

# Real-Time FPGA-Based Multi-Channel Spike Sorting Using Hebbian Eigenfilters

Bo Yu, Terrence Mak, *Member, IEEE*, Xiangyu Li, Fei Xia, Alex Yakovlev, *Senior Member, IEEE*, Yihe Sun, *Member, IEEE*, and Chi-Sang Poon, *Fellow, IEEE*

**Abstract**—Real-time multi-channel neuronal signal recording has spawned broad applications in neuro-prostheses and neuro-rehabilitation. Detecting and discriminating neuronal spikes from multiple spike trains in real-time require significant computational efforts and present major challenges for hardware design in terms of hardware area and power consumption. This paper presents a Hebbian eigenfilter spike sorting algorithm, in which principal components analysis (PCA) is conducted through Hebbian learning. The eigenfilter eliminates the need of computationally expensive covariance analysis and eigenvalue decomposition in traditional PCA algorithms and, most importantly, is amenable to low cost hardware implementation. Scalable and efficient hardware architectures for real-time multi-channel spike sorting are also presented. In addition, folding techniques for hardware sharing are proposed for better utilization of computing resources among multiple channels. The throughput, accuracy and power consumption of our Hebbian eigenfilter are thoroughly evaluated through synthetic and real spike trains. The proposed Hebbian eigenfilter technique enables real-time multi-channel spike sorting, and leads the way towards the next generation of motor and cognitive neuro-prosthetic devices.

**Index Terms**—Brain-machine interface, Hebbian learning, spike sorting, FPGAs, hardware architecture design.

## I. INTRODUCTION

Recently, multi-electrode arrays (MEAs) have become increasingly popular for neuro-physiological experiments *in vivo* [1][2][3][4] or *in vitro* [5][6][7][8]. Compared with other methods of signal acquisition, such as functional magnetic resonance imaging (fMRI) [9], Electroencephalography (EEG) [10] and Electrocorticographical (ECoG) [11], MEA provides the capability of recording neuronal spikes from specific regions of the brain with high signal-to-noise ratio [1][5]. The substantial temporal and spatial resolutions provided by MEAs facilitate the studies of neural network dynamic [12], plasticity [13], learning and information processing [14] and the developments of high performance brain-machine interface (BMI) for emerging applications, such as motor rehabilitation for paralyzed or stroke patients [15][16][17].

Bo Yu, Xiangyu Li, Yihe Sun are with Tsinghua National Laboratory for Information Science and Technology, Institute of Microelectronics, Tsinghua University Beijing 100084, China. e-mail: (yu-b06@mails.tsinghua.edu.cn; xiangyuli, sunyh@mail.tsinghua.edu.cn).

Terrence Mak, Fei Xia, Alex Yakovlev are with the School of Electrical, Electronic and Computer Engineering, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK. Terrence Mak is also with the Institute of Neuroscience, Newcastle Biomedicine, at the same University. e-mail: (terrence.mak, fei.xia, alex.yakovlev@ncl.ac.uk).

Chi-Sang Poon is with Harvard-MIT Division of Health Sciences and Technology, MIT, Cambridge, MA 02139, USA. e-mail: (cpoon@mit.edu).

Neuronal spike trains recorded by electrodes encompass noises introduced by measurement instruments and/or interferences from other vicinity neurons. Neural signal processing that extracts useful information from noisy spike trains is necessary for spike information decoding and neural network analysis in subsequent processes. In most neural signal processing flows, especially in the MEA based brain-machine interface (BMI) systems, spike sorting that discriminates neuronal spikes to corresponding neurons is among the very first steps of signal filtering [18][19][20] and its correctness significantly affects the reliability of the subsequent analysis [21].

Real-time spike sorting requires substantial computational capability to process continuous and high-bandwidth recordings from subjects and support implementation of feedbacks, such as neural stimulation, when necessary. Hardware systems providing dedicated circuits for specific computations can substantially outperform the corresponding computations using software in terms of computational performance and power dissipation. It also presents the unique advantages of portability and bio-implantability for different experimental needs. Hardware solutions are therefore necessary for neurophysiological signal recordings and analysis where these factors are crucial.

Principle component analysis (PCA) provides an effective solution for neuronal spike discrimination because of its capability of automatic feature extraction [19]. Implementation of PCA-based hardware systems for spike feature extraction has been reported in [22]. This system employs an algebra based PCA, in which computations for covariance matrix and eigenvalue decomposition are involved, and results in significant computational complexity. Direct realization of these numerical methods requires substantial hardware costs, such as power consumption and hardware area. In addition, most algebraic approaches compute all principal components whereas only the first few leading components are required for spike discrimination. Besides, the number of recording channels in a single MEA is rapidly increasing. New recording systems with thousands of channels, e.g. the MEA systems with 4096 channels [5], require substantial computational bandwidth to process the real-time recordings. Novel methodology and hardware architecture design are therefore vital to sustain competent performance for the rapidly increasing recording bandwidth.

Field programmable gate arrays (FPGAs) provide massively parallel computing resources and are suitable for real-time and high-performance applications. The reconfigurable abil-

ities of FPGAs provide substantial flexibilities for real-time multichannel recording systems, in which parameters, such as dimension of neuronal spikes and data word length, need to be tunable and adaptable for various circumstances. Compared to application specific integrated circuits (ASIC), system-on-chip (SoC) and programmable processor implementations, FPGAs provide high performance, good scalability and flexibility at the same time [23]. These are crucial criteria for multi-channel neuronal signal recording systems. As a result, FPGAs have been employed by a number of neuro-engineering research groups [5][7].

In this paper, we present a Hebbian eigenfilter approach based on general Hebbian algorithm (GHA) to approximate the leading principal components (PCs) of spikes. We also present a high-gain approach to speed up the convergence of the proposed eigenfilter. Further, we show that the eigenfilter can be effectively mapped to a parallel reconfigurable architecture to achieve high-throughput computation. The major contributions of this paper are:

- We propose a general Hebbian eigenfilter approach to approximate the principal component analysis. The proposed method provides an approximation to compute a selected number of eigenvectors. Also, high-gain strategies to speed up the Hebbian network convergence are discussed.
- An FPGA-based architecture, which exploits the intrinsic task-independence in the eigenfilter, is presented and has been integrated into a complete spike-sorting engine to provide real-time and high-throughput spike train analysis. To our knowledge, this is the first FPGA-based Hebbian eigenfilter used for spike sorting realization that can be readily employed in BMI or multi-channel recording systems.
- Both the Hebbian eigenfilter algorithm and the FPGA hardware implementation are rigorously evaluated. The spike sorting accuracy is evaluated based on synthetic spike trains that simulate the realistic inter-neuronal interferences and noises from the analogue circuit. The relationships between word length and hardware resource consumption, power dissipation and algorithm accuracy are also studied.

This paper consists of four Sections. Section II introduces the background of spike sorting and the proposed Hebbian eigenfilter. Section III describes the architectures of the proposed hardware and the implementation of eigenfilter. Section IV presents the evaluation results and discussion. Section V concludes the paper.

## II. HEBBIAN EIGENFILTER BASED SPIKE SORTING ALGORITHM

### A. Background

Spike sorting is one of the fundamental challenges in neurophysiological experiments. During recording, a micro-electrode always picks up action potentials from more than one neuron because of the uncontrolled environment around the electrodes [19][20]. Failing to distinguish different neuronal

spikes will compromise the performance of the neural prosthetic system [21]. As a result, spike sorting that discriminates detected spiking activities to corresponding neurons becomes crucial.

Typical spike sorting algorithms discriminate neuronal spikes according to intrinsic characteristics, namely features, in the time [24] or frequency domains [25][26]. PCA (time-based) and wavelet transformation (time-frequency-based) are the most widely used automatic feature extraction methods. PCA which has become a benchmark feature extraction method calculates orthogonal bases (principal components) that capture the directions of variation in data to characterize differences in data. Wavelet transformation provides a multi-resolution method to accomplish feature extraction that offers good time resolution at high frequency domain and good low frequency resolution at low frequency domain [27]. It is still controversial which method has better performance in the spike sorting scenario. However, most algorithms for the two methods are computationally intensive and require tremendous hardware resources if implemented on chip. In order to facilitate on-chip implementations, several computationally economic approaches have been proposed, such as discrete derivatives [28], integral transform [29], and zero-cross features [30]. However, the effectiveness of these hardware-friendly algorithms remains to be validated through real-data experiments. Instead of proposing a feature extraction method, in this paper, we present a hardware efficient PCA-based method. A neural network based approach is utilized to automatically learn and filter principal components from data.

Besides feature extraction, spike sorting requires a set of pre and post-processing steps including spike detection, spike alignment and clustering. Spike detection distinguishes neuronal spikes from background noises. A commonly used detection method is to compare absolute voltage of the recorded signal with a threshold that is derived from median value of the raw spike train [26]. However, hardware cost of obtaining the median value is high. The nonlinear energy operator (NEO) based spike detection method provides a hardware efficient alternative and also achieves high detection accuracy by considering both spike amplitude and frequency [31]. In general, detected high dimensional spikes need to be aligned at their maximum point for the following feature extraction. Through feature extraction, neuronal spikes are projected into a lower dimensional feature space that highlights the differences of aligned spikes. After feature extraction, clustering algorithms are always employed to automatically identify and differentiate clusters in the feature space. Fig. 1 exemplifies the clustering in the feature space, which consists of the first two principal components. In the feature space, each cluster represents a prospective neuron, and dots represent aligned neuronal spikes that are assigned to their closest cluster (centroid). Although investigation of spike detection, alignment and clustering are not the core of this paper, these pre and post-processing steps are important to ensure high quality spike sorting results and algorithms including NEO based detection and  $K$ -means clustering [32], have been incorporated in our software and hardware experiments.

Table I presents a list of parameters that will be used in this

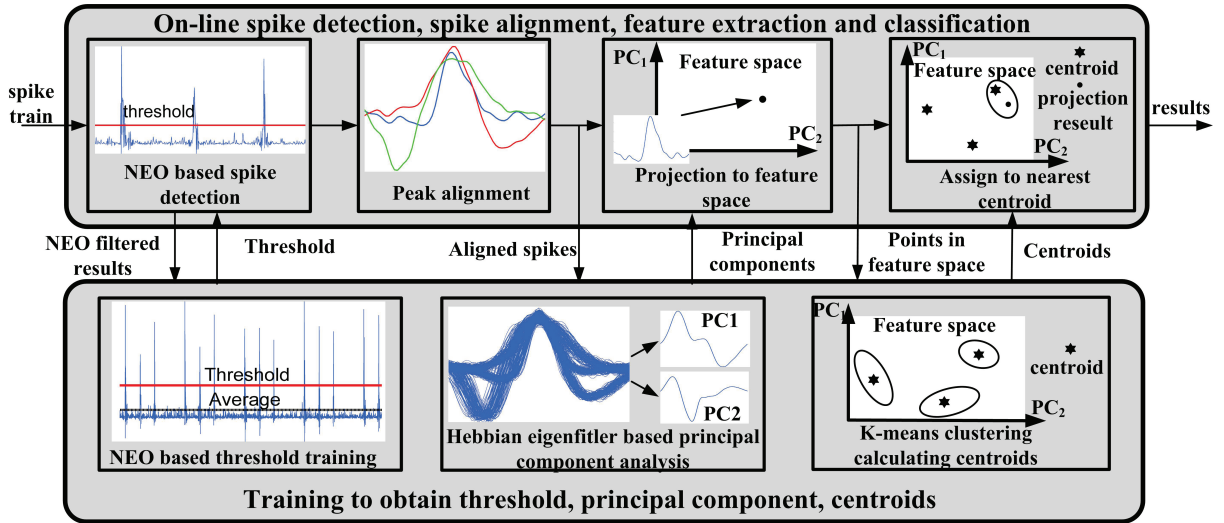


Fig. 1. Hebbian eigenfilter based spike sorting algorithm.

TABLE I  
PARAMETERS DEFINITION

Parameters	Definitions
$d$	Number of samples per aligned spike
$l$	Number of principal components used for spike sorting
$L$	Number of learning iterations
$n$	Total number of spikes
$M$	Total number of channels
$\alpha$	Number of parallel training modules
$\beta$	Number of parallel on-line modules
$N$	Folding ratio ( $\frac{M}{\beta}$ )
$T$	Training period
$r$	Number of registers required for a single channel
$a$	Number of arithmetic units required for a single channel
$S$	Sampling rate for each channel
$\Gamma$	Latency of an on-line processor
$p$	Number of clock cycles required for performing real-time spike sorting per channel
$P_i$	Total number of clock cycles required for performing $i$ -channel real-time spike sorting
$Q$	Number of estimated clusters

paper.

## B. Hebbian Eigenfilter for Spike Sorting

1) *Hebbian Spike Sorting Algorithm*: The overview of the Hebbian eigenfilter based spike sorting is shown in Fig. 1. The proposed method is composed of training and on-line processing parts. In the training module, a Hebbian eigenfilter is presented to extract principal components from the aligned spikes. This module also consists of individual algorithms for spike detection (NEO based method) and clustering ( $K$ -means clustering) for threshold and centroids estimations, respectively. The on-line algorithm detects neuronal spikes according to the estimated threshold and aligns excerpted spikes at the peak point. It then projects the aligned spikes to feature space based on the principal components, and then assigns the projected spikes to the nearest centroid.

Traditional algebra based PCA algorithms are computationally expensive because of involving complex matrix opera-

tions, such as covariance matrix computation and eigenvalue decomposition of covariance matrix [19][33][34]. Particularly, these algorithms compute a full spectrum of the eigenvectors as well as eigenvalues whereas having the first few most important eigenvectors would be sufficient for effective spike sorting.

Hebbian learning was originally inspired by the observation of the pre-synaptic and post-synaptic dynamics and synaptic weight updating [35]. It was later generalized into a number of different forms of algorithms for unsupervised learning [36], classification [37] and auto-association [38]. Particularly, the general Hebbian algorithm (GHA) [36] presents an efficient approach for realizing principal component analysis. The synaptic weights of the feed forward neural network evolve into the principal components of input data if a specific Oja's weight updating rule is followed [36].

We incorporate the Hebbian eigenfilter into the spike sorting framework and, thus, greatly reduce the computational complexity of the principal components computations.

Suppose we have  $n$  aligned spikes,  $\vec{x}(i), i \in [1, n]$ . Each aligned spike is  $d$  dimension (containing  $d$  sample points), *i.e.*  $\vec{x}(i) = [x_1(i), x_2(i), \dots, x_d(i)]^T$ . We let  $l$  be the dimension of feature space (the number of extracted principal components),  $\eta$  be the learning rate,  $\vec{W}(j) = [\vec{W}_1(j), \vec{W}_2(j), \dots, \vec{W}_l(j)]^T$  be a  $l \times d$  synaptic weight matrix that is initialized to  $\vec{W}(1)$ , and  $j$  be the iteration index.

The Hebbian spike sorting algorithm for calculating the first  $l$  principal components of  $n$  aligned neuronal spikes is summarized in Table II. Step 1 is the initialization process. Then, the mean vector of  $n$  aligned spikes,  $\vec{\mu}$ , is calculated in Step 2. After the mean vector is available, the mean vector is subtracted from all aligned spikes in Step 3. After that, iteration learning will be performed on zero-mean spikes. In the algorithm,  $LT[\vec{m}]$  is an operator that sets all the elements above the diagonal of matrix,  $\vec{m}$ , to zeros.  $\vec{W}(j)$  converges to the  $l$  most significant principal components of input data when  $j$  is large enough and the learning rate is appropriate.

Hebbian eigenfilter presents a simple and efficient mecha-

TABLE II  
ALGORITHM OF HEBBIAN EIGENFILTER

1. Initialize synaptic weight $\vec{W}(1)$ and learning rate $\eta$ .
2. Calculate the mean vector of the aligned spikes $\vec{\mu} = \sum_{i=1}^n \vec{x}(i)/n$
3. Zero-mean transformation $\vec{x}(i) = \vec{x}(i) - \vec{\mu} \quad 1 \leq i \leq n$
4. Perform Hebbian learning on zero-mean data $\vec{y}(j) = \vec{W}(j)\vec{x}(i)$ $\vec{L}T(j) = LT[\vec{y}(j)\vec{y}^T(j)]$ $d\vec{W}(j) = \eta(\vec{y}(j)\vec{x}^T(i) - \vec{L}T(j)\vec{W}(j))$ $\vec{W}(j+1) = \vec{W}(j) + d\vec{W}(j)$
5. If the network converges, the algorithm stops, otherwise $j = j + 1, i = i + 1, \quad 1 \leq i \leq n$ then go to step 4.

TABLE III  
COMPARISON BETWEEN HEBBIAN AND EVD BASED PCA

	Hebbian eigenfilter	EVD based PCA
Involving covariance matrix	No	Yes
Accuracy	Approximate	Accurate
Filtering leading eigenvectors	Yes	No
Computing eigenvalue	No	Yes
Computational complexity	$O(dL + dn)$	$O(d^2n + d^3)$

nism to compute principal components. It can be well adopted into the spike sorting routine. Comparing to conventional PCA algorithms, Hebbian eigenfilter does not involve covariance matrix computation and eigenvalue decomposition. It also has the capability to filter specified number of leading principal components. These advantages result in significant savings in terms of computational efforts.

2) *Complexity of Hebbian Eigenfilter*: In this section, we analyze the computational complexity of Hebbian eigenfilter in comparison with the algebra-based method.

Computational complexity of Hebbian eigenfilter is dominated by the number of iterations,  $L$ , dimension of spikes,  $d$ , and the number of spikes,  $n$ . The computational complexity for the Hebbian eigenfilter is  $O(dL + dn)$  (see Appendix A for the derivation). In contrast, the algebra-based PCA involves computation of covariance matrix and eigenvalue decomposition (EVD). Computational complexity for calculating a  $d \times d$  covariance matrix can be characterized as  $O(nd^2)$  (see Appendix B for the derivation). Eigen-decomposition of all the eigenvectors of a symmetric matrix ( $d \times d$ ) requires a complexity of  $O(d^3)$  [33]. As a result, the total computational complexity of the eigen-decomposition based PCA becomes  $O(nd^2 + d^3)$ . In general, number of spikes,  $n$  and data dimension,  $d$ , are large numbers and number of learning epochs,  $L$  is much smaller than  $nd$ . The computational delay of Hebbian eigenfilter will be significantly smaller than the eigen-decomposition based algorithms. This provides a critical advantage to real-time spike sorting using eigenfilter. The major characteristics of Hebbian eigenfilter and eigen-decomposition based algorithms are summarized in Table III.

3) *High-Gain Methods for Hebbian Spike Sorting*: An appropriate learning rate is important to the convergence of the Hebbian network. If the learning rate is too small, the network may need a large number of iterations to converge or may converge to a local solution due to the lack of momentum. If the learning rate is too large, the network may oscillate or

even become unstable causing numerical problems due to that the momentum is too large to be attracted.

Two high-gain methods for accelerating convergence of the eigenfilter are proposed in this section. One approach is to use non-uniform learning rates for each neuron. Because the most important principal component has the strongest attraction while the others are substantially weaker in the Hebbian network, assigning non-uniform learning rates to neurons, especially larger values (momentum) to weaker neurons can help the network converge more synchronously and quickly. The updating equation of weights becomes,

$$d\vec{W}(j) = \vec{\eta}(\vec{y}(j)\vec{x}^T(i) - \vec{L}T(j)\vec{W}(j)) \quad (1)$$

where  $\vec{\eta}$  is  $l \times l$  diagonal matrix,  $diag(\eta_1, \eta_2, \dots, \eta_l)$ ,  $\eta_1 < \eta_2 < \dots < \eta_l$ .

Another high gain approach is to apply ‘‘cooling-off’’ annealing strategies on iteration-varying learning rates to achieve precise solutions quickly. The basic idea of an annealing strategy is to use large learning rates at the beginning to bypass local minima and approach the final result quickly, and to use gradually smaller learning rates to achieve an accurate global minimum in the subsequent learning steps. The learning rate is then characterized using an annealing function  $\eta(j)$ , where  $j$  is the index of each iteration. The weight updating rule can be expressed as,

$$\begin{aligned} d\vec{W}(j) &= \vec{\eta}(j)(\vec{y}(j)\vec{x}^T(i) - \vec{L}T(j)\vec{W}(j)) \\ \vec{W}(j+1) &= \vec{W}(j) + d\vec{W}(j) \end{aligned} \quad (2)$$

where  $\vec{\eta}(j) = diag(\eta_1(j), \eta_2(j), \dots, \eta_l(j))$  is  $l \times l$  diagonal matrix,  $j$  is the iteration index,  $\eta_i(j)$  is a decreasing function. Because the learning rate can be a large number at beginning, the weight can be amplified significantly. To avoid weights become extremely large, we could normalize the weight at each learning epoch as,

$$\vec{W}(j+1) = \begin{cases} \frac{\vec{W}(j+1)}{\|\vec{W}(j+1)\|}, & \text{if } \|\vec{W}(j+1)\| > \delta \\ \vec{W}(j+1), & \text{otherwise} \end{cases} \quad (4)$$

where  $\|\cdot\|$  is the norm of a vector, and  $\delta$  is a constant threshold.

The annealing strategy could yield a better convergent rate. However, it requires additional computational cost for the normalization and empirical methods are needed for finding an optimized annealing strategy, namely  $\eta_i(j), \forall i, j$ . In contrast, using non-uniform learning rates for neurons is more hardware economical. However, empirical experiments are also required for finding the optimal learning rates. These high gain methods are beneficial to spike sorting algorithms, which demand low latency performance.

### III. HARDWARE ARCHITECTURE

#### A. Overall Hardware Architecture for Multi-channel Spike Sorting

Neural recording channels are independent to each other and present intrinsic data-level independencies that can be fully exploited to maximize the system performance. In a

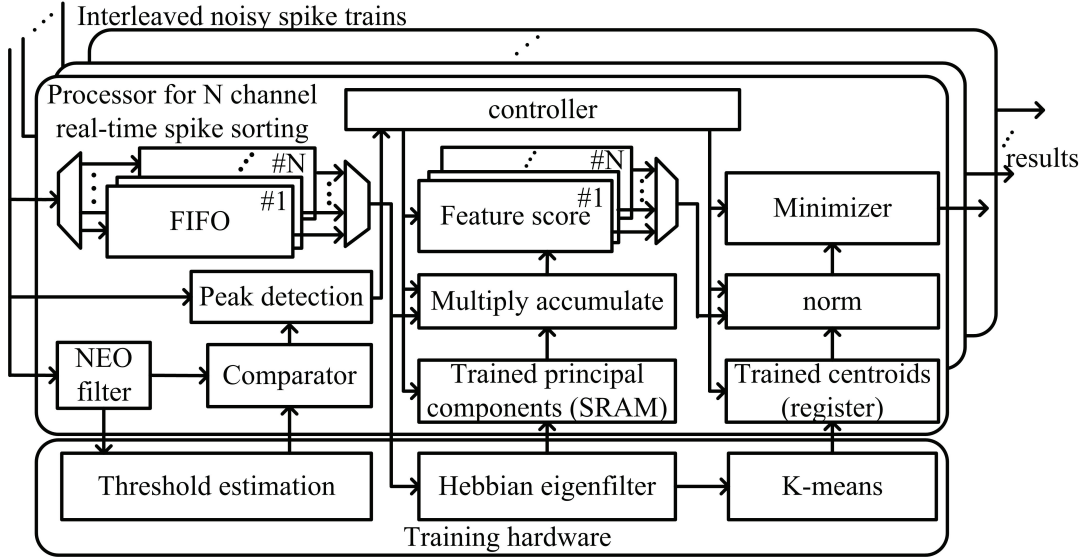


Fig. 2. Architecture of the multi-channel spike sorting hardware.

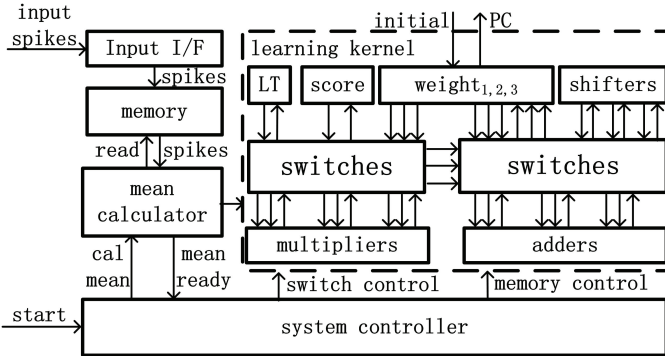


Fig. 3. Architecture of the hardware Hebbian eigenfilter.

fully parallel structure, independent on-line processing and training hardware supporting real-time multi-channel spike sorting are allocated for each channel. However, this fully parallel structure requires large hardware area and is not scalable and practical with the increasing number of channels. Folding and multiplexing techniques need be utilized to share computing resources among channels.

Our proposed system architecture of multi-channel spike sorting hardware is shown in Fig. 2. It consists of two major sub-systems, training and real-time processing modules.

The training module consists of hardware blocks of Hebbian eigenfilter,  $K$ -means and threshold estimation. The threshold estimation block takes the results of NEO filter as inputs and calculates the threshold for real-time detection circuits. The Hebbian eigenfilter operates on the excerpted spikes from real-time detection circuits and outputs principal components for real-time projection hardware. Taking the projection results as inputs, the  $K$ -means hardware calculates  $K$  centroids for the real-time module.

Suppose we have  $\alpha$  training modules and can be shared among channels. Also suppose the training period of each channel is  $T$ , there are  $M$  number of channels, and the latency

of the training hardware is  $t$ . Each training module takes in charge of  $\frac{M}{\alpha}$  channels. The time allocated for training at each of the channel is  $T \frac{\alpha}{M}$ . In order to finish training in such allocated period, the following relationship should be satisfied,  $T \frac{\alpha}{M} > t$ . In other words, the minimum number of training module becomes  $t \frac{M}{T}$ .

The system also consists of a real-time processing module, which is scalable and parameterizable. A number of identical architectures are employed to perform real-time spike sorting in parallel. Folding technique is employed to share computing resources, in which  $N$  channels are processed by a single processing element through multiplexing. The number of channels that are processed at one processor is referred as folding ratio. Therefore, we need  $\lceil \frac{M}{N} \rceil$  parallel real-time processing elements to process spike trains from  $M$  channels.

The real-time module consists of three parts: (i) spike detection and spike alignment, (ii) eigen-space projection and (iii) spike classification. A comparator is used to compare the input signals with the threshold to determine the occurrence of a spike. There are  $N$  FIFOs to buffer recorded samples from  $N$  different channels in each real-time processor. During real-time spike sorting, dot-product is computed between the outputs of FIFOs and principal components, which are pre-computed during training and stored in SRAMs. This projection is realized using the multiply-accumulate block and intermediate results are stored in registers. There are  $N$  registers allocated for  $N$  channels. During real-time spike classification, distance between feature score, which is the results of dot-product, and centroids are calculated. The feature score is assigned to its nearest centroid by the minimizer. This would give a classification of a spike.

### B. Hardware Architecture of Hebbian Eigenfilter

The structure of the Hebbian eigenfilter is presented in Fig. 3. This architecture provides capability of configuration through parameters, such as number of spikes, spike dimension

and number of learning iterations. The architecture consists of four parts, “system controller”, “learning kernel”, “mean calculator” and “interface and memory”. “System controller” controls the entire circuits. “Learning kernel” performs learning operations and consists of arithmetic units, storage units and switchers. “LT” stores the result of  $LT[\vec{y}\vec{y}^T]$ . “Score” stores result of  $\vec{y} = \vec{W}\vec{x}$ . “Weight” stores synaptic weights. Switchers route the signals between arithmetic units and storing elements. Only two adders and one multiplier are used for calculating one weight vector. The memory stores aligned spikes. “Mean calculator” calculates the mean vector of the aligned spikes before the mean vector of align spikes is ready. After the mean vector is obtained, “mean calculator” subtracts the mean vector from each aligned spike and sends the mean subtracted spikes to the learning kernel.

### C. Analysis of Folding Ratio

In a folded architecture, arithmetic units can be shared among channels using time division multiplexing. For each channel, intermediate results of computations should be kept temporarily for subsequent operations. Each channel, therefore, would have independent registers to store the intermediate results. As a result, the hardware consumption of an  $N$ -folded processor becomes  $N \times r + a$ , where  $r$  is the number of registers and  $a$  is the number of arithmetic units required for a signal channel real-time processing unit. In other words, the hardware consumption for each channel is  $r + \frac{a}{N}$ . Therefore, hardware consumption per channel can be reduced by increasing the folding ratio.

We could also evaluate the maximum latency of the system. Let  $S$  be the sampling rate of each channel. In order to provide a real-time processing capability, the data processing throughput should be larger than the data sampling rate. Considering our fully pipelined architecture, the data processing throughput is the product of the clock frequency,  $f_{clk}$ , and number of processing elements,  $\beta$ , and this should satisfy the inequality:

$$f_{clk}\beta \geq MS \quad (5)$$

The inequality can be further simplified by substituting  $M/\beta = N$  as:

$$f_{clk} \geq NS \quad (6)$$

For an  $N$ -folded real-time processing hardware, the latency,  $\Gamma$ , for processing a spike can be expressed as,

$$\Gamma = \frac{P_N}{f_{clk}} \quad (7)$$

where  $P_N$  is the number of clock cycles for processing  $N$  channels. Substituting Eq. 6 into Eq. 7, we can obtain:

$$\Gamma \leq \frac{P_N}{NS} \quad (8)$$

Suppose  $p$  is the number of clock cycle allocated for one channel.  $P_N$  will equal to  $Np$ . From Eq. 8, the maximum latency is expressed as,

$$\Gamma \leq \frac{Np}{NS} = \frac{p}{S} \quad (9)$$

This implies that the latency of a  $N$ -folded system is determined by the sampling rate and number clock cycles to accomplish the analysis for a single spike.

Moreover, dynamic power dissipation is proportion to clock frequency and capacitance of circuits. The dynamic power consumption of an  $N$ -folded architecture follows the following relationship,

$$\text{Power} \propto N \times \left(r + \frac{a}{N}\right) \times f_{clk} \quad (10)$$

Considering both Eq. 6 and 10, the power consumption per channel is proportion to,

$$\begin{aligned} \frac{N \times \left(r + \frac{a}{N}\right) \times f_{clk}}{N} &\geq \left(r + \frac{a}{N}\right) \times NS \\ &= (N \times r + a) \times S \end{aligned} \quad (11)$$

The power consumption per channel increases with the number of folding ratio.

## IV. EVALUATION RESULTS AND DISCUSSION

### A. Test Benchmarks and Experimental Flow

In order to quantitatively evaluate the performance of the Hebbian spike sorting algorithm, spike trains with known spike times and classifications should be employed. Clinical extracellular recording with realistic spike shapes and noise interferences is one option for qualitative studies. However, it is difficult to determine the precise classifications and the source of the spike from clinical measurement and these are the “ground truth” for any effective quantitative evaluation. Although these recordings can be further manually annotated by experts, it has been shown that manually clustered spikes have relative low accuracy and reliability [39]. For these reasons, synthetic spike trains from both [40] and a spike generation tool [41] were utilized to quantify the performance of our algorithm and to compare with other methods. These synthetic spike trains accurately model various background noises and neuronal spikes profile that appear at single-channel clinical recordings.

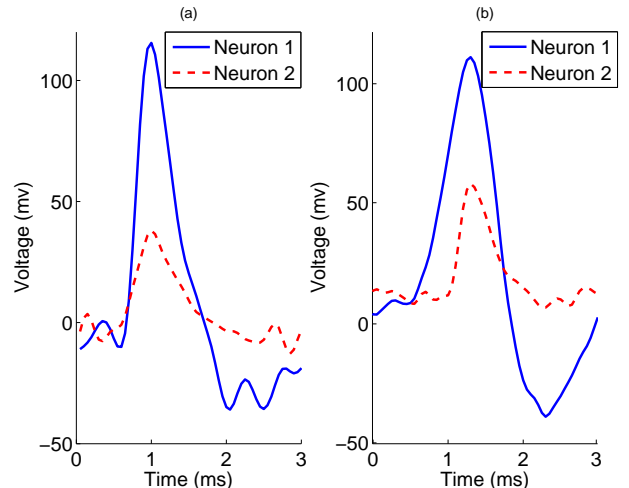


Fig. 4. (a) Clinical and (b) synthetic spike waveform of two neurons.

Both baseline [41] and sophisticated synthetic spike trains [40] were employed for the evaluation. These benchmarks were used to maximize representative different scenarios in real experiments. The baseline spike trains were generated from spike synthesis tool [41]. The tool accurately models factors affecting extracellular recordings, such as ion channels of the membrane, the electrical resistance and capacitance of the membrane, the extended spiking neural surface, background noises and inference from other neurons, and provides an approximation of realistic clinical data. Fig. 4 shows both the clinical and synthetic spike shapes. More importantly, parameters, such as the number of neurons contributing to the spike train, the waveform of neuronal spike, signal to noise ratio and the firing rate, can all be specified in the tool. Through adjusting these parameters, various spike trains can be generated for quantitative evaluations. For our evaluation, three groups of spike trains containing two, three and four neurons were generated. White noise and artifacts noises contributed by background neurons were considered when generating noisy synthetic spike trains. Each group contains spike trains with 11 different noise levels. Under the same noise level, a group of spike trains with neuron's firing rate from 5 Hz to 40 Hz was generated. All the data sets are 100s in length and generated at a sampling rate of 24 KHz. These spike trains provide ideal testing benchmarks to evaluate the proposed algorithm with a variety of noise immunity and realistic background noise. Because our method differentiates neuronal spikes according to spike profiles, it is not effective for bursting spikes that appear as concatenated and with decreasing amplitude. Although bursting spikes can be identified and ruled out with the help of inter-spike-interval histograms and cross-correlograms, addressing how to combine these methods with our work is beyond the scope of the paper. In this paper, we do not take bursting spikes into account.

The proposed algorithm and hardware architectures were thoroughly evaluated following the experimental flow shown in Fig. 5. Algorithms were implemented using Matlab. A built-in Matlab function, *princomp*, was used as the referenced PCA algorithm for comparing with the proposed eigenfilter. The accuracy of the proposed spike sorting algorithm was quantitatively evaluated using the synthetic spike trains. Hardware was modeled using Matlab fixed point tool and FPGA. The impacts of word length of the hardware on power, logic resources consumptions and accuracy of results were also studied using hardware models.

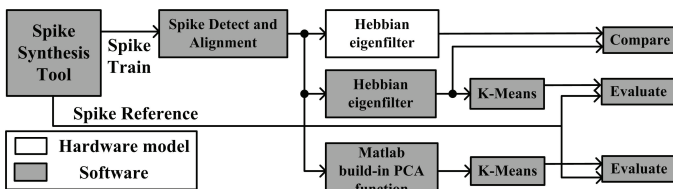


Fig. 5. Experimental flow for algorithm and hardware evaluations.

## B. Algorithm Evaluation

1) *Effectiveness of High Gain Approach*: In order to evaluate convergence efficiency of the Hebbian network, eigenvectors computed by the eigenfilter were compared with the benchmark eigenvectors, which were obtained using the Matlab *princomp* function. The accuracy metric is defined as the dot products between the two vectors, which is,

$$\text{Accuracy}_i = \frac{|\vec{PC}_i \cdot \vec{W}_i|}{\|\vec{PC}_i\| \times \|\vec{W}_i\|} \quad (12)$$

where  $\vec{W}_i$  and  $\vec{PC}_i$  refer to the  $i$ -th synaptic weight and principal component respectively. The results represent angles between the two vectors. When the two vectors have the same direction or the opposite direction, the result equals to one. Otherwise a value in  $[0, 1)$  is obtained.

Fig. 6 shows the learning curves of the Hebbian eigenfilters using uniform learning rate for three synaptic weights. The experiment is based on clinical data obtained from [40]. If a large learning rate,  $\eta = 1$ , is employed, the network begins to oscillate. In contrast, the network would require a significant number of epochs, e.g. around 4300 epochs, to converge, if a small learning rate,  $\eta = 0.1$ , is employed. Under a certain learning rate, the synaptic weight with stronger attraction has faster speed of convergence.

Increasing the uniform learning rate can speed up the convergence and lower the computation latency. However, a large uniform learning rate may also cause instability to the network. Instead of simply increasing uniform learning rates, we employed proposed high-gain approaches to accelerate the convergence.

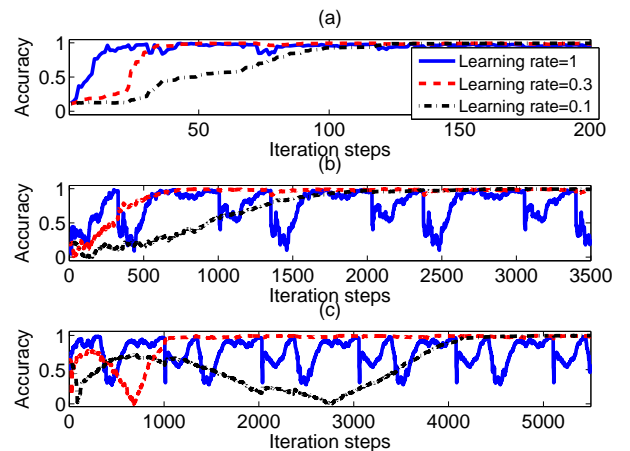


Fig. 6. The convergent curves of (a) the first synaptic weight, (b) the second synaptic weight, (c) the third synaptic weight of Hebbian eigenfilter using uniform learning rate for synaptic weights.

Fig. 7 shows the learning curves of the first three principal components using three different learning rate configurations: the uniform learning rate equals to 0.3; three non-uniform learning rates are specified as,  $\{0.3, 0.4, 0.5\}$ , for the first three synapses; the annealing strategy is defined as,  $\eta_i(n) = \frac{c_i}{1+d_i n}$ , where  $n$  is the number of learning epochs,  $c_i$  and  $d_i$  equal to

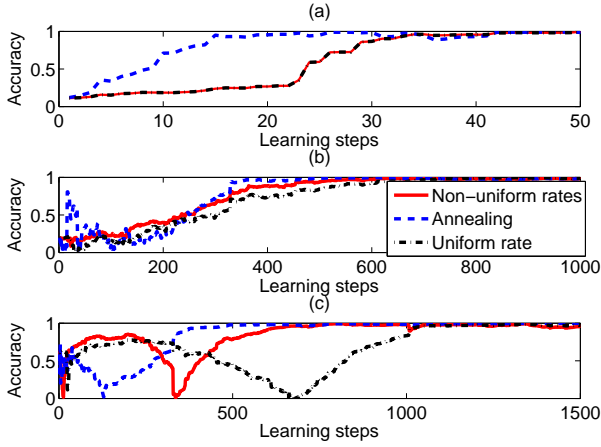


Fig. 7. The convergent curves of (a) the first synaptic weight, (b) the second synaptic weight, (c) the third synaptic weight of Hebbian eigenfilter using non-uniform learning rate, annealing learning rate and uniform learning rate (0.3).

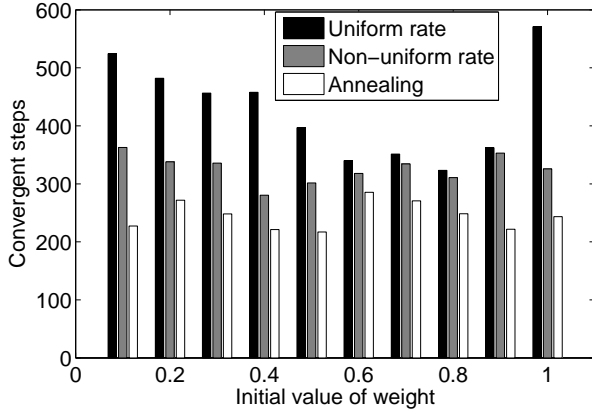


Fig. 8. Convergent speed of Hebbian eigenfilter with different initial conditions. Results are based on 10,000 runs of the the Hebbian eigenfilter algorithm with three different learning rate definitions.

$\{1, 10, 20\}$  and  $\{0.01, 0.05, 0.1\}$  for the first three synapses, respectively. Normalization are performed when the norm of the weights exceeds a threshold, e.g. threshold equals to 4. In general, the annealing approach converges fastest. The eigenfilter with non-uniform learning rates converges more synchronously and quickly than the network using a uniform rate.

Implementing annealing strategy requires more hardware resources than uniform and non-uniform approaches due to the additional annealing and normalization functions. Furthermore, programmable mechanism is required to generate various annealing functions, which complicates the hardware design.

Initialization of the weight vectors may affect the convergence of the Hebbian eigenfilter. The impacts of initialization of the weight vector on the convergent rate of the algorithm were studied. We performed a Monte Carlo simulation to evaluate the sensibility of convergence speed to the initial values. Fig. 8 shows the results of the Monte Carlo simulation

between convergent speed and the initial value of the weight vector. In order to avoid instability, which may be caused by rapid growth of the weights, initial values only vary between 0.1 and 1. Synthetic spikes generated by the spike synthesis tool with various noise levels are used for the evaluation. The results show that the average convergent steps are 426, 325, 245 for the uniform, non-uniform and annealing schemes, respectively. There are slight variations, 16.9%, 5.74% and 7.9% , in the converging speed with different initial values for the three learning rate schemes. However, the variation is small and might just due to statistical randomness.

2) *Algorithm Performance Evaluation*: We use the true positive rate (TPR) and the false positive rate (FPR) to evaluate the performance of our algorithm. The true positive rate is defined by Eq. 13, where  $Q$  is the number of estimated clusters, and  $Num_{correct\_classified\_spikes,i}$  and  $Num_{spikes,i}$  stand for the number of correctly classified spikes of neuron  $i$  and the total number of spikes of neuron  $i$ , respectively.

$$TPR = \frac{\sum_{i=1}^Q \frac{Num_{correct\_classified\_spikes,i}}{Num_{spikes,i}}}{Q} \quad (13)$$

The false positive rate is defined by Eq. 14, where  $Num_{false\_classified\_spikes,i}$  and  $Num_{false\_spikes,i}$  stand for the number of false spikes (not belonging to neuron  $i$ ) assigned to neuron  $i$  and the total number of false spikes for neuron  $i$ , respectively.

$$FPR = \frac{\sum_{i=1}^Q \frac{Num_{false\_classified\_spikes,i}}{Num_{false\_spikes,i}}}{Q} \quad (14)$$

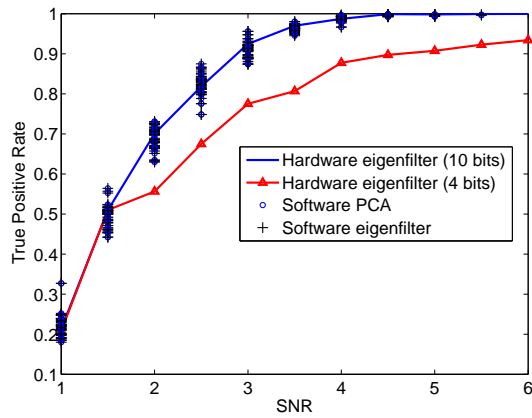
The noise immunity of our Hebbian-based spike sorting algorithm was studied. In the evaluation, a realistic NEO based detection was employed to detect neuronal spikes from noisy spike trains. The detected spikes were aligned at their peak point for Hebbian-based feature extraction. Then K-means clustering was incorporated to automatically cluster each spike. In this paper, we define the SNR of spike trains as the ratio of the mean peak signal level to the standard deviation of the background noise ( $\sigma$ ) [20], which is

$$SNR = \frac{E_k[|max_t(|s_k(t)|)|]}{\sigma} \quad (15)$$

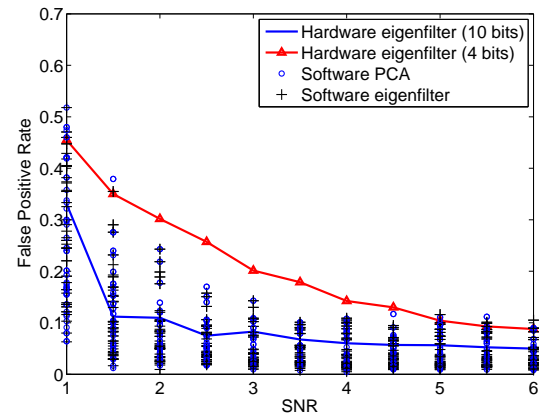
where  $s_k(t)$  is the  $k$ -th spike.

Fig. 9 shows the relationship between the true positive rate and SNR for spike sorting algorithms using the proposed Hebbian eigenfilter and Matlab built-in algorithm for PCA, *princomp*. Comparing Fig. 9 (a), (b) and (c), we can see that at the same SNR level, the smaller the neuron number is, the better classification results are. We can also see that there is little difference between Hebbian eigenfilter and Matlab *princomp* function used for spike sorting. As a result, although Hebbian eigenfilter only computes approximate eigenvectors, it has the same effect as other PCA algorithm in spike sorting process. We also calculate correct classification of hardware with different word length. Results show that word length has a significant impact on the accuracy.

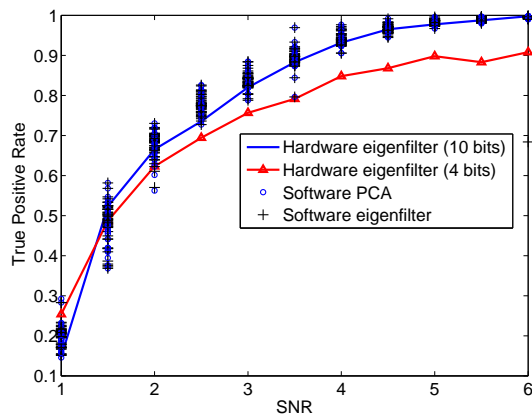




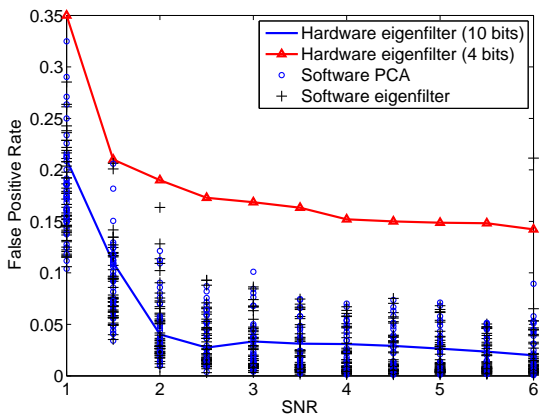
(a)



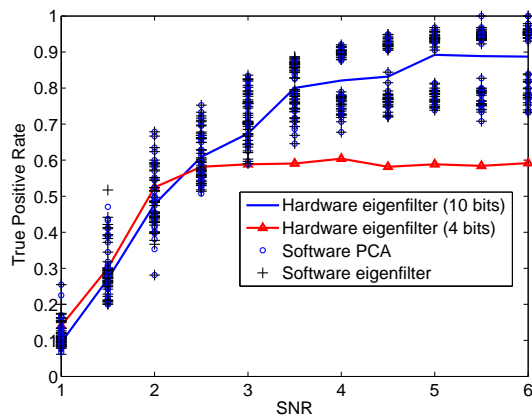
(a)



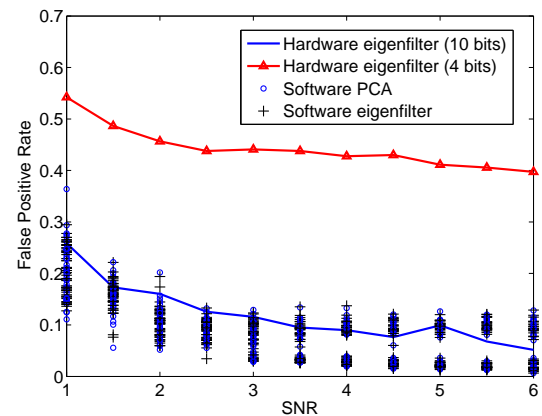
(b)



(b)



(c)



(c)

Fig. 9. Relationship between SNR and true positive rate using Hebbian eigenfilter and Matlab *princomp* function (a) 2 neurons (b) 3 neurons (c) 4 neurons. The hardware (4 and 10 bits) performances for different SNR are also shown in the figure.

Fig. 10. Relationship between SNR and false positive rate using Hebbian eigenfilter and Matlab *princomp* function (a) 2 neurons (b) 3 neurons (c) 4 neurons. The hardware (4 and 10 bits) performances for different SNR are also shown in the figure.

Fig. 10 shows the relationship between the false positive rate and SNR for spike sorting algorithms using Hebbian eigenfilter and Matlab built-in PCA. There is little difference between Hebbian eigenfilter and Matlab *princomp* function. For both methods, the false positive rate falls as the SNR increases. The false classifications of hardware with different word length

were also evaluated. Results show that word length has a significant impact on the performance of spike sorting. A reasonable false positive rate can be obtained with 10 bits word length.

We further compares the accuracy of spike sorting between Hebbian eigenfilter, PCA and wavelet transform approaches.

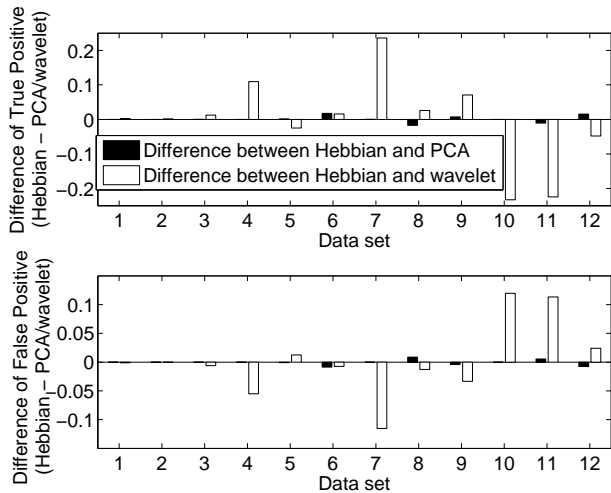


Fig. 11. The difference in true positive rate between Hebbian eigenfilter and PCA or wavelet transform (upper), and the difference in false positive rate between Hebbian eigenfilter and PCA or wavelet transform (lower). Benchmarks #1~#12 are C\_Easy1\_noise005-015, C\_Easy2\_noise005-015, C\_Difficult1\_noise005-015, C\_Difficult2\_noise005-015 taken from [40].

The benchmark data sets are from [40], which captures complex realistic spike shapes, various background noises, and interferences from neurons. Fig. 11 shows the evaluation results. It shows that Hebbian eigenfilter is on par with the PCA-based approach in terms of spike classification accuracy. The variation in accuracy is smaller than 1.8%. Hebbian eigenfilter is also comparable to the wavelet transform approach. Variations in performance have been observed for data sets #3 to #12. The variation in performance is smaller than 23.6%. Performance between Hebbian eigenfilter and wavelet is highly dependable on the data sets. Both the intrinsic property of the benchmark spikes in time and frequency domains, and the directions of feature spaces may affect the performance of spike sorting algorithm. Further discussions and comparisons between different spike sorting algorithms can be founded in [41][26].

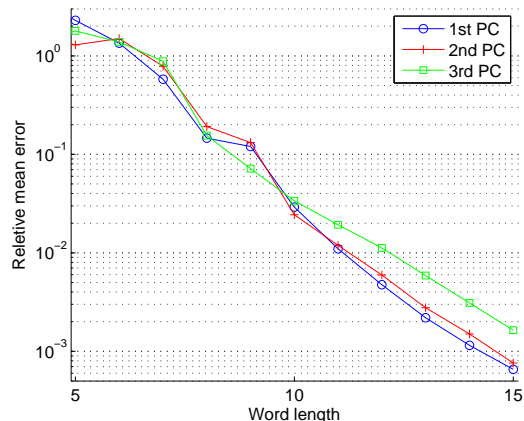


Fig. 12. Relationship between the hardware precision and the word length.

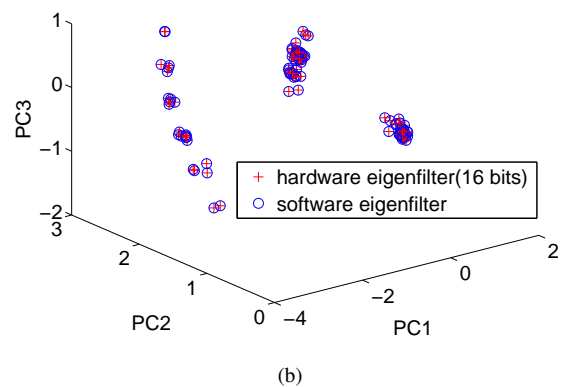
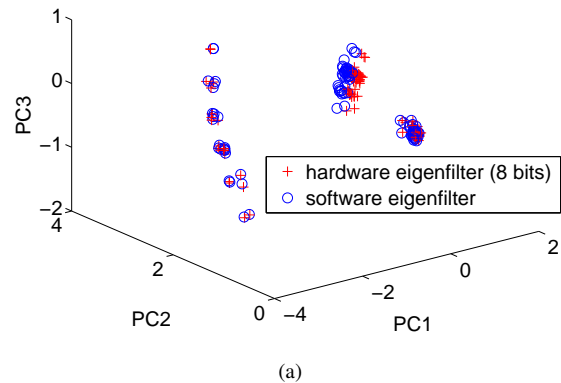


Fig. 13. Projection results of software Hebbian Eigenfilter and (a) 8 bits hardware, (b) 16 bits hardware in principal component space.

### C. Hardware Hebbian Eigenfilter

Word length has a direct impact on precision and also greatly affects system power dissipation and area consumption. For hardware evaluation, we studied the impacts of word length on hardware power, area and accuracy. The target device is a Xilinx FPGA (Spartan6 Low-Power XC6S-LX150L). Our hardware is designed using Xilinx System Generator. Hardware power under different word lengths is obtained by Xilinx Xpower. Xilinx ISE was employed to synthesize, place and route the design to the target FPGA. Hardware resources, such as look-up-tables (LUT, the basic logic element in FPGA) and embedded memory usage of the FPGA, were reported by ISE as well.

1) *Word length, Accuracy and Power Dissipation:* Although truncating hardware word length can lower power and resource consumption, it will also reduce the accuracy of hardware results. We use relative mean error (RME) that is shown in Eq. 16 to describe this accuracy loss, where  $m$  is the length of eigenvector,  $x_i^{software}$  and  $x_i^{hardware}$  are the  $i$ -th scalar element of eigenvector obtained from software and hardware. Fig. 12 shows the relationship between the word length and the precision of the hardware Hebbian eigenfilter. The relative mean error increases exponentially as the word length decreases.

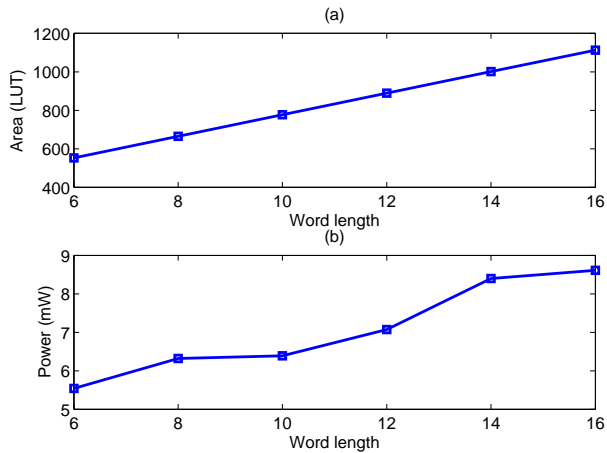


Fig. 14. Relationships between (a) word length and area consumption, (b) word length and power consumption. Area consumption is represented in the term of the number consumed LUT (look-up table), which is the basic logic resources in FPGAs.

TABLE IV  
AREA AND PERFORMANCE OF HEBBIAN EIGENFILTER

	Word Length = 10 bits	Word Length = 16 bits	Software (Intel Core2 E8400 @3GHz)
Number of Slice	777	1113	–
Number of BRAM	41	65	–
Power (mW)	6.4	8.6	–
Learning Latency (ms)	5.6	5.6	93.8
Projection Latency (ms)	$9.6 \times 10^{-4}$	$9.6 \times 10^{-4}$	$3.2 \times 10^{-2}$
Projection Throughput (spike/s)	$1.04 \times 10^6$	$1.04 \times 10^6$	$3.1 \times 10^4$

$$\text{RME} = \frac{\sum_{i=1}^m |x_i^{\text{software}} - x_i^{\text{hardware}}|}{\sum_{i=1}^m |x_i^{\text{software}}|} \quad (16)$$

Fig. 13 (a) and (b) show word length impact on hardware in feature space. In Fig. 13 (a), the word length of hardware Hebbian eigenfilter is 8 bits. A little difference between software and hardware can be seen. In Fig. 13 (b), the word length of hardware is 16 bits, in which the differences between hardware and software are hard to see. Fig. 14 shows the relationships between word length and hardware resources, and power consumptions. As the word length increases, hardware resource and power consumption also increase moderately.

2) *Implementation Results*: Table IV shows area and performance of our Hebbian eigenfilter. Learning latency is the time needed to obtain all principal components of input data. Projection latency is the time needed to accomplish one transformation that transfers a spike potential to a data point in feature space when the eigenvectors are given. Projection throughput is the number of transformations per second. Learning latency of hardware eigenfilter is 17 times faster than software eigenfilter running on a personal computer that has an Intel dual core processor E8400. Projection latency of hardware eigenfilter is 33 times faster than its software

TABLE V  
COMPARISON BETWEEN HEBBIAN EIGENFILTER, PCA AND WAVELET FOR COMPUTATIONAL COMPLEXITY AND HARDWARE PERFORMANCES

	#Operations (million)	Latency (ms)	Area ( $mm^2$ )	#Slice	Power (mW)
Wavelet [27]	49.15	42.4	198.3	30960	255
PCA [22]	48.4	41.8	49.5	7722	49.3
Hebbian eigenfilter (HE)	6.49	5.6	5	777	6.4
Improvement (PCA/HE)	7.5x	7.5x	9.9x	9.9x	7.7x
Improvement (wavelet/HE)	7.6x	7.6x	39.8x	39.8x	39.8x

counterpart.

Table V compares the hardware performance between the FPGA-based Hebbian eigenfilter and other existing hardware systems using PCA and wavelet approaches. Hardware results are varied if different technologies and devices are employed in the implementations. Especially, hardware performances using ASIC, FPGAs and DSP processors are different for a particular design. To obtain a fair comparison, we normalized<sup>1</sup> the hardware implementation results to FPGA equivalent and compared with our implementations. These would provide insightful quantitative evaluations between the different approaches in terms of hardware performance. But these results shouldn't be regarded as specifications of system performances.

In line with [27], number of operations was employed for the algorithm complexity evaluation. To normalize the operation as additions, subtraction is considered to be equal to an addition, and multiplication and division are considered 10 times as complex as an addition. The complexity of wavelet (using Lilliefors Test for dimension reduction [26]) is obtained from [27] and the PCA result is obtained from [22][44]. Covariance matrix calculation and orthogonal iteration based eigenvalue decomposition algorithm are considered for the principal components computation [44]. The complexity derivation of the Hebbian method is presented in Appendix A. Hebbian eigenfilter has the smallest complexity, which is about 13.4% and 13.2% of the PCA and Wavelet complexity measurements, respectively. This would lead to substantial advantage in hardware system development.

Suppose the PCA [22] and wavelet [27] hardware architectures have the same computing capability as the Hebbian eigenfilter, latency or delay will be proportional to the computational complexity. Latency (or delay) results are reported in Table V. The Hebbian eigenfilter approach has a significant advantage in delay that is 7.5 and 7.6 times faster than the PCA and wavelet approaches, respectively. This is critical to be employed in real-time spike sorting. The Hebbian eigenfilter approach has also significant improvement in hardware area. It reduces hardware area significantly from 9.9 to 39.8 times when compared to the PCA and wavelet approaches, respectively. Finally, we evaluate the power consumption for the

<sup>1</sup>We follow a widely acceptable approach [42][43] to convert the ASIC design into FPGA results. Particularly, FPGA implementation is 4.5 times slower than the corresponding ASIC design. In terms of area, FPGA implementation is 21 times larger than equivalent ASIC design in terms of area, and 128 logic counts in FPGAs equals to  $0.82 \text{ mm}^2$  in  $90 \text{ nm}$  CMOS technology [43]. Also, FPGA consumes 12 times more power than the equivalent ASIC.

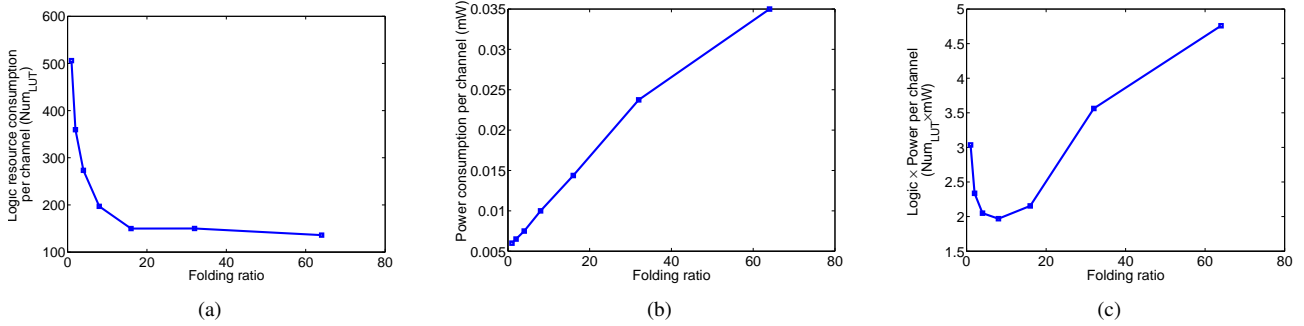


Fig. 15. (a) Relationships between folding ratio and logic resource consumption per channel (in terms of number of look-up table, LUT), (b) folding ratio and dynamic power consumption per channel and (c) folding ratio and production of resource and power consumption.

three methods. Assume that the three hardware systems run at the same clock frequency. Then, the dynamic power consumption is proportional to the hardware area. Results show that the Hebbian eigenfilter has 7.7 and 39.8 times improvement when compared to the PCA and wavelet hardware implementations in terms of power consumption.

#### D. Multi-channel System

Fig. 15 (a) shows the relationship between folding ratio and number of consumed logic resources per channel in terms of LUT. In folded structures, the consumption of logic resources per channel can be reduced by sharing computing resources among channels. Because storage units that cache intermediate results cannot be shared among channels, the area reduction per channel will become less significant as the folding ratio increases.

From Eq. 11, when the folding ratio increases, power consumption per channel also increases. The dynamic power consumption per channel with various folding ratio is shown in Fig. 15 (b).

The power consumption and logic resource costs are inversely related with the increase of folding ratio. If considering power and logic resource together in terms of the production of power and logic resource, an optimal value of folding ratio can be obtained, as shown in Fig. 15(c). An 8-folding ratio is the optimal case for our real-time processing hardware.

#### V. CONCLUSION

This paper presents a hardware efficient Hebbian eigenfilter for principal component analysis, which can be integrated effectively for spike sorting. The proposed algorithm is computationally efficient and is able to filter specified numbers of leading principal components. Two new high gain approaches are presented to improve the network convergence speed of the eigenfilter. New hardware architectures are presented to realize the eigenfilter and to deliver the capability of real-time multi-channel spike sorting. Parallel architectures are employed to exploit the intrinsic data-level independencies in multi-channel spike sorting. Folding technique is discussed to share the computing resources among parallel real-time processing elements. The proposed algorithm and hardware architectures are thoroughly evaluated using both synthetic and

clinical spike trains. When compared to software, the FPGA-based eigenfilter provides 17 and 33 times acceleration when compared to conventional PCA spike sorting in training and real-time projection, respectively. Also, the impacts of folding ratio on power and area consumption of real-time processing modules are studied. Compared with other conventional spike sorting algorithms, such as PCA and wavelet transform, our approach provides hardware implementations with the smallest power dissipation and hardware resource consumption.

#### APPENDIX A

##### COMPUTATIONAL COMPLEXITY OF GHA

Let  $C_{GHA}$ ,  $C_{mean}$  and  $C_{learn}$  be the computational complexity of GHA, mean centering and learning, respectively. Let  $d$ ,  $n$ ,  $l$  and  $L$  be the dimensionality of aligned spikes, the number of spikes for PCA, required leading principal components and iterations, respectively. We can have,

$$C_{GHA} = C_{mean} + C_{learn} \quad (17)$$

$C_{mean}$  is made up of the cost of mean calculation  $C_{mean\_vector}$  and zero-mean transformation  $C_{mean\_center}$ , so,

$$\begin{aligned} C_{mean} &= C_{mean\_vector} + C_{mean\_center} \\ &= (dnC_{add} + C_{div}) + dnC_{sub} \\ &= 2dnC_{add} + C_{mul} \end{aligned} \quad (18)$$

where we assume that  $C_{add} = C_{sub}$  and  $C_{div} = C_{mul}$  (because dividing a constant equals to multiply a constant).

Let  $C_y$ ,  $C_{LT}$ ,  $C_{dW}$ ,  $C_W$  be the cost for calculating  $\vec{y}$ ,  $LT[\vec{y}\vec{y}^T]$ ,  $d\vec{W}$  and  $\vec{W}$  respectively. The total computational cost of Hebbian learning is

$$\begin{aligned} C_{learn} &= (C_y + C_{LT} + C_{dW} + C_W)L \\ &= \left( \left( dlC_{mul} + (d-1)lC_{add} \right) + \frac{l(l+1)}{2}C_{mul} + \right. \\ &\quad \left. \left( \frac{l(l+5)}{2}dC_{mul} + \frac{l(l+1)}{2}dC_{add} \right) + dlC_{add} \right)L \\ &= \left( \left( \frac{l(l+7)}{2}d + \frac{l(l+1)}{2} \right)C_{mul} + \right. \\ &\quad \left. \left( \frac{l(l+5)}{2}d - l \right)C_{add} \right)L \end{aligned} \quad (19)$$

Combining Eq. 22, 18, 19, we obtain the cost of the GHA,

$$C_{GHA} = \left( \left( \frac{l(l+7)}{2}d + \frac{l(l+1)}{2} \right) L + 1 \right) C_{mul} + \left( \left( \frac{l(l+5)}{2}d - l \right) L + 2dn \right) C_{add} \quad (20)$$

$l$  is much smaller than  $d$ ,  $n$ ,  $N$  in spike sorting. The computational complexity of GHA can be estimated to be  $O(dL + dn)$ .

## APPENDIX B

### COMPUTATIONAL COST OF COVARIANCE MATRIX CALCULATION

For the covariance matrix calculation, let  $C_{cov}$ ,  $C_{mean\_cov}$  and  $C_{matrix\_cov}$  be the total computational cost, the computational cost of mean and covariance matrix calculations. We have

$$C_{mean\_cov} = ((n-1)C_{add} + C_{div})d + dnC_{sub} \quad (21)$$

$$C_{matrix\_cov} = d^2(nC_{mul} + (n-1)C_{add}) \quad (22)$$

We assume  $C_{add} = C_{sub}$  and  $C_{div} = C_{mul}$ , then,

$$C_{cov} = C_{mean\_cov} + C_{matrix\_cov} = (d^2n + 2dn - d^2 - d)C_{add} + (d^2n + d)C_{mul} \quad (23)$$

Therefore the computational complexity of covariance matrix calculation can be estimated to be  $O(nd^2)$ .

## REFERENCES

- [1] D. Warren, E. Fernandez, and R. Normann, "High-resolution two-dimensional spatial mapping of cat striate cortex using a 100-microelectrode array," *Neuroscience*, vol. 105, no. 1, pp. 19–31, 2001.
- [2] E. M. Maynard, C. T. Nordhausen, and R. A. Normann, "The Utah intracortical electrode array: A recording structure for potential brain-computer interfaces," *Electroencephalography and Clinical Neurophysiology*, vol. 102, no. 3, pp. 228–239, 1997.
- [3] R. A. Normann, E. M. Maynard, P. J. Rousche, and D. J. Warren, "A neural interface for a cortical vision prosthesis," *Vision Research*, vol. 39, no. 15, pp. 2577–2587, 1999.
- [4] P. J. Rousche, D. S. Pellinen, D. P. Pivin, J. C. Williams, R. J. Vetter, and D. R. Kipke, "Flexible polyimide-based intracortical electrode arrays with bioactive capability," *IEEE Transactions on Biomedical Engineering*, vol. 48, no. 3, pp. 361–371, 2001.
- [5] K. Imfeld, S. Neukom, A. Maccione, Y. Bornat, S. Martinoia, P. Farine, M. Koudelka-Hep, and L. Berdondini, "Large-scale, high-resolution data acquisition system for extracellular recording of electrophysiological activity," *IEEE Transactions on Biomedical Engineering*, vol. 55, no. 8, pp. 2064–2073, 2008.
- [6] L. Berdondini, P. Massobrio, M. Chiappalone, M. Tedesco, K. Imfeld, M. G. A. Maccione, M. Koudelka-Hep, and S. Martinoia, "Extracellular recordings from high density microelectrode arrays coupled to dissociated cortical neuronal cultures," *J. Neuroscience Methods*, vol. 177, pp. 386–396, 2009.
- [7] U. Frey, U. Egert, F. Heer, S. Hafizovic, and A. Hierlemann, "Micro-electronic system for high-resolution mapping of extracellular electric fields applied to brain slices," *Biosensor and Bioelectronics*, vol. 24, pp. 2191–2198, 2009.
- [8] A. Hierlemann, U. Frey, S. Hafizovic, and F. Heer, "Growing cells atop microelectronic chips: Interfacing electrogenic cells in vitro with CMOS-based microelectrode arrays," *Proceedings of the IEEE*, vol. 99, no. 2, pp. 252–284, 2011.
- [9] R. Sitaram, A. Caria, R. Veit, T. Gaber, G. Rota, A. Kuebler, and N. Birbaumer, "fMRI brain-computer interface: A tool for neuroscientific research and treatment," *Computational Intelligence and Neuroscience*, vol. 2007, 2007.
- [10] B. Blankertz, G. Dornhege, M. Krauledat, K. Müller, V. Kunzmann, F. Losch, and G. Curio, "The Berlin brain-computer interface: EEG based communication without subject training," *IEEE Transactions Neural Systems Rehabilitation Engineering*, vol. 14, no. 2, pp. 147–152, 2006.
- [11] A. Gunduz, J. C. Sanchez, P. R. Carney, and J. C. Principe, "Mapping broadband electrocorticographic recordings to two-dimensional hand trajectories in humans," *Neural Networks special issue on Brain-Machine Interfaces*, vol. 22, no. 9, pp. 1257–1270, 2009.
- [12] M. Gandolfo, A. Maccione, M. Tedesco, S. Martinoia, and L. Berdondini, "Tracking burst patterns in hippocampal cultures with high-density CMOS-MEAs," *Journal of Neural Engineering*, vol. 7, no. 5, 2010.
- [13] A. Vogt, G. Wrobel, W. Meyer, W. Knoll, and A. Offenhausser, "Synaptic plasticity in micropatterned neuronal networks," *Biomaterials*, vol. 26, no. 15, pp. 2549–2557, 2005.
- [14] J. Puchalla, E. Schneidman, R. Harris, and M. Berry, "Redundancy in the population code of the retina," *Neuron*, vol. 46, pp. 493–504, 2005.
- [15] G. Santhanam, S. I. Ryu, B. M. Yu, A. Afshar, and V. Shenoy, "A high-performance brain-computer interface," *Nature*, vol. 442, pp. 195–198, 2006.
- [16] M. A. L. Nicolelis, "Brain-machine interfaces to restore motor function and probe neural circuits," *Nature Reviews Neuroscience*, vol. 4, pp. 417–422, 2003.
- [17] L. R. Hochberg, M. D. Serruya, G. M. Fries, J. A. Mukand, M. Saleh, A. H. Caplan, A. Branner, D. Chen, R. D. Penn, and J. P. Donoghue, "Neuronal ensemble control of prosthetic devices by a human with tetraplegia," *Nature*, vol. 442, pp. 164–171, 2006.
- [18] M. Linderman, G. Santhanam, C. Kemere, and et al, "Signal processing challenges for neural prostheses," *IEEE Signal Processing Magazine*, vol. 25, no. 1, pp. 18–28, 2008.
- [19] M. Lewicki, "A review of methods for spike sorting: the detection and classification of neural action potentials," *Network: Computational Neural System*, vol. 9, pp. 53–78, 1998.
- [20] Z. Zumsteg, C. Kemere, S. O'Driscoll, G. Santhanam, R. Ahmed, K. Shenoy, and T. Meng, "Power feasibility of implantable digital spike sorting circuits for neural prosthetic systems," *IEEE Transaction Neural System Rehabilitation Engineering*, vol. 13, pp. 272–279, 2005.
- [21] J. Carmona, M. Lebedev, R. E. Crist, J. O'Doherty, D. Santucci, D. Dimitrov, P. Patil, C. Henriquez, and M. A. Nicolelis, "Learning to control a brain-machine interface for reaching and grasping by primates," *PLoS Biol*, vol. 1, no. 2, pp. 193–208, 2003.
- [22] 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*, pp. 434–435, 2009.
- [23] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2007.
- [24] G. L. Gerstein and W. A. Clark, "Simultaneous studies of firing patterns in several neurons," *Science*, vol. 143, pp. 1325–1327, 1964.
- [25] K. G. Oweiss, "A systems approach for data compression and latency reduction in cortically controlled brain machine interfaces," *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 7, pp. 1364–1377, 2006.
- [26] R. Quiroga, Z. Nadasdy, and Y. Ben-Shaul, "Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering," *Neural Computing*, vol. 8, pp. 1661–1687, 2004.
- [27] S. Gibson, J. Judy, and D. Marković, "Technology-aware algorithm design for neural spike detection, feature extraction and dimensionality reduction," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 18, no. 5, pp. 469–478, 2010.
- [28] Z. Nadasdy, R. Q. Quiroga, Y. Ben-Shaul, B. Pesaran, D. Wagenaar, and R. Andersen, "Comparison of unsupervised algorithms for on-line and off-line spike sorting," in *32nd Annu. Meeting Soc. Neurosci.*, 2002.
- [29] A. Zviagintsev, Y. Perelman, and R. Ginosar, "Low-power architectures for spike sorting," in *Proc. 2nd Int. IEEE EMBS Conf. Neural Eng.*, 2005.
- [30] K. Awais and M. Andrew, "On-chip feature extraction for spike sorting in high density implantable neural recording systems," in *Biomedical Circuits and Systems Conference*, 2010.
- [31] K. Kim and S. Kim, "Neural spike sorting under nearly 0-db signal-to-noise ratio using nonlinear energy operator and artificial neural-network classifier," *IEEE Transactions on Biomedical Engineering*, vol. 47, pp. 1406–1411, 2000.
- [32] J. MacQueen, "Some methods for classification and analysis of multivariate observations," *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press*, vol. 1, pp. 281–297, 1967.

- [33] K. Reddy and T. Herron, "Computing the eigen decomposition of a symmetric matrix in fix-point algorithms," in *IEEE Bangalore Section Tenth Annual Symposium*, 2001.
- [34] K. Fukunaga, *Introduction to Statistical Pattern Recognition, second edition*. New York: Academic Press.
- [35] D. Hebbian, *The Organization of Behavior*. New York: Wiley & Sons.
- [36] S. S. Haykin, *Neural Networks and Learning Machines*. New York: Prentice Hall.
- [37] D. Jeong and S. Lee, "Merging back-propagation and hebbian learning rules for robust classifications," *Neural Networks*, vol. 9, no. 7, pp. 1213–1222, 1996.
- [38] F. Palmieri, "Hebbian learning and self-association in nonlinear neural networks," in *International Joint Conference on Neural Networks*, 1994.
- [39] K. Harris, D. Henze, K. Csicsvari, and H. Hirase, "Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements," *Journal of Neurophysiology*, vol. 84, pp. 401–414, 2000.
- [40] R. Quiroga, "Wave clus." <http://www2.le.ac.uk/departments/engineering/research/bioengineering/neuroengineering-lab/spike-sorting.htm>, July 2009.
- [41] L. Smith and N. Mtetwa, "A tool for synthesizing spike trains with realistic interference," *Journal Neuroscience Methods*, vol. 159, pp. 170–180, 2007.
- [42] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *FPGA*, 2006.
- [43] K. Compton and S. Hauck, "Automatic design of area-efficient configurable ASIC cores," *IEEE Transactions on Computers*, vol. 56, no. 5, pp. 662–672, 2007.
- [44] G. Golub and C. Loan, *Matrix Computation*. The Johns Hopkins University Press, 1996.