Performance comparison of memristor crossbar-based analog and FPGA-based digital weight-memory-less neural networks

Chinamu Kawano Masanori Hashimoto
Dept. of Informatics, Kyoto University

***Abstract*—Memristor crossbar arrays have been studied as hardware accelerators for the efficient inference and learning of neural networks, eliminating the need to load network weights from memory. Conversely, FPGAs also facilitate the implementation of weight-memory-less neural networks by embedding weights within the combinational circuit. However, to the best of our knowledge, there have been no studies that quantitatively compare the inference performance of memristor crossbar arrays and FPGAs under identical conditions. In this paper, we examine the inference performance of neural networks implemented with crossbar arrays and FPGAs, focusing on inference latency and energy per inference. Experimental results show that the crossbar-based implementation achieves slightly lower latency and significantly reduces energy consumption compared to the FPGA implementation. Notably, the energy difference ranges from 10.4 to 22.0X in our test scenarios involving small neural networks with layers possessing tens of inputs and outputs.**

***Index Terms*—neuromorphic computing, memristor crossbars, neural networks, analog circuits, FPGA**

I. Introduction

The human brain is highly efficient in terms of energy consumption, performing complex tasks with relatively low power demand. For example, the most powerful supercomputers consume orders of magnitude more power than the human brain to accomplish similar tasks. Thus, designing computer systems that mimic brain architecture could lead to highly energy-efficient computing.

Motivated by this, neuromorphic computing has drawn a significant amount of attention. Specifically, the implementation of neural networks using crossbar arrays with memristors has been intensively studied [1]–[6]. Crossbar arrays can perform power-efficient computation of matrix-vector products in the analog domain [5]. Additionally, parallelism enables fixed-time matrix-vector products regardless of matrix size, suggesting efficient neural network inference with crossbar arrays and enabling large-scale neural network implementation.

Memristor crossbars offer dual functionality—computation and memory by storing matrix values as conductance values. Spatially deploying networks on these arrays eliminates memory access for parameters and the need to program memristors, and significantly reduces power consumption. This brain-inspired spatial neural network deployment eliminates memory requirements and avoids memristor endurance issues.

On the other hand, the memory-less digital computing architecture can be achieved through specialized multipliers with one fixed operand representing individual weights. This weight-embedded combinational circuit implementations with spatial unfolding has been explored in [7], [8], using binarized weights and activations. LogicNets [9] presents another notable approach by aggressively reducing the number of neuron inputs and activation bit-widths to suit FPGA hardware. These implementations are compatible with FPGAs, as FPGAs are well-suited for small-quantity hardware implementations dedicated to special purposes. While ASICs might offer better energy efficiency, the design and fabrication costs make ASIC implementations impractical.

Then, a simple question arises: *which is more efficient, a crossbar-based analog neural network or an FPGA-based digital weight-memory-less neural network?* Both crossbar arrays and FPGAs enable the updating of network parameters after manufacturing, making them potential alternatives. To answer this question, we compare the performance of these two implementations in this work. In order to ensure a fair comparison, we endeavor to align the inference accuracy of both implementations. To our knowledge, this is the first work to quantitatively compare the performance differences between crossbar arrays and FPGAs, each targeting the *weight-memory-less* architecture. This evaluation is pivotal for the prospective spatial unfolding of the entire network.

The rest of this paper is organized as follows: Section II reviews related works. Section III explains the neural network implementations whose performance is compared. Section IV describes how the implementations are evaluated and compared. Section V presents the comparison results, accompanied by a discussion. Finally, Section VI concludes our work.

II. Related works

The memristor is a two-terminal device that was theorized in 1971 [10]. Leveraging the property of the memristor, along with Ohm's law, it is capable of both storage and computation.

Memristor crossbar-based neural network implementations are studied in [1]–[6]. R. Hasan et al. present a memristor crossbar array implementation where network parameters are trained in software [1]. The authors of [3]–[5] propose implementations in which network parameters are trained within the circuits. Such implementations are expected to be more tolerant of device variations and faults than those where parameters are trained in software [5]. Actual hardware implementations are also demonstrated, for example, in [3], [6]. In addition to matrix computation, other computations, such as the activation function, can be implemented as analog hardware. F. Kiani et al. suggest that such implementations can

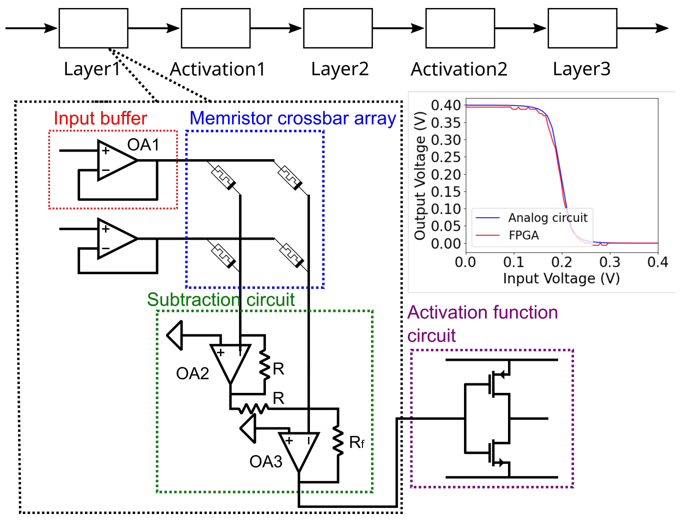

Fig. 1. Crossbar-based neural network implementation assumed in this study. The characteristics of the activation circuit are presented at the right top.

improve throughput more than implementations where peripheral circuits are implemented in digital circuits, since no A/D conversion is required. We thus consider an A/D conversion-less implementation in this work. Meanwhile, T. M. Taha et al. assert that neural networks utilizing memristor crossbar arrays outperform traditional setups like CPUs or GPUs [11]. However, conventional architectures consume a lot of power for memory access, and thus the performance superiority over memory-less digital architecture is unknown.

In the case of FPGA-based implementation, model parameters are often quantized to small bits to reduce storage and speed up computation. The binarized neural network (BNN) is an extreme version of this idea, which constrains parameters and activations to either +1 or -1 [12]. These networks can be mapped to parameter-embedded combinational circuits on FPGAs, eliminating the need to load parameters from memory [8]. However, BNNs often suffer from accuracy degradation. Y. Umuroglu et al. propose LogicNets for low-latency combinational circuit implementation, which is based on the equivalence of artificial neurons with quantized inputs/outputs and truth tables. However, the generality of LogicNets with aggressive fanin reduction has not been confirmed since only two benchmarks have been tested. Given these considerations, we have chosen to use a fixed-point representation for our FPGA implementation in the following sections.

III. Circuit structures for comparison

A. *Crossbar-based neural network*

Fig. 1 shows the diagram of the circuit that composes one neural network layer. This circuit consists of the following analog components, where the magnitudes of voltages and currents represent values such as activations: memristor crossbar arrays, input buffers, subtraction circuits, and activation function circuits. Fig. 1 also explains how a multi-layer neural network is organized. The network is composed by connecting single-layer circuits (as shown in Fig. 1) in serial and/or parallel configurations, according to the network structure.

1) Circuits: A memristor crossbar array, enclosed by the blue dotted line in Fig. 1, consists of memristors and can calculate dot-products of an input vector and a matrix. Let $\mathbf{V}$ be the vector of input voltages in the row direction, and $\mathbf{I}$ be the vector of output currents in the column direction. Then, we have $\mathbf{I} = \mathbf{GV}$ by Ohm's law and Kirchhoff's law, where $\mathbf{G}$ is the conductance matrix of the crossbar array. This enables the calculation of fully connected layers in neural networks

On the other hand, since the conductance of a memristor cannot take a negative value, a negative weight must be represented using two memristors combined with a subtraction circuit. This approach to realizing negative weights is widely used, e.g., in [2], [4], [5]. The subtraction circuit, enclosed by the green dotted line in Fig. 1, consists of two op-amps and three resistors. This circuit performs current-to-voltage conversion and subtraction, and the same circuit is used in [2], [4], [5]. R_f controls the gain of current-to-voltage conversion.

The circuit enclosed by the red box in Fig. 1 represents the input buffer used in this work due to its simplicity. This input buffer is responsible for injecting the voltage signals into the memristor crossbar array.

The circuit, enclosed by the purple dotted line in Fig. 1, represents the activation function circuit used in this work. It was adopted for its simplicity and ability to represent a sigmoid-like function. The output of this circuit is connected to the input buffer to facilitate a multi-layer structure. Fig. 1 includes the input-output characteristic of this circuit.

2) Parameter mapping: To map the parameters to the conductances of the memristors, we employed the same method as presented in [13]. In this method, the parameter matrix is decomposed into positive and negative parts, and these are mapped to the conductance of memristors representing positive and negative weights, respectively. For more details, please refer to [13].

B. *FPGA-based digital weight-memory-less neural network*

In the FPGA implementation, neural networks are spatially expanded and implemented as combinational circuits, enabling a memory access-less operation. The network parameters are embedded within the circuit, and numerous multipliers are present with fixed values for one of the two inputs, interconnected by adders. This design contrasts with ordinary implementations, where a limited number of multipliers are temporally shared and can accept arbitrary values for both inputs. An essential point to mention here is that logic synthesis tools propagate these constant values throughout the circuit, automatically leveraging them for logic minimization.

All calculations in our FPGA implementation are performed using fixed-point numbers. The activation function is implemented as a piecewise linear function, replicating the input-output relationship of the analog activation circuit, depicted in red in Fig. 1. This implementation of the activation function helps achieve similar inference accuracy, ensuring a fair comparison.

IV. Evaluation Setup

A. Network configuration

We use the MNIST dataset [14] to evaluate the performance of neural network inference, as it is widely used in related work. To reduce simulation time, the size of the input images was reduced from 28x28 to 8x8. The network has 10 outputs, and the index of the highest value is considered the inference result. We tested three different networks, whose structures are listed in Table I. All of them are composed of three fully connected layers. The PyTorch framework is used to learn the network parameters. The input-output characteristic of the activation function circuit shown in Fig. 1 is used as the activation function during training.

B. Operational amplifier design

Op-amp is a key circuit component affecting the inference performance significantly. This section explains and validates its design.

1) Baseline design: We first design a baseline op-amp that will be used in the subtraction circuits and input buffers, specifically for the NCSU 45 nm FreePDK [15]. Additional circuit tuning for OA1, OA2, and OA3 will be performed later. Fig. 3 shows the schematic. We refer partially to the design procedure outlined in [16].

We used $L = 90\,\mathrm{nm}$ for all transistors to cope with high power supply voltage and to mitigate channel length modulation. First, we determined the capacitance of C_c to attain a phase margin, such that C_c is 22% of the load capacitance of the op-amp; here, a 64×60 crossbar array is considered as the load. Then, we determined the current bias I_5 from the value of C_c, and the target slew rate is set to $1.0\,\mathrm{V/ns}$. Assuming that $V_{dd} = 0.8\,\mathrm{V}$, $V_{ss} = -0.8\,\mathrm{V}$ and the input range is $\pm 0.6\,\mathrm{V}$, the Ws of transistors M3, M4, M5 and M8 were determined to ensure they are in saturation region. Also, assuming that the required bandwidth is $5\,\mathrm{GHz}$, the size of transistors M1 and M2 was determined.

The size of transistor M7 determines how much output current can be provided by the op-amp. We observed that the subtraction circuit would fail if an insufficient amount of current flows. Therefore, we estimated the required current by simulating the circuit with some test patterns. The result shows that a current of $200\,\mu\mathrm{A}$ should be enough, and accordingly, we determined the size of M7. W of M6 is determined by $W_{M6} = 2W_{M4}\frac{W_{M7}}{W_{M5}}$.

2) Optimizing OA1, OA2 and OA3: We optimized the op-amps OA1, OA2, and OA3 in Fig. 1 with circuit simulations. To simplify the process, we simulated one layer of a crossbar array-based neural network, which includes input buffers, a memristor crossbar array, and subtraction circuits. The optimization metrics are the time for the outputs to converge (referred to as latency) and energy consumption, which is calculated as the integration of power dissipation until output convergence.

TABLE I
Network structures used for experiments.

Network configuration	# of neurons in each layer
Net1	$64 \rightarrow 60 \rightarrow 15 \rightarrow 10$
Net2	$64 \rightarrow 30 \rightarrow 30 \rightarrow 10$
Net3	$64 \rightarrow 20 \rightarrow 45 \rightarrow 10$

TABLE II
Determined op-amp parameters.

Op-amp	Parameters
OA1	$I_5 = 24.2\,\mu\mathrm{A}$, $W_{M1} = W_{M2} = 2.49\,\mu\mathrm{m}$, $W_{M3} = W_{M4} = 0.422\,\mu\mathrm{m}$, $W_{M5} = W_{M8} = 0.279\,\mu\mathrm{m}$
OA2	$I_5 = 48.4\,\mu\mathrm{A}$, $W_{M1} = W_{M2} = 1.25\,\mu\mathrm{m}$, $W_{M3} = W_{M4} = 0.845\,\mu\mathrm{m}$, $W_{M5} = W_{M8} = 0.559\,\mu\mathrm{m}$
OA3	$I_5 = 96.9\,\mu\mathrm{A}$, $W_{M1} = W_{M2} = 0.624\,\mu\mathrm{m}$, $W_{M3} = W_{M4} = 1.69\,\mu\mathrm{m}$, $W_{M5} = W_{M8} = 1.12\,\mu\mathrm{m}$

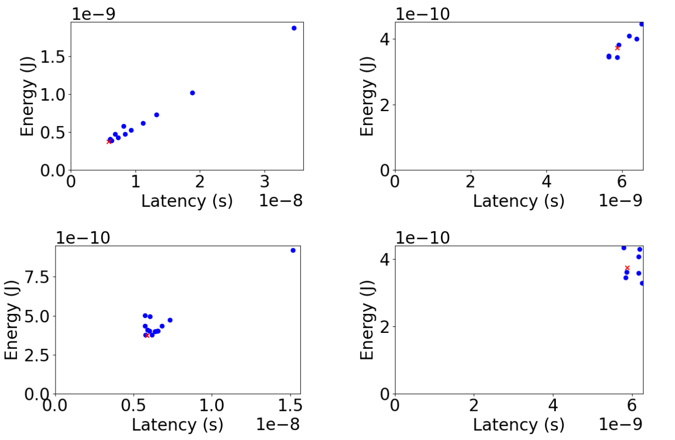

Fig. 2. Latency and energy with individual OA3 (top) and OA1 (bottom) designs. On the left, the bias current $I5$ is swept. On the right, the output current $I7$ of OA3 is swept. The red dots represent the selected designs.

First, we tuned the op-amp OA3 in subtraction circuits. We first changed the bias current I_5 while keeping the current flowing in M7 (referred to as I_7) unchanged by adjusting transistor sizes. We next fixed I_5 and changed I_7. Fig. 2 shows the results, where the red points represent the selected designs. From the left figure, 96.9 μA was determined to be the best value for I_5. While the best value was $100\,\mu\mathrm{A}$ for I_7, since the effect on latency and energy is small, we chose the value $200\,\mu\mathrm{A}$, which was derived during the initial op-amp design process. We next tuned OA1 and OA2 in the same method and determined I_5 and I_7. The determined parameters are listed in Table II. We chose $I_7 = 200\,\mu\mathrm{A}$, $W_{M6} = 6.97\,\mu\mathrm{m}$, $W_{M7} = 2.31\,\mu\mathrm{m}$, and $C_c = 24.2\,\mathrm{fF}$ for all op-amps.

Note that the optimal parameters may vary slightly depending on the crossbar size, but we use the same op-amps even when differently-sized crossbars are used.

C. Crossbar model

We employed a model for the crossbar in which the wiring is accomplished using column-direction and row-direction adjacent wiring layers, as defined in [15]. For accurate simulations, both wire resistance and capacitance are considered, as depicted in Fig. 4. The resistance was derived from the sheet resistivity provided in [15], and the wire capacitance was determined using the formula presented in [17]. The parasitic capacitance of the memristors was also taken into account, with a value of 0.14 fF, referring to the parasitic capacitance of a via-switch as reported in [18]. The chosen values are:

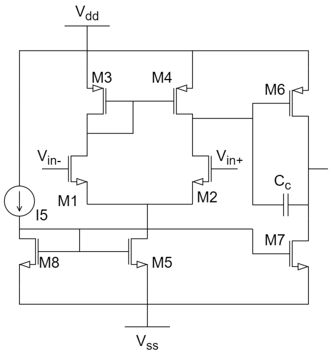

Fig. 3. Op-amp schematic.

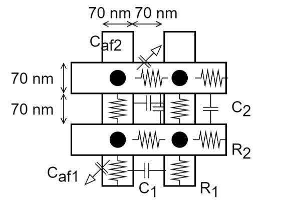

Fig. 4. Assumed crossbar model. Memristors are located at the intersections marked by black circles, sandwiched between the crossing metal lines.

$R_1 = R_2 = 0.250\,\Omega$, $C_1 = C_2 = 8.89 \times 10^{-18}\,$F, and $C_{af1} = C_{af2} = 5.59 \times 10^{-18}\,$F.

D. Performance metric evaluation

We compare the inference latency, energy per inference, and inference accuracy. The following will explain how we evaluate those metrics.

1) Memristor crossbar-based analog implementation: The inference latency, energy per inference, and inference accuracy are evaluated through SPICE simulation, using Xyce [19]. We employ the memristor model published in [20], which is also utilized in [4]. The range of memristor resistance in this work spans from $50\,\mathrm{k\Omega}$ to $10\,\mathrm{M\Omega}$.

The inference accuracy is assessed by connecting the inputs to a series of voltage sources that represent the network inputs and performing DC operating point analyses. We evaluate the accuracy using 500 images randomly selected from the MNIST test dataset.

Transient analyses are then performed using a small number of test images (10 images) to measure the inference latency and energy consumption per inference. The inference latency is defined as the time until all the outputs of the circuit converge after the input voltages are given. The convergence here is defined such that the difference between $V(t)$ and the final value of $V(t)$ is less than $10\,\%$ if the final value is greater than 0.1 V, or the difference is less than 0.01 V if the final value is smaller than 0.1 V.

2) FPGA-based digital implementation: To make the comparison fair, we design the FPGA-based implementation so that the inference accuracy would be close to that of the crossbar-based implementation. The necessary and sufficient numbers of bits in the integer and fractional portions of the fixed-point expression are experimentally derived. The target device is Zynq UltraScale+ MPSoCs (XCZU7CG-2FBVB900E). This device is chosen because of its large number of LUTs and DSPs, while it uses a more advanced technology of 14/16 nm than the 45 nm assumed for the analog implementation.

The network is implemented in Verilog as a combinational circuit. Logic synthesis, placement, and routing were executed with Vivado, aiming for the minimum delay implementation by incrementally adjusting the given delay constraint at 1-ns intervals. A post-implementation simulation is conducted to make the power report more accurate. Energy consumption is calculated from the power reported by Vivado, where the IO power is excluded, and the static power is assumed to be one-third of the dynamic power to make the comparison fair.

TABLE III
COMPARISON RESULTS. (I.F) IN FPGA IMPLEMENTATION REPRESENTS THE NUMBERS OF INTEGER AND FRACTION BITS, RESPECTIVELY. NUMBERS IN PARENTHESES IN THE ENERGY COLUMN REPRESENT THE ENERGY NORMALIZED BY THE CROSSBAR-BASED DESIGN.

Network	Implementation	Accuracy [%]	Latency [ns]	Energy [nJ]
Net1	Crossbar	86.6	27.7	3.69 (1.00)
	FPGA (6.7)	86.6	40.0	81.3 (22.0)
Net2	Crossbar	86.2	36.6	4.14 (1.00)
	FPGA (7.6)	87.6	39.0	43.1 (10.4)
Net3	Crossbar	84.6	24.6	3.01 (1.00)
	FPGA (5.6)	84.6	39.0	38.5 (12.8)

V. EXPERIMENTAL RESULTS AND DISCUSSION

A. Comparison

The inference accuracy, latency, and energy of each neural network implementation are listed in Table III. The bit widths of the integer and fractional parts used in the FPGA implementations are also shown in Table III.

In terms of the inference latency, the crossbar-based implementation was 1.07 to 1.59 times faster. The energy consumption per inference is calculated by multiplying the power dissipation by the latency. While both implementations achieved similar latencies, the FPGA-based implementation consumed 10.4 to 22.0 times more energy than the crossbar-based implementation, which means the crossbar-based implementation consumes much less power.

B. Discussion on energy in differently-sized layers

The above comparison indicates that both methods can achieve similar inference latency and that the crossbar-based implementation can surpass the FPGA-based implementation in terms of inference energy. However, it is unclear how different networks can affect the superiority of the crossbar-based implementation in inference energy. This section investigates energy differences in the case where a fully-connected layer with various numbers of inputs and outputs is implemented in both methods. With a proper pipeline structure, energy should increase proportionally to the number of layers in both implementations so that the discussion will focus on layer size.

In the crossbar-based implementation, the input buffers, subtraction circuits, and activation function circuits consume power. Supposing m and n are the numbers of inputs and outputs of the layer of interest, respectively, the energy for one-layer processing is estimated to be the sum of $m\times$(energy of input buffer) and $n\times\{$(energy of subtraction circuit) $+$ (energy of activation function circuit)$\}$. On the other hand, the latency is influenced by the parasitic capacitance and could increase in proportion to $n \cdot m$. To estimate the energy consumption for large m and n, we conducted simulations of the crossbar array varying m and n from 10 to 70 by 10 and calculated the energy consumption. The calculation of energy consumption

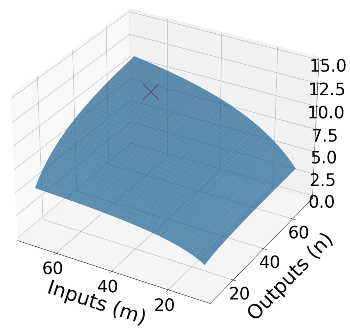

Fig. 5. Plot of $E_r(m, n)$. The red 'X' represents the point (58, 58), where E_r exceeds 10.

was done by integrating the power from the start time to the convergence time, where the integral interval corresponds to the latency definition. Note that these simulations did not include activation function circuits. Using the calculated energies, we performed fitting to $E_{xb} = a{\cdot}m{+}b{\cdot}n{+}c{\cdot}m{\cdot}n{+}d$, where a, b, c, and d are fitting coefficients. The results are $a = 4.5 \times 10^{-12}$, $b = 6.1 \times 10^{-12}$, $c = 2.2 \times 10^{-13}$, $d = -1.0 \times 10^{-11}$, with the coefficient of determination $R^2 = 0.96$, meaning that the quadratic expression of E_{xb} is appropriate in the crossbar-size range of interest.

The FPGA energy is estimated with a similar approach. We create various Verilog designs with varying (m, n) between 10 to 70 and perform synthesis, placement, and routing. The bit width is I=4 and F=4 for all designs. These designs do not include activation functions similar to the crossbar case. The clock frequency for each design is chosen such that it approaches its maximum possible value with trial and error. We then simulated each design using random inputs and estimated their energy consumption based on the switching activity data obtained from these simulations. Finally, we fitted the estimated energy consumption to $E_{FPGA} = a{\cdot}m{+}b{\cdot}n{+}c{\cdot}m{\cdot}n{+}d$. The results are $a = -2.8{\times}10^{-12}$, $b = -1.3{\times}10^{-11}$, $c = 4.3 \times 10^{-12}$, $d = 4.0 \times 10^{-11}$, with $R^2 = 1.00$.

To compare the effects of increasing m and n on the energy consumption, $E_r(m, n) = E_{fpga}(m, n)/E_{xb}(m, n)$ is illustrated on Fig. 5. The red point (58, 58) marks the point where E_r exceeds 10. We can see $E_r(m, n)$ is increased as the m and n become larger. This originates from the fact that the dependence of E_{fpga} on $m \cdot n$ is higher than that of E_{xb}. This result suggests that the crossbar-based implementation is more energy-efficient than the FPGA-based implementation when the size of a layer is large. Here, the E_r value is somewhat different from Table III because the activation function circuits are not considered in this subsection.

VI. Conclusion

We compared the performance of weight-memory-less neural network implementations using memristor crossbar arrays and FPGAs, maintaining the same inference accuracy. The crossbar-based implementation excelled in latency, being 1.07 to 1.59 times faster, and it was also more energy-efficient, consuming 10.4 to 22.0 times less energy. We found that the energy consumption of the crossbar-based implementation is less dependent on the product of the number of inputs and outputs than the FPGA-based implementation. This suggests that for larger neural network layers, the crossbar-based implementation could be more and energy-efficient. Our future work includes validating our findings on larger and various types of networks.

Acknowledgement

This work is based on results obtained from a project, JPNP21005, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).

References

[1] R. Hasan, C. Yakopcic, and T. M. Taha, "Ex-situ training of dense memristor crossbar for neuromorphic applications," in *Proc. NANOARCH*, 2015, pp. 75–81.
[2] B. R. Fernando, Y. Qi, C. Yakopcic, and T. M. Taha, "3d memristor crossbar architecture for a multicore neuromorphic system," in *Proc. IJCNN*, 2020.
[3] C. Li *et al.*, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," *Nature Communications*, vol. 9, no. 1, p. 2385, Jun 2018.
[4] R. Hasan, T. M. Taha, and C. Yakopcic, "On-chip training of memristor crossbar based multi-layer neural networks," *Microelectronics Journal*, vol. 66, pp. 31–40, 2017.
[5] ——, "On-chip training of memristor based deep neural networks," in *Proc. IJCNN*, 2017.
[6] F. Kiani, J. Yin, Z. Wang, J. J. Yang, and Q. Xia, "A fully hardware-based memristive multilayer neural network," *Science Advances*, vol. 7, no. 48, p. eabj4801, 2021.
[7] M. Rusci, L. Cavigelli, and L. Benini, "Design automation for binarized neural networks: A quantum leap opportunity?" in *Proc. ISCAS*, 2018.
[8] T. Murovič and A. Trost, "Massively parallel combinational binary neural networks for edge processing," *Elektrotehniski Vestnik/Electrotechnical Review*, vol. 86, pp. 47–53, 01 2019.
[9] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "Logicnets: Co-designed neural networks and circuits for extreme-throughput applications," in *Proc. FPL*, 2020.
[10] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.
[11] T. M. Taha, R. Hasan, and C. Yakopcic, "Memristor crossbar based multicore neuromorphic processors," in *Proc. SOCC*, 2014.
[12] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.
[13] C. Yakopcic and T. M. Taha, "Memristor crossbar based implementation of a multilayer perceptron," in *Proc. NAECON*, 2017, pp. 38–43.
[14] Y. LECUN, "The mnist database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*.
[15] N. S. University, "Freepdk45," *https://eda.ncsu.edu/freepdk/freepdk45/*.
[16] S. Vanaparthi, M. Mudavath, and P. Rangaree, "Design and implementation of 45nm operational amplifier," *Int'l Journal of Science & Eng. Development Research*, vol. 7, no. 9, pp. 452–458, Sep 2022.
[17] S.-C. Wong, G.-Y. Lee, and D.-J. Ma, "Modeling of interconnect capacitance, delay, and crosstalk in vlsi," *IEEE Transactions on Semiconductor Manufacturing*, vol. 13, no. 1, pp. 108–111, 2000.
[18] H. Ochi *et al.*, "Via-switch fpga: Highly dense mixed-grained reconfigurable architecture with overlay via-switch crossbars," *IEEE Transactions on VLSI Systems*, vol. 26, no. 12, pp. 2723–2736, 2018.
[19] Xyce, https://xyce.sandia.gov/.
[20] 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 *Proc. IJCNN*, 2013.