Automated Model-to-Transistor Design Framework for Memristor Crossbar-Based Spiking Neural Network Architectures

Hanwen Xuan, Zalfa Jouni, Mikhail Manokhin, Dimitri Galayko and Haralampos-G. Stratigopoulos
Sorbonne Université, CNRS, LIP6, Paris, France

***Abstract*—In-memory computing using memristor crossbar arrays provides an energy- and latency-efficient approach for implementing the matrix–vector multiplications required in neural network layers. In parallel, neuromorphic computing based on spiking neural networks (SNNs) offers a compelling alternative to conventional artificial neural networks (ANNs) in terms of energy efficiency and inference speed. Combining these two paradigms, namely executing SNNs on memristor crossbar arrays, presents a highly promising direction. In this work, we present a transistor-level design of a memristor crossbar array architecture for SNNs incorporating analog spiking neurons. A novel neuron circuit is developed to closely match the behavioral dynamics of neurons used in the widely adopted snnTorch framework for SNN training. The corresponding behavioral neuron model captures all key neuron parameters, enabling hardware-aware training within snnTorch. We further introduce an automated framework that generates SPICE netlists directly from trained models, including the mapping of trained weights to programmable memristor conductance values. The proposed design and framework are validated on two SNN benchmarks for ECG and MNIST classification, demonstrating close agreement between snnTorch and transistor-level inference results.**

***Index Terms*—Neuromorphic computing, spiking neural networks, in-memory computing, memristor crossbar arrays, design automation.**

I. Introduction

Computing capability has been a primary driver of progress in artificial intelligence (AI). As the demand for computational power continues to grow exponentially, there is an urgent need to explore alternative technology roadmaps for developing scalable and sustainable AI processor chips. Continued transistor scaling alone is no longer a sufficient solution, as the pace of Moore's law is slowing. A central design challenge lies in the excessive data movement between memory and processing units inherent to the conventional von Neumann architecture, which is a major source of both energy and speed inefficiency in AI hardware implementations.

In-memory computing has emerged as a promising approach to address this challenge by enabling computation directly within the memory array, rather than using memory solely for data storage. Among the proposed technologies, memristors are strong candidates for implementing synaptic weight storage. When arranged at the intersections of a crossbar array, memristors enable efficient vector-matrix multiplication: by applying voltages to the horizontal lines and sensing currents on the vertical lines, the array naturally computes $\mathbf{I} = \mathbf{G} \cdot \mathbf{V}$, where $\mathbf{G}$ represents the matrix of memristor conductances. This operation, being the dominant energy-consuming computation in neural networks, is performed in $\mathcal{O}(1)$ time complexity by directly exploiting Kirchhoff's and Ohm's laws.

Another promising approach to achieving low-energy operation is the replacement of conventional artificial neural networks (ANNs) with spiking neural networks (SNNs) [1]. SNNs emulate brain-inspired computation by processing information through discrete spikes, with the temporal information carried by the spikes playing a key role. Their energy efficiency arises from the sparsity of spike activity, the use of spike-driven computation that replaces energy-intensive multiplications with simpler addition operations, and the fact that spiking neurons remain idle unless triggered by spikes.

Bringing these two paradigms together, the implementation of SNNs on memristor crossbar arrays emerges as a particularly promising approach [2], [3]. Furthermore, due to their temporal data encoding, event-based communication across crossbars, and sparse activations, SNNs have the potential to efficiently tackle several major challenges in memristor crossbar array computation, including heat dissipation, memristor variability, digital-to-analog converter (DAC) and analog-to-digital converter (ADC) overhead, and sneak path currents [4].

In recent years, there have been several proof-of-concept demonstrations of memristor crossbar array-based SNN hardware implementations [5]–[14]. A major challenge in this context is training the SNN model while ensuring the target inference accuracy is preserved on hardware. Many studies [5], [8], [9], [12], [13] adopt spike-timing-dependent plasticity (STDP) [15] as an in-situ unsupervised learning rule. In STDP, memristor conductance is updated based on spike timing: if the presynaptic neuron fires before the postsynaptic neuron, the synapse is strengthened, whereas if it fires after, the synapse is weakened. However, STDP is difficult to scale to large, deep SNNs, and the repeated programming required can stress memristors, potentially degrading or destroying the device due to its endurance limits. Surrogate-gradient backpropagation

This work has been funded by the French National Research Agency (ANR) project CHAMELEON under Grant N^{o} ANR-22-CE94-0002-01 and by the European Union via the European Defense Fund project ARCHYTAS under Grant N^{o} 101167870. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

[16] offers a more robust training alternative by replacing the non-differentiable neuron threshold with a smooth surrogate during training. Yet, this approach is too complex for on-chip implementation. One could consider hardware-in-the-loop training, where the algorithm runs on a computer while the memristor conductances are updated iteratively based on the loss function computed on hardware. Beyond the limited endurance concern, this method is impractical for users and, in addition, chips typically need to be pre-programmed with a model for post-manufacturing testing [17], which must be completed in milliseconds, whereas training requires much longer times. Ideally, the memristor crossbar should function purely as an inference engine, with conductances programmed according to weights obtained from software [6]. To date, however, no work has demonstrated a fully automated pipeline from software training to hardware programming.

In this work, we present such a pipeline leveraging the widely used SNN training framework snnTorch [18], enabling direct weight transfer from software to a transistor-level design without any loss in accuracy. More specifically, we make the following contributions:

1) We design a fully analog SNN inference engine based on a memristor crossbar array, implemented with IHP's SG13S technology and the MEMRES module.
2) We design a novel analog spiking neuron that ensures compatibility with snnTorch.
3) We develop a design framework that automatically synthesizes the SPICE netlist of the SNN inference engine from a trained snnTorch model.
4) We train two SNNs in snnTorch on the ECG and MNIST datasets, demonstrating that model's weights-to-memristor conductances transfer preserves classification accuracy at the transistor level.
5) We conduct a corner simulation analysis of the circuit, confirming that baseline accuracy is maintained.
6) We demonstrate very low energy-per-spike operation in the order of 1nJ/spike.

The remainder of this article is organized as follows. Section II describes the SNN implementation on memristor crossbar array. Section III presents the automated design framework. Section IV reports the results of mapping SNN models onto the accelerator, and Section V concludes the article.

II. SNN Implementation on Memristor Crossbar Array

The circuit is designed in the IHP's SG13S technology and integrates memristors through the MEMRES module. Section II-A describes the architecture of the memristor crossbar array between two neuron layers. Section II-B presents the spiking neuron design and its behavioral model. Section II-C discusses the synaptic weight levels supported by the memristor's programmable conductance states, and shows how to perform the weight transfer from snnTorch to transistor-level. Finally, Section II-D shows a simulation of the neuron at transistor-level and demonstrates the correspondence between spiking activity in snnTorch and transistor-level simulations at the neuron level.

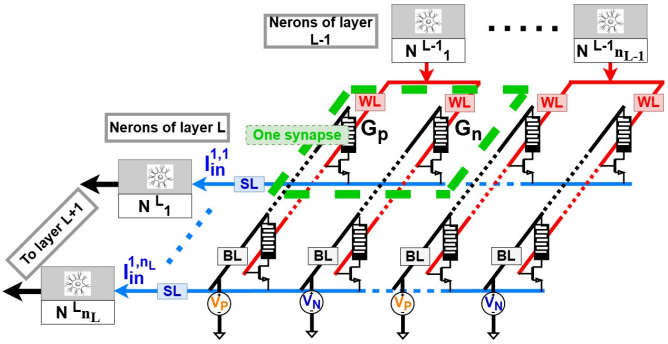


Fig. 1: Memristor-crossbar architecture for synaptic connections between neuron layers $L-1$ and L.

A. Architecture

Consider layer $L-1$ of n_{L-1} neurons, fully connected to the subsequent layer L of n_L neurons. Let N_i^{L-1} denote the i-th neuron in layer $L-1$ and N_j^L denote the j-th neuron in layer L, where $i = 1, \ldots, n_{L-1}$ and $j = 1, \ldots, n_L$.

Fig. 1 illustrates the implementation of synaptic connections between layers $L-1$ and L using a two-dimensional memristor crossbar array. Word-lines (WLs), shown in red, convey pre-synaptic spikes originating from neurons in layer $L-1$. Source-lines (SLs), shown in blue, collect the resulting synaptic currents and deliver them to the inputs of post-synaptic neurons in layer L. Bit-lines (BLs), shown in black, supply the bias voltages to the memristors.

Since a synapse can have either a positive or a negative weight, each synaptic connection is implemented as a 2T2R cell, comprising two memristors, each in series with an access transistor. Consequently, the output of a neuron in layer $L-1$ must fan out to drive two WLs, one per memristor branch. Fig. 1 highlights in green the synapse connecting neurons N_1^{L-1} and N_1^L.

At one terminal, the two memristors are connected to the BLs and are biased by V_P and V_N, respectively. These voltages are chosen such that the corresponding voltage drops satisfy $V_P - V_{ref} = V_s$ and $V_N - V_{ref} = -V_s$, where V_{ref} is the reference level fixed at the neuron input. At the other terminal, the two access transistors are driven by the output of neuron N_i^{L-1} via the WLs, and their sources are tied to the input of neuron N_j^L via the SLs.

Let $G_p^{i,j}$ and $G_n^{i,j}$ denote the conductances of the two memristors forming the synapse between neurons N_i^{L-1} and N_j^L. When neuron N_i^{L-1} fires, its spike is represented as logic 1 (logic 0 otherwise) and propagates along the WLs, turning on the access transistors. This enables the conduction path and injects into neuron N_j^L an input current $I_{in}^{i,j} = \left(G_p^{i,j} - G_n^{i,j}\right) \cdot V_s$. In this configuration, the $G_p^{i,j}$ branch provides an excitatory contribution (positive current), driving the neuron toward firing, whereas the $G_n^{i,j}$ branch supplies an inhibitory contribution (negative current), pulling the membrane potential away from threshold and reducing the neuron's

firing probability. The effective synaptic weight is therefore set by the differential conductance $G_p^{i,j} - G_n^{i,j}$, and is positive (negative) for $G_p^{i,j} > G_n^{i,j}$ ($G_p^{i,j} < G_n^{i,j}$).

For each neuron N_j^L, the SL connected to its input collects the currents contributed by all synapses activated by incoming spikes. These currents are summed along the SL, yielding the total input current I_{in}^j. Neuron N_j^L integrates the input current I_{in}^j to update its membrane potential. When this potential exceeds the firing threshold, the neuron emits a spike that is transmitted to the subsequent layer. In this architecture, each layer is realized using a distinct memristor crossbar array.

B. Spiking neuron design

Frameworks for training SNNs, such as snnTorch used in this work, operate with a global clock of period T. The simulation is divided into discrete timesteps separated by this period. At each timestep, all neurons are updated simultaneously, and no changes occur between clock ticks. Every neuron follows the same update procedure: it receives synaptic inputs from neurons in the previous layer, updates its membrane potential V_m accordingly, and checks whether V_m exceeds the firing threshold V_{th}. If the threshold is crossed, the neuron emits a spike and applies a reset, returning V_m to the resting potential V_{reset}. Additional mechanisms are often included: leakage, e.g., the membrane potential decays by a factor β at each timestep, and refractory period, T_{ref}, e.g., the neuron remains inactive and cannot fire following a spike.

We propose a novel spiking neuron design that adheres to the same clock-driven update scheme. Each hardware clock cycle corresponds to a single discrete timestep in snnTorch. We develop an equivalent behavioral model of the neuron that captures all key parameters, including T, V_m, V_{th}, V_{reset}, β, and T_{ref}, and can be seamlessly integrated into snnTorch. This approach enables hardware-aware training within snnTorch, ensuring compatibility between snnTorch and transistor-level simulations, and maintaining consistent inference accuracy across both domains.

The neuron design is fully analog, allowing direct integration into the memristor crossbar architecture shown in Fig. 1 without requiring ADCs on the SLs or DACs on the WLs. This reduces system complexity and area while improving energy efficiency. The key aspect is that, although the analog neuron operates in continuous time, the critical factor is its update behavior at each timestep which closely matches the neuron updates in snnTorch.

Fig. 2 shows the transistor-level circuit of the proposed spiking neuron. Its operation is explained using the functional diagram shown in Fig. 3.

1) Input stage: The first stage of the neuron (block A in Fig. 2) employs an op-amp to establish a fixed reference voltage V_{ref} at the input of the neuron, followed by an attenuator that scales the input current I_{in}^j by a factor K. Consequently, the current injected into the subsequent stage (block B in Fig. 2) is $I_{inj}^j = I_{in}^j/K$. The attenuator is used to regulate the current flowing into the membrane capacitor C_M of the neuron (block B), thereby enabling the use of a small capacitor, which is critical since capacitors dominate the circuit area. In particular, the low resistance of the memristor can produce large column currents that would otherwise require a much larger C_M.

2) V_m updates: The next stage (block B) is responsible for updating the membrane potential and implementing leakage. Neuron operation is governed by a clock signal *clk* with period T and duty cycle $\Delta t/T$. The clock starts low and then transitions high. Let t_k denote the k-th timestep.

During the low phase, the neuron applies the leakage with V_m^j relaxing toward V_{reset}. This leakage is implemented via a clock-controlled switched-capacitor circuit (block B) and is expressed as:

$$\beta = \frac{C_M}{C_M + C_L}. \tag{1}$$

At the end of the low phase $V_m^j(t_{k-1}+T-\Delta t) = \beta \cdot V_m^j[t_{k-1}]$.

During the high phase, the neuron performs spike integration, resulting in a change in its membrane potential. Let us denote the output of the i-th neuron in the previous layer by S_i. While snnTorch represents spikes as binary events at discrete timesteps, circuit-level simulations require continuous-time waveforms. Accordingly, a spike occurring at time t^f is modeled as a pulse spanning the interval $[t^f, t^f + T]$, such that a spike train is represented as a corresponding pulse train. Therefore, if the i-th neuron fires at timestep t^f, then $S_i(t) = 1$, $t \in [t^f, t^f + T]$, and $S_i(t) = 0$ otherwise. The change ΔV_m^j in membrane potential due to spike integration at the end of the high phase is:

$$\begin{aligned}
\Delta V_m^j &= \frac{I_{inj}^j}{C_M} \cdot \Delta t \\
&= \frac{\Delta t}{K \cdot C_M} \cdot I_{in}^j \\
&= \frac{\Delta t}{K \cdot C_M} \cdot \sum_{i=1}^{n_{L-1}} I_{in}^{i,j} \cdot S_i[t_{k-1}] \\
&= \sum_{i=1}^{n_{L-1}} \lambda \cdot \left(G_p^{i,j} - G_n^{i,j}\right) \cdot S_i[t_{k-1}],
\end{aligned} \tag{2}$$

where

$$\lambda = \frac{V_s \cdot \Delta t}{K \cdot C_M}. \tag{3}$$

Combining spike integration and leakage, at timestep t_k, the membrane potential is updated as follows:

$$V_m^j[t_k] = \beta \cdot V_m^j[t_{k-1}] + \Delta V_m^j. \tag{4}$$

3) Neuron fires: A tunable Schmitt trigger (block B) is used as a comparator to compare V_m^j with the neuron's threshold V_{th}, which is set directly through the bias voltage V_{bias1}:

$$V_{th} \propto V_{bias1}. \tag{5}$$

When $V_m^j[t_k]$ exceeds V_{th} a spike is generated at the end of the high phase using the D flip-flop (block D in Fig. 2). The output of the neuron is given by:

$$S_j(t) \;=\; \begin{cases} 1, & V_m^j[t_k] \geq V_{th} \\ 0, & V_m^j[t_k] < V_{th} \end{cases}, t \in [t_k, t_{k+1}]. \tag{6}$$

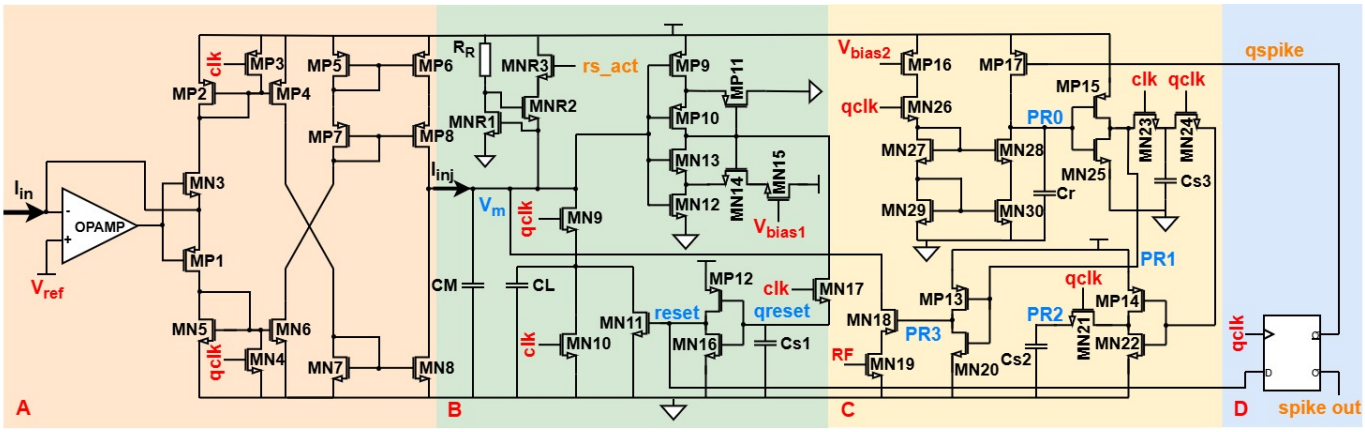


Fig. 2: Transistor-level schematic of the spiking neuron.

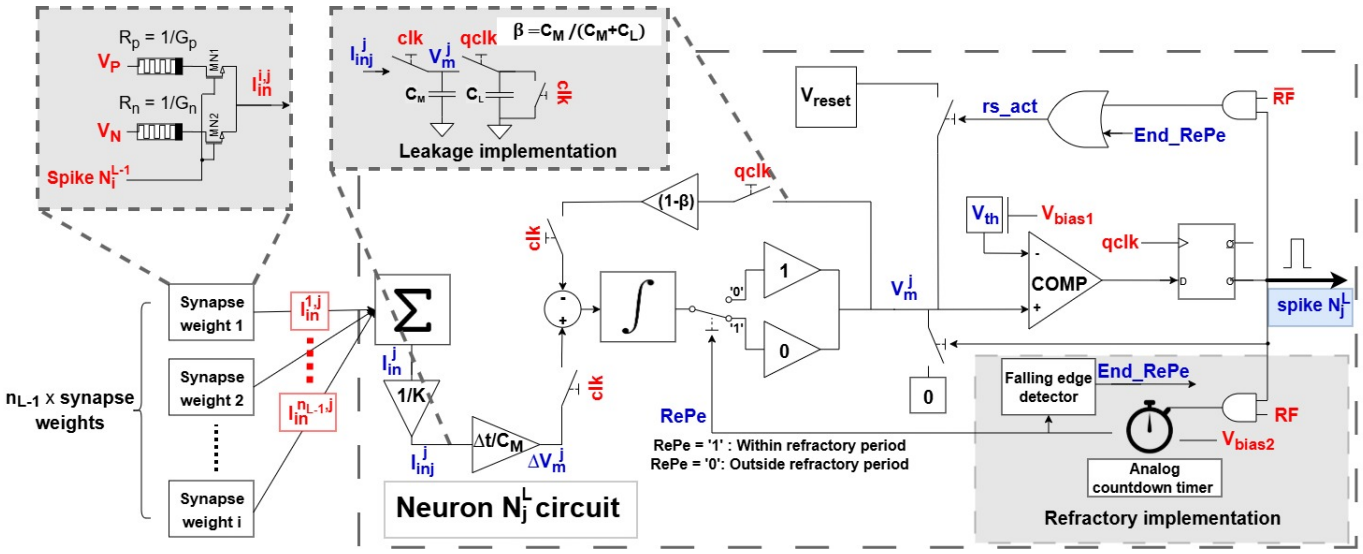


Fig. 3: Functional diagram of the spiking neuron.

4) Neuron resetting and refractoriness: If the neuron fires a spike, then V_m^j is instantaneously reset to V_{reset}:

$$V_m^j(t_k^+) = \begin{cases} V_{reset}, & S_j[t_k] = 1 \\ V_m^j[t_k], & S_j[t_k] = 0 \end{cases}. \tag{7}$$

V_{reset} is set by design and depends on the resistance R_R and the sizing of transistor MNR1 (block B). By neglecting channel-length modulation, the following relationship can be derived:

$$\left(\frac{W}{L}\right)_{\text{MNR1}} = \frac{V_{\text{DD}} - V_{reset} - V_t}{R_R\, K_n\, (V_{reset} - V_t)^2}, \tag{8}$$

where V_{DD} is the supply voltage, V_t is the threshold voltage of MNR1, and $K_n = \frac{1}{2} C_{\text{ox}} \mu_n$, with C_{ox} representing the gate-oxide capacitance per unit area and μ_n the electron mobility of the NMOS channel. For a specified target value of V_{reset}, Eq. (8) directly determines the required sizing of MNR1.

An optional refractory mechanism is realized using an analog countdown timer, which is enabled through the control signal *RF* (block C in Fig. 2). If then neuron fires at time t^f, V_m^j is clamped to zero for the duration of the refractory period T_{ref} and is reset to V_{reset} upon its completion:

$$V_m^j(t) = \begin{cases} 0, & t^f \le t < t^f + T_{ref} \\ V_{reset}, & t = t^f + T_{ref} \end{cases}. \tag{9}$$

T_{ref} is programmable via the bias voltage V_{bias2}:

$$T_{ref} \propto V_{bias2}. \tag{10}$$

C. Memristive synapses

IHP's SG13S technology, through the MEMRES module, provides a TiN/HfO$_{2-x}$/TiN resistive random-access memory (RRAM) device exhibiting memristive behavior. The RRAM cells in the crossbar array are programmed using the Incremental Step Pulse with Verify Algorithm (ISPVA) [19]. Beyond the conventional high-resistance state (HRS) and low-resistance state (LRS), the device can be reliably programmed to additional intermediate states [19]. Based on experimental characterization data [20], four stable resistance levels are achievable: 8 kΩ (LRS1), 10.5 kΩ (LRS2), 15 kΩ (LRS3), and 210 kΩ (HRS). The corresponding conductance levels are approximately $\{0.125, 0.0952, 0.0667, 0.0047\}$mS.

Therefore, using a 2T2R cell to implement a differential conductance $G_p - G_n$, a synapse in the crossbar can theoretically take 13 weight values

$\{0, \pm 0.0298, \pm 0.0583, \pm 0.1205, \pm 0.0286, \pm 0.0907, \pm 0.0621\}$ mS. However, there are closely spaced values, namely $\{0.0298, 0.0286\}$ and $\{0.0583, 0.0621\}$, which are susceptible to overlap under conductance drift, a well-known non-ideality of memristive devices. To ensure sufficient separation, these nearby values are grouped into a single level. This results in a reduced set of 9 distinct synaptic weight values: $\{0, \pm 0.0298, \pm 0.0583, \pm 0.1205, \pm 0.0907\}$mS. These 9 values constitute the *crossbar's synapse weight codebook*, denoted by $\mathcal{W} = \{w_1, \cdots, w_9\}$, representing the set of feasible synaptic weights that can be reliably programmed on the crossbar array.

In snnTorch, the arrival of a spike updates the membrane potential by adding or subtracting the absolute value of the synaptic weight that carried the spike, depending on whether the weight is positive or negative:

$$\Delta V_m^j = \sum_i w_{i,j} \cdot S_i[t_{k-1}], \tag{11}$$

where $w_{i,j}$ are the model weights. different from the physical weights implemented in the crossbar. Comparing Eqs. (2) and (11), the transfer of model weights from snnTorch to the crossbar requires that the following condition be satisfied:

$$\frac{w_{i,j}}{\lambda} = \left(G_p^{i,j} - G_n^{i,j}\right), \tag{12}$$

where λ, defined in Eq. (3), acts as a scaling factor to map the range of model weights to the available codebook values. The capacitor C_M is selected to be small, while K is chosen to reduce the current flowing through C_M. Therefore, λ can be tuned independently via V_s and Δt, or a combination of both. In turn, V_s is controlled by adjusting V_{ref}, V_P, and V_N, while keeping it lower than 0.5V to reduce the read disturb effect [21]. Once C_M is fixed, C_L is then set to determine β.

However, after training, the weights $w_{i,j}$ are represented as floating-point values, while only 9 discrete crossbar weights are available according to the codebook. For this purpose, we apply nearest-neighbor quantization to program the memristor conductances:

$$G_p^{i,j} - G_n^{i,j} = \min_{w_k \in \mathcal{W}} \left| \frac{w_{i,j}}{\lambda} - w_k \right|. \tag{13}$$

D. Neuron-level simulations

In both examples presented below, the transistor-level simulations use $V_{th} = 2.4$V (set by $V_{bias1} = 3.3$V), $\beta = 0.6$ (set by $C_M = 998$fF and $C_L = 660$fF), $V_{reset} = 1$V, $V_{ref} = 1.65$V, $V_P = 1.75$V, and $V_N = 1.55$V.

1) Analog spiking neuron: Fig. 4 illustrates a transient simulation of a single neuron over 20 clock cycles with period $T = 0.25\mu$s, corresponding to a spike integration duration of $\Delta t = 0.125\mu$s, e.g., duty cycle 50%. The refractory period is set to $T_{ref} = 1\mu$s via $V_{bias2} = 1.2$V. The neuron receives inputs from five presynaptic neurons, all generating the same spike train $S_i = \{0, 1, 1, 0, 1, 1, \cdots\}$. For all synapses, the effective conductance difference is set to $G_p - G_n = 0.0583$mS. The neuron is initialized to V_{reset} at $t = 0$. Each clock cycle consists of a leakage phase, during which the membrane potential decreases by a constant amount, followed by a spike integration phase in which the membrane potential increases linearly in response to incoming spikes. During the sixth clock cycle, the membrane potential V_m crosses the threshold V_{th}, causing the neuron to emit a spike at the beginning of the seventh clock cycle. Throughout the 1μs refractory period corresponding to four clock cycles, V_m^j is held at 0. At its conclusion, the neuron is reset and resumes spike integration.

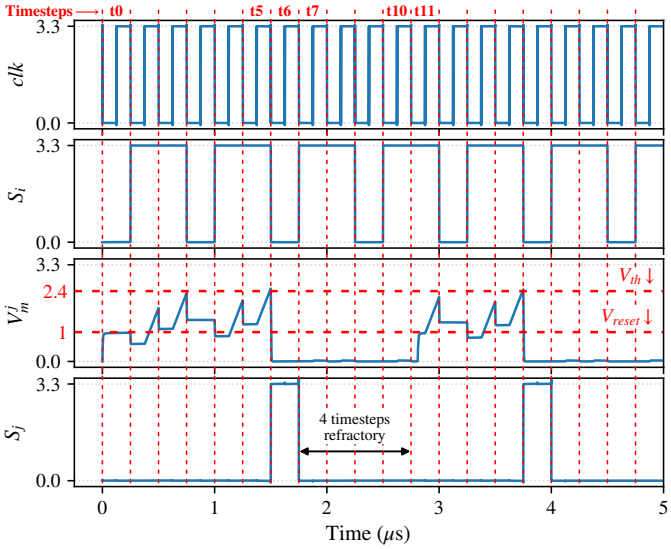


Fig. 4: Analog spiking neuron transient simulation.

2) Comparison of neuron activity in snnTorch and transistor-level simulations: Fig. 5 considers a single neuron driven by four neurons in the previous layer and compares neuron activity in snnTorch and transistor-level simulations. For the transistor-level simulation, we use a clock period of $T = 0.25\mu$s, and the experiment runs for 15 clock cycles. In snnTorch, we likewise use 15 timesteps $(t_0, \ldots, t_{14})$, although the temporal spacing between spikes is unitless. The refractory period is disabled, e.g., $V_{bias2} = 0$. The model weights are $\{1.10, 1.51, -0.71, 0.60\}$, corresponding to synapse weight codebook values of $\{0.0583, 0.0907, -0.0286, 0.0286\}$mS. The presynaptic spike trains are shown in Fig. 5a, where excitatory spikes are shown in green and inhibitory spikes in red. In the transistor-level simulation, signals are recorded in continuous time and converted to step-aligned traces by sampling the final V_m^j at the end of each clock cycle, enabling a direct comparison of V_m^j between the model and transistor-level simulations, as shown in Fig. 5b. As can be seen, the snnTorch and transistor-level simulations exhibit very similar membrane potential updates. Fig. 5c further shows perfect alignment of the output spikes in the two simulations.

III. Automated Design Framework

We propose a design automation framework that generates the SPICE netlist of the memristor crossbar array-based SNN accelerator directly from a model trained in snnTorch. The design flow, illustrated in Fig. 6, consists of the following steps:

1) Transferable neuron parameters: In Section II-B, we showed that the transistor-level neuron design has a behavioral

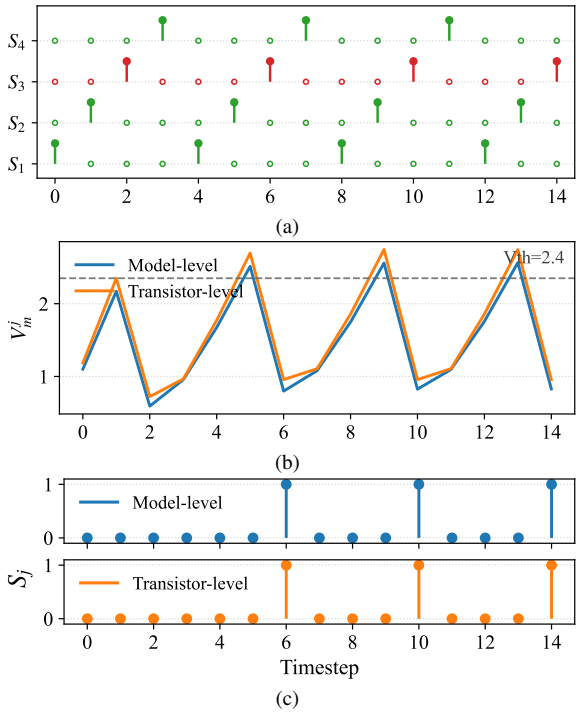


Fig. 5: Comparison of neuron spiking activity in snnTorch and at transistor-level: (a) input spike trains, (b) membrane potential, and (c) output spikes.

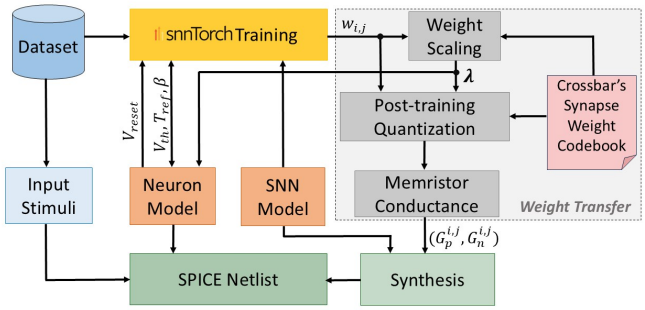


Fig. 6: Model–to-transistor automated design framework.

model consistent with that used in snnTorch with the neuron parameters, e.g., threshold V_{th}, leakage β, and refractory period T_{ref}, being programmable (see Eqs. (5), (1), and (10)), while V_{reset} is fixed by design (see Eq, (8)). This correspondence enables a direct transfer of neuron model parameters between snnTorch and the transistor-level implementation, and ensures that neuron activity matches across snnTorch and transistor-level simulations, as confirmed in Section II-D2.

2) SNN model training: Given a dataset and an SNN model, specified by the number of layers and neurons per layer, two training strategies can be adopted: (a) all neuron parameters are determined by the transistor-level design and used as fixed hyperparameters in snnTorch, with the synaptic weights being the only learnable parameters; (b) all neuron parameters (except V_{reset}) are treated as learnable hyperparameters in snnTorch. In this case, once training is completed, the circuit parameters V_{bias1}, C_L, and V_{bias2} are adjusted to realize the learned values of V_{th}, β, and T_{ref}, respectively.

3) Weight transfer: After training, the learned weights are mapped to memristor conductance levels using the procedure described in Section II-C, which includes scaling and quantization (see Eq. (13)).

4) Design synthesis: The transferability of neuron parameters and synaptic weights enables the direct mapping of a trained snnTorch model onto a memristor crossbar array-based SNN accelerator. This mapping is performed by a synthesis script that takes as inputs the configured hardware neuron model, the SNN architecture, and the memristor conductances, and automatically generates the SPICE netlist of the complete SNN accelerator, with one crossbar instantiated per layer. The script determines the crossbar dimensions based on the number of neurons in the connected layers and assigns memristor conductance values for each layer according to the corresponding layer's weights. The crossbars are then interconnected with the neuron circuits.

5) Input stimuli for inference at transistor-level: The datasets are provided or are encoded in spike format. Therefore, the inputs to the SNN model consist of spike trains. Each spike train is converted by a script into a corresponding pulse train, which serves as the input stimulus for transistor-level simulations (see Eq. (6) and Fig. 4).

6) Inference at transistor-level: Transistor-level inference requires a number of clock cycles equal to the number of timesteps used in snnTorch. Although the neuron operates in continuous time, the key observation is that at the discrete timesteps of snnTorch and at the end of high clock phases in the transistor-level implementation, the neuron models in snnTorch and at the circuit level coincide, yielding similar membrane potential updates and spiking activity, as shown in Fig. 5. As a result, consistent SNN inference accuracy is achieved across snnTorch and transistor-level simulations.

IV. Case Studies

We validate the transistor-level design and the automation flow using two SNN case studies: ECG classification on the MIT-BIH Arrhythmia Database and digit classification on the MNIST dataset. In addition, we perform corner simulations and we report the energy-per-spike. All transistor-level simulations are performed in Cadence. In snnTorch, the SNNs are trained with back-propagation using the Adam optimizer with a learning rate of 1.5×10^{-3} for ECG and 10^{-3} for MNIST. For both SNNs, we use $V_{th} = 2.4$V, $V_{reset} = 1$V, and $\beta = 0.6$, while the refractory period is disabled.

A. ECG Classification

The MIT-BIH Arrhythmia Database contains 448 half-hour ambulatory ECG recordings from 47 subjects [22]. Each record provides two leads sampled at 360 Hz. In this work, we use the MLII lead and exclude paced-beat records 102, 104, 107, and 217 [23]. Beats are labeled using the AAMI grouping [24] into four classes: N (normal), S (supraventricular ectopic), V (ventricular ectopic), and F (fusion). The Q (unclassifiable) class is omitted in this work. Beats from all recordings are

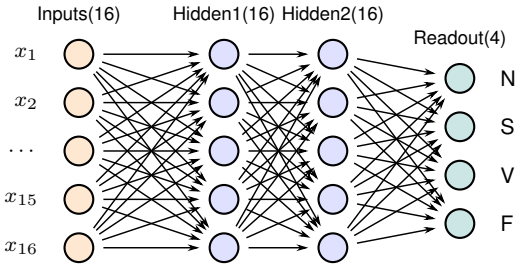


Fig. 7: SNN architecture for ECG classification.

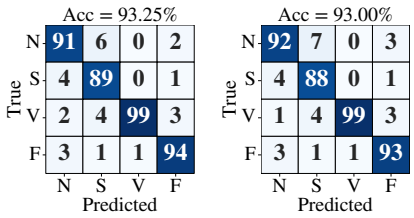


Fig. 8: Confusion matrices for ECG classification obtained from (a) post-quantized model inference and (b) transistor-level inference.

pooled and split into training and test sets (80%/20%) using an intra-patient split. The strong class imbalance (approximately 89.5% N, 2.8% S, 7% V, 0.8% F) is handled using random oversampling during training. ECG recordings are first segmented into individual beats by extracting a fixed window around each annotated R-peak (250ms pre-R and 450ms post-R). We apply z-score normalization to make beat segments comparable across recordings, setting each segment to zero mean and unit standard deviation. We then apply principal component analysis (PCA) to reduce each beat to a 16-dimensional feature vector. The resulting 16 coefficients form the input vector to the SNN.

We use the fully-connected SNN, illustrated in Fig. 7. For transistor-level inference, we use the 16-channel spike trains produced by the first hidden layer as input stimuli.

Fig. 8 compares the confusion matrices from post-quantized model inference and transistor-level inference on the same balanced test set with 100 beats per class. The overall accuracies are close, reaching 93.25% for the post-quantized model and 93% at the transistor level. The mismatches appear when the two strongest output classes produce nearly identical spike counts. This behavior arises because the membrane potential updates at the transistor level and in snnTorch are only approximately equivalent.

B. MNIST classification

MNIST is a handwritten digit dataset of 28×28 grayscale images labeled from 0 to 9 [25]. It provides a fixed split of 60 000 training images and 10 000 test images. Each image is converted to a 784-dimensional vector by flattening the 28×28 pixels, then scaled to $[0, 1]$ by dividing the pixel values by 255. PCA reduces each processed image to a 64-dimensional feature vector that preserves the main variations in the data. The 64 features form the SNN input.

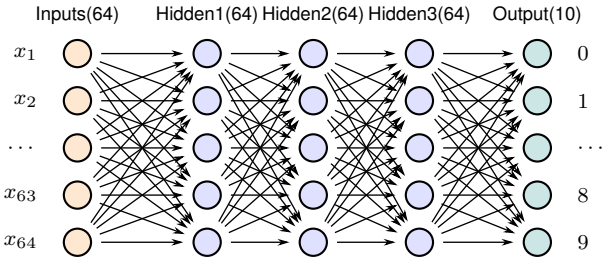


Fig. 9: SNN architecture for MNIST classification.

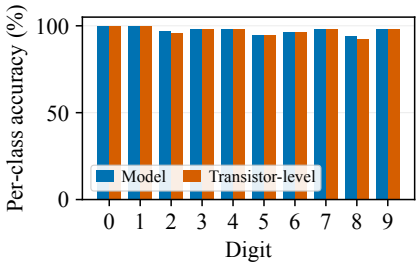


Fig. 10: Per-digit accuracy on MNIST for post-quantized model inference and transistor-level inference.

Fig. 9 shows the SNN architecture. For transistor-level inference, we implement the last two hidden layers and the output layer at transistor-level, using the 64-channel spike trains produced by the first hidden layer as input stimuli.

Fig. 10 compares per-digit accuracy from post-quantized model inference and transistor-level inference on a balanced MNIST subset of 1000 test samples (i.e., 100 samples per digit). Per-digit accuracies are nearly identical across the two inference modes, supporting consistent model-to-transistor-level translation within the proposed design flow. The overall accuracy is 97.46% for the post-quantized model and 96.78% at the transistor level.

C. Inference Time

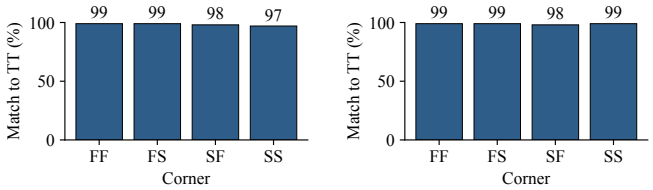
The inference time for one input sample is given by $T_{inf} = (N + (L-1)) \cdot T$, where $N + (L-1)$ is the number of clock cycles, N is the number of timesteps in snnTorch, L is the number of layers, and T is the clock period. The clock frequency in the transistor-level simulations is 1MHz for the ECG and 4MHz for the MNIST. Therefore, for the ECG and MNIST, the inference time is 26μs and 6.75μs, respectively.

D. Inference Accuracy Under Process Variations

We evaluate robustness to device non-idealities by performing transistor-level inference across standard process corners, including combinations of fast (F) and slow (S) NMOS and PMOS devices, denoted as FF, FS, SF, and SS. TT represents the nominal, typical design. As shown in Fig. 11, the results exhibit a high match rate relative to TT across all corners for both tasks, indicating that the accelerator maintains stable predictions under process variations.

E. Energy Efficiency

We evaluate static power consumption with all neurons inactive and dynamic power consumption during inference. Static



(a) ECG (b) MNIST

Fig. 11: Inference accuracy under process variations.

TABLE I: Power and energy-per-spike during ECG inference.

	Power (mW)		Energy-per-Spike (nJ)
Class	Total	Dynamic	Dynamic
F	7.81	5.81	1.04
S	6.83	4.83	1.07
V	7.96	5.96	1.11
N	7.24	5.24	1.20
Avg.	7.46	5.46	1.11

power scales with network size and is primarily dominated by the operational amplifiers, whereas dynamic power depends on spike sparsity and therefore varies across datasets and, in some cases, across classes within a dataset. A commonly used metric in SNNs is the energy-per-spike, defined as the total energy consumption over the inference time divided by the number of spikes generated during inference. Tables I and II report these metrics for the two case studies. The static power consumption is 2mW for the ECG network and 23.9mW for MNIST. For the ECG dataset, the power and energy-per-spike vary across classes, while for MNIST they remain nearly constant across digits. Overall, the average dynamic energy-per-spike is 1.11 nJ/spike for ECG and 0.405 nJ/spike for MNIST, demonstrating the very low power consumption of the proposed accelerator.

V. Conclusion

We presented an SNN hardware accelerator based on memristor crossbar arrays, along with an automated design framework that enables seamless mapping of SNN models trained in snnTorch onto the accelerator while preserving snnTorch-level inference accuracy at the transistor level. Furthermore, the inference accuracy is robust to process variations, and the inference power consumption is on the order of 1nJ/spike.

References

[1] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nat. Comput. Sci.*, vol. 2, no. 1, pp. 10–19, Jan. 2022.

[2] L. A. Camuñas-Mesa, B. Linares-Barranco, and T. Serrano-Gotarredona, "Spiking neural networks and their memristor-CMOS hardware implementations," *Materials*, vol. 12, no. 17, Aug. 2019, Article 2745.

[3] A. Mehonic, A. Sebastian, B. Rajendran, O. Simeone, E. Vasilaki, and A. J. Kenyon, "Memristors—from in-memory computing, deep learning acceleration, and spiking neural networks to the future of neuromorphic and bio-inspired computing," *Adv. Intell. Syst.*, vol. 2, no. 11, Aug. 2020, Art. no. 2000085.

[4] K. Eshraghian, J, X. Wang, and W. D. Lu, "Memristor-based binarized spiking neural networks: Challenges and applications," *IEEE Nanotechnol. Mag.*, vol. 16, no. 2, pp. 14–23, Jan. 2022.

[5] S. Kim *et al.*, "NVM neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, Dec. 2015, pp. 17.1.1–17.1.4.

TABLE II: Power and energy-per-spike during MNIST inference.

	Power (mW)		Energy-per-Spike (nJ)
	Total	Dynamic	Dynamic
Avg.	64.2	34.9	0.405

[6] A. Regev *et al.*, "Fully-integrated spiking neural network using SiOx-based RRAM as synaptic device," in *Proc. IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 145–148.

[7] S. R. Nandakumar, I. Boybat, M. L. Gallo, E. Eleftheriou, A. Sebastian, and B. Rajendran, "Experimental demonstration of supervised learning in spiking neural networks with phase-change memory synapses," *Sci. Rep.*, vol. 10, no. 1, May 2020.

[8] C. Mohan, L. A. Camuñas-Mesa, J. M. De La Rosa, E. Vianello, T. Serrano-Gotarredona, and B. Linares-Barranco, "Neuromorphic low-power inference on memristive crossbars with on-chip offset calibration," *IEEE Access*, vol. 9, pp. 38043–38061, Mar. 2021.

[9] L. A. Camuñas-Mesa, E. Vianello, C. Reita, T. Serrano-Gotarredona, and B. Linares-Barranco, "A CMOL-like memristor-CMOS neuromorphic chip-core demonstrating stochastic binary STDP," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 12, no. 4, pp. 898–912, Dec. 2022.

[10] F. Moro *et al.*, "Hardware calibrated learning to compensate heterogeneity in analog RRAM-based spiking neural networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2022, pp. 380–383.

[11] F. Nowshin, H. An, and Y. Yi, "Towards energy-efficient spiking neural networks: A robust hybrid CMOS-memristive accelerator," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 20, no. 1, pp. 5:1–5:20, Jan. 2024.

[12] S. K. Vohra, S. A. Thomas, M. Sakare, and D. M. Das, "Circuit implementation of on-chip trainable spiking neural network using CMOS based memristive STDP synapses and LIF neurons," *Integration*, vol. 95, Mar. 2024, Art. no. 102122.

[13] N. Garg *et al.*, "Versatile CMOS analog LIF neuron for memristor-integrated neuromorphic circuits," in *Proc. Int. Conf. Neuromorphic Syst. (ICONS)*, Jul. 2024, pp. 185–192.

[14] Z. Wang *et al.*, "Fully integrated memristive spiking neural network with analog neurons for high-speed event-based data processing," *arXiv:2509.04960*, 2025.

[15] H. Markram, W. Gerstner, and P. J. Sjöström, "Spike-timing-dependent plasticity: A comprehensive overview," *Front. Synaptic Neurosci.*, vol. 4, Jul. 2012.

[16] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 51–63, Nov. 2019.

[17] S. Raptis and H.-G. Stratigopoulos, "Minimum time maximum fault coverage testing of spiking neural networks," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar./Apr. 2025.

[18] J. K. Eshraghian *et al.*, "Training spiking neural networks using lessons from deep learning," *Proc. IEEE*, vol. 111, no. 9, pp. 1016–1054, Sep. 2023.

[19] E. Pérez, C. Zambelli, M. K. Mahadevaiah, P. Olivo, and C. Wenger, "Toward reliable multi-level operation in RRAM arrays: Improving post-algorithm stability and assessing endurance/data retention," *IEEE J. Electron Devices Soc.*, vol. 7, pp. 740–747, Jul. 2019.

[20] M. Uhlmann *et al.*, "LUT-based RRAM model for neural accelerator circuit simulation," in *Proc. IEEE Int. Symp. Nanoscale Architectures (NANOARCH)*, Dec. 2023.

[21] J. Wen, A. Baroni, E. Perez, M. Ulbricht, C. Wenger, and M. Krstic, "Evaluating read disturb effect on RRAM based AI accelerator with multilevel states and input voltages," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2022.

[22] G.B. Moody and R.G. Mark, "The impact of the MIT-BIH Arrhythmia Database," *IEEE Eng. Med. Biol. Mag.*, vol. 20, no. 3, pp. 45–50, May/Jun. 2001.

[23] C. Hansol, J. Park, J. Lee, and D. Sim, "Review on spiking neural network-based ECG classification methods for low-power environments," *Biomed. Eng. Lett.*, vol. 14, no. 5, pp. 917–941, Jun. 2024.

[24] "Testing and reporting performance results of cardiac rhythm and st-segment measurement algorithms," 1999.

[25] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist/, Online.