Design strategies for parallel iterative algorithms are presented. In order to further study different tradeoff strategies in design criteria for integrated circuits, a $10 \times 10$ Jacobi Brent-Luk-EVD array with the simplified $\mu$-CORDIC processor is used as an example. The experimental results show that using the $\mu$-CORDIC processor is beneficial for the design criteria as it yields a smaller area, faster overall computation time, and less energy consumption than the regular CORDIC processor. It is worth to notice that the proposed parallel EVD method can be applied to real-time and low-power array signal processing algorithms performing beamforming or DOA estimation.

1. Introduction

We are on the edge of many important developments which will require parallel data and information processing. The transmission systems are using higher and higher frequencies and the carrier frequencies are increasing to 10 GHz and above. Because of the smaller wavelength more antennas can be implemented on a single device leading to massive MIMO systems. Parallel VLSI architectures will be needed in order to provide the required computational power for 10 GHz and above, massive MIMO, and big data processing [1, 2].

In parallel matrix computation at the circuit level, implementing an iterative algorithm on a multiprocessor array results in a tradeoff between the complexity of an iteration step and the number of required iteration steps. Therefore, as long as the algorithm's convergence properties are guaranteed, it is possible to adjust the architecture, which can significantly reduce the complexity with regard to the implementation. Computing the parallel eigenvalue decomposition (EVD) as a preprocessing step to MUSIC or ESPRIT algorithm with Jacob's iterative method is used as an important example as the convergence of this method is extremely robust to modifications of the processor elements [3–6].

In [7], it was shown that Brent-Luk-EVD architecture with a modified CORDIC for performing the plane rotation of the Jacobi algorithm can be realized in advanced VLSI design. Based on it, a Jacobi EVD array is realized by implementing a scaling-free microrotation CORDIC ($\mu$-CORDIC) processor in this paper, which only performs a predefined number of CORDIC iterations. Therefore, the size of the processor array can be reduced for implementing a large-scale EVD array in parallel VLSI architectures. After that, several modifications of the algorithm/processor are studied and their impact on the design criteria is investigated for different sizes of EVD array ($10 \times 10$ to $80 \times 80$). Finally, a strategy to comply with the design criteria is established, especially in terms of balancing the number of microiterations and the computational complexity. The proposed architecture is ideal for real-time antenna array applications, such as a flying object carrying an antenna array for beamforming or DOA estimation that would require a real-time, low-power, and efficient architecture for EVD, or joint time-delay and frequency estimation using a sensor network.

This paper is organized as follows. Serial and parallel Jacobi methods are described in Section 2. In Section 3, the design issues of the parallel Jacobi EVD array are discussed, leading to the simplification from a regular full CORDIC to the $\mu$-CORDIC processor with an adaptive number of iterations. Section 4 shows the implementation results. Section 5 concludes this paper.
2. Parallel Eigenvalue Decomposition

2.1. Jacobi Method. An eigenvalue decomposition of a real symmetric matrix $A$ is obtained by factorizing $A$ into three matrices $A = Q \Lambda Q^T$, where $Q$ is an orthogonal matrix $(QQ^T = I)$ and $\Lambda$ is a diagonal matrix containing the eigenvalues of $A$. The Jacobi method approximates the EVD iteratively as follows:

$$A_{k+1} = Q_k A_k Q_k^T, \quad \text{with } k = 0, 1, 2, \ldots,$$

where $Q_k$ is an orthonormal plane rotation by the angle $\theta$ in the $(i, j)$ plane.

The plane rotations $Q_k$, where $k = 1, 2, 3, \ldots$, can be executed in various orders to obtain the eigenvalues. The most common order of sequential plane rotations $\{Q_k\}$ is called cyclic-by-row, meaning $(i, j)$ is chosen as follows:

$$(i, j) = (1, 2) (1, 3) \cdots (1, n) (2, 3) \cdots (2, n) \cdots (n-1, n).$$

The execution of all $N = n(n-1)/2$ index pairs $(i, j)$ is called a sweep. Matrix $A$ will converge into a diagonal matrix $\Lambda$ once $k$ sweeps are applied, where $\Lambda$ contains the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$:

$$\lim_{k \to \infty} A_k = \text{diag}[\lambda_1, \lambda_2, \ldots, \lambda_n] = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}. $$

2.2. Jacobi EVD Array. Instead of performing the plane rotations $Q_k$ one by one in a cyclic-by-row order, they can be separated into multiple subproblems and executed in parallel on a log $n$ dimensional multicore platform. Ahmed et al. [3] first presented a parallel array based on Jacobi’s method. It consists of $n/2 \times n/2$ PEs and each PE contains a $2 \times 2$ subblock of the matrix $A$. Figure 1 shows a typical $4 \times 4$ EVD array with 16 PEs. This Jacobi array can perform $n/2$ subproblems in parallel. Initially, each PE holds a $2 \times 2$ submatrix of $A$:

$$PE_{pq} = \begin{bmatrix} a_{2p-1,2q-1} & a_{2p-1,2q} \\ a_{2p,2q-1} & a_{2p,2q} \end{bmatrix},$$

where $p$ and $q = 1, 2, \ldots, n/2$.

A rotation angle has to be chosen in order to zero out the off-diagonal elements of the submatrix by solving a $2 \times 2$ symmetric EVD subproblem as shown in the following:

$$\begin{bmatrix} a'_{ij} & a'_{ji} \\ a'_{ji} & a'_{ij} \end{bmatrix} = R \begin{bmatrix} a_{ij} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix} R^T,$$

where $R = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$.

The maximal reduction $a'_{ij}a'_{ji} = 0$ can be obtained by applying the optimal angle of rotation $\theta_{opt}$:

$$\theta_{opt} = \frac{1}{2} \arctan \left( \frac{2a_{ij}}{a_{ij} - a_{ji}} \right).$$

3. CORDIC Approach

3.1. Regular CORDIC. Within each PE, a simple way to solve the subproblem of (5) in VLSI for zeroing out the off-diagonal elements is to use the CORDIC algorithm. An orthogonal CORDIC rotator is defined as [8, 9]

$$x_{i+1} = A_i \left[ x_i - y_i \cdot d_i \cdot 2^{-i} \right]$$

$$y_{i+1} = A_i \left[ y_i + x_i \cdot d_i \cdot 2^{-i} \right]$$

$$z_{i+1} = z_i - d_i \cdot \tan^{-1} 2^{-i}$$

$$A_i = \sqrt{1 + 2^{-2i}} \quad i = 1, 2, 3, \ldots, n$$

when $n \to \infty$, $A_n \approx 1.647$.

In the Cartesian coordinate system, the CORDIC orthogonal rotation mode can be used to compute (5) by separating
the two-sided rotation into two parts, \( G = [G_1^T; G_2^T] = A \cdot R^T \) and \( R \cdot A \cdot R^T \) that is computed by

\[
G_1 = \begin{bmatrix} a_{2i} \end{bmatrix}^T = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} a_{ij} \end{bmatrix}^T
\]

\[
G_2 = \begin{bmatrix} a_{2j} \end{bmatrix}^T = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} a_{ji} \end{bmatrix}^T,
\]

(8)

where the plane rotation with the desired rotation angle \( \theta_{\text{opt}} \) is executed using two CORDIC rotators. The CORDIC processors apply \( n \) steps, usually \( n = 32 \) for single floating precision. A constant scaling value \( K = 1/A_n = 0.6073 \) is subsequently required to fix the rotated vectors \( G_1 = [a_{ij}'; a_{ji}']^T \) and \( G_2 = [a_{ij}', a_{ji}']^T \) in order to retain the orthonormality. Similarly, these two CORDIC rotators can also be applied to compute \( R \cdot G \):

\[
[a_{ij}', a_{ji}']^T = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} a_{ij} \end{bmatrix}^T
\]

\[
[a_{ji}', a_{ij}']^T = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} a_{ji} \end{bmatrix}^T.
\]

(9)

Meanwhile, the angle \( \theta_{\text{opt}} \) can also be determined by using the CORDIC orthogonal vector mode. The CORDIC rotates the input vector through whatever angle is necessary to align the resulting vector with the \( x \)-axis:

\[
x_n = A_n \sqrt{x_0^2 + y_0^2} \\
y_n = 0 \\
z_n = z_0 - d_i \cdot \tan^{-1} 2^{-i}.
\]

(10)

The CORDIC with an orthogonal vector mode can compute the arctangent result iteratively \( \theta = \arctan(y/x) \), if the angle accumulator is initialized with zero \( (z_0 = 0) \).

In the VLSI design, two common approaches can be used to realize the CORDIC dependence flow graph in hardware: the folded (serial) or the parallel (pipelining) [10,11]. Note that we limit our efforts to the conventional CORDIC iteration scheme, as given in (7). In Figure 2(a), the structure of a folded CORDIC PE is shown, which requires a pair of adders for plane rotation and another adder for steering the next angle direction (computing the following \( z \) and \( d_i \)). All internal variables are buffered in the registers separately until the iteration number is large enough to obtain the result. The signs of all three intermediate variables are fed into a control unit that generates the rotation direction flags \( d_i \) to steer the add or suboperations and keep track of the rotation angle \( z \). For example, off-diagonal PE can directly apply the flags \( d_i \) from PE to \( G_1 \) and PE to \( G_2 \). After the rotation, the required scaling procedure can be obtained using the part of Figure 2(b) that fixes \( A_n \), where two multiplexers are required to select the inputs into the barrel shifterers. This folded dependence graph is typical for

3.2. Simplified \( \mu \)-Rotation CORDIC. As the process technologies continue to shrink, it becomes possible to directly implement a Brent-Luk-EVD array with the Jacobi method [12,13]. However, the size of the EVD array that can be implemented on the current configurable device with the regular CORDIC is still small, say, \( 4 \times 4 \). Therefore, we must simplify the architecture in order to integrate more processors. A scaling-free \( \mu \)-CORDIC for performing the plane rotation in (5) is used [5,6], where the number of inner iterations is reduced from 32 iterations to only one iteration.

The definition of \( \mu \)-CORDIC can be developed from (7) as

\[
x_{i+1} = \tilde{m} [x_i - y_i \cdot d_i \cdot 2^{-i}] \\
y_{i+1} = \tilde{m} [y_i + x_i \cdot d_i \cdot 2^{-i}] \\
\tilde{m} = \sqrt{\cos^2 \theta + \sin^2 \theta} = 1 + \epsilon,
\]

(11)

where \( \tilde{m} \) is the required scaling factor per iteration and \( \epsilon \) is the scaling error. The idea of the \( \mu \)-CORDIC rotation is to reduce the number of iterations of the full CORDIC to only a few iterations. Meanwhile, the scaling error \( \epsilon \) will be small enough to be neglected as long as the orthonormality is retained.

Figure 3 shows four different methods for different sizes of \( \mu \)-rotation angles and Table 1 shows a lookup table for the \( \mu \)-CORDIC, listing 32 approximated rotation angles for each \( \mu \)-rotation type, the required number of shift-add operations and its computation cycles. Note that the approximated angles are stored as two times of tan \( \theta \). When the rotation angle is very tiny (i.e., \( \epsilon \) is tiny, too), Type I with only one iteration will comply with the limited working range \( 1 - 2^-(n_{\mu} + 1) < \tilde{m} < 1 + 2^-(n_{\mu} + 1) \), if the selected \( n_{\mu} (n_{\mu} \in 1 \ldots 32) \) is larger than 16. In Figure 3(a), a pair of shift-add operations realizing one iteration step is sufficient. Furthermore, it is scaling free when the angle \( 2 \times \tan \theta \leq 3.05176 \times 10^{-5} \). These orthonormal \( \mu \)-rotations are chosen such that they satisfy a predefined accuracy condition in order to approximate the original rotation angles and are constructed by the least computation efforts.

Next, for the Type II rotation (as shown in Figure 3(b)), when \( n_{\mu} \) is selected from 8 to 15 for small angles, two pairs of shift-add operations are enough to retain the orthonormality. Moreover, when the \( n_{\mu} \) is selected from 5 to 7, Type III requires three \( \mu \)-rotations. No scaling is required by Types I through III. Finally, for large rotation angles, the scaling errors cannot be omitted. Figure 3(d) shows the corresponding dependence flow graph for Type IV. Besides the rotation
Figure 2: Flow graph of a folded CORDIC (recursive) processor.

itself, it requires two pairs of shift-add operations at the beginning of the flow graph, while 2 to 4 pairs of shift-add operations are required to fix the scaling factor $\tilde{m}$:

$$\tilde{m} = \left(1 + 2^{-2(k+1)}\right) \left(1 + 2^{-4(k+1)}\right) \cdots \left(1 + 2^{-2^M(k+1)}\right).$$

Note that the scaling costs $M = 2$ to 4 pairs of shift-add operations. In general, the cost of Type IV is bounded by $2 + M$ pairs of shift-add operations. For example, when the index $k$ is 2, the scaling is

$$\tilde{m}_2 = \left(1 + 2^{-6}\right) \left(1 + 2^{-12}\right) \left(1 + 2^{-24}\right).$$

3.3. Adaptive $\mu$-CORDIC Iterations. To improve the computational efficiency, the $\mu$-CORDIC has been modified to perform 6 iterations per cycle as CORDIC-6. As the global clock in a synchronous circuit is determined by the critical path, the maximum timing delay per iteration is 6 cycles (when the index $k$ is 1, Type IV). Therefore, the inner iteration steps of the angles are repeated until they are close to the critical one. The required number of repetitions is quoted in Table 1. For example, when the rotation angle index $k$ is 8, it will repeat three times from the index $k = 8$ to the index $k = 10$; when the rotation angle index $k$ is 20, it is repeated six times from the index $k = 20$ to the index $k = 29$. On the other hand, we can adjust the number of iterations by selecting the average angle during the last sweep and name it as CORDIC-mean.

4. Experimental Results

4.1. Matlab Simulation. The full CORDIC with 32 iteration steps, the $\mu$-CORDIC with one iteration step, and two different adaptive modes have been tested using numerous
Figure 3: Four $\mu$-CORDIC rotations.

random symmetric matrices $A$ of size $8 \times 8$ to $160 \times 160$ (i.e., EVD array sizes range from $4 \times 4$ to $80 \times 80$). Figure 5(a) shows the average number of sweeps needed to compute the eigenvalues/eigenvectors for each size of the EVD array, where the sweep number increases monotonically.

When the Jacobi EVD array size is $10 \times 10$, the $\mu$-CORDIC requires 12 sweeps while the full CORDIC only requires 6 sweeps per EVD computation. If we adjust the inner rotations to six times, the sweep number will be 10, smaller than the $\mu$-CORDIC but more than the full CORDIC. Note that using
Table 1: The lookup table for \(\mu\)-rotations CORDIC with 32-bit accuracy, showing the rotation type, the \(2 \times \tan \theta\) angle, the required shift-add operations for rotation and scaling, the required cycle delay, and repeat number for CORDIC-6.

<table>
<thead>
<tr>
<th>Index</th>
<th>Type</th>
<th>Angle (2 \times \tan \theta)</th>
<th>Shift-add</th>
<th>Cycle</th>
<th>Re.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IV</td>
<td>1.49070</td>
<td>4 8 6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>IV</td>
<td>0.54296</td>
<td>4 6 5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>IV</td>
<td>0.25501</td>
<td>4 6 5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>IV</td>
<td>0.12561</td>
<td>4 4 4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>III</td>
<td>6.25841 \times 10^{-2}</td>
<td>6 0 3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>III</td>
<td>3.12606 \times 10^{-2}</td>
<td>6 0 3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>III</td>
<td>1.56263 \times 10^{-2}</td>
<td>6 0 3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>II</td>
<td>7.81266 \times 10^{-3}</td>
<td>4 0 2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>II</td>
<td>3.90627 \times 10^{-3}</td>
<td>4 0 2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>II</td>
<td>1.95313 \times 10^{-3}</td>
<td>4 0 2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>II</td>
<td>9.76563 \times 10^{-4}</td>
<td>4 0 2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>II</td>
<td>4.88281 \times 10^{-4}</td>
<td>4 0 2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>II</td>
<td>2.44141 \times 10^{-4}</td>
<td>4 0 2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>II</td>
<td>1.22070 \times 10^{-4}</td>
<td>4 0 2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>II</td>
<td>6.10352 \times 10^{-5}</td>
<td>4 0 2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>I</td>
<td>3.05176 \times 10^{-5}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>I</td>
<td>1.52388 \times 10^{-5}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>I</td>
<td>7.62939 \times 10^{-6}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>I</td>
<td>3.81470 \times 10^{-6}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>I</td>
<td>1.90735 \times 10^{-6}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>I</td>
<td>9.53674 \times 10^{-7}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>I</td>
<td>4.76837 \times 10^{-7}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>I</td>
<td>2.38419 \times 10^{-7}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>I</td>
<td>1.19209 \times 10^{-7}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>I</td>
<td>5.90406 \times 10^{-8}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>I</td>
<td>2.98023 \times 10^{-8}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>I</td>
<td>1.49012 \times 10^{-8}</td>
<td>2 0 1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>I</td>
<td>7.45058 \times 10^{-9}</td>
<td>2 0 1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>I</td>
<td>3.72529 \times 10^{-9}</td>
<td>2 0 1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>I</td>
<td>1.86265 \times 10^{-9}</td>
<td>2 0 1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>I</td>
<td>9.31323 \times 10^{-10}</td>
<td>2 0 1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>I</td>
<td>4.65661 \times 10^{-10}</td>
<td>2 0 1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

4.2 VLSI Implementation. The \(\mu\)-CORDIC is modeled and compared to the folded Full CORDIC in VHDL with the resizing feature. These two methods have been integrated into parallel EVD arrays, with sizes \(4 \times 4\) and \(10 \times 10\), through a configurable interface separately. After that, they have been synthesized by using the Synopsys Design Compiler with the TSMC 45 nm standard cell library. Note that the word length is 32 bits with the IEEE 754 single floating precision for both CORDIC methods using the same floating point unit from shift-add operations than others. The adaptive CORDIC-6 method can offer a compromise between the hardware complexity and the computational effort.

Figure 5(c) shows the off-diagonal Frobenius norm versus the sweep numbers for each array size of \(80 \times 80\) with double floating precision. Each iteration method converges to the predefined stop criteria: \(\|A_{\text{off}}\|_F \times 10^{-6}\). The \(\|A_{\text{off}}\|_F\) is the Frobenius norm of the off-diagonal elements of \(A\) (i.e., \(A_{\text{off}} = A - \text{diag(diag}(A))\)).

Figure 5(d) shows the reduction of the off-diagonal Frobenius norm versus the sweep numbers for single floating precision. It can be noticed that the off-norms do not reach the convergence criteria, and each size of the EVD array has different stop criteria for each iteration method (default IEEE 754 single). Therefore, we can first analyze the Frobenius norm of the off-diagonal elements in Matlab and then observe it until it reaches its maximal reduction. Afterwards, a lookup table can be generated and directly assign these stop criteria to the target hardware circuit or IP component.
Table 2 lists the synthesis results for area, timing delay, and power consumption.

As expected, the combinational logic area and the power consumption of the $\mu$-CORDIC PE are much smaller than the Full CORDIC. Furthermore, in order to determine the time required to compute the EVD of a $n \times n$ symmetric matrix, it can be obtained by

$$T_{\text{total}} = T_{\text{delay}} \times K_{\text{iteration}} \times K_{\text{sweep}} \times [3 (n - 1) + \Delta + 3],$$

(14)

where $n = 8, 16, 20, 30, \ldots, 160, \Delta = n/2 - 1$.

The total timing delay per EVD operation is defined by the critical timing delay $\times$ the number of inner CORDIC rotations $\times$ average number of outer sweeps $\times$ size of the matrix $A$.

It can be observed that the total operation time is dependent on the relationship between the inner CORDIC rotations and the outer sweeps. Therefore, one obtains a speedup by a factor of 21.4 by reducing the number of inner CORDIC rotations. Although the reduction of power consumption is less significant due to an extra $\mu$-CORDIC's controller and multiplexers, it actually consumes much less energy per EVD computation due to the shorter computation time. Note that the $\mu$-CORDIC PE requires two inner iterations on average due to the different rotation cycles, from six to one inner iteration, as shown in Table 1. Figure 6 shows the energy consumption for sizes of the array from $4 \times 4$ to $80 \times 80$. Both rotation methods consume much less energy than the Full CORDIC, where the 6-CORDIC can obtain a factor of 40.9 and the $\mu$-CORDIC can obtain a factor of 104.3 on average for energy reduction compared to the Full CORDIC.

In [14], a Jacobi single cycle-by-row EVD algorithm [15] has been implemented with a single CORDIC processor.
Table 2: Comparison of 4 × 4 and 10 × 10 Jacobi EVD arrays.

<table>
<thead>
<tr>
<th>PE array size</th>
<th>Full CORDIC 4 × 4</th>
<th>μ-CORDIC 4 × 4</th>
<th>Full CORDIC 10 × 10</th>
<th>μ-CORDIC 10 × 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combinational</td>
<td>0.847 mm²</td>
<td>0.296 mm²</td>
<td>5.143 mm²</td>
<td>1.829 mm²</td>
</tr>
<tr>
<td>Noncombinational</td>
<td>0.390 mm²</td>
<td>0.123 mm²</td>
<td>2.306 mm²</td>
<td>0.833 mm²</td>
</tr>
<tr>
<td>Total</td>
<td>1.237 mm²</td>
<td>0.419 mm²</td>
<td>7.449 mm²</td>
<td>2.662 mm²</td>
</tr>
<tr>
<td>Power</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cell</td>
<td>62.283 mW</td>
<td>18.239 mW</td>
<td>388.379 mW</td>
<td>123.215 mW</td>
</tr>
<tr>
<td>Net</td>
<td>0.465 mW</td>
<td>0.433 mW</td>
<td>2.993 mW</td>
<td>2.678 mW</td>
</tr>
<tr>
<td>Leakage</td>
<td>11.909 mW</td>
<td>3.765 mW</td>
<td>86.136 mW</td>
<td>23.966 mW</td>
</tr>
<tr>
<td>Total</td>
<td>74.657 mW</td>
<td>22.437 mW</td>
<td>477.508 mW</td>
<td>149.859 mW</td>
</tr>
<tr>
<td>Timing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Critical</td>
<td>4.454 ns</td>
<td>1.213 ns</td>
<td>4.286 ns</td>
<td>2.247 ns</td>
</tr>
<tr>
<td>Frequency</td>
<td>224.5 MHz</td>
<td>824.4 MHz</td>
<td>233.3 MHz</td>
<td>445 MHz</td>
</tr>
</tbody>
</table>

Figure 6: The energy consumption per EVD operation with each size of EVD array (operating at 100 MHz).

5. Conclusions

The EVD was computed by the parallel Jacobi method, which was selected as an example for a typical iterative algorithm which exhibits very robust convergence properties. A configurable Jacobi EVD array with both Full CORDIC and μ-CORDIC is implemented in order to further study the tradeoff strategies in design criteria for parallel integrated circuits. The experimental results indicate that the presented μ-CORDIC method can reduce the size of the combinational logic, speed up the overall computation time, and improve the energy consumption.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

The authors would like to thank the National Science Council of Taiwan for the support under the contract of NSC 101-2218-E-150-001.

References


Submit your manuscripts at http://www.hindawi.com