UNICA

UNIVERSITÀ
DEGLI STUDI
DI CAGLIARI

Ph.D. DEGREE IN
Electronic and Computer Engineering

Cycle XXXV

Integrating Biological and Artificial Neural Networks Processing on FPGAs

ING-INF/01

Ph.D. Student: Gianluca Leone

Supervisor Prof. Paolo Meloni

Final exam. Academic Year 2021/2022
Thesis defence: February 2023 Session

Contents

Abstract

Neural interfaces are rapidly gaining momentum in the current landscape of neuroscience and bioengineering. This is due to a) unprecedented technology capable of sensing biological neural network electrical activity b) increasingly accurate analytical models usable to represent and understand dynamics and behavior in neural networks c) novel and improved artificial intelligence methods usable to extract information from recorded neural activity. Nevertheless, all these instruments pose significant requirements in terms of processing capabilities, especially when focusing on embedded implementations, respecting real-time constraints and exploiting resource-constrained computing platforms. Acquisition frequencies, as well as the complexity of neuron models and artificial intelligence methods based on neural networks, pose the need for high throughput processing of very high data rates and expose a significant level of intrinsic parallelism. Thus, a promising technology serving as a substrate for implementing efficient embedded neural interfaces is represented by APSoCs, that enable the use of configurable logic, organizable memory blocks and parallel DSP slices. In this thesis we assess the usability of APSoC in this domain by focusing on a) real-time processing and analysis of MEA-acquired signals featuring spike detection and spike sorting on 5,500 recording electrodes b) real-time emulation of a biologically-relevant spiking neural network counting 3,098 Izhikevich neurons and 9.6e6 synaptic interconnections c) real-time execution of spiking neural networks for neural activity decoding during a delayed reach-to-grasp task addressing low-power embedded applications.

Chapter 1

Introduction

Neural sensor development has been gaining pace during the last few years thanks to the combined effort of engineers and neuroscientists in both private companies and research centers. New generation CMOS multielectrode arrays (MEAs) and CMOS high-density multielectrode arrays (HDMEA) guarantee higher spatio-temporal resolution than previously adopted recording arrays of sensors. Companies in the field, such as 3Brain, commercialize several planar HDMEAs [1] featuring 4,096 recording sites sampled at 18 kHz, placed on a 64x64 grid with an electrode-to-electrode distance of 60 µm. Neuralink [2] presented a MEA embedding 3,072 recording sites sampled at 18.6 kHz, distributed across 96 threads of 32 electrodes with electrode-to-electrode distance in the range 50-75 µm and thread-to-thread spacing above 300 µm to foster wide area coverage over multiple brain regions. Conversely, Neuropixel 2.0 [3] is a high-density probe that allows sensing the activity on a reduced brain region compared to [2], but with a more densely populated array of sensors and in multiple cortical layers. Neuropixel 2.0 counts 5,120 electrodes sampled at 30 kHz, distributed over four shanks of 1,280 electrodes each. The shanks are 250 µm apart and 10 mm long; on each shank, the sensors are distributed on two columns spaced 32 µm, whereas the electrodes on the same column present a center-to-center distance of 15 µm.

The emerging CMOS neural sensor technologies count tens of times more recording sites, placed tens of times more densely than previously adopted arrays of sensors [4], thus, requiring new neural interfaces capable of keeping up the downstream processing on an augmented flow of data, providing low-latency responses to effectively exploit real-time interactions with

the biological tissue, permitting the realization of novel neuroprosthetic implants, or deepening the understanding of neural networks functioning principles. At the same time, increasingly accurate analytical neural models, when supported by enough computational power, permit the emulation of large portions of the brain neural dynamics with single-cell resolution, fostering neural networks' dynamic comprehension as well, and potentially bridging the gap between simple localized neural circuit behavior and complex cognitive processes distributed on wider brain regions. Furthermore, artificial intelligence models based on artificial neural networks are promising tools for decoding patients' intentions, positioning as a fundamental element in the control of neuroprosthetic implants. Emerging CMOS neural sensors provide a more accurate and dense sampling of the neural activity, presumably enabling more precise patient intentions decoding, but requiring the artificial intelligence model to process orders of magnitude more data while being subjected to stringent latency constraints.

A promising technology for addressing highly parallel and computational-intense real-time processing of neural data, real-time low-latency emulation of large-scale brain models, and real-time low latency execution of artificial neural network models is represented by All Programmable System on Chips (APSoCs). These devices embed: 1) hardwired Digital Signal Processor (DSP) slices, particularly well suited for the most computationally intense portion of the processing, constituted of multiplications and multiply-and-accumulate operations: 2) a fabric of programmable logic, flexible enough for accelerating even the more specific details of neural DSP algorithms and artificial neural networks inference; 3) configurable dual-port blocks of memory with selectable port widths, ideal to foster hardware adaptability over a different number of recording sites and emulated neurons; 4) an ARM-based Processing System (PS) [5], useful for taking care of housekeeping tasks, such as memory management and user interaction, and at the same time embedding enough computing power to execute refinement algorithms to increase the accuracy of the system as a whole.

APSoCs features are ideal to host accelerators to permit: a) real-time processing of MEAs and HDMEAs neural signals such as spike detection and spike sorting on thousands of recording channels; b) real-time emulation of large-scale analytical neuron models useful for representing and understanding dynamics and behavior of biological neural networks ; c) real-time execution of artificial intelligence methods based on artificial neural networks aiming to de-

code the neural signal, inferring the patient intentions, and providing a first step in the control of new neuroprosthetic implants operating in the central nervous system.
The main contributions and publications of the thesis can be summarized as follows:

- We demonstrated APSoCs are valid computing platforms for real-time neural signal analysis at the edge, especially when thousands of recording sites are considered. In **"ZyON: Enabling Spike Sorting on APSoC-Based Signal Processors for High-Density Microelectrode Arrays"** [6], we presented a spike detection and sorting system implemented on APSoC addressing up to 5,500 recording sites in closed-loop low-latency (2.3 ms) experiments. We exploited the parallel computational capabilities of the programmable logic to take care of the more demanding portion of the neural algorithm while the ARM-based processing system refined the programmable logic parameters in real-time to improve the quality and the yield of the sorting results. Our implementation allows real-time detection and sorting of the spikes on the highest number of recording channels at the state of the art;

- We demonstrated the physiological spike propagation delay of biological neural networks can be exploited in the real-time emulation of artificial neural networks, in particular, in **"A Bandwidth-Efficient Emulator of Biologically-Relevant Spiking Neural Networks on FPGA"** [7], we demonstrated APSoCs provide an adequate off-chip ram bandwidth for emulating in real-time arbitrarily connected spiking neural networks counting up to 3,098 neurons and 9.6e6 synaptic interconnections, considering a time resolution of 0.1 ms and a spike propagation delay of 3 ms, i.e. a system latency of 3 ms. Furthermore, we presented a study of the Izhikevich neuron model execution with fixed-point arithmetic on Xilinx DSP slices, providing a scheme for saving 39% of the memory necessary for storing the neurons' parameters and achieving a negligible difference in the spiking pattern when compared with a floating-point model;

- We demonstrated spiking neural networks are an adequate candidate for real-time low-power spike decoding by presenting a spike decoder implementation hosted by an APSoC composed of a multiplier-less spike detection processing chain cascaded with a spiking neural network based decoder. The system achieved results comparable with

other neural decoders at the state of the art when tested on the same delayed reach-to-grasp dataset, publicly available in [8]. The spiking neural network model required 7.36 times fewer parameters than the smallest neural decoder validated on the same dataset, and when tested on real recorded data 90% of the computations are saved due to spike sparsity.

The following chapters are organized as follows: Chapter 2 contains an exploration of several spike detection methods, focusing on accuracy, computational complexity, and real-time viability; Chapter 3 presents a spike detection and sorting system addressing up to 5,500 recording sites for closed-loop experiments published in [6]; Chapter 4 describes a real-time bio-realistic spiking neural network emulator of Izhikevich neurons counting 3,098 neurons and 9.6e6 synaptic interconnections published in [7]; Chapter 5 presents a neural decoder implementation exploiting a multiplier-less spike detection method and a spiking neural network for online spike decoding during a delayed reach-to-grasp task; Chapter 6 is left to conclusions and future works speculations.

Chapter 2

A study of real-time spike detection methods

Intracortical multielectrode arrays measure the occasional extracellular depolarization of neurons surrounding the electrode surface [9]. These events are called action potentials, or spikes, and have been widely used for studying several phenomena on an extended range of applications including spike sorting [10], prosthetic device control [11] and speech decoding [12]. Spike detection is the first step of several neural signal processing studies, addressing both online and offline analysis. In fact, embedding real-time spike detection in the recording system limits consistently the output data rate, permitting the acquisition of data from more recording sites also for offline analysis [13]. New generation multielectrode arrays count thousands of recording sites, thus, require appropriate spike detection algorithms that could scale and keep up with such an amount of data without exceeding the strict power limitations of implantable devices [14].

The first chapter of the Thesis is used to present accuracy and computational complexity comparisons between different spike detection methods. The results serve as a solid basis to guide the choices taken in the next chapters, where spike detection will be the prime component to enable more complex real-time neural signal analysis. The following sections are organized as follows: Section 2.1 presents in detail the typical processing steps composing spike detection algorithms, such as filtering, spike emphasis, and spike threshold evaluation; Section 2.2 and 2.3 contain comparisons between the methodologies studied in the previous section, respec-

tively by the point of view of the accuracy and the computational complexity; the results are discussed in Section 2.4, Section 2.5 is left to conclusions and indication of future directions for real-time spike detection algorithms.

2.1 Spike Detection Processing Steps

Spike Detection methods and algorithms aim to spot the spikes along the neural activity recorded by electrodes. Figure 2.1 introduces the typical steps that constitute the Spike Detection processing chain:

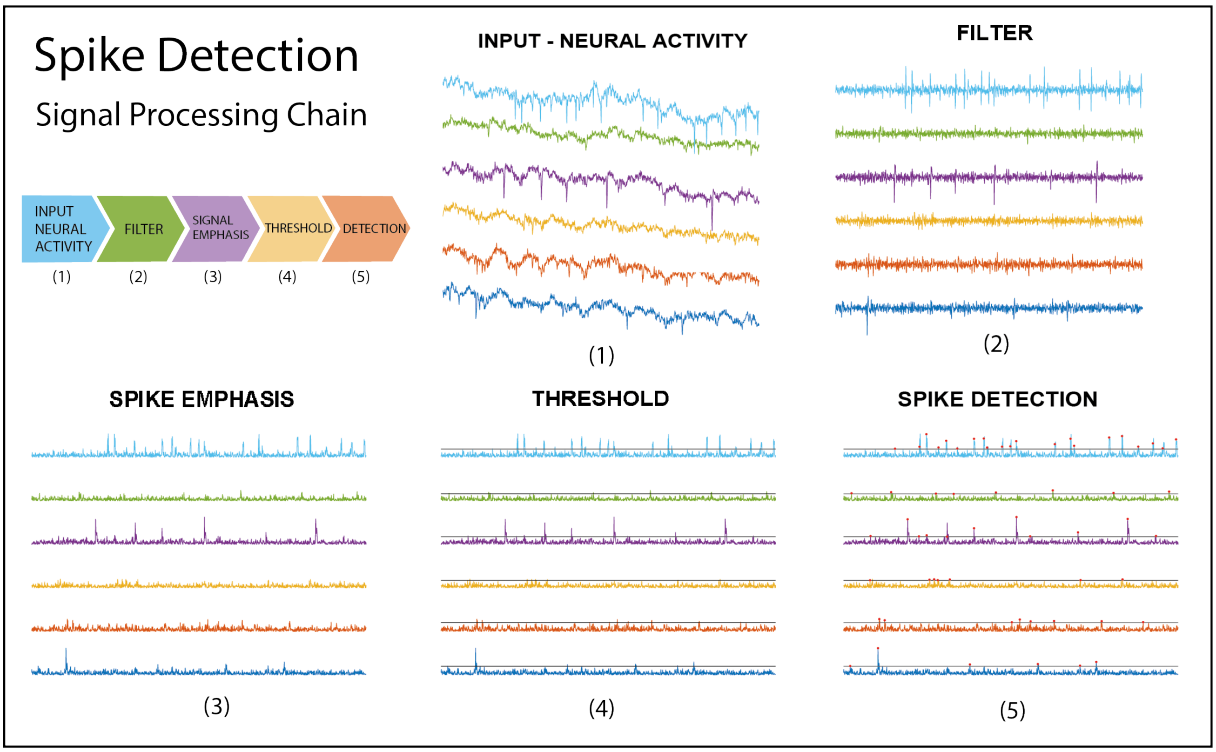


Figure 2.1: Spike detection processing chain can be divided into 4 sub-tasks: The input broadband recording (1) is filtered (2) to remove low-frequency components; the signal is emphasized to improve the signal-to-noise ratio (3); a measure of the noise is used to compute a threshold (4) used to detect the spikes when exceeded (5).

- Filtering: the frequencies outside the band of interest are filtered. The main focus lies in removing the frequency in the band [0, 300] Hz [15].
- Spike emphasis: relying on the pointed shape of the spikes, the signal is processed to amplify fast variations (the spikes) to increase the signal-to-noise ratio;

- Threshold computation: a measure of the noise, such as standard deviation, root mean square, or the mean of the signal is usually multiplied by a constant factor;

- Threshold crossing detection: the threshold is used as a point of comparison, when the signal exceeds the threshold value a spike is detected. A refractory period during which the neuron is not capable of responding to the stimuli and fires an additional spike is usually considered to avoid detecting multiple times the same spike.

2.1.1 Filtering

Digital filters are widely studied processing elements used for removing the signal frequency components out of the band of interest. In the spike detection domain, filters are mostly used to remove the lower frequency components in the range [0, 300] Hz. Digital filters can have an Infinite Impulse Response (IIR), characterized by Eq.2.1, or a Finite Impulse Response (FIR) characterized by Eq.2.2.

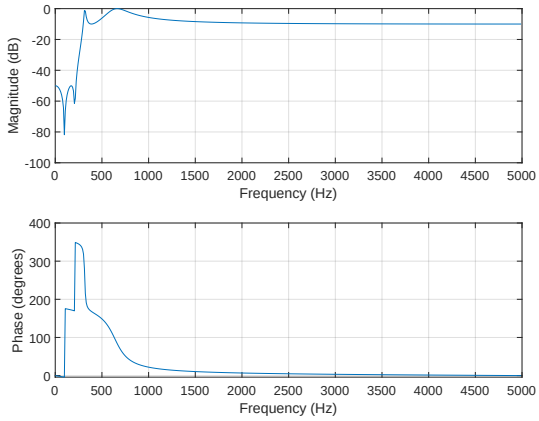
$$y(n) = \sum a_i y(n-i-1) + \sum b_i x(n-i) \tag{2.1}$$

$$y(n) = \sum b_i x(n) \tag{2.2}$$

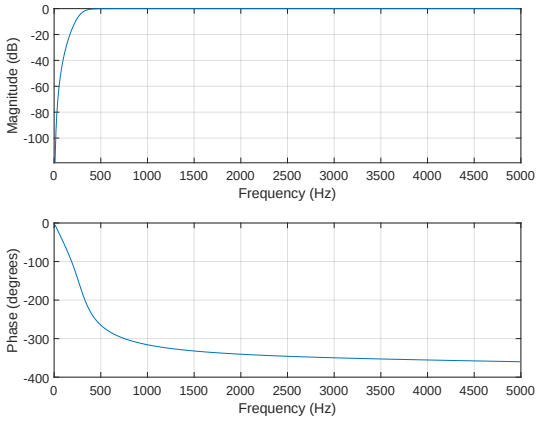
Where x is the digital input signal, y is the filter output, a and b are the filter coefficients, respectively poles, and zeros. Infinite Impulse Response filters, such as Elliptic and Butterworth filters, present good filtering performance at a lower order than FIR filters, because of the presence of the poles, thus, they have a retained computational complexity. Fig.2.2 and 2.3 show the frequency response of four digital filters, a 4th-order Elliptic IIR filter and a 4th-order Butterworth IIR filter in Fig.2.2, a 4th- and 60th-order FIR filter in Fig.2.3. All the filters are high-pass filters with a cut-off frequency of 300 Hz. It is visible how the roll-off[1] is steeper for IIR filters and it is required to increase the FIR filter order up to about 60 to obtain a similar behavior, as shown in Fig.2.3 (B).

However, in spite of their lower computational complexity, one of the drawbacks of using IIR filters is that they can become unstable. The stability condition for causal IIR filters entails

[1]Roll-off: filter transfer function steepness.

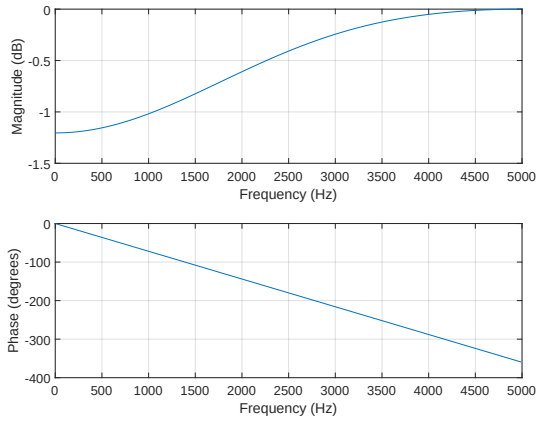


(a) 4th-order Elliptic IIR filter

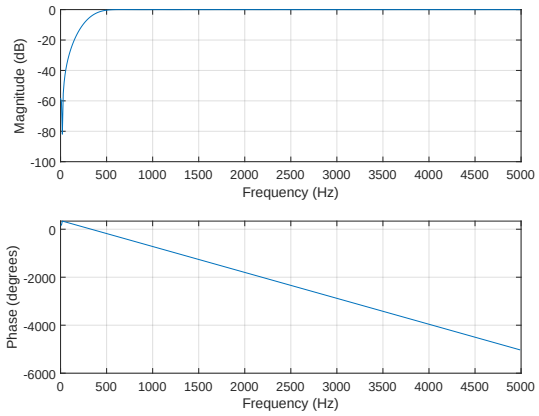


(b) 4th-order Butterworth IIR filter

Figure 2.2: IIR filters frequency response.



(a) 4th-order FIR filter



(b) 60th-order FIR filter

Figure 2.3: FIR filters frequency response.

all the poles *a* to be smaller than *1* as stated by Eq.2.3. Even though IIR filters coefficients are properly chosen during the design phase and respect Eq. 2.3, this condition could no longer apply once the filter is implemented in hardware, and the coefficients are approximated to the chosen fixed point representation.

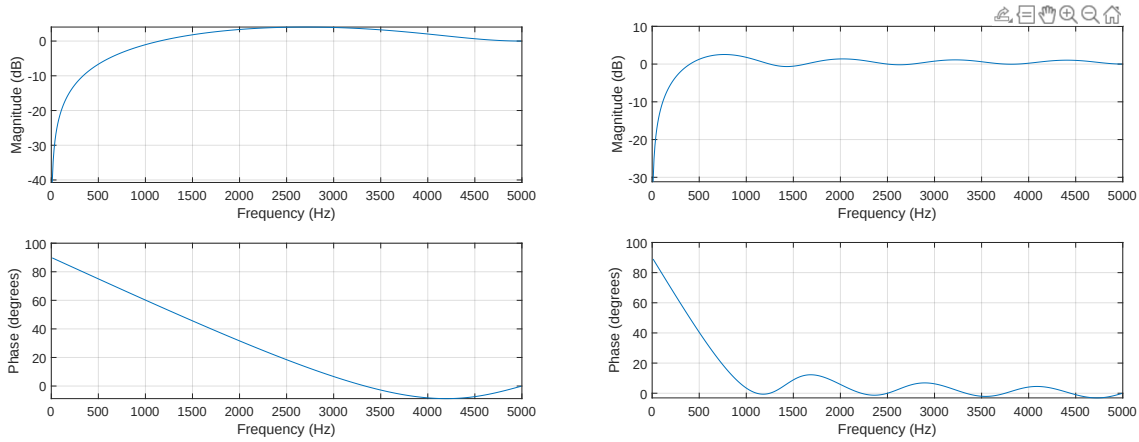
$$|a_i| < 1 \; \forall i \tag{2.3}$$

Usually, to circumvent this problem, it is preferred to use cascaded biquadratic filters.
Despite the higher computational complexity required by FIR filters to obtain a roll-off similar to IIR filters, they are sometimes preferred because of their intrinsic stability and their linear phase response, which guarantees no output phase distortion. In fact, in some spike detection follow-up signal analysis, such as spike sorting, it is crucial maintaining the spike shape intact, since depending on the spike shape it is possible to recognize when the spikes are fired by the same neuron. Therefore, FIR filters are still widely used.
Non-canonical FIR filters can be found in literature, such as the Moving Average (MA) filter, i.e. a FIR filter where all the coefficients are equal to the inverse of the filter order *N* as in Eq.2.4, or the case of the Moving Average Difference (MAD) filter [16], that implements a high-pass filter by subtracting to each sample the moving mean of the signal, computed thanks to a moving average filter as in Eq.2.5.

$$MA(n) = \frac{1}{N} \sum x(n-i) \tag{2.4}$$

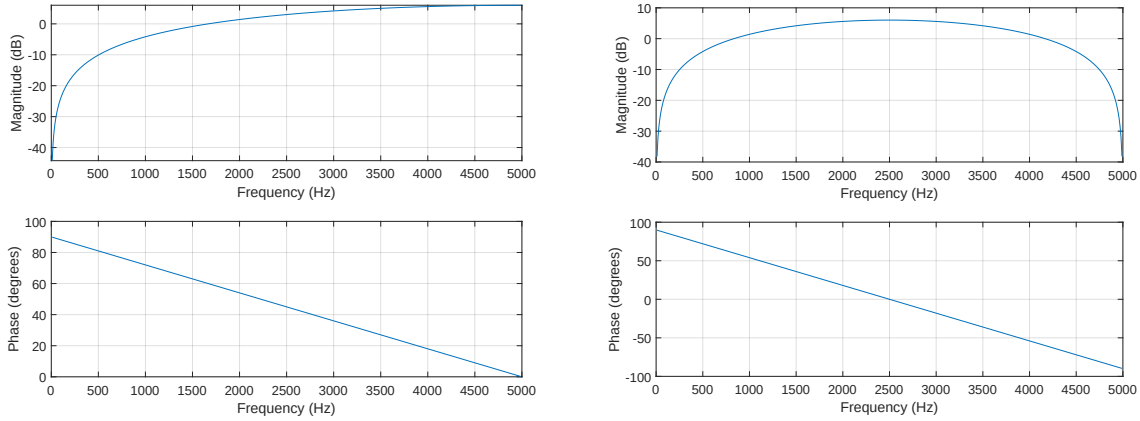
$$MAD(n) = x(n) - MA(n-1) \tag{2.5}$$

The moving average difference filter frequency response is shown in Fig.2.4 (A) and (B), where respectively a 2nd and an 8th-order Moving Average filters were used to compute the signal mean. The so obtained high-pass filters present a steeper roll-off than FIR filters of a similar order. Unfortunately, the linear phase response is lost because the filter coefficients are no more symmetric. Anyway, not having any poles, they are still intrinsically stable.
Another non-canonical but still effective filter is the difference filter [17]. In this case, the high-pass behavior is obtained by simply subtracting adjacent samples. Eq.2.6 and 2.7 model



(a) 2nd-order MAD filter (b) 8th-order MAD filter

Figure 2.4: Moving average difference filters frequency response.



(a) 1st-order difference filter (b) 2nd-order difference filter

Figure 2.5: Difference filters frequency response.

the behavior of first- and second-order difference filters.

$$y(n) = x(n) - x(n-1) \tag{2.6}$$

$$y(n) = x(n) - x(n-2) \tag{2.7}$$

The frequency responses of the difference filters are shown in Fig.2.5 (A) and (B). The 1st-order difference filter roll-off is lower than the 2nd-order moving average difference filter, however, the 2nd-order difference filter magnitude response is similar. As regards the phase response, the difference filter exhibit a linear phase response within the band of interest.

Moving Average Difference, and Difference filters reduce the filtering computational complexity, in fact, no multiplications take place, and as regards the division of moving average filters, it can be substituted by a *right-shift* if the order of the filter is conveniently chosen equal to a power of two.

Table 2.1 summarizes the main features of the filters analyzed in this subsection. It can be seen that IIR-type filters present a moderate computational complexity, introduce phase distortion, and are subject to numerical instability. Although FIR-type filters are always stable and do not introduce any phase distortion, it is required increasing the filter order to about 60 to obtain a magnitude response comparable to the one of 4th-order IIR filters, making FIR filters computationally intensive solutions. On the other hand, Moving Average Difference and Difference filters present a satisfying magnitude response at a limited order, i.e. between 2 and 8 for the Moving Average Difference filter and 1 and 2 for the Difference filter. Moreover, they are always stable, and their phase response is linear within the band of interest (except for the 8th-order MAD filter). Finally, MAD and Difference filters are the least computationally expensive models, since it is not required any multiplication, they are therefore the best candidates for implementing neural interfaces counting thousands of recording channels.

Filter type	Stability	Order	Phase-distortion	Computational complexity
IIR	Can become unstable	4th	Yes	Moderate
FIR	Always stable	60th	No	High
MAD	Always stable	2nd-8th	Low	Low
Diff	Always stable	1st-2nd	Low	Low

Table 2.1: Filters' features summary

2.1.2 Spike Emphasis

Spike emphasis algorithms are widely used ways to increase the signal-to-noise ratio of the signal and increase the spike detection accuracy at a low computational cost.

The simplest spike emphasis method consists of computing the absolute value of the neural signal [18]. The absolute value is not actually a real spike emphasis method, since it does not improve the SNR of the signal, it is indeed equivalent to applying both positive and negative thresholds [19].

Another popular spike emphasis method is the Non-linear Energy Operator (NEO) [20], described by Eq.2.8.

$$y(n) = x^2(n-1) - x(n)x(n-2) \tag{2.8}$$

The strategy behind NEO transformation is inherent to the typical pointed shape of the spikes. In fact, by considering the sample *x(n-1)* to be the spike crest, it is obvious how the previous and next samples, *x(n-2)* and *x(n)* would be much smaller, and therefore the pointed spike shapes would be amplified. However, this can either be or not be the case, depending on the spike steepness compared to the sampling frequency. To work around this problem, it is usually preferred to use a more general formula for NEO, that allows considering also further samples as shown in Eq.2.9

$$y(n) = x^2(n-k) - x(n)x(n-2k) \tag{2.9}$$

Along the lines of NEO, the Amplitude Slope Operator (ASO) was introduced in [16]. ASO multiplies the current value of the signal by its derivative, obtained by sample subtraction, as shown in Eq.2.10. In presence of a spike, both the amplitude and the first derivative of the neural signal are in fact larger than usual.

$$y(n) = x(n)(x(n) - x(n-k)) \tag{2.10}$$

Spike emphasis methods aim to increase the yield of spike detection. However, they inevitably introduce additional computations, that are carried out on each channel, and therefore, it should be verified if their insertion effectively improves the spike detection accuracy, i.e. are the added computations worth their cost? In the case of the absolute-value method, the computations are negligible, i.e. the technique consists in computing the 2's complement of the sample, whereas, in the case of NEO and ASO are introduced multiplications and additions, as well as memory elements, and the number of processing and memory elements grow linearly with the number of electrodes of the recording array. Table 2.2 summarises the complexity of the spike emphasis method from the point of view of the computational and memory require-

ments. In order to process the data sampled from thousands of recording channels it should be selected the least-expensive method that still guarantees an adequate solution to the target problem of the experiment. Therefore, it should be better to use in order: 1) the Absolute Value, 2) the Amplitude Slope Operator, 3) the Non-Linear Energy Operator, 4) or other more complex methodologies if needed. The accuracy improvements and the computational cost of the above-mentioned spike emphasis methods are described more in detail in Sections 2.2 and 2.3.

Method	Computational complexity	Memory requirements
Absolute Value	Low	None
NEO	High	High
ASO	Moderate	Moderate

Table 2.2: Computational and memory requirements of spike emphasis methods

2.1.3 Spike Threshold

The choice of the spike threshold has a key role in the accuracy of spike detection. The threshold is usually set by multiplying a constant factor, determined offline, by a measure of the noise.

Offline spike detection algorithms can rely on the noise measure on the whole neural signal recording, as well as more complex ways to evaluate the noise value. This is the case of [18], where the noise of the signal is computed by evaluating the median of the signal as shown in Eq.2.11.

$$THR_{median} = \alpha \times median\left(\frac{|x|}{0.6754}\right) \tag{2.11}$$

Where α is a constant factor and x is the input signal. This strategy is indeed unfeasible for real-time application, because the whole neural recording is used, and also because the median computation requires ordering the neural signal samples, which is a memory- and computational-intense task.

Real-time spike detection systems either rely on noise estimation based on fixed time windows or based on sliding time windows. Fixed time window-based noise estimators update the noise estimation every time the time window expires. Sliding time window-based noise estimators

update their value at every incoming sample, fitting the noise value more closely. However, it is required to store the samples of the whole window. In either case, the window size choice is of prime importance, since a too-short window could suffer too much because of the spikes within the window. To overcome the noise estimation error due to the spike presence, it is possible to remove the spikes samples from the noise estimation once detected, as in [21].
A widely used noise estimation method is the Mean Square (MS) value of the signal Eq.2.12, usually preferred to the Root Mean Square (RMS) value Eq.2.13 to avoid the square root computation. Note that W is the number of samples within the time window.

$$THR_{MS} = \frac{\alpha}{W} \sum_{i=0}^{M} x^2(i) \tag{2.12}$$

$$THR_{RMS} = \sqrt{\frac{\alpha}{W} \sum_{i=0}^{W} x^2(i)} \tag{2.13}$$

In place of the RMS or MS value, the standard deviation of the signal, or its square, the variance, can either be used Eq.2.14 and 2.15.

$$THR_{STD} = \sqrt{\frac{\alpha}{W} \sum_{i=0}^{W} [x(i) - mean(x(i))]^2} \tag{2.14}$$

$$THR_{VAR} = \frac{\alpha}{W} \sum_{i=0}^{W} [x(i) - mean(x(i))]^2 \tag{2.15}$$

$$\tag{2.16}$$

In addition, also the mean of the signal can be used to set the threshold, as in [17].

$$THR_{MEAN} = \frac{\alpha}{W} \sum_{i=0}^{W} x(i) \tag{2.17}$$

More sophisticated algorithms, such as [21], also consider the spike amplitudes during the threshold computation, implementing a hybrid mechanism that accounts for both the neural

signal noise μ_{noise} and the mean spike crest amplitudes μ_{spike} as shown in Eq.2.18.

$$THR_{DOUBLE} = \alpha\mu_{noise} + \beta\mu_{spike} \tag{2.18}$$

Where α and β are two constants to be determined. In general, the approach in [21] can be used in combination with all the signal noise estimation methods described above, by substituting the term μ_{noise} with the signal mean, the mean square, or the variance.

As regards the constant gain used to multiply the noise estimation α, it is worth noting that is application specific. Higher thresholds permit detecting the activity of the neurons closer to the electrode tip, whereas lower thresholds permit detecting the activity of further neurons [22].

Table 2.3 summarises the computational complexity and memory requirements of the three methods used to establish the signal noise: Mean Square (MS), Variance (VAR), and Mean. The mean value of the signal is the easiest way to evaluate the signal noise since it entails accumulating the raw samples, whereas the Mean Square uses the accumulation of the square of the samples. The variance computation is the most complex noise estimation method since it implies accumulating the squares of the difference between the incoming samples and the signal mean. As for the spike emphasis method choice, it is advisable to use the least expensive method that still allows achieving an acceptable accurate problem solution, therefore, in order: 1) Mean value, 2) Mean Square, 3) Variance. The accuracy and a detailed analysis of the memory and computational cost are available in Sections 2.2 and 2.3.

Method	Computational complexity	Memory requirements
MS	Moderate	Moderate
VAR	High	High
Mean	Low	Low

Table 2.3: Computational and memory requirements of spike threshold methods

2.2 Spike detection methods accuracy

Spike detection methods have several degrees of freedom: the choice of the filter, the spike emphasis algorithm, the type of threshold, its window size and type (fixed or sliding), and the choice of the refractory period. In this section, using as a benchmark the synthetic dataset [18], the accuracy of the three spike emphasis methods presented in Section 2.1.2 are analyzed: Absolute Value, Non-linear Energy Operator, and Amplitude Slope Operator; moreover, it is assessed the yield of the three different measures of the signal noise presented in Section 2.1.3: Mean Square, Variance, and Mean of the signal. Table 2.4 contains the list of the spike detection algorithms tested in this section.

The time window is kept fixed at 0.78 seconds, however, ten different time windows are tested with NEO and RMS methods only, to verify the effect of time window variations on accuracy. The sliding window is not considered because its implementation requires elevated memory resources, and scales poorly with the number of electrodes.

Finally, unfortunately, the dataset [18] does not comprise any low-frequency components, i.e. the local field potential is not simulated, therefore the filters cannot be tested effectively on this dataset.

Threshold	Emphasis
Mean square	Absolute value
Mean square	Non-linear Energy Operator
Mean square	Amplitude Slope Operator
Variance	Absolute value
Variance	Non-linear Energy Operator
Variance	Amplitude Slope Operator
Mean	Absolute value
Mean	Non-linear Energy Operator
Mean	Amplitude Slope Operator

Table 2.4: Spike detection configurations

2.2.1 Reference benchmark dataset

To test the accuracy of different detection methods is used [18] as a benchmark. The dataset contains 18 synthetic single-channel neural signal simulations, each simulation contains the

neural activity of three neurons. The simulations have different levels of noise: 0.05, 0.01, 0.15, 0.2, 0.25, and 0.35. The noise of each simulation indicates the standard deviation of the synthetic dataset. The simulations were created starting from real spike waveforms recorded in the neocortex[2] and basal ganglia[3], whereas the background noise is obtained by adding together random spikes.

2.2.2 Spike emphasis and spike threshold accuracy

Three spike emphasis methods: absolute value, Non-linear Energy Operator, and Amplitude Slope Operator have been tested on the 18 neural simulations of [18] using three different ways to estimate the threshold: the mean square, the variance, and the mean of the signal. The accuracy of the methods is assessed by using the F_{score} operator, described by Eq.2.19.

$$F_{score} = \frac{TP}{TP + 0.5(FP + FN)} \tag{2.19}$$

Where *TP* are the true positives, i.e. the correctly detected spikes, *FP* are the false positives, i.e. spikes detected by mistake, and *FN* are the false negatives, i.e. non-detected spikes.
The average accuracy results when the mean square threshold is used are shown in Table 2.5. The F_{score} is above 91% for the three methods, NEO and ASO perform slightly better than the absolute value method achieving respectively 93.86% and 94.48%. The absolute value method detects fewer true positives and has more false positives and false negatives than the other methods. ASO and NEO perform similarly, however, the number of false positives in the case of ASO is significantly lower, only 0.85% compared to the 3.64% of NEO. This feature should be considered in applications where the presence of false positives can significantly impact the system's behavior.

Table 2.6 contains the accuracy obtained by using the variance of the signal as a basis to set the thresholds. The same considerations still hold also in this case, however, the F_{score} of ABS, NEO, and ASO are in general lower than using the MS-based thresholds. Furthermore, the

[2]Neocortex: the brain area appointed to high-order functions, such as cognition, sensory perception, and motor control [23].

[3]Basal ganglia: a group of sub-cortical neural structures involved in functions as action selection and reinforcement learning [24].

Emphasis	F_{score}	TP	FP	FN
ABS	91.76%	89.40%	5.22%	10.60%
NEO	93.86%	91.19%	3.64%	8.81%
ASO	94.48%	90.30%	0.85%	9.70%

Table 2.5: Mean square-based threshold accuracy

number of false positives of NEO is significantly increased, reaching 14.03%.

Table 2.7 shows the accuracy of the spike detection when a mean-based threshold is used.

Emphasis	F_{score}	TP	FP	FN
ABS	86.66%	82.53%	2.33%	17.47%
NEO	90.51%	90.27%	14.03%	9.73%
ASO	93.29%	88.54%	0.96%	11.46%

Table 2.6: Variance-based threshold accuracy

ASO, which was the best candidate so far, does not work when coupled with the mean threshold. The reason lies in the fact that the signal average after the ASO operator has been applied is not a good metric to evaluate its noise. On the flip side, NEO, when associated with a mean-based threshold, achieves the best performances, reaching an F_{score} of 94.68% and true positives, false positives, and false negatives are all better than in the two previous cases. The absolute value method performs similarly to the case where it is paired with the Mean Square based threshold.

Emphasis	F_{score}	TP	FP	FN
ABS	91.64%	87.93%	2.82%	12.07%
NEO	94.68%	91.30%	1.60%	8.70%
ASO	19.98%	92.14%	730.18%	7.86%

Table 2.7: Mean-based threshold accuracy

2.2.3 Threshold time window size

In this section, the time window is let vary, while NEO and RMS threshold are used to detect the spikes, and their hyperparameters are left untouched (the constant to which the noise estimation is multiplied $alpha$). During the previous experiments, the window size was fixed at about 0.78 s. Table 2.8 shows the F_{score}, true positives, false positives, and false negatives for

9 different timing windows, varying from 0.043 seconds to 10.92 seconds. The 0.78 s window

W [s]	F_{score}	TP	FP	FN
10.92	81.19%	69.82%	2.49%	30.18%
5.46	89.20%	82.60%	3.01%	17.40%
2.73	92.35%	88.18%	3.24%	11.82%
1.37	93.50%	90.33%	3.35%	9.67%
0.78	93.86%	91.19%	3.64%	8.81%
0.34	93.72%	91.20%	4.01%	8.80%
0.17	92.82%	90.47%	5.28%	9.53%
0.085	89.54%	88.46%	10.33%	11.54%
0.043	75.77%	84.14%	39.34%	15.86%

Table 2.8: NEO and RMS accuracy at various window sizes

used in the previous experiment is the case with the highest F_{score}. Moving either to longer or shorter time windows causes a decrement in accuracy.

Decreasing the time window size the mean square value of the signal is computed with a higher timing resolution, leading to windows with a different number of spikes. The spike samples introduce errors in the noise evaluation, during windows with a high number of spikes the noise is estimated higher than it is, entailing a higher threshold that increases the false negatives, up to 15.86%, obtained for a window of 43 ms. In addition, during time windows with few spikes, where the noise estimation is more accurate, the hyperparameter chosen to multiply the noise estimation is too low. The low threshold causes an increased number of false positives of 39.34%, obtained for the 43 ms window.

Longer time windows have a more even spike distribution among them, consequently, the distribution of the thresholds is more compact, as clearly visible in Figure 2.6. However, once exceeded the value of 0.78 seconds, the improvements stop: the true positives start decreasing and the false negatives increase, causing in general a lower F-score.

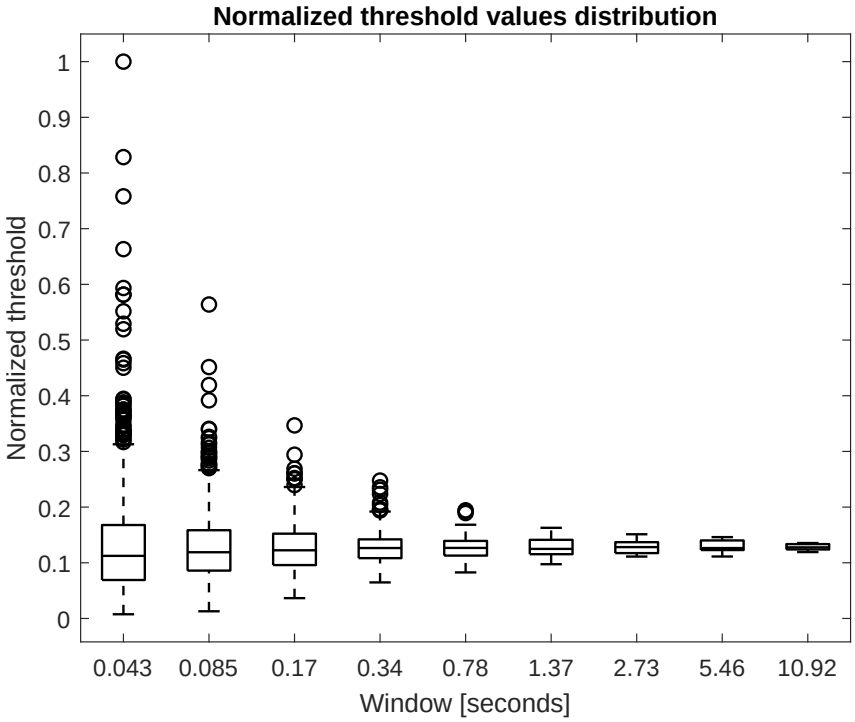


Figure 2.6: Threshold values distribution at the varying size of the threshold time window for RMS + NEO spike detection method

2.3 Spike detection methods complexity

In this section is presented the computational complexity and the memory requirements of the filtering, spike emphasis, and spike threshold methods.

Filtering

The operations (per sample and per channel) required by the filters are shown in Table 2.9. The 4th-order FIR filter is moderately expensive and could be used when it is not required a steep roll-off. The 60th-order FIR filter is the most expensive filter among the ones considered, requiring an order of magnitude more multiplications, additions, and registers to obtain a transfer function as steep as the IIR filter ones. However, differently from the IIR filters, the FIR filter is intrinsically stable and does not introduce any phase distortion.
The moving average difference filters complexity drop-down, in fact, no multiplications are required. The transfer function of the 8th-order MAD filter is similar to the one of the Butterworth filter. Moreover, not having any pole in their transfer function, the MAD filters are intrinsically stable. The only drawback constituted by using the 8th-order MAD filter is its phase response, which not being linear, introduces phase distortion.
The difference filters are the ones with the lowest requirements in terms of both computational complexity and memory. They are intrinsically stable as well and their phase response is linear within the band of interest. Difference filters and Moving Average filters are the best candidates to be embedded in neural interfaces addressing thousands of recording sites.

Filter	Order	Mul	Sum	Reg
FIR	4	5	4	4
FIR	60	61	60	60
IIR	4	10	9	8
MAD	2	0	2	2
MAD	8	0	8	8
DIFF	1	0	1	1
DIFF	2	0	1	2

Table 2.9: Filters computational cost and memory usage

Emphasis	Mul	Sum	Reg
ABS	0	1	0
NEO	2	1	$\geq$2
ASO	1	1	$\geq$1

Table 2.10: Spike emphasis methods' computational cost and memory usage

Threshold	Mul	Sum
MS	1	1
Var	1	3
Mean	0	1

Table 2.11: Spike threshold methods' computational cost

Spike emphasis

The operations (per sample and per channel) required by the spike emphasis methods are shown in Table 2.10. The absolute value is the simpler method, it only requires an addition to substitute the sample with its 2's complement when its value is negative. The operations of NEO and ASO are directly extracted by Equations 2.8 and 2.10. The NEO operator requires two multiplications and one subtraction. The ASO operator requires one multiplication and one addition. Furthermore, NEO and ASO operators also require to memorize the previous samples of the neural signal. When the parameter k of Equations 2.8 and 2.10 is *1*, two registers are needed for NEO and one for ASO. The number of registers increases linearly with *2k* and k respectively. Moreover, note that when the recording has multiple channels, registers are required on each channel.
Because of its simplicity, the absolute value method should be preferred to ASO and NEO algorithms for scaling on high-channel-count neural interfaces, even though the accuracy obtained with ASO and NEO algorithms is slightly higher, as seen in Section 2.2.

Spike threshold

The operations (per sample and per channel) required by the spike threshold evaluation are shown in Table 2.11. These values are directly obtained by analyzing Equations 2.12, 2.15, and 2.17. The multiplications by the constant factor α present in all the equations is not considered, since it only happens once per time window, and not at every incoming sample. According to the hypothesis of using a timing window with a number of samples equal to a power of two, the division by the window dimension W reduces to a right shift, and it is ignored as well. The mean computation requires a single addition per sample. The evaluation of the variance requires computing the mean of the signal, subtracting it from the current sample

value, computing the square of the difference, and adding it to the previous accumulated terms. It is then necessary one multiplication and three additions. The mean square of the signal is computed by evaluating the square of the incoming sample and adding it to the previously computed terms, thus, one multiplication and one addition are therefore required.

The Mean based threshold is the least computationally expensive noise estimation method, and unless the ASO operator is chosen, it is also the method that guarantees the highest accuracy, as shown in Section 2.2. The Variance of the signal is the most computationally intense noise estimation method, moreover, it achieves the poorest accuracy when used in combination with both NEO and the absolute value of the signal. The Mean Square evaluation is a trade-off between the Mean and the Variance from the computational cost point of view, it requires an addition and a multiplication: two additions less than the variance, but one multiplication more than the mean value computation, that is multiplication free. Unless significant accuracy improvements are found, the mean value is the best-suited method for implementing high-channel-count neural interfaces.

2.4 Discussion

The results obtained in the previous sections can be used to guide the choice of the most appropriate spike detection processing chain using as metrics the computational complexity, expressed as numbers of additions and multiplications, the memory requirements, expressed as number of registers, and the accuracy, expressed as F-score, number of true positives, false positives, and false negatives. In this Thesis, the main focus lies on providing suitable solutions for neural interfaces processing in real-time the data sampled by MEA and HDMEA counting thousands of recording channels. For this reason, computational- and memory-efficient methods are preferred to computationally expensive and memory-draining methods when they provide only small accuracy improvements.

The configuration we identified as the best suited for being used with new generation multi-electrode arrays is composed of 2-nd order Moving Average Filter, absolute value based spike emphasis method, and mean value based noise threshold estimation, computed on a fixed (non-sliding) time window of 0.78 seconds. This configuration is multiplication-free and reaches an F-score of 91.64%, only 3 points below the most accurate configuration, i.e. NEO and Mean value threshold, while it guarantees a minimum computational complexity of 2 additions per sample per channel (without considering the filtering stage), whereas NEO and Mean value threshold together requires 2 multiplications and 2 additions per sample per channel (without considering the filtering stage). In addition, the absolute value spike emphasis method does not require any additional memory element, NEO necessitates two memory elements per channel.

2.5 Conclusion

The study highlighted the characteristic of several algorithms used for filtering, spike emphasis, and spike threshold evaluation. In particular, the filter behavior of several filter topologies was analyzed and compared starting from their transfer function, considering both the magnitude and the phase response. Then, the accuracy of the permutations of three spike emphasis techniques and three spike threshold methods were measured using a benchmark dataset [18]. An additional exploration of the size of the time window used during the spike threshold computation was also carried out on the same dataset. Furthermore, the computational complexity and the memory requirements of filtering, spike emphasis, and spike threshold algorithms were extracted from their mathematical formulation.

All in all, it does not seem complex spike emphasis algorithms could improve the spike detection accuracy that much, conversely, they introduce additional computations and buffers to store the previous samples. The requirements are not that elevated per channel, but with the emerging technologies, they get multiplied by numbers in the order of 3k channels [2]. In a similar fashion, also filters and spike threshold complexity should be reduced as much as possible, by preferring lower computational- and memory-intense architectures when practicable, such as moving average difference or difference filters and mean value based estimation method for the threshold evaluation.

The configuration we selected as the best suited to be used in the design of new generation neural interfaces exploiting emerging CMOS-based MEAs and HDMEAs is a multiplication-free neural signal processing algorithm based on 2nd-order Moving Average Difference filter, absolute value spike emphasis method, and mean value based noise estimation. This configuration, without sacrificing the accuracy, having an F-score of 91.76%, only 3 points lower than the best performing solution found, i.e. NEO spike emphasis algorithm and mean value based threshold, allows to accurately detect the spikes with a minimum computational and memory requirements of four additions, zero multiplications, and three registers per channel.

Although in the next two chapters the Thesis focuses on spike sorting[4] and spiking neural network emulation, neglecting spike detection, a hardware implementation of the above-

[4]Spike sorting: recognizing and grouping the spikes generated by the same neuron.

mentioned neural signal algorithm is provided in Chapter 5, as long as its accuracy when used for detecting the spikes inside a more complex neural signal processing chain, aiming to decode the position of a handle by reading the neural activity generated in the motor cortex.

Chapter 3

Real-time spike sorting over thousands of channels

Abstract

Multi-Electrode Arrays and High-Density Multi-Electrode Arrays of sensors are key instruments in neuroscience research. Such devices are evolving to provide ever-increasing temporal and spatial resolution, paving the way to unprecedented results when it comes to understanding the behavior of neuronal networks and interacting with them. However, in some experimental cases, in-place low-latency processing of the sensor data acquired by the arrays is required. This poses the need for high-performance embedded computing platforms capable of processing in real-time the stream of samples produced by the acquisition front-end to extract higher-level information. Previous work has demonstrated that FPGA and All-Programmable System-On-Chip (APSoC) devices are suitable target technology for the implementation of real-time processors of Multi-Electrode Arrays data. In this chapter, we propose an APSoC-based implementation capable of sorting neural spikes acquired by the sensors. Our system, implemented on a Xilinx Z7020 APSoC is capable of executing online spike sorting on up to 5,500 acquisition channels, 43x more than state-of-the-art alternatives, supporting 18KHz acquisition frequency. We present an experimental study on a commonly used reference dataset, using an online refinement of the sorting clusters to improve accuracy up to 82%, with only 4% degradation with respect to offline analysis.

3.1 Introduction

During the past decades, understanding neural signals and interaction between neural units have been a topic of interest in the medical and biomedical scientific community. Lots of research efforts have been dedicated to advancing the knowledge in the field, mainly aimed at long-term important objectives, such as the comprehension of neural networks functional principles [25] and the implementation of neural prosthetic systems [26].

To foster studies on the behavior of neural units, researchers have developed a wide range of hardware and software instruments. Among these solutions, in the hardware domain, multi-electrode arrays (MEAs) [2] have been largely used for long-term multi-units recording. Multielectrode probes have been also proposed, well suited to monitor neurons in both superficial and deep brain structures [27]. Finally, High-Density MEAs (HDMEAs) permit retrieving information at the single cell level [3]. The increasing number of channels, growing from tens to thousands, drastically improves spatio-temporal resolution and the yields of the analysis and processing of the sampled activity. To be effectively exploitable, such evolution of the sensing hardware must be supported by the design of adequate processing platforms executing the analysis of the sensed signals. The large amount of collected data requires high throughput to comply with real-time constraints and to avoid data loss, especially when analysis must include *Spike Sorting* [28], i.e. extraction of high-level features, aimed at distinguishing the activity of different firing neurons recorded on the same track. Moreover, latency must be controlled, to support interaction with neural tissues in a closed-loop fashion.

To comply with such tight requirements, mainstream general-purpose processing systems (PCs and workstations), in this case, are hardly good target platforms, due to the low latency response required by the system dynamics, typically in the order of some milliseconds. Instead, ASIC- and FPGA-based embedded systems implementations are usually preferred. However, at the state-of-the-art, such devices only support a limited number of electrodes [29][30][31] or are designed to execute only the first steps of the neural signal processing chain [17][32][33], thus, do not match the requirements of the emerging technologies.

FPGAs are prospectively very well suited for parallel and highly DSP-intensive signal processing. New generation MEAs signal analysis requires operating in parallel on signals acquired

by a high number of channels, each one requiring a high number of multiply-and-accumulate operations, especially needed for removing noise, and other multiplications and arithmetic operations implementing the analysis of the main waveform features. Thus, this processing well matches the high number of DSP slices and BRAM tiles of modern programmable devices. Moreover, the flexibility provided by FPGA technology, permitting the hardware architecture to be reconfigured, is a key advantage in this kind of domain, where research efforts are often in an exploratory phase, requiring algorithms and methods to be refined easily during experiments.

To bring flexibility one step forward, we use All-Programmable SoCs (APSoCs), which allow (part of) the system functionality to be defined and refined in software, enabling tuning by researchers and users without hardware design and implementation expertise. A brief description of APSoC platforms is available in Appendix A.

In this chapter, we extended an existing neural signal processing system [32] named Zyon (Zynq-based On-line Neural processor), implemented on an All-Programmable System-On-Chip, that hosts on the same chip a dual-core ARM-based Processing Systems (PS) and a fabric of FPGA-based reconfigurable logic. In Zyon, the PS was used to close the loop and apply stimuli to the tissue, whereas the circuitry implemented on the FPGA executed the most computationally demanding portions of the processing operating in parallel on the streams of samples acquired by the different channels, implementing spike detection. In this chapter, we extend Zyon implementing support for spike sorting, i.e. recognize the spikes fired by the same neurons. We implement additional digital modules on the programmable logic, to speed up the most compute-intensive processing tasks within a typical spike sorting pipeline, such as extraction of signal features and feature-to-template comparison for classification. The results of such processing are made available to the PS, allowing the exploitation of common techniques used in machine learning, such as, for example, K-Means [34] and Self Organizing Map (SOM) [35]. In this way, the high-level intelligence implementing the sorting can be programmed in software and easily replaced or repeated over the same or different experiments, further improving the system's flexibility and adaptability to multiple analysis cases. Whereas in the previous chapter, we focused on spike detection algorithms addressing MEA and HDMEA counting thousands of electrodes, in this chapter, the analysis regards FPGA-

based solutions for spike sorting algorithms, and in particular, we focus on studying feature extraction and feature classification method, since the spike detection processing chain was already implemented and validated [32].
The main findings of the implementation of Zyon extension carried out in this chapter can be summarized as follows:

- we demonstrate the feasibility of an FPGA-based implementation of compute-intensive tasks within spike sorting, operating in real-time for high channel counts;
- we demonstrate the feasibility of a hybrid hardware-software approach that concurrently exploits Programmable Logic (PL) and Processing System (PS) inside the APSoC;
- We propose an example of cooperative use of PS and PL which periodically refines the identification of reference spike templates using different spike subsets, to reduce the impact of an unfavorable subset selection on the overall spike sorting accuracy;
- we validate our system architecture capabilities on a set of widely used reference benchmarks [18] and explore its parameters to validate and justify our design choices.

The remainder of this chapter is organized as follows: Section 3.2 contains an overview of existing online spike sorters and online spike sorting algorithms; Section 3.3 presents the target processing tasks and the overall structure of the sorting pipeline; Section 3.4 describes the processing system architecture and the involved functional blocks; Section 3.5 discusses the achieved results, presenting experiments to assess accuracy and performance; Section 3.6 is dedicated to a comparison with alternatives available in the literature; conclusions are reported in Section 3.7.

3.2 Related work

The landscape of different implementations and algorithms proposed in the last years to interact with the neural tissue and to sort neural data is multifaceted. Approaches available in the literature have a wide scope of objectives: the purpose may be to interact with the tissue [33], or to partially process the data to limit memory [36] and bandwidth requirements [37].

Some instruments are designed to operate offline, such as [38], which reaches outstanding performance on variable numbers of neurons, provides a graphic user interface, and could use a variable number of CPUs and GPUs to speed up the analysis. Other works, more related to the presented work, are focused on online analysis [39].

Moreover, spike sorting systems and algorithms have been implemented using a wide variety of target technology: researchers have developed software implementations executed on PC/-workstations [18], as well as custom hardware devices implemented on FPGA [32] or ASIC [40]. Custom implementations typically present lower latency than PC/workstation solutions and are preferred for real-time applications. However, software tools for real-time analysis exist, such as the case of pyNeurode [41], a Python-based platform capable of sorting 128 electrodes with a latency of around 160 ms, or 256 electrodes with a latency of around 255 ms.

Other works target different sorting strategies and focus on different steps of the sorting procedure. For instance, some works only implement online spike detection. The objective could be reducing memory requirements [17], or bandwidth requirements [37]. In some case spike detection is simply enough to enable real-time interaction with the neural tissue, without the need of adding additional latency introducing a spike sorting analysis [33].

Some work implements complete real-time spike sorters, integrating steps such as filtering, spike detection, extraction of features relevant for identifying the firing neuron among the ones surrounding the electrode tip, and classification or clustering. For instance, implementations such as [42] and [43] face the problem of online spike sorting, using significantly different sorting strategies. In [42], authors rely on feature extraction using the Zero Crossing Feature (ZCF) method [44], consisting in taking two different areas extracted from the spike waveform as features for classification. Subsequently, ZCF features are processed using a Moving Centroid K-Means (MCKM), an online clustering algorithm based on the K-Means (KM) algorithm. On the other hand, in [43] was directly used the raw spike waveforms for clustering. A similar approach was also used in [39], where authors directly clustered raw spike data employing a set of carefully chosen thresholds, to create and update clusters.

Selfsort [45] in contrast, despite keeping a structure similar to [42], firstly used a Self Organizing Map offline to get the cluster centers, then, took advantage of the approximated cluster centers, computed on the first set of incoming spikes, to simplify the system by implementing

in hardware a classifier instead of a clustering algorithm.

The Hierarchical Adaptive Means (HAM) algorithm [46] is another example of an online clustering approach, where the clusters are dynamically added, updated, or merged. In [46] a different feature extractor method called First and Second Derivative Extrema (FSDE) [47] has been exploited. FSDE estimates the derivative extrema and uses them as features for classification.

Multiple research efforts have also proposed in detail the hardware implementation of spike sorting systems. An example of ASIC based spike sorter is [40]: a 128-channel spike sorting chip designed for low-power applications. Initially, the spikes are emphasized using a linear algorithm that requires only sums and shifts, then, after detection, another linear transformation is applied to the isolated spike waveforms to extract valuable features for sorting the spikes, using an improved K-Means algorithm.

In [29] an Altera Cyclone III FPGA is used to prototype a spike sorting system. Post-synthesis results are also given. The neural signal processor embeds a binary decision tree (BDT) classifier based on a collection of two bits discrete wavelet transform (DWT) features, and it operates on 32 independent channels. The method provides a 50% memory reduction compared to distance-based methods.

Deep learning based spike sorters exist, such as [31], where a Convolutional Neural Network (CNN) was implemented on FPGA and used to sort the spike incoming from a 49-channel non-synthetic dataset. In [31] was also demonstrated how considering the geometrical properties of the multi-electrode array could boost the sorting accuracy. They added this feature to the Osort algorithm, in a similar fashion of [48], and incremented the sorting accuracy from 44% to 86% using a quantized 3.5 KB CNN. Most of the spike sorting systems and algorithms consider a maximum number of neurons present around the electrode and then a maximum number of clusters or templates to be matched. [40] considers six as the maximum number of clusters. HAM [46] and Selfsort [45] set a limit on the maximum number of clusters per channel referring to [49], where it was demonstrated that with the current technologies and algorithms it is possible to correctly identify up to eight to ten neurons per electrode.

Table 3.1 is a collection of real-time spike sorters and spike detectors we found in the literature related to this work. All the works address a small number of electrodes compared to the

STATE OF THE ART COMPARISON TABLE

Reference	Yang [29]	Saeed [42]	POSort [30]	SelfSort[45]	HAM [46]	Do [40]	Park [39]	Müller[33]	Zhang [17]	pyNeurode [41]	Yi [31]	This work
Year	2017	2013	2019	2017	2014	2018	2017	2013	2022	2022	2022	2020
Hardware	FPGA	PC	FPGA	PC	PC	ASIC	FPGA	FPGA	FPGA	PC	FPGA	FPGA
Channels	32	-	128	-	-	128	128	126	128	256	49	4096
Sampling [KHz]	20	-	24	-	-	25	32.5	20	7	-	30	18
Detection	TEO*	TEO	-	TEO	-	Filter	yes	yes	ASO	-	-	yes
Feature	DWT	ZCF	-	ZCF	FSDE	Filter	-	-	-		-	FSDE
Class/Clust	BDT	KM-Class	OSort	SOM-Class	HAM	KM-Class	Template	-	-	TM	CNN	K-Means-based
Resolution	-	10	16	10	-	9	16	8	10	-	4-8	12
Accuracy	(60%:80%)	82%	87%	93.4%	>90%	[72% : 86%]	-	-	95%	˜90%	86%	82%
SNR [dB]	[5:7]	7	[10:13]	[0:15]	[7:13]	[4.6:13]	-	-	[7:13]	-	-	[7:13]

* Modified threshold

Table 3.1: Related work summary

proposed system. This entails a very local monitoring of the neural tissue, or conversely a low resolution. Summarizing, our system:

- is the first closed-loop system that exploits the heterogeneous processing architecture of modern All-Programmable SoCs, fully embedding a spike sorting chain;
- increases by more than one order of magnitude the number of parallel recording channels processed in real-time while guaranteeing a closed-loop latency lower than 2.5 ms;
- takes advantage of APSoCs to guarantee a higher level of flexibility in the neural processing domain. We partition the spike sorting chain deployment between the PS and the PL. Hardware reconfigurability can be used at design time to change the parameters of the spike sorting sub-tasks operating on input samples. We combine it with software programmability, usable more easily during an experiment, to change higher-level sub-tasks operating on spike clusters and spike templates.

3.3 Target spike sorting pipeline

Spike sorting (SS) is a key step for the analysis of neural signals. It consists on the separation of the superimposed activities of neuronal cells sensed from the same electrode. At the end of the process, spikes generated from the same neuron are grouped together. The majority of spike sorting algorithms are constituted of a four steps processing chain [28] shown in Figure 3.1:

- Filtering - First, the acquired raw signal is filtered to remove noise as much as possible.
- Spike Detection - Spikes are usually detected by means of amplitude thresholding methods: the samples are compared with a threshold one after the other.
- Feature extraction - Once a spike is identified, its shape is considered for further analysis. Some of the main factors that determine the spike waveform are the position relative to the electrode and the neuron geometry [50]. Therefore, spikes coming from the same neuron will be morphologically similar. At this stage, valuable features are measured on the waveform, as an indication of the pertinence to a specific active neuron.

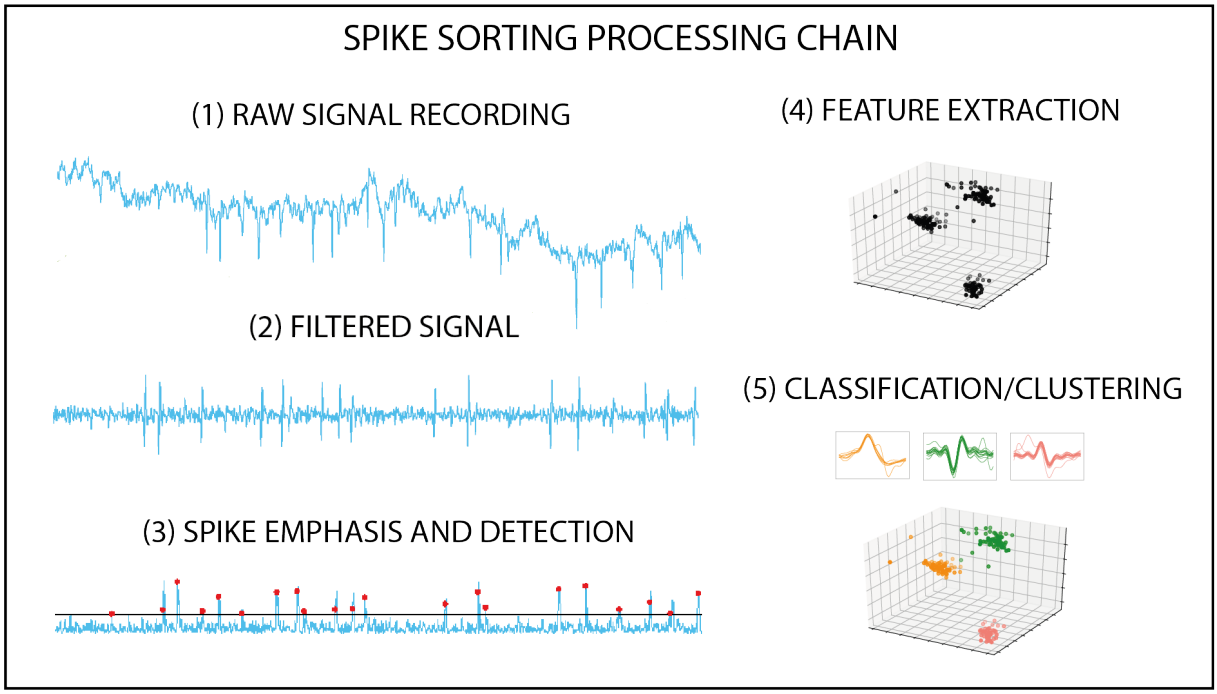


Figure 3.1: Spike sorting processing chain, the raw signal (1) is filtered (2), then, spikes are emphasized and detected (3). Valuable features are extracted from the spikes waveforms (4) and classified/clustered (5).

- Clustering - Feature values in detected spikes are considered to partition the feature space in clusters, that correspond to different spike shapes and, consequently, to different firing neurons. Clustering associates to each spike an ID, producing a sorted activity track in output for further analysis. When facing online spike sorting, the cluster definition cannot rely on the whole set of spikes involved in the experiment. Two main different kinds of approaches can be used. The first method runs a data-stream clustering algorithm, as in [46] and in [30]. The second possibility is, otherwise, to approximate the cluster centers considering a reduced recording time during the experiment, and consequently a limited number of spikes, and then use such centers to classify the incoming spikes during the remaining experiment duration, as in [45]. Thus, in this case, the final processing stage in Figure 3.1 can be considered as composed of two phases:

  - a proper *clustering*, which may take place on a *training* subset of spikes, e.g. at the startup, and identifies the clusters/templates to be considered during the rest of the experiment;

– a *classification* procedure, which evaluates online the similarity of incoming spikes to the templates identified by the clustering, completing the spike sorting process.

3.4 System architecture

The proposed processing system architecture is shown in Figure 3.2. The architecture is designed to exploit the characteristics shared by APSoC devices that are part of the Xilinx Zynq-7000 family. The architectural template can be configured at design time and parameterized to fit different devices of the family. However, the system configuration presented in this chapter is implemented on a Z-7020 device.

The system presented in this chapter is based on a previously presented platform, named Zyon

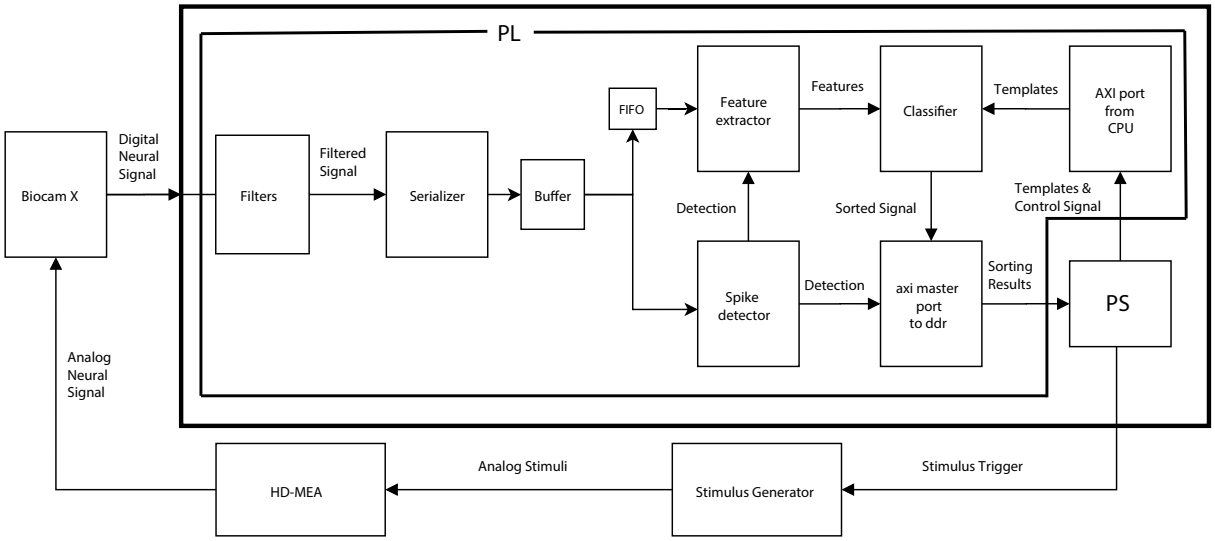


Figure 3.2: Schematic block diagram of the experimental setup. Biocam X provides Zyon with digital data of 4096 electrodes mounted on the HDMEA. Zyon processes the data in real-time, sorting the neural signals coming from 4096 channels, and generates the output stimuli through the Stimulus Generator.

[32]. Zyon, in its previous implementation, shares the principles of the architecture presented in this chapter. Both the PL and PS were used in cooperation. The PL was populated with modules, described in HDL, that implemented the front-end tasks of the neural signal processing chain, until the spike detection phase. The PS was programmed in C and, analyzing the detection results, evaluates higher-level metrics such as firing rate and spike locations, to take array-level decisions in real-time. As an example, in [51] the spike redundancy among the

channels of a High-Density Multi-Electrode Array (HD-MEA) was used to reduce the number of active electrodes and lower the computational burden of the signal analysis, in the case of retinal circuits. In this chapter are added further steps in the chain, implementing feature extraction and clustering.

As presented in [32], the system is instrumented to be interfaced with a BioCam X platform by 3Brain AG. The platform embeds an active CMOS-MEA device, capable of acquiring 4096 neural signals, sampled at a maximum frequency of 18KHz, digitalized, and transmitted to the external environment through a Camera Link interface. The interfacing logic implemented on the FPGA is modular and easily replaceable to interface with other HDMEA platforms, nevertheless, the BioCam X has been used as a reference to design the performance of our system, in terms of sampling frequency and channel count.

Zyon already embedded a *filtering* stage that we kept untouched, where the digital neural signals were multiplexed in time to be processed by a bank of 32 digital FIR filters of order 63, with cut-off frequency of 300 and 3400 Hz, implemented by the use of Vivado FIR Compiler. Since every filter completes the computation after 40 clock cycles, the overall filter bank throughput is equal to 0.8 (32/40) samples/cycles. The filtered signals were then serialized and processed by the downstream modules, which, exploiting an efficient hardware-level pipelining, implemented in the RTL description of each module, reached a throughput of 1 sample/cycle.

In this implementation, the *Spike Detector* reads the filtered samples from the *Serializer* and triggers the *Feature Extractor* when a spike is detected.

The *Feature Extractor* reads the samples from a BRAM-based FIFO, whose main functionality is buffering an adequate number of samples, serving as a short pre-threshold history of the sample stream.

Once the features are computed, the *Classifier* evaluates the distance between the feature vector and a set of pre-stored templates. It classifies the spikes by identifying the template producing the minimum distance.

All the mentioned modules, which take care of the data-crunching tasks in the pipeline, are implemented on the programmable logic. Furthermore, two dedicated AXI High-Performance ports were used to allow communication between such modules and the processing system. In this way, the processing system is available to access the results of the different processing

stages. When focusing on spike sorting, its main function is related to *clustering*: the PS can be used to receive feature vectors from the programmable logic, identify the cluster centers, and update the *Classifier* templates.
In addition, the *PS* takes care of:

- Implementing closed-loop interaction tasks;
- Refining the templates, if needed, considering the classification results;
- Housekeeping tasks, such as memory management, network communication, input/output, and interaction with the user;

Exploiting the peculiar characteristics of APSoCs for such purpose, in our approach, allows drastically increased flexibility, allowing for easier tuning/refinement of the clustering algorithm, *Classifier* templates, and stimulus patterns provided, based only on software tuning. Hardware changes are required only when lower-level algorithm parameters, such as detection method and classification metrics, have to be replaced.
In Figure 3.3 we use *Wavedrom*, an open-source digital timing diagram rendering engine, to show a waveform timeline representing the flow of data through the modules implementing the sorting pipeline.

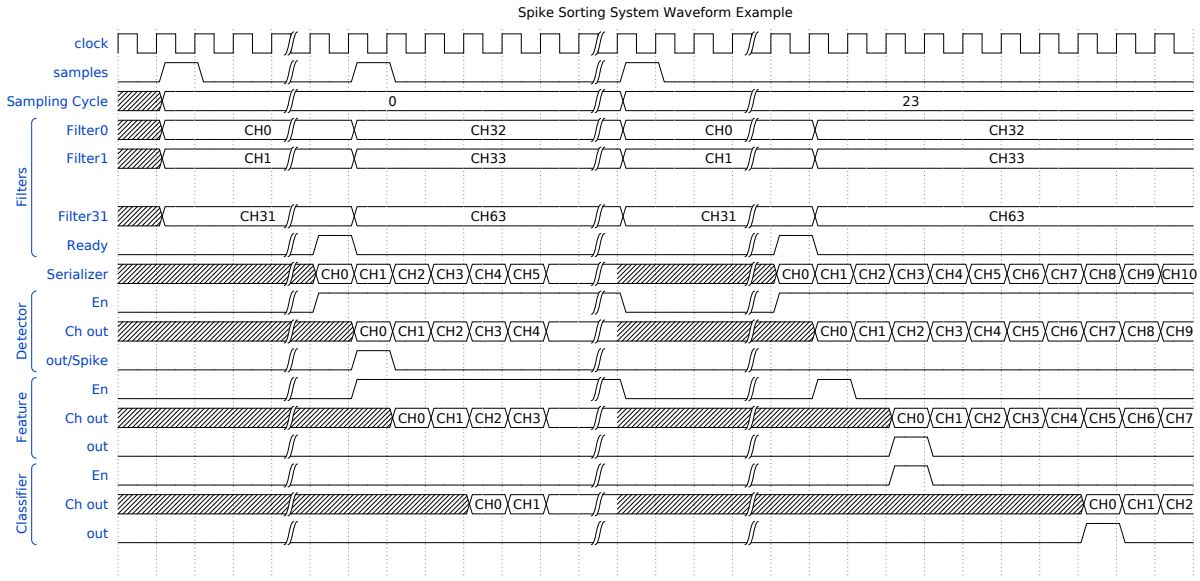


Figure 3.3: The channels are time-multiplexed and processed 32 at a time by the FIR filter bank. The filter bank has a latency of 40 clock cycles. When the firsts 32 samples are ready, they are further serialized and analyzed one by one in a time-multiplexed fashion by the *Spike Detector* module, that looks for samples above a certain threshold. As soon as the following 32 samples are computed by the filter bank, they are processed as well, up to the last set. When a spike is identified, i.e. when a sample is above the threshold, that in the example happens at the first sampling cycle for channel zero, the *Spike Detector* triggers the *Feature Extractor*, which collects the spike samples during the following 23 sampling cycles and computes the feature vector. When the feature vector is ready, the *Classifier* is enabled and the spike vector is classified.

3.4.1 Filter

The FIR filter bank, inherited from [32], is composed of 32 independent FIR filters of order 63, with cut-off frequency of 300 Hz and 3400 Hz. Every filter computes 32 MAC operations and serves 128 different channels in a time-multiplexing fashion. Every filter is implemented using a single DSP. A single DSP is sufficient to satisfy the requirement of 32 MAC operations per 128 channels as long as the system frequency is at least 32×128 = 4096 times bigger than the sampling frequency, being the sampling frequency set at 18 kHz, the minimum clock frequency for this system is about 74 MHz. The filter latency is determined from the FIR order N, and the sampling frequency f_s, through the formula: $N/(2 \cdot f_s)$. Being N equal to 63, and f_s equal to 18 kHz, the latency introduced by the filter bank is then 1.75 ms. The filter bank receives the neural data from an AXI interface, its output is connected to the *Serializer*.

3.4.2 Serializer

The *Serializer*, inherited from [32], employing a set of multiplexers, writes the incoming filtered samples in a BRAMs buffer, interfacing the filter bank with the spike detector.

3.4.3 Spike Detector

The *Spike Detector*, inherited from [32], reads the filtered samples from the *Serializer* and compares them with a threshold, evaluated according to Equation 3.1.

$$Thr^2 = \frac{\alpha^2}{M}\left[\sum_{k=0}^{M-1} x_{i-k}^2 - \frac{1}{M}\left(\sum_{k=0}^{M-1} x_{i-k}\right)^2\right] \tag{3.1}$$

Where *x* are the samples, α is a parameter and M is the size of the sliding window, which is limited to powers of two, to be easily managed in hardware. The spikes are detected when the following condition is satisfied:

$$Spike = x_i^2 > Thr^2 \tag{3.2}$$

3.4.4 Feature Extractor

The *Feature Extractor* implements the First and Second Derivative Extrema (FSDE) feature extraction algorithm [47]. FSDE based spike sorters use the minimum and maximum extrema of both first and second derivatives as features. However, usually not all the extrema are considered. By relying on the evidence proved in [47], the maximum of the first derivative and both the maximum and the minimum of the second derivative used together permit achieving the best results.

The first and second derivatives of the spike waveform are evaluated as the difference of adjacent samples and as the difference of adjacent first derivative values:

$$FD(i) = x(i) - x(i-1) \tag{3.3}$$

$$SD(i) = FD(i) - FD(i-1) \tag{3.4}$$

Where *FD* and *SD* are respectively, the first and second derivatives, and $x(i)$ is the i^{th} sample of the spike window. The features are computed within a window placed before the detection event. In order to retrieve the previous samples, a FIFO is placed between the *Serializer* and the *Feature Extractor*. The FIFO size is defined by Equation 3.5.

$$FIFO_{size} = D \times C \times S \tag{3.5}$$

Where D is the required number of samples, prior to the threshold trespassing, to be considered as the head of the spike waveform. D also determines a certain delay between a sample entering the *Feature Extractor* and its actual contribution to the feature evaluation. Whereas, C is the number of channels, and S is the dimension in bits of the recorded samples. The FIFO size grows linearly with the required delay D, and, its size has an impact on the accuracy. Limiting D removes too much information contained in the early sample of a spike, affecting the spike characterization and the overall clustering results. Therefore, this parameter needs to be carefully evaluated.

As soon as the *Spike Detector* triggers the *Feature Extractor*, it starts computing the features

on the delayed stream of samples coming from the FIFO. Figure 3.4 shows the *Feature Extractor* architecture. The internal buffers required to store the samples and the derivatives are implemented in BRAM, in fact, in our case study, acquiring from thousands of channels simultaneously, BRAM tiles are best suited than distributed LUT-ram memories. The *Feature Extractor* is composed of two main blocks: *Delta* and *Extrema*. *Delta* computes the derivatives, whereas *Extrema* computes the derivative extrema. *Delta* computes the First Derivative (FD) employing a subtractor and a BRAM buffer in which the samples of the previous sampling cycle are stored. The Second Derivative (SD) is computed in the same way. The FDs are stored inside a buffer and the SDs are evaluated using a second subtractor.

After initialization, starting from the second sampling cycle, the new *FD* and *SD* are compared with the contents of the buffers, as shown in Figure 3.4. When a new extrema is found, its old value is updated in the buffer. Since the algorithm runs for W sampling cycles and is executed independently on every channel, one counter per channel is also needed. The counters are stored in BRAM as well.

The data sampled by BioCam X are quantized using 12 bits. FSDE features can also be expressed using 12 bits since the subtraction of adjacent samples should not cause overflow conditions due to the limited distance between successive samples. This hypothesis was validated on the test dataset [18].

The *Feature Extractor* notifies the *Classifier* when a new feature vector is ready.

3.4.5 Classifier

Real-time sorting algorithms rely on an online classification process, that, based on a similarity metric, associates the incoming spike to a class. A wide variety of approaches, e.g. the strategy proposed by Selfsort [45], which identifies candidate clusters evaluating spikes in the firsts seconds of recording or other alternatives based on data stream clustering, such as Hierarchical Adaptive Means [46], share a common computational core: a classifier. Therefore, we decided to implement a classifier inside the programmable logic, in charge of computing the euclidean distance between points in the feature space, to compare each spike with a set of templates, representing the centers of the clusters. Centers may be updated and stored in the system by the *PS*, through one of the two AXI interfaces in use. Several metric distances are

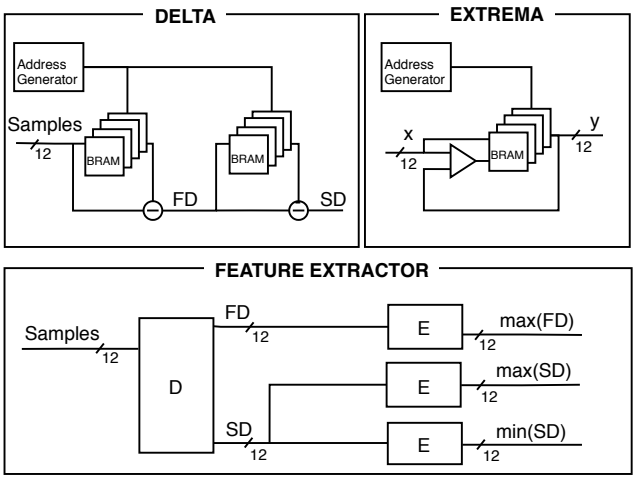


Figure 3.4: The *Feature Extractor* evaluates the derivative extrema of the spike waveform, the maximum of the first derivative FD_{max} and both the extrema of the second derivative SD_{max} and SD_{min}. The *Delta* block computes the derivatives. The *Extrema* block evaluates the derivative extrema.

possible, however, euclidean distance is a common choice that may serve different alternative clustering algorithms, such as, for example, K-Means [34], whose results are evaluated in more detail in the following, and SOM [35], which is also tested as an alternative to highlight the flexibility of the software programmability offered by the PS. Nevertheless, given the system modularity and the FPGA reconfigurability, replacing the *Classifier* with a different module computing a different metric is a straightforward process that does not require any modification to the system architecture.

The square of the euclidean distance is used in place of the euclidean distance to avoid computing a square root, and it is evaluated according to Eq.3.6. Where D_i is the square of the euclidean distance with the *i-th* template, T_i is the *i-th* template, and $i \in [1, K]$. With K the number of templates per channel.

$$\begin{aligned} D_i = (FD_{max} - T_{i_1})^2 + \\ + (SD_{max} - T_{i_2})^2 + \\ + (SD_{min} - T_{i_3})^2 \end{aligned} \tag{3.6}$$

The distances D_i are then compared to select the closest class:

$$neuron\ id = argmin(D_i) \tag{3.7}$$

The *Classifier* is triggered by the *Feature Extractor* at every incoming spike and provided with a new feature vector. As regards the templates, they are stored in a BRAM-based buffer by the PS, through one of the two AXI interfaces. The number of templates sets a limit to the maximum number of distinguishable neurons around the electrode and it is set to 8, according to the evidence given in [49].

Every classification entails three differences, three multiplications, and a final sum of the three products, implemented by the module *Distance*, whose architecture is shown in Figure 3.5. Subtractions and multiplications are embedded into DSP blocks, taking advantage of

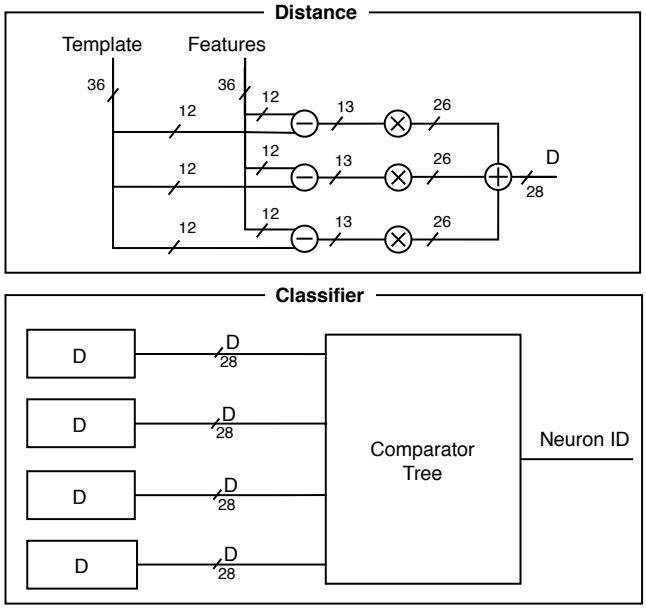


Figure 3.5: The *Classifier* schematic block, shown with 4 templates instead of 8 for display purposes. Each *Distance* block computes the Euclidean distance between the feature vector and a template. The *Comparator Tree* finds the smallest Euclidean distance identifying the firing neuron.

the DSP48E1 *pre-adder* and *multiplier*, whereas the three addends are added by means of a LUT-based *carry-save adder*. In fact, even though the final adder of a DSP block is a three terms adder, unfortunately, two addends are required to carry out the multiplication, so the

LUT-based adders are required.

The eight euclidean distances are computed concurrently by eight *Distance* blocks. Therefore, a total of 24 DSPs and *8 carry-save adders* are in use. Figure 3.5 shows the *Classifier* architecture for four templates. Subtractions and multiplications take advantage of the registers present inside the DSPs to pipeline the computation, whereas, other registers are added to guarantee low latency three-terms additions. Finally, the distances are compared to select the winning class. The comparison is implemented through a pipelined tree of comparators. The size of the three is directly related to the number of templates, that being eight, entails instantiating seven comparators.

To avoid numerical errors, during the whole classification process the number of bits was let grow. In particular, feature-to-template differences require an extra bit to avoid overflow, i.e. 13-bits, the square computation requires doubling the register size to 26-bits, and the three-addends final sum needs two extra bits, the euclidean distances are therefore represented in 28-bits. Although using wider data representation has an impact in terms of area, in both the Distance block and in the Comparator-Tree, the method guarantees the same accuracy of the floating-point representation, as shown in Section 3.5.4. Once classified, the *Communication* block is triggered to transmit the results.

3.4.6 PS-PL Communication

Communication between the PS and the PL takes place through two independent AXI ports. One is used to send bursts of processed data from the PL to the DDR memory, reachable through the PS interconnect. We have reserved for this stream a region in the DDR customizing the operating system configuration. The second AXI interface is used to set up the system by storing initialization data and to update the *Classifier* templates.

The system can be set in different communication-related operating modes, which may happen alternatively depending on the needs of the experiment. The first AXI interface can be set to transmit alternatively feature vectors and sorting results, sorting results only, or spike detection results. Furthermore, the second AXI interface can update the *Classifier* templates and the sorting parameters, by programming memory-mapped registers. In more detail:

- Transmission of feature vectors and sorted spikes: The programmable logic transmits the features of the detected spikes as two burst AXI transactions, the former contains the feature vectors, the latter the channel IDs to which the spikes belong. In the worst-case scenario, where a spike is present in every channel, the burst transmits a packet of 40KB (4096 channels x 64-bits + 4096 channels x 16-bits). Since this packet should be sent within the sampling cycle, and the maximum sampling frequency allowed by Bio CAM X is 18 kHz, the highest DDR bandwidth required for this stream is about 700 MB/s (40 KB x 18 kHz), which is below the maximum writing rate allowed between the PL and the PS [52]. A region of 40KB of the DDR should be reserved for this kind of transmission. Despite the high punctual transmission rate required, the physiology of the neurons is characterized by a refractory period corresponding to around 24 sampling cycles after each spike. The spike detection mechanism is designed consequently: when a spike is detected, the coming 24 sampling cycles are used to collect the samples composing the tail of the spike waveform. During this period, the detection module is paused and will not request new transmissions to DDR memory. Therefore, the bandwidth requirement cannot reach the worst-case peak of 700 MB/s in physiologically realistic experiments. This transmission mode is used during *Clustering*, where the PS scans the DDR region reserved for the previously described packets and fills a data structure collecting training spike feature vectors. Subsequently, it runs the clustering algorithm and updates the templates if needed. The PS data scan time takes less than 900 μs.

- Transmission of sorting results: The information related to each channel is encoded in 4 bits. The first bit declares the presence or absence of a spike in the channel, whereas the remaining three bits indicate the spike's class. A 2 KB packet (4 bits x 4096 channels) is written from the PL at each sampling cycle, once every 55.6 μs (18KHz). The worst-case to-DDR bandwidth required for this stream is 35 MB/s (18 KB x 4096 channels). A region of 4 KB is reserved in the DDR for this purpose, which is used as a double buffer. Buffer locations are used alternately to avoid overwriting. The PS reads the packets during the following sampling cycle, in about 20 μs, copying it outside the PL-dedicated memory region.

- Transmission of detected spikes: The programmable logic sends a stream of one bit per channel, to give information about the presence or absence of spikes. The bandwidth required for the transmission is about 8.8 MB/s (4096 channels x 1 bit x 18 kHz). This kind of transmission requires reserving a DDR region of 1 KB served as a double buffer (2 buffers x 4096 channels x 1 bit).

- Transmission of new templates: at the start-up, or when an update of the templates is required, the PS can run the clustering algorithm on a collection of features stored in the DDR to generate new templates and update the *Classifier*'s ones. The amount of data necessary to update a *Classifier* is 36 Bytes, which can be sent in 384 ns. It is possible to update the full battery of *Classifiers* by sending 144 KB of data through the dedicated AXI-port in 1.6 ms.

- Transmission of control signals and sorting parameters: The PS can set the transmission mode to DDR, enable/disable the sorting chain, threshold levels, and the DDR baseline address.

3.5 Experimental Results

In this section, we present our experimental results. First, we present a hardware-related evaluation of our implementation. Second, we present our experimental setup, the reference benchmark dataset used and the reference software implementation developed to choose the sorting algorithm and to validate our hardware implementation. Third, we assess the possibility of applying online classification after a template characterization performed on different numbers of *training* spikes, to assess the usability in real-life experiments. Furthermore, online template re-characterization is analyzed and the obtained accuracy is reported. Fourth, we assess our implementation by testing the selected feature set, comparing it with a ZCF [44] scheme, evaluating the impact of the used fixed-point data format, and exploring the trade-off between accuracy and memory requirements in the spike window centering problem.

3.5.1 Hardware report

The target device is the ZedBoard, a low-cost development board for the Xilinx Zynq 7020 All-Programmable SoC. The chip embeds 106400 Flip-Flops (FFs), 53200 Look-Up Tables (LUTs), 140 36Kb BRAMs tiles (RAMB36), and 220 DSP48E1 slices. Each DSP48E1 contains a 25-bits pre-adder, a 25x18 bits multiplier, and a 48-bits accumulator.

The FIR filter block is constituted of 32 FIR filters and every filter is implemented using one DSP48E1 only.

The *Feature Extractor* requires two LUT-based subtractors and three comparators to implement the FSDE algorithm. Furthermore, previous samples and previous derivatives need to be stored along the sampling cycles to carry on the FSDE algorithm. Thus, five BRAM-based buffers are instanced, with an entry of 12 bits per channel, requiring one RAMB36 and one RAMB18 block each. The FSDE algorithm also needs a counter per channel. The counters are stored in BRAM and need $\log_2 W$ bits each, where W is the dimension in samples of the spike window.

The *Classifier* requirements, in terms of FPGA resources, are highly related to process parameters like the number of templates per channel K and the number of features F. Depending on the number of features and templates, the number of operations changes as well as the memory required to store the templates, as shown in Table 3.2. The DSPs, the adders, and

CLASSIFIER RESOURCE REQUIREMENTS

(K, F)	DSP	Adders	Comparators	BRAMs (36, 18) Kb
(3, 2)	6	3	2	(8, 2)
(3, 3)	9	3	2	(12, 3)
(6, 2)	12	6	5	(14, 6)
(6, 3)	18	6	5	(21, 9)
(8, 2)	16	8	7	(22, 0)
(8, 3)	24	8	7	(33, 0)

Table 3.2: Classifier resource requirements for several pair of templates K and features F

the comparators are used to compute the euclidean distances between the feature vector and the templates; the BRAMs are used to store the templates. We select eight as the maximum number of neurons per electrode, however, the architecture is parametric and can be easily extended to support a different number. The limiting factor is the BRAMs, which poses the

limit to the number of neurons per channel to 16. By looking at Table 3.2, 33 BRAM tiles are required for K = 8. For K = 16, the number of BRAMs would increase by 2x, almost saturating (137 out of 140) the availability in the device.

The *post-implementation* results, obtained using *Vivado v2017.4*, are shown in Table 3.3.

Thanks to the hardware-friendly algorithms selected for this implementation, it is possible to

POST-IMPLEMENTATION RESOURCE REQUIREMENTS

Resource	Utilization	Available	%
LUT	28984	53200	54.48
LUTRAM	3753	17400	21.57
FF	26444	106400	24.85
BRAM	104	140	74.29
DSP	61	220	27.73

Table 3.3: Post-implementation resource requirements on Xilinx Zynq-7020 device

satisfy real-time constraints with low utilization of available DSP slices. However, due to the nature of the FSDE algorithm, the samples of the previous sampling cycle and the derivative extrema found up to that moment need to be stored, thus the BRAM utilization is relatively high. In addition, the FIFO storing pre-threshold samples of the spikes inside the *Feature Extractor* module, contributes to increase the BRAMs utilization. Overall, considering the resource requirements of the different modules and the throughput performance of the system, the Xilinx Zynq Z-7020 would be able of hosting up to 5500 channels while still satisfying the real-time constraints.

3.5.2 Experimental setup

Reference Benchmark Dataset

To assess the functionality of the system was used the dataset presented in [18] as a reference benchmark. The dataset is composed of four simulations: *Easy 1*, *Easy 2*, *Difficult 1* and *Difficult 2*, each including the activity of three neurons. Every track is available with different levels of noise: 0.05, 0.01, 0.15, and 0.2. The noise levels are intended to be the standard deviations σ of the neural tracks. The simulations were created starting from real spike waveforms recorded in the neocortex and basal ganglia, whereas the background noise is obtained by adding together random spikes. As explicated by the simulation names, the sorting is more challenging for the *Difficult* simulations and easier for the *Easy* simulations. Figure 3.6 shows the waveform models of the three neurons in the four datasets. The models are built by computing a sample-by-sample average between the spike belonging to the same neuron.

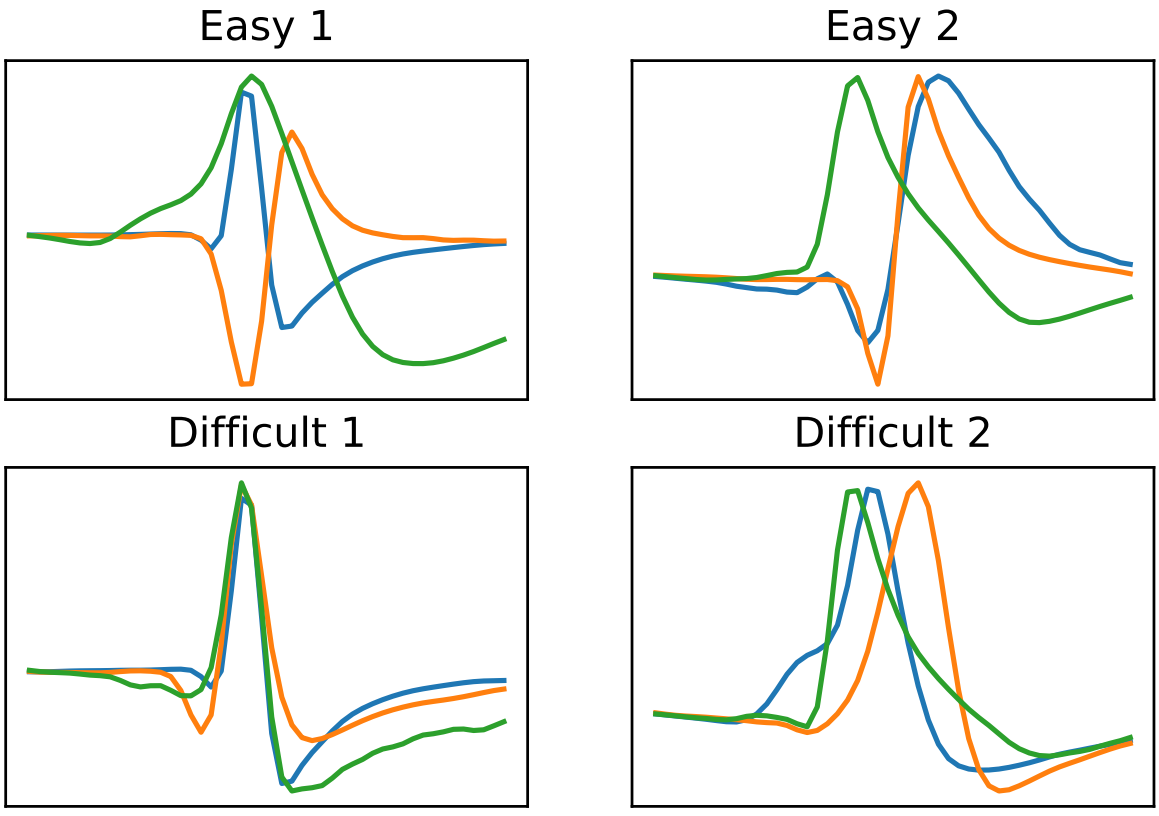


Figure 3.6: Spike waveform models of the four datasets presented in [18].

Experimental Setup

To test the device on the dataset [18] a PC is used to send the data samples through a UART interface operating at 115.200 baud/s. For this purpose, we have implemented a slightly modified design, integrating a Microblaze processor implemented on the PL, managing the streaming of the datasets. Short dataset segments corresponding to a track of 18k samples, converted to 12 bits, are sent to the FPGA, encoding each 12-bit sample in two UART packets. A simple program executed by the Microblaze receives UART packets, recomposes the samples, and stores them into a 64k local BRAM memory. Once the complete segment is received, the processor sends the same stream of samples to all the channels, through two AXI-Stream Broadcaster modules. The resource occupation overhead due to such testing infrastructure is shown in Table 3.4.

EXPERIMENTAL SETUP RESOURCE REQUIREMENTS

Resource	Utilization	Available	%
LUT	2774	53200	0.05
LUTRAM	140	17400	0.01
FF	5122	106400	0.05
BRAM	18	140	0.13
DSP	0	220	0.00

Table 3.4: Resource utilization overhead related with the additional logic used for testing, on the Xilinx Zynq Z-7020 obtained by using Vivado v2017.4.

Reference Software Implementation

To enable preliminary selection of the spike sorting strategy and comparison with available alternatives, before hardware development, we realized a software pipeline embedding the typical spike sorting processing steps [28] in Python, available for download and contribution as open source[1]. Thanks to such software implementation, it was possible to try different strategies of filtering, spike detection, feature extraction, and clustering, on both single and multi-channels data.

The platform embeds Finite Impulse Response (FIR) filters and the offline *Absolute Value Thresh-*

[1] `https://github.com/gianlucaleone/SpikeSorting`

old method proposed in [18]:

$$Thr = \alpha \times median\left(\frac{|x|}{0.6754}\right) \tag{3.8}$$

Where x is the neural signal and α is a parameter set to 4.0 as suggested in [18].
Furthermore, different feature extraction algorithms might be compared, such as Integral Transform, Zero Crossing Feature, and First and Second Derivative Extrema.
The coherence between software and hardware results was thoroughly verified. A comparison between processing results based on floating-point data format and the fixed point implemented on the hardware is presented in the following.

3.5.3 Accuracy evaluation

Several accuracy tests were carried out, comparing the spike-to-cluster association decided by the *Classifier* to the ground truth provided with the datasets. As mentioned, the spikes were detected using Equation 3.8, the features were extracted using the FSDE algorithm, and the online classification was carried out using euclidean based metric.
Figure 3.7 shows the cluster distribution in the feature space, after an offline analysis based on the K-means algorithm and FSDE features. The plots show only two out of the three features, the *first derivative maximum* on the x-axis and the *second derivative minimum* on the y-axis, to improve readability. It may be observed, looking at the clusters of the same dataset at different noise levels, that spikes in the same cluster spread out and the gap between the clusters decreases. It is also possible to observe that the spikes of the dataset *Easy 1* are much more distinguishable with respect to others at every noise level. On the contrary, at higher noise levels, in the other datasets, the clusters overlap.
In Figure 3.7, cluster centers are computed offline using the K-Means algorithm on all the spikes in the datasets. This kind of approach is not usable to implement online sorting, thus, is not suitable for any kind of closed-loop application. Conversely, we have evaluated the overall accuracy for each dataset, using a limited number of *training* spikes in Fig.3.8. The dashed lines represent the accuracy of the offline method. The average offline accuracy obtained is about 86%, ranging from 62% got in *Difficult 2* with a level of noise 0.2 to 95% got in *Easy 1* with a level

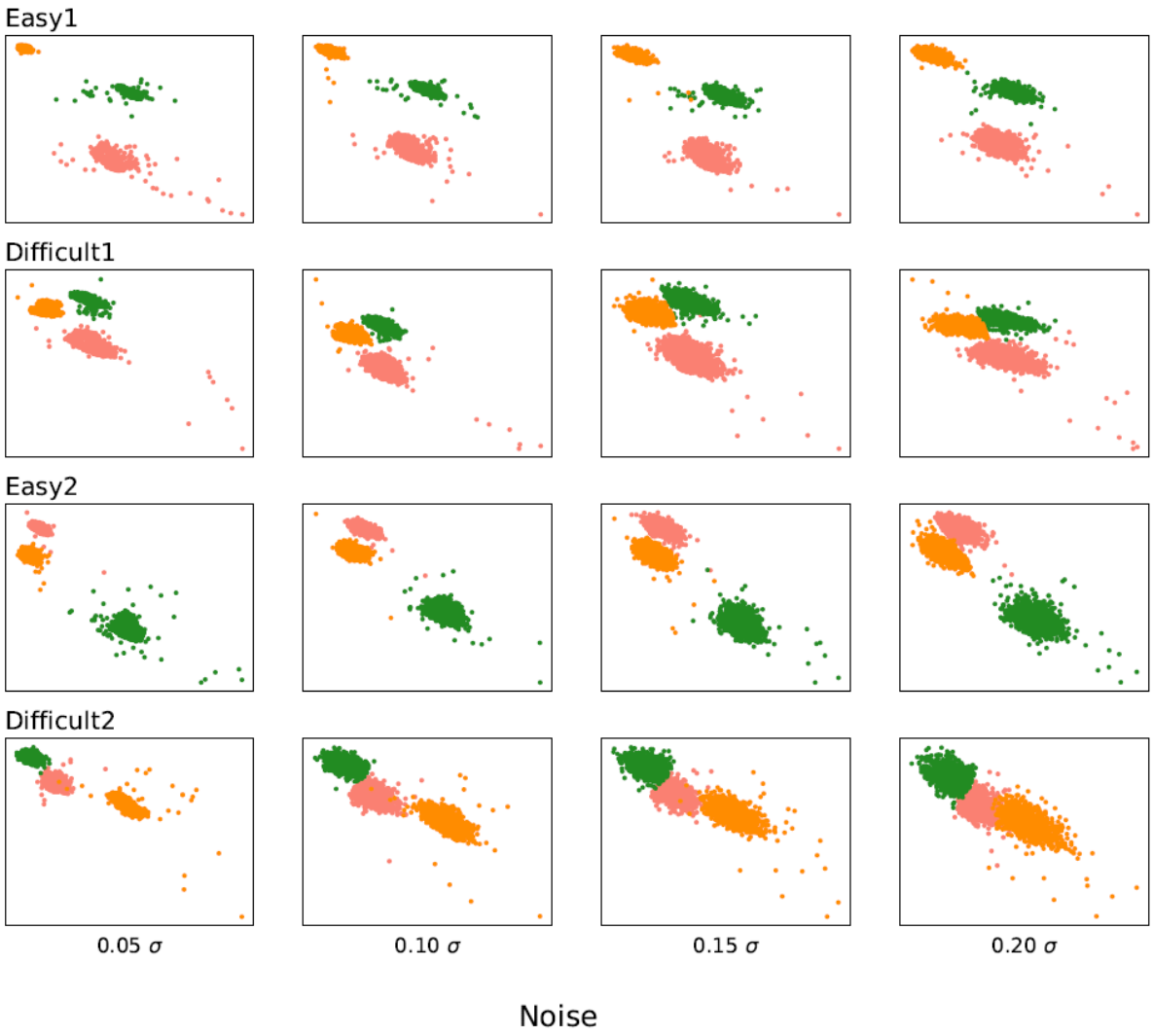


Figure 3.7: Cluster shapes obtained running K-Means on the FSDE features over 16 neural simulations [18]. The noise increases from left to right, the radius of the clusters increases with the noise, and the clusters get closer. The *Difficult* simulations exhibit closer clusters compared to the *Easy* simulations.

of noise 0.05. Every box-and-whisker plot, except the rightmost one, contains the results of 200 experiments where the training spikes were taken randomly from the dataset tracks. We varied the number of spikes used to run the K-Means algorithm along the x-axis, ranging from 100 up to 400. In most datasets, 100 training spikes are often sufficient to reach an accuracy similar to offline analysis, as may be noticed by the median value, which converges to the dashed line. However, in some datasets, e.g. *Easy 2* with a very low or very high level of noise, at least 300 spikes are required to converge to offline accuracy levels, set respectively to 0.94 and 0.73. In general, it is possible to notice a significant accuracy deviation from the median value, corresponding to larger boxes, i.e. depending on the set of feature vectors considered for the template creation, accuracy may change significantly. The *Easy 1* dataset does not show noticeable variability. *Difficult 2* shows limited variability, since, even if some corner cases determine significant degradation (up to 0.3 points), three quartiles of the experiments overlap with the offline accuracy level. *Difficult 1* and *Easy 2* accuracy changes significantly, i.e. results are less predictable for 100 and 200 *training* spikes. In these cases using 300 spikes appears to be the value minimizing, at the same time, variability and training set size.

Iterative clustering on the PS

Considering that, in general, most of the considered training sets result in an accuracy level close to the off-line analysis, we have tested a template definition methodology that repeats the K-means clustering along the duration of an experiment, limiting the effect of poorly-performing training sets of spikes, using each template set for a shorter time. The C-language K-Means implementation is taken from [53]. The cluster centers are initialized using K-Means++ [54]. The maximum number of iterations for refining the center was set to 10. To assess the possibility of repeating the clustering, we have measured the algorithm execution time on the PS. Table 3.5 shows the average execution time over 1000 runs. As may be noticed, the execution time changes for different datasets, since the algorithm requires a different number of iterations to converge. The execution is in general reasonably fast. The average time in table 3.5 is about 106 μs, corresponding to less than 4 sample times, confirming the possibility of executing the algorithm multiple times to refine the templates during an experiment.
The rightmost box-and-whisker of each plot in Figure 3.8 shows the accuracy obtained by run-

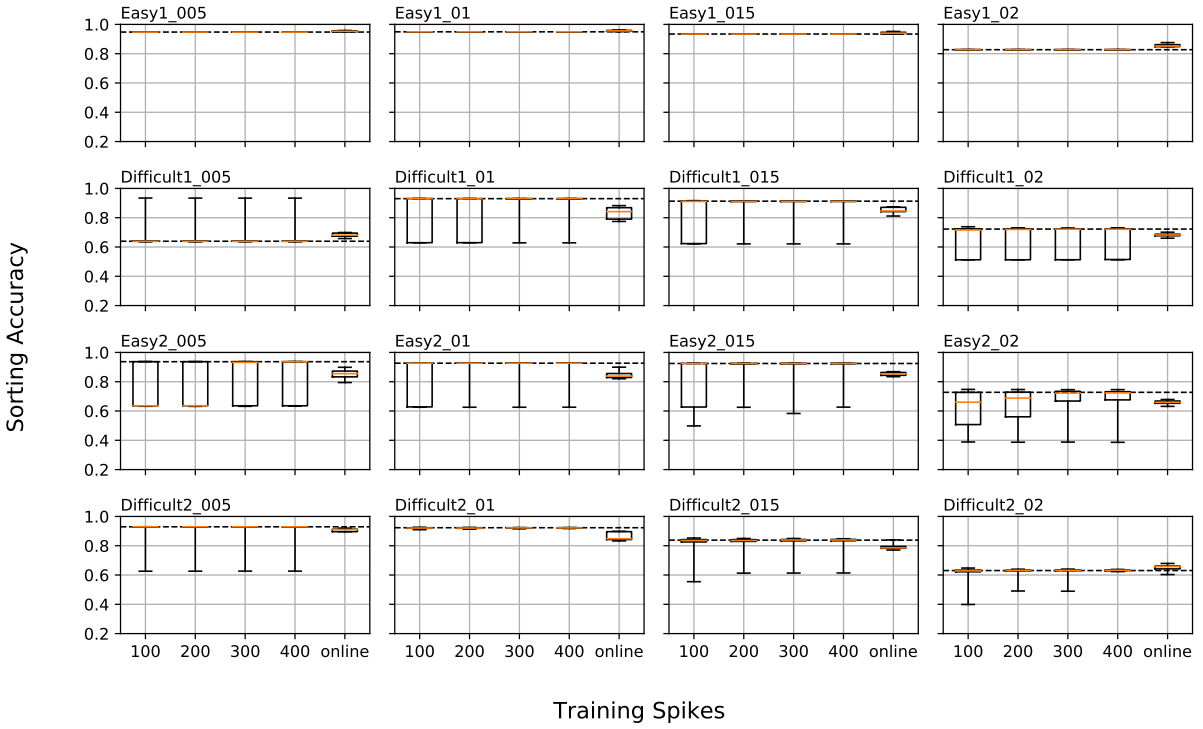


Figure 3.8: Sorting accuracy distribution over 200 randomly chosen sets of training spikes of variable dimension: 100, 200, 300, and 400. The box-and-whisker plots show the accuracy variability. The K-Means algorithm was run to determine the cluster-centers/classifier-templates, afterwards the spikes are classified. The noise increases along the columns, the datasets swipe along the rows. The dashed line represents the offline accuracy of the method, where the K-Means was run on the whole dataset; the last box-and-whisker plot of every figure shows the accuracy we achieved with the proposed real-time method.

$[\mu s]$	0.05σ	0.01σ	0.15σ	0.20σ
Easy 1	85	98	116	120
Difficult 1	102	143	190	203
Easy 2	73	80	71	75
Difficult 2	71	78	90	108

Table 3.5: K-Means run time on the PS, each run considers 300 spikes, every time measure is averaged over 1000 runs.

ning the K-Means clustering over 300 spikes once every three seconds (an execution rate that can be comfortably supported for thousands of channels considering the measures in Table 3.5). We run the experiment 10 times per each dataset selecting a different starting point in the neural signal, to obtain more reliable results. As may be noticed, variability is significantly reduced. Unfavorable corner cases are avoided and the worst-case accuracy is significantly improved. The obtained overall mean accuracy is about 82.4% and variability is much more contained, with values ranging from 79.8% to 84.9%.

To demonstrate the flexibility derived by software programmability, we have implemented on the PS a second clustering algorithm, based on Self-Organizing Map (SOM). A thorough accuracy evaluation for the SOM method in this case would require a more complex exploration of the algorithm hyperparameters, which is beyond the scope of this chapter. However, we have tested the execution with some basic settings to estimate the execution time. We have used the publicly available C-language implementation released as open-source under MIT license at [55]. The SOM algorithm is more complex than K-means, its run-time is the same for all the datasets since it stops when the maximum number of iterations is reached. By setting a 4x2 neural network and considering 200 training spikes, the average training time is about 2.67 seconds.

3.5.4 Implementation evaluation

Multiple tests were conducted prior to our design choice, evaluating the impact of architectural details on the overall accuracy.

Feature extraction and detection methods

To assess the impact of the chosen First and Second Derivative Extrema [47] algorithm on the accuracy, we have compared it with the Zero Crossing method [44], implementing it on our reference software pipeline. We also estimated the impact of the spike detection accuracy on the overall results. The K-Means clustering algorithm is used to measure the final sorting accuracy and to compare the methods.

Figure 3.9 shows four plots, one for each dataset track. The plots report the sorting accuracy at

four different noise levels. The dashed lines represent the sorting accuracy over the detected spikes only, as stated by Eq.3.9, where C are the spikes sorted correctly, and T_p are the true positives, i.e. the properly detected spikes. The solid lines represent the overall accuracy, where missed detections are accounted as well as misclassifications as stated by 3.10, where F_p are the false positives, and F_n are the false negatives.

$$Sorting\ Accuracy = \frac{C}{T_p} \tag{3.9}$$

$$Overall\ Accuracy = \frac{C}{T_p + F_p + F_n} \tag{3.10}$$

In simulation *Easy 1*, the FSDE accuracy is over 0.9 for the first three noise levels, and it is

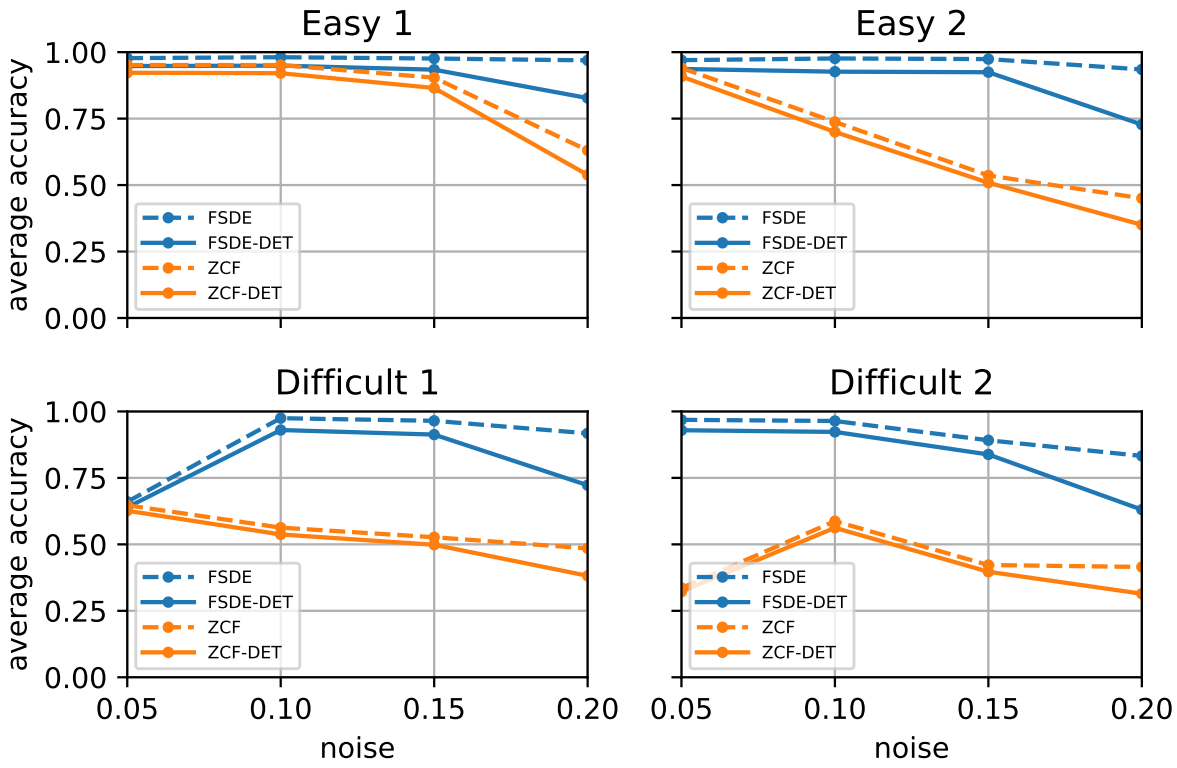


Figure 3.9: First and Second Derivative Extrema and Zero Crossing Feature extraction methods comparison.

only slightly better than the ZCF one. Nevertheless, the ZCF accuracy falls when the noise level is increased at 0.20σ, whereas the FSDE accuracy still is at 0.83. In simulations *Easy 2* and *Difficult 1*, the methods exhibit the same accuracy for 0.05σ. However, ZCF is not able to

maintain the same accuracy of FSDE for higher noise levels, dropping down to less than 0.5 in both simulations. In simulation Difficult 2, the accuracy performance of ZCF is constantly lower than FSDE.
FSDE appears to be dominant in every track, showing a better capability of extracting valuable features, at least when combined with the use of the K-Means algorithm. Furthermore, FSDE also appears more resilient to higher noise levels than ZCF.

Fixed-point implementation

The architecture embeds fixed-point processing elements rather than floating-point ones. The incoming samples data provided by BioCam X are 12-bits wide. We chose to allow data size to grow inside the datapath along the processing steps. This does not affect the transmission rate to DDR, since the information which needs to be sent in output are the FSDE features, encoded in 12-bits, and the classification results, encoded in 3-bits (to represent eight neurons per channel). This architectural choice, even though requires slightly more resources, permits the hardware implementation to be as accurate as the software floating-point simulation.

Spike's window centering exploration

We found a correct centering of the spike window around the spike detection event before extracting the features to be key for the overall accuracy. As previously mentioned, such centering is implemented by continuously keeping track of the recent samples using a FIFO. The number of preceding samples stored in the FIFO, as well as, obviously, the number of channels, has a direct impact on the utilization of BRAMs. Table 3.6 shows the average offline accuracy of the system at the varying of the spike window alignment with the spike event. The alignment is expressed as the number of samples stored in the FIFO. The BRAM deriving from each alignment choice is as well reported in Table 3.6. The configuration with 8 samples leads to the highest accuracy. Decreasing the length of the spike head memorized in the buffer, important information about the characteristics at the beginning of the spike waveforms are lost and the accuracy is negatively affected. Nevertheless, increasing the number of samples to 16 loses too much information from the tails of the spikes, leading to lower accuracy and an over-utilization of the BRAM resources.

Alignment	Accuracy	BRAM
2	74%	3
4	82%	5
8	86%	11
16	79%	22

Table 3.6: Offline sorting accuracy by varying the FIFO buffer size.

3.6 Comparison with State of the Art

Table 3.7 shows the main characteristics of the works we target to be compared to our implementation. To the best of our knowledge, literature does not present any implementation able to process 4096 electrodes simultaneously in real-time, including support for spike sorting.
In [39] Park et al. present a multichannel neural interface capable of sorting 128 channels simultaneously and stimulating the neural tissue from 8 electrodes. The neural interface presented is based on template matching and it is hosted by a Xilinx Kintex-7 XC7K160T. The device embeds 600 DSP slices and 325 36Kb BRAM tiles. No precise resource utilization is given, nevertheless, they require 6 kb of memory per channel, whereas we only need 0.92 kb per channel by adding together both the BRAMs and the registers utilization and dividing the sum by the number of channels (4096). Even considering 16 bits of resolution like in [39], instead of 12 bits, our memory would increase to 1.23 Kb per channel only.
The Parallel OSort algorithm (POSort), presented in [30], is prototyped on both a Xilinx Spartan-6 and a Xilinx Virtex-6 devices. Table 3.7 reports the Virtex-6 single channel implementation features, since it is the best version, in terms of accuracy and latency, between the fully documented ones shown in [30]. However, the POSort can handle up to 64 and 128 channels if hosted by high-end FPGAs like those in the Virtex and Kintex families. The memory required to operate on 64 and 128 channels is respectively 960 and 1920 BRAMs while this work is capable of sorting 4096 independent channels with 104 BRAMs only. Even though the POSort algorithm requires about half of our LUTs and a third of our registers, it needs more than double of our DSPs and about 590 more BRAMs per channel. Its accuracy is 87%, higher than our result, however, it could not scale up to 4096 channels unless an unreasonable amount of memory were available. The total POSort system latency is not provided, however, it is available the clustering latency which is about 0.25 μs. Although the total latency of this work is 2.3

ms, the main contribution is given by the FIR filter bank and the FIFO, and our classification latency is 0.08 μs, three times less than the POSort.

In [56] Dragas et al. present a 90-electrodes real-time spike sorting processor hosted by a Xilinx Virtex-6 FPGA. The presented system can process in real-time up to 650 neurons, which is 50 times less than our maximum number of neurons (we can consider 8 neurons per channel, per 4096 channels). Their work guarantees a latency of 2.65 ms, which is comparable to our result of 2.3 ms. The implementation requires 865 Kb of BRAM memory, 190,000 LUTs, and 29,000 REGs. No information about the DSP utilization is provided. Considering the significantly higher LUT utilization, it seems that the processing blocks have been implemented using arithmetic that does not map efficiently on DSP slices, using LUTs instead. Sorting accuracy is slightly less than 85%, which is 3 points above our result, and has been tested on a dataset with SNR above 5 dB.

In [57] is presented a single-channel real-time spike sorter hosted by a Xilinx Artix-7. The system can be instantiated multiple times (68 times), in order to handle an array of electrodes and sort up to 204 neurons, almost fully saturating the Xilinx Artix-7 LUT resources (98%). The reported system accuracy, of about 90%, was tested using a dataset with SNR in the range of 10-13 dB. To the best of our knowledge, it is the highest accuracy between the real-time spike sorters presented in the scientific literature. Unfortunately, by supporting 204 neurons only, this work is not compliant with the needs of more recent MEAs.

In [29] Yang et al. implement a 32 channels neural signal processor hosted by an Altera Cyclone III FPGA. The performance of the system in terms of accuracy is between 60-80% for signal-to-noise ratio in the range of 5-7 dB. Neither resource utilization nor system latency are provided; the reported number of channels and sorting accuracy are both lower than in the presented architecture.

In [58] Sungjin Oh et al. present a single-channel real-time spike sorter hosted by a Xilinx Spartan-6 FPGA and a PC. The neural signal is filtered, the spikes are detected, then, from the resulting spike waveforms, a technique similar to the ZCF is used to extract the features. The features are sent to a PC through an RS232 interface and clustered in real-time using the K-Means algorithm in MATLAB. The system was tested in-vivo, however, no accuracy data is provided to compare it to our work. In addition, no utilization data are even provided in terms

FPGA STATE OF THE ART COMPARISON TABLE

Reference	Park [39]	POSort [30]	Yang [29]	Dragas [56]	Valencia [57]	Sungjin Oh [58]	Schäffer [48]	Yi [31]	This work
Year	2018	2019	2017	2015	2019	2017	2020	2022	2020
FPGA	Xilinx Kintex-7	Xilinx Virtex-6	Altera Cyclone III	Xilinx Virtex-6	Xilinx Artix-7	Xilinx Spartan-6 + PC	Xilinx Zinq Ultrascale+	Xilinx Zinq Ultrascale+	Xilinx Zinq-7000
Input Channels	128	1	32	90	1	1	128	49	4096
Output Channels	8	0	0	0	0	0	0	0	16
Sampling [KHz]	32.5	24	20	20	24	50	20	30	18
Detection	yes	yes	TEO	-	TEO	2T	TEO	-	DT
Feature	no	no	DWT	-	-	ZCF	-	-	FSDE
Class/Clust	TM	OSort	BDT	BOTM	TM	K-Means	OSort	CNN	K-Means-based
Resolution [bits]	16	16	10	10	16	12	16	4-8	12
Accuracy	-	87%	(60%:80%)	$<$85%	90%	-	86%	86%	82%
SNR [dB]	-	[10:13]	[5:7]	$>$5	[10:13]	-	[3-10]	-	[7:13]
BRAM	-	29	-	865kb	0	-	98	27	104
DSP	-	130	-	-	5	-	60	6	61
LUT	-	16472	-	190000	6628	-	51674	8208	28984
REG	-	8444	-	29000	4880	-	17484	7057	26444
Frequency [MHz]	-	123	0.160	-	102	50	200	100	125
Clustering Latency [μs]	-	0.25	-	-	0.55	-	-	78	0.08
Latency [ms]	-	-	-	2.65	-	-	0.087*	$>$0.078**	2.3

*without filtering
*filtering and spike detection are not implemented

Table 3.7: FPGA-based spike sorters comparison summary

of LUTs, DSPs, BRAMs, and REGs.

In [48] Schäffer et al. present a real-time 128-channels spike sorter implemented on a Xilinx Zinq Ultrascale+ (ZCU106). For each spike detected a group of 3x3 channels is considered, centered in the channel sensing the highest absolute signal amplitude. The waveforms acquired by all 9 channels are processed by means of the Osort clustering algorithm. This allows for improved accuracy, 86% on average, tested on a dataset with SNR in the range of 3-10 dB, which outperforms the accuracy obtained in our work. However, due to the increased complexity, the number of processed channels in real-time is still 32 times lower compared to the capabilities of our architecture.

The work in [31] was also hosted by a Xilinx Zinq Ultrascale+ (ZCU-102) and implements a CNN-based classifier, without filtering and spike detection. The classifier reaches excellent accuracy performance (86%) and supports 49 channels with a latency of 78 μs. The classification latency is higher compared to the other works of Table 3.7, and this aspect should be considered in the case of closed-loop experiments. The hardware resource requirements are about a quarter of the hardware resources required by our implementation and a tenth in the case of the DSPs. This result is partially due to the fact that filters and spike detectors are not implemented. Moreover, scaling the number of channels of the architecture on more than 49 channels, 4096 for instance, such as the case of our implementation, to maintain the same latency, the hardware resources should increase of about 83.6 times (4096/49 channels).

3.7 Conclusion

We have defined a processing architecture supporting real-time spike sorting for MEA with thousands of channels. Such architecture, implemented on a Z7020 APSoC, can process in real-time up to 5,500 sample streams acquired at 18KHz. This outlines the possibility to use hardware implemented on FPGA-based reconfigurable logic to implement highly-parallel and low-latency neural signal processors. The selected set of hardware-friendly feature extraction and classification techniques effectively exploits DSP slices and BRAM storage resources available in the device, and effective pipelining can be applied to obtain reasonably high clock frequency. Moreover, we have demonstrated that the interaction with the integrated ARM-based processing system can be exploited online, to adapt to different experimental conditions. We have proved that the DDR memory available on the development board, reachable through the chip circuitry, provides sufficient storage capabilities and IO bandwidth to support data exchange between the data-crunching functional blocks implemented in the programmable logic and processing kernels executed by the hard cores. As an example, we have proposed an approach that repeats the clustering procedure during spike sorting, to limit the effects of unfavorable spike selection during the clustering definition process, improving accuracy to 82%, which corresponds to only 4% degradation with respect to off-line analysis. The proposed system increases by 43 times the supported number of channels compared to alternatives in the literature. The approach is suitable for closed-loop experiments since provides sorting results with a latency of 2.3 ms.

A prospective longer-term path of exploitation for our work derives from its complementarity with recent neuromorphic FPGA-based architectures, emulating different kinds of neurons [59][60][7]. Our spike sorter can be used to build an interface between such devices and MEAs, thus, the integration of these two approaches will pave the way to experiments involving the cooperation of biological and on-silicon neural networks.

Chapter 4

Enabling real-time SNN emulation with millions of synapses

Abstract

Closed-loop experiments involving biological and artificial neural networks would improve the understanding of neural cells functioning principles and lead to the development of new-generation neuroprosthesis. Several technological challenges require to be faced, such as the development of real-time spiking neural network emulators which could bear the increasing amount of data provided by new generation multielectrode arrays.

This chapter focuses on the development of a real-time spiking neural network emulator addressing fully-connected neural networks. It is presented a new way to increase the number of synapses supported by real-time neural network accelerators. The proposed solution has been implemented on the Xilinx Zynq 7020 All-Programmable SoC and can emulate fully connected spiking neural networks counting up to 3,098 Izhikevich neurons and 9.6e6 synapses in real-time, with a resolution of 0.1 ms.

4.1 Introduction

During the past decades, the comprehension of biological neural network phenomena has been at the center of researchers' interest in the medical and biomedical communities. Countless software and hardware instruments have been developed to enhance the understanding of neural cells' working principles. Whereas in the previous chapters the Thesis focused on providing a way to exploit real-time processing of neural data, it is worth mentioning other tools present in the scientific literature can simulate biological neural networks by relying on a wide range of mathematical models having a different level of detail [61], these kinds of tools can as well help the investigation of how neurons interact with each other, even though more and more often they are also exploited to address completely different problems, such as neuromorphic computing [62].

New generation multielectrode array, scaled from hundreds to thousands of recording sites [3], pushing for the development of signal processing systems capable of sorting order of magnitude more neural data in real-time than in the past [6], and artificial neural networks capable to keep up and process the incoming data. This translates into an imminent demand for bigger and more-connected neural networks. As a result, during the last years, the development of neural networks accelerator has increased consistently [63]. Moreover, networks of neural units are innately parallel, which means, standard Von Neumann architectures are not the best fit to simulate such networks. Therefore, also in this case, programmable accelerators, such as Field Programmable Gate Array (FPGA) based accelerators are best suited to the parallel and ever-changing demands nature of the experiments. Such hardware tools not only permit scaling down simulation time but also make possible real-time interactions between artificial and biological neural networks in a closed-loop fashion.

In this chapter is proposed a new method to increase the maximum number of synapses that can be emulated in real-time, without sacrificing the physiological dynamics and latency of biological neural networks. The method takes advantage of a physiological delay that affects the spike propagation along the cell's axon. This phenomenon, called axonal delay [64], makes possible to exploit the off-chip memory to store the synaptic weights. Furthermore, we applied the proposed method during the design of an FPGA-based hardware accelerator targeting fully

connected neural networks of Izhikevich spiking neurons [65]. Indeed, if on one hand, spiking neural networks encode information in the temporal domain, as biological neural cells do, emulating de facto more accurately the dynamics of biological cells, however, on the other hand, their implementation is more memory-demanding than non-spiking neural networks ones. Therefore, in the case of low-end FPGA implementing spiking neural networks, as for the Xilinx Z-7020 hosting the implementation presented in this chapter, where is not possible to store more than 5 Mb of data relying on the on-chip memory only, the sizes of the network cannot grow above a certain limit. The off-chip DDR memory has been used in other works that utilize spiking neural networks [66][67][68], however, the focus of these works was on image classification, rather than in the study of biological phenomena, therefore their architecture is not meant to be interfaced with a biological system which can require continuous interaction with a 0.1 ms resolution.
The main findings of this chapter can be summarized as follows:

- We demonstrate the physiological spike propagation delay present in biological neural networks can be exploited in the real-time emulation of spiking neural networks, guaranteeing a higher number of synaptic connections than by only using on-chip memory;

- We demonstrate Xilinx's APSoCs are eligible to apply the presented method, as their off-chip DDR memory has an adequate bandwidth to transfer the synaptic weights, and their use allows to increase the number of synapses that can be emulated in real-time;

- We demonstrate the Izhikevich neuron model equations [65] can be integrated into fixed-point arithmetic by relying on a few FPGA resources, such as DSPs and LUTs, without consistent behavioral variations.

The remainder of this chapter is organized as follows: Section 4.2 is an overview of existing FPGA-based neural network accelerators; Section 4.3 describes the utilized neuron model and studies its fixed-point accuracy. Section 4.4 is an overview of the hardware architecture; Section 4.5 presents the results in terms of accuracy and performances; Section 4.6 is a comparison with the state of the art; Section 4.7 is left to conclusions and future works.

4.2 Related work

A wide scope of software and hardware tools addressing spiking neural network emulation have been developed in the last few years. Software tools such as Nest [61], Neuron [69], and Brian [70] are well suited for biologically realistic simulation of spiking neural networks. They are flexible, and widely used by the scientific community for a wide range of experiments. However, they require larger and larger computer clusters for simulating high-count neural networks [71] and therefore are not the best fit for embedded applications.

Alternatively, parallel computing systems, implemented on a wide range of different platforms, such as CPU, GPU, and FPGA clusters, can achieve high throughput either. SpiNNaker [72] is a multiprocessor chip organized in a mesh of 48 neural computational cores, each made by 18 ARM968 processors. A board equipped with 4 SpiNNaker chips is capable of emulating in real-time a range of synapses going from 8e5 to 1.6e7 and a number of neurons ranging from 1,600 to 16,000, depending on the complexity of the neuron model used. NeuroFlow [73] is an FPGA-based spiking neural network simulation platform capable of emulating both Integrate-and-Fire (IF) and Izhikevich (IZ) neurons. When hosted by a cluster of 6 FPGAs it can simulate about 600,000 neurons, and from 1,000 to 10,000 synapses per neuron. The total amount of neurons decreases to 400,000 when the emulation is in real-time.

Moreover, at the state of the art, exists a broad collection of real-time FPGA-based spiking neural network accelerators more suited for embedded applications, having different scales, architectures, and use cases. Some work aims to implement low-power solutions, such as [74], where a neural network of 800 neurons and 12,544 synapses is implemented on a Xilinx Virtex-6 FPGA. The system implements a simplified Leaky Integrate-and-Fire (LIF) model [75] with a time resolution of 1 ms, and embeds real-time learning capabilities by integrating a simplified version of the Spike-Time Dependent Plasticity (STDP) algorithm [76]. Other works make use of the reprogrammable feature of FPGA and present configurable designs which could be exploited for a wider range of experiments, such as the work Snava [77]. Snava is a real-time programmable multi-model spiking neural network emulation system, capable of hosting up to 12,800 neurons and 20,000 synapses. The system, implemented on a Xilinx Kintex-7 FPGA, guarantees a resolution of 1 ms. The Snava system, employing a Graphical User Interface

(GUI), permits to monitor the spiking activity, to configure the neuron, the synapse model, and the interconnections.

The hardware implementation presented in [78] focuses on studying fully-connected neural networks; their real-time emulator targets closed-loop experiments, and it is hosted by a Xilinx Virtex-6 FPGA. The system implements 1,440 Izhikevich neurons with a resolution of 0.1 ms and a spike latency of 1 ms.

Other studies focus on more specific problems, such as minimizing the neurons' emulation latency down to 8 ns to increase the maximum number of neurons that can be emulated in a single FPGA chip, at the expense of the biological meaning [79]. This result has been achieved by designing a systolic array to integrate a simplified version of the Izhikevich neural model. By following the considerations found in [80], it is possible to decrease the computational load without renouncing the main emulating features of the Izhikevich model.

Conversely, Luo et. al [81] presented a bio-realistic cerebellum model, and propose it as the first step for the realization of neuroprosthesis systems with the purpose of substituting damaged motor control units in the brain. Luo et. al [81] propose a Network on Chip (NoC) hardware architecture, implemented on a Xilinx Virtex-7 FPGA, capable of emulating 101,000 LIF neurons [77] and 100,000 synapses in closed-loop experiments.

In Khodamoradi et. Al [82] is proposed an architectural solution to support several axonal delays without using extra FIFOs, schedulers, and separate routing networks for spiking feed-forward neural networks.

In Ambroise et. al [83], a folded low-resources architecture capable of emulating 117 Izhikevich neurons in real-time with a time resolution of 1 ms is presented. The system is implemented on a Xilinx Virtex-4 chip, and the interconnections of the neuron are configurable, ranging from zero to a fully connected network.

Finally, in Han et. Al [66] and Panchapakesan et. Al [67][68] Leak Integrate and Fire and Integrate and Fire based spiking neural networks are used to address image classification tasks on the MNIST and CIFAR-10 datasets on Xilinx Zynq devices, chip provided with both FPGAs and ARM processors. Their approaches take advantage of the off-chip DDR memory to store the weights of the network, however, not being designed as a biologically relevant neural network emulator, it is not applied any method to guarantee the physiological dynamics of biological

neural networks are respected.

Table 4.1 summarizes the main features of the above-mentioned FPGA works.

Table 4.1: Real-time FPGA-based neural network emulators comparison

Work	Year	Target FPGA	Neurons	Synapses
this work	2021	Xilinx Virtex-6	3,098	9.6e6
[74]	2020	Xilinx Virtex-6	800	1.25e4
[77]	2018	Xilinx Kintex-7	12,800	2.00e4
[78]	2017	Xilinx Virtex-6	1,440	2.07e6
[79]	2017	Altera Stratix IV	364	3.64e2
[81]	2016	Xilinx Virtex-7	101,000	1.00e5
[83]	2013	Xilinx Virtex-4	117	1.37e4
[66]	2020	Xilinx Kintex-7	2,842	1.86e6
[67]	2020	Xilinx ZCU102	2,410	-

4.3 Izhikevich neuron model

The Izhikevich model [65] permits the emulation of a large set of biological behaviors at a low computational cost. The model is composed of a two-dimensional system of ordinary differential equations 4.1, 4.2, plus a reset condition 4.3.

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{4.1}$$

$$\frac{du}{dt} = a(bv - u) \tag{4.2}$$

$$v > v_{th} \rightarrow \begin{cases} v = c \\ u = u + d \end{cases} \tag{4.3}$$

Where v is the membrane potential of the neuron, and u is the membrane recovery variable, both measured in mV. The term v_{th} is the threshold above which the modeled neuron fires a spike. When it happens, both the membrane potential and the membrane recovery variable are reset. The dimensionless parameters $a, b, c,$ and d permit tuning the model in order to emulate properly the behaviors of neocortical and thalamic neurons. I is the synaptic current, it permits taking into account the synaptic connection among neurons. Indeed, each synapse

can be described as an oriented and weighted connection between two neurons. When a neuron fires, its post-synaptic neurons counts the spike by adding to I the weight associated with their interconnection.

4.3.1 The quantization problem

The simplest and most common way to evaluate the Izhikevich model, nevertheless the way used in [65], is the one-step forward Euler scheme, described by 4.4, 4.5, and 4.6.

$$v_{k+1} = v_k + h(0.04v_k^2 + 5v_k + 140 - u_k + I + I_e) \tag{4.4}$$

$$u_{k+1} = u_k + ha(bv_k - u_k) \tag{4.5}$$

$$v_{k+1} > v_{th} \rightarrow \begin{cases} v_{k+1} = c \\ u_{k+1} = u_k + d \end{cases} \tag{4.6}$$

Where h is the time step, equal to 0.1 ms, and I_e is a parametric DC offset.
The above equations are solved by using fixed-point arithmetic so that a considerable amount of FPGA's resources could be saved. However, we found out that the accuracy and the convergence of the model, when operating in fixed-point, are not to be taken for granted. In order to investigate the behavior of the fixed-point implementation of the model, two MATLAB scripts have been developed. The former is used to provide a trustworthy ground truth for the experiments, which has been obtained by making use of floating-point arithmetic. The latter script is used to test the accuracy of the fixed-point solution at different levels of quantization.

4.4 Hardware spiking neural network

The spiking neural network emulator architecture is shown in Figure 4.1. The *Potential modules* integrate the Izhikevich equations, updating both the membrane potentials of the neurons and the spike conditions. The neural potentials and the spikes conditions are stored in two BRAM-based memories called *Potential mem* and *Spike mem*. The Izhikevich equations' parameters are stored in the BRAM-based memory *Param mem*.

The synaptic current is stored in an additional BRAM-based memory called *Current mem*,

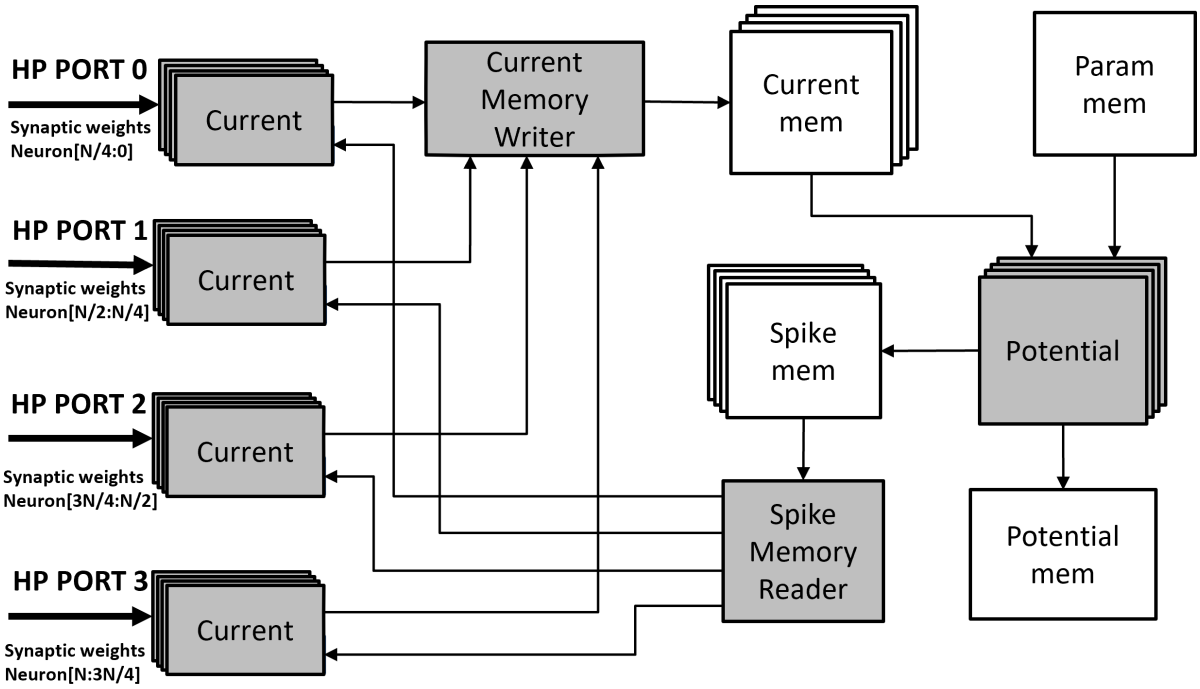


Figure 4.1: The block diagram of the neural network emulator. The synaptic weights are stream from the off-chip memory through four AXI High Performance ports to the programmable logic. Four clusters of current modules process the synaptic weights to evaluate the synaptic currents. The computed currents are stored through the Current Memory Writer, that permit sharing the single port of the current memory among four clusters of Current blocks. On a similar fashion, the module Spike Memory Reader allows four clusters of Current modules reading the spike conditions from a single port, avoiding data collisions. A set of Potential modules integrate the Izhikevich units, updating the neurons' internal state variable and the spike conditions.

updated by the *Current modules*. In order to update the *Current memory*, the *Current modules* read the stream of synaptic weights coming from the off-chip DDR through four axi-stream interfaces and the spike conditions from the *Spike mem*. The system is designed to exploit the

shared characteristics of the Xilinx Zynq-7000 family APSoC devices. The architecture is parametric, so the netlist can be generated to fit in different devices of the family and to emulate neural networks of different sizes. The system setup presented in this chapter is implemented on a Z-7020 chip.

The number of synaptic weights grows quadratically with the number of neurons in fully connected neural networks, and the Block-RAM (BRAM), which are the internal memories embedded in Xilinx's FPGA, are usually the bottleneck that prevents to increase the number of synapses over a certain limit. In the fully-connected neural network implemented in [78], the synaptic weights are stored on-chip, in the BRAMs, and the largest possible network which fits in is of about 1,440 neurons, obtained using 392 36 kb BRAM tiles in a Xilinx Virtex-6 XC6VLX240T chip. Indeed, if on one hand, the Programmable Logic (PL) is capable of performing heavy parallel computations, on the other hand, the available memory space is not enough to host larger fully-connected neural networks. Willing to overcome this result, we tried to exploit the off-chip DDR memory, that is the largest memory available in the Zedboard development board used for the implementation presented in this chapter. The DDR is 512 MB large, and it can be accessed concurrently from 4 High-Performance AXI ports (HP AXI ports), by using 4 AXI DMA operating at their maximum speed of 150 MHz [84], with an overall theoretical bandwidth of 4.8e9 B/s[84]. Keeping the same sampling frequency of [78], which is 10 kHz, it would be possible to move about 4.8e5 B in 0.1 ms, that by using synaptic weights of 8 bits each, would correspond to a fully-connected neural network of about 692 neurons. However, taking into consideration the biological delay that exists between the generation of a spike in the soma, and the propagation of the spike through the axon, towards the post-synaptic neurons, called Axonal Delay (AD), it is possible to relax the 0.1 ms deadline in favor of a looser one. Axonal delays' typical values can range significantly, from 0.3 ms for fast-conducting axons, such as cat visual thalamocortical axons, up to 130 ms, required to reach axon terminals in monkeys' visual cortex [64]. By using an axonal delay of 1 ms, as in [78], it would be possible to transfer the whole set of weights every millisecond instead of every tenth of a millisecond and reuse them 10 times to solve the Izhikevich equations. In this way, it would be possible to transfer 10x weights, which correspond to a fully connected neural network of 4.8e6 synapses, and therefore 2.191 neurons. The computational load would

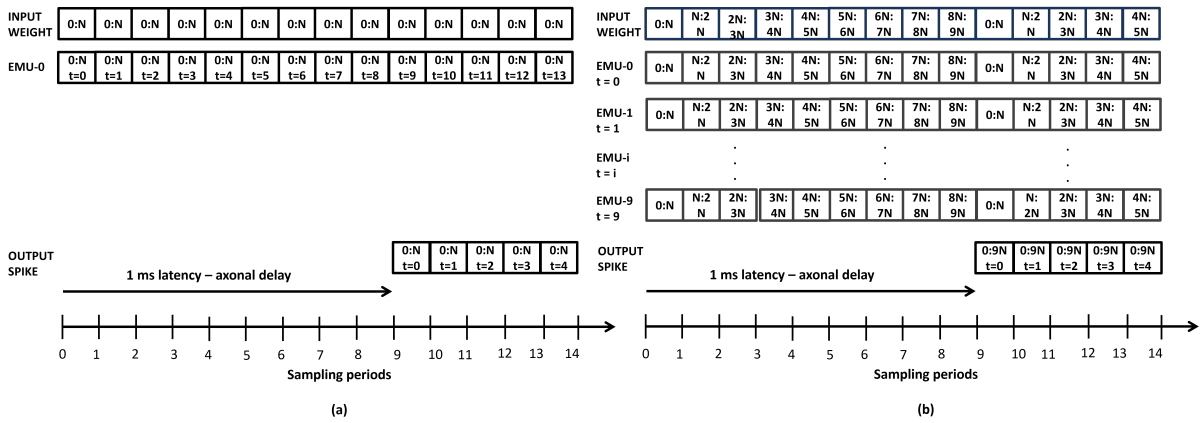


Figure 4.2: (a) The execution flow of a spiking neural network that takes into account axonal delay: the synaptic weights are stream from the external memory to the programmable logic at every sampling cycle; as the weights are available the neurons are integrated. (b) The execution flow of a spiking neural network that takes advantage of axonal delay: the synaptic weights are stream in during a set of sampling cycles, 10 in the example, allowing processing 10 times more synaptic weights; consequently, as the weights are available, it is computed in parallel the synaptic current of 10 sampling cycles.

increase consistently, however, this is obvious if more neurons and synapses are emulated in the same amount of time.

Figure 4.2 (a) shows the execution flow without taking advantage of the axonal delay: every neuron is updated at every integration step, and the spikes once ready, are forwarded 1 ms after to the other neurons because of the axonal delay. In Figure 4.2 (b) the synaptic weights are transferred during a longer period of time of 10 integration steps, the neuron integration is spread along this period, the spikes are still forwarded in output respecting the timing of configuration in Figure 4.2 (a).

4.4.1 Architectural overview

The main blocks of the biological neural network emulator and their interconnections are shown in Figure 4.1. The main actors are the *Potential modules*, that integrate the Izhikevich Equations 4.4, 4.5, and 4.6, and the *Current modules* that compute the synaptic current. Moreover, the spikes, the synaptic currents, the parameters of the Izhikevich model a, b, c, d, the membrane potentials, and the membrane recovery variables are stored inside BRAM-based memories, called after their contents, as shown in the schematic depicted in Figure 4.1.

The *Potential modules* are fully pipelined computational modules and have a throughput of one integrated neuron per clock cycle. By taking advantage of the parametric port size of Xilinx's BRAMs, it is possible to instantiate more potential modules in parallel when higher throughput is required. Moreover, the *Current modules* are fully pipelined computational modules as well, this allows to achieve a throughput of 8 summed synaptic weights per clock cycle. The synaptic weights are stored in the off-chip DDR and streamed through four AXI High-Performance Ports to the programmable logic. The stream is handled by four AXI DMAs. A different *Current module* is instantiated to handle each of the four streams of synaptic weights. With this setup, there is no need to store the synaptic weights on-chip, since they are processed as they arrive, and the BRAMs can be saved to store the neurons' model parameters.
If the axonal delay is higher than the integration frequency, more current modules can be deployed in parallel, and the currents of multiple time steps can be computed at once, as the weights are streamed in. Moreover, if that is the case, multiple instances of the *Current mem* and the *Spike mem* are required too, to store the currents and the spikes of all the time steps that fit in the axonal delay. The modules *Spike Memory Reader* and *Current Memory Writer* are used to write the synaptic currents in the *Current memory* and read the spike conditions from the *Spike memory*. These interface modules permit sharing a single memory port between the four sets of current modules in a time-multiplexed fashion, without creating any bottleneck, as explained in Section 4.4.2

Data transfer

The synaptic weights are moved from the DDR to the Programmable Logic (PL). The transaction is entrusted to 4 Xilinx AXI DMA IPs, each one connected to a different AXI High-Performance port. The response channels of the AXI buses are used to write back the spikes of the network in the DDR.

Potential module

The *Potential module* implements the Izhikevich Equations 4.4, 4.5, and 4.6. Additions and multiplications in Equation 4.4, 4.5, and 4.6 are mapped one-to-one on dedicated hardware resources. The architecture is shared among the neurons in a time-multiplexed fashion, guar-

anteeing a throughput of one integrated neuron per clock cycle. Multiple Potential modules can be instanced in the design.

The membrane potential pipeline makes use of three DSPs and a LUT-based multi-input adder, its architecture is shown in Figure 4.3. The first DSP is used to multiply the membrane po-

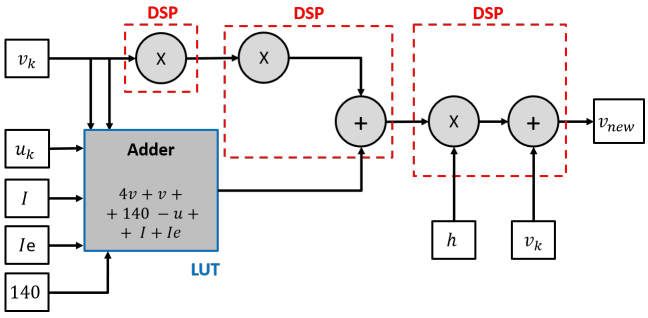


Figure 4.3: Block diagram of the membrane potential pipeline v_{k+1} from the Izhikevich model [18]

tential v_k to the constant 0.04. The product feeds the input of the second DSP block which multiplies $0.04v_k$ again by the membrane potential, obtaining $0.04v_k^2$. Concurrently, the addition $sum_v = 5v + 140 + u_k + I + I_e$ takes place by means of a LUT-based multi-input adder. The sum is the input of the post-multiplication adder embedded in the second DSP, so that the second DSP's output could be $\delta v = 0.04v_k^2 + sum_v$. Finally, the third DSP implements the operation $v_{new} = h\delta v + v_k$.

The membrane recovery variable pipeline is mapped into two additional DSPs. The block diagram, implementing Equation 4.5 is shown in Figure 4.4. The former DSP implements the

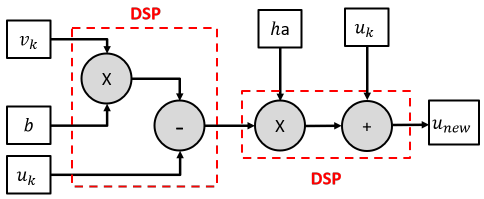


Figure 4.4: Block diagram of the membrane recovery variable pipeline u_{k+1} from the Izhikevich model [18]

operation $sum_u = bv_k + u_k$, and feeds the latter DSP, which multiplies sum_u by the precomputed parameter ha, and adds u_k to it, obtaining $u_{new} = u_k + ha(bv_k + u_k)$.

The reset/spike condition stated by Equation 4.6 is verified by a comparator, that in turn con-

trols two multiplexers as shown in Figure 4.5. When $v_{new} > v_{thr}$ the reset condition is activated, the second inputs of the multiplexers are chosen, therefore $v_{k+1} = c$ and $u_{k+1} = u_{new} + d$. Otherwise $v_{k+1} = v_{new}$ and $u_{k+1} = u_{new}$.

Once evaluated, the membrane potential v_{k+1}, the recovery variable u_{k+1}, and the spike con-

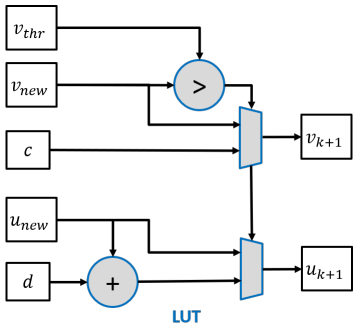


Figure 4.5: Block diagram of the reset condition architecture from the Izhikevich model [18]

dition, are stored in the *Potential memory* and in the *Spike memory*.

Current module

The *Current module* evaluates the synaptic current of every neuron of the network, so that Equation 4.4 could be integrated. The *Current module* architecture is shown in Figure 4.6. The synaptic weights are transmitted from the DDR through the AXI High-Performance ports, and processed on the fly, without the need to buffer them. Every synaptic weight is counted in the evaluation of the synaptic current if the pre-synaptic neuron is active. The spike conditions are read from the *Spike memory* as the weights come, in order not to count the weights of the inactive neurons. The weights of the inactive neurons are excluded from the addition by means of a *logical-and* operation involving each synaptic weight and its corresponding spike condition.

Every AXI High-Performance port transmits 64 bits per clock cycle, since the synaptic weights are 8 bits wide, the *Current module* processes 8 synaptic weights per clock cycle. Four clusters of *Current modules* are instanced in the design, one for each AXI High-Performance port, and every cluster is made by R *Current modules*, where R is the ratio between the selected axonal delay (AD) and the integration step, so that R synaptic currents could be evaluated in parallel, without retransmitting or storing the synaptic weights. The weights are added by means of a

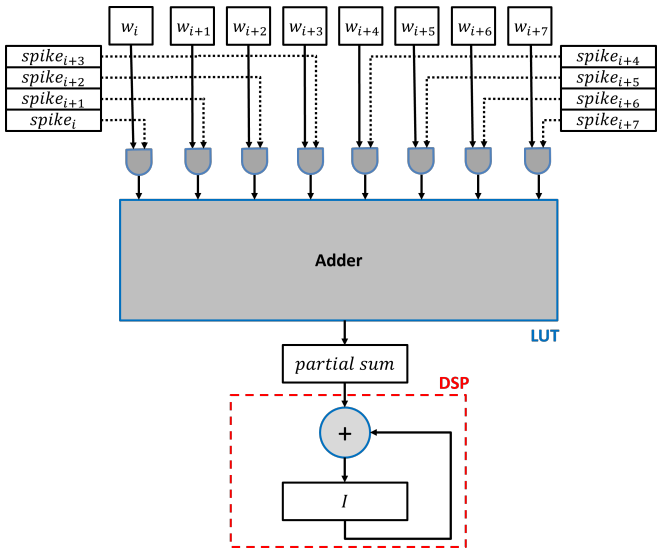


Figure 4.6: Block diagram of the Current module: the synaptic weights of the active synaptic connections are accumulated in the I register. The synaptic weights are read eight at a time, each weight is reset to zero if its corresponding spike condition is not active by means of a *logic-and* between the weights and the spike condition, then, the outcomes of the logical operations are added together and accumulated in the I register.

LUT-based adder, whose result drives a DSP-based accumulator, which permits computing the synaptic current during multiple clock cycles. Once evaluated, the synaptic current is stored in the *Current memory*.

4.4.2 Execution flow

The system execution flow repeats every time the selected axonal delay period expires. Each cycle can be described as follow:

- The Processing System (PS) enables four AXI DMA, which handle the transmission of the synaptic weights from the DDR to the PL, through 4 High-Performance AXI-Stream buses.

- Four clusters of R *Current modules* process the synaptic weights transmitted by the four DMAs through the four AXI High-Performance ports, concurrently, the *Spike memories* are accessed to identify active and inactive synaptic connections. A module called *Spike Memory Reader* arbitrates the accesses to the *Spike memory* through a single read port, by allowing only a cluster of *Current modules* per clock cycle to access, whereas the

others wait. The AXI-Stream transactions of the waiting clusters of *Current modules* are paused. Every time the *Spike Memory Reader* gets access to the spike memory on behalf of a cluster of *Current modules*, it reads in advance the spikes needed by that cluster for the next 4 clock cycles at once, taking advantage of the configurable width of the BRAM ports. Therefore, since 8 weights are transmitted per clock cycle, 32 spikes are read each time. By doing so, during steady-state processing, access conflicts do not take place and the *Current modules* clusters access one after the other without any conflict.

- Once a stream of weights starts, each *Current module* in the same cluster computes the synaptic current of the same neuron, just at a different point in time. This is possible because the computation of the synaptic currents requires the synaptic weights and the spike conditions previously evaluated and stored in the *Spike memory*. Once evaluated, the synaptic currents are stored in the *Current memory*. The *Current memory* is organized as a multi-banked memory of R banks, where R is the ratio between the axonal delay and the integration step. Each bank has an entry per neuron, whereas different banks host currents of different integration steps.

- As soon as the synaptic currents are available the neurons are integrated. To keep the potential fetching logic simple, four *Potential modules* integrate the Izhikevich equations of four different neurons concurrently. Once finished, the membrane potentials of the same neurons in the next integration steps are evaluated. When all the integration steps of those neurons are evaluated, the *Potential modules* start integrating 4 new neurons. The results are stored in the *Potential* and in the *Spike memories*.

- The evaluated spikes are written into the RAM. The spikes are moved by using the response channels of the AXI High-Performance ports.

4.5 Results

In this section, the performance, the hardware resource utilization, and the accuracy of the presented work are analyzed.

4.5.1 System performance

The design presented in this chapter is implemented on a ZedBoard, a low-cost development board for the Xilinx Zynq Z-7020 All-Programmable SoC. The architecture is parametric along multiple axes, as the number of neurons, synapses, and the axonal delay. The system setup chosen is the one that permits the emulation of the maximum number of fully connected neurons, which is 3,098, with 9.6e6 synapses and an axonal delay of 3 ms. To achieve this result, instead of storing the synaptic weights in the chip's internal BRAMs, which are not enough to memorize 9.6e6 bytes, the synaptic weights are stored in the off-chip DDR and transferred through the 4 AXI High-Performance ports present in every Zynq device. Four DMAs take care of the transmission of the weights, clocked at their maximum speed of 150 MHz [84].
Taking into account that the chosen emulation step is 0.1 ms, which is a common value in neuro-engineering applications, the system should process the whole set of synaptic weights every 0.1 ms. However, taking advantage of the axonal delay, a physiological latency that exists between the generation of a spike in the soma[1], and its propagation through the axon[2], towards the post-synaptic neurons [64], it is possible to generate the outputs with a certain latency without diverging from the physiological behavior. Taking advantage of this, it becomes possible to spread the transmission of the synaptic weights into more than a tenth of a millisecond, having more transmitted weights without violating the physiological dynamics of the neural cells as a result.
Increasing the axonal delay, from the performance point of view, permits to increase the operational intensity, i.e. the number of operations per byte, and therefore to enhance the FPGA throughput, at the expense of instantiating multiple current modules. The Roofline model, shown in figure 4.7, helps understanding how the operational intensity and the performances of the architecture change depending on the Axonal Delay (AD) value. The x-axis is the operational intensity measured in operations per byte ($ops/byte$), and the y-axis represents the overall performance in terms of the number of operations per second ($Gops$). The operational intensity rises up to 30 $ops/byte$ as the axonal delay increases. It is not possible to go beyond this limit, reached for an axonal delay of 3 ms, because the BRAM tiles saturate, mostly for the

[1]Soma: the portion of the neuron containing the nucleus.
[2]Axon: the portion of the neurons through which the spikes are propagated before reaching other neurons.

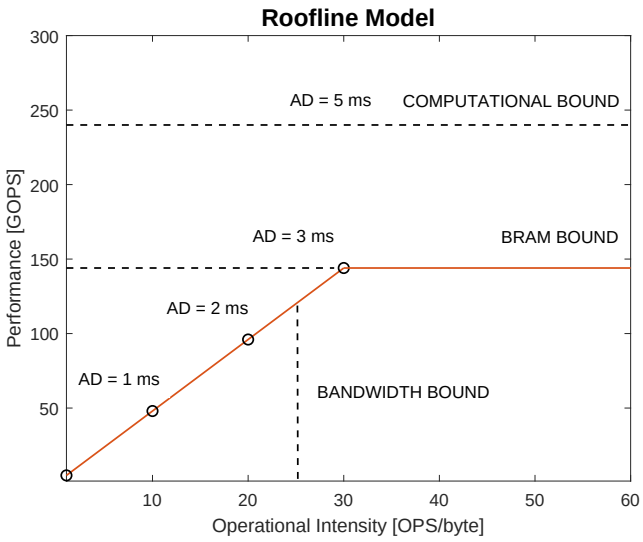


Figure 4.7: The Roofline model at the varying of the Axonal Delay (AD).

increment of the Izhikevich parameters and the synaptic currents memory requirements.

The computational bound of 240 Gops, represented by the upper horizontal dotted line in Figure 4.7, would be reached with an axonal delay equal to 5 ms, which would permit an operational intensity of 50 ops/byte. To achieve such performances, it is necessary to instantiate 50 current modules per axi port, for an overall number of 200 current modules, any of which would compute 8 additions per clock cycle at 150 MHz.

In the case of a 3 ms axonal delay: the theoretical number of weights that can be transmitted in real-time in 3 ms is 1.4e7, if the digital system is clocked at 150 MHz[84]. However, we experimented that is not possible to transmit more than 9.6e6 weights. To process the incoming weights, thirty *Current modules* are instantiated per AXI HP port, for an overall number of 120 *Current modules*. Moreover, four *Potential modules* read the synaptic currents once computed, integrate the Izhikevich equations, and evaluate the spike conditions. With this setup, the presented system is capable to emulate in real-time a fully connected neural network of 3,098 neurons and 9.6e6 synapses, with a resolution of 0.1 ms and an axonal delay of 3 ms.

4.5.2 Hardware report

The Zynq 7020 hosts 106,400 Flip-Flops (FFs), 53,200 LookUp Tables (LUTs), 140 36Kb BRAMs tiles, and 220 DSP48E1 slices.

The *Current module* is implemented using an array of and-gates, a LUT-based eight-inputs adder, and a DSP-based accumulator. To meet the timing constraints of 150 MHz, two pipeline stages were introduced inside the cascade of and-gates and multi-addend adder. The *retiming* option in the settings panel of the Vivado synthesizer was enabled during the synthesis step, allowing a more fine-grain retiming of the pipelining registers along the combinational paths. Table 4.2 shows the post-implementation resource requirement of a single *Current module*, obtained utilizing Vivado 2019.2. Note that 120 *Current module* instances are necessary to properly work in real-time since the selected axonal delay is 3 ms.

The *Potential module* exploits the computational capability of 5 DSPs, in addition, a LUT-based multi-input adder and a LUT-based comparator are also used. One DSP is used as a multiplier, whereas the remaining four DSPs are configured to use both the multiplier and the post-multiplier adder. To meet the timing constraints of 150 MHz, the *Potential module* was pipelined taking advantage of the pipeline stages embedded in each DSP. Three pipeline stages were exploited for multiplication and multiply-and-accumulate operations. Lut-based computations were pipelined as well: one pipeline stage was inserted on each addition and comparison. The architecture embeds a total of 10 pipeline stages. Table 4.2 shows the post-implementation resource requirement of a single *Potential module*, four of which are instanced in the design.

The *Current memory* has an entry of 15 bits per neuron and two 60 bits ports; 30 instances of the *Current memory* are present, since the currents of 30 consecutive integration cycles are computed at the same time, for a total amount of 60 BRAMs, as shown in Table 4.2.

The *Potential Memory* has an entry of 42 bits per neuron; one instance of the *Potential memory* requires 5 BRAMs, as shown in Table 4.2. A single instance of the *Potential memory* is sufficient in the design, since the old potential values, once used, can be overwritten. Moreover, 13.5 BRAMs are required to store the neuron parameters (a, b, c, d, I_e, h).

The *Spike memory* contains a bit per neuron, and 60 instances are required, two per integration step. it is not possible to overwrite the entries of the *Spike memory* while computing the new spike conditions, since the synaptic currents of the following neurons should be computed by relying on the same spike conditions. Every *Spike memory* requires a single 18Kb BRAM, for a total of 30 36 KB BRAM tiles, as shown in Table 4.2.

Table 4.2: Hardware resources distribution among the main modules

	Processing Elements		**Memories**			
Resource	Current	Potential	Current	Potential	Param	Spike
INSTANCE	120	4	30	1	1	60
LUT	45	173	0	0	0	0
LUTRAM	0	0	0	0	0	0
FF	78	452	0	0	0	0
BRAM	0	0	2	5	13.5	0.5
DSP	1	5	0	0	0	0

The post-implementation resource utilization report of the whole system is shown in Table 4.3. The Zynq 7020 chip is only partially used. As expected the BRAM tiles are almost fully occupied, as 93.21% is utilized. Since most of the data-crunchy computational logic was mapped into DSPs, 63.64% of the DSPs are used. The 66.96% of the FFs are available, as well as more than half of the LUTs (53.98%), and only 4.5% of the LUTRAMs are used.

Table 4.3: Resource utilization table

Resource	Utilization	Available	Utilization %
LUT	24,480	53,200	46.02
LUTRAM	802	17,400	4.61
FF	35,158	106,400	33.04
BRAM	130.5	140	93.21
DSP	140	220	63.64

4.5.3 Accuracy evaluation

Equations 4.1, 4.2, and 4.3 are solved by using fixed-point arithmetic, so that a considerable amount of FPGA's resources could be saved. However, we found out that the accuracy and the convergence of the Izhikevich model, when operating in fixed-point, are not to be taken for granted. To investigate the behavior of the fixed-point implementation of the model, two MATLAB scripts were implemented. The former is used to provide trustworthy ground truth for the experiments, generated from a floating-point implementation of the Izhikevich model. The latter script is used to analyze the fixed-point accuracy and understand how many bits are needed to smoothly move towards a fixed-point representation. To assess the behavior of the

fixed-point neural network a set of 1024 fully-connected Izhikevich neurons were simulated, of which 768 are of the excitatory type, and the remaining 256 are of the inhibitory type. Even if it is possible making use of regular spiking and fast-spiking cells to model the whole set of excitatory and inhibitory neurons respectively, to simulate a more heterogeneous network, the directives proposed in [65] were followed. The excitatory neurons are modeled by setting $a_i = 0.02$, $b_i = 0.2$, $c_i = -65.0+15r_i^2$ and $d_i = 8.0+6r_i^2$, with r_i a random variable uniformly distributed on the interval [0,1], and i the neuron index. On the value of r_i depends the kind of neuron dynamic obtained. With $r_i = 0$ the cell dynamic is the one of a regular spiking cell, with $r_i = 1$ is obtained the dynamic of a chattering cell, and with r_i around the value 0.8, is emulated the dynamic of an intrinsically bursting neuron. In a similar way, all the inhibitory cells parameters are randomly assigned by using the following rules $a_i = 0.02 + 0.08r_i^2$, $b_i = 0.25 - 0.05r_i^2$, $c_i = -65$ and $d_i = 2$; so that for $r_i = 1$ is obtained the dynamic of a fast spiking cell, and for $r_i = 0$ is simulated the dynamic of a low-threshold spiking cell. For all the excitatory cells the DC offset I_e is set at 4 pA, whereas for all the inhibitory cells the DC offset is set at 2 pA.

The functionality of the fixed-point network is assessed at both the single-cell and the network levels. The spike jitter, or spike lag, was verified neuron by neuron between the floating and fixed point networks and used as a comparison metric such in [85], as long as the mean firing rate, and the interspike interval. Moreover, the networks' bursts were analyzed: the mean bursting rate, the burst duration, and the interburst interval of the networks were compared. The bursts are identified as a sequence of more than 4 spikes, with an interspike interval of less than 100 ms.

Custom Data Width Selection

The data width of every input, output and internal signal was chosen to optimize at the same time the emulation accuracy and the resource utilization. We analyzed the dynamics of the membrane potential, the recovery variable, and the synaptic current by relying on the floating-point Matlab simulation of the Izhikevich model. We found out the membrane potential reaches maximum values which fit into 8 integer bits, the recovery variable into 6 integer bits, and as regards the synaptic current, it fits into 8 bits. In order to optimize the data map-

ping into Xilinx's DSP48E1, the DSP's input data width has not to be exceeded. Conversely, to obtain the best possible accuracy with such processing elements, once determined the size of the integer part of the data, the remaining bits of the DSP's inputs can be filled with fractional bits. The DSP48E1 contains a multiplier with two input ports of 25 and 18 bits, and an adder with three 48-bit input ports, two of which are used to carry out the multiplication. Therefore, in the case of the membrane potential, which goes in input to the DSP multiplier, it is possible to choose 18 bits or 25 bits data widths, corresponding to the formats $< 8.10 >$ or $< 8.17 >$ bits. We did not find any significant difference in accuracy between the two formats, therefore, we chose $< 8.10 >$ to save BRAM tiles. As regards the recovery variable, it does not go directly inside a multiplier, therefore its data width can go up to 48 bits, being 48 bits the input data width of the adder embedded in the DSP. However, we saw empirically that 24 bits, with the data format $< 6.18 >$, is a good trade-off between accuracy and BRAM utilization. When it comes to the synaptic current, since its integer part fits into 8 bits, and the synaptic weights have the format $< 1.7 >$ bits, the format $< 8.7 >$ bits was chosen. The membrane potential is computed as follows:

- The constant 0.04 is represented with the format $< 1.24 >$. The first multiplication $0.04v_k$ requires fewer integer bits than the sum of the two integer parts of the factors involved in the multiplication. In fact, 0.04 is smaller than one. Since $1/16$ is bigger than 0.04, and it is equivalent to a 4 digits shift to the right, multiplying a number by 0.04 reduces its integer part of 4 bits. Therefore, the format for $0.04v_k$ is $< 4.21 >$. The fractional part size is selected to fill the next multiplier input width, and maximize the emulation precision.

- The addition $sum_v = 4v_k + v_k + 140 - u_k + I + I_e$ is implemented by means of a multi-addend LUT-based adder. The addends' sizes are listed in Table 4.4. The fractional bits of the term sum_v are the same of the membrane recovery variable u_k, whereas its integer part is ten bits wide.

- The product $0.04v_k$ is multiplied by v_k and added to sum_v employing a DSP. The integer part of the product fits in eleven bits, whereas the fractional part size is truncated to fourteen bits before being multiplied by the integration step h in the next DSP.

- The integration step is equal to 0.1, therefore the output of the multiplication $h(0.04v_k^2 + sum_v)$ requires fewer integer bits than the input factors. The power of two 2^{-3} corresponds to a three digits right shift, and it is bigger than 0.1. Then, the integer part of the product surely fits into 8 bits.

- The term $h(0.04v_k^2 + sum_v)$ is added to v_k by means of the post-multiplication adder of the same DSP. Since the dynamic of v_k does not exceed 8 integer bits, as observed in the floating-point simulation, this sum still fits into 8 integer bits.

The membrane recovery variable pipeline is composed of two DSPs that implement the operations $bv_k - u_k$ and $ha(bv_k - u_k) + u_k$:

- The 25 bits input port of the first DSP is used for the parameter b, with the format $< 1.24 >$. This term's maximum value is 0.25, which corresponds to a two-digits right shift, therefore bv_k will require 6 integer bits at most.

- The multiplier output is subtracted by u_k using the adder embedded in the same DSP. The output, in the format $< 6.19 >$, fits into the 25 bits input of the second DSP implementing the operation $ha(bv_k - u_k) + u_k$.

- The term ha, directly stored pre-computed instead of a (to save a multiplier), can reach the maximum value of 0.01. Therefore it reduces by at least 6 bits the dynamic of $(bv_k - u_k)$. In any case, being $ha(bv_k - u_k) + u_k$ the new value of the membrane recovery variable, it cannot have a dynamic that goes above 6 integer bits, as observed during the floating-point simulation.

Single neuron behavior

This section presents a behavioral analysis of the single-cell Izhikevich model. In particular, it is shown the behavioral difference when the model is evaluated using floating-point arithmetic, our custom fixed-point arithmetic, and standard fixed-point arithmetic, at the varying of the data width. For fixed-point arithmetic, ten bits are kept fixed for the integer part, the remaining bits are used for the fractional part. In fact, using less than 10 bits causes data overflows, as pointed out in [83]. On the other hand, using more than 10 bits for the integer part

Table 4.4: Fixed-point data format of the Potential module signals

Data	Format	Data	Format
v_k	8.10	0.04	01.24
$0.04v_k$	4.21	$4v_k$	10.10
I	8.7	I_e	5.7
140	09.00	u_k	6.18
sum_v	10.18	$0.04v_k^2 + sum_v$	11.14
h	01.17	v_{new}	8.10
v_{thr}	08.10	c	08.10
b	01.24	$bv_k - u_k$	6.19
ha	01.17	u_{new}	6.18
d	06.18	$weight$	1.7

does not provide any accuracy benefits.

Figure 4.8 shows the superimposition of the fixed- and the floating-point simulation of the membrane potential for several fractional bits widths, within a time window of 200 ms. From left to write regular spiking, chattering, intrinsically bursting, fast-spiking, and low-threshold spiking cells. The parameters used to simulate each neuron are listed in Table 4.5. In the first row are used 10 fractional bits. The membrane potential superimposition shows evident differences with the floating-point model. Both the number and the timing of the spikes differ. In the second row, the number of fractional bits is increased to 16. Starting from this data format the spikes count between the floating- and the fixed-point cells is the same, for all the cell types. However, it is still possible to observe a significant timing lag among the spikes. In the third row, 22 fractional bits are used, for a total data size of 32 bits. This format permits obtaining two perfectly superimposed simulations. The last row shows the case where the custom format described in 4.5.3 is used. There are no significant behavioral differences between using 32 bits fixed-point arithmetic and the custom data width proposed in this chapter. However, the custom data width permits to map efficiently the computations into Xilinx's DSPs and LUTs. Moreover, the overall memory required per neuron is 371 bits, about the 61% required if 32 bits data width is used.

Table 4.5: Single Cell Simulation Parameters Values

Type	a	b	c	d	I_e
Regular Spiking	0.02	0.2 7	-65	8	4
Chattering	0.02	0.2	-55	4	4
Intrinsically Bursting	0.02	0.2	-50	2	4
Fast Spiking	0.1	0.2	-65	2	4
Low-Threshold Spiking	0.02	0.25	-65	2	4

Neural network behavior

The firing patterns of the hardware fully-connected neural network of 1024 Izhikevich neurons were compared to the floating-point Matlab model. The networks were compared along two seconds of activity.

The spike timing for each neuron of the network is shown in Figure 4.9 (a). The x-axis represents the time in samples (the integration step is 0.1 ms, so 20k samples correspond to 2 seconds), whereas the y-axis has an entry for each neuron of the network; neuron identifiers from 1 to 768 are of excitatory neurons, neuron identifiers from 769 to 1024 are of inhibitory neurons. The firing activity superimposition of Figure 4.9 (a) shows a match between the hardware and the Matlab reference models. Within 2 seconds of activity, a total amount of 20,874 spikes were fired from the Matlab reference model, whereas 20,886 were fired from the presented hardware implementation. The total number of spikes fired by the networks differs by about 0.06%. The mean firing rate of the two networks is shown in Figure 4.9 (b); in the case of the hardware network, the MFR is 10.1924 spikes per second, whereas it is 10.1982 spikes per second in the case of the Matlab reference model. Among the spikes fired from the Matlab reference model, 20,620 were correctly replicated from the hardware network, with a maximum timing jitter of 2 ms, which corresponds to the 98.78% of the spikes fired. The 1.27% of the fired spikes are instead false positives, and the 1.22% are false negatives. The spike jitter distribution is shown in Figure 4.10 (a), the 98.78% of the spikes are correctly reproduced with a maximum jitter of 2 ms, of which the 89.68% have a time jitter less or equal to 1 ms. The Inter-Spike interval (ISI) values are shown in Figure 4.10 (b). The first and the second columns depict respectively the excitatory and inhibitory neurons of the networks, whereas the first

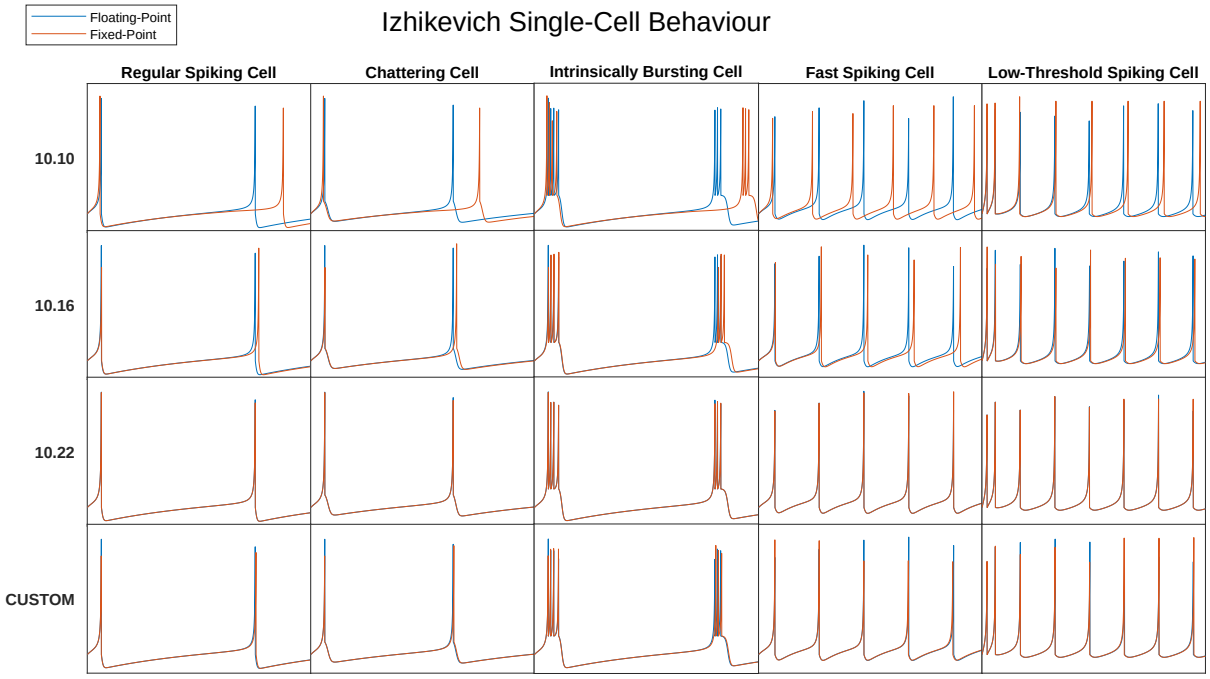
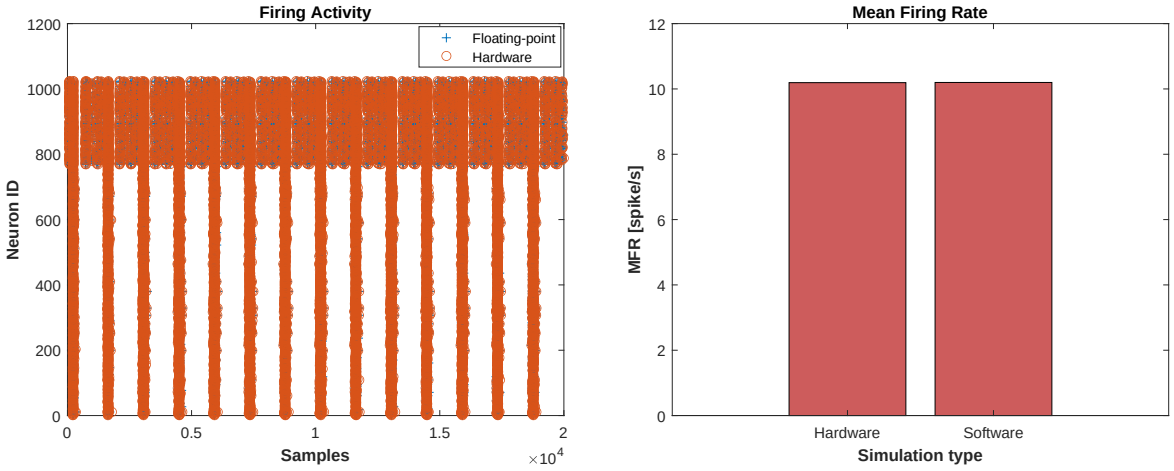


Figure 4.8: Fixed- and floating-point Izhikevich single-cell membrane potential superimposition at the varying of the fixed-point data width. The waveforms were captured within a time window of 200 ms. On the left is indicated the fixed-point data format. Each column depicts a different kind of neuron, from left to right regular spiking, chattering, intrinsically bursting, fast-spiking, and low-threshold spiking cells.

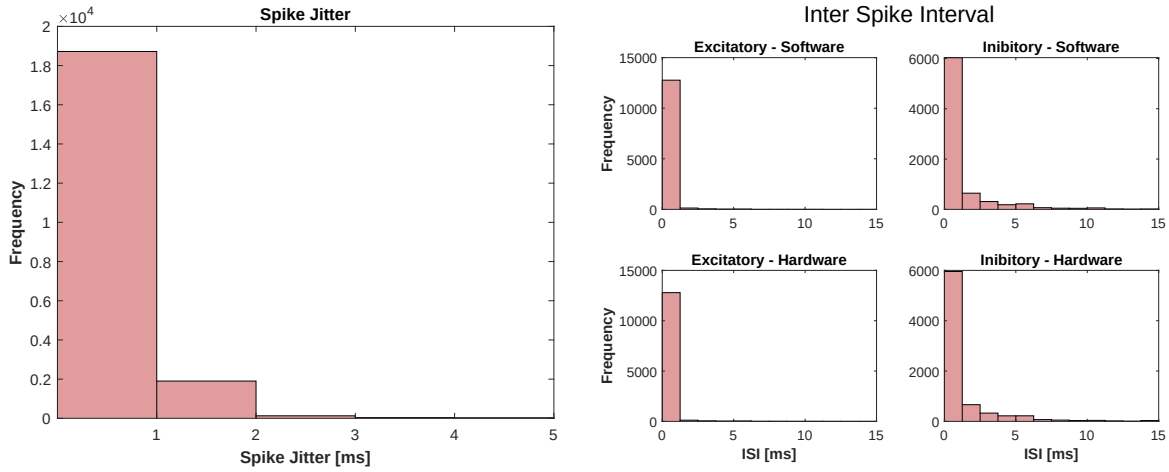
and the second rows show the Inter-Spike interval of the Matlab and the Hardware networks. There are no significant differences in the interspike time distributions of the hardware and Matlab networks, for both the inhibitory and the excitatory neurons.

The analysis of the bursting activity shows how the hardware network behavior still retraces the firing pattern of the Matlab reference model. Figure 4.11 shows the Mean Bursting Rate (MBR) of the two networks, the Burst Duration (BD), and the Inter-Burst Interval (IBI). The mean bursting rates are identical, in fact, the numbers of bursts of the two networks are the same. The average burst duration of the hardware network is 149.95 ms, whereas it is 150.55 ms for the reference network. The mean interburst interval of the hardware network is 123.49 ms, and the one of the software network is 123.03 ms.



(a) (b)

Figure 4.9: (a) Floating-point vs Hardware neural networks firing activity: the firing patterns are that similar to be indistinguishable in the image (b) Mean firing rate: the two networks present almost identical mean firing rate.



(a) (b)

Figure 4.10: (a) Spike time jitter: 98.78% of the fired spikes are correctly reproduced within a window of 2 ms, of which 89.68% within a 1 ms window. (b) Inter-Spike Interval: the first and the second rows show the Inter-Spike interval of the two networks, the first and the second columns depict the excitatory and inhibitory neurons of the networks. There are no significant differences in the interspike time distributions.

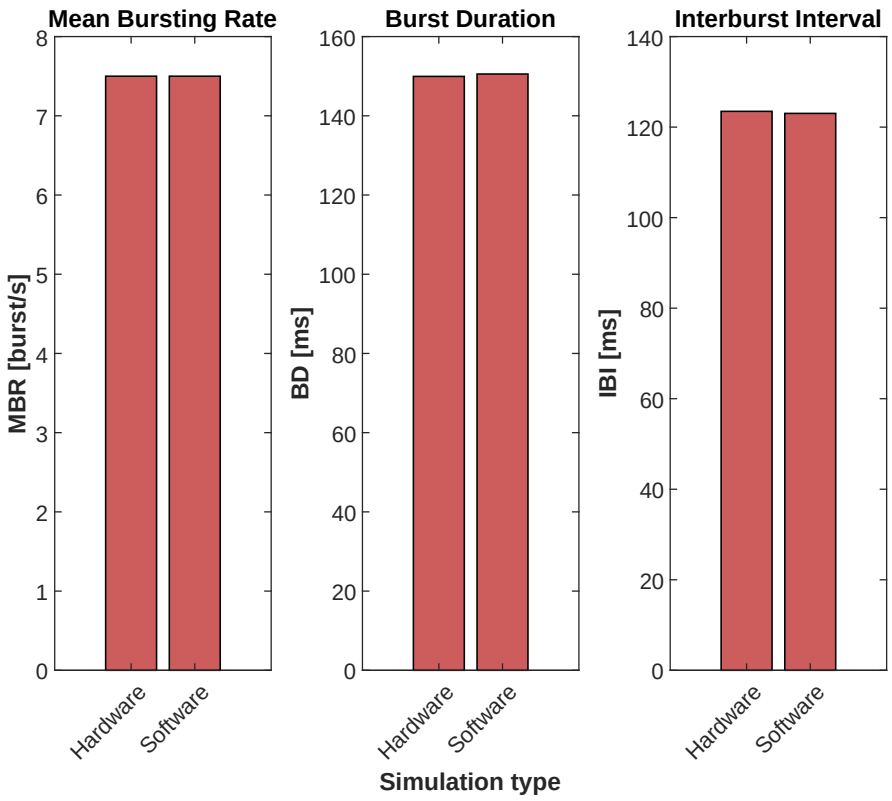


Figure 4.11: The burst metrics of the networks do not present any relevant variation, from left to right: Mean Bursting Rate, Burst Duration, Interbust Intervals.

4.6 Comparison with the State of the Art

The main characteristics of the FPGA accelerator we target for comparison with our work are shown in Table 4.6. The low-power embedded system implementation presented in [74] emulates 800 Leakage Integrate-and-Fire (LIF) neurons and 1.25e4 synapses. Our requirements in terms of DSP and registers are higher, 2.2x more DSPs and 1.5x more registers respectively. However, the work [74] uses 2.3x more LUTs than the presented system, and emulates only 25.82% of the neurons, and about 0.13% of the synapses of the proposed implementation. In addition, the neuron integration frequency is 10 times lower.

The digital system presented in [77] is capable of hosting 12,800 neurons and 20,000 synapses with a time resolution of 1.5 us. The network topology and the resolution used in [77] do not permit a fair comparison with the implementation presented in this chapter. In fact, this implementation emulates fully-connected neural networks with a time resolution of 0.1 ms, it counts 4.1 times fewer neurons, however, hosting 480 times more synaptic connections than in [77] can emulate arbitrarily connected neural networks.

The system presented in [78] is a fully connected neural network accelerator prototyped into a high-end Virtex-6 FPGA. The time resolution and the neuron model are the same as the presented work. Their digital system can emulate 1,440 neurons and 2.07e6 synapses, which are 46.48% and 21.56% of the proposed solution's result. Moreover, our work requires only the 43.80% of the LUTs, the 72.49% of the FFs, the 33.29% of the BRAMs, and the 34.31% of the DSPs compared to the implementation in [78].

The low-latency neural network accelerator presented in [79] is implemented on a Stratix-IV device, and can sustain a maximum clock frequency of 250 MHz. This allows to integrate the Izhikevich model in 8 ns and reuse many times the same neuron computational core within their 0.78 ms integrating step, as it happens in the presented work. However, the throughput of our *Potential module* guarantees an integrated neuron every 6 ns, which is higher than 8 ns, even though our maximum clock frequency is lower. In addition, the neuron interconnections scheme of the presented implementation has more biological meaning than in [79], where the neurons can have a single synaptic interconnection. The maximum number of neuron computational cores which fit in the Stratix-IV device is 364, but it is not explicitly declared the

Table 4.6: Real-time FPGA-based neural network emulators comparison

Work	Year	Family	Part	Neu	Syn	FC	mem	Model	Res	Data [bit]	LUT	FF	BRAM	DSP
this work	2021	Virtex-6	XC7Z020	3,098	9.6e6	✓	off-chip	Izhikevich	0.1 ms	custom	24,480	35,158	130.5	140
Gupta [74]	2020	Virtex-6	XC6VLX240T	800	1.25e4	✗	on-chip	Simp. LIF	1.0 ms	24	56,230	23,238	16	64
Sripad [77]	2018	Kintex-7	XC7K325T	12,800	2.00e4	✗	on-chip	Izhikevich	1.5 μs	16	148,774	97,824	213	100
Pani [78]	2017	Virtex-6	XC6VLX240T	1,440	2.07e6	✓	on-chip	Izhikevich	0.1 ms	32	55,884	48,502	392	408
Bandeira [79]	2017	Stratix IV	EP4SGX230	-	-	✗	on-chip	Izhikevich	0.78 ms	18	84,816*	84,816*	-	-
Luo [81]	2016	Virtex-7	XC7VX485T	101,000	1.00e5	✗	on-chip	LIF	-	40	268,455	176,424	960	2.304
Ambroise [83]	2013	Virtex-4	SX55	117	1.37e4	✓	on-chip	Izhikevich	1.0 ms	18	1,598	970	4	1
Han [66]	2020	Kintex-7	XC7Z045	2,842	1.86e6	✗	off-chip	LIF	-	16	5,381	7,309	40.5	-

*The quantity refers to the number of Adaptive Logic Module of Altera's FPGAs

maximum number of neurons their architecture can handle in real-time. Moreover, having a single synaptic connection per neuron, the comparison with the other architectures of Table 4.6 would not be fair.

The bio-realistic cerebellum model presented by Luo et. al [81] emulates 101,000 LIF neurons [77]. So as the presented work, the digital system can be coupled with biological neural networks in closed-loop experiments. Even though the number of neural units is consistently higher compared to the proposed implementation, it is to be taken into account that the presented system emulates Izhikevich neurons, which are far more computationally expensive than LIF neurons. Moreover, the number of synapses the presented architecture supports is 96 times higher than in [81] and this result is obtained by using the 9.11% of the LUTs, the 19.93% of the FFs, the 13.59% of the BRAMs, and the 6.07% of the DSPs of the implementation in [81].

The folded architecture presented in [83] permits saving resources by reusing the same processing elements along multiple clock cycles to evaluate the Izhikevich model. They instanced a single neural computational core that can be compared to our *Potential module*. Their core uses a single DSP, whereas ours makes use of 5 DSPs. However, due to this, their architecture requires about x9 more LUTs than our *Potential module*. Moreover, even though in a folded architecture is possible to save registers, by sharing them in time among multiple variables, the folded architecture in [83] still requires x3.5 more registers, probably because most of the registers used by the presented work are the ones embedded into the DSPs. The folded architecture needs 11 clock cycles to integrate the Izhikevich model, whereas our *Potential module* can integrate a neuron per clock cycle. In addition, the maximum clock frequency of the presented work is higher: 150 MHz against the 85 Mhz of the folded architecture in [83], and summing up, this leads to a throughput 19.4 times higher in favor of our *Potential module*.

The image classifier presented in [66] accelerates the execution of spiking neural networks of LIF neurons. Their approach takes advantage of the off-chip DDR memory to store the synaptic weights of the network, as in the proposed solution. As a matter of fact, their utilization in terms of BRAMs is lower than most of the other works in Table 4.6. Moreover, being the LIF model simpler than the Izhikevich model, and counting fewer parameters, their LUTs and FFs utilization is about a fifth of the presented implementation, and their BRAMs utilization is 31.03% of the presented work. However, even though the number of neurons is almost the

same (they emulate about 9% fewer neurons than the presented design), the synapses of our design are 5.16 times more. Finally, not being conceived as a biological-meaningful neural network emulator, [66] is not suited to be interfaced with biological neural networks.

4.7 Conclusion

We have presented a new method for increasing the synapses count of real-time neural network accelerators. We demonstrated the feasibility of the method by implementing a real-time neural network accelerator counting up to 3,098 neurons and 9.6e6 synapses into a Xilinx Zynq 7020 All-Programmable SoC, with a resolution of 0.1 ms. We showed that the off-chip DDR memory provides enough storage capability and I/O bandwidth to transfer the synaptic weights in real-time, and that by relying on the DDR memory, it is possible to overcome the number of synapses that a real-time spiking neural network emulator can store inside its BRAMs. In this chapter, it is demonstrated that it is possible to emulate highly-connected neural networks in real-time, paving the way to closed-loop experiments addressing biological and artificial neural network interaction, aiming to increase the actual comprehension of biological neural network functioning principles and neuroprosthesis development.
Moreover, we studied how to map the Izhikevich neuron model in fixed-point arithmetic so as to simultaneously find a good map into Xilinx's DSPs and LUTs, and degrade the accuracy of the network as little as possible. We found a difference of 0.06% in the total amount of fired spikes by the proposed fixed-point neural network and the floating-point reference model, with 98.78% of the spikes having a time jitter less than 2 ms.
A long-term purpose for our work is interfacing biological and artificial neural networks in real time. By relying on the support of a multielectrode array and a neural processing interface, it could be possible to provide input and output data exchange between the networks, making bio and artificial neural network cooperation possible.

Chapter 5

Exploiting SNN for efficient real-time neural decoding

Abstract

In the last decades deep learning neural decoding algorithms have been gaining pace in the broad landscape of neural interfaces and neural processing systems, however, these models must withstand strict computational and power limitations to be deployed on low-budget portable devices while operating in real-time. This Chapter presents a spike decoding system implemented on a low-end Zynq-7010 FPGA embedding a real-time low-power multiplier-less spike detection pipeline cascaded with a spiking neural network decoder mapped in the programmable logic. The system has been tested on two publicly available datasets achieving comparable results with State of the Art neural decoders based on more complex deep learning models, requiring 7.36 times fewer parameters than the smallest neural decoder 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 make this prototype a well-suited choice for low-power real-time neural decoding at the edge.

5.1 Introduction

Neural interfaces proved to be key components for improving the quality of life of disabled patients. Their range of applications stretches from hand movement decoding in patients affected by tetraplegia [86], seizure detection in pediatric subjects with intractable seizures [87], hand prosthesis control with sensory flow restoration in transradial amputees [88], speech decoding in patience with motor speech disorders [89], etc. Neural interfaces acquire neural signals, infer the patient's intention, and use this information to accommodate the patient's request. Among several recording solutions, intracortical sensors have shown to be valuable instruments in decoding multiple motor functions [90]. The neural data sampled with intracortical arrays of sensors can be seen as the contribution of two signals: the local field potentials (LFPs) and the action potentials, i.e. the spikes. The LFP is usually considered in the band [0.5, 300] Hz and can be obtained by filtering the recorded samples; the spikes are in the band above 300 Hz and necessitate being identified along the neural track after filtering, a neural signal processing task referred as spike detection [19] and that could be extended by recognizing the firing neurons, taking the name of spike sorting [28]. The output of spike detection and sorting are respectively called multi-unit activity (MUA) and single-unit activity (SUA). LFP, MUA, and SUA signals proved to be effective in several decoding tasks when supported by reliable decoding algorithms, such as Long-Short Term Memory (LSTM) used to decode LFPs [91], Recurrent Neural Network (RNN) used to decode MUA [92], Quasi-Recurrent Neural Network (QRNN) used to decode MUA, SUA, and Entire Spiking Activity (ESA) [93], i.e. a signal obtained by processing the spikes band without identifying the spikes.

Besides the application, neural interfaces are characterized by strict power and energy limitations, either because implantable chips must respect tight restrictions not to endanger the patient, limiting any temperature increase of the tissue to below 0.5°C [14], or because eventually, any neural interface should become portable, constraining its energy requirements in favor of extended battery life. In this sense, Spiking Neural Networks (SNN) are promising tools for low-power neural processing at the edge. SNNs differ from other neural network models as their fundamental units are spiking neurons, more similar to biological neurons, from which artificial neural networks derive. As biological neurons, spiking neurons have a

memory and are able to communicate exclusively receiving and transmitting action potentials, i.e. ones and zeros in the digital domain. The appeal of spiking neural networks lies in their computational efficiency, being able only to fire or not fire a spike, the synaptic connections among the units are not always active. The phenomenon is called spike sparsity and causes the number of computations to drop down, fostering reduced power consumption and latency. Even though 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 spiking neural network supervised learning [94], enabling wider use of these networks.

Furthermore, intracortical spikes and spiking neural network 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 Chapter presents a spike-based neural decoding system implemented on FPGA that exploits the computational efficiency of spiking neural networks to decode the displacement of a handle moved during a delayed reach-to-grasp task [8], and recorded by means of a 96-channel multi-electrode Utah array [4]. The neural signal is processed in real-time to extract the multi-unit activity by using a multiplier-less spike detector, mapped in the Programmable Logic (PL), presented in Chapter 2.

The key contributions of this Chapter are resumed as follows:

- A new computationally efficient real-time spike detection method is presented, along with its hardware implementation;

- An hardware spiking neural network is used for solving a continuous decoding task in real-time for the first time in the neural signal decoding domain (as far as we know);

- The accuracy of the system as a whole is assessed on a benchmark dataset, achieving accuracy comparable with the state-of-the-art and 7.36 times fewer parameters than the smallest neural decoder tested on the same dataset;

- Moreover, it is given proof of the computational efficiency of the spiking-neural-network-based decoder on real data, saving an average of 90% operations.

The following Sections are organized as follows: a description of the studies related to this work is presented in Section 5.2, the methods are described in Section 5.3, Section 5.4 reviews the hardware implementation, Section 5.5 contains the results, Section 5.6 comprises a comparison with the State of the Art, and Section 5.7 is left to the conclusions.

5.2 Related works

The Related works section is divided into two subsections. The former is used to describe the current State of the Art of neural activity decoders, the latter to provide an overview of the instruments available in the literature for the supervised training of spiking neural networks.

5.2.1 Neural activity decoders

Neural activity decoders, depending on the nature of the task, can be categorized into two families: gesture or motion classification and continuous motion or force decoding. The former constrains the output to a predefined set of classes and infers to which class the performed action belongs. The latter continuously tracks a target variable, either position, speed, or force, inferring the value by regression. In this section are analyzed several deep-learning neural decoders belonging to both categories, in addition, the models exploit several different arrays of sensors. The main features of the works analyzed in this section are summarized in Table 5.1.

Work	Year	Signal	Task	Decoder	Channels	Parameters	Platform
[95]	2022	sECoG	Regression	CNN+RNN	14	-	PC
[86]	2022	ECoG	Regression	2DCNN+LSTM	64	238,772	PC
[96]	2022	sEMG	Regression	FFNN	10	1,600	PC
[96]	2022	sEMG	Regression	CCFNN	10	1,928	PC
[96]	2022	sEMG	Regression	RBFNN	10	4,272	PC
[93]	2021	Intracortical	Regression	QRNN	96	-	PC
[97]	2022	sEMG	Regression	SNN	8	16,448†	PC
[98]	2022	sEMG	Classification	SNN	3	-	PC
[99]	2010	Intracortical	Classification	SNN	100	1,212†	PC
[100]	2019	EEG	Classification	CNN	10	-	FPGA
[101]	2016	ECoG	Classification	PCA+MLP	62	-	FPGA
[102]	2017	EEG	Classification	Bayesian Classifier	4	- deduced†	uC

Table 5.1: State of the art neural activity decoders

In [95] the stereo-electroencephalography (SECoG) signal is used for continuous force decoding during a hand-grasping task. The five participants had a total of 745 electrodes implanted, of which 9, 11, 14, 13, and 10 were used. The frequency components of each channel, over 5 different frequency bands, were extracted by exploiting five 6th-order Butterworth filters. The outputs of the filters were further processed to extract the power on each band by computing the square value of the Hilbert transform. Next, the obtained features were processed by a convolutional neural network followed by a recurrent neural network (CNN + RNN) composed of a temporal convolution block, a spatial convolution block, and a recurrent convolution block. In [86] the signal recorded by two ECoG implants of 8x8 grids is decoded during a 3D virtual hand translation task by means of a 2D CNN followed by a Long-Short Term Memory (LSTM) network. The patient, affected by tetraplegia caused by c4-c5 spinal cord injury, could control an avatar's right-hand movement by imagining moving his own right hand. The work focused on decoding the right/left-hand translations and use only half of the electrodes because of the bandwidth limit of the implant transmission rate. The signal was processed to extract frequency-domain features: from each of the 64 used channels, 15 continuous complex wavelets were extracted with central frequency regularly distributed between 10 and 150 Hz. The wavelets were computed over one-second windows with 90% overlap. Then, the absolute values of the wavelets were averaged over 0.1-second windows and used in groups of ten to feed the neural decoder after being *z-scored*, i.e. subtracted by the signal mean and divided by the signal's standard deviation. The features are given in input to the 2D CNN that performs a spatial convolution. The first layer of the LSTM, with its 50 units, analyzes the flattened output of the 2D CNN; the second layer of the LSTM, with its three units, provides the *xyz* coordinates of the hand trajectory.

Comparisons of different decoder types over the same dataset are hard to find. However, in [96] were compared the decoding accuracy of four artificial neural networks on a 15 joint angles continuous estimation task during nine wrist motions and nine grasp types. The neural activity was recorded using a 10-channel surface EMG (sEMG). The dataset is publicly available [103]. The surface EMG and the behavioral signals were zero-phase filtered by using two distinct 4th-order Butterworth filters with cutoff frequencies [10, 400] Hz and 10 Hz (low-pass) respectively, then, the signals were *z-scored*. The sEMG signals were normalized prior to com-

puting their RMS values. The RMS means over sliding windows of 150 ms, with 50% overlap, were used to feed five deep learning models, of which three are reported in Table 5.1: a Feed-forward Neural network (FFNN), a Cascade-Forward Neural Network (CCFFN) and a Radial Basis Function Neural Network (RBFNN) that outperformed the other models.

Whereas the previous works rely on surface electrodes, in [93] the intracortical recording sampled by means of a 96-electrode Utah array is used for assessing the performance on several decoders using different neural signal features. The datasets used as a benchmark contain the tracking of a handle moved during a reach-to-grasp task [8] and the tracking of the finger-tip movement during a reaching task [104]. Both linear decoders such as Wiener and Kalman filters were used, as well as deep learning decoders such as a Recurrent Neural Network, Long Short-Term Memory, and a Quasi-Recurrent Neural Network. Several neural signal features were tested in combination with the decoders, such as single and multi-unit activity, local field potential, and entire spiking activity. The decoding accuracy of the methods has been tested also over long-term neural recorded signals [104] to verify the signal degradation over time.

Third-generation neural networks are becoming part of the wide landscape of deep learning models used for neural signal decoding. Their computational efficiency is appealing, especially when the long-term goal is to deploy the deep learning model on an implantable SoC, or at least on a portable device, to really impact positively the patients' quality of life. However, in order to process the neural data with spiking neural networks it is necessary to convert continuous signals into spiking ones, adding an additional step in the neural signal processing chain. In [97] the Python implementation of a spiking neural network decoder trained to continuously track the elbow angle of four subjects moving their arm with 1.5 kg, 1 kg, and without any load is compared to the results obtained by a LSTM based decoder. The neural recording was acquired with an 8-channel surface EMG armband. Being the surface EMG signals continuous in time, to feed the spiking neural network the neural signal required further processing. Firstly, it is computed a time-domain feature called *waveform length*, defined as the aggregated length of the EMG waveform over the segment [105], computed over 100 samples in [97]. Then, this feature is input to a spiking layer of 64 units that convert the feature 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 a spiking neural network decoder is reported in [98], where a 3-channel surface EMG 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 called *A1*, *A2* and *B*, respectively in the band [15, 100], [100, 550] and [15, 550] Hz. The three signals are then smoothed by computing their root mean square value over a sliding window of 225 samples, corresponding to 150 ms. The signals are then normalized and an algorithm is run to convert the signals into spikes: the signal is accumulated sample by sample, when the accumulation value grows above a certain threshold it is reset and a spike is generated. Finally, the nine spike trains, three per electrode (*A1*, *A2*, and *B*), are processed by the decoder. The spiking neural network is a three-layer network with 9 inputs, 20 hidden units, and 5 output units. The neurons are of type 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.

We found only one work in literature that directly uses a spiking neural network for decoding spiking signals [99]. Their network analyzes single unit activity, recorded during a 3D reach-to-grasp task by five independent electrodes used multiple times over different trials, during which it was collected the activity of 979 units [106]. 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 movement was toward the left or the right and classifying one of the three possible orientations 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 orientations, and it is trained to spike at 51 ms for the orientation that represents or later, at 61 ms otherwise.

Although the approach used in [99] is similar to the method proposed in this chapter, 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.

Even though portability and low latency are key properties speaking of neural interfaces, we found prospectively few implementations addressing portable computing platforms such as FPGAs or microcontrollers (uCs) rather than PCs, that we reported in Table 5.1. In [100] a CNN deployed on a Field Programmable Gate Array is used for decoding in real-time the electroencephalographic signal acquired from 10 channels during a two-class motion imagery classification task. In [101] 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 into a dimensionality reduction performed through Principal Component Analysis (PCA) from 62 to 3 dimensions, then, a multilayer perceptron (MLP) is trained offline and used online to classify the finger movements. A microcontroller (uC) is used in [102] for implementing a low-cost neural interface solution for EEG-based neural decoding during an hand open/close/idle state classification task. The system is able of processing 4 EEG signals by means of a Bayesian classifier providing a results in about 2 seconds.

Although several hardware solutions have been proposed for intracortical neural signal processing, most of them stop at the spike detection phase [107][32] or at the spike sorting phase [6][39]. In the same way, several spiking neural network accelerators exist, some are oriented to understanding the brain functionalities, either FPGA-based prototypes such as [7][108], or higher-end emulators such as SpiNNaker [109]; whereas other hardware accelerators mostly focus on using spiking neural networks for their low-power properties, rather than exploit them for bio-realistic simulations. Among these implementations could be found both FPGA-based solutions [66][110] and custom ASIC designs, such as Loihi [111] from Intel, and TrueNorth [112] from IBM. However, it was never proposed, as far as we know, a custom neural interface prototyped on FPGA comprehensive of both neural signal analysis in the form of spike detection and neural decoding by means of a spiking neural network, as provided in

this chapter, ideal for low-cost neural signal processing and decoding at the edge.

5.2.2 Supervised learning for spiking neural network

Supervised learning is the most widely used way to compute neural network parameters. The process requires two input objects:

- A neural network model: the number of layers, the layers type, the number of neurons per layer, and the neurons type;
- A dataset: a list of input-output pairs that describes the neural network's desired behavior, where both inputs and outputs dimensions are arbitrary.

Supervised learning consists of letting the neural network model infer an output given a certain input, computing the error between the actual output and the target one, and using that error to adjust the parameters of the output layer. Furthermore, the error is *back-propagated* across the entire network, up to the input layer, allowing the refinement of the parameter values of all the network's layers. The process is usually performed for multiple input-output pairs of the dataset. This method is known as gradient descent back-propagating learning algorithm [113] and nowadays is the cornerstone of supervised learning algorithms.
Spiking neural networks, commonly referred to as III-Generation Artificial Neural Networks, differs from their predecessors for two main reasons: their neurons have memory, then, their outputs depend on the value of their internal state as well as from the value of their inputs, and their output is a spike, that is not differentiable. Especially this last aspect makes the gradient descent back-propagating learning algorithm not adequate for these kinds of networks, or at least not directly adequate.
Modified versions of the back-propagation algorithm suited for spiking neural networks exist in literature. The *SpikeProp* algorithm [114] overcomes the non-differentiability of the spike function by approximating it. Its main limitation is that the neurons can fire only once. Follow-up works introduced the possibility to learn other parameters aside from the weights, such as synaptic delays, and firing more than one spike [115].
Other strategies are possible, for instance, it is possible to first utilize the standard back-propagation method on a non-spiking neural network model, then, once the network is trained,

convert the model into a spiking one. In [116] this idea is applied to LIF neurons and demonstrated on CIFAR-10 and MNIST datasets, in [117] Integrate-and-Fire (IF) neurons are used and tested on the MNIST dataset.
Even though the algorithms differ, the common goal of spiking neural network learning algorithms is finding a set of synaptic weights that could reduce the difference between a computed spike train and a target one. The dataset, in the case of spiking neural networks, is in fact made of input spike trains and output spike trains. How the difference among spike trains is defined changes depending on the application case. It can either be defined as the difference between the spikes' time per neuron or as the sum of spikes fired during a certain interval of time by each neuron, which is usually the most suitable loss measure for classification tasks [118].
In a most recent work, SLAYER (Spike LAYer Error Reassignment) [118], was introduced the concept of back-propagation in time. In a spiking neural network, the error is indeed due to the present input, but also to the neuron's states. SLAYER during the learning process takes into account also the past inputs of the neurons, as well as the current ones.

5.3 Methods

In this chapter the intracortical recordings of a 96-channel MEA are used for decoding the displacement of a handle during a delayed reach-to-grasp task in real-time. The system is implemented on a low-end Zynq XC7Z010-1CLG400C and it 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) a spiking neural network that directly processes the detected spikes, inferring the handle displacement first derivative, i.e. the handle velocity.
This Section is organized as follows: a description of the dataset used to validate the system is provided in Section 5.3.1, in Section 5.3.2 is presented the spike detection pipeline, Sections 5.3.3 and 5.3.4 describe the neuron model and the spiking neural network, Section 5.3.5 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.

5.3.1 Dataset

The system is validated on two macaque monkeys' electrophysiological recordings [8]. The two monkeys were instructed to perform a delayed reach-to-grasp task. Each trial starts by informing the monkey on which type of action is required, two different types of grips and two levels of force were used, then, the monkey is required to pull a cuboid handle in that way. The two datasets, one for each macaque monkey, comprise a neural recording collected by means of a chronically implanted 10x10 multielectrode Utah array in the motor cortex sampled at 30 kHz, and behavioral data. We downsampled the neural recordings by a factor of 3 in this study. The behavioral data consists of the force applied to the cuboid handle, recorded by means of four force-sensitive sensors, and the handle displacement. The behavioral data is sampled at 1 kHz.

Monkey L was born on March 15, 2004. She started training in 2008, the Utah array was surgically implanted on September 15, 2010. The dataset was recorded on December 10, 2010. The Utah array bundle was cut on June 23, 2011. Monkey L was still alive in 2017. The recording is 11 minutes and 49 seconds long, the session took one hour and twenty-eight minutes. It contains 204 trials, of which 135 were successfully completed.

Monkey N was born on May 15, 2008. She started training in 2012, the Utah array was surgically implanted on May 22, 2014. The dataset was recorded on March 3, 2014. The Utah array bundle was damaged at the end of February 2015, then removed. Monkey L was also still alive in 2017. The recording is 16 minutes and 43 seconds long, the session took 51 minutes. It contains 160 trials, of which 141 were successfully completed.

Figures 5.1 (a) (b) (c) and (d) show how the handle position was processed before being used as the decoding variable during the training phase. The handle position, shown in Fig.5.1 (a), was smoothed with a moving average filter of order 64 Fig.5.1 (b), then its first derivative was computed to obtain the handle velocity Fig.5.1 (c). The first derivative was evaluated by subtracting adjacent samples. The handle velocity was then smoothed by using a mean average filter of order 16 Fig.5.1 (d) and used as a target variable for decoding. We chose the handle velocity as a variable for decoding to compare our neural decoding system accuracy with the extended analysis carried out in [93], where eight neural decoders and four neural signals were

tested on the same public neural dataset [8] using the handle velocity.

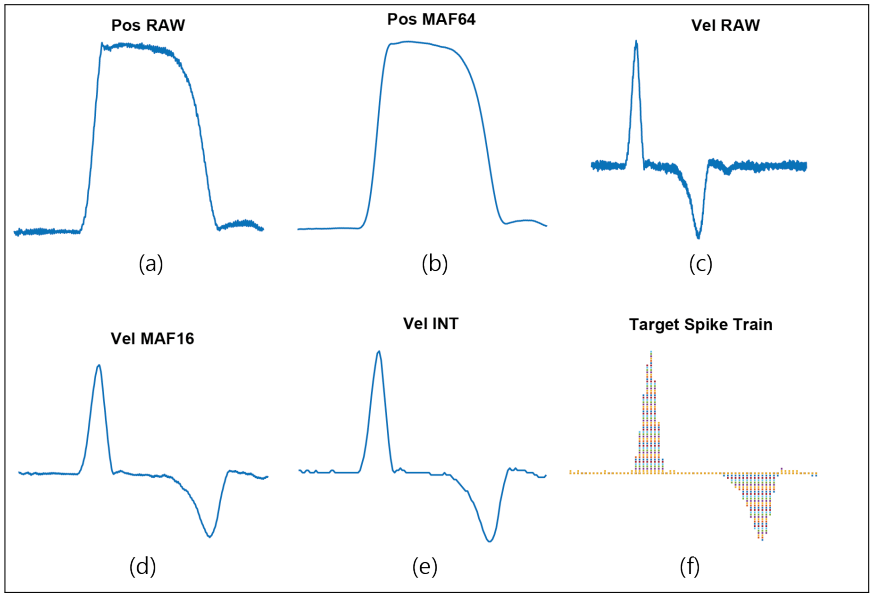


Figure 5.1: Target variable processing steps before training. (a) Raw handle position measurement. (b) Smoothed handle position, 60th-order moving average filter. (c) Handle velocity. (d) Smoothed handle velocity, 16th-order moving average filter. (e) Integer handle velocity in the range [-64,+63]. (f) Target spike train, downsampled by a factor of 10 for display purposes.

5.3.2 Spike detection

Spike detection is a key component of every neural interface based on spiking signals. Depending on the spike detection outcomes are based all the following steps of the signal processing chain. Therefore, poor accuracy during the spike detection phase would weaken the reliability of the whole chain. Differently from the neural decoder presented in this chapter, which operates with 1 kHz signals, the spike detector manages signals of an order of magnitude higher frequency, i.e. 10 kHz. Moreover, scaling towards an elevated number of channels, a compulsory point that will enable wider coverage and fine-grain resolution of the neural tissue, increases linearly the computational and memory requirements of the spike detection.

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. Relying on the extensive analysis carried out in Chapter 2, we chose the following deployment for the spike detection processing chain:

- Filter: 2nd order moving average difference filter;
- Spike emphasis: absolute value;
- Threshold: mean value;
- Time window: 0.82 ms;
- Refractory period: 1 ms;

Among the analyzed filters the 2nd-order moving average difference filter and both the 1st and 2nd-order difference filters were the more appealing because of the low computational and memory requirements. The choice went on the moving average difference filter because presented a steeper roll-off than the 1st-order difference filter and a flatter in-band response than the 2nd-order difference filter.
We did not observe any consistent accuracy improvement in using more sophisticated spike emphasis methods than the absolute value of the signal. Moreover, it only requires computing the 2's complement of the samples and enables using a single threshold (instead of two). In addition, it permits exploiting the mean value threshold, that we chose for this implementation, which is the least computationally expensive adaptive threshold estimation method.
We selected a time window of 0.82 ms for updating the threshold, since it appeared to be the best window considering the analysis summarized by Table 2.6 (a).
Finally, to match the frequency of the target variable, which is 1 kHz, we used spike bins computed over 1-ms windows as input of the spiking neural network, i.e. the spike count per channel within 1 ms. In addition, by setting the refractory period of each channel to 1 ms, we obtained the bins could exclusively be equal to zero or one.

5.3.3 Loihi Cuba neuron model

The basic processing element of the spiking neural network is the Loihi CUrrent BAsed Leaky Integrate and Fire (CUBA) Neuron, presented in [119]. The Loihi CUBA neuron extends the standard CUBA neuron [120] 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) [118], then extended and embedded as SLAYER 2.0 in the LAVA software

framework [121] used in this work. The Loihi Cuba Neuron requires two integrations, such as the Izhikevich neuron, however, its structure is simpler. 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:

$$\begin{aligned} S(t) &= \sum ws(t-1) \\ i(t) &= \alpha i(t-1) + S(t) \\ v(t) &= \beta v(t-1) + i(t) \\ s(t) &= v(t) > \theta \\ v(t) &= v(t)(1 - s(t)) \end{aligned} \tag{5.1}$$

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 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.

5.3.4 Spiking neural network

Spiking neural networks in LAVA [121] can use convolutional, dense (or fully-connected), 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 spiking neural network used in this work is made of two dense layers of 256 and 128 Loihi Cuba neurons whose parameters are shown in Table 5.2.

We tried to find the simplest model that could compete by the accuracy performance point of view with the other works in literature when tested on the selected dataset. We trained

Threshold	Current Decay	Voltage Decay
0.1	1.0	0.1

Table 5.2: Loihi Cuba neuron parameters

our model using 70% of the trials, then we tested its accuracy on the remaining 30%. The parameters used for training are shown in Table 5.3.

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

Table 5.3: Spiking neural network 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.

The memory required by the model, which uses weights of 15 bits, is reported per layer, and in total, in Table5.4.

L1	L2	Total
45 kB	60 kB	105 kB

Table 5.4: Spiking neural network memory requirements, from left to right: first layer L1 memory, second layer L2 memory, total memory.

Supervised learning algorithms, when the object is a III-Generation neural network, provide a parameter set that maps input spike trains to target output spike trains. In this case, the input of the spiking neural network is the binned spiking activity over a window of 1 ms. Being the refractory period per channel set at 1 ms, the binned activity can either be zero or one, therefore it is directly used as the input of the spiking neural network without further processing.

On the other hand, the desired output is a number, i.e. the target velocity. It is thus needed 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 target number, to train the network.

Our proposed solution is to attribute to every neuron of the output layer a weight that could be either *+1* or *-1*. Since in this case the positive and the negative maximum values of the target velocity are similar, half of the output neurons contribute positively, whereas the other half negatively. The target velocity is obtained by adding the fired spikes together at every

new inference of the spiking neural network, taking into account if the spikes come from the positive or the negative groups. For this purpose, the handle velocity was cast to integer, and its dynamic has been constrained in the range [-64,+63] as shown in Fig.5.1 (e), 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.5.1 (f) shows the target raster plot used to train the SNN, downsampled by a x10 factor for display purposes.

5.3.5 Spike sparsity

Spiking signals are characterized by being either active or inactive. This characteristic makes spiking neural networks event-driven systems. Figure 5.2 shows the timing raster plot of a random entry of the dataset, where on the x-axis is represented the time, and on the y-axis the detected spikes per channel. The red line is the sum of spikes across all the channels. It is observable how the number of concurrent spikes is limited, and how the spikes are concentrated before the handle velocity variation, shown in blue in Figure 5.2. The sparse activation condition is valid for both the output of the spike detection pipeline (which is the input of the spiking neural network), as well as for the connections among the layers of the spiking neural network, and its outputs. Taking advantage of the nature of these networks allows avoiding pointless waste of power, such as reading the entire weight memory when there is not any spiking activity on the input of the layers, or hardly any.

The layers of the spiking neural network considered in this chapter are fully-connected. Therefore, all the neurons of the same layer share the same set of inputs. That being the case, during the computation of the synaptic current, the spike memory is read again and again, once per each neuron of the layer. To avoid reading the spikes (and the weights) of the inactive inputs of the neural network layer, and to avoid computing the sums as well, is introduced a stack that stores the pointers to the active set of inputs (spikes and weights are stored in groups of four). The stack depth is equal to a quarter of the number of inputs of the layer (thus, 32 in the first layer and 64 in the second layer), whereas its word width is the base two logarithm of its depth (5 bits in the first layer and 6 bits in the second layer). At each synaptic current computation are read only the entries pointed by the stack, instead of reading the entire set of

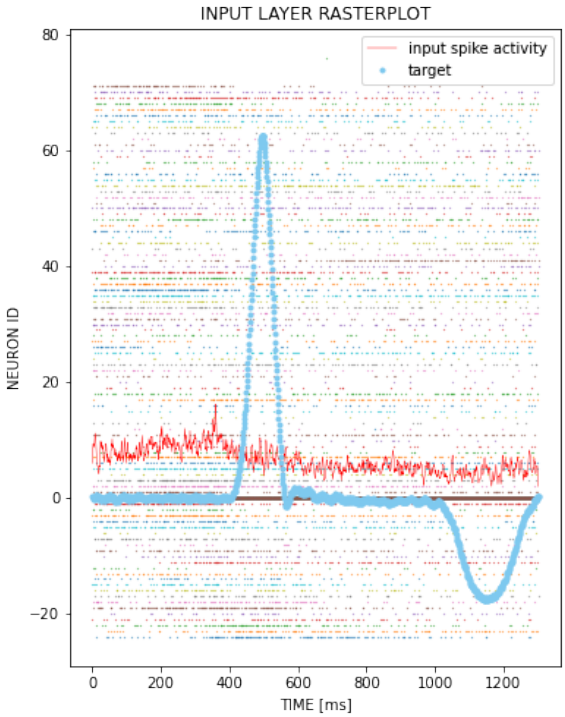


Figure 5.2: Measured spiking activity at the input of the spiking neural network

spikes and synaptic weights. This simple architectural expedient permits avoiding thousands of memory reads and sums per second as shown later in Section 5.5.2, as well as reducing the synaptic currents computations time by as many clock cycles.

5.4 System architecture

The hardware architecture, depicted in Fig.5.3, comprises the ARM-based Processing System (PS) embedded in the Zynq-family devices, two main hardware modules mapped in the programmable logic: a) a spike detector and b) a spiking neural network, and three AXI interfaces, used to 1) stream the broadband recording samples in the PL; 2) output the decoder inference; 3) import the synaptic weights in the PL. The spike detector is composed of three modules, used to filter the raw samples, detect the spikes, and compute the binned spiking activity, i.e. the number of spikes detected per millisecond on each channel. As regards as the spiking neural network, it consists of two dense layers cascaded with a spike-to-number converter, which translates the output spike train of the last layer into a number, i.e. the decoded target variable.

Moreover, a multi-electrode array of 128 channels is considered as input of the system, even though the datasets used to assess the accuracy use 96 channels.

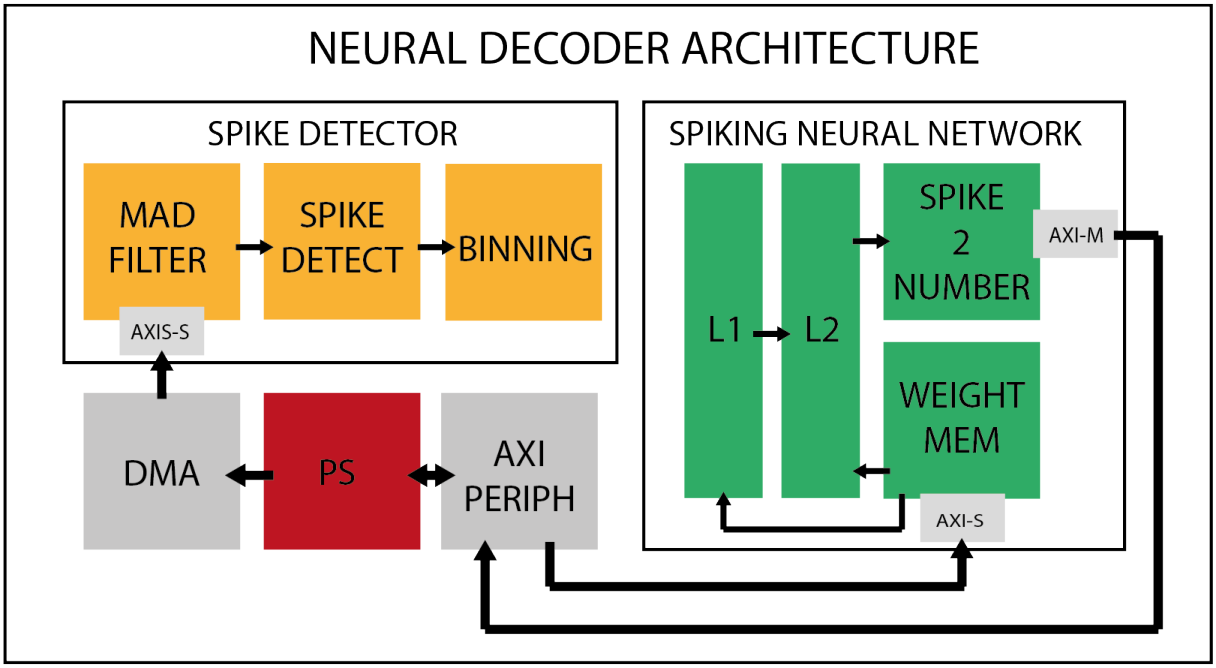


Figure 5.3: Neural interface architecture: the core of the neural decoder is composed of two cascaded modules: a spike detector and a spiking neural network. The former extracts the binned spiking activity directly from the raw neural signal by means of a Moving Average Difference (MAD) filter, a spike detector, and a binning module. The spiking neural network is composed of two dense layers, a weight memory, and a spike-to-number converter, that translates the output spike train generated by the second layer into the target decoding variable. The spike detector and the spiking neural network are connected to the Processing System (PS) through three AXI interfaces: 1) an AXI-stream interface is used to stream the neural samples in the programmable logic; 2) an AXI-lite interface is controlled from the PS to set up the neural decoder; 3) an AXI-lite interface is used to output the decoding results.

5.4.1 Spike detection and spike binning

The spike detection and the spike binning tasks are demanded 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 threshold is compared with the signal to verify the spike condition and updated. Finally, the spike binning module counts the detected spikes. A valid spike bin is forwarded to output every millisecond. The whole signal processing pipeline is shown in Fig.5.4.

Figure 5.4: The input broadband recording (1) is processed by five pipelined modules. The Moving Average Difference (MAD) filter (2) removes the low-frequency components; 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).

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 on the left of Fig.5.5.

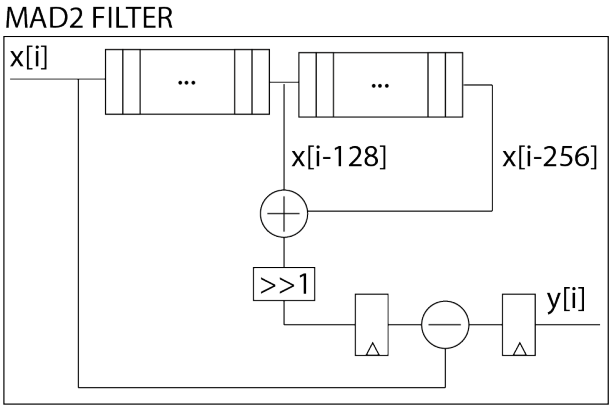


Figure 5.5: Second-order Moving Average Difference (MAD) filter

Spike emphasis

The filtered signal is rectified by means of 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 on the right of Fig.5.6.

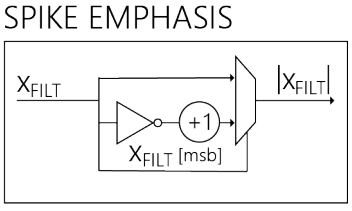


Figure 5.6: Absolute-value-based spike emphasis module

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, i.e. 128 entries. The BRAM memory is read when a new sample needs to be compared with the channel's threshold. Concurrently, when a new sample incomes, the future threshold of the respective channel from which the sample arrives 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 over 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 to update the value stored in the threshold memory, that 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.5.7.

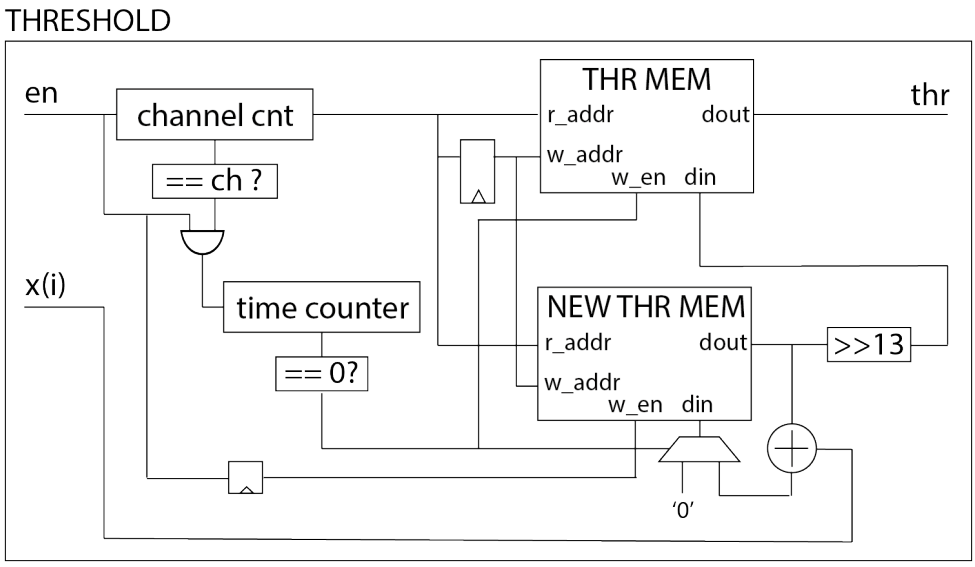


Figure 5.7: Mean-value-based spike threshold module

Spike detector

The spike detector compares the emphasized sample value with the respective channel threshold. In case the sample value exceeds the threshold and the last spike on the same channel happened more than 1 ms before, the spike is propagated to the output of the module. The spike detector module implements a refractory period rule that limits the spike rate to one spike per ms per channel.
A BRAM memory is used to keep track of the refractory period of each channel:

- When a new sample arrives, the refractory period memory is read, if the value is zero and the sample exceeds the threshold value the spike is forwarded to the output and the refractory memory entry is updated to one;

- When a new sample arrives, if the refractory period associated is greater than zero, its refractory period counter is incremented by one, and the spike (if present) is not propagated. The refractory period is incremented until it reset itself by overflow, after 8,192 sampling cycles.

The architecture of the detection module is shown in Fig.5.8.

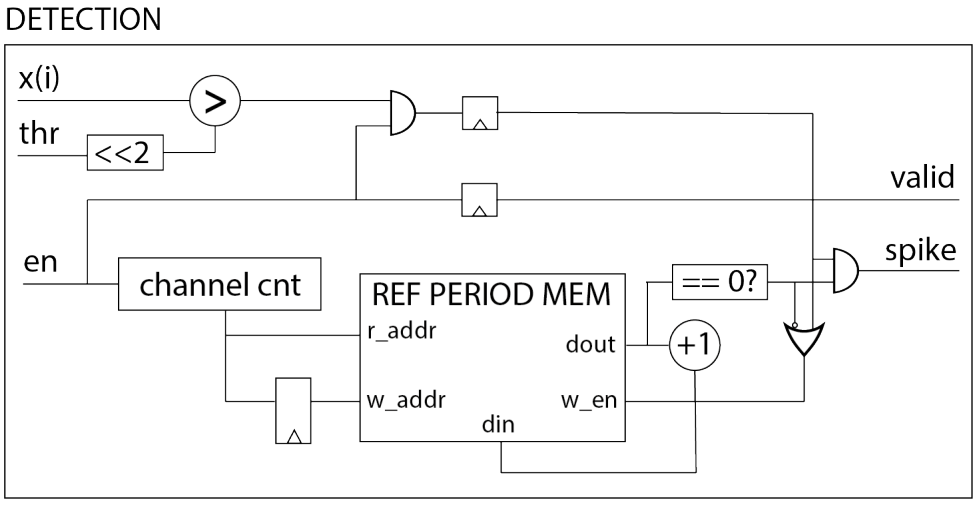


Figure 5.8: Spike detection module

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 spike 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, and the bin values reset. To stream out the bins is used a counter.

Since the spiking neural network processes four inputs at a time, a 4-bit shift register operates the serial to parallel conversion on the output stream of the spike binning module. Once four bins are collected, the bin set is stored in the spike memory of the first neural network's layer. Moreover, an additional flip flop and an *OR* gate are used to implement an *orator*, a structure that evaluates if any of the four bins is active, a sequential structure useful in the case the parallelism of the neural network is increased from 4 to 8, 16 or more. The *or-reduced* value of the bins is called *active set* in the design and it is used for low-power purposes during the spiking neural network inferences. The architecture of the spike binning module and the architecture of the serial to parallel converter are shown in Fig.5.9.

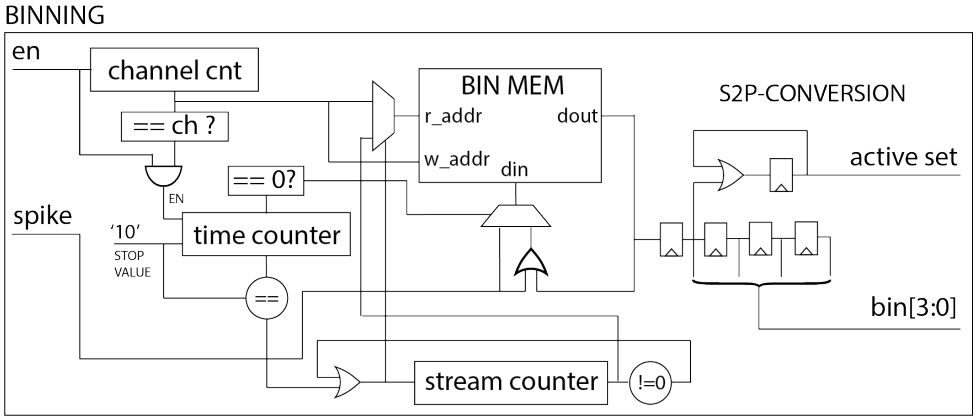


Figure 5.9: Spike binning module and serial to parallel (S2P) converter

5.4.2 Spiking neural network decoder

The spike decoder is composed of a spiking neural network and a spike-to-number converter. The spiking neural network contains two layers of 128 and 256 Loihi Cuba neurons. Each layer processes its neurons one by one. Every neuron state update requires computing the synaptic current, i.e. accumulating the weights of its active synapses, then the neuron can be integrated twice and its spike propagated to the output. Fig.5.10 shows the architecture of the layer module. Each layer module contains a weight and a spike memory, where respectively the spikes and the synaptic weights are stored in groups of four. Moreover, the spike memory is implemented as a double buffer to avoid stalling the pipeline. The synaptic current is computed by convolving spikes and weights; a stack is used to store the addresses of the active sets of synapses to avoid reading a row of four weights where no one is contributing to the synaptic current value. Once the synaptic current is ready, the corresponding Loihi neuron is integrated by using two integrator modules. The neurons' state variables are stored in two FIFOs with an entry per neuron that feedback their values to the inputs of the two integrators. The second integrator generates the output spike. The spikes are forwarded in output in groups of four to respect the spike memory structure of the next layer. The serial to parallel conversion is implemented by using four serial-cascaded flip-flops as in the spike binning module, the *active set* signal is computed as well and it is used to initialize the stack of the following layer.

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, each entry of the weight memory is composed of 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 the *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.

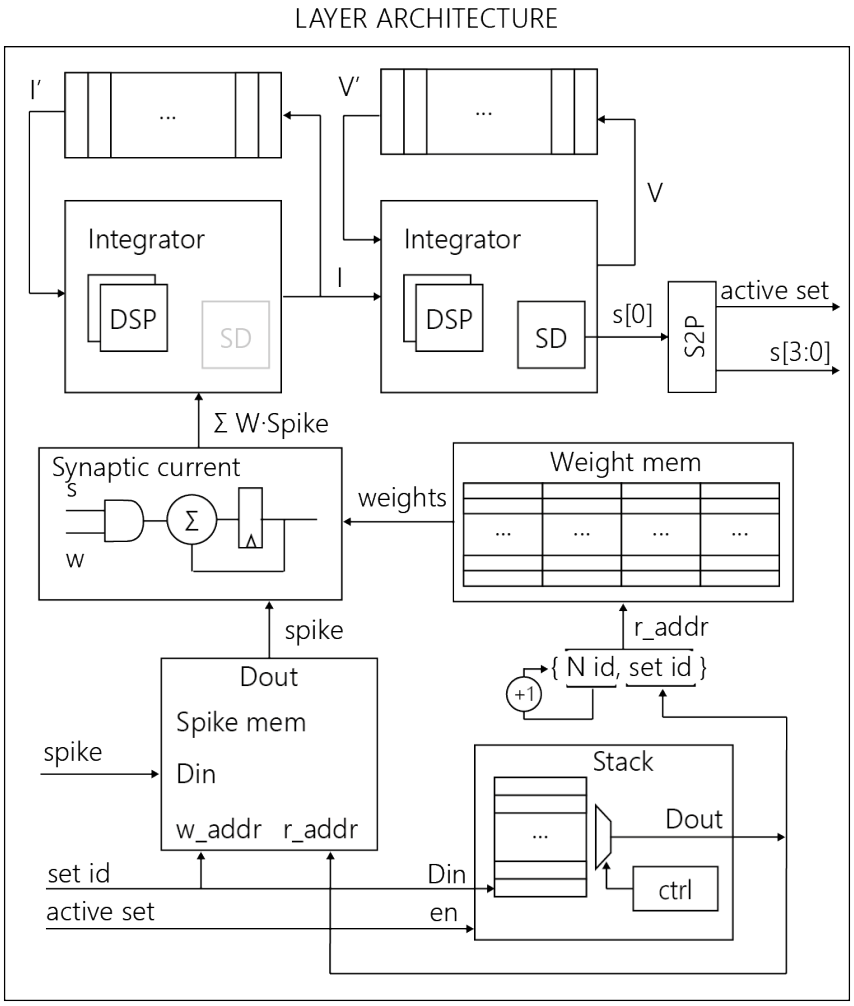


Figure 5.10: Spiking neural network's layer architecture: the layer computes the synaptic current neuron-by-neuron, the input spikes and the synaptic weights are read from the spike memory and the weight memory. The spike memory is written from the previous layer (if present) or by the spike detector, the weight memory is initialized at system start-up by the processing system, whereas the synaptic current computation is demanded to the synaptic current module. The spike and weight memories indexing are performed through a stack that stores the indexes of the active inputs only, by doing so, all the inactive set of inputs are skipped during the synaptic current computation phase. The stack is initialized by relying on the active set signal generated at the previous processing stage, i.e. either in the previous neural network layer or during the spike detection step. Once the current is ready, the Loihi Cuba Neuron is integrated by means of two identical cascaded integrator modules. 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, it is generated an or-reduced signal called active set which states if any of the spikes of the 4-bit group is active.

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 by means of two additional DSPs. The accumulation iterates until the synaptic current is computed adding four weights at every cycle. The hardware architecture of the module is shown in Figure 5.11

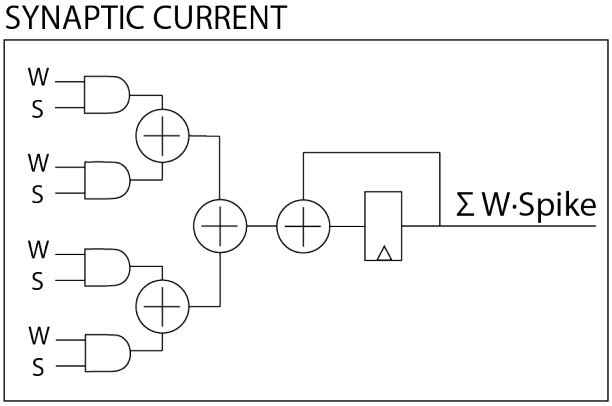


Figure 5.11: Synaptic current computation module

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 5.1. 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, with an enable to activate or deactivate the spike evaluation.

The two integrators are cascade-connected, the external input of the Current integrator is the convolved spiking activity, and 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 spiking neural network layer.

The Loihi Cuba model of the Lava library [121] 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 [121], 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 being added. The external input can either be the *synaptic current*, in the case of the first integration, or the *current* in the case of the second integration. When 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. The output of the comparison is also the spike, indeed. The hardware architecture of the module is shown in Figure 5.12.

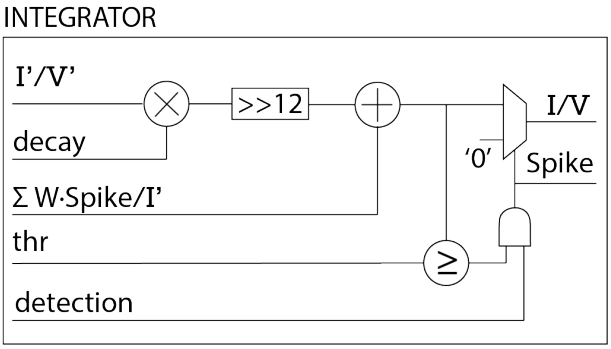


Figure 5.12: Loihi cuba neuron model integrator module

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 retrieve 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. Meanwhile, 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. The hardware architecture of the stack is shown in Figure 5.13.

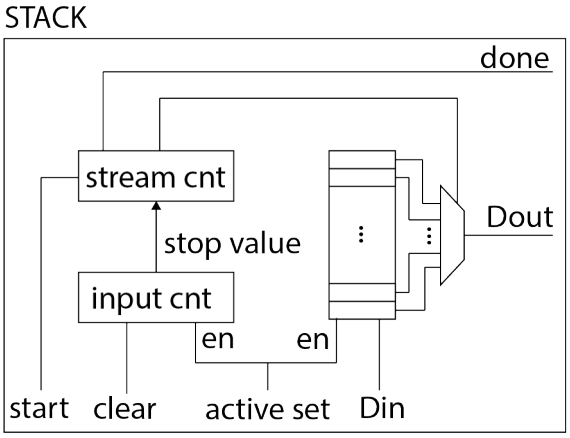


Figure 5.13: Spike sparsity stack architecture

Spike2Number converter

The output of the last SNN's layer requires being 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 fired from 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 can be 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. The hardware architecture of the spike-to-number converter is shown in Figure 5.14.

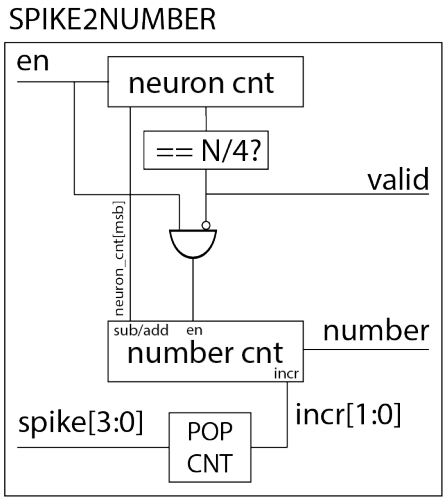


Figure 5.14: Spike to number converter architecture

5.5 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.

5.5.1 Accuracy

Two spiking neural network models were trained on the benchmark dataset. The first model was a single SNN dense layer of 128 units and 96 inputs. As tempting as this model was, because of its low computational and memory requirements, it performed poorly compared to the State of the Art, therefore we tried training a more complex model constituted by two dense layers. The former layer is made of 256 units and the latter of 128 units. Table 5.5 shows the parameter of the two spiking neural networks: number of layers, number of units, number of parameters, memory requirements and the achieved decoding accuracy measured with the Pearson correlation coefficient described by Eq.5.2 for the Dataset N and L [8].

$$CC(A,B) = \frac{1}{N-1}\sum_{i=1}^{N}\frac{A_i-\mu_A}{\sigma_A}\frac{B_i-\mu_B}{\sigma_B} \tag{5.2}$$

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, how-

Layers	Units	Parameters	Memory	CC N [8]	CC L [8]
1	128	12,288	22.5 kB	0.79	0.66
2	384	57,344	105 kB	0.83	0.78

Table 5.5: SNN models decoding accuracy

ever, it requires instantiating three times more units, 4.7 times more parameters, and thus, 4.7 times more memory.

The velocity inferred by the 2-layer spiking neural network model and the target handle velocity are plotted in Figure 5.15 for a random entry of the test set and shows graphically how the spike decoder is capable of tracking the target variable inferring its value from the neural recording. On the background of Figure 5.15 is shown a temporal raster plot, where on the x-axis is represented the time in milliseconds, and on the y-axis the spiking neural network' last layer neurons. The raster plot's points represent the output spike trains of the neural network. The output velocity corresponds to the sum of the spikes at each point in time.

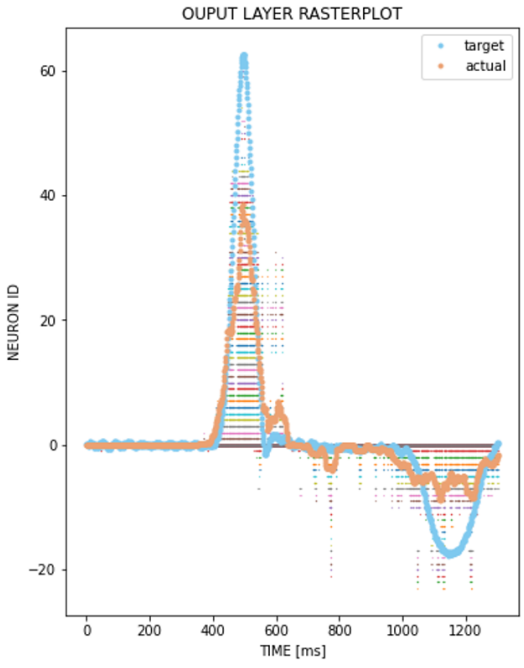


Figure 5.15: Spike decoder output behavior in the test set

5.5.2 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.

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. 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 and is clocked at 2 MHz. The resource requirements of the digital system are shown in Table 5.6. The first row indicates the overall resource usage by the system in terms of spike detector, spiking neural network, axi interfaces, and DMA. The second and third rows refer to the individual requirements of the spike detector and the spiking neural network.

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, being each SRLC32E LUTRAM primitive a 32-bit shift register: *32 bits x 80 primitives = 2,560 bits*. Note that *10 bits sample x 128 channels x 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 x 128 channels x 8 taps / 32 bits per SRLC32E primitive = 320*. The chosen filter permits saving 75% of the LUTRAMs compared to widely used 4th-order IIR filters.

The 14 DSPs instantiated are entirely used to speed up the SNN module. Being the SNN composed of 2 layers, each layer 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 each 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. Considering the weights memories are *128 inputs x 256 units x 15 bits* for the first layer and *256 inputs x 128 units x 15 bits* for the second layer, the net memory required by the neural network is 960 kb. Being the BRAMs 36 kb each, *ceil(960/36) = 27* BRAMs should be instanced. However, because of the suboptimal word size of 60 bits (4 weights), the number rises to 32 BRAMs. The spike detector requires a single BRAM, half of which it is used by the detector module to store the 128 refractory periods, whereas the other half is used by the threshold module to store the 128 thresholds. The last 1.5 BRAM is used by the *AXI_DMA*, instanced to provide the broadband recording stream during the test phase.

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

Table 5.6: Hardware utilization: the first row shows the overall resource utilization of the system; the second and third rows show respectively the resource usage of the spike detection module and the spiking neural network.

Spike sparsity savings

Spike sparsity has been exploited by instancing on each layer of the spiking neural network a stack to store the addresses pointing to the active input spike 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 [8]. The savings are reported in Tables 5.7 and 5.8, respectively for dataset *N* (16 minutes and 43 seconds long) and *L* (11 minutes and 49 seconds long) [8]. The saving is reported for each layer and for both.

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%. 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. 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

	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

Table 5.7: Operations savings for dataset *N* [8]

	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

Table 5.8: Operations savings for dataset *L* [8]

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 spiking neural network 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 or more complex neural networks. The computational and power savings are well balanced considering the resource requirements of the stacks reported in Table 5.9. Note that, the stacks are the only elements that permit the design to be aware of spike sparsity.

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

Table 5.9: Stacks' resources requirements

5.6 Comparison with the State of the Art

As far as we know, the presented neural interface is the first work that directly uses intracortical recorded spikes jointly to a spiking neural network for decoding the neural signal, either implemented on FPGA or ASIC. Table 5.10 reports software decoders tested on the same benchmark dataset used in this work [8]. Table 5.10 shows the type of decoder used in each study, the number of layers and units, the number of parameters, and the total required memory when available. Since the dataset comprises two different recordings, the accuracy, measured

in terms of correlation (CC), found on the monkey N test set is shown in the last but one column, whereas the accuracy measured on the test set of monkey L is shown in the last column.

On lines 2, 3, and 4 is shown the accuracy of three deep learning decoders [93], a Quasi-

Work	Type	Layers	Units	Parameters	Memory	CC N [8]	CC L [8]
This work	SNN	2	384	0.057M	105 kB	0.83	0.78
Ahmadi [93]	QRNN	1	400		-	0.84	0.73
Ahmadi [93]	GRU	1	200		-	-	0.78
Ahmadi [93]	LSTM	1	150		-	0.87	-
Yang [122]	RNN	2	128	0.42M	1.6 MB*	0.91	-
Yang [122]	GRU	2	128	1.19M	4.5 MB*	0.89	-
Yang [122]	LSTM	2	256	1.56M	5.9 MB*	0.91	-

* Deduced considering 32-bit parameters

Table 5.10: State of the art neural decoders comparison

Recurrent Neural Network (QRNN), a Gated Recurrent Unit (GRU), and a Long Short-Term Memory (LSTM). The best decoder found in [93] was a QRNN of 400 units, that performed similarly to the proposed model. On dataset N their decoder achieved 0.84 CC, whereas our spiking neural network got 0.83. On dataset L, the SNN 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 [93] is the one of the GRU model, which tied to the SNN; in the case of dataset N, the highest correlation was obtained using an LSTM model, which achieved 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 5.10 show the results obtained from other three deep learning decoders on dataset N only. In [122] an RNN, a GRU and an LSTM were implemented using the PyTorch library. The decoders achieved outstanding correlation results of 0.91, 0.89, and 0.91 respectively. In [122] was shown the number of parameters utilized for each model, making possible a memory requirements comparison with the SNN. 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 used in this chapter. The GRU model requires using 20.9 more parameters than the SNN, the LSTM necessitates 27.4 more parameters than the SNN. Moreover, it is possible to assume each parameter is a 32-bit floating point number, as commonly is in the PyTorch library, to

infer the memory usage of the models in [122]. They respectively require 1.6, 4.5, and 5.9 MB, whereas the SNN occupies only 105 kB. Therefore, the memory requirements for using the SNN are respectively 6.4%, 2.3 %, and 1.7% of the ones required by the models in [122].

5.7 Conclusion

We have presented a resource-power efficient intracortical neural interface system embedding a multiplier-less spike detection pipeline and a spike-sparsity-aware spiking neural network decoder. The spike detector is equipped with filter, dynamic threshold updates, refractory period spike burst limitation, and spike binning features and ensures reliable spike detection by only employing five additions per channel and zero multiplications. The spiking neural network 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 spiking neural network models on more datasets, their low-power characteristic makes them intrinsically attractive, especially in a field where the power budget is so constrained, as for neural interfaces. From the resource requirements point of view would be appealing to study the effect given by a reduction of the model's weight size on the decoding accuracy.

Chapter 6

Conclusions

The Thesis focused on providing a road map for next generation neural interface design based on available multielectrode arrays and high-density multielectrode arrays embedding thousands of recording sites. In particular, it provided an extensive exploration of spike detection algorithms where both the accuracy and computational complexity of the methods were analyzed, respectively in Sections 2.2 and 2.3, taking into consideration all the steps that make up the signal processing chain, and suggesting a multiplier-less detection pipeline, embedding a moving average difference filtering stage, an absolute value based spike emphasis algorithm, a dynamic mean value threshold update, and a refractory period control, that achieved comparable results with the State of the Art when tested on synthetic datasets, and when embedded in a neural activity decoder and tested on a delayed reach-to-grasp neural decoding task.
Moreover, in Chapter 3 was presented a spike sorting system hosted by a Xilinx All-Programmable System-On-Chip device, capable of sorting online the neural activity of about 5,500 recording sites with a latency of 2.3 ms, using the reconfigurable blocks, along with a new online method for minimizing poor spike set choice during the evaluation of the templates by exploiting the ARM-based processing system.
Furthermore, in Chapter 4, with the idea of enabling bio-realistic real-time emulation of spiking neural networks with arbitrarily interconnected neurons, was proposed a fully-connected spiking neural network of Izhikevich neurons deployed on a Xilinx All-Programmable System-On-Chip device and a new method to exploit the physiological spike propagation delay of biological neurons to take advantage of the off-chip ram to store the synaptic weights with-

out affecting real-time performances, achieving the possibility of emulating in real-time up to 3,098 neurons and 9.6e6 synaptic interconnections. Moreover, the Izhikevich model accuracy was assessed at both the single cell and network level when the model is integrated using fixed-point arithmetic for several data formats and a custom fixed-point point representation built respecting the Xilinx DSP48E1 data format. The custom fixed-point data mapping permits saving 39% of memory and achieves negligible behavioral differences with the floating-point model.

Finally, the feasibility of exploiting spiking neural networks as neural signal decoders was demonstrated in Chapter 5 by relying on the above-mentioned spike detection pipeline and a spiking neural network composed of two fully connected layers of Loihi Cuba neurons. Biological spikes are intrinsically in the proper format to be processed by SNNs, i.e. it is not necessary to convert continuous signals into spiking signals, as happens when SNNs are used in conjunction with EMGs or ECoGs. The spike decoder was prototyped on a Xilinx All-Programmable System-On-Chip and validated on a delayed reach-to-grasp task achieving comparable results with State of the Art neural decoders tested on the same dataset. However, its number of parameters was significantly lower, entailing inferior memory requirements as well as reduced computational complexity. In addition, when tested on real neural data the spiking neural network saved about the 90% of the computations, taking advantage of spike sparsity, proofing to be at the same time a better choice from the memory, power, and computational points of view.

Although this research provides new instruments and platforms for analyzing high-channel-count multielectrode arrays data and emulating spiking neural networks in real time, several challenges still need to be faced. Nowadays is extremely problematic to get access to datasets for neural decoding recorded using emerging CMOS MEAs and HDMEAs, whereas, new generation computing platforms would benefit from thoroughly testing over a multitude of neural decoding experiments. Furthermore, being power consumption a key aspect in the field of neural interfaces, it is crucial to study low-power solutions for both ASIC implementations and low-power FPGA prototyping to foster the development of viable solutions and truly enable quality of life improvements for people affected from neurological diseases.

Appendices

Appendix A

Target device: All Programmable System on Chip

All Programmable System on Chip (APSoC) devices are promising instruments for facing extremely parallel and computationally intensive real-time processing of neural data, real-time low-latency emulation of large-scale brain models, and real-time low latency execution of artificial neural network models.

All Programmable System on Chip (APSoC) devices from The Zynq-7000 family embed on the same chip a dual-core Cortex-A9 based processing system (PS) and a Xilinx Programmable Logic (PL) [123]. The PS includes on-chip memory, single and double precision Vector Floating Point Unit and a DDR3 external memory interface, enabling the execution of moderately compute-intense processes, as well as providing support for housekeeping tasks. Moreover, the PS is provided with a rich set of peripheral connectivity interfaces such as two 10/100/1000 tri-speed Ethernet MAC peripherals, two USB 2.0 peripherals, two CAN bus interfaces, two SD/SDIO controllers, two full-duplex SPI ports, two high-speed UARTs, two master and slave I2C interfaces, and four 32-bit banks General Purpose Inputs Outputs (GPIOs). Furthermore, the PS is provided with ARM AMBA AXI based high-bandwidth connectivity links for communicate with the PL.

The Xilinx Programmable Logic comprises Configurable Logic Blocks (CLB) embedding Look-Up Tables (LUT), flip-flops and cascadeable adders, 36 Kb Block RAMs, and Digital Signal Processor (DSP) blocks. The number of processing and memory elements vary depending on

the chosen device, however, the DSP blocks permit speeding up the most demanding portion of the algorithm that requires hardware support, taking advantage of the hardwired multiply-and-accumulate (MAC) blocks embedded in the DSPs; the look-up tables permit controlling the data-flow and solving boolean conditions; the adaptability of the BRAM consents adjusting the memory port widths, fostering the modularity of the design and permitting the re-use of the architecture for several experiments addressing different MEA and HDMEA recording systems.

Bibliography

[1] *3Brain Single-Well High-density microelectrode arrays with large sensing areas*. URL: https://www.3brain.com/products/single-well/hd-mea.

[2] Elon Musk et al. "An integrated brain-machine interface platform with thousands of channels". In: *Journal of medical Internet research* 21.10 (2019), e16194.

[3] Nick Steinmetz, Cagatay Aydin, Anna Lebedeva, Michael Okun, Marius Pachitariu, Marius Bauza, Maxime Beau, Jai Bhagat, Claudia Böhm, Martijn Broux, Susu Chen, Jennifer Colonell, Richard Gardner, Bill Karsh, Dimitar Kostadinov, Carolina Lopez, Junchol Park, Jan Putzeys, Britton Sauerbrei, and Timothy Harris. "Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings". In: (Nov. 2020). DOI: 10.1101/2020.10.27.358291.

[4] *Utah Array*. https://blackrockneurotech.com/.

[5] *Xilinx APSoCs*. URL: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[6] Gianluca Leone, Luigi Raffo, and Paolo Meloni. "ZyON: Enabling Spike Sorting on APSoC-Based Signal Processors for High-Density Microelectrode Arrays". In: *IEEE Access* 8 (2020), pp. 218145–218160. DOI: 10.1109/ACCESS.2020.3042034.

[7] Gianluca Leone, Luigi Raffo, and Paolo Meloni. "A Bandwidth-Efficient Emulator of Biologically-Relevant Spiking Neural Networks on FPGA". In: *IEEE Access* 10 (2022), pp. 76780–76793. DOI: 10.1109/ACCESS.2022.3192826.

[8] Thomas Brochier, Lyuba Zehl, Yaoyao Hao, Margaux Duret, Julia Sprenger, Michael Denker, Sonja Grün, and Alexa Riehle. "Massively parallel recordings in macaque motor cortex during an instructed delayed reach-to-grasp task". In: *Scientific data* 5.1 (2018), pp. 1–23.

[9] Vadim S Polikov, Patrick A Tresco, and William M Reichert. "Response of brain tissue to chronically implanted neural electrodes". In: *Journal of neuroscience methods* 148.1 (2005), pp. 1–18.

[10] Sarah Gibson, Jack W Judy, and Dejan Marković. "Spike sorting: The first step in decoding the brain: The first step in decoding the brain". In: *IEEE Signal processing magazine* 29.1 (2011), pp. 124–143.

[11] Jose M Carmena, Mikhail A Lebedev, Roy E Crist, Joseph E O'Doherty, David M Santucci, Dragan F Dimitrov, Parag G Patil, Craig S Henriquez, and Miguel A L Nicolelis. "Learning to control a brain–machine interface for reaching and grasping by primates". In: *PLoS biology* 1.2 (2003), e42.

[12] Christopher Heelan, Jihun Lee, Ronan O'Shea, Laurie Lynch, David M Brandman, Wilson Truccolo, and Arto V Nurmikko. "Decoding speech from spike-based neural population recordings in secondary auditory cortex of non-human primates". In: *Communications biology* 2.1 (2019), pp. 1–12.

[13] Reid R Harrison. "The design of integrated circuits to observe brain activity". In: *Proceedings of the IEEE* 96.7 (2008), pp. 1203–1216.

[14] Arto Nurmikko. "Challenges for large-scale cortical interfaces". In: *Neuron* 108.2 (2020), pp. 259–269.

[15] Hernan Gonzalo Rey, Carlos Pedreira, and Rodrigo Quian Quiroga. "Past, present and future of spike sorting techniques". In: *Brain research bulletin* 119 (2015), pp. 106–117.

[16] Zheng Zhang and Timothy G Constandinou. "Adaptive spike detection and hardware optimization towards autonomous, high-channel-count BMIs". In: *Journal of Neuroscience Methods* 354 (2021), p. 109103.

[17] Zheng Zhang, Oscar W Savolainen, and Timothy G Constandinou. “Algorithm and hardware considerations for real-time neural signal on-implant processing”. In: *Journal of Neural Engineering* 19.1 (2022), p. 016029.

[18] R. Quian Quiroga, Z. Nadasdy, and Y. Ben-Shaul. “Unsupervised Spike Detection and Sorting with Wavelets and Superparamagnetic Clustering”. In: *Neural Computation* 16.8 (2004), pp. 1661–1687. DOI: 10.1162/089976604774201631.

[19] Iyad Obeid and Patrick D Wolf. “Evaluation of spike-detection algorithms for a brain-machine interface application”. In: *IEEE Transactions on Biomedical Engineering* 51.6 (2004), pp. 905–911.

[20] J.F. Kaiser. “On a simple algorithm to calculate the ’energy’ of a signal”. In: *International Conference on Acoustics, Speech, and Signal Processing*. 1990, 381–384 vol.1. DOI: 10.1109/ICASSP.1990.115702.

[21] Zheng Zhang and Timothy G Constandinou. “Selecting an effective amplitude threshold for neural spike detection”. In: *bioRxiv* (2022).

[22] Emily R Oby, Sagi Perel, Patrick T Sadtler, Douglas A Ruff, Jessica L Mischel, David F Montez, Marlene R Cohen, Aaron P Batista, and Steven M Chase. “Extracellular voltage threshold settings can be tuned for optimal encoding of movement and stimulus parameters”. In: *Journal of neural engineering* 13.3 (2016), p. 036009.

[23] Simona Lodato and Paola Arlotta. “Generating neuronal diversity in the mammalian cerebral cortex”. In: *Annual review of cell and developmental biology* 31 (2015), p. 699.

[24] Andrea Stocco, Christian Lebiere, and John R Anderson. “Conditional routing of information to the cortex: A model of the basal ganglia’s role in cognitive coordination.” In: *Psychological review* 117.2 (2010), p. 541.

[25] Robert C Froemke and Yang Dan. “Spike-timing-dependent synaptic modification induced by natural spike trains”. In: *Nature* 416.6879 (2002), pp. 433–438.

[26] Hossein Kassiri, Sana Tonekaboni, M. Tariqus Salam, Nima Soltani, Karim Abdelhalim, Jose Luis Perez Velazquez, and Roman Genov. “Closed-Loop Neurostimulators: A Survey and A Seizure-Predicting Design Example for Intractable Epilepsy Treatment”. In:

IEEE Transactions on Biomedical Circuits and Systems 11.5 (2017), pp. 1026–1040. DOI: 10.1109/TBCAS.2017.2694638.

[27] James J Jun, Nicholas A Steinmetz, Joshua H Siegle, Daniel J Denman, Marius Bauza, Brian Barbarits, Albert K Lee, Costas A Anastassiou, Alexandru Andrei, Çağatay Aydın, et al. "Fully integrated silicon probes for high-density recording of neural activity". In: *Nature* 551.7679 (2017), pp. 232–236.

[28] Rodrigo Quian Quiroga. "Spike sorting". In: *Current Biology* 22.2 (2012), R45–R46.

[29] Yuning Yang, Sam Boling, and Andrew J. Mason. "A Hardware-Efficient Scalable Spike Sorting Neural Signal Processor Module for Implantable High-Channel-Count Brain Machine Interfaces". In: *IEEE Transactions on Biomedical Circuits and Systems* 11.4 (2017), pp. 743–754. DOI: 10.1109/TBCAS.2017.2679032.

[30] Daniel Valencia and Amirhossein Alimohammad. "A Real-Time Spike Sorting System Using Parallel OSort Clustering". In: *IEEE Transactions on Biomedical Circuits and Systems* 13.6 (2019), pp. 1700–1713. DOI: 10.1109/TBCAS.2019.2947618.

[31] Jinho Yi, Jiachen Xu, Ethan Chen, Maysamreza Chamanzar, and Vanessa Chen. "Multi-channel Many-Class Real-Time Neural Spike Sorting With Convolutional Neural Networks". In: *IEEE Open Journal of Circuits and Systems* 3 (2022), pp. 168–179.

[32] Giovanni Pietro Seu, Gian Nicola Angotzi, Fabio Boi, Luigi Raffo, Luca Berdondini, and Paolo Meloni. "Exploiting All Programmable SoCs in Neural Signal Analysis: A Closed-Loop Control for Large-Scale CMOS Multielectrode Arrays". In: *IEEE Transactions on Biomedical Circuits and Systems* 12.4 (2018), pp. 839–850. DOI: 10.1109/TBCAS.2018.2830659.

[33] Jan Müller, Douglas J Bakkum, and Andreas Hierlemann. "Sub-millisecond closed-loop feedback stimulation between arbitrary sets of individual neurons". In: *Frontiers in neural circuits* 6 (2013), p. 121.

[34] S. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

[35] Teuvo Kohonen. “Self-organized formation of topologically correct feature maps”. In: *Biological cybernetics* 43.1 (1982), pp. 59–69.

[36] Emilia Biffi, Diego Ghezzi, Alessandra Pedrocchi, and Giancarlo Ferrigno. “Development and validation of a spike detection and classification algorithm aimed at implementation on hardware devices”. In: *Computational intelligence and neuroscience* 2010 (2010).

[37] Biao Sun, Hui Feng, Kefan Chen, and Xinshan Zhu. “A Deep Learning Framework of Quantized Compressed Sensing for Wireless Neural Recording”. In: *IEEE Access* 4 (2016), pp. 5169–5178. DOI: 10.1109/ACCESS.2016.2604397.

[38] Pierre Yger, Giulia LB Spampinato, Elric Esposito, Baptiste Lefebvre, Stéphane Deny, Christophe Gardella, Marcel Stimberg, Florian Jetter, Guenther Zeck, Serge Picaud, et al. “A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo”. In: *Elife* 7 (2018), e34518.

[39] Jongkil Park, Gookhwa Kim, and Sang-Don Jung. “A 128-Channel FPGA-Based Real-Time Spike-Sorting Bidirectional Closed-Loop Neural Interface System”. In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 25.12 (2017), pp. 2227–2238. DOI: 10.1109/TNSRE.2017.2697415.

[40] Anh Tuan Do, Seyed Mohammad Ali Zeinolabedin, Dongsuk Jeon, Dennis Sylvester, and Tony Tae-Hyoung Kim. “An Area-Efficient 128-Channel Spike Sorting Processor for Real-Time Neural Recording With 0.175 μ W/Channel in 65-nm CMOS”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.1 (2019), pp. 126–137. DOI: 10.1109/TVLSI.2018.2875934.

[41] Wing-kin Tam and Matthew F Nolan. “pyNeurode: a real-time neural signal processing framework”. In: *bioRxiv* (2022).

[42] Maryam Saeed and Awais M. Kamboh. “Hardware architecture for on-chip unsupervised online neural spike sorting”. In: *2013 6th International IEEE/EMBS Conference on Neural Engineering (NER)*. 2013, pp. 1319–1322. DOI: 10.1109/NER.2013.6696184.

[43] Vaibhav Karkare, Sarah Gibson, and Dejan Marković. "A 75-µW, 16-Channel Neural Spike-Sorting Processor With Unsupervised Clustering". In: *IEEE Journal of Solid-State Circuits* 48.9 (2013), pp. 2230–2238. DOI: 10.1109/JSSC.2013.2264616.

[44] Awais M. Kamboh and Andrew J. Mason. "Computationally Efficient Neural Feature Extraction for Spike Sorting in Implantable High-Density Recording Systems". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 21.1 (2013), pp. 1–9. DOI: 10.1109/TNSRE.2012.2211036.

[45] Maryam Saeed, Amir Ali Khan, and Awais Mehmood Kamboh. "Comparison of Classifier Architectures for Online Neural Spike Sorting". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 25.4 (2017), pp. 334–344. DOI: 10.1109/TNSRE.2016.2641499.

[46] Sivylla E Paraskevopoulou, Di Wu, Amir Eftekhar, and Timothy G Constandinou. "Hierarchical Adaptive Means (HAM) clustering for hardware-efficient, unsupervised and real-time spike sorting". In: *Journal of neuroscience methods* 235 (2014), pp. 145–156.

[47] Sivylla E Paraskevopoulou, Deren Y Barsakcioglu, Mohammed R Saberi, Amir Eftekhar, and Timothy G Constandinou. "Feature extraction using first and second derivative extrema (FSDE) for real-time and hardware-efficient spike sorting". In: *Journal of neuroscience methods* 215.1 (2013), pp. 29–37.

[48] Laszlo Schäffer, Zoltan Nagy, Zoltan Kincses, Richard Fiáth, and Istvan Ulbert. "Spatial information based OSort for real-time spike sorting using FPGA". In: *IEEE Transactions on Biomedical Engineering* 68.1 (2020), pp. 99–108.

[49] Carlos Pedreira, Juan Martinez, Matias J Ison, and Rodrigo Quian Quiroga. "How many neurons can we see with current spike sorting algorithms?" In: *Journal of neuroscience methods* 211.1 (2012), pp. 58–65.

[50] Carl Gold, Darrell A Henze, Christof Koch, and Gyorgy Buzsaki. "On the origin of the extracellular action potential waveform: a modeling study". In: *Journal of neurophysiology* 95.5 (2006), pp. 3113–3128.

[51] Sara Zaher, Davide Lonardoni, Fabio Boi, Giovanni Pietro Seu, Gian Nicola Angotzi, Paolo Meloni, and Luca Berdondini. "A Closed-Loop System Processing High-Density Electrical Recordings and Visual Stimuli to Study Retinal Circuits Properties". In: *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*. 2019, pp. 652–656. DOI: 10.1109/NER.2019.8716913.

[52] "ynq-7000 SoC Technical Reference Manual". In: 22 (), p. 658. URL: https://www.xilinx.com/support/documentation/userguides/ug585-Zynq-7000-TRM.pdf.

[53] "K-means++ Clustering Code". In: (). URL: https://rosettacode.org/wiki/K-means%5C%2B%5C%2Bclustering#C.

[54] David Arthur and Sergei Vassilvitskii. *k-means++: The advantages of careful seeding*. Tech. rep. Stanford, 2006.

[55] "Self-Organizing Map (SOM) Kohonen artificial neural network code". In: (). URL: https://github.com/albertnadal/Kohonen.

[56] Jelena Dragas, David Jäckel, Andreas Hierlemann, and Felix Franke. "Complexity Optimization and High-Throughput Low-Latency Hardware Implementation of a Multi-Electrode Spike-Sorting Algorithm". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 23.2 (2015), pp. 149–158. DOI: 10.1109/TNSRE.2014.2370510.

[57] Daniel Valencia and Amirhossein Alimohammad. "An Efficient Hardware Architecture for Template Matching-Based Spike Sorting". In: *IEEE Transactions on Biomedical Circuits and Systems* 13.3 (2019), pp. 481–492. DOI: 10.1109/TBCAS.2019.2907882.

[58] Sungjin Oh, Sungmin Han, and Inchan Youn. "Real-time neural signal sensing and spike sorting system using a modified zero-crossing feature with highly efficient data computation and transmission". In: *Sensors and Materials* 29.7 (2017), pp. 1031–1042.

[59] Shuangming Yang, Jiang Wang, Qianjin Lin, Bin Deng, Xile Wei, Chen Liu, and Huiyan Li. “Cost-efficient FPGA implementation of a biologically plausible dopamine neural network and its application”. In: *Neurocomputing* 314 (2018), pp. 394–408.

[60] Shuangming Yang, Bin Deng, Jiang Wang, Huiyan Li, Meili Lu, Yanqiu Che, Xile Wei, and Kenneth A Loparo. “Scalable digital neuromorphic architecture for large-scale biophysically meaningful neural network with multi-compartment neurons”. In: *IEEE transactions on neural networks and learning systems* 31.1 (2019), pp. 148–162.

[61] Marc-Oliver Gewaltig and Markus Diesmann. “NEST (NEural Simulation Tool)”. In: *Scholarpedia* 2.4 (2007), p. 1430.

[62] Daria Lisitsa and Anton A. Zhilenkov. “Prospects for the development and application of spiking neural networks”. In: *2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. 2017, pp. 926–929. DOI: 10.1109/EIConRus.2017.7910708.

[63] Maxence Bouvier, Alexandre Valentian, Thomas Mesquida, Francois Rummens, Marina Reyboz, Elisa Vianello, and Edith Beigne. “Spiking neural networks hardware implementations and challenges: A survey”. In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 15.2 (2019), pp. 1–35.

[64] H. A. Swadlow and S. G. Waxman. “Axonal conduction delays”. In: *Scholarpedia* 7.6 (2012). revision #125736, p. 1451. DOI: 10.4249/scholarpedia.1451.

[65] E.M. Izhikevich. “Simple model of spiking neurons”. In: *IEEE Transactions on Neural Networks* 14.6 (2003), pp. 1569–1572. DOI: 10.1109/TNN.2003.820440.

[66] Jianhui Han, Zhaolin Li, Weimin Zheng, and Youhui Zhang. “Hardware implementation of spiking neural networks on FPGA”. In: *Tsinghua Science and Technology* 25.4 (2020), pp. 479–486.

[67] Sathish Panchapakesan, Zhenman Fang, and Nitin Chandrachoodan. “EASpiNN: Effective Automated Spiking Neural Network Evaluation on FPGA”. In: *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2020, pp. 242–242.

[68] Sathish Panchapakesan, Zhenman Fang, and Jian Li. "SyncNN: Evaluating and Accelerating Spiking Neural Networks on FPGAs". In: *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE. 2021, pp. 286–293.

[69] Nicholas T Carnevale and Michael L Hines. *The NEURON book*. Cambridge University Press, 2006.

[70] Dan Goodman and Romain Brette. "The Brian simulator". In: *Frontiers in Neuroscience* 3 (2009), p. 26. ISSN: 1662-453X. DOI: 10.3389/neuro.01.026.2009. URL: https://www.frontiersin.org/article/10.3389/neuro.01.026.2009.

[71] Alexander D Rast, Xin Jin, Francesco Galluppi, Luis A Plana, Cameron Patterson, and Steve Furber. "Scalable event-driven native parallel processing: the SpiNNaker neuromimetic system". In: *Proceedings of the 7th ACM international conference on Computing frontiers*. 2010, pp. 21–30.

[72] Eustace Painkras, Luis A. Plana, Jim Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R. Lester, Andrew D. Brown, and Steve B. Furber. "SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation". In: *IEEE Journal of Solid-State Circuits* 48.8 (2013), pp. 1943–1953. DOI: 10.1109/JSSC.2013.2259038.

[73] Kit Cheung, Simon R. Schultz, and Wayne Luk. "NeuroFlow: A General Purpose Spiking Neural Network Simulation Platform using Customizable Processors". In: *Frontiers in Neuroscience* 9 (2016), p. 516. ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00516. URL: https://www.frontiersin.org/article/10.3389/fnins.2015.00516.

[74] Shikhar Gupta, Arpan Vyas, and Gaurav Trivedi. "FPGA Implementation of Simplified Spiking Neural Network". In: *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 2020, pp. 1–4. DOI: 10.1109/ICECS49266.2020.9294790.

[75] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[76] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. "Spike-timing-dependent plasticity: a comprehensive overview". In: *Frontiers in synaptic neuroscience* 4 (2012), p. 2.

[77] Athul Sripad, Giovanny Sanchez, Mireya Zapata, Vito Pirrone, Taho Dorta, Salvatore Cambria, Albert Marti, Karthikeyan Krishnamourthy, and Jordi Madrenas. "SNAVA—A real-time multi-FPGA multi-model spiking neural network simulation architecture". In: *Neural Networks* 97 (2018), pp. 28–45. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2017.09.011. URL: https://www.sciencedirect.com/science/article/pii/S0893608017302150.

[78] Danilo Pani, Paolo Meloni, Giuseppe Tuveri, Francesca Palumbo, Paolo Massobrio, and Luigi Raffo. "An FPGA Platform for Real-Time Simulation of Spiking Neuronal Networks". In: *Frontiers in Neuroscience* 11 (2017), p. 90. ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00090. URL: https://www.frontiersin.org/article/10.3389/fnins.2017.00090.

[79] Vitor Bandeira, Vivianne Costa, Guilherme Bontorin, and Ricardo Reis. "Low Latency FPGA Implementation of Izhikevich-Neuron Model". In: Jan. 2017, pp. 210–217. ISBN: 978-3-319-90022-3. DOI: 10.1007/978-3-319-90023-017.

[80] Andrew Cassidy and Andreas G Andreou. "Dynamical digital silicon neurons". In: *2008 IEEE Biomedical Circuits and Systems Conference*. IEEE. 2008, pp. 289–292.

[81] Junwen Luo, Graeme Coapes, Terrence Mak, Tadashi Yamazaki, Chung Tin, and Patrick Degenaar. "Real-Time Simulation of Passage-of-Time Encoding in Cerebellum Using a Scalable FPGA-Based System". In: *IEEE Transactions on Biomedical Circuits and Systems* 10.3 (2016), pp. 742–753. DOI: 10.1109/TBCAS.2015.2460232.

[82] Alireza Khodamoradi, Kristof Denolf, and Ryan Kastner. "S2n2: A fpga accelerator for streaming spiking neural networks". In: *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2021, pp. 194–205.

[83] Matthieu Ambroise, Timothee Levi, Yannick Bornat, and Sylvain Saighi. "Biorealistic spiking neural network on FPGA". In: *2013 47th Annual Conference on Information Sciences and Systems (CISS)*. 2013, pp. 1–6. DOI: 10.1109/CISS.2013.6616689.

[84] Xilinx. “AXI DMA v7.1”. In: *LogiCORE IP Product Guide* (2019), p. 8.

[85] Michael Hopkins, Mantas Mikaitis, Dave R Lester, and Steve Furber. “Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural ordinary differential equations”. In: *Philosophical Transactions of the Royal Society A* 378.2166 (2020), p. 20190052.

[86] Maciej Śliwowski, Matthieu Martin, Antoine Souloumiac, Pierre Blanchart, and Tetiana Aksenova. “Decoding ECoG signal into 3D hand translation using deep learning”. In: *Journal of Neural Engineering* 19.2 (2022), p. 026023.

[87] Paola Busia, Andrea Cossettini, Thorir Mar Ingolfsson, Simone Benatti, Alessio Burrello, Moritz Scherer, Matteo Antonio Scrugli, Paolo Meloni, and Luca Benini. “EEGformer: Transformer-Based Epilepsy Detection on Raw EEG Traces for Low-Channel-Count Wearable Continuous Monitoring Devices”. In: *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. 2022, pp. 640–644. DOI: 10.1109/BioCAS54905.2022.9948637.

[88] Francesco M Petrini, Giacomo Valle, Ivo Strauss, Giuseppe Granata, Riccardo Di Iorio, Edoardo d’Anna, Paul Čvančara, Matthias Mueller, Jacopo Carpaneto, Francesco Clemente, et al. “Six-month assessment of a hand prosthesis with intraneural tactile feedback”. In: *Annals of neurology* 85.1 (2019), pp. 137–154.

[89] David A Moses, Sean L Metzger, Jessie R Liu, Gopala K Anumanchipalli, Joseph G Makin, Pengfei F Sun, Josh Chartier, Maximilian E Dougherty, Patricia M Liu, Gary M Abrams, et al. “Neuroprosthesis for decoding speech in a paralyzed person with anarthria”. In: *New England Journal of Medicine* 385.3 (2021), pp. 217–227.

[90] Marc W Slutzky. “Brain-machine interfaces: powerful tools for clinical treatment and neuroscientific investigations”. In: *The Neuroscientist* 25.2 (2019), pp. 139–154.

[91] Nur Ahmadi, Timothy G Constandinou, and Christos-Savvas Bouganis. “Decoding hand kinematics from local field potentials using long short-term memory (LSTM) network”. In: *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*. IEEE. 2019, pp. 415–419.

[92] David Sussillo, Paul Nuyujukian, Joline M Fan, Jonathan C Kao, Sergey D Stavisky, Stephen Ryu, and Krishna Shenoy. "A recurrent neural network for closed-loop intracortical brain–machine interface decoders". In: *Journal of neural engineering* 9.2 (2012), p. 026027.

[93] Nur Ahmadi, Timothy G Constandinou, and Christos-Savvas Bouganis. "Robust and accurate decoding of hand kinematics from entire spiking activity using deep learning". In: *Journal of Neural Engineering* 18.2 (2021), p. 026011.

[94] Xiangwen Wang, Xianghong Lin, and Xiaochao Dang. "Supervised learning in spiking neural networks: A review of algorithms and evaluations". In: *Neural Networks* 125 (2020), pp. 258–280.

[95] Xiaolong Wu, Guangye Li, Shize Jiang, Scott Wellington, Shengjie Liu, Zehan Wu, Benjamin Metcalfe, Liang Chen, and Dingguo Zhang. "Decoding continuous kinetic information of grasp from stereo-electroencephalographic (SEEG) recordings". In: *Journal of Neural Engineering* 19.2 (2022), p. 026047.

[96] Wafa Batayneh, Enas Abdulhay, and Mohammad Alothman. "Comparing the efficiency of artificial neural networks in sEMG-based simultaneous and continuous estimation of hand kinematics". In: *Digital Communications and Networks* 8.2 (2022), pp. 162–173.

[97] Yuwei Du, Jing Jin, Qiang Wang, and Jianyin Fan. "EMG-Based Continuous Motion Decoding of Upper Limb with Spiking Neural Network". In: *2022 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE. 2022, pp. 1–5.

[98] Yikang Yang, Jia Ren, and Feng Duan. "The Spiking Rates Inspired Encoder and Decoder for Spiking Neural Networks: An Illustration of Hand Gesture Recognition". In: *Cognitive Computation* (2022), pp. 1–16.

[99] Huijuan Fang, Yongji Wang, and Jiping He. "Spiking neural networks for cortical neuronal spike train decoding". In: *Neural Computation* 22.4 (2010), pp. 1060–1085.

[100] Xunguang Ma, Wenkai Zheng, Zujian Peng, and Jimin Yang. "Fpga-based rapid electroencephalography signal classification system". In: *2019 IEEE 11th International Conference on Advanced Infocomm Technology (ICAIT)*. IEEE. 2019, pp. 223–227.

[101] Mradul Agrawal, Sandeep Vidyashankar, and Ke Huang. “On-chip implementation of ECoG signal data decoding in brain-computer interface”. In: *2016 IEEE 21st International Mixed-Signal Testing Workshop (IMSTW)*. IEEE. 2016, pp. 1–6.

[102] Colin M McCrimmon, Jonathan Lee Fu, Ming Wang, Lucas Silva Lopes, Po T Wang, Alireza Karimi-Bidhendi, Charles Y Liu, Payam Heydari, Zoran Nenadic, and An Hong Do. “Performance assessment of a custom, portable, and low-cost brain–computer interface platform”. In: *IEEE Transactions on Biomedical Engineering* 64.10 (2017), pp. 2313–2320.

[103] Manfredo Atzori, Arjan Gijsberts, Claudio Castellini, Barbara Caputo, Anne-Gabrielle Mittaz Hager, Simone Elsig, Giorgio Giatsidis, Franco Bassetto, and Henning Müller. “Electromyography data for non-invasive naturally-controlled robotic hand prostheses”. In: *Scientific data* 1.1 (2014), pp. 1–13.

[104] Joseph E. O’Doherty, Mariana M. B. Cardoso, Joseph G. Makin, and Philip N. Sabes. *Nonhuman Primate Reaching with Multichannel Sensorimotor Cortex Electrophysiology*. This research was supported by the Congressionally Directed Medical Research Program (W81XWH-14-1-0510). JEO was supported by fellowship #2978 from the Paralyzed Veterans of America. JGM was supported by a fellowship from the Swartz Foundation. Zenodo, May 2017. DOI: 10.5281/zenodo.583331. URL: https://doi.org/10.5281/zenodo.583331.

[105] Salman Mohd Khan, Abid Ali Khan, and Omar Farooq. “Selection of features and classifiers for EMG-EEG-based upper limb assistive devices—A review”. In: *IEEE reviews in biomedical engineering* 13 (2019), pp. 248–260.

[106] Jing Fan and Jiping He. “Motor cortical encoding of hand orientation in a 3-D reach-to-grasp task”. In: *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE. 2006, pp. 5472–5475.

[107] Gerardo Saggese and Antonio Giuseppe Maria Strollo. “A Low Power 1024-Channels Spike Detector Using Latch-Based RAM for Real-Time Brain Silicon Interfaces”. In: *Electronics* 10.24 (2021), p. 3068.

[108] Shuangming Yang, Bin Deng, Huiyan Li, Chen Liu, Jiang Wang, Haitao Yu, and Yingmei Qin. "FPGA implementation of hippocampal spiking network and its real-time simulation on dynamical neuromodulation of oscillations". In: *Neurocomputing* 282 (2018), pp. 262–276. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.12.031. URL: https://www.sciencedirect.com/science/article/pii/S0925231217318751.

[109] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. "The spinnaker project". In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[110] Hasan Irmak, Federico Corradi, Paul Detterer, Nikolaos Alachiotis, and Daniel Ziener. "A dynamic reconfigurable architecture for hybrid spiking and convolutional fpga-based neural network designs". In: *Journal of Low Power Electronics and Applications* 11.3 (2021), p. 32.

[111] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. "Loihi: A neuromorphic manycore processor with on-chip learning". In: *Ieee Micro* 38.1 (2018), pp. 82–99.

[112] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip". In: *IEEE transactions on computer-aided design of integrated circuits and systems* 34.10 (2015), pp. 1537–1557.

[113] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[114] Sander M Bohte, Joost N Kok, and Han La Poutre. "Error-backpropagation in temporally encoded networks of spiking neurons". In: *Neurocomputing* 48.1-4 (2002), pp. 17–37.

[115] Satoshi Matsuda. "BPSpike: a backpropagation learning for all parameters in spiking neural networks with multiple layers and multiple spikes". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2016, pp. 293–298.

[116] Eric Hunsberger and Chris Eliasmith. “Spiking deep networks with LIF neurons”. In: *arXiv preprint arXiv:1510.08829* (2015).

[117] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing”. In: *2015 International joint conference on neural networks (IJCNN)*. ieee. 2015, pp. 1–8.

[118] Sumit Bam Shrestha and Garrick Orchard. “SLAYER: Spike Layer Error Reassignment in Time”. In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 1419–1428. URL: http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf.

[119] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. “Loihi: A neuromorphic manycore processor with on-chip learning”. In: *Ieee Micro* 38.1 (2018), pp. 82–99.

[120] Tim P Vogels and Larry F Abbott. “Signal propagation and logic gating in networks of integrate-and-fire neurons”. In: *Journal of neuroscience* 25.46 (2005), pp. 10786–10795.

[121] *Lava Software Framework*. 2022. URL: https://lava-nc.org/index.html.

[122] Shih-Hung Yang, Jyun-We Huang, Chun-Jui Huang, Po-Hsiung Chiu, Hsin-Yi Lai, and You-Yin Chen. “Selection of Essential Neural Activity Timesteps for Intracortical Brain–Computer Interface Based on Recurrent Neural Network”. In: *Sensors* 21.19 (2021), p. 6372.

[123] *Zynq-7000 SoC Data Sheet: Overview*. URL: https://docs.xilinx.com/v/u/en-US/ds190-Zynq-7000-Overview.