J Sign Process Syst (2016) 82:91–113
DOI 10.1007/s11265-015-0988-2

Cost-Effective and Variable-Channel FastICA Hardware Architecture and Implementation for EEG Signal Processing

Lan-Da Van · Po-Yen Huang · Tsung-Che Lu

Received: 1 July 2014 / Revised: 18 February 2015 / Accepted: 23 February 2015 / Published online: 21 March 2015

Abstract This paper proposes a cost-effective and variable-channel floating-point fast independent component analysis (FastICA) hardware architecture and implementation for EEG signal processing. The Gram-Schmidt orthonormalization based whitening process is utilized to eliminate the use of the dedicated hardware for eigenvalue decomposition (EVD) in the FastICA algorithm. The proposed two processing units, PU_1 and PU_2, in the presented FastICA hardware architecture can be reused for the centering operation of preprocessing and the updating step of the fixed-point algorithm of the FastICA algorithm, and PU_1 is reused for Gram-Schmidt orthonormalization operation of preprocessing and fixed-point algorithm to reduce the hardware cost and support 2-to-16 channel FastICA. Apart from the FastICA processing, the proposed hardware architecture supports re-reference, synchronized average, and moving average functions. The cost-effective and variable-channel FastICA hardware architecture is implemented in 90 nm 1P9M complementary metal-oxide-semiconductor (CMOS) process. As a result, the FastICA hardware implementation consumes 19.4 mW at 100 MHz with a 1.0 V supply voltage. The core size of the chip is 1.43 mm^2. From the experimental results, the presented work achieves satisfactory performance for each function.

Keywords Cost effective · EEG signal processing · FastICA algorithm · Gram-Schmidt · Hardware architecture · Variable channel

L.-D. Van (✉) · P.-Y. Huang · T.-C. Lu
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan 300, Republic of China
e-mail: ldvan@cs.nctu.edu.tw

1 Introduction

Electroencephalogram (EEG) research plays an important role for exploring the relation between human beings' behavior and brain [1, 2]. The EEG signals are the electrical potential recordings captured from the scalp sensors, where the electrical potentials mean that the components of the brain activity [3] are mixed. In addition, the EEG signals may be contaminated by the artifacts of eye activity or muscular activity [4–10]. Therefore, analyzing EEG signals becomes a challenge. In the brain research, the independent component analysis (ICA) algorithm is regarded as a useful method to study the brain activity through EEG signals [4–10]. The ICA algorithm is developed to solve the blind source separation (BSS) problem such that the mixed-up signals can be separated [11–19] and the brain activity information can be revealed.

Hardware architecture and implementation of ICA algorithms are challenges while considering cost, flexibility, speed and/or power. Many literatures have proposed the following ICA hardware implementations. Kim et al. [20] proposed a field-programmable gate array (FPGA) implementation of the 2-channel BSS constructed by the adaptive noise canceling (ANC) module. The 2-channel ANC model consumes 98.8 mW at 12.288 MHz at 1.8 V in FPGA. Du et al. [21–24] proposed parallel ICA algorithm and the corresponding FPGA implementation on a pilchard board, where several sub-processes run with multiple data parallelism. Jain et al. [25] proposed a parallel architecture for the FastICA algorithm. The floating-point arithmetic unit is adopted on pipelined FastICA design to increase the precision in [26]. One hardware efficient fixed-point FastICA is addressed in [27]. The fixed-point arithmetic is used on an FPGA-based INFOMAX ICA design [28]. Acharyya et al. [29] proposed a pioneering coordinate rotation digital computer (CORDIC)-based [30] *n*-dimensional FastICA algorithm and architecture by reusing the CORDIC module. Although

the proposed architecture [29] has low complexity characteristics by the algorithm analysis, the post-layout result of the proposed overall architecture is not available. Van et al. [31] proposed an energy-efficient eight-channel FastICA implementation with early determination scheme. The use of the dedicated eigenvalue decomposition (EVD) processor for eight-channel preprocessing in [31] adds the overhead to the hardware cost. Besides, it is predicted that the computational complexity for EVD based whitening process is largely increased when a higher channel number is requested [31]. Yang et al. [32] proposed a low-power eight-channel FastICA processor for epileptic seizure detection with fixed-point arithmetic. Roh et al. [33] proposed a 16-channel self-configured ICA implementation for the wearable neuro-feedback system.

The Gram-Schmidt [34] based whitening described in [12, 35, 36] has been pointed out that the Gram-Schmidt based whitening may be applied to the solution of BSS problem [35]. Concerning the computational complexity and hardware cost, in this paper, the Gram-Schmidt orthonormalization is applied in whitening process and fixed-point algorithm. Thus, it is expected that the computational complexity can be reduced and the hardware resource is possible for reuse. Note that the Gram-Schmidt orthonormalization has been adopted for the fixed-point iteration in the principle component analysis (PCA) algorithm to reduce the dimensions in [37]. To the best of our knowledge, the state-of-the-art dedicated hardware implementations do not adopt the Gram-Schmidt based whitening for the FastICA processing. On the other hand, for more flexibility, it is desired to support the variable channel selection and provide re-reference [1], synchronized average [1], and moving average [2] functions for EEG signal processing. Thus, we are motivated to propose a cost-effective FastICA architecture [38] that supports variable-channel FastICA operations and re-reference, synchronized average, and moving average functions. Herein, two reused processing units (PUs) are proposed to achieve cost-effective and variable-channel FastICA hardware implementation. The main contributions of this work are summarized as follows:

1) Propose a cost-effective and 2-to-16 channel floating-point FastICA architecture with two new reused PUs adopting Gram-Schmidt based whitening.
2) Implement the FastICA hardware architecture in the application-specific integrated circuit (ASIC) approach to support variable-channel FastICA, re-reference, synchronized average, and moving average functions.

As a result, the proposed FastICA architecture is implemented in the TSMC 90 nm 1P9M complementary metal-oxide-semiconductor (CMOS) process with the core size of 1.43 mm^2. The power consumption for 16-channel processing is 19.4 mW at 100 MHz.

The rest of this paper is organized as follows. The background for the FastICA algorithm is described in Section 2. The Gram-Schmidt orthonormalization based whitening is described and analyzed in Section 3. Next, the cost-effective and variable-channel FastICA architecture with the proposed PUs and the corresponding post-layout implementation are described in Section 4. In Section 5, we show the software and the corresponding post-layout simulation results for the validation of the proposed hardware implementation. Last, the conclusion and future work is remarked in Section 6.

2 Background for the FastICA Algorithm

2.1 Independent Component Analysis

Considering the number of blind source signals and the sample number are n and m, respectively, the BSS system can be modeled as (1) [26, 29, 31].

$$\mathbf{X} = \mathbf{AS} \tag{1}$$

where $\mathbf{X}$, $\mathbf{A}$, and $\mathbf{S}$ denote an n by m mixed-signal matrix, an n by n mixing matrix, and an n by m blind-source-signal matrix, respectively. $\mathbf{X}$ and $\mathbf{S}$ can be further expressed as

$$\mathbf{X} = [\mathbf{x}_1\, \mathbf{x}_2 \cdots \mathbf{x}_n]^T \tag{2}$$

$$\mathbf{S} = [\mathbf{s}_1\, \mathbf{s}_2 \cdots \mathbf{s}_n]^T \tag{3}$$

The mixed-signal vector $\mathbf{x}_i$ and the source-signal vector $\mathbf{s}_i$ can be respectively expressed in the following equations.

$$\mathbf{x}_i = [x_i(1)\, x_i(2) \cdots x_i(m)]^T, \text{ for } i = 1, 2, 3, \ldots, n, \tag{4}$$

$$\mathbf{s}_i = [s_i(1)\, s_i(2) \cdots s_i(m)]^T, \text{ for } i = 1, 2, 3, \ldots, n, \tag{5}$$

where $x_i(j)$ and $s_i(j)$ represent a mixed signal and a source signal at discrete time j, respectively. The ICA algorithm aims to find the matrix $\mathbf{S}$ without knowing the matrix $\mathbf{A}$, where the matrix $\mathbf{S}$ is assumed to have no more than one Gaussian-distributed source-signal vector. By observing the matrix $\mathbf{X}$, the inverse of the matrix $\mathbf{A}$ can be estimated as the weight matrix $\mathbf{W}^T$, and the blind source signal $\mathbf{S}$ can be obtained in (6) [31].

$$\mathbf{S} = \mathbf{W}^T\mathbf{X}, \tag{6}$$

where

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & & \vdots \\ w_{n1} & w_{n2} & & w_{nn} \end{bmatrix}. \quad (7)$$

The basic idea of ICA is to maximize the non-Gaussianity of $\mathbf{w}^T\mathbf{X}$, where $\mathbf{w}$ denotes a column vector of $\mathbf{W}$. Among the ICA algorithms, the FastICA algorithm proposed in [3, 13–16] has fast convergence rate and good performance on low signal-to-noise ratio (SNR) condition [8]. Therefore, the FastICA algorithm is chosen for the hardware implementation for EEG signal processing.

2.2 Preprocessing

In the FastICA algorithm, preprocessing is required to center and whiten the mixed signals. Through the preprocessing, the mixed signals can become zero mean and unit variance. The centering process can be expressed in (8).

$$\overline{x}_i(j) = x_i(j) - E\{x_i\}, \text{ for } i = 1, 2, 3, \ldots, n, \quad (8)$$

where $\overline{x}_i(j)$ and $E\{x_i\}$ denote the mixed signal value with zero mean and the expected value of the random variable $x_i(j)$ of vector $\mathbf{x}_i$ [31], respectively. Therefore, the centering matrix $\overline{\mathbf{X}}$ can be expressed in (9).

$$\overline{\mathbf{X}} = \begin{bmatrix} \overline{\mathbf{x}}_1^T \\ \overline{\mathbf{x}}_2^T \\ \vdots \\ \overline{\mathbf{x}}_n^T \end{bmatrix} = \begin{bmatrix} \overline{x}_1(1) & \overline{x}_1(2) & \cdots & \overline{x}_1(m) \\ \overline{x}_2(1) & \overline{x}_2(2) & \cdots & \overline{x}_2(m) \\ \vdots & \vdots & \cdots & \vdots \\ \overline{x}_n(1) & \overline{x}_n(2) & \cdots & \overline{x}_n(m) \end{bmatrix} \quad (9)$$

For the whitening process, EVD can be used to obtain unit-variance signals. Equation (10) can be obtained through the EVD on the covariance matrix $\mathbf{C}_\mathbf{x}$ of $\widetilde{\mathbf{x}}$, where $\widetilde{\mathbf{x}}$ denotes the random column vector of $\overline{\mathbf{X}}$.

$$\mathbf{C}_\mathbf{x} = E\{\widetilde{\mathbf{x}}\widetilde{\mathbf{x}}^T\} = \mathbf{E}\mathbf{D}\mathbf{E}^T, \quad (10)$$

where $\mathbf{E}$ denotes the eigenvector matrix of $\mathbf{C}_\mathbf{x}$ and $\mathbf{D}$ represents the eigenvalue matrix of $\mathbf{C}_\mathbf{x}$. $\mathbf{D}$ can be further expressed in (11) [31].

$$\mathbf{D} = diag(d_1, d_2, \cdots, d_n), \quad (11)$$

where $d_1, d_2, \ldots, d_n$ are the eigenvalues of $\mathbf{C}_\mathbf{x}$. The whitening process can then be written in (12) [31].

$$\mathbf{Z} = \mathbf{D}^{-1/2}\mathbf{E}^T\overline{\mathbf{X}} = \mathbf{P}\overline{\mathbf{X}} \quad (12)$$

After the whitening process, the covariance matrix of $\widetilde{\mathbf{z}}$ becomes the identity matrix $\mathbf{I}$ as shown in (13), where $\widetilde{\mathbf{z}}$ denotes the random column vector of $\mathbf{Z}$.

$$E\{\widetilde{\mathbf{z}}\widetilde{\mathbf{z}}^T\} = \mathbf{P}E\{\widetilde{\mathbf{x}}\widetilde{\mathbf{x}}^T\}\mathbf{P}^T = \mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{E}\mathbf{D}\mathbf{E}^T\mathbf{E}\mathbf{D}^{-1/2} = \mathbf{I} \quad (13)$$

Therefore, unit covariance of whitened signal can be guaranteed.

2.3 Fixed-Point Algorithm

The weight matrix is the key to find the independent component. The non-Gaussianity needs to be measured well in order to find the independence between vectors. In [15], the negentropy approximation is used to estimate the non-Gaussianity. With this approximation, the FastICA algorithm can find the maximum of non-Gaussianity by adjusting the weight matrix. For the training of the weight matrix, the fixed-point algorithm [15] is required. To estimate several independent components, either the symmetric orthogonalization or the deflationary orthogonalization can be utilized in the fixed-point algorithm [15]. The independent components are estimated in parallel for the fixed-point algorithm with the symmetric orthogonalization. On the other hand, the independent components are estimated one by one with the deflationary orthogonalization. In this work, we adopt the fixed-point algorithm with the symmetric orthogonalization such that the computation of the fixed-point algorithm can be parallelized in the hardware design. Assume $\overline{\mathbf{W}} = [\mathbf{w}_1\mathbf{w}_2\cdots\mathbf{w}_n]$, where $\mathbf{w}_i$, for i=1, 2, …, n, represents the column vector of $\overline{\mathbf{W}}$ [31], the fixed-point algorithm with the symmetric orthogonalization can be described as follows.

Step 1 Set the initial values for $\mathbf{w}_i$ with unit norm for i=1, 2, 3, …, n.

Step 2 Calculate the updating step [16], $\mathbf{w}_i^+ = E\{\widetilde{\mathbf{z}}[g(\mathbf{w}_i^T\widetilde{\mathbf{z}})]\} - E\{g'(\mathbf{w}_i^T\widetilde{\mathbf{z}})\}\mathbf{w}_i$, for i=1, 2, 3, …, n, where g is the derivative of the non-quadratic function $G(u) = \frac{1}{a}\log\cosh(au)$, where a is a constant parameter.

Step 3 Calculate the Gram-Schmidt orthonormalization for $\overline{\mathbf{W}}$ using (14), (15) and (16).

$$\mathbf{w}_1 = \frac{\mathbf{w}_1^+}{\|\mathbf{w}_1^+\|} \quad (14)$$

$$\mathbf{w}_{k+1}^\# = \mathbf{w}_{k+1}^+ - \sum_{i=1}^{k}\left[(\mathbf{w}_{k+1}^+)^T\mathbf{w}_i\right]\mathbf{w}_i, \text{ for } k = 1, 2, 3, \ldots, n-1 \quad (15)$$

$$\mathbf{w}_{k+1} = \frac{\mathbf{w}_{k+1}^{\#}}{\left\|\mathbf{w}_{k+1}^{\#}\right\|}, \text{ for } k = 1, 2, 3, \ldots, n-1 \quad (16)$$

Step 4 Go back to Step 2 if not converged.

Note that the symmetric orthogonalization is based on EVD in [15]. In this work, we adopt the Gram-Schmidt orthonormalization for the symmetric orthogonalization based fixed-point algorithm [31] in Step 3. Finally, when the convergence is achieved, the maximum of the non-Gaussianity of $\overline{\mathbf{W}}^T\mathbf{Z}$ can be estimated.

3 Gram-Schmidt Orthonormalization Based Whitening for the FastICA Algorithm

As mentioned in the previous section, the EVD based whitening process is generally adopted in the FastICA algorithm. In [31], since whitening process and fixed-point algorithm apply CORDIC and multiplier as well as adder modules, respectively, a specially designed CORDIC-based EVD processor is needed for EVD based whitening process. However, the hardware cost of a dedicated computing unit and the computational complexity of the EVD based whitening will be large while the number of channels increases in [31]. The pioneering work in [29] can reduce the hardware cost by reusing the CORDIC module to perform both EVD based whitening and the fixed-point algorithm of FastICA. In this work, the concept of Gram-Schmidt based whitening process [12, 35, 36] is adopted for the FastICA algorithm. The Gram-Schmidt based whitening process has lower computational complexity than the conventional EVD based whitening does. In this section, we will analyze and compare the computational complexity of the conventional EVD based whitening process and the Gram-Schmidt based whitening process in this section.

3.1 Computational Complexity Analysis

The Gram-Schmidt orthonormalization based whitening process for the FastICA algorithm is described as follows. First, the Gram-Schmidt orthonormalization for the mixed signal values with zero mean are illustrated in (17), (18) and (19).

$$\mathbf{z}_1^+ = \frac{\overline{\mathbf{x}}_1}{\left\|\overline{\mathbf{x}}_1\right\|} \quad (17)$$

$$\mathbf{z}_{k+1}^{\#} = \overline{\mathbf{x}}_{k+1} - \sum_{i=1}^{k} \left(\overline{\mathbf{x}}_{k+1}^T \mathbf{z}_i^+\right)\mathbf{z}_i^+, \text{ for } k = 1, 2, 3, \ldots, n-1 \quad (18)$$

$$\mathbf{z}_{k+1}^+ = \frac{\mathbf{z}_{k+1}^{\#}}{\left\|\mathbf{z}_{k+1}^{\#}\right\|}, \text{ for } k = 1, 2, 3, \ldots, n-1 \quad (19)$$

Such process gives an orthogonal matrix $\mathbf{Z}^+ = [\mathbf{z}_1^+ \mathbf{z}_2^+ \cdots \mathbf{z}_n^+]^T$, where the vectors $\{\mathbf{z}_1^+ \mathbf{z}_2^+ \cdots \mathbf{z}_n^+\}$ are mutually orthogonal. The covariance matrix of $\widetilde{\mathbf{z}}^+$ is then derived in (20).

$$E\left\{\widetilde{\mathbf{z}}^+ (\widetilde{\mathbf{z}}^+)^T\right\} = diag\left(\frac{(\mathbf{z}_1^+)^T \mathbf{z}_1^+}{m}, \frac{(\mathbf{z}_2^+)^T \mathbf{z}_2^+}{m}, \cdots, \frac{(\mathbf{z}_n^+)^T \mathbf{z}_n^+}{m}\right) = \frac{1}{m}\mathbf{I}, \quad (20)$$

where $\widetilde{\mathbf{z}}^+$ denotes the random column vector of $\mathbf{Z}^+$. Since unit variance is required for the whitening process, it is necessary to scale the orthogonal matrix $\mathbf{Z}^+$ with a factor $m^{1/2}$ as follows.

$$\mathbf{z}_{k+1} = m^{1/2}\mathbf{z}_{k+1}^+, \text{ for } k = 0, 1, 2, \ldots, n-1 \quad (21)$$

Therefore, the covariance matrix of $\widetilde{\mathbf{z}}$ can equal the identity matrix $\mathbf{I}$.

The computational complexity is defined as the sum of multiplications, divisions and square roots. Tables 1 and 2 summarize the computational complexity of the whitening process based on EVD and Gram-Schmidt orthonormalization, respectively, where mul, div and sqrt denote the corresponding multiplication, division and square root. Since it is a black box for us to know the detailed implementation of EVD function call in MATLAB, EVD is assumed to be performed by cyclic Jacobi method [39] for computational complexity in Table 1, where $\mathbf{J}$ denotes an n by n Jacobi rotation matrix [39]. Note that, the shift and add operations are excluded in Tables 1 and 2 since they contribute less complexity compared to the multiplication, division, and square root operations.

In Table 1, since the covariance matrix $\mathbf{C_x}$ is an n by n symmetric matrix, only $n(n+1)/2$ elements are required for the calculation of $\mathbf{C_x}$. Therefore, the computational complexity for the calculation of $\mathbf{C_x}$ is $m[n(n+1)/2]$ due to m multiplications for each element calculation of $\mathbf{C_x}$. The product of an n by n matrix and a Jacobi rotation matrix costs $4n$ multiplications since only $2n$ elements are needed to calculate, where each element needs two multiplications. Thus, $4n[n(n\text{-}1)/2\text{-}1]$ multiplications are required for the matrix multiplication of $[n(n\text{-}1)/2]$ $\mathbf{J}$ matrices to obtain the eigenvector matrix $\mathbf{E}$. In Table 2, $\left(\overline{\mathbf{x}}_{k+1}^T \mathbf{z}_i^+\right)\mathbf{z}_i^+$ needs to be calculated for $i=1, 2, 3, \ldots, k$ and $k=1, 2, 3, \ldots, n\text{-}1$. Since the calculation of $\left(\overline{\mathbf{x}}_{k+1}^T \mathbf{z}_i^+\right)\mathbf{z}_i^+$ needs $2m$ multiplications, the computational complexity for the calculation of $\mathbf{z}_{k+1}^{\#}$ can be obtained as $2m\sum_{k=1}^{n-1} k$.

The bar charts in terms of the number of multiplications and the sum of divisions and square roots versus number of channels with $m=512$ are shown in Fig. 1a and b, respectively. As can be seen in Fig. 1a and b, the computational complexity is mainly dominated by the multiplications for both EVD and Gram-Schmidt based orthonormalization whitening process.

Table 1 Computational complexity of the EVD based whitening process.

Procedure	Computational complexity
Calculate covariance matrix $\mathbf{C_x} = E\{\widetilde{\mathbf{x}}\widetilde{\mathbf{x}}^T\}$	$\left\{m\left[\frac{n(n+1)}{2}\right]\right\}$ muls
Calculate eigenvector matrix $\mathbf{E}=\mathbf{J}_{12}\mathbf{J}_{13}\cdots\mathbf{J}_{1n}\mathbf{J}_{23}\cdots\mathbf{J}_{n-1,n}$	$\left\{4n\left[\frac{n(n-1)}{2}-1\right]\right\}$ muls
Calculate eigenvalue matrix $\mathbf{D}=\mathbf{E}^T\mathbf{C_x}\mathbf{E}$	$(2n^3)$ muls
Calculate whitening matrix $\mathbf{P}=\mathbf{D}^{-1/2}\mathbf{E}$	(n^2) muls+(n) divs+(n) sqrts
Calculate whitened signal $\mathbf{Z} = \mathbf{P}\overline{\mathbf{X}}$	(mn^2) muls
Total	$\left[4n^3+\left(\frac{3}{2}m-1\right)n^2+\left(\frac{m}{2}-4\right)n\right]$ muls +(n) divs+(n) sqrts

As a result, the Gram-Schmidt orthonormalization based whitening process shows less computations than the EVD based whitening process does, especially when the channel number increases. Obviously, the computations for the eigenvalue matrix **D** and eigenvector matrix **E** are not required for the Gram-Schmidt based whitening. Therefore, EVD is not necessary in the Gram-Schmidt based whitening process such that the CORDIC module for EVD processing is not required in the proposed hardware architecture. Instead, from the analysis, the Gram-Schmidt based whitening process can be realized by general computing modules such as addition, multiplication, and inverse-square-root modules. Since these computing modules can be shared in the fixed-point algorithm of FastICA, the hardware cost is expected to be saved.

3.2 Influence of the Whitening Process on the FastICA Algorithm

Besides the computational complexity, two whitening approaches mentioned in the previous subsection need to be explored to see the influence on the separation quality of the FastICA algorithm. Therefore, the MATLAB coding resource [40] of FastICA algorithm is modified/recoded and then simulated with the EVD based and the Gram-Schmidt orthonormalization based whitening processes, respectively. The random mixing matrix **A** and the uniformly distributed source-signal matrix **S** are used to generate the mixed-signal matrix **X**. The mixed-signal matrix **X** is then processed with the FastICA algorithm by above two whitening approaches. The estimated source-signal vectors are then compared with the source-signal vectors using the performance index defined in (22).

$$\text{Performance Index} = \frac{1}{n}\sum_{i=1}^{n} abs\_corr\_coef(\widehat{\mathbf{s}}_i, \mathbf{s}_i) \tag{22}$$

where $\hat{\mathbf{s}}_i$ and $abs\_corr\_coef(\hat{\mathbf{s}}_i,\mathbf{s}_i)$ denote the i-th estimated source-signal vector computed by the FastICA algorithm and the absolute correlation coefficient between $\hat{\mathbf{s}}_i$ and $\mathbf{s}_i$, respectively. The absolute correlation coefficient [41] is expressed in (23) using the notations in this paper.

$$abs\_corr\_coef(\widehat{\mathbf{s}}_i, \mathbf{s}_i) = \left|\frac{E\{(\widehat{s}_i - E\{\widehat{s}_i\})(s_i - E\{s_i\})\}}{\sigma_{\widehat{s}_i}\sigma_{s_i}}\right| \tag{23}$$

where σ denotes the standard derivation. Note that the order of the estimated source-signal vector is sorted by associating the estimated source-signal vector and the source-signal vector with the maximum absolute correlation coefficient in the simulation.

Figure 2 shows the average performance index and the average iteration number versus number of channels, where the sample number is set to 512. The average performance index and the average iteration number are obtained with 500 test rounds. The maximum iteration number is set to 511 in the simulation. As can be seen, the performance indices and iteration numbers with EVD based and Gram-Schmidt orthonormalization based whitening processes are close. That

Table 2 Computational complexity of the Gram-Schmidt orthonormalization based whitening process.

Procedure	Computational complexity
Calculate $\mathbf{z}_1^+ = \frac{\overline{\mathbf{x}}_1}{\|\overline{\mathbf{x}}_1\|}$	(m) muls+(m) divs+(1) sqrt
Calculate $\mathbf{z}_{k+1}^{\#} = \overline{\mathbf{x}}_{k+1} - \sum_{i=1}^{k}\left(\overline{\mathbf{x}}_{k+1}^T\mathbf{z}_i^+\right)\mathbf{z}_i^+$ for k=1, 2, 3, …, n-1.	$\left(2m\sum_{k=1}^{n-1}k\right)$ muls
Calculate $\mathbf{z}_{k+1}^+ = \frac{\mathbf{z}_{k+1}^{\#}}{\|\mathbf{z}_{k+1}^{\#}\|}$ for k=1, 2, 3, …, n-1.	$[m(n-1)]$ muls+$[m(n-1)]$ divs+$(n-1)$ sqrts
Calculate $\mathbf{z}_{k+1}=m^{1/2}\mathbf{z}_{k+1}^+$ for k=0, 1, 2, …, n-1.	(mn) muls
Total	(mn^2+mn) muls+(mn) divs+(n) sqrts

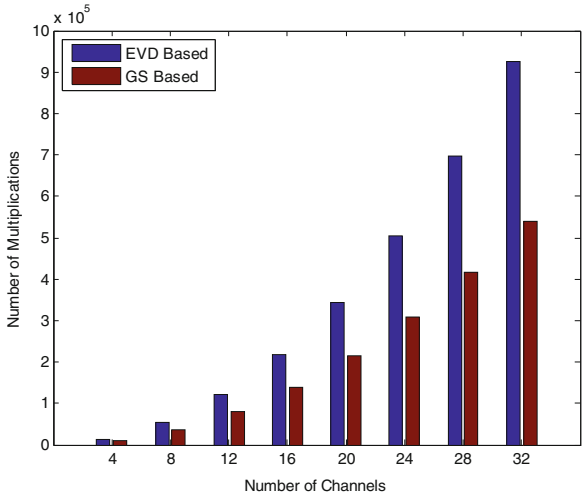


Figure 1 **a** Number of multiplications and **b** sum of divisions and square roots versus number of channels with EVD and Gram-Schmidt orthonormalization based whitening process.

means the Gram-Schmidt orthonormalization based whitening process can be an alternative way to replace the EVD based whitening process with similar separation quality in MATLAB simulation.

4 Cost-Effective and Variable-Channel FastICA Hardware Architecture and Implementation

In this section, the cost-effective floating-point FastICA hardware architecture and implementation for FastICA processing, re-reference, synchronized average and moving average functions are addressed. The reasonable sample size for the hardware implementation of the FastICA function is explored first. Then, the hardware architecture, hardware operations, and implementation of these four functions are revealed and illustrated.

4.1 Sample Size Analysis

Since FastICA is a statistical algorithm, increasing the sample size improves the separation quality. However, the memory size will become larger once the sample size increases. In addition, larger sample size will lead to larger computation complexity according to the discussion in Section 3. For the above reasons, the reasonable sample size for hardware implementation needs to be determined. Therefore, we simulate the average performance index versus the number of channels with the sample sizes of 256, 512, and 1024 in Fig. 3, where the simulation is performed by the double-precision floating-point (i.e., the most accurate precision) FastICA algorithm with the Gram-Schmidt orthonormalization based whitening process. The details of **A** and **X** in this section are the same as those of the simulation described in Section 3.2. From the simulation result, using 1024 samples can reach the best

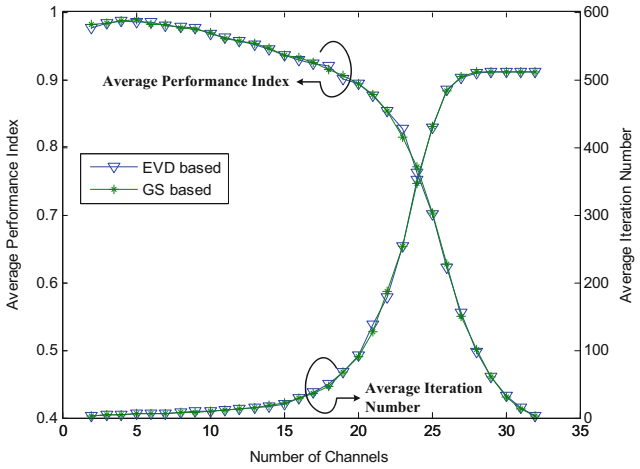


Figure 2 Average performance index and average iteration number versus number of channels with the whitening processes based on EVD and Gram-Schmidt orthonormalization.

performance among three different sample sizes, but the memory cost will become a penalty. On the other hand, although using 256 samples attains the lowest memory cost, the performance index is the worst among three different sample sizes. In summary, 512 is a tradeoff sample size for hardware implementation. To evaluate the effects of IEEE-754 single-precision floating-point and fixed-point arithmetic, the FastICA algorithm with the Gram-Schmidt orthonormalization based whitening process is performed with the 32-bit floating-point and 32-bit fixed-point simulations, respectively. The 11-bit fraction part of the fixed-point simulation is adopted for the random data and three datasets in Section 5. Figure 3 shows the performance index versus number of channels with floating-point and fixed-point simulation. It can be seen that the performance indices with 512 sample size manipulated by

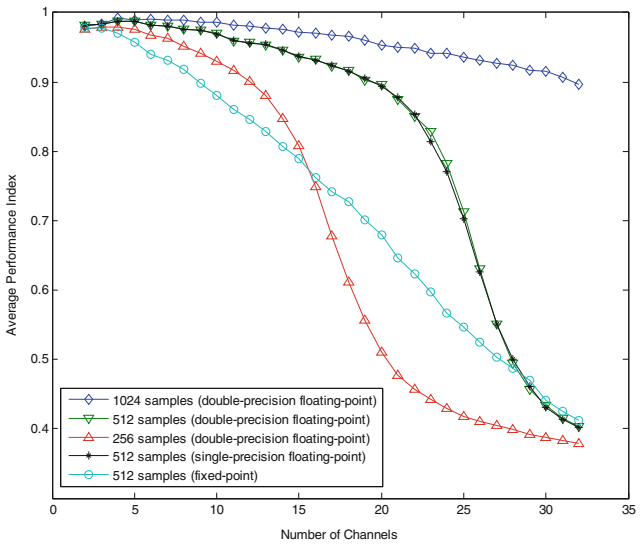


Figure 3 Performance index versus number of channels with different sample numbers.

double-precision and single-precision floating point are almost the same. For the targeted variable channel number with maximum of 16, the performance index with fixed-point simulation is lower than that of the floating-point simulation. It should be noted that the channel number should not be too high while considering an acceptable performance index. As a result, the maximum channel number, the sample size and the arithmetic format in our hardware implementation are set to 16, 512 and single-precision floating-point arithmetic format, respectively, according to the simulation result, where the details will be discussed in the next subsection.

4.2 Hardware Architecture

Figure 4 shows the system diagram of the proposed FastICA hardware architecture to realize the FastICA, re-reference, synchronized average and moving average functions. The proposed FastICA hardware architecture can perform different operations according to the instruction input, where the available instruction set is listed in Table 3 and is briefly described as follows. The LOAD instruction loads either fixed-point or floating-point data into the memory. The OUTPUT instruction outputs the processed data from the memory. The FASTICA instruction operates the FastICA algorithm. The REREF instruction produces the re-reference signal. The SYNAVG instruction offers the synchronized average signal. Finally, the MOVAVG instruction can make the proposed architecture act as a moving average filter for the input signal. In this work, the names of REREF and MOVAVG are the same as those in [42, 43] and MATLAB, respectively.

In the proposed hardware architecture, the IEEE-754 single-precision floating-point format in (24), where the notations are the same as those of [31], is adopted in the hardware design in order to preserve the computation accuracy.

$$L_{\text{float}} = (-1)^{s_L} \times (1.f_L) \times 2^{e_L - 127}, \tag{24}$$

where L_{float}, s_L, e_L and f_L represent the value of the IEEE-754 single-precision floating-point number, sign bit, 8-bit exponent bit and 23-bit fraction part of the mantissa, respectively. Since the input EEG signals will be in fixed-point format if they are captured by the scalp sensors and quantized by the analog-to-digital converter (ADC), the fixed-to-floating converter is utilized in this work to convert the fixed-point format into the IEEE-754 single-precision floating-point format. Afterwards, the input signals are stored at the data memory.

Herein, the processing units (PUs), PU_1 and PU_2, are utilized to process the data stored in the memory. The simplified architecture of the PUs is illustrated in Fig. 5. Each PU contains a multiply-and-add (MAA) unit, a temporary memory, and a division-by-2^{λ} operation unit, where the name of multiply-and-add (MAA) is the same as that in [44] and λ is a positive integer. In this work,

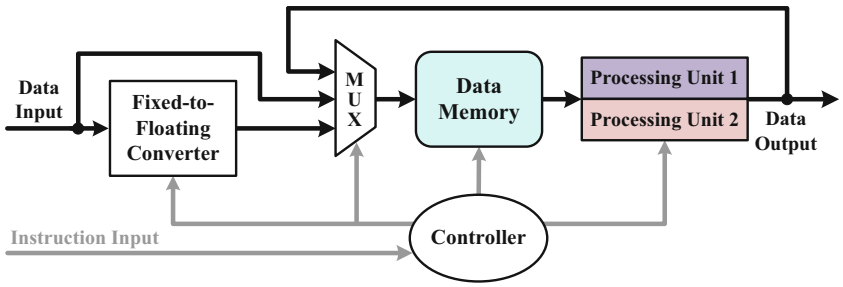


Figure 4 System diagram of the proposed FastICA hardware architecture.

PU_1 and PU_2 are reused in the centering operation of preprocessing and the updating step [16] (i.e., Step 2 of the fixed-point algorithm in Section 2.3) of the fixed-point algorithm. Through parallel manipulating the PU_1 and PU_2, the centering operation and the updating step can be accelerated. It is noted that, since the Gram-Schmidt orthonomalization processes the vectors sequentially, only PU_1 is reused for the Gram-Schmidt orthonomalization in preprocessing and fixed-point algorithm of the FastICA algorithm. Thus, PU_1 adopts the inverse-square-root unit based on the design of [45] for the Gram-Schmidt orthonomalization but PU_2 does not. The temporary memory is used to store the computation result such that the result can be reused in the subsequent computation or transferred to the data memory. Since the IEEE-754 single-precision floating-point format is utilized in the hardware design, the division-by-2^{λ} operation can be realized by subtracting λ for several functions as mentioned in Sections 4.3 and 4.5. For the FastICA function, PU_1 and PU_2 are reused to perform the centering operation of preprocessing and the updating step of the fixed-point algorithm, and PU_1 is reused for the computations of the Gram-Schmidt orthonormalization. Therefore, the dedicated EVD calculation unit is not required in the proposed hardware architecture. Other three function computations are realized by the two reused PUs in the proposed hardware architecture. Thus, the cost-effective hardware architecture can be attained. The hardware operations of the FastICA and other functions with the proposed PUs will be described in detail in the following subsections.

The 32-bit instruction format of the proposed hardware architecture is described in Table 4. The first 3 bits of the instruction represent the OP code for different operations. The remaining bits of the instruction are used to specify the user-defined parameters. The range of the OP code and the parameters are described in Table 5. Notably, the third parameter format of the FASTICA instruction is defined as the first 15 bits of IEEE-754 single-precision floating-point format. In order to simplify the hardware design, the first parameters can only be set to 2, 4, 8 or 16 for SYNAVG and MOVAVG instructions. Also, the second parameters of LOAD and OUTPUT can only be set to either 511 or 0 since they are used to specify the input data format or the output data type. With those functions and the user-defined parameters, the flexibility of the proposed hardware architecture can be attained.

For the chip implementation of the proposed FastICA hardware architecture, the cell-based design flow with the standard cell library in TSMC 90 nm 1P9M CMOS process is adopted. Artisan Memory Compiler is used to generate the memory module. Synopsys Design Compiler is employed to synthesize the proposed design with the timing constraint of 10 ns. Finally, Cadence SoC Encounter is used for the placement and routing procedure. Figure 6 shows the layout of the proposed FastICA hardware implementation, where the layout area occupies 1.43 mm^2.

Table 3 Instruction set of the proposed FastICA hardware architecture.

Instruction	Parameters	Operation
LOAD	Number of channels, input data type (fixed/floating)	Load data into the memory
OUTPUT	Number of channels, content type (weight/signal)	Output data from the memory
FASTICA	Number of channels, number of maximum iterations, threshold value	Perform FastICA algorithm
REREF	Number of channels, index of the baseline channel	Remove baseline
SYNAVG	Number of trails	Average all trails
MOVAVG	Number of samples on average, index of the target channel	Moving average filter

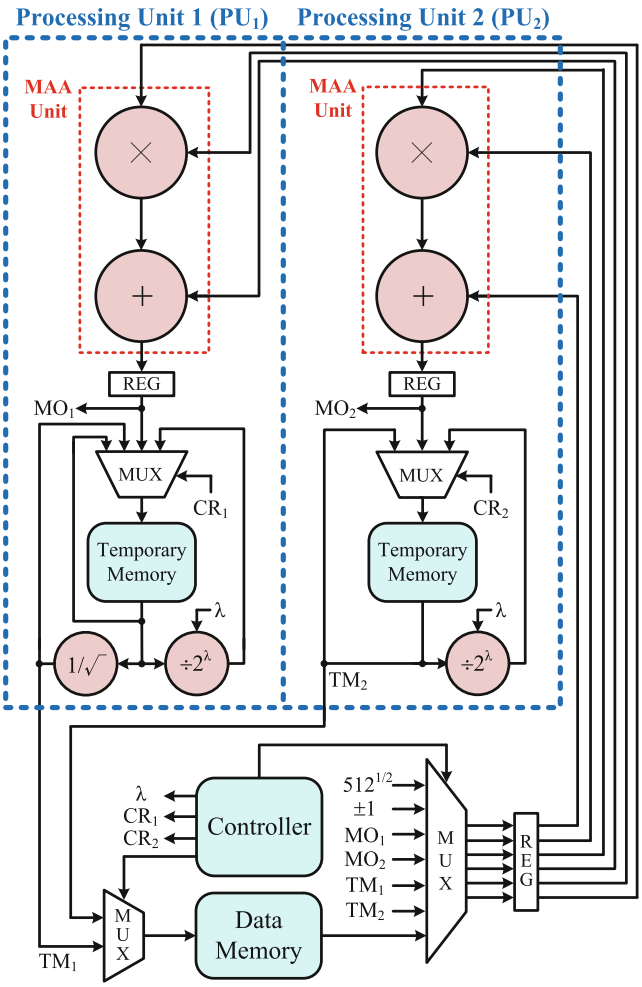


Figure 5 Simplified architecture of the processing units (*PUs*) and the corresponding controller.

4.3 Hardware Operations of the Preprocessing with Gram-Schmidt Orthonomalization Based Whitening Process of FastICA

To illustrate the hardware operations of the FastICA function using the PUs in Fig. 5, the computations of the preprocessing are described in this subsection. For the centering process, (8) can be recast in (25).

$$\bar{x}_i(j) = x_i(j) - \frac{1}{512}\sum_{l=1}^{512} x_i(l), \tag{25}$$

Table 4 Instruction format of the proposed FastICA hardware architecture.

OP Code	First parameter	Second parameter	Third parameter
3 bits	5 bits	9 bits	15 bits

Table 5 Range of the OP code and parameters of the instructions.

Instruction	OP code	First parameter	Second parameter	Third parameter
LOAD	0(000)	1~16	511/0 (fixed/floating)	–
OUTPUT	1(001)	1~16	511/0 (weight/signal)	–
FASTICA	2(010)	2~16	1~511	1~20
REREF	3(011)	1~16	0~15	–
SYNAVG	4(100)	2/4/8/16	–	–
MOVAVG	5(101)	2/4/8/16	0~15	–

where i=1, 2, …, n and j=1, 2, …, 512. The computation of (25) is described as follows. First, $x_i(l)$ are read from the data memory and used to calculate the result of $[\sum_{l=1}^{512} x_i(l)]/512$ using the MAA unit and the division-by-2^{λ} operation unit, where λ is set to 9. Next, the result of $x_i(j)$ minus $[\sum_{l=1}^{512} x_i(l)]/512$ is calculated on the MAA unit and saved at the data memory. The centering process can be either performed on the MAA unit of PU_1 or the MAA unit of PU_2. Therefore, PU_1 and PU_2 concurrently perform the computations of the centering process for different channels, respectively. For example, if n=16, $\bar{x}_1(j)$ and $\bar{x}_2(j)$ are calculated first with i=1 and i=2 in PU_1 and PU_2, respectively, for j=1, 2,…, 512. Next, $\{\bar{x}_3(j), \bar{x}_4(j)\}$, $\{\bar{x}_5(j), \bar{x}_6(j)\}$, …, $\{\bar{x}_{15}(j), \bar{x}_{16}(j)\}$ are calculated with i={3, 4}, i={5, 6}, …, i={15, 16} in PU_1 and PU_2 for j=1, 2, 3, …, 512, respectively.

After the centering process, (17), (18), (19) and (21) are used in the Gram-Schmidt orthonormalization based whitening process and can be recast as (26), (27) and (28), respectively.

$$\mathbf{z}_{k+1}^{\#} = \begin{cases} \bar{\mathbf{x}}_{k+1}, \text{ for } k = 0 \\ \bar{\mathbf{x}}_{k+1} - (\bar{\mathbf{x}}_{k+1}^T \mathbf{z}_1^+)\mathbf{z}_1^+ - \cdots - (\bar{\mathbf{x}}_{k+1}^T \mathbf{z}_k^+)\mathbf{z}_k^+, \text{ for } k = 1, 2, \ldots, n-1 \end{cases} \tag{26}$$

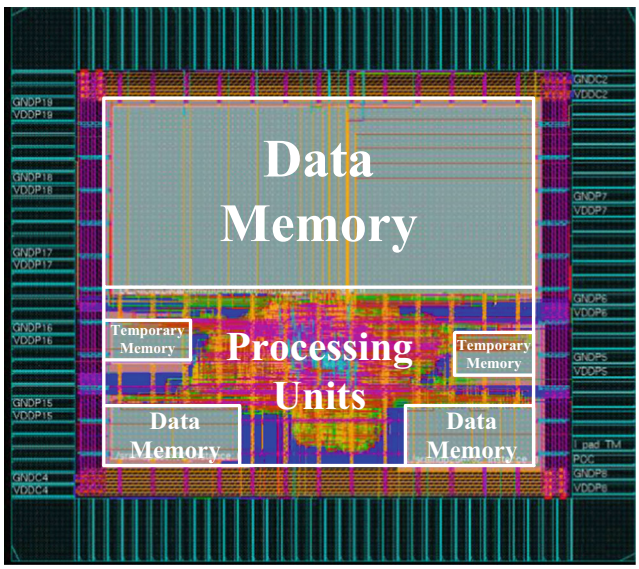


Figure 6 Layout of the proposed FastICA hardware implementation.

$$\mathbf{z}_{k+1}^{+}=\frac{\mathbf{z}_{k+1}^{\#}}{\left\|\mathbf{z}_{k+1}^{\#}\right\|}=\mathbf{z}_{k+1}^{\#}\left(1/\left\|\mathbf{z}_{k+1}^{\#}\right\|\right)=\mathbf{z}_{k+1}^{\#}\left[\left(\mathbf{z}_{k+1}^{\#}\right)^{T}\mathbf{z}_{k+1}^{\#}\right]^{-1/2} \tag{27}$$

$$\mathbf{z}_{k+1}=512^{1/2}\mathbf{z}_{k+1}^{+} \tag{28}$$

According to (26), (27) and (28), the computations of the Gram-Schmidt orthonormalization based whitening process are described as follows.

Step 1 Let i=1 and initially set $\mathbf{z}_{k+1}^{\#}=\overline{\mathbf{x}}_{k+1}$ at the temporary memory.
Step 2 Go to Step 6 if k=0. Otherwise, go to Step 3.
Step 3 Calculate $\overline{\mathbf{x}}_{k+1}^{T}\mathbf{z}_{i}^{+}$ on the MAA unit.
Step 4 Renew $\mathbf{z}_{k+1}^{\#}$ with the calculation result of $\mathbf{z}_{k+1}^{\#}-\left(\overline{\mathbf{x}}_{k+1}^{T}\mathbf{z}_{i}^{+}\right)\mathbf{z}_{i}^{+}$ using the MAA unit and increase i by 1.
Step 5 Repeat Step 3-Step 4 until $i>k$.
Step 6 Calculate $[(\mathbf{z}_{k+1}^{\#})^{T}\mathbf{z}_{k+1}^{\#}]^{-1/2}$ on the MAA unit and the inverse-square-root unit.
Step 7 Calculate $\mathbf{z}_{k+1}^{+}$ on the MAA unit.
Step 8 Repeat Step 1-Step 7 until n vectors $\{\mathbf{z}_1^{+},\mathbf{z}_2^{+},\cdots,\mathbf{z}_n^{+}\}$ of $\mathbf{Z}^{+}$ are obtained.
Step 9 Calculate $\mathbf{z}_{k+1}$ on the MAA unit. Repeat this step until n vectors $\{\mathbf{z}_1,\mathbf{z}_2,\cdots,\mathbf{z}_n\}$ of $\mathbf{Z}$ are obtained.

Note that the calculations of Step 1-Step 9 are only performed on the MAA unit of PU_1. In Step 1, $\overline{\mathbf{x}}_{k+1}$ is read from data memory. In Step 3, $\mathbf{z}_i^{+}$ is read from data memory to compute the dot product of $\overline{\mathbf{x}}_{k+1}$ and $\mathbf{z}_i^{+}$ on the MAA unit. In Step 4, the result of $\left(\overline{\mathbf{x}}_{k+1}^{T}\mathbf{z}_{i}^{+}\right)\mathbf{z}_{i}^{+}$ is calculated and subtracted from $\mathbf{z}_{k+1}^{\#}$ on the MAA unit. In Step 5, Step 3-Step 4 are repeated as $i\le k$ such that $\mathbf{z}_{k+1}^{\#}=\overline{\mathbf{x}}_{k+1}-\sum_{i=1}^{k}\left(\overline{\mathbf{x}}_{k+1}^{T}\mathbf{z}_{i}^{+}\right)\mathbf{z}_{i}^{+}$ can be obtained. In Step 6, the result of $(\mathbf{z}_{k+1}^{\#})^{T}\mathbf{z}_{k+1}^{\#}$ is calculated on the MAA unit and then processed with the inverse-square-root operation unit to obtain the result of $[(\mathbf{z}_{k+1}^{\#})^{T}\mathbf{z}_{k+1}^{\#}]^{-1/2}$. In Step 7, the result of $\mathbf{z}_{k+1}^{+}$ is calculated by multiplying $\mathbf{z}_{k+1}^{\#}$ and $[(\mathbf{z}_{k+1}^{\#})^{T}\mathbf{z}_{k+1}^{\#}]^{-1/2}$ on the MAA unit. In Step 8, Step 1-Step 7 are repeated to sequentially perform the calculations of n vectors $\{\mathbf{z}_1^{+},\mathbf{z}_2^{+},\cdots,\mathbf{z}_n^{+}\}$ of $\mathbf{Z}^{+}$ with k=0, 1, 2, …, n-1. In Step 8, when Step 1-Step 7 are done with k=n-1, the calculation of $\mathbf{Z}^{+}$ is completed. In Step 9, $\mathbf{z}_{k+1}$ is obtained by multiplying $512^{1/2}$ and $\mathbf{z}_{k+1}^{+}$ on the MAA unit. Step 9 is repeated until n vectors of $\mathbf{Z}$ are obtained. The result of $\mathbf{Z}$ is saved at the data memory. The proposed FastICA hardware architecture supports variable-channel processing, where the number of channels, n, is confined to 2-to-16 and defined by the user. In Step 8, the controller controls PU_1 to sequentially calculate $\{\mathbf{z}_1^{+},\mathbf{z}_2^{+},\mathbf{z}_3^{+},\cdots,\mathbf{z}_n^{+}\}$. While the controller detects that the last calculation of $\mathbf{z}_n^{+}$ is completed, the controller controls the PU_1 to execute the operation of Step 9. By setting different number of channels, n, the hardware operations of Gram-Schmidt orthonormalization can be realized on the same PU_1. For example, if n=16, $\mathbf{z}_1^{+}$ will be calculated first using Steps 1, 2, 6, and 7 with k=0. Next, $\mathbf{z}_2^{+}$ will be calculated through Step 1-Step 7 with k=1. Afterward, $\{\mathbf{z}_3^{+},\mathbf{z}_4^{+},\cdots,\mathbf{z}_{16}^{+}\}$ will be calculated sequentially with k=2, 3, …, 15, respectively. The vectors $\{\mathbf{z}_1,\mathbf{z}_2,\cdots,\mathbf{z}_{16}\}$ are calculated through Step 9 until $\{\mathbf{z}_1^{+},\mathbf{z}_2^{+},\cdots,\mathbf{z}_{16}^{+}\}$ are obtained. Similarly, if n=4, $\mathbf{z}_1^{+}$ will be calculated first and then $\{\mathbf{z}_2^{+},\mathbf{z}_3^{+},\mathbf{z}_4^{+}\}$ will be calculated sequentially.

4.4 Hardware Operations of the Fixed-Point Algorithm of FastICA

To implement the fixed-point algorithm using the two reused PUs, the equations can be recast in (29), (30) and (31).

$$\begin{aligned}\mathbf{w}_i^{+}&=E\{\widetilde{\mathbf{z}}[g(\mathbf{w}_i^{T}\widetilde{\mathbf{z}})]\}-E\{g'(\mathbf{w}_i^{T}\widetilde{\mathbf{z}})\}\mathbf{w}_i\\&=\frac{\sum_{j=1}^{512}\{\widetilde{\mathbf{z}}_j[g(\mathbf{w}_i^{T}\widetilde{\mathbf{z}}_j)]\}-\sum_{j=1}^{512}\{g'(\mathbf{w}_i^{T}\widetilde{\mathbf{z}}_j)\}\mathbf{w}_i}{512},\\&\cong\frac{\sum_{j=1}^{512}\{\widetilde{\mathbf{z}}_j[\alpha(\mathbf{w}_i^{T}\widetilde{\mathbf{z}}_j)+\beta]\}-\sum_{j=1}^{512}\{\alpha\}\mathbf{w}_i}{512}\end{aligned} \tag{29}$$

where $\widetilde{\mathbf{z}}_j$ denotes the j-th column vector of $\mathbf{Z}$, and α and β denote two parameters.

$$\mathbf{w}_{k+1}^{\#}=\begin{cases}\mathbf{w}_{k+1}^{+},\ \text{for}\,k=0\\ \mathbf{w}_{k+1}^{+}-\left[\left(\mathbf{w}_{k+1}^{+}\right)^{T}\mathbf{w}_1\right]\mathbf{w}_1-\cdots-\left[\left(\mathbf{w}_{k+1}^{+}\right)^{T}\mathbf{w}_k\right]\mathbf{w}_k,\ \text{for}\,k=1,2,\ldots,n-1\end{cases} \tag{30}$$

$$\mathbf{w}_{k+1}=\frac{\mathbf{w}_{k+1}^{\#}}{\left\|\mathbf{w}_{k+1}^{\#}\right\|}=\mathbf{w}_{k+1}^{\#}\left(1/\left\|\mathbf{w}_{k+1}^{\#}\right\|\right)=\mathbf{w}_{k+1}^{\#}\left[\left(\mathbf{w}_{k+1}^{\#}\right)^{T}\mathbf{w}_{k+1}^{\#}\right]^{-1/2} \tag{31}$$

The two non-linear functions $g(\mathbf{w}_i^{T}\widetilde{\mathbf{z}}_j)$ and $g'(\mathbf{w}_i^{T}\widetilde{\mathbf{z}}_j)$ in (29) are realized as $\alpha(\mathbf{w}_i^{T}\widetilde{\mathbf{z}}_j)+\beta$ and α, respectively, by the piecewise linear approximation method. The values of α and β are obtained by table look-up according to the value of $\mathbf{w}_i^{T}\widetilde{\mathbf{z}}_j$. Further discussions of the piecewise linear approximation method can be found in [38]. According to (29), (30) and (31), the computations of the fixed-point algorithm are described as follows.

Step 1 Let j=1 at the beginning.
Step 2 Calculate $\mathbf{w}_i^T\widetilde{\mathbf{z}}_j$ on the MAA unit.
Step 3 Calculate $\alpha\left(\mathbf{w}_i^T\widetilde{\mathbf{z}}_j\right)+\beta$ on the MAA unit.
Step 4 Calculate $\widetilde{\mathbf{z}}_j\left[\alpha\left(\mathbf{w}_i^T\widetilde{\mathbf{z}}_j\right)+\beta\right]$ on the MAA unit.
Step 5 Accumulate the results of $\widetilde{\mathbf{z}}_j\left[\alpha\left(\mathbf{w}_i^T\widetilde{\mathbf{z}}_j\right)+\beta\right]$ and α at the temporary memory using the MAA unit, respectively, and increase j by 1.
Step 6 Repeat Step 2-Step 5 until j>512.
Step 7 Calculate $\mathbf{w}_i^+$ on the MAA unit.
Step 8 Repeat Step 1-Step 7 until n vectors $\{\mathbf{w}_1^+, \mathbf{w}_2^+, \cdots, \mathbf{w}_n^+\}$ are obtained.
Step 9 Calculate the Gram-Schmidt orthonormalization for $\overline{\mathbf{W}}$.
Step 10 Repeat Step 1-Step 9 until $\overline{\mathbf{W}}$ is converged.

Note that the calculations of Step 1-Step 7 can be performed on the MAA unit in either PU_1 or PU_2. In other words, two $\mathbf{w}_i^+$ vectors can be concurrently calculated on the MAA units of PU_1 and PU_2, respectively. In Step 2, $\mathbf{w}_i$ and $\widetilde{\mathbf{z}}_j$ are read from the data memory to calculate their dot product on the MAA unit. In Step 3, the result of $\alpha\left(\mathbf{w}_i^T\widetilde{\mathbf{z}}_j\right)+\beta$ are calculated on the MAA unit. In Step 4, $\widetilde{\mathbf{z}}_j$ is read from the data memory and multiplied by $\alpha\left(\mathbf{w}_i^T\widetilde{\mathbf{z}}_j\right)+\beta$ on the MAA unit. In Step 5, the results of $\widetilde{\mathbf{z}}_j\left[\alpha\left(\mathbf{w}_i^T\widetilde{\mathbf{z}}_j\right)+\beta\right]$ and α are separately accumulated at the temporary memory using the MAA unit. In Step 6, Step 2-Step 5 are repeated as $j\leq 512$ to obtain $\sum_{j=1}^{512}\left\{\widetilde{\mathbf{z}}_j\left[\alpha\left(\mathbf{w}_i^T\widetilde{\mathbf{z}}_j\right)+\beta\right]\right\}$ and $\sum_{j=1}^{512}\{\alpha\}$. In Step 7, the result of $\sum_{j=1}^{512}\left\{\widetilde{\mathbf{z}}_j\left[\alpha\left(\mathbf{w}_i^T\widetilde{\mathbf{z}}_j\right)+\beta\right]\right\}$ minus $\sum_{j=1}^{512}\{\alpha\}\mathbf{w}_i$ is calculated on the MAA unit. The division-by-512 operation is not necessary in Step 7 since it does not affect the normalization result. In Step 8, Step 1-Step 7 are repeated to obtain n vectors $\{\mathbf{w}_1^+, \mathbf{w}_2^+, \cdots, \mathbf{w}_n^+\}$. According to the user-defined number of channels, n, the calculations of $\mathbf{w}_i^+$ vectors are repeated on both PU_1 and PU_2 to obtain all n vectors of $\mathbf{w}_i^+$ with i=1, 2, 3,…, n. For example, if n=16, $\mathbf{w}_1^+$ and $\mathbf{w}_2^+$ are calculated first with i=1 and i=2 in PU_1 and PU_2, respectively, through Step 1-Step 7. Next, $\{\mathbf{w}_3^+, \mathbf{w}_4^+\}$, $\{\mathbf{w}_5^+, \mathbf{w}_6^+\}$, …, $\{\mathbf{w}_{15}^+, \mathbf{w}_{16}^+\}$ are calculated with i=$\{3, 4\}$, i=$\{5, 6\}$, … and i=$\{15, 16\}$ in PU_1 and PU_2, respectively. In Step 9, the Gram-Schmidt orthonormalization is calculated for $\overline{\mathbf{W}}$. The calculation of the Gram-Schmidt orthonormalization is similar to Step 1-Step 8 of the whitening process in Section 4.3. In Step 10, the sum of absolute dot-products of the old weight vectors and the new weight vectors of $\overline{\mathbf{W}}$ is calculated on the MAA unit to determine whether $\overline{\mathbf{W}}$ is converged.

4.5 Implementation of the Re-reference, Synchronized Average and Moving Average Functions

In terms of the re-reference, synchronized average and moving average processing, the re-reference signal $y_{r,i}(j)$, the synchronized average signal $y_s(j)$, and the moving average signal $y_m(j)$ are defined in (32), (33), and (34), respectively.

$$y_{r,i}(j) = x_i(j) - x_{baseline}(j), \text{ for } j = 1, 2, 3, \ldots, m, \tag{32}$$

$$y_s(j) = \frac{1}{h}\sum_{trial=1}^{h} x_{trial}(j), \text{ for } j = 1, 2, 3, \ldots, m, \tag{33}$$

$$y_m(j) = \frac{1}{r}\sum_{k=0}^{r-1} x_{target\_channel}(j-k), \text{ for } j = 1, 2, 3, \ldots, m, \tag{34}$$

where i=1, 2, 3, …, n and h and r denote the number of trails and the number of samples on average, respectively. Since h and r are the first parameters of SYNAVG and

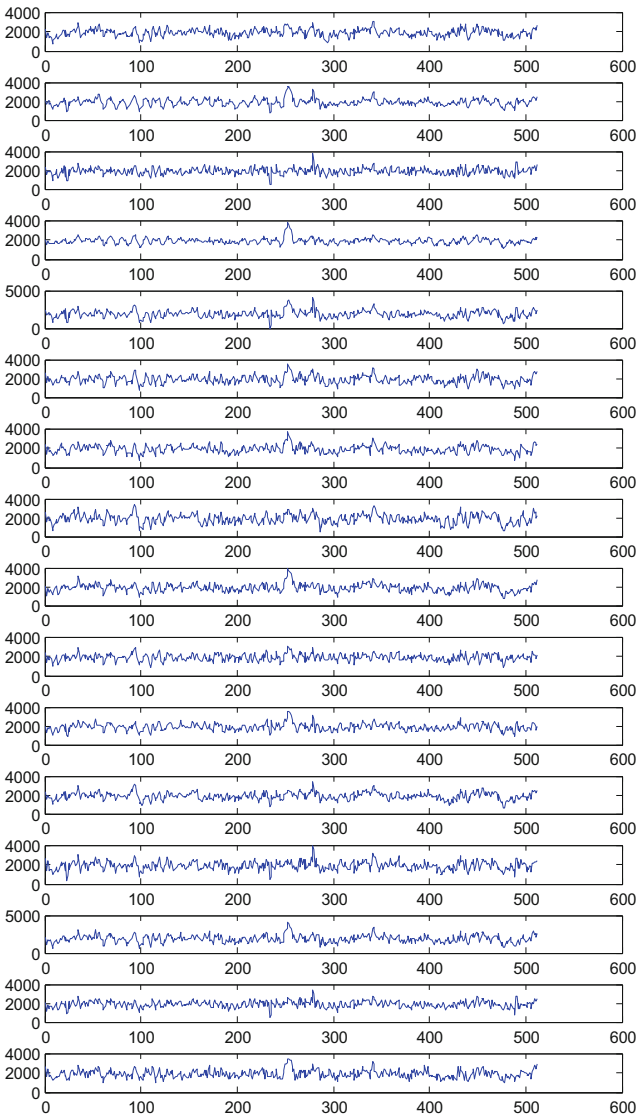


Figure 7 First dataset: 16-channel mixed signals generated by MATLAB and [42].

MOVAVG instructions, respectively, they can only be set to 2, 4, 8, or 16 in the proposed hardware architecture as mentioned in Section 4.2. Therefore, the division operation in (33) and (34) can be performed with the division-by-2^{λ} units in the PUs by setting λ to 1, 2, 3 or 4. The operations of addition and subtraction in (32), (33), and (34) are performed on the MAA units of PU_1 and PU_2. For the re-reference function, PU_1 and PU_2 concurrently perform the computations for different channels, respectively. For the synchronized average and moving average function, PU_1 and PU_2 concurrently perform the computations for different discrete time j, respectively.

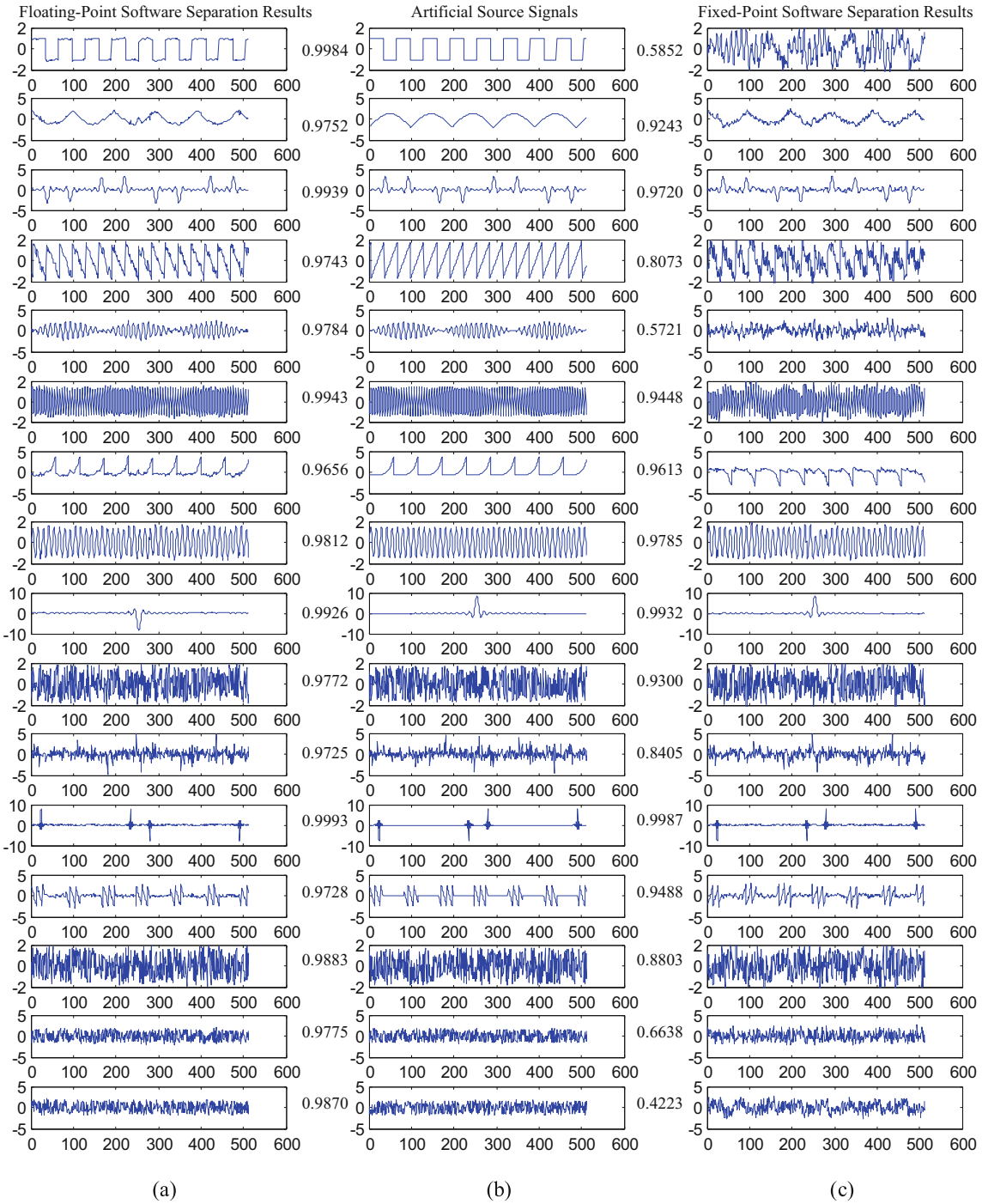


Figure 8 Comparison between (**a**) the floating-point software separation results, (**b**) the artificial source signals and (**c**) the fixed-point software separation results of the FastICA function for the first dataset.

5 Evaluation and Comparison Results

In this section, the double-precision floating point (i.e., the most accurate precision) software simulation in MATLAB is performed to verify the FastICA algorithm based on the Gram-Schmidt orthonormalization for whitening process and fixed-point algorithm. The software simulations of the re-reference, synchronized average and moving average functions are also performed in MATLAB. The hardware architecture is validated by the post-layout simulation results. Finally, the results are summarized and compared with other related works.

5.1 Software Simulation

To verify the separation quality with the FastICA algorithm based on the Gram-Schmidt orthonormalization for whitening process and fixed-point algorithm, the software simulation is performed. In our simulation, three datasets are adopted. The first dataset contains 16-channel artificial mixed signals which are obtained by randomly mixing the 16-channel artificial source signals in 12-bit fixed-point format as shown in Fig. 7. Note that the 16-channel artificial source signals contains two-channel sparse and very sparse source signals in [46] and other 14-channel source signals generated by MATLAB functions including abs, sin, square, rem, sinc, rand, log and randperm. Since the source signals in the first dataset are known, the software separation result can be compared to the source signals to determine the separation quality. Figure 8 shows the artificial source signals, the floating-point and fixed-point software separation results of the FastICA algorithm with the Gram-Schmidt orthonomalization based whitening process for the first dataset. The numbers between Fig. 8a and b denote the absolute correlation coefficients between the source signals and the floating-point separation results. On the other hand, the numbers between Fig. 8b and c denote the absolute correlation coefficients between the source signals and the fixed-point separation results. For the floating-point separation results, the average and minimum absolute correlation coefficients are 0.9830 and 0.9656, respectively. The signal-to-interference ratio (SIR) [47] results for 16-channel source signals and the software separation results with the first dataset are 25.0082, 13.0428, 19.1261, 12.8986, 13.6465, 19.4119, 11.6197, 14.2428, 18.2994, 13.4162, 12.5925, 28.7617, 12.6471, 16.2938, 13.4617, and 15.8396 dB, respectively. The average value of the SIR results is 16.2693 dB. As a result, the artificial mixed signals are well separated in the floating-point software simulation. For the fixed-point separation results, the average and minimum absolute correlation coefficients are 0.8389 and 0.4223, respectively. The average and minimum absolute correlation coefficients of the fixed-point separation results are lower than those of the floating-point simulation results.

The second dataset contains the 16-channel EEG signals which are adopted from the dataset in the EEGLAB toolbox [42, 43]. The details of the experiment were described in [6]. Figure 9 shows the 16-channel EEG signals which are captured from the positions of FPZ, F3, FZ, F4, C3, C4, CZ, P3, PZ, P4, PO3, POZ, PO4, O1, OZ, and O2 of the international 10–20 electrodes placement system. The 16-channel EEG signals are transformed to 12-bit fixed-point format with the sampling rate of 256 Hz and the windows size of two seconds. Figure 10a and b show the floating-point software simulation results and the fixed-point software simulation results for the second dataset, respectively. The numbers in the middle of Fig. 10 denote the absolute correlation coefficients between the floating-point simulation results and the fixed-point simulation results. For the floating-point simulation results, as can be seen in Fig. 10a, the independent components associated with the late positive event-related potential can be extracted

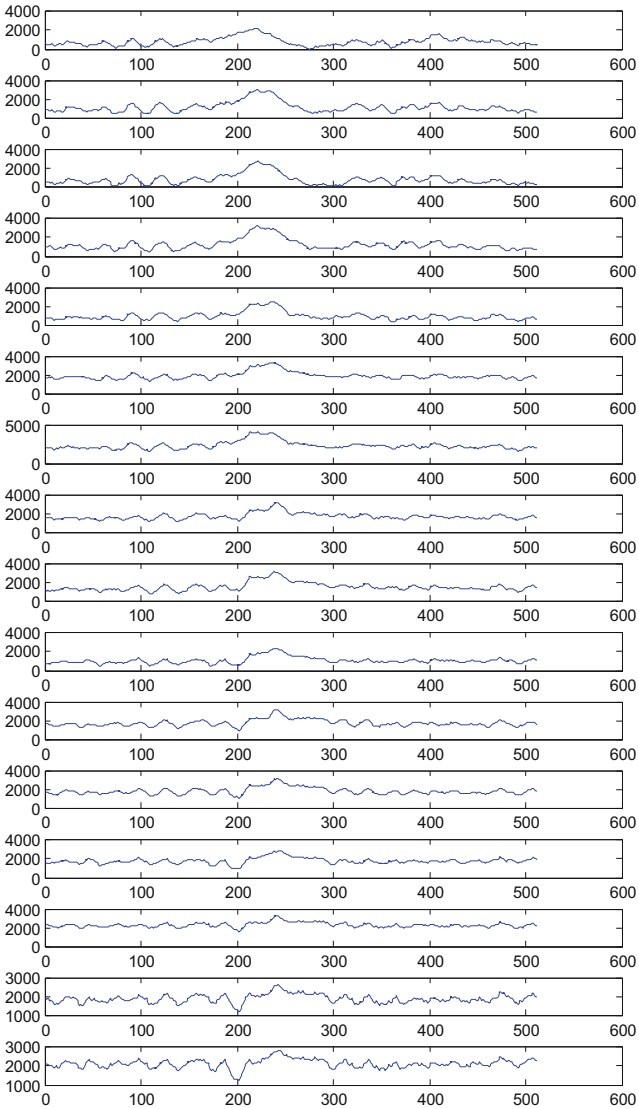


Figure 9 Second dataset: 16-channel EEG signals.

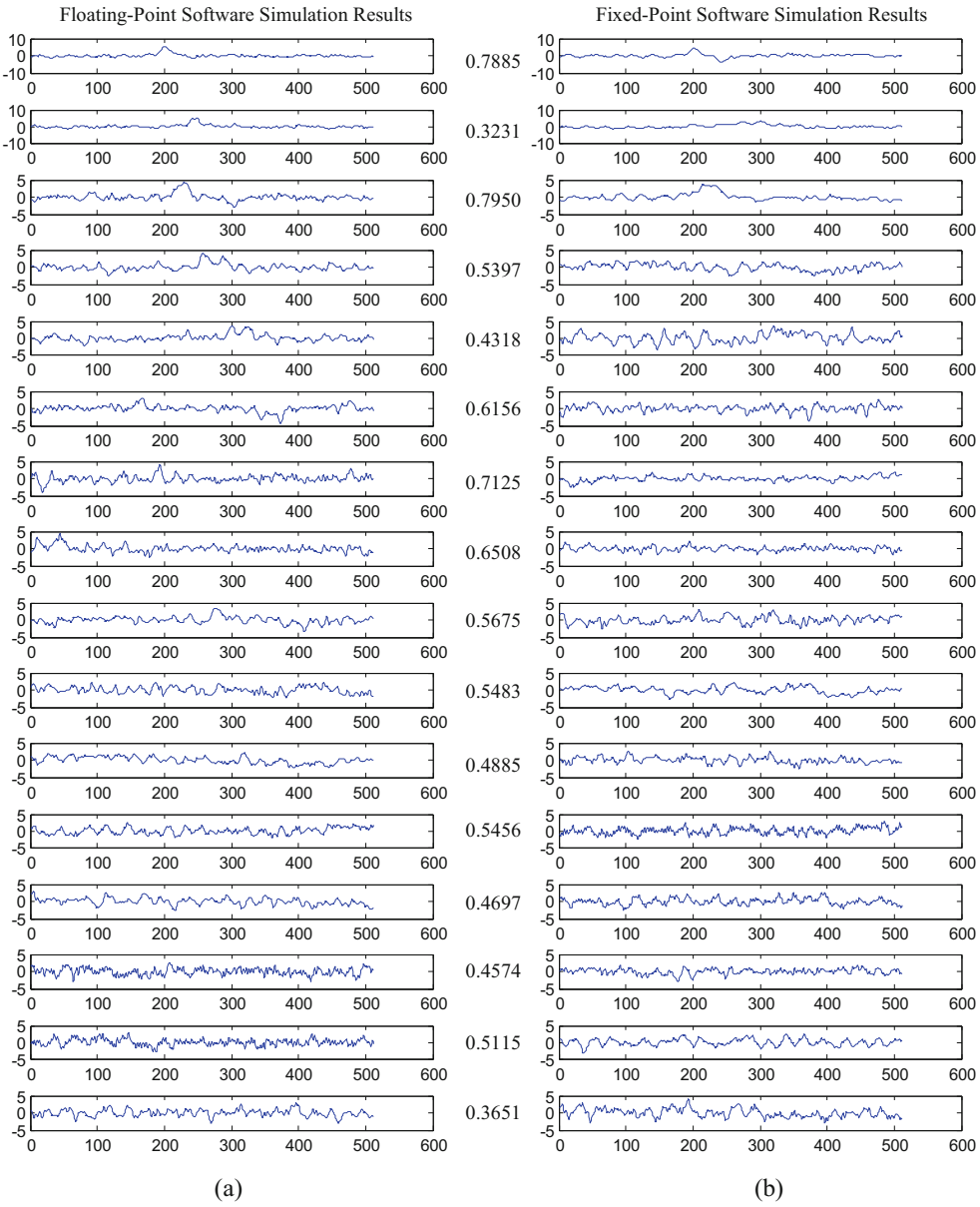


Figure 10 Software simulation result utilizing the FastICA algorithm with the Gram-Schmidt orthonormalization based whitening process for the second dataset in MATLAB: (**a**) floating-point simulation results and (**b**) fixed-point simulation results.

in the first three rows. The floating-point simulation results of Fig. 10a are similar to those reported in [6]. However, in Fig. 10b, one component cannot be easily observed in the second row. The average and minimum absolute correlation coefficients are 0.5507 and 0.3231, respectively.

The third dataset contains 4-channel EEG signals which have eye blinking artifact as shown in Fig. 11. The 4-channel EEG signals are captured from the positions of FP1, FP2, F3 and F4 of the international 10–20 electrodes placement system. The sampling rate and the format of the third dataset are the same as those of the second dataset. The software simulation results for the third dataset are shown in Fig. 12. As can be seen, the eye blinking component can be separated in the first row in Fig. 12. According to the software simulation results with the above three datasets, the validation

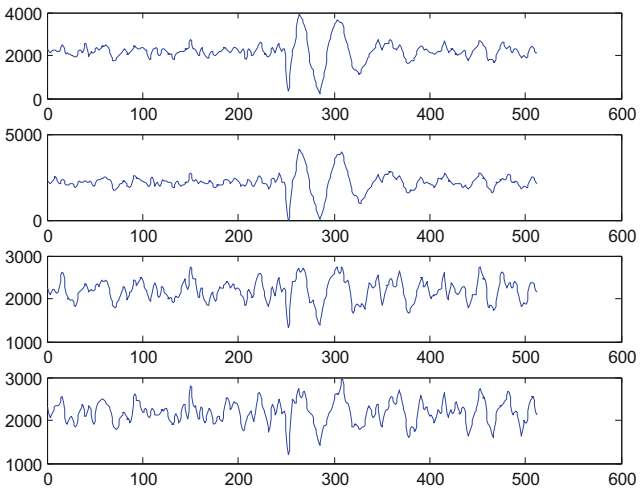


Figure 11 Third dataset: 4-channel EEG signals.

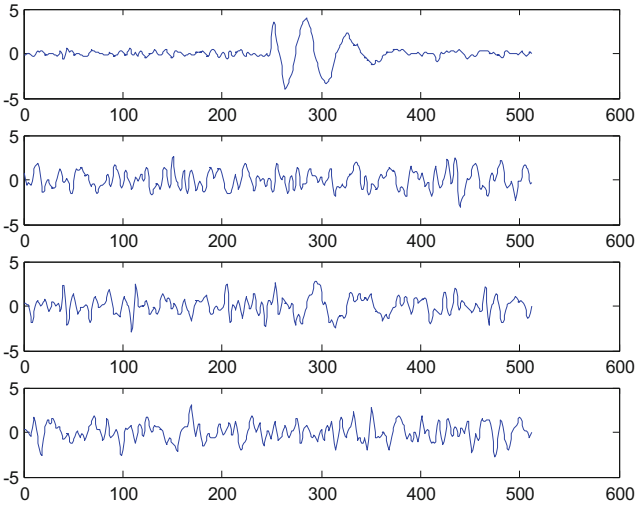


Figure 12 Software simulation result utilizing the FastICA algorithm with the Gram-Schmidt orthonormalization based whitening process for the third dataset in MATLAB.

of the FastICA algorithm with the Gram-Schmidt orthonormalization based whitening process can be achieved for both artificial mixed signals and EEG signals.

Figure 13 shows the software simulation result of the re-reference function, where the EEG signals at the positions of PZ and CZ in Fig. 9 are used for the input signal and the baseline signal, respectively. Figure 14 shows the software simulation result of the synchronized average function with 16 EEG trials captured from the position of CPZ. Finally, the EEG signal with high-frequency noise adopted from [46] and the corresponding software simulation result of moving average function are shown in Fig. 15. As can be seen, the high-frequency noise of EEG signal can be filtered after the moving average processing.

5.2 Post-Layout Simulation

In this subsection, the post-layout simulation is performed. The results of post-layout simulation are compared to the software simulation results done in the previous subsection to validate the function of the proposed cost-effective hardware implementation. For the FastICA function, the three datasets mentioned in the previous subsection are used for the post-layout simulation. Figures 16, 17 and 18 show the comparison results with the first dataset of the 16-channel artificial mixed

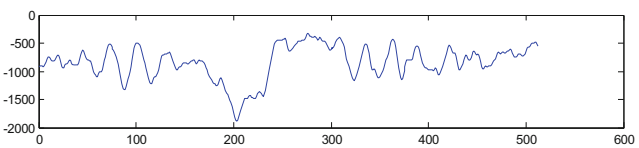
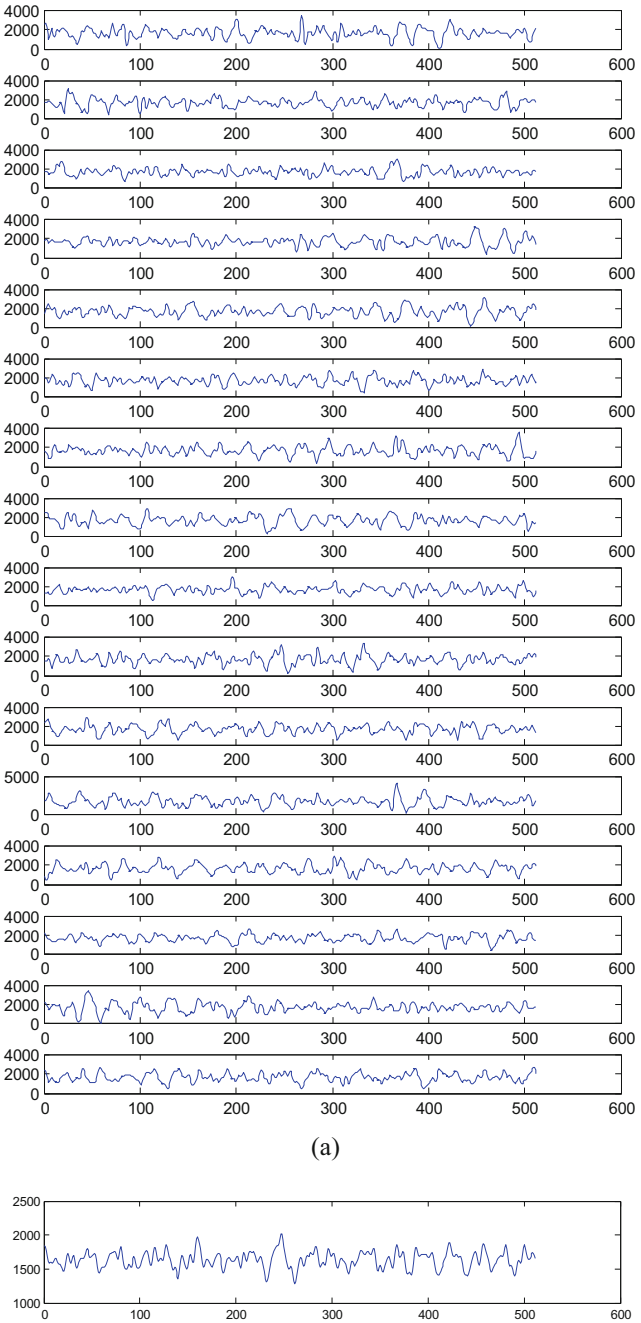


Figure 13 Software simulation result of the re-reference function in MATLAB.



(a)

(b)

Figure 14 **a** 16 trials of the EEG signals from the position of CPZ; **b** software simulation result of the synchronized average function in MATLAB.

signals in Fig. 7, the second dataset of the 16-channel EEG signals in Fig. 9 and the third dataset of the 4-channel EEG signals in Fig. 11, respectively. In Figs. 16, 17 and 18, the numbers in the middle of the figures denote the absolute correlation coefficients between the software simulation results and the post-layout simulation results. For the first dataset, the average and the minimum values of the absolute correlation

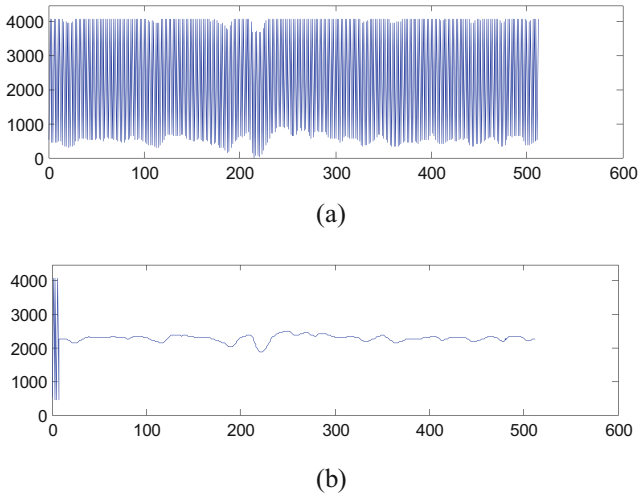


Figure 15 **a** EEG signal with high-frequency noise; **b** software simulation result of the moving average function in MATLAB.

coefficients are 0.9967 and 0.9852, respectively. For the second dataset, the average and minimum values of the absolute correlation coefficients are 0.9441 and 0.8424, respectively. For the third dataset, the average absolute correlation coefficient is 0.9868. Although the datasets in this work are different from those in [31], it should be noted that the minimum value of the absolute correlation coefficients of the second dataset (16-channel EEG signals), 0.8424, is near to that of the EEG signals in Fig. 17 of [31], 0.8499. According to the comparison results, the post-layout simulation result can attain enough accuracy compared to the software simulation results among three datasets.

Figure 19 shows the comparison result of the artificial source signals and the post-layout simulation results with the first dataset in Fig. 7, where the numbers in the middle of Fig. 19 denote the absolute correlation coefficients between

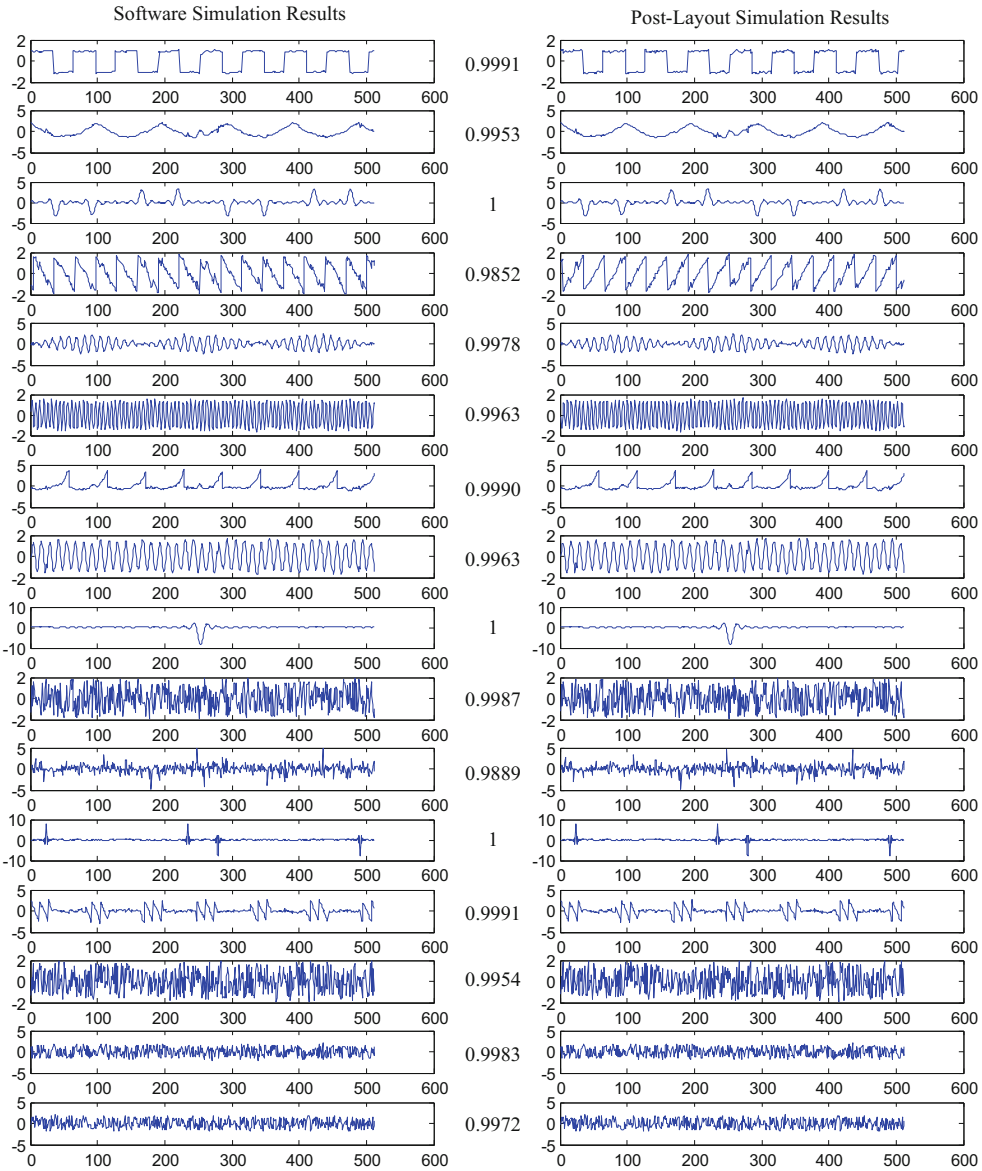


Figure 16 Comparison between the software simulation results and post-layout simulation results of the FastICA function for the first dataset in Fig. 7.

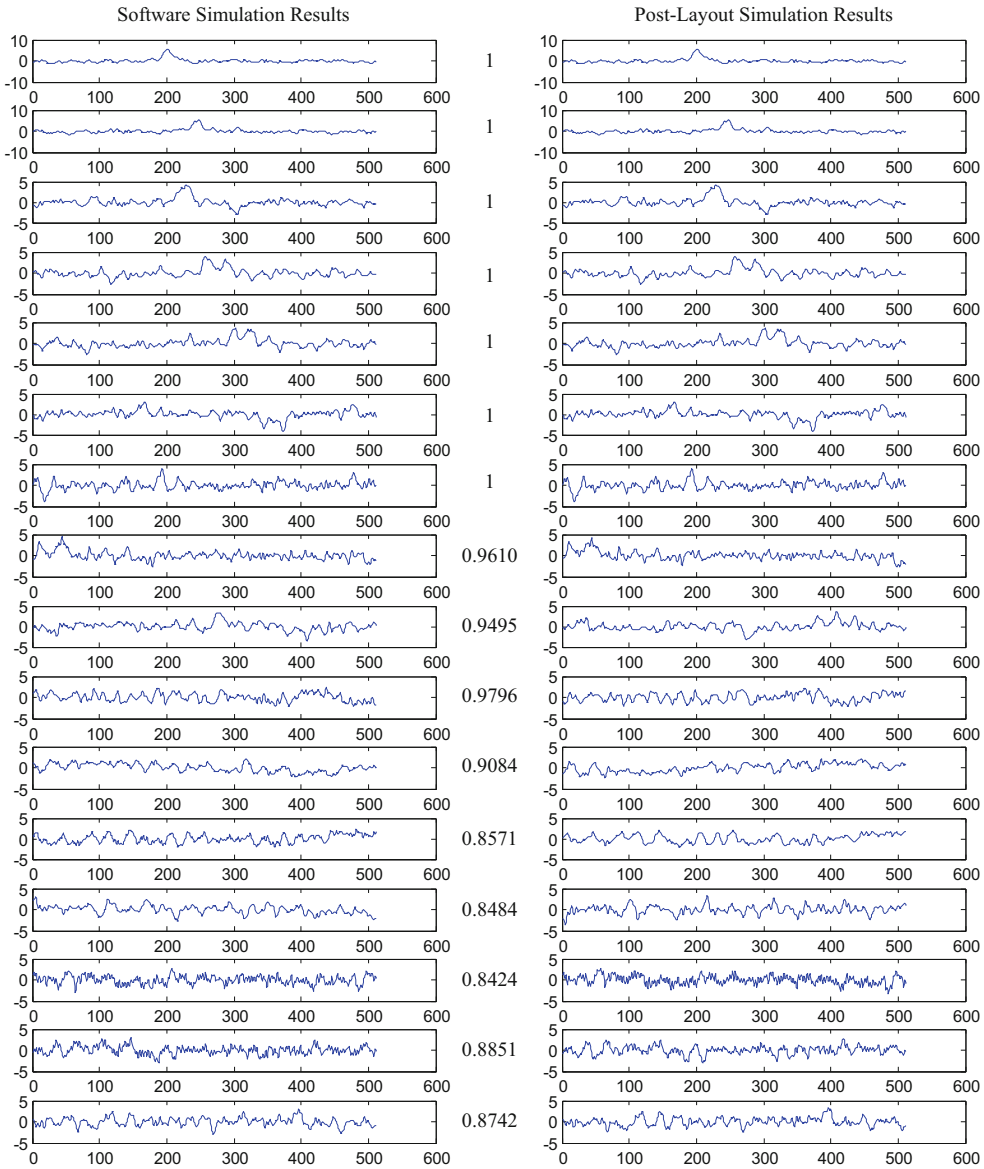


Figure 17 Comparison between the software simulation results and post-layout simulation results of the FastICA function for the second dataset in Fig. 9.

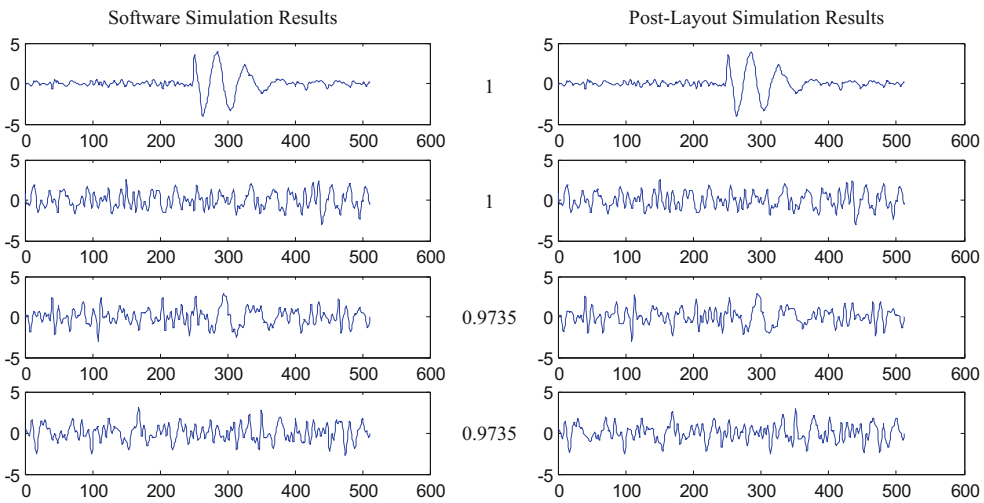


Figure 18 Comparison between the software simulation results and post-layout simulation results of the FastICA function for the third dataset in Fig. 11.

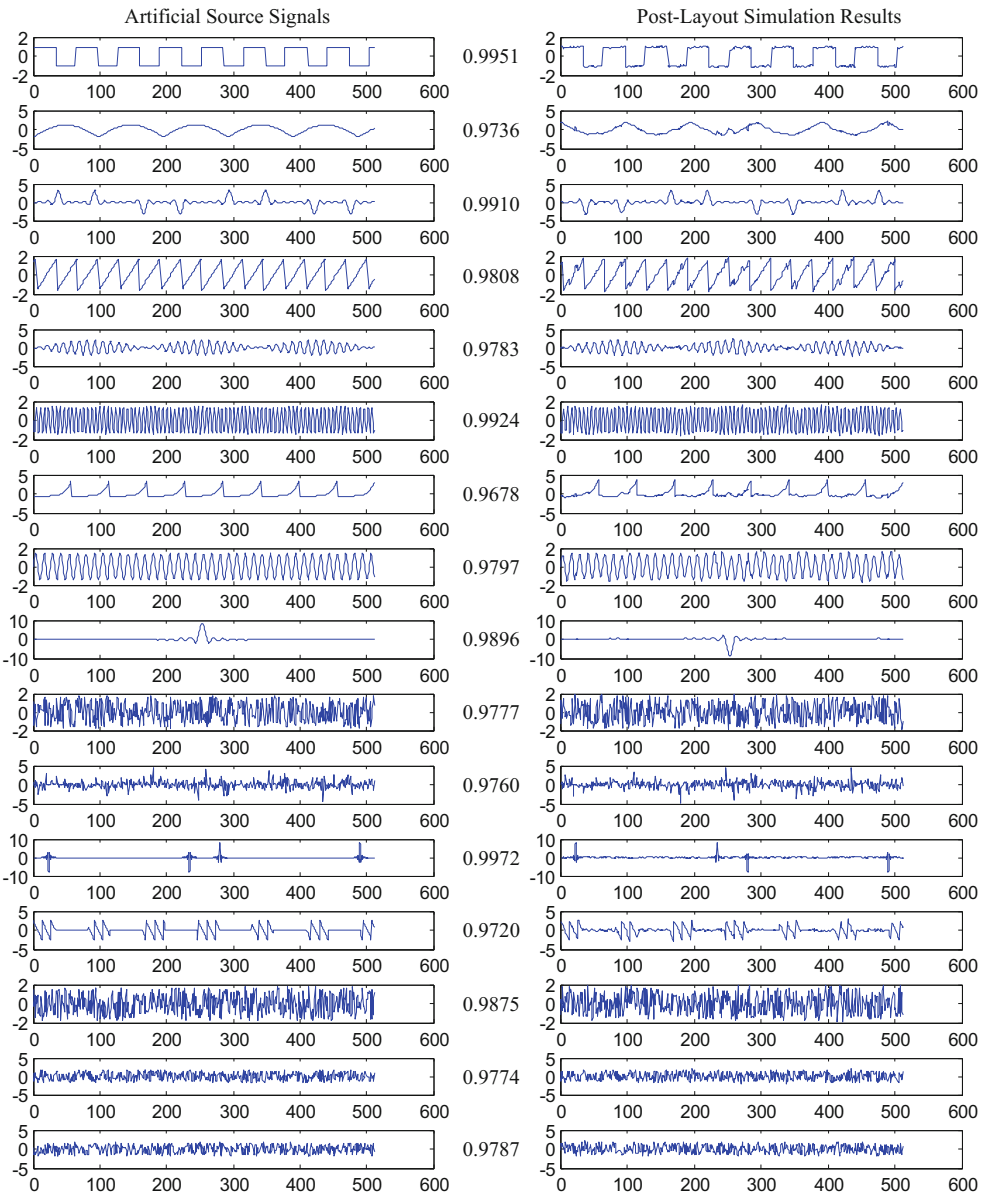


Figure 19 Comparison between the artificial source signals and post-layout simulation results of the FastICA function for the first dataset in Fig. 7.

the source signals and the post-layout simulation results. The average and minimum absolute correlation coefficients are 0.9822 and 0.9678, respectively. The SIR results for the 16-channel source signals and the post-layout simulation results with the first dataset are 22.2621, 13.0944, 18.5266, 14.6086, 14.0360, 19.4625, 12.1755, 14.3472, 17.7386, 13.8883, 13.5538, 27.8766, 12.8188, 16.7454, 13.8240 and 14.1117 dB, respectively, with an average value of 16.1919 dB. The results show that the separation quality of the proposed cost-effective FastICA hardware implementation is satisfactory. Figures. 20, 21, and 22 show the comparison results of the re-reference, synchronized average and moving

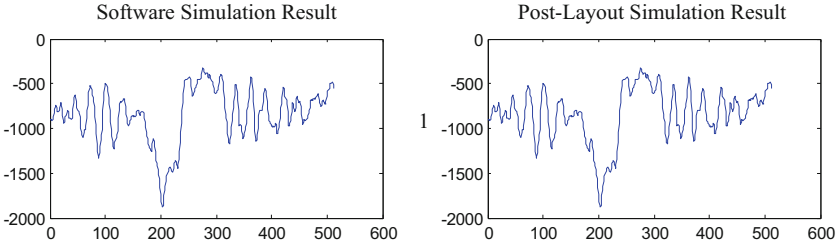


Figure 20 Comparison between the software simulation result and post-layout simulation result of the re-reference function.

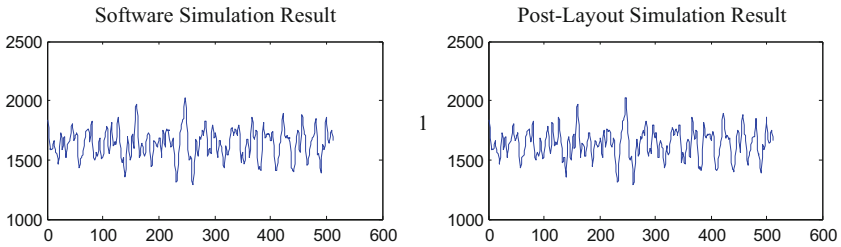


Figure 21 Comparison between the software simulation result and post-layout simulation result of the synchronized average function.

average functions, respectively. As can be seen, the absolute correlation coefficients can be up to 1 such that their accuracies can be guaranteed. According to the simulation results, since the post-layout simulation results are close to the software simulation results, the function of the hardware implementation can be validated with enough accuracy.

5.3 Evaluation and Comparison Results

According to the post-layout implementation results of the proposed FastICA hardware architecture, Table 6 summarizes post-layout characteristics. The power consumption results of the proposed hardware architecture for the FastICA processing are measured after RC extraction of the placed and routed netlist with Synopsys PrimeTime. The FastICA hardware architecture consumes 19.4 mW at 100 MHz when 16 channels are activated. Figure 23 shows the power consumption results among different number of channels. In the proposed FastICA hardware architecture, the power consumption results of different number of channels are close since the PU_1 and PU_2 are frequently utilized due to the heavy computations of the updating step [16] in (29) with the sample number of 512. On the other hand, Fig. 24 shows the energy evaluation results among different number of channels. As can be seen, with higher channel number, the energy consumption increases since the computation time increases due to the larger computational complexity. For the proposed FastICA hardware architecture, we target on the FastICA processing for the EEG data with the window size of 2 s and the sampling rate of 256 Hz. As can be seen in Table 6, since the maximum hardware computation time for the 16-channel FastICA processing is 1.85 s at 100 MHz, the maximum computation time is shorter than 2 s. That means the proposed FastICA hardware architecture can output 512 samples per channel every 2 s. In other words, the throughput is 256 samples/s. Thus, the proposed FastICA hardware architecture implementation can support enough throughput and real-time application in this work.

Table 7 lists comparison results in terms of application, algorithm, arithmetic, number of channels/weight vectors, sample size, speed, power dissipation, gate count, computation time, implementation approach, process technology, core area and variable channel among the post-layout implementation results of the proposed FastICA hardware architecture and the existing ICA implementations. In [20] and [26], FPGA implementations are used for two-channel ICA processing with the operating frequencies at 12.288 and 50 MHz, respectively. The number of gates in [20] is 11.469 thousand in FPGA approach; however, its computation time is larger than 60 s for ANC. Based on the parallel ICA algorithm, the operating frequency of the ICA implementation on FPGA is 20.161 MHz [21, 22] and between 21.357 and 35.921 MHz [23] for image processing. The design of [28] operates at 68 MHz on FPGA using INFOMAX algorithm for four-channel EEG signals processing. In [31, 32], the hardware implementation can achieve eight-channel FastICA processing with a dedicated EVD processor. In [33], the self-configured ICA implementation can achieve 16-channel processing with dedicated principal component decomposition engine (PCDE) for EVD processing. Since the implementation approach information and post-layout results of the overall architecture are not provided in [29], this low-complexity pioneer work is not listed in Table 7 for hardware comparison. In this work, the proposed hardware architecture can perform the FastICA processing with a variable number of channels from 2 to 16. By proposing two reused PUs for the preprocessing and the fixed-point algorithm in the FastICA algorithm, the gate count of this work is only 47.43 % higher than that reported in [31], while the processing capability in this work can be up to 16 channels and 512 samples that are two

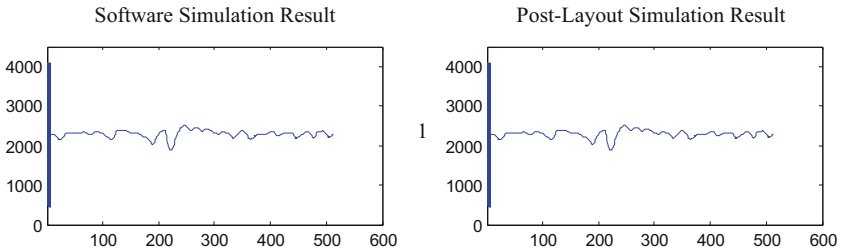


Figure 22 Comparison between the software simulation result and post-layout simulation result of the moving average function.

Table 6 Summary of the post-layout characteristics.

Power supply	1.0 V
Max. Clock	100 MHz
Core power for 16 channels	19.4 mW
Gate count	401K[a]
Core area	1.300×1.100 mm^2
Pin count	131
Process technology	TSMC 90 nm CMOS
Max. Computation time	1.85 s

[a] This number is counted using the size of two-input NAND gate

times as those provided in [31]. The gate count of this work with floating-point implementation is 5.79 times as that reported in [32] with fixed-point implementation; however, the FastICA processor in [32] is only for eight-channel electrocorticography (ECoG) signal processing. For 16-channel EEG signal processing in our work, the separation quality results in Section 5.2 could be guaranteed using the floating-point arithmetic and two times sample size. Although the ICA hardware [33] can provide the 16-channel FastICA processing with good separation quality, the variable-channel feature is not offered. On the other hand, the core size of [33] is larger than that of this work even considering the difference of process technology. In summary, this hardware architecture is cost effective while considering the satisfactory separation quality for 2-to-16 channel EEG signal processing application. In addition, the user-defined parameters as well as the functions of re-reference, synchronized average and moving average add the flexibility to the proposed hardware architecture.

6 Conclusion

In this work, we propose the cost-effective and 2-to-16 variable-channel FastICA hardware architecture for EEG signal processing. With the Gram-Schmidt based whitening in FastICA, the proposed two PUs can be shared between the preprocessing and the fixed-point algorithm to increase the hardware efficiency of the presented FastICA hardware architecture. The functions of re-reference, synchronized average and moving average as well as the user-defined parameters are also supported in the proposed FastICA hardware architecture to attain the flexibility. The evaluation and simulation results demonstrate that the proposed hardware architecture can achieve the satisfactory BSS quality and attain an efficient hardware usage. On the other hand, if more than 16-channel capability is demanded, the sample size must be largely increased, where the analysis is shown in Fig. 3, such that large memory size will be requested. This will be the near future work while considering separation quality and memory size with larger sample size for larger channel number.

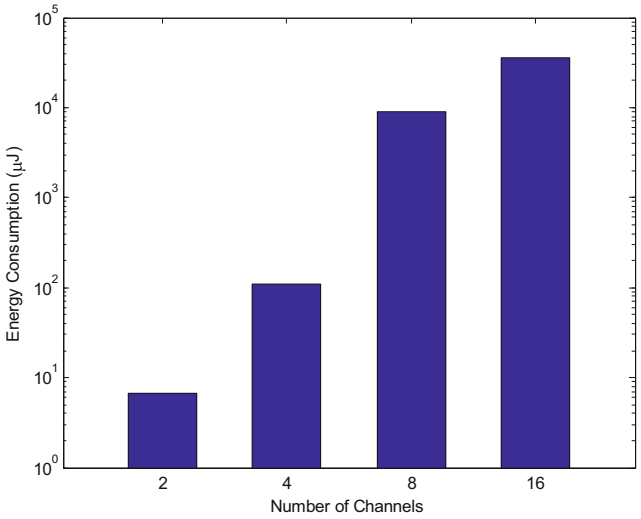


Figure 24 Energy consumptions in log scale of the proposed FastICA hardware architecture versus different number of channels.

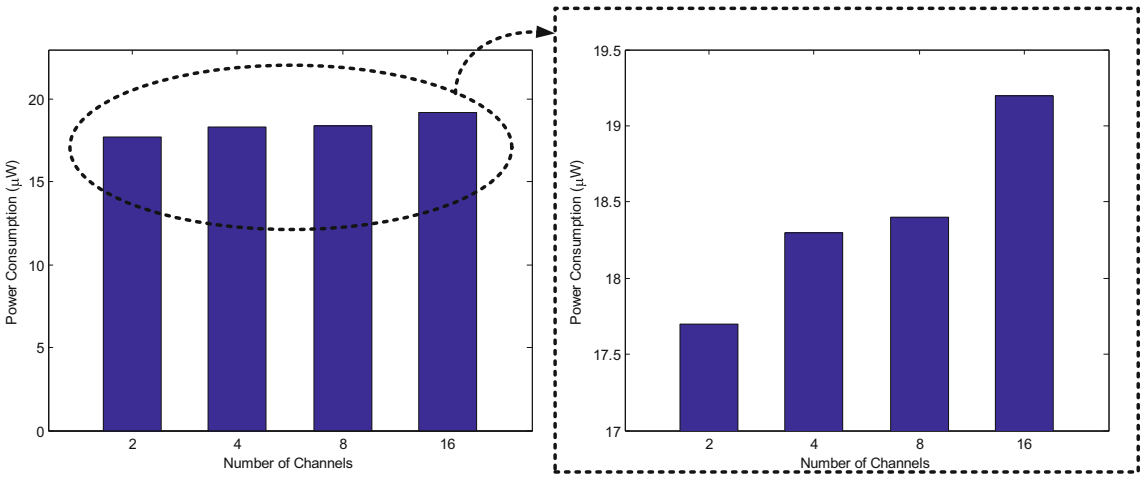


Figure 23 Power consumptions in linear scale of the proposed FastICA hardware architecture versus different number of channels.

Table 7 Hardware comparison results among various ICA implementations.

	Kim [20]	Du [21, 22]	Du [23][*1]	Shyu [26]	Huang [28]	Van [31]	Yang [32]	Roh [33]	This Work
Application	Speech	Image	Image	Speech	EEG	EEG	ECoG	EEG	EEG
Algorithm	ICA	Parallel ICA	Parallel ICA	FastICA	INFOMAX	FastICA	FastICA	Self-Configured ICA	FastICA
Arithmetic	Floating-point	Fixed-point	Fixed-point	Floating-point	Fixed-point	Hybrid	Fixed-point	N/A	Floating-point
No. of Channels/Weight Vectors (WVs)	2	4 (WVs)	4, 8, 12, 16, 20, 24 (WVs)	2	4	8	8	16	2–16
Sample Size	512	N/A	N/A	3000	512	256	256	512	512
Speed (MHz)	12.288, N/A	20.161, N/A	21.829[*2] 21.357[*3] 35.921[*4]	50	68	100	11	20	100
Power Dissipation (mW)	98.8, 14.5	N/A	N/A	N/A	N/A	16.35	0.0816	4.45[*6]	19.4[*7]
Gates ($\times 10^3$)	11.469, N/A (For ANC)	229.5, N/A	N/A	N/A	315	272	69.2	N/A	401
Computation Time (sec)	≥60, N/A	N/A	1129.5[*5]	0.003	N/A	0.29 (Max)	0.0842 (Max)	Variable	1.85 (Max)
Implementation Approach	FPGA, ASIC	FPGA, ASIC	FPGA	FPGA	FPGA	ASIC	ASIC	ASIC	ASIC
Process Technology (nm)	N/A, 350	N/A, 180	N/A	N/A	N/A	90	90	130	90
Core Area (mm^2)	N/A	1.41	N/A	N/A	N/A	1.49	0.40	7.02	1.43
Variable Channel	No	No	Yes	No	No	No	No	No	Yes

*1: Needs an external memory, *2: For sub-matrix, *3: For external decorrelation, *4: For comparison, *5: For 20 WVs, *6: For Mode3, *7: For 16 channels

Acknowledgments This work was supported in part by the Ministry of Science and Technology (MOST) Grants MOST 103-2221-E-009-099-MY3 and MOST 103-2911-I-009-101, and in part by Contract: 104W963. The authors thank Chip Implementation Center for their support.

References

1. Luck, S. J. (2005). *An introduction to the event-related potential technique*. Cambridge: MIT Press.
2. Bruce, E. N. (2005). *Biomedical signal processing and signal modeling*. New York: Wiley.
3. Hyvärinen, A., & Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Networks, 13*(4–5), 411–430.
4. Vigário, R. N. (1997). Extraction of ocular artefacts from EEG using independent component analysis. *Electroencephalography and Clinical Neurophysiology, 103*(3), 395–404.
5. Vigário, R., Särelä, J., Jousmäki, V., Hämäläinen, M., & Oja, E. (2000). Independent component approach to the analysis of EEG and MEG recordings. *IEEE Transactions on Biomedical Engineering, 47*(5), 589–593.
6. Makeig, S., Westerfield, M., Jung, T. P., Covington, J., Townsend, J., Sejnowski, T. J., & Courchesne, E. (1999). Functionally independent components of the late positive event-related potential during visual spatial attention. *Journal of Neuroscience, 19*(7), 2665–2680.
7. Castellanos, N. P., & Makarov, V. A. (2006). Recovering EEG brain signals: artifact suppression with wavelet enhanced independent component analysis. *Journal of Neuroscience Methods, 158*(2), 300–312.
8. Kachenoura, A., Albera, L., Senhadji, L., & Comon, P. (2008). ICA: a potential tool for BCI systems. *IEEE Signal Processing Magazine, 25*(1), 57–68.
9. Albera, L., Kachenoura, A., Comon, P., Karfoul, A., Wendling, F., Senhadji, L., & Merlet, I. (2012). ICA-based EEG denoising: a comparative analysis of fifteen methods. *Bulletin of the Polish Academy of Sciences Technical Sciences, 60*(3), 407–418.
10. Gao, J. F., Yang, Y., Lin, P., Wang, P., & Zheng, C. X. (2010). Automatic removal of eye-movement and blink artifacts from EEG signals. *Brain Topography, 23*(1), 105–114.
11. Lee, T. W. (1998). *Independent component analysis: Theory and applications*. Boston: Kluwer.
12. Cichocki, A., & Amari, S. (2002). *Adaptive blind signal and image processing: Learning algorithms and applications*. New York: Wiley.
13. Hyvärinen, A., & Oja, E. (1997). A fast fixed-point algorithm for independent component analysis. *Neural Computation, 9*(7), 1483–1492.
14. Hyvärinen, A. (1999). Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks, 10*(3), 626–634.
15. Hyvärinen, A., Karhunen, J., & Oja, E. (2001). *Independent component analysis*. New York: Wiley.
16. Oja, E., & Yuan, Z. (2006). The FastICA algorithm revisited: convergence-analysis. *IEEE Transactions on Neural Networks, 17*(6), 1370–1381.
17. Ollila, E. (2010). The deflation-based FastICA estimator: statistical analysis revisited. *IEEE Transactions on Signal Processing, 58*(3), 1527–1541.
18. Choi, S., Cichocki, A., & Amari, S. (2000). Flexible independent component analysis. *Journal of VLSI Signal Processing, 26*(1–2), 25–38.
19. Zarzoso, V., & Comon, P. (2010). Robust independent component analysis by iterative maximization of the kurtosis contrast with algebraic optimal step size. *IEEE Transactions on Neural Networks, 21*(2), 248–261.
20. Kim, C. M., Park, H. M., Kim, T., Choi, Y. K., & Lee, S. Y. (2003). FPGA implementation of ICA algorithm for blind signal separation and adaptive noise canceling. *IEEE Transactions on Neural Networks, 14*(5), 1038–1046.
21. Du, H., Qi, H., & Peterson, G. D. (2004). Parallel ICA and its hardware implementation in hyperspectral image analysis. *Proceedings of SPIE, 5439*, 74–83.
22. Du, H., & Qi, H. (2004). An FPGA implementation of parallel ICA for dimensionality reduction in hyperspectral images. *Proceedings of IEEE International Geoscience and Remote Sensing Symposium, 5*, 3257–3260.
23. Du, H., & Qi, H. (2006). A reconfigurable FPGA system for parallel independent component analysis. *EURASIP Journal on Embedded Systems, 2006*(23025), 1–12.
24. Du, H., Qi, H., & Wang, X. (2007). Comparative study of VLSI solutions to independent component analysis. *IEEE Transactions on Industrial Electronics, 54*(1), 548–558.
25. Jain, V. K., Bhanja, S., Chapman, G. H., Doddannagari, L., & Nguyen, N. (2005). A parallel architecture for the ICA algorithm: DSP plane of a 3-D heterogeneous sensor. *Proceedings of IEEE International Conference on Acoustics Speech and Signal Processing, 5*, 77–80.
26. Shyu, K. K., Lee, M. H., Wu, Y. T., & Lee, P. L. (2008). Implementation of pipelined FastICA on FPGA for real-time blind source separation. *IEEE Transactions on Neural Networks, 19*(6), 958–970.
27. Acharyya, A., Maharatna, K., Sun, J., Al-Hashimi, B.M., & Gunn, S.R. (2009). Hardware efficient fixed-point VLSI architecture for 2D kurtotic FastICA. *Proceedings of European Conference on Circuit Theory and Design*, 165–168.
28. Huang, W.C., Hung, S.H., Chung, J.F., Chang, M.H., Van, L.D., & Lin, C.T. (2008). FPGA implementation of 4-channel ICA for on-line EEG signal separation. *Proceedings of IEEE Biomedical Circuits and Systems Conference*, 65–68.
29. Acharyya, A., Maharatna, K., Al-Hashimi, B. M., & Reeve, J. (2011). Coordinate rotation based low complexity N-d FastICA algorithm and architecture. *IEEE Transactions on Signal Processing, 59*(8), 3997–4011.
30. Volder, J. E. (1959). The CORDIC trigonometric computing technique. *IRE Transactions on Electronic Computers, 8*(3), 330–334.
31. Van, L. D., Wu, D. Y., & Chen, C. S. (2011). Energy-efficient FastICA implementation for biomedical signal separation. *IEEE Transactions on Neural Networks, 22*(11), 1809–1823.
32. Yang, C. H., Shih, Y. H., & Chiueh, H. (2015). An 81.6 μW FastICA processor for epileptic seizure detection. *IEEE Transactions on Biomedical Circuits and Systems, 9*(1), 60–71.
33. Roh, T., Song, K., Cho, H., Shin, D., & Yoo, H. J. (2014). A wearable neuro-feedback system with EEG-based mental status monitoring and transcranial electrical stimulation. *IEEE Transactions on Biomedical Circuits and Systems, 8*(6), 755–764.
34. Strang, G. (2009). *Introduction to linear algebra* (4th ed.). Cambridge: Wellesley.
35. Cichocki, A., Osowski, S., & Siwek, K. (2004). Prewhitening algorithms of signals in the presence of white noise. *Proceedings of VI International Workshop "Computational Problems of Electrical Engineering"*, 205–208.
36. Woolfson, M. S., Bigan, C., Crowe, J. A., & Hayes-Gill, B. R. (2008). Method to separate sparse components from signal mixtures. *Digital Signal Processing, 18*(6), 985–1012.
37. Sharma, A., & Paliwal, K. K. (2007). Fast principal component analysis using fixed-point algorithm. *Pattern Recognition Letters, 28*(10), 1151–1155.
38. Huang, P.Y. (2011). Design and implementation of a multi-function cost-effective EEG signal processor. M. S. thesis, Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. (Advisor: Lan-Da Van).

39. Golub, G. H., & Van Loan, C. F. (1996). *Matrix computations* (3rd ed.). Baltimore: Johns Hopkins University Press.
40. The FastICA package for MATLAB. http://www.cis.hut.fi/projects/ica/fastica/.
41. Lan, T., Erdogmus, D., Pavel, M., & Mathan, S. (2006). Automatic frequency bands segmentation using statistical similarity for power spectrum density based brain computer interfaces. *Proceedings of International Joint Conference on Neural Networks*, 4650–4655.
42. Delorme, A., & Makeig, S. (2004). EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *Journal of Neuroscience Methods, 134*(2), 9–21.
43. EEGLAB Wiki – SCCN. http://sccn.ucsd.edu/wiki/EEGLAB.
44. Wu, A.Y. (Andy) (1995). Algorithm-based low-power digital signal processing system designs. Ph. D. thesis, Department of Electrical Engineering, University of Maryland, College Park, MD, USA.
45. Schulte, M.J., & Wires, K.E. (1999). High-speed inverse square roots. *Proceedings of 14th IEEE Symposium on Computer Arithmetic*, 124–131.
46. ICALAB for Signal Processing – benchmarks. http://www.bsp.brain.riken.jp/ICALAB/ICALABSignalProc/benchmarks/.
47. ICALAB for Signal Processing. http://www.bsp.brain.riken.jp/ICALAB/ICALABSignalProc/.

Lan-Da Van received the B.S. (Honors) and the M.S. degree from Tatung Institute of Technology, Taipei, Taiwan, in 1995 and 1997, respectively, and the Ph. D. degree from National Taiwan University (NTU), Taipei, Taiwan, in 2001, all in electrical engineering.

From 2001 to 2006, he was an Associate Researcher at National Chip Implementation Center (CIC), Hsinchu, Taiwan. Since Feb. 2006, he joined the faculty of Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, where he is currently an Associate Professor. His research interests are in VLSI algorithms, architectures, and chips for digital signal processing (DSP) and biomedical signal processing. This includes the design of low-power/high-performance/cost-effective 3-D graphics processing system, adaptive/learning systems, computer arithmetic, multidimensional filters, and transforms. He has published more than 50 journal and conference papers in these areas.

Dr. Van was a recipient of the Chunghwa Picture Tube (CPT) and Motorola fellowships in 1995 and 1997, respectively. He was an elected chairman of IEEE NTU Student Branch in 2000. In 2001, he has received IEEE award for outstanding leadership and service to the IEEE NTU Student Branch. In 2005, he was a recipient of the Best Poster Award at iNEER Conference for Engineering Education and Research (iCEER). In 2014, he was a recipient of the teaching award of Computer Science College, National Chiao Tung University. In 2014, he was a recipient of the best paper award in IEEE International Conference on Internet of Things (iThings2014). From 2009, he served as the officer of IEEE Taipei Section. In 2014, he served as Track Co-Chair of 22nd IFIP/IEEE International Conference on Very Large Scale Integration VLSI-SoC. He serves as Associate Editor and Guest Editor of Journal of Medical Imaging and Health Informatics. He served as a reviewer for several the IEEE transactions/journals.

Po-Yen Huang received the B.S. degree in EECS undergraduate honors program and the M.S degree in computer science from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 2010 and 2011, respectively. His research interests include VLSI information processing algorithm and architecture for biomedical signal processing.

Tsung-Che Lu received the B.S. degree in engineering science from National Cheng Kung University (NCKU), Tainan, Taiwan, in 2006, and the M.S. degree in engineering and system science from National Tsing Hua University (NTHU), Hsinchu, Taiwan, in 2008. He is currently working toward the Ph. D. degree with the Department of Computer Science, National Chiao Tung University (NCTU), Hsinchu, Taiwan. His research interests include analog and digital circuits for biomedical signal processing.