POLITECNICO DI TORINO
Repository ISTITUZIONALE

SFATTI: Spiking FPGA Accelerator For Temporal Task-Driven Inference - a Case Study on Mnist

Original
SFATTI: Spiking FPGA Accelerator For Temporal Task-Driven Inference - a Case Study on Mnist / Caviglia, A., Marostica, F., Carpegna, A., Savino, A., Di Carlo, S.. - ELETTRONICO. - (2025), pp. 59-64. (2025 IEEE International Conference on Image Processing Workshops (ICIP) Anchorage, AK, USA 14-17 September 2025) [10.1109/icipw68931.2025.11385983].

Availability:
This version is available at: 11583/3007964 since: 2026-02-24T10:03:45Z

Publisher:
IEEE

Published
DOI:10.1109/icipw68931.2025.11385983

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

19 November 2026

SFATTI: SPIKING FPGA ACCELERATOR FOR TEMPORAL TASK-DRIVEN INFERENCE - A CASE STUDY ON MNIST

Alessio Caviglia, Filippo Marostica, Alessio Carpegna, Alessandro Savino, Stefano Di Carlo

⋆ Politecnico di Torino,
Control and Computer Engineering Department, Torino, Italy,
e-mail:{firstname.lastname}@polito.it

ABSTRACT

Hardware accelerators are essential for achieving low-latency, energy-efficient inference in edge applications like image recognition. Spiking Neural Networks (SNNs) are particularly promising due to their event-driven and temporally sparse nature, making them well-suited for low-power Field Programmable Gate Array (FPGA)-based deployment. This paper explores using the open-source `Spiker+` framework to generate optimized SNNs accelerators for handwritten digit recognition on the MNIST dataset. `Spiker+` enables high-level specification of network topologies, neuron models, and quantization, automatically generating deployable HDL. We evaluate multiple configurations and analyze trade-offs relevant to edge computing constraints.

***Index Terms*—** FPGA, Spiking Neural Networks, MNIST, Edge Computing, Neuromorphic Computing

1. INTRODUCTION

Edge computing devices are increasingly leveraging Artificial Neural Networks (ANNs) to perform image processing directly at the data source. By enabling local inference, these systems reduce latency, lower energy consumption, and minimize dependence on remote servers. Processing visual data on-device not only enhances privacy but also supports real-time decision-making, key requirements for applications constrained by tight power budgets and timing deadlines.

Among various ANN paradigms, Spiking Neural Networks (SNNs) have emerged as particularly promising for edge applications due to their biologically inspired, event-driven computation [1]. Unlike traditional ANNs that rely on continuous activations and dense matrix multiplications, SNNs encode and process information as discrete spike events occurring over time, mimicking the sparse and temporally asynchronous communication of biological neurons. SNNs are well-suited for dynamic, temporally rich data due to their energy-efficient, sparse, and event-driven computation. They reduce memory and compute demands, making them ideal for audio processing, image processing, gesture recognition, and sensor fusion tasks. To process continuous-valued input, SNNs require encoding schemes that convert signals into spike trains [2].

Efficient deployment of SNNs on embedded hardware platforms poses additional complexity. Conventional hardware accelerators, optimized for dense numerical operations and synchronous computation, are ill-suited to the sparse, event-driven, and temporally dynamic nature of SNNs [3]. As a result, specialized neuromorphic hardware accelerators have been developed to exploit these unique characteristics. Examples include IBM's TrueNorth ASIC [4], which implements large-scale digital SNNs with extremely low power consumption, and Intel's Loihi processor [5], which supports on-chip learning and programmable synaptic plasticity. While these platforms demonstrate the feasibility of energy-efficient spike-based computation, they often lack integration flexibility and impose constraints on network structure and learning rules. Alternatively, Field Programmable Gate Array (FPGA)-based approaches have emerged as adaptable solutions capable of emulating diverse SNN models with customized architectures [6, 7, 8, 9, 10, 11, 12]. However, existing implementations frequently involve significant manual design effort, lack modularity, or are tightly coupled to specific toolchains and hardware configurations, hindering their usability and scalability.

In this paper, we present our contribution to the Digit Recognition Low Power and Speed Challenge @ ICIP 2025[1], which focuses on evaluating FPGA-based accelerators in terms of inference speed, energy efficiency, and classification accuracy on the MNIST dataset [13]. Our approach is centered around `Spiker+` [14], an open-source, end-to-end framework for the design, training, and deployment of SNNs on FPGA platforms. Unlike traditional fixed-function accelerators, `Spiker+` enables the automatic generation of optimized hardware tailored to specific tasks. This flexibility allows us to synthesize an accelerator specifically optimized for handwritten digit recognition, considering the trade-offs

THIS WORK WAS PARTIALLY SUPPORTED BY PROJECT SERICS (PE00000014) UNDER THE MUR NATIONAL RECOVERY AND RESILIENCE PLAN FUNDED BY THE EUROPEAN UNION.

[1] `https://mlunglma.github.io/challenge.html`

between speed, power, and accuracy.

The framework supports high-level SNN modeling in Python, integrates surrogate-gradient-based training through `snnTorch` [15], and performs automatic quantization to match hardware constraints. It then generates synthesizable HDL, streamlining deployment on target FPGA platforms. This seamless integration of training and hardware generation enables rapid design-space exploration and functional prototyping, also supported by companion tools such as SpikeExplorer [16] and `Optuna` [17]. In the context of the challenge, we demonstrate that using `Spiker+` we were able to effectively co-optimize both the neural network architecture and its hardware implementation, achieving high accuracy while meeting strict energy and latency requirements. Our results validate the viability of task-specific, reconfigurable SNN accelerators for low-power edge image processing applications.

2. METHODS

Figure 1 sketches the entire flow required to start from the MNIST database and to obtain a highly optimized SNN accelerator for digit recognition to be deployed on FPGAs through the use of the `Spiker+` framework, an open-source framework, publicly available on GitHub. To facilitate full reproducibility of the experiments presented in this paper, we prepared a dedicated repository containing detailed implementation instructions and supplementary materials. This repository is accessible at `https://github.com/smilies-polito/sfatti`

2.1. Spike Encoding

When processing non-neuromorphic datasets such as MNIST, SNNs require an input encoding stage to transform static, continuous-valued data into spike trains. This transformation is necessary to interface conventional data formats with the discrete, event-driven dynamics of spiking neurons. Rate coding is the most widely adopted encoding scheme when dealing with static data, such as images [2]. In rate coding, pixel magnitude is mapped to spike frequency within a fixed temporal window, offering robustness and simplicity. In our architecture, input encoding is handled via Poisson-based rate coding, realized using the `snntorch.spikegen` module. Each pixel intensity is interpreted as a firing probability at every timestep over a window of a fixed number of steps. This encoding procedure is applied before submitting the images to the accelerator and is also integrated into the training pipeline through a dedicated preprocessing class. Importantly, no additional encoding is required for inherently event-based inputs, such as those from neuromorphic sensors like Dynamic Vision Sensor (DVS) cameras, as the data are already represented in spike form.

2.2. Network Training

Training SNNs presents notable challenges primarily due to the discrete, non-differentiable nature of spike generation. This characteristic violates the assumptions underlying standard gradient descent methods, such as backpropagation, which rely on the availability of continuous and differentiable activation functions [18]. Consequently, direct application of gradient-based optimization to SNNs is infeasible. To address this, surrogate gradient methods have been introduced [19]. These methods approximate the discontinuous spike activation functions with smooth, differentiable surrogates during the backward pass of Back-Propagation Through Time (BPTT) [20], while preserving the original spike behavior during the forward pass [21]. This enables effective supervised training of SNNs using conventional optimization algorithms and loss functions. Recent advances have demonstrated that surrogate-gradient training can achieve performance comparable to traditional ANNs on standard benchmarks [1], closing the gap between bio-plausibility and practical efficacy. In our work, the training phase is handled entirely in software using the `SnnTorch` framework [18], which is fully integrated with `Spiker+`. `SnnTorch` supports surrogate-gradient learning, specifically BPTT with differentiable approximations of the spike function, and is used to optimize all synaptic weights and neuron thresholds in floating-point precision on a host CPU or GPU.

To ensure highly efficient hardware acceleration, `Spiker+` imposes a hardware-friendly constraint on the neuron model used during training. Specifically, `Spiker+` employs a standard Leaky Integrate and Fire (LIF) neuron model, where the membrane potential decays exponentially over time. This decay is governed by two parameters, α and β, which control the rate at which the potential and synaptic current diminish. In traditional implementations, these exponential decays would require multiplication operations, which are costly regarding hardware resources. To eliminate the need for multipliers and simplify the digital logic, `Spiker+` requires rounding these parameters to the nearest power of two during training. This allows the corresponding decay operations to be implemented as simple binary shifts on the FPGA, significantly reducing area and power consumption. While this approximation introduces a slight degradation in classification accuracy, the trade-off is justified by the substantial improvements in hardware efficiency. Once training is completed with these quantized decay parameters, the learned model is passed to the quantization module for further refinement and synthesis, ensuring that the final implementation meets the design constraints of the target FPGA platform.

To ensure reproducibility across environments, we fixed the random seed for all relevant components, including `Python`, `NumPy`, and `PyTorch`. This guarantees consistent behavior across runs, essential for two reasons: the spike encoding process uses Poisson-based rate coding, intro-

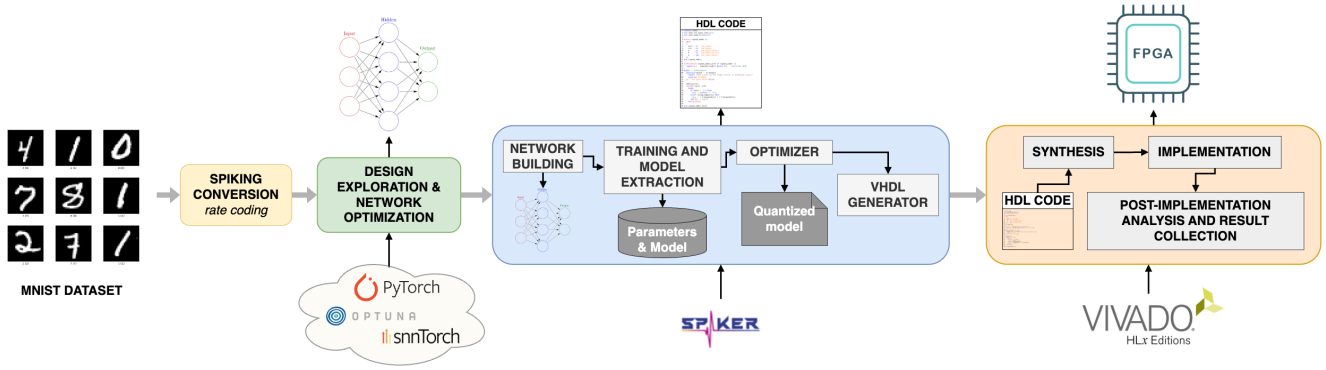


Fig. 1: Workflow scheme

ducing randomness in input generation; and training involves stochastic operations such as data shuffling and weight initialization. Even though the seed is fixed, results may still vary slightly due to nondeterministic operations in underlying libraries or differences in hardware backends. As highlighted in the PyTorch documentation [22], perfect reproducibility across all platforms and configurations cannot be guaranteed.

2.3. High-Level Design Space Exploration

Before committing to hardware synthesis, we performed a high-level exploration of the design space using purely software-based tools. This phase relied on `SnnTorch` for training and inference, and `Optuna` [17] for large-scale hyperparameter optimization. This initial analysis focused on identifying promising network configurations from an algorithmic standpoint. Various architectures, differing in neuron types, layer sizes, spiking conversion time-steps, and other training-specific parameters, were evaluated to assess their accuracy, trying to minimize those parameters that influence power and latency. The goal was to rapidly discard configurations that failed to meet target requirements through a computationally efficient approach that aimed at surveying the design space and identifying viable candidate networks.

2.4. Quantization Exploration with Spiker+

Following the high-level screening, we refined the design space using the quantization and hardware-aware simulation capabilities of `Spiker+`. The framework includes a quantization optimizer that reduces memory and computational costs by exploring efficient numerical formats for weights, membrane potentials, and fractional precision. It uses signed two's complement fixed-point arithmetic with overflow protection via value clipping. The optimizer accepts search ranges for bit widths (e.g., weights: 4–10 bits) and performs a combinatorial sweep, evaluating each configuration by running post-quantization inference and recording the resulting accuracy.

Along with the standard exploration used by `Spiker+`, we modified the framework to enable automated batch evaluations, logging all configurations and corresponding results into structured files. This eliminated the need for manual selection and ensured reproducibility, together with a faster study of the behaviour of many configurations. For each viable network architecture (e.g., fixed neuron model, variable layer sizes), we executed a full set of quantized simulations and applied application-specific constraints, such as a minimum test accuracy of 97.5%, to filter the results. This approach enabled a comprehensive yet automated exploration of trade-offs between accuracy, latency, and hardware cost, significantly accelerating the design of efficient and performant SNN accelerators.

2.5. HDL Generation

After optimizing all design parameters, `Spiker+` is used to generate a synthesizable hardware description of the target accelerator. To achieve this goal, `Spiker+` accepts the network architecture and trained and quantized network parameters, producing VHSIC Hardware Description Language (VHDL) source code describing the complete accelerator. This includes neuron modules, synaptic memory blocks, control logic, and I/O interfaces. `Spiker+` supports a full II-order LIF neuron model, where both α and β are non-zero. However, by adjusting the neuron parameters, it is also possible to implement simplified I-order LIF neurons ($\alpha = 0$), or even basic Integrate and Fire (IF) neurons by setting both α and β to zero.

`Spiker+` supports two generation modes: (i) simulation and verification, and (ii) direct deployment to physical FPGAs. In our case, we selected the deployable mode, which configures the memory blocks to match the on-chip BRAM of the target FPGA and sets all timing parameters for real-time operation. As mentioned before, the generated architecture avoids multipliers by approximating exponential decays with bit-shift operations (enabled by the power-of-two rounding of α, β), and processes input spikes sequentially to mit-

Table 1: Summary of SNNs configurations and hardware evaluation results. The network architecture summarizes the number of neurons per layer. The architecture is a feed-forward, fully connected network. Quantization is shown as weights bit-width (WB), membrane bit-width (MB), and fixed-point decimal bits (FPd). Utilization is shown as a percentage of used resources with respect to the total of Look Up Tables (LUTs), Flip Flops (FFs), and Block RAMs (BRAMs)

Neuron	Model	Quantization			Accuracy (%)	Power (mW)	Timesteps	Utilization (%)			Clock (MHz)	(Img/s)/W
		WB	MB	FPd				LUTs	FFs	BRAMs		
I-order LIF	784-25-10	4	4	4	73.50	162	10	1.92	0.95	1.23	188.6	/
		10	10	6	96.19	187	10	2.44	1.05	2.62	166.6	/
I-order LIF	784-50-10	4	4	4	83.29	177	10	2.31	1.06	2.15	188.6	/
		10	10	6	97.16	222	10	3.28	1.24	4.77	161.2	/
I-order LIF	784-75-10	4	4	4	86.26	191	10	2.74	1.17	2.92	185.2	/
		6	9	5	97.54	231	10	3.93	1.38	4.15	163.9	81.712,4
		10	10	6	97.86	254	10	4.13	1.43	6.92	153.8	69692.9
I-order LIF	784-100-10	4	4	4	73.53	202	10	3.15	1.28	3.85	172.4	/
		6	9	5	97.59	251	10	4.71	1.56	5.54	156.2	71696.4
		10	10	4	97.78	285	10	4.95	1.61	9.08	151.5	61227.4

igate bandwidth and memory limitations typical of low-end FPGAs. This results in compact and power-efficient implementations, as demonstrated in our evaluation.

2.6. Implementation and Results Gathering

The VHDL project generated by `Spiker+` was integrated into a Vivado project . The desgined was then synthesized and mapped to Xilinx Kintex XC7K160TFBG484-1 FPGA [23]. The final deployment requires two key steps. While `Spiker+` automatically instantiates all required Read Only Memorys (ROMs) in the design, the user is responsible for initializing their content. This is achieved through `.coe` files generated by `Spiker+`, which must be loaded into the memory generator Intellectual Propertys (IPs) within the Vivado design environment. Additionally, to correctly map the accelerator's I/O interfaces to the physical pins of the FPGA, the user must supply a custom `XDC` file. This file defines the clock and reset sources, along with the placement of data and control signals on the target device. After synthesizing and implementing the design, the bitstream is loaded onto the FPGA. During runtime, key performance metrics such as maximum clock speed, power consumption, and inference speed are collected. These metrics are correlated with the Design Space Exploration (DSE) results to validate the effectiveness of the automated design methodology.

3. RESULTS

Table 1 summarizes the DSE conducted to implement efficient SNN-based digit recognition on FPGA. The row highlighted in gray indicates the architecture with the highest energy efficiency, measured in images per second per watt, which was ultimately selected as our candidate design for the challenge.

All models were trained using surrogate gradient descent; each underwent training for 25 epochs using a batch size of 64 and the Adam optimizer with a learning rate of $5 \cdot 10^{-4}$ and hyperparameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The training objective was defined by the cross-entropy loss function. Training was performed in floating-point precision using the MNIST training set (60,000 samples), followed by post-training quantization inference to assess post-training accuracy. The final accuracy was measured via a bit-accurate software simulation of the accelerator using the standard MNIST test set (10,000 samples). To ensure consistency and reproducibility, the dataset was sourced from the official `torchvision.datasets` module.

Latency was assessed through hardware-level simulation of the quantized SNN architecture. Specifically, we measured the number of clock cycles required to perform inference on a single MNIST sample; this metric depends on the temporal encoding window (i.e., the number of timesteps). To estimate the number of images processed per second (img/s), we analyzed the results from the functional simulation. A parallel input interface was employed in the accelerator, with one input line per neuron. Since inputs are processed sequentially, reduced parallelism can be supported by adopting a dual-buffer interface: the two buffers alternate in supplying inputs to the accelerator. While one buffer is actively feeding the computation, the other is loaded with new data, ensuring continuous operation without introducing any latency overhead.

The latency estimation starts measuring during simulation the time Δt required by the accelerator to process a single input sample as defined in (1). The beginning of the inference is marked rising the `start` signal in the accelerator and the end is detected with an active value on the `ready` signal. This measure is done with a standard clock cycle of 20ns defined by Spiker+ in its testbench instead of the target clock cycle identified post synthesis for the best target architecture (see Table 1).

$$\Delta t = t_{\text{end}} - t_{\text{start}} = (347{,}510 - 173{,}810)\,\text{ns} = 173{,}700\,\text{ns} \tag{1}$$

The throughput R can be computed as the inverse of this

processing time according to (2):

$$R = \frac{1}{\Delta t} = \frac{1}{173{,}700 \times 10^{-9}} \approx 5{,}757 \text{ img/s} \quad (2)$$

Eventually, the number of clock cycles required to complete a single inference can be precisely determined by dividing the total inference time by the clock period, as shown in (3).

$$\#Cycles = \frac{173{,}700\,\text{ns}}{20\,\text{ns}} = 8{,}685 \text{ cycles} \quad (3)$$

Since this cycle count remains constant across all network configurations, the latency for each configuration can be derived by simply multiplying the number of cycles by the architecture's clock period. This methodology yields a realistic throughput estimation under timing-accurate conditions, as it captures the actual temporal behavior observed during simulation. The result illustrates, for a representative sample, the achieved image processing throughput of the accelerator, which is approximately 18,875 images per second under the given timing conditions (4).

$$\Delta t = 8{,}685 \text{ cycles} \cdot 6.1\ ns = 52{,}978.5\ ns$$
$$R = \frac{1}{\Delta t} = \frac{1}{52{,}978.5 \times 10^{-9}} \approx 18{,}875 \text{ img/s} \quad (4)$$

To estimate hardware costs and performance, each configuration was synthesized for the selected FPGA [23] using the Vivado tool, and post-synthesis reports were collected. Dynamic power consumption was estimated assuming a default toggle rate of 12.5% at the maximum achievable clock frequency, determined via timing analysis in Vivado. Architectures using reduced numerical precision and smaller hidden layers demonstrated lower power consumption with minimal impact on accuracy. Resource utilization was analyzed in terms of LUTs, flip-flops, and BRAMs (no DSP blocks are used). All tested designs fit comfortably within the resource constraints of the target device, leaving sufficient headroom for integration into complete systems.

4. CONCLUSIONS

This work presents an end-to-end Spiker+-based flow that converts high-level SNNs into deployable FPGA accelerators while systematically stripping away non-essential blocks to meet the low-power mandate of the Grand Challenge. The flow abstracts much of the manual effort traditionally required to bring SNNs onto resource-constrained devices. Beyond the convenience factor, the approach highlights how temporal information encoding fundamentally reshapes accelerator design: instead of a single, synchronous pass over dense tensors, computation unfolds over multiple timesteps, with sparse spike events. However, custom datapaths, memory tiling and efficient data conversion compress the effective decision window, keeping end-to-end latency competitive with ANN accelerators while lowering dynamic energy per inference.

Looking beyond the competition, the same flow can reenable the deferred features—on-chip learning, adaptive neurons, dynamic partial reconfiguration—and extend to fully event-based datasets. These additions are orthogonal to the challenge rules yet pivotal for future edge platforms that must adapt online, share resources and operate under stricter energy envelopes. Taken together, the methodology and its planned extensions narrow the gap between neuromorphic theory and deployable, power-efficient hardware, underscoring SNNs as a viable alternative to conventional networks when both energy and responsiveness matter

5. REFERENCES

[1] Guoqi Li, Lei Deng, Huajin Tang, Gang Pan, Yonghong Tian, Kaushik Roy, and Wolfgang Maass, "Brain-inspired computing: A systematic survey and future trends," *Proceedings of the IEEE*, 2024.

[2] Wenzhe Guo, Mohammed E. Fouda, Ahmed M. Eltawil, and Khaled Nabil Salama, "Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems," *Frontiers in Neuroscience*, vol. 15, 2021.

[3] Nitin Rathi, Indranil Chakraborty, Adarsh Kosta, Abhronil Sengupta, Aayush Ankit, Priyadarshini Panda, and Kaushik Roy, "Exploring neuromorphic computing based on spiking neural networks: Algorithms to hardware," *ACM Comput. Surv.*, vol. 55, no. 12, Mar. 2023.

[4] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, Brian Taba, Michael Beakes, Bernard Brezzo, Jente B. Kuang, Rajit Manohar, William P. Risk, Bryan Jackson, and Dharmendra S. Modha, "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*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.

[5] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.

[6] Daniel Neil and Shih-Chii Liu, "Minitaur, an Event-Driven FPGA-Based Spiking Network Accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2621–2628, Dec. 2014.

[7] Hanwen Liu, Yi Chen, Zihang Zeng, Malu Zhang, and Hong Qu, "A Low Power and Low Latency FPGA-Based Spiking Neural Network Accelerator," in *2023 International Joint Conference on Neural Networks (IJCNN)*, June 2023, pp. 1–8, ISSN: 2161-4407.

[8] Yarib Nevarez, David Rotermund, Klaus R. Pawelzik, and Alberto Garcia-Ortiz, "Accelerating Spike-by-Spike Neural Networks on FPGA With Hybrid Custom Floating-Point and Logarithmic Dot-Product Approximation," *IEEE Access*, vol. 9, pp. 80603–80620, 2021.

[9] Jindong Li, 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*, vol. 31, no. 8, pp. 1178–1191, Aug. 2023.

[10] Daniel Gerlinghoff, Zhehui Wang, Xiaozhe Gu, Rick Siow Mong Goh, and Tao Luo, "E3NE: An End-to-End Framework for Accelerating Spiking Neural Networks With Emerging Neural Encoding on FPGAs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 3207–3219, Nov. 2022.

[11] Alireza Khodamoradi, 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*, New York, NY, USA, Feb. 2021, FPGA '21, pp. 194–205.

[12] Sixu Li, Zhaomin Zhang, Ruixin Mao, Jianbiao Xiao, Liang Chang, and Jun Zhou, "A Fast and Energy-Efficient SNN Processor With Adaptive Clock/Event-Driven Computation Scheme and Online Learning," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 4, pp. 1543–1552, Apr. 2021, Conference Name: IEEE Transactions on Circuits and Systems I: Regular Papers.

[13] Li Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, Nov. 2012, Conference Name: IEEE Signal Processing Magazine.

[14] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo, "Spiker+: a framework for the generation of efficient spiking neural networks fpga accelerators for inference at the edge," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–15, 2024.

[15] Jason K. Eshraghian, Max Ward, Emre O. Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu, "Training Spiking Neural Networks Using Lessons From Deep Learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, Sept. 2023.

[16] Dario Padovano, Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo, "Spikeexplorer: Hardware-oriented design space exploration for spiking neural networks on fpga," *Electronics*, vol. 13, no. 9, 2024.

[17] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[18] Jason K Eshraghian, Max Ward, Emre O Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu, "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.

[19] Emre O Neftci, Hesham Mostafa, and Friedemann 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.

[20] Paul J Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[21] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in neuroscience*, vol. 12, pp. 331, 2018.

[22] PyTorch, "Reproducibility," [Online available:] `https://docs.pytorch.org/docs/stable/notes/randomness.html?utm_source=chatgpt.com`, 2025, Accessed: 2025-06-11.

[23] AMD, "AMD kintex™ 7 FPGA family," [Online available:] `https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/kintex-7.html`, 2025, Accessed: 2025-06-11.