

An FPGA-Based Spiking Neural Network for Spatiotemporal Pattern Learning and Classification

Jingang Jin, Yue Ma, Zhenhang Zhang and Qinru Qiu
Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, New York, USA
 {jjin24, yma183, zzhan281, qiqiu}@syr.edu

Abstract—Spiking Neural Networks (SNNs) are promising for energy-efficient spatiotemporal pattern recognition at the edge, but efficient deployment and on-device adaptation require specialized hardware. In this work, we present an FPGA implementation of multilayer SNNs for spatiotemporal pattern classification, in which both inference and training take place on the same hardware. Our design adopts a temporal-forward-only training algorithm that avoids the memory and latency overhead associated with traditional Backpropagation Through Time (BPTT). A mixed-precision scheme balances training stability with resource efficiency, and architectural optimization maximizes hardware and memory utilization. We further identify and resolve hazards that arise when inference and learning share the same memory, including port conflicts between the forward and backward access patterns and read-after-write dependencies in floating-point potential accumulation. Experimental evaluations on MNIST and five multivariate time-series classification datasets demonstrate that, with fully on-chip training, our SNN model achieves accuracy comparable to SNN or LSTM trained with BPTT. Compared to CPU and GPU based implementations, our FPGA design delivers 2.5× to 6× higher throughput while consuming an order of magnitude less energy per sample.

Index Terms—Neuromorphic Computing, Spiking Neural Network, In-hardware learning, FPGA

I. INTRODUCTION

Driven by the emergence of cyber-physical systems in recent years, a growing number of neural network models have been integrated into edge devices, such as smart vehicles and wearable sensors. Because user behavior, environmental conditions, and even sensor characteristics vary over time and across systems, it is highly desirable for neural networks to adapt to their specific hardware and application environment through in-hardware learning after deployment, in order to maintain performance.

There is also increasing demand for edge devices to process temporal sensor readings not as static, frame-based input, but as a stream with spatiotemporal structure. Compared to buffering an entire sequence and classifying all temporal samples at once, a state-space model (e.g., a recurrent neural network (RNN)) that processes input sample-by-sample incurs much lower computational cost and latency, making it more suitable for edge applications. However, a key challenge is that RNNs are trained using Backpropagation Through Time (BPTT), which unrolls the network over time, so its memory requirement grows linearly with sequence length. Moreover, training cannot begin until the entire sequence has been received, which again causes high latency and a bursty workload.

These properties make it difficult to train RNNs on edge devices.

Spiking neural networks (SNNs) [1] [2] present a promising alternative to conventional artificial neural networks. They are extremely energy efficient due to their event-driven dynamics and sparse computation. They are also well suited for temporal pattern recognition, since each neuron maintains a hidden state, making the network inherently an RNN. Finally, recent research on three-factor Hebbian learning [30] has made it possible to train such networks for spatiotemporal classification without unrolling the network in time [31]. Hence, unlike BPTT training, where memory requirements grow with the length of the input sequence, memory requirements under these algorithms remain constant throughout training.

Given that their computational patterns differ significantly from those of conventional ANNs, SNNs are not well-suited to traditional platforms like CPUs and GPUs, whose architectures are not optimized for sparse, event-driven, fine-grained parallelism. To leverage the full advantages of SNNs, researchers have proposed several dedicated neuromorphic processors, notably IBM TrueNorth [3] and Intel Loihi [4]. Although highly optimized, these neuromorphic chips often impose limitations on data precision, neuron models, and supported learning rules. As an emerging research area, SNN structure and training algorithms are evolving rapidly. These new dataflows and computing models require specialized hardware circuit, which current neuromorphic processors cannot efficiently support without approximation and simplification.

Field-Programmable Gate Arrays (FPGAs) occupy a strategic middle ground, offering a compelling balance of flexibility and efficiency. They also provide massive parallelism similar to GPUs, but at a more fine-grained level. Researchers have proposed numerous designs for SNN acceleration on FPGAs [5] [6] [7] [8], which exhibit significantly lower energy consumption than their CPU or GPU counterparts while offering greater flexibility than ASIC-based neuromorphic processors. However, existing work primarily focuses on accelerating the inference of pre-trained SNN models, largely overlooking on-line learning scenarios that are prevalent in edge applications.

When online learning is taken into consideration, SNN implementation complexity increases substantially. Numerous variables, including parameters and neuron potentials, are shared between the forward and backward propagation stages and accessed in diverse patterns, introducing both data and re-

source (e.g., memory port) dependencies. These dependencies hinder efficient hardware utilization on FPGAs.

In this work, we present an FPGA implementation of multilayer SNNs with on-chip learning capability. Our design integrates a stateful synapse filter structure within the spiking neuron model, a temporal forward-only training algorithm that eliminates the need for BPTT, and a conservative mixed-precision scheme to achieve high energy efficiency and processing throughput. We further introduce optimization strategies, such as loop unrolling, memory partition and read/write scheduling, for computation and memory access concurrency to fully exploit FPGA hardware resources. Memory hazards that are unique to systems hosting both forward and backward propagation are also identified and resolved.

Experimental evaluations on MNIST and five multivariate time-series classification datasets demonstrate that our implementation achieves accuracy comparable to SNN or ANN trained using BPTT. Compared to CPU- and GPU-based implementations, our FPGA acceleration delivers 2.5× to 6× higher throughput at roughly one-tenth the energy per sample.

The contributions of this work are summarized as follows:

- We implement multilayer SNNs on FPGAs with in-hardware learning capability, performing both training and inference entirely on-chip for spatiotemporal stream classification.
- We propose a resource-aware loop unrolling strategy guided by the empirical observation that both resource consumption and execution cycles scale linearly with the unroll factor, and a coarse-grained pipelining strategy that overlaps weight updates with backward passes.
- We propose a BRAM partitioning and interleaving scheme to resolve memory access conflicts between forward and backward passes, and use buffer insertion to resolve Read-After-Write (RAW) hazards.
- We evaluate our design on both static image (MNIST) and multivariate time-series classification tasks, demonstrating competitive accuracy against prior on-chip learning implementations and significant efficiency gains over CPU and GPU platforms.

II. BACKGROUND AND RELATED WORKS

A. Neuron Model

The Leaky-Integrate-and-Fire (LIF) [12] model is a widely utilized spiking neuron model. Its membrane potential performs a leaky integration of the synaptic input:

$$V[t] = \lambda V[t-1] + \sum_i w_i X_i[t] - V_{th} O[t-1] \quad (1)$$

where $V[t]$ represents the membrane potential of the neuron at time t , also considered as the state of the neuron, and w_i is the synaptic weight corresponding to the i -th input, $X_i[t]$. The neuron's output, $O[t]$, is a binary value determined by the Heaviside activation function $H(\cdot)$ and the threshold V_{th} , as detailed by the following expression:

$$O[t] = H(V[t] - V_{th}) \begin{cases} 0, & V[t] \leq V_{th} \\ 1, & \text{otherwise} \end{cases} \quad (2)$$

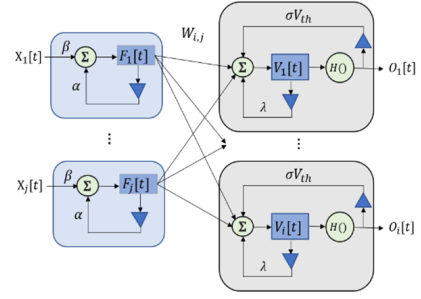


Fig. 1. Neuron model of the multilayer SNN

Recent neuroscience research [13] suggests that individual neurons' dendritic trees can perform localized computation, where a single neuron is functionally equivalent to an RNN capable of complex temporal responses. An enhanced neuron model incorporating the Post-Synaptic Potential (PSP) was proposed in [14], where PSP is defined as the output of a synapse filter governed by a generic IIR (Infinite Impulse Response) kernel with multiple forward taps and backward taps. [15] and [16] further demonstrated that competitive performance is achievable even with a first-order filter employing only one forward tap and one backward tap.

In this work, we use a first-order IIR filter for PSP. For the i -th input $X_i[t]$ at time t , the PSP of the synapse is given by the following equation:

$$F_i[t] = \alpha F_i[t-1] + \beta X_i[t]. \quad (3)$$

The coefficients α and β control the filter kernel and are shared by all neurons in our design. With this filter structure incorporated, the membrane potential $V_j[t]$ of the j -th neuron at time t is updated according to the following equation:

$$V_j[t] = \lambda V_j[t-1] + \sum_i w_{i,j} F_i[t] - \zeta V_{th} O_j[t-1]. \quad (4)$$

Here, ζ is a reset coefficient that controls the strength of the post-spike reset and is set to 0.5 in all experiments. Fig. 1 illustrates the neuron structure in one layer.

B. Online Training of SNNs

Online learning [19] and continual learning [17] are active research areas, but the common method for training SNNs poses a challenge. SNNs are often treated as a special class of RNN due to their internal memory and temporal filtering, making BPTT the standard training practice. However, BPTT's requirement to unroll the network across the entire input sequence results in computational and memory demands incompatible with resource-constrained edge devices. To overcome this limitation, learning algorithms such as SOLSA [15], OTTT [18], and e-prop learning [31] have been proposed for non-BPTT, gradient-based optimization of SNNs. These algorithms are based on similar principles that accumulate temporal gradients in a forward-only manner to avoid the need for unrolling the computation graph.

In this work, we adopt the SOLSA [15] learning algorithm. For each weight coefficient $w_{i,j}$ from pre-synaptic neuron i to

post-synaptic neuron j , SOLSA maintains a partial gradient $g_{i,j}[t]$ using the following equation:

$$g_{i,j}[t] = \sum_{t'=0}^t \mu_j[t'] \times \varepsilon_{i,j}[t'] \quad (5)$$

where $\mu_j[t]$ is the spatial (across-layer) gradient of the loss with respect to neuron j 's membrane potential, and $\varepsilon_{i,j}[t]$ is an accumulated trace on the ij -th post-synaptic potential (the ij -th input after the dendritic filter) that carries the temporal history. The trace is updated incrementally as $\varepsilon_{i,j}[t] = (\lambda - \zeta V_{th} H'(V_j[t])) \varepsilon_{i,j}[t-1] + F_i[t]$, where $H'(\cdot)$ is the surrogate derivative of the non-differentiable Heaviside firing function, taken as a Gaussian centered at V_{th} with width $\sigma = 0.5$. The spatial gradient is obtained by error backpropagation within a single time step, without unrolling over time: for an output neuron, $\mu_j[t] = (\partial \mathcal{L}[t] / \partial O_j[t]) H'(V_j[t])$; for a hidden neuron, it is propagated from the neurons k in the following layer, $\mu_j[t] = (\sum_k \mu_k[t] w_{j,k}) H'(V_j[t])$. Since $\mu_j[t]$ depends only on current-time-step quantities, it is evaluated in the same forward-only pass as $\varepsilon_{i,j}[t]$, which alone carries the temporal history.

C. FPGA implementations of SNNs

Driven by the efficiency of SNNs and the flexibility of FPGAs, many works have accelerated SNN inference on FPGA hardware. SyncNN [6] restructures event-driven SNNs into a synchronous dataflow for regular, hardware-friendly computation. FireFly v2 [11], building on its predecessor [8], exploits parallelism across input channels, output channels, pixels, and timesteps, and adds native hardware support for non-spike operations. The work in [7] co-designs the input encoding, neuron model, and architecture to achieve state-of-the-art accuracy and latency. While effective, these works deploy pre-trained SNNs on FPGAs without in-hardware learning.

Only a limited number of studies have explored on-chip SNN learning on FPGAs, and most of them rely on biologically inspired but relatively simple learning rules. CORDIC-SNN [22] implements STDP learning with Izhikevich neurons using CORDIC-based arithmetic to replace expensive multipliers with shift-and-add operations, but is limited to a two-layer topology with a single output neuron. TripleBrain [23] combines self-organizing maps with reinforcement-modulated STDP in a fully event-driven hardware core, achieving competitive accuracy on MNIST at low resource cost, yet remains restricted to a single trainable layer. These STDP-based approaches, while hardware-friendly, are inherently unsupervised or require reward signals, hence have limited applications in multilayer supervised learning.

Beyond STDP, the work in [26] takes a fundamentally different approach by employing evolutionary algorithms as a black-box training method for Izhikevich-neuron-based SNNs on a Zynq SoC, enabling continual in-field adaptation without gradient computation. However, evolutionary methods typically suffer from slow convergence and scale poorly with network size, making them impractical for larger architectures.

HaSiST [27] implements backpropagation on FPGA using a hard-sigmoid surrogate derivative, achieving high accuracy on MNIST, but the training procedure operates on a surrogate ANN that is subsequently converted to an SNN, rather than training the SNN directly. We instead train multilayer SNNs directly on FPGA with no ANN-to-SNN conversion.

III. METHODOLOGY

A. Design Rationale and Architecture Overview

Our proposed implementation comprises two core components: a datapath implemented using on-chip memory and programmable logic, and a workflow controller running on the embedded ARM processor. The controller streams in samples from the input time series and collects the classification results. It also provides labels for loss calculation during training. The datapath consists of per-layer forward propagation, backward propagation, and update modules, plus a loss calculation module. We adopted a High Level Synthesis (HLS) based design flow in this work.

SOLSA learning performs both forward and backward propagation once per input sample, doubling the volume of memory accesses compared to an inference only system. On the other hand, since it does not require unrolling the network or storing the history of neuron activities, it has much lower storage requirements than conventional learning algorithms. This allows us to train networks of decent size while constraining the neuron states and weight parameters to fit within the FPGA's on-chip Block RAM (BRAM), rather than relying on off-chip DRAM.

B. Mixed Data Precision

Prior FPGA acceleration of SNN inference exploited aggressive quantization, reducing weights to 4-bit fixed-point [6], ternary [28], or integer-only [29] representations. However, in-hardware learning imposes tighter precision requirements, as quantization errors accumulate through gradient computation and weight updates, degrading training convergence. We therefore adopt a conservative mixed-precision scheme: FP32 for neuron potential ($V_j[t]$) and FP16 for weight storage (w_{ij}). This decision is to balance the precision and memory size. Since a fully connected layer with N inputs and M neurons requires $N \times M$ weights but only M potentials, weights dominate the memory footprint. Applying FP16 to them yields a substantial reduction in both storage and resource utilization, while FP32 for the comparatively few potentials ensures stable training. For the similar reason, we adopt FP32 for F_i , $\varepsilon_{i,j}$, μ_j and $g_{i,j}$ introduced in Section II-B.

The binary nature of spikes enables additional simplification: using only 1 bit for the spike reduces the multiply-accumulate (MAC) operation on spike input to simple summations, and the spike output of the activation function can be represented by the negation of the sign bit of the difference between membrane potential and threshold.

C. Loop Unrolling

SNN computation involves extensive vector and matrix operations, which are initially implemented as nested loops in HLS. While sequential execution of these loops induces high latency, unrolling all loops for maximum parallelism is infeasible due to limited hardware resources. The challenge is to determine the optimal unroll factor for each loop under a constrained resource budget. To address this, we profile each loop by synthesizing it at different unroll factors and recording the resulting execution time (measured in clock cycles) and resource utilization. For nested loop, only inner loop is unrolled due to hardware resource limitation. The results demonstrate a linear relationship between the unroll factor and both resource consumption and execution time. Based on this profiling, two linear estimation models, $C_l(U_l)$ and $R_l(U_l)$, are trained for each loop l to predict its execution time and resource usage, respectively, under unroll factor U_l .

Guided by these models, we search for unroll factors using the greedy strategy depicted in Algorithm 1. It repeatedly increments the unroll factor of the loop with the highest estimated execution time, until the performance target is reached or no loop can be unrolled further without exceeding the hardware constraint. This directs hardware resources toward the current bottleneck at each step.

Algorithm 1 Resource-Aware Loop Unrolling

Require: Loop set L , resource budget $R_{\max}$, target latency C_{target} , Linear coefficients for $C_l(U_l)$ and $R_l(U_l)$

- 1: In Unroll Factors U , Initialize $U_l \leftarrow 1$ for each loop $l \in L$
- 2: Sort L by $C_l(U_l)$ in descending order
- 3: **while true do**
- 4: $l^* \leftarrow L[1]$
- 5: $U_{l^*} \leftarrow U_{l^*} + 1$
- 6: Estimate $C_{l^*}(U_{l^*})$ and R_{total} using linear model
- 7: **if** $\sum_{l \in L} C_l(U_l) \leq C_{\text{target}}$ **then**
- 8: Target Achieved
- 9: **break**
- 10: **end if**
- 11: **if** $R_{\text{total}} > R_{\max}$ **then**
- 12: $U_{l^*} \leftarrow U_{l^*} - 1$
- 13: Remove l^* from L
- 14: **else**
- 15: Insert $C_{l^*}(U_{l^*})$ back to the sorted list
- 16: **end if**
- 17: **end while**
- 18: **return** U

D. Coarse-grained Pipelining

In addition to the loop unrolling, we also consider coarse-grained datapath pipelining. Fig. 2 shows the data dependencies among all modules. In the training path, the forward modules execute sequentially ($L_1 \rightarrow L_2 \rightarrow L_3$), as each layer consumes the output of the previous one, and after loss calculation the backward modules execute in reverse order ($L_3 \rightarrow L_2 \rightarrow L_1$), since each layer's gradient depends on the gradient propagated from the layer above. Each layer's

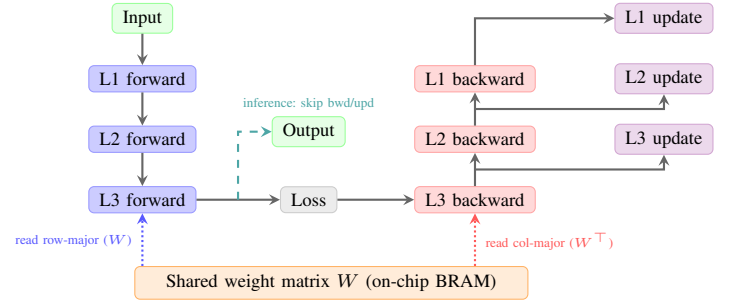


Fig. 2. Architecture and data flow of the multilayer SNN FPGA implementation. The forward and backward passes read the *same* on-chip weight matrix in conflicting orders—row-major (W) versus column-major (W^T)—motivating the partitioning and interleaving scheme.

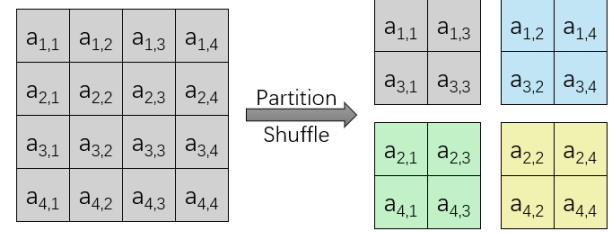


Fig. 3. Weight matrix partition and shuffle

weight update, however, depends only on its own backward module, hence can be executed in parallel with the backward module of the next layer as shown in Fig. 2.

The module-level overlap is fixed at synthesis time by the HLS dataflow structure. Unlike an inference-only system, where the entire forward propagation path can be fully pipelined, the backward path's data dependencies limit the opportunity for coarse-grained pipelining in our design. However, hardware resources can still be fully utilized through loop unrolling.

E. Memory Partitioning

There are some unique hazards associated to running both inference and learning on the same hardware. First, the forward and backward passes access the same weight matrix, but in conflicting orders: the forward path reads it row-major, while the backward path reads it column-major. This conflict is exacerbated by loop unrolling. Each BRAM block has only two I/O ports, so with an unroll factor of k , the module must fetch k weights per cycle; if those k elements fall within the same block, the accesses are serialized, negating the benefit of unrolling. Because the two passes require opposing access patterns, no single memory layout can eliminate conflicts for both simultaneously. A layout optimized for the forward pass reintroduces conflicts for the backward pass, and vice versa. An inference-only design, which performs only the forward pass, never encounters this conflict.

We resolve this hazard with memory partitioning and interleaving. The weight matrix is divided into $k \times k$ tiles, each mapped to a separate BRAM block with independent I/O ports; element $w_{i,j}$ is assigned to tile $[i \bmod k, j \bmod k]$, as illustrated in Fig. 3. Any k consecutive elements along either a row or a column are thus distributed across k distinct blocks,

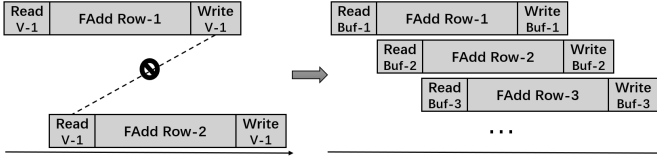


Fig. 4. Resolution of RAW hazard with buffer

so both the forward and backward passes fetch k elements in a single cycle without port conflicts.

F. Buffer Insertion

The second hazard unique to supporting on-chip learning is the Read-After-Write (RAW) dependency in the accumulating floating point values, such as $V_j = \sum_i w_{i,j} F_i[t]$. Learning on-chip requires high precision membrane potentials.

Single precision (FP32) floating point adder on Xilinx FPGA typically has a pipeline latency of 3–5 cycles. As shown in Fig. 4 (a), every iteration updates the same accumulator, iteration k must read the value written by iteration $k-1$; and because the floating-point adder has a multi-cycle latency L , that value is not ready in the next L cycles. Such RAW hazard will not occur in inference only system where fixed-point precision is typically used.

HLS automatically resolves such RAW hazards by raising the pipeline’s initiation interval (II) of the pipeline to L . However, this serializes the accumulation and negates the benefit of pipelining. We remove this dependency using an intermediate buffer that splits the accumulator into P partial sums: iteration k accumulates into line $(k \bmod P)$, so each line is updated only once every P iterations (Fig. 4 (b)). The dependency distance per line is then P cycles at $\text{II} = 1$, and choosing $P \geq L$ sustains $\text{II} = 1$ regardless of the reduction length. Since the measured latency of the floating point adder is about four cycles, four buffer lines suffice. A final step sums the partial sums into the potential at negligible cost.

IV. EXPERIMENTAL RESULTS

The experiments were conducted using the AMD ZCU102 Evaluation Board. This platform features a heterogeneous System-on-Chip that integrates a quad-core ARM Cortex-A53 processor with FPGA programmable logic. We developed the hardware implementation using the AMD Vivado Design Suite and Vitis Core Development Kit (version 2024.2).

A. Datasets and Hyper Parameters

We evaluated our proposed method on six classification datasets spanning both static image and multivariate time-series. Table I summarizes the key information of these datasets. MNIST is a widely used benchmark of handwritten digit images (28×28 grayscale), included to enable direct comparison with prior FPGA-based on-chip learning implementations. The remaining five datasets are multivariate time-series from the UEA archive [20], spanning motion sensing, biomedical signal processing, and physiological monitoring applications representative of edge computing scenarios. For SOLSA learning, we set both α and β in the PSP filter to be 0.9 and λ to be 0.5. The learning rate is set to be 0.0001.

TABLE I
DATASET INFORMATION

Dataset	Input Dim	Length	Classes	Type
MNIST	784	1	10	Static
BasicMotions[20]	6	100	4	Temporal
Epilepsy[24]	3	207	4	Temporal
FingerMovements[25]	28	50	2	Temporal
Racket Sports[20]	6	30	4	Temporal
Japanese Vowels[21]	12	29	9	Temporal

B. Classification Accuracy

We first compare against two prior FPGA-based implementations with on-chip learning capability: HaSiST [27] and TripleBrain [23] for their performance and hardware usage on the MNIST classification. Table II summarizes the results.

Our on-chip learning achieves 95.92% accuracy, surpassing TripleBrain’s 95.10%. This accuracy improvement can be attributed to our multilayer architecture with gradient-based learning, which enables richer feature representations than TripleBrain’s single-layer STDP approach. While HaSiST achieves higher accuracy at 97.5%, its training procedure operates on a surrogate ANN in full precision on a host computer, with trained weights subsequently transferred to the FPGA. Their in-hardware training engine can only handle 8×8 input. In contrast, our design performs both training and inference entirely on-chip in mixed precision, which is essential for edge scenarios requiring real-time adaptation without access to external computing resources.

For temporal datasets, we benchmark against the SNN models reported in [7] and [16]. The work in [7] trains SNN offline using BPTT and deploys inference on FPGA, while [16] implements SOLSA training on CPU/GPU. Both of them train the PSP kernel coefficients (α and β) in addition to the synapse weights (w_{ij}). To ensure a fair comparison, all models employ a three-layer structure ($D_{in} - 100 - 100 - D_{out}$), where D_{in} and D_{out} denote the input and output dimensions of the dataset, respectively. As a reference, an LSTM model is also trained, with its performance reported as a baseline representative of conventional artificial neural networks. Table III summarizes the results.

The results show a modest accuracy gap between our on-FPGA training and the SNNs trained on CPU/GPU. The difference can be attributed to the mixed-precision scheme and the adoption of a fixed PSP kernel. It also shows that the on-FPGA trained SNN performs comparable or even slightly better than the LSTM model.

C. Efficiency Comparison

In this section, we evaluate the hardware efficiency, including throughput, clock frequency, and energy consumption, of our FPGA-based SNN against the baseline SNN model implemented on both CPU and GPU. The efficiency comparison is conducted on the temporal datasets, representative of typical sensor data rates in edge computing applications. The experimental setup included an Intel Core i7 8700 CPU and an NVIDIA RTX 2080 Ti GPU. To ensure a fair comparison, all selected platforms are manufactured using comparable semiconductor process nodes (the ZCU102 uses a 16nm FinFET

TABLE II
MNIST ACCURACY COMPARISON AMONG FPGA-BASED ON-CHIP LEARNING IMPLEMENTATIONS

Impl.	Network	Synapses	Timesteps	FPGA	LUT	FF	DSP	BRAM	Acc.
HaSiST* [27]	784-200-10	158,800	15	Virtex-6	37.84 per synapse	2.63 per synapse	—	—	97.5%
TripleBrain [§] [23]	784-256	200,704	200	Zynq-7045	16,584 [†]	33,168 [†]	32	131	95.10%
Ours	784-100-100-10	89,400	10	ZCU102	87,577	63,642	712	202	95.92%

*HaSiST trains a surrogate ANN and transfers weights to SNN for inference. DSP and BRAM are not reported.

[§]TripleBrain use SOM-STDP and R-STDP. Resource reported as slices; TripleBrain uses a single-layer SNN with no multilayer gradient-based training.

[†]Converted from reported slices (Zynq-7000: 4 LUTs and 8 FFs per slice); upper-bound estimates.

TABLE III
TEMPORAL DATASET ACCURACY COMPARISON

Impl.	RS	JV	BM	Epi	FM
[7] [†]	0.869	0.969	1	0.961	0.62
[16] [‡]	0.864	0.973	1	0.965	0.612
LSTM	0.877	0.935	0.925	0.839	0.625
Ours	0.842	0.959	1	0.956	0.61

RS=Racket Sports, JV=Japanese Vowels, BM=BasicMotions, Epi=Epilepsy, FM=FingerMovements.

[†]Trained offline with BPTT; inference on FPGA.

[‡]Trained and evaluated on CPU/GPU.

TABLE IV
EFFICIENCY COMPARISON

Platform	Samples/s	Clock (MHz)	Energy (mJ/sample)
FPGA (ZCU102)	~50,000	125	0.5
GPU (2080 Ti)	~20,000	1,350	5
CPU (Core i7)	~8,000	3,600	8.125

Samples/s represents training throughput (forward + backward + update).

As illustrated in Fig. 5, resource consumption scales linearly with throughput for most metrics, and nearly all data points lie on the Pareto front. The only exception is BRAM usage, which remains constant once throughput drops below a certain threshold, corresponding to a set of low unroll factors. In this regime, the number of BRAMs required is bounded by the amount of data that must be stored, a fixed value that cannot be further reduced through scaling. These results demonstrate the high flexibility of our architecture in terms of performance-hardware tradeoff. As a result, it can be seamlessly migrated to smaller, cost-effective FPGAs for lower-throughput tasks without requiring architectural modifications.

V. CONCLUSION

We presented an FPGA implementation of a multilayer SNN that performs both inference and online learning entirely on-chip. By combining a temporal forward-only training algorithm with a conservative FP32/FP16 mixed-precision scheme and a set of architecture-level optimizations—resource-aware loop unrolling, coarse-grained pipelining, and BRAM partitioning with hazard-resolving buffers—the design sustains gradient-based learning under tight edge resource budgets. On MNIST it reaches 95.92% accuracy with fully on-chip training, surpassing TripleBrain (95.10%) and approaching the off-chip-trained HaSiST (97.5%), and stays within 3% of BPTT-trained baselines on five temporal datasets. It delivers 2.5× to 6× the throughput of CPU and GPU platforms at roughly one-tenth the energy per sample, with headroom for real-time edge learning. The linear scaling of resource usage lets the same architecture migrate across FPGA capacities without modification.

REFERENCES

- [1] Tavanaei, Amirhossein, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. "Deep learning in spiking neural networks." *Neural networks* 111 (2019): 47-63.
- [2] Ghosh-Dastidar, Samanwoy, and Hojjat Adeli. "Spiking neural networks." *International journal of neural systems* 19, no. 04 (2009): 295-308.

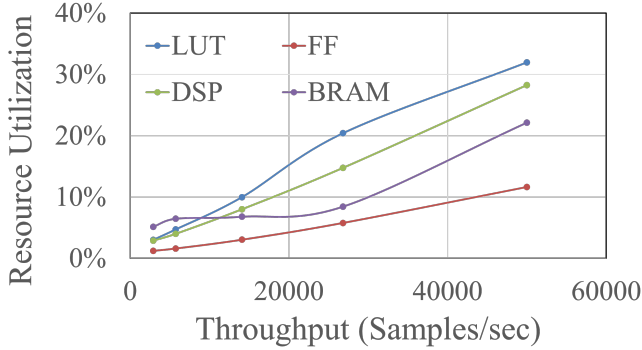


Fig. 5. Hardware resource utilization(Racket Sports Dataset)

process, the CPU uses 14nm, and the GPU uses 12nm), minimizing the influence of fabrication technology on the efficiency metrics.

As detailed in Table IV, our FPGA implementation achieves 2.5× to 6× throughput improvement for training while operating at a significantly lower clock frequency. Furthermore, it requires only 0.5 mJ per sample, an order of magnitude below the 5 and 8.125 mJ of the GPU and CPU baselines. Notably, the achieved throughput of 50,000 samples per second provides 5× to 50× headroom over the real-time sampling rates of the evaluated sensor datasets, confirming that our design can comfortably sustain online learning at real-time data rates in edge deployment scenarios.

D. Hardware Resource Utilization

In this section, we demonstrate the effectiveness of our loop-unrolling and memory management techniques and show that they produce Pareto optimum design points in resource and performance space. We varied the C_{target} and R_{max} to generate different loop unrolling strategies using Algorithm 1 to target distinct performance requirements. We then report the consumption of four primary resource types: Look-Up Tables (LUTs), Flip-Flops (FFs), Digital Signal Processing units (DSPs), and Block RAM (BRAM).

- [3] Akopyan, Filipp, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam et al. "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip." *IEEE transactions on computer-aided design of integrated circuits and systems* 34, no. 10 (2015): 1537-1557.
- [4] Davies, Mike, Narayan Srinivasa, Tsung-Han Lin, Gautham China, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou et al. "Loihi: A neuromorphic manycore processor with on-chip learning." *Ieee Micro* 38, no. 1 (2018): 82-99.
- [5] Khodamoradi, Alireza, Kristof Denolf, and Ryan Kastner. "S2n2: A fpga accelerator for streaming spiking neural networks." In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 194-205. 2021.
- [6] Panchapakesan, Sathish, Zhenman Fang, and Jian Li. "SyncNN: Evaluating and accelerating spiking neural networks on FPGAs." *ACM Transactions on Reconfigurable Technology and Systems* 15, no. 4 (2022): 1-27.
- [7] Fang, Haowen, Zaidao Mei, Amar Shrestha, Ziyi Zhao, Yilan Li, and Qinru Qiu. "Encoding, model, and architecture: Systematic optimization for spiking neural network in FPGAs." In *Proceedings of the 39th international conference on computer-aided design*, pp. 1-9. 2020.
- [8] Li, Jindong, Guobin Shen, Dongcheng Zhao, Qian Zhang, and Yi Zeng. "Firefly: A high-throughput hardware accelerator for spiking neural networks with efficient dsp and memory optimization." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 31, no. 8 (2023): 1178-1191.
- [9] Werbos, Paul J. "Generalization of backpropagation with application to a recurrent gas market model." *Neural networks* 1, no. 4 (1988): 339-356.
- [10] Robinson, Anthony J., and Frank Falls. *The utility driven dynamic error propagation network*. Vol. 11. Cambridge: University of Cambridge Department of Engineering, 1987.
- [11] Li, Jindong, Guobin Shen, Dongcheng Zhao, Qian Zhang, and Yi Zeng. "Firefly v2: Advancing hardware support for high-performance spiking neural network with a spatiotemporal fpga accelerator." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, no. 9 (2024): 2647-2660.
- [12] Tal, Doron, and Eric L. Schwartz. "Computing with the leaky integrate-and-fire neuron: logarithmic computation and multiplication." *Neural computation* 9, no. 2 (1997): 305-318.
- [13] Jones, Ilenna Simone, and Konrad Paul Kording. "Might a single neuron solve interesting machine learning problems through successive computations on its dendritic tree?." *Neural Computation* 33, no. 6 (2021): 1554-1571.
- [14] Fang, Haowen, Amar Shrestha, Ziyi Zhao, and Qinru Qiu. "Exploiting neuron and synapse filter dynamics in spatial temporal learning of deep spiking neural network." *arXiv preprint arXiv:2003.02944* (2020).
- [15] Zhang, Zhenhang, Jingang Jin, Haowen Fang, and Qinru Qiu. "SOLSA: Neuromorphic Spatiotemporal Online Learning for Synaptic Adaptation." In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 848-853. IEEE, 2024.
- [16] Jin, Jingang, Zhenhang Zhang, and Qinru Qiu. "Exploring Spike Encoder Designs for Near-Sensor Edge Computing." In *2025 Neuro Inspired Computational Elements (NICE)*, pp. 1-9. IEEE, 2025.
- [17] Ma, Yue, Huantao Ren, Boyu Wang, Jingang Jin, Senem Velipasalar, and Qinru Qiu. "LVP-CLIP: Revisiting CLIP for Continual Learning with Label Vector Pool." In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 231-240. 2025.
- [18] Xiao, Mingqing, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. "Online training through time for spiking neural networks." *Advances in neural information processing systems* 35 (2022): 20717-20730.
- [19] Jain, Lakhmi C., Manjeevan Seera, Chee Peng Lim, and Pagavathigounder Balasubramaniam. "A review of online learning in supervised neural networks." *Neural computing and applications* 25, no. 3 (2014): 491-509.
- [20] The Website of Time Series Classification Repository, timeseriesclassification.com/dataset.php, Accessed 21 Nov 2025.
- [21] Kudo, M., Toyama, J., Shimbo, M. (1999). Japanese Vowels [Dataset]. UCI Machine Learning Repository. <https://doi.org/10.24432/C5NS47>.
- [22] Heidarpur, Moslem, et al. "CORDIC-SNN: On-FPGA STDP learning with izhikevich neurons." *IEEE Transactions on Circuits and Systems I: Regular Papers* 66.7 (2019): 2651-2661.
- [23] Wang, Haibing, et al. "TripleBrain: A compact neuromorphic hardware core with fast on-chip self-organizing and reinforcement spike-timing dependent plasticity." *IEEE Transactions on Biomedical Circuits and Systems* 16.4 (2022): 636-650.
- [24] Villar, Jose R., et al. "Generalized models for the classification of abnormal movements in daily life and its applicability to epilepsy convulsion recognition." *International journal of neural systems* 26.06 (2016): 1650037.
- [25] Blankertz, Benjamin, Gabriel Curio, and Klaus-Robert Müller. "Classifying single trial EEG: Towards brain computer interfacing." *Advances in neural information processing systems* 14 (2001).
- [26] Otero, Andrés, et al. "Evolutionary fpga-based spiking neural networks for continual learning." *International Symposium on Applied Reconfigurable Computing*. Cham: Springer Nature Switzerland, 2023.
- [27] Siddique, Ali, Mang I. Vai, and Sio Hang Pun. "A low cost neuromorphic learning engine based on a high performance supervised SNN learning algorithm." *Scientific Reports* 13.1 (2023): 6280.
- [28] Li, Jiahao, et al. "Balancing accuracy and efficiency: co-design of hybrid quantization and unified computing architecture for spiking neural networks." *Frontiers in Neuroscience* 19 (2025): 1665778.
- [29] Liu, Hanwen, et al. "HardF-SNN: Hardware-Friendly Quantization for Spiking Neural Networks with Efficient Integer-Arithmetic-Only Inference." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 40. No. 3. 2026.
- [30] Frémaux, Nicolas, and Wulfram Gerstner. "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules." *Frontiers in neural circuits* 9 (2016): 85.
- [31] Gerstner, Wulfram, et al. "Eligibility traces and plasticity on behavioral time scales: experimental support of neohbbian three-factor learning rules." *Frontiers in neural circuits* 12 (2018): 53.