A High-Level Methodology to Evaluate and Optimize Digital Architectures Targeting Spike Encoding

Clémence Gillet, Adrien Vincent, Bertrand Le Gal, Sylvain Saïghi

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 18 August 2023, accepted 10 September 2023, date of publication 16 October 2023, date of current version 2 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3324877

RESEARCH ARTICLE

A High-Level Methodology to Evaluate and Optimize Digital Architectures Targeting Spike Encoding

CLÉMENCE GILLET, ADRIEN F. VINCENT, BERTRAND LE GAL, AND SYLVAIN SAÏGHI, (Member, IEEE)
Univ. Bordeaux, CNRS, Bordeaux INP, IMS, UMR 5218, F-33400 Talence, France

Corresponding author: Adrien F. Vincent (adrien.vincent@ims-bordeaux.fr)

This work was supported by the French National Research Agency (ANR) as part of the "Chaires Intelligence Artificielle (IA)" Program, Green Artificial Intelligence (GrAI) Project, under Grant ANR-19-CHIA-0003.

ABSTRACT Spiking Neural Networks (SNNs) are promising candidates for low-power and low-latency embedded artificial intelligence. However, those networks require event-based data produced by neuromorphic sensors which are not widely available, except for a few specialized devices like neuromorphic retinas. For other data types, a solution lies in the use of conventional sensors in conjunction with encoding layers. However, when performed in software, this solution can be detrimental to energy consumption or latency. Here we introduce a flexible design methodology for efficiently implementing, optimizing, and evaluating digital architectures of spike encoding integrating algorithms available in the literature. In order to quickly evaluate different hardware architectures and to tailor the solution to the application needs, our approach relies on High-Level Synthesis (HLS) tools and Python scripting. We illustrate the methodology by generating various digital architectures of two encoding algorithms taken from the literature and we evaluate their energy consumption and timing performances on Field Programmable Gate Arrays. This work could overcome the lack of neuromorphic sensors and accelerate the development of lower-power hardware SNNs.

INDEX TERMS Spiking neural networks, data encoding, high level synthesis, spike sorting, edge computing.

I. INTRODUCTION

Machine learning revolutionized many fields such as healthcare, industry, automotive, and smart home. Artificial Neural Networks (ANNs) are approaching human performances when addressing sound recognition [1] and object detection [2]. To do so, the complexity of those networks has been continuously increasing for the last 40 years leading to what is now called deep learning [3], [4] due to the many layers required to perform more and more complex tasks.

Until now, ANNs have been mostly implemented on traditional Von-Neumann architectures. However, sequential processing, as done on traditional computers, is not suited for those networks which process data in a parallel manner, leading to extreme amounts of power consumption and processing time. As a result, dedicated hardware accelerators have been developed to improve parallelism and lead to even more efficient systems.

Other types of neural networks, like Spiking Neural Networks (SNNs) [5], benefit greatly from these architectural improvements, paving the way for very low energy-consuming designs. SNNs are energy-efficient bio-inspired systems, making them candidates of choice for future embedded systems [6], [7]. SNNs typically need fewer layers than their Deep or Convolutional Neural Networks (DNNs or CNNs) counterparts [8], reducing the energy consumption and the required processing power, which makes them a promising approach to building systems better suited for embedded use.

The theoretical scope of application for neuromorphic systems is large but their real-world use cases are currently limited by the lack of sensors able to communicate directly

The associate editor coordinating the review of this manuscript and approving it for publication was Yassine Maleh.

with bio-inspired neural networks which are event-driven and thus communicate through spikes. Only a few sensors natively output spikes and they are often dedicated and optimized for a particular type of signal, e.g., retinomorphic sensors for event-based vision processing [9], [10]. The scarcity of event-based sensors leads developers to use regular data output by conventional sensors and to add dedicated encoding layers to generate spikes from frame-based data. Such conversion systems have been studied by simulation means in the literature [11], [12] or have reached hardware implementations for some of them [13], [14], [15]. The latter, however, remain difficult to easily and quickly adapt to an application different from the one used during their specific design.

This work introduces a methodology to generate, optimize and evaluate digital spike encoding architectures targeting FPGAs and paving the way for ASIC design. The workflow can implement many different encoding algorithms and adapt them to a broad panel of time series (or 1D data types). For demonstration purposes, we applied this methodology to a cost-optimized FPGA using a framework based of High-Level Synthesis tools and automated Python scripts. We chose to illustrate the methodology using two encoding algorithms taken from the literature and we evaluated their performances in terms of resource utilization, timing performances, and energy consumption.

This paper is organized as follows: in Section II, recent approaches for spike encoding are detailed. Section III describes the implementation methodology. The generated architectures are evaluated on FPGA devices in Section IV and conclusions are drawn in Section V.

II. RELATED WORKS

A. SPIKING NETWORKS

Spiking Neural Networks (SNNs) [5] are an alternative to now widely spread Deep Neural Networks which perform state-of-the-art pattern recognition and classification tasks [3], [4] but at the cost of a significant amount of hardware resources and energy consumption, despite optimizations like pruning [16], [17] and reduced-precision operations [18], [19], [20], [21].

SNNs take inspiration from the biology [22], [23], [24], [25], [26], [27], with a broad range of neuron models ranging from the most bio-realist to the least computationally demanding [28], [29], [30]. For example, they are especially suited for unsupervised learning: one of the main challenges of future artificial intelligence systems [31], e.g., when coupled to the bio-inspired *Spike Timing-Dependent Plasticity* (STDP) learning rule [32]. The key advantage of SNNs lies in their event-based nature contrary to conventional neural networks which consume and produce data on a regular time period. This data scarcity leads to fewer data movements and thus is the key to reducing the energy consumed by transmissions to and from the outside of the systems [33]. However, their event-based nature also prevents SNNs from being able to directly process data generated by most sensors on the market. It is thus necessary to either use inherently event-based sensors [34] which are limited in terms of application or to pre-process data acquired by conventional sensors and convert them into spikes.

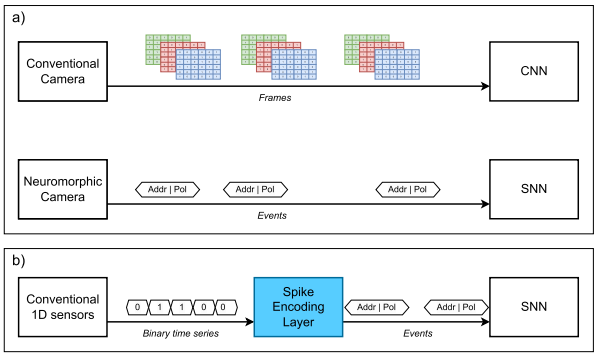

FIGURE 1. a) Existing methods for pattern recognition using CNNs and SNNs b) Pattern recognition flow using an SNN and data from a conventional sensor. The encoding layer is required to convert continuous digital data into events.

To the best of our knowledge, despite being promising for building such a generic interface between non-spiking sensors and SNNs, no generic hardware implementation of the bio-inspired spike-encoding methods presented in the next subsection has been reported in the literature so far.

B. CONVENTIONAL AND BIO-INSPIRED SENSORS

Electronic sensors are devices that monitor physical phenomena and translate information so electronic systems can use them. Conventional sensors used by CNNs output data in the form of continuous frames as shown in Figure 1. However, unlike CNNs, SNNs use an asynchronous flow of spikes that differ from frame-based data.

To process data using SNNs, a solution lies in the use of sensors that integrate bio-inspired hardware, such as neuromorphic cameras (see Figure 1a) implementing delta-modulation encoding, i.e., each pixel emits a spike when it receives a pre-determined change of brightness. To the best of our knowledge, video, audio streams, and olfactory data are the only data types to be addressed by such sensors as there is no spike-producing sensor for continuous streams of data such as space position or bio-signals acquired by electrodes for example.

The approach we chose to adapt frame-based sensor outputs to SNN consists in adding an encoding layer between the sensor and the network, as shown in Figure 1b.

For demonstration purposes, two algorithms taken from the literature are integrated into the methodology developed in this paper as they seem promising to encode time series generated by conventional 1D sensors and convert them into spikes suited for SNN processing. Both of these algorithms can be adapted to various dynamic ranges. The first algorithm is frequency-based and allows various neural encodings such as temporal coding, rate-based coding,

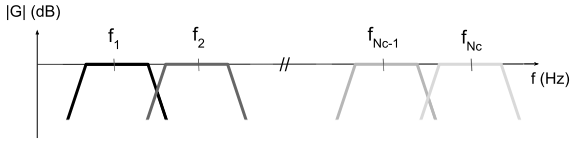

FIGURE 2. Theoretical bode magnitude plot of the different band-pass filters used in the frequency encoding algorithm.

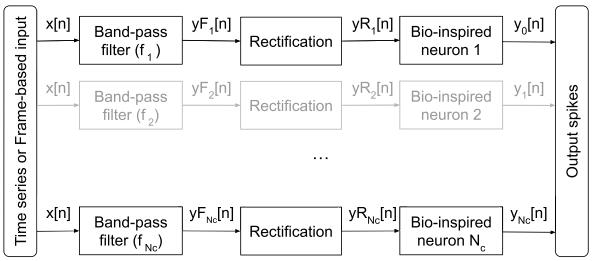

FIGURE 3. Generic frequency encoding with N_C channels. Each channel is composed of a band-pass filter, a rectification stage, and a spiking neuron.

Algorithm 1 Pseudocode for Frequency Encoding

$mp_k[0] \leftarrow mp_{\text{resting}}$
for $n = 1$ to Number of input samples **do**
  for $i = 0$ to M **do**
    $yF_k[n] \leftarrow yF_k[n] + b[i] \times x[n-i]$
  end for
  for $j = 1$ to N **do**
    $yF_k[n] \leftarrow yF_k[n] + a[j] \times yF_k[n-j]$
  end for
  $yR_k[n] \leftarrow \text{abs}(yF_k[n])$
  $mp_k[n] \leftarrow mp_k[n-1] + yR_k[n] - L_k$
  if $mp_k[n] > Thr$ **then**
    $mp_k[n] \leftarrow mp_{\text{resting}}$
    $y_k[n] \leftarrow 1$
  else
    $y_k[n] \leftarrow 0$
  end if
end for

or delta modulation [11] (see section II-B1. The second algorithm is amplitude-based [12] (see section II-B2).

1) FREQUENCY-BASED ENCODING

Frequency encoding [11] is a type of vector embedding used to transform time series (audio, biophysical recordings...) into vectors as done for image embedding but using a time-frequency representation. The frequency-based encoding relies on the fact that, in the case of neural networks used for clustering, the patterns to discriminate exhibit unique frequency identities and thus can be identified through a frequency representation rather than their amplitude span.

To perform this encoding, after normalizing the signal if needed, the first step consists in injecting the input signal through a bank of band-pass filters with central frequencies gradually increasing through a predefined frequency range, as illustrated in Figure 2.

To translate the signal into spikes that encode the density of information in different frequency ranges, the filter outputs are full-wave rectified and sent to spiking neurons, e.g., leaky integrate and fire neurons [29] as shown in Figure 3. The association of a band-pass filter, a rectification stage, and a spiking neuron forms a channel.

To obtain an encoding that suits the data, filters are designed to fit the frequency range of interest, and a trade-off between frequency and temporal resolution has to be found. The union of all the filter bandwidths must cover the frequency range of the signal. Consequently, a large number of filters will lead to a high-frequency description but it will also require more bio-inspired neurons to process the information. Infinite and Finite Impulse Response (IIR and FIR) filter models can be used depending on the precision, timing, and hardware constraints. The neuron layer in charge of encoding the signal into spikes is composed of bio-inspired neurons such as Leaky Integrate and Fire (LIF) neurons which mimic the biological behavior without being too computationally demanding. LIF neurons are defined by parameters such as their threshold value, a high threshold leading to fewer spikes for instance.

In order to be energy efficient while transmitting as much information as possible through spikes, a trade-off must be found between the neuron spiking rate and the impact on the result of the downstream SNN.

The pseudocode for the frequency encoding using LIF neurons and IIR filters is given in Algorithm 1 where:

- n is the input sample index;
- $x[n]$ is the frame-based input signal;
- k is the channel index;
- $y_k[n]$ is the channel output (spike);
- M is the feedforward filter order;
- $b[i]$ are the feedforward filter coefficients;
- N is the feedback filter order;
- $a[j]$ are the feedback filter coefficients;
- mp_k is the membrane potential of the k^{th} neuron;
- mp_{resting} is the resting potential of the neuron;
- Thr is the neuron threshold;
- L_k is the leak rate of the neuron.

2) AMPLITUDE-BASED ENCODING

The amplitude-based encoding algorithm [12] can be used to convert any type of continuous time series into a spike train. It relies on population neural coding and it is similar to position coding as the core units are "comparator neurons" that spike at a predefined rate if the input is within their sensitivity range, which is defined by a central value, and a range as shown in Figure 4.

Sensitivity centers are typically regularly spaced and neurons' sensitivity ranges can overlap so that for each input value, a fixed predetermined number of neurons $N_{overlap}$

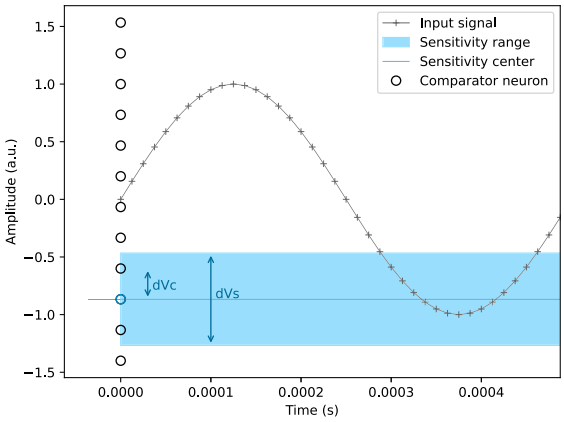

FIGURE 4. Illustration of the amplitude-based encoding algorithm. The input is a sine wave at 500 Hz and sampled at 80 kHz. The comparator neurons have a sensitivity range of dV_s and the space in between sensitivity centers is equal to dV_c.

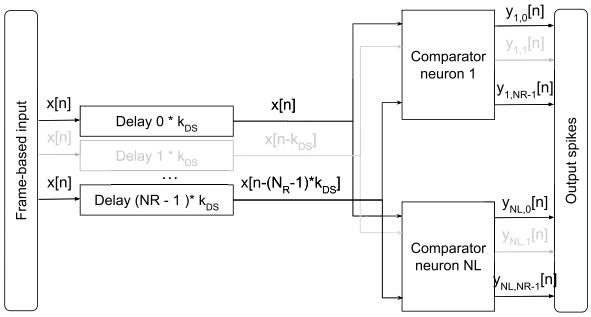

FIGURE 5. Algorithm of the amplitude encoding with N_L comparator neurons and N_R delays.

emits spikes. Consequently, the union of all sensitivity ranges covers all possible signal values. Neurons' sensitivity range dV_s is chosen depending on the noise level in the input signal.

To target SNNs that perform pattern recognition, a solution lies in ensuring that the entire waveform of a pattern to recognize is considered. That is why this algorithm does not only encode the current sample but also outputs the spikes corresponding to the encoding of the previous sample within a sliding time window fitting the pattern length as shown in Figure 5.

Various spiking behaviors can be chosen to encode the signal. For instance, the most naive behavior, called binary spiking is to have a neuron that emits a single spike every time its input is within its sensitivity range. Rate-based coding can also be integrated with neurons emitting spikes whose density depends on the difference between the input sample and the sensitivity center of the neuron. The closer the sample is to the neuron's sensitivity central value, the higher the spike density.

The pseudocode for the amplitude encoding is given in Algorithm 2 where:

- n is the input sample index;
- $x[n]$ is the frame-based input signal;
- N_R is the number of delays;
- N_L is the number of comparator neurons;

Algorithm 2 Pseudocode for Amplitude Encoding (binary Spiking version)

```
for n = 1 to Number of input samples do
  for i = 1 to N_L included do
    for j = 0 to N_R excluded do
      if LowThr[i] < x[n−j × k_DS] < UpThr[j] then
        y_{i,j}[n] ← 1
      else
        y_{i,j}[n] ← 0
      end if
    end for
  end for
end for
```

- k_{DS} is the optional (integer) down-sampling factor within the time window;
- dV_c is the amplitude difference between two comparator centers;
- dV_s is the sensitivity range of a comparator neuron;
- i is the neuron index;
- j is the delay index;
- $LowThr$ is the list of neuron lower thresholds;
- $UpThr$ is the list of neuron upper thresholds.

The algorithm can be viewed as a grid of $N_L \times N_R$ sensory neurons, where N_L is the number of neurons that are responsive to the same amplitude range but receive data corresponding to different processing delays of the input signal. On the other hand, N_R is the number of neurons within a row that have different sensitivity ranges but receive the same input.

The number of neurons N_L with different sensitivity centers (i.e., the number of lines) depends on:

- the upper and lower values $\mathrm{sig}_{\min}$ and $\mathrm{sig}_{\max}$ that can be reached by the signal;
- the number of neurons N_{overlap} whose sensitivity ranges are overlapping (i.e., number of neurons that will spike at the same time for the same value);
- the sensitivity range of each neuron $\mathrm{d}V_s$.

The number of lines N_L is given by the following formula:

$$N_L = \lceil(\mathrm{sig}_{\max} - \mathrm{sig}_{\min} + \mathrm{d}V_s) \times N_{\mathrm{overlap}}/\mathrm{d}V_s\rceil + 1$$

The value between two sensitivity centers $\mathrm{d}V_c$ is:

$$\mathrm{d}V_c = \mathrm{d}V_s/N_{\mathrm{overlap}}.$$

The number of neurons N_R with the same sensitivity centers and ranges but that see different delayed input values depends on the duration of the waveform to discriminate. For spike sorting applications, it has been shown that choosing a window duration T_{window} slightly shorter than the pattern of interest yields the best results [12].

For the time window, a good trade-off consists in choosing a downsampling frequency f_{ds} that is an integral multiple of the original frequency f_s, i.e., a downsampling factor $k_{ds} = f_s/f_{ds} \in \mathbb{N}$.

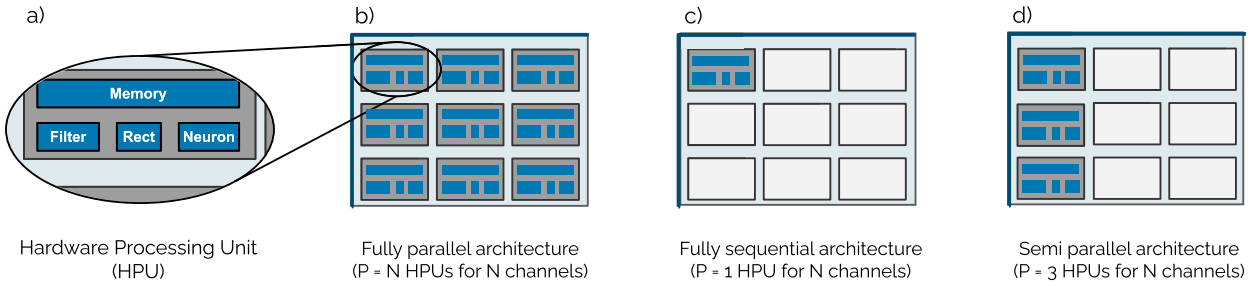

FIGURE 6. Various architectures to implement a frequency encoding algorithm as described in Algorithm 1 on FPGA through different HPU re-use rates.

A trade-off must be found regarding the number of spikes emitted. If there are too few spikes produced, it will result in a less accurate description of the signal. But on the other hand, generating too many spikes will be counterproductive in terms of data scarcity. Despite this encoding method being less computationally demanding than frequency encoding, it produces a large number of spikes as for each new input sample $k_{ds} \times N_R \times N_{\text{overlap}}$ neurons fire. The encoding algorithm must thus be chosen and adapted to the system constraints.

C. MOTIVATIONS

MATLAB implementations of both of these algorithms have been integrated into SNNs applied to spike sorting [11], [12]. Additionally, [11] explored hardware implementations that integrate analog components and nano-devices by simulation means that yielded satisfying results. Nano-devices lack maturity however and may suffer from a drift of performance which is why, no hardware implementation has been reported so far.

In this article, we propose a generic methodology to generate efficient hardware digital encoding layers implementing existing algorithm behavior, tackling the lack of hardware spiking sensors.

For demonstration purposes, the performances of different hardware implementations of the previously described algorithms are evaluated on Field Programmable Gate Arrays (FPGAs).

The advantage of FPGAs lies in their design flow that allows relatively easy programmability while achieving high performances at low power costs. FPGAs are chips that use configurable logic blocks and programmable interconnects. This enables updating the encoding layer and modifying the required model. Our methodology allows an exact determination of the optimal number of resources for a given encoding layer. Once optimized, the encoding architecture could be converted into an ASIC to further reduce energy consumption if needed.

III. METHODOLOGY / TARGETED ARCHITECTURE

A. INTRODUCTION

Encoding layers for SNNs can be used in various contexts that may require hundreds of channels for some and only a few in other cases with the same latency and/or throughput constraints. Additionally, a majority of systems require reduced energy consumption. Consequently, different kinds of digital architectures and optimization strategies are possible depending on the constraints.

A solution lies in the use of a system based on what we call Hardware Processing Units (HPUs). An HPU is composed of the digital hardware implementation of a core cell adapted from the targeted algorithm. In the case of frequency encoding, an HPU can be composed for instance of a band-pass filter, a rectification function, and a spiking neuron (Figure 6a). In the case of amplitude encoding, an HPU is a single "comparator" neuron and the encoding is done by sending the input vector composed of the latest samples to the HPU as well as the upper and lower bound of the neuron.

A naive approach consists in using as many HPUs as channels in the algorithmic description (Figure 6b). However, this approach results in a complex hardware system with a low hardware utilization rate, especially if the input data rate is slow compared to the latency of an HPU.

The opposite strategy consists in implementing a single HPU and reusing it as many times as needed to apply the encoding in a sequential manner (Figure 6c). In this case, the hardware complexity is drastically reduced, but a single HPU might not be fast enough to process all the channels one after the other under real-time constraints.

To determine which configuration better suits the constraints, virtually every HPUs combinations can be tested using the methodology (Figure 6d).

The optimal strategy consists in developing a hardware system that uses the smallest amount of resources to minimize its usage but with a latency that satisfies the real-time constraints. For instance, the implementation of the filters used in the frequency-based encoding can be sequential, semi-sequential, parallel, or pipeline depending on the timing and resource constraints [35], [36].

The number of possible solutions to develop and compare increases quickly leading to the need for an automatic methodology to explore the design space depending on the application targeted. We introduce in the rest of the paper a workflow based on HLS to address that problem by producing hardware architectures from behavioral models in an automated manner.

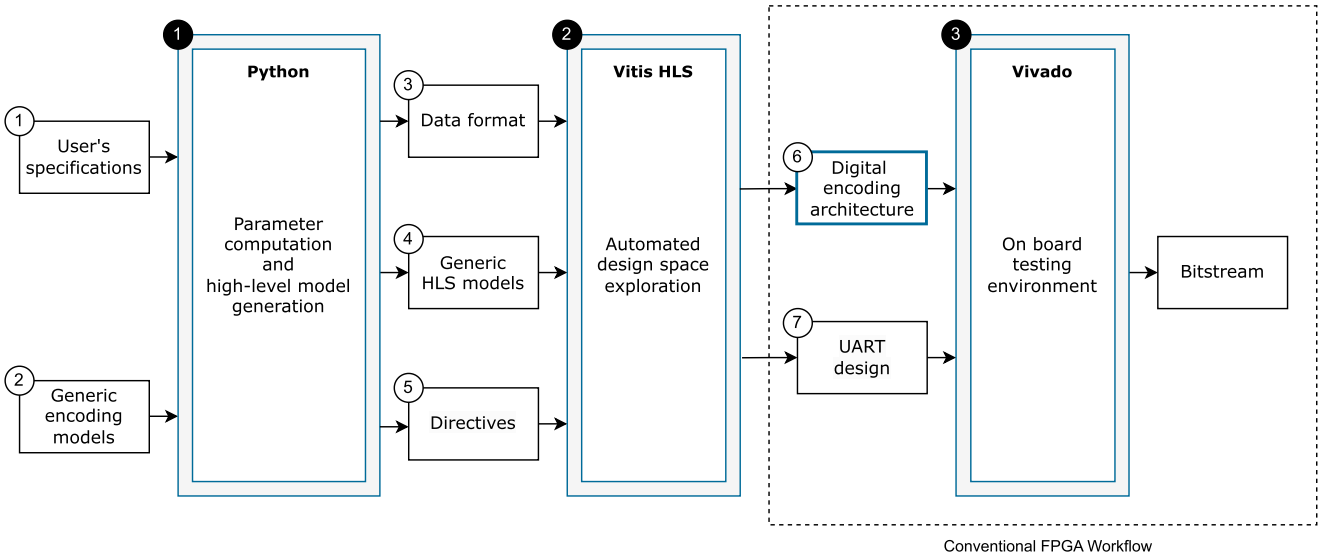

FIGURE 7. Workflow developed to generate the encoding IPs (f) using Python scripts (❶), Xilinx Vitis HLS tool (❷), and Xilinx Vivado software (❸).

B. HIGH LEVEL SYNTHESIS

To efficiently produce a large variety of digital architectures from algorithmic models, we chose to use HLS tools, which are especially well suited for the task [37], [38] and are widely used by academics and in the industrial community in other fields [39], [40]. HLS tools take an algorithmic description written in a high-level language like C/C++ and generate an architecture at register-transfer level (RTL). This hardware description of the architecture can then be integrated into more complex designs and can be processed by conventional logic synthesis tools to program FPGAs or design ASICs. We introduce an HLS-based methodology that provides an easy assessment of energy use and hardware complexity. As opposed to describing directly the architecture using conventional hardware description languages such as VHDL or Verilog which requires a long development process and thus limits the design-space exploration possibilities.

However, achieving acceptable performances in terms of timing and resource utilization requires well-written high-level behavioral models and finely-tuned directives to guide the HLS tool to meet the desired performance goals. Consequently, architectural optimizations were identified to structure the encoding algorithms and to focus on fine-grained microlevel architectural optimizations to reach the targeted performances. These optimizations are described in part III-D2.

C. DETAILED WORKFLOW

Our methodology analyzes, identifies possible architectures, optimizes quantification format, and then generates an architecture of a spike encoding algorithm according to application constraints, such as sensors' output data types, timing requirements, energy consumption targets, and the downstream SNN. The workflow is reported in Figure 7 where a Xilinx FPGA is used to demonstrate the abilities of this methodology relying on the use of Xilinx Vivado 2022.2 for implementation (❸) and Xilinx Vitis HLS 2022.2 for high-level synthesis and design space exploration (❷). However, the methodology could be adapted to other HLS software or even apply to ASIC development flow. To improve flexibility and genericity, a Python layer (❶) is added before the HLS layer and generates high-level models.

Virtually every encoding algorithm can be implemented using our methodology provided that a high-level behavioral description is available. For demonstration purposes, we implemented and reviewed the models described in section II-B1 and II-B2.

1) PARAMETER COMPUTATION AND HIGH-LEVEL MODEL GENERATION

The starting point of the methodology are the user's specifications (①) which include the parameters for the chosen encoding algorithm and the targeted performances in terms of timing and resource utilization. Generic models (②) for both encoding algorithms described above are used to generate HLS models that fit the user's needs.

The Python framework (❶) is used to:

- compute the most suitable data format (③);
- generate an HLS encoding model (④);
- produce the suitable HLS directives (⑤);
- generate synthetic data to test the system in simulation and on the hardware target (here, an FPGA).

The testing methodology is described in Section III-C3. Another goal of this phase is to evaluate the impact of the data format.

2) AUTOMATED DESIGN SPACE EXPLORATION

To obtain an estimation of the hardware complexity and timing performances of the system, we automatized design

space exploration using Vitis HLS tools (Figure 7B). This allows a quick and efficient evaluation of multiple digital architectures' ability to satisfy the constraints.

Generic HLS models (④) are selected and refined using HLS directives(⑤) and suitable data formats (③) to optimize memory usage, parallelism level, and architecture interfaces.

Once the design has met the timing constraints and the resource constraints (e.g., after having specified a suitable parallelism level), one can then try to further optimize it using finer-grain directives (to improve data access using a more suitable memory partitioning for instance) in order to explore more in depth the trade-offs in the design space around the previous hardware solution. One should note that such a tuning process is easier to iterate as the high-level description does not need heavy rewriting for further optimization (but rather minor changes in terms of HLS macros instead) contrary to what a more conventional RTL-based design approach relying directly on writing (lower-level) hardware description with languages like VHDL or Verilog would require.

3) ON BOARD TESTING ENVIRONMENT

The next step instantiates the generated architecture of the encoding layer (⑥) in a testing environment to observe the performances in a real environment. Testing tools are added to the system and integrate test data and communication interfaces (⑦). The energy consumption and the behavior of the system can be precisely obtained at this step enabling validation of the encoding architecture and the constraints.

Though energy consumption can be estimated by Vivado, we chose to use a Nordic Semi Power Profiler Kit 2 (PPK2) in order to get real-world values of the static and dynamic currents consumed by the board. Thus we can precisely evaluate if the energy consumption of the circuit matches the constraints if required [41], [42].

As shown in Figure 8, our testing environment is tailored to measure or validate 4 main parameters:

- energy consumption;
- latency and throughput;
- resource utilization;
- precision and accuracy.

D. SATISFYING THE CONSTRAINTS USING VITIS HLS

Spike generation depends on the chosen encoding algorithm. However, various levels of hardware optimization can be explored depending on the hardware and timing constraints.

1) DATA FORMAT OPTIMIZATION

In order to produce more efficient architectures, a fixed-point data format is used throughout the entire workflow instead of a floating-point data format. The latter may allow for a higher resolution but is not hardware friendly as it necessitates dedicated architecture and floating operations require significantly more clock cycles and hardware resources than fixed point data format. This latter, however, inherently leads

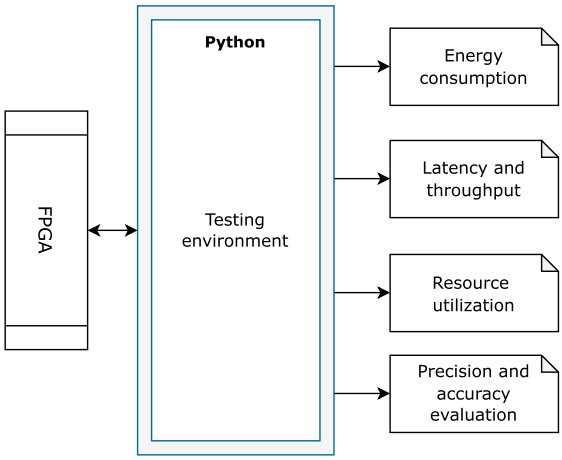

FIGURE 8. Testing environment for the encoding architectures.

to a loss in precision which has an impact on the entire data flow, especially in the case of an encoding algorithm that includes IIR filters where numerical instability may often be an issue. Before hardware generation, the Python framework computes the smallest data format factor for all the variables while keeping an acceptable precision loss through simulation. Floating point conversion into fixed-point format is done using the Vitis HLS `ap_fixed` library. This tool automatically converts floating point data into fixed point representation when given a word length, a number of bits used to represent the integer part, the quantization mode and the saturation mode.

Sections IV-B2 and IV-C2 provide the results of the data format optimization process for both encoding algorithms under test.

One can also note that other data formats, such as Residue Number System (RNS) or posit data format, could also be integrated into the methodology as long as one provides the relevant high-level model descriptions.

2) ARCHITECTURE OPTIMIZATIONS

In order to obtain the best implementation of the architectures, different strategies are applied. Architectures are tuned depending on application constraints and operations may be done in parallel if timing constraints require it. To make the best out of it, fine-grain-level parallelism is used. A rigorous description of the behavior is essential and the use of directives, such as the unroll directive emphasizes the architectures' parallel organization. In this case, memory partitioning is mandatory to allow parallel access to the data. Architectures are also pipelined to improve throughput and latency in order to mask memory access time and high-latency operations. All processing steps are finally put in parallel by dataflow directives.

IV. METHODOLOGY EVALUATION

A. HARDWARE DESIGN

To illustrate the flexibility of the methodology and to evaluate the performances of generated hardware architectures,

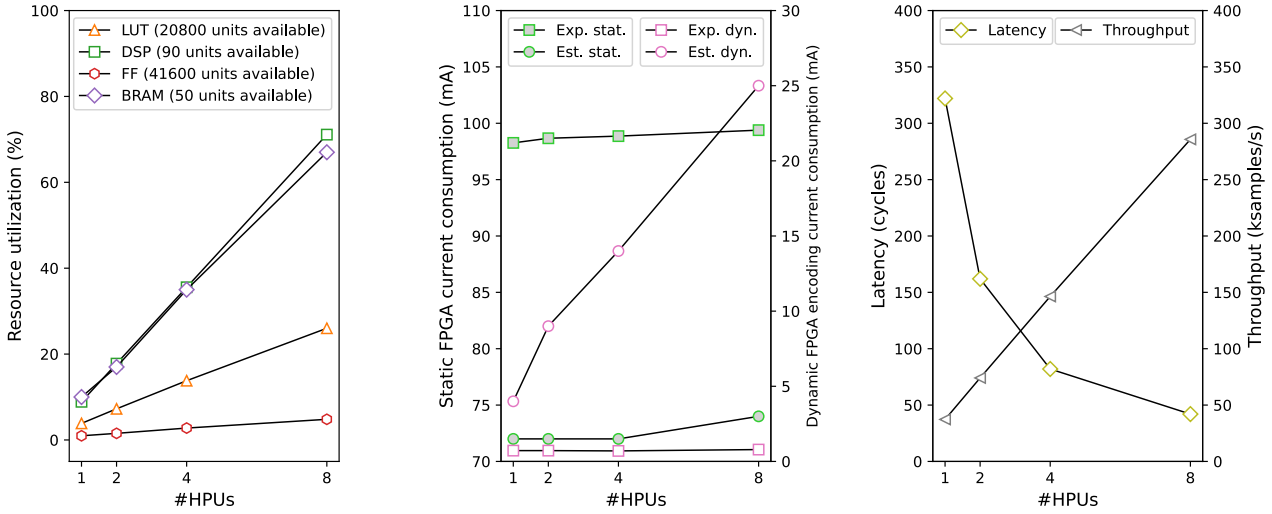

FIGURE 9. Frequency-based encoding layer: *Left panel:* resource usage on an Artix-7 FPGA embedded on a Digilent Cmod A7-35T board regarding the amount of hardware processing units, going from 1 to 8, for a layer composed of 32 channels. The results are produced with the proposed flow and the figures are post-place-and-route results. NB: LUTs stands for Look-Up Tables and FFs for Flip-Flops. *Middle panel:* static and dynamic current consumption of the architectures. Experimental results are measured by the Nordic Semiconductor Power Profiler Kit II (PPK2), and the estimated consumption is obtained using Vivado Power Estimator. *Right panel:* timing summary for 1, 2, 4, and 8 hardware processing units. The targeted clock frequency is 12 MHz and the sample rate is 25 kHz.

both encoding algorithms reported in the II-B related work section were considered.

Xilinx 2022.2 suite (Vitis HLS and Vivado) was used to perform logic synthesis, place-and-route, and bitstream generation. Timing performances were evaluated by Vitis HLS and hardware resource utilization was obtained after implementation by Vivado.

The architectures under study have been synthesized and deployed on a cost-optimized Xilinx Artix 7 35T FPGA (on a Digilent Cmod A7 board). The hardware complexity is given in real-world conditions and the current consumption is experimentally evaluated using a Nordic Semiconductor Power Profiler Kit II.

B. FREQUENCY ENCODING RESULTS AND OPTIMIZATION

1) MODEL PARAMETERS

To evaluate the efficiency of the frequency encoding, different trade-off architectures are designed and evaluated to encode biological neuron recordings similarly to what is reported in [11].

To match the work of [11], $N_L = 32$ channels are implemented. Butterworth band-pass filters are used to decompose the signal. Gammatone filters are also suitable to perform a similar frequency decomposition, but have not been selected for demonstration purposes in this work. The range of central frequencies spans from 100 Hz to 2 kHz and the bandwidth is 60 Hz. To limit latency and instability, the filter order is fixed at 2. Leaky Integrate-and-Fire neuron model is used to convert the rectified filtered data into spikes. The target input data rate is 25 kHz which is a good representation of typical biological recording data rates. Parameters are summed up in Table 1.

TABLE 1. Frequency encoding parameters.

Parameters	Description	Value
FM	Filter model	Band-pass Butterworth
$order$	Filter order	2
BW	Filter bandwidth	60 Hz
fc_{min}	Lowest central frequency	100 Hz
fc_{max}	Highest central frequency	2 kHz
N_C	Number of channels	32

2) OPTIMIZING DATA FORMAT

Digital infinite impulse response filters require high-precision data and coefficients to avoid instability. Data format was studied by comparing theoretical filter outputs using floating point data format to the behavior when using fixed point data format. In order to ensure a Root Mean Square Error (RMSE) smaller than 10^{-2} (a suitable threshold based on empirical observations) between fixed point format and floating point format filter output, coefficient data width is set at 32 bits and input data width is set at 8 bits.

3) ARCHITECTURE AND HARDWARE UTILIZATION

To evaluate different architectures, $P_{\text{HPU}} = 1, 2, 4,$ and 8 are instantiated, with a reuse rate N_L/P_{HPU} of 32, 16, 8, and 4 respectively. The timing, resource utilization, and current consumption are reported in Figure 9.

The resource utilization linearly increases until 8 HPUs are used. For 16 HPUs, the internal data become too large to fit in BlockRAMs (BRAMs) and the number of Look-Up Tables (LUTs) needed exceeds the amount available on the board. Consequently, an architecture instantiating 16 or more HPUs cannot be implemented on the cost-optimized targeted FPGA.

When taking timing performances into consideration, the design clock frequency must be adapted depending on the latency and the input sampling frequency (25 kHz in this example). For instance, with 8 HPUs implementing a total of 32 channels, 42 clock cycles (3.5 µs at 12 MHz) are needed to process one input signal sample. Consequently, the targeted clock period should be at most 40 µs / 42 ≈ 952 ns (1.05 MHz), which allows satisfying the real-time constraints as the critical path is up to 11.79 ns. On the opposite, using 1 HPU to process 32 channels requires 322 clock cycles (26.8 µs at 12 MHz) to encode one sample. If the data input rate is 25 kHz, the targeted clock period should be at least 40 µs / 322 ≈ 124 ns which is still enough to satisfy the constraints as it is longer than the critical path.

4) ENERGY CONSUMPTION AND TIMING PERFORMANCES

In addition to the throughput and resource utilization, energy consumption can be evaluated for each architecture under test. Depending on the application constraints, a suitable trade-off must be found between for instance low resource-consuming architectures that need a high clock frequency and more complex architectures running slower. As demonstrated in Figure 9, middle panel, experimental static current consumption measured on the target varies only slightly (by less than 2%) between an architecture with 1 HPU and an architecture with 8 HPUs though hardware resources are multiplied by 8. Additionally, experimental dynamic current consumption remains around 0.75 mA for every architecture. However, a higher number of HPUs leads to a higher throughput, which allows for a higher input data rate if needed. For instance, given a processing latency of $PL_{\text{HPU}=1} = 324$ cycles and an FPGA clock of $f_{\text{FPGA}} = 12$ MHz, 1 HPU allows for at most an input frequency of $f_{FPGA}/PL_{\text{HPU}=1} = 37$ kHz whereas 8 HPUs can process data arriving 8 times faster, i.e., at $f_{FPGA}/PL_{\text{HPU}=8} = 285$ kHz. Estimations using Vivado Power Estimator are not always reliable [41], [42]. Here, the dynamic consumption is highly overestimated whereas the static current is underestimated by 27%. Those results show the benefits of such a framework capable of efficiently generating many architectures to evaluate them in real conditions.

C. AMPLITUDE ENCODING RESULTS AND OPTIMIZATIONS

1) MODEL PARAMETERS

Similar to frequency encoding, amplitude encoding requires the design and evaluation of multiple trade-off architectures, utilizing parameters similar to those outlined in [43] (see Table 2).

Instead of processing the delayed input N_R times with different values to encode a pattern's behavior through a sliding time window, the result of the encoding of the current input sample is stored for a time T_{window} equal to the sliding window. That way, only $f_s \times T_{window} \times N_L$ bits are stored instead of N_R data that have to be processed again, with f_s being the input sampling frequency.

TABLE 2. Amplitude encoding parameters.

Parameters	Description	Value
$\mathrm{d}V_s$	Sensitivity range size	0.066 to 1.333 a.u.
N_{overlap}	Number of neurons triggered at the same time	10
f_{ds}	Sampling frequency within the time window	20 kHz
N_R	Number of delays	10
N_L	Number of comparator neurons	26 to 312
$A_{\max}$	Maximum amplitude of the input signal	1 a.u.

The signal amplitude range defines the number of neurons N_L that is necessary within a row of the encoding layer. The center of each neuron's sensitivity range depends directly on the span of the signal to encode, the number of neurons necessary, and the overlapping factor.

In the case of this encoding algorithm, an HPU consists in a simple fixed point precision comparator that takes three values: the current signal input and two bounds.

2) OPTIMIZING DATA FORMAT

To optimize the design, the first step consists in processing the most fitting data format factor for the input and the sensory neuron bounds. The data format factor is given by:

- $I_{\text{DATA}} = \lceil \log_2(A_{\max}) \rceil$
- $W_{\text{DATA}} = \lceil I_{\text{DATA}} + 1 - \log_2(\mathrm{d}V_c) \rceil$

with I_{DATA} the number of bits dedicated to the integer part, and W_{DATA} the total bit length of the word.

This step optimizes memory utilization and operations are thus done on the smallest data possible reducing the hardware complexity of the processing.

3) ARCHITECTURE, HARDWARE UTILIZATION, AND TIMING PERFORMANCES

The number of HPUs depends on the resources available, the number of comparator neurons, and the input data rate. As in [12], the input data rate f_s is set at 80 kHz, and the FPGA clock frequency f_{FPGA} is fixed at 12 MHz. The data format is defined by the formula given in section IV-C2. When done sequentially and given a safety margin ε and a processing latency $PL_{\text{HPU}=1} = 1$ cycle the maximum number of comparator neurons per HPU (or re-use rate) is given by $f_{\text{FPGA}}/f_s - \varepsilon$. With a safety margin of $\varepsilon = 6$ cycles being enough to ensure a throughput higher than the input data rate, the maximum HPU re-use rate is 144.

Resource usage and timing performances are displayed in Figure 10 (left and right panels). In particular, one can observe jumps in terms of BRAM usage, latency or throughput, when exceeding the maximum HPU re-use rate and thus switching from one to two HPUs.

4) ENERGY CONSUMPTION

Similar to what is done in IV-B4, Figure 10, middle panel, shows that the experimental static current consumption stays almost the same despite variations in the number

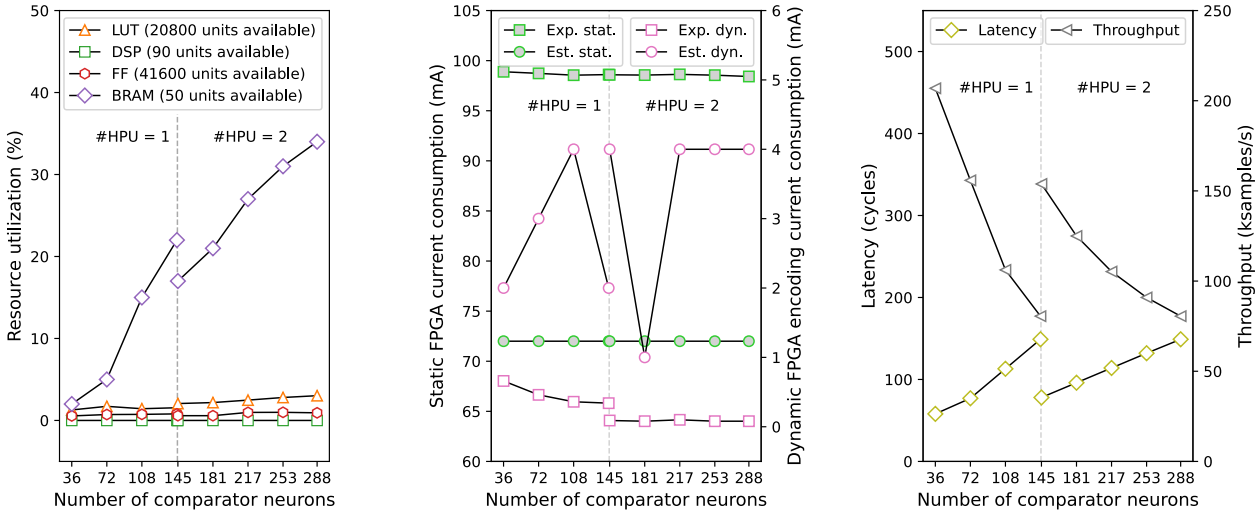

FIGURE 10. **Amplitude-based encoding layer: *Left panel:* resource usage on an Artix-7 FPGA embedded on a Digilent Cmod A7-35T board with 1 or 2 HPUs and 36, 72, 108, 144, 145, 181, 217, and 288 comparator neurons. The results are produced with the proposed flow and the figures are post-place-and-route results. NB: LUTs stands for Look-Up Tables and FFs for Flip-Flops. *Middle panel:* static and dynamic current consumption of the architectures. Experimental results are measured by the Nordic Semiconductor Power Profiler Kit II (PPK2), and the estimated consumption is obtained using Vivado Power Estimator. *Right panel:* timing summary for 36 to 288 comparator neurons. The targeted clock frequency is 12 MHz and the sample rate is 80 kHz.**

of comparator neurons and HPUs used. The experimental dynamic current consumption, on the other hand, tends to decrease when the number of HPUs and comparator neurons increases. Those results being very different from the estimation obtained through the Vivado power consumption evaluation tool, it reinforces the value of a complete framework able to evaluate a large amount of architectures in real conditions as non-linearities and resource usage may make optimal architectures difficult to anticipate.

V. CONCLUSION

We proposed a methodology to produce and evaluate digital architectures to convert data produced by conventional sensors into spikes to feed SNNs. To illustrate our work, we implemented two encoding algorithms available in the literature (with a frequency-based approach and an amplitude-based approach respectively). We integrated them into our High-Level Synthesis C/C++ framework which allows to identify leverages to optimize the RTL design in terms of resource utilization and energy consumption while ensuring real-time processing. The methodology automatically generates many digital architectures with various levels of parallelism and various quantification formats. All architectures were implemented on FPGAs and had their current consumption monitored as well as their timing performances (latency and throughput) evaluated. We showed that for each algorithm, various hardware architectures were possible depending on the application requirements. As an example, in the situation and hardware that we studied, the encoding architecture with the highest throughput is a frequency-based approach that can process 285,000 samples per second using 8 HPUs and requiring more than 70 % of the DSPs available on the board. This throughput is however reached at the cost of a slight increase of the dynamic current consumption (9.7 %).

The methodology can be used to benchmark various architectures in real-world conditions on FPGAs before converting the RTL into an ASIC to reduce even more the overall energy consumption. This preliminary work also brings hope for complete hardware SNNs systems designed with this methodology. It would allow designing and implementing more efficient event-based architectures and perform fairer comparisons with CNNs.

REFERENCES

[1] V. A. Trinh and M. Mandel, "Directly comparing the listening strategies of humans and machines," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 29, pp. 312–323, 2021.

[2] S. Dodge and L. Karam, "Human and DNN classification performance on images with quality distortions: A comparative study," *ACM Trans. Appl. Perception*, vol. 16, no. 2, pp. 1–17, Apr. 2019.

[3] A. Brock, S. De, S. L. Smith, and K. Simonyan, "High-performance large-scale image recognition without normalization," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 1059–1071.

[4] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64270–64277, 2018.

[5] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[6] P. Wijesinghe, A. Ankit, A. Sengupta, and K. Roy, "An all-memristor deep spiking neural computing system: A step toward realizing the low-power stochastic brain," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 5, pp. 345–358, Oct. 2018.

[7] M. Heidarpur, A. Ahmadi, M. Ahmadi, and M. R. Azghadi, "CORDIC-SNN: On-FPGA STDP learning with Izhikevich neurons," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 7, pp. 2651–2661, Jul. 2019.

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

[9] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120 dB 15 μs latency asynchronous temporal contrast vision sensor," *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb. 2008.
[10] C. Posch, D. Matolin, R. Wohlgenannt, and T. Maier, "A temporal contrast IR vision sensor," *Proc. SPIE*, vol. 7100, pp. 704–714, Sep. 2008.
[11] T. Werner, E. Vianello, O. Bichler, D. Garbin, D. Cattaert, B. Yvert, B. De Salvo, and L. Perniola, "Spiking neural networks based on OxRAM synapses for real-time unsupervised spike sorting," *Frontiers Neurosci.*, vol. 10, p. 474, Nov. 2016.
[12] M. Bernert and B. Yvert, "An attention-based spiking neural network for unsupervised spike-sorting," *Int. J. Neural Syst.*, vol. 29, no. 8, 2019, Art. no. 1850059.
[13] J. R. D. O. Neto, J. P. C. Cajueiro, and J. Ranhel, "Neural encoding and spike generation for spiking neural networks implemented in FPGA," in *Proc. Int. Conf. Electron., Commun. Comput. (CONIELECOMP)*, Feb. 2015, pp. 55–61.
[14] Y. Yi, Y. Liao, B. Wang, X. Fu, F. Shen, H. Hou, and L. Liu, "FPGA based spike-time dependent encoder and reservoir design in neuromorphic computing processors," *Microprocess. Microsyst.*, vol. 46, pp. 175–183, Oct. 2016.
[15] B. Fang, Y. Zhang, R. Yan, and H. Tang, "Spike trains encoding optimization for spiking neural networks implementation in FPGA," in *Proc. 12th Int. Conf. Adv. Comput. Intell. (ICACI)*, Aug. 2020, pp. 412–418.
[16] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" in *Proc. Mach. Learn. Syst.*, 2020, pp. 129–146.
[17] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2017, pp. 5687–5695.
[18] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.
[19] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5784–5789, Nov. 2018.
[20] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–23, Sep. 2018.
[21] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–10.
[22] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, and T. Masquelier, "SpykeTorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron," *Frontiers Neurosci.*, vol. 13, p. 625, Jul. 2019.
[23] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, Aug. 2019, Art. no. e47314.
[24] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma, "BindsNET: A machine learning-oriented spiking neural networks library in Python," *Frontiers Neuroinform.*, vol. 12, p. 89, Dec. 2018.
[25] M.-O. Gewaltig and M. Diesmann, "NEST (neural simulation tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.
[26] N. T. Carnevale and M. L. Hines, *The NEURON Book*. Cambridge, U.K.: Cambridge Univ. Press, 2006.
[27] J. M. Bower and D. Beeman, *The Book of GENESIS: Exploring Realistic Neural Models With the General Neural Simulation System*. Berlin, Germany: Springer, 2012.
[28] 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, Aug. 1952.
[29] L. F. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain Res. Bull.*, vol. 50, nos. 5–6, pp. 303–304, Nov. 1999.
[30] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.
[31] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[32] G.-Q. Bi and M.-M. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type," *J. Neurosci.*, vol. 18, no. 24, pp. 10464–10472, Dec. 1998.
[33] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.
[34] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A low power, fully event-based gesture recognition system," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7388–7397.
[35] F. G. Lorca, L. Kessal, and D. Demigny, "Efficient ASIC and FPGA implementations of IIR filters for real time edge detection," in *Proc. Int. Conf. Image Process.*, 1997, pp. 406–409.
[36] P. Priya and S. Ashok, "IIR digital filter design using Xilinx system generator for FPGA implementation," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Apr. 2018, pp. 0054–0057.
[37] R. Ye, K. Iordanou, G. Riley, and M. Luján, "Exploring sparse visual odometry acceleration on FPGA with high-level synthesis," *IEEE Access*, vol. 11, pp. 70741–70763, 2023.
[38] R. S. Molina, V. Gil-Costa, M. L. Crespo, and G. Ramponi, "High-level synthesis hardware design for FPGA-based accelerators: Models, methodologies, and frameworks," *IEEE Access*, vol. 10, pp. 90429–90455, 2022.
[39] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Design Test Comput.*, vol. 26, no. 4, pp. 18–25, Jul. 2009.
[40] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An introduction to high-level synthesis," *IEEE Design Test Comput.*, vol. 26, no. 4, pp. 8–17, Jul. 2009.
[41] D. Meintanis and I. Papaefstathiou, "Power consumption estimations vs measurements for FPGA-based security cores," in *Proc. Int. Conf. Reconfigurable Comput.*, Dec. 2008, pp. 433–437.
[42] D. Meidanis, K. Georgopoulos, and I. Papaefstathiou, "FPGA power consumption measurements and estimations under different implementation parameters," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2011, pp. 1–6.
[43] M. Bernert and B. Yvert, "An attention-based spiking neural network for unsupervised spike-sorting," *Int. J. Neural Syst.*, vol. 29, no. 8, Oct. 2019, Art. no. 1850059.

CLÉMENCE GILLET received the M.S. degree in electrical engineering from the ENSEIRB-MATMECA Engineering School, France, in 2020. She is currently pursuing the Ph.D. degree with the IMS Laboratory, University of Bordeaux, France. Her research interest includes efficient digital hardware implementation of spiking neural networks and related high-level synthesis tools.

ADRIEN F. VINCENT received the Ph.D. degree from the University of Paris-Saclay, France, in 2017, with a focus on the use of innovative memory devices as artificial synapses in neuro-inspired electronics. After graduating, he was a Postdoctoral Researcher with the Group of Pr. Dmitri Strukov, University of California at Santa Barbara, Santa Barbara, CA, USA, involved in the statistical modeling of TiO2 memristors and their potential for analog dot-product engines. Since 2018, he has been an Associate Professor with the Bordeaux INP, University of Bordeaux, Talence, France, and has joined the Bioelectronics Group, IMS Laboratory, Talence, where he works on hardware implementation of artificial intelligence systems and neuromorphic architectures.

BERTRAND LE GAL received the M.S. and Ph.D. degrees from the University of South Brittany, Lorient, France, in 2002 and 2005, respectively. In 2005, he joined the IRISA/INRIA Laboratory, University of Rennes 1, and the ENSSAT Graduate Engineering School, Lannion, France, as a Teaching Assistant. Since 2006, he has been an Associate Professor with the IMS Laboratory, ENSEIRB-MATMECA Engineering School, France. In 2023, he received the accreditation to supervise doctoral research with the University of Bordeaux, France. He is the author or coauthor of 31 peer-reviewed publications in international journals and more than 100 peer-reviewed articles in international conferences. His research focused on algorithm-architecture-matching of ECC decoders on both hardware (ASIC/FPGA) and software targets (CPU/DSP/GPU). His research interests include complete digital communication system implementations and RISC-V processor design underperformance and security constraints.

SYLVAIN SAÏGHI (Member, IEEE) received the Ph.D. degree from the University of Bordeaux, France, in 2004, with a focus on the design of analog operators dedicated to silicon neurons. In 2011, he was a Visiting Associate Professor with Johns Hopkins University, Baltimore, MD, USA, for six months, where he received a Fulbright Scholar Grant. He has performed pioneering work in developing biologically realistic and tunable silicon neurons. He is currently a Full Professor with the University of Bordeaux. He has authored and coauthored more than 60 peer-reviewed publications. His current research interest includes hardware implementation of neuromorphic systems for edge computing, with projects ranging from co-integrating emerging memristive nanodevices with CMOS circuits to strategies relying on more conventional digital systems.