IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 16, 2020, accepted November 26, 2020, date of publication December 2, 2020,
date of current version December 15, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3042034*

# ZyON: Enabling Spike Sorting on APSoC-Based Signal Processors for High-Density Microelectrode Arrays

**GIANLUCA LEONE, LUIGI RAFFO, (Member, IEEE), AND PAOLO MELONI, (Member, IEEE)**
Dipartimento Ingegneria Elettrica ed Elettronica, Università degli Studi di Cagliari, 09123 Cagliari, Italy

Corresponding author: Paolo Meloni (paolo.meloni@unica.it)

This work was supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant 780788.

**ABSTRACT** Multi-Electrode Arrays and High-Density Multi-Electrode Arrays of sensors are a key instrument 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 behaviour 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 Field-Programmable Gate Array and All-Programmable System-On-Chip devices are suitable target technology for the implementation of real-time processors of High-Density Multi-Electrode Arrays data. However, approaches available in literature can process a limited number of channels or are designed to execute only the first steps of the neural signal processing chain. In this work, we propose an All-Programmable System-On-Chip based implementation capable of sorting neural spikes acquired by the sensors, to associate the shape of each spike to a specific firing neuron. Our system, implemented on a Xilinx Z7020 All-Programmable System-On-Chip is capable of executing on-line spike sorting up to 5500 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 on-line refinement of the sorting clusters to improve accuracy up to 82%, with only 4% degradation with respect to off-line analysis.

**INDEX TERMS** Field programmable gate arrays, signal processing, neural engineering, APSoC, HDMEA, spike sorting.

## I. INTRODUCTION

During the past decades, understanding neural signals and interaction between neural units has been a topic of interest in the medical and biomedical scientific community. Lots of research efforts have been dedicated to advance the knowledge on the field, mainly aimed at long-term important objectives, such as the comprehension of neural networks functional principles [1] and the implementation of neural prosthetic systems [2].

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, literature presents Multi-electrode arrays (MEAs) [3] permit long-term multi-units recording. Multielectrode probes have been also proposed, well suited to monitor neurons in both superficial and deep brain structures. Probes host hundreds of recording sites in 5 mm length [4] and almost one thousand in 10 mm [5]. Finally, High-density MEAs (HDMEAs) permit to retrieve information at the single cell level [6], to study electrical- and light-evoked neural response and to acquire from tens of thousands recording sites. For example, [7] features 4096 recording sites, and [8] features 65,536 recording sites. In HDMEAs, the number of channels, growing from tens to thousands, drastically improve 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

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

analysis must include *Spike Sorting* [9], 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, thus do not match the requirements of HDMEA applications.

In our work, we focus on FPGA-based solutions, since FPGAs are prospectively very well suited for parallel and highly DSP-intensive signal processing. HDMEA 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 the 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.

In this work, we rely on a previously presented neural signal processing system [10], named ZyON (Zynq-based On-line Neural processor), implemented on a Xilinx Zynq APSoC, that hosts on the same chip a dual-core ARM-based Processing Sytems (PS) and a fabric of FPGA-based reconfigurable logic. In ZyON, the PS is used to close the loop and apply stimuli to the tissue, whereas the circuitry implemented on the FPGA is capable to execute the most computationally demanding portions of the processing operating in parallel on the streams of samples acquired by the different channels, such as filters and threshold monitoring for spike detection.

The main contribution of this work, we extend ZyON implementing support for spike sorting. We implement additional digital modules on the programmable logic, to speed-up 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 [11] and Self Organizing Map (SOM) [12]. 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 flexibility and adaptability to multiple analysis cases.

The main findings of this paper 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 co-operative 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 [13] and explore its parameters to validate and justify our design choices.

The remainder of this article is organized as follows. Section II contains an overview of existing online spike sorters and online spike sorting algorithms; Section III presents the target processing tasks and the overall structure of the sorting pipeline; Section IV describes the processing system architecture and the involved functional blocks; Section V discusses the achieved results, presenting experiments to assess accuracy and performance; Section VI is dedicated to a comparison with alternatives available in literature; conclusions are reported in Section VII.

## II. 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 literature have a wide scope of objectives: the purpose may be to interact with the tissue [14], or to partially process the data to limit memory [15] and bandwidth requirements [16]. Some instruments are designed to operate offline, such as [13] and [17], which reaches outstanding performance on different 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 [18]. 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 [13], as well as custom hardware devices implemented on FPGA [10] or ASIC [19].

Finally, different research works target different sorting strategies and focus on different steps of the sorting procedure. For instance, some works only implement online spike detection. However, the objective could be reducing memory requirements [15], reducing bandwidth requirement [16],

**TABLE 1. Related work summary.**

STATE OF THE ART COMPARISON TABLE

| Reference | Yang [27] | Saeed [20] | POSort [28] | SelfSort [23] | HAM [24] | Do [19] | Park [18] | Müller [14] | This work |
|---|---|---|---|---|---|---|---|---|---|
| Year | 2017 | 2013 | 2019 | 2017 | 2014 | 2018 | 2017 | 2013 | 2020 |
| Hardware | FPGA | - | FPGA | - | - | ASIC | FPGA | FPGA | FPGA |
| Channels | 32 | - | 128 | - | - | 128 | 128 | 126 | 4096 |
| Sampling [KHz] | 20 | - | 24 | - | - | 25 | 32.5 | 20 | 18 |
| Detection | TEO* | TEO | - | TEO | - | Filter | yes | yes | yes |
| Feature | DWT | ZCF | - | ZCF | FSDE | Filter | - | no | FSDE |
| Class/Clust | BDT | KM-Class | OSort | SOM-Class | HAM | KM-Class | Template | no | K-Means-based |
| Resolution | - | 10 | 16 | 10 | - | 9 | 16 | 8 | 12 |
| Accuracy | (60%:80%) | 82% | 87% | 93.4% | >90% | [72% : 86%] | - | - | 82% |
| SNR [dB] | [5:7] | 7 | [10:13] | [0:15] | [7:13] | [4.6:13] | - | - | [7:13] |

* Modified threshold

or stimulating the neural tissue in real-time, in response to the sensed activity [14].

Others focus on complete real-time spike sorters, integrating steps such as processing of raw data, extraction of features relevant for classification, clustering, i.e identification of spike classes to be considered, and assignment of incoming spikes to clusters. For instance implementations such as [20] and [21] face the problem of on-line sorting, using significantly different sorting strategies. In [20], authors rely on feature extraction using the Zero Crossing Feature (ZCF) method [22], 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, [21] avoids extracting features and relies on direct processing of the raw spike waveforms. Such as in [18], where authors directly cluster raw spike data employing a set of carefully chosen thresholds, to create and update clusters.

Selfsort [23] in contrast, despite keeping a structure similar to [20], firstly uses a Self Organizing Map offline to get the cluster centers, and, secondly, takes advantage of the approximated cluster centers, computed on the first set of incoming spikes only, to simplify the system by implementing in hardware a classifier instead of a clustering algorithm.

The HAM [24] algorithm is another example of an online clustering approach, where the clusters are dynamically added, updated, or merged. In [24] a different feature extractor method called First and Second Derivative Extrema (FSDE) [25] 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 [19]: a 128-channel spike sorting chip designed for low-power. A detection strategy similar to the Teager Energy Operator (TEO) [26], based on preemphasising the neural signal is proposed. However the method is linear and it doesn't need multiplications, it only uses sums and shifts. It also applies a similar linear transformation to the spike waveforms in order to extract features, which are therefore classified by the use of an improved K-Means algorithm.

Furthermore, an example of an FPGA based system is given in [27]. 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.

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. [19] considers six as the maximum number of clusters. HAM [24] and Selfsort [23] set a limit on the maximum number of clusters per channel referring to [29], 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.

Work in literature more related to ours is summarized in table 1. All the works address a small number of electrodes compared to the proposed work. This results in very local monitoring of the neural tissue or conversely in 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 profit from 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 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.

## III. TARGET SPIKE SORTING PIPELINE

Spike sorting (SS) is a key step for the analysis of neural signals. It consists of the separation of the superimposed activities of the neuronal cells sensed by the same electrode.

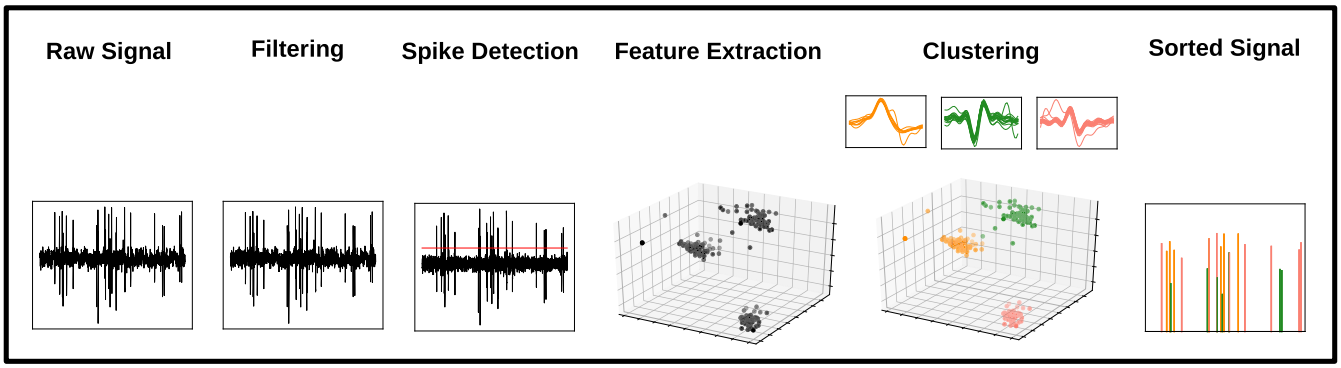


**FIGURE 1.** The spike sorting chain, from left to right: the raw signal and the filtered signal; Spike detection: the filtered signal is compared to a threshold represented by an horizontal red line; Feature extraction: valuable features are extracted from the spikes waveforms (the spikes are depicted in the feature space); Clustering process: the spikes are grouped depending on their features; Sorted signal: the spike trains can be identified with the respective firing neuron.

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 [9] shown in Figure 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 [30]. 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.
- 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 on-line 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. A first method runs a data-stream clustering algorithm, as in [24] and in [28]. A 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 [23]. Thus, in this case, the final processing stage in Figure 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 on-line the similarity of incoming spikes to the templates identified by the clustering, to perform the eventual sorting.

## IV. SYSTEM ARCHITECTURE

The proposed processing system architecture is shown in Figure 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 in different devices of the family. However, the system configuration presented in this paper is implemented on a Z-7020 device.

As mentioned, in this work we start from a previously presented platform, named ZyON [10]. Zyon, in its previous implementation, shares the principles of this paper. Both the PL and PS are used in cooperation. The PL is populated with modules, described in HDL, that implement the front-end tasks of the neural signal processing chain, until the spike detection phase. The PS is 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 [31] the spike redundancy among the channels has been used to reduce the number of active electrodes and lower the computational burden of the HDMEA signal analysis, in the case of retinal circuits. This work adds further steps in the chain to the pipeline, implementing and assessing feature extraction and clustering on several reference signal datasets.

As presented in [10], the system is instrumented to be interfaced with a BioCam X platform by 3Brain AG. Such platform embeds an active CMOS-MEA device, capable of acquiring 4096 signals with adequate electrodes, sampled at a maximum frequency of 18KHz and, 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 embeds a *filtering* stage, where the digital neural signals are multiplexed in time to be processed by a bank of 32 digital FIR filters of order 63, with cut-off frequency

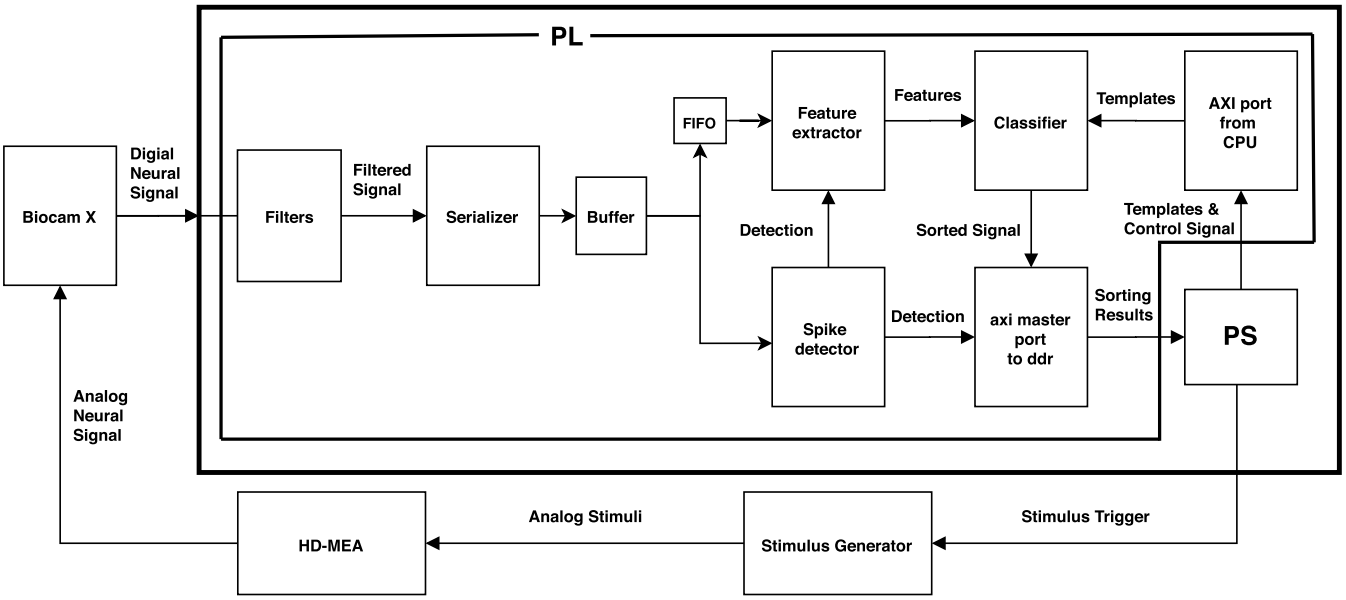


**FIGURE 2.** Schematic block diagram of the experimental setup. Biocam X provides ZyON with digital data of 4096 electrodes mounted on the HDMEA. ZyON processes the data and generates and applies the output stimuli through the Stimulus Generator.

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 are subsequently further serialized and processed by the downstream modules, that, exploiting an efficient hardware-level pipelining, implemented in the RTL description of each module, reach a throughput of 1 samples/cycles.

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, to be processed as soon as the threshold is exceeded and an activation signal is received from the spike detector.

Once the features are computed, the *Classifier* evaluates the distance metric between the feature vector and a set of pre-stored templates. It classifies the spikes by identifying the template producing the minimum distance. The features computations elapse for $W$ sampling cycles, where $W$ is the number of samples in the window representing the spike. All the mentioned modules, which take care of the data-crunching tasks in the pipeline, are implemented on the programmable logic. We have used two dedicated AXI High-Performance ports, available in Zynq-7000 devices to allow communication between such modules and the processing system. More details about specific communication items are available in Section IV-C. 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, to process them to identify cluster centers, and to store them in the *Classifier*. For example, in the main experiments presented in this paper, the PS has been used to execute a K-means clustering algorithm on a subset of spike feature vectors, to create templates to be considered during classification.

Moreover, the PS takes care of:

- implementing closed-loop interaction tasks;
- refining the templates, if needed, considering partial results of the classification during a spike sorting experiment;
- taking care of the system housekeeping tasks, such as memory management, network communication, input/output, 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 modification. Hardware changes are required only when lower-level algorithm parameters, such as detection method and classification metrics, have to be replaced.

In Figure 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.

### A. FEATURE EXTRACTOR

Our *Feature Extractor* implements the First and Second Derivative Extrema (FSDE) feature extraction algorithm [25]. FSDE based spike sorters use the minimum and maximum extrema of both derivatives as features. However, usually not all the extrema are considered. By relying on the evidence

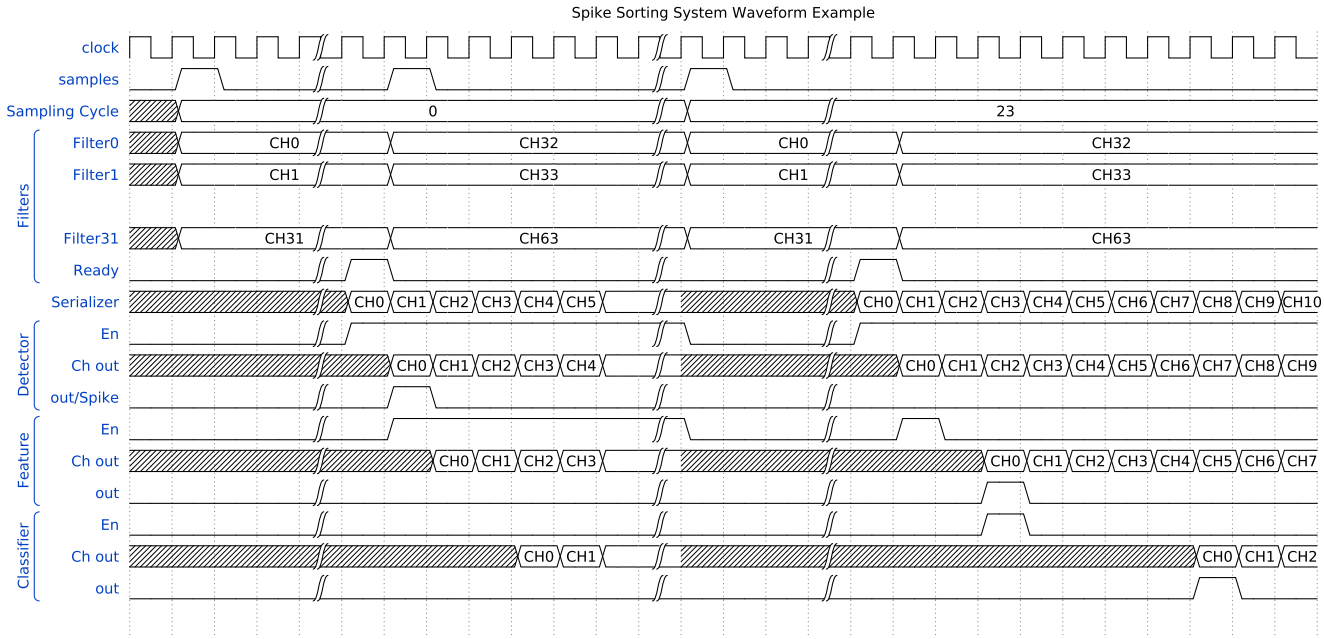


**FIGURE 3.** The 4096 channels are time-multiplexed and processed by the FIR filter bank. At the beginning of every sampling cycle the channels ranging from 0 to 32 are processed in parallel, the channels 33 to 63 follow after 40 clock cycles, and so on, up to the last group of channels (from channel 4064 to channel 4095). 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 predefined threshold. As soon as the following 32 samples are computed by the filter bank, they are processed as well, up to the last group (from channels 4064 to 4095). 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.

proved in [25], the maximum of the first derivative and both the maximum and the minimum of the second derivative used together permit to achieve the best possible accuracy.

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

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

$$SD(i) = FD(i) - FD(i-1) \tag{2}$$

where $FD$ and $SD$ are respectively, the first and second derivatives and $x(i)$ is the $i^{th}$ sample of the spike window. Then, the extrema of $FD$ and $SD$ may be computed as follow:

$$FD_{max} = max\big(FD(i)\big) \tag{3}$$

$$SD_{max} = max\big(SD(i)\big) \tag{4}$$

$$SD_{min} = min\big(SD(i)\big) \tag{5}$$

The features are computed within a window placed before the detection event. Performing max and min search without a-priori knowledge about the location of the spike detection event requires some memory. 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 6.

$$FIFO_{size} = D \times C \times S \tag{6}$$

where $D$ is the required number of samples, prior to the threshold trespassing that determines the spike detection, to be considered as head of the spike waveform. $D$ also determines a certain delay between a sample entering the *Feature Extractor* and its actual contribution to feature evaluation. $C$ 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, at the same time, has an impact on accuracy. Excessively 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 4 shows the *Feature Extractor* architecture. While, with a reduced number of channels, the internal buffers required to store the samples and the derivatives could use distributed Look-Up-Tables to create small RAM modules inside the datapath, in our case study, acquiring from thousands of channels simultaneously, BRAM blocks are more effectively used to create the buffers. The *Feature Extractor* is composed by 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, implementing Equation 1. 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.

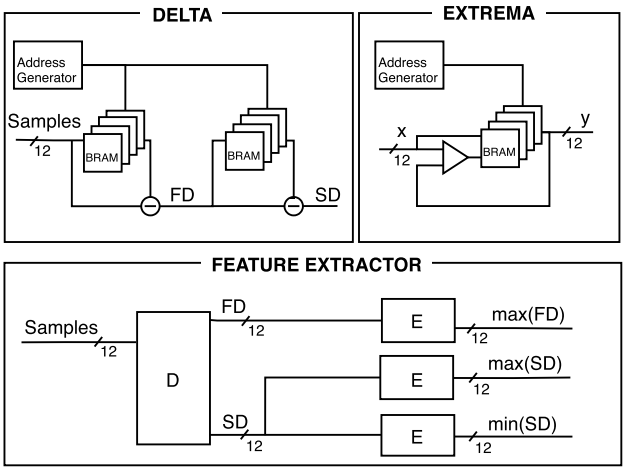


**FIGURE 4.** The *Feature Extractor* evaluates the derivative extrema of the spike waveform, the maximum of the first derivative $FD_{max}$ and both the extrema of the second one $SD_{max}$ and $SD_{min}$. The *Delta* block computes the derivatives. The *Extrema* block evaluates the derivative extrema.

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 4. When one of the updating conditions obtained by Equation 3-5 is satisfied, the old value of the extrema is updated. Since the algorithm elapses for many sampling cycles (equal to the dimension of the spike window in samples $W$) and is potentially executed independently on every channel, one counter per channel is also needed. It is possible to implement BRAM-based counters since every counter is accessed only two times per sampling cycle: to read and update the value if the FSDE algorithm is running.

The data sampled by BioCam X are quantized using 12 bits. FSDE features can also be expressed using 12 bits since subtraction of adjacent samples does not determine overflows. To validate such a choice, we tested it on the dataset [13], verifying that, due to limited distance between successive samples, overflows are never experienced.

The *Feature Extractor* drives the results of the algorithm on the output and triggers the *Classifier* when the features are ready.

### B. CLASSIFIER

Sorting algorithms rely on an on-line classification process, that, based on a similarity metric, associates incoming data to one element inside a set of pre-defined classes. Thus, a wide variety of approaches, e.g. the strategy proposed by Selfsort [23], 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 [24], 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. While several algorithms require a different kind of classifier, euclidean distance is a common choice that may serve different alternative clustering algorithm, such as, for example, K-Means [11], whose results are evaluated in more detail in the following, and SOM [12], 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 current *Classifier* implementation is based on equation 7, where the square of the euclidean distance $D_i$ is evaluated for each and every templates $T_i$, with $i \in [1, K]$. With $K$ the number of templates per channel.

$$\begin{aligned} D_i = &(FD_{max} - T_{i_1})^2 \\ &+(SD_{max} - T_{i_2})^2 \\ &+(SD_{min} - T_{i_3})^2 \end{aligned} \tag{7}$$

The distances $D_i$ are then compared to select the class whom the spiking neuron belongs:

$$neuron\ id = argmin(D_i) \tag{8}$$

The *Classifier* is triggered by the *Feature Extractor* once the spike waveform has been processed entirely and the feature vector is ready. However, a certain number of templates are needed to carry on the classification. The templates are stored into a BRAM-based buffer by the PS through one of the two AXI interfaces. The throughput required to load the templates depends on process parameters. In more detail, defining the number of templates per channel as $K$, the number of features as $F$ and the samples size as $S$, the required throughput is:

$$Templates_{thr} = K \times F \times S \tag{9}$$

whereas the size of the buffer is equal to:

$$Templates_{size} = C \times K \times F \times S \tag{10}$$

where $C$ is the number of channels. The *Classifier* accesses the templates memory and it evaluates the euclidean distances among the feature vector and the $K$ templates. Then, it compares the $K$ resulting euclidean distances to assign the incoming spikes to the class with the smallest value of distance. The number of templates represents the number of estimated neurons sensed by the electrode and it is a process parameter ($K$ is set to 8 in this hardware [29]).

Every evaluation consists of three independent differences, three independent multiplications, and a final sum of the three products. The module in charge of implementing such computations is the *Distance* block, shown in Figure 5. Subtractions and multiplications are embedded into DSP blocks, whereas the three terms additions are demanded to three inputs lut-based adders. Even though the final adder of a DSP block is a three terms adder, unfortunately, two addends are required to carry out the multiplication and so the three inputs lut-based adders are required. The euclidean distances are

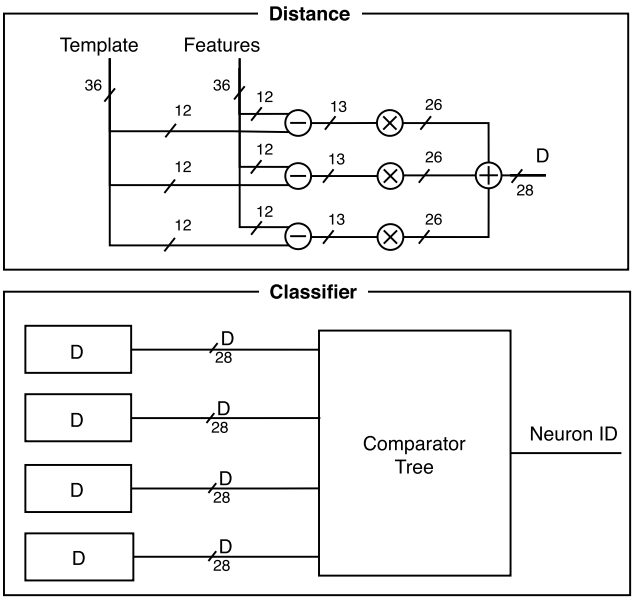


**FIGURE 5.** A simplified *Classifier* block with a number of templates *K* equal to 4 (instead of 8). The *Distance* block computes the Euclidean distance between the feature vector and a template. *K Distance* blocks are instanced, then the *Comparator Tree* figures out the smallest Euclidean distance identifying the firing neuron.

computed concurrently, therefore an instance of the *Distance* block is necessary for each template. Furthermore, since a DSP is needed for every multiplication, it is possible to estimate the number of DSPs required:

$$Classifier_{DSP} = T \times F \tag{11}$$

A LUT-based three terms adder is also needed for every *Distance* block, i.e. $K$ three-terms adders are also instanced. Figure 5 shows the *Classifier* architecture for $K$ equal to 4.

The subtractions and the multiplications take advantage of the registers present inside the DSPs to pipeline the computation, meanwhile, 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 $K$ and so as the latency:

$$Comparator_{latency} = log_2 K \tag{12}$$

The comparators needed to make up the tree:

$$Comparators = K - 1 \tag{13}$$

Differently from the feature computation results, which fit in 12-bits registers, the euclidean distance partial results require more bits. In particular, feature-to-template differences require an extra bit to avoid overflow, i.e. 13-bits, the square computation requires to double 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 V-D2. At the end of the classification, the *Classifier* triggers the *Communication* block to transmit the results.

### C. 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 to limit communication to spike detection results. Furthermore, it possible to use the second AXI interface both to update the *Classifier* templates and to set up the sorting parameters by programming some memory-mapped storage locations. In more detail:

- Transmission of output data: The programmable logic transmits the features of the detected spikes as two burst AXI transactions, the former is used to store feature vectors, and the latter burst is used to send information composed by a channel ID, which identifies the electrode where the spike has been detected, and the classification result. 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 [32]. 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 must be used during *Clustering*. During such procedure, the PS scans the DDR region reserved for the previously described packets and fills a data structure collecting training spikes. Subsequently, it runs the clustering algorithm and updates the templates if needed. The data scan of the 40KB packet elapses for less than 900 $\mu s$.

- Transmission of sorting results: 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 are the classification results. A 2 KB packet (4 bits x 4096 channels) is written from the PL at each sampling cycle, once every 55.6 $\mu$s (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 $\mu$s, to copy 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 bits 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 some kind of clustering algorithm on a collection of features stored in the DDR to generate new templates and update the *Classifier*. The amount of data necessary to update a single *Classifier* is 36 Byte, 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 level, and the DDR baseline address.

## V. 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 on-line 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 testing the selected feature set, comparing with a ZCF [22] scheme, and 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.

### A. HARDWARE REPORT

The target device is the ZedBoard, a low-cost development board for the Xilinx Zynq Z-7020 All-Programmable SoC.

**TABLE 2. Classifier resource requirements at the varying of the number of templates *K* and the number of features *F*.**

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) |

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 25 $\times$ 18 bits multiplier, and a 48-bits accumulator.

The FIR filter block is constituted by 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 described by Equations 1-5. 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 blocks each. The FSDE algorithm also needs a counter per channel. The counters are BRAM-based and they 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 2.

The DSPs, the adders and 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 because, being a power of two, it is a more hardware friendly parameter. The architecture can be easily extended to support a different number. The limiting factor is the number of BRAMs, which poses the limit to the number of neurons per channel to 16. By looking at Table 2, 33 BRAM tiles are required for K = 8. For K = 16, implementation would increase BRAMs by 2x, almost saturating (137 out of 140) the availability in the device.

After implementation, using Vivado v2017.4, it has been possible to obtain about the overall hardware resource utilization in the device, as shown in Table 3.

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

**TABLE 3. Post-implementation resource requirements of the Xilinx Zynq Z-7020 obtained by using Vivado v2017.4.**

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 |

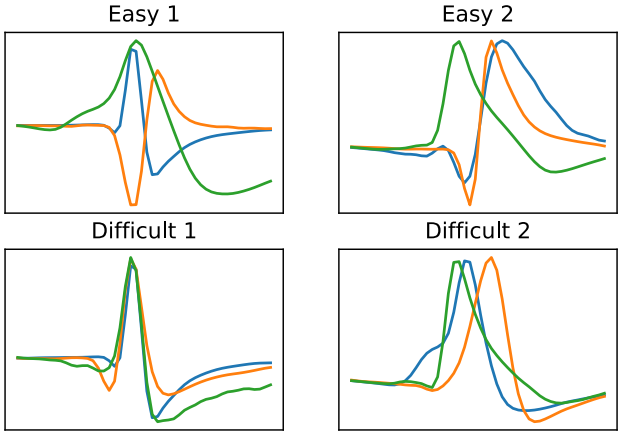


**FIGURE 6. Spike waveform models of the four datasets presented in [13].**

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 still satisfying the real-time constraints.

### B. EXPERIMENTAL SETUP

#### 1) REFERENCE BENCHMARK DATASET

To assess the functionality of the system, we have used, as a reference benchmark, the dataset presented in [13], composed of four simulations, named *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 $\sigma$ 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 6 shows the waveform models of the three neurons in the four datasets. The model waveforms are built by computing a sample by sample average between all the spike waveforms in the dataset.

#### 2) EXPERIMENTAL SETUP

To test the device on the dataset [13] 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-bits 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 filtering and sorting channels, through two AXI-Stream Broadcaster modules. The resource occupation overhead due to such testing infrastructure is shown in Table 4. For the accuracy evaluation used during preliminary design space exploration, we have used the software implementation described in Section V-B3.

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

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 |

#### 3) 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 [9] in Python, available for download and contribution as open source.[1] Thanks to such software implementation, it is 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 Thresholding* method proposed in [13]:

$$Thr = \alpha \times median\left(\frac{|x|}{0.6754}\right) \tag{14}$$

where $x$ is the neural signal and $\alpha$ is a parameter set to 4.0 as suggested in [13].

Furthermore, different feature extraction algorithms might be compared, such as Integral Transform, Zero Crossing Feature, First and Second Derivative Extrema. The coherence between results obtained by software and on the hardware platform has been 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.

### C. ACCURACY EVALUATION

We used our system to perform several accuracy tests, comparing the spike-to-cluster association decided by our *Classifier*, to the ground truth provided with the datasets. As mentioned, we have detected spikes using Equation 14, we used FSDE features and on-line classification based on Euclidean distance. Figure 7 shows the obtained cluster distribution over the feature space. The plots show two out of the three

[1] https://github.com/gianlucaleone/Spike_Sorting

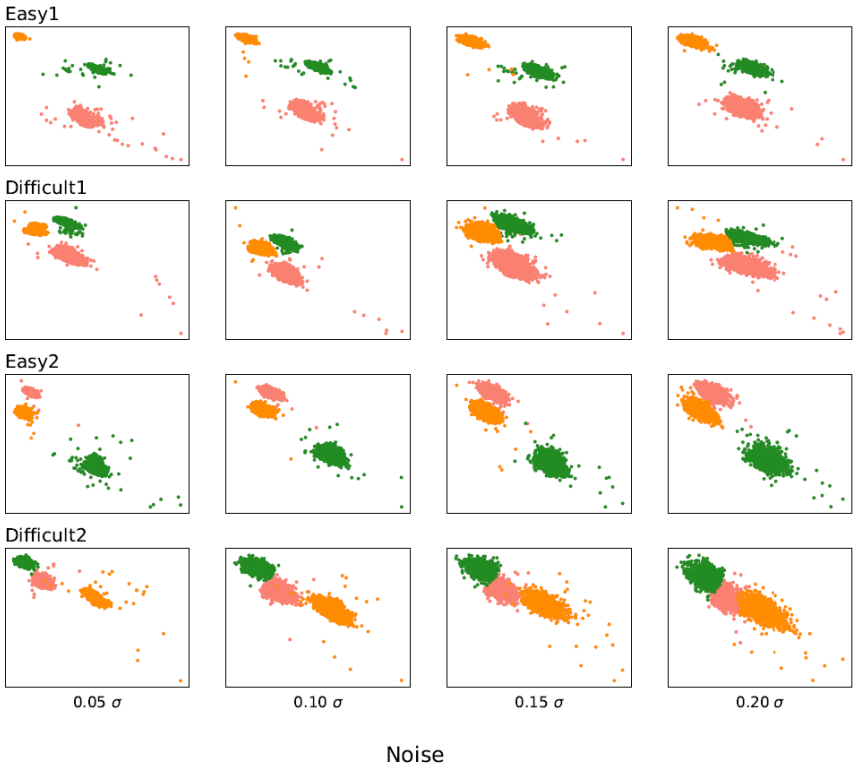


**FIGURE 7.** Cluster shapes of the 16 simulations [13]. The noise increases from left to right, the radius of the clusters increases with the noise, and the clusters get closer. The *Difficult* simulations exhibit closer clusters compared to the *Easy* simulations.

features, *first derivative maximum* on the x-axis and *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 some noise level, in the other datasets, clusters eventually start to be close or to overlap.

In Figure 7, cluster centers are computed off-line using a K-Means algorithm on all the spikes in the datasets. This kind of approach is not usable to implement on-line sorting, thus is not suitable for any kind of closed-loop application involving the HDMEA. Conversely, we have evaluated the overall accuracy for each dataset, using a limited number of *training* spikes to define the templates of the *Classifier*, through the use of the K-Means clustering algorithm. We have explored the number of training spikes to evaluate its impact on accuracy. Figure 8 shows the results. 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 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 vary 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. Moreover, in general, it is possible to notice a significant accuracy deviation from the median value, when selecting some specific spike sets for training, 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* show bigger boxes, 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.

### 1) 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, to limit the effect of poorly-performing training sets of spikes using each template set for a shorter time. The tested

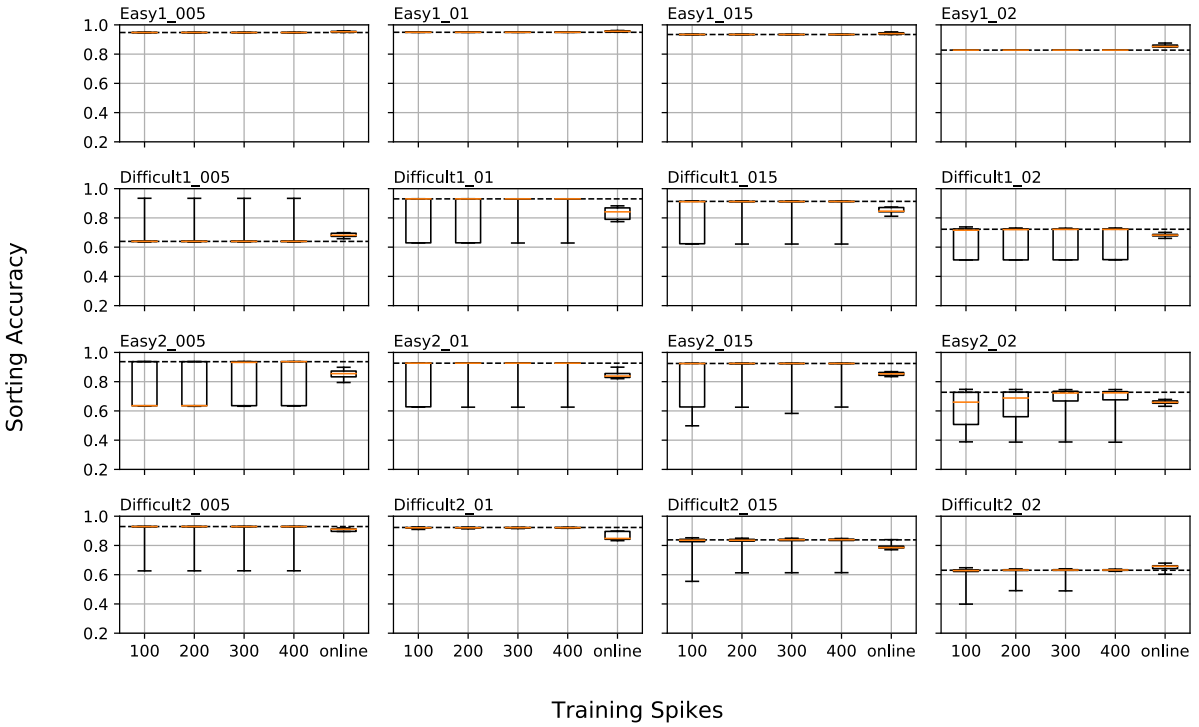


**FIGURE 8.** Sorting accuracy at the varying of the number of spikes used to prepare the *Classifier* templates trough the use of the K-Means clustering algorithm. The noise increases along the columns, the dataset changes along the rows. The dashed line states the offline accuracy of the method and the box-and-whisker plots show the collected accuracy obtained by running the k-means over 200 randomly chosen sets of spikes. The last box-and-whisker plot of every figure shows our real-time accuracy.

**TABLE 5.** K-Means run time on the PS. The dataset is constituted of 300 spikes. Every time measure is averaged over 1000 runs.

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

K-Means implementation, in C language, is taken from [33]. Cluster centers are initialized by the use of K-Means++ [34]. The maximum number of iterations for refining the center is set to 10. To assess the possibility of repeating the clustering, we have measured the execution time on the PS. Table 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.

Execution is in general reasonably fast. The average time in table 5 is about 106 $\mu s$, corresponding to less than 4 sample times, confirming the possibility of executing the algorithm multiple times to refine the coordinates of the centers during an experiment.

The rightmost box-and-whisker plots in Figure 8 show the accuracy obtained by repeating the K-Means clustering over 300 spikes once every three seconds (an execution rate that can be comfortably supported considering run-times in Table 5 for 4096 channels). 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. Unfavourable 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 algorithms, 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 paper. 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 [35]. 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 $4 \times 2$ neural network and considering 200 training spikes, the average training time is about 2.67 seconds.

### D. DETAILED EVALUATION OF THE IMPLEMENTATION

We have performed multiple tests to confirm our design choice, evaluating the impact of architectural details on the overall accuracy.

#### 1) EVALUATION OF FEATURE EXTRACTION AND DETECTION METHODS

To assess the impact of the chosen First and Second Derivative Extrema [25] algorithm on the accuracy, we have

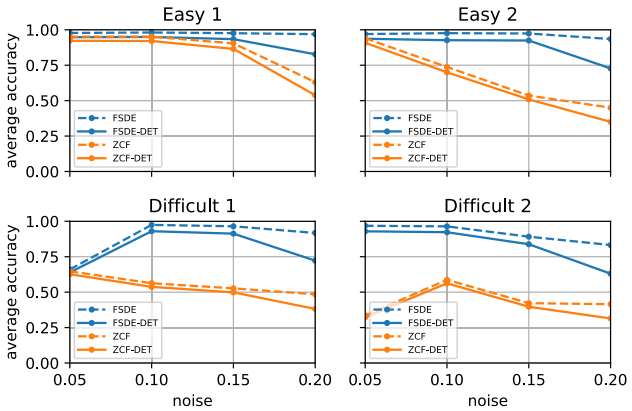


**FIGURE 9.** First and Second Derivative Extrema and Zero Crossing Feature extraction methods comparison.

compared it with a Zero Crossing Feature [22] implemented 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 9 shows four plots, one for each dataset track. The plots report the accuracy of the sorting at four different noise levels. In order to evaluate the impact of the spike detection phase, we report, besides the overall accuracy, commonly estimated taking into account the number of false positives $F_P$ and of false negative $F_N$ as in Equation 15 (solid lines in Figure 9), a detection-insensitive metric that only considers the ratio of correctly classified spikes over those detected by the system, as in Equation 16 (dashed lines).

$$Acc = \frac{Right\ Classified}{Detected + F_P + F_N} \tag{15}$$

$$Acc_{no\ det} = \frac{Right\ Classified}{Detected} \tag{16}$$

In the simulation *Easy 1*, the FSDE accuracy is over 0.9 for the first three noise levels, and it is only slightly better than the ZCF one. Nevertheless, the ZCF accuracy falls when the noise level is increased at $0.20\sigma$, whereas the FSDE accuracy still is at 0.83. In simulations *Easy 2* and *Difficult 1*, the methods exhibit the same accuracy for $0.05\sigma$. 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 the simulation Difficult 2, the accuracy performance of ZCF is constantly lower than FSDE. FSDE appears to be dominant in every simulation, 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 on the considered datasets [13]. Finally, it may be noticed that both algorithms suffer from some defects in the detection methodology. Using a static threshold appears to significantly affect accuracy when the noise increases to 0.20 (deviation between dashed and continuous lines in the graph). However, it may be noticed that, when disregarding mis-detected spikes, the accuracy gap between FSDE and ZCF is even bigger, especially for higher levels of noise.

**TABLE 6.** Spike sorting relative error of the fixed-point data flow. The error measures the accuracy difference between the fixed and floating-point algorithms.

FLOATING VS FIXED POINT SORTING ERROR

| Datasets | Level of Noise | | | |
|---|---|---|---|---|
| | $0.05\sigma$ | $0.10\sigma$ | $0.15\sigma$ | $0.20\sigma$ |
| *Easy 1* | 0.0 | 0.0 | 0.0 | 0.0 |
| *Easy 2* | 0.0 | 0.0 | 0.0 | 0.22 |
| *Difficult 1* | 0.0 | 0.0 | 0.0 | 0.0 |
| *Difficult 2* | 0.0 | 0.03 | -0.20 | 0.02 |

**TABLE 7.** Accuracy at the varying of the FIFO buffer size.

| Shift | Accuracy | BRAM |
|---|---|---|
| 2 | 74% | 3 |
| 4 | 82% | 5 |
| 8 | 86% | 11 |
| 16 | 79% | 22 |

### 2) EVALUATION OF FIXED POINT IMPLEMENTATION

The architecture embeds fixed-point processing elements rather than floating-point ones. In order to evaluate the accuracy penalty deriving from such approximated format, the fixed and floating-point sorting results are compared inside the software pipeline.

The data sampled by BioCam X is 12-bits wide. We chose to avoid the impact on accuracy deriving by adapting data representation to the same format after every processing step. Instead, we defined the width of internal bus signals to avoid overflows, at the expense of higher resource utilization. This does not affect the transmission rate to DDR, since the information which needs to be sent in output is only including FSDE features, encoded in 12-bits, and classification results, encoded in 3-bits to represent eight neurons per channel.

Table 6 shows the relative error of the fixed point spike sorting algorithm compared to the floating-point results. The spike sorting error for each simulation is given along the rows, whereas the levels of noise swipe along the columns. The relative error of the clustering is zero in most of the simulations, completely at zero for *Easy 1* and *Difficult 1*. The simulation *Easy 2* $0.2\sigma$ has a loss of accuracy of the 0.22%. The simulation *Difficult 2* presents a loss of the 0.03% for $0.1\sigma$ and a loss of the 0.02% for $0.2\sigma$. Finally, the simulation Difficult 2 $0.15\sigma$ presents an improvement of the accuracy of 0.2%. This demonstrates that there is no significant accuracy drop when moving toward fixed point algebra.

### 3) SPIKE WINDOW CENTERING EXPLORATION

We found a correct centering of the spike in the window of samples used to extract features to be key for the overall accuracy. As previously mentioned, such centering is implemented by continuously keeping track of the recent samples inside a FIFO, while waiting for the detection to trigger the *Feature Extractor*. 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 7 shows the average offline accuracy of the system on the 4 reference datasets [13], at the varying of the centering of the spike window, expressed in number of sampling cycles

**TABLE 8. Other FPGA-based spike sorters comparison summary.**

FPGA STATE OF THE ART COMPARISON TABLE

| Reference | Park [18] | POSort [28] | Yang [27] | Dragas [36] | Valencia [37] | Sungjin Oh [38] | Schäffer [39] | This work |
|---|---|---|---|---|---|---|---|---|
| Year | 2018 | 2019 | 2017 | 2015 | 2019 | 2017 | 2020 | 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-7000 |
| Input Channels | 128 | 1 | 32 | 90 | 1 | 1 | 128 | 4096 |
| Output Channels | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 16 |
| Sampling [KHz] | 32.5 | 24 | 20 | 20 | 24 | 50 | 20 | 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 | K-Means-based |
| Resolution [bits] | 16 | 16 | 10 | 10 | 16 | 12 | 16 | 12 |
| Accuracy | - | 87% | (60%:80%) | <85% | 90% | - | 86 | 82% |
| SNR [dB] | - | [10:13] | [5:7] | >5 | [10:13] | - | [3-10] | [7:13] |
| BRAM | - | 29 | - | 865kb | 0 | - | 98 | 104 |
| DSP | - | 130 | - | - | 5 | - | 60 | 61 |
| LUT | - | 16472 | - | 190000 | 6628 | - | 51674 | 28984 |
| REG | - | 8444 | - | 29000 | 4880 | - | 17484 | 26444 |
| Frequency [MHz] | - | 123 | 0.160 | - | 102 | 50 | 200 | 125 |
| Clustering Latency [$\mu$s] | - | 0.25 | - | - | 0.55 | - | - | 0.08 |
| Latency [ms] | - | - | - | 2.65 | - | - | 0.087* | 2.3 |

*without filtering

stored preventively in the FIFO, and the BRAMs required to implement the FIFO buffer.

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 therefore accuracy is affected. Nevertheless, increasing the number of samples to 16 loses too much information from the tails of the spikes, thus does brings to both a negative effect on accuracy and to over-utilization of the BRAM resources on the device.

## VI. COMPARISON WITH STATE OF THE ART

Table 8 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 [18] Park *et al.* present a multichannel neural interface capable of sorting 128 channels simultaneously and to stimulate the neural tissue from 8 electrodes. The neural interface presented is based on template matching and it is hosted by the Xilinx Kintex-7 XC7K160T. The device embeds 600 DSP slices and 325 36Kb BRAM tiles. No precise resources utilization are 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 shown in table 8 and dividing the sum by the number of the overall channels (4096). Even considering 16 bits of resolution like in [18], instead of 12 bits, our memory would increase to 1.23 Kb only.

The Parallel OSort algorithm (POSort), presented in [28], is prototyped on both the Xilinx Spartan-6 and the Xilinx Virtex-6 devices. Table 8 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 [28]. 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%, therefore greater than our 82%. However, it couldn't 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 $\mu$s. 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 $\mu$s, three times less than the POSort.

In [36] 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, over a 4096-electrodes HDMEA). The work in [36] guarantees a latency of 2.65 ms, which is comparable to our result, i.e. 2.3 ms. The implementation requires 865 Kb of BRAM memory, 190000 LUTs, and 29000 REGs. No information about the DSP utilization is provided. Considering the significantly higher LUT utilization, it seems that 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 [37] Valencia *et al.* present 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 (up to 204 neurons), and almost fully saturating the Xilinx Artix-7 LUT resources (98%). The reported system accuracy, of about 90%, has been tested using a dataset with SNR in the range 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 HDMEAs.

In [27] 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 has been assessed to be 60-80% for signal-to-noise ratio in the range

5-7 dB. Neither resource utilization nor system latency has been provided in [27]; the reported number of channels and sorting accuracy are both lower than in our implementation.

In [38] 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 filter cut-off frequencies are 300 and 5000 Hz, respectively. Then the spikes are detected, and from the resulting spike waveforms, a technique similar to the ZCF one is used to extract the features. The features are finally sent to a PC through an RS232 interface and clustered in real-time using the K-Means algorithm in MATLAB. The system has been tested in-vivo, therefore no accuracy data is provided to compare it to our work. In addition, no utilization data are even provided in terms of LUTs, DSPs, BRAMs, and REGs.

In [39] Schäffer *et al.* present a real-time 128-channels spike sorter implemented on a Xilinx ZCU106 SoC FPGA board. The system in [39] filters the neural signals using a III-order zero-phase Butterworth Infinite Impulse Response filter and detects the neural activity by using a NEO spike detector. For each spike detected a group of $3 \times 3$ channels is considered, centered in the channel sensing the highest absolute signal amplitude. The waveforms acquired by all the 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 3-10 dB, which outperforms the accuracy obtained in our work. However, due to the increased complexity, the number of processed channels is still 32 times lower compared to the capabilities of our architecture.

## VII. CONCLUSION

We have defined a processing architecture supporting spike sorting for neural signals acquired by means of HDMEAs. Such architecture, implemented on a Z7020 APSoC, can process in real-time up to 4096 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 programmable ARM-based processing system can be exploited on-line, 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 with respect to alternatives in 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 on-silicon [40], [41]. Our spike sorter can be used to build an interface between such devices and HDMEAs, thus the integration of these two approaches will pave the way to experiments involving the co-operation of biological and on-silicon neural networks.

## REFERENCES


[1] R. C. Froemke and Y. Dan, "Spike-timing-dependent synaptic modification induced by natural spike trains," *Nature*, vol. 416, pp. 433–438, Mar. 2002.

[2] H. Kassiri, S. Tonekaboni, M. T. Salam, N. Soltani, K. Abdelhalim, J. L. P. Velazquez, and R. Genov, "Closed-loop neurostimulators: A survey and a seizure-predicting design example for intractable epilepsy treatment," *IEEE Trans. Biomed. Circuits Syst.*, vol. 11, no. 5, pp. 1026–1040, Oct. 2017, doi: 10.1109/TBCAS.2017.2694638.

[3] M. E. Spira and A. Hai, "Multi-electrode array technologies for neuroscience and cardiology," *Nature Nanotechnol.*, vol. 8, no. 2, pp. 83–94, 2013.

[4] G. N. Angotzi, M. Malerba, S. Zucca, and L. Berdondini, "A 512-channels, whole array readout, CMOS implantable probe for acute recordings from the brain," in *Proc. 37th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, Milan, Italy, Aug. 2015, pp. 877–880, doi: 10.1109/EMBC.2015.7318502.

[5] J. J. Jun *et al.*, "Fully integrated silicon probes for high-density recording of neural activity," *Nature*, vol. 551, no. 7679, pp. 232–236, 2017, doi: 10.1038/nature24636.

[6] M. E. J. Obien, W. Gong, U. Frey, and D. J. Bakkum, "CMOS-based high-density microelectrode arrays: Technology and applications," in *Emerging Trends in Neuro Engineering and Neural Computation*. Singapore: Springer, 2017, pp. 3–39, doi: 10.1007/978-981-10-3957-7_1.

[7] *BioCAM X: Large-Scale Sensing Microelectrode Array Platform for Label-Free In-Vitro Electrophysiology*. Accessed: Feb. 1, 2020. [Online]. Available: https://www.3brain.com/biocamx.html

[8] D. Tsai, E. John, T. Chari, R. Yuste, and K. Shepard, "High-channel-count, high-density microelectrode array for closed-loop investigation of neuronal networks," in *Proc. 37th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, Milan, Italy, Aug. 2015, pp. 7510–7513, doi: 10.1109/EMBC.2015.7320129.

[9] Q. R. Quiroga, "Spike sorting," *Current Biol.*, vol. 22, no. 2, 2012.

[10] G. P. Seu, G. N. Angotzi, F. Boi, L. Raffo, L. Berdondini, and P. Meloni, "Exploiting all programmable SoCs in neural signal analysis: A closed-loop control for large-scale CMOS multielectrode arrays," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 4, pp. 839–850, Aug. 2018, doi: 10.1109/TBCAS.2018.2830659.

[11] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 129–137, Mar. 1982.

[12] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, pp. 59–69, Jan. 1982, doi: 10.1007/BF00337288.

[13] R. Q. Quiroga, Z. Nadasdy, and Y. Ben-Shaul, "Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering," *Neural Comput.*, vol. 16, no. 8, pp. 1661–1687, Aug. 2004, doi: 10.1162/089976604774201631.

[14] J. Müller, D. J. Bakkum, and A. Hierlemann, "Sub-millisecond closed-loop feedback stimulation between arbitrary sets of individual neurons," *Frontiers Neural Circuits*, vol. 6, p. 121, Jan. 2013, doi: 10.3389/fncir.2012.00121.

[15] E. Biffi, D. Ghezzi, A. Pedrocchi, and G. Ferrigno, "Development and validation of a spike detection and classification algorithm aimed at implementation on hardware devices," *Comput. Intell. Neurosci.*, vol. 2010, pp. 1–15, Mar. 2010, doi: 10.1155/2010/659050.

[16] B. Sun, H. Feng, K. Chen, and X. Zhu, "A deep learning framework of quantized compressed sensing for wireless neural recording," *IEEE Access*, vol. 4, pp. 5169–5178, 2016, doi: 10.1109/ACCESS.2016.2604397.
[17] P. Yger, G. L. Spampinato, E. Esposito, B. Lefebvre, S. Deny, C. Gardella, M. Stimberg, F. Jetter, G. Zeck, S. Picaud, J. Duebel, and O. Marre, "A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings *in vitro* and *in vivo*," *eLife*, vol. 7, p. e34518, Mar. 2018, doi: 10.7554/eLife.34518.
[18] J. Park, G. Kim, and S.-D. Jung, "A 128-channel FPGA-based real-time spike-sorting bidirectional closed-loop neural interface system," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 25, no. 12, pp. 2227–2238, Dec. 2017.
[19] A. T. Do, S. M. A. Zeinolabedin, D. Jeon, D. Sylvester, and T. T.-H. Kim, "An area-efficient 128-channel spike sorting processor for real-time neural recording with 0.175 $\mu$W/channel in 65-nm CMOS," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 1, pp. 126–137, Jan. 2019, doi: 10.1109/TVLSI.2018.2875934.
[20] M. Saeed and A. M. Kamboh, "Hardware architecture for on-chip unsupervised online neural spike sorting," in *Proc. 6th Int. IEEE/EMBS Conf. Neural Eng. (NER)*, San Diego, CA, USA, Nov. 2013, pp. 1319–1322, doi: 10.1109/NER.2013.6696184.
[21] V. Karkare, S. Gibson, and D. Marković, "A 75-$\mu$W, 16-channel neural spike-sorting processor with unsupervised clustering," *IEEE J. Solid-State Circuits*, vol. 48, no. 9, pp. 2230–2238, Sep. 2013, doi: 10.1109/JSSC.2013.2264616.
[22] A. M. Kamboh and A. J. Mason, "Computationally efficient neural feature extraction for spike sorting in implantable high-density recording systems," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 21, no. 1, pp. 1–9, Jan. 2013, doi: 10.1109/TNSRE.2012.2211036.
[23] M. Saeed, A. A. Khan, and A. M. Kamboh, "Comparison of classifier architectures for online neural spike sorting," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 25, no. 4, pp. 334–344, Apr. 2017, doi: 10.1109/TNSRE.2016.2641499.
[24] S. E. Paraskevopoulouа, D. Wu, A. Eftekhar, and T. G. Constandinou, "Hierarchical adaptive means (HAM) clustering for hardware-efficient, unsupervised and real-time spike sorting," *J. Neurosci. Methods*, vol. 235, pp. 145–156, Sep. 2014, doi: 10.1016/j.jneumeth.2014.07.004.
[25] S. E. Paraskevopoulou, D. Y. Barsakcioglu, M. R. Saberi, A. Eftekhar, and T. G. Constandinou, "Feature extraction using first and second derivative extrema (FSDE) for real-time and hardware-efficient spike sorting," *J. Neurosci. Methods*, vol. 215, no. 1, pp. 29–37, Apr. 2013, doi: 10.1016/j.jneumeth.2013.01.012.
[26] J. F. Kaiser, "On a simple algorithm to calculate the 'energy' of a signal," in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, Albuquerque, NM, USA, vol. 1, 1990, pp. 381–384, doi: 10.1109/ICASSP.1990.115702.
[27] Y. Yang, S. Boling, and A. J. Mason, "A hardware-efficient scalable spike sorting neural signal processor module for implantable high-channel-count brain machine interfaces," *IEEE Trans. Biomed. Circuits Syst.*, vol. 11, no. 4, pp. 743–754, Aug. 2017.
[28] D. Valencia and A. Alimohammad, "A real-time spike sorting system using parallel OSort clustering," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 6, pp. 1700–1713, Dec. 2019, doi: 10.1109/TBCAS.2019.2947618.
[29] C. Pedreira, J. Martinez, M. J. Ison, and R. Q. Quiroga, "How many neurons can we see with current spike sorting algorithms?" *J. Neurosci. Methods*, vol. 211, no. 1, pp. 58–65, Oct. 2012, doi: 10.1016/j.jneumeth.2012.07.010.
[30] C. Gold, D. A. Henze, C. Koch, and G. Buzsáki, "On the origin of the extracellular action potential waveform: A modeling study," *J. Neurophysiol.*, vol. 95, no. 5, pp. 3113–3128, May 2006, doi: 10.1152/jn.00979.2005.
[31] S. Zaher, D. Lonardoni, F. Boi, G. P. Seu, G. N. Angotzi, P. Meloni, and L. Berdondini, "A closed-loop system processing high-density electrical recordings and visual stimuli to study retinal circuits properties," in *Proc. 9th Int. IEEE/EMBS Conf. Neural Eng. (NER)*, San Francisco, CA, USA, Mar. 2019, pp. 652–656, doi: 10.1109/NER.2019.8716913.
[32] (Jul. 1, 2018). *Zynq-7000 SoC Technical Reference Manual*. Accessed: Feb. 15, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf
[33] *K-Means++ Clustering Code*. Accessed: Mar. 2, 2020. [Online]. Available: https://rosettacode.org/wiki/K-means%2B%2B_clustering#C
[34] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proc. 18th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.
[35] *Self-Organizing Map (SOM) Kohonen Artificial Neural Network Code*. Accessed: Mar. 2, 2020. [Online]. Available: https://github.com/albertnadal/Kohonen
[36] J. Dragas, D. Jackel, A. Hierlemann, and F. Franke, "Complexity optimization and high-throughput low-latency hardware implementation of a multi-electrode spike-sorting algorithm," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 23, no. 2, pp. 149–158, Mar. 2015, doi: 10.1109/TNSRE.2014.2370510.
[37] D. Valencia and A. Alimohammad, "An efficient hardware architecture for template matching-based spike sorting," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 3, pp. 481–492, Jun. 2019, doi: 10.1109/TBCAS.2019.2907882.
[38] S. Oh, S. Han, and I. Youn, "Real-time neural signal sensing and spike sorting system using a modified zero-crossing feature with highly efficient data computation and transmission," *Sensors Mater.*, vol. 29, no. 7, pp. 1031–1042, 2017.
[39] L. Schaffer, Z. Nagy, Z. Kincses, R. Fiath, and I. Ulbert, "Spatial information based OSort for real-time spike sorting using FPGA," *IEEE Trans. Biomed. Eng.*, early access, May 28, 2020, doi: 10.1109/TBME.2020.2996281.
[40] S. Yang, J. Wang, Q. Lin, B. Deng, X. Wei, C. Liu, and H. Li, "Cost-efficient FPGA implementation of a biologically plausible dopamine neural network and its application," *Neurocomputing*, vol. 314, pp. 394–408, Nov. 2018, doi: 10.1016/j.neucom.2018.07.006.
[41] S. Yang, B. Deng, J. Wang, H. Li, M. Lu, Y. Che, X. Wei, and K. A. Loparo, "Scalable digital neuromorphic architecture for large-scale biophysically meaningful neural network with multi-compartment neurons," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 1, pp. 148–162, Jan. 2020, doi: 10.1109/TNNLS.2019.2899936.

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

**LUIGI RAFFO** (Member, IEEE) has been a Full Professor of electronics with the University of Cagliari, Italy, since 2006. In 1994, he joined the Department of Electrical and Electronic Engineering, University of Cagliari, Italy. He teaches courses on system design, digital and analog electronics design, and processor architectures. Since 2012, he has been Rector's delegate for international research projects. He was a coordinator of the Course of Studies in Biomedical Engineering, from 2006 to 2012 and from 2017 to 2018. His research topics are in the field of the study, design, and development of systems and micro-systems for application where high performance, high efficiency, and low power are required. In such a field, he is an author of more than 200 scientific articles.

**PAOLO MELONI** (Member, IEEE) has been an Assistant Professor with the University of Cagliari, since 2012. He is currently a Scientific Coordinator of the ALOHA H2020 Project. He teaches advanced embedded systems at the University of Cagliari. He is the author of a significant track of international research papers. His research activity is on the development of advanced digital systems and the application-driven design and programming of multi-core on-chip architectures and FPGAs.