sensors

MDPI

Article

A Low Cost VLSI Architecture for Spike Sorting Based on Feature Extraction with Peak Search

Yuan-Jyun Chang †, Wen-Jyi Hwang *,† and Chih-Chang Chen

Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei 117, Taiwan; 60347009s@ntnu.edu.tw (Y.-J.C.); 60447062s@ntnu.edu.tw (C.-C.C.)

* Correspondence: whwang@csie.ntnu.edu.tw; Tel.: +886-2-7734-6670

† These authors contributed equally to this work.

Academic Editors: Andrew J. Mason, Haitao Li and Yuning Yang

Received: 3 October 2016; Accepted: 2 December 2016; Published: 7 December 2016

Abstract: The goal of this paper is to present a novel VLSI architecture for spike sorting with high classification accuracy, low area costs and low power consumption. A novel feature extraction algorithm with low computational complexities is proposed for the design of the architecture. In the feature extraction algorithm, a spike is separated into two portions based on its peak value. The area of each portion is then used as a feature. The algorithm is simple to implement and less susceptible to noise interference. Based on the algorithm, a novel architecture capable of identifying peak values and computing spike areas concurrently is proposed. To further accelerate the computation, a spike can be divided into a number of segments for the local feature computation. The local features are subsequently merged with the global ones by a simple hardware circuit. The architecture can also be easily operated in conjunction with the circuits for commonly-used spike detection algorithms, such as the Non-linear Energy Operator (NEO). The architecture has been implemented by an Application-Specific Integrated Circuit (ASIC) with 90-nm technology. Comparisons to the existing works show that the proposed architecture is well suited for real-time multi-channel spike detection and feature extraction requiring low hardware area costs, low power consumption and high classification accuracy.

Keywords: spike sorting; VLSI; brain machine interface

1. Introduction

There is an increasing demand in data-acquisition systems for neurophysiology to record simultaneously from many channels over long time periods [1]. These experiments accumulate large amounts of data, which would be processed by spike sorting systems for analyzing the activities of neurons. A typical spike sorting system [2,3] usually involves complicated feature extraction and spike classification operations for separating spikes from background noise and clustering the detected spikes. A large amount of spike trains would impose heavy computational load for a software spike sorting system, resulting in a long processing time.

One approach to reduce the computation time is to implement a spike sorting system by hardware. Hardware systems offering dedicated circuits substantially outperform their software counterparts in terms of computational performance. Hardware solutions are beneficial for neurophysiological signal recordings and analysis where real-time computing is crucial. There are two hardware approaches: Field Programmable Gate Array (FPGA) [4] and Application-Specific Integrated Circuit (ASIC) [5]. Some FPGA circuits [6,7] possess high area complexities and/or power consumption, which may be suited only for offline processing. On the contrary, ASIC architectures may have lower area costs and power dissipation. Many ASIC-based spike sorting implementations are then proposed for in vivo applications, where area- and power-efficient design is desired.

Many ASIC architectures based on Principal Component Analysis (PCA) [8–10] have been proposed for hardware spike sorting. Although they are effective for feature extraction, the inherent complexities for the computations of the covariance matrix and eigenvalue decomposition in the PCA algorithm may impose high hardware and power costs. The PCA variants, such as the Generalized Hebbian Algorithm (GHA) [11], are able to reduce the hardware costs by lifting the requirements for covariance matrix computation. In the GHA, the principal components are updated incrementally based on a set of training data. Nevertheless, its iterative training procedure may still be a bottleneck for online applications.

Alternatives to PCA and GHA include techniques such as discrete derivatives [12,13], integral transform [14] and zero-crossing [15] for feature extraction. The techniques feature low computational costs without additional training procedures. Nevertheless, some algorithms may be prone to noises due to the simple feature extraction procedure without taking noise into consideration. In addition, some of the algorithms have not been implemented by hardware. The effectiveness of the algorithms for ASIC implementation as compared with other techniques may still need to be evaluated.

The objective of this paper is to present a novel ASIC implementation for spike sorting featuring high classification accuracy, low area costs and low power consumption. The architecture is based on a novel feature extraction algorithm with low computational complexities. In the feature extraction algorithm, the location of the global minimum (or maximum) of a spike is first identified. Based on the location, the spike is then separated into two portions. The area of each portion is then used as a feature. Similar to the algorithms in [12–15], the proposed algorithm is simple to implement. In addition, it may be less susceptible to noise interference. Observe that the variations in the location of the global minimum (or maximum) of a spike due to noises may be small provided that the other local minimal (or maximal) values are significantly larger (or smaller) than the global one. This may be the case for some spike waveforms. The algorithm may then provide high immunity to noise corruption. Furthermore, the area of each position may be divided by the distance between the global maximum and minimum to enhance the classification accuracy.

A novel VLSI architecture is also presented for the novel feature extraction algorithm. In the architecture, the search for the global minimum and the computation of the area of portions separated by the minimum value are carried out concurrently. This is beneficial for pipelining operation for enhancing the throughput of the feature extraction. To further expedite the process, the local peak search and area computation over smaller segments of the spike can be carried out first. The local peak values and areas are subsequently merged with the global ones by a simple hardware circuit. In addition, due to its simplicity, the architecture can be operated in conjunction with other spike detection circuits. In this work, the circuit based on the Non-linear Energy Operator (NEO) [16] is employed for the multi-channel spike detection. To minimize the power consumption and area costs of the circuits, all of the channels share the same core for spike detection and feature extraction operations. A Clock Gating (CG) technique [17] is also employed to supply the system clock only to the active components of the circuit. A number of ASIC implementations are presented to demonstrate the effectiveness of the proposed architecture. Experimental results reveal that the proposed architecture is an effective alternative for in vivo multi-channel spike sorting with high classification accuracy, low power dissipation and low hardware area costs.

The remaining parts of this paper are organized as follows. The proposed feature extraction algorithm and the corresponding VLSI architecture are presented in Section 2. The architecture of the multi-channel spike sorting system supporting both the NEO and the proposed algorithm is presented in Section 3. Section 4 then evaluates the performance of the spike sorting system. Finally, Section 5 includes some concluding remarks.

2. The Proposed Feature Extraction Algorithm and Architecture

2.1. The Algorithm

2.1.1. The Proposed Algorithm for the Feature Extraction of Spikes

This section presents a novel algorithm for the feature extraction of spikes. Consider a spike $\mathbf{x}$ with length N, where $x_i, i = 1, ..., N$, is the i-th sample of $\mathbf{x}$. Let $i_{\min}$ and $i_{\max}$ be the index of the global minimum and maximum of $\mathbf{x}$, respectively. That is,

$$i_{\min} = \underset{1 \le i \le N}{\operatorname{argmin}}\, x_i, \quad i_{\max} = \underset{1 \le i \le N}{\operatorname{argmax}}\, x_i. \tag{1}$$

The index $i_{\min}$ (or $i_{\max}$) is used to separate the sample indices of spike $\mathbf{x}$ into two intervals $[1, i_{\min}]$ and $[i_{\min} + 1, N]$. Let a_1 and a_2 be the two features based on $i_{\min}$. They are computed by:

$$a_1 = \sum_{i=1}^{i_{\min}} (x_i - x_{i_{\min}}), \tag{2}$$

$$a_2 = \sum_{i=i_{\min}+1}^{N} (x_i - x_{i_{\min}}). \tag{3}$$

The a_1 and a_2 can then be viewed as the area of the spike in the two intervals with $x_{i_{\min}}$ as the reference level. Alternatively, $x_{i_{\max}}$ can be used as the reference level. In this case, the separation of $\mathbf{x}$ is based on $i_{\max}$, and the corresponding features, denoted by b_1 and b_2, are given by:

$$b_1 = \sum_{i=1}^{i_{\max}} (x_{i_{\max}} - x_i), \tag{4}$$

$$b_2 = \sum_{i=i_{\max}+1}^{N} (x_{i_{\max}} - x_i). \tag{5}$$

In addition to the areas, the difference between $i_{\min}$ and $i_{\max}$ could be a feature beneficial for spike sorting. One way to incorporate the feature is based on the division as shown below.

$$f_i = \begin{cases} a_i/(i_{\min} - i_{\max}) & \text{when } x_{i_{\min}} \text{ is the reference,} \\ b_i/(i_{\min} - i_{\max}) & \text{when } x_{i_{\max}} \text{ is the reference,} \end{cases} \tag{6}$$

where $f_i, i = 1, 2$, are the final features produced by the proposed algorithm.

An advantage of the proposed algorithm is that the feature vectors extracted by the algorithm may not be susceptible to noise interference. A contributing factor to the noise robustness is that the variations of the locations of peaks (i.e., $i_{\min}$ or $i_{\max}$) may not be high even for large background noises. For example, Figure 1 shows the locations and values of peaks of spikes corrupted by background noises with different SNR levels. Each spike contains 64 samples (i.e., $N = 64$). The spike data for the figure is obtained from the simulator developed in [18]. It can be observed from Figure 1 that the location variations are small for all of the SNR levels considered in the figure. In particular, when SNR = 1 dB, the location of the minimum value is 41, as shown in Figure 1d. From Figure 1a, we see that the true location of the minimum value is 38. Because the length of spikes is 64, the variation is only 4.68%. This may be beneficial for maintaining low variations of f_i in Equation (6) in the presence of noise.

The variations of the locations of peaks may also be small when a spike is overlapped by another one. Examples of spikes with different degrees of overlapping are revealed in Figures 2–4. To facilitate the observation, there is no background noise corruption in the spikes. In the examples, the location $i_{\min}$ of a spike is used to split the indices of spike samples into two intervals $[1, i_{\min}]$ and $[i_{\min} + 1, N]$.

For the sake of simplicity, we let $J_1 = [1, i_{\min}]$ and $J_2 = [i_{\min} + 1, N]$. In addition to the locations and values of peaks, the J_1 and J_2 are marked for each spike in the figures. This is beneficial for the observance of the impact on the extraction of feature vectors due to spike overlapping.

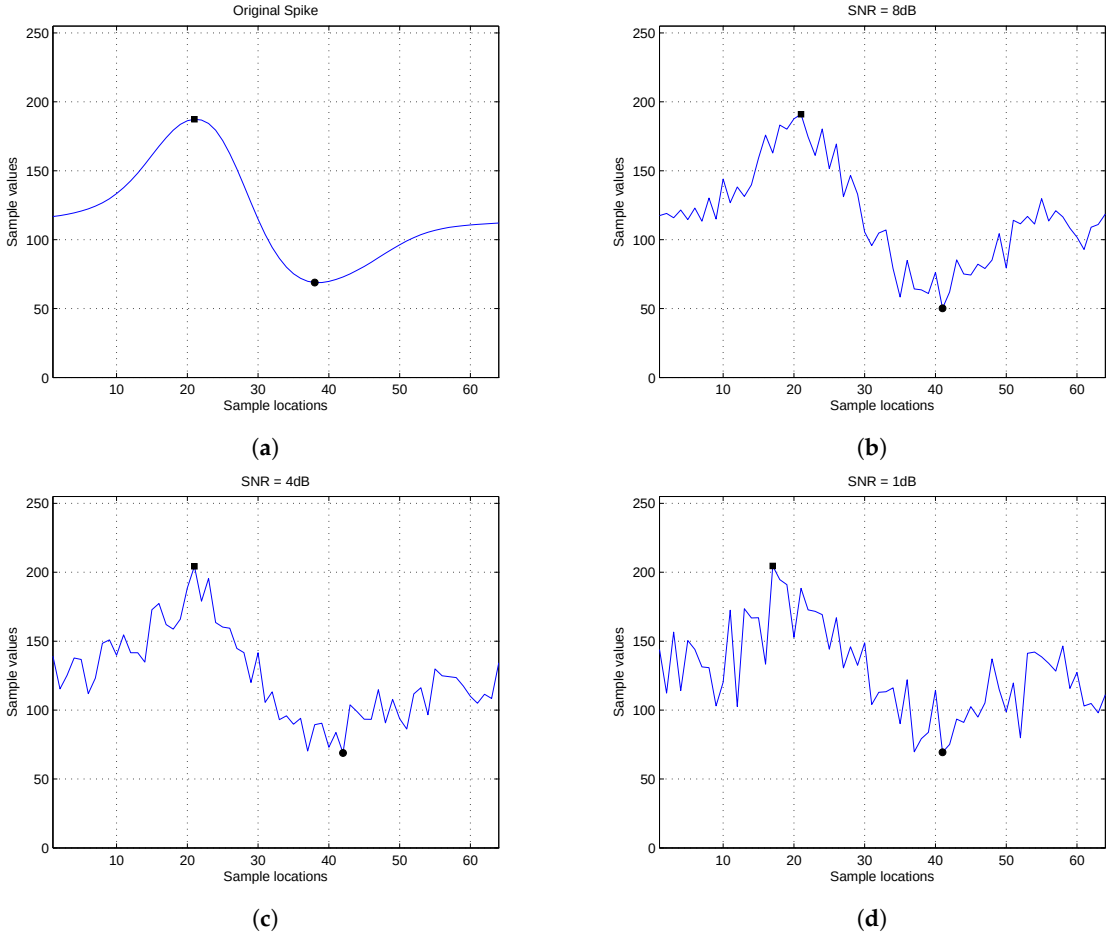

Figure 1. The locations and values of peaks of spikes corrupted by background noises with different SNR levels, where the black square and circle markers represent the maximum and minimum of spikes, respectively. (**a**) The original spike; (**b**) SNR = 8 dB; (**c**) SNR = 4 dB; (**d**) SNR = 1 dB.

We first consider the example without spike overlapping, as shown in Figure 2. There are two spikes in the example. The first and the second spike, denoted by Spike 1 and Spike 2, are located in the first 64 samples and final 64 samples in Figure 2a,b, respectively. Therefore, after the combination, the waveform of each individual waveform remains unaltered, as revealed in Figure 2c. That is, the peak locations stay the same, and the J_2 of Spike 1 is not overlapped with the J_1 of Spike 2. The feature vectors extracted from these two spikes will not be changed after the combination.

In the next example, Spike 2 is shifted leftward by 16 samples. Because the length of each spike is 64 samples, the Spike 1 and Spike 2 are overlapping by 25%, as shown in Figure 3. In this case, the combination of these two spikes introduces a slight distortion to each individual spike, as revealed in Figure 3c. Nevertheless, it can be observed that the variations in peak locations are small for each spike. Therefore, J_1 and J_2 can still be identified accurately for each spike. We can also see from Figure 3c that the J_2 of Spike 1 is overlapped with the J_1 of Spike 2. That is, one of the two areas separated by the $i_{\min}$ of Spike 1 also belongs to Spike 2, and vice versa. The a_2 of Spike 1 and a_1 of Spike 2 may then not be accurately extracted. However, because the remaining intervals (i.e., J_1 of Spike 1 and J_2 of Spike 2) are still non-overlapping intervals, the a_1 of Spike 1 and a_2 of Spike 2 can be computed accurately. The feature vectors may still be useful for subsequent spike classification.

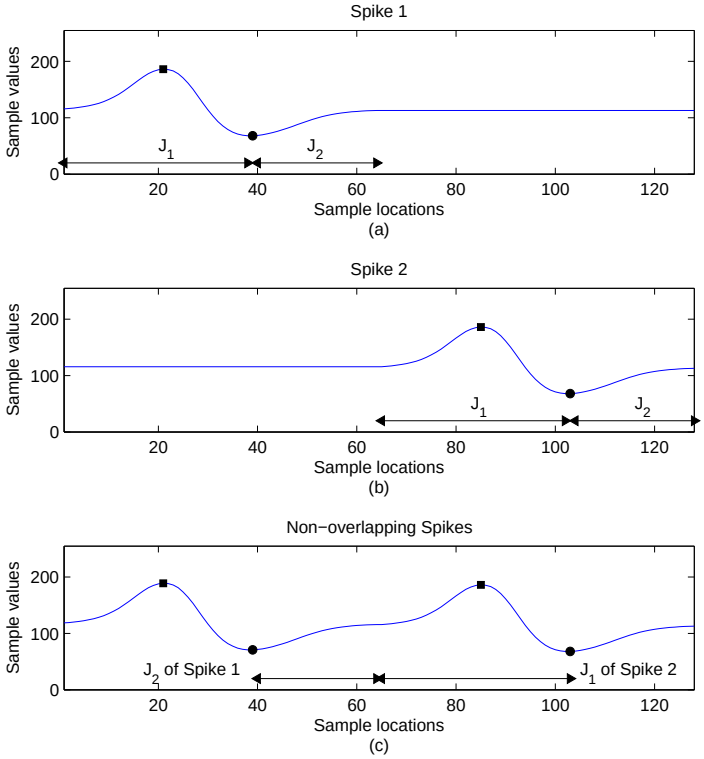

Figure 2. An example of two non-overlapping spikes for feature extraction. (**a**) Original Spike 1; (**b**) original Spike 2; (**c**) resulting waveform after the combination of the two spikes.

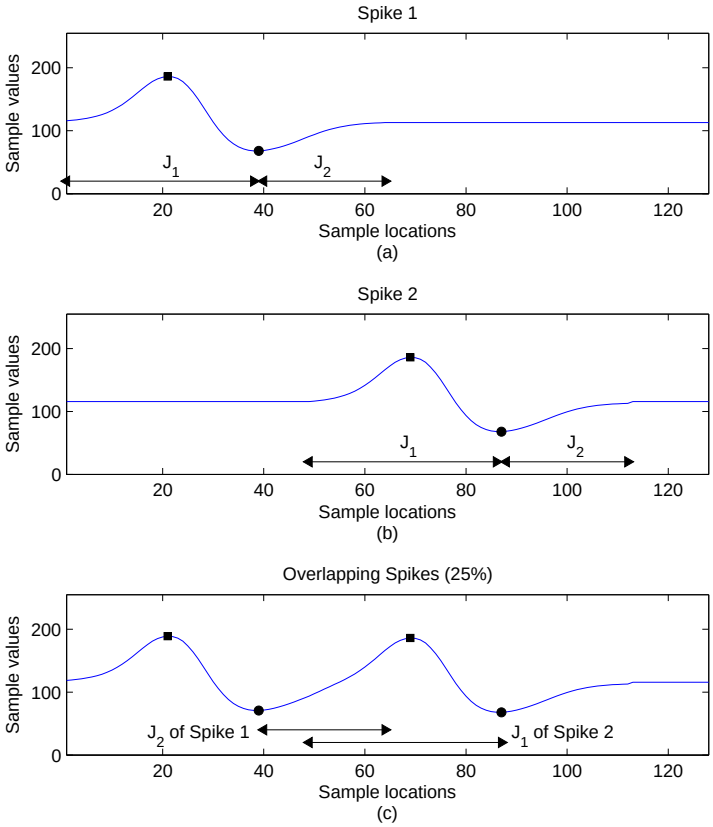

Figure 3. An example of two spikes overlapping by 25% for feature extraction. (**a**) Original Spike 1; (**b**) original Spike 2; (**c**) resulting waveform after the combination of the two spikes.

Figure 4 shows the third example, where Spike 1 and Spike 2 are overlapping by 50%. We can see from Figure 4 that the peak location variations are still small even for this case. This is helpful for the successful identification of J_1 and J_2 for each spike. Due to the large area overlapping, the distortion of spike waveforms may be visible. This is particularly true for Spike 1 by comparing the waveform of the first 64 samples in Figure 4a with that of Figure 4c. This can also be further confirmed by observing from Figure 4c that both the J_1 and J_2 of Spike 1 are overlapped with both J_1 of Spike 2. The correct classification of Spike 1 may then be difficult. Nevertheless, the successful classification of Spike 2 may still be possible because its J_2 is not overlapped with the intervals of Spike 1, and accurate extraction of a_2 and subsequently classification for Spike 2 may still be successful.

Finally, when Spike 1 and Spike 2 are overlapping by more than 50%, large distortion of spike waveforms may be observed. The deviations of peak locations from their original ones before spike combinations may then be large, resulting in inaccurate identification of J_1 and J_2. In addition, larger overlapping of J_1 and J_2 of both spikes is possible. Consequently, accurate extraction of a_1 and a_2 of Spikes 1 and 2 may become more difficult in this case.

In summary, based on the results provided by the above examples, we see that the proposed algorithm may be robust to noise interference. The locations of peaks may not be susceptible to background noises and spike overlapping. The variations may be small even when the SNR = 1 dB and/or the degree of spike overlapping is up to 50%. The proposed algorithm is based on the locations of peaks for the computation of feature vectors. The robustness of the peak locations observed from the examples is then beneficial for providing useful insight into the effectiveness of the proposed algorithm.

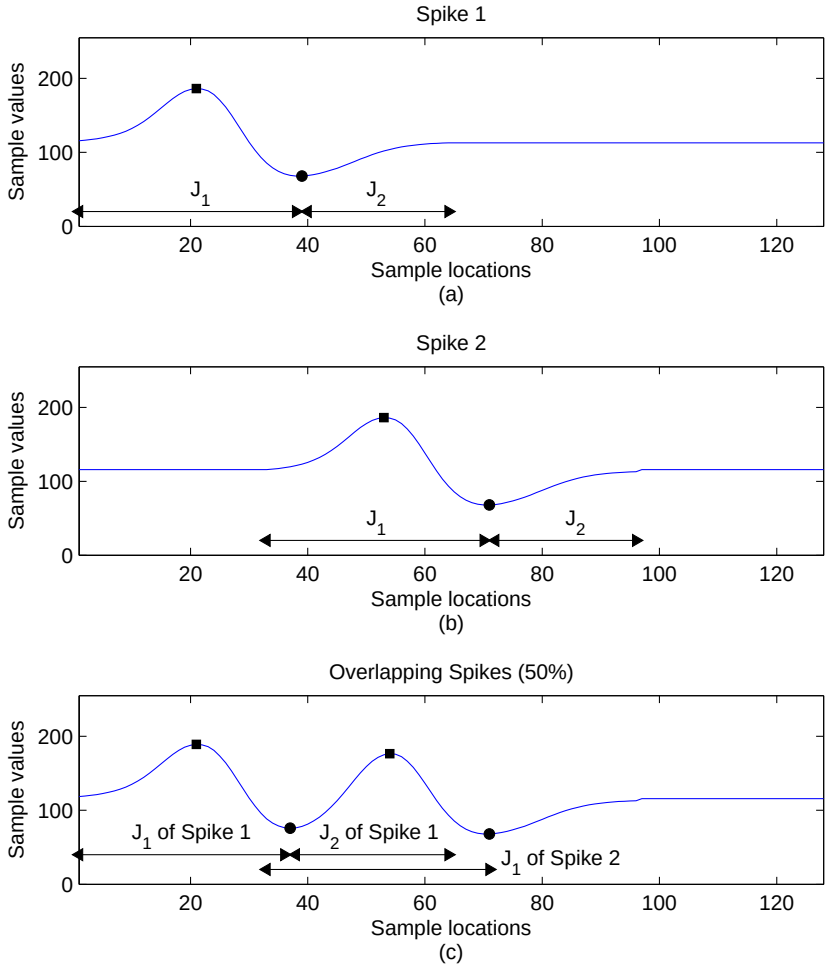

Figure 4. An example of two spikes overlapping by 50% for feature extraction. (**a**) Original Spike 1; (**b**) original Spike 2; (**c**) resulting waveform after the combination of the two spikes.

2.1.2. Operations of the Proposed Architecture Based on the Algorithm

From Equation (6), we see that the computation of f_i is dependent of the selection of $x_{i_{min}}$ or $x_{i_{max}}$ as the reference. For the sake of simplicity, only the case where $x_{i_{min}}$ is the reference is considered for the hardware implementation. The hardware design for the other case can be followed in a similar fashion.

One direct approach for the hardware implementation of the proposed algorithm is to first identify $i_{\min}$ and $i_{\max}$ from the spike $\mathbf{x}$. The computation of a_i and f_i is then followed. Although the approach is simple, it is necessary to store the spike $\mathbf{x}$. This is because the spike $\mathbf{x}$ is first used for finding $i_{\min}$ and $i_{\max}$, and then, it is re-used again for computing $a_i, i = 1, 2$. As a consequence, the latency of the circuit may be long. In addition, the circuit may need a buffer to store $\mathbf{x}$. When the dimension N of $\mathbf{x}$ is large, the area costs may be high.

The proposed architecture is able to alleviate the drawbacks stated above. It concurrently computes $i_{\min}$, $i_{\max}$ and $a_i, f_i, i = 1, 2$. To carry out the concurrent operation, a novel incremental approach is proposed. Before presenting the approach, we first note that a_i, $i = 1, 2$, in Equations (2) and (3) can be rewritten as:

$$a_1 = (\sum_{i=1}^{i_{\min}} x_i) - i_{\min} x_{i_{\min}}, \tag{7}$$

$$a_2 = (\sum_{i=i_{\min}+1}^{N} x_i) - (N - i_{\min}) x_{i_{\min}}. \tag{8}$$

Both a_1 and a_2 can be computed after all of the N samples of $\mathbf{x}$ are available. In the case that only the first j samples of $\mathbf{x}$ are available, we define:

$$p(j) = \operatorname*{argmin}_{1 \le i \le j} x_i, \quad q(j) = \operatorname*{argmax}_{1 \le i \le j} x_i. \tag{9}$$

as the current $i_{\min}$ and $i_{\max}$ up to the j-th sample of $\mathbf{x}$, respectively. Based on $p(j)$, the incremental versions of a_1 and a_2, denoted by $a_1(j)$ and $a_2(j)$, are defined as:

$$a_1(j) = S_1(j) - p(j) x_{p(j)}, \tag{10}$$

$$a_2(j) = S_2(j) - (N - p(j)) x_{p(j)}, \tag{11}$$

where:

$$S_1(j) = \sum_{i=1}^{p(j)} x_i, \quad S_2(j) = \sum_{i=p(j)+1}^{j} x_i. \tag{12}$$

Clearly, when $j = N$, it follows from Equations (1) and (9) that $p(N) = i_{\min}$. Therefore, $a_1(N) = a_1$ and $a_2(N) = a_2$. Based on $a_i(j), i = 1, 2$, we then define $f_i(j), i = 1, 2$, as:

$$f_i(j) = a_i(j) / (p(j) - q(j)). \tag{13}$$

The $f_i(j)$ can also be viewed as the incremental version of $f_i, i = 1, 2$. From Equations (6) and (13), it can also be easily shown that $f_i(N) = f_i$ when $x_{i_{\min}}$ is used as the reference.

It is interesting to note that the computation of $S_i(j), i = 1, 2$, can be carried out recursively from their predecessors $S_i(j-1), i = 1, 2$. To explore the recursive relationship, two cases are considered separately. The first case is $p(j) = p(j-1)$. This implies that the current $i_{\min}$ remains the same after the new sample x_j has arrived. In this case,

$$S_1(j) \mid_{p(j)=p(j-1)} = \sum_{i=1}^{p(j)} x_i = \sum_{i=1}^{p(j-1)} x_i = S_1(j-1), \quad (14)$$

$$S_2(j) \mid_{p(j)=p(j-1)} = \sum_{i=p(j)+1}^{j} x_i = (\sum_{i=p(j-1)+1}^{j-1} x_i) + x_j = S_2(j-1) + x_j. \quad (15)$$

In the second case, $p(j) \neq p(j-1)$. This occurs only when $x_j = \min_{1 \leq i \leq j} x_i$. Therefore, in this case, the current $i_{\min}$ is updated as $p(j) = j$. It then follows that:

$$S_1(j) \mid_{p(j)\neq p(j-1)} = \sum_{i=1}^{p(j)} x_i = \sum_{i=1}^{j} x_i = S_1(j-1) + S_2(j-1) + x_j, \quad (16)$$

$$S_2(j) \mid_{p(j)\neq p(j-1)} = 0. \quad (17)$$

All of the operations are based on the initial conditions $S_1(-1) = S_2(-1) = 0$. We can observe from Equations (14)–(17) that $S_1(j)$ and $S_2(j)$ can always be obtained from $S_i(j-1), i = 1, 2$, regardless of the variations of $p(j)$. In addition to providing incremental computation, another important advantage is that it is not necessary to reuse x_j for the computation of $S_i(k), i = 1, 2$, for any $k > j$. This is beneficial for hardware design because it is not necessary to adopt a buffer for storing **x** for data re-use.

2.2. The Architecture

2.2.1. Overview of the Proposed Architecture

Figure 5 shows the proposed architecture for feature extraction. As shown in Figure 5, it can be separated into three parts: the Global Minimum and maximum Search (GMS) unit, the ACCumulation (ACC) unit and the Feature Search (FS) unit. Based on an input sample x_j, the GMS unit computes $p(j)$ and $q(j)$, the current $i_{\min}$ and $i_{\max}$, respectively. The ACC unit then calculates $S_i(j), i = 1, 2$. Based on the results of GMS and ACC units, the FS unit produces the features $f_i(j), i = 1, 2$.

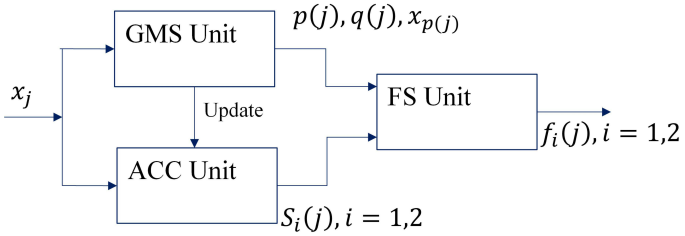

Figure 5. The proposed architecture for feature extraction.

2.2.2. GMS Unit, ACC Unit and FS Unit

The architecture of the GMS unit is depicted in Figure 6. The architecture updates and stores the current $i_{\min}$ and $i_{\max}$, as well as the current $x_{i_{\min}}$. It contained two modules: the GMS 1 module and the GMS 2 module. The goal of the GMS 1 module and GMS 2 module is to find the $p(j)$ and $q(j)$, respectively. Given an input sample x_j, in the GMS 1 module, the comparison between x_j and $x_{p(j-1)}$ is carried out first. When $x_j > x_{p(j-1)}$, it follows from Equation (9) that $p(j) = p(j-1)$. In this case, no updating occurs. Otherwise, we update $p(j) = j$, and $x_{p(j)} = x_j$. The module will also notify the occurrence of updating to the ACC unit. The GMS 2 module operates in a similar fashion for the updating of $q(j)$, which occurs when $x_j > x_{q(j-1)}$.

There are two components in the ACC unit, termed the ACC 1 module and the ACC 2 module, respectively. Their goal is to compute $S_1(j)$ and $S_2(j)$. The operations of the ACC 1 and ACC 2 modules are based on Equations (14)–(17), respectively. As shown in Figure 7, each module contains a register,

an adder and a multiplexer. As the input x_j enters the proposed architecture, the current values held by the register of the ACC 1 and ACC 2 modules are $S_1(j-1)$ and $S_2(j-1)$, respectively. The input sample x_j directly enters the ACC 2 module.

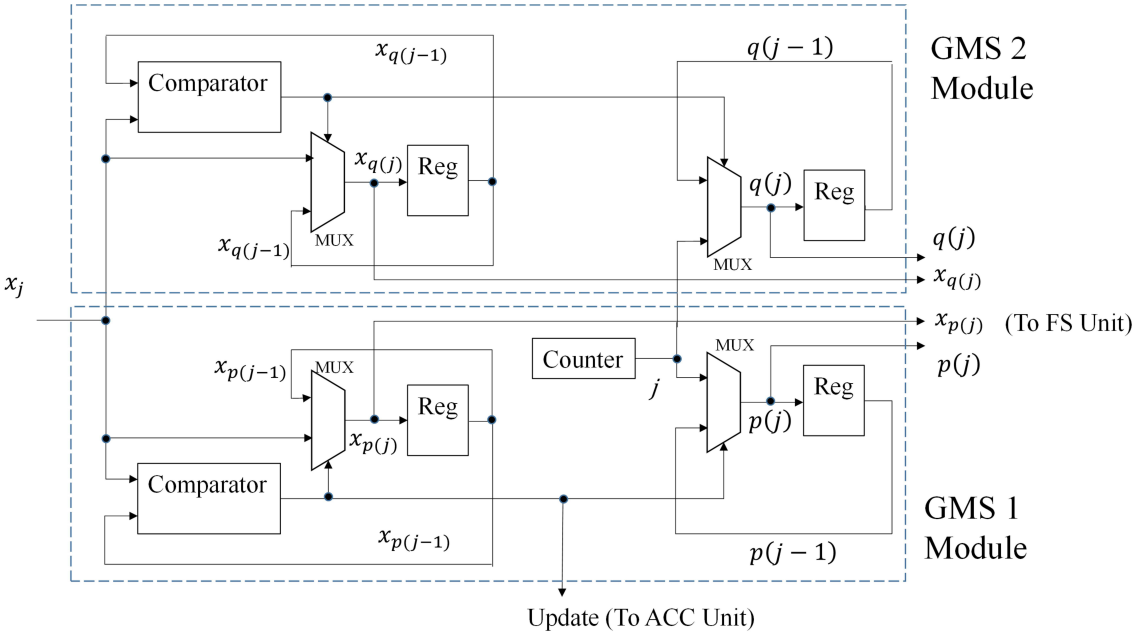

Figure 6. The architecture of the GMS unit.

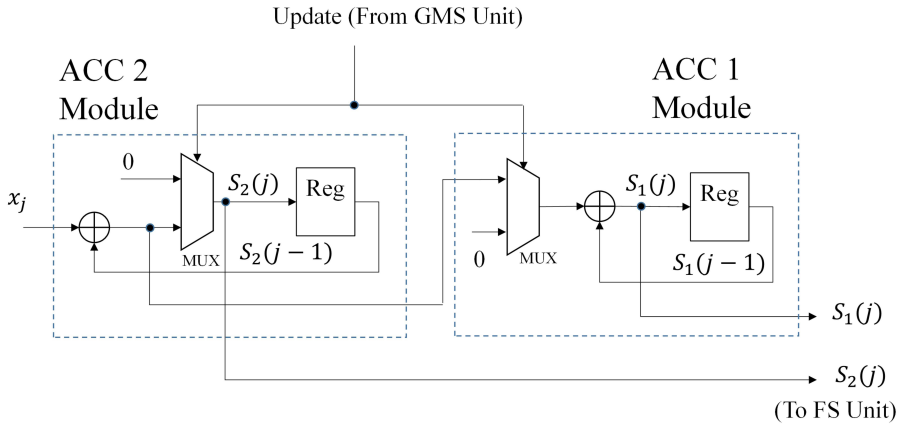

Figure 7. The architecture of the ACC unit.

In the case that no updating operations are required (i.e., $p(j) = p(j-1)$), ACC 2 will add x_j and $S_2(j-1)$ to compute $S_2(j)$ in accordance with Equation (15). In addition, from Equation (14), we see that $S_1(j-1)$ and $S_1(j)$ will be the same for ACC 1 module when no updating occurs. When $p(j) \neq p(j-1)$, ACC 2 will first compute $S_2(j-1) + x_j$ and then push this value to ACC 1. Meanwhile, based on Equation (17), the value of ACC 2 would then be reset to zero. Upon receiving the value $S_2(j-1) + x_j$ from ACC 2, ACC 1 adds the value to $S_1(j-1)$ according to Equation (16).

Figure 8 shows the architecture of the FS unit, which contains two components. The goal of the first component, termed the FS 1 module, is to compute $a_1(j)$ and $a_2(j)$ based on the results produced by the GMS unit and the ACC unit. The computation is carried out in accordance with Equations (10) and (11). Using the results produced by the FS 1 module, the second component of the FS unit, termed the FS 2 module, then computes the features $f_1(j)$ and $f_2(j)$ by Equation (13). As shown in Figure 8,

both the FS 1 module and the FS 2 module mainly contain arithmetic operators for the hardware implementation of Equations (10), (11) and (13).

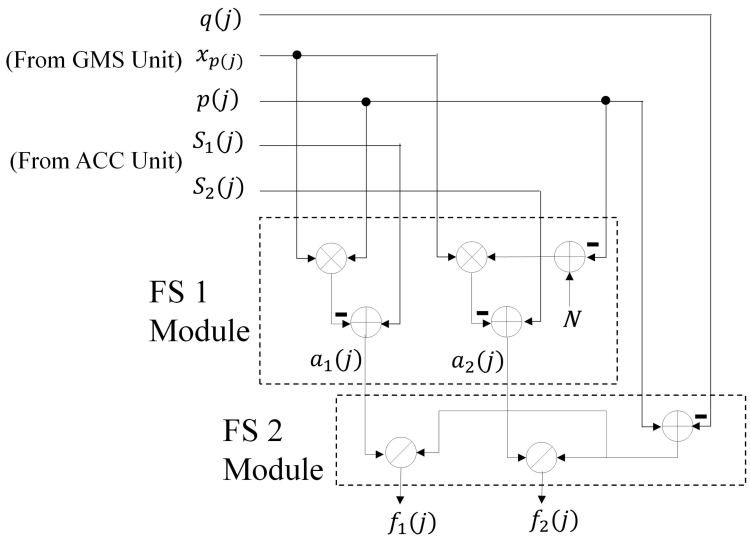

Figure 8. The architecture of the FS unit.

2.2.3. Extension of the Proposed Architecture for Parallel Computation

The proposed architecture in its basic form operates on spikes one sample at a time. When the number of samples N in a spike is large, the latency of the computation may be long. One way to solve the problem is to separate a spike into non-overlapping segments and then operate on the segments concurrently. The computation time can then be reduced.

Let K be the number of segments for the parallel computation, and K is a power of two. For the sake of simplicity, we first consider $K = 2$. The computation for other cases of $K > 2$ can be easily extended from this simple case. Figure 9 shows the proposed architecture for the concurrent operations with $K = 2$. Because $K = 2$, a spike will be separated into two segments. The samples x_j with $1 \leq j \leq (N/2)$ belong to the first segment and the others the second segment. There are two inputs: x_j and $x_{j+N/2}$, which are the j-th sample of the first and the second segments, respectively. There are two GMS units and two ACC units. As shown in the figure, the units denoted by GMS Unit A and ACC Unit A are for the first segment. The others are for the second segment. Their computation results are then combined by the circuits termed the GMS merger unit and the ACC merger unit. Finally, the FS unit computes the final features based on the data provided by the GMS merger unit and the ACC merger unit.

To discuss the operations of the circuits, we first define sets $A(j)$, $B(j)$, and $C(j)$ as:

$$A(j) = \{i : 1 \leq i \leq j\}, \quad B(j) = \{i : (N/2) + 1 \leq i \leq (N/2) + j\}, \quad C(j) = A(j) \cup B(j). \tag{18}$$

In addition,

$$p_A(j) = \underset{i \in A(j)}{\operatorname{argmin}} x_i, \quad p_B(j) = \underset{i \in B(j)}{\operatorname{argmin}} x_i, \quad p_C(j) = \underset{i \in C(j)}{\operatorname{argmin}} x_i. \tag{19}$$

$$q_A(j) = \underset{i \in A(j)}{\operatorname{argmax}} x_i, \quad q_B(j) = \underset{i \in B(j)}{\operatorname{argmax}} x_i, \quad q_C(j) = \underset{i \in C(j)}{\operatorname{argmax}} x_i. \tag{20}$$

Based on Equation (19), we further define:

$$S_{A,1}(j) = \sum_{i\in A(j), i\le p_A(j)} x_i, \quad S_{A,2}(j) = \sum_{i\in A(j), i> p_A(j)} x_i. \tag{21}$$

$$S_{B,1}(j) = \sum_{i\in B(j), i\le p_B(j)} x_i, \quad S_{B,2}(j) = \sum_{i\in B(j), i> p_B(j)} x_i. \tag{22}$$

$$S_{C,1}(j) = \sum_{i\in C(j), i\le p_C(j)} x_i, \quad S_{C,2}(j) = \sum_{i\in C(j), i> p_C(j)} x_i. \tag{23}$$

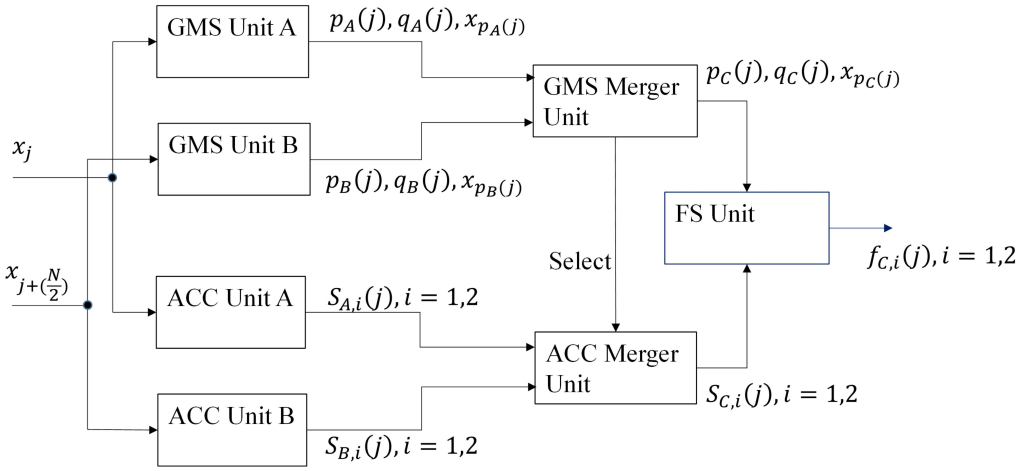

Figure 9. The proposed architecture for the concurrent operations with $K = 2$.

The goal of GMS Units A and B is to compute $p_A(j), q_A(j)$ and $p_B(j), q_B(j)$, respectively. Their architecture is identical to that shown in Figure 6. The ACC Units A and B find $S_{A,1}(j), S_{A,2}(j)$ and $S_{B,1}(j), S_{B,2}(j)$, respectively. These units can be operated by the architecture in Figure 7. Based on $p_A(j), q_A(j)$ and $p_B(j), q_B(j)$, the GMS merger unit produces $p_C(j), q_C(j)$ (as well as $x_{p_C(j)}$). The ACC merger unit then computes $S_{C,1}(j), S_{C,2}(j)$. Figures 10 and 11 shows the architectures of the GMS merger unit and the ACC merger unit, respectively.

It can be easily observed from Equations (19) and (20) that:

$$p_C(j) = \begin{cases} p_A(j) & \text{if } x_{p_A(j)} \le x_{p_B(j)} \\ p_B(j) & \text{if } x_{p_A(j)} > x_{p_B(j)} \end{cases} \tag{24}$$

$$q_C(j) = \begin{cases} q_A(j) & \text{if } x_{q_A(j)} \ge x_{q_B(j)} \\ q_B(j) & \text{if } x_{q_A(j)} < x_{q_B(j)} \end{cases} \tag{25}$$

Therefore, as shown in Figure 10, the GMS merger unit carries out the comparison operations over $x_{p_A(j)}$ and $x_{p_B(j)}$ (or $x_{q_A(j)}$ and $x_{q_B(j)}$) for finding $p_C(j)$ (or $q_C(j)$). Only comparators and multiplexers are required for the operations. We can also derive from Equations (21)–(24) that when $p_C(j) = p_A(j)$,

$$S_{C,1}(j)\ |_{p_C(j)=p_A(j)} = S_{A,1}(j), \tag{26}$$

$$S_{C,2}(j)\ |_{p_C(j)=p_A(j)} = S_{A,2}(j) + S_{B,1}(j) + S_{B,2}(j). \tag{27}$$

On the other hand, when $p_C(j) = p_B(j)$,

$$S_{C,1}(j)\ |_{p_C(j)=p_B(j)} = S_{A,1}(j) + S_{A,2}(j) + S_{B,1}(j), \tag{28}$$

$$S_{C,2}(j)\ |_{p_C(j)=p_B(j)} = S_{B,2}(j). \tag{29}$$

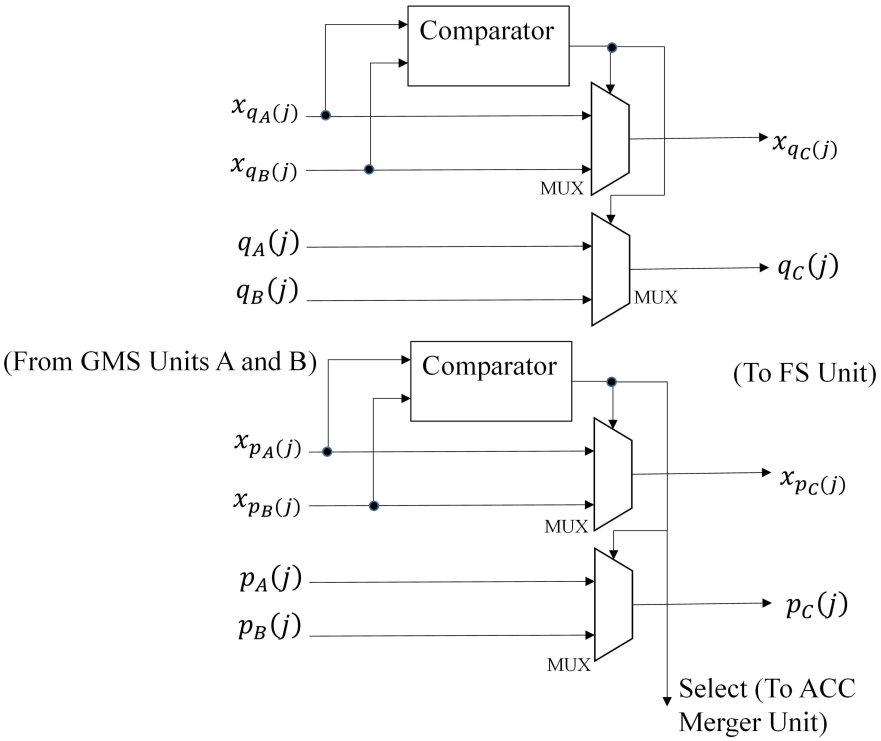

Figure 10. The architecture of the GMS merger unit.

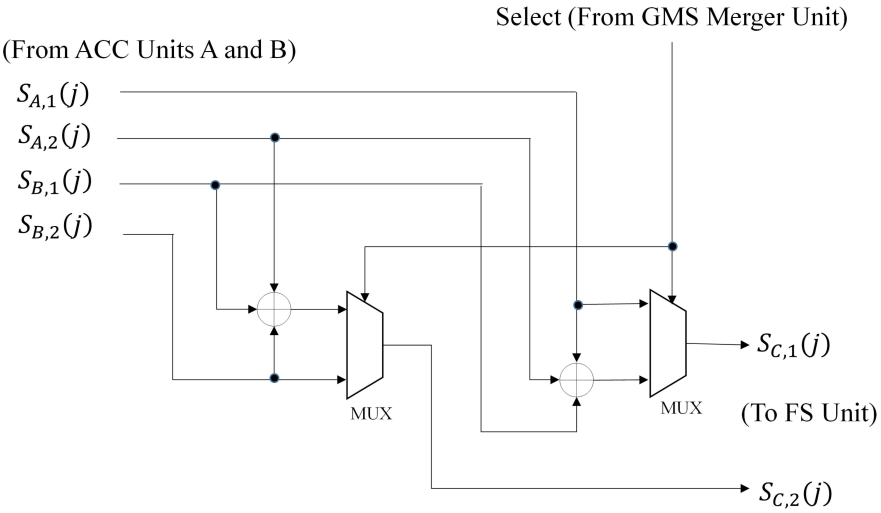

Figure 11. The architecture of the ACC merger unit.

The operations of the ACC merger unit then follow Equations (26)–(29). These operations can be carried out by only adders and multiplexers, as shown in Figure 11.

Based on the data produced by the GMS merger unit and ACC merger unit, the FC unit then computes $f_{C,1}(j)$ and $f_{C,2}(j)$, defined as:

$$f_{C,i}(j) = a_{C,i}(j)/(p_C(j) - q_C(j)), \tag{30}$$

where:

$$a_{C,1}(j) = S_{C,1}(j) - p_C(j)x_{p_C(j)}, \quad (31)$$
$$a_{C,2}(j) = S_{C,2}(j) - (N - p_C(j))x_{p_C(j)}, \quad (32)$$

The architecture of the FS unit for parallel computation is also identical to that of the basic circuit shown in Figure 8. It can be observed that, when $j = N/2$, $C(N/2) = \{i : 1 \leq i \leq N\}$. Therefore, $a_{C,i}(N/2) = a_i, i = 1, 2$. As a result, $f_{C,i}(N/2) = f_i, i = 1, 2$.

The proposed architectures can also be viewed as trees. The basic architecture shown in Figure 5 is a unitary tree with a leaf node and a root node. The parallel computation circuit in Figure 9 for $K = 2$ can be viewed as a simple binary tree with two leaf nodes, one intermediate node and one root node. Each leaf node contains a GMS unit and an ACC unit. Each intermediate node consists of a GMS merger unit and an ACC merger unit. The root node comprises the FS unit.

The parallel computation for a larger number of K, where K is a power of two, is a simple extension of $K = 2$. In this case, the circuit can be viewed as a binary tree with K leaf nodes and $K - 1$ intermediate nodes and a root node. There are $2 + \log_2 K$ layers for the parallel computation. The leaf nodes form the first layer. The intermediate nodes can be organized into the subsequent $\log_2 K$ layers. The root node is the final layer of the circuit. Figure 12 shows the examples of the binary trees for $K = 2$ and $K = 4$.

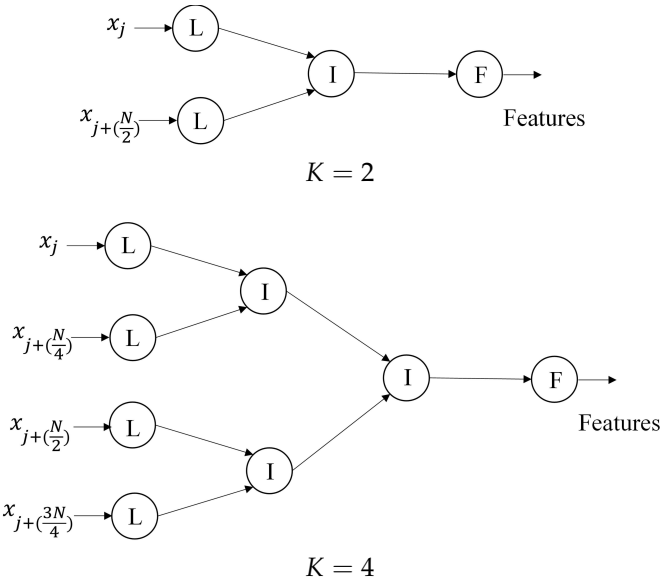

Figure 12. The binary tree representation of the proposed architecture for parallel computation with $K = 2$ and 4. The nodes denoted by L, I and F are the leaf nodes, intermediate nodes and root nodes, respectively.

The circuit with K a power of two operates by first separating a spike into K non-overlapping segments with length N/K. Each segment is assigned to a dedicated leaf node. Samples of each segment are delivered to the assigned leaf node one sample at a time. The data produced by the leaf nodes are then forwarded to the first layer of the intermediate nodes. Each layer would then receive data from its preceding layer and deliver the results to the next layer. The same process is repeated until the final layer is reached.

3. Proposed Architecture for Multi-Channel Feature Extraction

The proposed architecture can be easily operated in conjunction with the multi-channel spike detection circuit for multi-channel feature extraction. Figure 13 shows the architecture of a multi-channel spike sorting system based on the proposed feature extraction circuit. As depicted in

the figure, there are three components: spike detection circuit, spike buffer and the proposed feature extraction circuit.

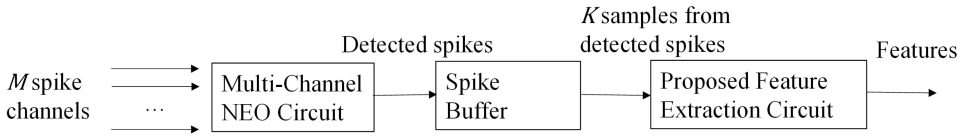

Figure 13. The architecture of a multi-channel spike sorting system based on the proposed feature extraction circuit.

The spike detection circuit is able to carry out the detection for multiple channels by the NEO algorithm. Let M be the number of channels for spike sorting. Assume all of the channels are sampled with the same sampling rate r_s. Let $T_s = 1/r_s$ be the sampling period. All of the channels are assumed to be sampled and multiplexed by a mixed mode circuit using the round robin approach. The mixed-mode circuit then delivers the samples one at a time to the spike detection circuit. Therefore, the circuit receives M samples during a time interval of length T_s. Different samples received during the interval are from different channels.

The spike detection circuit can be separated into two portions: the NEO buffer and the NEO detection unit. The NEO buffer is a $(M \times N)$-stage Serial-In-Serial-Out (SISO) shift register organized in a snake-like fashion. Each stage contains a sample of a spike train from a channel. It is therefore able to hold N consecutive samples of the spike trains from the M channels.

The bottom row of the buffer provides N consecutive samples of the spike train from a channel (say, channel h). It can be seen from Figure 14 that the bottom row of the NEO buffer is used for both NEO detection and peak alignment. The NEO detection unit takes three consecutive samples of the bottom row to carry out the NEO computation. The computation result is then compared to a given threshold γ. If the result is larger than the threshold, a hit event is issued. In addition, the entire last row is regarded as a detected spike and is copied to the spike buffer for spike alignment.

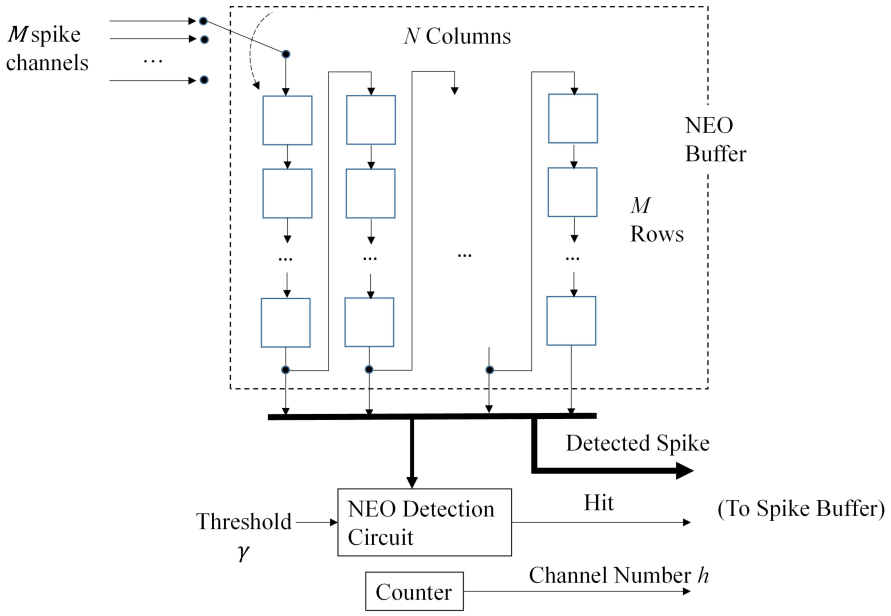

Figure 14. The architecture of the multi-channel NEO circuit.

The spike detection circuit is able to perform spike detection one channel at a time. After the spike detection of the channel h is completed in the current clock cycle, all of the spike samples already

in the NEO buffer are shifted to the next stage, and a new sample from the next channel (selected in a round robin fashion) enters the first stage of the NEO buffer. This allows the spike detection for the next channel to be carried out in the next clock cycle.

The goal of the spike buffer is to hold the detected spikes produced by the spike detection circuit, carry out the alignment and deliver the detected spikes to the proposed feature extraction circuit. As depicted in Figure 15, there are two stages in the spike buffer: the input stage and the output stage. The input stage is an N-input N-output buffer holding the detected spikes provided by the spike detection circuit. The output stage is an N-input K-output buffer fetching data N samples at a time from the input buffer. The buffer then delivers data K samples at a time to the proposed feature extraction circuit.

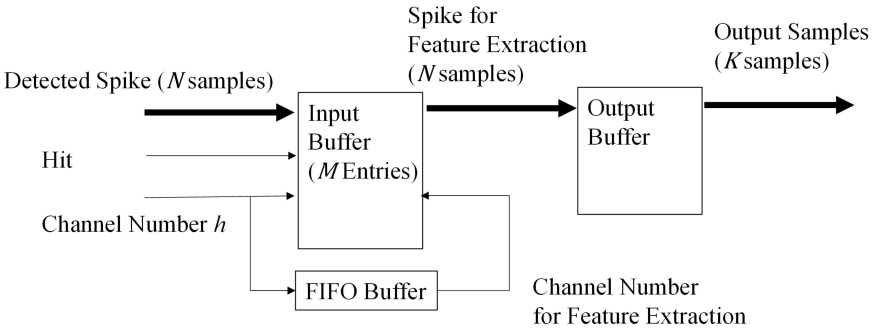

Figure 15. The architecture of the spike buffer.

The input stage contains M entries, where each entry i stores the detected spike for channel i. Given N, K, the maximum number of channels M can be evaluated. For any channel h in the circuit, a detected spike in that channel could be discarded when the spike is over-written in the spike buffer by the next detected spike from the same channel h before it can be further processed.

Recall from Section 2.2.3 that the proposed architecture supporting parallel computation is able to accept K samples at a time. Therefore, the proposed architecture is able to process a detected spike with N samples in N/K clock cycles. To find the maximum number of channels M, the worst case scenario is considered. In the scenario, the detected spikes from all of the M channels are stored in the spike buffer and are not processed by the feature extraction circuit yet. In addition, the feature extraction circuit is currently busy. Assume that the newest detected spike is from channel h. In this case, the spike buffer is able to accept another detected spike from channel h without discarding the old one only after the feature extraction circuit has processed M spikes. Because the latency for processing each detected spike is N/K clock cycles, it follows that the input buffer is able to accept another detected spike from channel h after MN/K clock cycles. Let Q be the minimum number of samples between the peak of successive spikes detected by the NEO circuit from the same channel. Assume that Q is the same for all of the channels. It then follows that a detected spike from channel h is not overwritten and discarded when:

$$\frac{MNT_c}{K} \leq QT_s, \tag{33}$$

where $T_c = 1/r_c$ is the clock period and r_c is the clock rate. It is interesting to know that the NEO circuit imposes an additional limit on the number of channels M. It is desired that the NEO circuit be able to receive one sample from each channel in a single sampling period T_s. Based on the round robin scheme for fetching samples for different channels, it is therefore necessary that:

$$MT_c \leq T_s. \tag{34}$$

Combining Equations (33) and (34), we see that the maximum number of channels, denoted by $M_{\max}$, should then satisfy:

$$M_{\max} \leq \min\{\frac{QT_sK}{NT_c}, \frac{T_s}{T_c}\}. \quad (35)$$

4. Experimental Results

This section presents the performance of the proposed architecture. The area complexities are first considered. Five types of area costs are evaluated: the number of comparators, adders/subtractors, multipliers/dividers, registers and multiplexers. All of the costs are expressed in terms of the asymptotic function (i.e., the big O function). Table 1 shows the area complexities of the proposed feature extraction circuit. It can be observed from Table 1 that all of the area costs of the GMS units, ACC units, GMS merger units, ACC merger units and FS units are independent of the spike length N and the number of channels M. However, for the parallel computation, the number of comparators, adders/subtractors, registers and multiplexers of all of the leaf nodes will grow with the number of segments K. This is because the total number of leaf nodes is dependent on K, and each leaf node contains a GMS unit and an ACC unit.

Table 1. The area complexities of the proposed feature extraction circuit.

	Comparators	Adders/Subtractors	Multipliers/Dividers	Registers	Multiplexers
GMS Unit	$O(1)$	0	0	$O(1)$	$O(1)$
ACC Unit	0	$O(1)$	0	$O(1)$	$O(1)$
GMS Merger Unit	$O(1)$	0	0	0	$O(1)$
ACC Merger Unit	0	$O(1)$	0	0	$O(1)$
FS Unit	0	$O(1)$	$O(1)$	0	0
Total Leave Nodes	$O(K)$	$O(K)$	0	$O(K)$	$O(K)$
Total Intermediate Nodes	$O(K)$	$O(K)$	0	0	$O(K)$
Total Root Nodes	0	$O(1)$	$O(1)$	0	0
Total	$O(K)$	$O(K)$	$O(1)$	$O(K)$	$O(K)$

Similarly, because the total number of intermediate nodes is dependent on K and each intermediate node contains a GMS merger unit and an ACC merger unit, the area complexities of comparators, adders/subtractors and multiplexers of all of the intermediate nodes are $O(K)$. On the contrary, there is only one root node for parallel computation, and the root node consists of only the FS unit. Among the units in the proposed architecture, the FS unit is the only unit containing multipliers/dividers. Therefore, the number of multipliers/dividers is fixed, and is independent of N, M and K. Summarizing the discussions stated above, we conclude that the complexities of the total number of comparators, adders/subtractors, registers and multiplexers of the proposed feature extraction circuit with K segments are $O(K)$. Moreover, the total number of multipliers/dividers is only $O(1)$.

Based on Table 1, the overall area complexities of the proposed spike sorting circuit are summarized in Table 2. To facilitate the evaluation, the area complexities of the NEO circuit and spike buffer are also included in the table. For the NEO circuit, it is necessary to store spike trains from all of the M channels for detection. In the spike buffer, each channel needs to have its own memory unit to hold the detected spikes for the subsequent feature extraction. Therefore, it can be observed from Table 2 that the number of registers in both the NEO circuit and spike buffer are dependent on the spike length N and the number of channels M. The total number of registers of the proposed spike sorting circuit is then $O(K + MN)$. In addition, because the NEO circuit has only a fixed number of adders and multipliers independent of N and M, the overall area complexities for the adders and multipliers in the spike sorting circuit are only $O(K)$ and $O(1)$, respectively.

Table 2. The area complexities of the proposed spike sorting circuit.

	Comparators	Adders/Subtractors	Multipliers/Dividers	Registers	Multiplexers
NEO Circuit	$O(1)$	$O(1)$	$O(1)$	$O(MN)$	0
Spike Buffer	0	0	0	$O(MN)$	$O(1)$
Feature Extraction	$O(K)$	$O(K)$	$O(1)$	$O(K)$	$O(K)$
Total	$O(K)$	$O(K)$	$O(1)$	$O(MN+K)$	$O(K)$

We next evaluate the actual hardware resource consumption of the proposed circuit. For the remaining evaluations of this section, the dimension of spikes is $N = 64$. The circuit implementation is carried out by ASIC with the Taiwan Semiconductor Manufacturing Company (TSMC) 90-nm technology and clock gating. The gate level design platform is the Synopsys Design Compiler. Table 3 shows the area (μm^2) of the proposed circuit for different numbers of channels M and numbers of segments K. From Table 3, we see that the area of the proposed circuit grows with M and K, which is consistent with the results shown in Table 2. Table 4 shows the normalized area (μm^2 per channel) of the proposed architecture. The normalized area is defined as the area of the circuit divided by the number of channels M. The normalized area can be viewed as the average area cost per channel. Note that all of the channels share the same computation cores in the NEO circuit (i.e., the NEO detection unit) and the proposed feature extraction circuit. Therefore, we can see from Table 4 that the average area per channel decreases as M increases. Consequently, because of the hardware resource sharing scheme, the proposed architecture shows a higher efficiency in area costs for a larger number of channels. Although the normalized area grows with K for a fixed M, the circuits with larger K have the advantages of lower latency for feature extraction. Therefore, as shown in Equation (35), the channel capacity $M_{\max}$ grows with K.

Table 3. The area (μm^2) of the proposed circuit for different numbers of channels M and segments K.

Number of Segments K	Number of Channels M					
	2	**4**	**8**	**16**	**32**	**64**
1	58,008	98,990	180,665	346,647	676,085	1,337,493
2	65,625	106,607	188,282	354,264	683,702	1,345,110
4	80,859	121,841	203,516	369,498	698,936	1,360,344

Table 4. The normalized area (μm^2/channel) of the proposed circuit for different numbers of channels M and segments K.

Number of Segments K	Number of Channels M					
	2	**4**	**8**	**16**	**32**	**64**
1	29,004	24,747	22,583	21,665	21,127	20,898
2	32,813	26,652	23,535	22,142	21,366	21,017
4	40,430	30,460	25,440	23,094	21,842	21,255

In addition to the area costs, the power consumption of the proposed architecture is also evaluated. Table 5 shows the normalized power dissipation (μW per channel) for different numbers of channels M with and without clock gating. The normalized power dissipation is defined as the total power consumption of the circuit divided by the number of channels M. In the experiment, the number of segment $K = 1$, and the clock rate r_c is 1 MHz. The power consumption is measured numerically by Synopsis Prime Time. When the number of channels M increases, because the normalized area decreases, we can see from Table 5 that the normalized power consumption also lowers for the circuit without clock gating. In addition, the clock gating is able to further reduce the power consumption by not supplying the clock signal to the inactive components. However, when M increases, the components of the proposed feature extraction circuit may become more busy because all

of the M channels share the circuit. As a result, when M increases from 32–64, it can be observed from Table 5 that the power consumption with clock gating is not lowered. Nevertheless, the the circuit with clock gating is still able to achieve a 33% power reduction as compared with its counterpart without clock gating.

We also compare the proposed architecture with other ASIC implementations [8–11] for spike sorting in Table 6. It may be difficult to directly compare these architectures because they may be implemented by different technologies and may be based on different spike detection and/or feature extraction algorithms. Moreover, their operation clock rates may be different. Nevertheless, it can be observed from Table 6 that, as compared with the existing architectures, the proposed architecture provides effective area-power performance while supporting both spike detection and feature extraction functions.

Note that, among these existing architectures, the proposed architecture and the one in [11] are based on the same clock rate, technology and spike detection scheme. We can see from Table 6 that the normalized area and normalized power of the proposed architecture are only 25.40% (i.e., 21,127 vs. 83,159) and 25.44% (i.e., 20.03 vs. 78.719) of those of the architecture in [11], respectively. This is because the proposed feature extraction algorithm has significantly lower computational complexities than those of the GHA algorithm adopted by [11].

Table 5. The normalized power consumption (μW/channel) of the proposed circuit with and without clock gating for different numbers of channels M.

	Number of Channels M					
	2	**4**	**8**	**16**	**32**	**64**
No Clock Gating	43.08	36.55	33.00	31.90	31.10	30.69
Clock Gating	32.98	26.58	23.13	21.69	20.03	20.53
Power Reduction	23%	27%	29%	32%	36%	33%

Table 6. The comparisons of various spike sorting architectures.

Architecture	**Normalized Area**	**Normalized Power**	**Clock Rate**	**# of Channels**	**Spike Dimension**	**Technology**	**Detection**	**Feature Extraction**
[8]	255,495 μm^2/ch.	521 μW/ch.	1 MHz	1	64	90 nm	No	PCA
[9]	1,770,000 μm^2/ch.	256.875 μW/ch.	N/A	16	N/A	350 nm	NEO	PCA
[10]	268,000 μm^2/ch.	8.589 μW/ch.	281.25 KHz	1	64	130 nm	No	SPIRIT
[11]	83,159 μm^2/ch.	78.719 μW/ch.	1 MHz	32	64	90 nm	NEO	GHA
Proposed	21,127 μm^2/ch.	20.03 μW/ch.	1 MHz	32	64	90 nm	NEO	Proposed

Although the proposed feature extraction algorithm has simple computation, it is effective for spike classification. Tables 7–10 show the Classification Success Rate (CSR) of the Fuzzy C-Means (FCM) [19] algorithm by clustering the feature vectors produced by various feature extraction algorithms. The CSR is defined as the number of spikes that are correctly classified divided by the total number of spikes. The PCA, GHA and zero-crossing [15] algorithms considered in the tables are implemented by MATLAB with double precision floating numbers. The proposed algorithm is implemented by hardware with 10-bit finite precision.

To see the robustness of the proposed algorithm, different types of noise interference are included in the experiments: background noises, interfering neurons and overlapping spikes. They are considered separately to facilitate our observation. The spike trains for the experiments in Tables 7–9 are obtained from the simulator developed in [18]. It also gives access to the ground truth about the spiking activity in the recording for quantitative assessment.

Table 7 shows the CSRs of various feature extraction algorithms for background noises with different SNR levels. In the experiments, the number of interfering neurons is set to be zero. In addition, 20% of the spikes are overlapping. It can be observed from Table 7 that the CSRs of the proposed algorithm are only slightly degraded as the SNR values decrease. In addition, the proposed algorithm

has CSR performance comparable to that of PCA and GHA for all of the SNR levels. Moreover, it outperforms the zero-crossing algorithm. When the SNR = 1 dB, the proposed algorithm is still able to achieve 96.47% CSR. As shown in Figure 1, the proposed algorithm is robust to background noise because the variations of peak locations due to background noise corruption are small. This leads to successful identification of intervals J_1 and J_2 for the computation of features a_1 and a_2.

Table 7. The CSRs of the various feature extraction algorithms for different SNR levels. The spike source data are obtained by the simulator in [18].

Algorithms	SNR (dB)			
	1	4	6	8
PCA [8]	99.57%	99.70%	99.64%	99.82%
GHA [11]	99.30%	99.58%	99.82%	99.82%
Zero-Crossing [15]	96.32%	96.71%	95.44%	95.57%
Proposed	96.47%	97.22%	97.13%	97.52%

The CSRs of various algorithms for different numbers of interfering neurons are revealed in Table 8. The interfering neurons are the nearby neurons whose spike times are correlated with the original spike times of the target neurons. We set SNR = 4 dB for the background noises. The percentage of the overlapping spikes for the experiments is 20%. From Table 8, we can see that only a small degradation is observed for the proposed algorithm as the number of interfering neurons grows. Its performance is also comparable to that of the PCA and GHA algorithms.

Table 8. The CSRs of the various feature extraction algorithms for different number of interfering neurons. The spike source data are obtained by the simulator in [18].

Algorithms	Number of Interfering Neurons			
	0	2	4	6
PCA [8]	99.70%	99.68%	99.76%	99.58%
GHA [11]	99.58%	99.66%	99.58%	99.51%
Proposed	97.22%	96.09%	95.56%	95.44%

The influences of the overlapping spikes on CSRs are considered in Table 9. In the experiments, the SNR level of background noises is 8 dB. In addition, there are no interfering neurons, so that the the impact of overlapping spikes can be easily observed. It appears from Table 9 that the proposed algorithm is able to maintain CSRs above 95% even when the percentage of the overlapping spikes is 30%. Because the proposed algorithm may still be able to find the J_1 or J_2 correctly for the overlapping spikes as revealed in Figures 3 and 4, successful feature extraction and classification may still be possible. As a result, the proposed algorithm may be able to maintain high CSRs in the presence of overlapping spikes.

Table 9. The CSRs of the various feature extraction algorithms for different percentages of overlapping spikes. The spike source data are obtained by the simulator in [18].

Algorithms	Percentage of Overlapping Spikes			
	15%	20%	30%	40%
PCA [8]	99.80%	99.82%	99.75%	99.35%
GHA [11]	99.76%	99.82%	99.75%	99.36%
Proposed	98.06%	97.52%	96.25%	94.84%

In addition to the simulator developed in [18], the spike trains in the wave_clus database are also considered in the experiments. Table 10 shows the resulting comparisons. Similar to the

results shown in Tables 7–9, we can also see from Table 10 that the CSR performances of the PCA, GHA and the proposed algorithm are comparable. In addition, the proposed algorithm outperforms the zero-crossing algorithm. Although PCA and GHA offer higher classification accuracy, the algorithms have higher computational complexities. We can see from Table 6 that their hardware implementations may have large area costs and power consumption. The computational complexities of zero-crossing are low. However, the algorithm has inferior CSR values. In particular, for the spike train data "C_Difficult2_noise005" in Table 10, the CSR of the zero-crossing is only 69.92%. By contrast, the proposed algorithm still maintain high accuracy (i.e., 82.13%) for the data. Therefore, the proposed algorithms have the advantages of low computational complexities for efficient hardware implementation and are capable of providing feature vectors for spike classification with high accuracy.

Table 10. The CSRs of the FCM algorithm by clustering the feature vectors produced by various feature extraction algorithms. The spike source data are obtained from the wave_clus database [20].

Algorithms	File Names			
	C_Easy1_Noise01	**C_Easy2_Noise01**	**C_Easy2_Noise005**	**C_Difficult2_Noise005**
PCA [8]	99.32%	96.68%	98.45%	98.57%
GHA [11]	99.32%	94.35%	98.15%	81.66%
Zero-Crossing [15]	89.61%	79.86%	92.26%	69.92%
Proposed	93.00%	90.57%	93.90%	82.13%

5. Conclusions

We have implemented the proposed architecture for spike sorting by ASIC with 90-nm technology. Experimental results reveal that the proposed architecture has the advantages of low area costs, low power consumption and high classification accuracy. For the 32-channel design example provided in the paper, the normalized area is 21,127 μm^2/channel, which is the lowest as compared with the existing designs considered in the paper. When operating at 1 MHz, the proposed architecture consumes normalized power of 20.03 μW/channel. The CSR values of the FCM based on the feature vectors provided by the proposed algorithm are also comparable to those of the PCA and GHA techniques. The proposed architecture therefore is an effective alternative to the applications where implantable spike sorting circuits with low power consumption, low area costs and high CSR are desired.

Acknowledgments: The authors would like to acknowledge the financial support of the Ministry of Science and Technology, Taiwan, under Grant MOST 105-2221-E-003-011-MY2.

Author Contributions: Yuan-Jyun Chang designed and implemented the spike sorting circuit and performed the experiments. Wen-Jyi Hwang conceived of and designed the spike sorting circuit and wrote the paper. Chih-Chang Chen performed the experiments.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ASIC	Application-Specific Integrated Circuit
CSR	Classification Success Rate
FCM	Fuzzy C-Means
FPGA	Field Programmable Gate Array
GHA	Generalized Hebbian Algorithm
GMS	Global Minimum and maximum Search
NEO	Non-linear Energy Operator
PCA	Principal Component Analysis
FS	Feature Search
ACC	ACCumulation

References

1. Einevoll1, G.T.; Franke, F.; Hagen, E.; Pouzat, C.; Harris, K.D. Towards reliable spike-train recordings from thousands of neurons with multielectrodes. *Curr. Opin. Neurobiol.* **2012**, *22*, 11–17.
2. Gibson, S.; Judy, J.W.; Markovic, D. Spike sorting: The first step in decoding the brain. *IEEE Signal Process. Mag.* **2012**, *29*, 124–143.
3. Reya, H.G.; Pedreira, C.; Quiroga, R.Q. Past, present and future of spike sorting techniques. *Brain Res. Bull.* **2015**, *119*, 106–117.
4. Meyer-Baese, U. *Digital Signal Processing with Field Programmable Gate Arrays*, 4th ed.; Springer: Berlin, Germany, 2014.
5. Goldshan, K. *Physical Design Essentials: An ASIC Design Implementation Perspective*; Springer: New York, NY, USA, 2007.
6. Yu, B.; Mak, T.; Li, X.; Xia, F.; Yakovlev, A.; Sun, Y.; Poon, C.S. A Reconfigurable Hebbian Eigenfilter for Neurophysiological Spike Train Analysis. In Proceedings of the IEEE International Conference on Field Programmable Logic and Applications, Milano, Italy, 31 August–2 September 2010; pp. 556–561.
7. Gibson, S.; Judy, J.W.; Markovic, D. An FPGA-based platform for accelerated offline spike sorting. *J. Neurosci. Methods* **2013**, *215*, 1–11.
8. Chen, T.-C.; Liu, W.; Chen, L.-G. VLSI Architecture of Leading Eigenvector Generation for On-Chip Principal Component Analysis Spike Sorting System. In Proceedings of the 30th Annual International Engineering in Medicine and Biology Society, Vancouver, BC, Canada, 20–25 August 2008; pp. 3192–3195.
9. Chen, T.-C.; Chen, K.; Yang, Z.; Cockerham, K.; Liu, W. A Biomedical Multiprocessor SOC for Closed Loop Neuroprosthetic Applications. In Proceedings of the IEEE International Solid-State Circuits Conference, San Francisco, CA, USA, 8–12 Febraury 2009; pp. 433–435.
10. Wu, T.; Yang, Z. Power-efficient VLSI implementation of a feature extraction engine for spike sorting in neural recording and signal processing. In Proceedings of the IEEE International Conference on Control Automation Robotics and Vision, Singapore, 10–12 December 2014; pp. 7–12.
11. Chen, Y.L.; Hwang, W.J.; Ke, C.E. An efficient VLSI architecture for multi-channel spike sorting using a generalized Hebbian algorithm. *Sensors* **2015**, *15*, 19830–19851.
12. Paraskevopouloua, S.E.; Barsakcioglua, D.Y.; Saberi, M.R.; Eftekhar, A.; Constandinou, T.G. Feature extraction using first and second derivative extrema (FSDE) for real-time and hardware-efficient spike sorting. *J. Neurosci. Methods* **2013**, *215*, 29–37.
13. Zamani, M.; Demosthenous, A. Feature extraction using extrema sampling of discrete derivatives for spike sorting in implantable upper-limb neural prostheses. *IEEE Trans. Neural Syst. Rehabil. Eng.* **2014**, *22*, 716–726.
14. Zviagintsev, A.; Perelman, Y.; Ginosar, R. Low-power architectures for spike sorting. In Proceedings of the 2nd International IEEE EMBS Conference on Neural Engineering, Arlington, VA, USA, 16–19 March 2005; pp. 162–165.
15. Kamboh, A.M.; Mason, A.J. Computationally efficient neural feature extraction for spike sorting in implantable high-density recording systems. *IEEE Trans. Neural Syst. Rehabil. Eng.* **2013**, *21*, 1–9.
16. Mukhopadhyay, S.; Ray, G.C. A new interpretation of nonlinear energy operator and its efficacy in spike detection. *IEEE Trans. Biomed. Eng.* **1998**, *45*, 180–187.
17. Kaeslin, H. *Digital Integrated Circuit Design*; Cambridge University Press: Cambridge, UK, 2008.
18. Smith, L.S.; Mtetwa, N. A tool for synthesizing spike trains with realistic interference. *J. Neurosci. Methods* **2007**, *159*, 170–180.
19. Oliynyk, A.; Bonifazzi1, C.; Montani, F.; Fadiga1, L. Automatic online spike sorting with singular value decomposition and fuzzy C-mean clustering. *BMC Neural Sci.* **2012**, *13*, 96.
20. Quiroga, R.Q.; Nadasdy, Z.; Ben-Shaul, Y. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural Comput.* **2004**, *16*, 1661–1687.