In-Memory Memristive Transformation Stage of Gaussian Random Number Generator

Xuening Dong[1], Amirali Amirsoleimani[2], Mostafa Rahimi Azghadi[3], and Roman Genov[1]

[1]Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada
[2]Department of Electrical Engineering and Compute Science, York University, Toronto ON M3J 1P3, Canada
[3]College of Science and Engineering, James Cook University, Queensland, Australia
Email: [1]xuening.dong@mail.utoronto.ca, [2]amirsol@yorku.ca,
[3]mostafa.rahimiazghadi@jcu.edu.au, [4]roman@eecg.utoronto.ca

***Abstract*—In this work, we present a modification to the digital Wallace-based Gaussian Random Number Generator (GRNG) by implementing an in-memory memristive dot-product engine in place of the vector-matrix multiplication (VMM) stage. The dot-product engine provides an analog interface to the GRNG with statistical robustness and better resource efficiency. One modification with three different structures is proposed and evaluated by the statistical test pass rates and benchmarked against the digital implementations. The best-proposed modification achieved a 95.8% test pass rate for 100 iterative small pool generation while requiring 23.6% and 44.4% less power and area consumption.**

***Keywords*—Memristor, Crossbar, Vector-Matrix Multiplication, Gaussian Random Number Generator**

I. Introduction

GAUSSIAN random numbers (GRNs) are widely used in fields such as Monte Carlo simulation [1], simulated annealing and white noise models [2] which a significant portion of the time is devoted in such applications for generating random numbers. Wallace method [3] has been considered to have the most considerable speedups for GRNs over conventional methods. Recent studies have been focusing on developing hardware GRNs to achieve faster throughput and better connectivity with peripheral circuits [4]. In particular, the Wallace method [3], which requires vector-matrix multiplication (VMM) and eliminates the use of hardware-expensive fundamental mathematical functions, gathers increased attention on its hardware implementation. State-of-the-art examples include [5] which implemented an FPGA-based GRNG based on Wallace, with the VMM step divided into Multiplication and Accumulation (MAC) processes. Though, the pipelining of components yields a long critical path and reduces the resource efficiency [6]. Consequently, alternative methods for implementing such operations are desirable and should be explored. One such approach may benefit from memristive in-memory computing.

Crossbars constructed using individual non-volatile memristive devices can perform the VMM operation [6] in one step [7]. By mapping the matrix onto the memristor conductance states, and inputting the vector as voltages applied to the rows of the crossbar, the resultant current sensed at the end of each column of the crossbar will reflect one element of the output vector. Memristive devices usually have a high resistance range ($1\mathrm{K}-1\mathrm{M}\Omega$) and few nanometers size, which leads to a better resource efficiency. However, a major concern with memristive crossbar VMM computation is their low accuracy [8]. Non-idealities such as sneak-path current, interconnect resistance, and ageing effect in the crossbar, and the quantization problem of the memristor itself jeopardize the precision of calculations carried on it To mitigate their impacts, additional techniques such as precision enhancement need to be applied to the crossbar.

In this paper, we propose to use in-memory computing using memristive crossbars for the VMM of the Wallace method to implement more efficient GRNGs. Generating random numbers from a gaussian distribution by utilizing true randomness of memristor crossbar switching is providing a higher quality random number generation in comparison with pseudo-random numbers algorithms typically used in computer programs. In this work, not only we are generating high quality random numbers by getting the benefit from memristor device switching randomness but also we are accelerating the hardware performance by using fully parallel VMM on memory crossbar array.

II. Preliminary

A. The Wallace Method

The Wallace method (Fig 1(a)) begins with an old pool of $N(=KL)$ GRNs generated by software method. The numbers are normalized for an average squared value that equals one and then stored in the memory. At each iteration, K numbers are randomly fetched, and organized into a vector X^T. The vector is then multiplied with an orthonormal transformation matrix A to form the new GRNs by $X'^T = AX^T$. Ideally, all the addresses in the old pool should have been covered and transformed into the new pool after each pass (defined as a complete formation of a new pool in L iterations). The new pool is then treated as an old pool in the next pass. The system ends up with a χ^2 correction step to decorrelate the numbers and reverse the normalization in the first step [9]. A random variate v can be approximated with the χ^2_N distribution by

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

978-1-6654-8356-8/22/$31.00 © 2022 IEEE

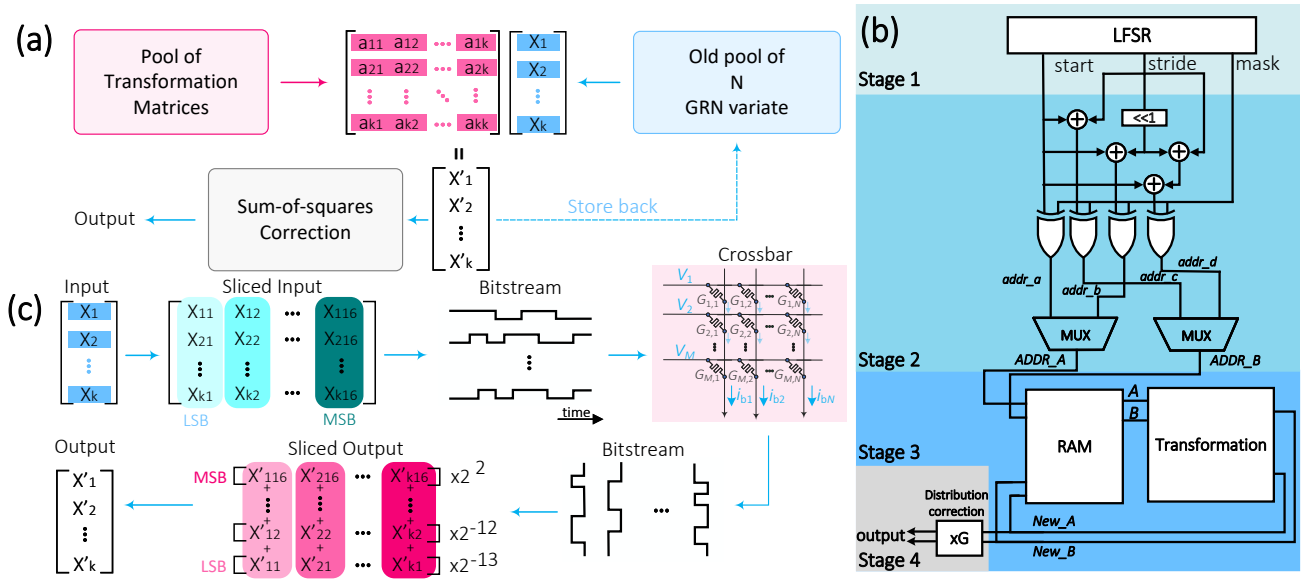

Fig. 1. (a) A graphic representation of the Wallace method. (b) The existing CMOS-based GRNG hardware implementation described in [5]. (c) The bit slicing technique used in the crossbar calculation to enhance accuracy and precision.

where x is randomly picked from the new pool. A correction factor $G = (\frac{v}{N})^{0.5}$ is then applied to each number in the pass.

B. GRNG Structure

An existing hardware GRNG in [5] is composed of two main parts: a random address generator (stages 1 and 2 in Fig. 1(b)), and a transformation-correction block (stage 3 and 4 in Fig. 1(b)). The old-pool GRNs are initially generated by software methods and stored in an on-chip dual-port RAM. The random address generator derived from conventional linear feedback shift registers (LFSRs) [10] generates bitstream that is sliced into three $log_2(L)$-bit patterns labelled as start, stride and mask. Each of the bit patterns are further logistically mixed by performing

$$addr_i = (start + stride \times i) \oplus mask \tag{2}$$

where $i \in \{1, ..., K\}$ to ensure the optimal coverage of addresses. Old GRNs are fetched from the RAM accordingly and transformed by MAC circuit. By benefiting from the only ± 1 entries in the Hadamard matrix, old GRNs can be preprocessed and operated by separate adders/subtractors in parallel. The optimal critical path then contains only one multiplexer and one subtractor. The result is stored back into the RAM per iteration and one new number from each pass is selected for approximating the distribution correction factor G, which is then applied to all the outputs.

III. Methodology

Here, we present modifications into the transformation stage (stage 3 of Fig. 1(b)) of the GRNG in [5] by replacing the logic circuits with a memristor crossbar to perform VMM.

A. Baseline Design

To enhance the calculation precision, a bit-slicing method (Fig 1(c)) is adopted in the computation. The digital numbers are sliced into 16 1-bit patterns and programmed into a bitstream to be fed to the crossbar. The output analog values are then transferred back to a $K \times 1$ output vector.

As shown in Fig 2(a), the old-pool GRNs fetched from RAM are written into an input buffer, grouped to a $K \times 1$ vector and sliced. The 16-bit input slices are converted to analog signals and fed into the rows of the crossbar. The transformation matrix entries are reflected by the resistance of memristors as shown in Fig 2(b) and pre-programmed into the crossbar in its transposed form by write-and-verify pulses in Fig 2(c). In this model, the matrix has been defined as a fixed Hadamard matrix to eliminate the need for reprogramming. The values (V) in the matrix are represented by two neighbouring columns, one for the positive part (V^+) and the other for the negative part (V^-) to form $V = V^+ - V^-$. The resultant current $I = I^+ - I^-$ is collected at the end of each two columns, converted to digital signal, accumulated in the output buffer, and written back into the memory.

B. Random Transformation Selection

To further reduce the correlation between the generated numbers, we use a pool of transformation matrices, from which one matrix is randomly selected and programmed into the crossbar per iteration. To adapt the method, extra on-chip memory is needed to store all possible matrices. During each writing, the row pulses are kept the same and the column inputs determine the final number written into the row as shown in Fig 2(b). Therefore, the matrices are stored as the bitline current values in the memory. The overall structure of the method is shown in Fig 2(d). Extra programming time of a crossbar is directly proportional to the number of rows, and the minimum switching time t_{min} of the memristor devices.

C. Multi-stage Transformation

Another enhancement on the baseline model changes the number of transformations performed during each iteration.

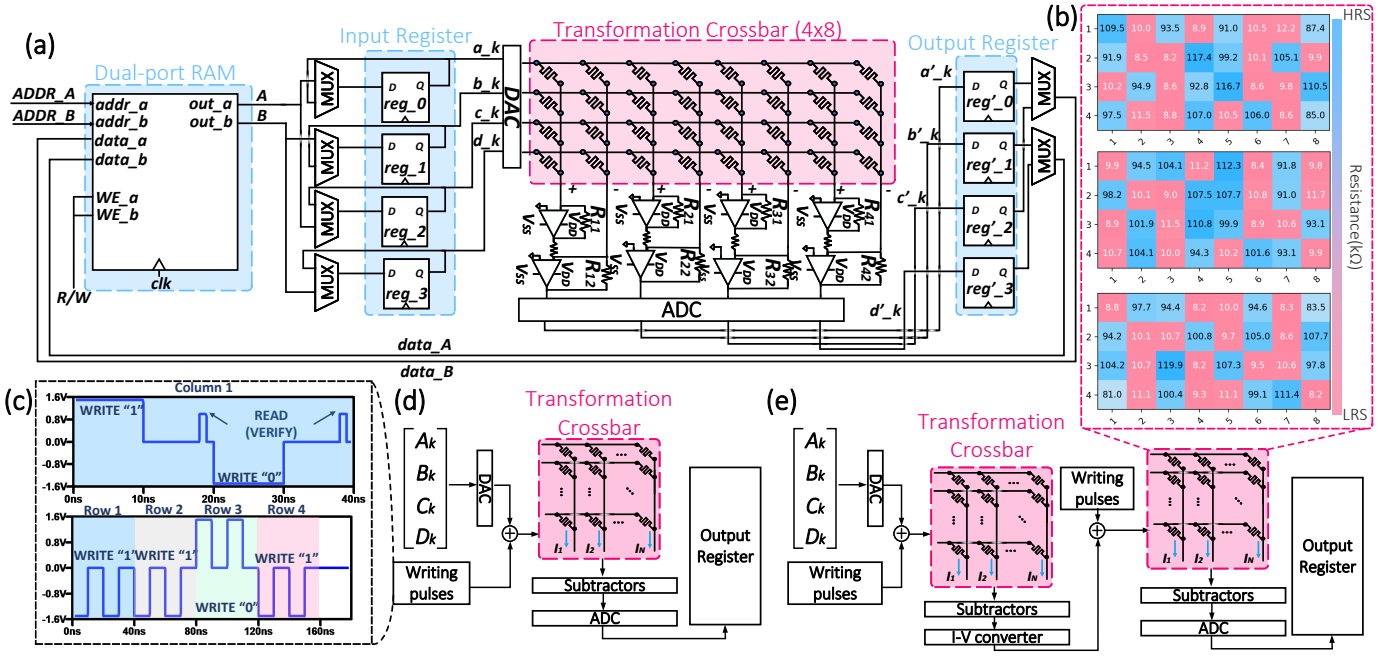

Fig. 2. (a) The proposed circuit structure for implementing the baseline model of the memristor-based transformation method. (b) Examples of memristor resistances mapped from the transformation matrices. (c) The Read-and-Verify pulses for programming e.g. [1 1 0 1] into the first column of the crossbar. (d) The first modification plan with a pool of transformation matrices. (e) The second modification plan with a pair of matrices multiplied with the input vector.

Each time, a pair of matrices are multiplied with the old-GRN vector [11]. Consequently, extra memory is needed to store all possible pairs of matrices. Similar to the previous method, the matrices are stored as bitline voltages in the memory. Another set of VMM engines, including the crossbar and the subtractor circuit, is appended at the end of the first one. The resultant current from the first crossbar is converted to voltage and passed to the input of second crossbar. The same calculation procedure is carried out again and the results are converted back to digital form. The overall structure of the method is shown in Fig 2(e). Since each crossbar bitline has its own power source, the programming of the two crossbars can be in parallel but the computations in crossbars are serial.

IV. Results and Discussion

To evaluate the proposed designs, a number of credibility tests are performed. All of the tests are carried out using the simulation platform for the memristor crossbar model built based on [12]. The device model in [13] is used by considering $\mathrm{LRS} = 10\mathrm{K}\Omega$ and $\mathrm{HRS} = 100\mathrm{K}\Omega$. The CMOS 32nm technology is used for digital blocks simulations. All calculations are on 16-bit (Q13.3) fixed-point arithmetic with 16 slices per number. All results have been summarized in Table I.

A. Goodness-of-Fit Tests

The four goodness-of-fit tests on the distribution of generated GRNs are: 1) Anderson-Darling (A-D) [17]; 2) Kolmogorov-Smirnov (K-S) [18]; 3) Shapiro-Wilk [19] and 4) D'Agostino K-squared [20], with null hypothesis stated as: the observed data follows a normal distribution. Acceptances are made on the test scores ($p > 0.05$), except for A-D test in which the statistics are compared with the critical value (stats $<$ crit. val) at each significance level. All tests are carried out with wire resistance of 20Ω and stuck probabilities of 0.1%.

The first tests are based on a relatively large pool of 4096 GRNs, with test scores compared directly. As shown in Fig 3(a), all three versions of the proposed design pass the statistical tests. The random transformation and multi-stage methods have shown at least $2.7\times$ and $2.3\times$ improvement on the test score over the baseline. Another set of tests are carried out on small pool sizes ($N = 128$ and 256) with 100 pools generated. The accumulated test pass rates are recorded in Fig 3(b). In both pool sizes, all three methods achieve a pass rate over 90%. An average increase of 3.58% and 1.69% is yielded by the latter two methods, respectively.

B. Robustness Against Non-idealities

Here we investigate the effect of three major non-idealities: sneak-path current, wire resistance between electric components, memristance variation and probabilities of stuck ON/OFF as the memristor ages. The effect of sneak-path current is considered and it is associated with the wire resistance and the state of each memristor. Therefore, it is considered with the two other factors simultaneously. In the test of wire resistance, the resistance varies in $[10\Omega, 110\Omega]$ and the pass rates of 100 pools of 128 random numbers are recorded in Fig 3(c). The memristor resistance state variation effect is considered 10% for both READ and programming stages. All pass rates decrease as the resistance increases, with the best case in the random transform method which only drops from 91.9% to 88%. The multi-stage transformation has a much steeper

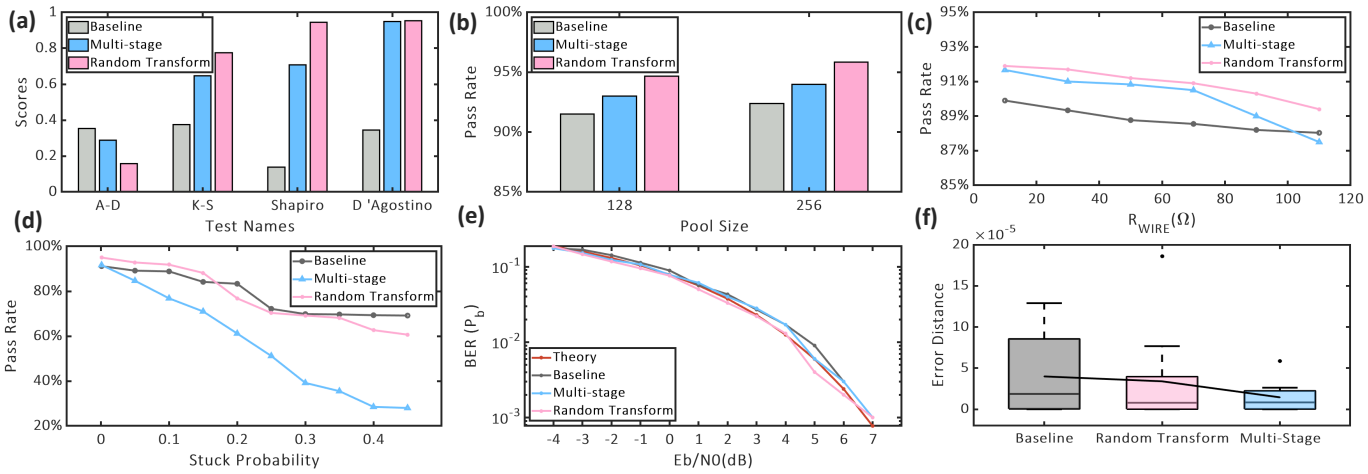

Fig. 3. (a) Large pool generation test result. (b) Small pool generation test result. (c) The impact of increasing wire resistance on the test pass rate. (d) The impact of memristor stuck ON/OFF on the test pass rate. (e) BPSK simulation result. (f) Squared error distance from the theory to the simulated BER.

TABLE I
GOODNESS-OF-FIT TEST RESULTS

Design		**Baseline Model (Section III.A)**			**Random Transform (Section III.B)**			**Multi-stage (Section III.C)**		
Tested Pool Size		128	256	4096	128	256	4096	128	256	4096
Best p-value		-	-	0.345	-	-	0.953	-	-	0.95
Test pass Rate		91.50%	92.40%	-	94.70%	95.80%	-	93%	94%	-
Test pass Rate Drop	**Wire Resistance**	2.08%			2.72%			4.55%		
	Stuck ON/OFF	24.20%			36.80%			69.20%		

TABLE II
PERFORMANCE COMPARISON OF EXISTING MEMRISTOR-BASED URNGS

Design	**[5]**	**[14]**	**[15]**	**[16]**	**This Work**		
					Baseline Model	**Random Transform**	**Multi-stage**
Method	Wallace	Box-Muller	Wallace	Physical randomness	Wallace		
Technology	FPGA (Xilinx)	FPGA (Xilinx)	FPGA (Altera)	Resistive RAM	RRAM + digital peripheral		
Power [mW]	54.78	48	560.25*	32.54*	41.83	42.98	44.23
Area [mm^2]	53.9	-	-	8.53E-06	29.97	29.973	29.975
Speed [MHz]**	155	138.8	117.63	33.3	125	35.7	29.4

* 64 units in parallel for stage 3 without standardizing the distribution, ** Random values generated per second.

descent after 70Ω and even drops below the baseline. Fig 3(d) shows tests based on the stuck probability of memristors, increased from 0.1% to 45%. Due to rapid reprogramming, both enhancement methods are more susceptible. The random transform and the multi-stage method each experienced a 36.8% and 69.2% descent, compared to the 24.2% decrease in the baseline model. The multi-stage method is the least robust due to the serial operation, which exacerbates the errors made earlier.

C. Application Test

This test uses GRNG as White Gaussian Noise sources and simulates the Bit Error Rate (BER) in the Binary Phase Shift Keying (BPSK) modulation [21]. Fig 3(e) illustrates the simulation result of BER versus the Signal-to-Noise Ratio. Simulated values show no large deviation from the theoretical values. To better visualize the differences, the squared error is calculated in Fig 3(f). All three methods have a mean error less than 5×10^{-5} with a 14.9% and a 63.7% decrease in the two enhancement methods from the baseline.

D. Overhead Analysis and Comparison

The overhead analysis in Table II are estimated on existing technology, using the Yakopcic Tantalum Oxide (TaO_x) memristor model [13] for memristor devices and 32nm CMOS technology referred from [22] in SPICE. The digital blocks implementation overhead are estimated based on power consumption equations in [23] and slice area estimated in [24]. The proposed baseline model consumes 41.83mW power per random number generated, 23.6% and 12.9% less than its digital counterpart in [5], [14] with different methods, while overhead power is required by two enhanced methods due to fetching of the new matrices and reprogramming of memristors. Another memristor-based GRNG implemented in [16] utilizing the inherent stochasticity consumed significantly less power and area but suffered from the problem of uncontrolled mean and variance of the distribution. Extra peripherals

are therefore required for standardizing the distribution. The baseline model also requires an area of $29.97\mathrm{mm}^2$, 44.4% less than the original. The area consumption increases in the latter two methods because of the extra memory and crossbar. The speed of the crossbar is considered with a fast transitioning binary Tantalum Oxide (TaO_x) memristor model in [13], and is proportional to the number of bit slices. Under the most optimistic situation where no slice is applied, the baseline can reach a maximum speed of 125MHz. The computation speed in the latter two methods drops significantly due to the reprogramming time and serial operation of crossbars.

V. Conclusion

In this work, we proposed modifications to the transformation stage of a Wallace-based hardware GRNG implementation. By exploiting the unique properties of memristive crossbars and by using statistical enhancement methods to perform VMM operations, we achieved a maximum of 95.8% in the statistical test pass rates with 23.6% less power consumption. Future implementations will be focusing on extending the memristor implementation to the other stages and improving its robustness against non-idealities.

References

[1] S. Raychaudhuri, "Introduction to monte carlo simulation," in *2008 Winter Simulation Conference*, 2008, pp. 91–100.

[2] J. S. Malik, J. N. Malik, A. Hemani, and N. D. Gohar, "An efficient hardware implementation of high quality awgn generator using box-muller method," in *2011 11th International Symposium on Communications Information Technologies (ISCIT)*, 2011, pp. 449–454.

[3] C. S. Wallace, "Fast pseudorandom generators for normal and exponential variates," *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 1, pp. 119–127, 1996.

[4] J. S. Malik and A. Hemani, "Gaussian random number generation: A survey on hardware architectures," *ACM Computing Surveys (CSUR)*, vol. 49, no. 3, pp. 1–37, 2016.

[5] D.-U. Lee, W. Luk, J. D. Villasenor, G. Zhang, and P. H. W. Leong, "A hardware gaussian noise generator using the wallace method," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 8, pp. 911–920, 2005.

[6] A. Amirsoleimani, F. Alibart, V. Yon, J. Xu, M. R. 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, no. 11, p. 2000115, 2020.

[7] M. Rahimi Azghadi, Y.-C. Chen, J. K. Eshraghian, J. Chen, C.-Y. Lin, A. Amirsoleimani, A. Mehonic, A. J. Kenyon, B. Fowler, J. C. Lee *et al.*, "Complementary metal-oxide semiconductor and memristive hardware for neuromorphic computing," *Advanced Intelligent Systems*, vol. 2, no. 5, p. 1900189, 2020.

[8] 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.

[9] R. P. Brent, "Some comments on cs wallace's random number generators," *The Computer Journal*, vol. 51, no. 5, pp. 579–584, 2008.

[10] P. P. Chu and R. E. Jones, "Design techniques of fpga based random number generator," in *Military and Aerospace Applications of Programmable Devices and Technologies Conference*, vol. 1, no. 999. Citeseer, 1999, pp. 28–30.

[11] C. Rüb, "On wallace's method for the generation of normal variates," 1998.

[12] A. Chen, "A comprehensive crossbar array model with solutions for line resistance and nonlinear device characteristics," *IEEE Transactions on Electron Devices*, vol. 60, no. 4, pp. 1318–1326, 2013.

[13] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Memristor spice model and crossbar simulation based on devices with nanosecond switching time," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1–7.

[14] J. Xu, Y. Shen, E. Chen, and V. Chen, "Bayesian neural networks for identification and classification of radio frequency transmitters using power amplifiers' nonlinearity signatures," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 457–471, 2021.

[15] R. Cai, A. Ren, N. Liu, C. Ding, L. Wang, X. Qian, M. Pedram, and Y. Wang, "Vibnn: Hardware acceleration of bayesian neural networks," *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018.

[16] A. Malhotra, S. Lu, K. Yang, and A. Sengupta, "Exploiting oxide based resistive ram variability for bayesian neural network hardware design," *IEEE Transactions on Nanotechnology*, vol. 19, pp. 328–331, 2020.

[17] M. A. Stephens, "Edf statistics for goodness of fit and some comparisons," *Journal of the American statistical Association*, vol. 69, no. 347, pp. 730–737, 1974.

[18] B. Murphy, "Handbook of methods of applied statistics," 1968.

[19] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

[20] R. B. d'Agostino, "An omnibus test of normality for moderate and large size samples," *Biometrika*, vol. 58, no. 2, pp. 341–348, 1971.

[21] A. Goldsmith, *Wireless communications*. Cambridge university press, 2005.

[22] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. 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, no. 3, pp. 14–26, 2016.

[23] 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.

[24] D. Wentzlaff, "Architectural implications of bit-level computation in communication applications," Ph.D. dissertation, Massachusetts Institute of Technology, 2002.