

This is the author-created version of the following work:

Dong, Xuening, Amirsoleimani, Amirali, Rahimi Azghadi, Mostafa, and Genov, Roman (2024) *WALLAX: A memristor-based Gaussian random number generator*. Neurocomputing, 566 .

Access to this file is available from:

<https://researchonline.jcu.edu.au/81672/>

© 2023 Elsevier B.V. All rights reserved.

Please refer to the original source for the final version of this work:

<https://doi.org/10.1016/j.neucom.2023.126933>

WALLAX: A Memristor-based Gaussian Random Number Generator

Xuening Dong, Amirali Amirsoleimani, Mostafa Rahimi Azghadi, and Roman Genov

Abstract—Generating Gaussian random numbers is essential in many applications such as cryptography, games, and computer simulations. Although software Gaussian Random Number Generators (GRNG) are widely used, hardware designs have been explored for their faster speed and lower computational cost. However, hardware GRNGs usually occupy large silicon areas when implemented in Complementary Metal Oxide Semiconductor (CMOS) technology, especially for their essential Uniform Random Number Generator (URNG) part. Here, we present a memristor-based GRNG, named WALLAX, conceived from the Wallace method to iteratively generate random numbers. This GRNG circuit benefits not only from the fully parallel analog-based Vector Matrix Multiplication (VMM) feature of memristive crossbars, but also harnesses the intrinsic stochastic switching behavior of the memristive devices to efficiently produce true random numbers. The vector-matrix multiplication of WALLAX is implemented on the memristive crossbar, while its random fetching step is realized by a URNG based on the stochastic switching nature of memristors. WALLAX successfully passes all the tests in the NIST 800-22 randomness test suite with 10^5 numbers generated and five goodness-of-fit tests with various pool sizes and effectively reduces the power and area consumption by 68.78% and 70.0% compared to digital implementations of the same GRNG method. The impact brought by memristor non-idealities is investigated by simulating the proposed structure with 1000 pools under various scenarios. Wire resistance and the stuck of state, each results in a 2.2% and 12.3% reduction in test pass rate within the tested range, respectively.

Index Terms—Memristor, Crossbar, Vector-Matrix Multiplication, Gaussian Random Number Generator.

I. INTRODUCTION

RANDOM number generators (RNGs), including the pseudo-random generators produced by deterministic algorithms [1] and the true-random generators produced by random physical processes [2], are widely applied to fields of stochastic computing, cryptography and Monte Carlo simulation (as shown in Fig. 1 (a-b)) [3]. Commonly seen RNGs include Uniform Random Number Generator (URNG) which produces numbers lying within a defined range with equal possibility of appearance and Gaussian Random Number Generator (GRNG) which draws random variables from a standard Gaussian distribution. The GRNGs are especially important as many real-valued random variables with unknown distributions converge to Gaussian distributions when the sample size is sufficiently large [4]. Therefore, they are widely applied to the simulation of real-world natural [5] and social science probabilistic models [6].

Xuening Dong and Roman Genov are with the Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, Ontario, Canada. Amirali Amirsoleimani is with York University, Canada. Mostafa Rahimi Azghadi is with the College of Science and Engineering, James Cook University, QLD 4811, Australia. Corresponding authors: Amirali Amirsoleimani and Mostafa Rahimi Azghadi, amirali.amirsoleimani@utoronto.ca, mostafa.rahimiazghadi@jcu.edu.au.

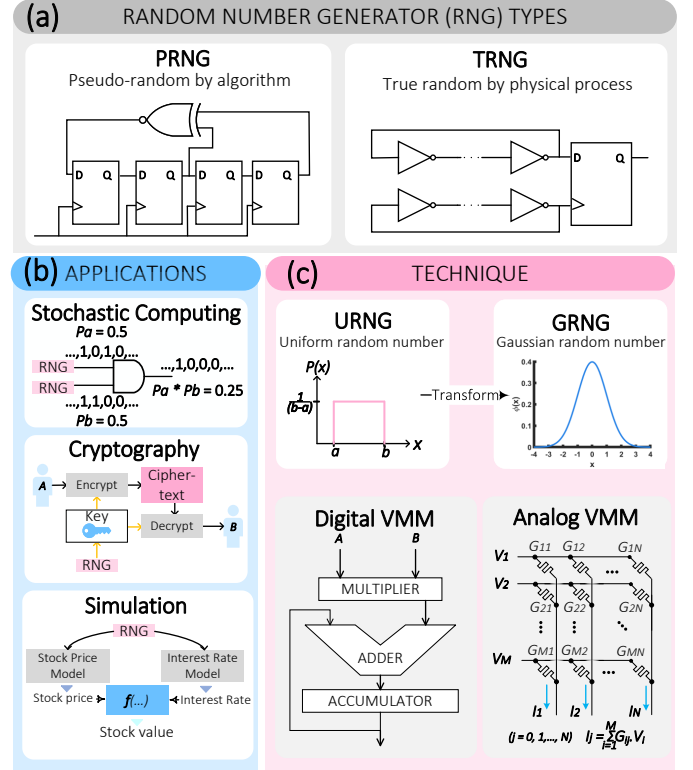


Fig. 1. RNG backgrounds: (a) RNG categories and current digital implementation technologies. (b) Examples of major applications of RNGs. (c) Major techniques for generating GRNs from URNs (top). The current inefficient digital technique for implementing VMM (bottom left), and the efficient memristive crossbar (bottom right) used in this work for VMM.

At present, several algorithms are available for generating GRNs, including Cumulative Distribution Function (CDF) inversion, transformation, rejection and recursion methods. However, these methods rely heavily on trigonometric, exponential and other logarithmic functions through software libraries [7]. Because, these mathematical functions are expensive to be accurately and efficiently implemented on hardware. The emergence of the Wallace method in [8] provided an alternative solution by replacing mathematical evaluations with recursive Vector Matrix Multiplications (VMM) to generate new GRNs from a pool of existing GRNs. Consequently, the Wallace method has been implemented in hardware systems such as FPGA with promising speed [9] [10]. However, the digital hardware devices cannot perform direct multiplication and addition efficiently; hence, such steps are divided into multiple shift registers and adder/subtractor circuits as depicted in Fig 1(c), which increase the area and power consumption [11].

Given the high power and area consumption of digital GRNGs, new techniques are being proposed using emerging

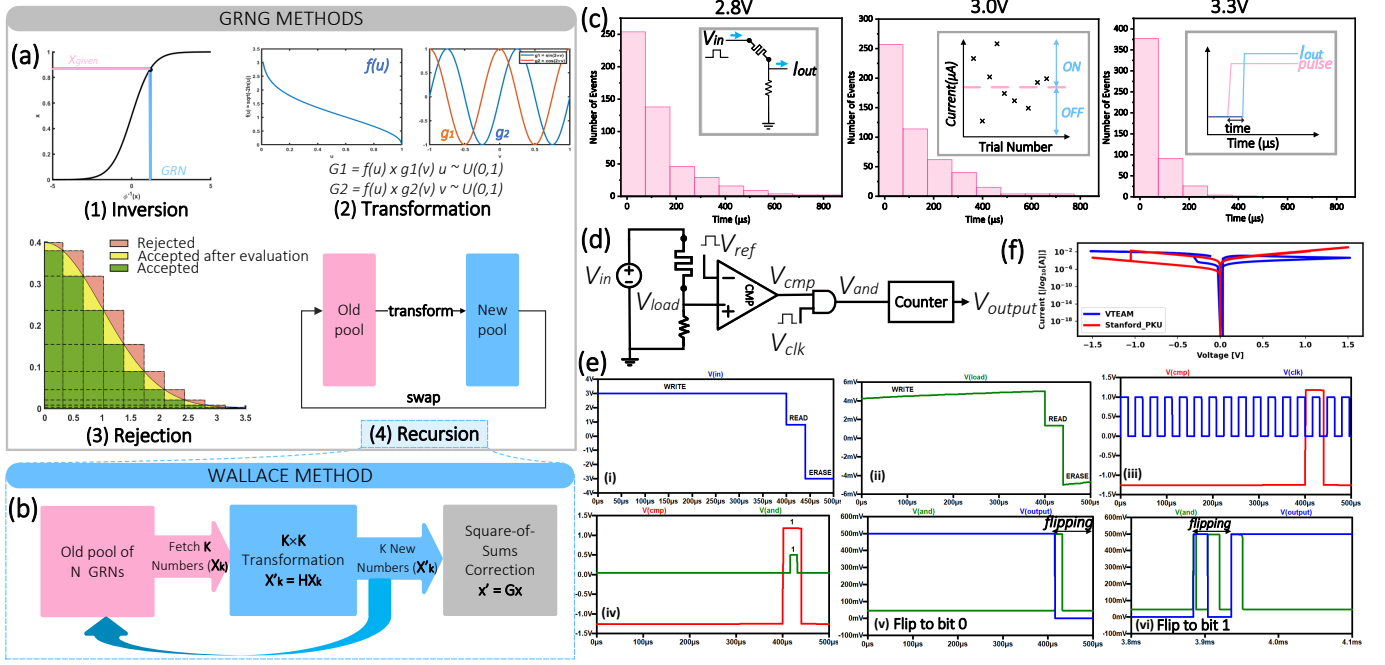


Fig. 2. (a) Four major methods for GRNG generation that are implemented in software/hardware. (b) Flow chart of the Wallace Method to be implemented in this work. (c) Random switching behaviour of memristor when different voltage pulses are applied, recorded as the number of switching events at different timestamps after the pulse is applied, following a Poisson distribution. (d) The URNG structure introduced in [12] that has the step-wise reconstruction of its output pulse as shown in (e), including (i) the input write-verify pulse, (ii) output pulse from the bottom electrode of memristor, (iii) output pulse of comparator (red) and the clock signal (blue), (iv) output pulse from the AND gate (green) and the final output from the counter indicating that the bit is flipped to (v) 0 and (vi) 1 in V_{output} . (f) the bipolar switching behaviour of models [13] [14] used for simulation.

analog devices. One such device is the memristor (resistive memory) that provides a highly efficient way for performing VMM based on Ohm's law for multiplication and Kirchhoff's law for addition. By organizing the devices into a crossbar architecture, the wordline voltages multiply with the conductance of individual devices to form the currents that sum up at the end of each bitline (as shown in Fig 1(c)) [11] [15]. All the operations are carried out in a single time step, therefore reduce the need for multiple pipelined stages in digital.

All aforementioned GRNG algorithms require URNGs for generating inputs. These URNGs are either directly implemented in software or using pseudorandom digital hardware such as linear feedback shift registers (LFSRs) which incurs high hardware cost [16]. The emerging memristive devices exhibit stochastic switching behaviour between ON/OFF states. This inherent feature that arises from the unpredictable formation of conductive filaments between the top and bottom electrodes [17], can be applied to implementing the URNG required for generating addresses in the Wallace method.

The use of memristors for building a GRNG, can lead to several advantages. First, the use of memristors in both VMM and URNG, can lead to lower power consumption and areas due to the low-power nanometer scale of memristive devices and their arrangement in a crossbar setting, which facilitates VMM operation in a power-efficient manner, compared to digital implementations. In addition, the Wallace method demands frequent operation on the memory unit for storing GRNs. This can be improved using an in-memory computing approach by minimizing read and write delays [18]. Finally, an effective way of implementing the Wallace method is to have multiple

distinct matrices for transformation in different iterations. The reprogramming ability of memristors allows for using the same crossbar to represent different matrices during runtime.

To benefit from the above-mentioned features, in this paper, we propose a memristor-based GRNG based on the Wallace method, with single-line parallel-connected memristors for URNG and memristor crossbars for VMM. The proposed GRNG is simulated with different non-ideal situations and thoroughly tested with various statistical and application tests. These tests show that our GRNG is capable of generating random numbers with similar test pass rates to the digital implementations. We also benchmark our design against existing implementations with respect to power and area consumption to show that our design significantly improves prior art.

Our specific contributions are as follows:

- 1) We propose the first memristive Wallace-based GRNG, which significantly reduces power and area consumption compared to literature;
- 2) We mitigate the accuracy loss during the crossbar computation by introducing a bit-slicing technique to reduce the required bit per cell;
- 3) We simulate and report the impact of various memristive non-idealities on the distribution of generated random numbers;
- 4) We replace the digital pseudo-random URNG with a memristor-based URNG from existing literature to enhance randomness by exploiting the intrinsic stochasticity in memristors.

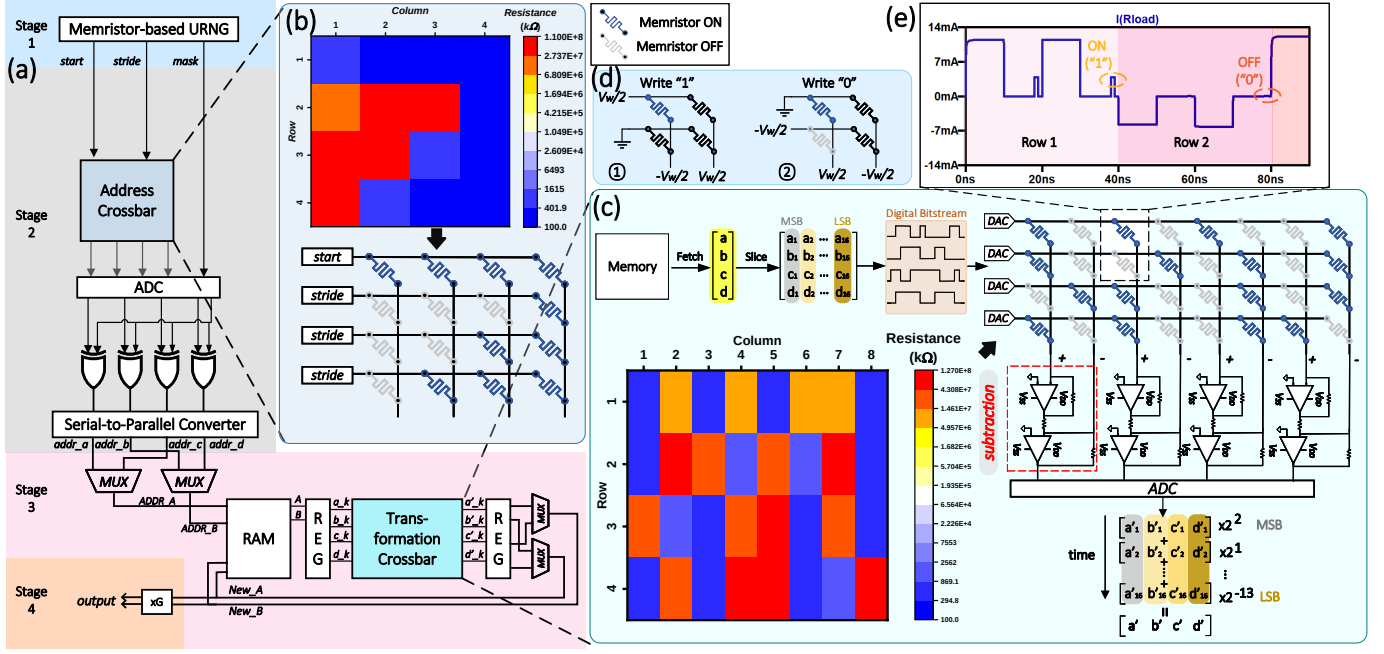


Fig. 3. WALLAX GRNG structure: (a) An overview of the proposed four-stage GRNG structure implemented in this work. (b) The heatmap for memristor resistance and the corresponding crossbar circuit used for generating random fetching addresses. (c) The heatmap for memristor resistance and the corresponding crossbar circuit used for transformation of GRNs to new numbers, together with the bit slicing technique adopted for enhancing the calculation accuracy. Both heatmaps use the red tiles indicating an "OFF" memristor at HRS and blue tiles indicating an "ON" memristor at LRS. (d) Methods adopted for writing "1" and "0" into the crossbar during the simulation. (e) Simulated current pulse of memristors being turned ON with current > 5 mA or OFF with current < 1 mA.

II. PRELIMINARY

A. Gaussian Random Number Generator Algorithms

The Gaussian distribution is a widely accepted statistical model for describing events in areas ranging from biology to sociology, and from physics to psychology. The essence of the distribution is the Central Limit Theorem (CLT), which states that when a sufficiently large amount of samples are obtained from an independent source population with replacement, the sample mean will invariably approximate the bell-curve shape [4]. This sampling process takes place in various situations in real life, especially in statistical related areas.

Major algorithms (as shown in Fig 2(a)) for generating GRN are categorized into four classes: 1) the Cumulative Distribution Function (CDF) inversion method, which directly infers GRN from its CDF [19], [20]; 2) transformation methods such as the Box-Muller [21]–[23], where the uniform random numbers are converted into GRN; 3) acceptance-rejection methods such as the Ziggurat algorithm [24], [25], which conditionally rejects some new values after transformation; 4) the recursion method such as the Wallace method, which loops through the linear combinations of available GRNs [8]. Most of these methods need evaluating complex mathematical functions such as logarithm, trigonometry and some contains conditional loops, which while can be easily implemented and approximated in software and on general-purpose processors, will require significant amount of power and computing resources to be realized. This limits their use in resource-constrained mobile and edge devices. Therefore, implementing them on energy- and resource-efficient hardware has drawn attention for research.

The CDF inversion methods, which are free of logic expression requirement, have been implemented using segmentation schemes and Lookup Tables (LUT) to approximate the CDF solution [19]. While this method is conceptually simple, it is computationally expensive. The design of LUT in the memory could simplify the number generation process, but is in trade off with the size of memory, which takes up several megabytes for a 16-bit arithmetic [20]. Both transformation and acceptance-rejection methods involve the evaluation of elementary functions. Hardware implementations of such methods have utilized Taylor's expansion [25], [26] or piece wise linear approximation [23], [27] to approximate the mathematical functions, compromising the memory space for higher accuracy and faster speed. Some recent work such as [28] have been focusing on the Polarization Decision Algorithm as an extension of the Box-Muller Method. These work benefited from the Intellectual Property (IP) cores on FPGA platforms to evaluate mathematical expressions for high computation precision yet with high power consumption [29]. Implementations of the recursion methods usually contains a memory for storing old GRNs and a computing unit for transformations. The works done in [9] and [10] have effectively developed on-chip units for generating GRNs recursively from a pool of pre-generated pseudorandom numbers. Though they have mathematically manipulated the transformation process to reduce the length of critical path, the number of logic gates involved in computations is still large. Also, this method results in large power and area consumption only in the transformation stage of the GRNG.

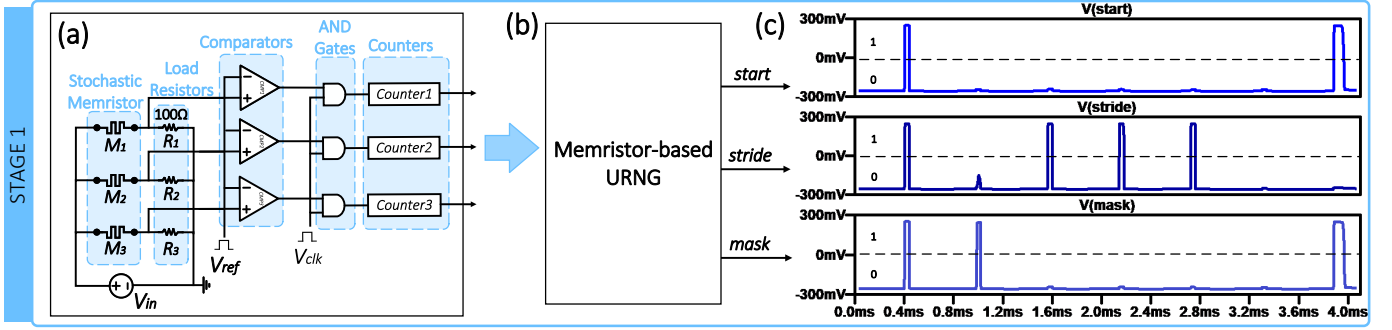


Fig. 4. Stage 1 (URNG): (a) circuit of URNGs connected in parallel, (b) block overview and (c) simulated output waveform of the proposed structure. The threshold voltage for deciding a bit "1" is 0mV.

B. The Wallace Method

The Wallace method (as shown in Fig 2(b)) effectively simplifies the process of obtaining Gaussian random numbers by eliminating the evaluations of elementary functions [8] [30]. It recursively performs orthogonal transformation to generate new random variables from a pool of $N = KL$ predetermined pseudonormal variates. Each iteration, a set of K pre-generated numbers are randomly drawn from the pool, forming a $K \times 1$ vector, named as X . The orthogonal identity matrix H is scrutinized to ensure the least amount of mathematical operations. The result X^1 will be calculated from $X^1 = HX$, labelled as the new set of variates and stored back into the pool. Since the linear combination of Gaussian random variables still follows a Gaussian distribution, and the involved variables $X_1, X_2, \dots, X_N$ are independent pseudonormal samples, the generated new variables should also follow a Gaussian distribution [8]. After a total of L iterations, the roles of the two pools are reversed and the transformation restarts. As the ideal fetching addresses are non-overlapping, the swap of pools can be simplified as rewriting the new GRNs back to the addresses from which the old pool numbers are drawn.

C. Uniform Random Number Generator

In each iteration of the Wallace method, K random addresses are required for fetching K numbers from the old-pool of GRNs. Current architectures of pseudorandom URNG used in hardware GRNG implementation are mainly single, multiple or combinational connections of LFSRs [31] [32]. These architectures inevitably employ large numbers of logical gates and trade-off between the statistical quality and the time/energy consumption. An alternative to these LFSR-based designs is the use of memristors which provide a promising way of generating random bitstreams with parallel-connected devices once at a time due to their stochastic switching nature. The resistance of a memristor is dependent on the current flow or voltage that has previously been applied to the device. The switching event of a memristor is either when the device turns ON or OFF by shifting between the High-Resistance State (HRS) and Low-Resistance State (LRS). At one instant, the resultant current flowing out of the memristor is randomly distributed across the defined threshold for ON state, as depicted in middle panel of the inset in Fig 2(c)). The switching mechanism vary randomly from device to device

due to fabrication differences and cycle to cycle due to the discrepancy in the conductive filament formation after a pulse is supplied. Therefore, this stochastic switching is intrinsic and natural, providing a reliable source of stochasticity for the generation of the required random bitstreams.

The switching delay is defined as the time difference between the onset of an applied pulse and the initial rise in the output current (see the insets of Fig 2(c) left and right panels). It is resulted from the different instances at which the actual switching takes place. As seen in Fig 2(c), this delay, follows a Poisson or Log Normal distribution when sampled in a large amount [33]. Due to this delay, the output currents of different memristors at different write-read cycles are different. Logic "1" or "0" then corresponds to whether the current is above or below a pre-defined threshold (see Fig 2(c) middle panel inset).

Several previous works [17], [34] have used this property to develop memristor-based URNGs. These implementations have been benchmarked against the National Institute of Standards and Technology (NIST) group of tests, and exhibited proven ability to generate bitstreams that are uniformly distributed and least likely to overlap. In this work, we focus on using the URNG introduced in [12] as shown in Fig 2(d). The URNG takes advantage of the random delay as mentioned before. The outputs from memristors are amplified and compared to a predefined threshold. The results are then sent to AND gates with high frequency clocks. The outputs of the AND gates then act as the clock to counters and the flipping of the counters' outputs eventually turn into a binary bitstream following a uniform distribution. The step-wise output pulse are reconstructed in Fig 2(e) with both bit 0 and 1 generated.

III. WALLAX ARCHITECTURE AND IMPLEMENTATION

Our proposed WALLAX system is simulated with both the VTEAM model [13] (for waveform simulations) and Stanford-PKU model [14] (for the subsequent validation tests) with the switching behaviour depicted in Fig 2(f). WALLAX consists of a four-stage hardware architecture (shown in Fig 3(a)) that produces $N - 1$ Gaussian random numbers at a time. The stages can be summarized as: 1) generating Uniformly-distributed Random Numbers (URNs); 2) converting URNs into addresses; 3) fetching the address corresponding GRN from memory, performing transformation and writing back;

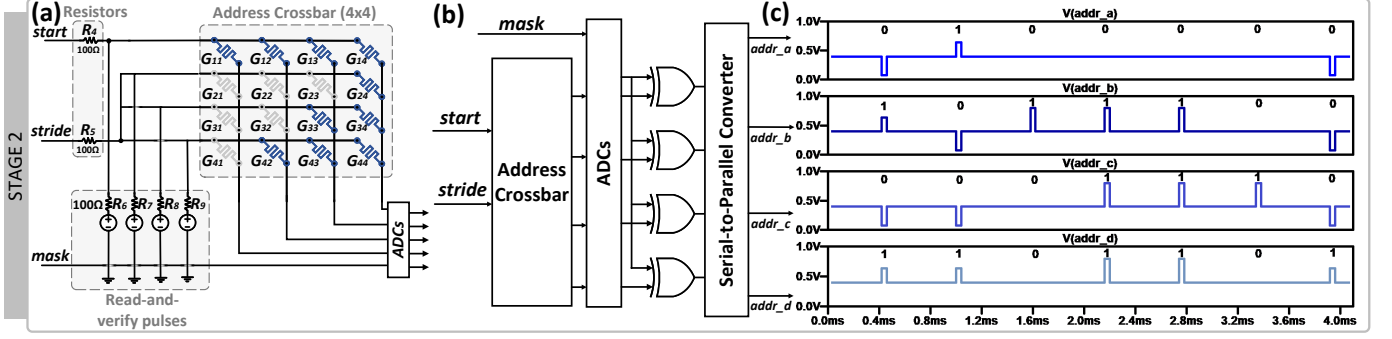


Fig. 5. Stage 2 (Address generation): (a) circuit of address crossbar, ADCs and the input read-and-verify pulses for writing bits into memristors, (b) block overview and (c) simulated output waveform of the proposed structure, with 0.5V recognized as a threshold for deciding whether a bit "1" is generated.

4) correcting the Chi-square correlation between each of the new numbers.

The initial pool of GRNs is generated by a software algorithm (e.g., Box-Muller), normalized to force their average squared value equals one and stored into the memory. Two memristor crossbars, one for generating the random addresses (Fig 3(b)) and another for the transformation of new GRNs (Fig 3(c)), are programmed with predefined matrices. These binary entries are written into the memristors by writing pulses in Fig 3(d) and verified by reading pulses in Fig 3(e).

For the consistency and simplicity of description, the pool size in this section is chosen to be $K = 4$ and $L = 128$, resulting in $N = K \times L = 512$. Each number in memory is represented by 16-bit fixed-point arithmetic (Q3.13).

A. Stage 1: URNG

This stage of the design is mainly the URNG to generate three random numbers, each with $\log_2(N) = 9$ bits, as shown in Fig 4(a) in details. The memristors are arranged with one terminal connected to the power source and the other terminal as the output [17]. In the stage 1 of our design, three URNGs from [12] are connected in parallel. The resultant three bitstreams are labelled as "start", "stride" and "mask" as shown in Fig 4(b) and (c), and passed to the second stage. This structure also allows for a more flexible number of bits generated, eliminating the waste of bits generated when the total number is not a base 2 number.

B. Stage 2: Address Generation

This stage is responsible for calculating the addresses of data in the memory using the URNs passed from stage 1. A detailed graph of this stage is included in Fig 5(a). These addresses should cover all of the numbers in the memory, while reduce the correlation between the numbers picked. Here, we adopt a technique following the description in [9] where the "start" bits are added to the "stride" bits or a multiple of them, and then XORed with the "mask" bits to build four addresses as:

$$addr_i = (start + stride \times i) \oplus mask, i \in \{0, 1, 2, 3\} \quad (1)$$

Benefiting from the multiply and accumulation nature of the memristor crossbar structure, the operations between "start" and "stride" is carried out in the crossbar. The generated bitstreams are then parallelized to form the fetching addresses,

as shown in Fig 5 (b) and (c), and used for fetching random numbers from the old GRN pool.

C. Stage 3: Transformation

This stage (as shown in Fig 6(a)) is where the transformation takes place. The randomly addressed GRNs in the old pool are grouped into a vector and then multiplied with an orthogonal matrix to achieve a linear combination of them. As mentioned before, the orthogonal matrix should be scrutinized to ensure the minimum calculation. Here we choose the Hadamard matrix:

$$H = \begin{bmatrix} 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} \quad (5)$$

to be consistent with the original work of Wallace. The entries of H are programmed into the crossbar before any calculation has taken place. Each entry of H is represented by the subtraction of the resistances of the two neighboring columns in the crossbar. Therefore, a value V should be represented as $V = V^+ - V^-$ to the crossbar. The resultant current of the two columns of the crossbars representing this value are then subtracted by the subtractor shown in Fig 6(b).

One prominent issue with the crossbar calculation is its precision. Due to non-idealities in the crossbar and the quantization problem in each memristor, the results computed in a crossbar can have large deviation from the correct value [35]. To address this issue, a bit-slicing technique, as shown in Fig 3(c) is applied to the crossbar. In this scheme, each of the 16-bit input GRNs are split bit by bit and form a digital bitstream from the most significant bit (MSB) to the least significant bit (LSB), as shown in the uncoloured region of Fig 6(d).

During each iteration, two numbers are fetched from the GRN memory and stored in the input register (see Fig 6(a) and 6(c)) until all four numbers have been collected. The four GRNs are grouped into a 4×1 vector X_k and fed into the rows of the transformation crossbar to perform $X'_k = HX_k$. The resultant current (as shown in Fig 6(e)) are sensed at the end of each column, stored into the output register and written back into memory two by two. The whole stage is governed by the control signals shown in Fig 6(f).

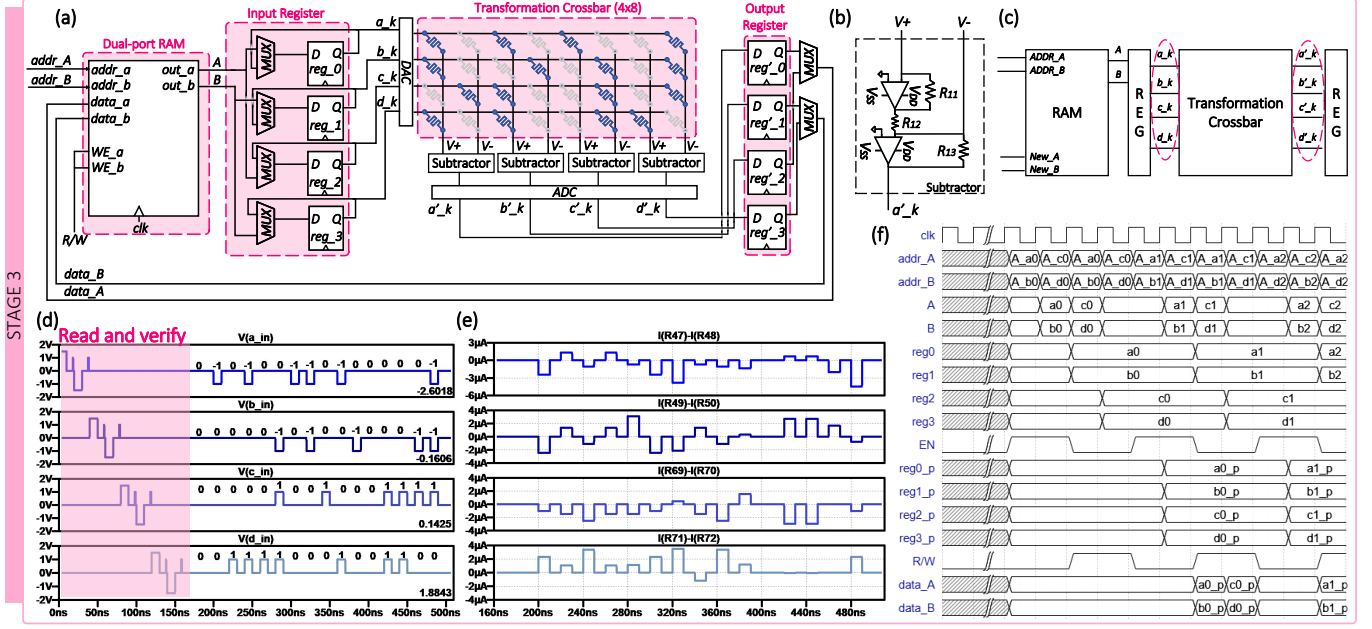


Fig. 6. Stage 3 (Transformation): (a) circuit of the memory block, transformation crossbar and buffers, with MUX control signals omitted, (b) subtractor circuitry for performing negative values in the array, (c) block-level overview, (d) simulated waveform for read-and-verify and bitstream representing the input old GRN values, (e) the resultant differential current flowing from the subtractors and (f) timing diagram of the proposed structure.

D. Stage 4: Chi-square correction

This stage is responsible for correcting the Chi-square relationships between the new numbers. When the GRNs are first generated for the initial pool, their sum of squares are standardized into 1. Due to the orthogonality of the matrix H , after the linear combination, the sum of squares of the new GRNs remains constant. Therefore, we need to retrieve their initial sum of squares ($\sum_{i=1}^N x_i^2$) to rewind the normalization.

The method used for Chi-square correction in this work is derived from [30], in which a scaling factor is applied to each of the outputs from stage 3. For the initial pool, their original sum of squares are calculated during the normalization and recorded as S_{init} . Therefore, the scaling factor G is initialized to be $\sqrt{\frac{S_{init}}{N}}$. However, due to the constant updates of the random numbers stored in the memory during the transformation, we need to approximate the Chi-square distribution followed by the sum of squares using

$$2\chi_N^2 \simeq (x + \sqrt{2N-1})^2 \quad (6)$$

where $x \sim \mathcal{N}(0,1)$. For a pool of N numbers, the target is to generate $N-1$ new Gaussian random numbers as the remained number x is used for approximating the subsequent Chi-square distribution. A variate $S \sim \chi_N^2$ is computed from equation (6) and the factor G is then:

$$G = \sqrt{\frac{S}{N}} = \sqrt{\frac{1}{2N}}(x + \sqrt{2N-1}). \quad (7)$$

For each new pool to be generated, x is fetched from the previous pool and N is a constant. This final step is carried out after the new GRN is stored in the memory, therefore, it has to be completely in digital. For the consistency of benchmarking, we decided to continually use the methodology introduced in

[9], that is, using a register to store the last iteration x value, perform equation (7) and multiply it with the new GRN.

IV. RESULTS AND DISCUSSION

The general requirements for RNGs are long-period and high quality of numbers generated [36]. In this section, we examine the performance of the proposed structure with various statistical and application tests. These tests are aimed at providing quantitative and qualitative measures on the robustness of randomness in the generated numbers and ensuring that the GRNs are applicable to real-world scenarios.

A. Simulation Methodology

The following tests were carried out using a simulation setup built upon the MemTorch platform [37] and [38]. Simulations performed using MemTorch accounted for two specific memristor models, i.e., VTEAM [13] and the Stanford-PKU RRAM model of [14] and focused on the influence of non-idealities on individual components such as temperature. Meanwhile, simulations performed on the passive crossbar arrays derived from [38] to investigate the overall effect of non-idealities to the crossbar structure by solving for an equilibrium solution for the given crossbar size and structure. All the parameters for the devices are listed in Table I. To maintain consistency between different test categories, each GRN is represented by 16-bit fixed-point numbers with 13 bits devoted to the decimal places (Q3.13). A bit-slicing into 1-bit patterns is applied to all the numbers involved in the crossbar computation.

B. Randomness Test

For a sampling population that follows a standard Gaussian distribution, there is equal probability of the CDF (Φ function)

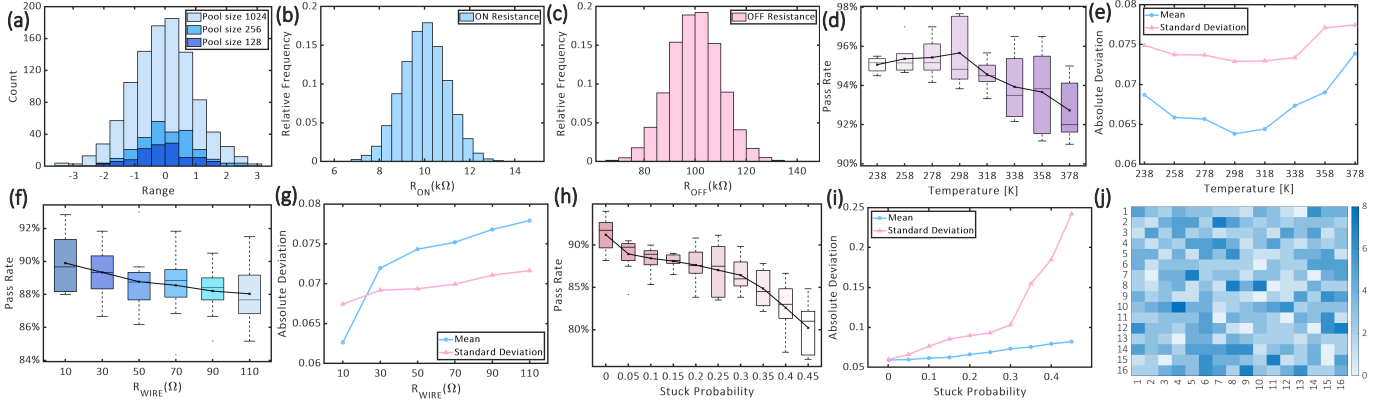


Fig. 7. (a) CDF visualization of the count of GRNs generated from three different pool sizes by WALLAX. (b) ON, and (c) OFF memristor resistance range during simulation of the proposed structure, following a normal distribution. (d) Change in statistical test pass rate due to the variation in ambient temperature. Each box represent the range of pass rates in 100 pools of 128 GRNs generated. The mean pass rates are connected to reflect the variation. (e) Fluctuations in the mean and standard deviation of the generated GRNs relative to temperature. (f) Change in statistical test pass rate due to the variation in wire resistance. Each box represent the range of pass rates in 1000 pools of 128 GRNs generated with the mean pass rates connected. (g) Fluctuations in the mean and standard deviation of the generated GRNs relative to the wire resistance. (h) Change in statistical test pass rate due to the variation in stuck probability of each memristor. Each box represent the range of pass rates in 1000 pools of 128 GRNs generated with the mean pass rates connected. (i) Fluctuations in the mean and standard deviation of the generated GRNs relative to the stuck probability. (j) The coverage of addresses generated by the stage 2 crossbar for 3 different pools of 256 random numbers.

TABLE I
SIMULATION PARAMETERS

VTEAM [13]		Stanford-PKU [14]		Passive [38]	
Crossbar Properties					
Ron (k Ω)	1	Ron (k Ω)	0.54	Ron (k Ω)	10
Roff (k Ω)	100	Roff (k Ω)	218.6	Roff (k Ω)	100
Ron/Roff variation range	$\pm 10\%$	Ron/Roff variation range	$\pm 10\%$	Ron/Roff variation range	$\pm 10\%$
Wire resistance	-*	Wire resistance	-*	Wire resistance	20
Stuck Probability	1%	Stuck Probability	1%	Stuck Probability	1%
Memristor Model Parameters					
d	3.00E-09	gap_init	2.00E-10	t_ox	1.20E-08
k_on	-10	g_0	2.50E-10	F_min	1.40E+09
k_off	0.0005	V_0	0.25	vel_0	10
alpha_on	3	I_0	0.001	E_a	0.6
alpha_off	1	read_voltage	0.1	a_0	2.50E-10
v_on	-0.2	T_init	298	delta_g_init	0.02
v_off	0.02	R_th	2100	model type	standard
x_on	0	gamma_init	16	T_crit	450
x_off	3.00E-09	beta	0.8	T_smth	500

* Not considered in the simulation platform.

of a number to be greater than or less than 0.5. Based on this fact, we performed a naive test for the distribution of random numbers by transforming the normal distribution into discrete uniform distribution $X \sim U(0, 1)$ using the determining rule of $\Phi(x) > 0.5$. The discrete uniform distribution was then tested with the NIST SP 800-22 test suite [39] with a threshold p-value of 0.01, and the results are included in Table II for a sample population of 409600 bits. The WALLAX structure successfully passed all of the test cases.

C. Goodness-of-Fit tests

The goodness-of-fit tests examine the distribution of sampling population subject to the normal bell-curve and we selected 1) the Chi-square (χ^2) test, 2) D'Agostino K-squared test, 3) K-S test and 4) A-D test to conclude the result. The null hypothesis is stated as: the observed data follows a normal distribution. The Chi-square (χ^2) test and D'Agostino K-squared test are based on the Chi-square probability, with the former comparing sample with expected frequencies in bins and the latter focusing on the skewness and kurtosis of the sample distribution [40]. The Anderson-Darling (A-D) Test [41] and Kolmogorov-Smirnov (K-S) test [42] are

TABLE II
NIST RANDOMNESS TEST RESULTS

Test Name	Result	
	p-value	Pass/Fail
Frequency Test	0.72944	Pass
Block Frequency Test	0.19454	Pass
Run Test	0.29514	Pass
Run Test (Longest Run of Ones)	0.77457	Pass
Binary Matrix Rank Test	0.03090	Pass
Discrete Fourier Transform (Spectral) Test	0.41441	Pass
Non-overlapping Template Matching Test	0.38492	Pass
Overlapping Template Matching Test	0.42927	Pass
Universal Statistical Test	0.36927	Pass
Linear Complexity Test	0.67442	Pass
Serial Test	0.37171	Pass
	0.73266	
Approximate Entropy Test	0.40345	Pass
Cumulative Sums	0.70450	Pass
	0.86926	
Random Excursion Test	0.76415	Pass
Random Excursion Variant Test	0.55431	Pass

used for checking the continuous properties, with the A-D test adopting more weights to the tail regions. In the first three tests, decisions are made on comparing the test score with a determined threshold of 0.05. For the A-D test, the statistics are compared with the critical value at each significance level, and rejected when the it is greater than the critical value.

The goodness-of-fit tests are carried out with different pool sizes with identical non-ideality settings. The results (p-values) from all tests have been summarized in Table III. Our design passes all tests regardless of the initial pool size. For the A-D test specifically, the statistic value follows the expected critical value at different significance levels. Also, by directly inspecting the CDF plot in Fig 7(a), the distribution of generated numbers well fits into a bell curve shape especially when the pool size is larger. Therefore, the design has general reliability in generating random numbers following the Gaus-

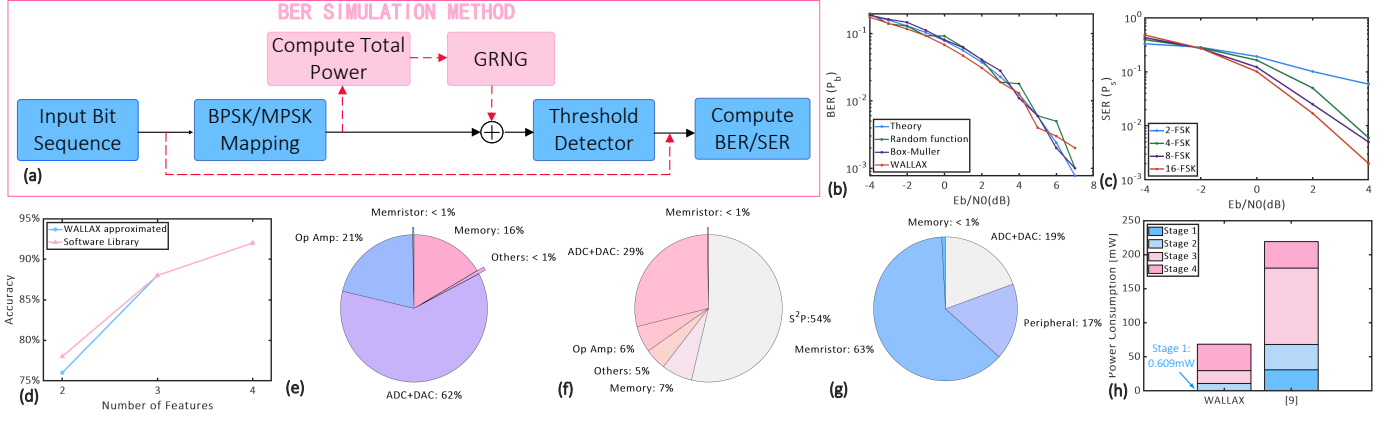


Fig. 8. (a) Flow chart of the process carried for the BER test. (b) BPSK and (c) MPSK BER simulation result when WALLAX is used as a Gaussian White Noise source. (d) The classification accuracy on the Iris dataset achieved using a WALLAX-based Gaussian Naive Bayes Classifier. Percentage breakdown of (e) power, (f) area and (g) latency consumption of the first three stages of proposed design (URNNG omitted in latency breakdown). (h) Stage-wise power consumption comparison between WALLAX and [9].

TABLE III
THE GOODNESS-OF-FIT TEST RESULT

Model	Pool Size	χ^2	A-D	K-S	D'Agostino
WALLAX	128	0.697	0.175	0.966	0.774
	256	0.674	0.418	0.368	0.065
	1024	0.111	0.485	0.733	0.350

sian distribution. When inspecting Table III, there are some p-values slightly above the boundary. These are mainly due to the arithmetic imprecision and non-idealities in the crossbars during the transformation such that the results shift away from their positions in the distribution. The conductance state of each memristor is manipulated by applying write pulses to the row and column across it. However, the ON and OFF resistance of a memristor can vary even if the same pulse is applied, due to cycle to cycle and device to device variations. As shown in Fig 7(b) and (c), R_{ON} and R_{OFF} values vary within a normal distribution in our model and could result in a deviated calculation. Moreover, as the sample size and transformation iteration increase, there are larger possibilities for the non-idealities and extreme conditions to affect the final result.

D. Ambient Temperature Effect

Variation in ambient temperature affects the charge carrier mobility of the memristor and consequently alter its I-V characteristics and ON/OFF state resistance [43]. Extreme temperatures could potentially lead to imprecise computations due to changing resistances. We investigate the impact of ambient temperature on the crossbar based on simulation of a temperature-dependent RRAM model in [14]. Simulation results have been shown in Fig 7(d) and (e), reporting on the goodness-of-fit test pass rate over 100 pools of $N = 128$ GRNs and the absolute deviation in mean and standard deviation over the generated distribution. The best pass rate and least deviations are encountered when the ambient temperature is close to room temperature at 298K. Although a drop in pass rate and ascent in deviation is observed when the temperature varies, the pass rate never drops below 92%. This shows

that the proposed design has good tolerance to temperature variation.

E. Robustness against Non-ideality

Non-idealities in the crossbar strongly effects the precision of crossbar computations. They bring unexpected fluctuations in the sensing current and lead to deviation in the final result. To test the influence of non-idealities on the computation quality of the crossbar, the following tests are carried out under different settings. The pool size ($N = 128$) is kept constant across all the tests. Results are recorded per 100 tests in a group and reported as the average of 10 groups to conclude the impact of non-idealities with different random seeds.

The sneak-path current is a prevailing non-ideality within the crossbar, referring to the undesired flow of current relative to the selected current path. Since we are using a passive memristor crossbar structure without transistor gates, the sneak-path current can flow bidirectionally across memristors and wires. Therefore, it significantly impacts the accuracy of calculation by injecting unwanted current to the bitlines. Consequently, we included sneak-path currents in our simulations in conjunction with memristive non-ideality scenarios to study its impact.

The wire resistance in connecting individual memristors in a crossbar degrades the signal passing through it, especially for cells further away from the source. The electric signal flowing into or out of the crossbar are altered by the wire resistance. The collection of sensing currents at the end of each column is susceptible to these fluctuations. To test the effect of wire resistance, we kept increasing the segment resistance between individual memristors in the crossbar within the range $[10, 110]\Omega$ and observed the pass rates of above goodness-of-fit tests in 1000 pools. As seen in Fig 7(f), there is a noticeable trend of dwindling pass rate as the wire resistance increases. The pass rate drops from 90% at 10Ω to 88.0% at 110Ω , yet the majority of decrements are less than 0.5%. There is also a 24.39% and 6.23% increase, respectively, in the absolute deviation from the standard μ and σ of the generated GRNs, as indicated in Fig 7(g). It is worth noting that, the size of crossbars used in the address calculation (4×4) and transformation (4×8) are relatively small, thus, even the

TABLE IV
STAGE-WISE POWER, AREA, AND LATENCY METRICS FOR THE WALLAX STRUCTURE.

Component	Params.	Stage 1: URNG				Stage 2: Address Generation				Stage 3: Transformation				
		Spec	Power (mW)	Area (mm ²)	Latency (μs)*	Spec	Power (mW)	Area (mm ²)	Latency (μs)*	Spec	Power (mW)	Area (mm ²)	Latency (μs)*	
ADC	Resolution	N/A				8 bits	10	6.00E-03	8.30E-04	8 bits	8	4.80E-03	8.30E-04	
	Number					5				4				
	Frequency					1.2GHz				1.2GHz				
DAC	Resolution	N/A				1 bit	1.56E-02	6.64E-07	8.00E-04	1 bit	1.56E-02	6.64E-07	8.00E-04	
	Number					4				4				
AND	Number	3	0.030	3.10E-05	33.333 [12]	N/A				N/A				
CMP	Number	3	0.542	5.83E-05		N/A				N/A				
Counter	Number	3	0.032	1.68E-03		N/A				N/A				
XOR	Number	N/A				4	2.07E-02	1.78E-05	8.20E-05	N/A				
S ² P	Number	N/A				4	0.126	2.02E-02	0.01	N/A				
Subtractor	Number	N/A				N/A				4	5.6	2.12E-03	5.00E-06	
Multiplexer	Number	N/A				N/A				8	0.24	5.21E-05	2.25E-05	
Dual-port RAM	Size	N/A				N/A				2KB	4.59	2.48E-03	8.21E-05	
IR + OR	Size	N/A				N/A				256B	0.23	7.70E-04	8.21E-05	
Memristors	Size	3	5.27E-03	6.28E-06		16	2.81E-02	3.35E-05	0.01	32	5.62E-02	6.70E-05	0.01	
	Bit per cell	2				2				2				
Total			0.609	0.002				10.291	0.026	0.044		18.592	0.010	0.032

* The latency is listed as individual element.

** IR = input register, OR = output register

cumulative wire resistances are comparatively smaller than the ON/OFF resistance of memristors, which are assumed to be 10^4 and 10^5 .

As the number of calculations performed on a crossbar increases, the crossbar inevitably experiences ageing effect which leads to potential stuck of state in individual memristors [44]. In addition, when crossbars are fabricated or initially programmed, some of their memristors may stuck at high or low resistance states. This leads to imprecise result especially when the problem pertains to the most significant bits of the number. Moreover, the leakage current will have its maximum impact when a memristor in its path is stuck at ON state, while it should have been OFF. Therefore, we carried out another set of tests following the same procedure as the wire resistance test, investigating a ranging stuck ON/OFF probability (p_{stuck}) from 0.001 to 0.45. The test results are summarized in Fig 7(h). We observe a clear decreasing trend with respect to increasing stuck probability, from 91.2% at $p_{stuck} = 0.001$ to 80.2% at $p_{stuck} = 0.45$. Steeper decrements appear when the probability approaches 0.5. Meanwhile, the μ and σ of the generated GRNs shift, respectively, 38.9% and 305% from the standard value, as shown in Fig 7(i). With more memristors in the crossbar stuck at ON/OFF state, more bits during the calculation are altered and the distribution eventually deviates.

Other factors such as the endurance and retention of the memristors are also considered in the simulation. However, since there is only one write operation and $N/4$ read operations per memristor, their impacts are negligible.

F. Address Crossbar Coverage

Even though we have adopted the add-XOR technique to ensure the optimal coverage of GRN addresses, the actual generated random addresses can hardly cover all of the addresses equally due to URNG itself and the imprecision in the crossbar. Fig 7(j) indicates the address coverage generated by stage 2 of the proposed WALLAX structure with a pool size of 256, repeated for three different pools. The darker the colour, the more frequent coverage of certain addresses of the memory. The plot shows an uneven distribution of the addresses, with the majority covered between [3, 5] times while some extreme cases have 0 or 8 coverage. Despite the

uneven address distribution, the three pools all successfully passed the above statistical tests, which demonstrate the lesser effect of address crossbar non-ideality to the performance of the overall GRNG.

G. Application test

GRNGs are embedded in different practical applications. To truly test the usefulness of our proposed WALLAX structure, we apply it to different real-world scenarios, while testing its two key properties, i.e. randomness and distribution.

The first application is the Binary Phase Shift Keying (BPSK) Bit Error Rate (BER) testing - based on the Monte Carlo simulation with 1000 symbols to transmit [45]. The overall work flow is shown in Fig 8(a). Here, WALLAX acts as a source of Gaussian white noise. To test its random nature, we compare the simulated BER when employing WALLAX, with theory, a software-based Box-Muller method, and the *randn* function in the Numpy library. The results are shown in Fig 8(b) with selected GRNG algorithms. There are no distinguishable peculiarities in all algorithms; however, we notice a gradually increasing deviation from the theoretical value as E_b/N_0 increases. This suggest that the imprecision and non-idealities in the calculation lead to a growing error rate when E_b/N_0 gets sufficiently large.

The test is then extended to the simulation of M -ary Phase Shift Keying (M-PSK) modulation, where M refers to the range of information symbols m that are created and transmitted ($m \in \{1, 2, \dots, M\}$). The result is shown in Fig 8(c), in which the trend of Symbol Error Rate (SER) is displayed. Similar to the BER testing, we randomly created 1000 symbols to transmit. As expected, when M increases, the SER shows a faster descending trend.

Another set of application tests was performed to test the nature of GRNs following a normal distribution. For this, the Gaussian Naive Bayes Classifier was used, which works on the Bayesian rule and assumes that the attributes to the class labels follows a Gaussian distribution [46]. The generated random numbers are used for approximating the PDF of the normal distribution and each time the probability density is read as the interception between the x-axis label and the estimated curve. In this test, the Iris dataset [47] was used with different

TABLE V
COMPARISON OF WALLAX AND PREVIOUS WALLACE-BASED GRNG DESIGNS

Design	[9]	[10]	WALLAX		
			Baseline	Optimal Power	Optimal Speed
Technology	CMOS	CMOS	CMOS +RRAM	CMOS +RRAM	CMOS +RRAM
Technology Node	150nm	N/A	32nm	32nm	32nm
Bitwidth	16	4-16	16	16	16
URNG	LFSR	LFSR	[12]	[12]	[49]
Pipelining	Yes	Yes	No	No	Yes
Speed [MHz]	155	117.63	0.12	0.118	125
Power [mW]	219.13	560.25*	68.41	56.45	137.25
Area [mm^2]	53.9	—	16.16	16.13	16.14

* 64 units in parallel for stage 3.

TABLE VI
PERFORMANCE COMPARISON OF EXISTING MEMRISTOR-BASED URNGS

	[12]	[49]	[50]	[51]
Material	Ag:SiO ₂	Ti/HfO _x	Cu _x Te _{1-x}	W/TiN/ TiON/SiO ₂ /Si
Source of Randomness	Delay	Coupling of memristors	Delay and Relaxation Time	Random Telegraph Noise
Bit Generation Rate [bs ⁻¹]	6K-300K	0.16K-1G	32K-100M	1K-100M
NIST Test	Passed	143/148 Passed	Passed	Passed

numbers of (2-4) features applied to the classifier. As shown in Fig 8(d), the WALLAX approximated Naive Bayes Classifier is able to achieve a similar accuracy to that of the *scikit-learn* library [48]. The WALLAX approximated classifier achieves a maximum of 92% accuracy over all four features.

V. OVERHEAD ANALYSIS AND COMPARISON

To fully demonstrate the improvement in resource consumption of the proposed design, we benchmarked it against existing digital GRNGs implementing the same algorithm. The estimation of resource consumption for WALLAX is based on the digital memristor model since only 0s and 1s are stored, and 32nm CMOS technology for the peripheral circuitry [52]–[60] with scaling factors derived from [61]. Detailed estimation with resource consumption disaggregated to device-level are recorded in Table IV and the consumption breakdown is depicted in Fig 8(e-g). Note that, since Stage 4 is purely digital and includes iterative calculations, its design is assumed to be identical to the Stage 4 in [9]. The cycle of random number generation ends up at the end of Stage 3 when the new numbers are written back into the memory. Therefore, Stage 4 is excluded from the latency calculation and resource consumption estimations in Table IV. The resource estimation of the existing models [9], [10] are derived from resources identified in their respective paper, and transformed into actual numbers by [62] [63]. The results are reported in Table V.

Due to the high resistance range (10 – 100k Ω), the power consumed by all 51 memristor components (3 parallelized memristors in stage 1, a 4 $\times$ 4 address crossbar in stage 2

and a 4 $\times$ 8 transformation crossbar in stage 3) is less than 0.1mW, which is only 0.2% of the overall power consumption of our design as shown in Fig 8(e). Apart from the digitally implemented Stage 4, ADCs take up the largest portion of power consumption in the design. WALLAX has an overall 68.391mW power consumption, which is an improvement of 68.78% compared to the similar 4-stage Wallace-based structure in [9]. This improvement can be seen in all the first three stages of the model as shown in Fig 8(h). Moreover, the memristors have a nanometer scale, which yields a small area consumption when compared with the digital implementations. When compared to the similar 4-stage structure in [9], WALLAX achieves a 70.0% area reduction.

There is a trade-off between the resource consumption and the throughput of various GRNG designs. Previous Wallace-based GRNG designs can all achieve a speed of over 117 MHz, at the significant expense of power and area consumption. In its baseline design, WALLAX can achieve a speed of 120 KHz without pipelining. The speed is largely limited by the URNG, which can only achieve an optimal bit generation rate of 300 kb/s [12]. Meanwhile, the speed of both crossbars in stage 2 and 3 is contingent on the length of bitstream and the number of bits in each slice. Under the scenario of 16-bit fixed-point numbers with 16 slices, each stage results in a latency of 0.044 μ s and 0.032 μ s, taking up the largest portion of latency as shown in Fig 8(g). The total latency for generating four GRNs is 33.41 μ s without any parallelized operations, which is much higher than digital implementations.

The performance of the proposed structure can be enhanced by changing the organization of the power consuming ADC units or using a faster URNG with pipelining. In Table V, the optimal power model is realized by sharing the ADCs in the crossbars. This will result in additional sample and hold units and longer latency, but the power consumption reduces by 17.5% compared to the baseline. The huge latency in the baseline design can be reduced by proper pipelining and the introduction of a URNG with higher throughput. In Table VI, we benchmarked four existing memristor-based URNGs with different randomness sources. The most optimal URNG attains a theoretical maximum speed of 1 Gbs⁻¹ with sacrifice in stability of randomness as NIST tests are not fully passed [49]. Using this speed-optimized GRNG, as shown in Table V, WALLAX can reach a 125MHz speed with proper pipelining.

VI. CONCLUSION

In this work, we proposed WALLAX, a low-power and compact memristor-based GRNG implementing the Wallace method. The four-stage structure of WALLAX iteratively generates new GRNs by performing efficient crossbar-based VMM and generating random addresses using a memristive crossbar. The credibility of our design was examined with a various range of statistical tests. When compared with the state-of-the-art digital GRNG implementations, WALLAX achieved an improvement of 68.78% and 70.0% in power and area consumption, respectively, while operating at a comparable speed. Future work may include mitigation or alleviation of memristive non-ideal impacts, and development of methods to accelerate WALLAX.

REFERENCES

- [1] P. L'Ecuyer, "Random number generation and quasi-monte carlo," *Wiley StatsRef: Statistics Reference Online*. Wiley, 2014.
- [2] B. Sunar, W. J. Martin, and D. R. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Transactions on Computers*, vol. 56, no. 1, pp. 109–119, 2007.
- [3] S. Raychaudhuri, "Introduction to monte carlo simulation," in *2008 Winter Simulation Conference*, 2008, pp. 91–100.
- [4] S. Kwak and J. Kim, "Central limit theorem: The cornerstone of modern statistics," *Korean Journal of Anesthesiology*, vol. 70, p. 144, Apr 2017.
- [5] T. B. Farver, "Chapter 1 - concepts of normality in clinical biochemistry," in *Clinical Biochemistry of Domestic Animals (Sixth Edition)*, J. J. Kaneko, J. W. Harvey, and M. L. Bruss, Eds. San Diego: Academic Press, 2008, pp. 1–25. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780123704917000015>
- [6] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 1082–1090. [Online]. Available: <https://doi.org/10.1145/2020408.2020579>
- [7] J. Malik and A. Hemani, "Gaussian random number generation: A survey on hardware architectures," *ACM Computing Surveys*, vol. 49, pp. 1–37, Nov 2016.
- [8] C. Wallace, "Fast pseudorandom generators for normal and exponential variates," *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, pp. 119–127, Mar 1996.
- [9] D.-U. Lee, W. Luk, J. Villasenor, G. Zhang, and P. Leong, "A hardware gaussian noise generator using the wallace method," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, pp. 911 – 920, Sep 2005.
- [10] R. Cai, A. Ren, N. Liu, C. Ding, L. Wang, X. Qian, M. Pedram, and Y. Wang, "Vibnn: Hardware acceleration of bayesian neural networks," *ACM SIGPLAN Notices*, vol. 53, pp. 476–488, Mar 2018.
- [11] A. Amirsoleimani, F. Alibart, V. Yon, J. Xu, M. Pazhouhandeh, S. Ecoffey, Y. Beilliard, R. Genov, and D. Drouin, "In-memory vector-matrix multiplication in monolithic complementary metal-oxide-semiconductor-memristor integrated circuits: Design choices, challenges, and perspectives," *Advanced Intelligent Systems*, vol. 2, p. 2000115, Aug 2020.
- [12] H. Jiang, D. Belkin, S. Savel'ev, S. Lin, Z. Wang, Y. Li, S. Joshi, R. Midya, C. Li, M. Rao, M. Barnell, Q. wu, J. J. Yang, and Q. Xia, "A novel true random number generator based on a stochastic diffusive memristor," *Nature Communications*, vol. 8, 2017.
- [13] S. Kvatinisky, M. Ramadan, E. G. Friedman, and A. Kolodny, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.
- [14] Z. Jiang, Y. Wu, S. Yu, L. Yang, K. Song, Z. Karim, and H.-S. P. Wong, "A compact model for metal-oxide resistive random access memory with experiment verification," *IEEE Transactions on Electron Devices*, vol. 63, no. 5, pp. 1884–1892, 2016.
- [15] M. Rahimi Azghadi, Y.-C. Chen, J. Eshraghian, J. Chen, C.-Y. Lin, A. Amirsoleimani, A. Mehonic, A. Kenyon, B. Fowler, J. Lee, and Y.-F. Chang, "Complementary metal-oxide semiconductor and memristive hardware for neuromorphic computing," *Advanced Intelligent Systems*, vol. 2, May 2020.
- [16] W. A. Borders, A. Z. Pervaiz, S. Fukami, K. Y. Camsari, H. Ohno, and S. Datta, "Integer factorization using stochastic magnetic tunnel junctions," *Nature*, vol. 573, no. 7774, pp. 390–393, 2019.
- [17] S. Gaba, P. Knag, Z. Zhang, and W. Lu, "Memristive devices for stochastic computing," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 2592–2595.
- [18] N. Verma, H. Jia, H. Valavi, Y. Tang, M. Ozatay, L.-Y. Chen, B. Zhang, and P. Deaville, "In-memory computing: Advances and prospects," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 3, pp. 43–55, 2019.
- [19] R. C. Cheung, D.-U. Lee, W. Luk, and J. Villasenor, "Hardware generation of arbitrary random number distributions from uniform distributions via the inversion method," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, pp. 952 – 962, Sep 2007.
- [20] J. Chen, J. Moon, and K. Bazargan, "Reconfigurable readback-signal generator based on a field-programmable gate array," *IEEE Transactions on Magnetics*, vol. 40, no. 3, pp. 1744–1750, 2004.
- [21] G. Box and M. Muller, "A note on the generation of random normal deviates," *Annals of Mathematical Statistics*, vol. 29, pp. 610–611, Jun 1958.
- [22] N. Kasiviswanathan and K. Srivatsan, "An efficient hardware implementation of gaussian random number generator," in *2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2)*, 2017, pp. 327–331.
- [23] M. Maamoun, H. A. Saadi, S. Dahmani, G. Zerari, N. Chabini, and R. Beguenane, "An optimized fpga based box-muller gaussian random number generator architecture for communication applications," in *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2021, pp. 0772–0777.
- [24] W. Tsang and G. Marsaglia, "The ziggurat method for generating random variables," *Journal of Statistical Software*, vol. 05, Jan 2000.
- [25] J. Su and J. Han, "An improved ziggurat-based hardware gaussian random number generator," in *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2016, pp. 1606–1608.
- [26] G. Zhang, P. Leong, D.-U. Lee, J. Villasenor, R. Cheung, and W. Luk, "Ziggurat-based hardware gaussian random number generator," in *International Conference on Field Programmable Logic and Applications*, 2005., 2005, pp. 275–280.
- [27] H. Edrees, B. Cheung, M. Sandora, D. Nummey, and D. Stefan, "Hardware-optimized ziggurat algorithm for high-speed gaussian random number generators," in *International Conference on Field Programmable Logic and Applications*, 2009, pp. 254–260.
- [28] Y. Hu, Y. Wu, Y. Chen, and G. Wan, "Gaussian random number generator: Implemented in fpga for quantum key distribution," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 2019.
- [29] A. Nafkha and Y. Louet, "Accurate measurement of power consumption overhead during fpga dynamic partial reconfiguration," in *2016 International Symposium on Wireless Communication Systems (ISWCS)*, 2016, pp. 586–591.
- [30] R. Brent, "Some comments on c. s. wallace's random number generators," *The Computer Journal*, vol. 51, 2010.
- [31] S. irfan ullah, M. Khan, and M. Khiyal, "Intelligent algorithm design of the lfsr based stream cipher," *World Applied Sciences Journal*, vol. 30, pp. 498–505, Jan 2014.
- [32] P. L'Ecuyer, "Maximally equidistributed combined tausworthe generators," *Math. Comput.*, vol. 65, pp. 203–213, Jan 1996.
- [33] R. Naoos, M. Al-Shedivat, and K. Salama, "Stochasticity modeling in memristors," *IEEE Transactions on Nanotechnology*, vol. 15, pp. 1–1, Jan 2015.
- [34] S. Lv, J. Liu, and Z. Geng, "Application of memristors in hardware security: A current state-of-the-art technology," *Advanced Intelligent Systems*, vol. 3, p. 2000127, 2020.
- [35] M. A. Zidan, Y. Jeong, J. Lee, B. Chen, S. Huang, M. J. Kushner, and W. D. Lu, "A general memristor-based partial differential equation solver," *Nature Electronics*, vol. 1, no. 7, pp. 411–420, 2018.
- [36] H. Nguyen, *Gpu Gems 3*, 1st ed. Addison-Wesley Professional, 2007.
- [37] C. Lammie, W. Xiang, B. Linares-Barranco, and M. R. Azghadi, "Mentorch: An open-source simulation framework for memristive deep learning systems," *Neurocomputing*, vol. 485, pp. 124–133, 2022.
- [38] A. Chen, "A comprehensive crossbar array model with solutions for line resistance and nonlinear device characteristics," *IEEE Transactions on Electron Devices*, vol. 60, pp. 1318–1326, Apr 2013.
- [39] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, and L. Bassham, "Nist special publication 800-22: A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications," *NIST Special Publication 800-22*, Apr 2010.
- [40] B. D'agostino, "An omnibus test of normality for moderate and large size samples," *Biometrika*, vol. 58, pp. 341–348, Aug 1971.
- [41] M. Stephens, "Edf statistics for goodness of fit and some comparisons," *Journal of The American Statistical Association - J AMER STATIST ASSN*, vol. 69, pp. 730–737, Sep 1974.
- [42] I. Chakravarti, R. Laha, and J. Roy, "Handbook of methods of applied statistics," *Journal of the Royal Statistical Society. Series A (General)*, vol. Volume 1, Jan 1967.
- [43] J. Singh and B. Raj, "Temperature dependent analytical modeling and simulations of nanoscale memristor," *Engineering Science and Technology, an International Journal*, vol. 21, no. 5, pp. 862–868, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2215098617314428>
- [44] C. Lammie, M. Rahimi Azghadi, and D. Ielmini, "Empirical metal-oxide rram device endurance and retention model for deep learning simulations," *Semiconductor Science and Technology*, vol. 36, 03 2021.

- [45] A. Goldsmith, *Wireless communications*. Cambridge university press, 2005.
- [46] X. Wu, V. Kumar, R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. Mclachlan, S. K. A. Ng, B. Liu, P. Yu, Z.-H. Zhou, M. Steinbach, D. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, Dec 2007.
- [47] G. GATES, "The reduced nearest neighbor rule," *IEEE Transactions on Information Theory - TIT*, May 1972.
- [48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [49] S. Balatti, S. Ambrogio, R. Carboni, V. Milo, Z. Wang, A. Calderoni, N. Ramaswamy, and D. Ielmini, "Physical unbiased generation of random numbers with coupled resistive switching devices," *IEEE Transactions on Electron Devices*, vol. 63, no. 5, pp. 2029–2035, 2016.
- [50] K. S. Woo, J. Kim, J. Han, J. M. Choi, W. Kim, and C. S. Hwang, "A high-speed true random number generator based on a cuxte1x diffusive memristor," *Advanced Intelligent Systems*, vol. 3, no. 7, p. 2100062, 2021. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202100062>
- [51] C.-Y. Huang, W. C. Shen, Y.-H. Tseng, Y.-C. King, and C.-J. Lin, "A contact-resistive random-access-memory-based true random number generator," *IEEE Electron Device Letters*, vol. 33, no. 8, pp. 1108–1110, 2012.
- [52] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. W. Strachan, M. Hu, S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 14–26, 2016.
- [53] A. Sharma, R. Singh, and P. Kajla, "Area efficient 1-bit comparator design by using hybridized full adder module based on ptl and gdi logic," *International Journal of Computer Applications*, vol. 82, pp. 5–13, Nov 2013.
- [54] R. Katreepalli and T. Haniotakis, "Power efficient synchronous counter design," *Computers & Electrical Engineering*, vol. 75, Jan 2018.
- [55] M.-S. Kim, Y. Yang, H. Koo, and H. Oh, "The demonstration of s2p (serial-to-parallel) converter with address allocation method using 28 nm cmos technology," *Applied Sciences*, vol. 11, p. 429, Jan 2021.
- [56] V. Saxena and R. J. Baker, "Compensation of cmos op-amps using split-length transistors," in *2008 51st Midwest Symposium on Circuits and Systems*, 2008, pp. 109–112.
- [57] R. Acar Vural and T. Yildirim, "Swarm intelligence based sizing methodology for cmos operational amplifier," in *12th IEEE International Symposium on Computational Intelligence and Informatics, CINTI 2011*, Nov 2011, pp. 525–528.
- [58] D. Cohen and R. Zakarevicius, "Operational amplifier integrators for the measurement of the delay times of microwave transistors," *Solid-State Circuits, IEEE Journal of*, vol. Sc-10, pp. 19 – 27, 1975.
- [59] T. Ghafouri and N. Manavizadeh, "Design and simulation of high-performance 2:1 multiplexer based on side-contacted fed," *Ain Shams Engineering Journal*, vol. 12, no. 1, pp. 709–716, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S209044792030109X>
- [60] S. Srinivas and A. B. Gudi, "Design and implementation of dual port sram memory architecture using mosfet's," in *2017 International Conference on Smart grids, Power and Advanced Control Engineering (ICSPACE)*, 2017, pp. 357–362.
- [61] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.
- [62] L. Deng, K. Sobti, and C. Chakrabarti, "Accurate models for estimating area and power of fpga implementations," in *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2008, pp. 1417–1420.
- [63] W. David, "Architectural implications of bit-level computation in communication applications," May 2014.