IMPLEMENTATION AND COMPARATIVE EVALUATION OF ICA WITH PIPELINED FASTICA FOR REAL-TIME BLIND SOURCE SEPARATION

K.R.Siva Bharathi

Assistant Professor
Sri Krishna College of Engineering and Technology, Coimbatore
Email id: sivabharathi@skcet.ac.in

Dr.X.Felix Joseph

Assistant Professor
Noorul Islam University, Kumaracoil
Email id: felix.jsph@gmail.com

Abstract—**Blind source separation of independent source signals from their mixtures is a common problem in real world multi-sensor applications. Independent component analysis is commonly reported for Blind Signal Separation (BSS). Independent Component Analysis (ICA) supports the normal sampling rate used in automatic speech recognition systems, which is incompatible to the systems with high sampling rates. So we propose the FastICA algorithm, which is a common offline method to identify artifacts and other interferences in the mixtures. FastICA becomes efficient for providing real time BSS when implemented in a Field Programmable Gate Array (FPGA). Added advantage of number precision is increased by the introduction of Floating point arithmetic. Increased sampling rate is provided by hand coding the FastICA. Using HDL language like VHDL facilitates the development of custom operators without significantly impacting the operator performance.**

Index terms--- **Blind Source Separation (BSS), Fast Independent Component Analysis (FastICA), Field Programmable Gate Array (FPGA), Floating point (FP), Hardware Description Language (HDL)**

I. INTRODUCTION

Biomedical signals such as electroencephalogram (EEG), magneto encephalography (MEG), and electrocardiogram (ECG) are generally measured from clinical sensors or instruments; however, the measured signals are polluted by the artifacts and other unknown noise signals, e.g., eye movements,muscle noise, and power noise from instruments[1]. This problem can be solved by independent component analysis (ICA) algorithm, which identifies artifacts or disturbances and extracts artifacts from the measured signals. The processing described previously is also named blind source separation (BSS)[2]. The meaning of "blind" is that both the original sources and the way the sources were mixed are all unknown, and only mixed signals or mixtures can be measured and observed. Many studies have been developed on the real time implementation of ICA [7]. These studies aimed at implementing ICA on FPGA"s [5] and pilchard boards and even combined it with adaptive noise cancellation techniques using adaptive filters [9].Independent Component Neural Networks (ICNN) [8] is also implemented in FPGA"s to separate the mixtures. The main drawbacks of these existing methods are poor performance due to the low sampling rates [3] and lack of pipelined architecture and the result is less accurate due to the use of fixed point arithmetic units [10].

ICA recovers independent source signals from their mixed signals by finding a lineartransformation thatmaximizes the mutual independence of mixtures. To improve the efficiency of ICA, FastICA algorithm is proposed. FastICA measures non-Gaussianityusing kurtosis to find the independent sources from their mixtures. In order to realize the real-time signal processing, the FastICA algorithm can be implemented on a field-programmable gate array (FPGA) to speed up the computation involving vector multiplications, matrix multiplications, and matrix inverses.

In this paper, the hardware FastICA is implemented by hand coding HDL code. Though there is software that translates the high-level languages such as C code, MATLAB, and even Simulink into HDL code, hand coding gives the implementation not only better performance but also less consumption of gate array in the FPGA. The proposed hardware FastICA implemented by hand coding provides high sampling rate up to 192 kHz. In addition, the numbers precision in hardware FastICA is increased by implementing the hardware floating-point(FP) arithmetic units. The FP arithmetic allows numbers to be represented in a wider range than fixed-point arithmetic [10], [11]. Finally, the ICA algorithm and pipelined architecture-based hardware FastICA is compared in this paper by means of simulation. The implementation of the algorithms is currently going on and the results will be analyzed soon.

II. ICA ALGORITHM

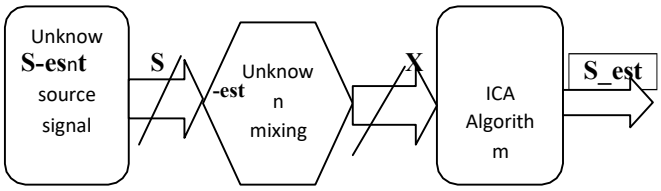

Fig.1. BSS using ICAalgorithm

Assume that the mixed signal matrix is defined as

$X = AS$ ..... **(1)**

Where $X = (x1, x2, \ldots . . xn)T$ and $S = (s1, s2, \ldots \ldots sn)T$ are the signal arrays, which have *n* observed mixed signals *n* and unknown independent sources, respectively, and **A** is a full-rank *n* by *n* mixing matrix. The goal ofICA algorithm isto recoverthe unknown source by estimating the mixingmatrix. Fig. 1is the illustration of the ICA processing. x and s are expressed as,

$x = [x(1), x(2) \ldots x(i-1), x(i)]$ ..... **(2)**

$s = [s(1), s(2) \ldots s(i-1), s(i)]$ ..... **(3**

Where i=1, 2 …m and m indicates the number of time samples.

A) PREPROCESS OF ICA

In order to reduce the complexity of the ICA algorithm calculation, it is necessary to preprocess the mixed signal matrix **X**. A popular preprocessing method is the principal component analysis (PCA) algorithm, which finds a linear transformation and translates correlative matrix **X** into **Z.** Then, components in **Z** become uncorrelated to each other.

The first step of PCA algorithm is called centering, which calculates the mean from the mixed signal x, and then subtracts the mean from x. The processing of centering is defined as

$$x(i) = x(i) - E\{x(m)\} \text{.....} \textbf{(4)}$$

Where **i=1, 2, 3…….. m** and m indicates the number of time samples. After centering, X becomes a zero mean matrix.

The second step of PCA algorithm is called whitening. The method of whitening utilizes eigenvalue decomposition (EVD) defined as

$$E\{XXT\} = Cx = EDET \text{ ….……..} \textbf{(5)}$$

Where $\mathbf{C_x}$ is the covariance matrix of X, $\mathbf{E=(e_1,e_2,……e_n)}$ is the orthogonal matrix of eigenvectors of $\mathbf{C_x}$, and $\mathbf{D=diag(\lambda_1,\lambda_2,\lambda_3,\text{.....}\lambda_n,)}$ is the diagonal matrix of eigenvalues of $\mathbf{C_x}$.

The whitening process can be described as

$$Z = PX \text{……} \textbf{(6)}$$

Where **P** is the whitening matrix and **Z** is a new matrix that is white will be defined as,

$$\mathbf{P = D^{-1/2} * E^T} \text{ ……} \textbf{(7)}$$

It is easy to show that the elements in **Z** are uncorrelated after whitening process because,

$$\mathbf{E\{ZZ^T\} = PE\{XX^T\}P^T = D^{-1/2}E^TEDE^TED^{-1/2} = I} \text{.....} \textbf{(8)}$$

After preprocessing, the mixing matrix **A** transforms into a new one. It can be received from,

$$Z = PX = PAS = \tilde{A}S \text{.....} \textbf{(9)}$$

The new mixing matrix is orthogonal. This means that the ICA problem of finding the full rank matrix is simplified to the estimation of the orthogonal mixing matrix.

III. FAST ICA ALGORITHM

TheFastICA algorithm uses the preprocessing steps centering and whitening. This makes use of an additional process called Kurtosis.Based on the central limit theorem, the distribution of the sum of independent random variables tends to be closer to a Gaussian. Thus, measurement of non-Gaussianity is used to find independent components. Traditional higher order statistics uses kurtosis or the named fourth-order cumulant to measure non-Gaussianity. The kurtosis of a zero-mean random variable y is defined by,

$$Kurt\{y\} = E\{y4\} - 3(E\{y2\})\,2 \text{……} \textbf{(10)}$$

Where $\mathbf{E\{y^{4}\}}$ is the fourth moment of y and $\mathbf{E\{y^2\}}$ is the second moment of y. for a Gaussian random variable y,$\mathbf{E\{y^{4}\}}$ equals 3($\mathbf{E\{y^2\})^2}$, so that the kurtosis is zero for a Gaussian random variable. If it is a non-Gaussian random variable, its kurtosis is either positive or negative. Therefore, non-Gaussianity is measured by the absolute value of kurtosis.

IV.IMPLEMENTATION:

A. IMPLEMENTATION OF FP ARITHMETIC UNITS:

Many scientific problems require FP arithmetic with high precision in their calculations. Moreover, a large dynamic range of numbers is necessary for signal processing. FP arithmetic has the ability to automatically scale numbers and allows numbers to be represented in a wider range than fixed-point arithmetic. The proposed 32-bit FP implementation includes adder, subtractor, multiplier, divider, and square rooter. Fig. 2 has the format of IEEE 745 standard 32-bit FP numbers. In it,s is the sign bit used to specify the sign of the FP number, is the 8-bit quantity called the exponent field, and has 23 bits used to store the binary representation of the FP number. The leading one in the mantissa does not appear in the representation; therefore, the leading one is implicit.

The FP value of F_p is computed by,

$$Fp = (-1)\text{^}5(1.f) * 2\text{^}(e - 127)$$

Sign Bias exponent fraction

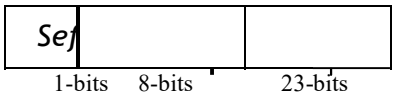

Fig.2.IEEE single precision FP format

B. IMPLEMENTATION OF FASTICA ALGORITHMS:

FPGA technology is very suitable to implement the digital signal processing algorithm for quickly verifying the algorithm in hardware. For real-time implementation of the FastICA algorithm, it requires high volume of mathematical operations in a very short time interval. Currently, most FPGAs have on-chip hardware multipliers and memory blocks, andare suitable for suchapplication.In this paper, the hardware FastICA is implemented by hand coding VHSIC hardware description language (VHDL). To increase the feasibility in the implementation of higher dimensions of FastICA, the design of implementation of FastICA is based on the concept of modules. The general Block diagram for implementing FastICA is illustrated below:

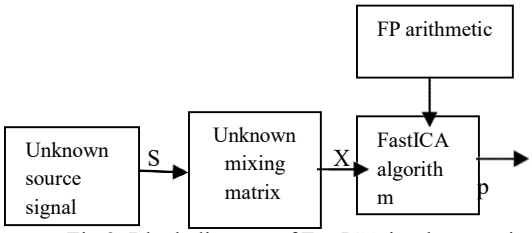

Fig.3. Block diagram of FastICA implementation

In order to store quantity of data, the on-chip memory in FPGA is used when the FastICA algorithm is processed. The memory is divided into four parts: memory-1, memory-2, memory-3, and memory-4. Memory-1 is used to store the measuredmixed signal **X** from analog-to-digital circuit module. Memory-2 stores the centered data. Furthermore, memory-3 and memory-4 are used to store the whitened data **Z** and the estimated independent component **S-est**, respectively.

Four blocks are needed to implement FastICA: centering, whitening, kurtosis, and FastICA controller. The detailed implementation of each block will be discussed as follows:

*1) **Centering:***The process of centering is to subtract the mixed signal means μ_1 and μ_2 from x_1 and x_2, respectively.

First, the mixed signals x_1 and x_2 will be read out from the memory-1_A and memory-1_B, respectively, and then it will accumulate the element of x one by one. After getting the summation of, will be obtained by dividing the summation by 3000. In order to speed up the processing, use multiplication operation (multiply by 1/3000) instead of the division, where 3000 is the number of time samples. Second, subtract the mean from the data in memory-1 forachieving centering and then save the results into memory-2.

*2) **Whitening:***The first step of whitening is to find the whitening matrix **P**, given by $\mathbf{D^{-1/2}*E^T}$, where $E = (e1, e2, \dots \dots en)$ is the orthogonal matrix of eigenvectors of $\mathbf{C_x}$, and

$D = diag(\ 1,\ 2,\ 3,\ \dots\ \dots\ \dots\ .\ .\ n,\ )$ is the diagonal matrix of eigenvalues of Cx. Covariance matrix, $Cx = E\{XXT\}$ is a 2 x2 matrix. It takes three multipliers to implement the calculation of $\mathbf{XX^T}$. Multiplier-1 is used for $x_1 * x_1$, multiplier-2 is used for $x_1 * x_2$, and multiplier-3 is used for $x_1 * x_2$, where,

$$(X) = \begin{matrix} x1 \\ x2 \end{matrix} = \begin{pmatrix} x1(1), x1(2) \cdots & x1(3000) \\ x2(1), x2(2) \cdots & x2(3000) \end{pmatrix}$$

Because $x_1 * x_2$ equals $x_2 * x_1$, it only needs to implement one of them in order to save the hardwareresources. The method described previously is a parallel operation. In other words, it performs three multiplications at the same time after reading two words out of data x_1 and x_2 and from memory-2_A and memory- 2_B, respectively. After 3000 accumulation and parallel multiplication operations, $\mathbf{C_x}$ thus can be derived by multiplying summation by 1/3000. The hardware operation described previously can be formulated as,

$$Cx = \begin{pmatrix} Cx00 & Cx01 \\ Cx10 & Cx11 \end{pmatrix}$$

$$Cx - 00 = (\sum x1\ (j)\ *\ x1(j))\ *\ 0.00033 \qquad \textbf{(10)}$$

$$Cx - 01 = Cx - 10 = (\sum x1(j) * x2(j)) * 0.000333 \ \textbf{..... (11)}$$

$$Cx - 11 = (\sum x2\ (j)\ *\ x2(j))\ *\ 0.00033\ \text{.....} \textbf{(12)}$$

Each centering result will be saved into memory-2 and will be used for calculating simultaneously. The implementation block diagram is presented in fig.3. When the calculation of $\mathbf{C_x}$ has been carried out, the next step is to calculate the eigenvalues λ_1 and $\lambda 2$ of Cxby the equation $a\lambda 2 + b\lambda + c = 0$. First, it simultaneously utilizesan adder to calculate the parameter $b = -(C_{00} + C_{11})$ and two multipliers associated with a subtractor to calculate the parameter c=det($\mathbf{C_x}$). The solutions λ_1 and $\lambda 2$ are both positive because C_x is a positive semi-definite. Practically, is positive definite for almost all natural data. Because the value of „a" equals unity, the operation of just utilizes left shifting operation on the c. Using a square rooter and a matrix inverse circuit, we can calculate $\mathbf{D^{-1/2}}$ easily. The algorithms for finding eigenvectors of E^T, normalization, and matrix transformation are all implemented. The matrix **P** is thus obtained by multiplying $\mathbf{D^{-1/2}}$ by $\mathbf{E^T}$**.** Finally, the white data could be obtained after multiplyingby **P** by **X.**. Then it is stored into memory-3. The block diagram of whitening is presented in Fig. 4.

*3) **Kurtosis:***The calculation of separating vector in the following equation has been already described in the previous section and it is expressed as,

$$\mathrm{w}\ (k) = E\ \{Z\ (ZT\mathrm{w}\ (k-1))3\} - 3\mathrm{w}\ (k-1) \textbf{.....(13)}$$

The calculation of$Z\ (ZT\mathrm{w}\ (k-1))\ 3$ is first implemented and the corresponding concept is presented in Fig. 5, where Memory-3_A and memory-3_B store the white data $z_1$and z_2 respectively. The parallel hardware in Fig. 5 must perform 3000 times and the operation is controlled by input "pre_w(k) Cal. Control." The hardware operation could be formulated as,

$$Pre\_\mathrm{w}\ (k) = \sum (\mathrm{z(j)}\ *\ (\mathrm{z(j)w}(k-1))3) \textbf{ ... (14)}$$

Where $Pre_\mathrm{w}\ (k)$iscalculation result of $\mathbf{Z(Z^Tw\ (k-1))}$ [3].

To save the logic element of FPGA, sequence hardware design is adopted. The remaining calculations will sequentially share the FP arithmetic components until the calculation of is finished.

*4) **FastICA Controller:***The works of FastICA controller are as follows. 1) Drive centering, whitening, and kurtosis blocks to be sequentially active, and control the memory-access cycle when each FastICA block requires accessing the data from memory. 2) When processing kurtosis block, FastICA controller will setup the initial value w (0), and then it compares the kurtosis result w (k) with w (k-1) for judging whether the "fixed-point iteration" in FastICA algorithm has converged.

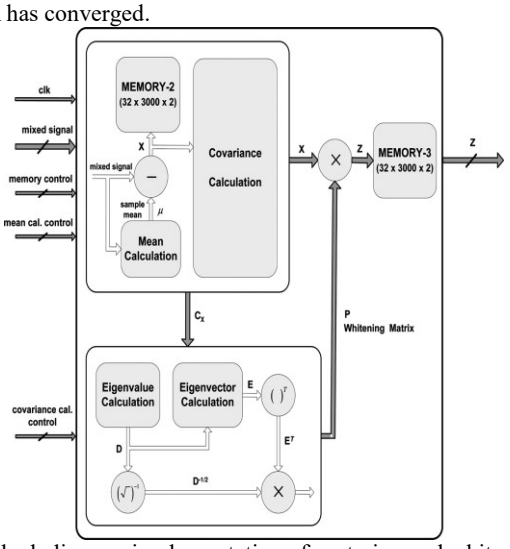

Fig.4. Block diagram implementation of centering and whitening.

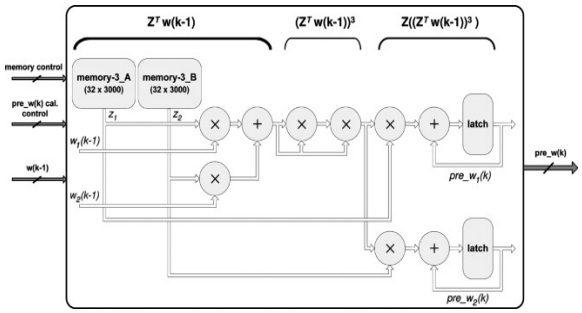

Fig.5. Block diagram implementation of the equation $:(\sum Z(\mathrm{j}) * (Z(\mathrm{j}))w(k-1))3)$

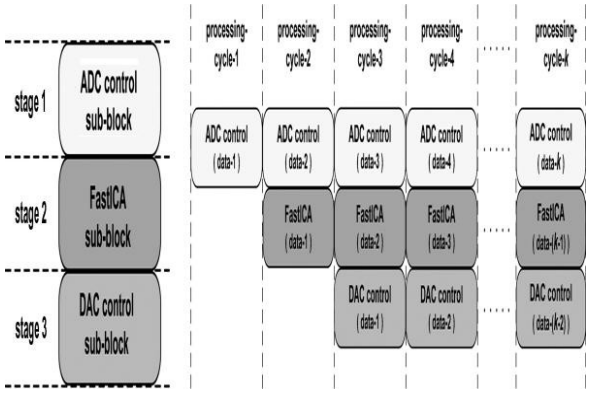

Fig.6.Illustration of proposed pipeline flow of pipelined FastICA

C. IMPLEMENTATION OF PIPELINED FASTICA ARCHITECTURE

This requires two sub blocks as explained below.

1) Overall Pipelined FastICA:In order to implement the pipeline architecture, the signal processing flow is divided into three sub blocks: analog-to-digital converter (ADC) control, FastICA calculation, and digital-to-analog converter (DAC) control. Besides three sub blocks, pipeline controller is needed to control the processing sequence of each sub block in pipeline architecture. The overall concept of pipelined FastICA is illustrated in Fig. 5. The process starts at processing cycle 1:

After first two cycles of processing, three sub-blocks will operate simultaneously in the same processing cycle. The estimated independent analog components output continuously after processing cycle 3. Utilizing the pipeline architecture, the implementation of FastICA achieves real-time continuous signal processing.

2) ADC and DAC Control Sub-blocks:The main concern is when the pipeline processing is the memory-accessing conflict problem. As shown in Fig. 5, at processing cycle 2, FastICA processing sub-block reads the data-1 out from memory-1 for processing, however, at the same time, ADC control sub-block has to store the sampling data, data-2, into memory-1. The same conflict happens at processing cycle 3. When FastICA sub-block saves the calculated results into memory-4, the DAC control sub-block has to read out the data from memory-4 for DAC operation in the same processing cycle. Therefore, memory-1 pair and memory-4 pair are proposed to avoid the memory conflict problem.

V. EXPERIMENTAL RESULTS:

In the simulation, sine and triangle waves generated from MATLAB are taken as source signals. Then, the mixed signals are produced by multiplying the random mixing matrix and the generated source signals. This mixing signal is pre-processed and the further processes are done using hand coding VHDL and the independent source signal is obtained at the output. The simulation results are illustrated below: Fig.6.indicates the mixed signal generated using MATLAB. Fig.7. represents the pre-processed signal that is further processed using PCA. Fig.8. indicates the obtained independent signal using hand coding VHDL.

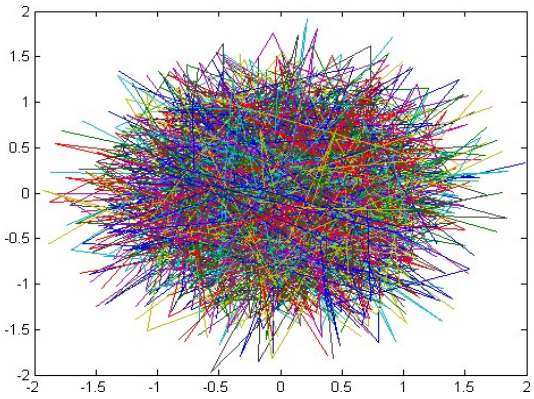

Fig.6. Generated mixed signal

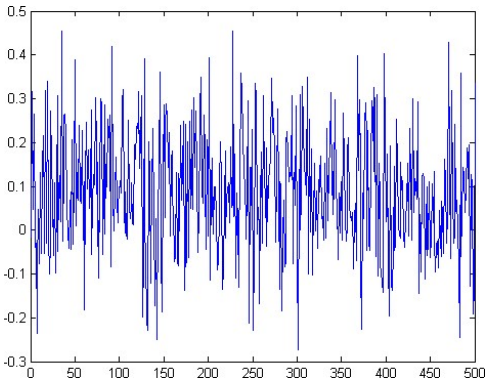

Fig.7. Pre-processed signal

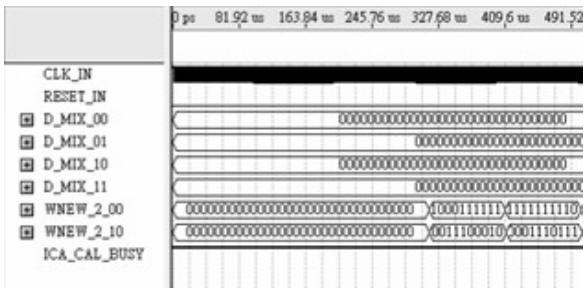

Fig.8. Processed signal using hand coding

The implementation will be done further and the results will be analyzed and comparative evaluation will be done with ICA.

VI. CONCLUSION:

This paper compares the performance of the ICA algorithm and the FastICA algorithm for real time implemented in FPGA. The FastICA algorithm proves the real time implementation of Blind Source Separation more feasible. ICA makes use of 12 KHz sampling with 12-bit A/D conversion, commonly used in automatic speech recognition system. With higher sampling rates, ICA require more computing power and memory bandwidth with given convolutive mixing channels, whereas the FastICA pipelined architecture allows high speed real time signal processing of FastICA with high sampling rate of up to 192 KHz by hand coding it in VHDL. Moreover , the FP arithmetic is implemented in the pipelined FPGA based FastICA to provide better precision and high dynamic performance. In future, the prototype system with FastICA algorithm could work without the need of a personal computer. Both simulation and experiment demonstrate that the proposed FPGA based FastICA system is capable of real-time signal processing such as speech signal enhancement and fetal heart rate (FHR) monitoring.

REFERENCES:

[1] Kuo-Kai shyu, Ming-Huan Lee, Yu-Te Wu and Po-Lei Lee: "*Implementation of pipelined FastICA on FPGA for Real-Time Blind Source Separation*,"IEEE Trans. On Neural Networks, Vol.19, no.6, June 2008.

[2] C. M. Kim, H. M. Park, T. Kim, Y. K. Choi, and S. Y. Lee, "*FPGA implementation of ICA algorithm for blind signal separation and adaptive noise canceling*," IEEE Trans. Neural Netw., vol. 14, no. 5, pp. 1038–1046, Sep. 2003.

[3] J. T. Chien and B. C. Chen, "*A new independent component analysis for speech recognition and separation*," IEEE Trans. Speech AudioProcess., vol. 14, no. 4, pp. 1245–1254, Jul. 2006.

[4] L. Tian, D. Erdogmus, A. Adami, M. Pavel, and S. Mathan, "*Salient EEG channel selection in brain computer interfaces by mutual information maximization*," in Proc. IEEE Annu. Int. Conf. Eng. Med. Biol.Soc., 2005, pp. 7064–7067.

[5] A. Nordin, C. Hsu, and H. Szu, "*Design of FPGA ICA for hyperspectral image processing*," in Proc. SPIE Signal Image Process, Mar. 2001, vol. 4391, pp. 444–454.

[6] H. Du and H. Qi, "*An FPGA implementation of parallel ICA for dimensionality reduction in hyperspectral images*," in Proc. IEEE Int. Symp.Geosci. Remote Sens., Sep. 2004, vol. 5, pp. 3257–3260.

[7] C. Charoensak and F. Sattar, "*A single-chip FPGA design for real-time ICA-based blind source separation algorithm,*" in Proc. IEEE Int. Symp.Circuits Syst., May 2005, vol. 6, pp. 5822–5825.

[8] A. R. Omondi and J. C. Rajapakse, "*Neural networks in FPGAs*," in Proc. Int. Conf. Neural Inf. Process., Nov. 2002, vol. 2, pp. 954–959.

[9] A. Celik, M. Stanacevic, and G. Cauwenberghs, "*Mixed-signal real-time adaptive blind source separation*," in Proc. IEEE Int. Symp. CircuitsSyst., May 2004, vol. 5, pp. V-760–V-763.

[10] A. Hyvärinen and E. Oja, "*A fast fixed-point algorithm for independent component analysis*," Neural Comput., vol. 9, pp. 1483–1492, 1997.

[11] N. Shirazi, A. Walters, and P. Athanas, "*Quantitative analysis of floating point arithmetic on FPGA based custom computing machines*,"inProc. IEEE Symp. FPGAs Custom Comput. Mach., 1995, pp. 155–162.