increases the generated random numbers in each dataset.
- In an RNG plane, the number of modules in each row doesn't directly impact the size of generated random numbers in a dataset. However, the total bits in each row of an RNG plane significantly impacts the number of generated random numbers in each dataset. Large datasets of random numbers can be generated when the total bits in each row of an RNG plane is a power of 2.
- The inputs of the XOR gates in each module also impact the size of the random number in each datasets. There are several pairs of XOR inputs that can generate large random number datasets.

In stochastic computing (SC), both autocorrelation and cross correlation of RNG datasets significantly impact the accuracy; the proposed design uses a sliding window method to increase the number of generated datasets. When increasing the size of the RNG plane, the number of generated datasets is also increased, however correlation between these datasets is high, especially datasets that are generated from the data in the middle of the RNG plane in which its most significant bits are shared. Flip-Flops are used to isolate these datasets and reduce their correlation.

Compared to other RNG designs found in the technical literature, the proposed design offers the best performance per RNG in terms of power dissipation, area and correlation (as product of the cross-and auto-correlations); it incurs the

largest delay. However, when considering the product of all these figures of merit (i.e., DPACP) as a combined metric, the proposed design has the best performance. When using the proposed design for SC in two stochastic circuits (multiplier and adder), results show that the mean square error for the SC multiplier and adder circuits when bitstreams are generated using the proposed RNG design, is the lowest, so very accurate.

REFERENCES

[1] S. A. Salehi, "Low-cost stochastic number generators for stochastic computing," *IEEE Trans. Very Large Integr. Syst.*, vol. 28, no. 4, pp. 992–1001, Apr. 2020.

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

[3] H. Ichihara, T. Sugino, S. Ishii, T. Iwagaki, and T. Inoue, "Compact and accurate digital filters based on stochastic computing," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 31–43, Jan.–Mar. 2019.

[4] R. Wang, B. F. Cockburn, and D. G. Elliott, "Design, evaluation and fault-tolerance analysis of stochastic FIR filters," *Microelectronics Rel.*, vol. 57, no. 2, pp. 111–127, 2016.

[5] B. D. Brown and H. C. Card, "Stochastic neural computation. I. computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, Sep. 2001.

[6] N. Nedjah and L. de Macedo Mourelle, "Stochastic reconFigurable hardware for neural networks," in *Proc. IEEE Euromicro Symp. Digit. Syst. Des.*, 2003, pp. 438–442.

[7] S. E. Lyshevski, V. Shmerko, M. A. Lyshevski, and S. Yanushchkevich, "Neuronal processing, reconFigurable neural networks and stochastic computing," in *Proc. IEEE 8th Conf. Nanotechnol.*, 2008, pp. 717–720.

[8] S. Li, Q. Wang, X. Liu, and J. Chen, "Low-cost LSTM implementation based on stochastic computing for channel state information prediction," in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, 2018, pp. 231–234.

[9] W. J. Gross and V. C. Gaudet, *Stochastic Computing: Techniques and Applications*. Berlin, Germany: Springer, 2019.

[10] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A survey of stochastic computing neural networks for machine learning applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 7, pp. 2809–2824, Jul. 2021.

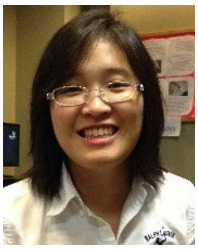
[11] F. Neugebauer, I. Polian, and J. P. Hayes, "Building a better random number generator for stochastic computing," in *Proc. IEEE Euromicro Conf. Digit. Syst. Des.*, 2017, pp. 1–8.

[12] H. Ichihara et. al., "Compact and accurate stochastic circuits with shared random number sources," in *Proc. IEEE 32nd Int. Conf. Comput. Des.*, 2014, pp. 361–3660.

[13] S. Mohajer, Z. Wang, K. Bazarga, M. Riedel, D. Lilja, and S. Faraji, "Parallel computing using stochastic circuit and deterministic shuffling networks," U.S. Patent 16/165713, Apr. 25, 2019.

[14] P. Junsangsri and F. Lombardi, "A pseudo-random number generator circuit for nanoscale stochastic computing (SC)," in *Proc. IEEE 23rd Int. Conf. Nanotechnol.*, 2023, pp. 299–304.

[15] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," in *Proc. IEEE 31st Int. Conf. Comput. Des.*, 2013, pp. 39–46.



PILIN JUNSANGSRI (Member, IEEE) received the B.Eng. degree in electrical engineering from Chulalongkorn University, Bangkok, Thailand, in 2006, and the M.S. degree in electrical and computer engineering, and the Ph.D. degree in computer engineering from Northeastern University, Boston, MA, USA, in 2010 and 2017, respectively. She is currently an Associate Professor with the School of Engineering, Wentworth Institute of Technology, Boston. Her past research included the simulation and design of the model of solar cells, design of non-volatile memory by using Emerging Technology such as memristor, phase change memory, programmable metallization cell, and racetrack memory. Her research interests include VLSI design, memory design, stochastic computing, and artificial intelligence.

FABRIZIO LOMBARDI (Life Fellow, IEEE) received the B.Sc. degree (Hons.) in electronic engineering from the University of Essex,Colchester, U.K., in 1977, the Diploma in microwave engineering, the master's degree in microwaves and modern optics, and the Ph.D. degree from the University of London, London, U.K., in 1978, 1978, and 1982, respectively. In 1977, he joined the Microwave Research Unit, University College London. He is currently the holder of the International Test Conference Endowed Chair Professorship with Northeastern University, Boston, MA, USA. His research interests include bio-inspired and nano manufacturing computing, VLSI design, testing, and fault/defect tolerance.