PARALLEL ARCHITECTURE FOR INDEPENDENT COMPONENT ANALYSIS ALGORITHM (ICA)

Independent component analysis (ICA) is a recently proposed method as a solution to the blind source separation problem. The objective is to recover the unobserved source signals from the observed mixtures without the knowledge of the mixing coefficients. It has the potential for a wide range of applications in industrial, medical, security, and military fields because it reduces the complex problem of dealing with high-dimensional statistical descriptions to products of one-dimensional density functions. For some applications, off-line ICA analysis on a workstation could be adequate, but for a vast majority it is desirable to have a VLSI chip for real-time analysis. In this chapter we propose some preliminary ideas on a parallel architecture for the real-time implementation of ICA algorithm on the reconfigurable J-platform. Admittedly it just begins to address the problem of VLSI implementation and explores some first steps toward actual high-speed implementation. The feasibility of mapping ICA to the J-platform is demonstrated, but much work remains to be performed as delineated in the 'discussion' section at the end of the chapter.

1 Motivation for ICA

Imagine that in a room, two people are speaking simultaneously and that there are two microphones which produce time signals denoted by $x_1(t)$ and $x_2(t)$. Each of these received signals is a weighted sum of the speech signals emitted by the two speakers denoted by $s_1(t)$ and $s_2(t)$. Then we can express the received signals in terms of the original signals as

$$x_1(t) = a_{11}\, s_1(t) + a_{12}\, s_2(t) \qquad (1)$$

$$x_2(t) = a_{21}\ s_1(t) + a_{22}\ s_2(t)$$

where a_{11}, a_{12}, a_{21}, and a_{22} are certain parameters that depend on the microphone characteristics and their distances from the speakers. Clearly, it would be very useful to recover the original speech signals from the received signals.

More generally, if there are *n* different signals and *n* received mixed signals, then the relationship can be expressed as

$$\begin{aligned} x_1(t) &= a_{11}\ s_1(t) + a_{12}\ s_2(t) + \ .\ .\ . + a_{1n}\ s_n(t) \\ x_2(t) &= a_{21}\ s_1(t) + a_{22}\ s_2(t) + \ .\ .\ . + a_{2n}\ s_n(t) \\ . \quad & \quad . \qquad\quad . \qquad . \quad . \quad . \\ x_n(t) &= a_{n1}\ s_1(t) + a_{n2}\ s_2(t) + \ .\ .\ . + a_{nn}\ s_n(t) \end{aligned} \tag{2}$$

or in matrix-vector notation $\underline{x}(t) = \boldsymbol{A}\ \underline{s}(t)$. Here, for example s_1 and s_2 could be speech signals, s_3 could be the sound produced by a motor vehicle, etc. In a biomedical environment s_1(t), … could represent a set of EEG signals, ECG signals, etc.

The recently developed technique called ICA, can be used to estimate ***A*** or its inverse $\boldsymbol{W} = \boldsymbol{A}^{-1}$ based on the information of their statistical independence, which then allows blind separation of the original signals from their mixtures. The technique is applicable not only to time signals but also to images. As a specific example consider the three images s_1, s_2 and s_3 shown in the Figure 1 (a), (b) and (c). Their histograms are shown in (d), (e) and (f). Suppose now that the observed images are the ones shown in Figure 1 (g), (h) and (i). We now pose the question whether and how we can recover the original images blindly (without the knowledge of the mixing information). The answer is a ‘yes’. Indeed, the images estimated by the application of the fast version of ICA, called Fast ICA, are shown in Figure 1 (j) (k) and (l).

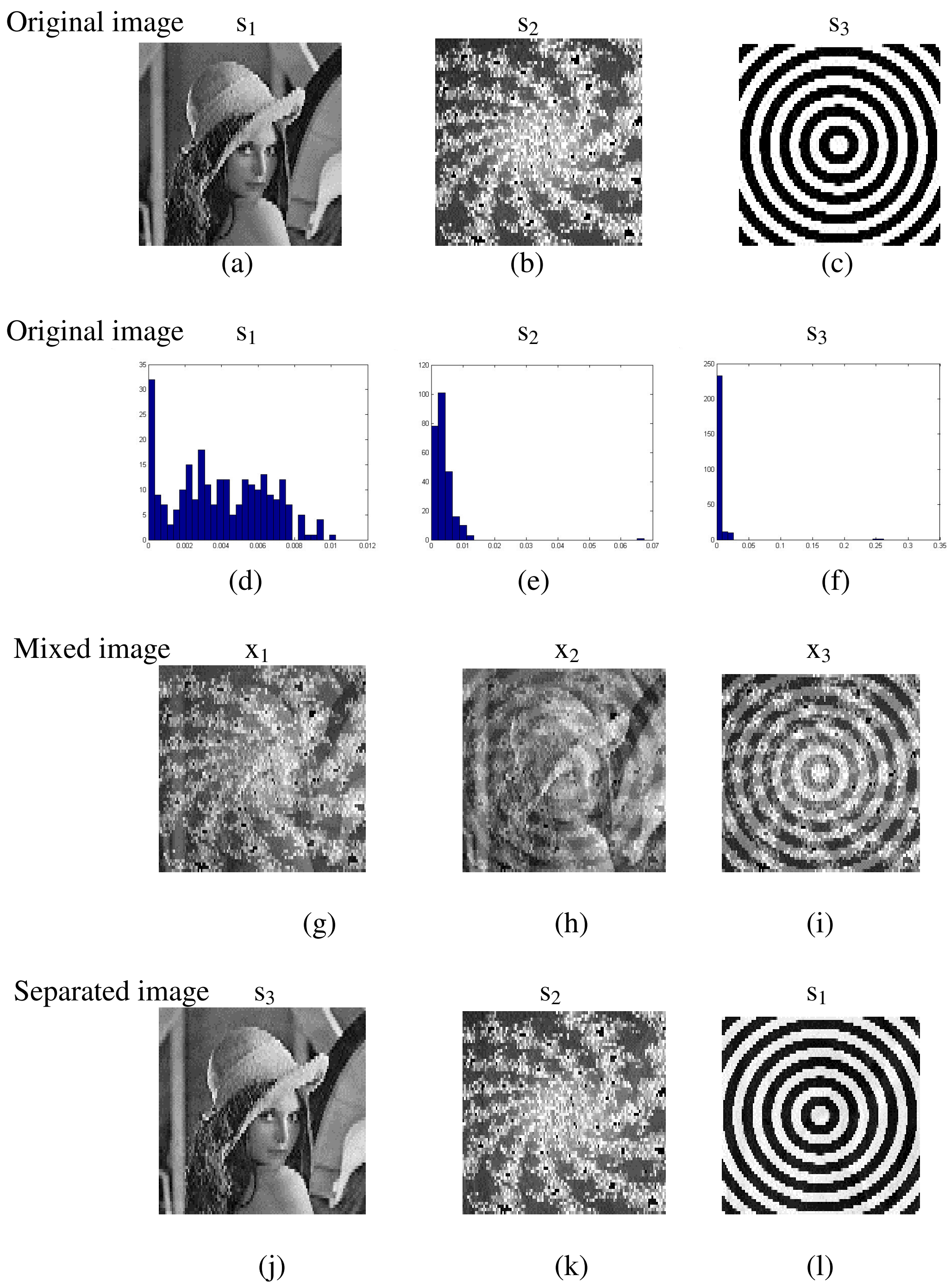
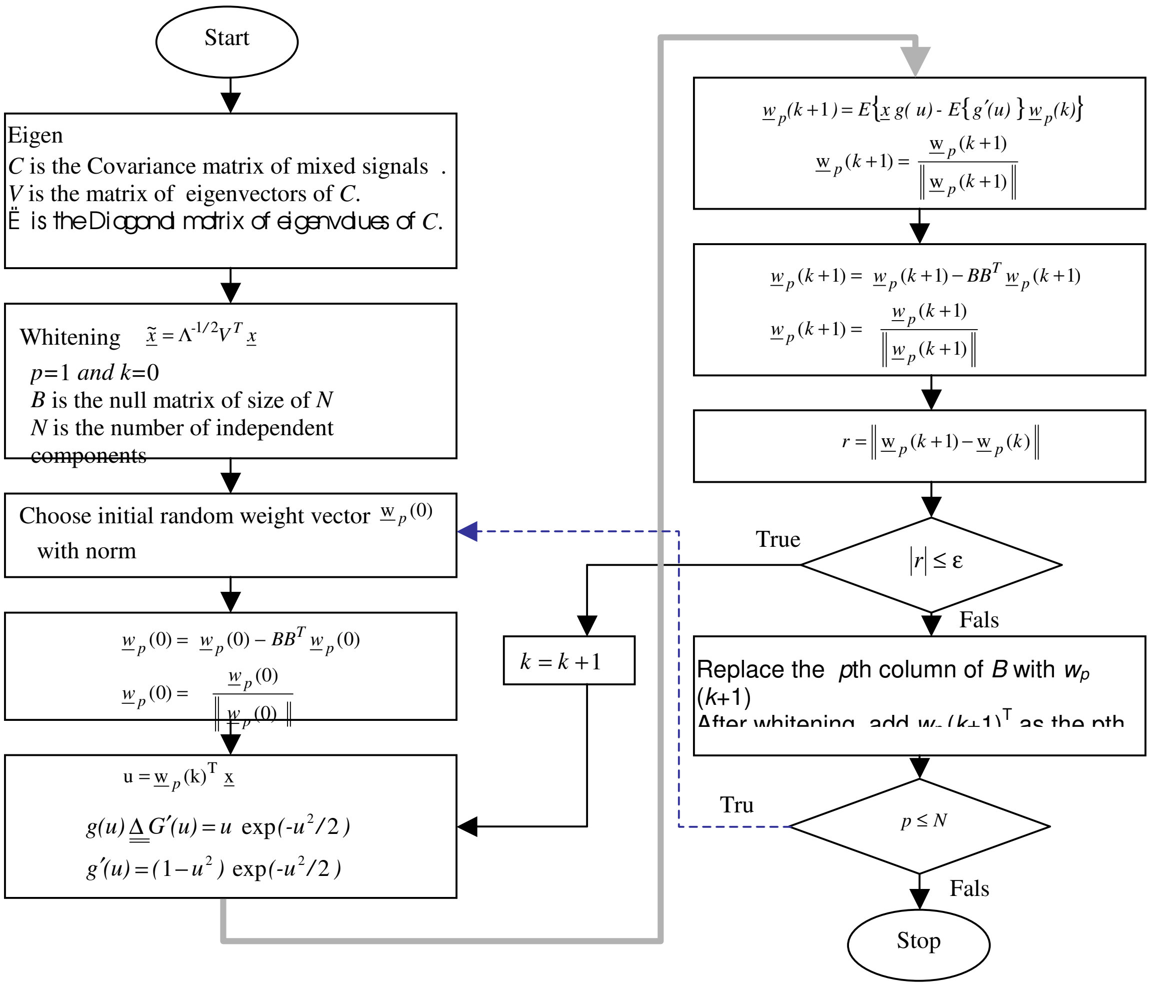


Figure 1. Separation of the original images using ICA algorithm

This algorithm has a wide range of applications in industrial and medical fields. For some specific application, ICA analysis on a workstation is adequate, but for a vast majority it is desirable to have a VLSI chip that can perform Independent component analysis (ICA) in real-time. In this chapter we propose a parallel architecture for the real-time implementation of ICA algorithm on the reconfigurable J-platform, developed in our laboratory.

2 Fast ICA Algorithm

The flowchart indicating the various steps involved in the Fast ICA algorithm is shown in the Figure 2



Start
Eigen
C is the Covariance matrix of mixed signals .
V is the matrix of eigenvectors of C.
Ë is theDiagonal matrix of eigenvalues of C.
Whitening $\underline{\tilde{x}} = \Lambda^{-1/2} V^T \underline{x}$
p=1 *and* k=0
B is the null matrix of size of N
N is the number of independent components
Choose initial random weight vector $\underline{w}_p(0)$ with norm
$\underline{w}_p(0) = \underline{w}_p(0) - BB^T \underline{w}_p(0)$
$\underline{w}_p(0) = \frac{\underline{w}_p(0)}{\| \underline{w}_p(0) \|}$
$u = \underline{w}_p(k)^T \underline{x}$
$g(u) \underline{\underline{\Delta}} G'(u) = u \exp(-u^2/2)$
$g'(u) = (1-u^2) \exp(-u^2/2)$
$\underline{w}_p(k+1) = E\{\underline{x}\, g(u) - E\{g'(u)\} \underline{w}_p(k)\}$
$\underline{w}_p(k+1) = \frac{\underline{w}_p(k+1)}{\| \underline{w}_p(k+1) \|}$
$\underline{w}_p(k+1) = \underline{w}_p(k+1) - BB^T \underline{w}_p(k+1)$
$\underline{w}_p(k+1) = \frac{\underline{w}_p(k+1)}{\| \underline{w}_p(k+1) \|}$
$r = \| \underline{w}_p(k+1) - \underline{w}_p(k) \|$
True
$|r| \le \varepsilon$
Fals
$k = k + 1$
Replace the pth column of B with w_p (k+1)
After whitening, add $w_p(k+1)^T$ as the pth
Tru
$p \le N$
Fals
Stop

Figure 2 Flowchart for ICA algorithm

A very high-level summary of the detailed flowchart is given in Figure 3. The details of the various blocks are discussed in the following sections.

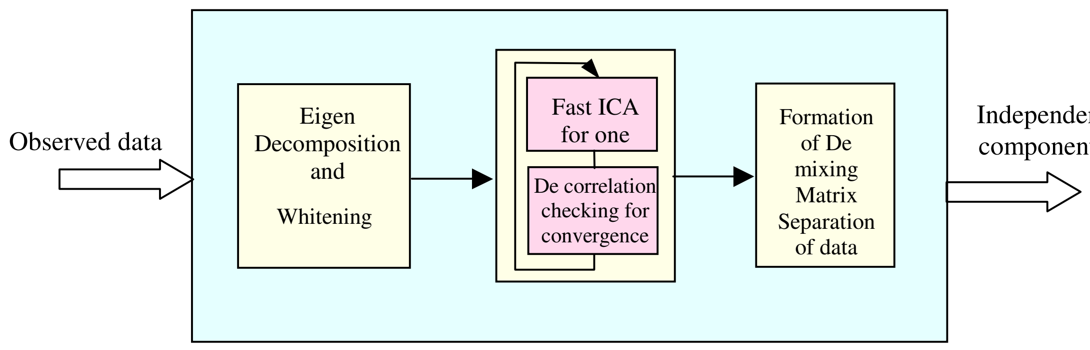

Observed data
Eigen Decomposition and Whitening
Fast ICA for one
De correlation checking for convergence
Formation of De mixing Matrix Separation of data
Independent components

Figure 3 Simplified diagram for ICA algorithm

3 Preprocessing for Fast ICA

Before applying the ICA algorithm on the given data, it is useful to perform preprocessing. Some preprocessing techniques that can make the problem of ICA estimation simpler and better conditioned are centering, whitening and band pass filtering. In this chapter we discuss only whitening.

3.1 Whitening

Whitening reduces number of parameters to be estimated. Whitened data $\underline{\tilde{x}}$ has its components uncorrelated and their variances equal unity. In other words, the covariance matrix of $\underline{\tilde{x}}$ is an identity matrix. Whitening can be done using eigenvalue decomposition (EVD) of the covariance matrix of mixed signals $\underline{x}$, $\boldsymbol{C}$. Let $\boldsymbol{V}$ be the matrix of eigenvectors of $\boldsymbol{C}$ and $\boldsymbol{\ddot{E}}$ the Diagonal matrix of eigenvalues of $\boldsymbol{C}$.

Whitening is done by

$$\underline{\tilde{x}} = \boldsymbol{\ddot{E}}^{-1/2} \boldsymbol{V}^{T} \underline{x} \qquad (3)$$

Whitening transforms mixing matrix A into $\underline{\tilde{A}}$ where $\underline{\tilde{A}} = \Lambda^{-1/2} V^T A$. A parallel architecture for whitening is shown in Figure 4

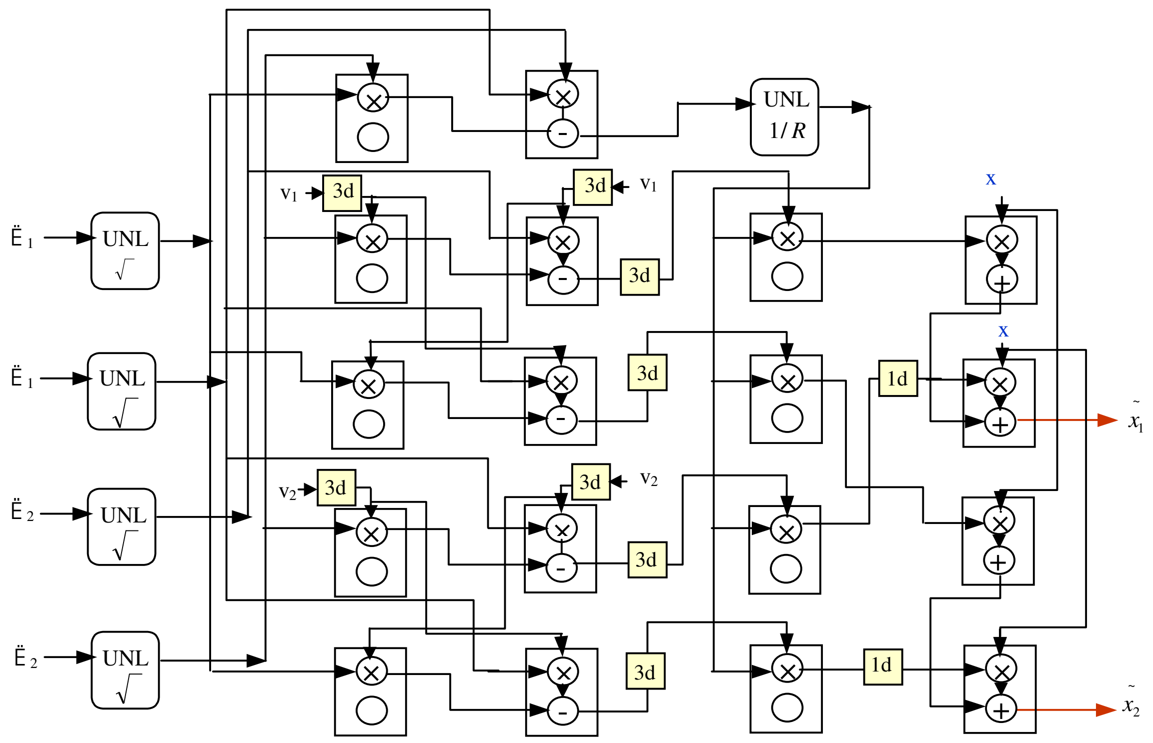


Figure 4 Parallel architecture for whitening

4 Iterative Computation

The fast ICA finds a direction, i.e. a unit vector $\underline{w}$ such that the projection $\underline{w}^T\underline{x}$ maximizes non-gaussianity. Non gaussianity is measured by negentropy $J(\underline{w}^T\underline{x})$ where

$$J(u) \approx [E\{G(u)\} - E\{G(z)\}]^2 \tag{4}$$

z is Gaussian variable of zero mean and unit variance (i.e. standardized). The variance of $\underline{w}^T\underline{x}$ has to be unity for the measure to be valid. For whitened data, this is equivalent to constraining the norm of w to be unity. To prevent different vectors from converging to the same maxima, we must decorrelate them after each iteration. For this, when we have estimated p vectors, $\underline{w}_1$, $\underline{w}_2$, …, $\underline{w}_p$, we run the algorithm for $\underline{w}_{p+1}$, and after every iteration step subtract from $\underline{w}_{p+1}$ the projection matrix B, whose columns are $\underline{w}_1$,$\underline{w}_2$, …, $\underline{w}_p$.

The algorithm consists of the following steps:

Step 1*: *Initialization: Choose initial random weight vector $\underline{\mathrm{w}}_n(0)$ with norm 1. Let B be the null matrix of the size of number of independent components.

Step 2*: *Iteration: Let the non-linear function be $G(u) = -\exp(-u^2/2)$. Then

$$\begin{aligned} g(u) &\underset{=}{\Delta}\ G'(u) = u\ \exp(-u^2/2) \\ g'(u) &= (1-u^2)\ \exp(-u^2/2) \end{aligned} \qquad (5)$$

The update of the nth row of $\boldsymbol{W}$ is given by $\underline{\mathrm{w}}_n(k+1)^T$.

$$\underline{\mathrm{w}}_n(k+1) = \mathrm{E}\{\underline{\mathrm{x}}\ \mathrm{g}(\underline{\mathrm{w}}_n(k)^{\mathrm{T}}\ \underline{\mathrm{x}})\ - \mathrm{E}\{\mathrm{g}'(\underline{\mathrm{w}}_n(k)^{\mathrm{T}}\ \underline{\mathrm{x}})\}\,\underline{\mathrm{w}}_n(k)\} \qquad (6)$$

$$\underline{\mathrm{w}}_n(k+1) <= \frac{\underline{\mathrm{w}}_n(k+1)}{\left\| \underline{\mathrm{w}}_n(k+1) \right\|} \qquad (6)$$

The Fast ICA algorithm can be mapped on to the J-platform as shown in Figure 5.

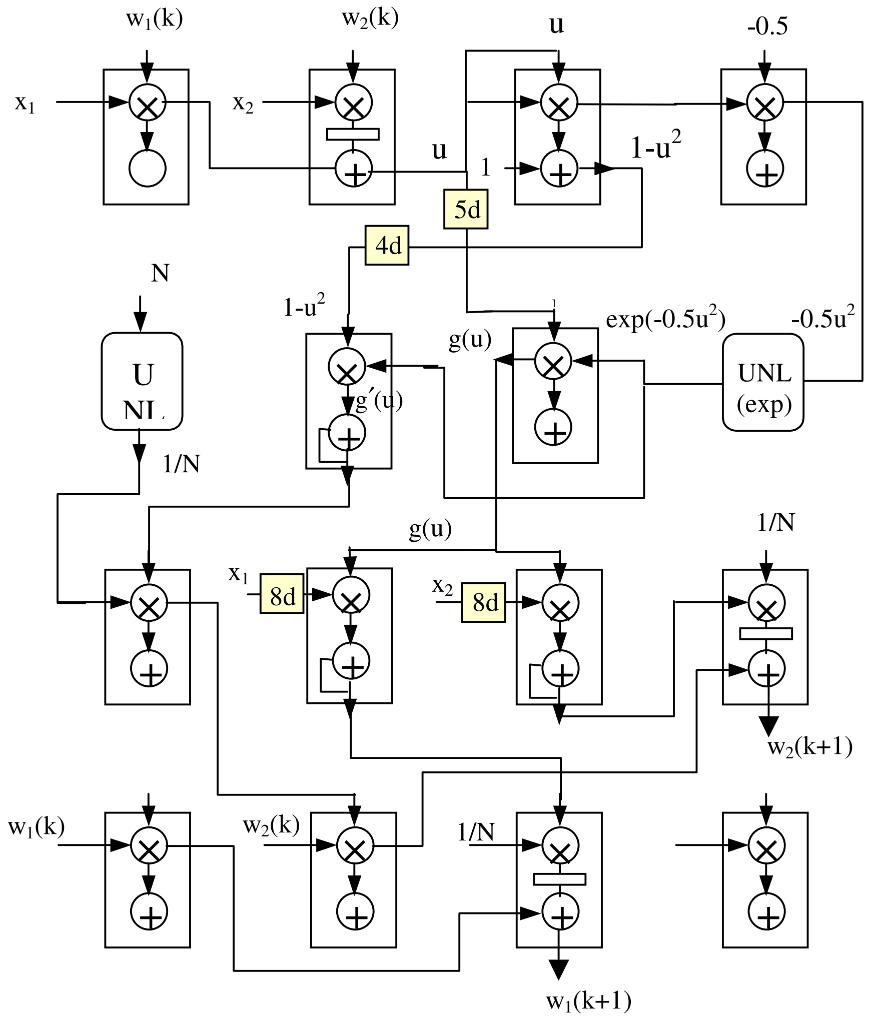


Figure 5 Fast ICA Algorithm

***Step 3*: *Decorrelation*:** To prevent the different vectors from converging to the same maxima, it needs to be decorrelated.

$$\underline{w}_n(k+1) = \underline{w}_n(k+1) - BB^T\,\underline{w}_n(k+1) \tag{7}$$

$$\underline{w}_n(k+1) = \frac{\underline{w}_n(k+1)}{\|\,\underline{w}_n(k+1)\,\|} \tag{8}$$

Step 4: If $\underline{\mathrm{w}}_n(k+1)$ and $\underline{\mathrm{w}}_n(k)$ have converged, then goto step 5, else increment *k* to *k* + 1 and goto step 2.

Step 5*:** Replace the nth column of ***B with $\underline{\mathrm{w}}_n(k+1)$. After whitening, add $\underline{\mathrm{w}}_n(k+1)^T$ as the nth row of ***W***. Increment n to $n+1$ and set $k = 0$. If $n \leq$ number of independent components, then goto step 2 else stop.

The parallel architecture for the decorrelation step is shown in Figures 6

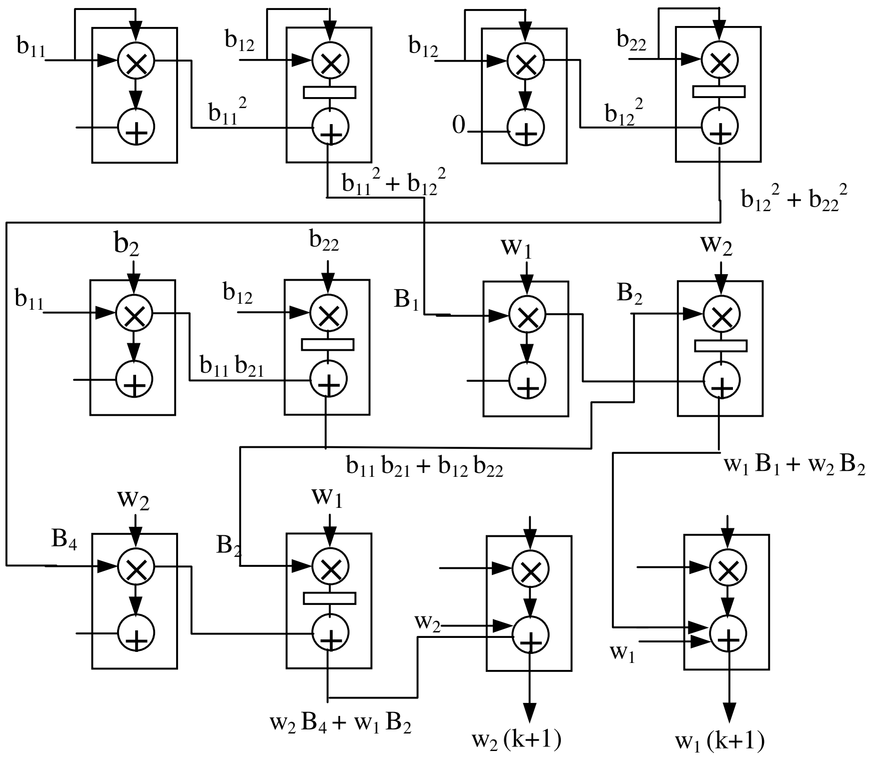


Figure 6 Parallel architecture for Decorrelation

5 Parallel architecture for Eigenvalue decomposition

In (3), ***V*** is the orthogonal matrix of eigenvectors and ***Ë*** is the diagonal matrix of its eigenvalues. ***Ë*** can be obtained by eigenvalue decomposition of the covariance matrix ***C***. A simplified diagram of the parallel architecture of the EVD algorithm is shown in the Figure 7. The details of the Figure 7 are shown in Figure 8 and Figure 9.

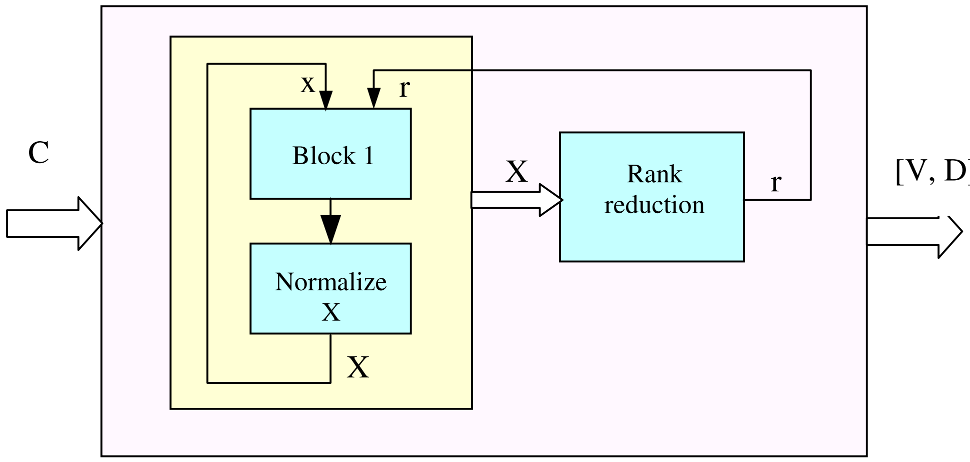


Figure 7 Symbolic architecture for EVD for a 4x4 symmetric real matrix

Because ***C*** is real and symmetric, its eigenvalues are real and non-negative. Therefore, a special algorithm can be used for its decomposition. The EVD algorithm has the following steps

Step 1*: *Initialization:

Choose a initial random $\underline{x}$ and normalize it Let M be is the size of the covariance matrix ***C***. Initialize ***R*** = ***C***.

Step 2*: *Iteration:

Find the smallest eigenvalue and corresponding eigenvector of ***R*** say *X*

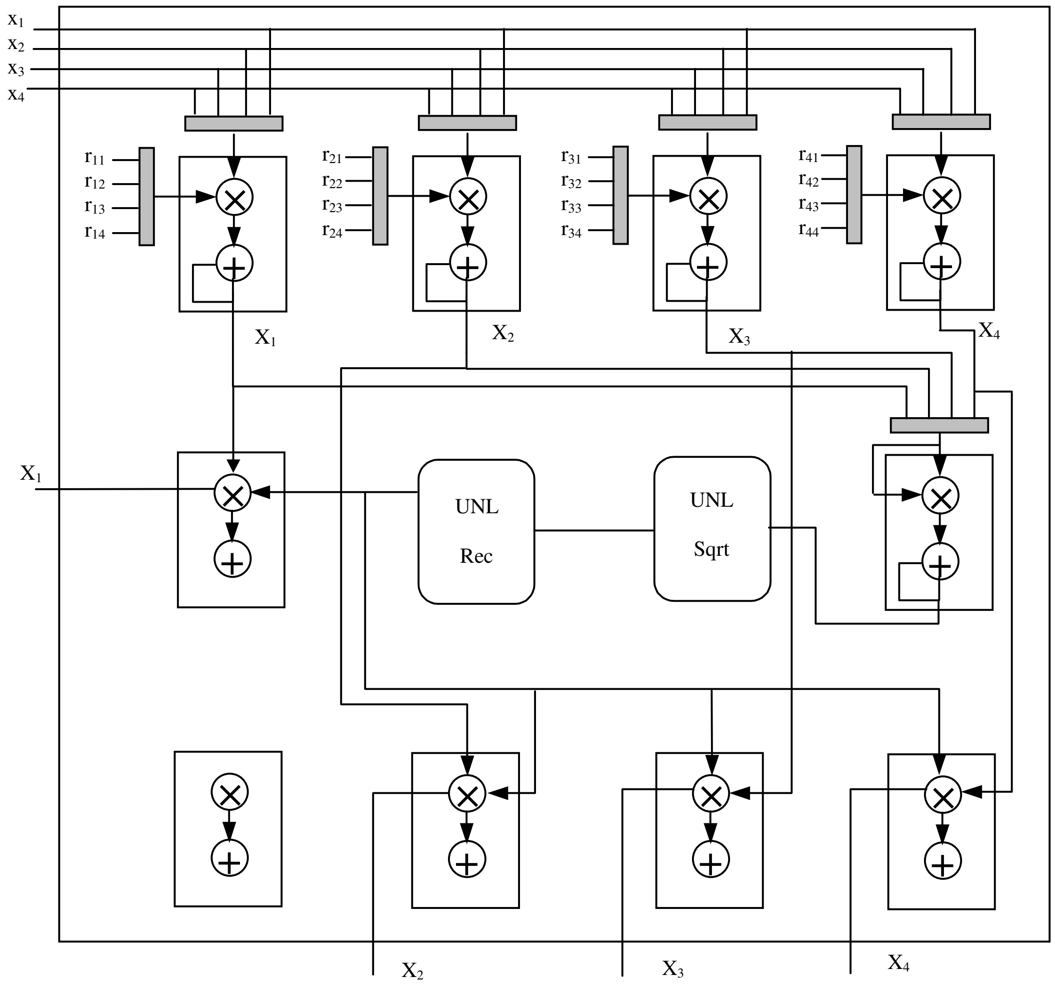


Figure 8 Details of Block 1 and Normalize X (Architecture for finding one eigen value and one eigen vector)

Step 3*: *Rank reduction:

The rank of the covariance matrix should be reduced by using the following expression

$$R^{+} = R - \lambda * X * X^{T} \tag{9}$$

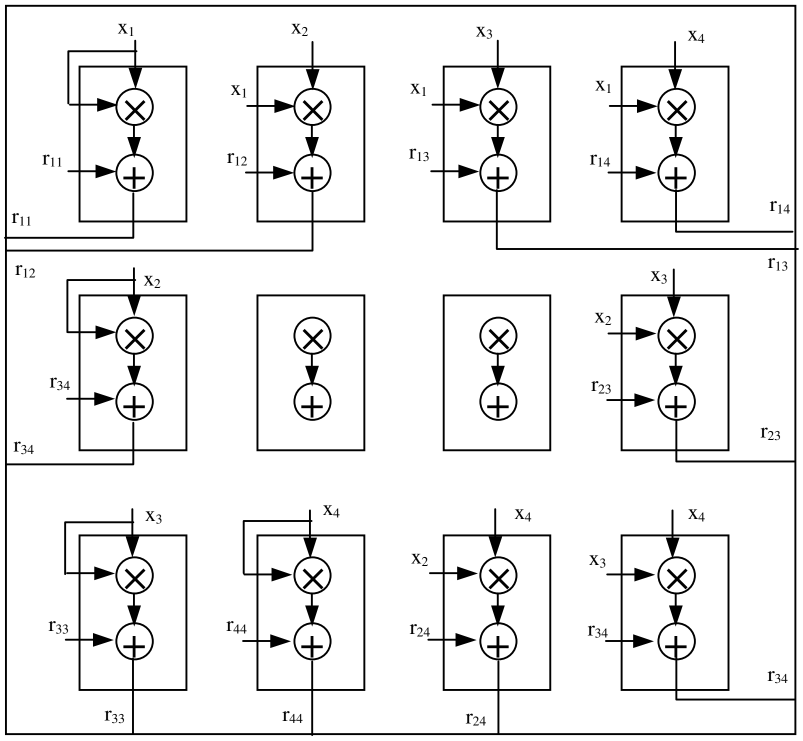


Figure 9 Details of Block 2

For the sake of simplicity, let the size of the covariance matrix be 4x4. Block1 finds the minimum eigenvalue and the corresponding eigenvector of a matrix *R*. The architecture requires 14 MA_PLUSs and 3 UNLs including the spares. The second block finds the norm of the input and normalizes the input vector. The third block is the Rank reduction block. This follows (2). The parallel architecture of EVD on J-Platfrom requires 28 MA_PLUSs and 3 UNLs including spares.

6 Discussion

In this chapter we have proposed some preliminary parallel architectures to implement the Fast ICA algorithm on J-platform. By considering individual steps of the algorithm for simple cases, it was shown that they can be mapped to the platform's coarse-grain cells: MAPLUS, UNL, and DF. Although an analysis was not performed, hardware versions have the potential for reducing the latency compared to the workstation analysis enormously. Some of the areas that should be explored in future are 1) mapping to limited and fixed resources on a J-platform chip, 2) timing consideration for synchronous data flow in the architecture, 3) reusability of the resources, 4) generalization to arbitrary number of signals, 5) block-by-block analysis of data, 6) precision considerations, and much more.