

Dynamic Neural Thresholding on Mixed-Signal Neuromorphic Processors Enabled by Integrated Learning and Hardware Design

Kyuseung Han^{a*}, Kwang-Il Oh^{a*}, Sukho Lee^a, Hyeonguk Jang^a, Jae-Jin Lee^a, Sooyoung Jang^{b†}

^a*Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea*

^b*Hanbat National University, Daejeon, Republic of Korea*

Abstract—Spiking neural networks (SNNs) can improve inference accuracy through joint optimization of synaptic weights and neuronal thresholds. However, mixed-signal neuromorphic processors, which are designed for energy efficiency using analog circuits, face practical limitations. In particular, digital to analog converters (DACs) often lack sufficient resolution to represent the large threshold values required by joint optimization. To address this issue, we propose a mixed-signal neuromorphic processor architecture that shifts threshold control to digital logic. This approach removes the need for high-resolution DACs and allows dynamic threshold adjustment without modifying the analog neural core. We also propose a learning method tailored to this architecture. We evaluate the proposed design on five image classification benchmarks, measuring accuracy, latency, and energy consumption. The results show that our architecture consistently improves accuracy across benchmarks while incurring only minimal latency and energy overhead. This demonstrates that the proven benefits of joint weight and threshold learning can be realized in energy efficient analog hardware.

Index Terms—neuromorphic processor, spiking neural network, SoC, mixed-signal, threshold

I. INTRODUCTION

Neuromorphic computing is a computational paradigm inspired by the structure and function of the human brain. At the heart of neuromorphic systems are spiking neural networks (SNNs), which encode and transmit information via discrete spikes of electrical activity. Compared to traditional artificial neural networks (ANNs), SNNs offer event-driven processing that more closely mirrors biological neuronal behavior. This property not only enhances the biological plausibility of the models but also enables low-power, real-time data processing, rendering SNNs particularly compelling for resource-constrained and embedded applications [1]–[3].

In hardware implementations, the basic building blocks of SNNs, namely neurons and synapses, can be realized using either digital or analog circuits. While digital designs offer ease of programmability and systematic integration [4]–[9], analog circuits can yield advantages in energy efficiency and more directly emulate biological behavior [10], [11]. Nonetheless,

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2025-02263706, Development of an analog-digital mixed ultra-low-power neuromorphic edge SoC)

*K. Han and K. Oh are contributed equally to this work as first authors.

†Corresponding author: Sooyoung Jang (syjang@hanbat.ac.kr).

relying solely on analog circuits can limit the flexibility needed for more intricate computational tasks. To address these limitations, recent studies have adopted mixed-signal architectures that combine analog and digital components to balance energy efficiency with design flexibility [12]–[21].

Recently, research in neuromorphic computing has advanced significantly, especially in learning mechanisms for SNNs [22], [23]. Early efforts predominantly focused on spike-timing-dependent plasticity (STDP) [24], [25], which updates synaptic weights based on the precise timing of neuronal spikes. While STDP offers strong temporal learning capabilities with low energy consumption, its accuracy has historically lagged behind that of ANNs. Consequently, researchers have begun incorporating backpropagation-based methods into SNNs [26]–[28], leveraging advanced optimization strategies commonly used in ANNs [29], [30]. This integration narrows the performance gap, enabling high-accuracy, low-power neuromorphic systems.

Despite recent advances, integrating backpropagation-based methods into analog circuits remains challenging. In particular, the implementation of dynamic neural thresholding in analog SNNs is a critical yet underexplored issue. This mechanism adaptively modulates a neuron’s firing threshold based on recent activity. Recent studies [31]–[33] have demonstrated that jointly optimizing weights and thresholds can significantly improve model accuracy. However, when thresholds are optimized jointly with weights, they can become significantly larger than typical weight magnitudes, creating practical challenges for analog hardware, particularly digital-to-analog converters (DACs). To address this, we propose a mixed-signal neuromorphic processor architecture that offloads threshold control to digital circuitry. By preserving the analog core, this approach circumvents the resolution limitations of analog DACs. We demonstrate the effectiveness of this approach through a 28 nm CMOS implementation.

II. RELATED WORK

Backpropagation methods inherently require subtraction and the handling of negative values, as both weights and membrane potentials are signed. These challenges have been addressed in analog neuron designs even before the adoption of backpropagation [34]–[37]. However, existing approaches typically rely on a single capacitor per neuron, which presents significant implementation hurdles—such as the need for a negative voltage

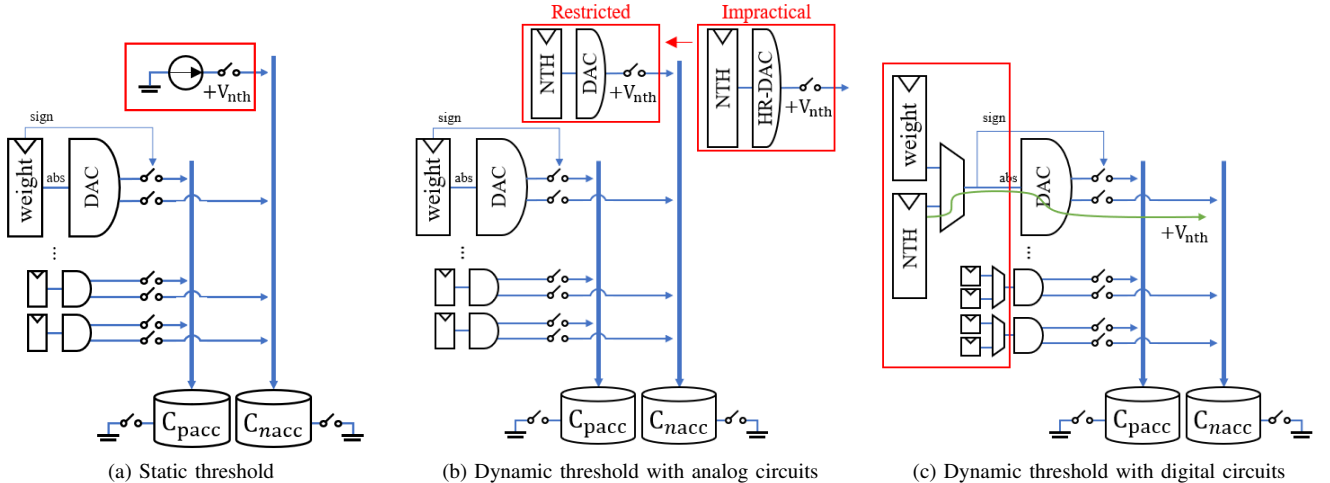


Fig. 1: Block diagrams of mixed-signal circuits for a single neuron and multiple synapses.

supply or increased DAC resolution. To address these limitations, recent studies have proposed the use of two capacitors instead [17]–[19]. Since this work also targets backpropagation-based learning, we build upon the dual-capacitor architecture to support dynamic threshold adjustment.

To date, analog circuit based designs, whether purely analog or mixed signal, have paid limited attention to threshold adjustment [12]–[21], [34]–[37]. In most cases, threshold adjustment is either entirely absent or not described in sufficient detail. Even when such functionality is implemented at the circuit level, it often lacks consideration of learning dynamics, and its system level validity remains unverified. To address this gap, this work identifies the specific requirements for threshold control in the context of learning, and proposes a circuit architecture that incorporates these requirements and demonstrates its effectiveness through experimental validation.

III. THRESHOLD ADJUSTMENT

The integrate-and-fire model with a static threshold is illustrated in Fig. 1a. Each synapse is implemented using a weight register and a DAC, while the membrane potential is represented by two capacitors, C_{pacc} and C_{nacc} , corresponding to positive and negative accumulation, respectively. During operation, when a synapse is activated, it deposits charge into one of two capacitors. For positive weights, the charge is added to C_{pacc} ; for negative weights, the charge is added to C_{nacc} . Given that C_{nacc} is precharged to V_{nth} , which represents the neural threshold value, the effective membrane potential is represented by the difference between the voltages on C_{pacc} and C_{nacc} . As long as the voltage on C_{pacc} is lower than that on C_{nacc} , the charge on the two capacitors continues accumulating. When the voltage of C_{pacc} exceeds that of C_{nacc} , a spike is fired and the voltages of C_{pacc} and C_{nacc} become 0 and V_{nth} , respectively, which is referred to as a hard-reset. Alternatively, in a soft-reset, C_{pacc} is maintained while C_{nacc} is incremented by V_{nth} . Further details on these circuits can be found in [18], [19].

By replacing the fixed current source with a threshold register and an additional DAC, the threshold value can be customized, as shown in the top-right part of Fig. 1b. When the learned neural threshold value (NTH) is stored in the register, the DAC converts it into a current that increases the voltage of C_{nacc} by V_{nth} . While the method is conceptually simple, it poses a major challenge—our joint training experiments show that optimal thresholds may reach up to 19 times the typical weight values. This discrepancy implies that the DAC for the threshold would require approximately 5 additional bits of resolution compared to the original DAC. Therefore, we denote the DAC used for the threshold as a high-resolution DAC (HR-DAC) in the figure. However, DACs are typically designed with the highest feasible resolution, considering factors such as process technology, chip area, and synapse structure, since a wider weight range enhances accuracy. Consequently, designing an HR-DAC becomes nearly impractical [38], forcing the use of the original DAC despite its constrained representational range.

To overcome this challenge, we propose a viable approach that eliminates the need for a high-resolution DAC for threshold adjustment. Digital circuits are employed to artificially generate spikes, enabling threshold values to function analogously to negative weights (see Fig. 1c). This method enables dynamic threshold adjustment while preserving the analog circuitry from the conventional design shown in Fig. 1a. The detailed architecture will be described in Section IV-C.

IV. MIXED-SIGNAL NEUROMORPHIC PROCESSOR

A. Overall architecture

Fig. 2 illustrates the overall architecture of the proposed neuromorphic processor, designed for SNN inference in lightweight embedded systems. The neural accelerator is connected to the system interconnect via five AMBA interfaces [39]. Four AXI interfaces facilitate access to large data arrays, while a Rocket core [40] controls the accelerator through an APB interface, executing applications stored in the main memory. The accelerator comprises three main components: the neural core, load/store units, and a controller.

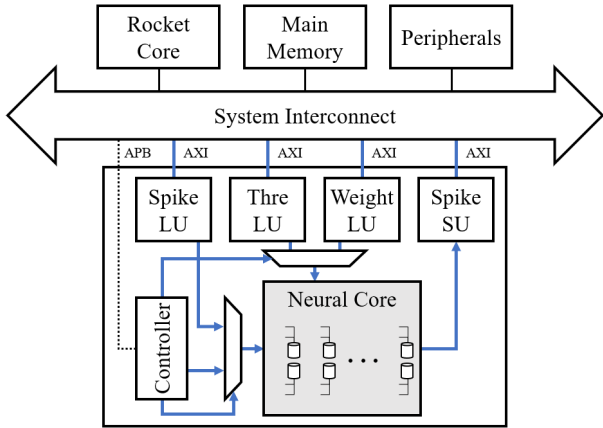


Fig. 2: Overall architecture of the proposed neuromorphic processor. The gray part is analog, while the rest are digital.

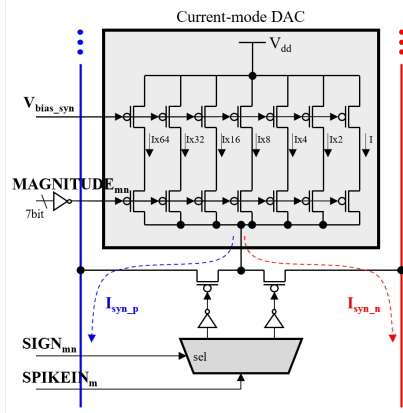


Fig. 3: The schematic of implemented analog synapse circuits.

The neural core, consisting of synapses and neurons where the actual computation occurs, is implemented using analog circuits to achieve low-power operation. It is precisely designed to produce binary output signals—0 or V_{dd} —corresponding to logical 0 and 1, respectively [18], [19]. These binary outputs make it possible to implement a digital neural core that is functionally equivalent to the analog one, allowing FPGA-based early-stage prototyping. Load units (LUs) read input spikes (Spike LU), thresholds (Thre LU), and weights (Weight LU) from memory, while the store unit (SU) writes output spikes (Spike SU) back to memory. The controller orchestrates these components based on configurations provided by the Rocket core, enabling the accelerator to compute one layer at a time and, through iteration, execute the complete SNN.

B. Analog neural core

The analog neural core is structured as a 32×32 array, where each of the 32 neurons is connected with 32 synapses. Fig. 3 presents the schematic of the synapse, each of which is realized as a current-mode DAC that converts a 7-bit magnitude into an analog current, transmitting either I_{syn_p} or I_{syn_n} depending on the sign bit. Since inhibitory and excitatory signals are mutually exclusive, a single DAC is shared within each synapse.

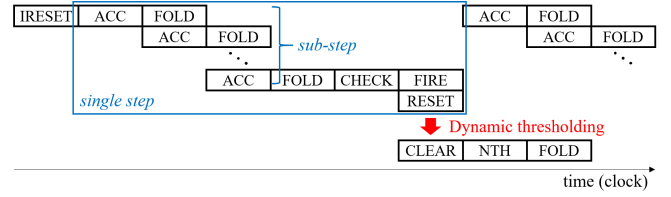


Fig. 4: Computation of one layer based on the neuron's single cycle operations.

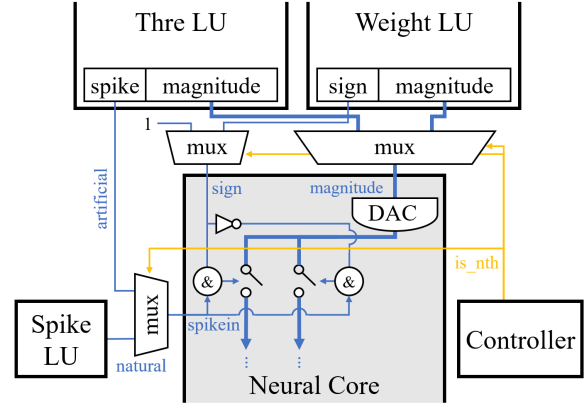


Fig. 5: Modified input control of the neural core.

C. Digital circuits

The sequence of neuron operations for computing a single layer at the clock level is illustrated in Fig. 4. When the layer size exceeds the hardware size, charge accumulation is repeated over multiple iterations. Additionally, due to the limited capacity of the capacitors, the accumulation occurs gradually, followed by voltage folding to prevent overflow. Consequently, generating a single spike may require several sub-steps of accumulation and folding, executed pipelined within one processing step. After the accumulation process, a CHECK operation compares the two capacities to determine if a spike should occur; if the firing condition is met, a spike is generated in the next cycle, and the capacitors are reset simultaneously. If the threshold values remain static, reset operations are handled entirely by the analog circuitry and can be completed within a single cycle. In contrast, if the threshold values are determined dynamically based on the register, the reset operations span three cycles due to the control sequence between the digital and analog circuits. Although this change increases the number of cycles per step by two, operational overlap ensures that the actual overhead amounts to just one extra cycle.

To integrate threshold management alongside weight management, we implement a method to control the neural core's inputs. This control mechanism is illustrated by two trapezoidal regions in Fig. 2 and is associated with two neuron operations: the ACC operation, where the neural core processes 'natural' input spikes and weights, and the NTH operation, where artificial input spikes and threshold values are utilized. The term 'natural' distinguishes the inputs used in the ACC operation from the artificially generated ones used in the NTH operation.

Fig. 5 provides a detailed view of a single synapse operation.

TABLE I: Splitting the trained threshold to multiple registers when the maximum of weights is 127

Threshold Value	NTH0	NTH1	NTH2	...
200	(1,127)	(1,73)	(0,0)	(0,0)
300	(1,127)	(1,127)	(1,46)	(0,0)

The synapse was carefully designed to enable seamless and efficient execution from software to hardware circuitry. The trained weights, initially stored in two’s complement format, are converted into a sign-and-magnitude format compatible with the DACs. This conversion occurs once during application start-up, incurring negligible overhead. Similarly, the threshold value is also converted into a sign-and-magnitude format. However, since the sign for the threshold is always 1 (indicating an increase in C_{nacc}), there is no need to store it in a register. Instead, the single bit is used as a flag to indicate whether to generate an artificial spike. The threshold magnitude is then distributed across multiple synapses. TABLE I lists the register values (NTH0 to NTH2) that include both the artificial spike flag and the magnitude values. For example, a threshold of 200 can be partitioned into two synapses with values 127 and 73 since a single synapse register can represent a maximum of 127. This partitioning, along with the weight format conversion, is performed in software before runtime. Finally, the values stored in the Thre LU and Weight LU are selected via three muxes along with the original inputs and then passed to the neural core. The controller generates appropriate selection signals for the muxes based on whether the neuron operation is in ACC or NTH mode. In contrast, the conventional architecture delivers both spikes and synaptic weights (comprising signs and magnitudes) directly to the neural core without additional input control.

V. MODULAR LEARNING METHOD

In mixed-signal neuromorphic processors, quantization is a fundamental hardware constraint rather than a mere optimization technique. We therefore adopted the widely used quantization-aware training (QAT) technique proposed in [41], where the quantization range is dynamically determined based on the maximum and minimum values of the weights observed during training. However, since the scaling factor is derived from these dynamic ranges, the threshold value may exceed the upper integer limit of the quantization levels, Q_{max} . To address this, it is necessary to fix the quantization range to [1.0,1.0], so that the default threshold value (1.0) is always quantized to Q_{max} in the weight-only method.

As the proposed architecture enables dynamic adjustment of neural thresholds, it removes constraints on their magnitude and facilitates the learning of thresholds alongside weights. This not only makes joint learning possible but also allows the method in [41] to be directly applied without modification. However, initial experiments on five representative tasks using the soft-reset mechanism (TABLE II) showed that naïvely applying this method led to suboptimal performance compared to the weight-only approach. This degradation likely results from increased learning complexity due to the expanded parameter space,

TABLE II: Degraded accuracy from the naïve approach

Dataset	Accuracy (%)	
	Weight-only	Naïve
MNIST	97.64	97.85
KMNIST	90.98	90.73
FMNIST	80.12	76.11
GTSRB	89.65	87.03
CIFAR-10	55.96	54.23

TABLE III: Improved accuracy from the proposed modular approach on GTSRB application using soft-reset

Method	Q_{max} : NTH	Quantization Range	Threshold Range	Accuracy (%)
Weight-only	N/A	[1.00,1.00]	1.00	89.65
Naïve	N/A	Dynamic	[0.00,1.00]	87.03
Modular	1:1~1:2	[-0.50,0.50]	[0.50,1.00]	89.93
	1:2~1:4	[-0.25,0.25]	[0.50,1.00]	90.15
	1:4~1:10	[-0.10,0.10]	[0.40,1.00]	90.57

which interacts poorly with quantization effects. These results highlight the need for more structured learning strategies.

To overcome this, we propose a *modular* learning approach that divides the solution space into independent subproblems, each defined over a predefined parameter range. In each subproblem, the learnable threshold range is constrained, and the quantization range is fixed based on the ratio between weights and thresholds. As demonstrated in TABLE III for the GTSRB application, this approach reduces learning complexity and improves performance compared to the baseline. Since the subproblems are independent, they can be processed in parallel, thereby accelerating the overall training process. After all subproblems have been completed, their results are aggregated, and the model with the highest performance is selected as the final output. On the other hand, when the representable range of thresholds is limited due to the DAC constraints illustrated in Fig. 1b, the quantization range of the weights must be determined based on the learned threshold values. The methods discussed in this section are summarized in TABLE IV.

VI. EXPERIMENTAL RESULTS

A. Dataset and training methodology

We employed *snnTorch* [30] to train both synaptic weights and neural thresholds simultaneously. We selected five image-classification benchmarks to evaluate the impact on accuracy: digits (MNIST), Japanese characters (Kuzushiji-MNIST, KMNIST), clothing items (Fashion-MNIST, FMNIST), traffic signs (GTSRB), and general objects (CIFAR-10). While the first three datasets are relatively small, the latter two pose larger-scale tasks. Since the DACs operate with 7-bit resolution in our system, 8-bit QAT is applied to include the sign bit.

TABLE IV: Characteristics of the QAT variations

Method	Quantization Range	Threshold	Quantized Threshold	Circuit
Weight-only	[1.0,1.0]	1.0	Q_{max}	Fig. 1a
Restricted	[NTH,NTH]	Learnable	Q_{max}	Fig. 1b
Modular	Predefined	Learnable	Dynamic	Fig. 1c

TABLE V: Area and power breakdown of the mixed-signal processor implemented in 28nm CMOS process

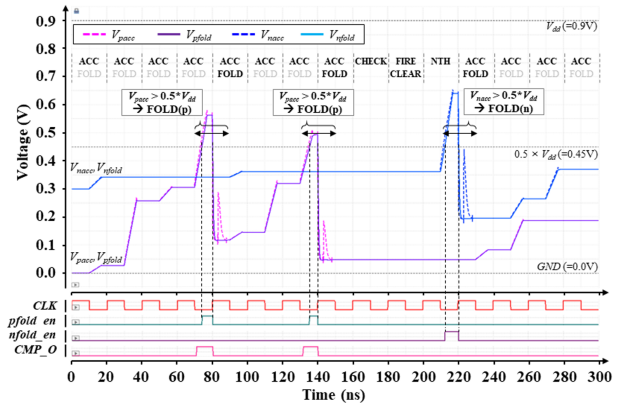
	Module	Area (μm^2)	Power (mW)
Accelerator	Synapse $\times(32\times32)$	67276 (13%)	0.44 (5%)
	Neuron $\times32$	25319 (5%)	0.09 (1%)
	LU $\times2 + \text{SU}\times1$	49088 (10%)	1.51 (16%)
	Thre LU + Muxes + α	4266 (1%)	0.13 (1%)
	Controller	47642 (9%)	1.51 (16%)
	Overall	193591 (39%)	3.68 (39%)
	Rocket core	156402 (31%)	0.73 (8%)
	The Rest	152486 (30%)	4.92 (53%)
	Overall	502479 (100%)	9.33 (100%)

B. Implementation of the proposed processor

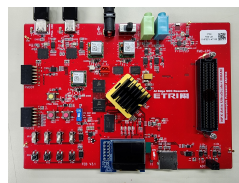
The proposed mixed-signal processor is designed to operate at 50MHz using a 28nm CMOS process technology. In this design, all components except for the analog neural core are implemented as digital circuits developed in Verilog HDL. The digital circuits are synthesized and verified with Synopsys Design Compiler and PrimeTime [42], while the analog neural core is designed and simulated using Cadence Virtuoso and Spectre [43]. Specifically, the analog circuit was designed with careful consideration of process variations using Monte Carlo simulations. In particular, the neuron input offset variation was as small as 3.34mV (2σ), which is an important factor for accurate analog computation. A detailed breakdown of the area and power consumption is provided in TABLE V. Here, the notation ‘LU $\times2$ +SU $\times1$ ’ refers to the conventional Spike LU, Weight LU, and Spike SU. In contrast, the newly introduced Thre LU along with the muxes and the modified part of the controller, as illustrated in Fig. 5, occupy 2.2% ($\approx 4266/193591$) of the accelerator’s total area and 3.5% ($\approx 0.13/3.68$) of its total power consumption, respectively. These overheads are reduced to approximately 1% when considered in the context of the entire processor.

The operation of the analog circuitry under modified control signals from digital logic was verified through simulation. Fig. 6a demonstrates the stepwise operation of the neuron shown in Fig. 4, controlled by digital signals which are synchronized with the clock. The detailed circuit characteristics are beyond the scope of this paper, as they remain nearly unchanged from those with a static threshold.

The entire processor was then validated using an FPGA prototype. As described in Section IV-A, the neural core operates with binary inputs and outputs, which enables its functional replication in a digital form. This approach facilitates application development and performance analysis prior to fabrication. For prototyping, an FPGA board was designed based on the Xilinx Artix UltraScale+ [44], as shown in Fig. 6b. The trained models for the five applications were converted into C header files, compiled with a software layer capable of running them, and then loaded into the prototype in binary form. All these compilation processes, from applications to the RISC-V binary, are automated by modifying the NPX framework [45]. We selected 10 samples from each application and thoroughly validated the prototype by ensuring that all



(a) Simulation of analog circuits



(b) A prototype

Application	Execution Time (ms)
MNIST	467
KMNIST	465
FMNIST	245
GTSRB	12030
CIFAR-10	6598

(c) Execution time

Fig. 6: Verification of the mixed-signal neuromorphic processor.

intermediate values matched during the execution of these 50 samples. After verification, we measured the execution times of the applications, with results summarized in Fig. 6c. This implies that the proposed architecture is carefully designed to ensure the seamless execution of application software.

C. Performance analysis

To investigate the effectiveness of dynamic threshold adjustment in mixed-signal neuromorphic processors, we evaluated three distinct hardware architectures, illustrated in Fig. 1. As introduced earlier, pure analog implementations cannot support negative weights [10], [11], making them incompatible with backpropagation-based training methods that have consistently demonstrated superior accuracy. In contrast, digital-only circuits offer high algorithmic flexibility but suffer from significant power consumption [19]. Mixed-signal architectures aim to strike a balance between these two extremes, offering a promising trade-off between flexibility and energy efficiency. The three mixed-signal architectures considered in this work are summarized as follows:

- The *Baseline* architecture, shown in Fig. 1a, lacks threshold programmability. This approach is commonly found in previous mixed-signal designs, including [19].
- The *Alternative* architecture, illustrated in Fig. 1b, is based on [18]. Although it supports threshold programmability, the threshold values must remain within the same dynamic range as the weights, which introduces training constraints.
- The *Proposed* architecture, presented in Fig. 1c, incorporates digital control logic that enables threshold values to be programmed independently of weight magnitudes.

The experimental results are summarized in TABLE VI. Accuracy was evaluated as the average over five-fold cross-validation for both hard-reset and soft-reset training schemes.

TABLE VI: Comparison of accuracy, latency, and energy consumption across applications for different approaches

Approach		Metric	Application									
(Circuits / Joint Learning)			MNIST		KMNIST		FMNIST		GTSRB		CIFAR-10	
		Static Threshold	127		127		127		127		127	
Baseline (Fig. 1a / Weight-only)	Accuracy	Hard	97.18	-	90.73	-	74.83	-	86.93	-	54.39	-
		Soft	97.64	-	90.98	-	80.12	-	89.65	-	55.96	-
	Latency	1.76	-	1.76	-	1.60	-	69.52	-	108.64	-	
	Energy	6.31	-	6.31	-	5.73	-	249.10	-	389.27	-	
		Learned Threshold	84 ~ 114		72 ~ 124		84 ~ 106		47 ~ 127		45 ~ 127	
Alternative (Fig. 1b / Restricted)	Accuracy	Hard	96.93	(-0.25)	87.23	(-3.50)	75.06	(+0.23)	72.88	(-14.05)	47.10	(-7.29)
		Soft	97.53	(-0.11)	91.22	(+0.24)	80.99	(+0.87)	87.29	(-2.36)	56.00	(+0.04)
	Latency	1.76	(0.0%)	1.76	(0.0%)	1.60	(0.0%)	69.52	(0.0%)	108.64	(0.0%)	
	Energy	6.51	(+3.2%)	6.51	(+3.2%)	5.91	(+3.2%)	256.96	(+3.2%)	401.56	(+3.2%)	
		Learned Threshold	129 ~ 228		664 ~ 1050		162 ~ 228		489 ~ 2368		395 ~ 1299	
Proposed (Fig. 1c / Modular)	Accuracy	Hard	97.25	(+0.07)	90.87	(+0.14)	76.04	(+1.21)	89.99	(+3.06)	56.66	(+2.27)
		Soft	97.91	(+0.27)	91.92	(+0.94)	81.32	(+1.20)	90.57	(+0.92)	59.49	(+3.53)
	Latency	1.92	(+9.1%)	1.92	(+9.1%)	1.76	(+10.0%)	69.74	(+0.3%)	108.80	(+0.1%)	
	Energy	7.12	(+13.0%)	7.12	(+13.0%)	6.53	(+13.9%)	258.75	(+3.9%)	403.68	(+3.7%)	

*The units of accuracy, latency, and energy consumption are %, μ s, and nJ, respectively.

*The blue indicates an improvement compared to the baseline, while the red does a worsening.

Since the learned threshold values vary across layers, they are reported as ranges. Latency was defined as the maximum single-step execution time across all network layers, assuming that all layers are pipelined and executed in parallel; thus, it is inversely proportional to throughput. Power consumption was measured by accounting for the circuit overheads associated with each component. The additional DAC for threshold control in the *Alternative* is identical to the synapse DAC, so its overhead can be inferred from TABLE V. The overheads of newly introduced circuits in the *Proposed* are listed separately in the same table. Finally, energy consumption of the accelerator was computed as the product of latency and power consumption.

The experimental results showed that while the *Alternative* introduces joint weight-threshold optimization, it failed to outperform the *Baseline* in accuracy. In some cases, the *Alternative* even yielded lower accuracy, likely because forcing thresholds to fit the same range as weights constrain the search space and prevent the optimizer from finding an optimal threshold-weight configuration. Consequently, the *Alternative* does not provide a viable means to improve accuracy over the *Baseline*. In contrast, the *Proposed* enlarges the representational range for threshold values, reaching up to 2368 on GTSRB, which is 19 times the maximum value supported by the DAC. As a result, it delivered measurable accuracy improvements—up to 1.21% on smaller tasks (MNIST, KMNIST, FMNIST) and up to 3.53% on larger, more complex tasks (GTSRB and CIFAR-10).

While increased flexibility in threshold adjustment introduces additional operational cycles, the latency penalty of the *Proposed* varied considerably across datasets. For smaller tasks, latency increased by around 10%, but for larger tasks, this overhead dropped to 0.1%~0.3%, indicating that the overall computation time is primarily dominated by the layer size rather than threshold adjustment. The energy consumption

likewise increased in proportion to the additional cycles and the associated digital logic, resulting in 3.7%~3.9% overhead.

These results underscore the importance of dynamic neural threshold adjustment in mixed-signal environments. What matters is not the absolute accuracy, but rather the fact that higher accuracy can be achieved within the same network. In this context, the advantages of joint threshold-weight optimization [31]–[33] are effectively realized in mixed-signal neuromorphic processors, thereby bridging the gap between software-based learning frameworks and practical hardware implementations.

VII. CONCLUSION

In this work, we addressed a critical limitation of mixed-signal neuromorphic processors: the difficulty of supporting large neural thresholds in analog circuitry, which prevents these systems from obtaining any benefit from joint weight-threshold learning. We proposed an architecture that represents thresholds as artificial spikes with negative weights, thereby shifting threshold control to the digital domain and alleviating the analog design burden. We further developed a learning strategy aligned with this architecture, enabling integrated optimization of synaptic weights and neural thresholds.

Our approach was validated through simulations and FPGA prototyping, achieving up to 3.53% accuracy improvement on CIFAR-10 with only 0.1% latency and 3.7% energy overhead, while also delivering consistent gains across MNIST, KMNIST, FMNIST, and GTSRB. These results confirm that dynamic neural thresholding can be effectively realized in energy-efficient mixed-signal hardware. By bridging advanced learning methods with practical circuit-level implementation, this work establishes a foundation for future large-scale neuromorphic systems based on integrated learning-hardware design.

REFERENCES

- [1] M. Bouvier *et al.*, “Spiking neural networks hardware implementations and challenges: a survey,” *ACM JETC*, vol. 15, no. 2, 2019.
- [2] J. D. Nunes, M. Carvalho, D. Carneiro, and J. S. Cardoso, “Spiking neural networks: a survey,” *IEEE Access*, vol. 10, pp. 60 738–60 764, 2022.
- [3] A. Basu, L. Deng, C. Frenkel, and X. Zhang, “Spiking neural network integrated circuits: a review of trends and future directions,” in *Proc. of CICC*, 2022.
- [4] O. Moreira *et al.*, “NeuronFlow: a neuromorphic processor architecture for live AI applications,” in *Proc. of DATE*, 2020, pp. 840–845.
- [5] E. Lemaire *et al.*, “Synaptic activity and hardware footprint of spiking neural networks in digital neuromorphic systems,” *ACM TECS*, 2022.
- [6] W. Guo *et al.*, “Toward the optimal design and FPGA implementation of spiking neural networks,” *IEEE TNNLS*, vol. 33, no. 8, pp. 3988–4002, 2022.
- [7] W. Ye, Y. Chen, and Y. Liu, “The implementation and optimization of neuromorphic hardware for supporting spiking neural networks with MLP and CNN topologies,” *IEEE TCAD*, vol. 42, no. 2, pp. 448–461, 2023.
- [8] Intel, <https://www.intel.com/content/www/us/en/research/neuromorphic-computing-loihi-2-technology-brief.html>, accessed 8 Sep. 2025.
- [9] H. Jang, S. Lee, J.-J. Lee, and K. Han, “NeuGEMM: A reordering-free unified GEMM-Conv2D accelerator for lightweight neuromorphic processors,” in *Proc. of FPL*, 2025.
- [10] C. D. Schuman *et al.*, “A survey of neuromorphic computing and neural networks in hardware,” *arXiv preprint arXiv:1705.06963*, 2017.
- [11] M. El-Masry, T. Werner, A. Zjajo, and R. Weigel, “Toward efficient system-on-module for design-space exploration of analog spiking neural networks,” *IEEE TCS-I*, vol. 71, no. 8, pp. 3538–3549, 2024.
- [12] M. Bavandpour *et al.*, “Mixed-signal neuromorphic inference accelerators: recent results and future prospects,” in *Proc. of IEDM*, 2018.
- [13] A. Neckar *et al.*, “Braindrop: a mixed-signal neuromorphic architecture with a dynamical systems-based programming model,” *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2019.
- [14] J. Büchel *et al.*, “Supervised training of spiking neural networks for robust deployment on mixed-signal neuromorphic processors,” *Scientific reports*, vol. 11, no. 1, p. 23376, 2021.
- [15] D. Zendrikov, S. Solinas, and G. Indiveri, “Brain-inspired methods for achieving robust computation in heterogeneous mixed-signal neuromorphic processing systems,” *IOP NCE*, vol. 3, no. 3, p. 034002, 2023.
- [16] W. Fang, Z. Xuan, S. Chen, and Y. Kang, “An 1.38 nJ/inference clock-free mixed-signal neuromorphic architecture using ReL-PSP function and computing-in-memory,” in *Proc. of BIOCAS*, 2023, pp. 1–5.
- [17] Y. Ko *et al.*, “A 65 nm 12.92-nJ/inference mixed-signal neuromorphic processor for image classification,” *IEEE TCS-II*, vol. 70, no. 8, pp. 2804–2808, 2023.
- [18] S. Kim *et al.*, “Neuro-CIM: ADC-less neuromorphic computing-in-memory processor with operation gating/stopping and digital-analog networks,” *IEEE JSSC*, vol. 58, no. 10, pp. 2931–2945, 2023.
- [19] K. Han *et al.*, “STARC: crafting low-power mixed-signal neuromorphic processors by bridging SNN frameworks and analog designs,” in *Proc. of ISLPED*, 2024.
- [20] O. Richter *et al.*, “DYNAP-SE2: a scalable multi-core dynamic neuromorphic asynchronous spiking neural network processor,” *IOP NCE*, vol. 4, no. 1, p. 014003, 2024.
- [21] M. Isik, N. Howard, S. Miziev, and W. Pawlak, “Advancing neuromorphic computing: mixed-signal design techniques leveraging brain code units and fundamental code units,” in *Proc. of IJCNN*, 2024, pp. 1–8.
- [22] A. Taherkhani *et al.*, “A review of learning in biologically plausible spiking neural networks,” *Neural Networks*, vol. 122, pp. 253–272, 2020.
- [23] F. Liu *et al.*, “Advancing brain-inspired computing with hybrid neural networks,” *National Science Review*, vol. 11, no. 5, pp. 1–16, 2024.
- [24] C. Lee, G. Srinivasan, P. Panda, and K. Roy, “Deep spiking convolutional neural network trained with unsupervised spike-timing-dependent plasticity,” *IEEE TCDS*, vol. 11, no. 3, pp. 384–394, 2019.
- [25] A. Vigneron and J. Martinet, “A critical survey of STDP in spiking neural networks for pattern recognition,” in *Proc. of IJCNN*, 2020, pp. 1–9.
- [26] S. K. Esser *et al.*, “Backpropagation for energy-efficient neuromorphic computing,” in *Proc. of NEURIPS*, vol. 28, 2015.
- [27] Z. Zhou *et al.*, “Spikformer: when spiking neural network meets transformer,” in *Proc. of ICLR*, 2023.
- [28] M. Dampfhofer, T. Mesquida, A. Valentian, and L. Anghel, “Backpropagation-based learning techniques for deep spiking neural networks: a survey,” *IEEE TNNLS*, vol. 35, no. 9, 2024.
- [29] W. Fang *et al.*, “SpikingJelly: an open-source machine learning infrastructure platform for spike-based intelligence,” *Science Advances*, vol. 9, no. 40, p. eadi1480, 2023.
- [30] J. K. Eshraghian *et al.*, “Training spiking neural networks using lessons from deep learning,” *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.
- [31] A. Zhang *et al.*, “Low latency and sparse computing spiking neural networks with self-driven adaptive threshold plasticity,” *IEEE TNNLS*, pp. 1–12, 2023.
- [32] S. Li *et al.*, “Joint threshold learning convolutional networks for intelligent fault diagnosis under nonstationary conditions,” *IEEE TIM*, vol. 72, pp. 1–11, 2023.
- [33] H. Sun *et al.*, “A synapse-threshold synergistic learning approach for spiking neural networks,” *IEEE TCDS*, vol. 16, no. 2, pp. 544–558, 2024.
- [34] A. Valentian *et al.*, “Fully integrated spiking neural network with analog neurons and RRAM synapses,” in *Proc. of IEDM*, 2019.
- [35] S. Hwang *et al.*, “Impact of the sub-resting membrane potential on accurate inference in spiking neural networks,” *Scientific Reports*, vol. 10, no. 1, p. 3515, 2020.
- [36] J. Kim *et al.*, “Design of a 180nm CMOS neuron circuit with soft-reset and underflow allowing for loss-less hardware spiking neural networks,” *Advanced Intelligent Systems*, vol. 6, no. 1, p. 2300460, 2024.
- [37] K. Park, S. Kim, M.-H. Oh, and W. Y. Choi, “Resting-potential-adjustable soft-reset integrate-and-fire neuron model for highly reliable and energy-efficient hardware-based spiking neural networks,” *Neurocomputing*, vol. 590, p. 127762, 2024.
- [38] B. Choi *et al.*, “AR-CIM: A 460.22 TOPS/W SRAM-based analog reconfigurable computing-in-memory macro with 1/2/4/8-bit variable precision,” in *Proc. of ESSERC*, 2024, pp. 361–364.
- [39] arm, <https://developer.arm.com/Architectures/AMBA>, accessed 8 Sep. 2025.
- [40] Berkeley, <https://github.com/chipsalliance/rocket-chip>, accessed 8 Sep. 2025.
- [41] B. Jacob *et al.*, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proc. of CVPR*, 2018, pp. 2704–2713.
- [42] Synopsys, <https://www.synopsys.com>, accessed 8 Sep. 2025.
- [43] Cadence, <https://www.cadence.com>, accessed 8 Sep. 2025.
- [44] AMD, <https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/artix-ultrascale-plus.html>, accessed 8 Sep. 2025.
- [45] Neuromorphic Processor eXpress, <https://riscvexpress.github.io/npx>, accessed 8 Sep. 2025.