IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 15, 2021, accepted March 31, 2021, date of publication April 12, 2021, date of current version April 21, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3072495

An Efficient VLSI Architecture for FastICA by Using the Algebraic Jacobi Method for EVD

MUHAMMAD SAJJAD[1], MOHD ZUKI YUSOFF[1], (Member, IEEE), NORASHIKIN YAHYA[1], (Member, IEEE), AND ALI SHAHBAZ HAIDER[2], (Member, IEEE)

[1]Centre for Intelligent Signal and Imaging Research, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Malaysia
[2]Department of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA

Corresponding author: Muhammad Sajjad (muhammad_19001746@utp.edu.my)

This work was supported in part by the Ministry of Education Malaysia through the Higher Institutional Centre of Excellence (HICoE) Scheme awarded to the Centre for Intelligent Signal and Imaging Research (CISIR), in part by the Graduate Assistantship Scheme of Universiti Teknologi PETRONAS, and in part by the Yayasan Universiti Teknologi PETRONAS (YUTP) Fund under Grant 015LC0-239.

ABSTRACT Blind source separation (BSS) is a problem that appears in many research fields. Fast Independent components analysis (FastICA) is one of the techniques to solve the problem. The researchers have verified the effectiveness of the technique through the offline analysis of the public datasets. The development of a real-time portable system involving such a computationally complex analysis requires an efficient hardware implementation of FastICA. A Field programmable gate array (FPGA) and an application-specific integrated circuit (ASIC) are two promising hardware platforms to implement FastICA. This work proposes a new method, called ALgebraic Jacobi Method (ALJM), for performing eigenvalue decomposition (EVD) required for the implementation of FastICA. We use a simplification, a polynomial approximation, and the Newton-Raphson method for calculating the Jacobi rotation. In this way, we ensure hardware reusability between the EVD stage and the weight vector estimation (WVE) stage of FastICA which reduces the computational complexity and the power consumption, without compromising its computation speed. We evaluate the ALJM-based FastICA by performing BSS on the linear mixtures of the deterministic and the random signals and comparing the performance results with the existing methods. After verifying its functionality and numerical stability, we propose a scalable systolic processing array (SPA) for the ALJM-based FastICA and implement it on Spartan-6 FPGA. By comparing the existing implementations of FastICA, in terms of speed, area, and power, we conclude that the ALJM-based FastICA is one of the most efficient methods for prototyping and commercializing a real-time portable system comprising FastICA.

INDEX TERMS Application-specific integrated circuit, ASIC, blind source separation, commercialization, eigenvalue decomposition, field-programmable gate array, FastICA, Fixed-point Designer, independent components analysis, Jacobi method, VLSI.

I. INTRODUCTION

Independent component analysis (ICA) is a statistical technique for decomposing a multivariate signal into statistically independent components (ICs) having non-Gaussian probability distributions [1]. This technique has many variants, such as FastICA [2], [3], [4], and the scope of its applications is ever-growing [5]. After evaluating the technique's effectiveness through offline analysis on the available public datasets [6], the researchers have started deploying this computationally complex technique in real-time portable applications [7], [8]. Many efforts have been made for developing the portable hardware implementing ICA followed by some artificial intelligence (AI) or machine learning (ML) algorithms in real-time [9]. Some of them are summarized below.

A wearable neuro-feedback system (NFS) is developed in [10]. This system implements FastICA for removing the artifacts in real-time. A hardware design of FastICA for epileptic seizure detection is proposed in [11]. In [12], a hardware architecture for a cost-effective 16-channels FastICA is reported for the processing of electroencephalographic (EEG) signals in real-time. Another wearable mental state monitoring system is developed in [13]

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

by implementing FastICA in the ASIC for removing the artifacts. A VLSI design of 3-channel ICA is presented in [14] for separating and localizing the acoustic sources. In [15], [16], ICA is also used for removing the artifacts in real-time. A System-on-Chip (SoC) based implementation of FastICA for dynamic background subtraction in real-time and portable vision devices is deployed in [17]. A field-programmable gate array (FPGA) based implementation of ICA is adopted in [18] for digital pre-distortion (DPD) in wireless communication systems. Moving average ICA for real-time applications is evaluated in [19]. Another FPGA implementation of FastICA, using a reference input for rearranging the independent components (ICs), is reported in [20].

The literature listed above shows that FastICA is the most frequently used variant of ICA. In FastICA data whitening is followed by weight vector estimation (WVE). Data whitening mainly consists of eigenvalue decomposition (EVD), whereas WVE consists of matrices multiplications and non-linearity computation. In hardware, EVD is computed by the Jacobi method [21] because it can be parallelized on a systolic processor array (SPA), as in [22], [23]. The Jacobi method can be implemented in two ways, the exact Jacobi method (EJM) and the approximate Jacobi method (AJM). The EJM, having quadratic convergence, is implemented using COordinate Rotation DIgital Computer (CORDIC) blocks [24], [25], [26], whereas the AJM, having almost linear convergence, is implemented with shift-and-add operations [27], [28]. The EVD stage occupies more than 50% of chip area in the FastICA implementation [11]. To share the hardware between the two stages, [24] implements FastICA using CORDIC blocks, whereas [12] uses the Gram-Schmidt orthonormalization. But these approaches save silicon area at the expense of computation speed.

The approaches listed above, either optimize both the stages separately or merge the hardware for both the stages by using the slow iterative processes. So, the computational complexity of FastICA is still limiting its use for real-time portable platforms.

To solve the issue stated above, the current work contributes in the following three ways.

1) We propose a new method for computing EVD in the whitening process of FastICA. This method is named as ALgebraic Jacobi Method (ALJM) based on its computational nature.
2) The theoretical performance of FastICA based on the ALJM, the EJM, and the AJM is evaluated by varying the wordlength and the dimensionality (number of channels).
3) A SPA for FastICA is proposed based on the ALJM. The SPA based architecture is area-efficient, high-speed, scalable, and can fit into a low-cost FPGA device.

Contrary to the EJM or the AJM, the ALJM does not use CORDIC elementary angles for computing the Jacobi rotations. We use a polynomial approximation followed by an inverse square root operation, using the Newton-Raphson method, for computing the tangent of the Jacobi rotation angle. Fixed-point Designer tool of MATLAB [29] is used for the theoretical evaluation because it supports the datatypes and the arithmetic operations of embedded hardware for bit-true simulation. Quantization error level, convergence speed, and cross-correlation coefficient (*CCC*) are used as the performance metrics. The proposed method is also evaluated on the hardware level using Spartan-6 FPGA. The evaluation proves that the ALJM-based FastICA can be used for rapid prototyping and commercialization of the real-time portable systems, as demonstrated in Fig. 1.

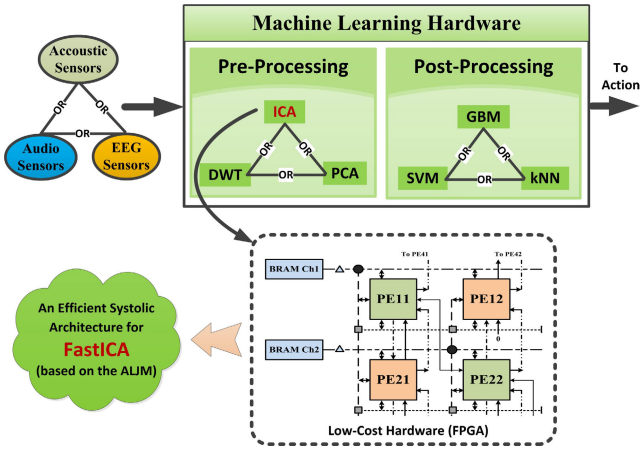

FIGURE 1. A Real-time Portable System Capable of FastICA.

We organize this article into seven sections. In Section II, we review the existing methods and formulate the problem. The ALJM is described in Section III followed by the proposed VLSI architecture for the ALJM-based FastICA in Section IV. The theoretical evaluation of the method is detailed in Section V followed by the hardware evaluation in Section VI. Finally, the conclusion and future work is listed in Section VII. In the article, we use bold italic capital letters for naming matrices and bold italic small letters for denoting one-dimensional vectors. The transpose of a matrix or a vector is represented by superscript ^{T} on its name.

II. LITERATURE REVIEW AND PROBLEM FORMULATION

The efficient implementation of FastICA on a hardware platform demands some knowledge of statistics, numerical linear algebra, advanced digital design, and modern digital design technologies. So, in the following sub-sections, we discuss the relevant concepts for understanding the research problem.

A. MATHEMATICAL DESCRIPTION OF ICA

Let $\boldsymbol{S}$ and $\boldsymbol{X}$ be the two matrices of order $n \times m$ representing a set of some non-Gaussian latent source signals and a set of the corresponding observable signals, respectively. If the observable signals are linear mixtures of the source signals, as in (1), then we can use FastICA to compute $\boldsymbol{W}$ for estimating the source signals, as in (2). Here, $\hat{\boldsymbol{S}}$ represents the estimate of

the source signals, n is the number of the observable signals, m is the number of the trials or observations, $\boldsymbol{A}$ is the mixing matrix, and $\boldsymbol{W}$ is the matrix of weight vectors.

$$\boldsymbol{X} = \boldsymbol{AS} \tag{1}$$
$$\hat{\boldsymbol{S}} = \boldsymbol{W^T X} \tag{2}$$

1) DATA WHITENING PROCESS

FastICA is used for estimating super-Gaussian and sub-Gaussian latent sources (including one Gaussian source), but it demands zero-mean and unit-variance mixed signals as its inputs [30]. So, the mean of each mixed signal is subtracted and a new zero-mean mixed-signal matrix $\tilde{\boldsymbol{X}}$ is used to update $\boldsymbol{X}$, as in (3). The update operator := is used throughout the text to indicate the update operation of the scalers, vectors, or matrices, as in computer programming.

$$\boldsymbol{X} := \tilde{\boldsymbol{X}} \tag{3}$$

Covariance matrix $\boldsymbol{C}_{XX}$ of order $n \times n$ is obtained from $\boldsymbol{X}$, as in (4), where $\xi[]$ denotes the expectation operator of statistics. Using EVD of $\boldsymbol{C}_{XX}$ [31], the eigenvalues and the eigenvectors are computed using the diagonal matrix $\wedge$ and the orthogonal matrix $\boldsymbol{E}$ respectively, as in (5), and these are used for the whitening process, as in (6), where $\boldsymbol{H}$ is the whitening matrix and $\boldsymbol{Z}$ is the zero-mean unit-variance data matrix.

$$\boldsymbol{C}_{XX} = \xi\left[\boldsymbol{XX^T}\right] \tag{4}$$
$$\boldsymbol{C}_{XX} = \boldsymbol{E} \wedge \boldsymbol{E}^T \tag{5}$$
$$\boldsymbol{H} = \wedge^{-1/2}\boldsymbol{E}^T \Longrightarrow \boldsymbol{Z} = \boldsymbol{HX} \tag{6}$$

2) WEIGHT VECTORS ESTIMATION

FastICA separates the sources by measuring the non-Gaussianity of $\boldsymbol{Z}$. An approximation to negentropy of the data can be used as a measure of non-Gaussianity, and $g(x) = \tanh(x)$ can be used as a derivative of the non-quadratic function used for approximating negentropy. So, each column of $\boldsymbol{W}$, denoted by $\boldsymbol{w}_i$, is updated, as in (7) and (8), whereas $\|\boldsymbol{w}_i\|$ is the Euclidean norm of $\boldsymbol{w}_i$.

$$\boldsymbol{w_i} := \xi\left[\boldsymbol{Z}\left(g\left(\boldsymbol{w}_i^T\boldsymbol{Z}\right)\right)^T\right] - \xi\left[g'(\boldsymbol{w}_i^T\boldsymbol{Z})\right]\boldsymbol{w_i} \tag{7}$$
$$\boldsymbol{w_i} := \boldsymbol{w_i}/\left\|\boldsymbol{w_i}\right\|, \quad whereas\ i = 1, 2, \ldots, n \tag{8}$$

Using (9), $\boldsymbol{W}$ is normalized, and then its symmetric decorrelation is performed, as in (10), iteratively until the convergence of $\boldsymbol{W}$.

$$\boldsymbol{W} := \boldsymbol{W}/\sqrt{\left\|\boldsymbol{WW}^T\right\|} \tag{9}$$
$$\boldsymbol{W} := 1.5\boldsymbol{W} - 0.5\boldsymbol{WW}^T\boldsymbol{W} \tag{10}$$

After the convergence of $\boldsymbol{W}$ in (10), the computations from (7) to (10) are repeated for the overall convergence of $\boldsymbol{W}$, so that it can be used for the estimation of the source signals, as in (2). In (9), $\left\|\boldsymbol{WW}^T\right\|$ is 2-norm of $\boldsymbol{WW}^T$ and (11) can also be used for estimating the sources signals.

$$\hat{\boldsymbol{S}} = \boldsymbol{W^T Z} \tag{11}$$

B. EVD BY THE EXACT JACOBI METHOD

By using EVD, a real square symmetric matrix $\boldsymbol{S}$ of order $n \times n$ can be expressed in the form of (12), where $\boldsymbol{V}$ is an orthogonal matrix with the columns as eigenvectors of $\boldsymbol{S}$ and $\boldsymbol{D}$ is a diagonal matrix with the diagonal entries as eigenvalues of $\boldsymbol{S}$. By using the Jacobi method [21], the two matrices are obtained by rotating $\boldsymbol{S}$ through a series of two-sided orthogonal rotations (also called the Jacobi rotations) until it is diagonalized, as in (13). The purpose of these rotations is to annihilate off-diagonal elements of $\boldsymbol{S}$. These rotations are performed iteratively, as in (14), where $\overleftrightarrow{\boldsymbol{S}}$ denotes a rotated version of $\boldsymbol{S}$, and after each rotation, $\boldsymbol{S}$ is updated as $\boldsymbol{S}:=\overleftrightarrow{\boldsymbol{S}}$.

$$\boldsymbol{S} = \boldsymbol{VDV}^T \tag{12}$$
$$\boldsymbol{D} = \boldsymbol{V}^T\boldsymbol{SV} \tag{13}$$
$$\overleftrightarrow{\boldsymbol{S}} = \boldsymbol{Q}_i^T\boldsymbol{SQ}_i \tag{14}$$

$\boldsymbol{Q}_i$ is the Jacobi rotation during the i^{th} iteration, as in (15). The Jacobi rotation differs from the identity matrix only at p^{th}&q^{th} rows and columns. For i^{th} iteration, the pair (p, q) is selected according to the sequence of (II-B) or (17), which have a one-to-one correspondence. The completion of these $n(n-1)/2$ rotations is called a sweep, and Γ denotes the total number of the two-sided Jacobi rotations in one sweep.

$$\boldsymbol{Q_i} = \begin{bmatrix} 1 & \ldots & 0 & \ldots & 0 & \ldots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \ldots & c & \ldots & s & \ldots & \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \ldots & -s & \ldots & c & \ldots & \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \ldots & 0 & \ldots & 0 & \ldots & 1 \end{bmatrix} \begin{matrix} \\ \\ \leftarrow p \\ \\ \leftarrow q \\ \\ \\ \end{matrix}$$
$$\uparrow p \qquad \uparrow q \tag{15}$$
$$(p, q) = (1, 2), (1, 3), \ldots (1, n), (2, 3), \ldots (2, n), \ldots (n-1, n) \tag{16}$$
$$i = 1, 2, \ldots n-1, n, \ldots, 2n-2, \ldots \Gamma$$
$$whereas\ \Gamma = n(n-1)/2 \tag{17}$$

In the EJM, c&s are calculated using (18), where a_{pq} is the element of $\boldsymbol{S}$ from p^{th} row and q^{th} column. The result of the first sweep can be represented by $\boldsymbol{S}_\Gamma$, as in (19) and (20).

$$\theta_i = \frac{1}{2}tan^{-1}\frac{2a_{pq}}{a_{pp} + a_{qq}}, \quad c = cos\theta_i,\ s = sin\theta_i \tag{18}$$
$$\boldsymbol{S}_\Gamma = \left(\prod_{i=1}^{\Gamma}\boldsymbol{Q}_i^T\right)\boldsymbol{S}\left(\prod_{i=1}^{\Gamma}\boldsymbol{Q}_i\right) \tag{19}$$
$$\boldsymbol{V}_\Gamma = \left(\prod_{i=1}^{\Gamma}\boldsymbol{Q}_i\right) \tag{20}$$

$$\boldsymbol{S} := \boldsymbol{S}_\Gamma \tag{21}$$

$$\boldsymbol{D} = \boldsymbol{S}_\Gamma \text{ and } \boldsymbol{V} = \boldsymbol{V}_\Gamma \tag{22}$$

If $\boldsymbol{S}_\Gamma$ is not a diagonal matrix at the end of a sweep, then the next sweep is performed by updating $\boldsymbol{S}$ with $\boldsymbol{S}_\Gamma$, as in (21), and if $\boldsymbol{S}_\Gamma$ becomes a diagonal matrix, then (22) is used for finding $\boldsymbol{D}$ and $\boldsymbol{V}$ matrices.

C. CORDIC OR VOLDER'S ALGORITHM

CORDIC algorithm is a hardware-friendly iterative technique developed for implementing the elementary functions of mathematics by using only add and shift operations [26]. It can be used for computing sine, cosine, and arctangent in EVD by the EJM [32], [33]. If N is the total number of CORDIC iterations, which usually is equal to the wordlength of the hardware architecture, then mathematically CORDIC can be described using the equations from (23) to (26), where $i = 0, 1, 2, \ldots, N$ and $d_i = \pm 1$.

$$x_{i+1} = K_i \left(x_i - y_i.d_i.\ 2^{-i}\right) \tag{23}$$

$$y_{i+1} = K_i \left(y_i + x_i.d_i.\ 2^{-i}\right) \tag{24}$$

$$z_{i+1} = z_i - d_i.tan^{-1}(2^{-i}) \tag{25}$$

$$K_i = 1/\sqrt{1 + 2^{-2i}} \tag{26}$$

A lookup table with N entries is required to store the elementary angles, $\theta\,(i) = 45.00, 26.56, 14.03, \ldots, 0.0017$ (expressed in degree), for calculating d_i during each iteration. In vectoring mode, CORDIC can be used to compute $\theta_r = \tan^{-1}(Num/Den)$. During each iteration, d_i is selected such that $y_{i+1} \rightarrow 0$. In this mode, the initial and the final states of CORDIC are given by (27) and (28), respectively. In rotation mode, CORDIC can be used to compute $\sin(\theta_r)$ and $\cos(\theta_r)$ of an input angle θ_r. During each iteration, d_i is selected such that $z_{i+1} \rightarrow 0$. In this mode, the initial and the final states of CORDIC are given by (29) and (30), respectively.

$$x_0 = Den, y_0 = Num \text{ and } z_0 = 0 \tag{27}$$

$$x_N \approx \sqrt{x_0^2 + y_0^2}, y_N \approx 0 \text{ and } z_N \approx \theta_r \tag{28}$$

$$x_0 = {}^1\!/_{A_N}, y_0 = 0 \text{ and } z_0 = \theta_{rot} \tag{29}$$

$$x_N \approx \cos(\theta_r)\,, y_N \approx sin(\theta_r) \text{ and } z_N \approx 0 \tag{30}$$

D. EVD BY THE APPROXIMATE JACOBI METHOD

We know that the symmetric matrix $\boldsymbol{S}$ is rotated by using the Jacobi rotations to annihilate its off-diagonal elements a_{pq} and convert it to a diagonal matrix $\boldsymbol{D}$. But complete annihilation is neither possible nor mandatory because the annihilation efforts from one rotation are destroyed by the subsequent rotations during each sweep except for the last few sweeps before the convergence. This observation leads to a new method called the AJM [27], which is faster and more efficient than the EJM, as reported by [28]. According to the AJM, any Jacobi rotation which satisfies the condition in (31) can be used for annihilating a_{pq}. In other words, the AJM computes c *and* s approximately.

$$a_{pq} := da_{pq} \text{ whereas } 0 \leq |d| < 1 \tag{31}$$

$$a_D = a_{qq} - a_{pp} \tag{32}$$

$$\tau = \frac{a_D}{2a_{pq}} \tag{33}$$

$$k = exp\left(|a_D|\right) - exp(\left|a_{pq}\right|) \tag{34}$$

$$\left(2^l - 2^{-l+1}\right)|a_D| \leq |a_D| + 2^{-1}|a_D| < \left(2^{l+1} - 2^{-l}\right)|a_{pq}| \tag{35}$$

$$l = \begin{cases} 0 & if\ k \leq -2 \\ 0, 1 & if\ k = -1, 0 \\ k-1, k, k+1 & if\ k > 0 \end{cases} \tag{36}$$

Here, d in (31) represents the depth of the annihilation, and $exp()$ in (34) represents the exponent of a number when it is expressed in base-2 scientific notation. In this method, k is calculated using (34), and then the possible values of l are chosen using the conditions in (36). The comparison operations of (35) are used to select a specific value for l. For avoiding the square root operation in the scaling factor, l is incremented as in (37), whereas (38) is used for computing approximate values of c and s. Similarly, for avoiding the division operation of the scaling factor K_l^2, mathematical recursion is used, as in (40). The inequality in (41) represents the accuracy limit of fixed-point hardware architecture with a wordlength of w-bit.

$$l := l + 1 \tag{37}$$

$$c \approx K_l^2\left(1 - 2^{-2l}\right), \quad s \approx sign\,(\tau)\,K_l^2 2^{-l+1} \tag{38}$$

$$K_l^2 = \frac{1}{1 + 2^{-2l}} \tag{39}$$

$$K_{i+1}^2 = K_i^2(1 + 2^{-2^{i+1}l}), K_1^2 = 1 - 2^{-l} \tag{40}$$

$$2^{i+1}l \leq w \tag{41}$$

So, using the AJM, the division and the square root can be avoided but at the expense of losing quadratic convergence to almost linear convergence for EVD. For the AJM, the maximum absolute annihilation has an upper bound of 0.6.

E. SYSTOLIC PROCESSING ARRAY FOR EVD

The Jacobi rotation $\boldsymbol{Q_i}$ only affects the elements of $\boldsymbol{S}$ from $p^{th}\&p^{th}$ rows and columns. For parallelizing the computations of the Jacobi rotations $\boldsymbol{Q_i}$ on a SPA, 2×2 matrices $\boldsymbol{Q_i^{sub}}, \boldsymbol{V^{sub}}$, and $\boldsymbol{S^{sub}}$ are introduced, as in (42). A SPA of $n/2 \times n/2$ processing elements (PEs), as shown in Fig. 2, is used for performing $n/2$ disjoint Jacobi rotations (which do not share any row or column) in parallel as in [22]. A sweep is completed after performing n-1 steps of the $n/2$ disjoint rotations. The diagonal processing elements (DPEs) on the systolic array compute $c\&s$ using (18) or (38) and then rotate their elements by $\boldsymbol{Q_i^{sub}}$. The non-diagonal processing elements (NPEs) only rotate their elements by the rotation parameters provided on their dotted-line inputs by the NPEs. After each step of the $n/2$ disjoint rotations, row-wise and

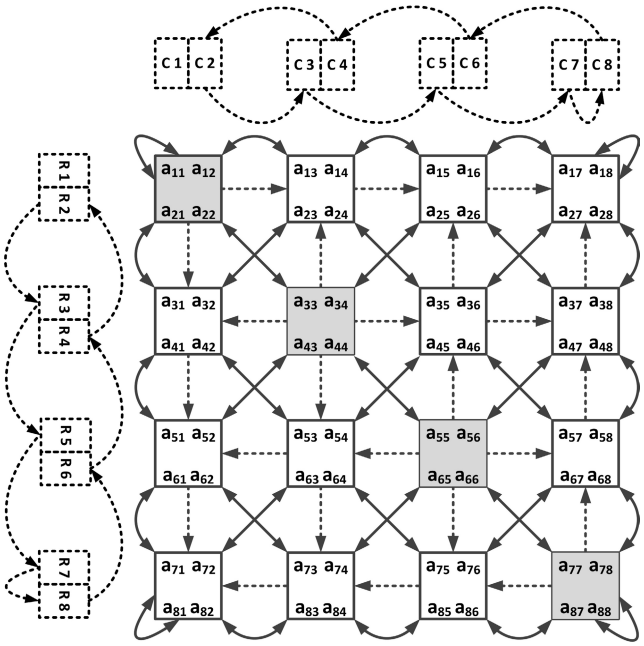

FIGURE 2. A Systolic Processing Array for EVD of an 8 × 8 Matrix.

column-wise swapping of the elements of $\boldsymbol{S}$ is performed according to the patterns shown on the top and left of Fig. 2. After the performing the swapping for n-1 times, the elements come to their original position. The crossed diagonal interfaces between the PEs are used for the swapping of the elements. In this way, we get eigenvalues and eigenvectors of $\boldsymbol{S}$.

$$\boldsymbol{Q}_i^{sub} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}, \quad \boldsymbol{S}^{sub} = \begin{pmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{pmatrix}, \; \boldsymbol{V}^{sub} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{42}$$

$$\begin{aligned} a_{pp} &:= c^2 a_{pp} + s^2 a_{qq} - cs(a_{pq} + a_{qp}) \\ a_{pq} &:= c^2 a_{pq} - s^2 a_{qp} + cs(a_{pp} - a_{qq}) \\ a_{qp} &:= c^2 a_{qp} - s^2 a_{pq} + cs(a_{pp} - a_{qq}) \\ a_{qq} &:= c^2 a_{qq} + s^2 a_{pp} - cs(a_{pq} + a_{qp}) \end{aligned} \tag{43}$$

$$\boldsymbol{V}^{sub} := \begin{pmatrix} ca_{pp} - sa_{pq} & ca_{pq} + sa_{pp} \\ ca_{qp} - sa_{qq} & ca_{qq} + sa_{qp} \end{pmatrix} \boldsymbol{V}^{sub} \tag{44}$$

F. PROBLEM FORMULATION

The EJM and the AJM have been proposed for computing eigenvalues and eigenvectors using shift and add operations. These operations are the only options for implementing mathematical functions in the first generation of FPGAs [22], [24], [27]. With the availability of hardware-based multipliers (in the form of DSP blocks) in the second/third generations of FPGAs and the popularity of ASIC-based designs, the scaling and the rotation operations of both methods are implemented using the hardware multipliers [28], [33]. To implement the EJM on the SPA, each DPE needs to be equipped with two CORDIC [34] blocks for computing $\theta_i, c,$ *and* s as in (18). CORDIC algorithm can be realized with shift and add operations but it has a slow convergence rate. To overcome this issue, unrolled CORDIC [35], parallel CORDIC [36], or pipelined CORDIC blocks are introduced but the EJM cannot be implemented using these blocks on the SPA, at least under the constraints originating from real-time portable systems. Non-systolic architectures for the EJM [13], [37] are also efficient with a small channel count but scalability cannot be achieved with such arrangements. Systolic architectures for the AJM using hardware multipliers for the scaling operation of (39) are also used to implement FastICA [28], [20]. But for avoiding the square root of the scaling operation on the SPA, K_l^2 is computed by recursion, as in (45). For calculating l, as in (36), in a single clock cycle a lot of variable-shifters and priority decoders along with some adders are required within each DPE. So, it is difficult to bring the required hardware efficiency even with the AJM. According to [28] the AJM is superior to the EJM in terms of efficiency (eigenvalues produced per second per unit chip area) but both methods are still in use [23], [20], [38], [39] for implementing FastICA on FPGAs or ASICs. In the AJM, c *and* s are approximated to the closest CORDIC iteration values, that is why the existing two methods are considered as CORDIC-based methods.

$$K_l^2 = \left(1 - 2^{-2l}\right)\left(1 + 2^{-4l}\right)\left(1 + 2^{-8l}\right)\left(1 + 2^{-16l}\right) \quad when\ l = 0, K_l^2 = 1/2 \tag{45}$$

ASIC-based implementations of FastICA can be optimized with respect to speed or power [32] and hardware complexity can be neglected to some extent because even a complex design can be optimized for power by carefully selecting technology node and operating voltage as in [11], [13]. ASIC-based designs are useful at the commercialization stage. But from [9], we know that 65% of AI algorithms are being implemented on FPGAs because of their flexibility and rapid prototyping capability. With the availability of low-cost and high-performance FPGAs [40], we can also use them at the commercialization stage after optimizing the algorithms for power. In the EJM and AJM based implementations of FastICA, the nonreusable EVD stage makes the implementations area-inefficient and power-hungry (because of the quiescent current flowing through this stage) [41]. Based on this observation we are going to propose a new method, called the ALJM, for computing EVD for FastICA. Th ALJM also leads to a new scalable efficient SPA for FastICA.

III. INTRODUCING THE ALJM

The EJM, the AJM, and the ALJM differ in the way they compute the values of sine and cosine of the rotation angle, as in (18), (38), and (46). These values should be computed such that the Jacobi rotation remains an orthogonal transformation (in other words, $c^2+s^2 = 1$). The EJM uses CORDIC block, and its outputs satisfy this condition. The AJM and the ALJM approximately compute the tangent of the angle, as in (46). This way helps in orthogonalizing the approximate Jacobi rotation, as explained in [27]. The relation between

$tan\theta_i$ and τ is shown in (47). From (33) we can see that τ keeps growing in magnitude during the EVD computation because a_{pq} is annihilated in each step. This observation leads to the simplification in (48). The quantities e and f_r of (49) can be easily computed at hardware level using a cascade connection of priority decoder and variable left shifter.

$$t \approx tan\theta_i, \quad c \approx \frac{1}{\sqrt{1+t^2}}, \ \mathrm{s} \approx ct \tag{46}$$

$$tan\theta_i = \frac{sign\,(\tau)}{|\tau| + \sqrt{1+\tau^2}} \tag{47}$$

$$t \approx \frac{sign\,(\tau)}{2\,|\tau|} \approx \frac{a_{pq}}{a_D} \tag{48}$$

$$e = exp(a_D), \quad f_r = 2^{-e}a_D - 1 \tag{49}$$

$$P = 0.3275{f_r}^2 - 0.8042f_r + 0.9861 \tag{50}$$

$$t_r = 2^{-e}a_{pq}P \tag{51}$$

$$t \approx \begin{cases} 1 & if\ t_r \geq 1 \\ t_r & otherwise \end{cases} \tag{52}$$

After computing P of (50), t can be computed using (51) and (52) and this is a very good approximation for $tan\theta_i$, as compared to the approximation used in the AJM, because it keeps the convergence rate almost intact. Then the inverse square root by the Newton-Raphson method [42], described from (53) to (55), is used to compute $c\ and\ s$, as in (56). This is how we can avoid the square root and division operations in a non-CORDIC implementation of EVD and the same set of multipliers can be used for calculating the Jacobi rotation entries, the scaling operation, and performing the two-sided rotation. In the ALJM, s and c can be computed within 7 clock cycles including the five iterations of the Newton-Raphson method. The AJM also takes 6 clock cycles for computing s and c because it performs the comparison operations of (35) and the recursion of (40). But the AJM uses a lot of reconfigurable logic for computing $c\ and\ s$ and the DSP blocks for the scaling whereas the ALJM uses the DSP blocks for both the tasks.

$$x = 1 + t^2 \tag{53}$$

$$x_0 = 2^{-exp(x)/2} \tag{54}$$

$$x_1 = x_0\frac{x}{2}, x_0 := x_0(1.5 - x_0x_1) \tag{55}$$

$$c = x_0\ and\ s = ct \tag{56}$$

Almost all modern FPGAs are equipped with such DSP blocks [40], [43], [44]. These blocks can be operated at frequencies greater than 300MHz and the number of the blocks can vary from 10 to 3600 within a single FPGA. They have a 25×18 or 18×18 multiplier followed by a 48-bit accumulator which ensures accuracy and overflow control. The blocks can operate in an asynchronous mode, or in pipelined mode using the built-in high-speed registers. The DSP blocks are more optimized for speed and power than the configurable logic blocks (CLBs) and transferring the computational load of any algorithm to these units results in a good balance of operating speed and power consumption. So, we can expect that the ALJM-based FastICA, which engages the blocks during EVD and WVE, can beat the existing methods with respect to area, speed, and power. The speed performance of the ALJM is expected because it retains the quadratic convergence of the EVD stage, contrary to the AJM which has almost linear convergence.

Modern FPGA devices have all the necessary features to implement a SPA which can be used for both EVD and WVE stages. The advanced silicon modular block (ASMBL) is adapted to overcome geometric layout constraints for combining blocks of different functionality. In ASMBL, DSP blocks are stacked in a column-wise arrangement, which is parallel as well as adjacent to Block RAM column-wise arrangement. Moreover, in case of higher dimensionality, a super logic region (SLR), containing DSP blocks, can be connected to the neighboring SLRs using super logic lines (SLL) in Xilinx 7-Series FPGAs as explained in [45]. The SLL has very low latency compared to the reconfigurable logic.

IV. PROPOSED HARDWARE ARCHITECTURE

In the previous section, we have demonstrated how the whole EVD stage can be mapped to a SPA based on the DSP blocks of modern low-cost FPGAs. For performing both the EVD and the WVE stages on a single SPA, which is the primary purpose of the current work, we also need to simplify the operations of the WVE stage. Therefore, the following subsections are dedicated to implementing both the stages using a two-dimensional array of DSP blocks connected by a network of data buses. Moreover, we are currently targeting low-cost FPGAs for the ALJM-based FastICA, but the concept is equally valid for ASIC platforms which will be considered in the future. We need to replace the DSP blocks with fixed-point multiply-accumulate (MAC) IPs and the Block RAM (BRAM) modules with dual-port flipflop-based memory.

A. TOWARDS A SPA FOR FastICA

For the implementation of the WVE stage of FastICA, we need the three types of matrix multiplications, as in (57), where the subscripts are showing the dimensions of the matrices involved, $\mathbf{P}$ denotes the product matrix, $\mathbf{L}$ denotes the left-hand side matrix, and $\mathbf{R}$ denotes the right-hand side matrix. Calculating $\tanh x$ at high-speed is also required in the WVE stage. As shown in (58), after piecewise linearization, this function can be implemented by using some comparators and the DSP blocks of FPGAs. The comparators are required for selecting the values of $A\,(x)$ and $B(x)$ based on the value of x.

$$\boldsymbol{P}_{n\times m} = \boldsymbol{L}_{n\times n}\boldsymbol{R}_{n\times m}, \quad \boldsymbol{P}_{n\times n} = \boldsymbol{L}_{n\times m}\boldsymbol{R}_{m\times n}$$
$$\boldsymbol{P}_{n\times n} = \boldsymbol{L}_{n\times n}\boldsymbol{R}_{n\times n} \tag{57}$$

$$tanhx = A\,(x)\,x + B(x) \tag{58}$$

In addition, there are the two *norm* operations followed by the two *division* operations for the WVE stage, as in (8)

and (9). The simulation trials performed during the theoretical evaluation of FastICA, as described in the next section, show that the two *norm* operations be simplified, as in (59) and (60), without any significant effect on the separation quality of the extracted signals $\hat{\boldsymbol{S}}$ of (2).

$$\boldsymbol{w_i} := \boldsymbol{w_i}/max(abs(\boldsymbol{w_i})) \tag{59}$$

$$\boldsymbol{W} := \boldsymbol{W}/max(abs(\boldsymbol{W}\boldsymbol{W}^T)) \tag{60}$$

We can also replace the *division* operations, as verified in the next section using the fixed-point mathematical models for FastICA, with the shift operations as shown in (61) and (62) without significantly degrading *CCC* between the actual sources and the extracted sources. So, we can approximately perform the divisions by shifting the input operand by the binary places equal to the output of a signed priority encoder. Now we can describe the design of the SPA for FastICA based on the ALJM.

$$\boldsymbol{w_i} := \boldsymbol{w_i}/2^{exp(max(abs(\boldsymbol{w_i})))} \tag{61}$$

$$\boldsymbol{W} := \boldsymbol{W}/2^{exp(max(abs(\boldsymbol{W}\boldsymbol{W}^T)))} \tag{62}$$

B. THE PROPOSED SPA FOR FastICA

The proposed SPA for performing 4-channel FastICA is shown in Fig. 3. This architecture consists of a 2-D array of processing elements labeled with their position (PEij), a 1-D array of BRAM modules for each channel, a network of the bidirectional buses, and a distribution of the configurable connectors for routing one bus to another bus. The BRAM modules are labeled (BRAM Chi) according to the channel they store. Each bus is represented by a different line style and has a name like B_{XYZ} which describes its functionality, listed below the architecture. There are three types of the configurable connectors and each type is represented by a different shape. The PEs on the main diagonal (DPEs, colored in light green) and the off diagonal/non-diagonal PEs (NPEs, colored in light orange) have different functionality and architecture.

The NPEs have a built-in controller whereas the NPEs only have an instruction decoder. The controller inside each DPE dictates the operation of all the NPEs in the same column. It also controls the BRAM module for the corresponding channel. The NPEs of each column receive instructions from the corresponding controller, decode them, and act accordingly. The instructions specify the operation (addition, multiplication, or MAC) and the operands involved. The bus named B_{NPE} is used to carry these instructions from a DPE to the NPEs in the same column.

The buses named B_{VMM} and B_{HMM} are used for implementing the matrix multiplications. DSP blocks inside each PE are used for performing the MAC operations during the matrix multiplications. The multiplications are performed over the data from the BRAM modules or distributed RAM (DRAM, implemented using LUTs) inside each PE. The bus B_{TMM} is used to implement multiplication involving transpose of a matrix such as the multiplications during the symmetric decorrelation of FastICA. The B_{MAC} is used for cascading the outputs of the DSP Blocks for performing the decomposition operations of the EVD and the WVE stages (decomposing $\boldsymbol{X}$ and $\boldsymbol{Z}$ using $\boldsymbol{H}$ and $\boldsymbol{W}$ respectively). The two buses B_{MCV} and B_{MVM} are used for the sorting operations performed on $\boldsymbol{W}$. B_{MCV} is used for directing the maximum value of a column vector (such as $\boldsymbol{w_i}$ during the *norm* or normalization step) to the DPE for normalizing each column vector whereas B_{MVM} is used for directing the maximum value of the matrix (such as $\boldsymbol{W}$) to the DPEs so that the matrix can be normalized as in (62).

The configurable connectors represented by the circles are used for directing B_{HMM} to B_{VMM} during the covariance matrix calculation as in (4) and (7). The configurable connectors represented by the triangles are used for directing input or output ports of the RAM modules to B_{HMM} during the covariance and data whitening operations whereas the configurable connectors represented by the squares are used for directing B_{TMM} to B_{VMM} during the symmetric decorrelation operation, as in (9) and (10). The buses connected to the PEs through double-headed arrows are bi-directional buses (they carry both the inputs and the outputs to/from the PEs). Some hardware platforms, such as modern FPGA, cannot synthesize bi-directional signals and use multiplexers to implement bidirectional logic as shown in Fig. 4.

C. THE ARCHITECTURE FOR DPE

The architecture of a DPE and its connections to the surrounding buses is shown in Fig. 5(a). The control unit block not only controls the operation of all components inside the DPE, but it also dictates the operation of the NPEs within the same column, as shown by its connection to B_{NPE}. Three multiplexers are used for directing different operands to the DSP block. The operands can come from the DRAM block or through the surrounding buses. Dedicated lines are used for directing 1 and 0 to multiplier and adder of the DSP block, respectively, because of their frequent use. The DRAM block stores the linearization coefficients of tanh() function, the polynomial coefficients used in (50), one element from each of $\boldsymbol{D}$, $\boldsymbol{V}$, and $\boldsymbol{W}$, and intermediate variables generated while performing FastICA. The signed priority encoder (which operates on the absolute value of its input), variable shifter (which shifts its input in the direction and the position determined by its control input) and Rounding & Truncation block are used for generating initial guess for the square root operation during data whitening. These blocks are also used for performing the inverse square root operation during EVD. The data from the DSP block and the surrounding buses can be stored in the DRAM block. The circle having symbol >= is showing the sorting operation performed on the input data from B_{MCV} and the element of $\boldsymbol{W}$ stored in the DRAM block. The tanh() Select block computes the region of linearization for tanh() function and the control unit selects the corresponding coefficients from the DRAM block. Abs() and 2sC() blocks are used to take absolute and 2's complement (whenever it is required, otherwise they are bypass by the control unit) of the contents addressed in the DRAM block, respectively. The size

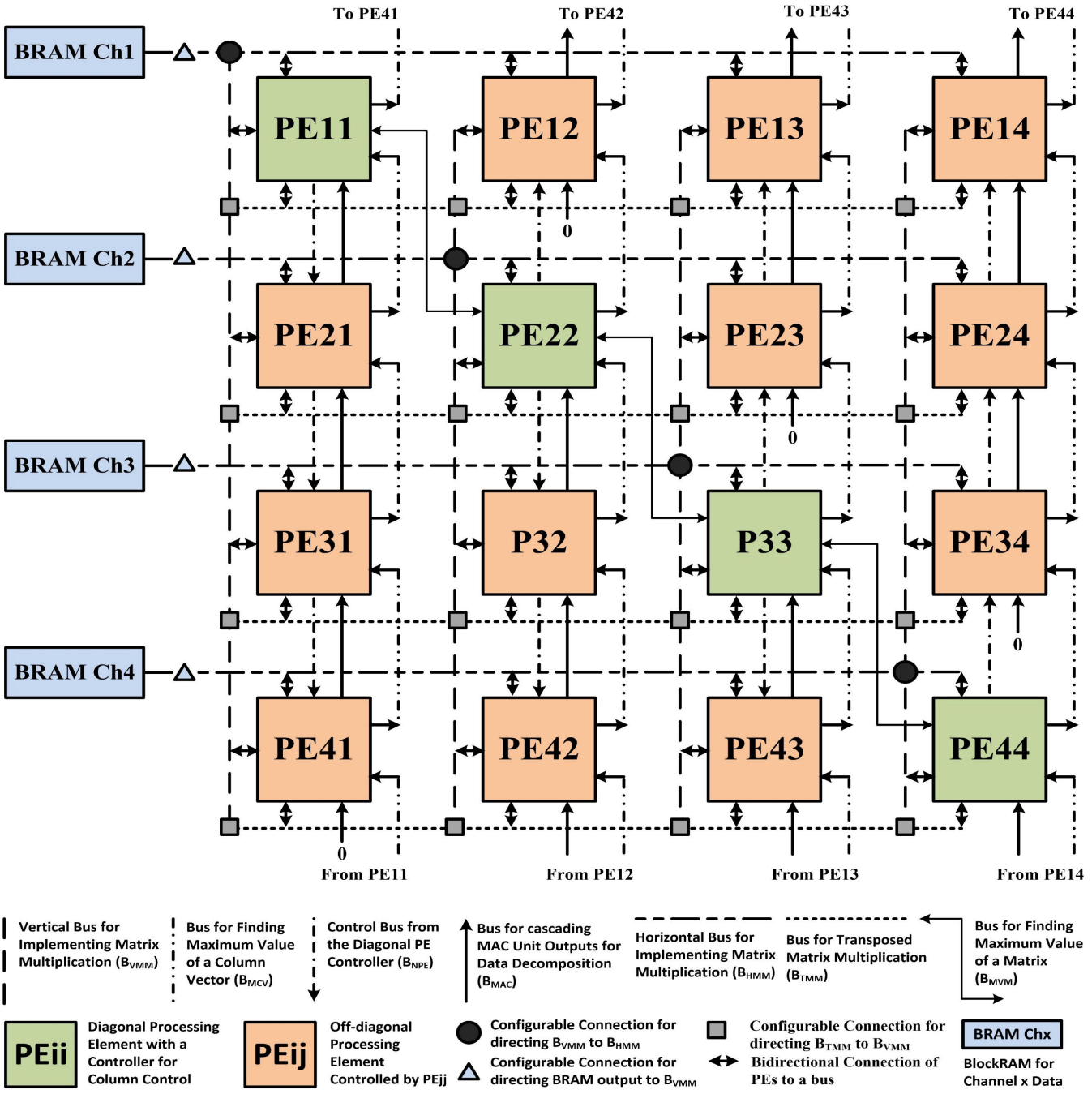

FIGURE 3. Architecture of the Proposed Systolic Processing Array (SPA) for implementing the ALJM-Based FastICA in Hardware (Spartan-6).

of the DRAM block in the case of the ALJM-based FastICA is typically 16×18.

D. THE ARCHITECTURE FOR NPE

The architecture of NPE is a simplified version of the DPE architecture. The NPEs run in slave mode and the instructions are decoded by the Command Decoder block, as in Fig. 5(b). The size of the dual-port RAM block in the case of the ALJM-based FastICA is typically 4×18. Both the types of PEs can direct data from B_{HMM} and B_{VMM} to the DSP block and place the addressed content of the DRAM block to B_{TMM}.

E. BRAM AND ITS INTERFACING

Symmetric decorrelation of FastICA is redefined, as in (63), in terms of matrix multiplications. This redefinition helps understanding memory allocation and mapping FastICA on the SPA. There is a BRAM module for every channel on the SPA. The size of the module is 1024×18 for calculating FastICA of 512 samples. Initially, one channel of multivariate signal $\boldsymbol{X}$ is stored in the first 512 locations of the module. Then during the centering operation, each sample of $\boldsymbol{X}$ is replaced by $\tilde{\boldsymbol{X}}$. During data whitening, matrix $\tilde{\boldsymbol{X}}$ is replaced by $\boldsymbol{Z}$ in the module. The last 512 locations of the BRAM

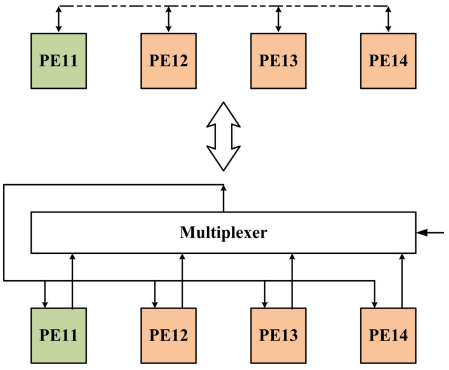

FIGURE 4. Implementing a Tri-State bus in Modern FPGAs.

modules are used for storing $\boldsymbol{H}$. After the convergence of $\boldsymbol{W}$, $\hat{\boldsymbol{S}}$ is estimated and it replaces $\boldsymbol{H}$ at the last 512 locations. At the end of the computations, we have $\boldsymbol{Z}$ at the first 512 locations and $\hat{\boldsymbol{S}}$ at the last 512 locations of the module. So, we can say that the SPA for the ALJM-based FastICA is memory efficient as compared to the existing architectures.

$$\begin{aligned} \boldsymbol{W} &:= \frac{1}{m}\left(\boldsymbol{Z}\boldsymbol{H}^T - diag\left(\sum\left(1-\boldsymbol{H}^2\right)\right)\boldsymbol{W}\right) \\ \boldsymbol{H} &= tanh\left(\boldsymbol{W}^T\boldsymbol{Z}\right) \end{aligned} \tag{63}$$

F. MAPPING EVD ON THE SPA

The SPA for computing EVD, as shown in Fig. 2, can be merged into the SPA for FastICA, as shown in Fig. 3. The swapping operations and data exchange between the PEs can be performed using the buses such as B_{VMM}, B_{HMM}, and B_{TMM}. For computing s *and* c, we divide SPA for FastICA into the groups of four PEs labeled as PEii, PEij, PEji, and PEjj whereas i = 1, 3, 5, …Cc and j = 2, 4, 6, …Cc. Here Cc is the channel count for FastICA and the first value of 'i' and 'j' select the first group and the second value selects the second group and so on. During the first cycle of the EVD stage, each group is configured as shown in Fig. 6(a). In this configuration, f_r^2 and $2^{-e}a_{pq}$ of (51) are computed and stored in the DRAM block of PEii. In the second cycle, each group is configured as shown in Fig. 6(b). The stored variables from the previous clock cycle are used to calculate the initial guess for the inverse square root of (55), and the guess is stored in the DRAM block. During the next five clock cycles, this initial guess is updated, as shown in Fig. 6(c), and finally, we get the inverse square root. In this way, each group computes s *and* c for performing the Jacobi rotation.

The first group performs the Jacobi rotation on the 1st & 2nd rows and columns of the SPA during the next two clock cycles and updates the corresponding elements of $\boldsymbol{D}$ and $\boldsymbol{V}$ for computing eigenvalues and eigenvectors. The elements a_{ii}, a_{ij}, a_{ji} *and* a_{jj} of any group takes six additional clock cycles for completing update operation, meanwhile, the second group starts updating the 3rd & 4th rows and columns for the subsequent two clock cycles. In this way, the Jacobi rotations are computed in parallel and performed in pipelined mode. When all the groups complete the rotations by updating their elements, the swapping operation is performed using the relevant buses and the next step of the ongoing sweep is performed.

V. THEORETICAL PERFORMANCE EVALUATION

Usually, hardware platforms execute algorithms using fixed-point representation. But the development of such algorithms is carried out using a floating-point representation on a numerical computing environment such as MATLAB [46]. Fixed-point Designer [29] is a MATLAB tool, which facilitates optimizing and evaluating the hardware-oriented algorithms. This tool moves the analysis from the hardware domain to the software domain hence saving time and effort. It offers customizable datatypes and configurable modes for mathematical and logical operations. We use this tool, contrary to the analysis carried out in [47], [48], for evaluating and comparing the theoretical performance of the ALJM-based FastICA with the existing methods. This performance counts for functional accuracy, numerical stability, convergence rate, and quality of separation. The settings of the tool are customized to match the properties of the arithmetic operations performed on the targeted hardware. For example, the tool offers the two options, *'Round'* and *'Zero'* for truncation. We choose *'Zero'* because in hardware, truncation is done by dropping the extra bits. Rounding fractions to the nearest integer needs additional hardware, and usually, it is not implemented for fixed-point computations. Similarly, the tool also offers the two actions, *'Wrap'* and *'Saturate'* for overflow. But we choose *'Wrap'* because during the fixed-point computations, the results are wrapped at overflows. Moreover, the precision for both sum and product modes is specified according to the widths of the adder and multiplier in the DSP block.

A. EVALUATION SETUP AND BENCHMARKING

We evaluate the proposed and the existing algorithms under the same environment with the same datasets. We develop parameterized fixed-point software models of the three algorithms so that their sensitivity to the parametric variations can be evaluated before implementing them on actual hardware. In real-time applications that perform FastICA such as BCIs, the latent sources are supposed to be non-Gaussian. So, we generate non-Gaussian random signals as the source signals $\boldsymbol{S}$, as in (1), and then create their linear mixtures, represented by $\boldsymbol{X}$, as in (1). EVD, whitening, and FastICA are performed separately using each of the three algorithms. We evaluate and compare the algorithms by measuring their average performance over 50 trials.

B. INTEGER LENGTH OPTIMIZATION

In the fixed-point implementation of an algorithm, wordlength (*WL*) is defined as the sum of inter length (*IL*) (including the sign bit) and fraction length (*FL*). *FL* is related to the computational accuracy of the algorithm. For minimizing the required hardware resources, knowing the

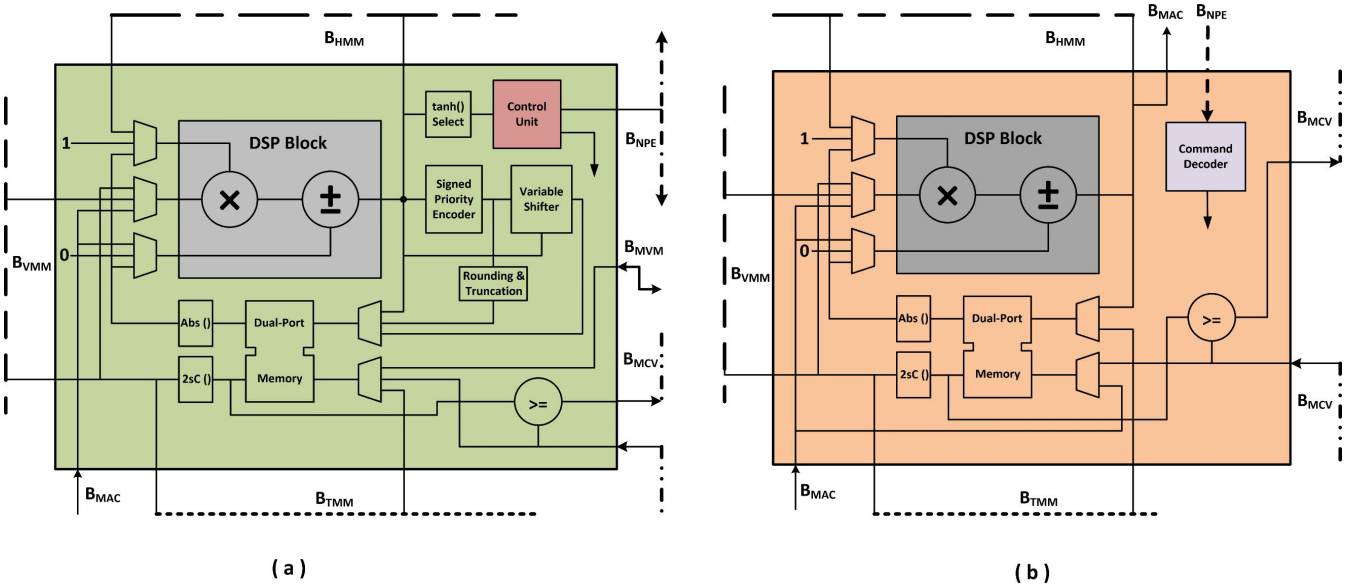

FIGURE 5. Digital Architecture for DPE and NPE of the SPA used for Implementing the ALJM-based FastICA.

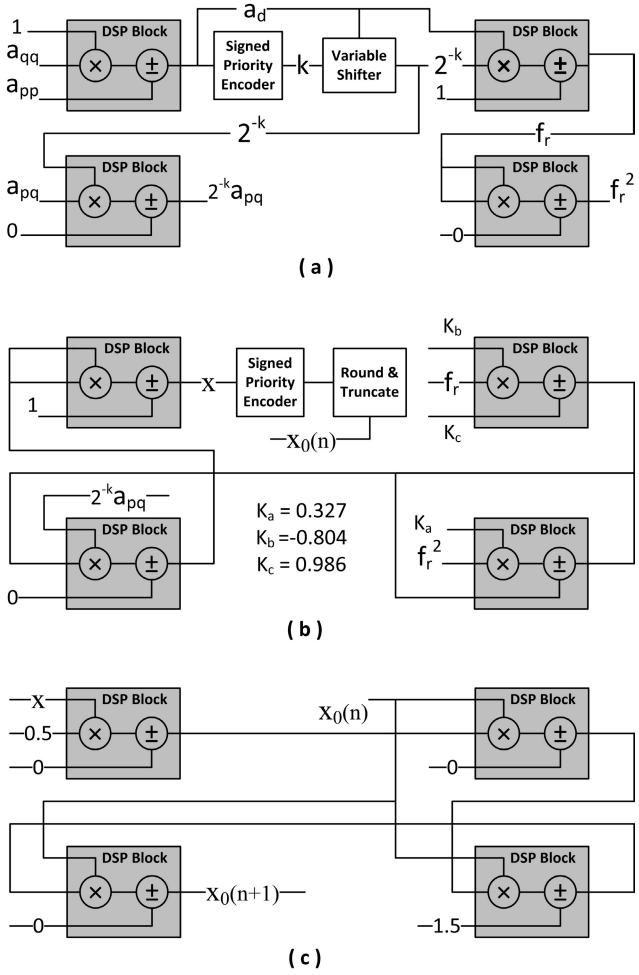

FIGURE 6. Configurating the SPA to Compute the ALJM-based EVD.

bare minimum *IL* is the first step towards the optimization of an algorithm. We use '*buildInstrumentedMex*' and '*showInstrumentationResults*' features of Fixed-point Designer for optimizing *IL* for the proposed method. After scaling the covariance matrix $\boldsymbol{C_{XX}}$ of (5) so that its elements lie in the range $(-1, +1)$ inclusive, $IL = 4$ is recommended by the tool over the 50 trials of simulation for the EVD stage, and $IL = 1$ is recommended for the WVE stage for increasing the separation quality.

C. DEFINING PERFORMANCE METRICS

From the simulation results of the algorithms with Fixed-point Designer, we note that the amount of error in each eigenvalue depends on its magnitude. So, for measuring the quality of the eigenvalues $\hat{\sigma}_i$ computed by the algorithms, we use the relative error in each eigenvalue, as defined in (64), where n is the order of C_{XX}. Reference for the relative error is the eigenvalues σ_i computed by MATLAB 2020b *eig* function.

$$max[100 \times abs(\sigma_i - \hat{\sigma}_i)/\sigma_i] \quad i = 1, 2, \ldots .n \tag{64}$$

As we know that eigenvalues are unique, but the corresponding eigenvectors can vary with the computation tools and methods. But they are always orthogonal to each other. So, we decide to measure the quality of their orthogonality. So, the percent error in orthogonality between any two eigenvectors $\hat{\upsilon}_i$ and $\hat{\upsilon}_j$, computed by any of the three algorithms, is defined in (65).

$$100 - min[100(abs(\hat{\upsilon}_i.\hat{\upsilon}_i - \hat{\upsilon}_i.\hat{\upsilon}_j)/\hat{\upsilon}_i.\hat{\upsilon}_i] \quad i = 1, 2, \ldots .n, \quad j = 1, 2, \ldots n, i \neq j \tag{65}$$

After computing the EVD of $\boldsymbol{C_{XX}}$, the inverse square roots of the eigenvalues $\hat{\sigma}_i$ are calculated to obtain the whitening matrix $\boldsymbol{H}$, as in (6). In each of the algorithms, the inverse square roots are calculated using the Newton-Raphson method. The whitening matrix $\boldsymbol{H}$ is used to obtain

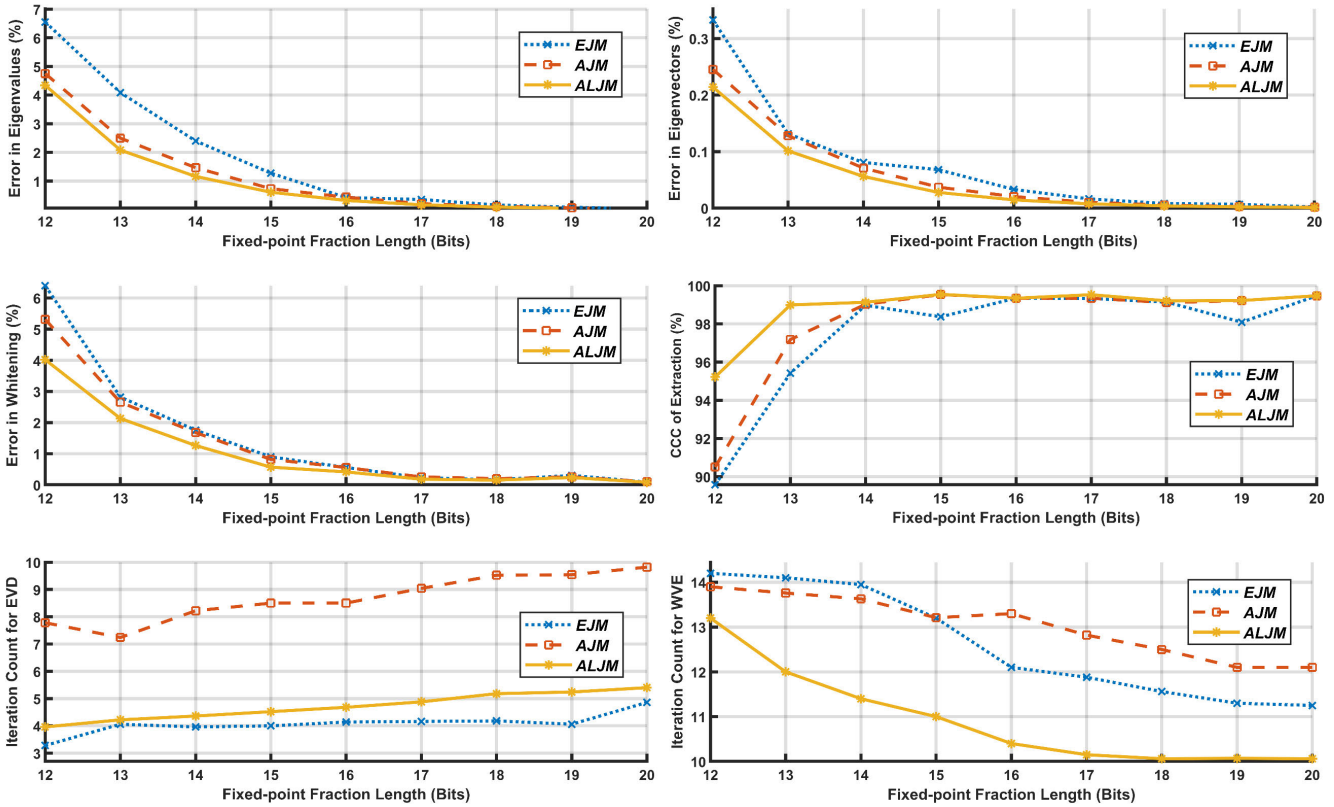

FIGURE 7. The relative Errors and the Iteration Count at Different Fraction Length (*FL*) for 8-channel the ALJM-based FastICA.

zero-mean unit variance (whitened) data matrix $\boldsymbol{Z}$, as in (6). After this, $\boldsymbol{C_{XX}}$ for $\boldsymbol{Z}$ is computed with each of the algorithms. Ideally, $\boldsymbol{C_{XX}}$ should be the identity matrix, but due to the quantization in the fixed-point implementations, it is only close to the identity matrix. So, for comparing the quality of the whitening process, we use (66) to compute percent errors, where $\hat{c}_i$ and $\hat{c}_j$ are any two columns of $\boldsymbol{C_{XX}}$.

$$\begin{aligned} &100 - min[(abs(\hat{c}_i.\hat{c}_i - \hat{c}_i.\hat{c}_j)/\hat{c}_i.\hat{c}_i] \\ &i = 1, 2, \ldots .n, \quad j = 1, 2, \ldots n, i \neq j \end{aligned} \tag{66}$$

The quality of separation is estimated by calculating the cross-correlation coefficient $CCC(x, y)$ between the input source signals and the extracted signals. This coefficient is calculated as in (67), where x is the input signal and y is the estimated signal. When $CCC\ (x, y) = 100$ then the source signal and the extracted signal are identical.

$$CCC(x, y) = \frac{100}{n-1} \sum_{i=1}^{n} \left(\frac{x - \mu_x}{\sigma_x}\right)\left(\frac{y - \mu_y}{\sigma_y}\right) \tag{67}$$

Each of the three algorithms computes $1/\tau$, directly or indirectly, as defined in (33) before performing the Jacobi rotation. When this factor is equal to zero (or less than the accuracy of the fixed-point representation used) then the Jacobi rotation becomes a unity matrix. During the 50 trials, the maximum absolute value for a_D is always less than 4. In (68) and (69) we calculate the annihilation limit of a_{pq} which is set as the convergence limit for the EVD stage.

$$1/\tau = \frac{2a_{pq}}{a_D} \leq 2^{-FL} \tag{68}$$

$$a_{pq} \leq 2^{-FL+1} \tag{69}$$

During the WVE stage, the convergence of weight vectors matrix $\boldsymbol{W}$ is defined in (70) and (71) where $\boldsymbol{W_i}$ is weight vectors matrix during ith iteration. Because of the normalization, as in (8) and (9), elements of $\boldsymbol{W_i}$ are always less than unity. So, we can say that $\boldsymbol{W}$ converges within 1% and this is enough to ensure a cross-correlation coefficient of 99.8%, as observed during the trials.

$$\boldsymbol{\Delta W_i} = \boldsymbol{W_i} - \boldsymbol{W_{i-1}} \tag{70}$$

$$max\ (\boldsymbol{\Delta W_i}) \leq 0.01 \tag{71}$$

D. EFFECTS OF LIMITED WORDLENGTH

Currently, most of the FPGAs have built-in 18 × 18 multipliers. In addition to this, intellectual properties (IPs) for 16-bit to 24-bit ASIC designs are also available on every technology node. As we know that $IL = 4$ is recommended by Fixed-point Designer, so we decide to vary *FL* from 12-bit to 20-bit for analyzing the algorithms over *WL* of 16 to 24 bits. In Fig. 7, we plot the mean value of the three errors (defined in (64), (65), and (66)), *CCC* (defined in (67)), and the two iteration counts (defined in (69) and (71)) against the different values of *FL* for the 50 trials performed with each of the three

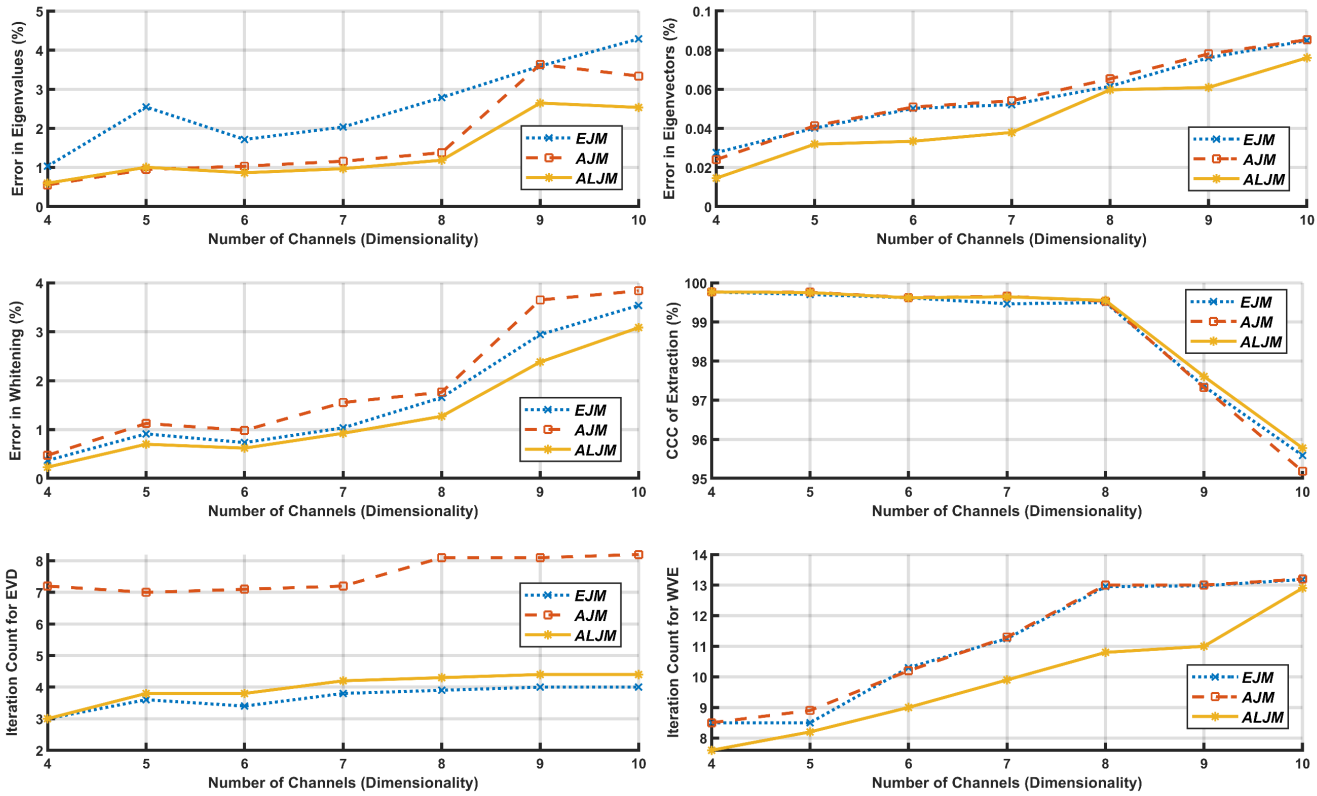

FIGURE 8. The relative Errors and the Iteration Count at Different Dimensionality (n) for 8-channel the ALJM-based FastICA.

algorithms. In each trial, we use eight different non-Gaussian random source signals $\boldsymbol{S}$ and mix them using a different randomly generated mixing matrix $\boldsymbol{A}$ of order 8×8. We observe that the EJM is performing the worst over the initial values of *FL*, contrary to the expectations. The reason for such behavior is that the performance of CORDIC, the algorithm used by the EJM, is poor over these values of *FL*. Orthogonality error in the computation of *c and s* is the main reason for such behavior. The other two methods use an approximation for tangent of the rotation angle before computing *c and s* which produces better orthogonality even at the initial values of *FL*. The error in data whitening originates from both the error in eigenvalues and the error in eigenvectors. So, the EJM is again performing poorly compared to the other two methods at the initial values of *FL*. Similarly, the imperfection in *CCC* is also determined by the errors in data whitening or EVD because the WVE stage of all the methods is the same. The EJM has the least *CCC* due to the progression of the errors from the EVD stage at the initial values of *FL*. The iteration count for EVD which is the number of iterations required for diagonalizing the matrix $\boldsymbol{S}$ by the Jacobi rotations is almost 4. The reason for such a low iteration count is the quadratic convergence of the EJM. The ALJM is very close to the EJM in terms of the iteration count for EVD because it uses a very good approximation compared to the AJM. The iteration count for the WVE stage or the number of iterations required for convergence of $\boldsymbol{W}$ also depends on the quality of the data whitening. This is the reason for the poor iteration count for the initial values of *FL*. From all the six graphs showing the average relative errors, *CCCs*, or the iteration counts, we can easily observe that the ALJM is performing better than or at least equal to the other two methods. Such performance of the ALJM ensures its superior separation quality at the hardware level before actually creating the hardware.

E. EFFECTS OF CHANGING DIMENSIONALITY

We can estimate from the systolic array architecture that the required amount of hardware resources varies with the square of dimensionality or channel count. This is why the real-time portable applications of ICA, as in [10], [13], can hardly accommodate eight channels. So, we decide to change dimensionality from 4 to 10 for studying the quantization errors in the case of low-cost FPGAs with $WL = 14 + 4 = 18$ to match the built-in DSP blocks. In Fig. 8, we plot the mean value of the three errors (defined in (64), (65), and (66)), *CCC* (defined in (67)), and the two iteration counts (defined in (69) and (71)) against the dimensionality (from 4 to 10 channels inclusive) for another set of 50 trials. In each trial, we use non-Gaussian random source signals $\boldsymbol{S}$ and mix them using a different randomly generated mixing matrix $\boldsymbol{A}$ of order $n \times n$. We set $WL = 18$ to deeply analyze the dimensionality variations in case of the low-cost FPGAs such as Spartan-6.

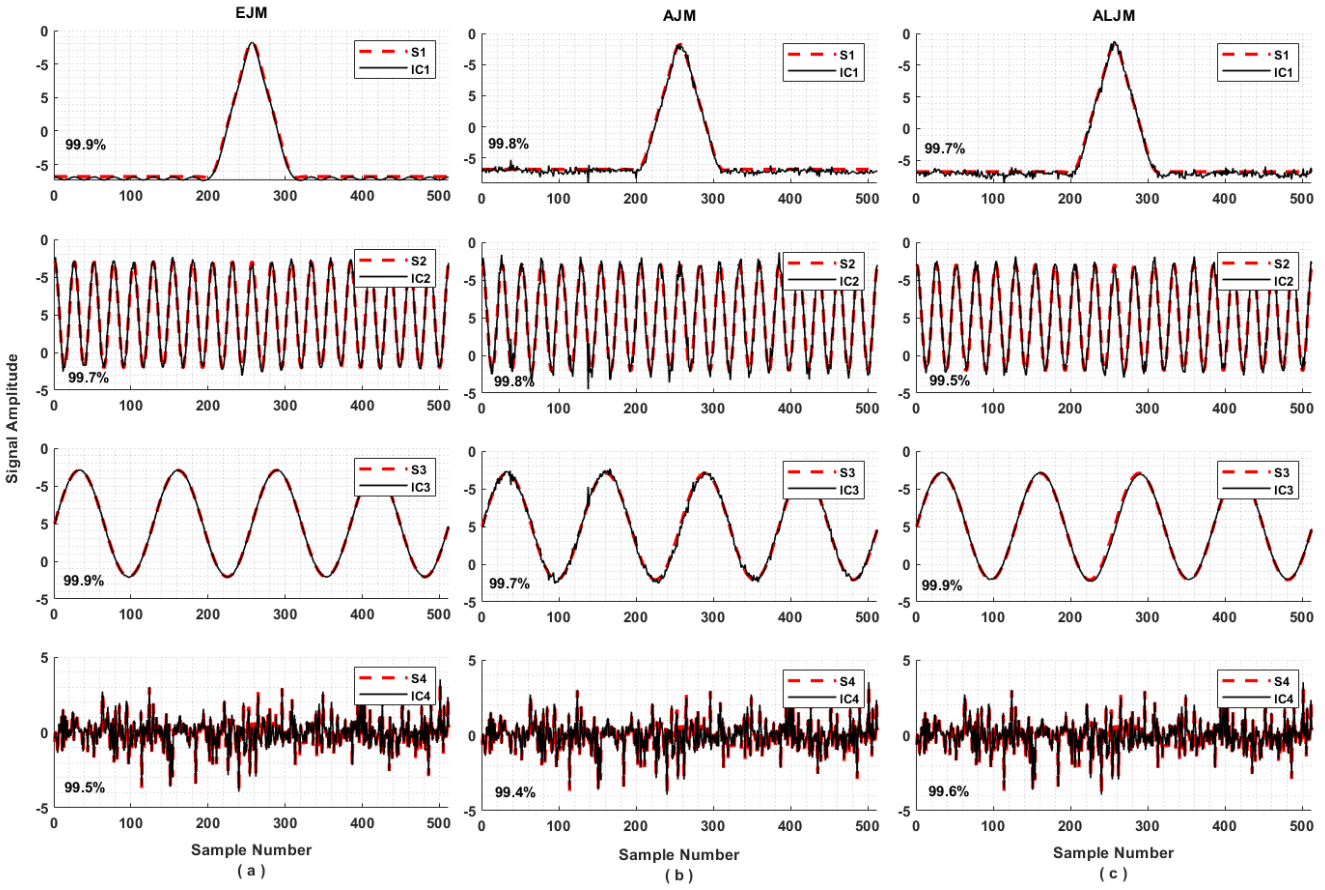

FIGURE 9. Comparing the Three Implementations of FastICA using the Four Synthetic Signals.

The reason for showing the performance results with the varying dimensionality is to see how the errors, caused by the quantization, grow with the increasing amount of computation. We know that the amount of computation increases with the square of the dimensionality. The error in eigenvalues, eigenvectors, and data whitening is growing with the increasing dimensionality. This is due to the growing amount of computation. *CCC* is also dropping with the increasing dimensionality and this indicates that for getting a *CCC* better than 95%, in the case of more than 10-channel, the required *FL* is greater than 14. The iteration count for EVD is almost independent of the dimensionality at least for the EJM and the ALJM, whereas the iteration count for WVE reflects the errors and imperfections from the data whitening and the EVD stage.

F. EXTRACTING THE DETERMINISTIC SIGNALS

For visually inspecting the performance of FastICA based on the three methods, we use four deterministic signals and generate the four mixed signals by randomly mixing them (by using a randomly generated mixing matrix $\boldsymbol{A}$ of order 4×4). These signals are selected to model some of the common contamination sources in EEG acquisition (sudden muscle movement, power line interference, and thermal noise) and an EEG signal itself (from gamma frequency band). In Fig. 9 (a), (b), and (c), we plot the sources with red dotted style and the extracted independent components (ICs) black solid style for the three algorithms. In this part of the evaluation, we set $FL = 14$ and $WL = 18$ to target the DSP blocks of the FPGAs for designing the SPA for FastICA. S1 to S4 denote the source signals, and IC1 to IC4 denote the extracted signals. We can see the performances of the algorithms under this setup are almost equal, as indicated by the cross-correlation label and visual inspection. We know that FastICA has the capability of extracting one Gaussian source [2]. We can conclude that the ALJM-based FastICA can also extract a Gaussian source and the other signals with a *CCC* of 99.5%.

We perform these trials to study and compare the numerical stability of the ALJM-based FastICA with limited *WL* in the fixed-point implementations. With the help of the 50 trials with random source signals and their randomly generated mixtures, we test the ALJM-based FastICA against any abnormal behavior resulting from some abnormal conditions due to the fixed-point computation. When very small eigenvalues appear ($\sigma_i < 0.002$ with $FL = 14$), then each of the algorithms stops to meet the convergence, so we replace such a small eigenvalue with zero to ensure the numerical

stability and convergence of the algorithms. All the observations made during this section are not possible without generating such software models of the methods. In the hardware domain comparing the methods to such dynamic conditions is tedious and time taking. So, the current work, in contrast to the existing approaches, introduces a new methodology for comparing, evaluating, and optimizing the hardware-oriented algorithms before realizing them on real hardware platforms.

VI. HARDWARE PERFORMANCE EVALUATION

After verifying the functionality, statistical performance, and numerical stability of the ALJM-based FastICA, we now analyze the hardware performance and efficiency of the method. The architecture for FastICA presented in Section IV can be implemented in different FPGAs and ASIC technology nodes. For proving the efficiency of the architecture, we implement it on Spartan-6 FPGA because this is a low-cost in-production FPGA series and meets the requirement of real-time portable system development. It is also available in small size packages. The DSP Blocks in Spartan-6 are called DSP48A1. The performance analysis is based on resource utilization, computation time, and power consumption. The functionality of the method is also verified through the post-layout simulation results and actual hardware implementation results. Performing FastICA gives a useful insight into the nature of latent sources. The actual sources can be extracted by FastICA only if they have non-Gaussian (sub/super-Gaussian) distributions and the mixed signals are linear combinations of the source signals. That is the reason for using the linear mixtures of non-Gaussian source signals throughout the theoretical evaluation of the FastICA. In the hardware evaluation, we also use eight random (non-Gaussian) source signals represented by $\boldsymbol{S}$ and mix them using a random mixing matrix $\boldsymbol{A}$ for generating the mixed signals $\boldsymbol{X}$.

A. BENCHMARK FOR THE HARDWARE FUNCTIONALITY

For setting a benchmark for the functionality, we use the floating-point double-precision MATLAB model of FastICA [30] (developed and distributed freely for research purposes by Helsinki University of Technology, Finland). This model is fed with the mixed multivariate signal $\boldsymbol{X}$ and it extracts the multivariate signal $\hat{\boldsymbol{S}}$. The model for FastICA is customized (by choosing tanh() as the non-linearity function and symmetric method for decorrelation), to match the hardware implementation, through the provided MATLAB GUI.

We use the Fixed-point Designer model of the ALJM-based FastICA for segregating quantization errors (errors caused by fixed-point 2's complement representation with $WL = 18$) and the hardware implementation errors (errors caused by imperfect translation and hardware circuitry). As explained in Section V, this model is also customized for implementation on Spartan-6 FPGA. The quantized versions of both $\boldsymbol{X}$ (denoted by $\boldsymbol{X_q}$) and $\hat{\boldsymbol{S}}$ (denoted by $\hat{\boldsymbol{S}}_q$) are used for comparison with the hardware results in the next subsection. $\boldsymbol{X_q}$ is produced when $\boldsymbol{X}$ is imported to Fixed-point Designer (fixed-point 2's complement representation with $WL = 14 + 4$) whereas $\hat{\boldsymbol{S}}_q$ represents the extracted sources from the model in Fixed-point Designer.

B. POST-LAYOUT SIMULATION AND IMPLEMENTATION

After hand-coding of the 8-channel the ALJM-based FastICA for Spartan-6 XC6SLX75-2FGG484C in Verilog HDL, we generate a testbench, post-place & route model, and a standard data format (SDF) file using Xilinx Integrated Software Environment (ISE 14.7 Webpack) running on Windows 10 64-bit. $\boldsymbol{X_q}$ is loaded into the BRAM modules by generating COE files. Similarly, the linearization coefficients for tanh() are loaded in the DRAM blocks. All these files are used for performing post-layout simulation using Isim tool, integrated with Xilinx ISE 14.7, and ModelSim PE 10.4 (Student Edition) tool from Mentor Graphics. The extracted signals from the Post-layout simulation $\hat{\boldsymbol{S}}_q$ are imported in MATLAB 2020b for comparison with the extracted sources by the model in Fixed-point Designer.

After the post-layout simulation, we use XPower Analyzer tool of ISE 14.7 for calculating the quiescent and dynamic power consumption of the design using the Switching Activity Interchange Format (SAIF) file from the post-layout simulation. We interface a Verilog HDL-based memory reading module with the model of the ALJM-based FastICA so that we can read the content of the BRAM modules and write the content to simulation output files for comparison.

After successfully performing the post-layout simulation, the bitstream file of the HDL model is downloaded to a custom board, containing XC6SLX75-2FGG484C and supporting a 10MHz external clock source, through a USB JTAG programming and debugging interface. The HDL model of FastICA is accompanied by the memory reading module and a ChipScope Definition and Connection (CDC) file for reading back the content of the Block RAM modules representing the extracted sources. Using appropriate trigger signals, the content of the BRAM modules is read using ChipScope Pro tool of ISE 14.7 and then the data files are imported to MATLAB for comparing them with the extracted sources from both the post-layout simulation and the Fixed-point Designer model. The comparison of the extracted sources from the three fixed-point platforms is discussed in the next subsection.

C. EXTRACTING THE RANDOM SOURCE SIGNALS

In Fig. 10 (a) we plot $\boldsymbol{X}$ (represented by dashed blue lines with an offset of 2 for visibility) and $\hat{\boldsymbol{S}}$ (represented by solid red lines without any offset) which are the inputs and the outputs of the double-precision MATLAB model of FastICA, respectively. Here, S is used to denote a source, and IC is used to represent the corresponding independent component or extracted source signal. Only 60 samples, out of 512, are shown for maintaining clarity in the graphs. CCC between the input and the extracted outputs is also shown below the legend. This is considered as the theoretical limit for the separation quality because there are no truncation or quantization errors in this model.

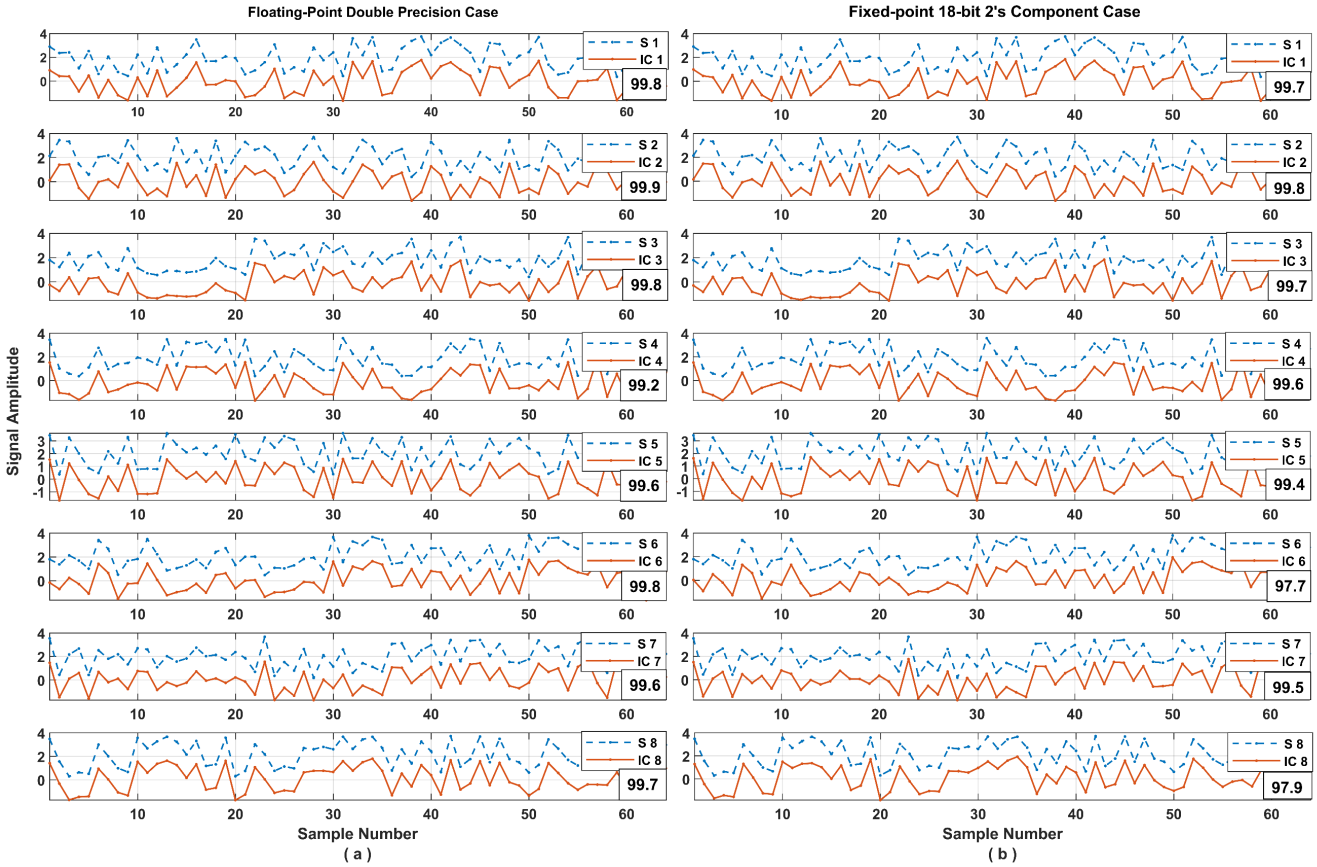

FIGURE 10. Comparing Floating-point (Software-based) and Fixed-point (Hardware-Based) Implementations of the ALJM-based FastICA.

The extracted signals $\hat{S}_q$ are generated by the Fixed-point Designer and they are treated as the benchmark for hardware implementation of the ALJM-based FastICA because this model only accounts for quantization errors and no hardware-related error appears at this stage. So, we set our target for generating these output signals, first from the post-layout simulation and then from the hardware implementation. That can ensure the functionality of the hardware design at the operating frequency recommended by ISE 14.7 during the post-layout simulation. We compare the results imported to MATLAB from the post-layout simulation and the hardware implementation with these benchmark results from Fix-point Designer. The three sets of results are identical point by point. This not only verifies the validity of the post-layout simulation followed the real implementation but also the capability of Fixed-point Designer to simulate the algorithm for its hardware behavior. To avoid redundancy, we only plot the results from the real implementation in Fig. 10 (b) where $\boldsymbol{X_q}$ (represented by dashed blue lines with an induced offset of 2) and $\hat{S}_q$ (represented by solid red lines, no offset). We label the results as fixed-point 18-bit 2's complement version of the ALJM based FastICA because it applies to all three platforms which produce the same results. As we can see from the figure that all the independent components extracted are, at least, 97% correlated with their theoretical versions (those extracted by the double-precision MATLAB model).

The degradation in *CCC*, as compared to Fig 10 (a), is due to the quantization resulting from the fixed-point arithmetic which is necessary for saving hardware resources. As we can see from [10], [11] that the performance of most of the AI or ML algorithms remains acceptable with a small degradation in *CCCs*. By compromising *CCC* to some extent, we can increase computation speed and save hardware resources as well as power consumption manyfold. That is clear from the figure that a 64-bit implementation (double-precision MATLAB model) of the algorithm is extracting the sources with a *CCC* of 99.7% whereas the 18-bit implementation (fixed-point FPGA) extracts the sources *CCC* up to 97.5%.

D. COMPARING DESIGN PARAMETERS

Table 1 lists the current and the past FastICA hardware implementations (published during the past decade with sufficient implementation detail) along with the design and performance parameters. We can see that most of the implementations are on ASIC platforms. Power consumption can be better optimized on ASICs than on FPGAs as explained in Section III. But ASIC implementations of AI or ML algorithms are not suitable for the rapid prototype development during the research phase. There is no recommended way

TABLE 1. Comparing design specifications of the hardware implementations of FastICA.

	(2008) [50]	(2011) [33]	(2014) [10]	(2015) [52]	(2015) [11]	(2015) [51]	(2016) [12]	(2018) [49]	(2019) [53]	(2019) [38]	(2020) [20]	This Work
EVD Method	QRD	EJM	EJM	EJM	AJM	EJM	GSO	QRD	GSO	EJM	AJM	ALJM
Channel Count	2	8	16	4	8	4	2-16	2	6	8	4	8
Window Size	3000	256	512	256	256	256	512	64000	1024	1024	512	512
Architecture Width (bits)	32	32	NP	NP	16	16	32	32	32	32	18	18
Design Speed (MHz)	50	100	20	172	11	100	100	100	240	100	100	10
Power Demand (mW)	NA	16.35	4.45	NP	0.0816	NP	16.35	4200	0.5703	65.0	321	152
Gates Required (x1000)	NA	272	NA	NA	69.2	NA	401	NA	NA	840	NA	NA
Computation Time (ms)	3	290	Variable	2.5	84.2	10	1850	68.1	NA	150	7.5	3.2
Process Node (nm)	90	90	130	40	90	45	90	28	90	90	28	45
Platform	FPGA	ASIC	ASIC	FPGA	ASIC	FPGA	ASIC	SoC	ASIC	ASIC	FPGA	FPGA
Correlation Coefficient (%)	> 99	> 99	> 95	NP	> 95	NP	> 96	NP	96	> 96	> 96	> 96

QRD: QR-Decomposition, **GSO**: Gram-Schmidt Orthogonalization, **NA**: Not Applicable, **NP**: Not Provided, **SoC**: System on Chip (Xilinx Zynq 7000)

for comparing the silicon area between ASIC and FPGA implementations of an algorithm. Higher design speed can be achieved with ASIC implementation because FPGA has programmable interconnects which brings ultimate flexibility but reduce design speed. Some other methods such as QR-Decomposition (QRD) and Gram-Schmidt Orthogonalization (GSO) are also reported for computing EVD. These methods use square root and division, and they are frequently used on software platforms because of their hardware complexity. Using a divider and a square root circuit to perform all the operations of FastICA can save hardware resources but speed performance cannot be achieved in such cases as in [12]. In [49] a System on Chip (SoC, programable logic along with an Arm processor) is used for implementing these operations. In [50] no hardware utilization or power consumption is reported so we cannot have a fair comparison.

We can see that our work is the fastest 8-channel FastICA implementation with a *CCC* greater than 96. It meets the speed and accuracy requirements for the most real-time portable applications while leaving a sufficient execution time for any subsequent post-processing algorithm. For developing a prototype for system-level evaluation of a real-time portable system that incorporates FastICA (for example, epileptic seizure detection using EEG or background noise removal in audio systems), the proposed method is the most suitable approach.

In our implementation, we transfer computational load from reconfigurable logic to the dedicated DSP blocks and BRAM modules. In this way, we take advantage of ASIC design while maintaining the design flexibility offered by FPGA. That is only possible by using the ALJM-based FastICA. So, we can say this is the reason that the design efficiency of the current work is not only better than the other FPGA implementations but also competing with the ASIC implementations in all three aspects of a digital design (area, speed, and power).

E. COMPARING COMMERCIAL ASPECTS

After comparing our work in terms of speed with the previous works, we now compare our design in terms of hardware resources and power consumption with the most relevant designs. We know that FPGA resources consist of BRAM, DSP blocks, lookup tables, and flipflops. Table 2 lists 4-channel FPGA-based FastICA implementations. We scale the ALJM-based FastICA to 4-channel and map it to Spartan-6 device XC6SLX9-2CPG196C to perform a fair comparison. The FPGA implementations are also compared in terms of the commercial aspects such as power, size, and cost.

In [51] a non-scalable implementation based on the EJM is reported. Although non-scalable implementations can be better optimized compared to scalable implementations but adapting them to changing dimensionality needs a redesign. In [52] a high-speed scalable implementation is reported for a Virtex-6 device. These devices are high performance but have large packaging sizes and high cost. So, they cannot meet the requirements of a low-cost real-time portable system development. Another implementation for the AJM based FastICA [20] is reported using Xilinx 7-series device Kintex-7. This is a middle-range device with respect to cost and performance. This is a scalable implementation and uses the AJM along with the hardware-based multipliers. This is one of the most optimized implementations, from the previous works, for developing a real-time portable system.

TABLE 2. Comparing the FPGA-based FastICA implementations.

	(2015) [52]	(2015) [51]	(2020) [20]	This Work
EVD Method	EJM	EJM	AJM	ALJM
Scalability	Yes	No	Yes	Yes
Device Series	Virtex-6	Spartan-6	Kintex-7	Spartan-6
Target Device Price (USD)	715	25	160	18
Target Device Size (mm^2)	23×23	8×8	23×23	8×8
Computation Time (ms)	2.5	10	7.5	1.9
Power Demand (mW)	NP	NP	321	91
Slice Utilization LUT Flip-Flop	NP 132811 160629	14760 NP NP	NP 12821 13595	1056 4180 114
Block RAM (KB)	45	60	18	9
DSP48 Slices	440	29	25	16

TABLE 3. Comparing the EVD methods on the SPA for FastICA.

	EJM	AJM	ALJM
Additional Hardware for EVD Stage	72 Registers 10 Shifters 12 Adders	12 Encoders 20 Shifters 20 Adders	Nothing
Convergence Rate	Quadratic	Almost Linear	Almost Quadratic
Clock Cycles for $\cos\theta i$ & $\sin\theta i$	36	5	6
Maximum Sweeps for Convergence	4	9	5
Total Clock Cycles for EVD	1568	1575	840

The results of the current work show that the ALJM-based FastICA saves 70% lookup tables, 40% DSP blocks, and 50% BRAM modules and still achieves a speed of three times compared to the state-of-the-art approach in [20]. Moreover, 70% power saving is also achieved by using the power-optimized hardware like the DSP blocks (DSP48A1) and the BRAM modules. Based on these results we can say that the current work is the most suitable implementation for the targeted systems.

F. HARDWARE SAVING BY THE SPA

As stated before, one of the contributions of the current work is introducing the scalable SPA for FastICA implementation. The SPA can be used for FastICA using any of the three methods for EVD but the ALJM is the most optimized method because it uses no additional component and the components of the WVE stage are reused for computing EVD. In Table 3, we list the number of additional components required for the EJM and the AJM, in the case 8-channel FastICA is implemented on the SPA. The 72 registers are 18-bit wide and are used to implement CORDIC blocks for the EJM. The shifters are variable signed shifters. They are combinational circuits and the amount and direction of the shifts are determined by the signed integer at the selection input of the circuit. The encoders represent the signed priority encoders that find the binary exponent of a number in 2's complement. The EJM requires the maximum number of clock cycles for computing $\cos\theta i$ & $\sin\theta i$ because of the slow convergence of CORDIC blocks. The required number of clock cycles can be reduced for the AJM by using more memory, shifters, and adders. But such hardware components increase quiescent power because they cannot be reused in the WVE stage. So, we can say that the ALJM-based FastICA is again the most optimized implementation.

VII. CONCLUSION AND FUTURE WORK

We know that ICA is the most widely used technique for artifact removal and feature extraction from a multivariate signal and FPGAs are the most popular hardware platform for implementing AI or ML algorithms. For rapidly prototyping real-time portable systems capable of performing ICA followed by some ML algorithm, we have proposed a new method, named the ALJM, for computing EVD during FastICA (the most powerful and frequently used variant of ICA). Using Fixed-point Designer, we have verified the superior theoretical performance of the ALJM-based FastICA. Quantization errors, iteration counts, and *CCC* have been used as performance metrics for the comparison. Based on the ALJM, we have proposed the SPA architecture for performing FastICA on the hardware platforms. We have implemented the SPA on Spartan-6 FPGA and compared its performance with the previous works. The hardware performance comparison is based on the parameters such as computation time (latency), channel count, power consumption, and design cost/size. This comparison concludes that the ALJM-based FastICA implemented, even on low-cost FPGA devices, performs better than the existing implementations with the same channel count. So, the proposed SPA is one of the best choices for prototyping and commercializing FastICA based AI systems.

In this work, we have hand-coded the design to generate its HDL model for low-cost FPGAs. The process of hand-coding requires time, effort, and relevant experience and these are rarely available to the researchers developing algorithms in MATLAB. So, in future work, we will translate the Fixed-point Designer model to the HDL model using different high-level synthesis (HLS) tools and compare their translation performance in terms of hardware design aspects. This methodology will help the researchers in the rapid prototyping of their systems. Moreover, we have compared the ALJM with the existing methods only in the context of FastICA implementation. In future work, we will also extend this comparison to solely EVD/SVD computation efficiency.

REFERENCES

[1] P. Comon, "Independent component analysis, a new concept?" *Signal Process.*, vol. 36, no. 3, pp. 287–314, Apr. 1994.

[2] A. Hyvarinen, "Fast and robust fixed-point algorithms for independent component analysis," *IEEE Trans. Neural Netw.*, vol. 10, no. 3, pp. 626–634, May 1999.

[3] A. Hyvärinen, "The fixed-point algorithm and maximum likelihood estimation for independent component analysis," *Neural Process. Lett.*, vol. 10, no. 1, pp. 1–5, 1999.
[4] G. Sahonero-Alvarez and H. Calderón, "A comparison of SOBI, FastICA, JADE and Infomax algorithms," in *Proc. 8th Int. Multi-Conf. Complex., Informat. Cybern.*, Mar. 2017, pp. 17–22.
[5] G. R. Naik and D. K. Kumar, "An overview of independent component analysis and its applications," *Informatica*, vol. 35, no. 1, pp. 63–81, 2011.
[6] X. Jiang, G.-B. Bian, and Z. Tian, "Removal of artifacts from EEG signals: A review," *Sensors*, vol. 19, no. 5, p. 987, Feb. 2019.
[7] C.-T. Lin, L.-W. Ko, M.-H. Chang, J.-R. Duann, J.-Y. Chen, T.-P. Su, and T.-P. Jung, "Review of wireless and wearable electroencephalogram systems and brain-computer interfaces—A mini-review," *Gerontology*, vol. 56, no. 1, pp. 112–119, 2010.
[8] X. Wu, B. Zhou, Z. Lv, and C. Zhang, "To explore the potentials of independent component analysis in brain-computer interface of motor imagery," *IEEE J. Biomed. Health Informat.*, vol. 24, no. 3, pp. 775–787, Mar. 2020.
[9] M. A. Talib, S. Majzoub, Q. Nasir, and D. Jamal, "A systematic literature review on hardware implementation of artificial intelligence algorithms," *J. Supercomput.*, vol. 77, pp. 1–42, May 2020.
[10] T. Roh, K. Song, H. Cho, D. Shin, and H.-J. Yoo, "A wearable neurofeedback system with EEG-based mental status monitoring and transcranial electrical stimulation," *IEEE Trans. Biomed. Circuits Syst.*, vol. 8, no. 6, pp. 755–764, Dec. 2014.
[11] C.-H. Yang, Y.-H. Shih, and H. Chiueh, "An 81.6 μW FastICA processor for epileptic seizure detection," (in English), *IEEE Trans. Biomed. Circuits Syst.*, vol. 9, no. 1, pp. 60–71, Feb. 2015, doi: 10.1109/Tbcas.2014.2318592.
[12] L.-D. Van, P.-Y. Huang, and T.-C. Lu, "Cost-effective and variable-channel FastICA hardware architecture and implementation for EEG signal processing," *J. Signal Process. Syst.*, vol. 82, no. 1, pp. 91–113, Jan. 2016.
[13] T. Roh, S. Hong, H. Cho, and H.-J. Yoo, "A 259.6 μW HRV-EEG processor with nonlinear chaotic analysis during mental tasks," *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 1, pp. 209–218, Feb. 2016, doi: 10.1109/Tbcas.2014.2369576.
[14] M. Stanacevic, S. Li, and G. Cauwenberghs, "Micropower mixed-signal VLSI independent component analysis for gradient flow acoustic source separation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 7, pp. 972–981, Jul. 2016.
[15] A. Jafarifarmand, M.-A. Badamchizadeh, S. Khanmohammadi, M. A. Nazari, and B. M. Tazehkand, "Real-time ocular artifacts removal of EEG data using a hybrid ICA-ANC approach," *Biomed. Signal Process. Control*, vol. 31, pp. 199–210, Jan. 2017.
[16] A. Acharyya, P. N. Jadhav, V. Bono, K. Maharatna, and G. R. Naik, "Low-complexity hardware design methodology for reliable and automated removal of ocular and muscular artifact from EEG," *Comput. Methods Programs Biomed.*, vol. 158, pp. 123–133, May 2018.
[17] F. Carrizosa-Corral, A. Váazquez-Cervantes, J.-R. Montes, T. Hernández-díaz, L. Barriga-Rodríguez, J. A. Soto-Cajiga, and H. Jiménez-Hernández, "ICA-based background subtraction method for an FPGA-SoC," *Electron. Imag.*, vol. 2017, no. 13, pp. 36–41, Jan. 2017.
[18] P. Jaraut, M. Rawat, and P. Roblin, "Digital predistortion technique for low resource consumption using carrier aggregated 4G/5G signals," *IET Microw., Antennas Propag.*, vol. 13, no. 2, pp. 197–207, Feb. 2019.
[19] I. Rejer and P. Górski, "MAICA: An ICA-based method for source separation in a low-channel EEG recording," *J. Neural Eng.*, vol. 16, no. 5, Sep. 2019, Art. no. 056025.
[20] Z. Li, L. Feng, J. Zhang, and X. Li, "VLSI design of a fast one-stage independent component extracting system based on ICA-R algorithm," *J. Circuits, Syst. Comput.*, vol. 30, no. 3, Mar. 2021, Art. no. 2150044, doi: 10.1142/s0218126621500444.
[21] G. E. Forsythe and P. Henrici, "The cyclic Jacobi method for computing the principal values of a complex matrix," *Trans. Amer. Math. Soc*, vol. 94, no. 1, pp. 1–23, 1958.
[22] R. P. Brent and F. T. Luk, "The solution of singular-value and eigenvalue problems on systolic arrays," in *Proc. Math. Program. Numer. Anal. Workshop Centre Math. Appl.*, 1984, pp. 38–64.
[23] Z. Shi, Q. He, and Y. Liu, "Accelerating parallel Jacobi method for matrix eigenvalue computation in DOA estimation algorithm," *IEEE Trans. Veh. Technol.*, vol. 69, no. 6, pp. 6275–6285, Jun. 2020.
[24] A. Acharyya, K. Maharatna, B. M. Al-Hashimi, and J. Reeve, "Coordinate rotation based low complexity N-D FastICA algorithm and architecture," *IEEE Trans. Signal Process.*, vol. 59, no. 8, pp. 3997–4011, Aug. 2011.
[25] C.-C. Sun, J. Götze, and G. E. Jan, "Parallel jacobi EVD methods on integrated circuits," *VLSI Design*, vol. 2014, pp. 1–9, Jul. 2014.
[26] J. Volder, "The CORDIC computing technique," in *Proc. Western Joint Comput. Conf.*, Mar. 1959, pp. 257–261.
[27] J. Gotze, S. Paul, and M. Sauer, "An efficient Jacobi-like algorithm for parallel eigenvalue computation," *IEEE Trans. Comput.*, vol. 42, no. 9, pp. 1058–1065, Sep. 1993.
[28] Y. Liu, C.-S. Bouganis, and P. Y. K. Cheung, "Hardware architectures for eigenvalue computation of real symmetric matrices," *IET Comput. Digit. Techn.*, vol. 3, no. 1, pp. 72–84, Jan. 2009.
[29] The MathWorks. (2020). *Fixed-Point Designer Getting Strated Guide R2020b*. Accessed: Oct. 7, 2020. [Online]. Available: https://www.mathworks.com/help/pdf_doc/fixedpoint/fixedpoint_gs.pdf
[30] A. Hyvärinen and E. Oja, "Independent component analysis: Algorithms and applications," *Neural Netw.*, vol. 13, nos. 4–5, pp. 411–430, Jun. 2000.
[31] L. N. Trefethen and D. Bau III, *Numerical Linear Algebra*. Philadelphia, PA, USA: SIAM, 1997.
[32] C.-C. Sun and J. Götze, "A VLSI design concept for parallel iterative algorithms," *Adv. Radio Sci.*, vol. 7, pp. 95–100, May 2009.
[33] L.-D. Van, D.-Y. Wu, and C.-S. Chen, "Energy-efficient FastICA implementation for biomedical signal separation," *IEEE Trans. Neural Netw.*, vol. 22, no. 11, pp. 1809–1822, Nov. 2011, doi: 10.1109/Tnn.2011.2166979.
[34] O. Mencer, L. Semeria, M. Morf, and J.-M. Delosme, "Application of reconfigurable CORDIC architectures," *J. VLSI Signal Process. Syst. Signal, Image Video Technol.*, vol. 24, nos. 2–3, pp. 211–221, Mar. 2000.
[35] S. M. R. Shahshahani and H. R. Mahdiani, "A high-performance scalable shared-memory SVD processor architecture based on Jacobi algorithm and Batcher's sorting network," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 6, pp. 1912–1924, Jun. 2020.
[36] T.-B. Juang, S.-F. Hsiao, and M.-Y. Tsai, "Para-CORDIC: Parallel CORDIC rotation algorithm," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 8, pp. 1515–1524, Aug. 2004.
[37] I. Bravo, M. Mazo, J. L. Lázaro, P. Jiménez, A. Gardel, and M. Marrón, "Novel HW architecture based on FPGAs oriented to solve the eigen problem," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 12, pp. 1722–1725, Dec. 2008.
[38] L.-D. Van, T.-C. Lu, T.-P. Jung, and J.-F. Wang, "Hardware-oriented memory-limited online FastICA algorithm and hardware architecture for signal separation," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 1438–1442.
[39] Y. Wang, J.-J. Lee, Y. Ding, and P. Li, "A scalable FPGA engine for parallel acceleration of singular value decomposition," in *Proc. 21st Int. Symp. Qual. Electron. Design (ISQED)*, Mar. 2020, pp. 370–376.
[40] Xilinx. (2014). *Spartan-6 FPGA DSP48A1 Slice Spartan-6 Family User Guide UG389*. Accessed: Oct. 5, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug389.pdf
[41] D. Markovic, B. Nikolic, and R. W. Brodersen, "Power and area minimization for multidimensional signal processing," *IEEE J. Solid-State Circuits*, vol. 42, no. 4, pp. 922–934, Apr. 2007.
[42] L. V. Moroz, V. V. Samotyy, and O. Y. Horyachyy, "Modified fast inverse square root and square root approximation algorithms: The method of switching magic constants," *Computation*, vol. 9, no. 2, p. 21, Feb. 2021.
[43] Intel Corporation. (2018). *Variable Precision DSP Block*. Intel Arria 10 Device Overview. Accessed: Oct. 5, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-10/a10_overview.pdf
[44] Xilinx. (2018). *7 Series DSP48E1 Slice Xilinx User Guide UG479*. Accessed: Oct. 5, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf
[45] Xilinx. *7 Series FPGAs Configurable Logic Block User Guide UG474 2016*. Accessed: Oct. 5, 2020. [Online]. Available: https://www.xilinx.com/supportdocumentation/user_guides/ug474_7Series_CLB.pdf
[46] *Fixed-Point Designer 2020b*, The MathWorks, 3 Apple Hill Drive, Natick, MA, USA, 01760-2098, 2020.
[47] D. P. Acharya, G. Panda, and Y. V. S. Lakshmi, "Effects of finite register length on fast ICA, bacterial foraging optimization based ICA and constrained genetic algorithm based ICA algorithm," *Digit. Signal Process.*, vol. 20, no. 3, pp. 964–975, May 2010.
[48] D. Patil, N. Das, and A. Routray, "Implementation of fast-ICA: A performance based comparison between floating point and fixed point DSP platform," *Meas. Sci. Rev.*, vol. 11, no. 4, pp. 118–124, Jan. 2011.

[49] F. Carrizosa-Corral, A. Vázquez-Cervantes, J.-R. Montes, T. Hernández-Díaz, J. C. S. Vargas, L. Barriga-Rodríguez, J. A. Soto-Cajiga, and H. Jiménez-Hernández, "FPGA-SoC implementation of an ICA-based background subtraction method," *Int. J. Circuit Theory Appl.*, vol. 46, no. 9, pp. 1703–1722, 2018.
[50] K.-K. Shyu, M.-H. Lee, Y.-T. Wu, and P.-L. Lee, "Implementation of pipelined FastICA on FPGA for real-time blind source separation," *IEEE Trans. Neural Netw.*, vol. 19, no. 6, pp. 958–970, Mar. 2008.
[51] R. Tang, H. Wu, Y. Pak, Y. Liu, Q. Wang, and Y. Zhao, "Optimized FPGA implementation of ICA based on negentropy maximization," in *Proc. 5th Int. Conf. Instrum. Meas., Comput., Commun. Control (IMCCC)*, Sep. 2015, pp. 551–555.
[52] D. Zhao, J. Jiang, C. Wang, B. Lu, and Y. Zhu, "FPGA Implementation of FastICA Algorithm for On-line EEG Signal Separation," in *Computer Engineering and Technology*. Berlin, Germany: Springer, 2015, pp. 59–68.
[53] S. Bhardwaj, S. Raghuraman, and A. Acharyya, "Simplex FastICA: An accelerated and low complex architecture design methodology for *n*D FastICA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1124–1137, Dec. 2019.

MUHAMMAD SAJJAD received the B.Sc. (Eng.) degree in electrical engineering from the University of Engineering and Technology Taxila, Pakistan, in 2008. He is currently pursuing the M.Sc. (by Research) degree from Universiti Teknologi PETRONAS (UTP), Seri Iskandar, Malaysia.

He is also attached to the Centre for Intelligent Signal and Imaging Research (CISIR), UTP. His area of research interest includes efficient hardware architecture for brain signal processing and electroencephalography (EEG). He was a Senior Digital Design Engineer at the Center for Advanced Research in Engineering, Pakistan, from 2008 to 2018.

Mr. Sajjad received the UTP Graduate Assistantship Scheme for full-time research students in 2019.

MOHD ZUKI YUSOFF (Member, IEEE) received the B.Sc. degree in electrical engineering from Syracuse University, in 1988, the M.Sc. degree in communications, networks, and software from the University of Surrey, in 2001, and the Ph.D. degree in electrical and electronic engineering from Universiti Teknologi PETRONAS (UTP), Malaysia, in 2010.

He has accumulated over 29 years of experience working with various industries and academic/training institutions, such as Celcom Academy, Politeknik Sultan Abdul Halim Mu'adzam Shah (POLIMAS), the Malaysian Institute of Microelectronic Systems (MIMOS), and Singatronics (M) Sdn., Bhd. He is currently an Associate Professor with UTP, where he is also the Director of the Centre for Intelligent Signal and Imaging Research (CISIR) and a member of the Institute of Health and Analytics (IHA). His research interests include brain–computer interface, transport safety, and telecommunications. He has international publications and holds some patents. He is also a member of the following learned societies and professional body: Tau Beta Pi—the National Engineering Honorary Society, Eta Kappa Nu—the Electrical and Computer Engineering Honorary Society, and the Board of Engineers Malaysia (as a Graduate Engineer). He is a certified Curriculum Designer and Developer awarded by the Sepang Institute of Technology (SIT) and the Douglas Mawson Institute of TAFE, in March 1999.

NORASHIKIN YAHYA (Member, IEEE) received the B.Eng. degree (Hons.) in electronic engineering from The University of Sheffield, U.K., in 2001, the M.Sc. degree in electrical engineering from Lehigh University, USA, in 2004, and the Ph.D. degree in electrical engineering from Universiti Teknologi PETRONAS (UTP), Seri Iskandar, Malaysia, in 2015.

She is currently a Senior Lecturer with UTP, where she is also a member of the Centre for Intelligent Signal and Imaging Research (CISIR), one of the Malaysia National Higher Institution Centre of Excellence (HICoE) status focusing on neuro signal and image analysis as its research niche area. Her current active research works are on image segmentation and pattern recognition techniques involving biomedical signals and images, seismic signals, and acoustic emission signals using deep learning architecture.

ALI SHAHBAZ HAIDER (Member, IEEE) received the B.Sc. degree in electrical engineering from the University of Engineering and Technology (UET) Taxila, Pakistan, in 2008, and the M.Sc. degree in systems engineering from the Pakistan Institute of Engineering and Applied Sciences (PIEAS), Islamabad, Pakistan, in 2010.

He has been a Consultant Engineer of Dehlsen Associates LLC, Santa Barbra, CA, USA. He has been involved in the collaborative research with the U.S. Department of Energy, Sandia National Labs, USA, and Pacific Marine Energy Center (PMEC), USA. He has been a Ph.D.-Fulbright-USA Scholar with the Department of Electrical and Computer Engineering, Oregon State University, USA, since 2017. His research interests include digital control, power electronics, and electric drives.