SUPER - CORDIC: LOW DELAY CORDIC ARCHITECTURES FOR COMPUTING COMPLEX FUNCTIONS

A Thesis
Presented to
The Academic Faculty

by

Tushar Supe

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2015

SUPER - CORDIC: LOW DELAY CORDIC ARCHITECTURES FOR COMPUTING COMPLEX FUNCTIONS

Approved by:

Professor David V. Anderson, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Vijay Madisetti
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Marilyn Wolf
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: 03 December 2015

To my family

and friends

ACKNOWLEDGEMENTS

I would like to thank my advisor and the reading committee for their support and encouragement. I would especially like to thank my advisor Dr. David V. Anderson for his guidance and valuable inputs on my work throughout this past year. Finally I would like to thank my friends and family for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis proposes an optimized Co-ordinate Rotation Digital Computer (CORDIC) algorithm in the rotation and extended vectoring mode of the circular co-ordinate system. The CORDIC algorithm computes the values of trigonometric functions and their inverses. The proposed algorithm provides the result with a lower overall latency than existing systems. This is done by using redundant representations and approximations of the required direction and angle of each rotation. The algorithm has been designed to provide the result in a fixed number of iterations n for the rotation mode and $3\lceil n/2 \rceil + \lfloor n/2 \rfloor$ for the extended vectoring mode; where, n is a design parameter. In each iteration, the algorithm performs between 0 and p/n parallel rotations, where, p is the number of precision bits and n is the selected number of iterations. A technique to handle the scaling factor compensation for such an algorithm is proposed. The results of the functional verification for different values of n and an estimation of the overall latency are presented. Based on the results, guidelines to choosing a value of n to meet the required performance have also been presented.

CHAPTER I

INTRODUCTION

In several signal processing applications, it is necessary to compute the precise values of trigonometric functions in real-time. There exist different means to do so, including Taylor series approximations and CORDIC (Co-Ordinate Rotation Digital Computer). However, these methods either involve the use of a multiplier or are iterative, both of which result in a high output latency. The CORDIC algorithm is an iterative algorithm used to compute trigonometric functions without the use of a multiplier [12]. While the original algorithm worked in the circular co-ordinate system, the unified CORDIC extends the algorithm to be used with the hyperbolic and linear co-ordinate systems as well; thus enabling it to compute various more complex functions [13]. However, the algorithm in general takes about p iterations to provide an output having p bit precision. This high iteration count implies a high output latency. The algorithm also provides a scaled result, although the scaling factor is constant for basic CORDIC algorithms. Therefore, a scaled initial vector can be loaded before performing the iterations to the same effect.

These algorithms have been widely used for various applications such as matrix transforms [6], [17], decimal-binary conversions [2], singular value decompositions [3], etc. The critical path for the original CORDIC algorithm is the determination of the sign of the residual angle W. The sign of the residual angle is used to determine the direction of rotation of the next iteration. Some of the approaches to reducing this critical path delay are to use carry-save and signed-digit representations [3], binary to bipolar recoding (BBR), micro-rotation angle recoding (MAR) [4], modified vector rotational CORDIC (MVR-CORDIC) [15], Hybrid CORDIC [14], etc. These are

discussed further in Section II. A summary of many ways to optimize the CORDIC algorithm is presented in [9].

This thesis proposes an optimized algorithm to compute trigonometric functions in the circular co-ordinate system using the rotation mode. In the conventional algorithm, the rotations are performed in the direction determined by the residual angle; however, the magnitude of each rotation is predefined. The idea here is to perform rotations in the direction and of the magnitude, that provides lower residual value. While it is computationally intensive to find the magnitude of rotation that provides minimum error, it is relatively easy to find an approximate rotation magnitude that provides a decrease in the residue. This approximation does not provide the rotations enabling the minimum iteration count as provided by the trellis based searching schemes [16]; however, it enables us to perform the rotations in parallel, resulting in a reduction in the overall latency. This thesis also proposes an optimized algorithm to calculate the inverse of trigonometric functions using the extended vectoring mode. It is shown that the acceleration of convergence of this mode by considering the magnitude of rotations is difficult, but it can be done after the residue is sufficiently small. A redundant representation at a higher level of granularity has been used in both modes to provide an estimate of the value of the variable with good accuracy, while also reducing the critical path time of calculating the residual angle. The proposed algorithm has been designed and verified to work in the circular co-ordinate system, although it can be extended to support the other co-ordinate systems as well.

The structure of this thesis is as follows. Section II describes the original CORDIC algorithm in further detail and describes some noteworthy approaches to reduce the latency. In Section III, the proposed and generalized algorithm is described in detail as well as the proposed scaling compensation technique. A design implementation of a specific case with an example is described. Section IV analyzes the performance of the same. The conclusions of the paper are presented in Section V.

CHAPTER II

FUNDAMENTALS AND EXISTING APPROACHES

2.1 The CORDIC Algorithm

The original CORDIC algorithm has two modes of operation: rotation mode, which can be used to calculate the values of the sine and cosine functions, and the vectoring mode, which can be used to calculate the inverse of the same. The function of the algorithm can also be interpreted as the conversion of a unit vector from the polar to cartesian co-ordinate system and vice versa. It can be designed to accept input (provide output) in any unit, such as degrees, radians and binary fractions of a half revolution for the rotation (vectoring) mode. In general, it takes p iterations to converge to a result with precision of p bits for both modes of the algorithm. Also, each iteration involves a p bit operation on the critical path. Therefore, each iteration takes $O(p)$ time, while the entire algorithm takes $O(p^2)$ time.

$$\begin{aligned} x' &= cos(\beta - \alpha) \cdot (x - tan(\beta - \alpha) \cdot y) \\ y' &= cos(\beta - \alpha) \cdot (y + tan(\beta - \alpha) \cdot x) \end{aligned} \tag{1}$$

The Figure 1 shows two vectors in the circular co-ordinate systems. Given the co-ordinates of a vector (x, y, α), the co-ordinates of the other vector can be computed as shown in Eq. 1. The CORDIC algorithm performs rotations using similar equations. However, the rotations are performed about a predefined set of angles. These angles are of the form $\beta - \alpha = tan^{-1}(2^{-k})$. These values are chosen such that the multiplication of the tangent function with the co-ordinates, as shown in Eq. 1, can be performed as simple binary shifts. It has been shown in [12] that convergence is guaranteed by using these set of angles. Also, the multiplication of the cosine function

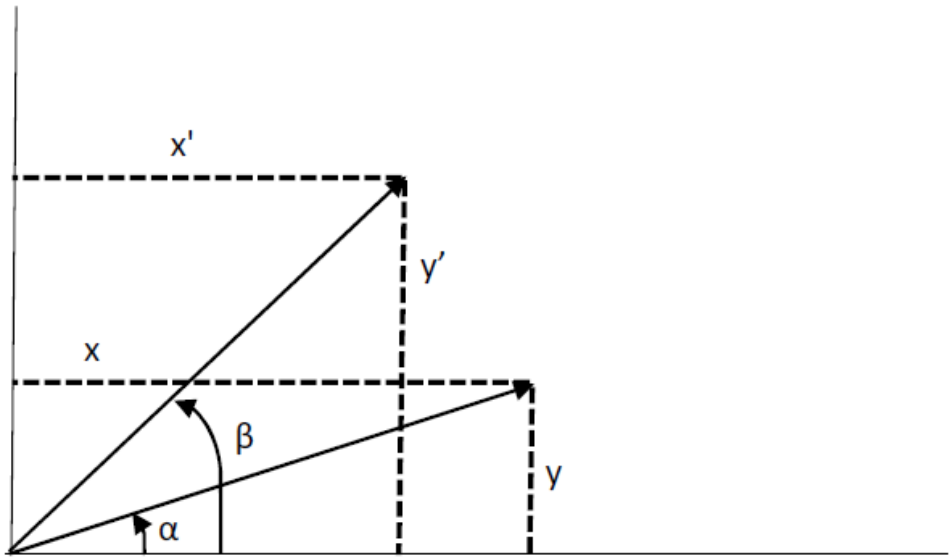

Figure 1: Vectors in Circular Co-ordiante System

is ignored in the CORDIC algorithm and thus, results in scaled vectors. However, this completely avoids the need for a multiplier and makes each rotation a simple shift and add operation.

In the rotation mode of the CORDIC, we start with a vector of known value which is generally chosen as 0 radian. This vector is then rotated about the predefined angles to converge onto the required input angle. In each iteration k, the vector is rotated by an angle of $tan^{-1}2^{-k}$ in the determined direction, while also accumulating a certain scaling in its magnitude. While the angle converges to the required angle, we get the X and Y co-ordinates as the value of the two trigonometric functions cosine and sine; however, the output is scaled by a certain factor. Although, since the set of angles used for each rotation is predefined, the scale factor introduced for a constant precision of p is constant. This can therefore be accounted for by loading a scaled initial vector instead [13]. The formula for each iteration is as shown in Eq. 2.

$$
\begin{aligned}
X_{i+1} &= X_i - \sigma_i \cdot 2^{-i} \cdot Y_i \\
Y_{i+1} &= Y_i + \sigma_i \cdot 2^{-i} \cdot X_i \\
W_{i+1} &= W_i - \sigma_i \cdot tan^{-1}(2^{-i}) \\
\sigma_i &= sign(W_i) \qquad\qquad (Rotation\ Mode) \\
\sigma_i &= -sign(Y_i) \qquad\qquad (Vectoring\ Mode)
\end{aligned} \tag{2}
$$

Here, X_k, Y_k are the co-ordinates of the vector after k^{th} iteration, W_k is the residual angle after the k^{th} iteration and $\sigma_k \in \{-1, 1\}$ is the direction of rotation for the $k + 1^{th}$ iteration. This conventional algorithm examines the sign of the residual angle after each iteration, to decide the direction of the next. Therefore, this algorithm determines the direction of rotation that could reduce the error, but the magnitude of the rotation is independent of any of these values. It is rather predefined for every iteration.

In the vectoring mode of the algorithm, the value of a vector is taken as input and stored as X and Y. The value of W is initialized to zero. Rotations are performed similar to the rotation mode, except that the direction of rotation is determined by the sign of Y instead. As the vector is rotated to converge onto the value of Y to 0, the angle corresponding to the input vector is converged upon in W. Here as well, the magnitude of the vector gets scaled with every rotation. However, this does not affect convergence or the final result. This is because the algorithm only depends on the sign of Y and not the value. Also, any change in magnitude has no effect on the angle of the vector. Therefore, this mode of the algorithm can be used to determine the angle of a vector with known cartesian co-ordinates with any magnitude. The drawback is that it requires both the co-ordinates. Its use to calculate inverse trigonometric functions therefore, requires as input the values of both sine and cosine.

2.2 Existing Approaches

It was soon realized that the critical data path for both modes of the algorithm, was the computation of the direction of rotation based upon the sign bit of the residual angle/Y co-ordinate. There have been approaches utilizing redundant number based implementations [3] to reduce this critical path delay and obtain a fast carry-free computation. This approach also enabled σ_i to take a value of 0 which meant that it could choose to not perform rotations for certain angles. Using this representation, each iteration effectively takes $O(1)$ time. Therefore, the entire algorithm takes $O(p)$ time. However, in this case the direction of rotation cannot be determined directly from the residual angle as it now utilizes a redundant representation. The direction therefore, is obtained by an estimate instead. This affects the accuracy of the result. Also, this raises the issue of a non-constant scaling factor as the magnitude of rotations performed differs in each case. The double rotation method [11] provides a constant scaling factor with redundant representations; however, the number of rotations performed is doubled compared to the original algorithm. Another approach was to use a radix-4 CORDIC algorithm [1]. In this method, the direction of rotation is allowed to assume one of five possible values. This algorithm requires half the number of iterations as compared to the conventional technique. However, the computation time of an iteration, also increases.

Angle Recoding algorithms utilizing the greedy algorithm [5], extended angle sets [16] and parallel angle recoding [10] successfully reduce the number of iterations required. These algorithms however, require rigorous computation to determine the rotation angle and direction, but provide very low iteration count of approximately $p/3$ iterations for p bit precision. It can therefore be used in cases where the input values are known beforehand, such as calculation of twiddle factors. However, its use to calculate these functions in real-time, would be unproductive. For a precision of p bits, the Hybrid CORDIC algorithm [14] provides an algorithm to compute the

result with $p/3 + 1$ iterations. The underlying idea is that after approximately $p/3$ iterations, the rotations can be performed in parallel. This is because the direction of rotation can be simply obtained by examining the respective bit in the residual angle. These methods also require a complex scaling factor compensation, which adds to the overall latency. The BBR and MAR techniques described by [4] provide great benefit in being able to perform rotations in parallel, but the recoding by itself takes considerable computation time.

$$
\begin{aligned}
X'_i &= X_i - \sigma_i \cdot 2^{-i-1} \cdot Y_i \\
X_{i+1} &= X'_i - \sigma_i \cdot 2^{-i-1} \cdot Y'_i \\
Y'_i &= Y_i + \sigma_i \cdot 2^{-i-1} \cdot X_i \\
Y_{i+1} &= Y'_i + \sigma_i \cdot 2^{-i-1} \cdot X'_i \\
W'_i &= W_i - \sigma_i \cdot tan^{-1}(2^{-i-1}) \\
W_{i+1} &= W'_i - \sigma_i \cdot tan^{-1}(2^{-i-1}) \\
T_{i+1} &= T_i + (T_i \cdot 2^{-2i}) \\
\sigma_i &= (T_i > X_i)\ ?\ -1\ :\ 1 \qquad (Cosine\ Matching) \\
\sigma_i &= (T_i > Y_i)\ ?\ 1\ :\ -1 \qquad (Sine\ Matching)
\end{aligned}
\tag{3}
$$

The vectoring mode of the algorithm can also be optimized using redundant representations [3]. However, its use to calculate inverse functions remains limited. A technique to obtain the inverse sine or cosine values, without the knowledge of the other, was proposed in [8] and [7]. This mode is called as the extended vectoring mode. The idea here was to perform CORDIC rotations to match the target value for X (Y) to calculate inverse cosine (sine). The equations for the same are shown in Eq. 3. Here, X_k, Y_k and W_k are the co-ordinates and the residual angle, similar to the rotation mode. The variable T_k is the current target to be matched, while σ_k is the direction of rotation. In this mode of the algorithm, rotations are performed

to converge to $X = T$ ($Y = T$), for cosine (sine) matching. However, since every rotation also scales the magnitude of the vector, a similar scaling needs to be added to the target T as well. Therefore, double rotations are performed here, which helps to scale the target value with only simple shift and add operations. This has been explained in further detail in Section 3.2.2. This technique also requires p iterations to converge to a result with p bit precision, while the computation time of each iteration has now tripled. This is because in every iteration we are performing a double rotation and then a comparison to determine the sign for the next iteration.

CHAPTER III

THE PROPOSED ALGORITHM AND ARCHITECTURE

3.1 Rotation Mode

The rotation mode of the proposed algorithm can be used to compute values of trigonometric functions such as sine and cosine, with lower latency. The following description of the algorithm represents the input and residual angle as binary fractions of a half revolution i.e $W \in [-1, 1) \mapsto [-\pi, \pi)$. However, these representations can be applied to radian representations as well.

Here, the residual angle W and the values of X and Y have been implemented using a redundant representation; however the redundancy for W is at a higher level of granularity. As described earlier, each rotation in the original algorithm is a p bit operation, but it guarantees the direction providing lower residue. Using the redundant representation, each rotation becomes a 1 bit operation; however, the direction is determined by an estimate. Therefore, the idea here is to use an intermediate representation for W. This would provide an intermediate computation time between the two initially described representations, while also providing adequate accuracy about its sign and value. Also, another salient feature of the proposed algorithm is that the magnitude of the residue is used to determine the rotation angle. That is, for example, for a very small input angle value, the original CORDIC algorithm would still start rotation by the maximum angle i.e. $tan^{-1}1$. Although, the algorithm stills converges onto the input angle, this iteration would increase the residue. However, the proposed algorithm examines the value of the residue to determine the rotation angle. While the greedy algorithm [5] and the trellis based searching schemes [16] also do the same, they involve rigorous computation. The proposed algorithm determines

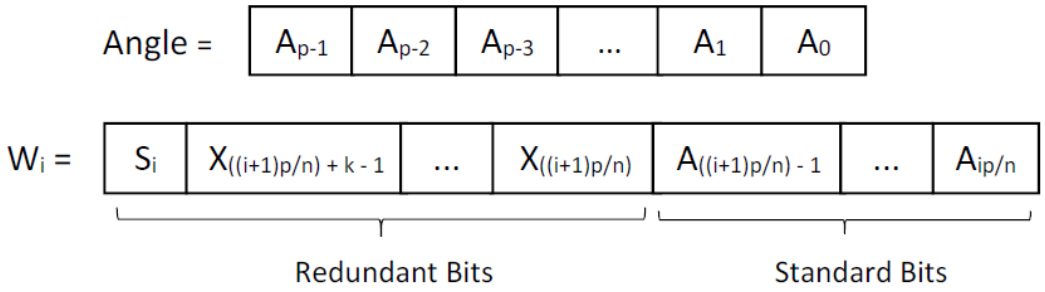

Figure 2: Set Representation

an approximate rotation angle, using negligible computation. Therefore, approximations of the required rotation angle and direction can be obtained to perform them in parallel.

For a fixed precision of p bits for the input angle and the output values, we divide the input angle into n sets of p/n bits respectively, where n is a fixed constant. Let these sets be W_i with $i \in [0, n-1]$. Each of these n sets is viewed as a counter with a particular weight of $2^{p((i/n)-1)}$ for the angle mapping $[0, 1] \mapsto [0, \pi]$. These sets are augmented to contain extra (redundant) bits, enabling them to assume a larger range of signed values as shown in Fig. 2. The sets are initialized such that the standard bits for each set are taken directly from the input angle, while the redundant bits are set to 0. Therefore, the residual angle after any rotation can be evaluated by Eq. 4. The representation used for X and Y can be similar to the carry-save representations, since it provides least computation time and the determination of the direction and magnitude of rotation does not depend on these values.

$$W = \sum_{i=0}^{n-1} W_i \times 2^{p((i/n)-1)} \tag{4}$$

The algorithm has been designed such that each iteration j would simply work on reducing the value of set W_{n-j} to 0 and it thus takes n iterations to compute the results. For the case of $n = p$, we would have a similar representation of the data as

Theorem 1. *For W being the residual angle and $i \in [2, \infty)$, if $2^{1-i} > |W/\pi| \geq 2^{-i}$ then $||W| - tan^{-1}(2^{2-i})| < |W|$*

Proof. If $W > 0$ and $W \geq tan^{-1}(2^{2-i})$ then

$$W - tan^{-1}(2^{2-i}) < W \tag{5}$$
$$tan^{-1}(2^{2-i}) > 0 \tag{6}$$

This is true for all values of $i \in [2, \infty)$

If $W > 0$ and $W < tan^{-1}(2^{2-i})$ then

$$-W + tan^{-1}(2^{2-i}) < W \tag{7}$$
$$tan^{-1}(2^{2-i}) < 2W \tag{8}$$

But W can only have a minimum value of $\pi 2^{-i}$

$$tan^{-1}(2^{2-i}) < \pi 2^{1-i} \tag{9}$$

This is also true for all values of $i \in [2, \infty)$
Similarly, this can be proven for negative values of W □

used in the carry-save and signed-digit redundant representations; whereas, for $n = 1$, the representation would be similar to the conventional CORDIC algorithm. However, the way this algorithm works is significantly different than either of these. Therefore, for intermediate values of n, we get an intermediate computation time and accuracy of the value. In the proposed algorithm, we implement a mechanism to provide the *direction* of rotation which guarantees the direction that can provide a reduction in error, while the *angle* or *magnitude* of rotation is determined by approximation. Also, each rotation here, is effectively a p/n bit operation.

3.1.1 Rotation Direction and Angle Determination

Here, each iteration comprises of multiple steps. While the conventional CORDIC would perform one rotation per iteration, the proposed algorithm performs multiple rotations in parallel. In each iteration j, the algorithm performs a rotation till the value of the respective residual set W_{n-j} is reduced to zero. The direction of rotation

for each step is determined by examining the sign of W_{n-j}, similar to the conventional algorithm. Whereas, the angle of rotation is chosen by the location of the most significant bit in W_{n-j}, that has a value which is the complement of its sign bit. The idea here is to simplify the determination of rotation angle and to perform a rotation of magnitude that reduces the error. Theorem 1 shows that for all values of the residual angle $W < \pi/2$, the residual angle reduces after each rotation based on this angle selection technique. A similar proof can also be shown for rotation magnitude determined in a similar fashion from W represented in radians. This provides a reduction in error; however, it does not ensure that it provides the minimal error. Therefore, this guarantees convergence provided the angle and direction of rotation are maintained as stated here. That is, convergence is guaranteed for sufficiently low values of n since the effect of redundancy decreases with n. To provide more accurate estimates of the angle, the algorithm incorporates a sign compensation technique which has been discussed further in Section 3.1.3. After each rotation, the values of the residual angle sets are updated to reflect the rotation and this process is repeated till W_{n-j} is zero. Besides, at the end of every iteration there is an overflow handling function, which adds the redundant bits of W_{n-j-2} to W_{n-j-1} ensuring optimum representation for the next iteration. The number of rotations performed in each iteration would range from $[0, p/n]$. The technique to handle the scaling factor introduced has been discussed in the following subsection.

3.1.2 Scaling Compensation

The algorithm performs rotations using a predefined set of angles taking the form $tan^{-1}(2^{-r})$, where $r \in [0, p-1]$. The size of the set of angles required, is equal to the number of precision bits p. In contrast to the conventional algorithm, the proposed algorithm performs a rotation about only a subset of this set of angles. Also, the angle used for every rotation is not predefined, but estimated from the value of the

respective residual set W_{n-j}. Therefore, there is a need to appropriately calculate and eliminate the scaling in the end result. The scaling factor introduced due to each rotation of $tan^{-1}2^{-r}$, is $(cos(tan^{-1}2^{-r}))^{-1}$.

$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} - ... \tag{10}$$

$$\lim_{x \to 0} tan^{-1}x = x \tag{11}$$

$$\begin{aligned} cos(tan^{-1}2^{-r}) &= 1 - 2^{-2r-1} + x \\ \text{where, } x &< 2^{-p} \\ r &\geq p/4 \end{aligned} \tag{12}$$

The scaling compensation technique employed here is based on a few heuristics. The Taylor series expansion of cosine has been shown in Eq. 10. Also, the value of arctan can be estimated for small values using Eq. 11. On combining these two equations, we get the relation presented in Eq. 12. This shows that the scaling factor introduced can be estimated using a simple shift and add operation. It can also be seen that for values of $r \geq p/2$, the value can be estimated as 1 and therefore, no additional operations need to be performed. The scaling factor introduced due to the angles in the range $tan^{-1}(2^{-p/2})$ to $tan^{-1}(2^{-p})$ is therefore, negligible.

For the $p/4$ angles for which the scaling factor cannot be estimated using Eq. 12, we can employ the double rotation method described in [11]. Here, we perform two rotations for every angle such that for a rotation in direction σ_i for angle $tan^{-1}2^{-r}$, we instead perform two rotations in the direction σ_i of angle $tan^{-1}2^{-r-1}$. While, if a rotation about any angle is not to be performed, we can instead perform two rotations about that angle, but in opposite directions. Thereby still accumulating the scaling factor, while maintaining the angle. This technique enables us to skip rotations about certain angles as desired by the algorithm. Therefore, the scaling factor remains constant. Here, we are using a carry-save implementation for X and

Algorithm 1: Super-CORDIC Rotation (W)

```
Initialize values of W[n − 1 : 0]
Load_Scaled_Initial_Vector
for j = 0 to n − 1 do
  iter = n − 1 − j
  for rot = 1 to p/n do
    if W_iter ! = 0 then
      Angle_determination(W_iter)
      W = Modify_residue(W)
      Perform_rotations
    else
      W = Overflow_handle(W)
      break
    end if
  end for
end for
```

Figure 3: Pseudo-Code for the proposed Super-CORDIC Rotation architecture

Y, therefore, the doubled number of rotations for the first $p/4$ angles is still not on the critical path as long as n is chosen so that $p/n > 4$.

3.1.3 Architectural Implementation

The architectural design for implementing this algorithm is described in further detail in this section. Here, in each iteration j, the angles that may be used for rotation, belong to the set $tan^{-1}(2^{-r})$ where $r \in [jp/n, (j+1)(p/n) - 1]$. During the first iteration, the scaled initial vector is loaded and double rotations are performed about every determined angle. In the second iteration, we perform an additional rotation to compensate for the scaling factor using the approximation as stated by Eq. 12. The entire algorithm is therefore, as described by the pseudo-code presented in Fig. 3. The presented algorithm provides convergence over the input domain of $[-\pi/4, \pi/4]$. The input domain has been explained further in the next subsection.

To provide a more accurate estimate of the residual angle, a sign compensation technique has been used. The technique avoids an error in angle and direction selection due to the redundant representation. In the carry save representation, a single

Table 1: Example of Sign Compensation for Input Angle 10 Degrees

Angle	W	W_4	W_3	W_2	W_1
10	0x1C71C7	0x7	0x7	0x7	0x7
Rotation $tan^{-1}\ 2^{-4}$		0x3FE	0x3DE	0x3D0	0x3CB
6.42367	0x124593	0x5	0x3E5	0x3D7	0x3D2
Rotation $tan^{-1}\ 2^{-4}$		0x3FE	0x3DE	0x3D0	0x3CB
2.84733	0x8195D	0x2	0x3	0x3A7	0x39D

carry is propagated to the next bit position in every step. Similar is the case with signed digit representations. The sign compensation technique described here is analogous to the same. It propagates a sign change to the higher residual set, without adding any additional latency. For the j^{th} iteration, it checks if the sign bits of W_{n-j} and W_{n-j-1} are different. In such a case, it complements the initial carry to the modify residue function for W_{n-j} and adds an equivalent number to W_{n-j-1}. However, in the case when the sign bits are different and $|W_{n-j}| = 1$, the rotation is not performed since the result of the sign compensation would result in $W_{n-j} = 0$. The pseudo-code for the entire modify residue function with the sign compensation technique has been described in Fig. 4. An example of sign compensation has been showed in Table 1. The algorithm has been applied on an input angle of 10 degrees for $p = 24$ and $n = 4$ and the first double rotation of $tan^{-1}\ 2^{-4}$ has been shown. The representation of the residual sets is, $W_i[5:0]$ are the standard bits and $W_i[9:6]$ are the redundant bits, including the sign bit. After the first rotation, the sign of W_4 and W_3 are different and therefore, sign compensation has been performed in the following rotation. It can be observed that the effective value of residue as computed by Eq. 4, is still consistent.

Determining the angle of rotation based on Theorem 1, provides a simple approach for positive values of W and low values of n. This is because lower the value of n, lower is the effect of redundancy. Also, the implementation of the same is essentially just an examination of the respective bits. However, this becomes a little complex in

Algorithm 2: Modify_residue (W)

$sign\_comp = W_{n-j}[s] \oplus W_{n-j-1}[s]$
for $set = n-1$ to 0 **do**
  $mod_{set} = (angle \times 2^{p-(set \times n)}/\pi) \bmod 2^{p/n}$
  $mod_{set} = (p/n)\{direction\} \oplus mod_{set}$
  $mod\_carry_{set} = direction$
end for
$mod\_carry_{iter} = sign\_comp \oplus direction$
$mod\_carry_{iter-1} = sign\_comp \ \& \ direction$
for $set = n-1$ to 0 **do**
  $W_{set} = W_{set} + mod_{set} + mod\_carry_{set}$
end for
return W

Figure 4: Approximate pseudo-code for Modify Residue Function

Algorithm 3: Angle_Determination (W_{iter})

$tangent = (p/n) \times (n - iter - 1) + 2$
$direction = W_{iter}[s]$
for $bit = 0$ to $(p/n) - 1$ **do**
  if $W_{iter}[(p/n) - 1 - bit] = W_{iter}[s]$ **then**
    $tangent \mathrel{+}= bit - W_{iter}[s]$
    break
  end if
end for
$angle = tan^{-1}(tangent)$

Figure 5: Approximate pseudo-code for Angle Determination Function

the case of negative values of W, since checking for the validity of the input condition of Theorem 1 is no longer a simple bit examination. Therefore, using the same logic as has been presented by the pseudo-code in Fig. 5, does not follow the angle determination logic of the theorem strictly and is more of an approximation.

The advantage of using this redundant representation is that each addition operation now becomes only a p/n bit operation. This implies that if there is a carry generated in any set W_i instead of propagating it to W_{i+1}, it may be stored within W_i as a overflow counter. To accommodate these overflow values, each set has some redundant bits as shown in Fig. 2. The required number of redundant bits k would

Algorithm 4: Overflow_Handle (W)

if $W_{iter} = 0$ **then**
  $W_{iter-1} += (p/n + 1)\{W_{iter-2}[sign]\} + W_{iter-2}[X]$
  $W_{iter-2}[X] = 0$
else
  $W_{iter-1} += W_{iter} << p/n$
  $W_{iter} = 0$
end if

Figure 6: Approximate pseudo-code for Overflow Handle Function

Table 2: Example for rotation mode for $p = 24, n = 4$ and input angle 25 degrees

Iteration	Residual Angle (W)	W_3	W_2	W_1	W_0	Rotation
	0x471C72	0x11	0x31	0x31	0x32	
Iteration 1	0x1F2F91	0x8	0x3F3	0x3FE	0x11	14.03624
	0xFFF742B0	0x3FF	0x3B5	0x3CB	0x3F0	14.03624
	0xFFF9CE88	0x3FF	0x3DD	0x3FA	0x8	-0.89517
	0xFFFC5A60	0	0x3C5	0x29	0x20	-0.89517
Iteration 2	0xFFFEE638	0	0x3EE	0x18	0x38	-0.89517
	0x2C29	0	0x3	0x3EF	0x69	-0.44761
	0x36B	0	0	0xD	0x2B	5.59528e-2
Iteration 3	0xE0	0	0	0x3	0x20	3.49705e-3
	0x3E	0	0	0	0x3E	8.74264e-4
Iteration 4	0x16	0	0	0	0x16	2.18566e-4
	0x2	0	0	0	0x2	1.09283e-4
	0	0	0	0	0	1.36603e-5

be as stated in Eq. 13 to ensure proper functionality even in the worst case. The number of redundant bits here, including the sign bit, would generally be lesser than in the case of other redundant approaches [3]. The pseudo-code for the overflow handle function has been presented in Fig. 6. It shows that, if there is case where an iteration j has not reduced W_{n-j} to zero, it is moved over into the W_{n-j-1} set allowing the next iteration to reduce it further. If W_{n-j} is zero, then the redundant bits from W_{n-j-2} are moved into W_{n-j-1} to ensure optimal representation for the next iteration.

To demonstrate the working of this algorithm, an example has been presented in table 3.1.3 for values $p = 24$ and $n = 4$. Here the input angle has been chosen as 25 degrees and is represented as a binary fraction of a half revolution. It can be seen from iteration 1 that double rotations are performed to preserve the scaling factor. The algorithm converges to $W_{n-j} = 0$ in each iteration j. The final error present in the values of X and Y is 1.493×10^{-7} and 3.062×10^{-7} respectively.

$$\begin{aligned} 2^{k+(p/n)} - 1 &\geq (2^{p/n} - 1) + (2^{log_2(p/n)} - 1) \\ &\quad + (p/n)(2^{p/n} - 1) \\ k &\geq log_2(p/n) + 1 \end{aligned} \tag{13}$$

The maximum computation time required for an iteration is p/n times the time required to modify the residue, plus a p/n bit operation for overflow handling. This evaluates to $(p/n) \times ((p/n) + 1)$ bit operations per iteration. Therefore, the choice of value of n is crucial here. Considering the scaling factor heuristics presented above, it follows that one should have $n \leq 4$ to avoid scaling error. Therefore, the ideal value of n can be considered to be 4 to keep the hardware cost minimal, while avoiding scaling error. The total latency of the algorithm can be viewed as n times the computation time of one iteration, which evaluates to $p((p/n) + 1)$. This implies that greater the value of n, lower is the total latency of the algorithm. However, for high values of n the error in the output due to the scaling factor and also due to the difficulty to maintain the angle determination logic based on Theorem 1, increases considerably. Note, in comparison to the conventional CORDIC algorithm, the effective iteration count of the proposed algorithm can be considered to be $(p/n) + 1$ as per the total latency of the algorithm. This is because, each iteration in the original CORDIC algorithm was a p bit operation. For a design with each iteration taking one cycle, this algorithm would therefore take n cycles. The algorithm could also be designed such that each

rotation takes one cycle. In this case, the total number of cycles required would vary between 0 to $p+n$. However, the time period of the cycle would be much smaller than the original algorithm, since the computation time of each rotation has decreased by a factor of n.

3.1.4 Input Domain Extension

The use of redundant representations pose problems in detection of a sign change, since the generated carry may or may not always propagate till the sign bit. For input angles very close to 0 radian, the proposed algorithm would perform rotations of very small magnitude to converge to the result and therefore, the vector never crosses the zero boundary where the value of Y would incur a sign change. In contrast, for input angle values close to $\pi/2$, the proposed algorithm performs rotations starting with a large magnitude, which may cause the vector to cross the $\pi/2$ boundary, where the value of X incurs a sign change. Therefore, the convergence domain for the proposed algorithm has been defined to be $[-\pi/4, \pi/4]$.

However, the algorithm can be extended to converge over the entire 2π domain as follows. Since the algorithm begins with loading a known and scaled vector value which is taken to be at angle 0 radian, we can simply change the the input domain by instead loading a different vector. The value of the residual sets W_i would also be initialized differently according to the loaded vector. Therefore, we cover the entire input domain of 2π by following the relations mentioned in Eq. 14. It also shows that for the different initial vectors, the value of the co-ordinate is either loaded in X_0 or Y_0 and it may be negative or positive. The absolute value required is the same and therefore, no additional storage is required.

$$
\begin{array}{llll}
W \in [-\pi/4, \pi/4) & \theta = 0 & X_0 = 1/K & Y_0 = 0 \\
W \in [\pi/4, 3\pi/4) & \theta = \pi/2 & X_0 = 0 & Y_0 = 1/K \\
W \in [3\pi/4, 5\pi/4) & \theta = \pi & X_0 = -1/K & Y_0 = 0 \\
W \in [5\pi/4, 7\pi/4) & \theta = 3\pi/2 & X_0 = 0 & Y_0 = -1/K
\end{array} \tag{14}
$$

where, W = Input angle

θ = Angular Co-ordinate of Initial Vector

$X_0,\ Y_0$ = Scaled Initial Vector Co-ordinates

When the representation used for W is that of binary fractions of the half revolution, checking for the domain of the input angle and accordingly loading the initial vector is simple. It can be achieved simply by examining the three MSB bits of the input angle. Also, initializing the residual sets W_i also does not require any additional overhead in terms of computation. In the case of a radian representation for W, it is not as simple to detect what domain the input angle lies in. However, the domains mentioned in Eq. 14 are not strict and are still valid as long as the input angle does not lie close to $\theta + \pi/2$. Therefore, we can have obscure boundaries for the domains which can be realized with a simple examination of the MSB bits as well. Although, initializing the value of the residual sets W_i has a computational overheard. We need to compute the value of $W - \theta$ and use this value to initialize the residual sets. This is a p bit operation and therefore, can be considered as an additional iteration. However, the benefit of using this representation is that for small values of the residue, the rotation direction and angle can be determined in parallel as presented in Hybrid CORDIC[14]. This enables us to perform all the iterations that do not involve any double rotation or scaling compensation, to be performed in parallel. For the ideal proposed value of $n = 4$, this implies that one less iteration is required. Therefore, the overall effect of the domain extension with the use of the Hybrid CORDIC technique

adds no additional latency.

3.2 Extended Vectoring Mode

The vectoring mode of the conventional CORDIC algorithm has limited uses as it requires the value of sine and cosine both, to compute the angle of its input vector. Therefore, it cannot be used to calculate the values of inverse sine and inverse cosine if one of these values is not known. As described above, the CORDIC extended vectoring as proposed by [8] provides a technique to calculate these functions. This is done by performing rotations of a known vector to match the input target value, opposed to the conventional vectoring mode where the initial vector itself is taken as input. However, to compensate for the scaling factor added after every rotation, the double rotation technique has been utilized. It has been shown in the following section how the double rotations help to estimate the scaling factor, without requiring any multiplication. Besides, the direction of each rotation is determined by the sign of the difference between the target and the corresponding value of the vector. This results in each iteration being thrice in length, as compared to the conventional CORDIC. Also, it can also be shown that only a output precision of $2^{-(p/2)}$ can be obtained for an input precision of p bits and this requires p number of iterations.

The differences between the approach for this mode in contrast to the rotation mode are as follows. In the rotation mode, the scaling factor can cause an error in the result and needs to be handled. In the vectoring mode, there is no requirement for this since any scaling in magnitude would not affect the sign or the angle of the vector. Although, in extended vectoring we are comparing the value of the vector to the input target value. Therefore, the algorithm in designed to modify the target by the same scaling factor, as is introduced by the CORDIC and then performing the comparison. Another approach to handling the scaling, as described in [7], is to start with a scaled initial vector to counter the scaling introduced by CORDIC. However,

in this case the initial few rotations are performed based on an approximation, while the scaling factor is eliminated.

Another crucial difference between the two modes is that in the rotation mode, the direction of rotation is determined by the residual angle. This value tends to relate directly to the angle values utilized here by the relation mentioned in Eq. 11. It is therefore, easier to approximate the magnitude of the required rotation as well, from the value of the residue. However, in the case of vectoring the direction is determined by the Y co-ordinate, while in the extended vectoring it is determined by the difference in target and co-ordinate values. Here, although the sign of these variables provide us the direction of rotation required, an estimate of the magnitude is difficult to obtain, since there is no direct relation between these variables and the set of angles used. Therefore, the scope of accelerating convergence by taking into account the magnitude of rotation as well as the direction, as was done in the rotation mode, is limited in this case.

$$\begin{aligned} T_{i+1} &= T_i + T_i \cdot 2^{-2i} \\ D_{i+1} &= X_{i+1} - T_{i+1} \quad (Cosine\ Matching) \\ D_{i+1} &= Y_{i+1} - T_{i+1} \quad (Sine\ Matching) \end{aligned} \tag{15}$$

Here, we define two new variables D and T which are computed as stated by the Eq. 15. The rest of the variables are the same as defined in Eq. 3. In the proposed algorithm, the representation of D is the same as that of W in the rotation mode, as shown in Fig. 2. The other variables i.e. X, Y, W and T can be represented using a carry save representation. This is because the value of D would be used to determine the direction and angle of rotation, while there is no such dependency on the other variables. Therefore, the non-redundant value of D after any rotation in the algorithm can be computed by the Eq. 16.

$$D = \sum_{i=0}^{n-1} D_i \times 2^{p((i/n)-1)} \tag{16}$$

3.2.1 Rotation Direction and Angle Determination

Similar to the rotation mode, each iteration here as well, takes multiple steps. Although, while it was essentially a p/n bit operation to perform a rotation in the rotation mode, here it is a $3p/n$ bit operation for every rotation that introduces a scaling factor. Whereas, the rotations that do not introduce a scaling factor and thus, do not need a double rotation, can be performed as a p/n bit operation. To keep the time period of the input clock similar to that of the rotation mode, here, three iterations are used to reduce each set D_i to zero for all sets D_{n-j} where $j < n/2$. This is because only $p/2$ angles from the entire set of predefined angles, introduce a scaling as shown in Section 3.1.2. The number of iterations requiring the double rotations is $3\lceil n/2 \rceil$; while, the number not requiring double rotations is $\lfloor n/2 \rfloor$. Therefore, the total iteration count for the proposed algorithm is $3\lceil n/2 \rceil + \lfloor n/2 \rfloor$.

As mentioned above, it is difficult to approximate the magnitude of rotation required for the extended vectoring mode. Therefore, for the first $3\lceil n/2 \rceil$ iterations, the rotations are performed sequentially and are predefined, as was the case with the conventional algorithm. However, a double rotation is performed since the scaling factor introduced due to it can be easily estimated using a shift and add operation. Therefore, for these iterations the direction is determined by the sign of the higher valued non-zero set D_i and the magnitude of rotations are predefined.

For the remaining $\lfloor n/2 \rfloor$ iterations, since the accumulated angle would have converged to a value close to the required value, the relation stated in Eq. 17 can be used to also determine an approximate magnitude of the rotation to be performed. This relation only shows the case for cosine matching, but a similar case for sine matching can also be shown. Therefore, we can get an estimate of the magnitude of rotation

to be performed once the angle accumulated in W is close enough to the required θ. However, it requires the value of cosec of W and requires a multiplication operation. Here, we use the sectioning approach. Since the set of angles to be used is predefined, we do not require an exact value or even an estimate of cosec, rather we can use this relation just by having boundaries as to which power of two, the value of cosec is closest to. We could define obscure boundaries for the same, which can be realized by simple bit examination. Therefore, this can be implemented without the use of a multiplier and does not require additional storage either. Therefore, for these iterations the direction of rotation is determined in a similar fashion as described above; however, the magnitude of rotation is also estimated.

$$\begin{aligned} D &= cosW - cos\theta \\ &= 2 \cdot sin((\theta + W)/2) \cdot sin((\theta - W)/2) \\ &\approx 2 \cdot sin(\theta) \cdot ((\theta - W)/2) \\ D \cdot cosec(\theta) &\approx \theta - W \end{aligned} \tag{17}$$

3.2.2 Scaling Factor Estimation

As mentioned above, the scaling factor in this case need not be compensated, but the target needs to be updated with an equivalent scaling. This is done using the double rotation technique. Although, this technique was also used in the rotation mode for the first iteration, the purpose of using the same is different in the two cases. In the rotation mode, double rotation enables the algorithm to skip a rotation about an angle, while still introducing the scaling factor attributed to it. Here, double rotations are performed because it is easier to estimate the scaling factor. The scaling factor introduced due to a rotation of $tan^{-1}2^{-k}$ is $(cos(tan^{-1}2^{-r}))^{-1}$. On performing double rotations, this scaling factor can be estimated as shown in Eq. 18. Therefore, we perform double rotations while the target T is also updated to account for this

Algorithm 5: Super-CORDIC Extended Vectoring (T)

```
Initialize values of D[n − 1 : 0]
Load_Initial_Vector
for j = 0 to 3⌈n/2⌉ do
  for rot = 1 to p/3n do
    Direction_determination(D)
    D = Modify_residue(D)
    Perform_double_rotations
  end for
  D = Overflow_handle(D)
  break
end for
for j = 0 to ⌊n/2⌋ do
  for rot = 1 to p/n do
    Angle_and_Direction_determination(D)
    D = Modify_residue(D)
    Perform_rotations
  end for
  W = Overflow_handle(W)
  break
end for
```

Figure 7: Pseudo-Code for the proposed Super-CORDIC Extended Vectoring architecture

scaling factor. Thus, the comparison between the co-ordinate and the target has a constant scaling in both the terms.

$$\begin{aligned} K &= (cos(tan^{-1}2^{-r}))^{-1} \quad (Single\ Rotation) \\ K &= \sqrt{1+2^{-2r}} \\ K^2 &= 1+2^{-2r} \quad (Double\ Rotation) \end{aligned} \tag{18}$$

3.2.3 Architectural Implementation

The input target value is placed into the standard bits of D, while the redundant bits are set to zero. During the first iteration, a known initial vector is loaded and rotations are performed on this vector till the value of the required co-ordinate matches the input target value. While doing so, the angle accumulated over the rotations in register W is the inverse cosine/sine of the input target value and is output at the

end of the algorithm. Rotations are performed by determining the direction and magnitude of rotation as described in the previous section. In each of the initial $3\lceil n/2 \rceil$ iterations, $p/3n$ double rotations are performed. While doing so, the target value is also updated to reflect the accumulated scaling and W is updated to reflect the angle of the current vector. Finally, the register D is computed as the difference between the corresponding co-ordinate and the target value and is used further for the next rotation.

In the remaining $\lfloor n/2 \rfloor$ iterations, a variable number of rotations are performed in each iteration ranging from 0 to p/n. The rotations performed in these iterations do not contribute any scaling factor and thus, there is no need to update the target value. Therefore, each of these rotations is a p/n bit operation. The magnitude of rotations for these iterations are determined by the relation presented in Eq. 17. Here the update of the D register is also performed with sign compensation to reduce the effect of redundancy. Also, at the end of each of these iterations the overflow handling has also been employed similar to the rotation mode. The entire algorithm is shown by the pseudo-code in Fig. 7.

The total number of redundant bits required in D would be the same as was the case with the rotation mode, since it accounts for the worst case. The domain of this algorithm has been restricted to $[0, 1]$ to avoid the need additional hardware to detect sign changes. The output range of the algorithm is mapped onto $[0, \pi/2]$. The inverse of negative values of sine and cosine can be easily computed by mapping them onto the domain of this algorithm. It can also be noted that since $cosec(\pi/2 - \theta) = sec(\theta)$, the number of values required for approximations of the magnitude of rotation based on Eq. 17, remain the same and do not need to be doubled to incorporate the computation of sine inverses. The total computation time of this algorithm is $(p/3n)((3p/n) + 1)(3\lceil n/2 \rceil) + (p/n)((p/n) + 1)(\lfloor n/2 \rfloor)$. In comparison to the original algorithm with each iteration as a p bit operation, the effective iteration count here is

Table 3: Example of extended vectoring for $p = 24, n = 4$ and input cos 25

Residue (D)	X	Y	W	T	Rotation
0x17FC35	0xFFFFFF	0	0	0xE803CA	
0xFF97BF9	0xF00000	0x7FFFFE	0x7D6DD7	0xF68406	14.036
0x11E1E7	0x10C3FFE	0x420000	0x3DC262	0xFA5E15	-7.125
0x79B4D	0x102F3C0	0x6345FE	0x5DB7BD	0xFB5872	3.576
0xE75D	0xFC7EA5	0x735C67	0x6DB667	0xFB9747	1.789
0xFFFD2CFB	0xF8D3FC	0x7B3925	0x75B63C	0xFBA700	0.895
0xFFFF120C	0xFABCFB	0x7753EA	0x71B642	0xFBAAEE	-0.447
0xFFFFFFBC	0xFBAAA6	0x755DFB	0x6FB643	0xFBABE9	-0.223
0x739B	0xFC1FC3	0x746235	0x6EB644	0xFBAC27	-0.111
0x394D	0xFBE584	0x74E03C	0x6F3643	0xFBAC36	5.595e-2
0x1C11	0xFBC84A	0x751F32	0x6F7642	0xFBAC39	2.797e-2
0xD6E	0xFBB9A7	0x753EA8	0x6F9641	0xFBAC39	1.398e-2
0x61C	0xFBB255	0x754E62	0x6FA640	0xFBAC39	6.994e-3
0xFFFFFEC9	0xFBAB02	0x755E1B	0x6FB63F	0xFBAC39	1.398e-2
0x9D	0xFBACD6	0x755A2F	0x6FB240	0xFBAC39	-3.497e-3
0xFFFFFFB5	0xFBABED	0x755C25	0x6FB43F	0xFBAC39	1.748e-3
0x28	0xFBAC61	0x755B2C	0x6FB340	0xFBAC39	-8.742e-4
0xFFFFFFEE	0xFBAC27	0x755BA8	0x6FB3BF	0xFBAC39	4.371e-4
0x9	0xFBAC42	0x755B6A	0x6FB380	0xFBAC39	-2.185e-4
0xFFFFFFFD	0xFBAC36	0x755B88	0x6FB39F	0xFBAC39	1.092e-4
0xFFFFFFFE	0xFBAC38	0x755B82	0x6FB398	0xFBAC39	-2.732e-5

$(1/3n)((3p/n) + 1)(3\lceil n/2 \rceil) + (1/n)((p/n) + 1)(\lfloor n/2 \rfloor)$. For the ideal value of $n = 4$, the effective iteration count evaluates to $(p/2) + 1$.

An example of the algorithm has been presented in Table 3.2.3. It can be seen that during the first two iterations, double rotations are performed with the rotation magnitude being predetermined. It can also be seen that the the target value T is also scaled with each rotation to ensure the scaling in the target and the co-ordinates is the same. Also, this scaling has no effect after the first two iterations and therefore, double rotations are not required in the following iterations. In contrast to the rotation mode, the value of the co-ordinates may take values greater than 1 as shown in the first rotation of the first iteration in the given example. This is again due to the scaling and therefore, one extra bit is required to represent these values.

In the third and final iteration, the rotations are accelerated using the magnitude approximation presented in Eq. 17. It can be seen that, as the value of X approaches the value of T i.e. as the value of D reduces, the result is converged upon in W. The error in the final result of this example is 1.8206×10^{-6}.

CHAPTER IV

PERFORMANCE ANALYSIS

4.1 Rotation Mode

The rotation mode of the proposed algorithm was simulated for different values for p, n and (p/n). The error in the results obtained is as shown in tables 4, 5 and 6. These values are based on 10000 samples evenly distributed over the range $[0, \pi/4]$. As described above, greater the value of n, lower is the overall latency, although, higher is the error in the result. It can be seen that for a constant value of p/n, increasing the number of precision bits p, does not do much benefit. Also, the results for the optimal value of $n = 4$ show the minimum error. On decreasing the value of n to 3, it can be seen that there is a slight decrease in error. However, here the hardware cost as well as the overall latency would increase considerably. On increasing the domain of the rotation mode of the algorithm, the results obtained are as shown in Table 7. These values are based on 10000 samples evenly distributed over the range $[0, 2\pi]$. For this mode of the algorithm, the average expected error would be around 2^{-p}, assuming there is no scaling error. However, a slightly higher absolute error is obtained due to truncation errors.

Table 4: Error values for rotation mode with constant p/n = 4

Error	P = 12	P = 16	P = 20	P = 24	P = 28
Avg. in X	1.898e-4	1.440e-5	3.319e-6	3.287e-6	3.286e-6
Avg. in Y	2.617e-4	2.078e-5	2.116e-6	1.396e-6	1.365e-6
Max. in X	8.264e-4	6.422e-5	1.247e-5	1.014e-5	1.006e-5
Max. in Y	1.245e-3	8.887e-5	1.008e-5	6.720e-6	6.657e-6

Table 5: Error values for rotation mode with constant n = 4

Error	P = 12	P = 16	P = 20	P = 24	P = 28
Avg. in X	1.888e-4	1.440e-5	1.055e-6	8.092e-8	5.210e-9
Avg. in Y	2.558e-4	2.078e-5	1.637e-6	1.055e-7	8.200e-9
Max. in X	9.231e-4	6.422e-5	5.449e-6	3.900e-7	2.529e-8
Max. in Y	3.101e-3	8.887e-5	8.671e-6	4.466e-7	3.476e-8

Table 6: Error values for rotation mode with constant p = 24

Error	N = 8	N = 6	N = 4	N = 3
Avg. in X	5.904e-5	3.287e-6	8.092e-8	7.298e-8
Avg. in Y	2.944e-5	1.396e-6	1.055e-7	8.620e-8
Max. in X	8.667e-4	1.014e-5	3.900e-7	3.568e-7
Max. in Y	3.021e-3	6.720e-6	4.466e-7	4.246e-7

4.2 Extended Vectoring Mode

This mode of the proposed algorithm does not provide as much flexibility in terms of designing for speed or accuracy. Since the initial $p/2$ rotations have to be performed sequentially and cannot be accelerated, the choice of the value of n would have to be greater than or equal to 4. The algorithm was simulated and tested for $p = 24$ and $n = 4$ for 10000 samples evenly distributed over the range $[0, \pi/4]$. The average error obtained was 1.411×10^{-5}, while the maximum error obtained was 9.049×10^{-4}. For this mode of the algorithm, the average expected error would be around $2^{-p/2}$. Again, a slightly higher absolute error is obtained due to truncation errors.

Table 7: Error values for rotation mode with domain extension for constant p = 24

Error	N = 6	N = 4	N = 3	N = 2
Avg. in X	8.378e-5	1.299e-6	1.643e-7	2.179e-7
Avg. in Y	8.377e-5	1.290e-6	1.650e-7	2.221e-7
Max. in X	2.606e-4	4.406e-6	7.346e-7	8.275e-7
Max. in Y	2.606e-4	4.428e-6	7.484e-7	8.305e-7

CHAPTER V

CONCLUSION

The Super-CORDIC rotation mode algorithm presents a generalized algorithm to design a CORDIC processor based on the requirements in terms of hardware cost, overall latency and absolute error in the output. The value of the design parameter n can be chosen to meet required performance. The higher the value, the higher is the error in the final result and lower is the latency. For the proposed ideal value of design parameter $n = 4$, the Super-CORDIC presents an rotation mode algorithm that has no scaling error and an effective iteration count of $(p/n) + 1$. This iteration count is lower than any existing rotation mode CORDIC system. The hardware cost of the algorithm would however, be higher than the original algorithm but lower than the other redundant CORDIC systems. The only limitation on the value of n is that $p/n \geq 4$, so that the double rotations of the first iteration do not add additional latency.

The extended vectoring mode of the algorithm of the Super-CORDIC has been optimized to provide the result in $3\lceil n/2 \rceil + \lfloor n/2 \rfloor$ iterations. Therefore, the effective iteration count for the ideal case of $n = 4$ is $p/2 + 1$. The choice of the design parameter n has more strict limitations here and can be only chosen such that $n \leq 4$. However, it is possible to design a Super-CORDIC processor for both modes, with the ideal value of $n = 4$. Therefore, the Super-CORDIC algorithm presents an optimized low latency algorithm to compute the values of trigonometric functions and their inverses.

REFERENCES

[1] Antelo, E., Villalba, J., Bruguera, J. D., and Zapata, E. L., "High performance rotation architectures based on the radix-4 cordic algorithm," *Computers, IEEE Transactions on*, vol. 46, no. 8, pp. 855–870, 1997.

[2] Daggett, D., "Decimal-binary conversions in cordic," *Electronic Computers, IRE Transactions on*, no. 3, pp. 335–339, 1959.

[3] Ercegovac, M. D. and Lang, T., "Redundant and on-line cordic: Application to matrix triangularization and svd," *Computers, IEEE Transactions on*, vol. 39, no. 6, pp. 725–740, 1990.

[4] Hsiao, S.-F., Hu, Y.-H., and Juang, T.-B., "A memory-efficient and high-speed sine/cosine generator based on parallel cordic rotations," *Signal Processing Letters, IEEE*, vol. 11, no. 2, pp. 152–155, 2004.

[5] Hu, Y. H. and Naganathan, S., "An angle recoding method for cordic algorithm implementation," *Computers, IEEE Transactions on*, vol. 42, no. 1, pp. 99–102, 1993.

[6] Hu, Y. H. and Wu, Z., "An efficient cordic array structure for the implementation of discrete cosine transform," *Signal Processing, IEEE Transactions on*, vol. 43, no. 1, pp. 331–336, 1995.

[7] Lang, T. and Antelo, E., "Cordic-based computation of arccos and arcsin," in *Application-Specific Systems, Architectures and Processors, 1997. Proceedings., IEEE International Conference on*, pp. 132–143, IEEE, 1997.

[8] Mazenc, C., Merrheim, X., and Muller, J.-M., "Computing functions cos^{-1} and sin^{-1} using cordic," *IEEE Transactions on Computers*, vol. 42, no. 1, pp. 118–122, 1993.

[9] Meher, P. K., Valls, J., Juang, T.-B., Sridharan, K., and Maharatna, K., "50 years of cordic: Algorithms, architectures, and applications," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 56, no. 9, pp. 1893–1907, 2009.

[10] Rodrigues, T. K. and Swartzlander Jr, E. E., "Adaptive cordic: Using parallel angle recoding to accelerate rotations," *Computers, IEEE Transactions on*, vol. 59, no. 4, pp. 522–531, 2010.

[11] Takagi, N., Asada, T., and Yajima, S., "Redundant cordic methods with a constant scale factor for sine and cosine computation," *Computers, IEEE Transactions on*, vol. 40, no. 9, pp. 989–995, 1991.

[12] Volder, J. E., "The cordic trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. 8, pp. 330–334, Sept. 1959.

[13] Walther, J. S., "A unified algorithm for elementary functions," in *Proceedings of the May 18-20, 1971, spring joint computer conference*, pp. 379–385, ACM, 1971.

[14] Wang, S., Piuri, V., and Swartzlander Jr, E. E., "Hybrid cordic algorithms," *IEEE Transactions on Computers*, no. 11, pp. 1202–1207, 1997.

[15] Wu, C.-S. and Wu, A.-Y., "Modified vector rotational cordic (mvr-cordic) algorithm and architecture," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 48, no. 6, pp. 548–561, 2001.

[16] Wu, C.-S., Wu, A.-Y., and Lin, C.-H., "A high-performance/low-latency vector rotational cordic architecture based on extended elementary angle set and trellis-based searching schemes," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 50, no. 9, pp. 589–601, 2003.

[17] Yu, S. and Swartzlander Jr, E. E., "A scaled dct architecture with the cordic algorithm," *Signal Processing, IEEE Transactions on*, vol. 50, no. 1, pp. 160–167, 2002.