Abstract—The emergence of microelectrode array enables the study of real-time neurophysiological activities across multiple regions of the brain. However, the real-time extracellular voltage potentials recorded on any electrode represent the simultaneous electrical activity of an unknown number of neurons which present a critical challenge to the accuracy of interpretation and identification of the neural circuitry in the subsequent analysis. In this paper, we present a principal component analysis approach utilizing Hebbian eigenfilter to identify the corresponding electrical activities to each neuron, or namely spike sorting. The Hebbian eigenfilter greatly simplifies the computational complexity of eigen-projection. An efficient and low-power hardware architecture is proposed. The performance, accuracy and power consumption of our Hebbian eigenfilter are thoroughly evaluated through synthetic spike trains. The proposal enables real-time spike sorting and analysis, and leads the way towards future motor and cognitive neuroprosthetics.

I. INTRODUCTION

Recently, there has been much interest in developing brain-machine interface (BMI) technology to help improve the quality of life, to restore motor or cognitive functions for people with severe disabilities and injuries [1]. In general, BMI includes means of sensing neurophysiological signals, methods of decoding and mapping of the neuronal signals into a collection of motor or behavior primitives.

Neurophysiological signals can be detected or sensed in various ways. For non-invasive systems, such as functional magnetic resonance imaging (fMRI) [2] and Electroencephalography (EEG) [3], making use of methods for recording electrical or magnetic field provide a convenient approach at the cost of accuracy and mobility. Alternatively, multi-electrode array implant provides a promising solution for recording neurophysiological activities across multiple regions of brain and enables a comprehensive understanding of the interactions between groups of neurons. Applications of such methodology in neuroprosthetics to paralysed or spinal cord injured patients have been reported in both experimental and clinical studies [4].

Despite that implanted electrodes can accurately measure the extracellular action potentials, the recorded action potentials are often contributed from more than one neuron. The inter-neuronal interferences can greatly corrupt the temporal patterns from the spike train and degrade the signal decoding and interpretations in the subsequent steps of BMI. Distinguishing the action potentials of different neurons detected on the same electrode, or namely spike sorting, becomes the utmost challenging task.

A number of spike sorting algorithms using different signal processing and machine learning methods have been proposed in the literature [5]. Techniques that employ substantial statistics or complex numerical methods may improve the spike sorting accuracy but at an expense of throughput performance. Very often, microprocessors or DSP processors are employed in BMI and the throughput and power dissipation of multi-channel spike processing become a major bottleneck [6]. In this paper, we propose a Hebbian eigenfilter approach that is based on general Hebbian algorithm (GHA) to approximate the eigen projection of spikes in principal component analysis (PCA) such that the computational complexity can be greatly reduced. 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

1. 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, which is the most computational demanding part in spike sorting.

2. An FPGA-based architecture, which exploits the intrinsic task-independency in the eigenfilter, is presented and that 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 spike sorting realization and, can be readily employed in BMI or multi-channel recording systems.

3. 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 noise from the analogue circuit. The relationship between word length and power dissipation is also studied.

This paper consists of five parts. The second part introduces basic ideal of neurophysiological multi-channel recording and our PCA based spike sorting algorithm. The third part describes the structure of our hardware that implements GHA. The fourth part sets up experiment environment that generates ground truth known synthetic spike trains to verify our software and hardware. The fifth part evaluates our GHA based
spike sorting algorithm and GHA hardware using synthetic spike trains.

II. SPIKE SORTING USING HEBBIAN EIGENFILTER

A. Neurophysiological Multi-electrode Recording and Spike Sorting

Using multi-electrode arrays implanted neurosurgically in the brain region of interest is the most effective method to detect and study neuron activities in multi-channel recording systems. In such system it is incorrect to assume that action potentials appearing on one electrode just come from one neuron [7]. As a result, associating recorded action potentials to their neural source, namely spike sorting, is needed. Another reason that spike sorting is needed in real time multi-electrodes recording system is its ability to compress raw data coming from electrodes [8]. This compression can reduce the bandwidth needed to send data or the memory needed to store data.

The flow of spike sorting is shown in Figure 1, which consists of three basic process, 1) Spike detection and alignment, 2) Feature extraction, 3) Classification. Spike detection process detects neuron spikes based on some feature of the spike such as amplitude. All detected spikes are aligned in alignment process. According to predefined spikes’ features such as peak-to-peak amplitude, width or principal component, aligned spikes can be clustered into different groups.

In this paper, we proposed a general Hebbian algorithm (GHA) [9] based spike sorting algorithm. Spike detection is based on an amplitude threshold. According to [10], we set the threshold to

\[ \text{Thr} = 4\lambda_n \quad ; \quad \lambda_n = \text{median}\frac{|x|}{0.6745} \]  

(1)

where \( x \) is the noisy signal and \( \lambda_n \) is an estimate of the standard deviation of the background noise. For each detected spikes, 64 samples are saved for further analysis. All spikes are aligned to their maximum point at the 30th sample point. Aligned spikes are projected into principal component space for classification by principal component analysis (PCA). In our algorithm, the GHA is applied to compute principal components. At last k-means cluster algorithm [9] clusters data points in principal component space into different groups. Each group presents spikes coming from one certain neuron.

B. Hebbian Eigenfilter for PCA

Feature extraction is a critical part in spike sorting process. Many algorithms for feature extraction have been developed, such as principal component analysis (PCA), independent component analysis (ICA), and wavelet-transform, etc. But none of these algorithms can beat other algorithms in all aspect. In this paper we use PCA based algorithm to extract spike features.

Principal component analysis (PCA) is a mathematical procedure that transfers a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components. PCA is an effective way to identify patterns in data and actually projects data to principal component space that highlights the similarities and differences in data. This projection in PCA is a feature extraction process and can be used by spike sorting algorithm. As a result PCA is basic and widely used method in spike sorting process. The advantage of PCA is that it is very effective for high dimensional data. Principal components (PCs) are eigenvectors of the covariance matrix of input data. Due to eigenvectors computing, PCA is characterized by high computational complexity that is a major drawback effecting PCA to be used in power and time constrained BMI application.

However, in the case of spike sorting, the first few principal components account for almost all the variability in the data. There is no need to compute all the principal components of input data. An algorithm that can approximate only the first few principal components is enough. In this paper we use generalized Hebbian algorithm (GHA) to get the first few principal components of input data. It is based on Hebb’s postulate of learning and is shown as

\[ y_i = \sum_{i=1}^{m} w_{ji}(n)x_i(n) \quad j = 1, 2, 3, \ldots, l \]  

(2)

\[ w_{ji}(n+1) = w_{ji}(n) + \delta w(n) \]  

(3)

\[ \delta w_{ji}(n) = \eta(y_j(n)x_i(n) - y_j(n)\sum_{k=1}^{j} w_{ki}(n)y_k(n)) \quad j = 1, 2, \ldots, l; \quad i = 1, 2, \ldots, m \]  

(4)

where \( x_i(n) \) is the \( i \)th component of input vector \( x(n) \), \( y_j(n) \) is the \( j \)th component of output vector \( y(n) \), \( w_{ji}(n) \) is a component of the \( l \times m \) synaptic weight matrix. The initial value of synaptic weights, \( w_{ji}(0) \), are initialized to small random values. Learning rate parameter \( \eta \) should be assigned to a small positive value. For large \( n \), the synaptic weight

\[ w_j(n) = [w_{j1}(n), w_{j2}(n), w_{j3}(n), \ldots, w_{jm}(n)] \]

converges to the \( j \)th principal component of input vector \( x(n) \). From above equations we can see that GHA is low computational complexity due to its approximation feature. And it has the nature to approximate only the first few principal components. As result we think the GHA is a better choice to be used in PCA based spike sorting algorithm.
III. CIRCUIT IMPLEMENTATION OF HEBBIAN EIGENFILTER

We proposed a structure of Hebbian eigenfilter that uses GHA to filter the first few eigenvector, which is shown in Figure 2. This structure can be divided into two parts, one part uses aligned spikes (input data of Hebbian eigenfilter) and synaptic weight to compute output, and LT(output · outputT) that is lower triangular matrix of output · outputT; the other part updates synaptic weight. Word length is a parameter in our structure, which can be changed to get different hardware configuration. At last principal component will be got and stored in RAM. Output will be stored in RAM and used subsequent process.

The data path computing output and LT(output · outputT) is shown in Figure 3. The output is the inner product of input data and synaptic weights. The product and sum unit in Figure 3 computes inner product. After inner production output will be stored in register and used to get lower triangular matrix of output · outputT. At this time inner production has already finished, so product unit for inner product can be reused. The results of LT(output · outputT) are stored in registers for further used to compute new synaptic weights.

The hardware resources needed depends on the number of principal components needed, the number of spikes used for learning and the sample points of each spike. For example, in our case the first three principal components are needed, 120 spikes are used for learning and for each spike there are 64 sample points. Three product and sum units are needed to update synaptic weights. 3 × 64 memory cells are used to store synaptic weight. 3 × 64 memory cells stores the output and 120 × 64 memory cells stores input data.

Figure 4 is the data path to update synaptic weights. It is just a general structure. The actual number of multipliers and adders depends on how many principal components are needed. In our case, three principal components are computed at the same time. So nine multiplier and nine adder are needed.

A circuit implementing k-means algorithm is also designed by us. Hebbian eigenfilter and k-means classification hardware compose a full spike sorting engine that can do feature extraction and classification. The basic idea of k-means is to partition data into some number K of clusters. In order to accomplish this, k-means computes K centroids that are centers of each cluster and then assigns data points to its nearest centroid. The object of k-means algorithm is to minimize the object function below,

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \| x_n - \mu_k \|^2$$  \hspace{1cm} (5)

Where \( \mu_k \) presents centroids of each cluster, \( x_n \) is data point and for each data point \( x_n \), \( r_{nk} \) is a corresponding set of binary indicator variables, \( r_{nk} \in \{0, 1\}, wherek = 1, \ldots, K \) describing which of the K clusters the data point \( x_n \) is assigned to, so that if data point \( x_n \) is assigned to cluster \( k \) then \( r_{nk} = 1 \), and \( r_{nj} = 0 \) for \( j \neq k \). At first \( \mu_k \) are randomly initialized. Data points are assigned to its nearest centroids. Then the centroids are updated according to

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}$$ \hspace{1cm} (6)

The above iteration will not stop until object function \( J \) reach its minimum value. At this moment k-mean algorithm accomplishes clustering.

The structure of our k-means hardware is shown in Figure 5. Input data in RAM that are results of Hebbian eigenfilter are data points needed to be clustered. Registers storing
centroids of each cluster are initialized by some data in RAM. Accumulators, dividers and the counters are units computing new centroids. The arithmetic unit that computes distance between input data point and centroids outputs results to compare unit. Compare unit tells controller that which is the nearest centroid to the data point. According to this, controller assigns the data point to corresponding centroid and enables corresponding accumulator and counter. After all input data points are assigned, dividers are enabled. New centroids are the results of dividers and stored into registers.

IV. EXPERIMENT ENVIRONMENT SET UP

A. Input Data Preparation

Two kinds of data can be used to verify our GHA based spike sorting algorithm and Hebbian eigenfilter, one is clinical spike trains and the other is synthetic spike trains. Clinical spike trains are real neurons activities, but no one knows the ground truth of these data, i.e. how many neurons contribute to these spikes, and each neuron’s spikes. Synthetic spike trains are generated by some spike synthesis tool that uses mathematical methods to model physical mechanism of spike generation. The advantage of using synthetic data is that ground truth of data is known and evaluation on software and hardware can be easily done. In this paper we use a tool developed by University of Stirling [11] to generate synthetic spikes and evaluate our proposed algorithm and structure.

Stirling’s spike synthesis tool can generate a spike train that consists of spikes coming from a certain number of neurons, and a record of each neuron’s spike. This record referred as spike reference in this paper contains the exact time of each neuron’s spikes, from which we can know the ground truth of the spike train. By changing some parameter in Stirling’s tool, we can determine the number of neuron contribution to the spike train, the shapes of each neuron’s spike, signal noise ratio of the spike train, the distribution of each neuron’s spike on time domain and sampling rate. In this paper, we generate two kinds of spike train, one contains spikes from two neurons and the other one contains spikes from three neurons. Figure 6 shows spike shapes of two neurons, spikes in the left are generated by synthesis toll, spikes in the right are clinical data.

The sample rate is 25600 sample points per second. All neuron generates Gaussian distribution spikes.

B. Experiment Environment

The experiment environment for evaluating algorithm and hardware is shown in Figure 7. Spike synthesis tool generates noisy spike train and spike reference. Spike detection and alignment block uses noisy spike train to generate aligned spikes to hardware GHA, software GHA and Matlab build-in function separately. Using spike reference and result of K-Means, correct classification rate can be calculated. By comparing correct classification rate of spike sorting algorithm using GHA and Matlab build-in eigenvector computing function seperately, we can know if GHA has the same effect as other eigenvector computing algorithm in spike sorting process. Results of hardware GHA and software GHA will be compared to find differences between software and hardware when changing word length in hardware.

V. RESULT

A. Algorithm Evaluation

The most important characteristic of spike sorting algorithm is the ability to accurately classify neural spikes. How well the spike sorting algorithm is depends on the number of neurons, the characteristic of signal (i.e. the SNR of signal, the distribution of each neuron’s spike on the time domain), and how different the waveforms are. In this paper we use a
The definition of SNR that is the power ratio between the signal and the noise,

\[
SNR_{\text{dB}} = 10 \times \log_{10}(P_{\text{signal}} / P_{\text{noise}})
\]  

(7)

where \( P \) is average power. The average firing rate of each neuron is 30 spikes per seconds. And spikes generated by each neuron are Gaussian distribution. We generate spike trains with different SNR. Using these spike trains we calculate correct rate of spike sorting algorithm using GHA and Matlab function. Figure 8 and Figure 9 shows correct rate of spike sorting algorithm when neuron number equals to 2 and 3. Correct rate is defined as,

\[
\text{CorrectRate} = \frac{N_{\text{correctclassifiedspikes}}}{(N_{\text{detectedspikes}} + N_{\text{missedspikes}})}
\]  

(8)

where \( N \) presents the number. Comparing Figure 8 and 9, we can see that at the same SNR level, the less neuron number is, the better classification results are got. We can also see that there is little different difference between GHA and Matlab build-in eigenvector computing algorithm. As a result, although GHA only compute approximate eigenvectors, it has the same effect as other eigenvector computing algorithm in spike sorting process. We also compare the result of hardware with different word length. Word length also has a significant impact on the result.

B. Hardware Evaluation

For hardware evaluation, we studied the impact of word length on hardware power and result. It is a guideline for hardware designer and helps them decide the word length or the tunable range of word length in their system. Because we want to change word length each time, FPGA is the best choice for such work due to its reconfigurable ability. The target device is Xilinx Virtex6 FPGA chip xc6vsx315t with package 3ff1156. Our hardware is designed using Xilinx System Generator. Hardware power under different word lengths is also got by power analysis in System Generator.

1) Power: We can see the relationship between power and word length from Figure 10. There is a linear relationship between power and word length. When the word length changes from 56 bits to 35 bits, the power decreases nearly 3 times.

2) Mean Error between Hardware and Software: Although truncating hardware word length can lower power, it will also make the accuracy of result become worse. We use relative mean error (RME) that is shown in equation (8) to describe this accuracy loss,

\[
RME = \frac{\sum_{i=1}^{n} |x_i^{\text{software}} - x_i^{\text{hardware}}|}{\sum_{i=1}^{n} |x_i^{\text{software}}|}
\]  

(9)

where \( n \) is the length of eigenvector, \( x_i^{\text{software}} \) and \( x_i^{\text{hardware}} \) are the \( i \)th scalar element of eigenvector got from software
and hardware. Figure 10 shows the relationship between word length and average mean error between eigenvectors got from software and hardware. This result is got by change all parts’ word length in our Hebbian eigenfilter except the constant multiplier.

Figure 11 and Figure 12 compare GHA result with our Hebbian eigenfilter’s result under different word length. In Figure 11, the word length of Hebbian eigenfilter is 35 bits. A little difference between software and 35 bits hardware can be seen. In Figure 12, the word length of hardware is 48 bits, which has the same result as the software.

3) Area and Throughput: Table 1 shows area and performance of our Hebbian eigenfilter with word length equals to 35 and 56. Learning latency is defined as the time needed to get all principal components of input data. Projection latency is defined as the time needed to accomplish one transformation that transfer a spike potential to a data point in principal component space.

VI. CONCLUSION

In this paper we proposed a PCA based spike sorting algorithm using general Hebbian algorithm (GHA) that is suit for real time BMI application. A hardware implementing GHA, Hebbian eigenfilter, is designed and implemented in FPGA. Both GHA based spike sorting algorithm and Hebbian eigenfilter are thoroughly evaluated using a spike synthesis tool. Proposing and implementing a complete high throughput, power aware spike sorting system that suits for real-time, multi-electrode recording BMI application is our future work.

REFERENCES