IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 28 February 2023, accepted 19 April 2023, date of publication 24 April 2023, date of current version 2 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3269598

RESEARCH ARTICLE

On-FPGA Spiking Neural Networks for End-to-End Neural Decoding

GIANLUCA LEONE, LUIGI RAFFO, AND PAOLO MELONI
Dipartimento Ingegneria Elettrica ed Elettronica, University of Cagliari, 09123 Cagliari, Italy
Corresponding author: Paolo Meloni (paolo.meloni@unica.it)

ABSTRACT In the last decades, deep learning neural decoding algorithms have gained momentum in the field of neural interfaces and neural processing systems. However, to be deployed on low-budget portable devices while maintaining real-time operability, these models must withstand strict computational and power limitations. This work presents a spike decoding system implemented on a low-end Zynq-7010 FPGA, which includes a multiplier-less spike detection pipeline and a spiking-neural-network-based decoder mapped in the programmable logic. We tested the system on two publicly available datasets and achieved comparable results with state-of-the-art neural decoders that use more complex deep learning models. The system required 7.36 times fewer parameters than the smallest architecture tested on the same dataset. Moreover, by exploiting the spike sparsity property of the neural signal, the total amount of computations is reduced by about 90% during a test carried out on real recorded data. The low computational complexity of the chosen spike detection setup, combined with the power efficiency of spiking neural networks, makes this prototype a well-suited choice for low-power real-time neural decoding at the edge.

INDEX TERMS Neural decoding, spike detection, spiking neural network, FPGA, low-power.

I. INTRODUCTION

Neural interfaces proved to be key components for enhancing the quality of life of disabled patients. Their range of applications stretches from hand movement decoding in patients affected by tetraplegia [1], seizure detection in pediatric subjects with intractable seizures [2], hand prosthesis control with sensory flow restoration in transradial amputees [3], speech decoding in patients with motor speech disorders [4], etc. Neural interfaces acquire neural signals, interpret the patient's intention, and use this information to accommodate the patient's request.

Among several recording solutions, intracortical sensors have proven to be valuable instruments in decoding multiple motor functions [5]. The neural data sampled with intracortical arrays of sensors is usually seen as the contribution of two signals: the local field potentials (LFPs) and the action potentials, i.e. the spikes. The LFP is usually found in the frequency range [0.5, 300] Hz and can be obtained by filtering the recorded samples. Spikes, on the other hand, occur above 300 Hz and necessitate being identified along the neural track after filtering, a neural signal processing task referred as spike detection [6] and that could be extended by recognizing the firing neurons, taking the name of spike sorting [7]. The output of spike detection and sorting are respectively called multi-unit activity (MUA) and single-unit activity (SUA). LFP, MUA, and SUA signals have proven effective in several decoding tasks when supported by reliable decoding algorithms, such as Long-Short Term Memory (LSTM) used for LFP decoding [8], Recurrent Neural Network (RNN) used for MUA decoding [9], Quasi-Recurrent Neural Network (QRNN) used to decode MUA, SUA, and Entire Spiking Activity (ESA) [10], which involves processing the spike band without identifying individual spikes.

Regardless of the specific target application, neural interfaces are subject to strict power and energy limitations. For instance, implantable chips must respect tight restrictions to avoid endangering the patient, limiting any temperature increase of the tissue to below 0.5°C [11]. Furthermore, it is advisable to ensure long battery life for portable devices, which constrains neural interfaces' energy requirements.

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

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

In this context, Spiking Neural Networks (SNNs) are promising tools for low-power neural processing at the edge. Unlike other neural network models, SNNs use spiking neurons as their fundamental units, which are more similar to biological neurons, from which artificial neural networks derive. As biological neurons, spiking neurons have a memory and can communicate exclusively receiving and transmitting action potentials, i.e. ones and zeros in the digital domain. The appeal of SNNs lies in their computational efficiency: the synaptic connections among spiking neurons are not always active since the neurons can only fire o not fire a spike. The phenomenon is called spike sparsity and causes the number of computations to drop down, which in turns lead to reduced power consumption and latency. Despite the non-differentiability of the spike function held back from extensively using these models in the past, the effort of the scientific community during the last decades paved the way for new algorithms suitable for SNN supervised learning [12], enabling wider use of these networks.

Furthermore, intracortical spikes and SNN models are already intrinsically compatible, no further processing is required to transform a continuous signal into a spiking one, as happens when other forms of neural activity are used in place of action potentials.

This work presents a spike-based neural decoding system implemented on FPGA that exploits the computational efficiency of SNNs to decode the displacement of a handle moved during a delayed reach-to-grasp task [13], recorded using a 96-channel multi-electrode array (MEA) [14]. Moreover, the neural signal is pre-processed in real-time to extract the MUA by using a multiplier-less spike detector, mapped in the Programmable Logic (PL).

The key contributions of this study are resumed as follows:

- It is presented a new computationally efficient real-time spike detection method along with its hardware implementation;
- A hardware SNN is used for solving a continuous decoding task in real-time for the first time, to the best of our knowledge, in the neural signal decoding domain;
- The system's accuracy has been assessed on a benchmark dataset, achieving accuracy comparable to state-of-the-art methods. Remarkably, it used 7.36 times fewer parameters than the smallest neural decoder tested on the same dataset;
- The study demonstrates the computational efficiency of spiking-neural-network-based decoders in neural applications, with an average of 90% operations saved.

The following Sections are organized as follows: a description of the studies related to this work is presented in Section II, the methods are described in Section III, Section IV reviews the hardware implementation, Section V contains the results, Section VI compares the results with the state of the art, and Section VII is left to the conclusions.

II. RELATED WORKS

Neural activity decoders can be categorized into two families, depending on the nature of the task: gesture or motion classification and continuous motion or force decoding. The former produces an output from a predefined set of classes and determines which class the performed action belongs to. The latter continuously tracks a target variable, such as position, speed, or force, and infers its value through regression. In this section, we analyze several deep-learning neural decoders that belong to both categories and utilize several sensor arrays. Table 1 summarizes the main features of the works analyzed in this section.

In [1] the signals recorded by two electrocorticography (ECoG)-based implants are decoded during a 3D virtual hand translation task using a CNN followed by an LSTM block, using one-second windows with 90% overlap. Similarly, in [15] the stereo-electroencephalography (sEEG) signal is used for continuous force decoding during a hand-grasping task. The neural signal is used to feed a CNN followed by an LSTM block, in windows of two seconds with a stride of 50 milliseconds.

Both works reached state-of-the-art accuracy, showing that deep-learning models are valid instruments for neural decoding. However, considering the ultimate goal consists in delivering low-power neural interfaces, using CNN for analyzing long time-series data characterized by long periods of idleness interleaved with bursts of activity is not power-efficient. SNNs better exploit the event-based nature of the task, allocating resources only when needed by the application.

In [16] the Python implementation of an SNN decoder trained to continuously track the elbow angle of four subjects moving their arm is compared to the results obtained by an LSTM-based decoder. The neural recording was acquired with an 8-channel surface EMG armband. The surface EMG signal is continuous in time, thus, it requires to be converted in spike trains. Firstly, a time-domain feature called *waveform length* is computed, defined as the aggregated length of the EMG waveform over the segment [23], computed over 100 samples in [16]. Then, this feature is given as input to a spiking layer of 64 units that converts it into spikes. The spike trains are then processed by two hidden layers of 128 and 64 units. The output of the last hidden layer, in form of spike trains, is converted into a continuous value by introducing a final layer, composed of one spiking neuron only, where the membrane potential of the neuron was used as continuous output, rather than its spikes.

Another example of an SNN decoder is reported in [17], where a 3-channel surface EMG (sEMG) signal was used to carry out a hand gesture recognition task counting five gestures. The continuous sEMG signal is processed to obtain frequency-domain features, then converted into spike trains to be decoded by the SNN. The signal is filtered three times with three distinct filters to obtain three separated signals, then smoothed by computing their root mean square value over a sliding window of 225 samples, corresponding to 150 ms.

TABLE 1. State of the art neural activity decoders.

Work	Year	Signal	Task	Type	Decoder	Channels	Parameters	Platform
[1]	2022	ECoG	Imagined hand translation	Regression	CNN+LSTM	64	238,772	PC
[15]	2022	sEEG	Hand grasping force	Regression	CNN+LSTM	14	-	PC
[16]	2022	sEMG	Elbow joint angle	Regression	SNN	8	16,448†	PC
[17]	2022	sEMG	Hand gestures	Classification	SNN	3	-	PC
[18]	2010	Intracortical	Arm movements	Classification	SNN	100	1,212†	PC
[10]	2021	Intracortical	Hand velocity	Regression	QRNN	96	-	PC
[19]	2021	Intracortical	Hand velocity	Regression	LSTM	96	-	PC
[20]	2019	EEG	Motor imagery	Classification	CNN	10	-	FPGA
[21]	2016	ECoG	Finger movements	Classification	PCA+MLP	62	-	FPGA
[22]	2017	EEG	Hand gestures	Classification	Bayesian Classifier	4	- deduced†	uC

The RMS arrays are normalized and converted into spikes: the RMS values are accumulated sample by sample until a certain threshold is reached and a spike is generated. Finally, the spike trains are used to feed the SNN decoder. The SNN decoder is a three-layer network with 9 inputs, 20 hidden units, and 5 output units. The neurons are of type Leaky Integrate-and- Fire (LIF), and in particular, the units of the last layer have their threshold set to infinite. The classification result is determined by applying a *winner take all* strategy where the neuron with the higher potential is chosen.

In [16] and [17] is demonstrated SNNs are capable of decoding different kinds of neural signals and can be exploited for both regression and classification tasks. However, the conversion process from continuous signals to spike trains and vice versa requires additional computations. The use of spiking signals could simplify the processing pipeline and lead to more efficient neural interfaces.

We found only one work in literature that directly uses SNNs for decoding spiking signals [18]. Their network analyzes SUA recorded during a 3D reach-to-grasp task [24]. The single neuron activity was processed offline and only the more correlated units were kept. The system was trained to decode the object direction and orientation, using respectively the one hundred and ninety-seven more correlated neurons. The decoding tasks consisted of classifying whether the movements were toward the left or the right and which was the orientation of the target. The former task was carried out by using a hidden layer of 12 units and by training one output neuron to fire a spike at 51 ms for left prediction, or later, at 61 ms for right prediction. The latter was implemented by a 12-unit hidden layer and by applying a winner-take-all strategy to three output neurons where each neuron represents one of the three possible orientations. Although the approach used in [18] is similar to the method proposed in this work, it is difficult to make a comparison with this study because the dataset was not acquired using a multi-electrode array, instead, the same experiment was repeated multiple times to sample the neural activity of different brain areas, furthermore, their model was used for classification, not for continuous regression.

To assess the accuracy performance of our SNN-based approach we chose to use the dataset reported in [13], which comprises two intracortical recordings sampled using a 96-electrode Utah array during a delayed reach-to-grasp task. The same dataset is used in two works used as benchmarks to compare the accuracy of our results. In [10] several decoders, such as QRNN, Gated Recurrent Unit (GRU), and LSTM, are tested on different neural signal features, such as SUA, MUA, LFP, and ESA. In [19] RNN, GRU, and LSTM models are tested on one of the two neural recordings available in [13], permitting only a partial comparison with our result.

Even though portability and low latency are key properties in the neural interface domain, we found prospectively few implementations addressing portable computing platforms such as FPGAs or microcontrollers (uCs) rather than PCs, that we reported in Table 1. In [20] a CNN deployed on FPGA is used for decoding in real-time the electroencephalographic (EEG) signal acquired from 10 channels during a two-class motion imagery classification task. The input of the CNN is structured as a series of frames, with each frame representing a one-second segment of the EEG signal across three frequency bands. Each frame overlaps the preceding one by 50%. During inference, the hardware architecture performs a sequence of matrix multiplications that maps the CNN's layers execution, fully utilizing the resources of the hosting Xilinx z7020 device. In [21] an FPGA is used to accelerate a two steps process aiming at decoding a 62-channel ECoG signal during an online finger movement classification task. The first step of the data processing pipeline consists of a dimensionality reduction performed through Principal Component Analysis (PCA) from 62 to 3 dimensions, then, a multilayer perceptron (MLP) is used to classify the finger movements. A uC is used in [22] for implementing a low-cost neural interface solution for EEG-based neural decoding during a hand open/close/idle state classification task using a Bayesian classifier.

Although several hardware solutions have been proposed for intracortical neural signal processing, most of them stop at the spike detection phase [25] or at the spike sorting phase [26]. Similarly, among numerous existing SNN accelerators, some are oriented to understand the brain functionalities, exploiting either FPGA-based prototypes [27], or higher-end emulators such as SpiNNaker [28]. In [27] is presented a bio-realistic emulator of Izhikevich neurons

addressing a single fully-connected layer structure, implemented on a Xilinx z7020 chip. The implementation is optimized for biological real-time emulation of SNNs, allowing arbitrary connections among neurons and a per-neuron parameter set, which consents to reproduce several output spiking patterns. In [28] is presented a high-end biologically relevant emulator of spiking neurons where nodes of 48 integrated circuits, embedding 18 ARM968 processor cores each, can be interconnected in a cluster, counting over a million processors to accelerate the neural network emulation, requiring a peak electrical power of 75 kW.

In this work, we focus on designing an end-to-end accelerator for neural processing and decoding at the edge exploiting low-power properties of SNNs. The existing alternatives include FPGA-based solutions [29], where is presented a hybrid spiking and convolutional FPGA-based neural network design deployed on a low-end Xilinx z7020, and custom ASIC neuromorphic multi-core accelerators, such as Loihi from Intel [30], and TrueNorth from IBM [31].

However, to the best of our knowledge, this is the first custom neural interface prototyped on FPGA comprehensive of both neural signal analysis in the form of spike detection and neural decoding utilizing an SNN, ideal for low-cost neural signal processing and decoding at the edge.

III. METHODS

In this work, the intracortical recordings of a 96-channel MEA are used for decoding the velocity of a handle moved during a delayed reach-to-grasp task in real time. The system is composed of two main modules: 1) a spike detector, that processes the raw samples provided by the array of sensors, detecting the spikes, and 2) an SNN that directly processes the detected spikes, inferring the handle velocity. The modules are implemented on a low-end Zynq 7010 FPGA (XC7Z010-1CLG400C) which allows fast prototyping, moreover, the FPGA programmability feature permits to modify the deployment of the parametric design on the FPGA to match the requirements of several experimental setups, that could involve a different dataset, a change in the spike detection hyper-parameters, the choice of a different neural network topology.

This Section is organized as follows: a description of the dataset used to validate the system is provided in Section III-A, in Section III-B is presented the spike detection pipeline, Sections III-C and III-D describe the neuron model and the SNN architecture, Section III-E explain the spike sparsity phenomenon and how to take advantage of it for reducing at the same time the power consumption and the system latency.

A. DATASET

The system has been tested using electrophysiological recordings from two macaque monkeys [13]. During the experiments, the monkeys were trained to perform a delayed reach-to-grasp task that involved pulling a cuboid handle with two types of grips and two levels of force. The datasets, one for each monkey, consisted of neural recordings obtained from a chronically implanted 10 × 10 multielectrode Utah array in the motor cortex. The neural recordings were sampled at 30 kHz, but were downsampled by a factor of 3 for this study. In addition to neural data, the datasets included behavioral data, such as the handle displacement and the force applied to the cuboid handle, which was recorded using force-sensitive sensors. The behavioral data were sampled at 1 kHz.

Figure 1 shows the stages of the handle position processing before being used as the decoding variable during the training phase. Initially, the handle position (Fig. 1 (a)) underwent a smoothing process with a moving average filter of order 64 (Fig. 1 (b)). Next, the first derivative of the smoothed handle position was computed to obtain the handle velocity (Fig. 1 (c)). The first derivative was evaluated by subtracting adjacent samples. To further refine the handle velocity, a mean average filter of order 16 was applied (Fig. 1 (d)). The resulting smoothed handle velocity was then used as the target variable for decoding during the training phase. We selected the handle velocity as the decoding variable to enable a comparison of our neural decoding system's accuracy with that of similar studies [10] and [19], which also used handle velocity as the decoding variable.

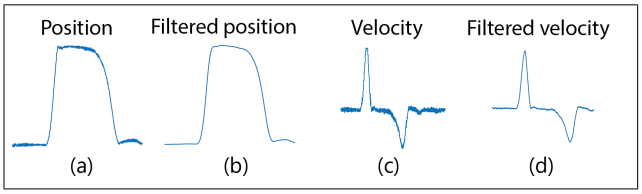

FIGURE 1. Target variable processing steps before training: (a) raw handle position measurement; (b) smoothed handle position (64th-order moving average filter); (c) handle velocity (subtraction of adjacent samples); (d) smoothed handle velocity (16th-order moving average filter).

B. SPIKE DETECTION

Spike detection is a key component of neural interfaces based on spiking signals. Spike detection accuracy directly affects the reliability of the subsequent steps of the signal processing chain, thus the reliability of the whole method. In this study, the spike detector operates on signals sampled at a frequency of 10 kHz, which is ten times higher than the frequency of the signals processed by the SNN, which is 1 kHz. Moreover, as the number of channels increases to either cover a larger area of the neural tissue, sample at a finer resolution, or both, the computational and memory demands of spike detection also increase linearly. This presents a challenge when scaling up the system to accommodate a greater number of channels.

For all these reasons we chose to keep as limited as possible the computational and memory usage per channel, aiming to contribute with a design well suited for scaling both in frequency and in the number of channels, proposing a configuration that only requires five additions per channel and zero multiplications. Fig. 2 shows the spike detection processing chain, which is also resumed in the following points:

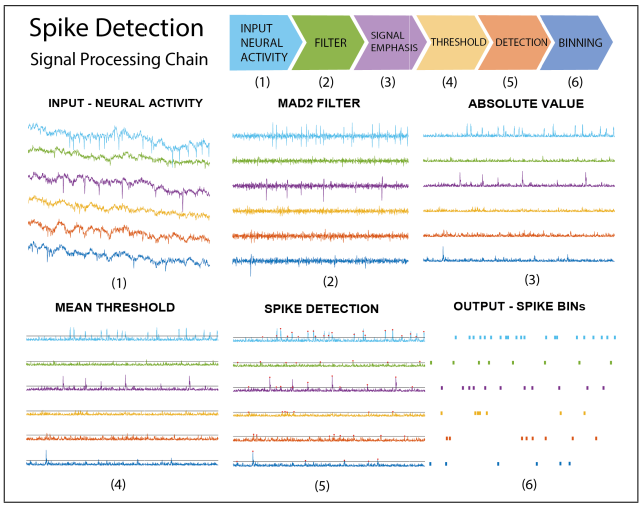

FIGURE 2. The input neural activity (1) is processed by five pipelined modules: the MAD filter removes the low-frequency components (2); the signal is rectified by the spike emphasis module (3); the mean of the rectified signal is used as a basis to compute the threshold (4) to detect the spikes (5); one-millisecond spike bins are forwarded in output (6).

- Filter: second-order moving average difference (MAD) filter - to remove low-frequency components (LFPs);
- Spike emphasis: absolute value - to emphasize the spike shape and improve the detection reliability;
- Threshold: mean value (times four) - value above which is detected a spike event;
- Threshold update time: 0.82 ms - how often the spike threshold is updated;
- Refractory period: 1 ms - time of insensitivity after the spike detection event (per channel);
- Spike bins: 1 ms - spikes count over a certain interval of time (per channel).

The MAD filter proposed in [32] is a good candidate to implement a high-pass filter behavior, since its application only consists in subtracting from each input sample the moving mean of the input signal, computed thanks to a moving average filter, as stated by Eq. 1.

$$MAD(n) = x(n) - \frac{1}{N}\sum_{i=1}^{N} x(n-i) \tag{1}$$

where x is the input signal and N is the filter order. Note that, by choosing N equal to a power of two, as in this study where it has been chosen $N = 2$, the filter is implemented by only $N + 1$ sums and one right-shift operation, no multiplications and/or divisions take place.

As regards the spike emphasis method, we did not observe any consistent accuracy improvement in using more sophisticated solutions than the absolute value of the signal. Moreover, computing the absolute value of a number does not require any additional memory element or multiplication contrarily to other widely used methods such as Non-Linear Energy Operator [33] or Amplitude Slope Operator [32].

The time window size is set at 0.82 ms, corresponding to 8,192 samples, a value we observed empirically is the best trade-off among true positives, false negatives, and false positives, maximizing the detection accuracy.

Finally, to match the frequency of the target variable to decode, which is 1 kHz, we used spike bins computed over 1-ms windows as input of the SNN. In addition, by setting the refractory period of each channel to 1 ms, we obtained that the bins could exclusively be equal to one or zero, matching the input pattern required by the SNN.

C. LOIHI CUBA NEURON MODEL

The basic processing element of the SNN is the Loihi CUrrent BAsed Leaky Integrate and Fire (CUBA) Neuron, presented in [30]. The Loihi CUBA neuron extends the standard CUBA neuron [34] by introducing an additional internal state variable. The Python implementation of the neuron model is derived by the PyTorch package SLAYER (Spike LAYer Error Reassignment) [35], then extended and embedded as SLAYER 2.0 in the LAVA software framework [36] used in this work. The Loihi CUBA Neuron requires two integration stages, similarly to the Izhikevich neuron [37], but with a simpler structure. The state variables update equations are identical: the spiking activity, convolved with the synaptic weights, feeds the first integrator, whose output feeds the second one. The second state variable is used as a metric for the output spike generation. The model equations follow:

$$\begin{aligned} S(t) &= \sum ws(t-1) \\ i(t) &= \alpha i(t-1) + S(t) \\ v(t) &= \beta v(t-1) + i(t) \\ s(t) &= v(t) > \theta \\ v(t) &= v(t)(1 - s(t)) \end{aligned} \tag{2}$$

where w is the set of synaptic weights of the neuron, s is the input spike vector, and $S(t)$ is the convolution between spikes and weights. $S(t)$ is the Loihi CUBA neuron input. $i(t)$ is the current state variable, which is computed by multiplying its previous value by a decay factor α and adding the convolved spiking activity $S(t)$ to it. $v(t)$ is the voltage state variable, its equation maintains the same structure seen for $i(t)$. Its previous value is multiplied by a decay factor β and added to the current $i(t)$. The neuron fires a spike when the value of $v(t)$ exceeds the threshold θ. When it happens, the value of $v(t)$ is reset to zero.

D. SPIKING NEURAL NETWORK

SNNs in LAVA [36] can use convolutional, dense, recurrent layers, etc. Moreover, many types of neurons are supported as well, aside from the Loihi CUBA neuron discussed previously, Resonate & Fire Izhikevich model, Adaptive Leaky Integrate and Fire model, and others, are available.

The SNN used in this work is constituted of two dense layers of 256 and 128 Loihi CUBA neurons. The script used

for training is based on the *Spike to Spike Regression: Oxford*[1] example given in the platform, the neuron parameters are left untouched and are shown in Table 2.

TABLE 2. Loihi CUBA neuron parameters.

Threshold	Current Decay	Voltage Decay
0.1	1.0	0.1

We tried to find the simplest model enabling an accuracy competitive with other works in literature tested on the same dataset. We trained our model using the Slayer library in the Lava framework [36], using the spike times as a metric to compute the loss. The training counts 200 epochs, 60% of the trials are used for training, 10% for validation, and 30% for testing. We selected the model with the best loss on the validation set. As for the neuron parameters, we left the training parameters as found in the Lava example template. The parameters used for training are shown in Table 3.

TABLE 3. SNN training parameters, from left to right: units in the first layer L1, units in the second layer L2, learning rate, weight decay factor, number of epochs.

Units L1	Units L2	Learning rate	Weight decay	Epochs
256	128	0.001	1e-5	200

Table 4 reports the memory required by each layer and in total, considering 15-bit representation for the weights.

TABLE 4. SNN memory requirements, from left to right: first layer L1 memory, second layer L2 memory, total memory.

L1	L2	Total
45 kB	60 kB	105 kB

SNNs are trained to generate target spike trains in response to input spike trains. Even though in this work the input is effectively a spike train, the desired output is a number, i.e. the target velocity. A way to easily transform the output spikes into a number during the online neural activity decoding, and on the contrary, a way to generate a target spike pattern starting from the desired output number during the network training is thus needed.

Our proposed solution is to attribute to every neuron of the output layer a weight that could either be $+1$ or -1 and compute the target velocity as the sum of the spikes fired by the output layer. In this case, the maximum and minimum values of the target velocity are similar, thus, half of the output neurons contribute positively, whereas the other half negatively. For this purpose, the handle velocity was cast to integer and constrained in the range $[-64, +63]$ as shown in Fig. 3 (a), i.e. the output is represented with a resolution of 7 bits. The firing pattern is established so that, depending on the value of the target variable at each time step, the same number of units fire a spike. Fig. 3 (b) shows the target raster plot used to train the SNN, downsampled by a factor of ten for display purposes.

[1] https://lava-nc.org/lava-lib-dl/slayer/notebooks/oxford/train.html

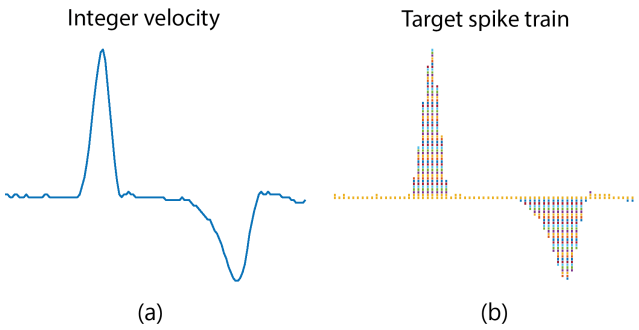

FIGURE 3. Target variable conversion into spike trains prior training: (a) the handle velocity is cast to integer, in the range [−64, +63]; (b) the integer values have a one-to-one correspondence with the number of spikes that compose the target spike train used for training.

E. SPIKE SPARSITY

Spiking signals are characterized by being either active or inactive. This characteristic makes SNNs event-driven systems. Figure 4 shows the timing raster plot of a portion of the dataset, where on the x-axis is represented the time, and on the y-axis the detected spikes per channel. The spiking activity increases at three points in Figure 4: right before 2.5 seconds, at 7.5 seconds, and right after 12.5 seconds; in correspondence with these three points the spikes are denser in the raster plot. The dark blue line on the bottom of Figure 4 represents the sum of the spikes across all the channels. It is observable how the number of concurrent spikes is limited to about twenty, in addition, the superimposed light blue line is the smoothed sum of the spikes, plotted to better visualize the trend of the neural activity, which increases where the spikes are denser. The orange line is the handle velocity, it is possible noticing that the handle movement happens as soon as the augmented neural activity is detected in the motor cortex.

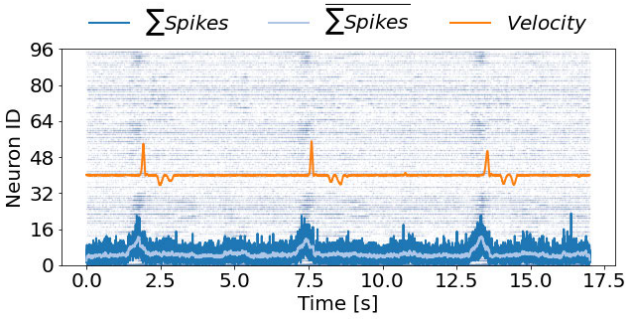

FIGURE 4. Measured spiking activity at the input of the SNN. The raster plot on the background shows the spikes detected on each channel by the spike detection module; on the bottom is shown the sample-by-sample sum of the spikes in dark blue and its mean value in light blue; in orange is shown the handle velocity, i.e. the target variable.

The sparsity property interests the input layer of the SNN, as well as the following layers. Because of the nature of these

networks is possible to avoid wastes of power, such as reading the entire weight memory when there is no spiking activity on the input of the layers or hardly any. The SNN exploited in this work is based on fully-connected layers. Therefore, all the neurons of the same layer share the same inputs, which are read multiple times from the spike memory. To avoid reading the spikes and computing the sums of the inactive inputs of the layer, we introduce a stack storing the pointers to the active set of inputs, stored in groups of four. At each synaptic current computation, only the entries pointed by the stack need to be read, instead of the entire set of spikes and synaptic weights. As will be discussed in Section V-B2, this simple architectural expedient enables avoiding hundreds of thousands of memory reads and sums per second, as well as greatly reducing the synaptic current computation time.

IV. SYSTEM ARCHITECTURE

This section describes the digital system hardware architecture, which supports spike detection and spike-based neural activity decoding. The tasks are deployed to two custom computational cores implemented on the PL. The system is designed to handle inputs from a 128-channel MEA, although the accuracy assessment datasets are limited to 96 channels. The system receives broadband recordings through a slave AXI stream interface and outputs the decoder inference through a master AXI lite interface. The neural decoder architecture is shown in Fig. 5.

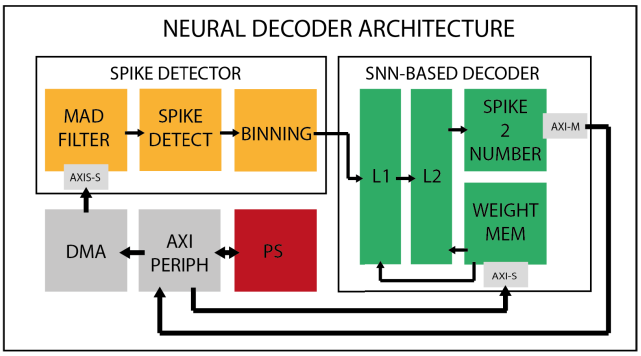

FIGURE 5. Neural decoder architecture: the core of the neural decoder is composed of two cascaded modules: a spike detector and an SNN-based decoder. The former extracts the binned spiking activity directly from the raw neural signal using a MAD filter, a spike detector, and a binning module. The SNN is composed of two dense layers, a weight memory, and a spike-to-number (S2N) converter, that translates the output spike train generated by the SNN's second layer into the target variable. The neural decoder is connected to the Processing System (PS) through three AXI interfaces: 1) an AXI-stream interface is used to stream the neural samples in the PL; 2) an AXI-lite interface is controlled from the PS to set up the neural decoder; 3) an AXI-lite interface is used to transmit the decoding results.

A. SPIKE DETECTION AND SPIKE BINNING

The spike detection and the spike binning tasks are appointed to five pipelined modules that process the input channels in a time-multiplexed fashion. The broadband neural signal is first filtered and emphasized. Then, the signal is compared with a mean-value-based threshold to verify the spike condition. Finally, the spike binning module counts the detected spikes per millisecond and forwards the bins to the SNN-based decoder.

1) MEAN REMOVING FILTER

The low-frequency components of the neural signals are removed by subtracting from every incoming sample its moving mean value. The mean value is obtained by right shifting by one position the sum of 2 previous samples. The spike detector processes 128 channels in a time-multiplexed fashion, therefore, it internally contains 256 registers (to store two samples per channel). At every incoming sample, the registers shift by one position, the incoming sample is stored, and the oldest sample is lost. The output of the registers in positions 128 and 256 are added and right-shifted by one position to compute the signal mean. The mean value is subtracted from the incoming sample to obtain a high-pass behavior. The architecture of the 2nd-order MAD filter is shown in Fig. 6 (1).

2) SPIKE EMPHASIS

The filtered signal is rectified using a 2-ways multiplexer that selects either the sample or its 2's complement depending on the value of its sign bit. The architecture of the spike emphasis module is shown in Fig. 6 (2).

3) SPIKE THRESHOLD

The spike detection threshold is equal to four times the mean value of the rectified signal computed during the past 0.82 ms (8,192 samples). The thresholds are stored in a BRAM-based memory with one entry of 10 bits per channel and are read when a new sample needs to be compared with the channel's threshold. Concurrently, at every new incoming sample, the future threshold is updated. The future thresholds are stored in an additional BRAM-based memory of 128 entries of 23 bits (10 bits is the sample size + 13 bits due to the accumulation of 8,192 samples). The future threshold is read, added to the new incoming sample, and stored back in the same memory location. The samples are accumulated during the 0.82 ms time window, then they are right-shifted by thirteen positions and used to overwrite the value stored in the threshold memory, which will not change for the next 0.82 ms. After updating the threshold memory, the corresponding new threshold memory location is reset. A thirteen bits counter is used to keep track of time. The architecture of the threshold module is shown in Fig. 6 (3).

4) THRESHOLD CROSSING DETECTOR

The threshold crossing detector compares the emphasized sample value with the respective channel threshold. In addition, it implements a refractory period rule that limits the spike rate to one spike per millisecond per channel. The refractory periods are stored in a BRAM memory with an entry per channel. When a new sample arrives, the refractory period memory is read, if the value is zero and the sample

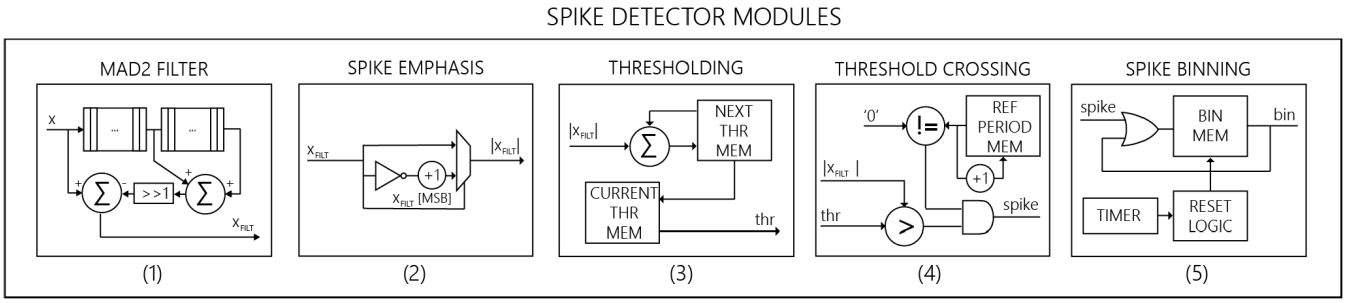

FIGURE 6. Spike detector internal modules: (1) second-order MAD filter, (2) absolute-value-based spike emphasis module, (3) mean-value-based spike threshold module, (4) threshold crossing detector, (5) spike binning module.

exceeds the threshold value, the spike is forwarded to the output. Moreover, once a spike is detected, the refractory period is incremented by one at every sampling cycle, until is reset by overflow, after 8,192 sampling cycles. The architecture of the threshold crossing detector is shown in Fig. 6 (4).

5) SPIKE BINNING

The spike binning module accumulates the spikes of every channel along a time window of 1 ms. However, because of the refractory period rules implemented by the threshold crossing detector, which limits the number of spikes per channel to one per millisecond, the output of the spike binning module can either be one or zero. Therefore, the bin memory is a single-bit memory with 128 entries, implemented using a 128-bit register.

When a spike is fired, the spike binning module updates the flip-flop associated with the firing channel. The module embeds a counter that keeps track of the time steps elapsed, incremented every time the whole set of channels is processed. At the tenth iteration, the accumulated spikes can be forwarded to the output. The architecture of the spike binning module is shown in Fig. 6 (5).

B. SNN-BASED DECODER

The spike decoder is composed of an SNN and a spike-to-number (S2N) converter. The SNN contains two layers of 128 and 256 Loihi CUBA neurons that process the neural activity in the form of spikes detected by the spike detection module. Finally, the output spike trains inferred by the SNN are converted into the handle velocity by the S2N converter.

1) LAYER MODULE

The layer module is composed of the spike and the weights memories, a stack where are stored the pointers to the active input synapses, a synaptic current computation module, two identical cascade connected integrator modules used to evaluate the Loihi CUBA neuron model described by Eq. 2, and a serial to parallel (S2P) converter. The layer module updates the neuron one by one: 1) the addresses of the active inputs are read from the stack, 2) the pointers are used to read the spike and the weight memories, 3) the synaptic current computation module accumulates the synaptic weights, 4) the integrator modules update the neurons' states that are stored in two

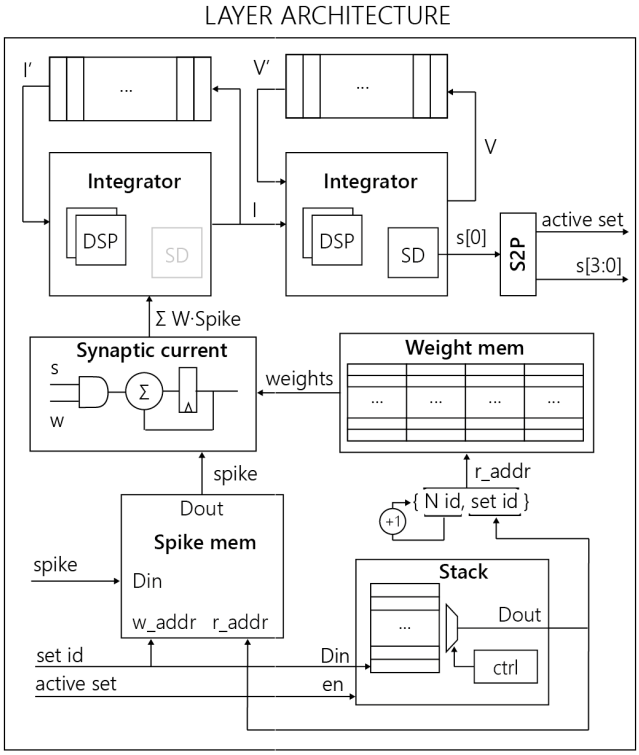

FIGURE 7. SNN's layer architecture: two memories store the spikes and the synaptic weights; a stack structure stores the pointers to the active set of inputs used to read the weights that contribute to the synaptic current value. The synaptic current module evaluates the synaptic current accumulating the weights over multiple clock cycles. Two identical cascaded integrator modules integrate the Loihi CUBA Neuron model. The neuron internal state variables are stored in two FIFO memories connected in feedback between the integrators' input and output. The serial output spikes are converted in a 4-bit parallel signal, in addition, the active set signal is generated as the *or-reduce* of the output spikes.

FIFOs connected between the integrator's input and output, 5) the second integrator generates a spike, 6) the S2P converter re-organizes the single-bit stream of spikes into a stream of four bits and appends the active set signal, which is the *or-reduced* value of the spikes. Fig. 7 shows the architecture of the layer module.

2) SYNAPTIC CURRENT

The synaptic current is the convolution between the input spike array of a neuron and its set of synaptic weights.

Its computation entails adding the synaptic weights of the active synaptic interconnections.

The synaptic weights are stored in a BRAM-based memory, where each entry allocates four weights. The spike memory has the same structure as the weight memory, therefore, with only one read operation four spikes and four weights are read. Weights and spikes are read sequentially, and a *logic-and* between each weight and its corresponding spike is computed, so that if the spike is at *logic-1* the weight value is preserved, otherwise is set to zero not to contribute to the evaluation of the synaptic current.

A single DSP used in Single-Instruction Multiple-Data (SIMD) mode can be used to compute two additions, the resulting partial sums are then accumulated employing two additional DSPs. The accumulation iterates until the synaptic current is computed adding four weights at every cycle.

3) LOIHI CUBA NEURON IMPLEMENTATION

The Loihi CUBA Neuron is constituted by two almost identical cascaded integrators that update the state variables by multiplying their previous values by a decaying factor and adding an external input, as described by equations 2. The only difference is that the second integrator verifies if its output exceeds the spike threshold, resets the state variable if it is the case, and outputs the spike. All considered a single hardware module was implemented, including an enable signal to activate the spike evaluation.

The two integrators are cascade-connected, the input of the current integrator is the synaptic current, whereas its output feeds the voltage integrator. A FIFO is used to feedback the old integrated values. The FIFO is BRAM-based and its depth is equal to the number of neurons of the SNN layer.

The Loihi CUBA model of the Lava library [36] represents the numbers in fixed-point; the state variables, as well as the synaptic weights, utilize 12 fractional bits, whereas the integer bits vary depending on the number of inputs of the neuron. This hardware implementation respects the data width used in the library [36], therefore no accuracy degradation is expected.

The multiplication between the state variable and the decay factor is mapped on a DSP. The state variable is the *current* when the integrator module is used to solve the first integration or the *voltage* during the second integration. The multiplication output, having 24 fractional bits, is right-shifted by 12 bits to be aligned with the external input before their addition. In the second integrator, where the spike evaluation is enabled, the output of the adder is compared with the spike threshold: depending on the outcome a multiplexer forwards either zero or the sum to the output, and a spike is produced.

4) SPIKE SPARSITY STACK

The spike sparsity stack allows taking advantage of the spike sparsity property of neural signal. The stack stores the addresses of the active spike sets by relying on the *active set* signal, which is used as a write enable. When the synaptic current computation starts, the stack streams out the addresses of the active spike sets. The address stream permits to retrieves spikes and weights of the active sets of inputs, whereas the inactive ones are skipped. The address stream is directly connected to the read address port of the spike memory, instead, to read the weight memory, the address stream is concatenated to the neuron identifier.

The stack is a flip-flop-based memory, its depth is equal to a quarter of the number of inputs of the layer where it is instantiated, whereas its word width is the base-two logarithm of its depth:

- Layer 1: 32 entries of 5 bits;
- Layer 2: 64 entries of 6 bits.

To stream out the stack content are used two counters. The former counts the stack entries during the writing phase; the latter is activated when the address stream starts, i.e. the second counter is initialized with the number of valid entries stored in the former counter and decremented until it reaches zero. The counter value drives the addresses to the output port through a multiplexer.

5) Spike2Number CONVERTER

The output of the last SNN's layer requires to be converted from a spike vector of 128 elements to a number. The number obtained is the regression output.

The conversion is performed by counting the spikes output of the last layer. The spike to number conversion problem is thus a population counting problem. Since the layer outputs the spikes four at a time, the spike-to-number converter is implemented as a variable-step counter, that at every new set of spikes increments its value depending on their sum.

Finally, since half of the neurons contribute positively, and the second half negatively, the counter increments its value when it receives the spikes of the first group and decrements it when it receives the spikes of the second group.

C. EXTERNAL INTERFACES

The architecture communicates by means of three AXI interfaces:

- AXI-4 stream slave: used to read the broadband signal;
- AXI-4 lite slave: used to initialize the weight memories;
- AXI-4 lite master: used to transmit the decoder prediction.

For test purposes, it is used a DMA to stream the sample from the PS to the PL.

V. RESULTS

The result section comprises two subsections; the former presents the decoding accuracy reached by the model on the benchmark dataset, and the latter shows the resource requirements of the hardware implementation.

A. ACCURACY

Two SNN models have been trained on the benchmark dataset. The first model was a single SNN dense layer of 128 units and 96 inputs. Although this model had low

computational and memory requirements, it performed poorly compared to the state of the art. Therefore, we tried training a more complex model consisting of two dense layers, the former layer having 256 units and the latter having 128 units. Table 5 shows the parameter of the two SNNs, including the number of layers, units, parameters, and memory requirements. Additionally, Table 5 shows the achieved decoding accuracy, which is measured with the Pearson correlation coefficient described by Eq. 3.

$$CC(A,B) = \frac{1}{N-1}\sum_{i=1}^{N}\frac{A_i-\mu_A}{\sigma_A}\frac{B_i-\mu_B}{\sigma_B} \tag{3}$$

where CC is the Pearson correlation coefficient of two random variables A and B, μ_A and σ_A are the mean and standard deviation of A, μ_B and σ_B are the mean and standard deviation of B. The two-layer model outperformed the single-layer model on both the used datasets, however, it requires instantiating three times more units, 4.7 times more parameters, and thus, 4.7 times more memory.

TABLE 5. SNN models decoding accuracy.

Layers	Units	Parameters	Memory	CC N [13]	CC L [13]
1	128	12,288	22.5 kB	0.79	0.66
2	384	57,344	105 kB	0.83	0.78

The velocity inferred by the 2-layer SNN model and the target handle velocity are plotted in Figure 8 using 37 seconds of recording taken from the test set. Figure 8 shows graphically how the spike decoder is capable of tracking the target variable, inferring its value from the neural recording.

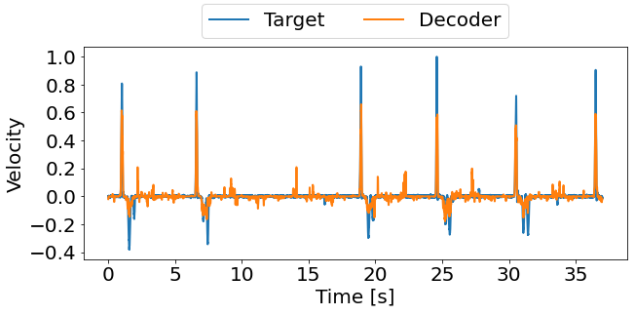

FIGURE 8. Spike decoder output behavior.

B. HARDWARE REPORT

This subsection analyzes the resource requirements of the presented neural interface and shows the benefit of designing a spike sparsity-aware architecture, expressing the savings as the number of saved additions.

1) UTILIZATION

The neural decoder implementation can handle up to 128 input channels. The choice went on 128 since it is the closest power of two bigger than 96, that is the channels of the Utah array used for recording the benchmark dataset. However, the architecture can be synthesized also for a lower or a higher number of channels. The design is hosted on a Zybo, a low-cost development board for Xilinx Zynq All-Programmable SoCs, clocked at 2 MHz, a frequency at which the architecture can comfortably process the data in real time, ensuring a response in 1.8 ms on average, and in about 3 ms in the worst-case scenario. The post-implementation resource requirements of the digital system are shown in Table 6. The first row indicates the overall resource usage of the design under test (DUT), the second and the third rows refer to the individual requirements of the spike detector and the SNN.

The overall required LUTs are about 2.5 k, of which 160 are instanced in the spike detector, and 756 in the SNN. The registers and the LUTRAMs are about 3.4 k and 209 respectively, of which 251 and 80 serve the spike detector, and about 1 k and 30 the decoder. Most of the spike detector LUTRAMs (72) are employed for storing the previous samples inside the filter, in fact, each SRLC32E LUTRAM primitive is a 32-bit shift register, thus: *32 bits* $\times$ *80 primitives* $=$ *2,560 bits*. Note that *10 bits sample* $\times$ *128 channels* $\times$ *2 taps* $=$ *2,560 bits* as well. By using more demanding filters, such as a 4th order IIR filter, the number of LUTRAM primitives necessary to implement the shift register would rise to about *10 bits sample* $\times$ *128 channels* $\times$ *9 taps/32 bits per SRLC32E primitive* $=$ *360*. The chosen filter permits saving 77.78% of the LUTRAMs compared to widely used 4th-order IIR filters.

The 14 DSPs instantiated are entirely used to speed up the SNN module, thus, each layer of the SNN takes advantage of the computational power of 7 DSPs. Three DSPs are used to accumulate the synaptic weights during the synaptic current computation, the remaining 4 are used for the double integration of the Loihi neuron, 2 per integrator. As regards the filter, it does not require any DSP, since the samples are not multiplied by any parameter. In a 4th-order IIR filter, to maintain the same throughput of one output sample per clock cycle, 9 multiplications should be computed in parallel, requiring 9 DSPs.

The BRAM requirement is 34.5 tiles, of which 1.5 BRAMs are used by the DMA, instanced to provide the broadband recording during the test phase, 1 BRAM is used by the spike detector to store the thresholds and the refractory periods, and the remaining 32 are used by the SNN. In the SNN architecture, two BRAMs are used by the integrators to implement the FIFO memory where are stored the neurons' state (half of a BRAM for each integrator), and the remaining 30 are used for storing the synaptic weights. Considering the weight memories of the two layers have the same size: *128 inputs* $\times$ *256 units* $\times$ *15 bits* $=$ *480 kb* for the first layer and *256 inputs* $\times$ *128 units* $\times$ *15 bits* $=$ *480 kb* for the second layer, and that each BRAM is 36 kb each, *ceil(480/36)* $=$ *14* BRAMs should be instanced per layer. However, because of the suboptimal word size of 60 bits (4 weights), the number rises to 30 BRAMs instead of 28, as if it was used 16-bit weights.

TABLE 6. Post-implementation hardware utilization: the first row shows the resource utilization of the design under test that comprises the spike detector, the SNN-based decoder and the AXI DMA used to stream the input samples during the test phase, the second row and the third rows represent the utilization of the spike detector and the spiking neural network.

Entity	LUT	REG	LUTRAM	DSP	BRAM
DUT	2,458	3,365	209	14	34.5
Detection	160	251	80	0	1
SNN	756	1,013	30	14	32

2) SPIKE SPARSITY SAVINGS

Spike sparsity has been exploited by instancing on each layer of the SNN a stack to store the addresses pointing to the active input sets. The effect of this architectural choice has been assessed by counting the number of saved additions during the synaptic current computation using real test data [13]. The savings are reported in Tables 7 and 8, respectively for dataset *N* (16 minutes and 43 seconds long) and *L* (11 minutes and 49 seconds long) [13]. The saving is reported separately for each layer and for both.

TABLE 7. Operations saved in dataset *N* [13].

	L1	L2	Total
Done	6.14e6 (5 %)	4.54e7 (14 %)	5.15e7 (12 %)
Saved	1.14e8 (95 %)	2.76e8 (86 %)	3.90e8 (88 %)
Total	1.20e8	3.21e8	4.41e8

On dataset N the spike sparsity aware architecture saves 95% of the sums in the first layer, and 86% in the second, for a total saving of 88%.

TABLE 8. Operations saved in dataset *L* [13].

	L1	L2	Total
Done	5.94e6 (7 %)	2.11e7 (9 %)	2.70e7 (9 %)
Saved	7.92e7 (93 %)	2.06e8 (91 %)	2.85e8 (91 %)
Total	8.51e7	2.27e8	3.12e8

The additions saved on dataset L are 93% in the first layer, and 91% in the second, for a total saving of 91%, even more than for dataset N.

It must be considered that the saved sums not only reduce the switching power of the digital system but also the time necessary for evaluating the decoder output. Considering the clock period of the system is set for the worst case scenario, where all the synapses are active, but on average less than the 12% are, the SNN on average terminates in the 12% of the time and could be put in sleep mode for the remaining 88% of the time, saving also static power, or on the contrary, it could be used to process larger sensor arrays. The computational and power savings are well balanced considering the resource requirements of the stacks reported in Table 9. Note that, the stacks are the only elements that alone permit the design to be aware of spike sparsity.

TABLE 9. Stack modules utilization.

	LUT	REG	LUTRAM
L1	75	171	0
L2	139	397	0
TOT	214	568	0

VI. COMPARISON WITH THE STATE OF THE ART

The presented neural interface is the first work that directly uses intracortical recorded spikes jointly with an SNN for decoding the neural signal, either implemented on FPGA or ASIC, to the best of our knowledge. Table 10 reports software decoders tested on the same benchmark dataset used in this work [13]. Table 10 reports the type of decoder used in each study, the number of parameters, and the total required memory when available. Moreover, since the dataset comprises two different recordings, the accuracy, measured in terms of the Pearson correlation coefficient (CC), is reported in the two last columns of the table.

TABLE 10. State of the art neural decoders comparison.

Work	Decoder	Feature	Params	Memory	CC N	CC L
This work	SNN	MUA	0.057M	105 kB	0.83	0.78
[10]	QRNN	MUA	-	-	0.84	0.73
[10]	GRU	ESA	-	-	-	0.78
[10]	LSTM	ESA	-	-	0.87	-
[19]	RNN	SUA	0.42M	1.6 MB†	0.91	-
[19]	GRU	SUA	1.19M	4.5 MB†	0.89	-
[19]	LSTM	SUA	1.56M	5.9 MB†	0.91	-
Deduced considering 32-bit parameters†						

On lines 2, 3, and 4 we show the accuracy of three deep learning decoders [10], a QRNN, a GRU, and an LSTM. The deep-learning models have been tested with several neural signal features, such as SUA, MUA, and ESA. The best model found in [10] for decoding the MUA was the QRNN of 400 units, which performed similarly to our work. On dataset N their decoder achieved 0.84 CC, whereas our SNN got 0.83. On dataset L, this work outperformed the QRNN with a CC of 0.78 compared to the correlation of 0.73 found by the QRNN. The best correlation found for dataset L in [10] is the one of the GRU model, which tied to our result using the ESA feature rather than the MUA; in the case of dataset N, the highest correlation was obtained using an LSTM model and the ESA feature, achieving 0.87. As regards the correlation obtained on dataset N for the GRU model, and on dataset L for the LSTM model, they were not available, but it can be assumed they were lower than the result of the QRNN since the QRNN is the best decoder found in their study overall.

Lines 5, 6, and 7 of Table 10 show the results obtained from three deep learning decoders: RNN, GRU, and LSTM [19], implemented using the PyTorch library, and tested on dataset N only. The models tested in [19] based their inferences on the SUA, computed using the Plexon Offline Sorter, which found 156 units. The decoders achieved outstanding correlation results of 0.91, 0.89, and 0.91 respectively. Moreover, in [19] is shown the number of parameters utilized for each model, making possible a memory requirements comparison

with our work. The RNN, which is the smallest model of the three, makes use of 0.42 million parameters, about 7.4 times more than the SNN model presented in this work. The GRU model requires using 20.9 more parameters than the SNN, the LSTM necessitates 27.4 more parameters than the SNN. Moreover, we assume a 32-bit floating-point representation for the parameters, as it is common in the Pytorch library, to infer the memory usage of the models in [19]. They respectively require 1.6, 4.5, and 5.9 MB, whereas our model occupies only 105 kB. Therefore, the memory requirements for using the presented model are respectively 6.4%, 2.3 %, and 1.7% of the ones required by the models in [19].

VII. CONCLUSION

We have presented a resource-power efficient intracortical neural interface system embedding a multiplier-less spike detection pipeline and a spike-sparsity-aware SNN decoder. The spike detector is equipped with a filter, dynamic threshold updates, refractory period spike burst limitation, and spike binning features, ensuring reliable spike detection by only employing five additions per channel and zero multiplications. The SNN model, used to decode the neural signal, takes advantage of the spike sparsity feature of intracortical recordings, by dynamically indexing the active synaptic weights during the computation of the synaptic currents, avoiding waste of dynamic power and accelerating the layers' inference. The effectiveness of the method was proved on two datasets, where the 88% and the 91% of the sums during the computation of the synaptic currents were saved, at the expense of 568 REGs and 214 LUTs.

Further improvements of this work are possible in several directions. It would be interesting to verify the accuracy of SNN models on more datasets, explore the impact on the decoding accuracy given by a reduced precision for the weights representation and investigate the effect of convolutional layers to exploit MEA spatial properties.

REFERENCES

[1] M. Śliwowski, M. Martin, A. Souloumiac, P. Blanchart, and T. Aksenova, "Decoding ECoG signal into 3D hand translation using deep learning," *J. Neural Eng.*, vol. 19, no. 2, Apr. 2022, Art. no. 026023.

[2] P. Busia, A. Cossettini, T. M. Ingolfsson, S. Benatti, A. Burrello, M. Scherer, M. A. Scrugli, P. Meloni, and L. Benini, "EEGformer: Transformer-based epilepsy detection on raw EEG traces for low-channel-count wearable continuous monitoring devices," in *Proc. IEEE Biomed. Circuits Syst. Conf. (BioCAS)*, Oct. 2022, pp. 640–644.

[3] F. M. Petrini et al., "Six-month assessment of a hand prosthesis with intraneural tactile feedback," *Ann. Neurol.*, vol. 85, no. 1, pp. 137–154, Jan. 2019.

[4] D. A. Moses, S. L. Metzger, J. R. Liu, G. K. Anumanchipalli, J. G. Makin, P. F. Sun, J. Chartier, M. E. Dougherty, P. M. Liu, G. M. Abrams, A. Tu-Chan, K. Ganguly, and E. F. Chang, "Neuroprosthesis for decoding speech in a paralyzed person with anarthria," *New England J. Med.*, vol. 385, no. 3, pp. 217–227, 2021.

[5] M. W. Slutzky, "Brain–machine interfaces: Powerful tools for clinical treatment and neuroscientific investigations," *Neuroscientist*, vol. 25, no. 2, pp. 139–154, May 2018.

[6] I. Obeid and P. D. Wolf, "Evaluation of spike-detection algorithms for a brain–machine interface application," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 6, pp. 905–911, Jun. 2004.

[7] R. Q. Quiroga, "Spike sorting," *Current Biol.*, vol. 22, no. 2, pp. R45–R46, 2012.

[8] N. Ahmadi, T. G. Constandinou, and C.-S. Bouganis, "Decoding hand kinematics from local field potentials using long short-term memory (LSTM) network," in *Proc. 9th Int. IEEE/EMBS Conf. Neural Eng. (NER)*, Mar. 2019, pp. 415–419.

[9] D. Sussillo, P. Nuyujukian, J. M. Fan, J. C. Kao, S. D. Stavisky, S. Ryu, and K. Shenoy, "A recurrent neural network for closed-loop intracortical brain–machine interface decoders," *J. Neural Eng.*, vol. 9, no. 2, 2012, Art. no. 026027.

[10] N. Ahmadi, T. G. Constandinou, and C.-S. Bouganis, "Robust and accurate decoding of hand kinematics from entire spiking activity using deep learning," *J. Neural Eng.*, vol. 18, no. 2, Apr. 2021, Art. no. 026011.

[11] A. Nurmikko, "Challenges for large-scale cortical interfaces," *Neuron*, vol. 108, no. 2, pp. 259–269, Oct. 2020.

[12] X. Wang, X. Lin, and X. Dang, "Supervised learning in spiking neural networks: A review of algorithms and evaluations," *Neural Netw.*, vol. 125, pp. 258–280, May 2020.

[13] T. Brochier, L. Zehl, Y. Hao, M. Duret, J. Sprenger, M. Denker, S. Grün, and A. Riehle, "Massively parallel recordings in macaque motor cortex during an instructed delayed reach-to-grasp task," *Sci. Data*, vol. 5, no. 1, pp. 1–23, Apr. 2018.

[14] *Utah Array*. Accessed: Jan. 10, 2023. [Online]. Available: https://blackrockneurotech.com/

[15] X. Wu, G. Li, S. Jiang, S. Wellington, S. Liu, Z. Wu, B. Metcalfe, L. Chen, and D. Zhang, "Decoding continuous kinetic information of grasp from stereo-electroencephalographic (SEEG) recordings," *J. Neural Eng.*, vol. 19, no. 2, Apr. 2022, Art. no. 026047.

[16] Y. Du, J. Jin, Q. Wang, and J. Fan, "EMG-based continuous motion decoding of upper limb with spiking neural network," in *Proc. IEEE Int. Instrum. Meas. Technol. Conf. (IMTC)*, May 2022, pp. 1–5.

[17] Y. Yang, J. Ren, and F. Duan, "The spiking rates inspired encoder and decoder for spiking neural networks: An illustration of hand gesture recognition," *Cognit. Comput.*, pp. 1–16, May 2022.

[18] H. Fang, Y. Wang, and J. He, "Spiking neural networks for cortical neuronal spike train decoding," *Neural Comput.*, vol. 22, no. 4, pp. 1060–1085, Apr. 2010.

[19] S.-H. Yang, J.-W. Huang, C.-J. Huang, P.-H. Chiu, H.-Y. Lai, and Y.-Y. Chen, "Selection of essential neural activity timesteps for intracortical brain–computer interface based on recurrent neural network," *Sensors*, vol. 21, no. 19, p. 6372, Sep. 2021.

[20] X. Ma, W. Zheng, Z. Peng, and J. Yang, "FPGA-based rapid electroencephalography signal classification system," in *Proc. IEEE 11th Int. Conf. Adv. Infocomm Technol. (ICAIT)*, Oct. 2019, pp. 223–227.

[21] M. Agrawal, S. Vidyashankar, and K. Huang, "On-chip implementation of ECoG signal data decoding in brain-computer interface," in *Proc. IEEE 21st Int. Mixed-Signal Test. Workshop (IMSTW)*, Jul. 2016, pp. 1–6.

[22] C. M. McCrimmon, J. L. Fu, M. Wang, L. S. Lopes, P. T. Wang, A. Karimi-Bidhendi, C. Y. Liu, P. Heydari, Z. Nenadic, and A. H. Do, "Performance assessment of a custom, portable, and low-cost brain–computer interface platform," *IEEE Trans. Biomed. Eng.*, vol. 64, no. 10, pp. 2313–2320, Oct. 2017.

[23] S. M. Khan, A. A. Khan, and O. Farooq, "Selection of features and classifiers for EMG-EEG-based upper limb assistive devices—A review," *IEEE Rev. Biomed. Eng.*, vol. 13, pp. 248–260, 2020.

[24] J. Fan and J. He, "Motor cortical encoding of hand orientation in a 3-D reach-to-grasp task," in *Proc. Int. Conf. IEEE Eng. Med. Biol. Soc.*, Aug. 2006, pp. 5472–5475.

[25] G. Saggese and A. G. M. Strollo, "A low power 1024-channels spike detector using latch-based RAM for real-time brain silicon interfaces," *Electronics*, vol. 10, no. 24, p. 3068, Dec. 2021.

[26] G. Leone, L. Raffo, and P. Meloni, "ZyON: Enabling spike sorting on APSoC-based signal processors for high-density microelectrode arrays," *IEEE Access*, vol. 8, pp. 218145–218160, 2020, doi: 10.1109/ACCESS.2020.3042034.

[27] G. Leone, L. Raffo, and P. Meloni, "A bandwidth-efficient emulator of biologically-relevant spiking neural networks on FPGA," *IEEE Access*, vol. 10, pp. 76780–76793, 2022.

[28] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, Feb. 2014.

[29] H. Irmak, F. Corradi, P. Detterer, N. Alachiotis, and D. Ziener, "A dynamic reconfigurable architecture for hybrid spiking and convolutional FPGA-based neural network designs," *J. Low Power Electron. Appl.*, vol. 11, no. 3, p. 32, Aug. 2021.

[30] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.
[31] F. Akopyan et al., "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.
[32] Z. Zhang and T. G. Constandinou, "Adaptive spike detection and hardware optimization towards autonomous, high-channel-count BMIs," *J. Neurosci. Methods*, vol. 354, Apr. 2021, Art. no. 109103.
[33] J. F. Kaiser, "On a simple algorithm to calculate the 'energy' of a signal," in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, vol. 1, 1990, pp. 381–384.
[34] T. P. Vogels and L. F. Abbott, "Signal propagation and logic gating in networks of integrate-and-fire neurons," *J. Neurosci.*, vol. 25, no. 46, pp. 10786–10795, Nov. 2005.
[35] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2018, pp. 1419–1428. [Online]. Available: http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf
[36] (2022). *Lava Software Framework*. [Online]. Available: https://lava-nc.org/index.html
[37] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

GIANLUCA LEONE received the B.S. degree in electronics engineering from the University of Cagliari, Cagliari, Italy, in 2016, and the M.S. degree in electronics engineering from Politecnico di Torino, Turin, Italy, in 2019. He is currently pursuing the Ph.D. degree in electronic and computer engineering with the University of Cagliari. His research interest includes the development and optimization of hardware accelerators on FPGA.

LUIGI RAFFO joined the Department of Electrical and Electronic Engineering, University of Cagliari, Italy, in 1994. He has been a Full Professor of electronics with the University of Cagliari, since 2006. He has been the Coordinator of the course of studies in biomedical engineering, from 2006 to 2012 and from 2017 to 2018. He teaches courses on system design, digital and analog electronics design, and processor architectures. Since 2012, he has been a Rector's Delegate for international research projects. His research interests include the study, design, and development of systems and micro-systems for applications where high performance, high efficiency, and low power are required. In such a field, he is the author of more than 200 scientific articles.

PAOLO MELONI has been an Assistant Professor with the University of Cagliari, since 2012. He teaches advanced embedded systems with the University of Cagliari. He is currently the Scientific Coordinator of the ALOHA (www.aloha-h2020.eu) H2020 Project. He is the author of a significant track of international research articles. His research interests include the development of advanced digital systems and the application-driven design and programming of multi-core on-chip architectures and FPGAs.

Open Access funding provided by 'Università degli Studi di Cagliari' within the CRUI CARE Agreement