Dedicated Processor for Real-time Spike Sorting

Leonardo Braga Natal
leonardo.natal@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

July 2016

Abstract

Spike Sorting consists of a technique that maps each action potential (spike) detected in an extra-cellular signal to a putative neuron. It is assumed that for the same neuron some key features of its spike shape do not significantly vary over time. Directly, its importance relies on the impossibility of analyzing each neuron in isolation, and, indirectly, on its application in neural rehabilitation systems. The goal of this thesis is to develop a dedicated processor that implements a real-time Spike Sorting algorithm. It must follow essentially two restrictions: low computational demand and low power consumption. To do so, it was studied the state-of-the-art of the main algorithms proposed to solve this problem. The relevant criteria are presented and NEO and OSort algorithms are considered the ones which meets them best. To facilitate their execution, it is developed a hardware architecture. The overall solution proved to be able to correctly detect and classify all the spikes contained in the simulated spike-train, as well as process around 100 channels simultaneously. Finally, it is addressed the possible improvements mentioned in the literature.

Keywords: real-time spike sorting, extra-cellular neural signal, neural rehabilitation, dedicated processor, ultra low power, low computational demand

1. Introduction

One of the biggest issues in every society today is the inclusion of physically disabled people. Although the last decades have seen some developments looking to improve their quality of life, the difficulties they face are still present. The challenge of equating disabled peoples life conditions with everybody else’s is a perennial problem. One possible way to mitigate the gap is to adapt the environment so it becomes more accessible for all. This would certainly be an important step forward. However, fully including disabled people in a community’s activities will not suffice. Another possibility is to provide an apparatus capable of replacing the functionality of the affected limbs. This is usually achieved through prosthetic devices. However, these prostheses still don’t enable people to perform movements as they are expected to, as they are difficult to control and frequently assume a mere aesthetic purpose.

Thus, one of the neural related areas of research that has shown the most growth recently is the area dedicated to the study of the new neural interfaced prosthetic systems. The appeal of such devices comes from the fact that they are fully integrated into the human nervous system and can provide a completely new perspective to disabled patients. Unlike static prostheses, which allow very limited to no freedom to perform movements, a neural interfaced prosthesis is expected to be fully assimilated by patients as part of their body.

Laboratory tests showed the feasibility of such an approach. However, these tests have been conducted under very specific and controlled circumstances. The current challenge consists in improving the technology so that it would be possible to incorporate this solution in a person’s daily routine. To do so, the bottleneck nowadays is not associated to the prosthesis itself, which replaces a limb, nor to the actuators on it, but to the neural interface responsible for interpreting the cortical neural activity and generating the corresponding control signals to the actuators, so that it can perform the desired action correctly. Hereupon, and considering all the aspects already discussed, the approach supported by the majority of the scientific community is to develop an implantable mechanism. Such an approach entails strong restrictions, especially regarding the power consumption and the size of the hardware. With respect to power consumption, there are two main aspects to consider. Firstly, replacing the power supply of an implantable device too often is not a desirable practice (a widely accepted replacement frequency has an order of magnitude of years). Secondly, heat dissipation, which has a particularly strict standard limit when the goal is to insert the apparatus in a place close to
the brain. In reference to the latter restriction, the size of the hardware, it should be said that this presents a more challenging scenario since reducing the size of the implantable device improves the patients experience, but it also increases the density of its heat dissipation.

Moreover, the electrodes adopted to acquire the electric extra-cellular signals have a strong influence in the whole process. On the one hand, the fact that one single electrode senses more than one neuron simultaneously is still an open question [1]. On the other hand, the techniques to build electrodes evolved significantly over the past two decades and made it possible to reduce its size and improve its sensibility. Consequently, the amount of data generated while recording neural activity is a prohibitive factor since this data has to be transmitted to the external world [4]. For instance, a 1024-channel system with a sampling frequency of 10 KSamples/second and an 8-bit resolution analog-to-digital converter requires a bandwidth of \( \sim 82 \) MBit/s, which is much higher than what is usually provided in neural prostheses (around 1 MBit/s).

### 1.1. Spike Sorting Process

Given this scenario, the primary purpose of a neural interface is to detect and interpret each action potential (spike) present in the spike-train recorded, generating enough information for the rest of the system. Additionally, it reduces the bandwidth required to transmit the necessary data. This is done by associating each spike detected to a putative neuron. Thus, it is enough to know the exact time a given spike is generated (fired) and a label corresponding to the putative neuron associated to it. In this way, it is possible to identify patterns on neural activity and use them to control the prosthesis while significantly reducing the data transmission rate and the risks to the patient. This process is known as SSP.

However, before the SSP itself, a previous step is required: the acquisition step. Currently, the general approach to do this is to amplify the signal obtained through the difference of potential between a given electrode and the pre-established ground (which is usually obtained by placing a wire under the scalp) and then apply a band-pass filter. Next, the analog signal is sampled with an A/D converter. Then, this digital signal is processed according the SSP. The literature presents many different alternatives to do so. However, the vast majority splits the whole solution into four smaller and simpler blocks with a well-defined interface between them. For this reason, this study follows the same pattern and each of these simpler blocks is evaluated separately.

Hence, the first stage of the SSP itself is the Filtering stage. Although the signal had already been filtered and converted to a digital representation, it includes a background noise captured with the information of interest, and so, requires an digital filtering stage to remove such undesirable component on the signal. At a later stage, SDA, a spike detection procedure is performed to detect any spike that is generated. Still in this stage, the spike is aligned according to some predefined criteria (e.g. zero crossing moment or the peak value). This is a critical part of the process since the correct identification and alignment of the spikes is essential to the rest of the process. Next, it is present the FE stage. In this stage that it is found the minimum set of features that allows the classification of the spikes according to the generator neuron. This is performed by assuming a strong hypothesis: the shape of all the spikes generated by the same neuron does not change significantly over time. Then, the matching of the recorded spikes with the correspondent neurons is performed in the next and final stage, Clustering. It applies techniques that group the similar spikes based on the features extracted previously. The final result is an one-to-many relation between the putative neurons and the spikes generated by them. The Figure 1 presents a diagram with these four stages and their inputs and outputs.

This study focus on the last three stages of the previous division of the whole problem. The preceding steps already have several accepted solutions addressed and the bottleneck nowadays becomes the processing of the digital spike-train.

![Figure 1: Standard approach of solving the SSP problem.](image)

### 1.2. State-of-the-Art

To the stage SDA, four classes of algorithms were considered in this study: ST, EB, TM and SWTP. The first class of algorithm to be discarded is the TM. The reason for that is the presence of a prohibitive feature: it needs previous knowledge. Although it is possible to implement a training step before running this algorithm to detect the spikes, the literature points to the fact that the spikes are not completely stationary. Thus, in real-time application, one pre-defined template would not represent an average spike after some time of execution. The other three classes, ST, EB and SWTP, were compared using four different criteria: (1) rate of detection, which functionally evaluates the performance of the classes through the percentage of the existing spikes that are detected, (2) compu-
tional complexity of its implementation, which nor-
malizes the operations required to implement each
solution using additions (each comparison operation
is equivalent to one addition, while each multipli-
cation is equivalent to ten additions), (3) capability
to be implemented in a real-time environment and
(4) the requirement for a training step. In a first
analysis, what draws most attention is the discrep-
ancy between SWTP and the other methods. It
is more than twenty times more complex than the
second place. The last two options are the ST and
EB classes. Considering the significant increase in
the percentage of detection, around 35%, and the
permissible increase in the complexity in relation to
the absolute value, the EB class of algorithm is the
one that suits the purpose of this project best.

Next, to the stage FE, five algorithms were con-
sidered. Two of them are offline algorithms but
have become a benchmark in this type of applica-
tion and are presented for comparison purposes
only. They are the PCA and the DWT [3]. The
other three are DD, IT and ZCF and were evalu-
ated according to five criteria: (1) accuracy, which
represents a functional performance, (2) complex-
ity, measuring the equivalent number of addition
operations necessary to implement each solution
(the same normalization as before), (3) capability
to be implemented in a real-time environment, (4)
the need for a training step before start executing
and (5) the necessity of updating the parameters of
the algorithm, when this is the case. Besides not
supporting real-time application, PCA and DWT
algorithms have a complexity much higher than
that of other methods (PCA requires more than
278 times the number of operations the third places
does, while the DWT more than 28 times). Apart
from them, the IT has proven to be a simple so-
lution in which there is a small constant number
of features to be used in clustering. However, it
requires a frequent update of its parameters, and
most importantly, it is significantly less accurate
than the other two alternatives presented. The DD
and the ZCF show the same accuracy as the offline
algorithms. The difference between these last two
lies in their complexity: while the DD computes
the slope of every two samples in a given spike, the
ZCF has a smaller number of features. Therefore,
the complexity of the ZCF is smaller and it is the
one that best meets the requirements of this study.

The last stage, Clustering, is the one that im-
poses the highest computational cost. Therefore, it
is particularly important to mitigate the complexity
of this part of the solution. K-means, BC, SC and
OSort algorithms were studied. Some evaluation
criteria are the same used before: Accuracy, Com-
plexity, Real-time capability. Additionally, they are
evaluated according to their capability of running
without supervision, no need for parameters and
the ability to automatically adapt their execution
to changes in the environment. It is noticeable that
the k-means would be an interesting option since it
has a relative low complexity and can be executed in
a shorter period of time. However, it is not unsup-
vised, i.e. it needs prior knowledge of the number
of clusters, which, in the present case, corresponds
to the total number of real spikes the electrode is
sensible to. On top of that, there is a more im-
portant restriction that is not met: it is not possible
to implement a real-time version of the K-means al-
gorithm because the spikes are not stationary, which
makes it possible to create a training phase to pre-
determine the number of clusters. A different stra-
 tegy is proposed in the BC. It is a statistical ap-
proach which aims to find the optimal solution as-
suming hypothesis about the correlation between
the point in the data-set. The drawback then is the
complexity of this algorithm. The higher number
of operations needed to implement this solution is
prohibitive. The same can be said about the SPC;
even though it is unsupervised and non-parametric,
its complexity, both in number of operations and
in memory terms, as well as the fact that it is an
offline solution, makes it an inadequate choice in
the context of this study. The last approach con-
considered was the OSort. It is the only one which
can be implemented to run automatically and online.
Besides, it offers a good trade-off between accuracy
and computational complexity. Apart from that,
OSort presents a competitive advantage: it does
not require a FE step since it performs the clus-
tering directly to the detected spikes. As a result,
this last algorithm is identified as the one that best
meets the needs of a real-time application.

1.3. Objective
Given the relevance of the topic, the objective of the
present thesis was to design a dedicated hardware
that implements a real-time SSP and that can be
used as the basis of an implantable neural interface
to a prosthetic device. The strategy adopted was
top-down. It starts with a high-level solution and
gradually increase its level of detail until reach a
description which is enough to be implemented in
hardware.

1.4. Report Outline
Next section describe the proposed solution to solve
the SSP. It is followed by the presentation of the
architecture to facilitate the execution of such a so-
lution. Then, the overall execution is evaluated in
different steps until the calculation of the maximum
number of channels the DSP is able to process si-
multaneously. Finally, a conclusion is presented
discussing a possible strategy to further improve-
ments.
2. Online Spike Sorting

As discussed in Section 1, the most adequate algorithm to each of the three stages addressed in this study is to apply NEO to detect and align the spikes, ZCF to extract their features and OSort to classify them. However, as described in the corresponding section, OSort does not require the features to be extracted in a previous stage and is able to handle the spikes directly. On the one hand, the fact that the spikes are described in a high-dimension space represents a drawback since to execute the classification a higher number of operations is required, as well as memory size. On the other hand, the increase in this stage is well offset by the gain of suppressing one whole stage of the solution.

The input to the NEO block is the discretized spike-train. For the purposes of this study, the solution to this block was subdivided into three parts. The first part, pre-emphasizer, focus on enhancing the spike characteristics while reducing the influence of the DC component of the signal. It consists of a non-linear filter according to the expression:

\[ \psi(x_n) = x_n^2 - x_{n+1}x_{n-1}, \]  

where \( x_n \) is the signal value at instant \( n \). The sample associated to localized high frequencies are amplified while the others are diminished. In the next part, Threshold Calculator, it is calculated the scaled mean of the signal according to the expression:

\[ Thr = C \frac{1}{N} \sum_{n=1}^{N} \psi[x_n], \]

where \( Thr \) is the threshold value, \( C \) is the threshold multiplier, \( N \) is the number of samples in the signal, \( x_n \) is the signal value at instant \( n \) and \( \psi \) is given by the Equation 1 [7]. The threshold multiplier \( C \) demonstrated to be critical. The correct selection of this parameter is essential to set the threshold above the noise and bellow the peak value of the spikes (considering the low-frequency components had already been removed). In the last part, spike aligner, a temporal windows, calculated based on the refractory period, is centered in the local maximum of the neighboring sample which values are higher than the pre-defined threshold. In this way, the same number of samples is considered before and after the peak value of the detected spike. It is important to notice that, even though the signal is pre-emphasized to identify the local maximums, the window is centered in the spike-train, preserving the original characteristics of the spikes. The influence of the window size on the final results is critical. It happens because if it is chosen a small windows, there is no enough information to differentiate spikes from different neurons and if it is chosen an excessively big windows, it may end up with samples from another spike.

After a spike is detected and aligned according to the peak value, it has to be classified. However, first, it is necessary to define how it is calculated the distance between two clusters and between a spike and a cluster. As both, the spikes and the clusters, are described in the same \( n \)-dimensional space (where the number of coordinates \( n \) is calculated based on the refractory period, as mentioned before), the Euclidean distance can be straightforward calculated:

\[ d = \sum_{n=1}^{N_{spk}} (s_n - c_n)^2, \]

where \( N_{spk} \) is the number of samples in a given spike, \( s_n \) is the value of each sample and the \( c_n \) is the cluster center. Once the distance calculation is defined, the algorithm is completely described. Whenever a spike is detected, it is calculated its distance to each of the clusters. If the distance to the closest cluster is higher than the Threshold Sorting, a new cluster is created and it is centered on that spike. Otherwise, if it is sufficiently close to one of the exiting clusters, it is assigned to it. This assignment, however, has to be followed by the update in the center of this cluster since it should take into account the influence of the new spike:

\[ c_n' = \frac{(N - 1) \cdot c_n + s_n}{N}, \]

where \( c_n' \) and \( c_n \) are, respectively, the new cluster and old cluster coordinates, the \( s_n \) is the spike coordinates and \( N \) is the number of spikes in the cluster after adding the new spike. This update in turn, may lead to a situation in which two clusters are very close to each other and should be merged. Therefore, it is necessary to calculate the distances between the clusters. Then, these distances are compared to the Threshold Merge and if they are smaller than it, the clusters are merged by computing the new cluster center through the average of the centers of each of the two clusters weighted by the number of spikes contained in each of them:

\[ c_n' = \frac{N_i \cdot c_i + N_j \cdot c_j}{N_i + N_j}, \]

where \( c_i, c_j \) and \( c_n' \) are, respectively, the \( i, j \) and new cluster coordinates, and \( N_i \) and \( N_j \) the number of spikes in cluster \( i \) and \( j \).

As can be noticed, this is an adaptive algorithm that can accommodate changes in the environment during the execution. For example, if the electrode slightly changes its position and starts to sense a new neuron, the algorithm is able to handle this situation since the only change is that a new cluster
will be created based on the shape of the spikes from this new neuron.

3. Architecture

Once the algorithm to be implemented by the DSP is defined, it is necessary to design the hardware architecture to facilitate its execution. To do so, three different scenarios are considered. The first one consists in a completely dedicated hardware. Every calculation could be optimized to share as much resource as possible. Also, it could be reduced the number of operations required in the overall execution. In this way, both requirements are met: the performance and the hardware size (which is associated to the power consumption). However, the cost-benefit related to such approach represent a prohibitive factor. The complexity derived from the software control-flow in the case of this project, especially regarding the classification stage in which the conditional blocks of execution are predominant, summed to the time necessary to design and optimize the usage of all the required resources turn this scenario into a poor option. On the other extreme, there is the possibility to employ a general purpose microprocessor to fully implement the algorithm. The most appealing feature in such approach is the enormous flexibility it offers. Such arrangement implements a hardware platform that supports a wide range of application and transfers the specific implementation to the software layer. Therefore, the control-flow issue is straightforward solved and does not require additional effort. Notwithstanding, the significant decrease in performance when executing demanding parts of the algorithm, especially the ones that corresponds to repetitive calculations, represent an serious drawback. The third scenario emerges in between the previous two as a way of composing their advantages while avoiding their weaknesses. It employs a microprocessor as the kernel of the architecture and develop additional pieces of hardware to perform specific and demanding calculations. In this way, both requirements are met. The designing cost of implementing a complex control-flow is mitigated using the general purpose microprocessor, while the critical parts of the algorithm are speed-up using specific purpose devices, the co-processors. For these reasons, this hybrid approach is the strategy adopted here.

The general purpose microprocessor selected is the MicroBlaze. It is result of a Thesis project developed in Delft University of Technology, The Netherlands [5]. The primary goal of such a project was to make available an open-source, robust and highly configurable hardware description to be utilized in a wide variety of applications. Due to strategic reasons, a light-weight implementation compatible with the architecture of the commercial processor was chosen. Additionally, the pursuit for a high performance, small implementation size and high simulation speed represent relevant aspects that make this an appealing choice. Moreover, the MB-LITE can be extended using existing Intellectual Property components, which might be considered an important feature since these components may take advantage of particular resources present in FPGAs of the same manufacture.

Next, to develop the co-processors, the main blocks of the two algorithms that compose the proposed software were characterized according to two criteria: calculation demand and control-flow complexity. These criteria are justified by their relevance when designing new pieces of hardware. The first one is associated to the gain that is obtained when moving a heavy calculation block from a general purpose processor to a co-processor designed exclusively to this task, while the second is useful to evaluate the difficulty in such designing derived from the control-flow. Accordingly, the first block to which it is designed a co-processor is pre-emphasizer. It has a heavy calculation demand and a simple control-flow, thus the gain of implementing it separately is high, while the complexity associated with the control-flow is low. Still in the NEO algorithm, the other blocks do not offer a major advantage when implemented in a dedicated hardware. The spike aligner in particular, would even imply in a high complexity device with relative low gain. The same strategy was employed to OSort algorithm. The two blocks identified as having simple control-flow and heavy calculation demand correspond to the operations of updating the cluster centers and merging them. In this case, however, one of them is a degenerated case of the other. The routine used to update a cluster center after a spike is add on it is the same routine used to merge two cluster. It is just to consider one of the clusters as having only one spike. For this reason, the design of only one co-processor is capable of handling both cases.

The data-path of Co-processor 1 is shown in the Figure 2. It calculates the pre-emphasized sample at instant \( n \) based on spike-train samples of instants \( n-1, n \) and \( n+1 \) according to the Expression 2. In this way, the result is available with one clock cycle of delay. It can be interpreted as two blocks, one sequential and on combinational. The first block corresponds to the registers that store the three samples of the spike-train. The current sample is stored in register \( rb \), while the previous sample is stored in register \( rc \) and the next sample is stored in register \( ra \). The second block is the combinational block and consists of three functional units, two multipliers and one subtractor. Since the co-processor aims to speed-up its execution time at
most, the multiplications are implemented through partial products instead of a sequential approach. In this way, it takes advantage of the parallelism of this operation. Also, the subtractor is implemented with an adder and one inverter. Thus, it is calculated the complement of the subtrahend and the carry-in of the adder is set to ‘1’. It is executed synchronously to the rising edge of the clock signal and the bases of its operations is the update of the registers. Thus, when a new rising edge of the clock is detected, the content from register \( r_b \) is copied to register \( r_c \) and the same id done from \( r_a \) to \( r_b \). Then, the income sample is copied to register \( r_a \). It is important to mention the hypothesis assumed in this control unit: the interval of time required to update the registers summed to the interval of time required to the combinational block generate an stable result signal at the output is smaller than one clock period.

The data-path of Co-processor 1 is shown in Figure 2. The new cluster coordinates are calculated according to Expression 5. As discussed previously, however, it can also be applied to the update the cluster centers, which is a particular case. The four operands required to calculate the coordinates of the new cluster’s coordinates are: the coordinates of cluster 1 and cluster 2, corresponding respectively to registers \( c_1 \) and \( c_2 \), and the number of spikes in each cluster, corresponding to registers \( n_1 \) and \( n_2 \). Like in the Co-processor 1, it can be interpreted as composed by a sequential block together with a combinational one. The first one is formed by the four registers, while the second is formed by four functional units: two multipliers and two adders. The multipliers are also implemented through partial products as before. The difference lies on the divider. This is the critical unit of the Co-processor 2. The division operation was implemented through a sequential algorithm since a combinational approach would lead to a prohibitive number of hierarchical levels, i.e. the overall delay would be the sum of the delays in the worst case which contains many chained gates. Then, with a sequential algorithm, the number of clock periods required to obtain a stable signal at the output is not constant and depend on the operands. It is also executed synchronously to the rising edge of the clock signal and its basis of operations is the update of the registers. Nonetheless, in this case the operands are independent. Thus, when a new rising edge of the clock is detected, the control unit checks the address signal to identify which operand is being written. It is important to notice that for simplify the drawing the control-unit as well as its signals (e.g. address signal mentioned before) are omitted in the Figure 3. Thereby, the trigger to start the coordinate update is the write in the \( c_2 \) register. Once it is done, the combinational blocks generates the operands to the divider, which sets its output valid signal to ‘0’ until the result is available on its output, when this signal changes to ‘1’ and the co-processor finishes its execution.

The data-path of Co-processor 2 is shown in Figure 3. The final architecture including the MB-Lite and the two co-processors is presented in Figure 4. The general purpose microprocessor is responsible for executing the algorithm codified in the Instruction Memory (ROM). Additionally, it delegates the execution of the critical parts of this algorithm to the co-processors. It is done through the mapping of these devices in the memory address of the microprocessor. In this way, the Memory Decoder is responsible for for splitting the whole memory map between the two co-processors and the Data Memory. It is worth it to mention that the spike-train signal is accessible from both, the Co-processor 1 and the MB-Lite. It is justified because the first pre-emphasizes the signal while the second uses it to identify the spikes as well as copy it directly from the original signal. Additionally, the Co-processor 2 is invoked whenever a new spike is assigned to an existing cluster or two clusters are merged. Besides sharing the memory map with the co-processors, the Data Memory stores all the meta-data required in the whole execution, as well as the detected and classified spikes. Although it is omitted the control-signals in the figure, all components are synchronous and sensible to the rising edge of a clock signal.
4. Evaluation of the Solution

The proposed DSP is assessed in several steps. To do so, it is processed spike-train available in the literature. First, in the functional validation, the results of the implementation in the high-level programming language is discussed, which is followed by the results for the validation implemented in the lower-level description. Then, the solution including the hardware architecture is evaluated according to the execution time required to process the subset of the database. In this last case, two scenarios are considered. The first scenario executes the whole algorithm only in the microprocessor, while the second includes the two co-processors designed. It is still addressed the execution time of the solution divided in routines and subroutines as well as a discussion in which the execution time evaluation is extrapolated to calculate how many different spike-trains the proposed DSP is able to handle in simultaneous. Finally, as explained before, the final goal is to maximize the number of channels the processor is able to process simultaneously without missing the strict requirements for such a type of device. In this fashion, the implementation of further improvements and simplifications suggested in the literature are considered in the end of this chapter.

4.1. Spike-train Processed in the Simulation

The spike-train used to simulate the solution was extracted from a database proposed by Quiroga et al. 2004 in [6]. Due to limitations in the simulation infrastructure, a subset of the whole database was simulated, corresponding to the first 4,096 samples. It was built using 594 different average spike shapes obtained from real recordings of the neocortex and basal ganglia. First, it was generated the background noise. To do so, these spike shapes were randomly selected and superimposed at random instants of time for half of the times of the samples. Then, three of these shapes were selected to represent the actual spikes to be detected in the simulated signal and later classified. They were superimposed in random times, each of them with a Poisson distribution of the inter-spike interval time, i.e. the interval of time between two spikes from the same neuron. It is used a 20 Hz mean of firing rate and their amplitude were normalized to have the peak value equal to 1. The SNR is 0.05dB and was generated using the standard deviation. In a first moment, the data was simulated with a sampling rate of 96 kHz and using interpolation to the waveform of the spikes. Later, it was down-sampled to 24 kHz sampling rate to simulate the real conditions, in which the peak of the spike may not be aligned with a sample, i.e. it happens between two samples. It is important to notice that constructing the signal in this way reproduces the biological variability in the spike shape [2]. Additionally, there is no need for the filter stage because the data was already filtered.

4.2. Functional Validation

As mentioned before, the first simulation aimed to validate the functionality of the proposed algorithms. NEO was implemented using the scaled mean threshold multiplier $C$ (see Equation 2) equal to 8. This value is suggested in the literature and proved to be enough to avoid the noise and maintain the sensibility to detect the spike peak value. Additionally, the window size was calculated according to the refractory period of a neuron and to the sampling rate applied to acquire the signal recorded in the database. It leads to 49 samples and the window was centered around the highest value of the spike. The original spike-train can be seen seen in Figure 5, while the pre-emphasized one with the threshold used are presented in Figure 6. It is possible to notice the correspondence between the spikes described in both signal, before and after the application of the non-linear filter. Also, the fluctuation in the low-frequency components is drastically attenuated. Later, after identifying the local maximas, the spikes are recorded from the original signal. At the end, the NEO function returns all the spikes contained in the subset of the database. This is shown in Figure 7. From inspection of the spike shapes, three classes of spikes are distinguished. It corresponds to the three templates used to generate the database. The next step was to cluster these spikes. It was implemented the original version of the OSort algorithm, i.e. using euclidean distance and dynamic cluster centers calculations. This algorithm, like the previous one, has a critical parameter settings. It needs two predefined thresholds, TS and TM, which are crucial to the performance of the system since these parameters define how a given spike is assigned to a cluster and how the clusters are merged. Their best value are defined empirically.
and, in the case of this database, they are equal to 1.8 and 1.5, respectively. The classification result is presented in the Figure 8. Comparing with the description in [6], the two functions together were able to correctly detect and cluster the simulated spikes. There is a correspondence between the final cluster of the spikes and its shapes shown previously. The three different classes of spike generated three different clusters, as well as the quantity of spikes per cluster was the same for all of them.

Figure 5: Spike-train digital signal.
Figure 6: Pre-emphasized signal.
Figure 7: Detected spikes.
Figure 8: Cluster Histogram.

The next description, in C programming language, helped to detail the solution. As in the previous case, it implements two functions, one for each of the two stages considered in this study. The major difference is the memory management and the operation over numerical real variables. The simulation shows a slight difference in the cluster center’s position calculation. This is explained due to the use of fixed-notation to model real numbers, which is not as precise as a floating point variable. After several operations this difference can already be noticed. However, the simulation proves it does not interfere in the final result as this lower-level description is able to correctly detect and align all spikes as well as classify them.

4.3. Execution Times
On the next step, it is simulated the previous solution being executed in the hardware architecture description developed for the DSP. The execution times are extracted from this simulations. First, the machine code was obtained compiling the code from language C to the commercial architecture MicroBlaze is applied. Next, the spike-train input is emulated in memory. In this way, the samples are first pre-processed and adapted to the data format the MB-Lite microprocessor is able to interpret. After that, they are utilized to pre-initialize the data memory (Memory RAM), as well as another smaller memory encapsulated inside the Co-processor 1, which also needs this input signal to perform the pre-emphasis step. Additionally, it is created a special routine to save all the output from the two functions in the data memory as well. It includes the detected spikes, their arrival time and the final cluster centers as well as the spikes assigned to them. Considering the trade-off between the performance and heat-dissipation, the clock signal frequency is defined to 1GHz. This is an intermediate value that can vary depending on the technology this description is implemented.

First, it is simulated the execution of the whole solution only in the microprocessor. The Table 1 presents the results for each routine and their subroutines. NEO represents approximately 60% of the total execution time. The reason for this is the necessity of pre-emphasize every sample in the spike-train. In turn, the OSort algorithm processes only the spikes detected previously, which leads to 35% of the total execution time. Apart from that, there is the Save Results routine. It, however, does not have a significant execution time. Considering the NEO subroutines, once again, due to the requirement of processing every sample and for each of them execute a complex calculation, the pre-emphasizer & threshold step corresponds to approximately 80% of the total time required to execute the NEO algorithm. The 20% remaining corresponds to the detection and alignment that only consider the pre-emphasized samples that are higher than the pre-defined threshold and, among them, process only the ones that are local maxima. Following the same pattern, the OSort algorithm is unbalanced between the two big blocks it consists of. Spike Assignment corresponds approximately to 75% of the total execution time while Cluster Merging corresponds to the rest 25%. Analogously, it can be explained by the necessity for assigning every spike, while the cluster merging block only require demanding calculations when there are two clusters very close to each other.

The second scenario in which the solution is evaluated includes the two co-processors. The code snippets responsible for implementing the calculations are replace to memory accesses in order to interact with the other devices. The execution times for this second scenario is presented in Table 2. NEO and OSort routines are divided as before. The subroutines affected by integration of the co-
processors into the architecture are Pre-emphasis & threshold, Spike Assignment and Cluster Merging. In the first, the execution time is reduced in more than 70%, while in the second and third in more than 40% and almost 30%, in the second and third, respectively. It demonstrates the relevance of the development of these specific purpose devices to replace the general purpose microprocessor in demanding operations such as calculation. Besides the reduction, the execution is now more balanced. The time the NEO algorithm takes to run is slightly higher than the time OSort does. This is an important aspect since in a real-time system the different stages should be as balanced as possible to avoid bottlenecks. The same characteristic can be seen in NEO subroutines: the reduction in the Pre-emphasis & threshold block promoted a more balanced routine. The exception is the OSort algorithm. Even though it had a reduction of almost 40% in the overall routine execution time, it was marginally reflected in the balance between the two subroutines. This is explained due to the use of the Co-processor 2 in both parts of the algorithm. As explained before, the Co-processor 2 is used to speed up the OSort routine in two different parts and in the case of this evaluation, one of them is in the Spike Assignment while the other is in Cluster Merging.

Finally, regarding the speedup promoted by the two co-processors in relation to the previous architecture, the values are 2.4 and 1.6 for NEO and OSort algorithms, respectively. Thus, discarding the secondary routines, the overall speedup is 2.1 (the NEO routine has a longer execution time).

4.4. Simultaneous Spike-train Processing

Under the aforementioned requirements, the ultimate purpose of this study is to maximize the number of channels the proposed DSP is able to process simultaneously without losing important information. Thereby, in order to calculate this number, the first assumption taken is related to the real-time operation: the execution time to process a spike-train signal is directly proportional to the number of spikes contained in it. This is an appropriate approximation since the major demanding operations are done when a spike is detected. Besides, the process associated to every sample is performed by Co-processor 1, which is optimized to this task and does not represent a bottleneck. Then, the next step is to consider the minimum interval of time the same neuron generates an spike, the refractory period. The time considered in the estimation presented here is 5 ms. It means that, once the DSP finishes to process one channel, it has 5 ms to process all the other channels. Also, from Table 2, the total time to execute the whole algorithm is 523,447 ns and the number of spikes detected is 10 (the time required to initialize the microprocessor as well as the time necessary to save the results are not considered here). Therefore, the DSP needs approximately 50 ns per spike to detect, align and classify it. In turn, the maximum number of channels this architecture is able to handle is around 100 channels.

5. Conclusions

The main aspects of the SSP were discussed. Its importance relies on the necessity of interpreting the neural activity to facilitate final-systems. Among them, the present study focused on neural interfaced prosthetic devices, which distinguishes from the others due to the possibility of substantially increase the quality of life of those who suffer from a disability. For this reason, the main algorithms available in the literature related real-time SSP were considered. The limitation in size and power consumption constrains the solution and leads to the application of NEO as spike detector and aligner, due to its relative simplicity and good accuracy. It also leads the application of OSort to perform the classification due to its relative simplicity as before but also because there it suppresses the need for an extra stage, the feature extraction. This solution was evaluated throughout many steps. First, a functional validation was done in order to guarantee this combination is able to correctly process the neural signal. The results obtained demonstrated the viability of this solution: all the spikes contained in the neural signal processed were correctly detected and classified. Then, the algorithm was refined to a more detailed description. Memory management and numerical modeling issues were
explicitly handled. The simulation results of this last version of the algorithm showed less precision in the calculation of the final clusters’ center when compared to the previous one. However, it keep the capability of detecting and correctly classifying all the spikes in the input signal. Furthermore, as the desired result at the end is just the instant of time a spike is detected and a label corresponding to its neuron, this drawback in the clusters’ center position can be ignored.

Next, to facilitate the execution of the algorithm, it was designed an specific hardware architecture. It was adopted an hybrid approach that combines the flexibility of a general purpose microprocessor with the performance of a dedicated device in demanding calculations. The MB-Lite microprocessor is chosen to be the kernel of the architecture. It has an open-source code, which means that the original description can be adapted to the needs of this study. Additionally, as it implements a commercial architecture widely used, there is a fully functional compiler available. Regarding the extra devices, it were identified three parts of the algorithm with both, demanding computational calculations and simple control-flow. The first device, Co-processor 1, is applied in one of these parts: the pre-emphasizing step in NEO routine. On the other hand, the second device, Co-processor 2, is applied in the other two parts: update of a cluster center and cluster merging, both in OSort routine.

The simulation of the overall solution with the two co-processors showed an speedup of more than 2.4 for the NEO algorithm and more than 1.6 for the OSort algorithm in relation to the scenario in which the solution is executed only in the general purpose microprocessor. It demonstrates the importance of having specialized pieces of hardware compounding the architecture. Moreover, the adoption such devices promoted a more balanced system. None of the block of the algorithm represent a bottlenecks. In the end, the number of different spike-trains this solution is able to process simultaneously is found to be around 100 channels. This is a relevant number considering the size and ultra-low power requirements imposed in the beginning. It is worth mentioning that the final power-consumption associated to this solution depends on the technology it is implemented in.

Finally, it is mentioned some possible improvements suggested in the literature. Altogether, three modifications are discussed. The first one is the substitution of the Euclidean distance definition for the absolute sum of the difference in each coordinate. It came to be implemented but the first validation proved it to have an prohibitive impact on the final accuracy when classifying the spikes. The second proposes to change the threshold calculation. It is justified through the assumption that the amplitude of the spikes does not change over time. Last of all, the third assumes that after some time of execution a given cluster contain so many spikes that its center position is very close to the ideal point, therefore it is not necessary to update this point anymore, nor to check if two clusters are too close to each other. In this way, the potential gain associated to these last two modifications in the algorithm is calculated according the maximum number of channels it could process. This number is found to be around 170 channels. It would represent a speedup of 1.7, which means that this is a promising strategy for the future work.

References


