OpenSpike: An OpenRAM SNN Accelerator

Farhad Modaresi
Dept. of Electrical Engineering
Allameh Mohaddes Nouri University
Nur, Mazandaran, Iran
f.modaresi@mohaddes.ac.ir

Matthew Guthaus
Dept. of Computer Science and Engineering, UC Santa Cruz
Santa Cruz, CA, United States
mrg@ucsc.edu

Jason K. Eshraghian
Dept. of Electrical and Computer Engineering, UC Santa Cruz
Santa Cruz, CA, United States
jeshragh@ucsc.edu

Abstract*—This paper presents a spiking neural network (SNN) accelerator made using fully open-source EDA tools, process design kit (PDK), and memory macros synthesized using OpenRAM. The chip is taped out in the 130 nm SkyWater process and integrates over 1 million synaptic weights, and offers a reprogrammable architecture. It operates at a clock speed of 40 MHz, a supply of 1.8 V, uses a PicoRV32 core for control, and occupies an area of 33.3 mm^2. The throughput of the accelerator is 48,262 images per second with a wallclock time of 20.72 μs, at 56.8 GOPS/W. The spiking neurons use hysteresis to provide an adaptive threshold (i.e., a Schmitt trigger) which can reduce state instability. This results in high performing SNNs across a range of benchmarks that remain competitive with state-of-the-art, full precision SNNs. The design is open sourced and available online: *https://github.com/sfmth/OpenSpike

***Index Terms*—ASIC, accelerator, open source, OpenRAM, spiking neural network**

I. Introduction

The open source community has enabled advances in deep learning (DL) to proliferate over the past decade. Much of these advances have successfully been ported to biologically plausible spiking neural networks (SNNs). SNNs have been used to model brain function, and can perform DL using gradient-based or Hebbian learning rules, often on a CUDA backend [1]–[4]. To achieve this, the training methods that have been popularized by DL (e.g., error backpropagation) have also been adopted to training SNNs to much success [4], [5]. The benefits of SNNs are limited when ported onto CUDA-accelerated backends as the underlying instruction-set is constrained to SIMD/SIMT (single-instruction multiple-data/thread) processing.

Demonstrating the value of SNNs goes beyond measuring test set accuracy on toy problems. Evaluating the performance of SNNs extends to its efficiency, often in terms of energy, latency, or synaptic operations per second. Hardware such as Loihi and SpiNNaker offer power profiling tools to estimate the overhead of SNN workloads, and programming packages such as SpikingKeras can provide coarse estimates based on the number of operations in a model [6]–[8]. These tools have lowered the barrier to access neuromorphic algorithm development and have provided promising empirical demonstrations of spike-based computing.

Neuromorphic research is still highly exploratory, and despite the various engineering benefits that have been demonstrated, the research community remain uncertain with what feature sets, neuron models, and neural computations should be integrated on-chip. On the one hand, why should we implement features in silicon that are not commonly used? On the other hand, how do we know what benefits such features have to offer without hardware baselines?

The recent expansion of open source electronic design automation (EDA) and VLSI tooling is primed to do for neuromorphic chip design what has already been done for DL [9], [10]. The explosion of new DL methods as applied to SNNs only keeps coming [11]–[19], and lowering the barrier to develop ASICs can ultimately profile these workloads in a way that can demonstrate or dispute energy and latency advantages [20]–[22].

To guide the direction of neuromorphic research along an open and reproducible trajectory, this paper presents a fully open source SNN accelerator, including the process design kit (PDK), the tooling used to synthesize the design, and the memory macros used to store synaptic weights. Our design was done in the SkyWater 130nm (SKY130) process, cleared all pre-tape-out checks, and includes 1,059,840 synaptic weights, and operates at a clock speed of 40 MHz. Small-scale SNNs (i.e., on the order of 1,000-10,000 neurons) tend to be memory-limited in their performance, so we have used OpenRAM macros in the design to promote further optimization of near-memory compute tooling.

II. Neuron Model

The most common neuron model used in SNNs trained via gradient descent is the leaky integrate-and-fire neuron model [23], [24]. It offers a reasonable approximation of neurons while retaining simplicity. A discrete-time version can be represented by the following equation:

$$u_t = \beta u_{t-1} + x_{t-1} w - z_{t-1} u_{\text{thr}}, \tag{1}$$

where u is the membrane potential of the neuron, x is the input to the neuron, w is the weight attached to x, β is the decay rate of the membrane potential, and the subscript t refers to time. When the membrane potential exceeds the threshold u_{thr}, an output spike is generated:

$$z_t = \begin{cases} 1, & \text{if } u_t > u_{\text{thr}} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Introducing a Schmitt trigger to the thresholding action of the neuron has the following effect:

$$z_t = \begin{cases} 1, & \text{if } u_t > u_{\text{thr}}^h \text{ and } i_t = 0 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

where i_t refers to spike inhibition:

$$i_t = \begin{cases} 0, & \text{if } z_t = 0 \text{ and } u_t < u_{\text{thr}}^l \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

As shown in Refs. [15], [25], [26], SNNs are highly tolerant to weight quantization. In the extreme, the trainable weights of an SNN can be binarized to $w \in \{-1, +1\}$ with small performance degradation as performance is 'propped up' by non-binarized variables, including membrane potential and time.

III. Architecture of the Accelerator

We propose a time-multiplexed accelerator designed for the SKY130 process with an open-source PDK, synthesized and hardened using fully open-source EDA tools. The accelerator core is illustrated in Figure 1. When carrying out neuromorphic computations, the memory complexity of DL and SNN workloads scales with $\mathcal{O}(n^2)$ where n is the number of neurons, which is dominated by the network parameters. Fully-integrated accelerators on a monolithic substrate tend to be bottelenecked by weight access.

To address this in our OpenSpike core, spiking neuron modules are re-used and time-multiplexed by parallelizing neuronal computations with the loading of weights, along with time-multiplexing inter-neuron computations. Neuron re-use optimizes for area with a marginal impact on latency. This is possible due to the parallelization of weight access and neuronal computations (i.e., state update, and spike triggering). Furthermore, using binarized weights balances the cost of weight reads from memory with neuron computations.

A. Network Architecture

To implement a neural network, our accelerator uses 1,024 hardware neurons to process the network one layer at a time. For example, in an SNN with an architecture of 1024-1024-10 dense connections across three layers, a total of 1,024 neurons would be needed on chip. A drawback to this approach is the need to save and load neuron data for each layer, but this can be performed in parallel with weight read-out with a marginal impact on performance.

B. Neuron Processor

Each of the 1,024 neurons consists of a multiply-accumulate (MAC) unit and a potential adder. The MAC unit sums the product of inputs x with weights w over time, and stores the result in a register. The final result is then passed to the potential adder which accumulates present time membrane potential u_t with the previous time decayed potential βu_{t-1}.

While the MAC unit is computing the incoming potential for the current layer, the potential adder holds the accumulated potential from the previous layer. The membrane potential

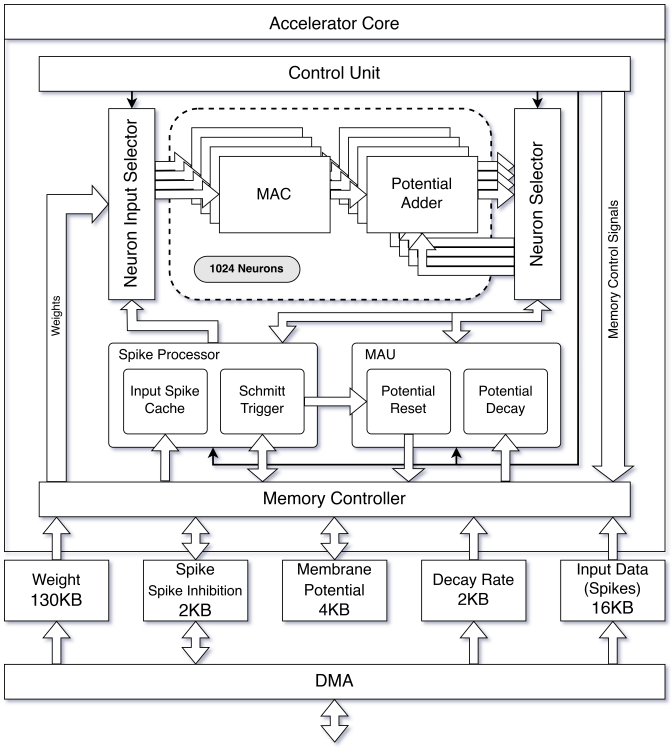

Fig. 1: System architecture: Data flow between different modules of the OpenSpike core.

arithmetic unit (MAU) computes the decayed potential, and then loads the result in the potential adder. As decayed potentials are loaded to the potential adder, the result is used to calculate the spike responses which are saved and re-read at a slower rate for MAC units, as they need the previous layer's spikes. Upon completion of reading decayed potentials, the new membrane potential values for each neuron are written as a neuron selector addresses into necessary neurons. The reset status is determined by the spike response from the Schmitt trigger module. This response determines whether a spike has occured, which determines whether the membrane potential has to be reset prior to saving or not.

Each MAC unit can accumulate up to four incoming connections at once. This process is repeated for all fan-in spikes.

C. Control Unit

The control unit is a state machine that has three main states:

1) Input Layer State: The input layer state consumes only 3 clock cycles, as the input layer neurons each have only 1 input. It only takes one accumulation cycle for MAC units to calculate the accumulated potential, as the input spike cache is already loaded in the output layer state. Saving results from the output layer also takes one cycle since it only has 10 output neurons and the accelerator core is designed to save up to 16 neurons in each cycle. Initialization of the state in the control unit takes one cycle to complete.

2) Hidden Layer State: In this state, the MAC units iterate through 1,024 input connections for each neuron 4 connections at a time, such that 256 cycles are required to compute the

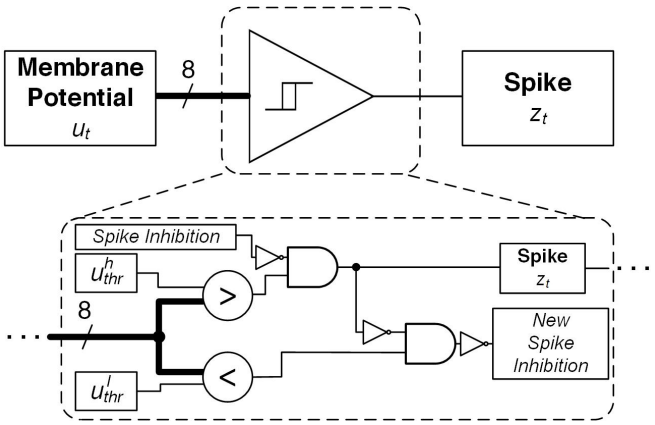

Fig. 2: Spike emission process.

potential values. While MAC units are processing, the neuron selector activates the potential adders 16 at a time, and it therefore takes 64 cycles to finish processing and storing the hidden layer potentials.

3) Output Layer State: In this state, only 10 MAC units are needed to compute over 256 cycles. At the same time, the input spike cache is loaded from the input spike SRAM in 8 cycles. Additionally, the potential adders will complete generating results from the hidden layer. This completes the operations needed for a single time step.

D. Spike Processor

The spike processor has two functions: i) spike emission using the Schmitt trigger as a neuron selector reads the updated membrane potential (Figure 2), and ii) to act as a 128-byte cache for MAC units to be used in the input layer and to be pre-loaded in the output layer state.

E. Membrane Potential Arithmetic Unit

The MAU receives the final potential value from the neuron selector and resets the potential if that neuron has triggered a spike in the present time step. The final value is stored, and the decayed potentials are loaded into the potential adder, where the membrane potential and the decay rate β are read from SRAM and multiplied with a combinational circuit and routed to its respective potential adder using the neuron selector. To decrease latency, a combinational circuit was used for multiplication where the membrane potential is divided into smaller fractions using shift operations, and then they are added back together in different combinations based on the decay rate to compute the final decayed membrane potential value (Figure 3).

F. OpenRAM Macros

Memory macros have classically been highly proprietary and manually optimized circuits, as memory cells rely on layouts that are so dense they must often bypass design rule violations. Designs are often limited by the availability of memory designs and arrays, and memory compilers are

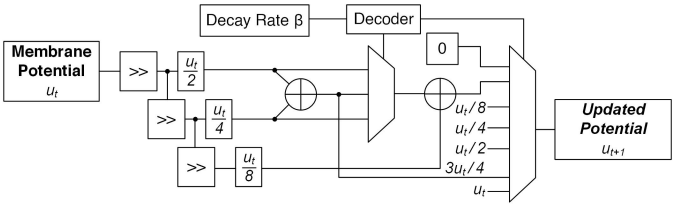

Fig. 3: Membrane potential arithmetic unit uses shift-and-add to implement varying decay rates with low computational cost.

TABLE I: Network Timing for a Dense SNN

Process	Latency
Input Layer	0.120 μs
Hidden Layer	10.3 μs
Output Layer	10.3 μs
Total	20.72 μs

* Network architecture: 1024-1024-10 Dense SNN.

scarcely available with existing PDKs. OpenRAM is an open-source memory compiler that helps with addressing these issues [27]. By using a high-level Python configuration file, the layout and netlists of SRAMs can be generated by passing in data word size, the number of words in memory, for a given PDK. Process, voltage and temperature corners can also be specified for SRAM characterization.

The proposed design uses approximately 154 kB worth of dual-port 2 kB SRAM macros that consume a total area of 21.89 mm^2 for storing spikes, adaptive thresholds from the Schmitt triggers which enables inhibition, membrane potential, the decay rate, weights, and incoming spikes. The SRAMs operate at 40 MHz while the core is clocked at 20 MHz. The memory controller addresses spikes to all fan-out weights, and interfaces all SRAMs to the core. The weight read lines are multiplexed within one clock cycle of the accelerator core to increase bandwidth.

IV. Results

A. Backend Flow

The OpenLane flow was used to harden the accelerator core, turning synthesized Verilog into a GDSII layout [28]. Logic synthesis, floorplanning, placement, clock routing and optimization, global and detailed routing are performed within this flow. Hardening the accelerator core took 10 hours with a peak of 42.5 GB of RAM utilized, generated 128,776 physical cells, and the resulting GDSII layout is illustrated in Figure 4.

B. Timing

The critical path of the core is 24.56 ns due to the neuron state calculation in the MAU. To optimize the MAU, the throughput of the membrane potential decay step is doubled by including a pair of adders. The maximum clock frequency is 24.39 MHz. The SRAMs can operate at 40 MHz and take two cycles per memory access, and so to balance weight access with computation, the core is driven at 20 MHz.

TABLE II: Power consumption at the fastest corner

Group	Internal	Switching	Leakage	Total	%
Sequential	31.4 mW	2.32 mW	0.37 μW	33.9 mW	28.5%
Combinational	44.1 mW	40.7 mW	2.37 μW	84.8 mW	71.5%
Total	75.5 mW	43.2 mW	2.74 μW	119 mW	100.0%
%	63.6%	36.4%	∼0.0%	100%	

TABLE III: Accuracy of OpenSpike across different datasets compared to full precision counterparts

Dataset	Accuracy (OpenSpike)	Accuracy (Full precision)
MNIST	99.12%	99.42%
FashionMNIST	88.12%	91.02%
DVSGesture	92.36%	93.06%

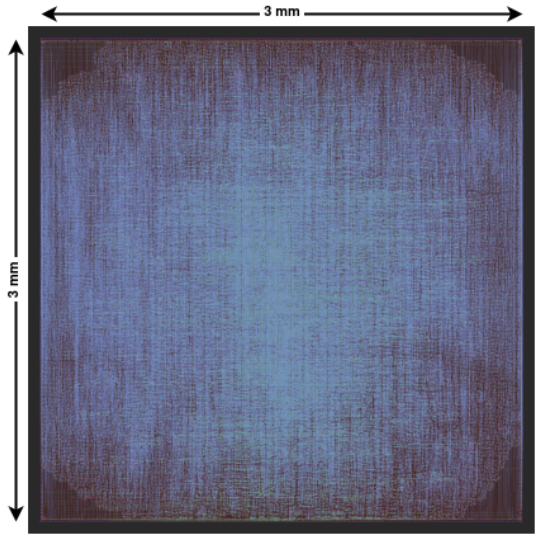

Fig. 4: GDSII of the accelerator core.

The timing of various stages of the network are provided in Table I, noting that this is only for one case out of any number of possible architectural configurations. Initialization involves loading firmware onto the PicoRV32 processor to configure on-chip logic analyzers, 38 general purpose IOs (maximum bandwidth of 50 MHz), and a wishbone bus to interface the accelerator with the firmware. The input layer takes 0.12 μs to complete, where input images pre-loaded on a 16kB bank of SRAM. The input is treated as a flattened single-channel 32×32 image. The input is then passed to the hidden layer, where each input is weighted by 1,024 weights and takes a total of 10.3 μs. The final layer has almost the same latency, as the same number of neuron inputs must be processed. The wallclock time for a forward-pass is 20.72 μs. The total throughput without pipelining is therefore 48,262 images per second.

C. Power

The OpenSpike core and its SRAM macros consume about the same amount of energy. SRAM macros during read and write operations of the core consume 106.54 mW of power in total. The dominant portion of power consumption of the core arises from dynamic power in combinational logic. A power breakdown at the fastest corner is provided in Table II. Internal power refers to dynamic power in standard cells, and switching power accounts for dynamic power across routing and capacitances external to cells. The dynamic power is the total of the internal and external switching power, and amounts to 119 mW for a fully activated network (i.e., all inputs are spiking) in the worst-case. This can be significantly reduced by training sparse networks, e.g., by using sparse input data from event cameras [29], and by imposing objectives that aim to reduce spike count [4], [5], [12], [30].

D. Accuracy

The accuracy of SNNs with binarized weights, where the network weights $w \in \{-1, +1\}$, has been measured on several image datasets, including MNIST, FashionMNIST, and DVSGesture [29]. We tested the performance of small convolutional SNN architectures (16Conv5-64Conv5-10) on classification tasks in a supervised learning setup where the best performance for the MNIST dataset was 99.12%, the FashionMNIST dataset was 88.12%, and for DVSGesture 92.36% was achieved (16Conv5-32Conv5-11). These represent very small performance hits when compared to their full precision counterpart networks, where MNIST only degraded by 0.3%, FashionMNIST by 2.9%, and DVSGesture by 0.7%. The same hyperparameters and surrogate gradients as in Ref. [26] are adopted, where the binarization operator is replaced with a straight-through-estimator during the backward-pass.

V. Discussion and Conclusion

The proposed SNN accelerator core based on OpenRAM memories aims to drive neuromorphic hardware research in the direction of reproducibility, in much the same way algorithms and software development has gone. The work gone into making neuromorphic accelerators available for broad, public use are extremely useful for applications engineers in the pursuit of applied neuromorphic research, though is of less use in the chip design flow outside of algorithms exploration. Frenkel *et al.* moved towards open-sourcing a neuromorphic chip for online learning, and though it relies on closed-source PDKs and memory macros, making all other aspects of the design available can help the neuromorphic community with both education and expanding the reach of custom hardware [31]. Our accelerator addresses the challenges of open hardware by using a *fully* open-source design flow. While open flows are presently constrained to legacy nodes, industry involvement and more advanced technologies have already become better integrated within open-EDA toolchains within the past year, including GlobalFoundries 180 nm and SkyWater's RRAM-CMOS process. Open neuromorphic accelerators have the potential to do what the open-source community has achieved for DL.

REFERENCES

[1] J. C. Knight and T. Nowotny, "Larger GPU-accelerated brain simulations with procedural connectivity," *Nature Computational Science*, vol. 1, no. 2, pp. 136–142, 2021.

[2] J. P. Turner, J. C. Knight, A. Subramanian, and T. Nowotny, "mlgenn: accelerating snn inference using gpu-enabled neural networks," *Neuromorphic Computing and Engineering*, vol. 2, no. 2, p. 024002, 2022.

[3] M. Stimberg, D. F. Goodman, and T. Nowotny, "Brian2genn: a system for accelerating a large variety of spiking neural networks with graphics hardware," *bioRxiv*, p. 448050, 2018.

[4] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *arXiv preprint arXiv:2109.12894*, 2021.

[5] 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 Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019.

[6] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[7] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[8] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith, "Nengo: A python tool for building large-scale functional brain models," *Frontiers in Neuroinformatics*, vol. 7, p. 48, 2014.

[9] R. T. Edwards, "Google/SkyWater & the promise of the open pdk," in *Proc. Workshop on Open-Source EDA Technol.(WOSET)*, 2020.

[10] A. Ghazy and M. Shalan, "Openlane: The open-source digital asic implementation flow," in *Proc. Workshop on Open-Source EDA Technol.(WOSET)*, 2020.

[11] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature Communications*, vol. 11, no. 1, pp. 1–15, 2020.

[12] A. Henkes, J. K. Eshraghian, and H. Wessels, "Spiking neural network for nonlinear regression," *arXiv preprint arXiv:2210.03515*, 2022.

[13] Y. Yang, N. D. Truong, J. K. Eshraghian, A. Nikpour, and O. Kavehei, "Weak self-supervised learning for seizure forecasting: a feasibility study," *Royal Society Open Science*, vol. 9, no. 8, p. 220374, 2022.

[14] K. M. Stewart and E. Neftci, "Meta-learning spiking neural networks with surrogate gradient descent," *Neuromorphic Computing and Engineering*, 2022.

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

[16] R.-J. Zhu, Q. Zhao, T. Zhang, H. Deng, Y. Duan, M. Zhang, and L.-J. Deng, "Tcja-snn: Temporal-channel joint attention for spiking neural networks," *arXiv preprint arXiv:2206.10177*, 2022.

[17] N. Perez-Nieves and D. Goodman, "Sparse spiking gradient descent," *Advances in Neural Information Processing Systems*, vol. 34, pp. 11 795–11 808, 2021.

[18] Y. Venkatesha, Y. Kim, L. Tassiulas, and P. Panda, "Federated learning with spiking neural networks," *IEEE Transactions on Signal Processing*, vol. 69, pp. 6183–6194, 2021.

[19] M. E. Elbtity, P. S. Chandarana, B. Reidy, J. K. Eshraghian, and R. Zand, "APTPU: Approximate computing based tensor processing unit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 12, pp. 5135–5146, 2022.

[20] P. Zhou, D.-U. Choi, W. D. Lu, S.-M. Kang, and J. K. Eshraghian, "Gradient-based neuromorphic learning on dynamical rram arrays," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 12, no. 4, pp. 888–897, 2022.

[21] S. M. Kang, D. Choi, J. K. Eshraghian, P. Zhou, J. Kim, B.-S. Kong, X. Zhu, A. S. Demirkol, A. Ascoli, R. Tetzlaff *et al.*, "How to build a memristive integrate-and-fire model for spiking neuronal signal generation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 12, pp. 4837–4850, 2021.

[22] J. K. Eshraghian, S.-M. Kang, S. Baek, G. Orchard, H. H.-C. Iu, and W. Lei, "Analog weights in reram dnn accelerators," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2019, pp. 267–271.

[23] P. Dayan and L. F. Abbott, *Theoretical neuroscience: Computational and mathematical modeling of neural systems*. MIT press, 2005.

[24] L. Lapique, "Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization." *J. of Physiol. and Pathology*, vol. 9, pp. 620–635, 1907.

[25] J. K. Eshraghian, C. Lammie, M. R. Azghadi, and W. D. Lu, "Navigating local minima in quantized spiking neural networks," *arXiv preprint arXiv:2202.07221*, 2022.

[26] J. K. Eshraghian and W. D. Lu, "The fine line between dead neurons and sparsity in binarized spiking neural networks," *arXiv preprint arXiv:2201.11915*, 2022.

[27] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "Openram: An open-source memory compiler," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–6.

[28] T. Ajayi and D. Blaauw, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," in *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019.

[29] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7243–7252.

[30] F. Zenke and T. P. Vogels, "The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks," *Neural computation*, vol. 33, no. 4, pp. 899–925, 2021.

[31] C. Frenkel and G. Indiveri, "Reckon: A 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 1–3.