P-CORDIC: A Precomputation Based Rotation CORDIC Algorithm

Martin Kuhlmann  
Broadcom Corporation, Irvine, CA 92619, USA  
Email: kuhlmann@broadcom.com

Keshab K. Parhi  
Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA  
Email: parhi@ece.umn.edu

Received 30 August 2001 and in revised form 14 May 2002

This paper presents a CORDIC (coordinate rotation digital computer) algorithm and architecture for the rotation mode in which the directions of all micro-rotations are precomputed while maintaining a constant scale factor. Thus, an examination of the sign of the angle after each iteration is no longer required. The algorithm is capable to perform the CORDIC computation for an operand word-length of 54 bits. Additionally, there is a higher degree of freedom in choosing the pipeline cutsets due to the novel feature of independence of the iterations \( i \) and \( i - 1 \) in the CORDIC rotation.

Keywords and phrases: CORDIC, computer arithmetic, constant scale factor, precomputation, rotation mode.

1. INTRODUCTION

CORDIC (coordinate rotation digital computer) [1, 2] is an iterative algorithm for the calculation of the rotation of a 2-dimensional vector, in linear, circular, or hyperbolic coordinate systems, using only add and shift operations. It has a wide range of applications including discrete transformations such as Hartley transform [3], discrete cosine transform [4], fast Fourier transform (FFT) [5], chirp Z transform (CZT) [6], solving eigenvalue and singular value problems [7], digital filters [8], Toeplitz system and linear system solvers [9], and Kalman filters [10]. It is also able to detect multiuser in code division multiple access (CDMA) wireless systems [11].

The CORDIC algorithm consists of two operating modes, the rotation mode and the vectoring mode, respectively. In the rotation mode, a vector \((x, y)\) is rotated by an angle \(\theta\) to obtain the new vector \((x', y')\) (see Figure 1). In every micro-rotation \(i\), fixed angles of the value \(\arctan(2^{-i})\) are subtracted or added from/to the angle remainder \(\theta_i\), so that the angle remainder approaches zero. In the vectoring mode, the length \(R\) and the angle towards the \(x\)-axis \(\alpha\) of a vector \((x, y)\) are computed. For this purpose, the vector is rotated towards the \(x\)-axis so that the \(y\)-component approaches zero. The sum of all angle rotations is equal to the value of \(\alpha\), while the value of the \(x\)-component corresponds to the length \(R\) of the vector \((x, y)\). The mathematical relations for the CORDIC rotations are as follows:

\[
\begin{align*}
x_{i+1} &= x_i + m \cdot \sigma_i \cdot 2^{-i} \cdot y_i, \\
y_{i+1} &= y_i - \sigma_i \cdot 2^{-i} \cdot x_i, \\
z_{i+1} &= z_i - \frac{1}{m} \cdot \sigma_i \cdot \arctan\left(\sqrt{m}2^{-i}\right),
\end{align*}
\]

where \(\sigma_i\) is the weight of each micro-rotation and \(m\) steers...
the choice of rectangular \((m = 0)\), circular \((m = 1)\), or hyperbolic \((m = -1)\) coordinate systems. The required micro-rotations are not perfect rotations, they increase the length of the vector. In order to maintain a constant vector length, the obtained results have to be scaled by a scale factor \(K\). Nevertheless, assuming consecutive rotations in positive and/or negative directions, the scale factor is constant and can be precomputed according to

\[
K = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} (1 + \sigma_i^2 \cdot 2^{-2i})^{1/2}.
\]

(2)

The computation of the scale factor can be truncated after \(n/2\) iterations because the multiplicands in the last \(n/2\) iterations are 1 due to the finite word-length and do not affect the final value of \(K_1\),

\[
K_1 = \prod_{i=0}^{n/2} (1 + \sigma_i^2 \cdot 2^{-2i})^{1/2}.
\]

(3)

There are two different approaches for the computation of the CORDIC algorithm. The first one uses consecutive rotations in positive and/or negative direction, where the weight of each rotation is 1. Hence, \(\sigma_i\) is either \(-1\) or 1, depending on the sign of the angle remainder \(z(i)\). In every iteration a significant amount of time is used to examine the most significant bit in case of a binary architecture or the most significant three digits of a redundant architecture to predict the sign of \(z(i)\) and hence the rotation direction \(\sigma_i\). In comparison to the CORDIC implementations with constant scale factor, other implementations use a minimally redundant radix-4 or an even higher radix number representation [12, 13, 14]. These architectures make use of a wider range of \(\sigma_i\). In case of a minimally redundant radix-4 architecture, \(\sigma_i \in \{-2, -1, 0, 1, 2\}\). By using this numbering system, the number of iterations can be reduced. However, the computation time per iteration increases, since it takes more time to differentiate between five different rotation direction values and to generate five different multiples of arctan\((2^{-i})\). The scale factor also becomes variable and has to be computed every time, due to the absence of consecutive rotations leading to an increase in area.

To speed up the computation time of the CORDIC algorithm, either the number of iterations or the delay of each iteration have to be minimized. The proposed algorithm introduces a novel approach, in which the rotation direction can be precomputed by adding the rotation angle \(\theta\), a constant and a variable adjustment which is stored in a table. Hence, a significant speedup of the delay per iteration is obtained. Since all rotation directions are known before the actual rotation begins, more than one rotation can also be performed in one iteration leading to a reduction in latency. The proposed architecture also eliminates the \(z\)-datapath and reduces the area of the implementation.

This paper is organized as follows. Section 2 presents the theoretical background for the novel CORDIC algorithm for rotation mode and Section 3 presents the novel architecture. Section 4 performs an evaluation of different CORDIC architectures while Section 5 concludes the paper.

2. THE NOVEL CORDIC ALGORITHM

2.1. Mathematical derivation using Taylor series

The summation of all micro-rotation with their corresponding weight \(\sigma_i\) is equivalent to the rotation angle \(\theta\)

\[
\theta = \sum_{i=0}^{n} \sigma_i \cdot \arctan (2^{-i}),
\]

(4)

where \(\sigma_i \in \{-1, 1\}\), corresponding to the addition and subtraction of the micro-angles \(\theta_i\). Since consecutive rotations are employed, the scale factor is constant. The value of \(\sigma\) can be interpreted as a number in radix-2 representation. The goal of the proposed method is to compute the sequence of the micro-angles \(\theta_i\) in a significant speedup of the delay per iteration is obtained.

Adding and subtracting \(2^{-i}\) to (4) results in

\[
\theta = \sum_{i=0}^{\infty} (2d_i - 1) \cdot (2^{-i} - 2^{-i} + \arctan (2^{-i}))
\]

(5)

\[
= \sum_{i=0}^{\infty} (2d_i - 1) \cdot 2^{-i} - \sum_{i=0}^{\infty} (2d_i - 1) \cdot (2^{-i} - \arctan (2^{-i}))
\]

(6)

\[
= 2d - 2 + \sum_{i=0}^{\infty} (2^{-i} - \arctan (2^{-i})) - \sum_{i=0}^{\infty} 2d_i (2^{-i} - \arctan (2^{-i}))
\]

(7)

\[
= 2d - c_1 - \text{sign}(\theta) \cdot 2 (1 - \arctan(1))
\]

(8)

where \(c_1\) corresponds to \(c_1 = 2 - \sum_{i=0}^{\infty} (2^{-i} - \arctan(2^{-i}))\). Solving (8) for \(d\) results in

\[
d = 0.5 \theta + 0.5c_1 + \text{sign}(\theta) \cdot (1 - \arctan(1))
\]

\[ - \sum_{i=1}^{\infty} d_i \cdot (2^{-i} - \arctan (2^{-i}))
\]

(9)

\[
= 0.5 \theta + c + \text{sign}(\theta) \cdot \epsilon_0 - \sum_{i=1}^{\infty} d_i \cdot \epsilon_i
\]

where \(c\) corresponds to \(0.5c_1\). Table 1 shows the values of the partial offsets \(\epsilon_i\) for the first 10 values of \(i\) and indicates that the value of \(\epsilon_i\) decreases approximately by a factor of 8 with increasing \(i\). Hence, the summation of \(d_i \epsilon_i\) can be limited to

\[
d = 0.5 \theta + c - \text{sign}(\theta) \cdot \epsilon_0 - \sum_{i=1}^{n/3} d_i \cdot \epsilon_i
\]

(10)

\[
d = 0.5 \theta + c - \text{sign}(\theta) \cdot \epsilon_0 - \delta.
\]
Rather than storing the partial offsets $c_i$ and computing the sum over all $i$ of the product $d_i c_i$, $\delta = \sum_{i=1}^{n/3} d_i c_i$ can be precomputed and stored. Hence, the only difficulty consists of determining which offset corresponds to the input $\theta$. This can be achieved by comparing the input $\theta$ with a reference angle $\theta_{ref}$. The reference angles $\theta_{ref}$ correspond to the summation of the first $n/3$ micro-rotation. To be certain to obtain the correct offset, $\theta$ has to be larger than the reference angle $\theta_{ref}$. All reference angles are stored in a ROM and are accessed by the most significant $n/3$ bits of $\theta$. In addition to the reference angles, the values of $\delta$ are stored. In case of a negative difference $\theta_{ref} - \theta$, the corresponding $\delta$ is selected, otherwise the next smaller value of $\delta$ is chosen to be subtracted from $\theta + c - \arctan(c) \cdot c_0$.

**Example 1.** Assuming we have a word-length of 16 bits and $\theta = 0.9773844$. According to Table 2, $\theta_{ref}$ corresponds to 0.97337076 and $\delta = 0.03644375$. Hence, $d$ is computed as

$$d = 0.5 \cdot \theta + 1 - 0.5 \cdot c + \sum_{i=0}^{n/3} d_i c_i$$

$$= 0.5 \cdot 0.9773844 + 1.08624513 + 0.03644375$$

$$= 1.6113811 = 1.10011101000001_{2},$$

$$\sigma = 1111111111111111_{1}.$$  

### 2.2. High precision

By using a mantissa of $n = 54$ bits (corresponding to the floating point precision), the ROM for storing all offsets would require $2^{18}$ entries. This is rather impractical since the required area to implement the ROM will exceed by far the area for the CORDIC implementation. To reduce the area for the ROM, $\delta$ can be split into two parts,

$$\delta = \delta_{ROM} + \delta_r,$$

where $\delta_{ROM}$ is stored in a ROM while $\delta_r$ is computed. By examining the Taylor series expansion of $\arctan(2^{-i})$, it becomes obvious that the partial offset $c$ for iteration $i$ and $i+1$

Table 1: The values of $\epsilon_i$ of the first 10 values of $i$.

<table>
<thead>
<tr>
<th>Iteration $i$</th>
<th>Partial offset $c_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.214601836602551690</td>
</tr>
<tr>
<td>1</td>
<td>3.63523909999176e-02</td>
</tr>
<tr>
<td>2</td>
<td>5.021336873135844e-03</td>
</tr>
<tr>
<td>3</td>
<td>6.450054325335649e-04</td>
</tr>
<tr>
<td>4</td>
<td>8.11900040265152e-05</td>
</tr>
<tr>
<td>5</td>
<td>1.016656973172375e-05</td>
</tr>
<tr>
<td>6</td>
<td>1.27137523169197e-06</td>
</tr>
<tr>
<td>7</td>
<td>1.589398898887035e-07</td>
</tr>
<tr>
<td>8</td>
<td>1.986803302817237e-08</td>
</tr>
<tr>
<td>9</td>
<td>2.483521181314878e-09</td>
</tr>
</tbody>
</table>

Table 2: The reference angles of the rotation mode and their corresponding values of $\delta$ for an operand word-length of 16 bits.

<table>
<thead>
<tr>
<th>$\theta_{ref}$</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.01640412</td>
<td>0.00008119</td>
</tr>
<tr>
<td>0.04607554</td>
<td>0.00009136</td>
</tr>
<tr>
<td>0.10746824</td>
<td>0.00064501</td>
</tr>
<tr>
<td>0.16964791</td>
<td>0.00065517</td>
</tr>
<tr>
<td>0.23230586</td>
<td>0.00072620</td>
</tr>
<tr>
<td>0.29478553</td>
<td>0.00073636</td>
</tr>
<tr>
<td>0.34871558</td>
<td>0.00502134</td>
</tr>
<tr>
<td>0.41119525</td>
<td>0.00503150</td>
</tr>
<tr>
<td>0.47355320</td>
<td>0.00510253</td>
</tr>
<tr>
<td>0.53603287</td>
<td>0.00511269</td>
</tr>
<tr>
<td>0.59742557</td>
<td>0.00566634</td>
</tr>
<tr>
<td>0.65990524</td>
<td>0.00567651</td>
</tr>
<tr>
<td>0.72226319</td>
<td>0.00574753</td>
</tr>
<tr>
<td>0.78474286</td>
<td>0.00575770</td>
</tr>
<tr>
<td>0.84605347</td>
<td>0.03635239</td>
</tr>
<tr>
<td>0.84853314</td>
<td>0.03636256</td>
</tr>
<tr>
<td>0.91089109</td>
<td>0.03643358</td>
</tr>
<tr>
<td>0.97337076</td>
<td>0.03644375</td>
</tr>
<tr>
<td>1.03476346</td>
<td>0.03699737</td>
</tr>
<tr>
<td>1.09724313</td>
<td>0.03700756</td>
</tr>
<tr>
<td>1.15960108</td>
<td>0.03707859</td>
</tr>
<tr>
<td>1.22208075</td>
<td>0.03708875</td>
</tr>
<tr>
<td>1.27601080</td>
<td>0.04137373</td>
</tr>
<tr>
<td>1.33849046</td>
<td>0.04138389</td>
</tr>
<tr>
<td>1.40084842</td>
<td>0.04145492</td>
</tr>
<tr>
<td>1.46332808</td>
<td>0.04146508</td>
</tr>
<tr>
<td>1.52472079</td>
<td>0.04201873</td>
</tr>
<tr>
<td>1.58720045</td>
<td>0.04202890</td>
</tr>
</tbody>
</table>

$$\epsilon_i = 2^{-i} - \arctan(2^{-i})$$

$$= \left( -\frac{2^{-3i}}{3} + \frac{2^{-5i}}{5} - \frac{2^{-7i}}{7} + \frac{2^{-9i}}{9} - \cdots \right).$$

$$\epsilon_{i+1} = 2^{-i-1} - \arctan(2^{-i-1})$$

$$= \left( -\frac{2^{-3i-3}}{3} + \frac{2^{-5i-5}}{5} - \frac{2^{-7i-7}}{7} + \frac{2^{-9i-9}}{9} - \cdots \right).$$

By comparing (13) and (16), it can be seen that (13) is about $2^3$ times larger than (16). Assuming a word-length of $n$ bits and $i \geq \lceil n/5 \rceil - 2$, the factor is $2^3$. Hence, the term $\epsilon_{\lceil n/5 \rceil - 1} = -2^{-3(\lceil n/5 \rceil - 1)/3} + 2^{-5(\lceil n/5 \rceil - 1)/5}$ can be stored in a ROM and
the remaining offset \( \delta_r \) is computed as

\[
\delta_r = \sum_{j=\lceil n/5 \rceil -1}^{\lceil n/5 \rceil} d_j \cdot \epsilon_{\lceil n/5 \rceil -1} \cdot 2^{-3(j-\lceil n/5 \rceil +1)} < 2^{-3(\lceil n/5 \rceil -1)}.
\]  

(17)

The largest magnitude of \( \delta_r \) is smaller than \( 2^{-3(\lceil n/5 \rceil -1)} \).

**Example for high precision**

Assume that we have a word-length of 50 bits and \( \theta = \frac{0.977384381116}{0.03644501895249} \) can be obtained. Hence, \( d \) is computed according to

\[
d = 0.5 \cdot \theta + 1 - 0.5 \cdot c + \delta_{ROM} + \delta_r
\]

\[
= 0.5 \cdot \frac{0.9773843811160 + 1.08624514683872}{0.03644501895249 + 2.483521181241566}
\]

\[
- 09 \cdot (1 + 2^{-18} + 2^{-21} + 2^{-24})
\]  

(18)

\[
= 1.61138235883274
\]

\[
= 1.100110010000011100110111001000100110111000112.
\]

**2.3. The rotation mode in hyperbolic coordinate systems**

Similar to the circular coordinate system, a simple correlation between the input angle \( \theta \) and the directions of the micro-rotation can be obtained. Due to the incomplete representation of the hyperbolic rotation angle \( \theta_i \), some iterations have to be performed twice. In [2], it was recommended that every 4th, 13th, \((3k + 1)\)th iteration should be repeated to complete the angle representation.

Similar to the rotation mode in circular coordinate system, the rotation angle \( \theta \) is equivalent to the summation of all micro-rotation with their corresponding weight. This leads to

\[
\theta = \sum_{i=0}^{\infty} \alpha_i \cdot \arctanh (2^{-i}) + \sigma_4^{\text{extra}} \cdot \arctanh (2^{-4})
\]

\[
+ \sigma_3^{\text{extra}} \cdot \arctanh (2^{-13}) + \sigma_0^{\text{extra}} \cdot \arctanh (2^{-40}) + \ldots.
\]

(19)

Performing a Taylor series expansion and applying \( \alpha_i = 2d_i - 1 \) results in

\[
d = 0.5 \theta + 0.5c_1 + \text{sign}(\theta) \cdot (2(0.5 - \text{arctanh}(0.5))
\]

\[
+ \left( d_4^{\text{extra}} - \frac{1}{2} \right) \cdot \text{arctanh} (2^{-4})
\]

\[
+ \left( d_3^{\text{extra}} - \frac{1}{2} \right) \cdot \text{arctanh} (2^{-13})
\]

\[
+ \left( d_0^{\text{extra}} - \frac{1}{2} \right) \cdot \text{arctanh} (2^{-40})
\]

\[
= 0.5 \theta + c + d_4^{\text{extra}} \cdot \text{arctanh} (2^{-4}) + d_3^{\text{extra}} \cdot \text{arctanh} (2^{-13})
\]

\[
+ d_0^{\text{extra}} \cdot \text{arctanh} (2^{-40}),
\]

(20)

where \( c \) corresponds to \( c = 1 - 0.5 \sum_{i=1}^{\infty} (2^{-i} - \text{arctanh}(2^{-i})) - 0.5 \cdot (\text{arctanh}(2^{-4}) + \text{arctanh}(2^{-13}) + \text{arctanh}(2^{-40}) + \ldots) \).

Since these extra rotation are not known in advance, an efficient high precision VLSI implementation is not possible. However, for signal processing applications using a word-length of less than 13 bits, the ROM size corresponds to only 14 entries.

**3. The novel rotation-CORDIC architecture**

For an implementation with the operand word-length of \( n \) bits, the pre-processing part consists of a ROM of \( 2\lceil n/5 \rceil -2 \) entries in which the reference angles \( \theta_{ref} \) and the corresponding offsets \( \delta \) are stored, respectively (see Figure 2). To avoid a second access to the ROM in case of \( \theta_{ref} > \theta \) the next smaller offset \( \delta_{k-1} \) is additionally stored in the \( k \)th entry of the ROM.

The ROM is accessed by the \( \lceil n/5 \rceil -2 \) MSB bits of \( \theta \). A binary tree adder computes, whether \( \theta \) is smaller or larger than the chosen reference angle \( \theta_{ref} \) and selects the corresponding offset (either \( \delta_k \) or \( \delta_{k-1} \)). Using a 3 : 2 compressor and another fast binary tree adder, the two required additions to obtain \( d_{\text{approx}} = 0.5\theta + c_2 + \delta_{ROM} \) can be performed, where \( c_2 \) corresponds to \( c + \text{sign}(\theta)c_3 \). Using the bits \( d_{\lceil n/5 \rceil -1} \) to \( d_{\lceil n/5 \rceil} \), \( \delta_c \) can be computed according to (17) and has to be added to \( d_{\text{approx}} \).

For the worst case scenario, there is a possible ripple from the bit \( d_{\lceil n/5 \rceil -1} \) to the bit \( d_{\lceil n/5 \rceil} \) which would call for a time consuming ripple adder. However, by employing an extra rotation for \( d_{\lceil n/5 \rceil -1} \) this limitation can be resolved. This extra rotation corresponds to the overflow bit of the addition from the bits \( d_{\text{approx}}^{\lceil n/5 \rceil -1} \) and \( \delta_r \). The additional rotation also does not affect the scale factor, since \( 3(\lceil n/5 \rceil -1) > n/2 \). For a precision of \( n \leq 16 \) bits, there are less than 32 offsets which can be stored in a ROM and the additional overhead to compute \( \delta_r \) can be removed.

The alternative architecture can be chosen by realizing that the directions of the micro-rotations are required in a most significant bit first manner (see Figure 2). As in the previous architecture, a fast binary adder is employed to determine which offset has to be selected. A redundant sign digit adder adds \( 0.5\theta, c \), and \( \delta_{\text{ROM}} \) and on the-fly converter starts converting resulting into the corresponding binary representation. Normally, the most significant bit cannot be determined until the least significant digit is converted. However, such worst cases do not exist in the CORDIC implementation, due to the redundant representations of the angles \( \text{arctan}(2^{-i}) \), where

\[
\text{arctan}(2^{-i}) < \sum_{k=1}^{\infty} \text{arctan}(2^{-k}),
\]

as opposed to the binary representation

\[
2^{-i} > \sum_{k=1}^{\infty} 2^{-k}.
\]

(22)

Therefore, it is not possible that there are more than \( l - 1 \) consecutive rotations in the same direction. In case that there are \( l - 1 \) consecutive rotations in the same direction, the \( l \)th
Each next rotation has to be performed in the negative direction. This happens if the angle remainder $z_i \approx 0$. Table 3 shows the maximum number of consecutive unidirectional rotations depending on the iteration number $i$. This limitation leads to a reduction in the complexity of the online converters and its most significant bits can already be used to start the rotations in the $x/y$ datapath.

**Example 2.** Assuming an angle $\theta = 0.001$. Hence, the angle remainder $\theta_i$ correspond to

\[
\begin{align*}
\theta_0 &= 0.001, \\
\theta_1 &= \theta_0 - \arctan(1) = -0.7844, \\
\theta_2 &= \theta_1 + \arctan(0.5) = -0.3208, \\
\theta_3 &= \theta_2 + \arctan(0.25) = -0.0758, \\
\theta_4 &= \theta_3 + \arctan(0.125) = 0.0486.
\end{align*}
\]

The next rotation has to be performed in the negative directions, since $\theta_4 > 0$. Hence, it is not possible to obtain rotation sequence like $\phi_{0\ldots4} = 01111$ but it has to be $\phi_{0\ldots4} = 01110$.

### 3.1. Evaluation of the $z$-datapath

**Delay analysis**

In this paper, we assume a similar delay model as proposed in [14]. Nevertheless in [14], the unit delay is set to a gate delay while in our evaluation the unit delay is set to a full-adder delay. Hence, the delays for 2-input (NAND, NOR) gate, XOR, multiplexer, register, and full-adder are 0.25, 0.5, 0.5, 0.5, 0.5, and $1_{FA}$.

The determination of which offset has to be chosen consists of the delay of the decoder, the ROM, a fast binary $n$-bit tree adder and a multiplexer. Assuming a delay of $\log_2(m)$ gate delays for the decoder, where $m$ corresponds to the number of rows in the ROM ($m < \log_2(n) + 1$), one for the word-line driver and another for the ROM, $\log_2(n) \cdot t_{Max}$ for the fast binary adder and $0.5 \cdot t_{FA}$ for the multiplexer, we can obtain the correct value of $\delta_{ROM}$ after a delay of $(0.5 \log_2(n) + 1 + 0.25 \log_2(\log_2(n))) \cdot t_{FA}$.

A $3 : 2$ compressor can be employed to reduce the number of partial products to two. An additional fast binary tree adder can compute the final value of $d_{approx}$. Hence, the entire delay to obtain $d_{approx}$ corresponds to

\[
(0.5 \log_2(n) + 1 + 0.25 \log_2(\log_2(n))) + (1 + 0.5 \log_2(n)) \cdot t_{FA} = (\log_2(n) + 2.25) \cdot t_{FA}.
\]

After obtaining the bits $d_{[n/5-1]}$ to $d_{[n/3]}$, $\delta_r$ can be computed. Since the value of $\delta_r$ is smaller than $2^{3([n/5]-1)}$ and the value of $d_{approx} + \delta_r$ is not required before $2^{3([n/5])} t_{FA}$ the computation of $\delta_r$ is not in the critical path.

Alternatively to the $3 : 2$ compressor and the tree adder, a minimally redundant radix-$4$ sign digit adder can be employed which has a delay of two full-adders. Hence, all output digits are available after these two full-adder delays. An additional on-the-fly converter converts the digits into its equivalent binary representation starting with the MSD. It requires a delay of multiplexer and four NANDS/NORs to convert one digit which results in $1.5 t_{FA}$ per digit (1 digit = 2 bits). The last digit is converted after a delay of $(n/2 + 1) \cdot 1.5 t_{FA}$. As already described in Table 3, bit $n/3$ is stable as soon as the last digit (corresponding to bit $n$) has been converted. Hence, the $n/3$ rotation can be performed after a delay of $(n/2 + 1) \cdot 1.5 t_{FA}$. Therefore, the iterations $i = 0$ can already be performed after a delay of $(n/2 + 1) \cdot 1.5 t_{FA} - n/3 \cdot 2 t_{FA} = (1/12 \cdot n + 1) t_{FA}$. Note that the conversion of one redundant digit is performed faster than the addition/subtraction of the $x/y$ datapath. Hence, an initial delay of $(1/12 \cdot n + 1) t_{FA} + (\log_2(n) + 2.25) t_{FA} = (1/12 \cdot n + \log_2(n) + 3.25) t_{FA}$ has to be added to the delay of the $x/y$ datapath.

**Area analysis**

Previously, the area of the $z$-datapath consists of $n/2$ iterations in which $(n + \log_2(n + 2))$ multiplexers and $(n + \log_2(n + 2))$ full-adders and registers are employed. Additionally, due to the Booth encoding, in the last $n/4$ iterations, about $2(n + \log_2(n + 2))$ multiplexers and $(n + \log_2(n + 2))$ full-adders are required. Assuming $A_{FA} = 1.61 \cdot A_{max}$ and $A_{reg} = A_{FA}$ (areas are based on layouts), the hardware complexity of the $z$-datapath results in $A_z = 1.7 \cdot n(n + \log_2(n + 2))A_{FA}$. Assuming a word-length of 54 bits and neglecting the required area for
the examination of the most significant three digits, about 5700AFA are required.

The proposed architecture utilizes a ROM of word-length \(n\) and \(2^{\lceil \log_2(n/5) \rceil - 2}\) entries, requiring an area of \(n \cdot 2^{\lceil \log_2(n/5) \rceil - 2} \cdot A_{FA} \cdot 1/50\) resulting in 552AFA for a word-length of 54 bits. The implementation of the decoders can be done in multiple ways. NOR based decoders with precharge lead to the fastest implementation. However, the decoder area becomes larger. The decoder size per word-line corresponds to \(A_{dec} = 0.83A_{FA}\). Since \(2^{\lceil \log_2(n/5) \rceil - 2}\) decoders are required, the area for all decoder corresponds to \(A_{dec, total} = 0.83 \cdot 2^{\lceil \log_2(n/5) \rceil - 2} \cdot 424A_{FA}\), assuming a 54 bit word-length. The ROM has to store \(\theta_{ref}\), \(\delta_t\), and \(\delta_{k-1}\). This results in a total area for the ROM and the decoder of about 2080AFA. The computation of \(\delta_t\) requires \(n/5 - n/5 + 2 = 2n/15 + 2\) rows of CSA (carry-save-adders) and Muxes and a final fast binary tree adder. Note that each row of CSA adders and Muxes only consists of \((n - 3n/5 + 6)\) bits (the more significant bits are zero). The required area corresponds to \(10 \cdot 27A_{FA} + 10 \cdot 27A_{max} + 5 \cdot 27A_{FA}\) respectively. Hence, the computation of \(\delta_t\) requires 540AFA. Moreover, the two redundant sign digit adder require \(2n \cdot A_{FA}\), while the converter consists of about \(0.5n^2 + n\)A_{max}. This corresponds to 108 and 696AFA for a word-length of 54 bits. This makes a total of 3426AFA, which is about 60% of the \(z\)-datapath previously employed.

3.2. Evaluation of the \(x/y\) datapath

In the first \(n/2\) micro-rotations, the critical path of the \(x/y\) rotator part consists of a multiplexer and a 4 : 2 compressor, which has a combined critical path of 2 full-adders. The last \(n/2\) micro-rotations can be performed only using \(n/4\) iterations, since Booth encoding can be employed. However, the delay of the selection for the multiple of the shifted \(x/y\) components requires slightly more time, resulting in a delay of about one full-adder delay. The delay for the 4 : 2 compressor remains 1.5 full-adder. Hence, the critical path of the entire \(x/y\) rotator part consists of \(n/2 \cdot 2t_{FA} + n/4 \cdot 2 \cdot 2.5 \cdot t_{FA} = 1.625n \cdot t_{FA}\). Note that the direction of the first iteration is already known; hence, the first iteration is not in the critical path. Therefore, the critical path of the entire \(x/y\) rotator part consists of \((1.625n - 2)t_{FA}\).

As an example, for a word-length of \(n = 16\) bits, the \(x/y\) datapath delay and the entire delay of the CORDIC algorithm corresponds to 24 and 32.5 full-adder delays, respectively.

3.3. Scale factor compensation

Since the scale factor is constant, the \(x\) and \(y\) values can already be scaled while the rotation direction is being computed. The scaling requires an adder of word-length \((n + \log_2(n))\) bits. Using a binary tree adder, this results in a delay of \(\log_2(n + \log_2(n)) \cdot t_{max}\). For the scale factor, a CSD (canonic signed digit) representation can be used, leading to at most \(n/3\) nonzero digits. Applying a Wallace-tree for the partial product reduction, the total delay of the scaling results into \((0.5\log_2(n + \log_2(n)) + \log_13(n/3)) \cdot t_{FA} < (1/12 \cdot n + \log_2(n) + 3.25) \cdot t_{FA} = t_{min}\). Hence, the scaling of the \(x\) and \(y\) coordinates does not affect the total latency of the novel algorithm.

4. OVERVIEW OF PREVIOUSLY REPORTED CORDIC ALGORITHMS

The delay of every iteration can be decomposed into two different time delays, \(t_d,\sigma\) and \(t_{d,y}\), where \(t_{d,x}\) corresponds to the time delay to predict the new rotation direction while \(t_{d,y}\) corresponds to the time delay of the multiplexer/add structure of the \(x/y\) datapath. Various implementations have been proposed to obtain a speedup of the CORDIC algorithm. Improvements have been especially made in the reduction of \(t_{d,x}\).

In [17], the angle remainder has been decomposed every \(k = 3k + 1\) iteration. From the given angle \(\theta\), the first four rotation directions can be immediately determined. After performing the corresponding addition/subtraction of the terms \(\alpha_1 \cdot \alpha_2\) from the input angle \(\theta\) using CSA arithmetic, a fast binary tree adder computes the nonredundant result \(z_i\). The bits 4 to 13 of \(z_i\) deliver the rotation direction \(\sigma_i\) to \(\sigma_{i+3}\) which are used to perform the rotation in the \(x/y\) datapath and the computation of the next angle remainder \(z_{i+4}\). Hence, a low latency CORDIC algorithm is obtained. However, a significant reduction in latency is achieved at the cost of an irregular design. Furthermore, it is difficult to perform a \(\pi/2\) initial rotation or the rotation of index \(i = 0\) for circular coordinates, as it would force a conversion from redundant to conventional arithmetic for the \(z\) coordinate just after the first micro-rotation which is costly in time and area. Hence, this parallel and nonpipelined architecture only converges in the range of \([-1, 1]\). The overall latency of this architecture corresponds to about \(2n \cdot \log_2(n) + \log_2(n)\) full-adder delay.

In [18], a direct correlation between the \(z\) remainder after \([n/3]\) rotations and the remaining rotation direction have been shown. Hence, no more examination of the direction of the micro-rotation has to be performed leading to a considerable reduction in latency. However, in the first \([n/3]\) iteration a conventional method has to be employed.

In [19], the directions of the micro-rotation have been recoded using an offset binary coding (OBC) [20]. The obtained correlation is approximately piecewise linear since small elementary angles can be approximated by \(a(i) = \arctan(2^{-i}) \approx s \cdot 2^{n-1}/s\), where \(s\) is the slope of the linearity. This is valid for \(i \geq m\), where \(m\) is an integer which makes the approximation tolerable (normally \(m = \lceil n/3 \rceil\)). Hence, the following correlation can be obtained:

\[
\sum_{i=m}^{n-1} \sigma_i 2^i \approx s \cdot \sum_{i=m}^{n-1} \sigma_i \cdot 2^{-i-1}.
\]

By performing some arithmetic computations, the following correlation of the rotation direction can be obtained:

\[
\sum_{i=0}^{n-1} \sigma_i \cdot 2^{-i} = \sum_{i=0}^{n-1} \sigma_i \cdot 2^{-i-1} - \sum_{i=m}^{n-1} \sigma_i \cdot 2a(i)/s.
\]
Hence, a multiplication by the inverse of the slope $s$ is required. This multiplication can be simplified to two stages of addition for an operand word-length of 9 bits. However, in most digital signal processing application, the operands have a word-length of up to 16 bits. Hence, for those applications, the presented method requires more stages of addition to compensate the multiplication resulting in a more complex implementation and an increase in delay.

In [21], a double rotation method is introduced which compensates for the scale factor while performing the regular $x/y$ rotations. However, due to the double rotation nature of this method, $t_{d,xy}$ is increased to about twice its original value.

To reduce the latency of the CORDIC operation, [22] proposed an algorithm using online arithmetic. However, this results in a variable scale factor. This drawback is removed in [23]. In every iteration a significant amount of time is used to examine the most significant three digits to predict $\sigma_i$. The employed random logic requires a delay of about 1.5 full-adder delays. Since the $x/y$ datapath consists of a 4-2 compressor, it requires also a delay of 2 full-adders. Hence, the overall iteration delay corresponds to 3.5 full-adder delays. To maintain a constant scale factor, consecutive rotations are required in the first $n/2$, where $n$ corresponds to the word-length of the operands. For the computation of the last $n/2$ bits, Booth encoding can be employed reducing the number of iterations by a factor of 2. However, the selection of multiple of the shifted $x$ and $y$ operands requires an additional multiplexer delay and increases the overall iteration delay to 4 full-adder delays. Hence, the number of iteration is equivalent to 0.75$n$ which corresponds to a total latency of $3n$ full-adders (this does not include the scale operation and the conversion).

Other implementations like [24] remove the extra rotations by a branching mechanism in case that the sign of the remainder cannot be determined (most significant three digits are zero). Hence, no extra-rotations are required while the required implementation area is doubled. Nevertheless, the most significant three digits (or most significant six bits) still have to be examined for the prediction of the next rotation direction. In [25], the double step branching CORDIC algorithm is introduced which performs two rotations in a single step. Nevertheless, this method requires an examination of the most significant six digits to detect two rotation directions. Since some of the digits can be examined in parallel, the delay increases only to $2t_{FA}$. The computation time of a double rotation in the $x/y$ datapath is slightly reduced compared to two normal $x/y$ rotations. Hence, the total amount of computation time corresponds to $0.5n(2t_{FA} + 3t_{FA}) = 2.5t_{FA}$.

In [26], the signs of all micro-rotations are computed serially. However, a speed up of the sampling rate is achieved by separating the computation of the sign and the magnitude of every $z_i$ or $y_i$ remainder. The sign of every remainder is computed by a pipelined carry-ripple adder (CRA) leading to an initial latency of $n$ full-adders before the first CORDIC rotation can be performed. Nevertheless, after this initial latency, the following signs can be obtained with a delay of only one full-adder. This leads to an overall latency of $3n$ full-adder delays.

In comparison to the CORDIC implementations with constant scale factor, other implementations use a minimally redundant radix-4 or an even higher radix number representation [12, 13, 14]. By using this number system, the number of iterations can be reduced. However, the prediction of the $\sigma_i$ becomes more complicated, since there are more possible values for $\sigma_i$. In addition, the scale factor becomes variable and has to be computed every time, due to the absence of consecutive rotations. An online computation of the scale factor and a parallel scaling of the $x$ and $y$ operands can be achieved. Depending of the use of CSA or fast carry-propagate-adders (CCLA), the number of iterations can be reduced to $2[n/3] + 4$ and $n/2 + 1$, respectively. The iteration delay $t_{d,CSA}$ of the architecture using CSA adders corresponds to the same delay as already described for the last $n/2$ iteration in the constant scale factor using Booth-encoding, while the architecture employing the fast CCLA adders requires $1.5 - d,CSA$ [14]. Hence, the overall latency of these CORDIC algorithm using a minimally redundant radix-4 digit set corresponds to about $2n$ full-adder delays.

Table 4 provides a delay comparison between the proposed algorithm and other CORDIC implementations. Some of the delays have been taken from [14, 17, 26].

5. CONCLUSION

This paper presented a CORDIC algorithm for the rotation mode which computes the directions of the required micro-rotation before the actual CORDIC computations start while maintaining a constant scale factor. This is obtained by using a linear correlation between the rotation angle $\theta$ and the corresponding direction of all micro-rotations for the rotation mode. The rotation directions are obtained by adding the rotation angle $\theta$ to a constant and a variable offset which is stored in a ROM. An implementation for high precision is also provided which reduces the size of the required ROM. Hence, neither extra or double rotations nor a variable scale factor are required. The implementation is suitable for word-lengths up to 54 bits, while maintaining a reasonable ROM size.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Delay in $t_{FA}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>proposed</td>
<td>$1.625n + 1/12 - n + \log_2(n) + 1.25$</td>
</tr>
<tr>
<td>[14]</td>
<td>$2n + 6$</td>
</tr>
<tr>
<td>[26]</td>
<td>$3n + 1$</td>
</tr>
<tr>
<td>[21]</td>
<td>$3.75n$</td>
</tr>
<tr>
<td>[27]</td>
<td>$5.25n$</td>
</tr>
<tr>
<td>[25]</td>
<td>$2.5n$</td>
</tr>
<tr>
<td>[17]</td>
<td>$2n + \log_3(n) + \log_5(n)$</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENT
This work was supported by the Defense Advanced Research Projects Agency under contract number DA/DABT63–96-C-0050. Prof. Parhi is on leave from the Department of Electrical and Computer Engineering of the University of Minnesota, Minneapolis, MN, USA.

REFERENCES

Martin Kuhlmann received his Diplome Ingenieur and Ph.D. degrees in electrical engineering from the University of Technological Aachen, Germany in 1997 and from the University of Minnesota in 1999, respectively. Currently, he is a staff design engineer at Broadcom Corporation, Irvine, CA, USA. His research interests include computer arithmetic, digital communication, VLSI design, and deep-submicron crosstalk.

Keshab K. Parhi is a distinguished McKnight University Professor of Electrical and Computer Engineering at the University of Minnesota, Minneapolis, where he also holds the Edgar F. Johnson Professorship. He received the B.Tech., M.S.E.E., and Ph.D. degrees from the Indian Institute of Technology, Kharagpur (India) (1982), the University of Pennsylvania, Philadelphia (1984), and the University of California at Berkeley (1988), respectively. His research interests include all aspects of physical layer VLSI implementations of broadband access systems. He is currently working on VLSI adaptive digital filters, equalizers and beamformers, error control coders and cryptography architectures, low-power digital systems, and computer arithmetic.