Chapter 1

Efficient Hardware Architecture for Correlation-Based Spike Detection and Unsupervised Clustering

Chien-Min Ou and Wen-Jyi Hwang

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/66105

Abstract

This chapter presents a novel hardware architecture for correlation-based spike detection and unsupervised clustering. The architecture is able to utilize the information extracted from the results of spike clustering for efficient spike detection. The architecture supports the fast computation for the normalized correlation and OSORT operations. The normalized correlation is used for template matching for accurate spike detection. The OSORT algorithm is adopted for unsupervised classification of the detected spikes. The mean of spikes of each cluster produced by the OSORT algorithm is used as the templates for subsequent detection. The architecture adopts postnormalization technique for reducing the area costs. Modified OSORT operations are also proposed for facilitating unsupervised clustering by hardware. The proposed architecture is implemented by field programmable gate array (FPGA) for performance evaluation. In addition to attaining high detection and classification accuracy for spike sorting, experimental results reveal that the proposed architecture is an efficient design providing low area cost and high throughput for real-time offline spike sorting applications.

Keywords: spike sorting, spike detection, spike clustering, field programmable gate array, 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 [2] for analyzing the activities of neurons. A typical spike sorting system usually involves complicated spike detection and classification operations for separating spikes from background noise and clustering the detected spikes. Large amount of spike-trains would impose heavy computation load for a software spike sorting system, resulting in long processing time.

One approach to reduce the computation time is to implement a spike sorting system by hardware. A number of spike sorting systems based on field programmable gate array (FPGA) [3] have been proposed. In [4], a spike classification architecture using probabilistic neural networks [5] is proposed. Although high speed-up over its software counterpart is observed, the system does not provide spike detection. In addition, the architecture requires the user to input the number of clusters. Therefore, it does not support a fully unsupervised hardware classification. Similarly, the architecture proposed in [6] also focuses on the classification. The generalized Hebbian algorithm (GHA) [7] is implemented by FPGA in this work for feature extraction. The features produced by the architecture are then clustered by the k-means algorithm. The architecture does not include the hardware implementation of the k-means algorithm. In addition, the number of clusters still needs to be prespecified in the k-means algorithm. The architecture in [8] is able to carry out the feature extraction and clustering in hardware. In the architecture, the feature extraction and clustering are based on GHA and fuzzy c-means [9] algorithm, respectively. Therefore, similar to the architecture in [6], fully unsupervised classification is difficult for the architecture in [8] because the number of clusters still needs to be known beforehand.

An alternative FPGA-based hardware architecture [10] is to adopt OSORT algorithm [11] for spike-sorting. The OSORT algorithm is able to perform clustering without the prior knowledge of the number of clusters. Unsupervised classification therefore can be carried out. The architecture also includes a spike detection circuit based on the nonlinear energy operators (NEOs) [12], which is an energy-based detection algorithm. Similar to other energy-based algorithms [13, 14], the NEO algorithm is simple and effective. However, although the energy-based algorithms can operate in conjunction with a number of automatic threshold algorithms [12–14], proper selection of threshold values for these algorithms may still be difficult when noise becomes large. Therefore, their performance may deteriorate rapidly as noise energy increases.

The template-based spike detection algorithm may be suited for the detection of spikes from the source sequences with high noise level. A matched filter [15, 16] is a typical technique based on templates. To detect the presence of the templates, the filter correlates known templates with the input spike trains. This operation can also be viewed as a likelihood ratio detection (LRT) [17]. A drawback of the matched filter is that the templates are required. Although the adaptive generation of templates is possible [18], only a single template is produced for spike trains. However, because spike trains in general are formed from two or more neurons, a single template may not be sufficient for the detection of all the spikes generated by the neurons. The template-based algorithm presented by [19] has been found to be effective for the detection of noisy spikes. It adopts the OSORT algorithm for automatic template generation. Normalized correlation operations then carry out the spike detection using the templates produced by the OSORT algorithm. Nevertheless, in the algorithm, both the template generation and correlation computation have high computational complexities.

The software implementation of the algorithm may not be suitable for fast analysis of spike trains.

The objective of this chapter is to present a novel FPGA-based hardware architecture for efficient spike detection and clustering. The architecture supports both the normalized correlation operations for spike detection and the OSORT operations for spike clustering. Moreover, the clustering results produced by the OSORT algorithm are used as the templates for spike detection. The architecture is the hardware implementation of the algorithm in [19]. It then has the advantages of accurate spike detection, fully unsupervised spike clustering, and fast computation.

We have implemented a spike sorting system on a network-on-chip (NOC) platform for the evaluation of the proposed architecture. The platform is based on FPGA. It consists of a soft-core processor [20] and the proposed architecture. The proposed spike sorting architecture is used as a hardware accelerator of the soft-core processor for spike sorting. The simulator developed in [21] is adopted to generate extracellular recordings. In this paper, comparisons with the existing software and hardware implementations are made. Experimental results show that the proposed architecture attains a high speed-up over its software counterpart for spike sorting. It also has a lower area cost over existing hardware architectures. Experimental results reveal that the proposed architecture is an effective alternative for real-time spike sorting with accurate detection and clustering.

2. The algorithm

Before presenting the hardware architectures for spike sorting, we first review the spike detection and clustering algorithms adopted by this work. Detailed discussions of these algorithms can be found in [19].

2.1. Normalized correlation

Consider a spike train $\mathbf{X}$, where the mth sample of $\mathbf{X}$ is denoted by $x[m]$. Moreover, the mth segment of the spike train $\mathbf{X}$ is denoted by $\mathrm{x}_m = [x[m], x[m-1], \ldots, x[m-N+1]]^T$, where N is the length of the segment. Suppose the spike train is processed by a matched filter with template $\mathrm{t} = [t[0], \ldots, t[N-1]]^T$. Let $\overline{\mathbf{x}}_m$ and $\overline{\mathbf{t}}$ be the normalized version of $\mathbf{x}_m$ and $\mathbf{t}$, respectively. That is,

$$\overline{\mathbf{x}}_m = \frac{\mathbf{x}_m}{\left\|\mathbf{x}_m\right\|},\ \overline{\mathbf{t}} = \frac{\mathbf{t}}{\left\|\mathbf{t}\right\|}. \tag{1}$$

The normalized output at m, denoted by, $\bar{y}[m]$, is computed from

$$\bar{y}\left[m\right] = \sum_{k=0}^{N-1} \bar{x}\left[m-k\right]\bar{t}\left[k\right] = \bar{\mathbf{x}}_m^T \bar{\mathbf{t}} \tag{2}$$

This is the inner product of segment $\bar{\mathbf{x}}_m$ and template $\bar{\mathbf{t}}$, which indicates the normalized correlation between these two vectors. The segment $\mathbf{x}_m$ is detected as a spike when $\bar{y}[m]$ is larger than a prespecified threshold η. Let $d\left(\bar{\mathbf{x}}_m, \bar{\mathbf{t}}\right)$be the squared distance between $\bar{\mathbf{x}}_m$and $\bar{\mathbf{t}}$. It can be shown that

$$d(\bar{\mathbf{x}}_m, \bar{\mathbf{t}}) = 2 - 2\bar{\mathbf{x}}_m^T \bar{\mathbf{t}}. \tag{3}$$

Because $d\left(\bar{\mathbf{x}}_m, \bar{\mathbf{t}}\right) \geq 0$,

$$\bar{\mathbf{x}}_m^T \bar{\mathbf{t}} \leq 1. \tag{4}$$

Our normalized correlation operations are based on $\bar{\mathbf{x}}_m$ and $\bar{\mathbf{t}}$. When $\bar{\mathbf{x}}_m^T \bar{\mathbf{t}} \geq \eta$, then $\mathbf{x}_m$is detected as a spike. From Eq. (4), it follows that

$$\eta \leq 1 \tag{5}$$

The normalized correlation operations shown in Eq. (2) may have high computational complexities. Although the template $\bar{\mathbf{t}}$ can be computed beforehand, the computation of $\bar{\mathbf{x}}_m$ still needs to be carried out online, involving the calculation of$||\mathbf{x}_m||$and $\bar{\mathbf{x}}_m = \frac{\mathbf{x}_m}{\left\|\mathbf{x}_m\right\|}$. Note that N multiplications, N−1 additions, and one squared root operation are required for the computation of $||\mathbf{x}_m||$. Moreover, N divisions are needed for $\bar{\mathbf{x}}_m$. Finally, the inner product of $\bar{\mathbf{x}}_m^T \bar{\mathbf{t}}$ requires N multiplications and N−1 additions. In total, the basic implementation of the normalized correlation operations requires 2N multiplications, (2N−2) additions, N divisions, and 1 squared root operation. When the fast computation is an important concern, the hardware implementation may then be desirable.

2.2. OSORT algorithm

The OSORT algorithm is an unsupervised template-based clustering algorithm for spike sorting. It does not require feature extraction, and the number of clusters c is automatically determined by the algorithm. Define $\mathcal{C}_i$, $i = 1, ..., c$, as the clusters of spikes generated by the OSORT algorithm, where $\mathbf{t}_i$, $i = 1, \ldots, c$, is the average value of the spikes belonging to $\mathcal{C}_i$.

Given a current detected spike $\mathbf{s}$ for clustering, in the OSORT algorithm, the squared distances $d_i = d(\mathbf{s}, \mathbf{t}_i)$ for $i = 1, \ldots, c$ are first computed. The minimum distance d_{i*} is then identified, where $i^* = \arg \min_i d_i$. We will assign $\mathbf{s}$ to $\mathcal{C}_{i*}$ when d_{i*} is less than a prespecified threshold τ_1. In this case, because $\mathcal{C}_{i*}$ has a new member, its mean value $\mathbf{t}_{i*}$ will also be updated. Otherwise, a new cluster is created, where $\mathbf{s}$ is its only member. After the updating of $\mathbf{t}_{i*}$, the cluster merging process will be activated. The process involves the computation of the distance between $\mathbf{t}_{i*}$ and $\mathbf{t}_j$, $i^* \neq j$. Two clusters $\mathcal{C}_{i*}$and $\mathcal{C}_{j*}$will be merged when $d\left(\mathbf{t}_{i*}, \mathbf{t}_{j*}\right) < \tau_2$, where$j^* = \arg \min_{j, j \neq i^*} d\left(\mathbf{t}_{i*}, \mathbf{t}_j\right)$. **Figure 1** summarizes the operations of the OSORT algorithm.

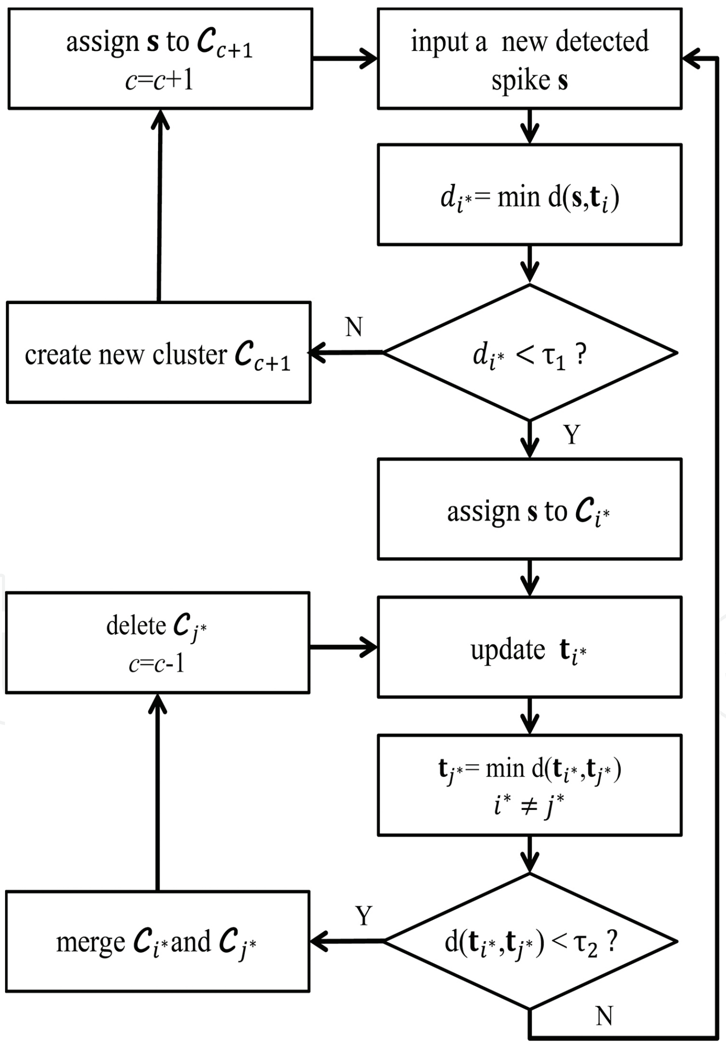
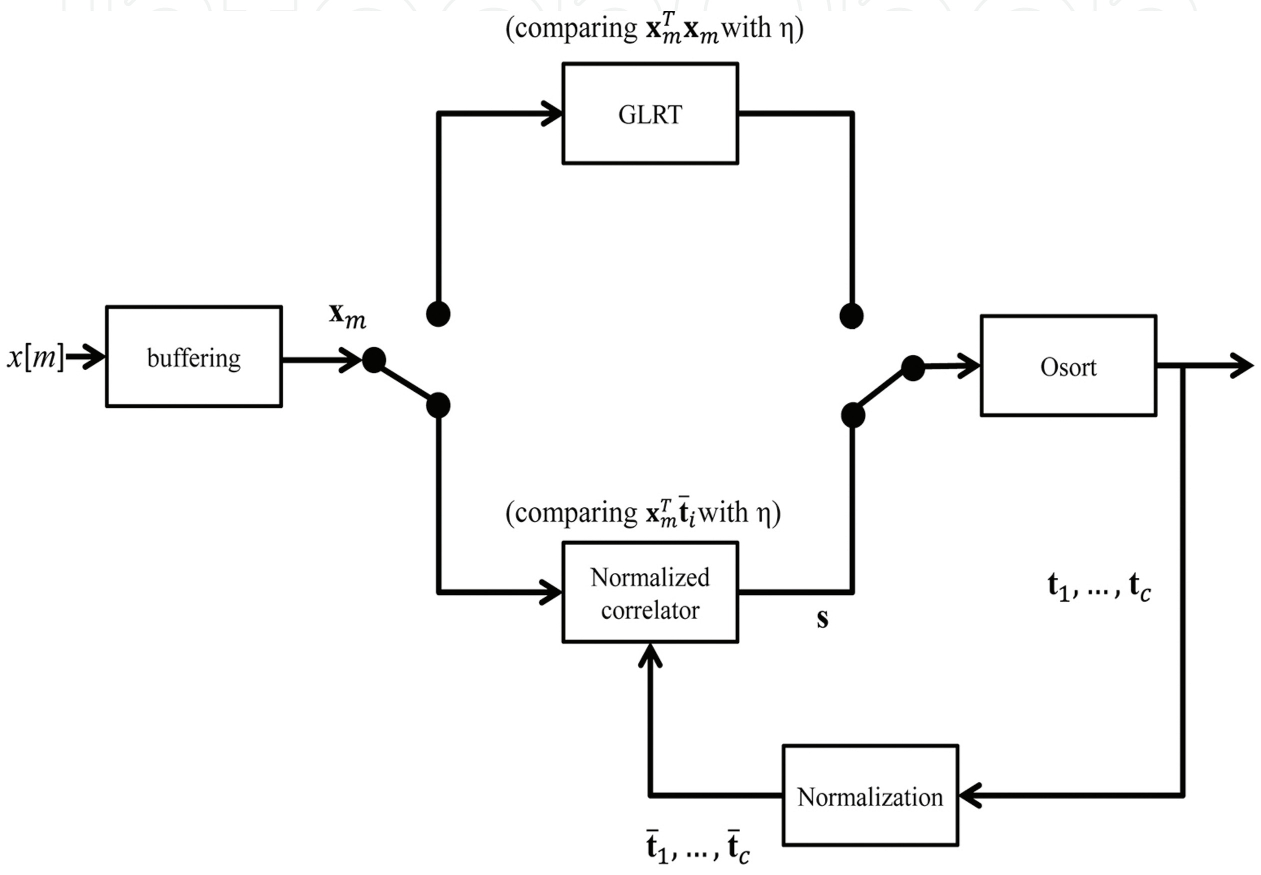

assign s to $\mathcal{C}_{c+1}$
c=c+1
input a new detected spike s
d_{i^*}= min d(s,$\mathbf{t}_i$)
N
create new cluster $\mathcal{C}_{c+1}$
$d_{i^*} < \tau_1$?
Y
assign s to $\mathcal{C}_{i^*}$
delete $\mathcal{C}_{j^*}$
c=c-1
update $\mathbf{t}_{i^*}$
$\mathbf{t}_{j^*}$= min d($\mathbf{t}_{i^*}$,$\mathbf{t}_{j^*}$)
$i^* \neq j^*$
Y
merge $\mathcal{C}_{i^*}$and $\mathcal{C}_{j^*}$
d($\mathbf{t}_{i^*}$,$\mathbf{t}_{j^*}$) < τ_2 ?
N

Figure 1. The flowchart of OSORT operations.

2.3. Normalized correlation and OSORT algorithm for spike sorting

By combining the normalized correlation with the OSORT algorithm, an effective spike sorting system for both spike detection and classification can be realized. The system is a feedback system capable of automatic template generation for spike detection and unsupervised clustering for the classification. The block diagram of the system is revealed in **Figure 2**.



(comparing $\mathbf{x}_m^T\mathbf{x}_m$ with η)
GLRT
$x[m]$
buffering
$\mathbf{x}_m$
Osort
(comparing $\mathbf{x}_m^T\bar{\mathbf{t}}_i$ with η)
Normalized correlator
$\mathbf{s}$
$\mathbf{t}_1, \ldots, \mathbf{t}_c$
Normalization
$\bar{\mathbf{t}}_1, \ldots, \bar{\mathbf{t}}_c$

Figure 2. The block diagram of the spike detection/sorting system based on GLRT test, normalized correlator, and OSORT algorithms.

Initially, the clusters and templates produced by the OSORT are not available. As a result, it may be difficult to carry out the normalized correlation for spike detection. One way to solve this problem is to use only the block energy for the detection of spikes. The detected spikes are then clustered by the OSORT algorithm for the generation of initial templates.

After the templates become available, the spike detection is then based on the normalized correlation $\bar{\mathbf{x}}_m^T\bar{\mathbf{t}}$. The input block is detected as a spike when any of the c normalized correlation exceeds the threshold η. Because of the normalized correlation operations, the threshold value is bounded as shown in Eq. (5). Note that the templates for spike detection will be updated regularly so that the variations of input signals can be tracked to improve the spike detection performance.

The hardware architecture for implementing the spike sorting system is depicted in **Figure 3**. The architecture contains two modules and one controller. The first module of the architecture, termed normalized correlator module, is responsible for the spike detection. It is capable of performing both the GLRT and normalized correlation operations. The second module, termed OSORT module, carries out the unsupervised OSORT spike sorting. The global controller coordinates the operations of these two modules. The architecture and detailed operations of the normalized correlator module and OSORT module are presented in the following two sections.

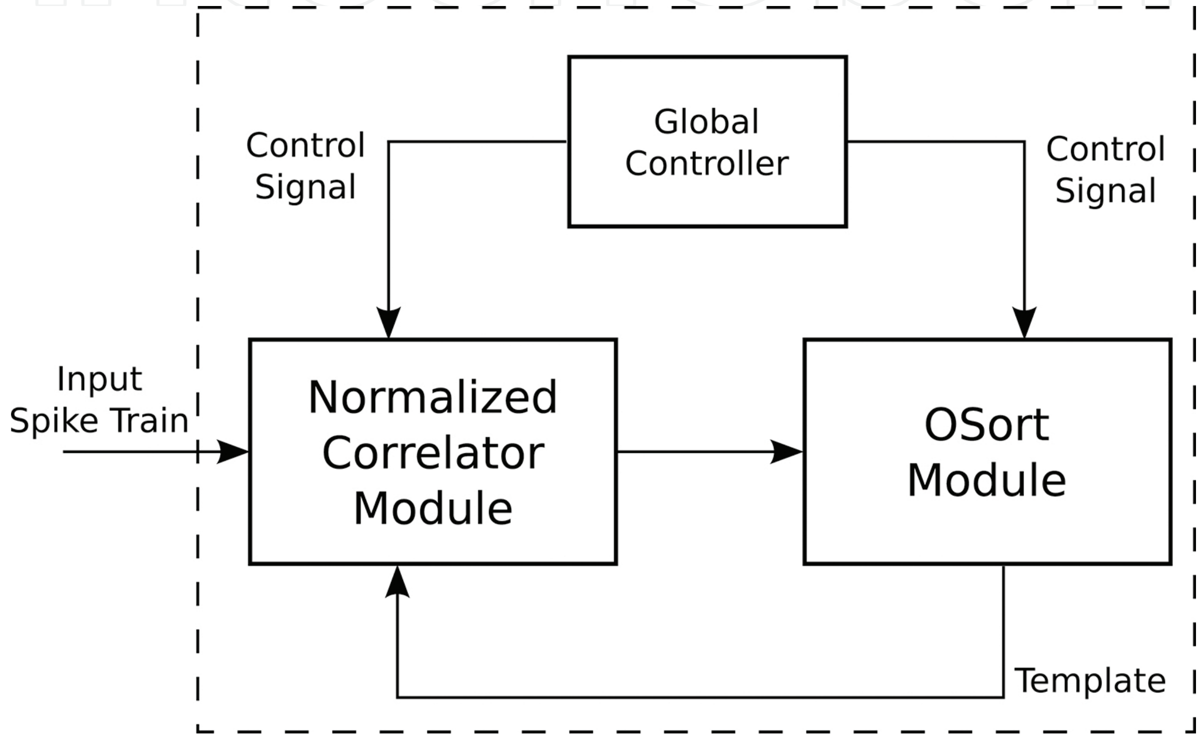


Figure 3. The hardware architecture of the proposed spike sorting system.

3. Architecture of the normalized correlator module

The block diagram of the normalized correlator module is revealed in **Figure 4**. The module supports the filtering, block energy computation, correlation computation, detection, and buffering. The filtering operation is the preprocessing step for the spike detection. It reduces the DC offset and noises. The objective of the block energy computation is to find $||\mathbf{x}_m||^2$, which is then followed by correlation computation for calculating $\bar{\mathbf{x}}_m^T\bar{\mathbf{t}}$. The detection results are then produced by the comparison operations. The detected spikes are stored in the switch buffer, which can be accessed by the OSORT module for subsequent clustering operations. Without loss of generality, the length of spike is set to be $N = 64$ for our discussion.

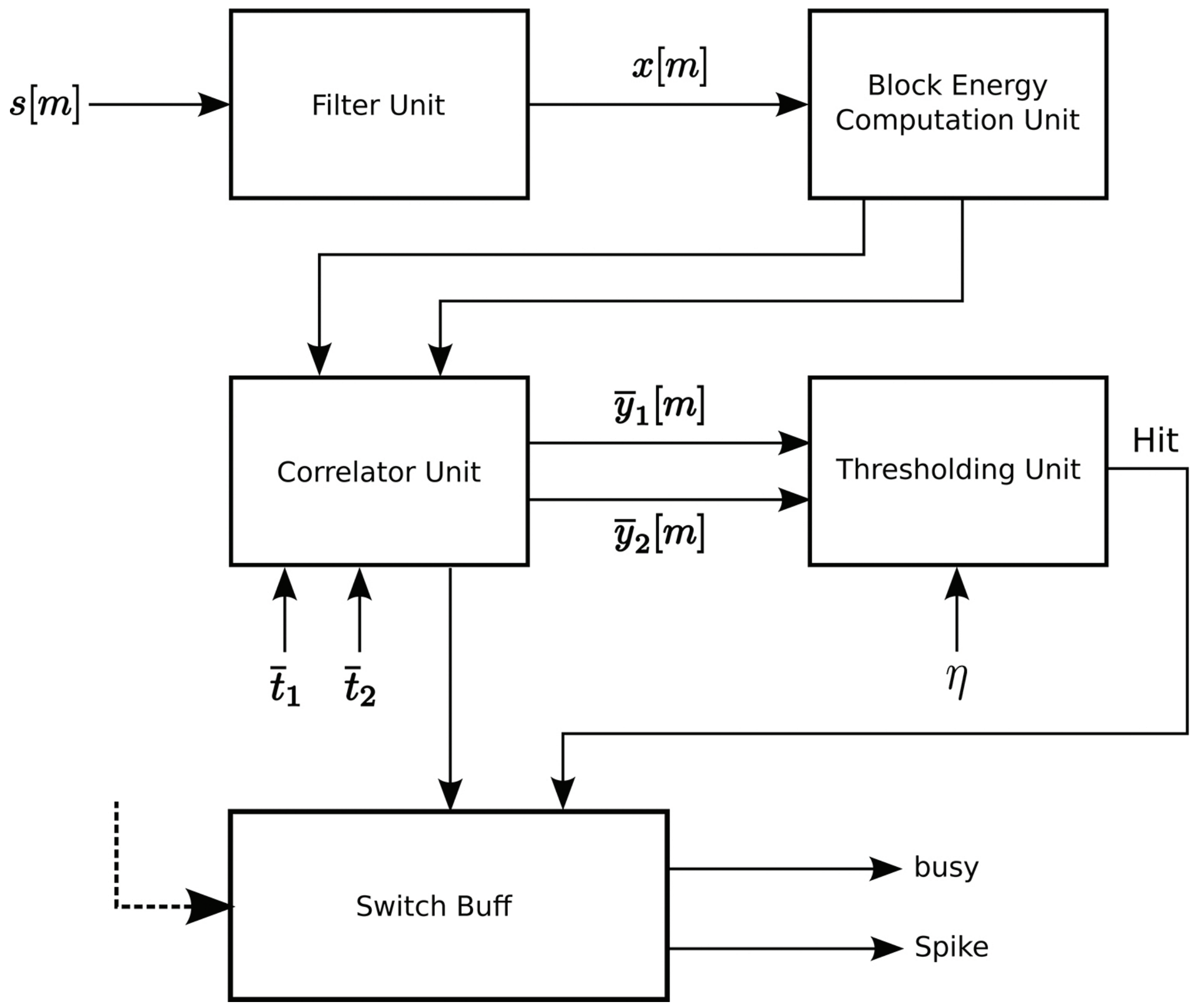


Figure 4. The block diagram of the normalized correlator module.

3.1. Filter unit and block energy computation unit

The filter unit is a hardware implementation of a band-pass Butterworth filter. The filter unit contains multipliers, shift registers, and adders. The architecture is a simple realization of the direct form I of IIR filters. To implement the block energy computation unit, we first note that a basic approach may involve N multiplications for the energy computation, resulting in high-area costs. The proposed approach is based on the fact that

$$\left\|\mathbf{x}_m\right\|^2 = \left\|\mathbf{x}_{m-1}\right\|^2 + x^2\left[m\right] - x^2\left[m-N\right]. \tag{6}$$

Consequently, the calculation of $||\mathbf{x}_m||^2$ needs only two multiplications. This is because $||\mathbf{x}_{m-1}||^2$ (i.e., the block energy of the previous block) is already available. **Figure 5** shows the resulting design, which contains two multiplier, a single N-stage shift register, and two adders. The goal of the shift register is to store the previous samples (i.e., $x[k]$, $k = m - 1,\dots, m - N$) of

$x[m]$. The shift register therefore is able to offer the sample $x[m - N]$ for the calculation of $x^2[m - N]$. In addition, the shift register can be employed for the correlation computation.

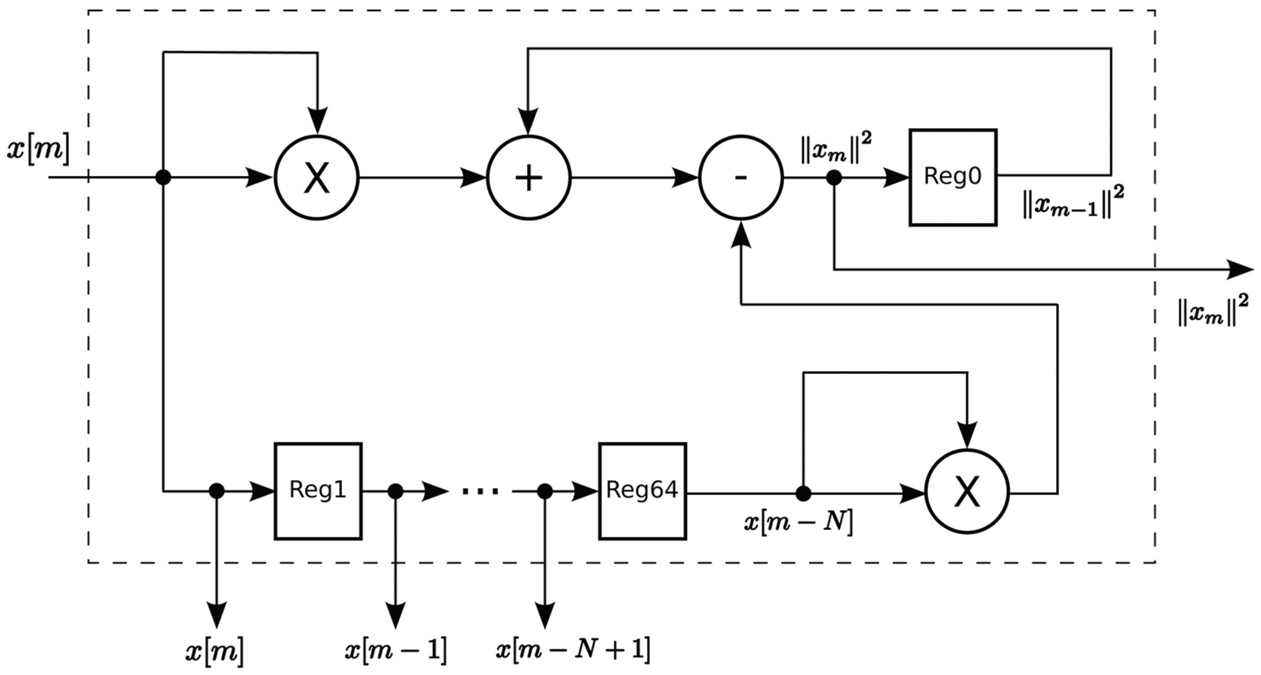


Figure 5. Architecture of block energy computation unit.

3.2. Correlator unit

The goal of the unit is to carry out the normalized correlation $\bar{\mathbf{x}}_m^T\bar{\mathbf{t}}$. Note that the normalized template $\bar{\mathbf{t}}$ can be obtained offline from the OSORT circuit. Therefore, it is only necessary to find $\bar{\mathbf{x}}_m$ online. One simple approach to compute $\bar{\mathbf{x}}_m$ is to divide each sample of $\mathbf{x}_m$ by $||\mathbf{x}_m||$. Because the block $\mathbf{x}_m$ contains N samples, N dividers are required. In the proposed architecture, a novel postnormalization approach is employed, where the inner product $\mathbf{x}_m^T\bar{\mathbf{t}}$ is computed first. Because $\mathbf{x}_m^T\bar{\mathbf{t}}$ is a scalar, we can then use only one divider to compute $\bar{\mathbf{x}}_m^T\bar{\mathbf{t}}$ by dividing $\mathbf{x}_m^T\bar{\mathbf{t}}$ by $||\mathbf{x}_m||$.

Figure 6 shows the architecture of the correlator unit for the case of two templates. The circuit consists of $2N$ multipliers, one squared root circuit, two accumulators, and one divider. Moreover, there are two registers for storing the normalized templates $\bar{\mathbf{t}}_1$ and $\bar{\mathbf{t}}_2$. Recall that the shift register in the block energy computation unit contains the samples of $\mathbf{x}_m$. Based on $\mathbf{x}_m$ and $\bar{\mathbf{t}}_i$, $i = 1, 2$,the computation of each $\mathbf{x}_m^T\bar{\mathbf{t}}_i$, $i = 1, 2$, is carried out in parallel. Moreover, the multiplication results are accumulated in a pipelined fashion. The accumulation results are then scaled by a factor of $1/||\mathbf{x}_m||$. Because the block energy computation unit provides

$||\mathbf{x}_m||^2$, only a squared root circuit and an inverse circuit are needed for the calculation of $1/||\mathbf{x}_m||$, as shown in **Figure 6**.

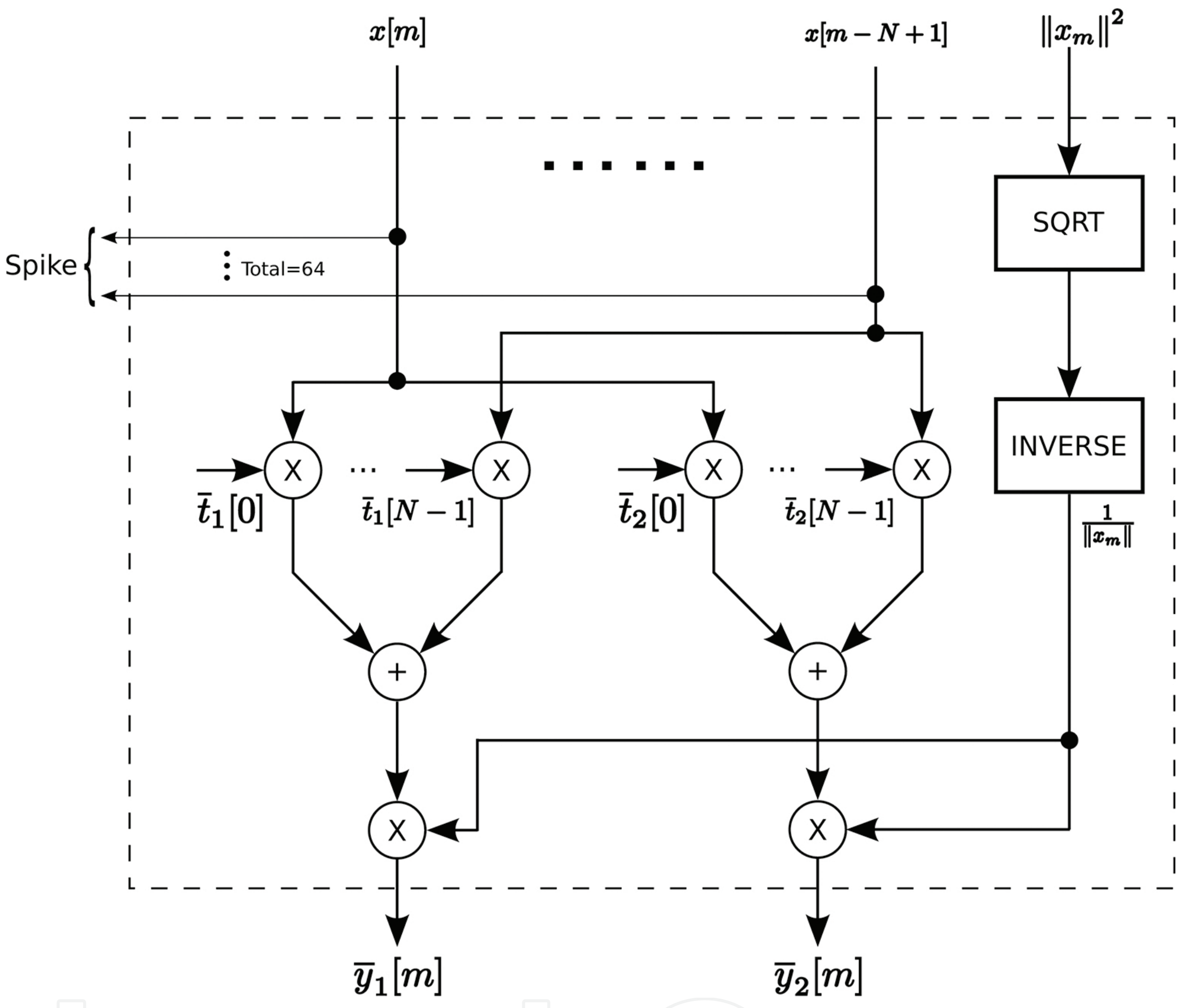


Figure 6. Architecture of correlator unit.

3.3. Threshold unit

Although the operations of the unit can be easily accomplished by a simple comparison circuit, the detection accuracy may be further improved by taking the detection results of the neighboring blocks into consideration. Because the neighboring blocks are overlapping, they may be similar. As a result, the normalized correlation values of the neighboring blocks may also be similar. Therefore, it is likely that an occurrence of a single spike may result in the issues of multiple hits.

To solve this problem, when the normalized correlation value of a block is above the threshold, a hit is not immediately declared. The architecture will then examine the normalized correlation values of the previous blocks. A hit would actually be issued only if *k* out of *K* preceding blocks have normalized correlation values above a threshold. In this way, the false alarm rate (FAR) can be effectively lowered. **Figure 7** shows the corresponding architecture, which contains a *K*-stage shift register storing the comparison results of the *K* previous blocks. Each stage of the shift register contains only a single-bit information, where 1 indicates that the corresponding block has normalized correlation value above the threshold η, and 0 otherwise. Therefore, if the sum of all the *K* stages is larger or equal to *k*, then at least *k* preceding blocks have normalized correlation value above the threshold. In this case, the architecture issues a hit.

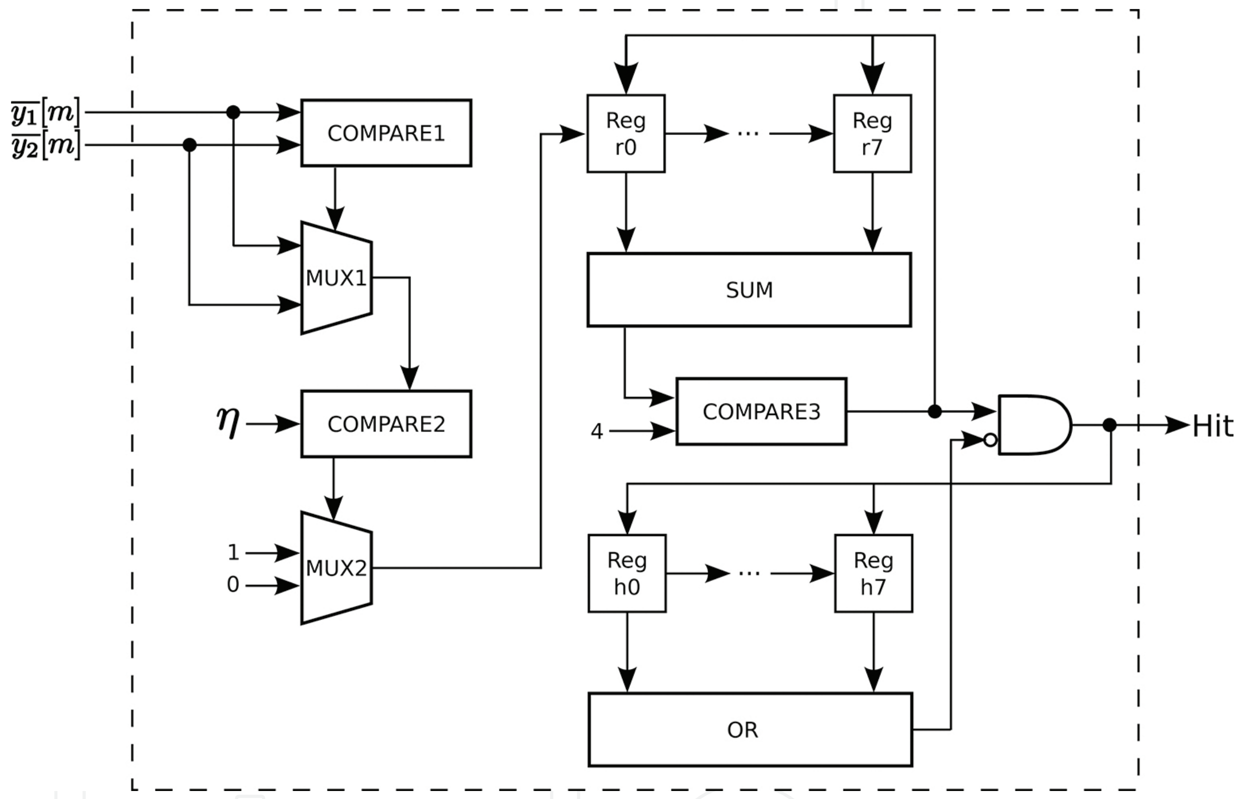


Figure 7. Architecture of threshold unit.

3.4. Switch buffer

The goal of switch buffer is to store the detected spikes for subsequent clustering operations. As shown in **Figure 8**, there are two buffers (denoted as Buffer x and Buffer y) in the circuit. When one of the buffers stores the detected spikes, the other provides the detected spikes to the OSORT module for clustering operations. The switch controller in the circuit is responsible for the determination of the buffer to store the detected spikes. The flowchart of the operations of the switch controller is shown in **Figure 9**. From the flowchart, it can be observed that the controller assigns the detected spikes to a buffer in accordance with the availability of that buffer. A buffer is available when it has empty cells for storing new detected spikes, and is not currently providing spikes to the OSORT module.

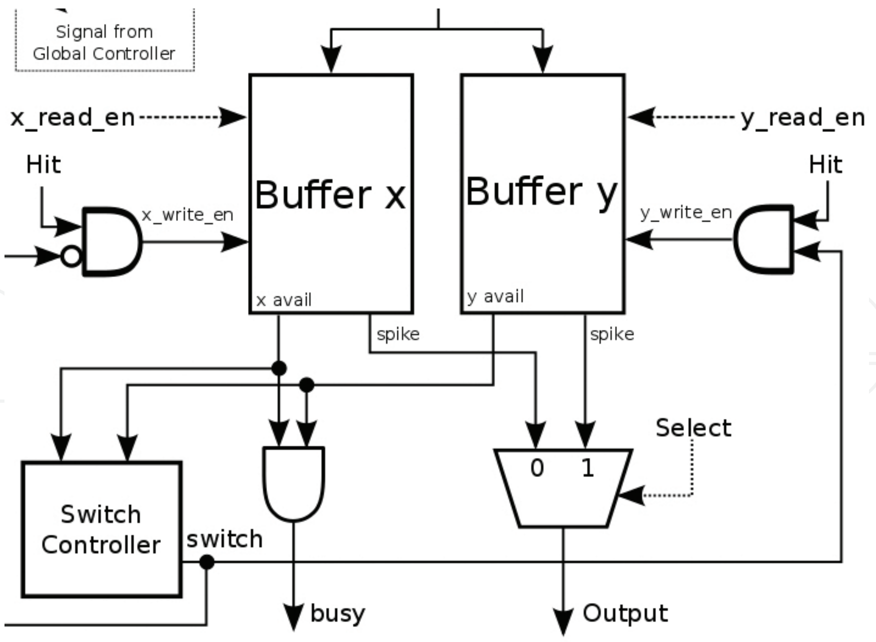


Figure 8. Architecture of switch buffer.

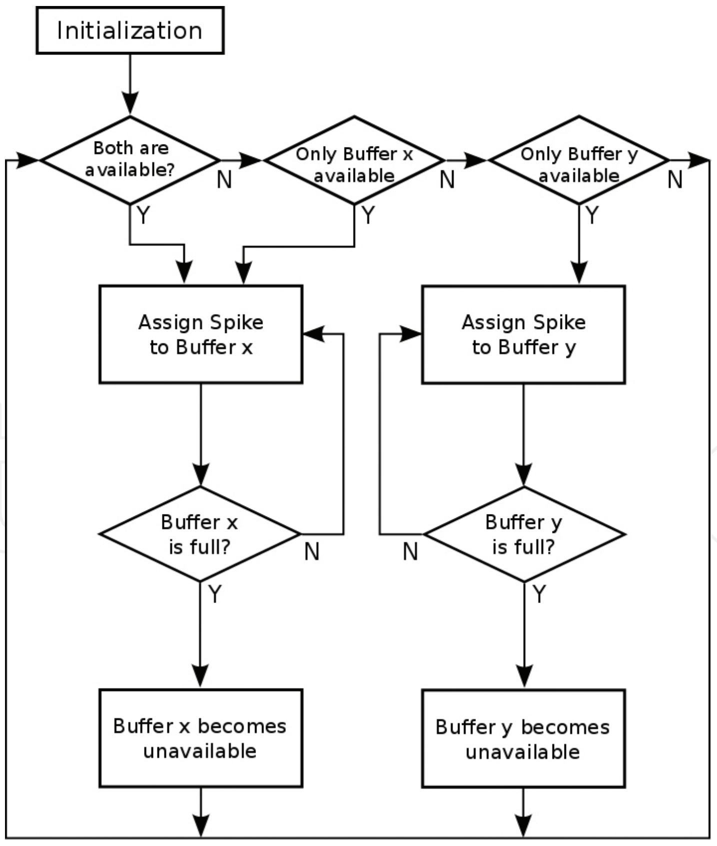


Figure 9. Flowchart of switch controller.

4. The architecture of OSORT module

The OSORT module contains buffers, distance computation unit, mean updating unit, comparator, and controller, as shown in **Figure 10**. The centroid and the size of each cluster are stored in the buffers. The distance computation unit and mean updating unit are responsible for squared distance computation and the updating of centroid of the clusters, respectively. The control unit coordinates different components of the OSORT module for carrying out the unsupervised clustering operations.

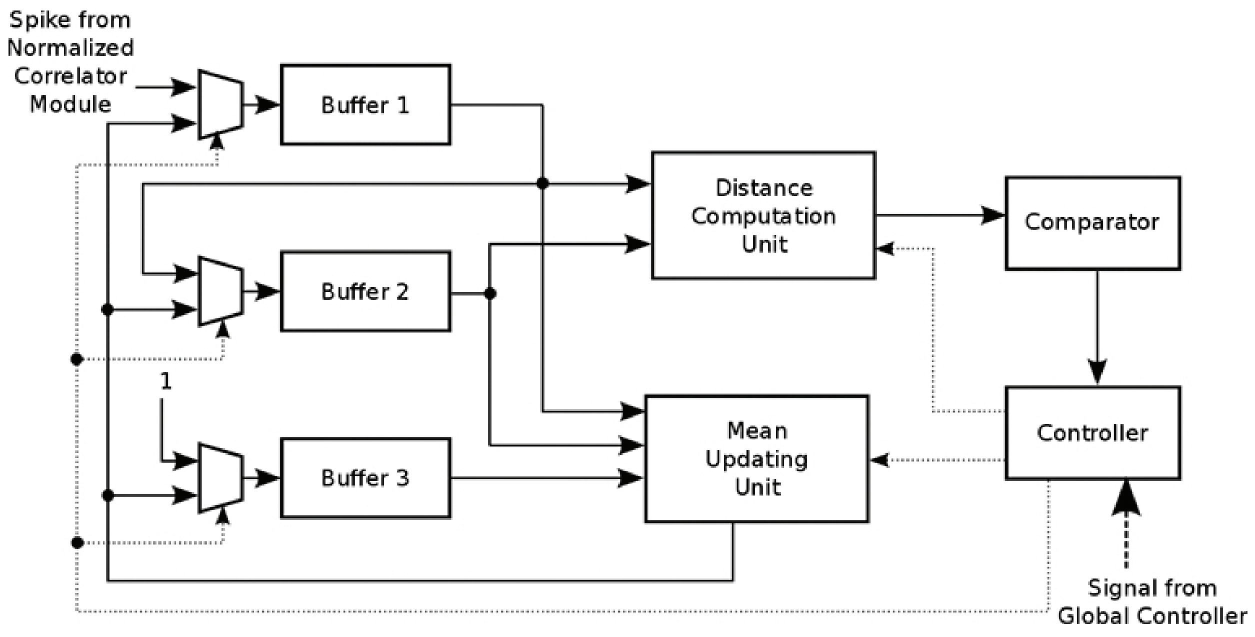

Spike from Normalized Correlator Module
Buffer 1
Distance Computation Unit
Comparator
Buffer 2
1
Mean Updating Unit
Controller
Buffer 3
Signal from Global Controller

Figure 10. Architecture of OSORT module.

4.1. Buffers

There are three buffers in the OSORT module, which are denoted by Buffer 1, Buffer 2, and Buffer 3, respectively. Buffer 1 holds an input spike detected by the correlators. Buffers 2 and 3 contain the mean value and size of each cluster, respectively. Buffer 1 is a simple N-stage shift register, fetching or delivering one sample of the input spike at a time. As shown in **Figure 11**, Buffer 2 contains QN-stage shift registers. Each shift register holds the mean value of a cluster. Therefore, Q is the upper-bound of the number of clusters. Buffer 2 updates or provides mean values of clusters one at a time. Because the mean value $\mathbf{t}_i$ of a cluster $\mathcal{C}_i$ is the average value of the spikes mapping to that cluster, the mean value also contains N samples. Accessing the mean value of the cluster is also carried out one sample at a time. Buffer 3 records the size of each cluster. There are Q entries in the buffer. The ith entry contains the number of spikes in the cluster $\mathcal{C}_i$.

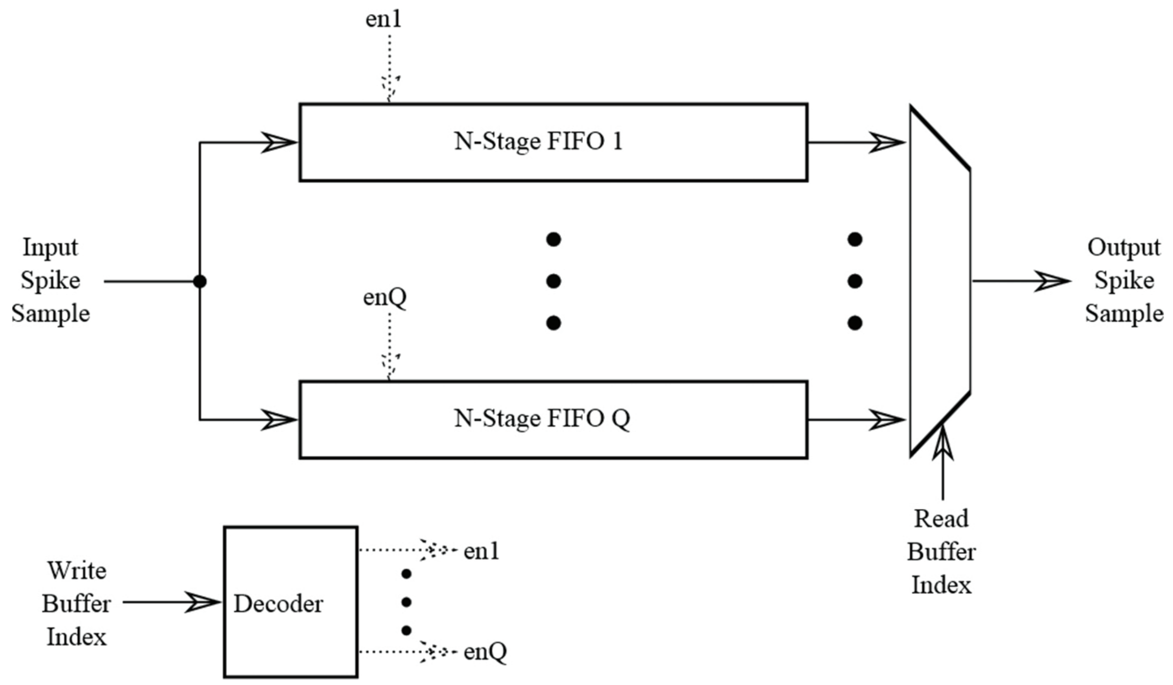


Figure 11. Architecture of Buffer 2.

4.2. Distance computation unit and mean updating unit

Because buffers in the memory unit can be accessed one sample at a time, the circuits in the distance computation unit and mean updating unit provide only sample-wise computations. This is beneficial for reducing the area costs. There are two cases when the distance computation unit needs to be activated. In the first case, a new spike is arrived. To find the cluster for the new spike, the squared distance computation is required. In the second case, it is desired to merge two clusters. The squared distance calculation is needed for finding the closest clusters. The distance computation unit takes the samples fetched from Buffer 1 and Buffer 2 as inputs. The unit finds the squared distance between the spikes stored in Buffer 1 and the mean value of a cluster selected in Buffer 2. Upon the completion of the squared distance computation, the comparison of the new squared distance with the current minimum distance stored in a register of the unit is carried out. If the new squared distance is smaller than the current minimum distance, then it becomes the new current minimum distance. The same squared distance computation and comparison operations will be repeated for until all the mean values in Buffer 2 are searched. This scheme is useful for finding the best matching mean value stored in Buffer 2 to the spike stored in Buffer 1.

The mean updating unit is activated after a new spike is assigned to a cluster, or after two clusters are merged. In these cases, the mean of the updated cluster needs to be computed. Note that the clusters to be updated are determined by the distance computation unit. The mean updating unit is only responsible for the computation of the new mean of the updated clusters. The circuit takes waveforms stored in Buffer 1 and Buffer 2, and the cluster size stored in Buffer 3 as inputs. The updated mean is the weighted sum of the waveforms obtained from Buffer 1 and Buffer 2, as shown in **Figure 12**. The weights are determined from the cluster sizes from Buffer 3. The updated results are then stored back to Buffer 2 and Buffer 3.

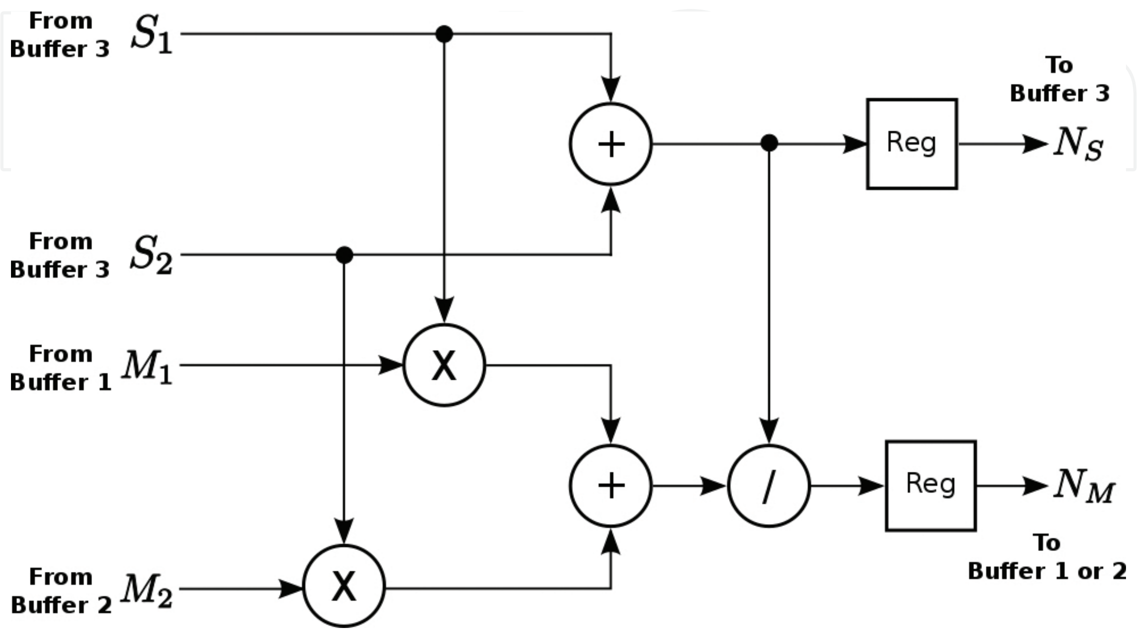


Figure 12. Architecture of mean updating unit.

4.3. Control unit

The control unit activates components of the memory unit and cluster computation unit for the unsupervised clustering. The states of the control unit are summarized in **Table 1**. In addition to the tasks carried out by each state, the activated circuit components associated with each state are also included in the table. **Figure 13** shows the flowchart of the OSORT algorithm in terms of the states defined in **Table 1**.

States	Activated components	Operations
State 1	Buffer 1 Distance computation unit	Fetch a new spike to Buffer 1
State 2	Buffers 1, 2	Find the best matching cluster to the spike in Buffer 1
State 3	Buffers 1, 2, 3	Creating a new cluster
State 4	Buffers 1, 2, 3 Mean updating unit	Update the mean and size of a cluster
State 5	Buffers 2, 3	Remove a cluster

Table 1. States of the control unit in OSORT module.

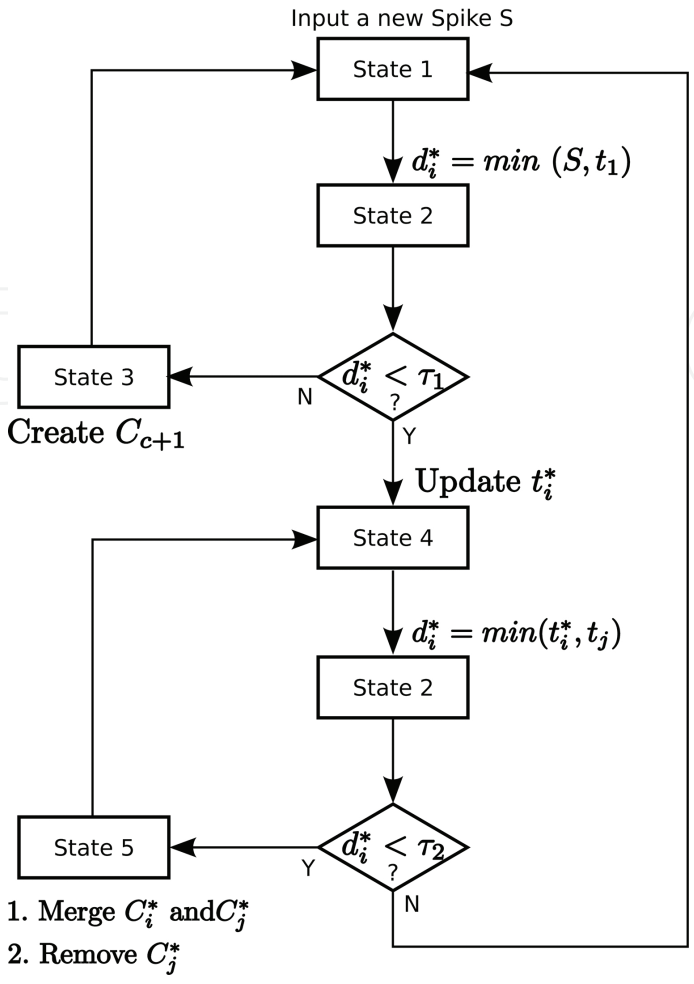


Figure 13. The flowchart of the controller in the OSORT module.

The flowchart in **Figure 13** is consistent with that shown in **Figure 1**. Although the combinations of the states shown in **Table 1** are able to implement the OSORT algorithm, additional modifications may still be desirable to facilitate the hardware implementation. One modification implemented in the controller is to handle the cases when the current number of clusters reaches the upper limit Q, and the creation of new cluster is still desired. In this case, the least recently updated cluster will be replaced by the new cluster. To carry out this modification, an additional field is added to each entry of Buffer 3. The entry indicates the number of updates in the past for the corresponding cluster. This modification can be viewed as an additional function supported in State 3 in **Table 1** for the creation of a new cluster.

5. Global controller and NOC

As depicted in **Figure 3**, the global controller in the proposed spike sorting system coordinates the operations of the normalized correlator module and OSORT module. The major goal of the global controller is to fetch detected spikes from the normalized correlator module and deliver them to the OSORT module. When a buffer in the switch buffer of the normalized correlator module becomes full, the global controller starts to fetch the detected spikes one at a time from the buffer to the OSORT module.

The fetching operations are repeated until all the spikes stored in the buffer are fetched. At this time, the buffer becomes available again for storing the new detected spikes, as shown in **Figure 14**. The proposed hardware spike sorting system is configured as a user component in a NOC system, which is designed by the QSYS platform. In addition to the proposed system, we see from **Figure 15** that the NOC contains the NIOS II processor, a DMA controller, an on-chip RAM, and a hardware timer.

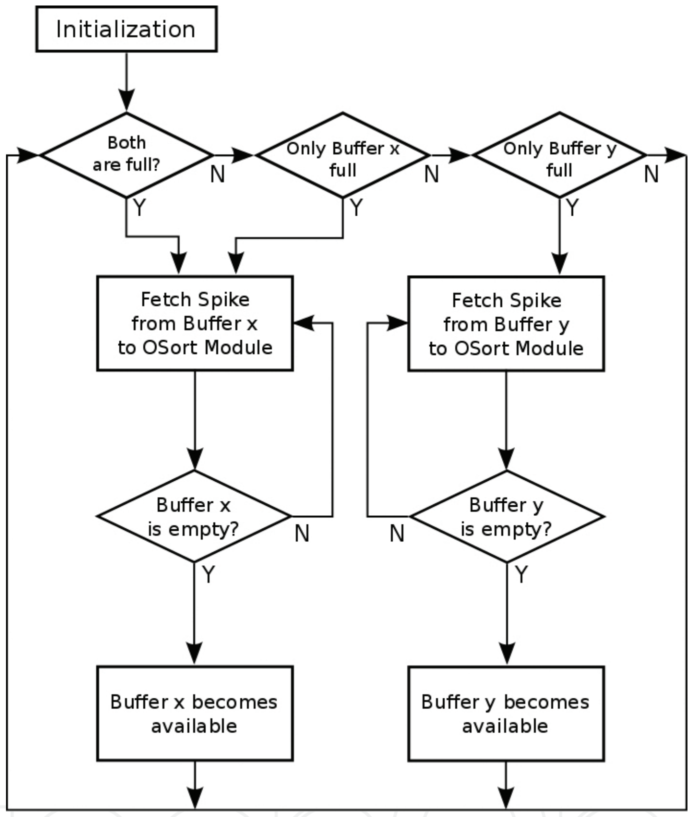

Initialization
Both are full?
Only Buffer x full
Only Buffer y full
N
Y
Fetch Spike from Buffer x to OSort Module
Fetch Spike from Buffer y to OSort Module
Buffer x is empty?
Buffer y is empty?
Buffer x becomes available
Buffer y becomes available

Figure 14. The flowchart of the global controller.

The raw spike trains are stored in the on-chip RAM. The DMA controller is responsible for delivering the spike trains to the proposed hardware system without the intervention of the NIOS II processor. The DMA controller is able to halt the delivery of spike trains automatically when both buffers in the switch buffer of the normalized correlator module are unavailable and/or full. The hardware timer is used to measure the computation speed of the proposed system. The NIOS II processor integrates different components in the NOC. It activates the DMA controllers for the delivery of spike trains. After that, the processor collects the spike sorting results from the OSORT modules of the proposed spike sorting system. The processor is also able to read the information provided by the hardware timer for the measurement of the computation speed of the proposed circuit.

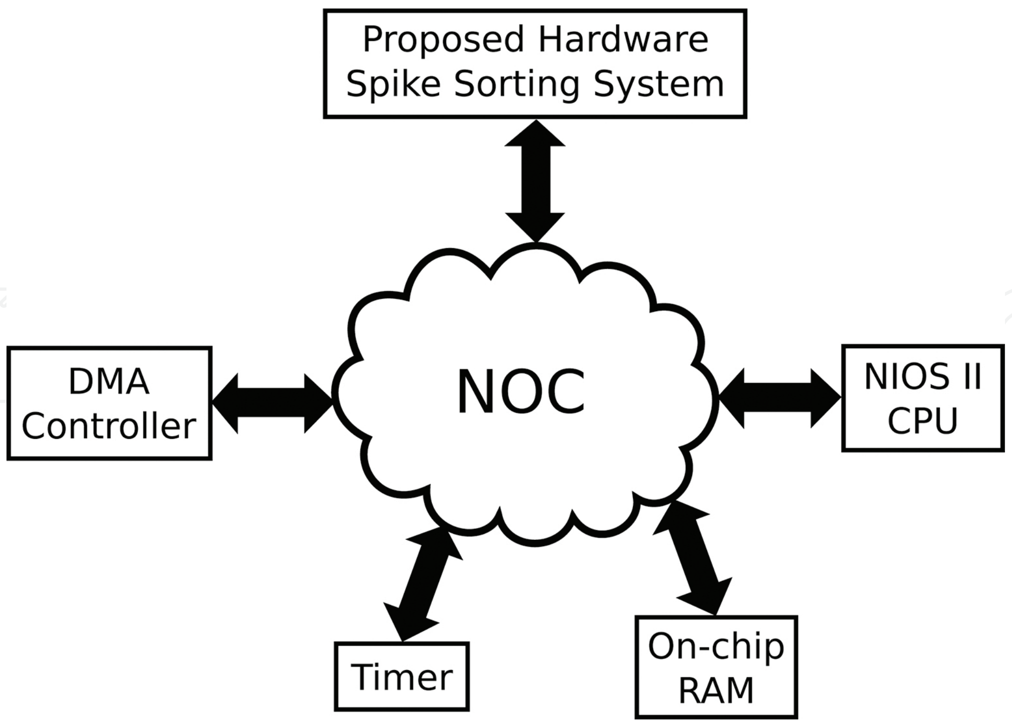


Figure 15. The proposed NOC system for spike sorting.

6. Experimental results

This section presents some experimental results of the proposed architecture. The extracellular recordings for the experiments are based on the simulator developed in [21], where the ground truth about spiking activity can be accessed. Each spike has length 2.67 ms. The sampling rate for the spike recording is 24,000 samples/s. Therefore, there are 64 samples (i.e., $N = 64$) in each spike.

The performance of the proposed architecture for spike detection is first evaluated. The performance evaluation is based on the true positive rate (TPR) and false alarm rate (FAR). The TPR of a detection algorithm is defined as the number of true spikes detected by the algorithm divided by the total number of true spikes. The FAR of a detection algorithm is the number of silent segments, which are falsely detected as spikes by the algorithm, divided by the total number of the segments detected by the algorithm. The TPR and FAR of various detection algorithms are included in **Table 2**. In the experiments, the spike trains are from two neurons. Therefore, there are two templates (i.e., $c = 2$) for the proposed normalized correlator.

Because the normalized correlation is effective for detecting real spikes and ignoring silent segments, we can observe from **Table 2** that the proposed architecture has superior performance over the other algorithms. We use the example shown in **Figure 16** to further demonstrate this fact. In the example, a noisy spike train with SNR = −3 dB is used for the spike detection. **Figure 16** reveals the normalized correlation values $\bar{y}_i[m]$, $i = 1, 2$, for the spike train. From

Figure 16, we see that, because of large noise corruption, it is difficult to locate spikes even by direct eye inspection. However, based on the normalized correlation values provided by the proposed architecture, the location of true spikes can still be effectively identified.

SNR (dB)		Normalized correlator	Noncoherent energy detector [17]	NEO [12]	SWT [22]	Matched filter [16]
10	TPR	93.64%	91.37%	93.10%	94.82%	89.65%
	FAR	0.40%	5.35%	3.57%	6.77%	2.80%
1	TPR	90.04%	88.03%	87.21%	92.43%	82.90%
	FAR	0.92%	6.36%	22.49%	79.36%	3.02%
−3	TPR	82.71%	82.60%	80.53%	86.66%	80.31%
	FAR	1.06%	9.52%	57.87%	82.43%	8.92%

Table 2. TPR and FAR values of various spike detection algorithms.

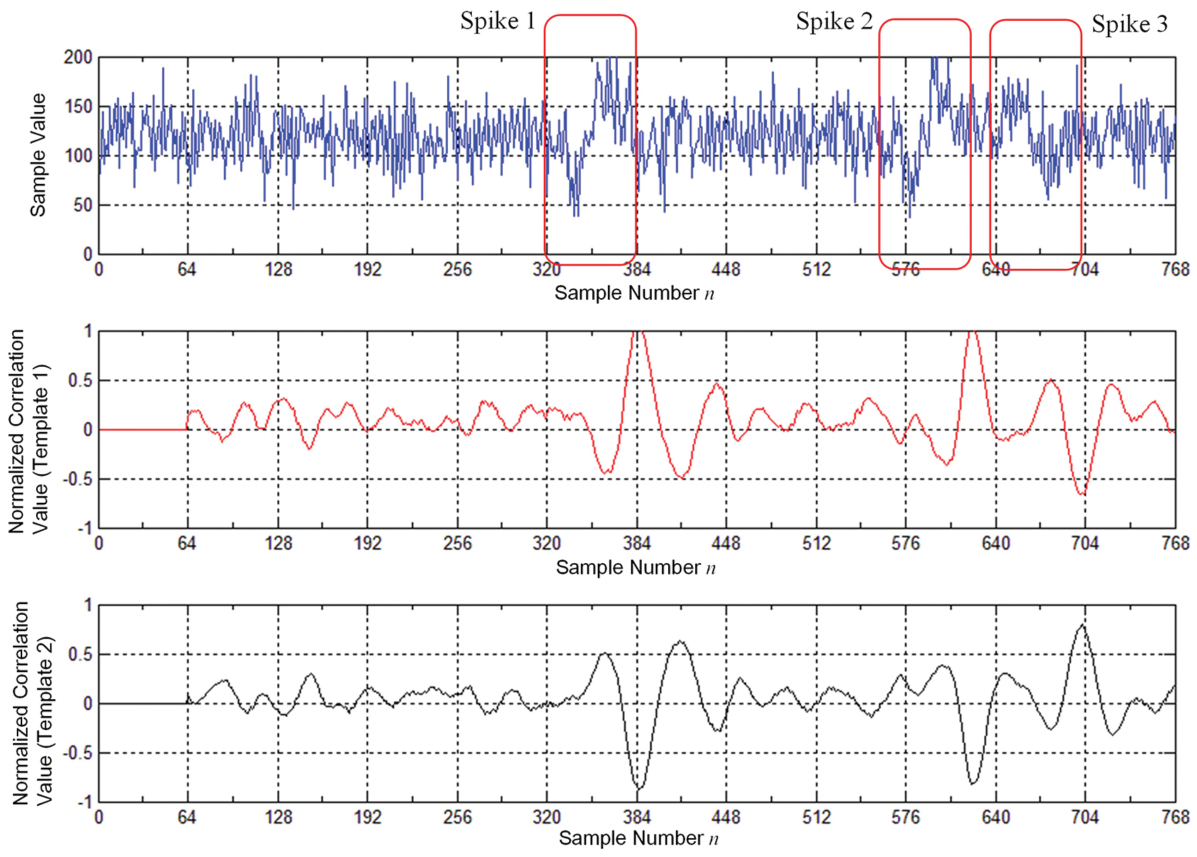


Figure 16. An example of the proposed normalized correlator for noisy spike detection with SNR = −3 dB for $c = 2$ templates.

Next we evaluate the area complexities. Because adders, multipliers, dividers, comparators, and registers are the basic building blocks of the proposed architecture, the area complexities are separated into five types: the number of adders, multipliers, dividers, comparators, and registers. **Tables 3** and **4** show the area complexities of the normalized correlator and OSORT modules, respectively. It can be observed from **Table 3** that, in the normalized correlator module, the correlator unit and switch buffer have larger area complexities. The number of adders, multipliers, and registers grows with the block dimension *N* and the number of templates *c* in the correlator unit. Let *L* be the capacity (i.e., the maximum number of spikes) of each buffer in the switch buffer. The number of registers in the switch buffer therefore is dependent on *L* and *N*, as shown in **Table 3**. The area complexities of the other types are of *O*(1). Therefore, the proposed circuit has low consumption of dividers and comparators. From **Table 4**, we observe that only the area complexities of the buffers in the OSORT module grow with *N*. The other parts of the OSORT module have fixed area complexities.

	Filter unit	Block energy computation	Correlator	Thresholding unit	Switch buffer	Subtotal
Adders	$O(1)$	$O(1)$	$O(cN)$	$O(1)$	0	$O(cN)$
Multipliers	$O(1)$	$O(1)$	$O(cN)$	$O(1)$	0	$O(cN)$
Dividers	0	0	1	0	0	1
Comparators	0	0	0	$O(1)$	0	$O(1)$
Registers	$O(1)$	$O(N)$	$O(cN)$	$O(1)$	$O(LN)$	$O(cN + LN)$

Table 3. Area complexities of the normalized correlator module.

	Buffer	Distance computation unit	Mean updating unit	Subtotal
Adders	0	$O(1)$	$O(1)$	$O(1)$
Multipliers	0	$O(1)$	$O(1)$	$O(1)$
Dividers	0	0	$O(1)$	$O(1)$
Comparators	0	$O(1)$	0	$O(1)$
Registers	$O(cN)$	$O(1)$	$O(1)$	$O(cN)$

Table 4. Area complexities of the OSORT module.

The proposed architecture has been implemented by FPGA for performance measurement. The target FPGA device for the hardware implementation is Altera STRATIX IV EP4SGX230. The design platform for the experiments is the Altera QUARTUS II with QSYS. **Table 5** shows the hardware utilization of the proposed architecture. There are four different FPGA hardware resources considered: adaptive look-up tables (ALUTs), dedicated logic registers, block memory bits, and DSP blocks. The DSP blocks are dedicated to the implementations of adders, multipliers, dividers, and comparators. The ALUTs, dedicated logic registers, and block memory bits can be used for the implementation of registers, as well as adders, multipliers, dividers, and comparators. It can be observed from **Table 5** that the consumption of DSP blocks of normalized correlator is higher than that of the OSORT module. This is because the normalized correlator requires more number of arithmetic operators. There are 182,400 ALUTs, 182,400 dedicated logic registers, 1288 DSP blocks, and 14,625,792 block memory bits in the target FPGA device. It can be observed from **Table 5** that only limited hardware resources are consumed by the proposed circuit.

	ALUTs	**Dedicated logic registers**	**Block memory bits**	**DSP blocks**
Normalized correlator module	13,966	29,733	0	532
OSORT module	22,604	22,562	320	88
Total	39,355/182,400 (21.57%)	52,517/182,400 (28.79%)	320/14,625,792 (<0.1%)	642/1288 (49.84%)

Table 5. The utilization of FPGA resources of the proposed circuit. The switch buffer capacity for the measurement is $L = 40$.

In addition to consuming low hardware resources, the proposed architecture is able to provide high throughput. **Table 6** reveals the throughput of the proposed architecture for various clock rates and switch buffer size L. The throughput is defined as the number of spike samples which can be processed by the proposed architecture per second. The unit of the throughput in the table therefore is mega samples per second (Msamples/sec). It can be observed from **Table 6** that the throughput grows with L and/or clock rate. In particular, when $L = 32$ and clock rate is 100 MHz, the throughput is 25.04 Msamples/sec. The throughput of its software counterpart running on Intel I7-930 processor at clock rate 2.8 GHz and 16 GB RAM is only 0.69 Msamples/sec. The throughput of the proposed architecture therefore is 36 times higher than that of its software counterpart.

Clock rate	$L = 20$	$L = 40$	$L = 80$
50 MHz	9.22 Msamples/sec	10.25 Msamples/sec	13.31 Msamples/sec
75 MHz	13.00 Msamples/sec	15.34 Msamples/sec	20.00 Msamples/sec
100 MHz	17.80 Msamples/sec	20.25 Msamples/sec	25.04 Msamples/sec

Table 6. The throughput of the proposed circuit for various clock rates and switch buffer capacities L.

7. Concluding remarks

The proposed architecture has been found to be effective for real-time spike sorting. It features high accuracy, low hardware resource consumption, and high throughput. The combination of the normalized correlation and OSORT algorithm is beneficial for accurate spike detection with high TPR and low FAR even for low SNR values. The postnormalization approach adopted by the normalized correlator circuit is also able to reduce the area costs for the normalization operations. In addition, the switch buffer in the correlation circuit can effectively coordinate the operations of spike detection and classification for achieving high throughput. Experimental results reveal that the proposed architecture achieves TPR = 82.71% and FAR = 1.06% for SNR = −3 dB. The ALUT consumption is only 21.57% for the FPGA device STRUTIX IV EP4SGX230. The throughput is 25.04 Msamples/sec for the clock rate 100 MHz. All these facts demonstrate the effectiveness of the proposed architecture.

Acknowledgements

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 details

Chien-Min Ou[1] and Wen-Jyi Hwang[2*]

*Address all correspondence to: whwang@csie.ntnu.edu.tw

1 Department of Electronics Engineering, Chien-Hsin University of Science and Technology, Taoyuan, Taiwan

2 Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei, Taiwan

References

[1] Einevoll, G. T.; Franke, F.; Hagen, E.; Pouzat, C.; Harris, K. D. Towards reliable spike-train recordings from thousands of neurons with multielectrodes, Current Opinion in Neurobiology 2012, 22, 11–17.

[2] Gibson, S.; Judy, J. W.; Markovic, D. Spike sorting: the first step in decoding the brain, IEEE Signal Processing Magazine 2012, 29, 124–143.

[3] Hauck, S.; Dehon, A. Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing, Morgan Kaufmann: San Francisco, CA, USA, 2008.

[4] Zhu, X.; Yuan, L.; Wang, D.; Chen, Y. FPGA implementation of a probabilistic neural network for spike sorting. In Proceeding of the IEEE International Conference on Information Engineering and Computer Science, Piscataway, New Jersey, USA. 2010; pp. 26–29.

[5] Dutta, K.; Prakash, N.; Kaushik, S. Probabilistic neural network approach to the classification of demonstrative pronouns for indirect anaphora in Hindi. Expert Systems with Applications 2010, 37, 5607–5613.

[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, Piscataway, New Jersey, USA. 2010; pp. 556–561.

[7] Haykin, S. Neural Networks and Learning Machines, 3rd ed.; Pearson: Upper Saddle River, NJ, USA, 2009.

[8] Hwang, W. J.; Lee, W. H.; Lin, S. J.; Lai, S. Y. Efficient architecture for spike sorting in reconfigurable hardware, Sensors 2013, 13, 14860–14887.

[9] Miyamoto, S.; Ichihashi, H.; Honda, K. Algorithms for Fuzzy Clustering, Springer: Berlin/Heidelberg, Germany, 2010.

[10] Gibson, S.; Judy, J. W.;Markovic, D. An FPGA-based platform for accelerated offline spike sorting, Journal of Neuroscience Methods 2013, 215, 1–11.

[11] Rutishauser, U. Online detection and sorting of extracellularly recorded action potentials in human medial temporal lobe recordings, in vivo, Journal of Neuroscience Methods 2006, 154, 204–224.

[12] Mukhopadhyay, S.; Ray, G. C. A new interpretation of nonlinear energy operator and its efficacy in spike detection, IEEE Transactions on Biomedical Engineering 1998, 45, 180–187.

[13] Quiroga, R. Q.; Nadasdy, Z.; Ben-Shaul, Y. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering, Neural Computation 2004, 16, 1661–1687.

[14] Gibson, S.; Judy, J. W.; Markovic, D. Technology-aware algorithm design for neural spike detection, feature extraction, and dimensionality reduction, IEEE Transactions on Neural Systems and Rehabilitation Engineering 2010, 18, 469–478.

[15] Mtetwa, N.; Smith, L. S. Smoothing and thresholding in neuronal spike detection, Neurocomputing 2006, 69, 1366–1370.

[16] Sato, T.; Suzuki, T.; Mabuchi, K. Fast template matching for spike sorting, Electronics and Communications in Japan 2009, 92, 57–63.

[17] Oweiss, K.; Aghagolzadeh, M. Detection and classification of extracellular ***action potential recordings, Chapter 2 of Statistical Signal Processing for Neuroscience, Academic Press, Tokyo., pp.15–74, 2010.

[18] Kim, S.; McNames, J. Automatic spike detection based on adaptive template matching for extracellular neural recordings, Journal of Neuroscience Methods 2007, 165, 165–174.

[19] Hwang, W. J.; Wang, S. H.; Hsu, Y. T. Spike detection based on normalized correlation with automatic template generation, Sensors 2014, 14, 11049–11069.

[20] NIOS II Processor Reference Handbook; Altera Corporation: San Jose, CA, USA, 2015. Available online: http://www.altera.com/literature/lit-nio2.jsp (accessed on 8 April 2015).

[21] Smith, L. S.; Mtetwa, N. A tool for synthesizing spike trains with realistic interference. Journal of Neuroscience Methods 2007, 159, 170–180.

[22] Kim K.; Kim, S. A wavelet-based method for action potential detection from extracellular neural signal recording with low signal-to-noise ratio, IEEE Transactions on Biomedical Engineering 2003, 50, 999–1011.