Low-Latency and Reconfigurable VLSI-Architectures for Computing Eigenvalues and Eigenvectors Using CORDIC-Based Parallel Jacobi Method

Rahul Sharma, *Graduate Student Member, IEEE*, Rahul Shrestha, *Senior Member, IEEE*, and Satinder K. Sharma, *Senior Member, IEEE*

***Abstract*— This article proposes a low-latency parallel Jacobi-method-based algorithm for computing eigenvalues and eigenvectors of $n \times n$-sized real-symmetric matrix. It is a coordinate rotations digital-computer (CORDIC)-based iterative algorithm that comprises multiple rotations and hence the key contribution of our work is to reduce the time cost of each rotation. Thus, alleviating the total latency for computing eigenvalues and eigenvectors using the parallel Jacobi method. Based on this proposed algorithm and additional architectural optimizations, a new low-latency and highly accurate VLSI-architecture has been presented in this manuscript for computing eigenvalues and eigenvectors of real-symmetric matrix. Subsequently, this work proposes a reconfigurable algorithm and its VLSI-architecture for computing eigenvalues and eigenvectors of complex Hermitian (CH), complex skew-Hermitian (CSH), and real skew-symmetric (RSS) matrices. Performance analysis of the proposed architectures has demonstrated minimal error-percentage of 0.0106% which is adequate for the wide range of real-time applications. The proposed architectures are hardware implemented on Zynq Ultrascale+ field-programmable gate array (FPGA)-board that consumed short latency of 9.377 μs while operating at maximum clock frequency of 172.75 MHz. Comparison of our implementation results with the reported works showed that the proposed architecture incurs 43.75% lower latency and 89.4% better accuracy than the state-of-the-art implementation.**

***Index Terms*— Coordinate rotations digital-computer (CORDIC), digital VLSI architectures, eigenvalues, eigenvectors, field-programmable gate array (FPGA), matrix theory, VLSI.**

I. Introduction

FUNDAMENTALLY, eigenvalues and eigenvectors are imperative information of matrix having significant role in various fields of science and engineering. In the year of 1948, Shannon [1] calculated theoretical limit of peak information that can be transmitted through the communication channel by computing its eigenvalues as well as eigenvectors and water-filling the eigenvalues. Such calculations are also extensively applied in the contemporary fields of computer vision, machine learning, image coding, object recognition, and image classification where eigenvalues and eigenvectors of a covariance matrix determine the principal directions of variation among the collection of images [2]–[4]. From an implementation aspect, cameras are the most suitable sensors for aforementioned image-processing applications that provide tremendous information related to the environment at high frame rates and hence lower latency is the prime requirement for its design. Subsequently, higher precision computation of such information is another key feature that enables to display images and videos with extremely high resolution.

Direction on arrival (DOA) estimation algorithms are widely used in array signal-processing applications [5]. However, its real-time hardware implementation is a challenging task due to heavy computational load incurred by eigenvalue decomposition of real-valued covariance matrix [6]–[8]. It results in large-area design that requires longer delay or latency to generate the output. Thus, low-latency hardware architecture with moderate area consumption for these algorithms is an utmost necessity for its real-time applications. In nutshell, determination of eigenvalues and eigenvectors is computationally complex and time-consuming process that is imperative to wide variety of applications. Our research goal is to design a hardware architecture that computes eigenvalues and eigenvectors of matrix with higher precision, lower latency, and moderate area requirement. Literature shows that QR [9] and Jacobi [10] methods are extensively used for generating eigenvalues and eigenvectors. Former has been designed for the general matrices, while the later (i.e., Jacobi method) is limited to only real-symmetric matrix. Nevertheless, Jacobi method delivers better performance with higher precision than QR method for computing eigenvalues and eigenvectors of real-symmetric matrix [10]. Due to inherent parallelism, contemporary hardware implementations are compliant to the Jacobi method which makes them highly appropriate for distributed resource systems [11]. Its parallel version (i.e., parallel Jacobi method) has a time complexity of $O\{n \times (\log_2 n)\}$ where n denotes the size of matrix whose eigenvalues and eigenvector are to be determined [12]. Such parallel Jacobi method is an iterative process in which every rotation consumes a constant amount of time, and the number of such rotations is fixed for a given value of n which primarily affects its time complexity. Thereby, optimizing such rotation time is an efficient way to alleviate an overall converge time cost of the parallel Jacobi method.

Manuscript received November 18, 2021; revised March 9, 2022 and April 5, 2022; accepted April 24, 2022. *(Corresponding author: Rahul Shrestha.)*

The authors are with the School of Computing and Electrical Engineering, IIT Mandi, Mandi 175075, India (e-mail: s18018@students.iitmandi.ac.in; rahul_shrestha@iitmandi.ac.in; satinder@iitmandi.ac.in).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2022.3170526.

Digital Object Identifier 10.1109/TVLSI.2022.3170526

In line with this notion, various reported works seek to reduce the time cost of the parallel Jacobi method [14]–[18]. These contributions apply a fundamental principle of dividing the matrix into sub-blocks with the aid of multiprocessor arrays and then performing plane rotation using the coordinate rotations digital-computer (CORDIC) algorithm. It simplifies trigonometric operations by converting them to hardware-friendly shift operations [19], and such trigonometric functions are extensively involved in the computations of parallel Jacobi method. As a result, CORDIC algorithm conserves the resource utilization while implementing the parallel Jacobi method in well-known hardware platforms like field-programmable gate array (FPGA) and application specific integrated circuit (ASIC). Therefore, the time cost of every rotation in parallel Jacobi method is primarily determined by the total computation time (latency) of CORDIC algorithm that is equivalent to the total number of CORDIC periods and hence it is desirable to alleviate such time [19], [21]. In the reported work by Brent and Luk [12], decompositions of singular-value and eigenvalue using the multiprocessor arrays have been presented that can be implemented with parallel architecture. However, it requires three CORDIC periods in each rotation to compute the eigenvalues and two CORDIC periods in every rotation to determine the eigenvectors. Later in [15], based on the hardware architecture from [14], Tao and Wei suggest the use of a sign-set instead of computing the rotation angle to alleviate the time cost of Jacobi method where only one CORDIC period per rotation is required to compute the eigenvectors. However, the off-diagonal processors in this technique consume two CORDIC periods to complete double rotation for eigenvalue calculation because the matrix elements of off-diagonal processors rotate with the angles received in both horizontal and vertical directions. Recently, Shi *et al.* [18] reported the one-off rotation-acceleration approach based on sign set computation to avoid the calculations of difference and summation for horizontal and vertical angles. This approach mitigates the time required to determine the rotation angles in advance. Thus, its time cost for each rotation is only two CORDIC periods for computing the eigenvalues that is better than the time required by [15] and [14]. On the other hand, our article proposes low-latency algorithm for the parallel Jacobi method that further accelerates the computations of eigenvalue and eigenvector. Here, the key contribution lies in the design of hardware architecture for CORDIC module based on the proposed low-latency algorithm that requires only one such CORDIC-module to compute the rotation matrix value. Subsequently, four more clock cycles are needed by the multiply-and-accumulate (MAC) units in our design to process this rotation-matrix value to perform double rotation. Therefore, the proposed work in this article requires one CORDIC period plus four clock cycles in each rotation of parallel Jacobi method for the computations of eigenvalues and eigenvectors. Furthermore, the reported systolic-array-based architectures in [14]–[18] for parallel Jacobi method have been designed to process only real-symmetric matrix. However, they are not proposed for processing CH and complex skew-Hermitian (CSH) matrices. López-Parrado and Velasco-Medina [20] reported a systolic-array-based architecture to calculate the eigenvalues of only CH matrix using complex arithmetic. In addition to low-latency algorithm as well as hardware architecture for computing eigenvalues and eigenvectors, this article also proposes a reconfigurable VLSI architecture for calculating these entities for CH, CSH, and real skew-symmetric (RSS) matrices, using parallel Jacobi method based on real arithmetic. Notion of designing such reconfigurable VLSI architecture is to conceive single architecture that is capable of supporting multiple applications. Such contribution makes our design suitable for the wide range of applications that alleviates area and power requirements, compared to individual designs for computing eigenvalues and eigenvectors of CH, CSH, and RSS matrices. For example, eigenvalue computations of complex Hermitian and real-symmetric matrices are used in massive multiple-in multiple-out (MIMO) and single-input single-output (SISO) systems, respectively [24]. Thereby, the proposed reconfigurable VLSI-architecture is applicable for both these systems in a single transceiver for better power and area conservations. Highlights of our contributions are as follows.

1) We propose a low-latency algorithm for computing eigenvalues and eigenvectors of $n \times n$ sized real-symmetric matrix by selectively applying the CORDIC algorithm.
2) Overall VLSI architecture and micro-architectures of its submodules for computing eigenvalues as well as eigenvectors of real-symmetric matrix that consumes lower latency have been presented here.
3) Additionally, this work presents a parallel Jacobi method-based low-latency and reconfigurable algorithm for calculating eigenvalues and eigenvectors of CH, CSH, and RSS matrices. Its corresponding reconfigurable VLSI-architecture has also been suggested in this article.
4) Subsequently, comprehensive analyses of error comparison and hardware complexity of the proposed VLSI architectures are carried out.
5) Suggested stand-alone and reconfigurable architectures for real-symmetric and CH/CSH/RSS matrices, respectively, are hardware implemented on FPGA platform and functionally validated, using the real-world hardware test-setup. Eventually, their implementation results are compared with the state-of-the-art works.

II. Preliminaries

A real-symmetric matrix $A \in \mathcal{R}^{n\times n}$ of $n \times n$ order can be transformed to a diagonal matrix using the sequence of Givens rotation as [22]

$$A_{k+1} \overset{\triangle}{=} \left[a_{i,j}^{k+1}\right] = R_{p,q,\theta^{(k)}}^{k} \cdot A_k \cdot \left\{R_{p,q,\theta^{(k)}}^{k}\right\}^{\tau} \tag{1}$$

where $k = \{1, 2, 3, \ldots\}$ rotations, τ denotes the matrix transpose, and $a_{i,j}$ represents element of ith row and jth column in A matrix. Note that $A_1 = A$ is the original real-symmetric matrix and $R_{p,q,\theta^{(k)}}^{k} \overset{\triangle}{=} [r_{i,j}^{(k)}]$ is the Givens rotation of order n. The $R_{p,q,\theta^{(k)}}^{k}$ matrix that represents each rotation is expressed as

$$R_{p,q,\theta^{(k)}}^{k} = \begin{array}{cc} \begin{array}{ccccccc} & & & & q\text{th} & & \end{array} & \\ \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & r_{p,p}^{(k)} & \cdots & r_{p,q}^{(k)} & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & r_{q,p}^{(k)} & \cdots & r_{q,q}^{(k)} & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} & \begin{array}{c} \\ \\ p\text{th} \\ \\ \\ \\ \\ \end{array} \end{array}. \tag{2}$$

For a given p and q values of kth rotation, the non-zero elements $(r_{i,j}^{(k)})$ of R^k matrix in (2) are represented as

$$\begin{aligned} r_{i,i}^{(k)} &= 1 \quad \forall\ i \neq p\ \&\ q \\ r_{p,p}^{(k)} &= r_{q,q}^{(k)} = \cos\left(\theta_{p,q}^{(k)}\right) \\ r_{p,q}^{(k)} &= -r_{q,p}^{(k)} = \sin\left(\theta_{p,q}^{(k)}\right), \quad p < q \end{aligned} \tag{3}$$

and rest of the elements are $r_{i,j}^{(k)} = 0$. Based on the classical Jacobi method, p and q values are obtained by equating

$$\left|a_{p,q}^{k}\right| = \max_{i \neq j} \left|a_{i,j}^{k}\right|. \tag{4}$$

Further, the angle $\theta_{p,q}^{(k)}$ is computed as

$$\theta_{p,q}^{(k)} = \frac{1}{2}\tan^{-1}\left(\frac{2 \times a_{p,q}^{k}}{a_{p,p}^{k} - a_{q,q}^{k}}\right), \quad \theta \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right] \tag{5}$$

to satisfy $a_{p,q}^{k+1} = 0$ and consequently, $a_{p,q}^{k}$ is observed to be zero in A_{k+1} matrix from (1). Such process is iteratively performed until all the off-diagonal elements of A matrix are zero. This is mathematically expressed as $\lim_{k\to\infty} A_{k+1} = \Lambda$ where $\Lambda \in \mathcal{R}^{n\times n}$ is a diagonal matrix in which eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ of A matrix are the diagonal elements of Λ matrix (i.e. $\Lambda = \mathbf{diag}\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$). Furthermore, the classical Jacobi method computes maximum value of $|a_{i,j}^{k}|$ in every rotation to satisfy (5) that is computationally complex process. Such method can be modified to exclude more than one off-diagonal portion in every rotation by selecting p and q values of some fixed orders. This makes Jacobi method suitable for the parallel computations [13].

If $A \in \mathcal{R}^{4\times 4}$ represents a real-symmetric matrix of the order 4×4 then the expression for first orthogonal rotation R_1 is

$$R_1 = \begin{bmatrix} r_{1,1}^{(1)} & r_{1,2}^{(1)} & 0 & 0 \\ -r_{1,2}^{(1)} & r_{2,2}^{(1)} & 0 & 0 \\ 0 & 0 & r_{3,3}^{(1)} & r_{3,4}^{(1)} \\ 0 & 0 & -r_{3,4}^{(1)} & r_{4,4}^{(1)} \end{bmatrix} \tag{6}$$

where its (p, q) elements are placed according to (3). Here, $\theta_{1,2}^{(1)}$ and $\theta_{3,4}^{(1)}$ angles are independently selected based on (5) such that the upper off-diagonal elements $a_{1,2}^{2}$ and $a_{3,4}^{2}$ of the $A_2 = R_1 \cdot A_1 \cdot R_1^{\tau}$ are excluded. Similarly, the subsequent rotations result new matrices like R_2 and R_3. With the aid of these matrices, pairs of elements $a_{1,3}^{3}, a_{2,4}^{3}$ and $a_{1,4}^{4}, a_{2,3}^{4}$ of A_3 and A_4 matrices, respectively, are reduced to zero. In this way, each of the $n(n-1)/2$ off-diagonal elements (above the main diagonal) are discarded after $(n-1)$ rotations where $n/2$ such elements are removed in each orthogonal rotation. Here, every $(n-1)$ orthogonal rotation is referred as sweep [13]. Subsequently, the second sweep comprises aforementioned process for R_4, R_5, and R_6 orthogonal matrices having the same constructions as R_1–R_3 matrices, respectively, for $A \in \mathcal{R}^{4\times 4}$ where $n = 4$. Such computations are iteratively performed until A matrix transforms into a diagonal matrix. In general, for an even-ordered real-symmetric A matrix, the components of each $(n-1)$ orthogonal rotations of R_k matrices for $k = 1, 2, 3, \ldots, (n-1)$ are given by [13]

$$\begin{aligned} r_{p,p}^{(k)} &= r_{q,q}^{(k)} = \cos\left(\theta_{p,q}^{(k)}\right) \\ r_{p,q}^{(k)} &= -r_{q,p}^{(k)} = \begin{cases} -\sin\left(\theta_{p,q}^{(k)}\right), & p > q \\ \sin\left(\theta_{p,q}^{(k)}\right), & p < q \end{cases} \end{aligned} \tag{7}$$

TABLE I
LIST OF (p, q) PAIRS OF ELEMENTS FOR R_k MATRIX (OF 16×16 ORDER) IN $k = 1, 2, 3, \ldots, 15$ DIFFERENT ROTATIONS

k				(p, q)	Pairs			
1	(1,14)	(2,13)	(3,12)	(4,11)	(5,10)	(6,9)	(7,8)	(15,16)
2	(1,12)	(2,11)	(3,10)	(4,9)	(5,8)	(6,7)	(13,15)	(14,16)
3	(1,10)	(2,9)	(3,8)	(4,7)	(5,6)	(11,15)	(12,14)	(13,16)
4	(1,8)	(2,7)	(3,6)	(4,5)	(9,15)	(10,14)	(11,13)	(12,16)
5	(1,6)	(2,5)	(3,4)	(7,15)	(8,14)	(9,13)	(10,12)	(11,16)
6	(1,4)	(2,3)	(8,12)	(9,11)	(5,15)	(6,14)	(7,13)	(10,16)
7	(1,2)	(8,10)	(3,15)	(4,14)	(5,13)	(6,12)	(7,11)	(9,16)
8	(1,15)	(2,14)	(3,13)	(4,12)	(5,11)	(6,10)	(7,9)	(8,16)
9	(1,13)	(2,12)	(3,11)	(4,10)	(5,9)	(6,8)	(7,16)	(14,15)
10	(1,11)	(2,10)	(3,9)	(4,8)	(5,7)	(6,16)	(13,14)	(12,15)
11	(1,9)	(2,8)	(3,7)	(4,6)	(5,16)	(11,14)	(12,13)	(10,15)
12	(1,7)	(2,6)	(3,5)	(4,16)	(10,13)	(9,14)	(11,12)	(8,15)
13	(1,5)	(2,4)	(3,16)	(7,14)	(8,13)	(6,15)	(10,11)	(9,12)
14	(1,3)	(2,16)	(7,12)	(4,15)	(5,14)	(6,13)	(8,11)	(9,10)
15	(1,16)	(2,15)	(3,14)	(4,13)	(5,12)	(6,11)	(7,10)	(8,9)

where p and q are the sequences defined as follows.

1) For

$$\begin{aligned} k &= 1, 2, 3, \ldots, n/2 - 1 \\ q &= n/2 - k + 1, n/2 - k + 2, \ldots, n - k \\ p &= \begin{cases} (n - 2k + 1) - q, & n/2 - k + 1 \leq q \leq n - 2k \\ 2(n - k) - q, & n - 2k < q \leq n - k - 1 \\ n, & n - k - 1 < q. \end{cases} \end{aligned} \tag{8}$$

2) For

$$\begin{aligned} k &= n/2, n/2 + 1, \ldots, n - 1 \\ q &= n - k, n - k + 1, \ldots, (3/2)n - k - 1 \\ p &= \begin{cases} n, & q < n - k + 1 \\ 2(n - k) - q, & n - k + 1 < q \\ & \quad \leq 2(n - k) - 1 \\ (3n - 2k - 1) - q, & 2(n - k) - 1 < q. \end{cases} \end{aligned} \tag{9}$$

On the other side, rest of the elements in R_k matrix are zero and $\theta_{p,q}^{k}$ angles for every kth rotation are calculated such that the $a_{p,q}^{k}$ elements are reduced to zero for all (p, q) pairs. For example, if $n = 16$ then all k (p, q) pairs of elements are determined based on (9), as shown in Table I. If A matrix becomes diagonal matrix after s sweeps or $r = s(n - 1)$ rotations then the diagonal entries of $A_{r+1} = W \cdot A \cdot W^{\tau}$ matrix are the eigenvalues and corresponding eigenvectors are the columns of $W^{\tau} = V_1^{\tau}, V_2^{\tau}, \ldots, V_s^{\tau}$ where $V_j^{\tau} = (R_1^{\tau})_j, (R_2^{\tau})_j, \ldots, (R_{n-1}^{\tau})_j$ for jth sweep.

III. PROPOSED ALGORITHM AND VLSI ARCHITECTURES

A. Proposed Low-Latency Algorithm

The proposed low-latency algorithm based on parallel Jacobi method has been presented in Algorithm 1 that processes $n \times n$ elements of a real-symmetric $A \in \mathcal{R}^{n\times n}$ matrix. It is only valid for even natural value of n; nevertheless, if n is an odd natural number then an additional row and a column of zero elements can be inserted in A matrix to convert n into an even natural number. After the initialization process, all the elements $\{(r_{pp}^{k})_i, (r_{qq}^{k})_i, (r_{pq}^{k})_i, (r_{qp}^{k})_i\}\ \forall\ i = \{1, 2, \ldots, n/2\}$ of R_k matrix are calculated using the proposed CORDIC-based algorithm, as illustrated in lines 11–22 of Algorithm 1. Here, each line calls ELEMENTS function whose processing is

Algorithm 1 Proposed Low-Latency Algorithm for Eigenvalues and Eigenvector Computations

```
1: Input: $A \in \mathcal{R}^{n\times n}$ matrix where $n$ is even natural number.
2: Input-2: $N$ = No. of iterations ($N$ represents bit-quantization of each element of $A_k$ matrix).
3: Input-3: $C$ = Scale-factor for $N$-bit accuracy.
4: Output: The Eigenvalues($\Lambda$) and Eigenvectors($W$) of $A$ matrix
5: Start the computations of eigenvalues and eigenvectors.
6: Initialize: $A_{k=1} = A$ and $V_{k=1} = I_n$.
7: $s = 0$
8: Repeat
9: $s = s + 1$
10: for $k \longrightarrow 1 \ \ to \ \ n-1$ do
11:     Start Parallel Execution        ▷ Compute $R_k$ matrix using equations (3-5) & Table I
12:     $\left\{(r^k_{pp})_1, (r^k_{qq})_1, (r^k_{pq})_1, (r^k_{qp})_1\right\}$=
13:             Elements($r_{pp}, r_{qq}, r_{pq}, r_{qp}, (a^k_{pq})_1, (a^k_{pp})_1, (a^k_{qq})_1, N, C$),
14:
15:     $\left\{(r^k_{pp})_2, (r^k_{qq})_2, (r^k_{pq})_2, (r^k_{qp})_2\right\}$=
16:             Elements( $r_{pp}, r_{qq}, r_{pq}, r_{qp}, (a^k_{pq})_2, (a^k_{pp})_2, (a^k_{qq})_2, N, C$),
17:
18:             ⋮                    ⋮
19:
20:     $\left\{(r^k_{pp})_{n/2}, (r^k_{qq})_{n/2}, (r^k_{pq})_{n/2}, (r^k_{qp})_{n/2}\right\}$=
21:        Elements($r_{pp}, r_{qq}, r_{pq}, r_{qp}, (a^k_{pq})_{n/2}, (a^k_{pp})_{n/2}, (a^k_{qq})_{n/2}, N, C$).
22:     End Parallel Execution
23:     $B = R_k \cdot A_k$
24:     $A_{k+1} = B \cdot R^{\tau}_k$
25:     $V_{k+1} = V_k \cdot R^{\tau}_k$
26: Until $A_{k+1}$ will not converge to diagonal matrix
27: Return $A_{k+1}$ and $V_{k+1}$
28: $\Lambda = A_{k+1}$                          ▷ Eigenvalues for A matrix
29: $W = V_{k+1}$                          ▷ Eigenvector for A matrix
30: End the computations of eigenvalues and eigenvectors.
31: Start of ELEMENTS function.
32: function Elements( $r_{pp}, r_{qq}, r_{pq}, r_{qp}, a_{pq}, a_{pp}, a_{qq}, N, C$)
33:     Initialize: $i = 0$, $d_{\theta_i}$ = -sign($X^i_v \cdot Y^i_v$), $Y^0_v = 2 \cdot a_{pq}$, $X^0_v = (a_{pp} - a_{qq})$,
        $X^0_r = C$, $Y^0_r = 0$                    ▷ Start Parallel Jacobi Method
34:     repeat:
35:         if $d_{\theta_i} < 0$, then
36:             $d_i$= -1
37:         else
38:             $d_i$= 1
39:         Start Parallel Execution
40:         Vector Mode: $X^{i+1}_v = X^i_v - d_i \cdot Y^i_v \cdot 2^{-i}$,
41:                      $Y^{i+1}_v = Y^i_v + d_i \cdot X^i_v \cdot 2^{-i}$;
42:         Rotation Mode: $X^{i+1}_r = X^i_r - d_i \cdot Y^i_r \cdot 2^{-i}$,
43:                        $Y^{i+1}_r = Y^i_r + d_i \cdot X^i_r \cdot 2^{-i}$;
44:         End Parallel Execution
45:         $i$=$i + 1$,
46:     until $i = N$, return $r_{pp} = r_{qq} = X^i_r, r_{pq} = Y^i_r, r_{qp} = -Y^i_r$.
47: End of ELEMENTS function.
```

shown in lines 31−47 of Algorithm 1. For the information, processing of ELEMENTS function in a conventional Jacobi method [13] can be described in the following manner. At first, the inputs are applied to compute $2 \times \theta^{(k)}_{p,q}$ value in vector mode and then right-shifted by one-bit to obtain $\theta^{(k)}_{p,q}$ value, as discussed earlier in (5). Further, this $\theta^{(k)}_{p,q}$ value is processed in the rotation mode to generate $\sin(\theta^{(k)}_{p,q})$ and $\cos(\theta^{(k)}_{p,q})$ that consume two CORDIC-periods ($2 \times T_c$) [19], [21]. Unlike such conventional method, ELEMENTS function of the proposed Algorithm 1 (illustrated in lines 31−47) uses CORDIC algorithm only in the rotation mode to determine elements of R_k matrix that shortens the computation time. Therefore, equations governing such CORDIC algorithm with N-bit accuracy for the rotation mode are

$$\begin{aligned} X^{(i+1)}_r &= X^i_r - Y^i_r \times d_i \times \tan(\phi_i) \\ Y^{(i+1)}_r &= Y^i_r + X^i_r \times d_i \times \tan(\phi_i) \\ \theta^{(i+1)}_r &= \theta^i_r - d_i \times \tan(\phi_i) \end{aligned} \tag{10}$$

where $d_i \in \{1, -1\}$ is the sign of CORDIC-rotation angle and ϕ_i is smaller rotation angle whose elements are $\phi_i = \tan^{-1}(2^{-i}) \ \forall \ i = \{0, 1, 2, \ldots, N-1\}$. This d_i value can be assigned based on the sign of θ_i in every smaller rotation such that the variable is sign d_i in each CORDIC operation. Note that the sign set of $2 \times \theta^i_{p,q}$ is represented as $d_{2\times\theta_i}$. It can also be determined based on the sign of $\tan(2 \times \theta^{(i)}_{p,q})$ in each CORDIC iterative process, instead of calculating the $2 \times \theta_{p,q}$ angle using the CORDIC in vector mode. Hence, the sign of $d_{2\times\theta_i}$ can be computed as

$$d_{2\times\theta_i} = \tan\left(2 \times \theta^{(i)}_{p,q}\right) = \left(\frac{2 \times a^{(i)}_{p,q}}{a^{(i)}_{p,p} - a^{(i)}_{q,q}}\right) \tag{11}$$

$\forall \ i = \{0, 1, 2, \ldots, N-1\}$. In parallel Jacobi method, there exist a condition when $|\theta^{(k)}_{p,q}| \leq \pi/4$ and thereby, $\theta^i_{p,q}$ is given by

$$\theta^{(i+1)}_{p,q} \approx \sum_{i=0}^{N-1} d_{\theta_i} \cdot \tan(\phi_i/2) \tag{12}$$

$\forall \ i = \{0, 1, 2, \ldots, N-1\}$. Thus, the sign set of $\theta_{p,q}$ is equivalent to $2 \times \theta_{p,q}$ (i.e., $d_{\theta_i} = d_{2\times\theta_i} \in \{1, -1\}$) and $\phi_i = \phi_i/2$ in (12). Therefore, ith elements of the sign set d_{θ_i} is processed by ith operation of CORDIC process. Thus, calculation of sign set and rotation can be performed in parallel, consuming one CORDIC-period T_c, as illustrated in lines 31−47 of Algorithm 1. Now, the new scale-factor $C = \prod \cos(\phi_i)$ for N-bit accuracy is precalculated as

$$\prod \cos(\phi_i) = \prod \cos(\phi_i/2). \tag{13}$$

In general, for an input A matrix of order n, $n/2$ number of such ELEMENTS functions are required to calculate all the rows of R_k matrix. Following that, each ELEMENTS function iterates N times where N is the bit-quantization of A matrix elements.

After the aforementioned computation of R_k matrix, implementation friendly double-rotation has been performed using the elements of R_k matrix to determine the eigenvalues (Λ), and subsequently the eigenvector (W), referring lines 23−29 of Algorithm 1. Here, the conventional computation of double rotation from (1) [13] has been segregated into two operations (i.e., lines 23 and 24 in Algorithm 1) for its hardware-friendly implementation. First, all elements of the R_k matrix are multiplied with A_k matrix to realize (1). For such multiplication, only upper-diagonal matrix of A_k needs to be multiplied with R_k matrix in each rotation that consequently generates $B = R_k \times A_k$ matrix, as presented in line-23 of Algorithm 1. Subsequently, the product of transpose of R_k and B matrices is computed, that corresponds to real-symmetric A_{k+1} matrix from line-24 in Algorithm 1. On the other hand, V_k matrix is initialized with unit matrix I_n of the order $n \times n$, as shown in line-6 of Algorithm 1. For the eigenvector computation, V_k matrix is multiplied with the transpose of R_k matrix to realize V_{k+1} matrix from line-25 in Algorithm 1. Aforementioned process iteratively runs till A_{k+1} matrix transforms into diagonal Λ matrix where all diagonal entries are the eigenvalues of original real-symmetric A matrix, as presented by line-28 in Algorithm 1. Furthermore, corresponding eigenvectors are the columns of matrix $W = \{w_{\lambda_1}, w_{\lambda_2}, \ldots, w_{\lambda_n}\}$, referring line-29 of Algorithm 1. Such segregation-based proposed hardware-friendly double-rotation process enables our VLSI architecture (for computing eigenvalues and eigenvectors) to resource share the MAC operations, as it will be presented in Section III-D.

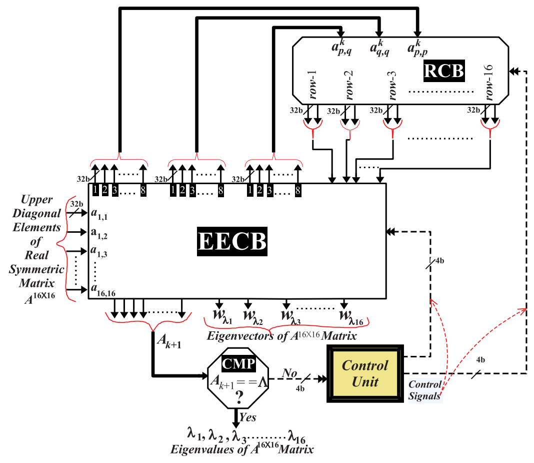
$a^k_{p,q}$
$a^k_{q,q}$
$a^k_{p,p}$
RCB
row-1
row-2
row-3
row-16
32b
EECB
Upper Diagonal Elements of Real Symmetric Matrix A^{16X16}
$a_{1,1}$
$a_{1,2}$
$a_{1,3}$
$a_{16,16}$
1 2 3 8
4b
A_{k+1}
$w_{\lambda1}$ $w_{\lambda2}$ $w_{\lambda3}$ $w_{\lambda16}$
Eigenvectors of A^{16X16} Matrix
CMP
$A_{k+1}==\Lambda$?
No
Yes
Control Unit
Control Signals
$\lambda_1, \lambda_2, \lambda_3 \ldots \lambda_{16}$
Eigenvalues of A^{16X16} Matrix

Fig. 1. Proposed system-level design for computing eigenvalues and eigenvectors of the real-symmetric $A \in \mathcal{R}^{16\times16}$ matrix.

Consequently, it enhances the hardware efficiency of suggested design by sharing the costly unit like multiplier.

B. System-Level Design for Computing Eigenvalues and Eigenvectors of Real-Symmetric Matrix

Proposed system-level design for computing eigenvalues and eigenvectors of the real-symmetric $A \in \mathcal{R}^{16\times16}$ matrix based on suggested Algorithm 1 is shown in Fig. 1. Primarily, all the upper triangular elements of $A^{16\times16}$ matrix (each represented as $a_{i,j}$ format with a bit width of 32-bit) are fed as inputs to this design. As discussed in Algorithm 1, parallel Jacobi method is an iterative process that begins with two simultaneous initializations: $A_1 = A^{16\times16}$ and $V_1 = I^{16\times16}$ in eigenvalue and eigenvector computation block (EECB), as shown in Fig. 1. At first, all the input $a_{i,j}$ elements of A matrix are buffered (using the registers) in EECB. Subsequently, elements corresponding to the kth rotation (denoted as $a^k_{i,j}$) are fed to R_k-matrix computation block (RCB) based on Table I where every row contains eight (p, q) pairs. Here, each of these pairs maps three elements: $a^k_{p,p}$, $a^k_{q,q}$, and $a^k_{p,q}$ from the registers in EECB. Thus, it transfers 24 elements (bundled into three groups with eight elements in each one) to RCB, as shown in Fig. 1. These elements are processed by RCB to generate $R_k \in \mathcal{R}^{16\times16}$ matrix. Consecutively, 16 rows (two elements from each row) of R_k matrix are routed to EECB to perform double rotation $A_{k+1} \triangleq [a^{k+1}_{i,j}] = R_k \cdot A_k \cdot R^{\tau}_k$ and $V_{k+1} = [v^{k+1}_{i,j}] = V_k \cdot R^{\tau}_k$ transformations. In EECB, computed elements of A_{k+1} matrix are buffered in the same register of input $a_{i,j}$ elements and are further fed to RCB, as shown in Fig. 1. Such processes in EECB iterate until A_{k+1} matrix converts into diagonal matrix (Λ). Thereafter, the diagonal elements (i.e., $\lambda_1, \lambda_2, \ldots, \lambda_{16}$) of Λ matrix are the eigenvalues of real-symmetric A matrix. Eventually, eigenvectors are the columns of V_{k+1} matrix that is generated by EECB as $W = V_{k+1} = \{w_{\lambda_1}, w_{\lambda_2}, \ldots, w_{\lambda_{16}}\}$, referring line-29 in Algorithm 1. On the other hand, the suggested system-level architecture described above is also capable of handling real-symmetric matrices with the orders that are less than 16×16. For example, to calculate eigenvalues and eigenvectors of a matrix of size 12×12, this matrix must be first transformed into a 16×16 real-symmetric matrix by adding zero entries in its last four rows and columns. However, the current architecture shown in Fig. 1 defers to support any real-symmetric matrix with an order higher than 16×16. Nevertheless, this design can be upgraded for higher order matrix that still delivers lower latency and enhanced hardware efficiency, at the cost of time to design.

In the general case of an input A matrix of order n, the proposed EECB-architecture receives $n(n+1)/2$ upper-triangular elements of A matrix. Referring Table I, $n/2$ number of (p, q) pairs are precomputed using (8) and (9). Each of these pairs corresponds to three elements: $a^k_{p,p}$, $a^k_{q,q}$, and $a^k_{p,q}$. They are fed to RCB in three groups of $n/2$ elements each for generating the R_k matrix. Consecutively, n rows (two entries from each row) of the R_k matrix are routed to EECB for double rotation $A_{k+1} \triangleq [a^{k+1}_{i,j}] = R_k \cdot A_k \cdot R^{\tau}_k$ and $V_{k+1} = [v^{k+1}_{i,j}] = V_k \cdot R^{\tau}_k$ transformations. This process is iteratively performed until A_{k+1} matrix is converted into a diagonal Λ matrix where all its diagonal elements are the eigenvalues of input real-symmetric A matrix. Additionally, columns of W matrix have the corresponding eigenvectors that are represented as $W = \{w_{\lambda_1}, w_{\lambda_2}, \ldots, w_{\lambda_n}\}$.

C. VLSI Architecture of RCB

This module processes 24 elements of A matrix bundled into $a^k_{p,p}$, $a^k_{q,q}$, and $a^k_{p,q}$ which are generated from EECB for the kth rotation corresponding to various (p, q) pairs, as listed in Table I. The proposed RCB architecture is shown in Fig. 2(a) where eight (p, q)-pairs processing blocks (PQBs) are fed with 24 input elements which are segregated in such a way that each PQB processes three elements (of 32 bit each) from $a^k_{p,p}$, $a^k_{q,q}$, and $a^k_{p,q}$ bundles. Each PQB design primarily incorporates iterative CORDIC computation for realizing the elements of R_k matrix based on (7) and Algorithm 1. Subsequently, these elements from PQBs are passed into a multiplexing network that generates 16 rows of R_k matrix, as shown in Fig. 2(a). Here, every row comprises of only two elements of R_k-matrix (instead of 16 elements) and rest are nullified, as discussed earlier in Section II. Subsequently, the generalized VLSI-architecture of RCB requires $n/2$ number of PQBs while processing the input matrix of order n. Each PQB generates output lines, thus r^k_{pp}, r^k_{pq}, and r^k_{qq}. These output lines are transferred to a multiplexing network to arrange the rows of the R_k matrix for the kth rotation. Furthermore, $2 \times n$ number of $\{(n-1){:}1\}$-sized multiplexers are needed in the general RCB multiplexing-network block to map the rows of R_k matrix. On the other hand, VLSI architecture of PQB from Fig. 2(b) remains unchanged for different values of input matrix order (i.e., n). However, the size of all computation units in a PQB-i will upscale with the increase in bit widths of $(a^k_{p,p})_i$, $(a^k_{q,q})_i$, and $(a^k_{p,q})_i$.

The proposed micro-architecture of single PQB—first PQB in Fig. 2(a)—has been presented in Fig. 2(b). It processes three 32-bit elements: $(a_{pp})_1$, $(a_{qq})_1$, and $(a_{pq})_1$ which are tapped from $a^k_{p,p}$, $a^k_{q,q}$, and $a^k_{p,q}$ bundles, respectively, to generate elements of the R_k-matrix, referring (7). To begin with, input element $(a_{pq})_1$ is left-shifted by one-bit position and other two elements $(a_{pp})_1$ and $(a_{qq})_1$ are subtracted, based on (5). Furthermore, $Y = |2 \cdot (a_{pq})_1|$ and $X = |(a_{pp})_1 - (a_{qq})_1|$ are obtained from absolute computation units (ACUs) and are fed to sign estimator (SEM) of the CORDIC block, as presented in line-32 of Algorithm 1 and shown in Fig. 2(b). Here, SEM generates the sign set d_{θ_i} $\forall$ $i = \{0, 1, 2, \ldots, N-1\}$ where

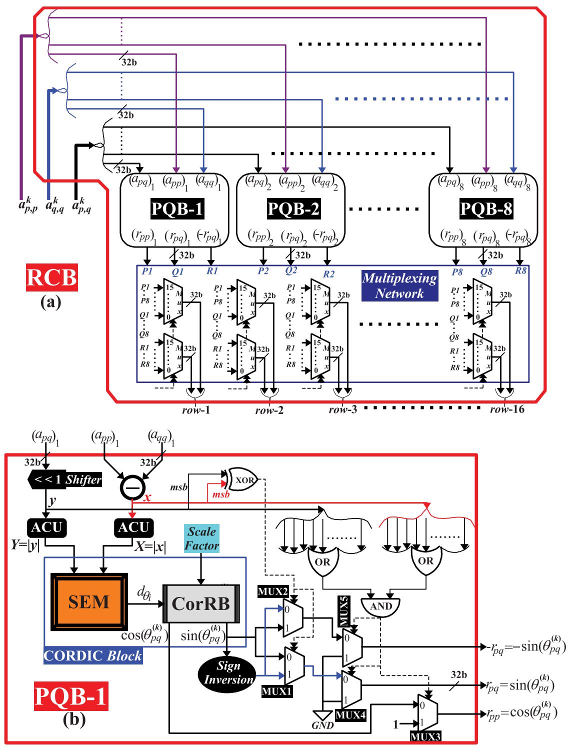

Fig. 2. (a) Proposed VLSI architecture of RCB for computing the elements of R_k matrix. (b) Micro-architecture for one of the CORDIC-based PQBs used in RCB.

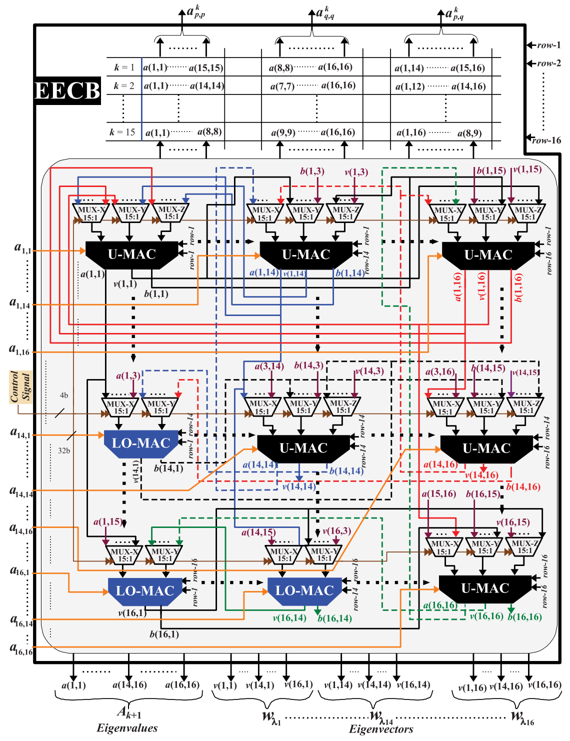

Fig. 3. Proposed VLSI architecture of EECB for real-symmetric $A^{16\times16}$ matrix.

N represents the maximum number of CORDIC iterations. These d_{θ_i} signs are sequentially passed to CORDIC rotation-mode block (CorRB) along with the precalculated scale factor (i.e., C), as discussed earlier in Section III-A and shown in Fig. 2(b). Subsequently, CorRB generates $\sin(\theta_{pq}^{(k)})$ and $\cos(\theta_{pq}^{(k)})$, referring (5). For $|\theta_{pq}^{(k)}|\leq\pi/4$, $\cos(\theta_{pq}^{(k)})$ is always a positive value and $\sin(\theta_{pq}^{(k)})$ has the same sign as $2\cdot(a_{pq})_1/\{(a_{pp})_1-(a_{qq})_1\}$. The later condition has been realized using MUX1 multiplexer whose select line is generated by XORing the most significant bits (MSBs) of x and y values, as shown in Fig. 2(b). On the other hand, if $(a_{pq})_1=(a_{pp})_1=(a_{qq})_1=0$ then it is an undefined form and such case must be skipped while estimating the sign set d_{θ_i}. To mitigate this, a fixed value of $|\sin(\theta_{pq}^{(k)})|=0$ and $|\cos(\theta_{pq}^{(k)})|=1$ has been assigned in our design. This is realized using MUX3–MUX5 multiplexers and their select lines are obtained by bitwise ORing and ANDing x and y values in our PQB-architecture from Fig. 2(b). Thus, rest seven PQBs of RCB work in similar manner, as discussed above. Eventually, all the outputs of eight PQBs are arranged into 16-rows of R_k matrix with the aid of 16:1 multiplexers for the kth rotation, as illustrated in Fig. 2(a), and further applied to EECB.

D. VLSI Architecture of EECB

As shown in Fig. 1, all the upper-triangular $a_{i,j}$ elements (i.e., $a_{1,1}$–$a_{16,16}$) of real-symmetric $A^{16\times16}$ matrix are iteratively processed by EECB to generate its eigenvalues and eigenvectors based on the double rotation and $V_{k+1}=[v_{(i,j)}^{k+1}]=V_k\cdot R_k^{\tau}$ transformation, respectively. The double rotation is performed in two parts: first part computes $B=[b_{i,j}]=R_k\cdot A_k$ matrix and second part calculates $A_{k+1}=B\cdot R_k^{\tau}$ matrix, using the previously computed elements of $B=[b_{i,j}]$ matrix, as illustrated by lines 23–24 in Algorithm 1. Thereafter, EECB computes V_{k+1} matrix based on $V_{k+1}=[v_{(i,j)}^{k+1}]=V_k\cdot R_k^{\tau}$ transformation that eventually generates eigenvectors of the real-symmetric $A^{16\times16}$ matrix, referring line-25 of Algorithm 1. On the other side, 16 rows of R_k matrix from RCB are fed to EECB in every kth rotation, as shown in Fig. 1. The proposed VLSI architecture of EECB is shown in Fig. 3 that comprises of two kinds of MAC units: 1) MAC unit for upper-triangular element (i.e., U-MAC) and 2) MAC unit for lower off-diagonal element (i.e., LO-MAC). Here, the double rotation and V_{k+1} matrix transformation for real-symmetric $A^{16\times16}$ matrix are performed with the aid of 136 U-MACs and 120 LO-MACs. An overview of interconnections among these U-MACs and LO-MACs along with the steering logic is presented in Fig. 3. It shows that the input upper-triangular $a_{i,j}$ elements are first fed to respective MAC units. In addition, each U-MAC is fed with the outputs of three 15:1 multiplexers: MUX-X, MUX-Y, MUX-Z; and two rows (i.e., *row*-1/2/3/. . ./16) of R_k matrix from RCB, referring Figs. 1 and 3. Furthermore, Fig. 3 shows that the outputs of two 15:1 multiplexers (MUX-Y and MUX-Z) and two rows of R_k matrix are fed to LO-MAC. To perform double rotations in every kth rotation, inputs to MUX-X are upper-triangular elements $a_{i,j}$ which are the outputs of other U-MACs from previous $(k-1)$th rotation. Similarly, inputs of MUX-Y are $b_{i,j}$ elements and they are the outputs of other U-MACs and LO-MACs in the same kth rotation, as shown in Fig. 3. After the calculation of the double rotation in the MAC units, EECB computes V_{k+1} matrix based on $V_{k+1}=[v_{(i,j)}^{k+1}]=V_k\cdot R_k^{\tau}$ transformation for the real-symmetric $A^{16\times16}$ matrix. Prior to the transformation begins in EECB, one of the registers (i.e., REG-4) in all the MAC units are initialized with 0/1 to

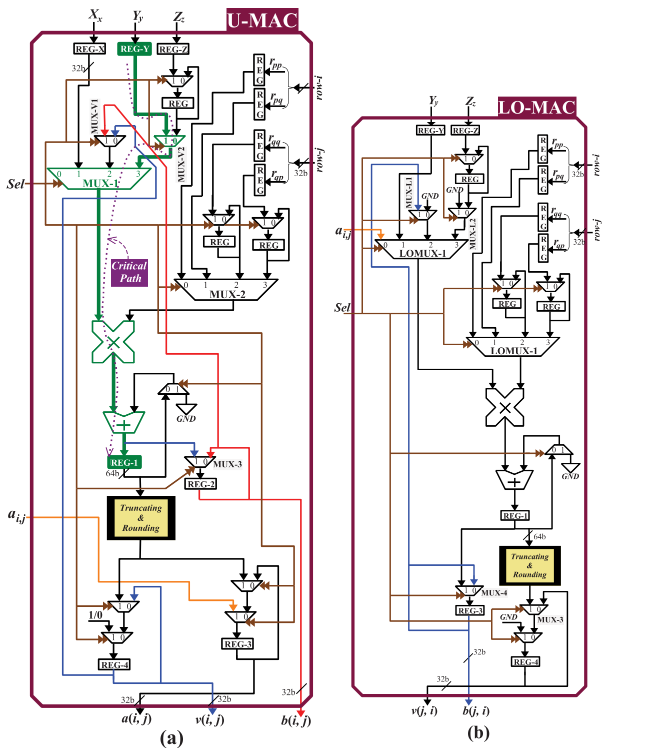

Fig. 4. Suggested micro-architectures of two MAC units of EECB: (a) U-MAC and (b) LO-MAC.

load $V_{k=1} = I_{16\times16}$, as presented in line-2 of Algorithm 1. Furthermore, each MAC unit is connected with an output of 15:1 multiplexer (i.e., MUX-Z) at the input side, as shown in Fig. 3. It also shows that the outputs of all MAC units are $\upsilon(i, j)$ elements of $(k-1)$th rotation and they are the inputs of MUX-Zs for each kth rotation. In the first two clock cycles, outputs of each U-MAC and LO-MAC are the elements of B matrix. In the next two clock-cycles, U-MAC outputs are upper off-diagonal elements of A_{k+1} matrix. In the subsequent two clock-cycles all MAC-outputs are the elements of $V_{k+1} = [\upsilon(i, j)]$ matrix of every kth rotation.

Subsequently, output elements of A_{k+1} matrix of all MAC units are also fed back to the RCB in every kth rotation, as shown in Fig. 1. In EECB architecture, such double rotation is performed iteratively until the A_{k+1} matrix transforms into a diagonal Λ matrix. Thereafter, the diagonal entries of $A_{k+1} = [a_{i,i}^{k+1}]$ matrix $\{\lambda_1, \lambda_2, \ldots, \lambda_{16}\}$ are delivered as the output eigenvalues of real-symmetric $A^{16\times16}$ matrix with the aid of comparator, as shown earlier in Fig. 1. Instant the A_{k+1} matrix is transformed into Λ diagonal matrix, all $\upsilon(i, j)$ outputs of all MACs in EECB generates the eigenvectors $\{\omega_{\lambda_1}, \omega_{\lambda_2}, \ldots, \omega_{\lambda_{16}}\}$ of real-symmetric $A^{16\times16}$ matrix that corresponds to its eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_{16}\}$, as shown in Fig. 2. To determine eigenvalues and eigenvector of A matrix of order n using the proposed EECB architecture, following units are required: $n(n+1)/2$ number of U-MACs and $n(n-1)/2$ number of LO-MACs. For 32-bit width, the suggested architectures of U-MAC and LO-MAC remain the same for any order size (i.e., n) of A matrix. Additionally, U-MAC and LO-MAC units are coupled with three $\{(n-1){:}1\}$-sized multiplexers (i.e., MUX-X, MUX-Y, and MUX-Z) and two $\{(n-1){:}1\}$-sized multiplexers (i.e., MUX-X and MUX-Y), respectively.

1) Micro-Architectures of U-MAC and LO-MAC Units:

Suggested U-MAC micro-architecture is shown in Fig. 4(a) that processes X_x, Y_y, Z_z (outputs of MUX-X, MUX-Y, and MUX-Z of EECB, respectively), and two rows from RCB to generate upper triangular elements of $A^{k+1} = [a_{i,j}^{k+1}]$, $B = [b_{i,j}]$, and $V_{k+1} = [\upsilon_{i,j}^{k+1}]$ matrices. It comprises of two 4:1 multiplexers (MUX-1 and MUX-2) where the inputs of MUX-1 are the outputs from REG-3 register, X_x, MUX-$V1$ and MUX-$V2$. Similarly, MUX-2 has been fed with the rows of R_k matrix from RCB, as shown in Fig. 4(a). It shows that the select line (i.e., Sel) of these multiplexers routes their outputs to a multiplier, only after RCB finishes the computation of rows. This product is cumulatively added with the content of REG-1 register and stored in it. If Sel = 1 then this adder output is separated using MUX-3 and stored in REG-2 register which is the element of $B = [b_{i,j}]$ matrix, as illustrated in Fig. 4(a). Subsequently, the stored data from REG-1 register is transferred to REG-3 register, when the select line value Sel = 4. At this moment, the content of REG-3 register is the final output elements of $A^{k+1} = [a_{i,i}^{k+1}]$ matrix. In the next four clock-cycles when the Sel = 8, REG-4 register stores the output upper-triangular element of $V_{k+1} = [\upsilon_{i,j}^{k+1}]$ matrix. In this manner, all the U-MACs of EECB operate in the aforementioned way for every kth rotation.

On the other hand, Fig. 4(b) presents an LO-MAC micro-architecture of EECB that processes outputs of MUX-Y (Y_y), MUX-Z (Z_z), and two rows of R_k matrix from RCB in every kth rotation. The LO-MAC outputs are lower off-diagonal elements of $V_{k+1} = [\upsilon_{i,j}^{k+1}]$ and $B = [b_{j,i}]$ matrices, respectively, as shown in Fig. 4(b). For computing the elements of $B = [b_{j,i}]$ matrix in LO-MAC architecture, two 4:1 multiplexers (LOMUX-1 and LOMUX-2) steer the inputs to a multiplier for the MAC operation. The inputs of LOMUX-1 are conjugate $a(i, j)$ element of the A^{k+1} matrix, Y_y, MUX-$L1$, and MUX-$L2$ outputs. Similarly, the inputs of LOMUX-2 are the rows of R_k matrix, as shown in Fig. 4(b). Outputs from LOMUX-1 and LOMUX-2 are multiplied, added and subsequently stored in REG-1 register. Its content is further transferred to REG-3 register via MUX-4, when the value of Sel = 1. Eventually, this value in REG-2 register is the output element of $B = [b_{j,i}]$ matrix, as illustrated in Fig. 4(b). Once the value of Sel = 8, output of REG-1 register is transferred to REG-4 register via MUX-3. Finally, this REG-4 content is the final $\upsilon(j, i)$ output-element of V_{k+1} matrix in every kth rotation.

E. Timing Analysis

Detailed analysis of latency consumed by the proposed VLSI architecture that iteratively computes eigenvalues and eigenvectors of real-symmetric $A^{n\times n}$ matrix has been presented here. Referring Algorithm 1, $k = n-1$ rotations are first performed and thereafter, $s = N_s$ sweeps (i.e., equivalent to $r = N_s \times (n-1)$ rotations) are performed until $A^{n\times n}$ matrix transforms into $\Lambda^{n\times n}$ diagonal matrix (i.e., N_s sweeps are required to convert $A^{n\times n} \rightarrowtail \Lambda^{n\times n}$). For better understanding, computation time required for each kth rotation has been segregated into two parts: T_1 and T_2. Referring the overall architecture in Fig. 1, the operations executed during these durations are as follows.

- *T_1:* To calculate the rows of R_k matrix in RCB and are made available at the inputs of EECB.
- *T_2:* To perform one double-rotation in EECB.

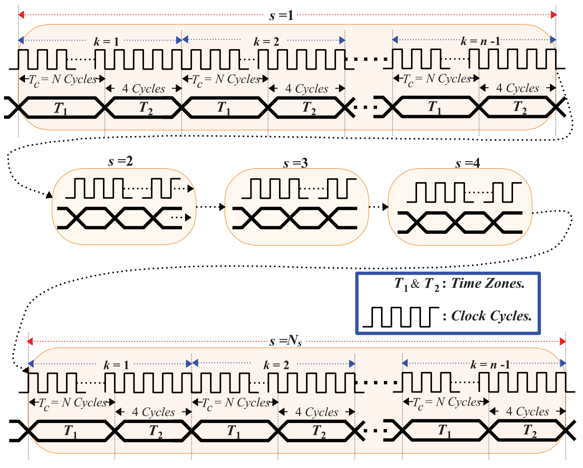

Fig. 5. Timing diagram for quantifying the latency of proposed VLSI architecture for computing eigenvalues and eigenvectors of real-symmetric $A^{n\times n}$ matrix.

Algorithm 2 Proposed Reconfigurable Algorithm to Compute Eigenvalues and Eigenvectors of CH or CSH Matrices

```
1: Input: $M = [m^r + i\cdot m^i] \in \mathcal{C}^{n\times n}$ matrix where $m^r \in \mathcal{R}^{n\times n}$ & $m^i \in \mathcal{R}^{n\times n}$.
2: Input: Input control-signal $\phi$ = 0/1.
3: if $\phi$ == 1 then
4:     $S_{CH} = \begin{bmatrix} m^r & (m^i)^\tau \\ m^i & m^r \end{bmatrix}_{2n\times 2n}$   ▷ $M^{n\times n}$ matrix converts into a real-symmetric $S$ matrix for CH matrix.
5: else
6:     $S_{CSH} = \begin{bmatrix} m^i & (m^r) \\ (m^r)^\tau & m^i \end{bmatrix}_{2n\times 2n}$   ▷ $M^{n\times n}$ matrix converts into real-symmetric $S$ matrix for CSH matrix.
7: Equate $A^{2n\times 2n} = S^{2n\times 2n}$.
8: Repeat the steps between lines 2–12 from Algorithm 1.
9: $\Lambda$ = diag($\lambda_1$,$\lambda_1$, $\lambda_2$,$\lambda_2$, $\cdots$, $\lambda_n$,$\lambda_n$)= $A_{k+1}$   ▷ Twice eigenvalues for $M^{n\times n}$ matrix.
10: $W = V_{k+1}$   ▷ Eigenvector that corresponds to the eigenvalues ($\Lambda$).
```

For the overall computation of latency, timing diagram of the proposed architecture has been presented in Fig. 5. Here, T_1 duration is equivalent to one CORDIC-period T_c at the RCB to compute R_k matrix. Note that T_c is quantified by the input bit width N of CorRB architecture in RCB design (i.e., $T_c = N$ clock cycles), as shown in Fig. 2(b). On the other hand, T_2 duration requires four clock cycles for computing one double rotation. Consequently, Fig. 5 shows that an aggregated time required for a single kth rotation is given by $N_{\text{orc}} = T_c + 4$ and thus, total-time/latency ($\mathcal{L}$) needed to compute eigenvalues and eigenvectors of real-symmetric $A^{n\times n}$ matrix is

$$\mathcal{L} = \sum_{s=1}^{N_s}\left[\sum_{k=1}^{n-1}(N_{\text{orc}})_k\right] = \sum_{s=1}^{N_s}\left[\sum_{k=1}^{n-1}(T_c+4)_k\right] \tag{14}$$

clock cycles.

IV. Reconfigurable Algorithm and VLSI Architecture

A. Proposed Reconfigurable Algorithm

In addition to eigenvalues and eigenvectors calculations of the real-symmetric matrix, parallel Jacobi method is also widely used for computing these entities for CH and CSH

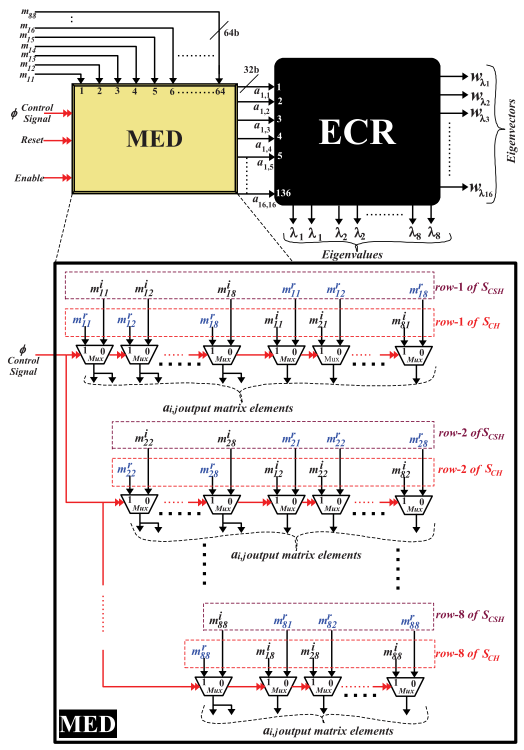

Fig. 6. Proposed reconfigurable architecture for computing eigenvalues and eigenvectors of CH or CSH matrices, and VLSI architecture of MED multiplexer network.

matrices [23]. These CH and CSH matrices are first transformed into a real-symmetric matrix which is then processed using the same algorithm (i.e., Algorithm 1), presented in Section III-A to obtain their eigenvalues and eigenvectors. This article presents an efficient method for transforming such CH and CSH matrices into real-symmetric matrices. Further, such transformation has been tailored with Algorithm 1 to conceive a reconfigurable algorithm that is presented in Algorithm 3. It has been designed to calculate eigenvalues and eigenvectors of CH or CSH matrices, depending on the magnitude of control signal ϕ. Let $M = [m^r + i\cdot m^i] \in \mathcal{C}^{n\times n}$ be input CH or CSH matrix (i.e., $\phi = 1/0$ for CH/CSH matrix). Here, m^r and m^i represent real and imaginary parts, respectively, of all $n\times n$ elements in M matrix. The suggested algorithm for transforming the input which is a complex matrix into a real-symmetric matrix $S_{\text{CH/CSH}} \in \mathcal{R}^{2n\times 2n}$ has been presented from lines 1–6 in Algorithm 3. Subsequently, such $S^{2n\times 2n}_{\text{CH/CSH}}$ matrix is being processed by lines 1–12 of Algorithm 1 (from Section III-A) for computing eigenvalues and eigenvectors of CH or CSH matrix, depending on the ϕ value. Here, $A^{2n\times 2n} = S^{2n\times 2n}_{\text{CH/CSH}}$ matrix converges to a diagonal Λ matrix of **diag**(λ_1, λ_1, λ_2, $\lambda_2, \ldots, \lambda_n$, λ_n) that preserves twice the number of eigenvalues of $n \times n$ CH or CSH matrix, without alleviating the accuracy. Eventually, corresponding eigenvectors are the columns of the $W = V_{k+1}$ matrix, referring to lines 9 and 10 from Algorithm 3.

B. Suggested Reconfigurable VLSI-Architecture

A system-level design for computing eigenvalues and eigenvectors of CH/CSH matrices based on Algorithm 3 is shown in Fig. 6. It is an aggregation of two major blocks: 1) matrix element distributor (MED) and 2) eigenvalues-and-eigenvectors computation-unit for real-symmetric-matrix (ECR) which is earlier presented in Fig. 1. Here, MED architecture has been designed to process CH/CSH matrix of the size 8×8 (i.e., $M \in \mathcal{C}^{8\times8}$) such that it generates 16×16 real-symmetric $S^{16\times16}$ matrix. Thus, it becomes suitable input for the subsequent ECR to process real-symmetric $A^{16\times16}$ matrix, as discussed earlier in Section III-B. At the input side, all 64 elements of $M = [m^r + i \cdot m^i] \in \mathcal{C}^{8\times8}$ CH or CSH matrix are first fed to MED where each element has been quantized with 64-bit such that 32 MSBs represent its real part and rest 32 least-significant-bits (LSBs) denote the imaginary part, referring line-1 in Algorithm 3. As shown in Fig. 6, ϕ control-signal reconfigures MED to convert input complex-matrix $M^{8\times8}$ to either real-symmetric $S_{\text{CH}}^{16\times16}$ or $S_{\text{CSH}}^{16\times16}$ matrices, referring lines 2–6 in Algorithm 3. Further, $S_{\text{CH/CSH}}^{16\times16}$ matrix is equated to $A^{16\times16}$ matrix whose upper-triangular elements are fed to ECR, as illustrated in Fig. 6. Here, ECR processes these matrix-elements based on operation that is presented in Section III-B to generate eigenvalues and eigenvectors of CH (i.e., S_{CH}) or CSH (i.e., S_{CSH}) matrix, as illustrated by lines 9–10 of Algorithm 3. These eigenvalues are generated in the repeated format as $\Lambda = \textbf{diag}\{\lambda_1, \lambda_1, \lambda_2, \lambda_2, \ldots, \lambda_8, \lambda_8\}$, without losing the precision. In addition, aforementioned design can also be reconfigured for the RSS matrix by assigning a zero matrix m^i (i.e., $m^i = O^{n\times n}$) for the input $M = [m^r + i \cdot m^i] \in \mathcal{C}^{n\times n}$ matrix.

The proposed micro-architecture of MED is also shown in Fig. 6 which is a multiplexer network that transforms complex $M^{8\times8}$ matrix into real-symmetric upper-triangular $S_{\text{CH/CSH}}^{16\times16}$ matrix. Considering $M^{8\times8}$ matrix of 64 elements where each complex element is $m_{ij} = m_{ij}^r + i \cdot m_{ij}^i$ such that m_{ij}^r and m_{ij}^i are real and imaginary parts, respectively. Suggested MED architecture transforms these 64 complex elements (each of 64 bit) into 136 upper-triangular real-elements of $S^{16\times16}$ matrix which is equivalent to upper-triangular real-symmetric $A^{16\times16}$ matrix, as illustrated by line-7 in Algorithm 3. Based on ϕ value, such transformation has been performed with two patterns for CH and CSH matrices. In the proposed MED architecture, row-1 elements of $S_{\text{CSH}}^{16\times16}$ and $S_{\text{CH}}^{16\times16}$ matrices are multiplexed via series of multiplexers in the first layer, as shown in Fig. 6. It also shows that the inputs of all 2:1 multiplexers are connected with the elements from respective rows of $S_{\text{CSH}}^{16\times16}$ and $S_{\text{CH}}^{16\times16}$ matrices upto eight rows. Remaining elements from last eight rows of both S_{CH} and S_{CSH} matrices are the replicated versions of their upper-triangular elements. Therefore, these elements in MED architecture are obtained by tapping the outputs of selected 2:1 multiplexers, as presented in Fig. 6. Consecutively, all elements of the upper triangular $A^{16\times16}$ matrix (i.e., equivalent to $S^{16\times16}$ matrix) from MED are fed to ECR that computes eigenvalues (Λ) and eigenvectors (W) of the $M^{8\times8}$ matrix, referring Fig. 6. Generally, to connect all the rows of S_{CH} and S_{CSH} matrices, the proposed reconfigurable architecture requires $n(3n+1)/2$ number of 2:1-sized multiplexers.

V. Performance Analyses, Hardware Implementation, Comparisons, and Complexity Analyses

A. *Performance Analyses*

To evaluate the performance of proposed algorithm based on parallel Jacobi method and its corresponding architecture, there are three configuration parameters that must be analyzed for numerical precision: number of sweeps N_s, bit width N, and CORDIC-period T_c. To begin with, an experimental error analysis that determines the N_s value to obtain adequate results (i.e., magnitudes of eigenvalues) from the proposed architecture has been carried out. Here, error percentage is computed between the $Q_{2.30}$ fixed-point eigenvalues ($\hat{\epsilon}_i$) from FPGA implementation of the proposed architecture and floating-point eigenvalues (ϵ_i) obtain from the extensive simulations (i.e., using the standard *eig* function in MATLAB tool). Each error percentage ($\mathbb{E}$) for single N_s value has been obtained for multiple 16×16 real-symmetric random matrices as

$$\mathbb{E} = \left(\frac{1}{n} \sum_{i=1}^{n} \left| \frac{\hat{\epsilon}_i - \epsilon_i}{\epsilon_i} \right| \right) \times 100\%. \tag{15}$$

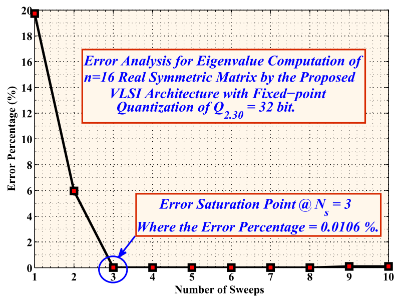

Fig. 7. Error analysis between the eigenvalues computed by proposed architecture and conventional simulation of real-symmetric 16×16 matrix with respect to different sweep values.

Since the error percentages of eigenvalues and eigenvectors are equivalent, this work presents error analysis of the former only, as illustrated in Fig. 7. It shows that the error percentage of eigenvalues from $n = 16$ real-symmetric matrices alleviates with the increasing value of N_s and the error eventually saturates beyond $N_s = 3$. Hence, this value of sweep delivers adequately accurate magnitudes of eigenvalues and eigenvectors from our design. Similar error analyses for other three input matrices: CH, CSH, and RSS matrices, for the size of $n = 8$ have been carried out where $N_s = 3$ delivered sufficiently accurate results.

As discussed earlier in lines 23–24 and line-25 of Algorithm 1 from Section III-A, double rotation for eigenvalues computation and single rotation for eigenvectors calculation, respectively, are necessary operations. In the proposed architecture, these information are calculated by EECB (using U-MAC and LO-MAC units), as shown in Figs. 3 and 4, respectively. Primary operations in these MAC units are multiplication and addition. In every kth rotation, such multiplication doubles the bit width of its output. Thereby, truncation and rounding of the LSBs of such multiplication outputs are necessary to alleviate the proliferation of hardware required by these multipliers. However, these operations degrade the accuracy of results which is directly proportional to the N bit-width value. Hence, it is important to analyze the error percentages with increasing N values of the input elements for various matrices supported by the proposed architecture for an adequate number of sweeps $N_s = 3$, as presented in Fig. 7. Hence, this article presents average truncation and rounding errors of the proposed VLSI architecture for computing the eigenvalues of real-symmetric, CH, CSH, and RSS matrices when the bit-width N ranges from 16 to 32 bit, as shown in Fig. 8. Aforementioned average error $= (1/\phi)\sum_{j=1}^{\phi} \mathbb{E}_j$ for

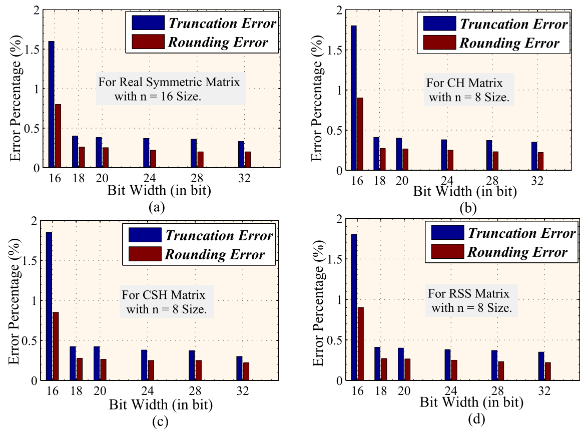

Fig. 8. Average truncation and rounding errors for computing eigenvalues of (a) real-symmetric, (b) CH, (c) CSH, and (d) RSS matrices for varying bit widths of input matrix elements and $\phi = 15$.

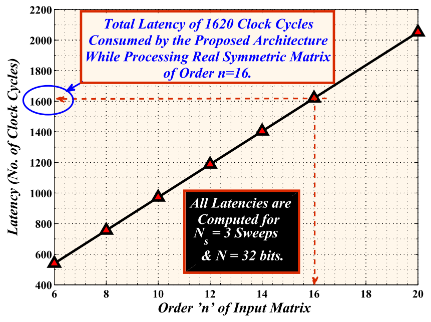

Fig. 9. Analysis of various latencies of the proposed architecture for different input matrix sizes while computing eigenvalues of real-symmetric matrix.

ϕ different real-symmetric matrices and $\mathbb{E}$ is presented in (15). Such analyses show that average rounding and truncation errors diminish sharply when the N value is increased from 16- to 20-bit. These errors remain constant beyond $N = 20$ bit, as shown in Fig. 8. Therefore, for better accuracy and fair comparison with the state-of-the-art implementation [18], $N = 32$ bit has been considered for the design of the proposed architecture. It is to be noted that the error results shown in Fig. 8 are obtained for $n = 16$ size of real-symmetric matrix and $n = 8$ size for other CH, CSH, and RSS matrices.

On the other side, latency ($\mathcal{L}$) of the proposed design is proportional to the order ($n \times n$) of input matrices, referring (14). Therefore, analysis of $\mathcal{L}$ values (in number of clock cycles) for different n values of real-symmetric input matrix for $N_s = 3$ sweeps and $N = 32$ bit is shown in Fig. 9. It illustrates that the proposed design consumes $\mathcal{L} = 1620$ clock cycles while processing the real-symmetric matrix of order $n = 16$. Since the MED micro-architecture is purely combinational design, it does not consume any extra clock cycle. Therefore, the proposed reconfigurable architecture also requires 1620 clock cycles to compute eigenvalues and eigenvectors of CH, CSH, and RSS matrices of the order $n = 8$.

B. FPGA Implementations and ASIC Designs

In this article, the proposed stand-alone and reconfigurable VLSI-architectures for computing eigenvalues and eigenvectors of real-symmetric, CH, CSH, and RSS matrices have been simulated as well as synthesized using Xilinx Vivado-2017.1 design-suite. For the real-symmetric matrix, the suggested Algorithm-1 has been simulated using MATLAB-2018a simulation platform on a host system. Subsequently, the proposed architectures are RTL-coded and their gate-level netlists are placed-and-routed and hardware-implemented on Virtex-7 evaluation FPGA-board. As a result, Table II lists the resource utilization of our designs where the input data-bus is 32-bit fixed-point represented in $Q_{2.30}$ format. Thus, all input data must be scaled between -1 and 1 to alleviate quantization error.

Critical (longest) path of the proposed architecture lies in the U-MAC unit, highlighted in Fig. 4(a), such that its combinational delay is given as $\partial_{\text{UMAC}} = (\partial_{2:1\text{MUXV2}} + \partial_{4:1\text{MUX1}} + \partial_{\times} + \partial_{+})$ where $\partial_{2:1\text{MUXV2}}$, $\partial_{4:1\text{MUX1}}$, $\partial_{\times}$ and ∂_{+} represent the delays of 2:1 multiplexer, 4:1 multiplexer, multiplier, and adder, respectively. Hence, expression for the critical path delay of our architecture is

$$\partial_{\text{crit}} = t_{\text{cq,REGY}} + \partial_{\text{UMAC}} + t_{\text{su,REG1}} \tag{16}$$

where $t_{\text{cq,REGY}}$ and $t_{\text{su,REG1}}$ denote clock-to-Q delay of REG-Y register and setup-time of REG-1 register, respectively, in the U-MAC unit from Fig. 4(a). Therefore, static timing analysis of the proposed design for processing input-matrix of the order $n = 8$ indicates that it operates at a maximum clock frequency $f_{\max} = (1/\partial_{\text{crit}})$ of 172.75 MHz. It is to be noted that one-rotation (T_{ort}) and total-converge (T_{tot}) times (in second) for the proposed architectures are expressed as

$$T_{\text{ort}} = (T_c + 4) \times \frac{1}{f_{\max}} \;\; \& \;\; T_{\text{tot}} = T_{\text{ort}} \times (n-1) \times N_s \tag{17}$$

respectively. As discussed earlier, T_c is the total number of clock cycles consumed by one CorRB architecture and the number of CORDIC iterations performed by such CorRB is equivalent to the bit-width N of input matrix elements. Therefore, T_c value equals 32 clock cycles in our design where the input has a bit-width N of 32 bit. Here, $(n-1)$ is the number of rotations in each sweep and N_s is the total number of sweeps. An optimum N_s value is 3 for the matrix size of $n = 16$, based on the analysis presented in Fig. 9. Using aforementioned design parameters, the proposed architecture delivers T_{ort} of 0.208 μs and T_{tot} of 9.377 μs for computing eigenvalues and eigenvectors of real-symmetric input matrix.

For the other three matrices: CH, CSH, and RSS, suggested reconfigurable architecture presented in Fig. 6 has been hardware implemented on same FPGA platform and its implementation results are presented in Table II. This design also possesses the same critical path delay, as expressed in (16), and operates at the maximum clock frequency of 172.75 MHz. On the other hand, each complex-valued input matrix function has been represented as a 64-bit value (32 bit each for real and imaginary parts). As shown in Fig. 6, the MED architecture converts 64-bit input elements into 32-bit values, as well as the matrix range n to $2n$. As a result, n and T_c are replaced by $2n$ and $T_c = N/2$, respectively, in (17). Thus, T_c equals 64/2 clock cycles and T_{ort} is 0.208 μs. The suggested reconfigurable architecture processes $n = 8$ sized input matrices and therefore, $T_{\text{tot}} = 9.377$ μs for $n = 2 \times 8$ size. On the other hand, both the proposed stand-alone and reconfigurable architectures that compute eigenvalues and eigenvector of real-symmetric and CH/CSH/RSS matrices, respectively, are ASIC synthesized as well as post-layout simulated in UMC 90-nm CMOS process using the standard electronic-design-automation (EDA) tools. Thus, the implementation results obtained from this ASIC-design process are presented in

TABLE II
COMPARISON OF PROPOSED ARCHITECTURE WITH PRIOR REPORTED IMPLEMENTATIONS

	This work			[22]	[19]	[18]	[24]
FPGA Board	**Virtex-7**	**Virtex-7**		Virtex-7	Altera	Virtex-II pro	Altera
Matrix Type	**CH, CSH, & RSS**	**Real Symm.**		Real Symm.	Real Symm.	Real Symm.	CH
Architecture Type	**Reconfigurable**	**Stand-alone**		Stand-alone	Stand-alone	Stand-alone	Stand-alone
Matrix Size (n)	**8**	**8**	**16**	8	8	16	10
Bit-Width (N)	**32 bit**	**32 bit**	**32 bit**	32 bit	16 bit	18 bit	18 bit
Clk Cycles for One Rotation (N_{orc})	T_c **+ 4**	T_c **+ 4**	T_c **+ 4**	$2 \cdot T_c$	$8\log_2 N + T_{sc}{}^{✠} + T_e{}^{♣}+36$	–	$5 \cdot T_c + 6$
Clk Cycles for Latency ($\mathcal{L}$)‡	$f(T_c$ **+ 4)**	$f(T_c$ **+ 4)**	$f(T_c$ **+ 4)**	$f(2 \cdot T_c)$	$f(8\log_2 N + T_{sc}{}^{✠} + T_e{}^{♣}+36)$	–	$f(5 \cdot T_c + 6)$
Error Percentage ($\mathbb{E}$)	**0.11%, 0.0251%, & 0.4986%**§	**0.0106 %**	**0.0106 %**	0.10%$^⊛$	0.78 %	0.432 %	0.11 %
Eigenvalue & Eigenvector	**Both**	**Both**	**Both**	Both	Only Eigenvalue	Both	Only Eigenvalue
No. of DSP Cores	**1026**	**-**	**1024**	–	–	–	768
No. of Slice LUTs	**134249**	**122127**	**127425**	258348†	-	-	21813
No. of Slice Registers	**53165**	**14418**	**50166**	113458†	-	-	8356
Area (No. of Logic Elements)	-	-	-	-	16314	-	-
No. of Slices	-	-	-	-	-	1800	-

1) ✠: T_{sc} is the latency of scale-factor (CORDIC has a gain of 1.646) correction module such that for $N<16$, $N=16-22$, $N=23-26$,$N=27$,$N=28-30$,$N=31-33$ is 8, 9, 10, 11, 12, 13, respectively. ♣: T_e represents the time for matrices interchange among off-diagonal processors and is equal to one if the matrix size is 4 else it is equal to two, as reported [20].
2) ‡: Total latency $\mathcal{L} = f(N_{orc})$ clock cycles where the function $f(\cdot)=\sum_{s=1}^{N_s}\left[\sum_{k=1}^{n-1}(\cdot)_k\right]$.
3) ⊛: This is the DOA estimation root-mean-square error for an array of eight sensors.
4) †: Area utilization reported with the DOA estimation (correlation matrix + eigenvalue computation + MUSIC algorithm).
5) §: Error percentages incurred while processing CH, CSH and RSS matrices are 0.11%, 0.0251% and 0.4986%, respectively.

TABLE III
ASIC SYNTHESIS AND POST-LAYOUT SIMULATION RESULTS OF THE PROPOSED ARCHITECTURES

Design Metrics	Stand-alone Arch.	Reconfigurable Arch.
CMOS Technology	90 nm	90 nm
Supply Voltage	0.9 V	0.9 V
Matrix Size (n)	16	8
Bit-Width (N)	32 bit	32 bit
Core Area	10.051 mm^2	10.053 mm^2
Standard Cell Count	10051063 Cells	10053402 Cells
Max. Clock Frequency	100 MHz	100 MHz
Latency	16 μs	16 μs

Table III where our designs are capable of delivering a latency of 16 μs, while operating at a maximum clock frequency of 100 MHz with the supply voltage of 0.9 V.

C. Hardware Prototyping and Functional Validation

This work presents hardware prototyping of the proposed ECR architecture in Zynq Ultrascale+ ZCU102 evaluation FPGA board (xczu9eg-ffvb1156-2-i). It computes eigenvalues of the real-symmetric ($A_{16\times16}$) matrix of order $n = 16$. Real-world snapshot and schematic representation of the test setup used in this work have been illustrated in Fig. 10(a) and (b), respectively. Here, 136 upper-triangular elements (each of 32-bit) of $A_{16\times16}$ matrix are the test vectors, generated by MATLAB environment in the host computer, and they are first stored in block random access-memory (BRAM) of FPGA via universal serial bus (USB). These stored test-vectors of 136 $\times$ 32-bit BRAM are transferred to the ECR FPGA-core via on-board parallel buses that processes

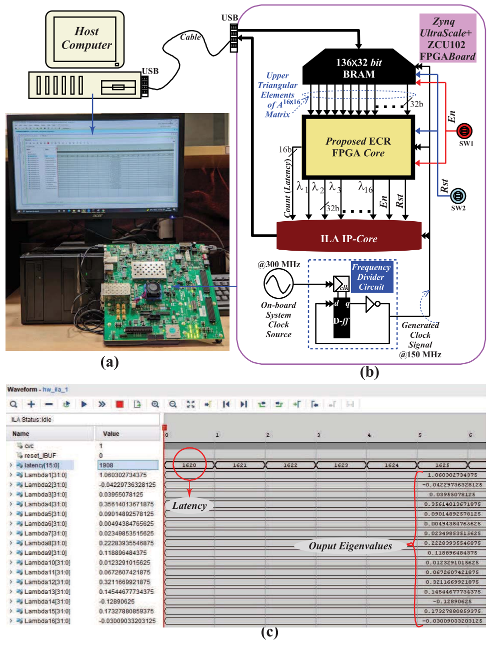
Host Computer
USB
Cable
USB
Zynq UltraScale+ ZCU102 FPGA Board
136X32 bit BRAM
Upper Triangular Elements of $A_{16\times16}$ Matrix
32b
16b
Proposed ECR FPGA Core
En
SW1
Rst
SW2
Count (Latency)
λ_1 λ_2 λ_3 λ_{16}
32b
ILA IP-Core
@300 MHz
On-board System Clock Source
Frequency Divider Circuit
D-ff
Generated Clock Signal @150 MHz
(a)
(b)
Waveform
Latency
Ouput Eigenvalues
(c)

Fig. 10. (a) Real-world and (b) detail schematic of the test setup for the validation of proposed hardware prototype. (c) Snapshot of the output waveform containing the measured output eigenvalues and latency count of our design.

them to generate the eigenvalues of $A_{16\times16}$ matrix. Consecutively, using the Vivado in-built integrated-logic-analyzer (ILA), these output eigenvalues and the latency $\mathcal{L}$ (i.e., count output from on-board implemented-counter) are transferred

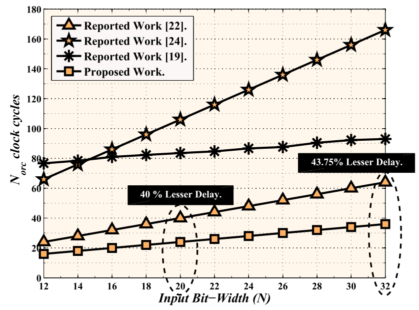

Fig. 11. Comparative analysis of latency (N_{orc}) with respect to scaling bit-width (N) values of proposed implementation with the reported ones.

to host computer via USB to be displayed on its screen, as shown in Fig. 10. Note that the system frequency of Zynq Ultrascale+ ZCU102-FPGA board is 300 MHz and our hardware prototype can be operated up to 172.25 MHz of maximum clock frequency for the validation. Hence, we employed a frequency divider on FPGA-board that divides the frequency of system clock by two to generate an operating clock signal of 150 MHz that is routed to ILA core, BRAM, and the proposed ECR FPGA-core, as shown in Fig. 10(b). Two of the on-board FPGA switches (SW1 and SW2) are used for generating enable (*En*) and reset (*Rst*) signals for the proposed design. Here, *En* switch initiates the process of transferring the BRAM data (i.e., test-vectors) to the ECR FPGA-core and simultaneously starts the counter that tracks $\mathcal{L}$ value of the hardware prototype. Eventually, the snapshot of ILA screen (from host computer) that shows the output waveform of hardware-generated eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_{16}$ has been presented in Fig. 10(c). In comparison with the simulated values, the eigenvalues of our hardware prototype incurs an error of 0.0106%. Additionally, Fig. 10(c) presents the latency value of 1620 clock cycles via 16-bit count value. Finally, aforementioned process clearly illustrates the functional validation of proposed design in FPGA platform with minimal error, and subsequently demonstrates its lower latency requirement.

D. Comparisons and Discussion

Implementation results of the proposed stand-alone and reconfigurable VLSI-architectures are compared with the reported works from literature, as presented in Table II. It shows that the proposed design for a real-symmetric matrix consumes the least number of clock cycles for one rotation (N_{orc}) and latency ($\mathcal{L}$), compared to the state-of-the-art implementations from [15], [18]. For $n = 8$ sized input-matrix and the input bit-width $N = 32$ bit, suggested architectures are capable of computing both eigenvalues and eigenvectors. However, both (i.e., eigenvalues and eigenvectors) and only eigenvalues are computed for input-matrix of order $n = 8$ in [18] (for $N = 32$ bit) and [15] (for $N = 16$ bit), respectively. Furthermore, both eigenvalues and eigenvectors are computed in [14] for $n = 16$ sized input-matrix. In addition, error percentage of our design is 89.4% better than the contemporary implementation of [18] (both are 32-bit quantized designs), and lesser than the errors incurred by other related works [14], [15], [20], as shown in Table II.

On the other hand, we proposed a reconfigurable architecture for processing CH, CSH, RSS input-matrices, as shown in Fig. 6. In this design, N_{orc} has been alleviated by 4× compared to the reported implementation by Parrado *et al.* [20] which uses complex arithmetic equations to calculate eigenvalues using only the CH for $n = 10$ sized matrix-dimension. Unlike, we use only real arithmetic calculations to generate eigenvalues and eigenvectors of CH, CSH, and RSS matrices which is less complex in terms of hardware. However, area utilizations of the proposed architectures for $n = 8$ sized real-symmetric matrix and $n = 8$ sized CH, CSH, and RSS matrices are moderate compared to reported works. Additionally, we present the analysis of N_{orc} values of proposed and reported implementations for increasing N values, as shown in Fig. 11. It clearly shows that our design is scalable with respect to delay and the proposed architecture delivers 43.75% lower delay for $N = 32$ bit input matrix elements compared to the state-of-the-art implementation [18]. In nutshell, the suggested architecture is reconfigurable for computing eigenvalues and eigenvectors of real-symmetric, CH, CSH, and RSS matrices, while incurring minimal error and consuming lowest latency that makes our design suitable for high-speed applications.

Apart from the aforementioned CORDIC-based implementations, there are other eigenvalues and eigenvector computation methods based on Jacobi method, without using the CORDIC algorithm, like the approximation Jacobi method (AJM) [25], [26] and the algebraic Jacobi method (ALJM) [27]. Here, the AJM has almost linear convergence and its computations are performed using the shift-and-add operations [25], [26]. Unlike, ALJM with almost quadratic convergence uses simplification, polynomial approximation, and Newton–Raphson method to perform the Jacobi rotation for the computation of eigenvalues [27]. Our article presents the comparison of the proposed stand-alone architecture (that computes eigenvalues and eigenvector of real-symmetric matrix) with the reported architectures based on AJM and ALJM from [26] and [27], respectively, in Table IV. It shows that the suggested design consumes lesser number of clock cycles (i.e., latency $\mathcal{L}$) in comparison with the reported implementations from [26], [27]. In addition, our architecture is capable of calculating both eigenvalues and eigenvectors. Similarly, [27] computes both these values; however, [26] calculates only the eigenvalues. With the same bit quantization, proposed design delivers better error percentage than both the reported works [27] and [26], as shown in Table IV. Compared to the state-of-the-art work [27], the proposed design has acceptable and moderate hardware consumption. Note that our design is memoryless whereas the reported work of [27] requires 9 kB of BRAM.

E. Hardware-Complexity Analyses

For computing eigenvalues and/or eigenvectors for the input matrix of $n \times n$ order, this article presents the hardware complexity analysis of the proposed architectures with the conventional parallel Jacobi method [13] as well as other reported works in Table V. In the proposed Algorithm 1, there are three primary parts: step-1 calculates R_k matrix {lines 11−22}, step-2 performs a double rotation {lines 23 and 24}, and step-3 computes eigenvalues and eigenvectors {lines 28 and 29}. Hardware complexity of the conventional parallel Jacobi method [13] is estimated by assuming that the R_k matrix is obtained using adders, subtractors, CORDIC processors and read-only-memories (ROMs), as shown in Table V. Subsequently, its double rotation and eigenvector calculation are performed using multipliers, adders, and subtractors. Table V shows that the conventional parallel Jacobi method has total computational hardware complexity of $8 \cdot n^2 + 2 \cdot n$ with memory

TABLE V
HARDWARE COMPLEXITY COMPARISON OF PROPOSED VLSI-ARCHITECTURES WITH REPORTED WORKS

	Conventional Parallel Jacobi Method [17]	This Work		[22]	[19]	[18]	[24]
Matrix Type	Real Symm.	Real Symm.	CH, CSH, RSS	Real Symm.	Real Symm.	Real Symm.	CH
Matrix Size (n)	even	even	even	even	even	even	even
Eig.val. & **Eig.vec.**	Both	Both	Both	Only Eigenvalue	Both	Both	Only Eigenvalue
Number of Multipliers	$5n^2$	n^2 {lines 23,24,25}	$4n^2$ {lines 23,24,25 }	$n/2$	$(3n^2 - n)/2$	$n(3n + 2)/4$	n^2
Number of Adders/ Subtractors	n + $3n^2$	$2n$ {line 11-22} + $(n^2 + n)$ {lines 23,24,25}	$4n$ { line 11-22 } + $(4n^2 + 2n)$ {lines 23,24,25}	n^2	$n/2$	$n/2$	$n/2$
Number of CORDIC$^{\spadesuit}$	n	n {line 11-22}	$2n$ {line 11-22}	$n(n + 2)/4$	$n(3n + 2)/8$	$3n(n^2 + 2)/8$	$n(n^2 + 2)/2$
Number of ROM	$n^{\maltese}$	– –	– –	$n(n - 2)/2^{\maltese}$	$n/2^{\maltese}$ $+n/2^{\clubsuit}$	$n^{\maltese}$ $+n/2^{\clubsuit}$	$(n^2 + 2)/2^{\maltese}$ $+n^2/4^{\diamond}$

1) $\spadesuit$: Here, each CORDIC is an iterative processor that consumes three adders and two N-bit shifters [23], [25].
2) ROM Sizes of $\maltese$: $N \times N$ bit, $\clubsuit$: $4 \times N$ bit, and $\diamond$: $8 \times N$ bit where N represents input bit-width.

TABLE IV
COMPARISON OF CORDIC-BASED PROPOSED ARCHITECTURE AND PRIOR REPORTED DESIGNS WITHOUT USING CORDIC

Design Metrics	**Proposed Method**	[30]-**AJM**	[31]-**ALJM**
Device series	Virtex- 5	Virtex- 5	Spartan-6
Matrix Size(n)	8×8	8×8	8×8
Matrix Type	Real Symm.	Real Symm.	Real Symm.
Bit-widths(N)	21 bit (18 bit$^{\maltese}$)	21 bit	18 bit
Error (%)	0.23 (0.26$^{\maltese}$)	0.55	<1
Eig.val. & Eig.vec.	Both	Only Eig.val.	Both
Convergence Rate	Logarithmic	Linear	Quadratic
Max. Sweeps(N_s)	3	5-10	5
Total Clock Cycles($\mathcal{L}$)	525 (462$^{\maltese}$)	–	840
Frequency(MHz)	80.456 (90.992$^{\maltese}$)	145.022	–
Latency(μs)	6.525 (5.077$^{\maltese}$)	500-600	–
No. of DSP Slices	– (64$^{\maltese}$)	–	16
Slice Utilization	– (11007$^{\maltese}$)	–	1056
No. of Slice LUTs	59197 (19425$^{\maltese}$)	26875	4180
No. of slice Registers	8534 (10935$^{\maltese}$)	40305	114
Block RAM(kB)	– (–$^{\maltese}$)	–	9

1) $\maltese$: Design metrics of the proposed design obtained from its implementation on Spartan-6 FPGA board.

requirement of n number of $N \times N$-bit ROMs where N is input bit width. To compute only eigenvalues in [18], its architecture comprises of two processing units: diagonal and off-diagonal processors. Here, R_k matrix and the diagonal elements of double rotation are determined using $n/2$ number of such diagonal processors. Hence, the hardware complexity is calculated using n subtractors, n adders, n CORDICs, and $n/2$ multipliers, as presented in Table V. In $n(n-2)/8$ number of off-diagonal processors, an off-diagonal element is calculated using $n(n-2)/2$ ROMs, $n(n-2)/2$ adders, $n(n-2)/2$ subtractors, $n(n-2)/4$ CORDICs, and $n(n-2)/2$ multipliers. Similarly, the designs presented in [15] and [14] compute eigenvalues and eigenvectors using diagonal and off-diagonal processors that consume CORDICs, adders/subtractors, multipliers and ROMs, as listed in Table V. It shows that the proposed architectures have lower hardware complexity than the conventional parallel Jacobi method [13] and higher computational hardware complexity than [14], [15], [18]. Nevertheless, our designs are memory less that further alleviates the hardware requirement. In general, an overall computational complexity of the proposed stand-alone architecture for computing eigenvalues and eigenvectors of n ordered input matrix has two parts: space and time complexities. Here, complete space complexity $\mathbb{S}(n) = \kappa(n) + \vartheta(n) + \omega(n)$ where $\kappa(n) = n^2$ represents multiplier complexity and $\vartheta(n) = n^2 + 5 \times n$ is the adder/subtractor complexity. Similarly, the multiplexer complexity is given by $\omega(n) = \Omega_{n:1}(n) + \Omega_{4:1}(n) + \Omega_{2:1}(n)$ such that $\Omega_{n:1}(n) = 5n(n+1)/2$, $\Omega_{4:1}(n) = 2n^2$, and $\Omega_{2:1}(n) = 10n^2 + n$ denote the space complexities of $(n-1)$:1, 4:1 and 2:1 multiplexers, respectively. Second, an overall time-complexity is given as $\mathbb{T}(n) = O\{\log_2 n \times (n-1) \times (N+4)\}$ where $N + 4$ is the number of clock cycles consumed for single rotation with N bit-width input matrix, and $\log_2 n$ is the number of sweeps such that each sweep comprises of $(n-1)$ rotations. On the other side, eigenvalues for the CH matrix are obtained using the complex arithmetic in [20]. Table V shows that the computational hardware complexity of [20] is lower than the proposed reconfigurable architecture. However, our reconfigurable architecture is capable of computing eigenvalues and eigenvector of CH, CSH, and RSS matrices using the real arithmetic, without the memory requirement. Therefore, an overall time complexity of the proposed reconfigurable architecture can be derived by replacing n with $2n$ in $\mathbb{T}(n)$ and is given by $\mathbb{T}_{\text{recon}}(n) = \mathbb{T}(n \rightarrow 2n) = O\{\log_2 2n \times (2n-1) \times (N+4)\}$. Eventually, the space complexity of our reconfigurable architecture is expressed as $\mathbb{S}_{\text{recon}}(n) = \mathbb{S}(n \rightarrow 2n) + n(3n+1)/2$ where $n(3n+1)/2$ represents the additional space complexity of 2:1 multiplexers used in MED architecture, as discussed earlier in Section IV-B.

VI. CONCLUSION

This work presented lower latency and reconfigurable algorithms for the computation of eigenvalues and eigenvectors of wide variety of matrices like real-symmetric, CH, CSH, and RSS matrices. Comprehensive explanations of stand-alone and reconfigurable VLSI architectures for such computations were presented in our article. Detail error analysis and implementation results showed that the proposed algorithms incurred minimal error and lowest latency. Thus, these designs are extremely useful for various real-time applications. Our article illustrated the complete process of transforming a complex algorithm for eigenvalues and eigenvectors computation into implementation-friendly (i.e., low-latency and reconfigurable) algorithms, transforming them into efficient VLSI architectures and their FPGA implementations.

References

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jun. 1948.

[2] S. Deshmukh and A. Dubey, "Improved covariance matrix estimation with an application in portfolio optimization," *IEEE Signal Process. Lett.*, vol. 27, pp. 985–989, 2020.

[3] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Mar. 1991, pp. 586–591.

[4] R.-L. Hsu, M. Abdel-Mottaleb, and A. K. Jain, "Face detection in color images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 696–706, May 2002.

[5] Y. Xie, C. Peng, X. Jiang, and S. Ouyang, "Hardware design and implementation of DOA estimation algorithms for spherical array antennas," in *Proc. IEEE Int. Conf. Signal Process., Commun. Comput. (ICSPCC)*, Aug. 2014, pp. 219–223.

[6] C. Zhou, Y. Gu, S. He, and Z. Shi, "A robust and efficient algorithm for coprime array adaptive beamforming," *IEEE Trans. Veh. Technol.*, vol. 67, no. 2, pp. 1099–1112, Feb. 2018.

[7] C. Zhou, Y. Gu, Z. Shi, and Y. D. Zhang, "Off-grid direction-of-arrival estimation using coprime array interpolation," *IEEE Signal Process. Lett.*, vol. 25, no. 11, pp. 1710–1714, Nov. 2018.

[8] K.-C. Huarng and C.-C. Yeh, "A unitary transformation method for angle-of-arrival estimation," *IEEE Trans. Signal Process.*, vol. 39, no. 4, pp. 975–977, Apr. 1991.

[9] C. F. van Loan and G. H. Golub, *Matrix Computations*. vol. 52. Baltimore, MD, USA: Johns Hopkins Univ. Press, 1983.

[10] J. Demmel and K. Veselić, "Jacobi's method is more accurate than QR," *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 4, pp. 1204–1245, 1992.

[11] S. Pal, S. Pathak, and S. Rajasekaran, "On speeding-up parallel Jacobi iterations for SVDs," in *Proc. IEEE 18th Int. Conf. High Perform. Comput. Commun.; IEEE 14th Int. Conf. Smart City; IEEE 2nd Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Dec. 2016, pp. 9–16.

[12] R. P. Brent and F. T. Luk, "The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays," *SIAM J. Sci. Stat. Comput.*, vol. 6, no. 1, pp. 69–84, 1985.

[13] A. H. Sameh, "On Jacobi and Jacobi-like algorithms for a parallel computer," *Math. Comput.*, vol. 25, no. 115, pp. 579–590, Jan. 1971.

[14] I. Bravo, P. Jimenez, M. Mazo, J. L. Lazaro, and A. Gardel, "Implementation in FPGAs of Jacobi method to solve the eigenvalue and eigenvector problem," in *Proc. Int. Conf. Field Program. Logic Appl.*, Aug. 2006, pp. 1–4.

[15] T. Wang and P. Wei, "Hardware efficient architectures of improved Jacobi method to solve the eigen problem," in *Proc. 2nd Int. Conf. Comput. Eng. Technol.*, vol. 6, 2010, pp. 22–25.

[16] A. Ahmedsaid, A. Amira, and A. Bouridane, "Improved SVD systolic array and implementation on FPGA," in *Proc. IEEE Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2003, pp. 35–42.

[17] S. Zhang, X. Tian, C. Xiong, J. Tian, and D. Ming, "Fast implementation for the singular value and eigenvalue decomposition based on FPGA," *Chin. J. Electron.*, vol. 26, no. 1, pp. 132–136, Jan. 2017.

[18] 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.

[19] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proc. ACM/SIGDA 6th Int. Symp. Field Program. Gate Arrays (FPGA)*, 1998, pp. 191–200.

[20] A. Lopez-Parrado and J. Velasco-Medina, "Efficient systolic architecture for Hermitian eigenvalue problem," in *Proc. IEEE 4th Colombian Workshop Circuits Syst. (CWCAS)*, Nov. 2012, pp. 1–6.

[21] G. L. Haviland and A. A. Tuszynski, "A CORDIC arithmetic processor chip," *IEEE J. Solid-State Circuits*, vol. 15, no. 1, pp. 4–15, Feb. 1980.

[22] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*. vol. 87. Oxford, U.K.: Clarendon Press, 1965.

[23] C. F. van Loan and H. G. Golub, *Matrix Computations*. Baltimore, MD, USA: Johns Hopkins Univ. Press, 1983.

[24] K. Kuang-Chi Lee and C.-E. Chen, "An eigen-based approach for enhancing matrix inversion approximation in massive MIMO systems," *IEEE Trans. Veh. Technol.*, vol. 66, no. 6, pp. 5480–5484, Jun. 2017.

[25] 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.

[26] A. Ibrahim, M. Valle, L. Noli, and H. Chible, "Assessment of FPGA implementations of one sided Jacobi algorithm for singular value decomposition," in *Proc. IEEE Comput. Soc. Annu. Symp. (VLSI)*, Jul. 2015, pp. 56–61.

[27] M. Sajjad, M. Z. Yusoff, N. Yahya, and A. S. Haider, "An efficient VLSI architecture for FastICA by using the algebraic Jacobi method for EVD," *IEEE Access*, vol. 9, pp. 58287–58305, 2021.

Rahul Sharma (Graduate Student Member, IEEE) received the Bachelor of Technology degree in electronics and communication engineering from Rajasthan Technical University, Kota, India, in 2017. He is currently working toward the M.S. degree (by research) at the School of Computing and Electrical Engineering, IIT Mandi, Mandi, India.

His research interest focuses on developing efficient VLSI architectures and implementation-friendly algorithms for the spectrum sensing process of cognitive radio technology.

Rahul Shrestha (Senior Member, IEEE) received the Bachelor of Engineering degree in telecommunication engineering from the B. M. S. College of Engineering, Bengaluru, India, in 2008, and the Ph.D. degree in electronics and electrical engineering from IIT Guwahati, Guwahati, India, in 2014.

From 2014 to 2016, he worked as an Assistant Professor with the Center for VLSI and Embedded Systems Technologies, International Institute of Information Technology Hyderabad, Hyderabad, India. Since 2016, he has been holding the position of an Assistant Professor with the School of Computing and Electrical Engineering, IIT Mandi, Mandi, India. In 2018, he served as a Visiting Assistant Professor with Blekinge Institute of Technology, Karlskrona, Sweden, under the European Erasmus+ International Credit Mobility Program. His primary research interest is designing efficient algorithms as well as digital VLSI architectures and its transformation into ASIC-chip or field-programmable gate array (FPGA)-prototype for the real-world applications of signal processing, wireless communication, deep neural networks, forward-error-correction channel decoders, cognitive radio, and cooperative spectrum sensing.

Satinder K. Sharma (Senior Member, IEEE) received the M.S. degree in physics (electronic science) from Himachal Pradesh University, Shimla, India, in 2002, and the Ph.D. degree from the Department of Electronic Science, Kurukshetra University, Kurukshetra, India, in 2007.

From 2007 to 2010, he was a Postdoctoral Fellow with the DST Unit on Nanoscience and Nanotechnology, Department of chemistry (CHE), IIT Kanpur, Kanpur, India. From 2010 to 2012, he worked as a Faculty with the Electronics and Microelectronics Division, Indian Institute of Information Technology, Allahabad, India. Since 2012, he has been working as a Faculty with the School of Computing and Electrical Engineering, IIT Mandi, Mandi, India. In 2015, he worked as a Visiting Faculty with the Institute of Semiconductor Engineering, University of Stuttgart, Stuttgart, Germany. He has published more than 92 publications in the international peer-reviewed journals, and presented several invited talks and research papers at more than 65 international and national conferences. He holds eight submitted patents. His current research interests include microelectronics circuits and systems, CMOS memories and 2-D-FETs fabrication and characterization, MEMS/nanoelectromechanical system (NEMS), sensors, and next generation lithography.