Efficient FPGA-Based Architecture for Spike Sorting Using Generalized Hebbian Algorithm

Wen-Jyi Hwang, Chi-En Ke, Sheng-Ying Lai, Jung-Gen Wu
Department of Computer Science and
Information Engineering
National Taiwan Normal University
Taipei, Taiwan
Email: whwang@csie.ntnu.edu.tw, gdset@hotmail.com, ddt0218@hotmail.com, jgwu@csie.ntnu.edu.tw

***Abstract*—An efficient VLSI architecture for fast spike sorting is presented in this paper. The architecture is able to perform feature extraction based on the Generalized Hebbian Algorithm (GHA). The employment of GHA allows efficient computation of principal components for subsequent clustering and classification operations. The hardware implementation of GHA features high throughput and high classification success rate. The proposed architecture is implemented by Field Programmable Gate Array (FPGA). It is embedded in a System-On-Programmable-Chip (SOPC) platform for performance measurement. Experimental results show that the proposed architecture is an efficient spike sorting design with high speed computation for spike trains corrupted by large noises.**

Keywords-Spike Sorting; FPGA; Generalized Hebbian Algorithm.

I. INTRODUCTION

Spike sorting [1] is often desired for the design of brain machine interface (BMI) [2]. It receives spike trains from extracellular recording systems. Each spike train obtained from the system is a mixture of the trains from neurons near the recording electrodes. The goal of spike sorting is to segregate the spike trains of individual neurons from this mixture. Spike sorting is a difficult task due to the presence background noise and the interferences among neurons in a local area. A typical spike sorting algorithm involves computationally demanding operations such as feature extraction. One way to carry out these complex tasks is to deliver spike trains to external computers. Because the delivery of raw spike trains requires high bandwidth, wireless transmission may be difficult. Existing spike sorting systems may therefore be wired, restraining patients and test subjects from free movement.

Hardware spike sorting is an effective alternative for BMI applications. It allows the spike sorting to be carried out at the front-end so that data bandwidth can be reduced for wireless communication. A common approach for hardware design [3] is based on Application Specific Integrated Circuits (ASICs). A major drawback of ASICs is the lack of flexibility for changes. With the wide range of spike sorting algorithms that already exist and the continual design and improvement of algorithms, the ability to easily change a spike sorting system for new algorithms is usually desired. However, the modification in ASIC is very difficult, especially when chips are implanted in the brain. In addition, the high Non-Recurring Engineering (NRE) costs and long design and verification efforts for fabricating ASICs can severely limit the applicability of emerging BMI applications. The field programmable gate array (FPGA) is an effective alternative to ASIC for hardware implementation with lower NRE costs. Moreover, the circuits in an FPGA are reconfigurable; thereby providing higher flexibility to a spike sorting architecture for future extensions.

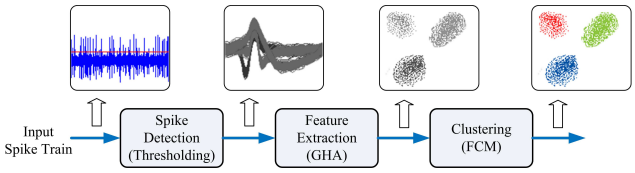


Figure 1: The operations of a typical spike sorting system.

The objective of this paper is to present an effective FPGA-based hardware architecture for spike sorting. The architecture is able to perform online training for feature extraction in hardware. The feature extraction is based on the generalized Hebbian algorithm (GHA) [4]. The proposed architecture is used as a hardware accelerator of a spike sorting system on a System-On-Programmable-Chip (SOPC) platform for performance evaluation. The computation time of spike sorting based on the SOPC is measured and compared with existing works. Experimental results reveal that the proposed architecture is able to perform feature extraction in real time with low hardware resource consumption.

The remaining parts of this paper are organized as follows: Section 2 gives a brief review of the spike sorting operations and the GHA algorithm. Section 3 describes the proposed GHA architecture. Experimental results are included in Section 4. Finally, the concluding remarks are given in Section 5.

II. PRELIMINARIES

A. Spike Sorting Operations

Figure 1 shows the operations of a typical spike sorting system, which consists of spike detection, feature extraction and clustering. The spike detection identifies and aligns spikes from a noisy spike train. A simple spike detection technique

 ISBN: 978-1-61208-319-3

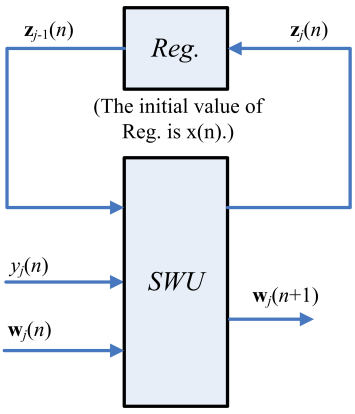


Figure 2: The hardware implementation of (10) and (12).

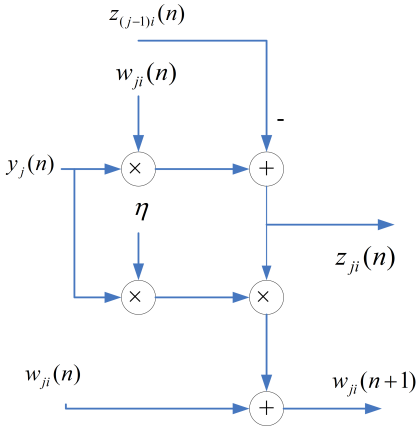


Figure 3: The architecture of each module in the SWU unit.

is to perform thresholding based on the absolute value of spike samples. The feature extraction finds the feature vectors from the detected spikes. The GHA algorithm has been found to be an effective technique for feature extraction. Based on the feature vectors, the final step of the spike detection is to perform clustering and classification using unsupervised clustering methods such as the fuzzy c-means (FCM) algorithm [5]. Detailed discussions of each step can be found in [1].

B. GHA Algorithm

Let

$$\mathbf{x}(n) = [x_1(n), \ldots, x_m(n)]^T, n = 1, ..., t, \quad (1)$$
$$\mathbf{y}(n) = [y_1(n), \ldots, y_p(n)]^T, n = 1, ..., t, \quad (2)$$

be the n-th input and output vectors to the GHA, respectively. In addition, m, p and t are the vector dimension, the number of Principal Components (PCs), and the number of input and output vectors for the GHA, respectively. The output vector $\mathbf{y}(n)$ is related to the input vector $\mathbf{x}(n)$ by

$$y_j(n) = \sum_{i=1}^{m} w_{ji}(n) x_i(n) \quad (3)$$

where the $w_{ji}(n)$ stands for the weight from the i-th synapse to the j-th neuron at iteration n.

Let

$$\mathbf{w}_j(n) = [w_{j1}(n), \ldots, w_{jm}(n)]^T, j = 1, \ldots, p \quad (4)$$

be the j-th synaptic weight vector. Each synaptic weight vector $\mathbf{w}_j(n)$ is adapted by the Hebbian learning rule:

$$w_{ji}(n+1) = w_{ji}(n) + \eta[y_j(n)x_i(n) - y_j(n)\sum_{k=1}^{j} w_{ki}(n)y_k(n)] \quad (5)$$

where η denotes the learning rate. Given an input vector $\mathbf{x}(n)$, the GHA algorithm involves the computation of $y_j(n)$ in (3), and $\mathbf{w}_j(n)$ in (5) for $j = 1, ..., p$. After a large number of iterative computation and adaptation, $\mathbf{w}_j(n)$ will asymptotically approach to the eigenvector associated with the j-th eigenvalue λ_j of the covariance matrix of input vectors, where $\lambda_1 > \lambda_2 > \cdots > \lambda_p$. A more detailed discussion of GHA can be found in [4].

C. GHA Algorithm for Spike Sorting

The GHA can be used for feature extraction of spikes. To use GHA for feature extraction, the $\mathbf{x}(n)$ in (2) is the n-th spike in the spike train. Therefore, the vector dimension m is the number of samples in a spike. Let

$$\mathbf{w}_j = [w_{j1}, \ldots, w_{jm}]^T, j = 1, \ldots, p \quad (6)$$

be the synaptic weight vectors of the GHA after the training process has completed. Based on $\mathbf{w}_j, j = 1, ..., p$, the GHA feature vector extracted from training vector $\mathbf{x}(n)$ (denoted by $\mathbf{f}_n$) is computed by

$$\mathbf{f}_n = [f_{n,1}, ..., f_{n,p}]^T, \quad (7)$$

where

$$f_{n,j} = \sum_{i=1}^{m} w_{ji} x_i(n) \quad (8)$$

be the j-th element of $\mathbf{f}_n$. The set of feature vectors $F = \{\mathbf{f}_1, ..., \mathbf{f}_t\}$ are then for subsequent classification and clustering.

III. Proposed Architecture

The proposed GHA architecture consists of three functional units: the memory unit, the synaptic weight updating (SWU) unit, and the principal components computing (PCC) unit.

A. SWU Unit of GHA

To reduce the complexity of computing implementation, (5) can be rewritten as

$$w_{ji}(n+1) = w_{ji}(n) + \eta y_j(n)[x_i(n) - \sum_{k=1}^{j} w_{ki}(n)y_k(n)] \quad (9)$$

The design of SWU unit is based on (9). Although the direct implementation of (9) is possible, it will consume large hardware resources. One way to reduce the resource consumption is by observing that (9) can be rewritten as

$$w_{ji}(n+1) = w_{ji}(n) + \eta y_j(n) z_{ji}(n), \quad (10)$$

where

$$z_{ji}(n) = x_i(n) - \sum_{k=1}^{j} w_{ki}(n)y_k(n), j = 1, \ldots, p. \quad (11)$$

 ISBN: 978-1-61208-319-3

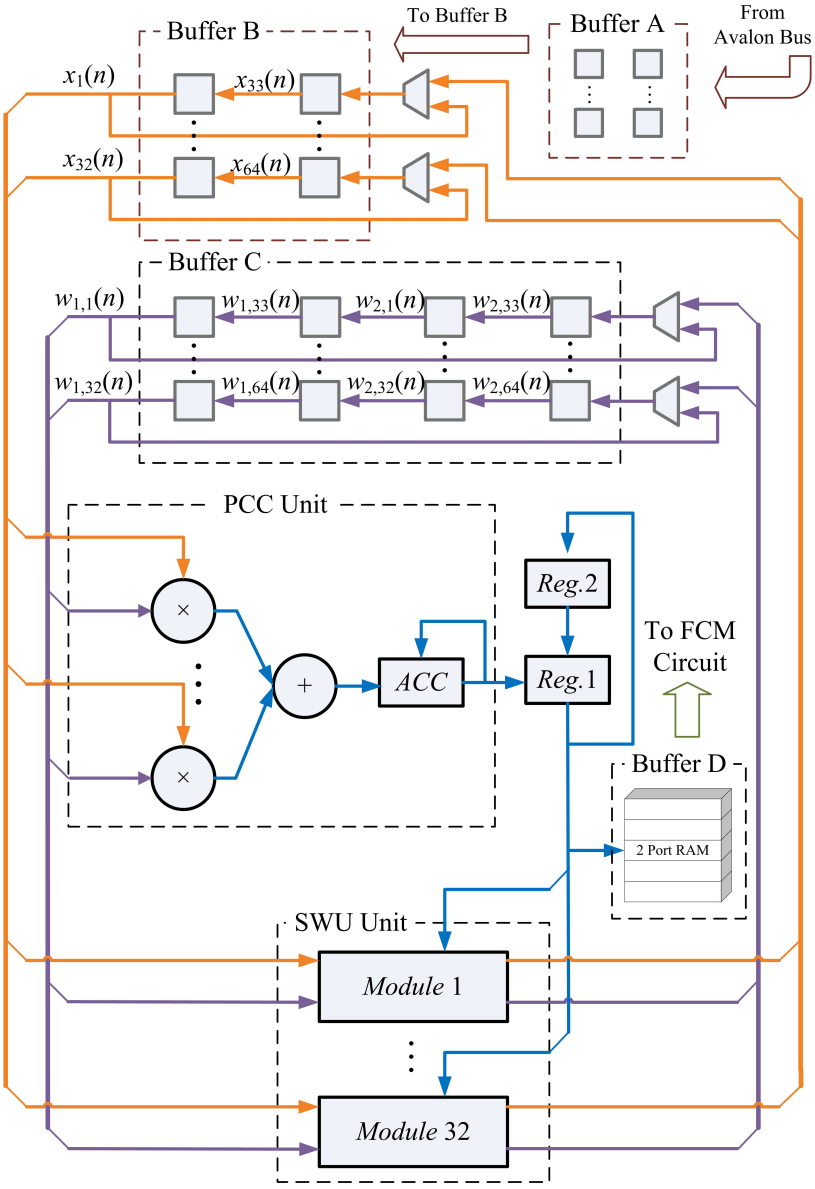


Figure 4: The GHA circuit for $m = 64$, $p = 2$, $b = 2$ and $q = 32$

and $\mathbf{z}_j(n) = [z_{j1}(n), \ldots, z_{jm}(n)]^T$. The $z_{ji}(n)$ can be obtained from $z_{(j-1)i}(n)$ by

$$z_{ji}(n) = z_{(j-1)i}(n) - w_{ji}(n)y_j(n), j = 2, \ldots, p \qquad (12)$$

When $j = 1$, from (11) and (12), it follows that

$$z_{0i}(n) = x_i(n) \qquad (13)$$

Therefore, the hardware implementation of (10) and (12) is equivalent to that of (9). Figure 2 depicts the hardware implementation of (10) and (12). As shown in the figure, the SWU unit produces one synaptic weight vector at a time. The computation of $\mathbf{w}_j(n+1)$, the j-th weight vector at the iteration $n+1$, requires the $\mathbf{z}_{j-1}(n)$, $\mathbf{y}(n)$ and $\mathbf{w}_j(n)$ as inputs. In addition to $\mathbf{w}_j(n+1)$, the SWU unit also produces $\mathbf{z}_j(n)$, which will then be used for the computation of $\mathbf{w}_{j+1}(n+1)$. Hardware resource consumption can then be effectively reduced.

One way to implement the SWU unit is to produce $\mathbf{w}_j(n+1)$ and $\mathbf{z}_j(n)$ in one shot. However, m identical modules, individually shown in Figure 3, may be required because the dimension of vectors is m. The area costs of the SWU unit then grow linearly with m. To further reduce the area costs, each of the output vectors $\mathbf{w}_j(n+1)$ and $\mathbf{z}_j(n)$ is separated into b blocks, where each block contains q elements. The SWU unit only computes one block of $\mathbf{w}_j(n+1)$ and $\mathbf{z}_j(n)$ at a time. Therefore, it will take b clock cycles to produce complete $\mathbf{w}_j(n+1)$ and $\mathbf{z}_j(n)$.

B. PCC unit of GHA

The PCC operations are based on (3). Therefore, the PCC unit of the proposed architecture contains adders and multipliers. Because the number of multipliers grows with the vector dimension m, the direct implementation using (3) may consume large hardware resources when m becomes large.

Similar to the SWU unit, the block based computation is used for reducing the area costs. In fact, (3) can be rewritten as

$$y_j(n) = \sum_{k=1}^{b} \sum_{i=1}^{q} w_{j,(k-1)q+i}(n) x_{(k-1)q+i}(n). \qquad (14)$$

The implementation of (14) needs only q multipliers, a q-input adder, and an accumulator.

C. Memory Unit of GHA

The memory unit contains four buffers: Buffers A, B, C and D. Buffer A fetches and stores spike $\mathbf{x}(n)$ from the main memory. Buffer B contains $\mathbf{z}_j(n)$ for the computation in PCC and SWU units. Buffer C consists of the synaptic weight vectors $\mathbf{w}_j(n)$. The feature vectors $\mathbf{f}_1, ..., \mathbf{f}_t$ are stored in Buffer D. The Buffers A, B and C are shift registers. Buffer D is a two-port RAM for the subsequent access by the FCM unit.

D. Operations of the GHA unit

In typical spike sorting implementations [8], a spike may contain 64 samples. In addition, two PCs may suffice for feature extraction [1]. Therefore, without loss of generality, the GHA unit for $m = 64$ (i.e., vector dimension is 64) and $p = 2$ (i.e., number of PCs is 2) is considered in this subsection. In the GHA unit, each vector is separated into 2 blocks. Moreover, the dimension of each block is 32. Therefore, we set $b = 2$ and $q = 32$ for the circuit implementation. Figure 4 shows the resulting GHA circuit for $m = 64$, $p = 2$, $b = 2$ and $q = 32$. The operations of the GHA circuit can be separated into 4 states, as revealed in Figure 5. The most important operations of the GHA circuit are the PCC operations in State 3 and SWU operations in State 4. These two operations are further elaborated below.

Assume the input vector $\mathbf{x}(n)$ is available in Buffer B. In addition, the *current* synaptic weight vectors $\mathbf{w}_1(n), \mathbf{w}_2(n)$ are stored in the Buffer C. Based on $\mathbf{x}(n)$ and $\mathbf{w}_1(n), \mathbf{w}_2(n)$ the PCC unit produces output vector $y_1(n), y_2(n)$. The computation of $y_j(n)$ is separated into two steps. The first step finds $\sum_{i=1}^{32} w_{j,i}(n)x_i(n)$. The second step computes $\sum_{i=33}^{64} w_{j,i}(n)x_i(n)$, and then accumulate the result with that of the previous step to find $y_j(n)$. These two steps share the same circuit in the PCC unit.

Upon the completion of PCC operations, the SWU unit will be activated in State 4. Using $\mathbf{x}(n)$, $y_j(n)$ and $\mathbf{w}_j(n), j = 1, 2$, the SWU unit computes the new synaptic weight vectors $\mathbf{w}_j(n+1), j = 1, 2$, which are then stored back to Buffer C for subsequent training. Similar to the computation of $y_j(n)$, the computation of $\mathbf{w}_j(n+1)$ consists of two steps. The first step computes the first half of $\mathbf{w}_j(n+1)$ (i.e., $w_{j,1}(n+1)$,...,$w_{j,32}(n+1)$). The second step calculates the second half. These two steps also share the same circuit in the SWU unit. Moreover, the computation of $\mathbf{w}_1(n+1)$ also produces $\mathbf{z}_1(n)$, which is stored back to Buffer B.

After the training process of GHA circuit is completed, the Buffer C contains the synaptic weight vectors $\mathbf{w}_1$ and $\mathbf{w}_2$. Based on the synaptic weight vectors, the PCC unit can be used for feature extraction operations. The computation results are stored in Buffer D as feature vectors $\mathbf{f}_n, n = 1, ..., t$, for the subsequent FCM clustering operations.

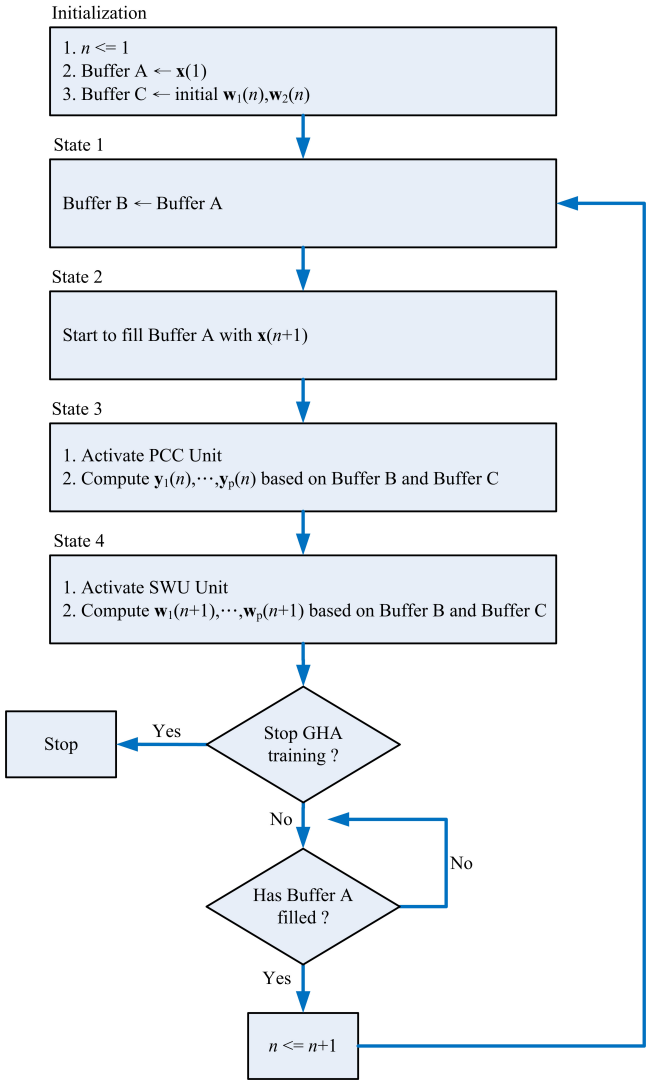


Figure 5: The training operations of the GHA circuit.

E. NOC-based GHA System

The proposed architecture is used as a custom user logic in an NOC system consisting of softcore NIOS CPU, DMA controller and on-chip RAM. An NOC is a new platform for implementing advanced SOCs in an interconnection network strategy, which solves the problems of traditional bus architecture, such as communication efficiency, latency and single clock synchronization. In a typical spike sorting system, delivery of spike signals, feature vectors, and classification results are required. The NOC therefore is beneficial for enhancing transmission speed and throughput of the proposed architecture.

In the NOC system, all the detected spikes are stored in the on-chip RAM and then transported to the proposed GHA circuit for feature extraction. The DMA-based training data delivery is performed so that the memory access overhead can be minimized. The softcore NIOS CPU coordinates different components in the NOC. It is responsible for circuit activation and control. The results of feature extraction are stored in the memory unit of the GHA circuit for subsequent clustering

 ISBN: 978-1-61208-319-3

TABLE I: CCRs OF THE PROPOSED ARCHITECTURE FOR THE SPIKE SORTING WITH DIFFERENT SNR LEVELS.

		SNR (dB)					
		1	2	4	6	8	10
$c = 2$	t	1651	1638	1621	1656	1662	1653
	$\bar{t}$	1644	1632	1617	1654	1660	1651
	CCR	99.58%	99.63%	99.75%	99.88%	99.88%	99.88%
$c = 3$	t	1850	1860	1842	1870	1873	1828
	$\bar{t}$	1571	1672	1737	1791	1812	1769
	CCR	84.92%	89.89%	94.30%	95.78%	96.74%	96.77%

operations.

IV. EXPERIMENTAL RESULTS

In order to evaluate the performance of the proposed architecture for spike sorting, the simulator developed in [8] is adopted to generate extracellular recordings. The simulation gives access to ground truth about spiking activity in the recording and thereby facilitates a quantitative assessment of architecture performance since the features of the spike trains are known a priori. Various sets of spikes with different signal-to-noise (SNR) ratios and interference levels have been created by the simulator for our experiments. All the spikes are recorded with sampling rate 24000 samples/sec. The length of each spike is 2.67 ms. Therefore, each spike has 64 samples. The dimension of vectors for GHA training therefore is $m = 64$. The number of PCs is $p = 2$ for the circuit design.

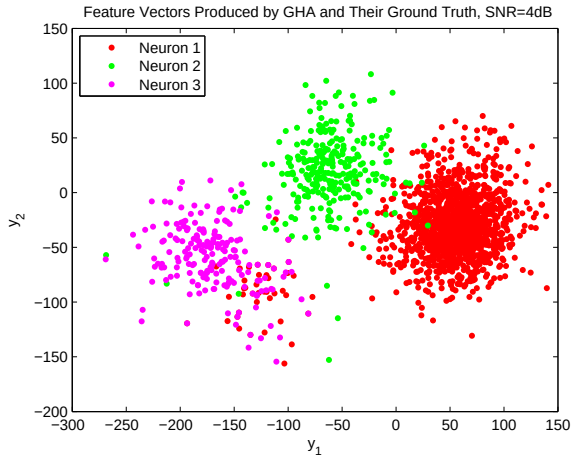
We first consider the classification correct rate (CCR) of the proposed architecture. The CCR for spike sorting is defined as the number of spikes which are correctly classified by the total number of spikes. To show the robustness of the proposed architecture against noise interference, various SNR ratios are considered, ranging from SNR=1 dB to 10 dB. Table I shows the resulting CCRs for the spike trains with two target neurons ($c = 2$) and three target neurons ($c = 3$). The duration of the spike trains is 14 seconds. The spikes extracted from the spike trains are used for the GHA and FCM training, as well as spike classification. The total number of spikes used for training and classification (t), and the number of spikes which are correctly classified ($\bar{t}$) are also included in the table. Because the performance of FCM training may be dependent on the selection of initial vectors, each CCR value in the table is the average CCR values of 40 independent executions. From the table, it can be observed that the proposed architecture is able to attains CCR above 84 % for $c = 3$ when SNR=1 dB.

Table II compares the CCRs of the GHA- and PCA- based spike sorting algorithms. They all use the FCM method for clustering. It can be observed from the table that the GHA algorithm has slightly higher CCRs over the PCA for various SNR levels. The hardware implementation of the PCA may be difficult because it requires the covariance matrix of input data. Therefore, the GHA algorithm is well suited for spike sorting due to its simplicity for hardware design, and its high CCRs.

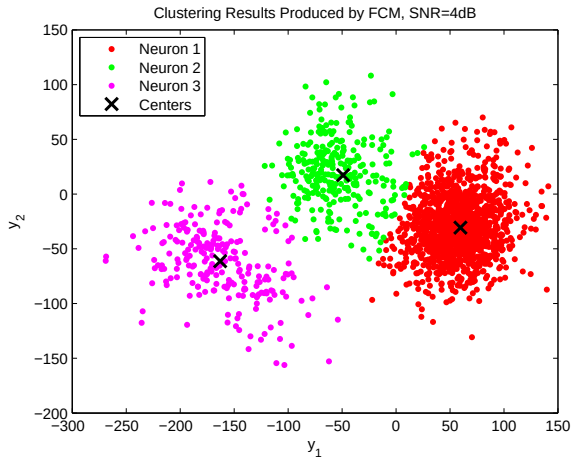
To further elaborate the effectiveness of the GHA and FCM algorithms for spike sorting, Figure 6 shows the distribution of GHA feature vectors of spikes for SNR=4, and the results of

TABLE II: CCR VALUES OF VARIOUS SPIKE SORTING ALGORITHMS.

	SNR (dB)				
	1	4	6	8	10
GHA	84.92 %	94.30 %	95.78 %	96.74 %	96.77 %
PCA [5]	84.21 %	94.08 %	95.72 %	96.69 %	96.72 %



(a)



(b)

Figure 6: The distribution of GHA feature vectors of spikes, and the results of FCM clustering for SNR=10. (a) Ground truth of neuron spikes, (b) Clustering results produced by FCM.

FCM clustering. The center of each cluster produced by FCM are also marked in the figure. By comparing Figure 6.(a) with Figure 6.(b), we see that the proposed GHA and FCM circuits are able to correctly separate spikes even for large noises.

The proposed NOC-based spike sorting system features fast computation. Table III shows the training time of the proposed GHA system for various clock rates. The design platform for the experiments is Altera Quartus II with Qsys [9]. The target FPGA device is Altera Cyclone IV EP4CGX150DF31. The training time of its software counterpart running on Intel I7 processor is also included in the table for the comparison purpose. The training set for these experiments consists of 800 spikes. The number of epoches for GHA training is 100. It can

TABLE III: THE TRAINING TIME OF THE PROPOSED GHA CIRCUIT FOR VARIOUS CLOCK RATES.

Implementation	NOC-based GHA						Software GHA
Processor	Altera NIOS II						Intel I7
Clock Rate	50 MHz	200 MHz	400 MHz	600 MHz	800 MHz	1 GHz	2.61 GHz
GHA (ms)	35.60	8.92	4.46	2.97	2.23	1.78	181.38

TABLE IV: COMPARISONS OF THE PROPOSED GHA CIRCUIT WITH OTHER FPGA-BASED FEATURE EXTRACTION IMPLEMENTATIONS.

Arch.	FPGA Devices	Logic Cells or LEs	DSP elements or Multipliers	Embedded Bits	Max. Clock Rate	Throughput
Proposed GHA Arch.	Altera Cyclone IV EP4CGX150 Xilinx	15688	128	63488	1G Hz	4.50×10^7
GHA Arch. in [6]	Virtex 6 XC6VSX315T Xilinx	12610	12	0	100M Hz	1.60×10^6
GHA Arch. in [7]	Cyclone IV EP4CGX150	9144	432	63448	50M Hz	2.75×10^6

be observed from Table III that the propose NOC-based spike sorting system is able to operate up to 1 GHz clock rate. In addition, the GHA training time decreases linearly with the clock rate. When clock rate becomes 1 GHz, the total training time of the proposed NOC-based spike sorting system is only 1.99 ms. By contrast, the training time of the Intel I7 processor is 193.18 ms. The speedup of the proposed hardware system over its software counterpart is therefore 97.08.

In Table IV, we compare the area costs and throughput of the proposed GHA circuit with those of other FPGA-based hardware implementations [6], [7] for feature extraction. The throughput is defined as the number of input training vectors the circuit can process per second. It can be observed from Table IV that the proposed GHA architecture attains highest clock rate and throughput at the expense of higher area costs. In fact, the proposed architecture has throughput 28.125 (4.50×10^7 vs. 1.60×10^6) and 16.3 times (4.50×10^7 vs. 2.75×10^6) higher than that of architectures in [6] and [7], respectively. The proposed algorithm has superior performance because it is based on shift registers for storing weight vectors and input vectors for high speed computation. In addition, although the proposed architecture has higher area costs, it only consumes a small fraction of the hardware resources available in the target FPGA. In fact, there are 149760 LEs, 6635520 embedded memory bits, and 360 multipliers in the target FPGA Cyclone IV EP4CGX150DF31. The proposed architecture utilizes only 10.78 %, 0.96 % and 35.56 % of the LEs, memory bits and multipliers of the target FPGA, respectively. All these facts show the effectiveness of the proposed architecture.

V. CONCLUSION

The proposed architecture has been implemented on the Altera FPGA Cyclone IV for physical performance measurement. The architecture is used as an hardware accelerator to the NIOS CPU in a NOC platform. Experimental results reveal that the proposed spike sorting architecture has advantages of high CCR and high computation speed. For SNR=10, its CCR is above 96 % for three target neurons. When SNR becomes 1 dB, it is still able to retain CCR above 84 %. The architecture is able to achieve 1 GHz clock rate. The speedup over its software counterpart running on Intel I7 processor is above 97. The GHA circuit has higher computation speed as compared with existing hardware implementations for GHA feature extraction. These results show that the proposed system implemented by FPGA is an effective realtime training device for spike sorting.

REFERENCES

[1] M. S. Lewicki,"A review of methods for spike sorting: the detection and classification of neural action potentials," Network Computer Neural System, vol. 9, 1998, pp. R53-R78.

[2] M. A. Lebedev and M. A. L. Nicolelis, "Brainmachine interfaces: past, present and future," Trends in Neurosciences, vol.29, 2006, pp. 536-546.

[3] T. Chen, K. Chen, Z. Yang, K. Cockerham, and W. Liu, "A biomedical multiprocessor SoC for close-loop neuroprosthetic application," in International Solid-State Circuits Conference, 2009, pp. 434-435.

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

[5] A. Oliynyk, C. Bonifazzi1, F. Montani, and L. Fadiga1, "Automatic online spike sorting with singular value decomposition and fuzzy C-mean clustering," BMC Neural Science, vol. 13, 2012, pp. 1-9.

[6] B. Yu, T. Mak, X. Li, F. Xia, A. Yakovlev, Y. Sun, and C. S. Poon,"A Reconfigurable Hebbian Eigenfilter for Neurophysiological Spike Train Analysis," Proc. International Conference on Field Programmable Logic and Applications, 2010, pp. 556-561.

[7] W. J. Hwang and H. Chen, Efficient VLSI Architecture for Spike Sorting Based on Generalized Hebbian Algorithm, Proc. European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2013, pp. 71-76.

[8] L. S. Smith and N. Mtetwa, "A tool for synthesizing spike trains with realistic interference," Journal of Neuroscience Methods, vol. 159, 2007, pp. 170-180.

[9] Altera Corporation, Quartus II Handbook Version 13.0. 2013. Available online: http://www.altera.com/literature/lit-qts.jsp (accessed on 26 June 2013).

 ISBN: 978-1-61208-319-3