On-chip Principal Component Analysis with a Mean Pre-estimation Method for Spike Sorting

Tung-Chien Chen[1,2], Kuanfu Chen[2], Wentai Liu[2], and Liang-Gee Chen[1]

[1]National Taiwan University, Taipei, Taiwan; [2]University of California, Santa Cruz, CA, USA;

Abstract— **Principal component analysis (PCA) spike sorting hardware in an integrated neural recording system is highly desired for wireless neuroprosthetic devices. However, a large memory is required to store thousands of spike events during the PCA training procedure, which impedes the on-chip implementation for the PCA training engine. In this paper, a mean pre-estimation method is proposed to save 99.01% memory requirement by breaking the algorithm dependency. According to the simulation result, 100 dB signal-to-error power ratio can be preserved for the resulting principal components. According to the implementation result, 6.07 mm^2 silicon area is required after a 283.16 mm^2 area saving for the proposed PCA training hardware.**

I. Introduction

On-chip implementation of neural signal processing along with the recording circuitry can significantly reduce the data bandwidth and is a key to enable the wireless neural recording system with a large amount of electrodes [1]. Without such data processing, large amount of data need to be transferred to a host computer, and typically a cable is required. In this case, patients and test subjects are restrained from free movement, which disturbs the measurement and observations and impedes the advance in fundamental neuroscience research and wireless neural prosthetic devices for patients.

A promising approach to achieve the bandwidth reduction is to extract spike features immediately after spike detection on the implant site [1–5]. Only the event times and some additional features about classification are transmitted after the signal processing. This approach achieves more than 100-fold data reduction while preserving the neuron signature for discrimination of individual neuron signal sources. The principal component analysis (PCA) [1–3] and wavelet transformation [4, 5] are the most widely used tools for this approach. In this paper, we propose to achieve a dedicated co-processor for PCA-based spike sorting algorithm.

A complete PCA-based spike sorting hardware has two major parts—the parameter training and the on-the-fly feature extraction. After spike detection, the training collects the detected spike events and extracts the major characteristic vectors, the principal components (PCs). In general, these vectors can optimally differentiate neurons in least square terms. The operations of the covariance matrix and the leading eigenvectors are two major mathematic calculations in this part. In the on-the-fly feature extraction, the feature scores are extracted by projecting the detected spike events on the PCs. The operation of inner product is required here. Note that periodic re-training is frequently required because of the movement of the electrodes and the change of the environment [6].

In realizing a complete on-chip PCA-based spike sorting co-processor, the most challenging problem is to design the PCA training engine to calculate PCs from the detected spike events.

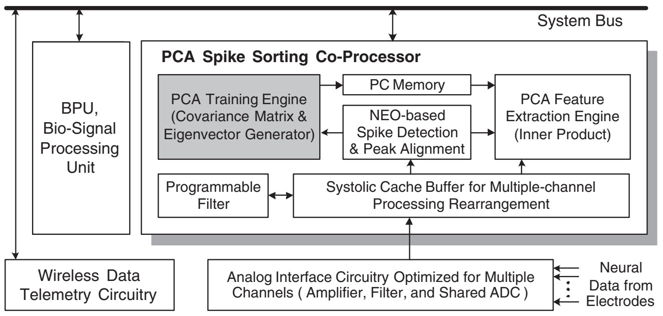
System Bus
PCA Spike Sorting Co-Processor
BPU, Bio-Signal Processing Unit
PCA Training Engine (Covariance Matrix & Eigenvector Generator)
PC Memory
NEO-based Spike Detection & Peak Alignment
PCA Feature Extraction Engine (Inner Product)
Programmable Filter
Systolic Cache Buffer for Multiple-channel Processing Rearrangement
Wireless Data Telemetry Circuitry
Analog Interface Circuitry Optimized for Multiple Channels (Amplifier, Filter, and Shared ADC)
Neural Data from Electrodes

Fig. 1. The proposed neural signal processing platform supporting the PCA spike sorting algorithm.

There are two main difficulties. First, eigenvalue decomposition (EVD) of the covariance matrix is typically required to generate the eigenvector. The EVD procedure requires matrix diagonalization that is very complicated for hardware implementation. Second, thousands of detected spike events are recommended to used for PCA training. It is necessary to store all these spike events in a large on-chip memory because of the algorithm dependency. The large on-chip memory requirement impedes the low-area implementation.

In [7], the first eigenvector generation hardware is proposed based on the iterative eigenvector distilling algorithm [8]. In this paper, we will focus on the second problem to complete a self-contained PCA spike sorting engine. The rest of this paper is organized as follows. In Section II, a big map of the neural signal processing platform is proposed along with the PCA spike sorting co-processor. Section III describes the memory problem and the proposed solution for on-chip PCA training engine. The corresponding architecture design is described in Section IV. Finally, Section V presents the implementation and simulation results, and Section VI concludes this work.

II. Neural Signal Processing Platform and PCA Spike Sorting Co-processor

Figure 1 shows the proposed neural signal processing platform supporting the PCA spike sorting algorithm. The platform architecture is a versatile hardware structure that can provide the interoperability for different functional modules, the scalability for different application requirements, and the adaptability for future extensions without substantial modifications. Further, a bio-signal processing unit (BPU) is embedded to provide the flexibility in algorithm development while a dedicated co-processor for PCA spike sorting is designed to efficiently accelerate the computationally intensive algorithm. Under the real-time, low-power, and small-area constraints, this platform structure along with the general-purpose BPU and the dedicated

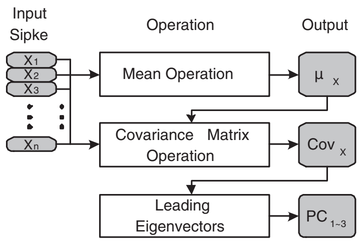

Fig. 2. Flow chart and IO requirement for the original PCA training algorithm.

spike sorting coprocessor can provide both flexibility and efficiency for signal processing in neural prosthetic devices.

For the PCA spike sorting co-processor, the neural data are amplified, filtered, and digitized by the analog frontend interface circuitry (AFIC) and then fed to the co-processor. A band-pass digital filter is first performed to reject the low-frequency field potential and high-frequency thermal noise. Then, the nonlinear energy operator (NEO) spike detector [9, 10] calculates the energy function for the neural waveform. Once the result reaches the threshold at the peak of the convex curve, a find-spike-event signal is sent. If a spike event is detected, a section of neural waveform is sent to both the PCA training engine and feature extraction engine. The training engine calculates the covariance matrix and eigenvectors of the detected spike events. The results of leading eigenvectors, or the PCs, are stored in the PC memory. After training, the feature extraction engine performs inner product between the trained PCs and the subsequent detected spike events to produce the feature scores.

As for signal processing for multiple channels, the traditional parallel hardware that duplicates the processing unit multiple times for multiple channels results in a large area overhead. In order to save the hardware cost, we start with the optimized AFIC [11] that shares an ADC and part of amplifier by multiple channels in a multiplexed fashion. Then, a memory hierarchy of systolic cache buffer is proposed in [12] to efficiently share one processing unit for multiple channels and do sequential scheduling on cyclic basis without a bulky memory and processing latency. Note that this cycle-basis scheduling among different channels is automatically controlled by a local finite state machine (FSM) for the filtering, spike detection, and feature extraction hardware units. For the PCA training engine, the training schedule and period among channels are controlled by BPU and can be programmed according to the application requirements.

III. Problem Statement and Proposed Method

A. Memory Problem for PCA Training Hardware

During the parameter training, thousands of detected spike events are recommended to be used in PCA training. It is necessary to store all these spike events in a large on-chip memory because of the algorithm dependency. The large memory requirement makes the small-area implementation impossible for the PCA training engine.

In order to have a clear idea between the hardware requirement and algorithm dependency, Fig. 2 describes the flow chart and the IO requirements of the PCA algorithm. The corresponding mathematic equations are listed as follows:

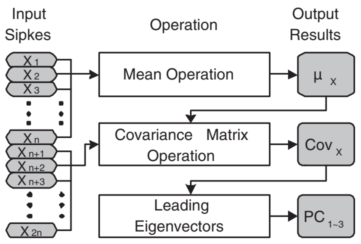

Fig. 3. Flow chart and IO requirement for the PCA training algorithm with the proposed mean pre-estimation method.

$$\begin{aligned} \mu_x &= \frac{\sum_{i=1}^{n} X_i}{n} \\ Cov_x &= \frac{\sum_{i=1}^{n}((X_i-\mu_x)(X_i-\mu_x)^T)}{n} \end{aligned} \tag{1}$$

The PCA algorithm has three main operations. It calculates the mean, μ_x, and covariance matrix, Cov_x, of the detected spike events, $X_1 \sim X_n$, and then calculates the leading eigenvectors, PC_{1-3}, of the covariance matrix.

The spike events, $X_1 \sim X_n$, are the input for both mean and covariance matrix operations. Next the covariance matrix operation must be performed right after the mean operation since the output of mean operation, μ_x, is the input of covariance matrix operation. Therefore, when a spike event is detected and input from the spike detector to calculate the mean vector, this spike event is required to be stored on chip for the subsequent covariance matrix operation. Suppose that one spike sample has 9-bit precision, 32 samples are used to represent one spike event, and up to 4096 spike events can be used for the PCA training procedure, a memory size of more than 1 M bits is required.

B. Proposed Mean Pre-estimation Method

In order to save the large memory for training spike events, we need to break the algorithm dependency between mean and covariance matrix operations without affecting the accuracy of PCA results. A mean pre-estimation method is proposed as shown in Fig. 3. The corresponding mathematic equations are listed as follows:

$$\begin{aligned} \mu_x &= \frac{\sum_{i=1}^{n} X_i}{n} \\ Cov_x &= \frac{\sum_{i=n+1}^{2n}((X_i-\mu_x)(X_i-\mu_x)^T)}{n} \end{aligned} \tag{2}$$

In the proposed algorithm, after the mean operation, we use the subsequent thousands of spike events, $X_{n+1} \sim X_{2n}$, for the covariance matrix operations. That is to say, the mean vector here is estimated from the previous thousands of spike events, $X_1 \sim X_n$. In this way, during the operations of both mean and covariance matrix, we can on-the-fly process the input spike event, accumulate the results, and then immediately throw away this input spike event. No large memory is required.

The proposed mean pre-estimation method should not affect the accuracy of the final PCA results. The characteristic shapes of the recorded spike events are determined by the morphology of neurons' dendritic trees and the distance and orientation relatives between neurons' and the recording electrode. These physical factors do not change within a period of time. Therefore, the mean vector of the previous n spike events should be very similar to the mean vector of the subsequent n spike events if n is large enough.

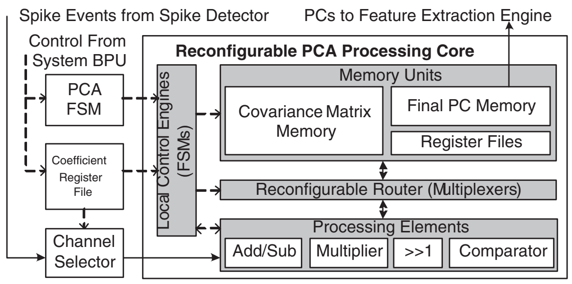
Spike Events from Spike Detector
PCs to Feature Extraction Engine
Control From System BPU
Reconfigurable PCA Processing Core
PCA FSM
Coefficient Register File
Channel Selector
Local Control Engines (FSMs)
Memory Units
Covariance Matrix Memory
Final PC Memory
Register Files
Reconfigurable Router (Multiplexers)
Processing Elements
Add/Sub
Multiplier
>>1
Comparator

Fig. 4. The block diagram of PCA training engine.

IV. Architecture Design

The block diagram of the PCA training engine is shown in Fig. 4. Before the training procedure, the BPU will program the essential training parameters such as the selected channel and the number of training spike events into the coefficient register file. Afterwards, the BPU will sent a start signal to PCA FSM to trigger the PCA training procedure. During the training procedure, the reconfigurable processing core calculates the mean vector, the covariance matrix, and the corresponding eigenvectors in turn. All intermediary and final PC results are stored in the memory units.

The algorithms to calculate the mean, the covariance matrix, and the eigenvector (with the modified algorithm proposed in [7]) have very similar mathematic operations. Therefore, a reconfigurable PCA processing core is designed to support all the essential operations with the same processing elements along with a reconfigurable router. In the rest of this section, we will show how this reconfigurable engine can support the mean and covariance matrix operations. As for the hardware configuration for the eigenvector operation, please refer to [7].

A. Mean Configuration

The processing flow to calculate the mean vector is shown in Fig. 5. When the start-mean signal is sent from PCA FSM, the hardware enters the listening mode waiting for new spike events. When a new spike event is detected, the hardware enters the accumulation mode. The detected spike event is accumulated in the register files as the intermediary result of the mean vector. The input spike event is threw away right after the accumulation. After the 2^m spike events are input and processed, the hardware enters the division mode. The right shift operation is performed m times for each sample of the mean vector in the register files. Finally, the result of mean vector is stored in the register files for the subsequent covariance matrix operation. Note that “m” can be programmed as 2 to 14 by the BPU during the hardware initialization.

B. Covariance Matrix Configuration

The processing schedule to calculate the covariance matrix is shown in Fig. 6. Before the processing, the register file should have a mean vector, μ_x, averaging from the previous 2^m spike events. When the start-covariance-matrix signal is sent by PCA FSM, the hardware enters the listening mode waiting for new spike events. When a new spike event is detected, the detected spike event is first subtracted by the mean vector. The resultant column vector, Y_i, is stored back to the register file. The input spike event can be threw away after the subtraction. Then, the

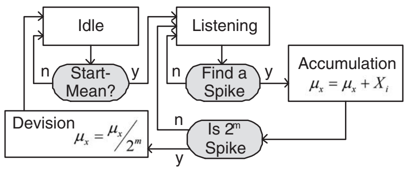

Fig. 5. The processing schedule to calculate the mean vector with the reconfigurable PCA processing core.

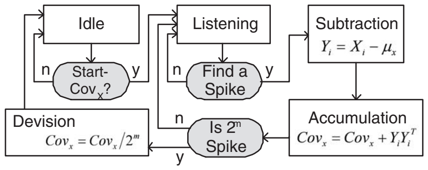

Fig. 6. The processing schedule to calculate the covariance matrix with the reconfigurable PCA processing core.

column vector, Y_i, is multiplied by its transposed vector, Y_i^T. The resultant square matrix is accumulated in the covariance matrix memory. Y_i can be erased from the register file after the accumulation. After the 2^m spike events are input and processed, the hardware enters the division mode. The right shift operation is performed m times for each entry of the covariance matrix. Finally, the resultant covariance matrix is stored in the covariance matrix memory for the subsequent eigenvector generation.

V. Simulation and Implementation Results

In the beginning of this section, it is necessary to summarize specifications of the proposed hardware. In our PCA training engine, the precision of the input spike sample and the output PC sample is nine bits. 32 samples are used to represent one spike event and PC. Up to 16384, 2^{14}, spike events can be used for one PCA training procedure. The hardware has the memory capability to store 48 trained PCs for 16 channels.

A. SNR Simulation of Mean Pre-Estimation Method

The proposed mean pre-estimation algorithm is verified with the neural data download from [13]. The simulation is done with the hardware C model. There are more than 3096 spike events, named X_1 to X_{3096}, in each neural sequence. We set “m” to 10 and perform PCA training algorithm for the spike events of $X_{2049} \sim X_{3096}$. The resultant mean and the first three PCs are used as the benchmark. Two experimental cases are compared. In the first case, we use the spike events of $X_{1025} \sim X_{2048}$ for mean estimation. In the second case, the spike events of $X_1 \sim X_{1024}$ are used for mean estimation. Note that the NEO spike detection algorithm [9] is used in this experiment. After spike detection, the detected spike events are aligned horizontally and vertically according to their peaks. 16 samples are used before and after the spike peak to represent each spike event. Table I summarizes the signal-to-noise ratio (SNR), or signal-to-error power ratio, of the mean vector and final PCs between the benchmark and two experimental cases. It clearly shows that the proposed mean pre-estimation method preserves more than 100 dB SNR and does not affect the accuracy of the PCA training results.

B. Memory Requirement Comparison

The memory requirement for the original PCA algorithm and the proposed method are summarized in Table II. For the

TABLE I

SNR Between the Benchmark and Two Experimental Cases of the Proposed Mean Pre-estimation Method

(Unit: dB)

	Sequence #	#01	#02	#03	#04	#05	#06	#07	#08	#09	#10	#11	#12	Average
Case#1	Mean	70.3	56.4	65.1	83.8	81.0	62.9	90.2	78.4	75.3	76.1	95.1	75.4	75.8
	PC#1	200.7	180.4	221.3	143.0	134.6	123.6	135.5	129.7	121.9	127.9	155.4	120.4	149.5
	PC#2	138.2	134.9	133.8	211.9	143.2	81.5	132.1	116.6	97.9	113.7	116.0	105.8	127.1
	PC#3	119.2	97.8	133.4	149.9	136.3	80.8	102.7	119.3	95.3	90.6	125.6	96.2	112.3
Case#2	Mean	73.1	74.5	73.3	88.5	84.1	73.8	90.3	88.4	75.2	79.0	82.0	74.4	79.7
	PC#1	146.9	111.6	136.0	141.4	121.7	92.3	151.2	102.3	121.1	146.8	163.8	124.9	130.0
	PC#2	159.5	110.5	119.8	120.2	117.7	80.6	161.7	98.4	147.3	146.2	160.0	131.2	129.4
	PC#3	90.7	97.0	165.9	102.3	124.4	81.8	100.2	115.9	103.8	69.8	125.1	126.7	108.6

Note: neural sequences #01–12 are C_Easy1_noise005–015, C_Easy2_noise005–015, C_Difficult1_noise005–015, C_Difficult2_noise 005–015 from [13].

TABLE II

Memory Size Comparison between Original Algorithm and Proposed Mean Pre-estimation Method

On-chip Storage Requirement (bits)	Original Algorithm	Proposed Method
Spike Events	4718592	0
Mean Vector	736	736
Covariance Matrix	32768	32768
Final PC results	13824	13824
Total	4765920	46592
Estimated Area (mm^2)	285.96	2.80

Note: 60 $\mu m^2/bit$ is used in the area estimation.

original algorithm, to store all training spike events requires $2^{14} \times 32 \times 9$ bits memory. To store all PC results for 16 channel requires $3 \times 16 \times 32 \times 9$ bits memory. The maximum precisions for intermediary values of mean vector and covariance matrix are 23 and 32 bits respectively during the accumulation. Therefore, the required memory sizes are 32×23 and $32 \times 32 \times 32$ bits. With the proposed algorithm, the memory to store the training spike events is saved. The unit area for each memory bit is about 60 μm^2 in .35 μm 2P4M CMOS process. As the result, the proposed mean pre-estimation method can save 283.16 mm^2 (99.01%) memory area and enable the low cost implementation for on-chip PCA training engine.

C. Implementation Result

We implement the proposed on-chip PCA training engine in verilog language, verify it with the test patterns generated by hardware C model, and synthesize the hardware in .35 μm 2P4M CMOS process. The chip is operated with 640 kHz operation frequency and 5 V supply voltage. According to the synthesized results, totally 6.07 mm^2 core area is required. Table III summarizes the area profile of this on-chip PCA training engine.

VI. Conclusion

In this paper, a mean pre-estimation method is proposed to ease the large memory requirement of the on-chip PCA training engine. 99.01% memory to store training spike events is saved by breaking the algorithm dependency. According to the simulation result, 100 dB signal-to-error power ratio can be preserved for the final PCA results. According to the implementation result, 6.07 mm^2 silicon area is required after a 283.16 mm^2 area saving with the proposed method.

TABLE III

Area Profile of the PCA Training Engine

Module Block	Area (mm^2)	Ratio (%)
PCA FSM	0.037	0.62
MMR Register File	0.053	0.88
Channel Selector	0.148	2.43
Local FSMs	0.127	2.10
Covariance Matrix Memory (1 of 1024x32-b SRAM)	1.874	30.88
Trained PC Memory (3 of 512x9-b SRAMs)	1.133	18.67
Register Files	1.214	20.01
Reconfigurable Router and Processing Elements	1.481	24.41
Total	6.067	100.00

References

[1] Z. Zumsteg et al., “Power feasibility of implantable digital spike sorting circuits for neural prosthetic systems,” *IEEE Trans. on Neural Systems and Rehabilitation Engineering*, vol. 13, no. 3, pp. 272–279, 2005.

[2] K. G. Oweiss et al., “Multispike train analysis,” in *Proc. of IEEE*, May. 1977, vol. 65, pp. 762–773.

[3] M.D. Linderman et al., “Signal processing challenges for neural prostheses,” *IEEE Signal Processing Magazine*, vol. 25, no. 1, pp. 18–28, 2008.

[4] J. C. Letelier et al., “Spike sorting based on discrete wavelet transform coefficients,” *Journal of Neuroscience Methods*, vol. 101, no. 2, pp. 93–106, 2000.

[5] E. Hulata et al., “A method for spike sorting and detection based on wavelet packets and shannon's mutual information,” *Journal of Neuroscience Methods*, vol. 117, no. 1, pp. 1–12, 2002.

[6] K. Shenoy, G. Santhanam, S. Ryu, A. Afshar, B. Yu, V. Gilja, M. Linderman, R. Kalmar, J. Cunningham, C. Kemere, A. Batista, M. Churchland, and T. Meng, “Increasing the performance of cortically controlled prostheses,” in *Proc. 28th Annu. Conf. IEEE Engineering in Medicine and Biology Society*, Aug. 2006, pp. 6652–6656.

[7] T.-C. Chen et al., “Vlsi architecture of leading eigenvector generation for on-chip principal component analysis spike sorting system,” *to appear, Proc. of IEEE conferenc on Engineering in Medicine and Biology Society*.

[8] A. Sharma et al., “Fast principal component analysis using fixed-point algorithm,” *Pattern Recognition Letters*, vol. 28, no. 10, pp. 1151–1155, 2007.

[9] K. H. Kim et al., “Neural spike sorting under nearly 0-db signal-to-noise ratio using nonlinear energy operator and artificial neural-network classifier,” *IEEE trans. on Biomedical Engineering*, vol. 47, no. 10, pp. 1406–1411, 2000.

[10] M. Chae et al., “A 128-channel 6mw wireless neural recording ic with on-the-fly spike sorting and uwb transmitter,” in *Proc. of IEEE International Solid-State Circuits Conference)*, Feb. 2008, vol. 7, pp. 146–147.

[11] M. Chae et al., “Design optimization for integrated neural recording systems,” *to appear, IEEE Journal of Solid-State Circuits*.

[12] T.-C. Chen et al., “Neusort2.0: A multiple-channel neural signal processor with systolic array buffer and channel-interleaving processing schedule,” *to appear, Proc. of IEEE conferenc on Engineering in Medicine and Biology Society*.

[13] R. Quian Quiroga, “http://www2.le.ac.uk/departments/engineering/extranet/research-groups/neuroengineering-lab/software,” .