VLSI Architecture of Leading Eigenvector Generation for On-chip Principal Component Analysis Spike Sorting System

Tung-Chien Chen[1,2], Wentai Liu[1,3] and Liang-Gee Chen[2]
[1]University of California, Santa Cruz, CA, USA; [2]National Taiwan University, Taipei, Taiwan;
[3]National Chiao-Tung University, Hsinchu, Taiwan; Email: djchen@video.ee.ntu.edu.tw

***Abstract*— On-chip spike detection and principal component analysis (PCA) sorting hardware in an integrated multi-channel neural recording system is highly desired to ease the bandwidth bottleneck from high-density microelectrode array implanted in the cortex. In this paper, we propose the first leading eigenvector generator, the key hardware module of PCA, to enable the whole framework. Based on the iterative eigenvector distilling algorithm, the proposed flipped structure enables the low cost and low power implementation by discarding the division and square root hardware units. Further, the proposed adaptive level shifting scheme optimizes the accuracy and area trade off by dynamically increasing the quantization parameter according to the signal level. With the specification of four principal components/channel, 32 samples/spike, and nine bits/sample, the proposed hardware can train 312 channels per minute with 1MHz operation frequency. 0.13 mm^2 silicon area and 282μW power consumption are required in 90 nm 1P9M CMOS process.**

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 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]–[4]. 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 and classification of individual neuron signal sources. The principal component analysis (PCA) [1], [2] and wavelet transformation [3], [4] are the most widely used tools for this approach. In this paper, we propose to achieve the first hardware prototype for PCA-based approach.

The PCA-based spike sorting has two major parts—the parameter training and the on-the-fly feature extraction. After spike detection, the training collects the detected spike waveforms and extracts the major characteristic vectors, the principal components (PCs), that can optimally differentiate neurons in least square terms. Calculating the covariance matrix and the leading eigenvectors are two major mathematic operations in this part. In the on-the-fly feature extraction, the feature scores are extracted by projecting the detected spike waveforms on the PCs, and the operation of inner product is required. Note that periodic re-training is frequently required because of the movement of the electrodes and change of the environment [5].

TABLE I
Description of iterative eigenvector distilling algorithm

1, Choose h, the number of eigenvectors required to estimate, Choose r, the iteration number of eigenvector distilling, Compute covariance matrix $\sum_{\mathrm{cov}}$, and set p and q to 1.
2, Initialize eigenvector φ_{p} up, e.g. randomly.
3, Do the eigenvector distilling process : $\varphi_{\mathrm{p}} = \sum_{\mathrm{cov}} \varphi_{\mathrm{p}}$
4, Do the Gram – Schmidt orthogonalization process : $\varphi_{\mathrm{p}} = \varphi_{\mathrm{p}} - \sum_{j=1}^{p-1} \left(\varphi_{\mathrm{p}}^{T} \varphi_{\mathrm{j}}\right) \varphi_{\mathrm{j}}$
5, Normalize up φ_{p} by dividing it by its norm : $\varphi_{\mathrm{p}} = \varphi_{\mathrm{p}} / \left\| \varphi_{\mathrm{p}} \right\|$
6, Increase counter $\mathrm{q} = \mathrm{q} + 1$ and go to step 3 until q equals r.
7, Increase counter $\mathrm{p} = \mathrm{p} + 1$ and go to step 2 until p equals h.

In realizing an on-chip PCA-based spike sorting circuit, the most challenging problem is to design a hardware engine to calculate leading eigenvectors. There are many algorithms to calculate eigenvectors from a covariance matrix, but most of them can hardly be mapped into an efficient VLSI architecture. In this paper, based on a new computationally fast and hardware friendly algorithm [6], the first VLSI architecture to calculate the leading eigenvectors is proposed. To facilitate the description, we name this algorithm as iterative eigenvector distilling algorithm. The reminder of this paper is organized as follows. In section II, the algorithm is briefly introduced. For the detailed mathematic proof of the algorithm, please refer to [6]. In section III, the low power and low area VLSI architecture is proposed with a flipped structure and adaptive level shifting method. Section IV presents the implementation and simulation results, and Section V concludes this work.

II. Iterative Eigenvector Distilling Algorithm

Table I depicts the fast PCA algorithm based on the iterative eigenvector distilling algorithm. "h" is the required number of the PCs. "r" is the algorithm iteration number, and "Σ_{cov}" is the covariance matrix calculated from the detected spike waveforms. In the beginning, the eigenvectors, "φ_p", are initialized randomly. Afterwards, the leading eigenvectors of the covariance matrix are calculated one by one in a reducing order of dominance. The calculation of each eigenvector

has r iterations and each of the iterations has two procedures-the eigenvector distilling process and the orthogonal process.

The key of this algorithm is to intensify the major component on the initial eigenvector through continuously multiplying the initial eigenvector with the covariance matrix. This procedure is called the eigenvector distilling process. The most PC can be simply derived after several iterations of this distilling process. For the remaining $h-1$ PCs, an additional orthogonal process is required. In order to continuously intensify the pth PC on the initial eigenvector, the previously measured $p-1$ components are removed from the intermediate results of φ_p by the orthogonal process after every iteration of distilling process. Note that the Gram-Schmidt method is used in our orthogonal process.

This algorithm has several advantages in terms of hardware implementation. First, this algorithm is free from eigenvalue decomposition, matrix diagonalization, symmetric rotation, and matrix inverse. Second, the algorithm exactly meets the requirement without calculating all the eigenvalues and the minor eigenvectors. Third, the algorithm can globally converge in a few iterations without the need for any specific initial setting. Also, the algorithm has a very regular procedure.

III. Architecture Design

A. Flipped Structure

In the original algorithm, four kinds of math operations are required—addition, multiplcation, division, and square root. Generally speaking, division and square root hardware units require much more silicon area and consume much more power compared with multipliers and adders. In order to optimize the power consumption and silicon area, the flipped structure is proposed to discard these hardware-expensive operations.

First, we discard the normalization process of $\varphi_p = \varphi_p / \|\varphi_p\|$, and change the orthogonal process to $\varphi_p = \varphi_p - (\varphi_p^T \varphi_j / \|\varphi_j\|)(\varphi_j / \|\varphi_j\|)$. Then, we multiply the whole equation by $\|\varphi_j\|^2$. Since $\|\varphi_j\|^2 = \left(\varphi_j^T \varphi_j\right)$, the orthogonal process finally becomes $\varphi_p = \left(\varphi_j^T \varphi_j\right)\varphi_p - (\varphi_p^T \varphi_j)\varphi_j$. After this transformation, the norm of the previously calculated PC is flipped to the dividend part in the orthogonal process. The division and square root operations are thus replaced by addition and multiplication. In this way, we can not only save the silicon area and power consumption but also increase the hardware utilization by reusing the uncomplicated processing units of the adders and the multipliers.

Note that this method works with an assumption that the scaling of the eigenvectors does not affect the final sorting performance. The new principal components are the scaled versions of the origonal normalized ones. The scaled PCs force the feature scores to be scaled equally for all spike waveforms. The classification algorithms such as K-means and mean-shift usually consider only the geographic relativity of these feature scores. Therefore, the scaling of the PCs does not affect the final sorting results.

```
for(p = 1 : 4)
{   φp = [ 1,  1,  ....,  1, ]
    for(i = 1 : 10)
    {   φp = Σcov φp
        Level_Check_And_Shift(φp)*
        for(j = 1 : p − 1)
        {   φp = (φj^T φj) φp − (φp^T φj) φj
            Level_Check_And_Shift(φp)*}}}

* Level_Check_And_Shift(φp) :
While(max(φp) ≥ 2^(bw−1) || min(φp) < −2^(bw−1))
{   φp = (φp + 1) >> 1   }
```

Fig. 1. Pseudo-code of the iterative eigenvector distilling algorithm with the flipped structure and adaptive level shifting scheme.

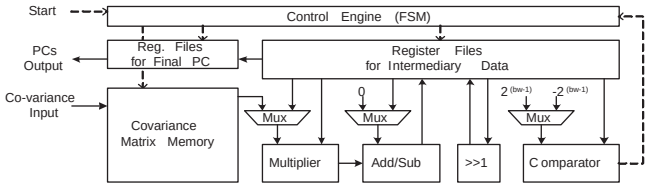

Fig. 2. The block diagram of the proposed architecture for the leading eigenvector generation.

B. Adaptive Level Shifting Scheme

After the Flipped structure, there is no division and square root operation, and the φ can be easily represented in a fixed-point integer number during the processing. It should be advantageous since the fixed-point integer DSP system is very friendly in terms of VLSI implementation. However, the dynamic range of φ increases rapidly during the iterations. For example, suppose the input covariance matrix is a 32×32 matrix, and each entry has 16-bit precision. The dynamic range of φ is increased by 16+5 bits for every eigenvector distilling process. If the current dynamic ranges of φ_j is n bits, the dynamic range of φ_p is increased by $2\times$n+5 bits for every orthogonal process. After several iterations, the final dynamic range become prohibitively large, which impedes a low area and low power implementation.

An adaptive level shifting scheme is proposed to optimize the hardware in terms of processing accuracy per hardware cost. The idea is to use the floating point concept in a fixed point DSP system. It is realized by dynamically increasing the quantization parameter according to the signal level until the limited bit-width can completely cover the quantized signals for each processing step. Fig. 1 shows the pseudo code of the proposed flipped structure combining with the proposed adaptive level shifting scheme. After either the eigenvector distilling process or the orthogonalization

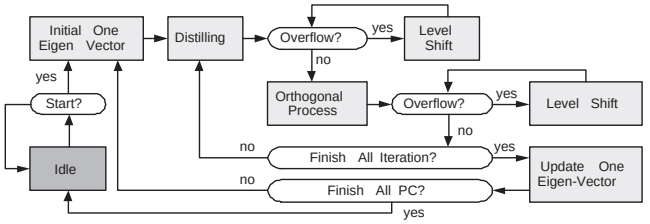

Fig. 3. Finite state machine in the control engine.

process, the level check and shift procedure is applied to compress the dynamic range according to the current level. The level check and shift procedure is shown in the bottom of Fig. 1. "bw" is the pre-defined bit-width of the system outputs of the final eigenvectors. During the level check and shift procedure, φ_p is continuously rounded by 2 until it can be completely represented in the pre-defined bit-width.

C. Architecture Design

Based on the modified algorithm, the block diagram of the proposed architecture for the leading eigenvector generation is shown in Fig. 2. The input is a covariance matrix calculated from the detected spike waveforms. The outputs are several leading eigenvectors of the covariance matrix, or the so-called PCs of the detected spike waveforms. Four major processing units are implemented in this architecture. The multiplier and adder (also used as a subtractor) units form a multiply-accumulate (MAC) structure and are used for the eigenvector distilling process and the orthogonalization process. The right-shift and comparator units are used for the level check and shift procedure. The whole algorithm is folded into these four processing units and processed sequentially. All the intermediary data are stored in the register files. The control engine mainly constructed of a finite state machine (FSM) is responsible for the scheduling and resource allocation during the processing.

After the architecture is constructed, the next challenge is how to do the scheduling and resource sharing. Fig. 3 shows the FSM of the control engine. Suppose each spike waveforms has n samples, and Σ_{cov} is an $n \times n$ matrix while φ is an $n \times 1$ vector. During the state of eigenvector distilling, "Σ_{cov}" and"φ_p" is input to MAC and the new "φ_p" is stored back to the register file. Note that every eigenvector distilling state takes $n \times n$ cycles. During the orthogonal process, "$\varphi_j^T \varphi_j$" is first computed, and both inputs of MAC are "φ_j". Afterwards, "φ_p" and "φ_j" are input to MAC for "$\left(\varphi_p^T \varphi_j\right)$". Then, "$\left(\varphi_j^T \varphi_j\right)$" and "$\varphi_p$" are input for "$\left(\varphi_j^T \varphi_j\right) \varphi_p$". As the final step in the orthogonal process, the MAC is initialized with "$\left(\varphi_j^T \varphi_j\right) \varphi_p$", and input with "$\left(\varphi_p^T \varphi_j\right)$" and "$\varphi_j$" in the subtraction mode. After that, the orthogonalization state is finished, and the "φ_j" component is totally removed from "φ_p". The result is also stored back to the register files. Note that the orthogonal process to remove each pre-calculated eigenvector, "φ_j", takes $n \times 4$ cycles. During the overflow checking state, "φ_p" is input to the comparator with $2^{(bw-1)}$ and $-2^{(bw-1)}$. The checking result is fed back to the control engine. If an overflow occurs, the FSM will enter the level shift state, and "φ_p" is input to the right shift engine to quantize the signal by 2. This procedure will continue until the overflow checking fails. It takes n cycles to pass each overflow checking and level shift state.

IV. Implementation Results

A. Implementation Results

We use 90 nm 1P9M process to implement the proposed leading eigenvector generator for various specifications with one MHz operation frequency. Table II reports the implementation results of different processing accuracies. The input bit-width specifies the precision of the given covariance matrix while the output bit-width specifies the precision of the required PCs. The sample number of spike waveforms is fixed to support as large as 32 samples while the PC number and iteration number can go up to four and 128 respectively. Note that if the input/output (I/O) bit-width is n, the maximum bit-width of internal circuit is $3 \times n + 5$ which happens after the orthogonal process. The size of the on-chip static random access memory is $32 \times 32 \times n$ bits to store the covariance matrix. When the bit width goes high, the area of covariance matrix memory, register files, and processing units increase in order to store and process more data. The hardware costs almost linearly increase in this case.

TABLE II

Results of Different Processing Accuracy

Spec.: Max. Bit Width		Entire Core		Covariance Memory Matrix	Processing Units + Register Files
I/O	Internal Circuit	Area (um^2)	Power (uW)	Area (um^2)	Area (um^2)
2	11	45996	93	17667	26465
4	17	70027	150	20859	47303
8	29	119956	255	29154	88937
16	53	222950	469	45744	175341

TABLE III

Results of Different Sample Number Per Spike Waveform

Spec.: samples per Spike Waveform	Entire Core		Covariance Memory Matrix	Processing Units + Register Files
	Area (μm^2)	Power (μW)	Area (μm^2)	Area (um^2)
16	79209	152	18135	59322
32	132021	282	31228	98929
64	255495	521	75481	178071

TableIII reports the implementation results of different sample numbers of spike waveforms. This time the I/O bit width is fixed to 9 bits. The PC number and iteration number can still go up to 4 and 128 respectively. If sample numbers of spike waveform is m, the size of the on-chip static random access memory is $m \times m \times 9$ bits. When the sample number of spike waveforms goes high, the dimensions of Σ_{cov} and φ increase. This fact increases the area of the covariance matrix memory and the register files. The area cost also linearly increases in this case.

Table IV shows the hardware capability of different hardware parameters. The processing capability is defined as the number of channels that can be trained in PCA algorithm within one minute. The number is reported for the worst case (which requires the maximum cycles for level checking and shifting) and with iteration number of 20, required principal number of 4, and 1MHz operation frequency. In the maximum specification of 64 samples per spike and 16 bit bit-width of each input spike sample and output eigenvector sample, our hardware can perform PCA analysis for 90 channels within one minute.

TABLE IV

THE HARDWARE CAPABILITY OF DIFFERENT HARDWARE PARAMETERS

Samples per Spike (#)	I/O Bit-width (bit)	Required Cycles for each Channel (#)	Channel Number per Minute (#)
64	16	666k	90
32	16	246k	244
32	9	192k	312
16	9	73k	822

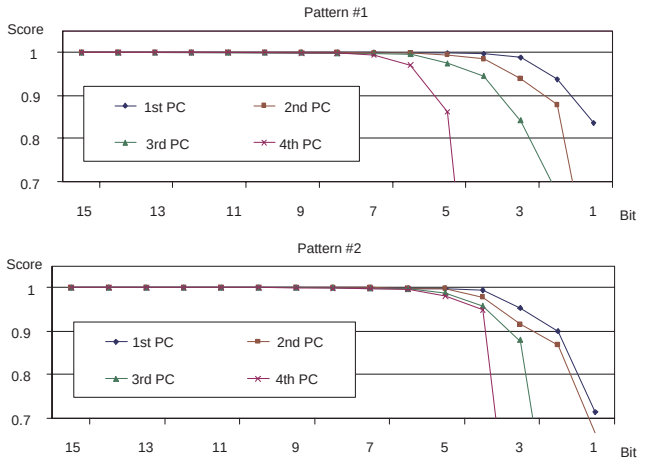

Fig. 4. The comparison between the principal components generated by Matlab function and our hardware Verilog model. The test patterns are downloaded form [7]. Pattern # 1 and # 2 are C_Easy1_noise005 and C_Difficult1_noise005.

B. Precision Analysis and Case Study

The precision analysis is made in order to establish the tradeoff between hardware cost and sorting performance. We use the neural data from [7] to validate our hardware. The Matlab "eig" function is used as our benchmark. Note that the nonlinear energy operation algorithm [8] is applied as the spike detection method, and we build a classification algorithm based on the watershed segmentation algorithm [9]. After spike detection, the detected spike waveforms are aligned horizontally and vertically according to their peaks and 8/12 samples are used before/after the peak to represent each spike waveform. The iteration number for each PC is set to 20 in this analysis.

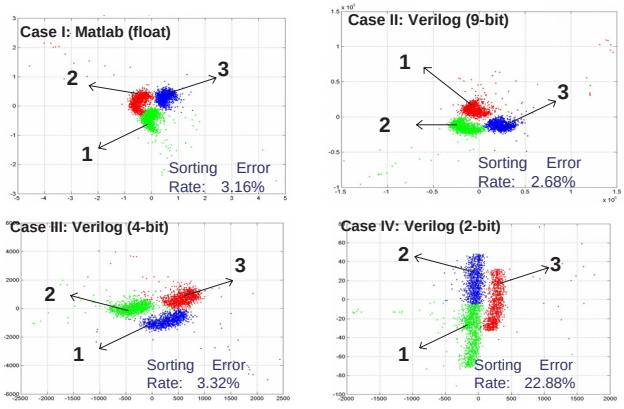

Fig. 5. Case studying using pattern #2. The first case uses Matlab function with floating point input neural data and eigen vectors, while the second to fourth cases use our hardware verilog model with 9-bit input neural data and 9-/4-/2-bit eigen vectors.

Fig. 4 shows the comparisons between the principal components generated by the Matlab function and our hardware verilog models. We use the correlation function, $\|\varphi_{Verilog}^{T}\varphi_{Matlab} \big/ norm(\varphi_{Verilog}) \times norm(\varphi_{Matlab})\|$, as the similarity score. The simulation results show that the hardware with 9-bit precision is a good trade-off point. For this precision, the hardware is minimized without affecting the accuracy of output PCs.

The sorting performances with Matlab function and our Verilog model are demonstrated in Fig. 5. Compared to the first case that uses Matlab function with the floating point data, the second case with 9-bit PC has almost the same 2-D feature map and sorting result. In the third case with 4-bit PC, the feature map has some distortion, but the sorting performance is still maintained. In the last case with 2-bit PC, the feature map is totally distorted, which ruins the sorting result.

V. CONCLUSION

In this paper, the first VLSI architecture for leading eigenvector generation was designed for the PCA-based spike sorting system. The iterative eigenvector distilling algorithm is used because of its simple and regular nature. The proposed flipped structure enables the low cost and low power implementation, while the adaptive level shifting scheme optimizes the accuracy and area trade off. According to the implementation results with specification of four PCs/channel, 32 samples/spike and 9bit/sample, the proposed hardware can train 312 channels per minute at 1MHz operation frequency and consumes 132k μm^2 silicon area and 282μW power in 90 nm 1P9M process.

REFERENCES

[1] Z. Zumsteg, C. Kemere, S. ODriscoll, G. Santhanam, R. Ahmed, K. Shenoy, and T. Meng, "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, D. J. Anderson, and M. M. Papaefthymiou, "Multispike train analysis," in *Proc. of IEEE*, May. 1977, vol. 65, pp. 762–773.

[3] J. C. Letelier and P. P. Weber, "Spike sorting based on discrete wavelet transform coefficients," *Journal of Neuroscience Methods*, vol. 101, no. 2, pp. 93–106, 2000.

[4] E. Hulata, R. Segev, and E. Ben-Jacob, "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.

[5] 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.

[6] A. Sharma and K. K. Paliwal, "Fast principal component analysis using fixed-point algorithm," *Pattern Recognition Letters*, vol. 28, no. 10, pp. 1151–1155, 2007.

[7] R. Quian Quiroga, "Simulated extracellular recordings," .

[8] K. H. Kim and S. J. Kim, "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.

[9] D. Wang, "Unsupervised video segmentation based on watersheds and temporal tracking," *IEEE trans. on Circuits System on Video Technology*, vol. 8, no. 7, pp. 539–546, 1998.