POLITECNICO DI TORINO
Repository ISTITUZIONALE

Spiker+: a framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge

Original
Spiker+: a framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge / Carpegna, A., Savino, A., Di Carlo, S.. - In: IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING. - ISSN 2168-6750. - ELETTRONICO. - 13:3(2025), pp. 784-798. [10.1109/tetc.2024.3511676]

Availability:
This version is available at: 11583/2995441 since: 2025-09-23T01:58:28Z

Publisher:
IEEE

Published
DOI:10.1109/tetc.2024.3511676

Terms of use:

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

Publisher copyright

(Article begins on next page)

25 November 2026

Spiker+: A Framework for the Generation of Efficient Spiking Neural Networks FPGA Accelerators for Inference at the Edge

Alessio Carpegna, *Student Member, IEEE*, Alessandro Savino, *Senior Member, IEEE*, and Stefano Di Carlo, *Senior Member, IEEE*

***Abstract*—Including Artificial Neural Networks in embedded systems at the edge allows applications to exploit Artificial Intelligence capabilities directly within devices operating at the network periphery, containing sensitive data within the boundaries of the edge device. This facilitates real-time decision-making, reduces latency and power consumption, and enhances privacy and security. Spiking Neural Networks (SNNs) offer a promising computing paradigm in these environments. However, deploying efficient SNNs in resource-constrained edge devices requires highly parallel and reconfigurable hardware implementations. We introduce Spiker+, a comprehensive framework for generating efficient, low-power, and low-area SNN accelerators on Field Programmable Gate Arrays for inference at the edge. Spiker+ presents a configurable multi-layer SNN hardware architecture, a library of highly efficient neuron architectures, and a design framework to enable easy, Python-based customization of accelerators. Spiker+ is tested on three benchmark datasets: MNIST, Spiking Heidelberg Dataset (SHD), and AudioMNIST. On MNIST, it outperforms state-of-the-art SNN accelerators in terms of resource allocation, with a requirement of 7,612 logic cells and 18 Block RAMS (BRAMs), and power consumption, draining only 180 mW, with comparable latency (780 μs/img) and accuracy (97%). On SHD and AudioMNIST, Spiker+ requires 18,268 and 10,124 logic cells, respectively, requiring 51 and 16 BRAMs, consuming 430 mW and 290 mW, with an accuracy of 75% and 95%. These results underscore the significance of Spiker+ in the hardware-accelerated SNN landscape, making it an excellent solution for deploying configurable and tunable SNN architectures in resource and power-constrained edge applications.**

***Index Terms*—Spiking neural networks, LIF, FPGA, neuromorphic accelerator, edge computing, artificial intelligence, frugal AI, electronic design automation, high-level synthesis.**

Received 28 December 2023; revised 24 September 2024; accepted 3 December 2024. Date of publication 11 December 2024; date of current version 12 September 2025. This work was supported by the European Union's Horizon Europe Research and Innovation Programme under Grant 101070238. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. *(Corresponding author: Alessio Carpegna.)*

The authors are with the Department of Control and Computer Engineering, Politecnico di Torino, 10129 Torino, Italy (e-mail: alessio.carpegna@polito.it; alessandro.savino@polito.it; stefano.dicarlo@polito.it).

Digital Object Identifier 10.1109/TETC.2024.3511676

I. Introduction

INTEGRATING Artificial Neural Networks (ANNs) at the edge is a pivotal computing advancement, enabling direct application of Artificial Intelligence (AI) capabilities in devices and systems at the periphery of networks, resulting crucial in domains like autonomous vehicles, industrial automation, and healthcare [1]. Using ANNs at the edge allows systems to process substantial data locally, reducing latency, power consumption, and reliance on external data centers or cloud services. Additionally, it enhances privacy and security by keeping sensitive data within the edge device . This approach boosts the effectiveness and agility of embedded applications, paving the way for innovative breakthroughs across sectors and ushering in a new era of intelligent, decentralized computing.

Neural networks, tailored for diverse computational tasks, include feed-forward networks for pattern recognition, recurrent networks for sequential data, convolutional networks for image analysis, and transformers for Natural Language Processing (NLP). Amidst this variety, Spiking Neural Networks (SNNs) emerge as a promising paradigm for edge computing in resource-constrained environments, particularly intriguing for researchers in neurosciences and Machine Learning (ML). Unlike other models, SNNs mimic biological neurons that efficiently process dynamic, time-sensitive data, making them valuable in edge scenarios, where real-time decision-making is crucial [2].

In this context, hardware accelerators are vital for SNNs applications, significantly improving computational efficiency and speed for real-time processing in resource-constrained environments [3]. However, the flexibility of specialized hardware design poses a challenge, with applications often requiring diverse network architectures, encoding methods, and neuron models. While awaiting the maturity of SNN accelerators based on emerging technologies, existing literature proposes various digital hardware solutions (refer to Section III). Unfortunately, these solutions often constrain network topology to circuit architecture, limiting exploration of the broader design space, whether considering Application-Specific Integrated Circuits (ASICs) or Field Programmable Gate Arrays (FPGAs). We propose an alternative strategy, optimizing the network architecture for specific needs and leveraging FPGAs for deploying custom hardware blocks. This approach enables efficient and low-power SNN inference engines at the edge, supporting real-time data

© 2024 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

processing. FPGAs provide high parallelism and reconfigurability, making them ideal for accelerating neural network computations with minimal latency.

To support this trend, this paper presents Spiker+, a complete framework for generating efficient low-power and low-area customized SNN accelerators on FPGAs for inference at the edge. Spiker+ introduces several pivotal contributions. At its core, it provides a fully configurable multi-layer hardware architecture implementing both fully connected and recurrent SNNs. This architecture is a significant step from the preliminary Spiker model, initially presented in [4]. It introduces a library of highly efficient architectures delving into a range of approximation techniques to implement remarkably low-area and low-power neurons, thus optimizing resource utilization while maintaining high performance. Notably, Spiker+ brings a complete design framework to the forefront, a comprehensive toolkit for developing complex SNNs accelerators. This framework empowers researchers and developers to describe target network architectures with great flexibility, enabling the specification of layers, neuron types, and input encoding techniques using a few lines of Python code. Integrating sophisticated off-line server-based training algorithms like Back-Propagation Through Time (BPTT) [5] ensures the network has cutting-edge learning capabilities. Additionally, Spiker+ emphasizes the significance of optimizing networks through quantization techniques, reducing complexity while judiciously balancing approximation and accuracy. Finally, the framework seamlessly generates a VHDL model of the accelerator, primed for deployment on Xilinx™ FPGA boards. These contributions make Spiker+ a robust solution in the hardware-accelerated SNN landscape. Spiker+ has been tested on the well-known MNIST dataset [6] and compared to state-of-the-art SNN accelerators for FPGAs, demonstrating superior performance. Moreover, two accelerators for the Spiking Heidelberg Dataset (SHD) [7] and AudioMNIST [8] have been generated and evaluated to illustrate the framework's flexibility when handling different problems. The primary aim of Spiker+ is to offer an Electronic Design Automation (EDA) framework that simplifies the design of SNN accelerators for FPGA, addressing a gap that is still underrepresented in the literature. The tool does not include features for searching optimal architectures or optimizing hyper-parameters. However, it seamlessly integrates with existing tools such as SpikExplorer [9], able to find an optimal network configuration, which can then be used by Spiker+ to generate the corresponding hardware accelerator.

The rest of the paper is organized as follows: Section II presents some background on SNNs, focusing on the models used in Spiker+; Section III reviews relevant literature on accelerating SNNs. Section IV describes the Spiker+ architecture, with all the design choices that it involves, and Section V introduces the framework able to configure, design, and generate custom hardware accelerators. Finally, Section VI presents the results obtained by applying the designed accelerators on the MNIST, SHD, and AudioMNIST, and Section VII concludes the paper.

II. Background

This section overviews foundational knowledge on SNNs, required to understand the remaining parts of the paper.

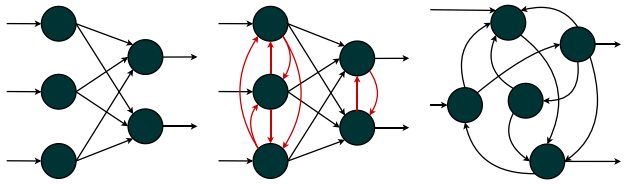


Figure 1. Typical architectures of an SNN.

A. Spiking Neural Networks

SNNs distinguish themselves through their unique information encoding based on *spikes*, inspired by neuroscience. Indeed, this neuron model mimics biological neurons and synaptic communication mechanisms based on action potentials. The information is thus represented as a flow of spikes with various neural coding techniques, shifting the computational complexity from the *spatial* dimension to the *temporal* dimension. Spike encoding methods in SNNs range from real current or voltage pulses in specialized analog circuits to numerical representations in software or dedicated digital implementations. This paper focuses primarily on the latter. Spikes convey information through temporal organization, and in the digital domain, they can be approximated as binary values: 'one' for received spikes and 'zero' otherwise. Essentially, neurons are translated into compact computational units that exchange data, i.e., their *activations* through binary bit streams.

In SNNs, neuron connections can take various forms, including Feed-Forward Fully-Connected (FF-FC) (Fig. 1(a)), Fully-Connected Recurrent (FC-R) (Fig. 1(b)), and Randomly-Connected Recurrent (RC-R) organizations. Spiker+ prioritizes FF-FC and FC-R structures, ideal for solving diverse classification tasks [10]. In particular, in FC-R SNNs, neurons connect not only to all nodes in the subsequent layer but also within their layer. This strengthens the joint activity of groups of neurons while inhibiting unrelated ensembles, fostering a competitive environment, which results in a better specialization of the single neurons.

B. Neuron Models

Over the past decades, numerous neuron computational models have emerged, originating from nerve electrical conductance measurements and mathematical modeling. The Hodgkin-Huxley model [11], inferred from squid giant axon measurements in 1952, is realistic but complex and has evolved into the simplified Izhikevich model [12]. However, for hardware implementations, balancing accuracy and complexity commonly leads to using the Leaky Integrate and Fire (LIF) model and its simplified version, Integrate and Fire (IF) [13]. These models aim to mathematically describe the behavior of a biological neuron focusing on its membrane, which is crucial to defining the internal electrical processes. The membrane selectively permits specific ion passage, accumulating charge and creating a *membrane potential* (V_m) defining the neuron's state and behavior.

The LIF model in Spiker+ encompasses a family of neuron models with varying levels of simplification. Equation (1) introduces the discrete-time formulation of a Synaptic Conductance-based II-order LIF model [13]. Operating in discrete time enables the iterative solution of the differential equations governing the temporal evolution of the membrane potential (V_m). In this model, input spikes (s_{in}) are integrated by synapses with conductance weights (W), influencing the membrane potential based on the input's significance ($W \cdot s_{in}$). The integrated spikes form the synaptic current (I_{syn}), which undergoes capacitive discharge ($\alpha \cdot I_{syn}$, with $0 < \alpha < 1$). The membrane integrates this current, resulting in increased or decreased potential based on the current sign. The current sign is influenced by the excitatory or inhibitory nature of the input spikes, impacting the neuron's firing probability. This effect can be positive or negative. The capacitive component (β) discharges the membrane toward a resting state in the absence of stimuli ($\beta \cdot V_m$, with $0 < \beta < 1$). Lastly, the reset parameter (r) models the reset process, as explained later in this section.

$$\begin{cases} I_{syn}[n] = \alpha \cdot I_{syn}[n-1] + W \cdot s_{in}[n] \\ V_m[n] = (\beta \cdot V_m[n-1] + I_{syn}[n-1]) \cdot r \end{cases}. \tag{1}$$

An action potential, represented by a separate variable s_{out}, occurs when the membrane voltage (V_m) surpasses a threshold value (V_{th}). This dual-variable approach (membrane and action potential), simplifies the description by treating the spike as a binary variable. Equation (2) illustrates the relationship between the output spike and membrane potential V_m.

$$s_{out}[n] = \begin{cases} 1, & \text{if } V_m > V_{th} \\ 0, & \text{if } V_m \leq V_{th} \end{cases}. \tag{2}$$

Finally, to model the complete discharge of the membrane when the neuron fires, a reset term is used (r in (1)). There are different alternatives to applying the reset. Spiker+ supports two modes: a *hard-reset*, shown in (3), in which the membrane is instantly brought to zero when a spike is generated, and a *subtractive-reset*, detailed in (4). In the latter, the threshold value is subtracted by the membrane potential.

$$r = 1 - s_{out}[n-1] \tag{3}$$

$$r = 1 - \frac{V_{th}}{\beta \cdot V_m[n-1] + I_{syn}[n-1]}. \tag{4}$$

The LIF model, being a family of models, allows for various simplifications to derive different LIF descriptions. For instance, setting $\alpha = 0$ transforms the II-order model in (1) into a I-order LIF, where input spikes directly influence the membrane potential. Additionally, with $\beta = 1$, a basic IF model is obtained, maintaining a constant membrane value without input spikes.

These different models serve diverse tasks. The II-order LIF excels in handling input sequences with high temporal information content, capturing longer correlations in precise spike sequences. On the other hand, I-Order LIF and IF models are simpler and preferred for processing static data converted into spikes, e.g., images. Working solely with LIF offers six distinct models: II- and I-order LIF, and IF, each with a hard or subtractive reset. These models, supported by Spiker+, combined with the architectures in Fig. 1, address various classification and regression problems.

C. Training

Training an SNN involves tuning it for specific problem-solving, such as classifying input data. The training process adjusts synaptic weights (W) and internal neuron parameters like threshold (V_{th}) and time constants (α and β) to enhance model accuracy.

Training SNNs presents notable challenges, mainly due to the non-differentiability of the SNN activation function. This paper adopts the Surrogate Gradient approach [5], replacing the non-differentiable gradient with a surrogate function, like the spike function itself or a Gaussian function. This enables the application of standard supervised learning techniques, such as BPTT, overcoming non-differentiability and facilitating effective SNN training.

Alternative solutions, like e-prop [14] for Recurrent Spiking Neural Network (RSNN) and Spike-Timing-Dependent Plasticity (STDP) for unsupervised weight tuning based on spike timing, exist. However, as Spiker+ focuses on inference using pre-trained parameters, these methods fall outside the paper's scope.

III. Neuromorphic Accelerators: Related Work

In the past, SNNs were primarily implemented using software frameworks like Brian/Brian2 [15]. However, their unique features, including high parallelism, temporal evolution, and event-driven computation, are ill-suited for dominant Von-Neumann Central Processing Unit (CPU) architectures with one or a few powerful computational units. Unfortunately, Single Instruction Multiple Data (SIMD) architectures, such as General Purpose Graphic Processing Units (GPGPUs) and Tensor Processing Units (TPUs), optimized for standard ANNs workloads, are also not well-equipped for efficiently processing event-driven information across multiple timesteps [16]. Furthermore, the binary spike encoding of SNNs does not align with the typical 64, 32, or 16-bit numeric representations of these SIMD architectures. Therefore, dedicated neuromorphic hardware is crucial for ensuring the widespread adoption of SNNs. Fig. 2 provides an overview of state-of-the-art accelerators for SNNs, offering a general perspective rather than an exhaustive list of solutions. For detailed information, refer to [17], [18].

Contributions in this domain span various design dimensions, including application-driven solutions focused on specific applications and those aimed at modeling biological neuron dynamics. However, this paper primarily emphasizes the hardware technology dimension. The research effort is divided between analog solutions based on emerging technologies and efficient digital implementations [18]. In the digital realm, presented solutions differ on the target platform (ASIC or FPGA) and accelerator size, tailored for either large-scale systems or small applications.

Examining large network models, the SpiNNaker system developed at Manchester University is implemented using standard 32-bit ARM M4F CPUs simulating neuron activity. It optimizes

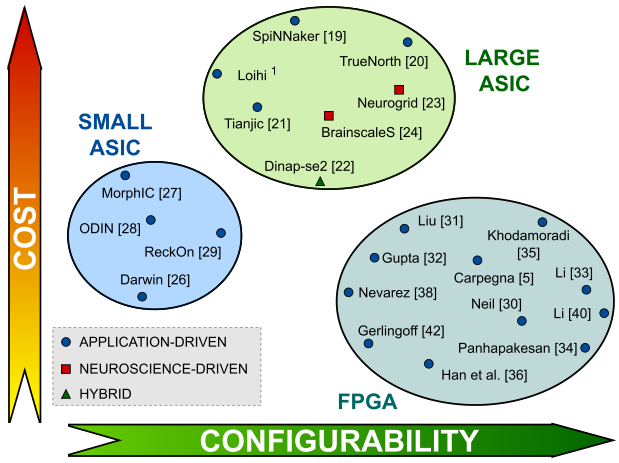


Figure 2. Landscape of neuromorphic hardware.

spike routing between units [19]. Promisingly, major computer companies invest in developing their neuromorphic accelerators, such as Intel Loihi[1] and IBM True-North [20]. These accelerators provide additional optimizations using specialized hardware to execute the neuron model. Another solution, Tianjic [21] from the University of Beijing, aims to implement a hybrid SNN/ANN model, benefiting from both domains.

An alternative design approach is recognizing that real biological neural networks function as analog physical systems. Emulating their efficiency involves using hardware components that approximate biological elements. Chips like Dynap-se2 [22] by SynSense, a spinoff of the Institute of Neuro-Informatics (INI) in Switzerland, exemplify this concept. Neurogrid [23] from Stanford University and the European project BrainscaleS [24] focus on faithfully simulating portions of a biological brain. Due to their complexity, these systems have programming tools for automatic SNN configuration. Tools like Rockpool[2] by SynSense and Nengo[3] facilitate automatic configuration and acceleration of SNNs on neuromorphic hardware based on Python model descriptions.

While these accelerators suit large-scale neuromorphic systems with good programmability, applications like the Internet of Things (IoT), wearable devices, and biomedical sensors demand small sizes, computation robustness, and low power consumption. In such cases, designing specialized digital hardware accelerators to execute specific tasks, such as classification or regression, efficiently becomes a viable solution.

To pursue this direction, the first option is designing a compact, programmable ASIC supporting various architectures and models. The focus is on digital solutions for networks ranging from hundreds to a few thousand neurons. A preliminary comparison with non-spiking hardware accelerators, dedicated to efficient convolution execution in Convolutional Neural Networks (CNNs), is discussed in [25], highlighting the energy efficiency of SNNs. These accelerators often use the Address Event Representation (AER) protocol for compatibility with neuromorphic sensors. An example is found in [26], developed at Zhejiang University. Charlotte Frenkel's work at the University of Delft and Eidgenössische Technische Hochschule (ETH) Zurich introduces three chips — MorphIC [27], ODIN [28], and RecKon [29] — exploring online learning on small, low-power, and efficient accelerators.

The final digital accelerator option, central to this paper, involves developing a specialized FPGA-based accelerator, offering advantages like cost reduction and increased flexibility by bypassing tape-out design needs. Many IoT edge systems now integrate FPGAs for task acceleration, potentially expanding the application of neuromorphic processors. An early example is the event-driven Minitaur [30], and subsequent alternatives feature diverse update policies, neuron models, architectural choices, and network sizes [4], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40]. Another key advantage of using an FPGA is its intrinsic reconfigurability. It allows hardware reprogramming to modify the network architecture or neuron model, tailoring the accelerator to specific application requirements. However, existing accelerators still need to exploit this characteristic efficiently. The first example of FPGA-oriented Design Space Exploration (DSE) is found in [41], focusing on the best encoding technique for input data translation into spike sequences. Conversely, E^3NE[42] provides a block library to configure networks for specific applications, optimizing data movement and hardware utilization, with a dedicated section for input encoding.

In this scenario, there is still a lack of a comprehensive framework that conceals the internal details of the architecture, enabling the user to operate at higher abstraction levels. For instance, a Python description of the model could be automatically translated into custom blocks. Spiker+ moves in this direction, trying to address this challenge.

IV. Spiker+ Architecture

This section presents the Spiker+ hardware architecture, which serves as the central component of the Spiker+ SNN hardware acceleration framework. The architecture is introduced top-down, beginning with the high-level network model and then delving into the neurons and input/output interfaces.

A. Network Architecture

The SNN architecture presented here builds upon the initial Spiker architecture introduced in [4]. While our earlier work provided a proof of concept tailored for inference on the MNIST dataset [6], derived from the SNN model by Diehl and Cook [43], Spiker+ focuses on a generic and fully configurable architecture adaptable to various problems.

Fig. 3 depicts the high-level architecture of a toy example of a three-layer FF-FC architecture used to introduce the three hierarchical levels of Control Units (CUs) that characterize Spiker+: i) the *network CU*, responsible for synchronizing the various components within the network; ii) the *layer CUs*, orchestrating the update of the neurons of a layer based on a

[1]https://www.intel.com/content/www/us/en/research/neuromorphic-computing.html
[2]https://rockpool.ai
[3]https://www.nengo.ai

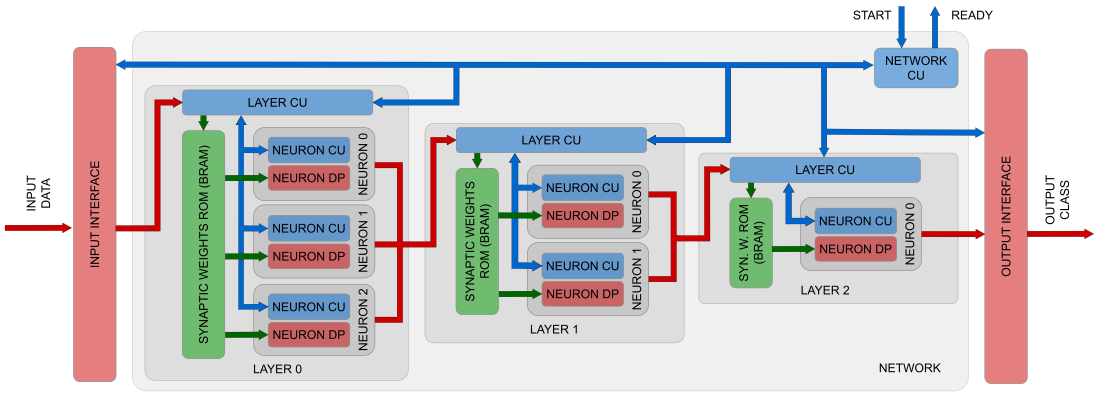


Figure 3. Spiker+ example of FF-FC architecture. The example includes three layers with different numbers of neurons and depicts all the architecture's main control blocks.

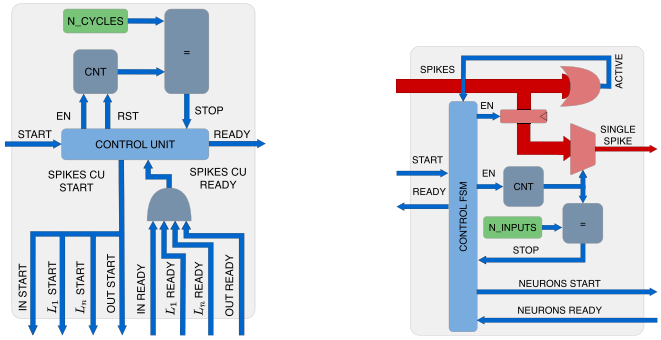
set of input spikes; iii) the *neuron CU*: the accelerator core controlling the update of the membrane potential in each neuron. This organization represents a highly optimized architecture in terms of performance and space utilization.

Block communication is based on a simple two-signal (`start/ready`) handshake protocol to ensure high modularity while minimizing design complexity. When a block (i.e., a neuron or a layer) is ready to work, it notifies the corresponding CU through the `ready` signal and awaits a new `start` signal to begin the computation. Consequently, if two blocks need synchronization, combining the two `ready` signals with an AND gate ensures that the CU waits for both before initiating a new computation. This protocol is also employed at the interface with the external world. Such an approach maintains modularity in the design and paves the way for various architectural solutions.

B. Network CU: Global Synchronization

The primary function of the Network CU is to coordinate the temporal evolution of the neurons of the different layers during an inference. As previously mentioned, in an SNN, information is encoded as trains of spikes (i.e., sequences of bits) received on every input. Each train is characterized by a given duration (i.e., the number of transmitted bits) denoted as `N_cycles`. Thus, the network performs inference by evolving over `N_cycles` temporal steps to analyze the temporal patterns.

The Network CU reported in Fig. 4(a) receives a `start` signal when a new inference must be initiated. It then manages an iterative process. At every iteration, it awaits the `ready` signal from all Layer CUs composing the network to ensure all layers are prepared to work (i.e., `L1 ... Ln ready`). The Network CU also synchronizes with the input/output interface to ensure that data are available and results are correctly delivered (i.e., `IN/OUT ready`). Subsequently, it asserts a set of `start` signals, enabling all connected blocks to perform a computation step. At the same time, the internal counter (`CNT`) increases to track the computation length. Upon reaching the desired cycle count (`N_cycles`), the loop concludes, and the `ready` signal is asserted to indicate the end of the inference process.



(a) Network CU (b) Layer CU

Figure 4. Internal architecture of the Network and Layer CUs.

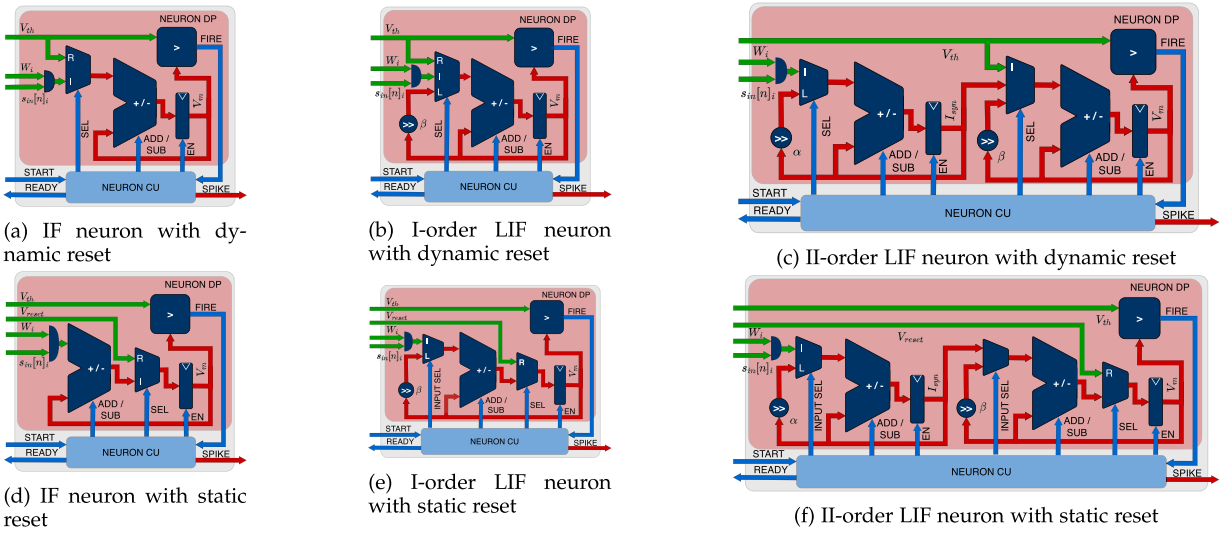
C. Layer CU: Deliver Spikes to Neurons

In a fully connected multi-layer SNN as the one proposed in Fig. 3, a parallel update of each layer involves three dimensions: i) the number of neurons, ii) the number of inputs processed by each neuron, and iii) the temporal dimension of each input, representing the number of cycles.

The last dimension is inherently sequential and cannot be parallelized, as it depends on the temporal evolution of the inputs. This dimension is managed by the Network CU discussed in Section IV-B, which manages the update during each cycle.

If the network is sufficiently small, updating all neurons with their inputs in parallel within a single cycle could be feasible. However, the network and input data sizes are typically too high to achieve such a degree of parallelism. Consequently, Spiker+ exploits only one dimension to obtain parallelism, concurrently updating all neurons within a layer while sequentially providing inputs to each neuron. The Network CU, depicted in Fig. 4(b), oversees this process.

Once again, this circuit operates based on a `start/ready` protocol. The control unit receives a `start` signal from the Network CU and enters a loop: it awaits the readiness of the neurons composing the layer to process a new spike (`neurons ready` signal), initiates the computation (`neurons start` signal), and increments the internal counter (`CNT`). The counter directly selects the spike to be processed from the sampled inputs. When all input spikes have been provided to the neurons



(a) IF neuron with dynamic reset

(b) I-order LIF neuron with dynamic reset

(c) II-order LIF neuron with dynamic reset

(d) IF neuron with static reset

(e) I-order LIF neuron with static reset

(f) II-order LIF neuron with static reset

Figure 5. Spiker+ neuron architectures in order of increasing complexity from left to right (IF, I-order LIF and II-order LIF), with the subtractive reset on top and the fixed one on the bottom. In the multiplexers, channels labeled with I indicate the beginning of the Integrate path, R the beginning of the Reset path, and L the beginning of the Leakage path.

(`N_INPUTS`), the loop concludes, and the control unit asserts the `ready` signal.

An additional component visible in the upper section of Fig. 4(b) is an `OR` gate utilized to verify if there is at least one active spike among the inputs. Currently, no encoding or compression has been applied to the spikes. However, considering that SNNs typically exhibits sparse activity, avoiding unnecessary computations when there are no spikes can significantly save time and power. The role of the `OR` gate is to compress along the time dimension: if there is no active spike in a particular cycle, looping over all inputs to provide no spike to the neuron becomes unnecessary.

D. Neuron Models

All the different LIF neuron models presented in Section IV-B are translated into dedicated hardware implementations in Spiker+, trying to minimize the required components. Building upon the groundwork laid in [4], the proposed neuron functions as a Multiply and Accumulate (MAC) unit, augmented with additional components and controls to manage its various states. Fig. 5 shows the obtained architectures, in order of increasing complexity from left to right (IF, I-order LIF and II-order LIF), with the subtractive reset on top and the fixed one on the bottom.

From a hardware perspective, the most critical factors of the characteristic equation of the neurons are the multiplications. Four of them can be found in (1):

1) The synapses weighting: $W \cdot s_{in}[n]$
2) The reset: $V[n-1] \cdot r$
3) The exponential decays: $\alpha \cdot I_{syn}[n-1]$ and $\beta \cdot V_m[n-1]$

For the first one, exploiting the binary nature of the spikes reduces the operation to a simple selection: zero if there is no spike, W if a spike is present. This can be implemented as a bitwise `AND` between the weight and the spike.

The reset operation can be applied in two ways, as shown in (3) and (4). The first case exploits the binary nature of the spike: either the membrane is kept at its value or reset to zero, so this is again a selection process more than a multiplication. The hardware implementation is a bit more general since it allows to explicitly choose the value of V_{reset}, which in this case can also be different from 0, as shown in Fig. 5(d), (e) and (f). The second reset method can be obtained by simply subtracting the threshold voltage V_{th} from the computed value of V_m, as shown in Fig. 5(a), (b) and (c).

At this point, the last critical multiplication is the one required to compute the step-by-step exponential decay of the membrane. The problem exists only for the two LIF models (in the IF model, the membrane is kept fixed without stimuli), with one multiplication needed in the I-order version and two multiplications in the II-order one. The criticality is solved once again, exploiting the characteristics of binary operations. If one of the operators is representable as a power of two, the multiplication can be reduced to a simple bit-shift. Since there is no control on the values of I_{syn} and V_m, which evolve dynamically during the update of the network, the only parameters on which it is possible to act are the constant hyper-parameters α and β. The values can vary between 0 and 1, with larger values corresponding to slower exponential decay. Generally, a value near to 1 is observed. In this case α and β can be approximated as $\alpha = 1 - \alpha'$ and $\beta = 1 - \beta'$, where α' and β' are negative powers of 2. As shown in [4], the overall accuracy has no notable impact if such an approximation is applied during the training phase.

E. Synapses

The primary advantage of implementing SNNs on dedicated hardware, alongside the execution parallelism, lies in the opportunity to integrate memory and computation. On an FPGAs, this integration can be achieved through two distinct methods.

For relatively small memory requirements, such as the internal parameters of the neurons, the internal Look Up Tables (LUTs) can serve as a viable memory solution. This approach offers superior speed, leveraging Flip Flops (FFs) and registers. However, the available space is limited, primarily due to the necessity of accommodating the logical functions of the network within the LUTs.

In scenarios where a larger memory capacity is necessary, particularly for synaptic weights, many FPGAs grant access to discrete units of Static Random Access Memory (SRAM) strategically positioned close to the computing elements, commonly referred to as Block RAM (BRAM).

Spiker+ provides a synapse interface, implementing the `start/ready` handshake protocol, and relies on an initialization file containing quantized weights. Weighs are stored into BRAMs. Spiker+ expects all neurons to access their respective weights in parallel upon activating the ready signal by the synapse; therefore, it strongly relies on the high parallelism provided by on-board BRAMs. Spiker+ also permits storing weights in an external Dynamic Random Access Memorys (DRAMs) when on-board space is insufficient. In such situations, the synaptic interface loads the weights for the current cycle before asserting the ready signal, impacting the accelerator's speed.

A secondary configurable attribute concerning synapses involves incorporating feedback connections, as mentioned in Section IV-A. Spiker+ can be configured to include or exclude these connections, depending on the application requirements.

F. I/O Interface

Spiker+ requires an input/output interface to receive data and transmit results. Spiker+ supports two scenarios. In the simple scenario, inputs have already been encoded as spikes. For instance, these data may originate from neuromorphic sensors, such as a Dynamic Vision Sensor (DVS) cameras or a silicon cochlea. Alternatively, they could be pre-encoded by an external block before being stored.

In a more complex scenario, data are stored in a raw numeric format and converted on-board into spike streams. There are different methods available for this conversion, depending on the type of input data [41]: i) firing rate coding (i.e., information is encoded using the instantaneous average firing rate), ii) population rank coding (i.e., information is encoded using the relative firing time of a population of neurons), or iii) temporal coding (i.e., information is encoded with the exact timing of individual spikes). An efficient rate encoding structure such as the one proposed in [4] can be directly connected to Spiker+. Furthermore, several possibilities exist concerning data transmission: data may arrive as a continuous stream directly from a sensor or be transmitted through an external link. Alternatively, data may be stored in memory, necessitating memory access by the accelerator. To accommodate these diverse scenarios, Spiker+ uses the `start/ready` handshake protocol to manage the communication with the input interface.

At the accelerator output, decisions are usually taken based on the firing activity of the last layer. Spiker+ implements the output interface using simple counters, one for each output neuron, that can be post-processed outside the accelerator (e.g., the most active neuron wins). This simple implementation is only one possible option and can be easily customized to specific needs. The only requirement of the output interface is to implement the `start/ready` protocol to synchronize with the network.

It has to be noted that the latency measures reported in the paper have been taken considering input and output interfaces that can keep the pace of the accelerator. Overall, the network performance strongly depends on the interfaces. Results in Section VI aim to show the maximum performance Spiker+ can reach.

V. Configuration Framework

Spiker+ goes beyond being a mere hardware accelerator; it is a comprehensive design framework that facilitates easy customization of the SNN accelerator for specific applications. As detailed in Section IV, the platform encompasses six distinct neuron models, a modular layer interface allowing instantiating any desired number of layers, and customizable inter-layer feedback connections. However, manually defining the architecture at the Register Transfer Level (RTL) requires substantial effort. To tackle this challenge, Spiker+ incorporates a Python-based configuration framework, streamlining the customization process to just a few lines of code. The customization and tuning flow for a specific target application within Spiker+ is depicted in Fig. 6.

The configuration of the network, its training, and the optimization towards its hardware realization are performed at the software level, using Python functionalities. These first steps are built on top of the open-source framework `snntorch`.[4] The description of the network is based on a Python dictionary that contains various parameter settings, such as the SNN model and architecture (e.g., FF-FC or FC-R SNN), specifying relevant parameters like the number of layers, neurons per layer, neuron type, and timing constants.

First (step 1), the dictionary is processed by the `net-builder` module, which converts it into a Python object that serves as a key component for subsequent steps. Once the network is constructed, it undergoes training (step 2), utilizing `snntorch` and `pytorch`. During training, as per [4], the time constants α and β are rounded to the nearest power of two to adjust for this approximation. The outcome of the training phase includes a set of learned parameters such as weights and thresholds along with an accuracy evaluation of the trained model.

As training typically occurs in floating-point precision, unsuitable for edge hardware accelerators, Spiker+ implements functions to quantize the network, automating the exploration of the quantization impact on SNN accuracy (step 3). Spiker+ employs a two's complement N-bit fixed-point representation, and if any values exceed the representable range during quantization, they are saturated to either the maximum or minimum value. Users can define the search interval (default: 16 to 0), and

[4]https://snntorch.readthedocs.io/en/latest/

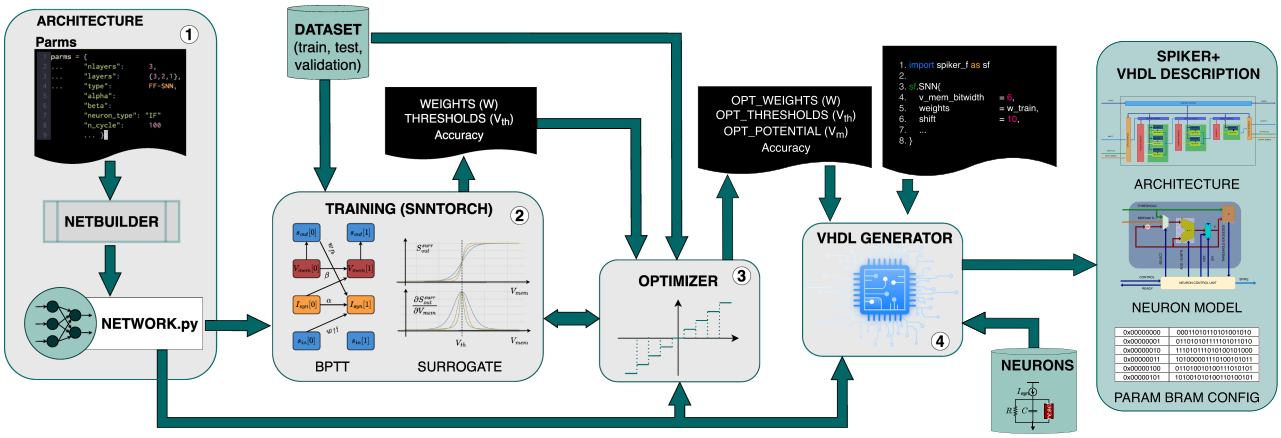


Figure 6. Spiker+ configuration framework.

the tool iterates through these values, performing full inferences and returning the resulting accuracy values. Currently, the `optimizer` performs a complete grid search over the quantization range, but future improvements may include more advanced search techniques, such as Bayesian optimization, to accelerate the process. Additionally, a Quantization Aware Training (QAT) could help mitigate the accuracy loss. Plans also include merging the `trainer` and `optimizer` using tools like Brevitas[5] to handle quantization.

Thanks to the modularity of the Spiker+ blocks and their interfaces, the tool can integrate seamlessly with other open-source frameworks like `spikingjelly` [44]. This allows users to configure custom `netbuilder`, `trainer`, and `optimizer` modules based on other tools. However, the built-in functionality of Spiker+ relies on `snntorch`, benefiting from its active community and extensive documentation. The goal is to offer an open-source, user-friendly tool that aligns with ongoing efforts to streamline the development and deployment of neuromorphic systems.

In the final step (step 4), the chosen model, architecture, and trained parameters are delivered to the `VHDL Generator`, which automatically translates them into a VHSIC Hardware Description Language (VHDL) description of the SNN. This stage capitalizes on the modularity of the proposed architecture and utilizes an available library of neuron models. This is the core and the main contribution of Spiker+ tool. It relieves the user from the need to have specific skills in hardware design: starting from a high-level description of the network, it automatically generates the desired architecture using the VHDL language.

Furthermore, the tool generates configuration files for storing the trained parameters in the FPGA memory. This provides flexibility to the user in selecting the type of memory. Modern hardware design tools, such as Xilinx Vivado™, support the automatic generation of Read Only Memory (ROM) memories, allowing users to choose the desired hardware platform, such as LUTs, BRAMs, or distributed Random Access Memory (RAM). These tools expect a configuration file as input, precisely what Spiker+ provides.

[5]https://zenodo.org/records/12800880

To the best of our knowledge, Spiker+ is the first complete high-level synthesis tool for an SNN, generating VHDL code automatically from a high-level Python description of the network. Additionally, the VHDL description of the accelerator includes a testbench for enhanced development and simulation convenience. This testbench allows inputs to be read from a file, enabling users to provide the desired input vector stimuli. It provides a straightforward interface to ensure the device synthesizes on the selected FPGA. Only recently, a new intermediate representation, Neuromorphic Intermediate Representation (NIR), was introduced to decouple the network description from its specific implementation [45]. NIR enables the description of any SNN as a graph, making explicit the connection patterns, computational primitives, and other key aspects. Given that Spiker+ specifically targets Fully-Connected (FC) and FC-R architectures, incorporating a translation into NIR at this stage would have added unnecessary complexity. However, this approach shows great potential for future iterations of Spiker+. A `vhdl generator` capable of interpreting NIR descriptions would further decouple the network configuration from the hardware architecture, enhancing the modularity and flexibility of the framework, in line with its current design principles.

VI. Experimental Results

Spiker+ has been evaluated using two widely recognized benchmark datasets, i) MNIST [6] and ii) SHD [7], plus an additional audio dataset, the AudioMNIST [8].

MNIST comprises grayscale images of handwritten digits from 0 to 9, commonly used to benchmark AI algorithms. This dataset is ideal for comparing Spiker+ with other SNN accelerators. Images are converted into spikes using Poisson-distributed rate encoding. Due to the dataset simplicity, a basic I-order LIF model with a FF-FC structure suffices for accurate classification.

SHD is explicitly designed as an SNN benchmark, containing recordings of people pronouncing numbers in English and German. It requires a more complex neuron model, specifically a II-order LIF, and a network architecture with inter-layer recurrent connections to account for the importance of the time dimension in achieving acceptable classification accuracy.

TABLE I
SUMMARY OF THE EXPERIMENTAL SET-UP ON THE TWO DATASTES

	MNIST	SHD	Audio MNIST
Data type	Gray-scale images	Audio recordings	Audio recordings
# inputs	784	700	40
Spikes time-steps	100	100	73
Encoding	Rate code	Custom [7]	Binarized spectrogram
Training samples	60,000	8,156	24,000
Test samples	10,000	2,264	6,000
Net type	FF-FC SNN	FC-R SNN	FF-FC SNN
Net arch	784-128-10	700-200-20	40-150-10
Neuron model	I-order LIF	II-order LIF	I-order LIF
Reset	Subtractive	Subtractive	Subtractive
Training method	BPTT with SG	BPTT with SG	BPTT with SG
Surrogate functioon	Arctan	Fast Sigmoid	Arctan
Model accuracy	96.83%	75.44%	95.23%

AudioMNIST is again a dataset of spoken numbers from 0 to 9, pronounced in English. To convert audio samples into spikes, spectrograms were first generated using 40 Mel-spaced band-pass filters. These spectrograms were then binarized by setting values to 1 only where the spectrogram exceeded a defined threshold. This process reduces the dataset's size compared to the other two, making it suitable for real-time applications where an electronic system could leverage Spiker+ support. The AudioMNIST dataset has not yet been widely explored with SNNs in the literature, making it an intriguing case.

These datasets differ significantly, demanding SNNs with distinct models and complexities, providing an opportunity to test Spiker+ reconfigurability for three different tasks. All the models are trained offline employing BPTT with two different surrogate gradient functions. Table I summarizes the considered setup.

The remainder of this section is structured as follows: Section VI-A presents results from tuning Spiker+ on the three target datasets and compares it with state-of-the-art SNN accelerators on FPGA. Sections VI-D analyze various Spiker+ configurations, examining the influence of architectural choices and data characteristics on area, latency, and power consumption. The goal of this analysis is to show the importance of a tool able to automatically generate different designs when exploring different alternatives to solve a specific task.

A. Benchmarking

Table II provides a comprehensive comparison of Spiker+ with recent state-of-the-art FPGA accelerators designed for SNNs on the MNIST dataset. The table is split into two sections. The upper section covers Spiking Convolutional Neural Networks (SCNN) accelerators, where spiking layers are strategically placed after or interleaved with standard convolutional layers, gradually identifying key features in input images. The lower section considers pure spiking accelerators with fully connected layers of either IF or LIF neurons. Notably, the comparison focuses on works published from 2020 onward, while references such as [4] and [40] provide information on the performance of older accelerators.

In image classification tasks, SCNNs accelerators achieve superior accuracy, peaking at 99.30% with larger and more complex networks, often utilizing DSP cores on the FPGA (e.g., [33], [35], [38]). However, despite Spiker+ being explicitly designed to trade off accuracy with other design dimensions (i.e., power, area, latency), it still achieves a commendable accuracy of 93.85%. Among purely spiking accelerators, it is only surpassed by [36], which employs an accelerator with higher latency, power, and size. Remarkably, Spiker+ excels in compactness and low power consumption. On a low-end Xilinx™ XC7Z020 FPGA board, it uses only 7,612 logic cells (4.8% of available cells) and 18 BRAMs (13% of available BRAMs), with a total power requirement of 180 mW. This makes it an optimal solution for limited space or power-constrained applications. It is important to note that Table II measures the area with a general "logic cells" value obtained by combining LUTs and FFs for a concise overview. For Spiker+, these values are 4,314 and 3,298, respectively. Detailed values for other accelerators can be found in their respective papers.

A noteworthy observation from the comparison in Table II is that the top power-efficient accelerators employ a clock-driven update policy. This counterintuitive finding contradicts the general literature assertion favoring event-driven approaches for power efficiency. A clock-driven approach seems the most effective solution for relatively small-sized accelerators lacking sufficient sparsity in spiking activities. Li et al. attempted to address this with a hybrid architecture, adapting its update strategy to input sparsity. However, this strategy did not yield significant power savings, even with added complexity [40]. Importantly, the power consumption of the accelerator reported in [4] is unrealistically high. This is due to an error in mapping I/O ports during the accelerator implementation, leading to an overestimated value.

Regarding latency, Spiker+ requires 780 μs to classify an input image. Although not the fastest result in Table II, this achievement is noteworthy considering the limited hardware resources and power consumption. Factors influencing this latency include: i) the clock frequency capped at 100 MHz due to BRAM access time; ii) the image encoding using 100 time-steps (the window size impacts inference time directly); iii) the speed of the input and output interfaces; iv) the input spiking activity affecting the classification time, as explained in Section VI-B. For comparison, Carpegna et al. [4] achieved 220 μs image classification time with a 3500 time-step encoding window. This work employed a different encoding that privileged spike sparsity, impacting accuracy (i.e., 73.96%) but demonstrating how sparsity can reduce inference time.

To better compare Spiker+ with accelerators Fig. 7 shows the Pareto optimal curves comparing the considered architectures and highlights the optimal trade-offs between different metrics. Looking at Fig. 7(a) it can be observed that Spiker+ lies a little behind the majority of the others in terms of accuracy, with an

TABLE II
COMPARISON OF SPIKER+ TO STATE-OF-THE-ART FPGA ACCELERATORS FOR SNNS

Design	Liu et al. [31]	Nevarez et al. [38]	Li et al. [33]	Gerlinghoff et al. [42]	Panchapakesan et al. [34]	Khodamoradi et al. [35]
Year	2023	2021	2023	2022	2021	2021
f_{clk}[MHz]	100	200	300	200	200	N/R
Neuron bw	8	8	12	N/R	4	N/R
Weights bw	8	8	8	3	4	N/R
Update	Clock	Clock	Clock	Clock	Event	Event
Model	IF	Spike-by-Spike (SbS)	LIF	LIF	IF	LIF
FPGA	XA7Z020	XC7Z020	XCZU3EG	XCVU13P	XCZU9EG	XA7Z020
Avail. BRAM	140	140	216	2688	912	140
Used BRAM	N/R	16	50	N/R	N/R	40.5
Avail. DSP	220	220	360	12288	2520	220
Used DSP	0	46	288	0	N/R	11
Avail. logic cells	159,600	159,600	211,680	3,088,800	822,240	159,600
Used logic cells	27,551	23,704	15,000	51,000	N/R	39,368
Arch	28×28-32c3-p2-32c3-p2-256-10	28x28x2-32c5-p2-64c5-p2-1024-10	28x28-16c3-64c3-p2-128c3-p2-256c3-256c3-10	32×32×1–6c5–p2–16c5–p2–120c5–120–84–10	28x28-32c3-p2-32c3-p2-256-10	28x28-16c7-24c7-32c7-10
#syn	8,960	75,776	2,560	25,320	10,752	320
T_{lat}/img [ms]	0.27	1.67	0.49	0.29	0.08	N/R
Power [W]	0.28	0.22	2.55	3.40	N/R	N/R
E/img [mJ]	0.076	0.37	1.250	0.986	N/R	N/R
E/syn [nJ]	8.48	4.88	488	38.9	N/R	N/R
Accuracy	99.00%	98.84%	98.12%	99.10%	99.30%	98.50%
Design	**Han et al. [36]**	**Gupta et al. [32]**	**Li et al. [40]**	**Carpegna et al. [4]**	**SPIKER+ (this work)**	
Year	2020	2020	2021	2022	2024	
f_{clk}[MHz]	200	100	100	100	100	
Neuron bw	16	24	16	16	6	
Weights bw	16	24	16	16	4	
Update	Event	Event	Hybrid	Clock	Clock	
Model	LIF	LIF	LIF	LIF	LIF	
FPGA	XC7Z045	XC6VLX240T	XC7VX485	XC7Z020	XC7Z020	
Avail. BRAM	545	416	2,060	140	140	
Used BRAM	40.5	162	N/R	45	**18**	
Avail. DSP	900	768	2,800	220	220	
Used DSP	0	64	N/R	0	0	
Avail. logic cells	655,800	452,160	485,760	159,600	159,600	
Used logic cells	12,690	79,468	N/R	55, 998	**7,612**	
Arch	784-1024-1024-10	784-16	784-200-100-10	784-400	784-128-10	
#syn	1,861,632	12,544	177,800	313,600	101,632	
T_{lat}/img [ms]	6.21	0.50	3.15	0.22	0.78	
Power [W]	0.477	N/R	1.6	59.09	**0.18**	
E/img [mJ]	2.96	N/R	5.04	13	0.14	
E/syn [nJ]	1.59	N/R	28	41	1.37	
Accuracy	97.06%	N/R	92.93%	73.96%	93.85%	

average ranking in terms of latency. From the point of view of power consumption and area utilization, expressed in terms of the required logic cells, Spiker+ is dominant. This becomes even more evident in Fig. 7(b) and (c), where the hardware-related features are compared against one another. Across all three plots, Spiker+ consistently emerges as the optimal choice, highlighting the effectiveness of the design strategies discussed in Section IV. It is important to note that, because the focus was primarily on hardware design, there is still potential for accuracy improvements using more advanced training and parameter tuning techniques. Notably, a multi-objective Bayesian optimization conducted in [9] enabled us to optimize the encoding process, reducing the required number of time steps, directly linked to latency, and improving accuracy by approximately 2%. Thus, there is still room for further enhancements at the algorithmic level. Ultimately, what stands out is that Spiker+ offers the best balance in terms of the consumption-area-latency trade-offs.

Apart from the MNIST use case, Spiker+ is the first accelerator tested on SHD and AudioMNIST. Consequently, results reported in Table III are presented independently, without comparison to other architectures.

The hardware requirements for processing the SHD exceed those used for MNIST and AudioMNIST due to several factors: the network architecture required to process this complex dataset is larger, the neuron model is a more complex II-order LIF (i.e., double size compared to a I-order LIF), and the accelerator uses a FC-R model featuring inter-layer feedback connections with weights stored in BRAM. The bit widths of the neuron membrane potential and weights are also higher. However, Spiker+ remains smaller and more power efficient than most accelerators in Table II. Latency is reduced from 780 μs for

TABLE III
BENCHMARKING ON THE SHD AND AUDIOMNIST DATASETS

	SHD	AudioMNIST
f_{clk}[MHz]	100	100
Avail. logic cells	159,600	159,600
Used logic cells	18,268	10,124
Neuron bw	8	8
FF weights bw	6	5
Arch	700-200-20	40-150-10
RR weights bw	5	0
#syn	184,000	7,500
Update	Clock	Clock
T_{lat}/recording [ms]	0.54	0.08
T_{lat}/timestep [us]	5.4	1.0
Model	II order LIF	I order LIF
Power [W]	0.43	0.29
FPGA	XA7Z020	XA7Z020
Energy/recording [mJ]	0.23	0.02
Energy/synapse [nJ]	1.25	2.67
Avail. BRAM	140	140
Used BRAM	51	16
Accuracy	72.99%	89.82%

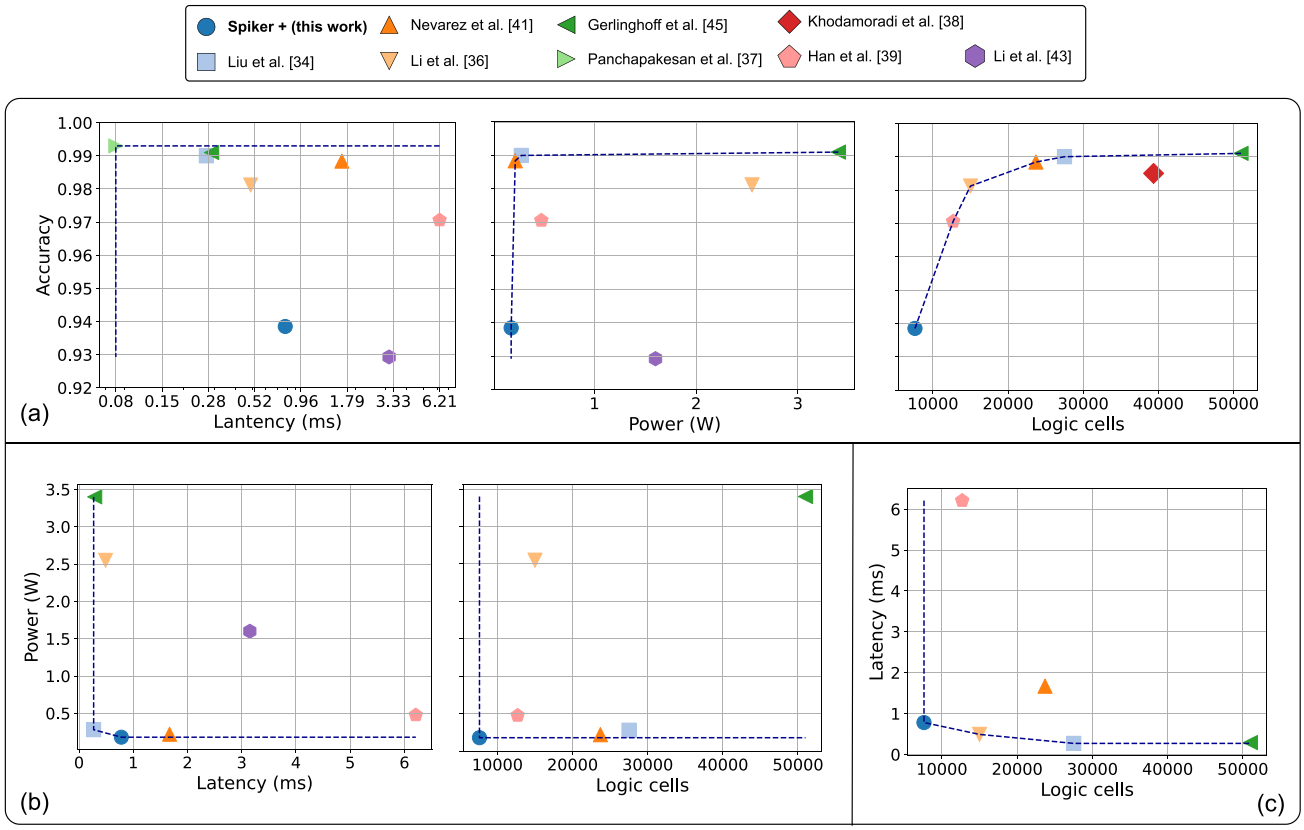


Figure 7. Pareto optimal curves of the accelerators presented in Table II. (a) Accuracy is plotted against the three key hardware metrics: latency, which reflects computation speed, power consumption, and area utilization, measured by the number of logic cells required by the accelerator. Memory usage is closely tied to area utilization, so it is not shown here. (b) Power consumption is plotted against latency and area utilization. (c) Area utilization is plotted versus latency. Spiker+ emerges as the optimal solution for all hardware-related metrics, though it still falls slightly short in terms of accuracy.

MNIST to 540 μs for SHD thanks to lower input activity in the biologically inspired encoding used for this dataset. Similarly, latency is reduced to 80 μs for the AudioMNIST, thanks to the lower dimensionality of the inputs (40 spike channels are sufficient to capture the most relevant features of the spoken digits), and a smaller amount of timesteps. Differently from the MNIST task, both the SHD and AudioMNIST are based on time-varying signals, which are intrinsically more suitable for SNN processing. Table III reports the latency required for a single time step, which is 5.4 μs and 1 μs respectively. This makes Spiker+ on its own able to process in real-time signals sampled up to 185 kHz for the SHD and 1 MHz for the AudioMNIST. While this does not account for input pre-processing or communication between Spiker+ and the processor or main memory, these tasks are not computationally intensive, particularly for AudioMNIST, due to the conversion protocol we developed and the reduced size of the input data.

B. Performance versus Input Activity

As previously mentioned, the input spiking activity, influenced by the encoding method, plays a critical role in the accelerator's performance. Before delving into a detailed analysis, Fig. 8 illustrates the average number of active cycles across different network layers. A significant variation is observed among the datasets. In MNIST, activity consistently decreases as it progresses through the network. For SHD, there is a peak in activity in the first hidden layer, while in AudioMNIST, the activity remains steady in the first two layers and increases in the output layer. This discrepancy could be attributed to the inter-layer feedback in SHD, which causes more synchronized activity compared to the FF-FC architecture used for the other two datasets. Other contributing factors include differences in input data statistics and the impact of the training process

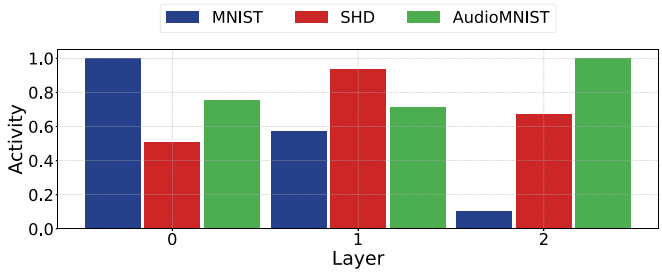


Figure 8. Visual representation of the average number of active cycles at various stages of the network.

Since all layers update in parallel and process inputs sequentially, latency is determined by the slowest layer (i.e., the

layer handling the largest set of inputs). For the MNIST and SHD architectures, detailed in Table I, the slowest layer is the input layer, processing 784 and 700 inputs respectively. In this layer, 100% of time-steps contain at least one spike for MNIST, while for SHD, the percentage is around 48%. Consequently, SHD, with the combination of a lower number of inputs and lower activity in the input layer, enables increased inference speed. Since both models use the same number of time steps and clock frequency, and the difference in the number of inputs is not significant, one might expect about 48% inference time reduction for SHD compared to MNIST due to the reduced activity (i.e., about 0.37 ms). However, the observed value in Table III is 0.54 ms. The higher latency is explained by the FC-R model used by SHD incorporating inter-layer feedback connections, processed sequentially. Therefore, an additional set of 200 feedback inputs must be processed for the first layer, with an average of 93% active time steps. For AudioMNIST, the latency is primarily determined by the hidden layer, which consists of 150 neurons with an average activity of around 71%. Given the reduced number of time steps compared to the other datasets, the resulting latency is significantly lower.

The input activity not only impacts inference latency, as explained earlier. When the activity decreases, there is a higher probability of having spare cycles with no spikes, allowing the Layer CU to skip the sequential processing of all inputs. This reduction in calculations also affects power consumption. Fig. 9 analyzes how power, latency, and energy (i.e., power times latency) change with different input activities on the two datasets, highlighting counterintuitive behaviors.

Examining Fig. 9(a) that reports the average power consumption of an inference task, we observe two distinct behaviors. In the case of MNIST and AudioMNIST, reducing input activity increases power consumption, reaching a limit of around 200 mW and 300 mW respectively as activity approaches zero and stabilizing at 180 mW and 295 mW under full activity. This behavior is explained by the clock-driven nature of the accelerator, where every clock cycle triggers a network update regardless of active spikes in the input. As mentioned earlier, Spiker+ skips time steps without active inputs. However, the exponential decay of the membrane is computed step by step. Therefore, without input stimuli, Spiker+ dedicates one clock cycle to decay all membranes before returning to an idle state, awaiting the next input set. The two situations are similar from the perspective of neuron switching power, as the membrane is updated in both cases. However, the layer CU continually switches between two states, resulting in a high total switching activity and higher power consumption. Conversely, when executing a sequential update on the inputs, the CU enters the update state and then awaits the completion of the loop. This behavior is not observed in SHD. Given the larger architecture and higher weight bit-width, power consumption in SHD is likely dominated by BRAMs access during input processing.

In conclusion, latency heavily depends on input activity. Without active spikes, the execution time tends to $\texttt{N_cycles} \times T_{clk}$, as a single clock cycle is sufficient to decay the membranes. Consequently, overall energy consumption is reduced with decreasing input activity.

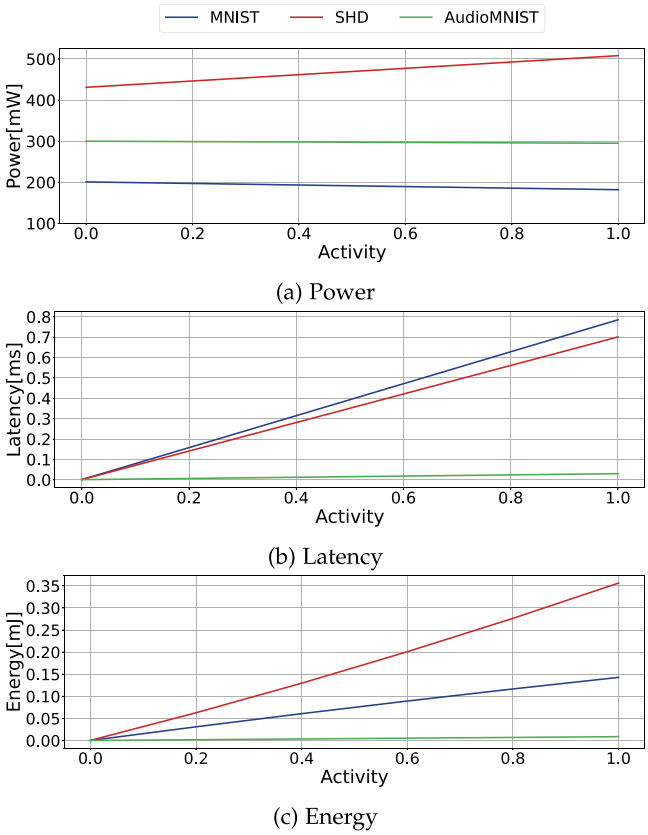


Figure 9. Impact of input activity variations on energy, power, and latency of inference using Spiker+.

C. Performance versus Quantization

As one of the key features of Spiker+ is the optimization of the accelerator through quantization of weights and membrane potentials, it is crucial to examine how these design choices influence performance. Latency remains unaffected by the selected bit-widths, as it is determined by the required amount of processing, which depends on: i) input size, ii) number of time steps, iii) network size, where each layer processes the outputs from the previous one, and iv) spike sparsity. Since all neurons operate in parallel, latency is dictated by the layer that needs to process the largest set of spikes. The primary presumed impacts of quantization are on power consumption and network accuracy. Fig. 10 illustrates the results of quantization on the considered datasets.

The resilience of SNNs to quantization is notable. In all the models, despite their differences and the unique information encoding methods employed, the decrease in accuracy with different quantization values is minimal. However, reducing precision by even a single bit has an evident effect on power consumption, owing to the large number of units working in parallel.

Finally, as explained in Section VI-D, it is crucial to note that the primary constraint on Spiker+ size is the number of weights that can be accessed in parallel. A smaller weight bit-width reduces the required interconnections and the amount of

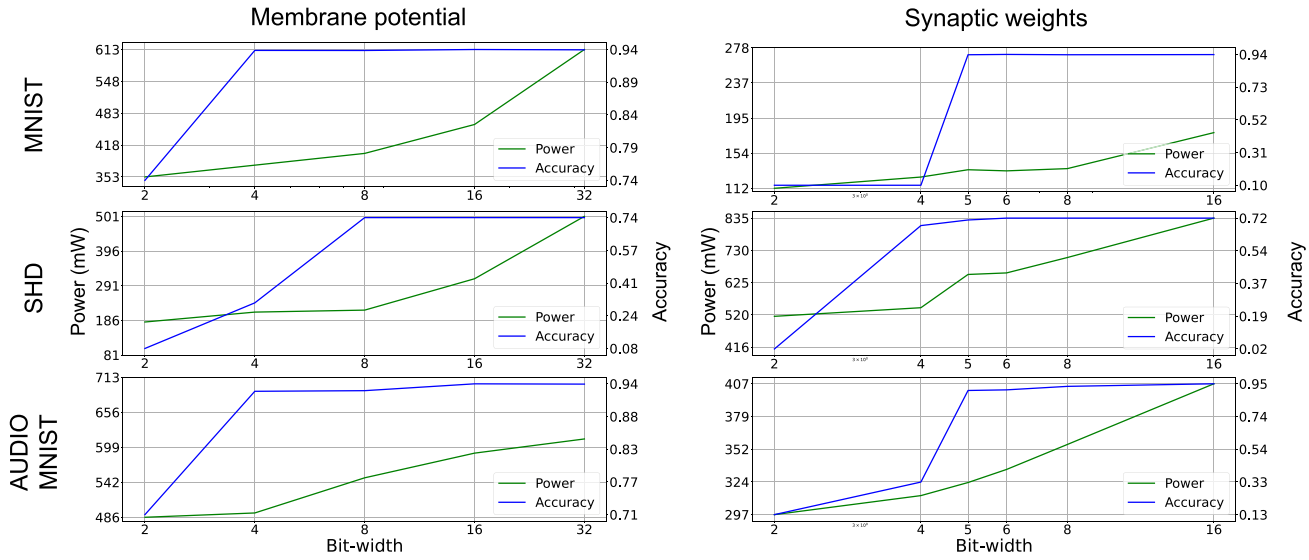


Figure 10. Impact of quantization of neuron membrane potentials and synaptic weights on inference accuracy and power consumption for target datasets.

TABLE IV
SYNTHESIS OF MAXIMUM SIZE ACCELERATOR ON DIFFERENT XILINX™ FPGAS BOARDS

FPGA	Net	Neur.	BRAM	LUT	Pow(mW)
XC7Z020 / XA7Z020	FFFC	1224	138	44330	1070
	RSNN	550	135	24,660	720
XCZU3EG	FFFC	1900	215	62989	1200
	RSNN	690	213	29,809	780

memory used, including the total number of BRAMs. This not only conserves power but also facilitates the implementation of larger architectures.

D. Performance versus Size

Finally, let us explore the model complexity achievable with Spiker+ on selected Xilinx™ FPGA boards. Synthesis results for three Xilinx™ boards, particularly low-end ones suitable for resource-constrained edge applications, are presented in Table IV.

On the Xilinx™ XC7Z020/XA7Z020 boards discussed in Section VI-A, the largest FF-FC network possible using a I-order LIF consists of 1,220 neurons, utilizing 138 BRAMs (98.5%) and 42,430 LUTs (26.7%). The BRAM size of the FPGA emerges as the limiting factor. On the slightly more advanced Xilinx™ XCZU3EG board, a larger 1,900-neuron architecture, utilizing 215 BRAMs (99.5%) and 62,989 LUTs (29.8%), can be implemented. Notably, the place-and-route algorithm encounters no obstacles, reinforcing that BRAM limitations govern the network size.

For FC-R architectures using a II-order LIF, where both feed-forward and feedback weights need storage in BRAM, the maximum network size is influenced. Specifically, it is capped at 550 neurons for Xilinx™ XC7Z020/XA7Z020 boards and 690 neurons for the Xilinx™ XCZU3EG board.

Potential issues such as place-and-route complexities or excessive power consumption due to the fully parallel nature of the accelerator are foreseeable. A prospective solution involves implementing a certain degree of time multiplexing to address these challenges while expanding the architecture size. This approach entails sharing hardware components between neurons, introducing a trade-off with performance, and is currently under study as an enhancement.

VII. CONCLUSION

This paper introduced Spiker+, a versatile framework to design low-power and resource-efficient hardware accelerators for SNNs targeting edge inference on FPGA platforms. It features a Python configuration framework that facilitates easy reconfiguration of the accelerator, allowing users to choose from six neuron models (IF, I-order LIF, and II-order LIF, each with the option of a *hard* or *subtractive* reset) and two network architectures (FF-FC and FC-R). The tool enables the automatic selection of training and quantization parameters directly through Python. The results are significant, boasting a 93.85% accuracy on MNIST, with a classification latency of 780 μs per image and power consumption of 180 mW. Additionally, it achieves a 72.99% and 89.82% accuracy on SHD and AudioMNIST respectively, corresponding to a 540 μs latency and power consumption of 430 mW for the SHD, and 80 μs latency and 290 mW power for the AudioMNIST. These metrics are highly competitive compared to state-of-the-art FPGA accelerators for SNNs, demonstrating high performance in both power efficiency and area. This work lays a solid foundation for deploying specialized, low-power, and efficient SNN accelerators in resource and power-constrained edge applications. Spiker+ is a live project, and ongoing work focuses on enlarging the library of available neurons, input encoding blocks, and network architectures.

DATA AVAILABILITY

To encourage research in this field, Spiker+ is available as an open-source project at https://github.com/smilies-polito/Spiker.

ACKNOWLEDGMENT

The authors would like to thank Domenico Sabella for their valuable assistance in revising and cleaning the final version of the code published on the open repository.

REFERENCES

[1] Z. Chang, S. Liu, X. Xiong, Z. Cai, and G. Tu, "A survey of recent advances in edge-computing-powered artificial intelligence of things," *IEEE Internet Things J.*, vol. 8, no. 18, pp. 13849–13875, Sep. 2021.

[2] J. Xue et al., "EdgeMap: An optimized mapping toolchain for spiking neural network in edge computing," *Sensors*, vol. 23, no. 14, pp. 6548, 2023.

[3] P. Dhilleswararao, S. Boppu, M. S. Manikandan, and L. R. Cenkeramaddi, "Efficient hardware architectures for accelerating deep neural networks: Survey," *IEEE Access*, vol. 10, pp. 131788–131828, 2022.

[4] A. Carpegna, A. Savino, and S. Di Carlo, "Spiker: An FPGA-optimized hardware accelerator for spiking neural networks," in *Proc. 2022 IEEE Comput. Soc. Annu. Symp. VLSI*, 2022, pp. 14–19.

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

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[7] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 7, pp. 2744–2757, Jul. 2022.

[8] S. Becker, J. Vielhaben, M. Ackermann, K.-R. Müller, S. Lapuschkin, and W. Samek, "AudioMNIST: Exploring explainable artificial intelligence for audio analysis on a simple benchmark," *J. Franklin Inst.*, vol. 361, no. 1, pp. 418–428, Jan. 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0016003223007536

[9] D. Padovano, A. Carpegna, A. Savino, and S. Di Carlo, "SpikeExplorer: Hardware-oriented design space exploration for spiking neural networks on FPGA," *Electronics*, vol. 13, no. 9, pp. 1744, Jan. 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/9/1744

[10] B. Yin, F. Corradi, and S. M. Bohté, "Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks," *Nature Mach. Intell.*, vol. 3, no. 10, pp. 905–913, Oct. 2021.

[11] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol.*, vol. 117, no. 4, pp. 500–544, 1952.

[12] E. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

[13] R. Brette and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity," *J. Neuriophysiol.*, vol. 94, no. 5, pp. 3637–3642, 2005.

[14] G. Bellec et al., "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature Commun.*, vol. 11, no. 1, Jul. 2020, Art. no. 3625.

[15] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, Aug. 2019, Art. no. e47314.

[16] N. Rathi et al., "Exploring neuromorphic computing based on spiking neural networks: Algorithms to hardware," *ACM Comput. Surv.*, vol. 55, no. 12, Mar. 2023, Art. no. 243.

[17] A. Basu, C. Frenkel, L. Deng, and X. Zhang, "Spiking neural network integrated circuits: A review of trends and future directions," Mar. 2022, *arXiv:2203.07006*.

[18] F. Pavanello et al., "Special session: Neuromorphic hardware design and reliability from traditional CMOS to emerging technologies," in *Proc. IEEE 41st VLSI Test Symp.*, 2023, pp. 1–10.

[19] C. Mayr, S. Hoeppner, and S. Furber, "SpiNNaker 2: A 10 million core processor system for brain simulation and machine learning," Nov. 2019, *arXiv: 1911.02385*.

[20] F. Akopyan et al., "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.

[21] L. Deng et al., "Tianjic: A unified and scalable chip bridging spike-based and continuous neural computation," *IEEE J. Solid-State Circuits*, vol. 55, no. 8, pp. 2228–2246, Aug. 2020.

[22] O. Richter et al., "DYNAP-SE2: A scalable multi-core dynamic neuromorphic asynchronous spiking neural network processor," Nov. 2023, *arXiv:2310.00564*.

[23] B. V. Benjamin et al., "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[24] C. Pehle et al., "The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity," *Front. Neurosci.*, vol. 16, 2022, Art. no. 795876.

[25] S. Narayanan, K. Taht, R. Balasubramonian, E. Giacomin, and P.-E. Gaillardon, "SpinalFlow: An architecture and dataflow tailored for spiking neural networks," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit.*, 2020, pp. 349–362.

[26] D. Ma et al., "Darwin: A neuromorphic hardware co-processor based on spiking neural networks," *J. Syst. Archit.*, vol. 77, pp. 43–51, Jun. 2017.

[27] C. Frenkel, J.-D. Legat, and D. Bol, "A 65-nm 738k-Synapse/mm^2 quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning," in *Proc. 2019 IEEE Int. Symp. Circuits Syst.*, 2019, pp. 1–5.

[28] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm^2 12.7-pJ/SOP 64k-Synapse 256-Neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 1, pp. 145–158, Feb. 2019.

[29] C. Frenkel and G. Indiveri, "ReckOn: A 28 nm Sub-mm^2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales," in *Proc. 2022 IEEE Int. Solid- State Circuits Conf.*, 2022, pp. 1–3.

[30] D. Neil and S.-C. Liu, "Minitaur, an event-driven FPGA-Based spiking network accelerator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 12, pp. 2621–2628, Dec. 2014.

[31] H. Liu, Y. Chen, Z. Zeng, M. Zhang, and H. Qu, "A low power and low latency FPGA-Based spiking neural network accelerator," in *Proc. 2023 Int. Joint Conf. Neural Netw.*, 2023, pp. 1–8.

[32] S. Gupta, A. Vyas, and G. Trivedi, "FPGA implementation of simplified spiking neural network," in *Proc. 27th IEEE Int. Conf. Electron. Circuits Syst.*, 2020, pp. 1–4.

[33] J. Li, G. Shen, D. Zhao, Q. Zhang, and Y. Zeng, "FireFly: A high-throughput hardware accelerator for spiking neural networks with efficient DSP and memory optimization," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 8, pp. 1178–1191, Aug. 2023.

[34] S. Panchapakesan, Z. Fang, and J. Li, "SyncNN: Evaluating and accelerating spiking neural networks on FPGAs," in *Proc. 31st Int. Conf. Field-Program. Log. Appl.*, 2021, pp. 286–293.

[35] A. Khodamoradi, K. Denolf, and R. Kastner, "S2N2: A FPGA accelerator for streaming spiking neural networks," in *Proc. 2021 ACM/SIGDA Int. Symp. Field- Program. Gate Arrays*, New York, NY, USA, 2021, pp. 194–205.

[36] J. Han, Z. Li, W. Zheng, and Y. Zhang, "Hardware implementation of spiking neural networks on FPGA," *Tsinghua Sci. Technol.*, vol. 25, no. 4, pp. 479–486, Aug. 2020.

[37] G. Zhang et al., "A low-cost and high-speed hardware implementation of spiking neural network," *Neurocomputing*, vol. 382, pp. 106–115, Mar. 2020.

[38] Y. Nevarez, D. Rotermund, K. R. Pawelzik, and A. 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.

[39] H. Asgari, B. M.-N. Maybodi, M. Payvand, and M. R. Azghadi, "Low-energy and fast spiking neural network for context-dependent learning on FPGA," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 67, no. 11, pp. 2697–2701, Nov. 2020.

[40] S. Li, Z. Zhang, R. Mao, J. Xiao, L. Chang, and J. Zhou, "A fast and energy-efficient SNN processor with adaptive clock/event-driven computation scheme and online learning," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 68, no. 4, pp. 1543–1552, Apr. 2021.

[41] N. Abderrahmane, E. Lemaire, and B. Miramond, "Design space exploration of hardware spiking neurons for embedded artificial intelligence," *Neural Netw.*, vol. 121, pp. 366–386, Jan. 2020.

[42] D. Gerlinghoff, Z. Wang, X. Gu, R. S. M. Goh, and T. Luo, "E3NE: An end-to-end framework for accelerating spiking neural networks with emerging neural encoding on FPGAs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 3207–3219, Nov. 2022.

[43] P. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Front. Comput. Neurosci.*, vol. 9, 2015.

[44] W. Fang et al., "SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence," *Sci. Adv.*, vol. 9, no. 40, Oct. 2023, Art. no. eadi1480, doi: 10.1126/sciadv.adi1480.

[45] J. E. Pedersen et al., "Neuromorphic intermediate representation: A unified instruction set for interoperable brain-inspired computing," 2023. [Online]. Available: https://arxiv.org/abs/2311.14641



Alessio Carpegna (Student Member, IEEE) received the MSc degree in electronic engineering from the Politecnico di Torino, Torino, Italy, in 2021, with a specialization in digital systems design. In the same year he entered the Italian national PhD program in Artificial Intelligence. His current research interests include neuromorphic systems, and in particular the design of neuromorphic hardware accelerators, Operating Systems and Embedded Systems in general.



Alessandro Savino (Senior Member, IEEE) received the MS equivalent and PhD degrees in computer engineering and information technology from the Politecnico di Torino, Italy, in 2005 and 2009, respectively. He is an associate professor with the Department of Control and Computer Engineering, Politecnico di Torino, Italy. His research contributions include approximate computing, reliability analysis, safety-critical systems, software-based self-test, and image analysis. He has been part of the program and organizing committee of several IEEE and INSTICC conferences and has served as a reviewer of IEEE conferences and journals. His research interests include operating systems, imaging algorithms, machine learning, evolutionary algorithms, graphical user interface experience, and audio manipulation.



Stefano Di Carlo (Senior Member, IEEE) received the MSc degree in computer engineering and the PhD degree in information technologies from Politecnico di Torino, Italy, where he is a full professor. His research interests include DFT, BIST, and dependability. He has coordinated several national and European research projects. He has published more than 200 papers in peer-reviewed IEEE and ACM journals and conferences. He regularly serves on the Organizing and Program Committees of major IEEE and ACM conferences. He is a Golden Core member of the IEEE Computer Society.

Open Access provided by 'Politecnico di Torino' within the CRUI CARE Agreement