CORDIC Vectoring with Arbitrary Target Value *

Tomás Lang
Dept. Electrical and Computer Eng.
University of California at Irvine
Irvine CA. USA
tomas@ece.uci.edu

Elisardo Antelo
Dept. Electrónica e Computación
Universidade de Santiago de Compostela
15706 Santiago de Compostela. SPAIN.
elisardo@usc.es

Abstract

The computation of additional functions in the CORDIC module increases its flexibility. We consider here the extension of the vectoring mode (angle calculation) so that the vector is rotated until one of the coordinates (for instance y) attains a target value t (in contrast to the value 0, as in standard vectoring). The main problem in the algorithm is that the modulus of the vector is scaled in each CORDIC iteration so that a direct comparison of y[j] with t does not assure convergence. We present a scheme that overcomes this and in which the implementation consists of a standard CORDIC module plus a module to determine the direction of rotation. This improves over a previous proposal in which more complex iterations are introduced as part of the CORDIC algorithm.

1. Introduction

The computation of additional functions in the CORDIC module increases its flexibility. Here we consider an extension of the CORDIC vectoring mode. Specifically, as shown in Figure 1, instead of rotating the vector (x_a, y_a) until one of the coordinates (for example y) is zero, the rotation is done until y attains a target value t, with the restriction $|t| \leq M$ with $M = \sqrt{x_a^2 + y_a^2}$. This extension is considered in [6], where it is shown that using a standard CORDIC module it would require three vectorizations. The applications mentioned are inverse kinematics computations and the solution of equations of the type $b\cos\theta \pm a\sin\theta = c$. CORDIC modules have been previously proposed for this application in [4] [9] [11].

The modified vectoring mode also allows the computation of $\sqrt{a^2 + b^2 - c^2}$ and the angle between two vectors of the same modulus. In this latter case, using a standard CORDIC module would require two vectoring operations and one subtraction. Finally, arcsin and arcos are special cases. CORDIC–based algorithms for these functions have been proposed in [10] and [7].

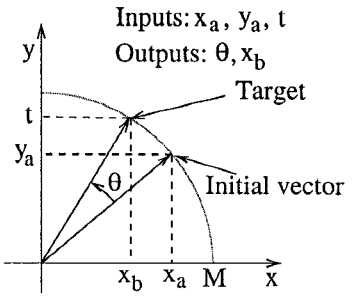

Figure 1. Extended CORDIC vectoring with target t.

In [2] a method to compute elementary functions is proposed in which the control variable converges to a constant value, dependent on the function to be computed. The method is applied to the computation of exponentials, logarithms, ratios, and square–root. In all these cases, the constant value is either zero or one. Later, in [1] it was shown that applying this method to complex numbers results in CORDIC algorithms. However, the convergence constant is also zero.

To simplify the description we consider that the initial vector corresponds to the first quadrant and $t \geq 0$. The generalization to other cases is straight–forward. A more detailed description of the algorithms presented in this work can be found in [8].

Consider first an "ideal" rotation–based algorithm for this extended vectoring. In such a case, the initial vector (x_a, y_a) is rotated until $y = t$ (see Figure 2(a)). If the rotation is obtained by a sequence of microrotations of primitive angles α_j, and σ_j is the direction of rotation in iteration j,

*E. Antelo was supported in part by the Ministry of Education and Science (CICYT) of Spain under contract TIC96-1125

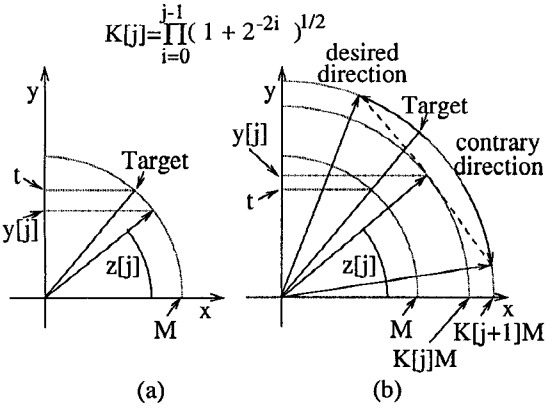

Figure 2. Rotation–based algorithm (a) and Effect of scale factor on the direction of microrotation (b)

then

$$x[0] = x_a \;\; y[0] = y_a \;\; z[0] = 0$$

and

$$\sigma_j = \begin{cases} -1 & \text{if } (y[j] \geq t) \text{ OR } (x[j] \leq 0) \\ 1 & \text{otherwise} \end{cases} \qquad (1)$$

$$(x[j+1], y[j+1]) = ROT[(x[j], y[j]), \sigma_j \alpha_j]$$

$$z[j+1] = z[j] + \sigma_j \alpha_j$$

The angle θ is accumulated in z. Moreover, the final value of coordinate x is $x_b = \sqrt{x_a^2 + y_a^2 - t^2}$.

Now consider the use of the CORDIC algorithm [6]. In this algorithm, in iteration j the microrotation is by an angle $\pm \tan^{-1}(2^{-j})$ and the modulus of the vector is scaled by the factor $(1 + 2^{-2j})^{1/2}$. As a result of this scaling, obtaining the direction of rotation by expression (1) does not result in a convergent algorithm, as illustrated in Figure 2(b). To obtain a convergent algorithm, the scaling can be accounted in one of three ways:

1. Initializing to the vector (x_a, y_a) and determining the direction of rotation by comparing $y[j]$ with $K[j]t$, where $K[j]$ is the scaling of the modulus introduced up to iteration j. This approach is used in [10] for the special case of calculating arcos and arcsin. However, the determination of $K[j]t$ is complicated, so double CORDIC iterations are used to utilize $K^2[j]$ instead of $K[j]$. This either complicates the implementation of the iteration or doubles the number of cycles. Moreover, the additional recurrence to compute $K^2[j]t$ is implemented with an adder and a variable shifter.

2. Initializing to the vector (x_a, y_a) and compensating the scaling factor in each iteration so that the compensated $y[j]$ is compared with t. This is used in [6]. Again, to simplify the scaling, microrotations with double shifts and repetitions are performed, leading to a CORDIC module which is more complex than the standard CORDIC [1].

3. Initializing the vector to $(x_a/K, y_a/K)$, where K is the scaling introduced by all iterations, and determining the direction of rotation by comparing $y[j]$ with $(K[j]/K)t$. This is the approach we present here. As discussed later, this method is advantageous because it is easy to obtain a reasonable approximation of $(K[j]/K)t$. To assure convergence, we combine an approximation that is simple to implement with a few repetitions.

We present the algorithm, determine a suitable approximation of $(K[j]/K)t$ and the location of the required repetitions, and show an implementation. In contrast to the previous proposals, this implementation consists of a standard CORDIC module plus another module to determine the direction of rotation. It has the following characteristics:

- The CORDIC module can be used for the standard CORDIC functions.
- The iteration time remains the same as that of the standard CORDIC module.
- The number of iterations is practically the same as that of the standard CORDIC algorithm. This is true when the compensation of the scaling factor in the standard CORDIC is done by a combination of scaling iterations and repetitions (as described in [3] [5] among others), which is one of the preferred methods because the compensation can be performed with the same hardware as the microrotations.
- The additional module is simple.

The special case of computing arcos and arcsin was considered in [7]. Although the general strategy is the same, the special case allows simplifications, as discussed further in Section 5.

2. Algorithm

As outlined in the Introduction, the **ideal** CORDIC–based algorithm we propose is as follows (for the first quadrant):

[1] The actual algorithm presented in [6], in which the compensation of $y[j]$ is done by scalings and repetitions, has cases for which it does not converge.

Initial condition

$$x[0] = \frac{x_a}{K} \quad y[0] = \frac{y_a}{K} \quad z[0] = 0$$

where K is the scaling factor introduced by all microrotations of the actual CORDIC algorithm.

Note that this scaling operation has to be performed also in the standard CORDIC algorithm, so no additional overhead is required. It can be done, for example, by a sequence of scaling iterations using the same hardware as the microrotations. Moreover, repetitions can be introduced so that the total number of scaling iterations and repetitions is minimized.

Iteration. The iteration consists of a CORDIC step with the value of σ_j determined by the result of the comparison of $y[j]$ with $R[j]t$ (see Figure 3(a)), where $R[j] = K[j]/K$ and $K[j]$ is the scaling up to iteration j. That is,

$$\sigma_j = \begin{cases} -1 & \text{if } (y[j] \geq R[j]t \geq 0) \\ 1 & \text{otherwise} \end{cases} \qquad (2)$$

$$x[j+1] = x[j] - \sigma_j 2^{-j} y[j]$$
$$y[j+1] = y[j] + \sigma_j 2^{-j} x[j]$$
$$z[j+1] = z[j] + \sigma_j \tan^{-1}(2^{-j})$$

At the end, $y[n+1] = t$ and the angle is $z[n+1] = \theta$. Moreover, $x[n+1] = \sqrt{x_a^2 + y_a^2 - t^2}$.

Precision and range. We consider the case in which the input operands x_a and y_a are positive fractions with n bits, and $0 \leq t \leq M \leq \sqrt{2}$ has n fractional bits. The results are also given with n fractional bits. The range of the angle is $[-\pi/2, \pi/2]$. Note that because of the characteristics of the arcsin function, the fact that the operands are given with n fractional bits reduces the number of significant bits of the angle in the region close to $\pi/2$. This effect is considered further in [7] for the arcos case.

2.1. Practical algorithm with an approximation of R[j] and repetitions

The implementation of the ideal algorithm proposed is complicated because of the need to compute $R[j]t$. To avoid the exact calculation, we use a combination of two approaches, as follows:

1. Use an approximation $A[j]t$ of $R[j]t$. That is, the direction of rotation is determined by

$$\sigma_j = \begin{cases} -1 & \text{if } (y[j] \geq A[j]t \geq 0) \\ 1 & \text{otherwise} \end{cases} \qquad (3)$$

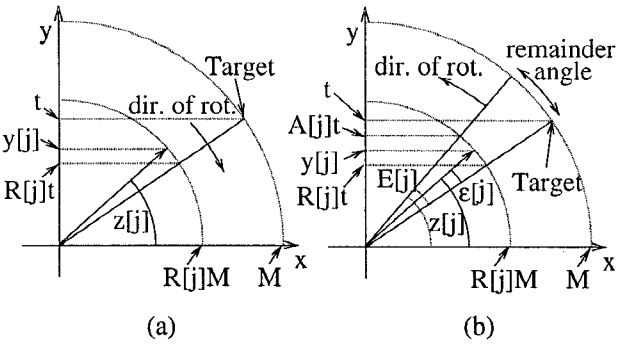

Figure 3. Ideal algorithm (a) and using approximation (b)

2. Introduce repetitions of some CORDIC steps.

When the comparison is done using the approximation $A[j]t$ instead of the exact value $R[j]t$, the situation is similar to the case described in Figure 2(b) but the region in which the contrary direction occurs is reduced. Figure 3(a) describes the ideal algorithm and Figure 3(b) shows the situation when using the approximation. To have convergence, the remainder angle $\varepsilon[j]$ has to be compensated by subsequent iterations. We now consider the following items:

- Determine the maximum remainder angle in iteration j produced by using the approximation instead of the exact value. This is indicated by $E[j]$ in the Figure.
- Determine a condition on this remainder to assure convergence.
- Propose an approximation and repetitions so that the convergence condition is satisfied. The approximation should be simple to implement and the number of repetitions small.

2.1.1. Maximum remainder in iteration j

We now determine the maximum remainder angle $E[j]$. We first consider this maximum for a given $z[j]$ and then determine the overall maximum in the range $0 \leq z[j] \leq \pi/2$.

We consider separately the cases $A[j] \geq R[j]$ and $A[j] < R[j]$.

1. $F[j] = \frac{R[j]}{A[j]} \leq 1$. As shown in Figure 3(b), the maximum remainder for a given $z[j]$ is

$$E[j] = z[j] - \sin^{-1}(\frac{t}{M})$$

This maximum value corresponds to $y[j] = A[j]t$, so

$$E[j] = z[j] - \sin^{-1}(\frac{y[j]}{A[j]M})$$

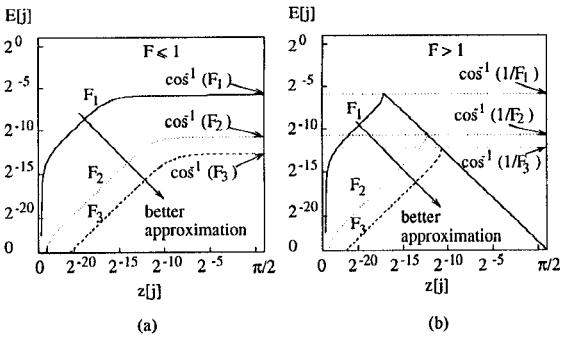

Figure 4. Variation of the remainder angle with z[j] for different values of F[j] (example for j=3).

Moreover, the CORDIC iterations produce $y[j] = R[j]M \sin z[j]$ so that

$$E[j] = z[j] - \sin^{-1}(F[j] \sin z[j])$$

This occurs for $t = F[j]M \sin z[j]$.

Now consider the maximum remainder in the range $0 \leq z[j] \leq \pi/2$ (note that for $z[j] < 0$ no contrary microrotation can occur). Since for $F[j] \leq 1$ we have $F[j] \sin z[j] \leq 1$, the maximum remainder is produced for $z[j] = \pi/2$ so that

$$E[j]_{max} = \pi/2 - \sin^{-1}(F[j]) = \cos^{-1}(F[j])$$

as illustrated in Figure 4(a).

2. $F[j] = \frac{R[j]}{A[j]} > 1$. Since now $F[j] \sin(z[j])$ can be larger than 1, the resulting remainder is

$$E[j] = \sin^{-1}(\min(F[j] \sin(z[j]), 1)) - z[j]$$

Consequently, for this case ($F[j] > 1$), as $z[j]$ varies in the range $[0, \pi/2]$, Figure 4(b) shows that for $z[j] \leq \sin^{-1}(1/F[j])$ the behavior is similar to that of the previous case and thereafter it is linear. The maximum value is for angle $\pi/2 - \cos^{-1}(1/F[j])$ and

$$E[j]_{max} = \cos^{-1}(1/F[j])$$

Combining both cases above we get

$$E[j]_{max} = \cos^{-1}(\min(F[j], 1/F[j])) \quad (4)$$

2.1.2. Condition for convergence

The remainder angle has to be compensated by subsequent iterations. Consider the case in which the direction of rotation in iteration j is contrary to that which would be produced when comparing with $R[j]t$. The maximum

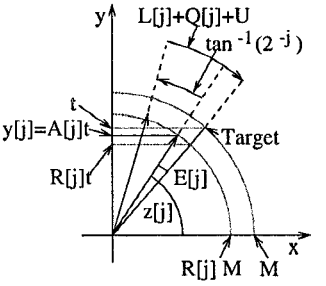

Figure 5. Condition for convergence.

remainder before the microrotation j is $E[j]_{max}$ and the rotation angle in iteration j is $\tan^{-1}(2^{-j})$. Since this rotation is in the contrary direction, the total angle that has to be compensated is $E[j]_{max} + \tan^{-1}(2^{-j})$. This compensation is obtained as a result of two types of microrotations: basic CORDIC iterations (that is, one iteration for each j) and repetitions. The angle for the first type is

$$L[j] = \sum_{i=j+1}^{n} \tan^{-1}(2^{-i})$$

since in the worst case after a microrotation in the contrary direction, all subsequent microrotations are in the correct direction.

Similarly, for the repetitions,

$$Q[j] = \sum_{i \in REP[j]} \tan^{-1}(2^{-i}) \quad (5)$$

where $REP[j]$ is the set of indices of the repetitions after iteration j. Therefore, as shown in Figure 5, the condition for convergence is

$$E[j]_{max} \leq L[j] + Q[j] - \tan^{-1}(2^{-j}) + U \quad (6)$$

where U is an upper bound on the final error in the angle. Substituting $E[j]_{max}$ we get

$$\cos^{-1}(\min(F[j], 1/F[j])) \leq L[j] + Q[j] - \tan^{-1}(2^{-j}) + U \quad (7)$$

Note that the right-hand side is positive because of two reasons:

1. $L[j] > \tan^{-1}(2^{-j})$ since $\tan^{-1}(2^{-i}) < 2\tan^{-1}(2^{-(i+1)})$.
2. The introduction of repetitions.

For $j = n$, a microrotation in the contrary direction can occur, without further compensation. Consequently, the bound on the final error is

$$U = \cos^{-1}(\min(F[n], 1/F[n])) + \tan^{-1}(2^{-n})$$

2.1.3 Appropriate approximation and repetitions

We now determine a suitable approximation and the required repetitions so that the convergence condition (7) is satisfied. There is a tradeoff between the closeness of the approximation (and therefore, its complexity of implementation) and the number of repetitions. Namely, if the approximation is sufficiently close, no repetitions are needed whereas for a "coarser" approximation more repetitions are required. Consequently, we are concerned with a suitable tradeoff between these two aspects.

Since $A[j]t$ tends to t we define $B[j]$ such that $A[j]t = t - B[j]$ and we implement the iteration

$$\begin{aligned} B[j+1] &= 2^{-b}B[j] + C[j] \\ C[j+1] &= 2^{-c}C[j] \end{aligned}$$

in which b and c are constants to be determined. As we show in Section 4.1 this iteration is simple to implement.

So let us now determine an approximation of this type. From the definition of $R[j]$ we have

$$A[j] \approx R[j] = \frac{K[j]}{K} = \prod_{i=j}^{n}(1+2^{-2i})^{-1/2} \prod_{i \in REP[j]} (1+2^{-2i})^{-1/2}$$

where, as before, $REP[j]$ is the set of indices for the repetition iterations after step j.

For the approximation, we consider only the first product. That is, we develop an approximation which is not dependent on the repetitions. This simplifies the approximation and, as we see later, still allows convergence with a small number of repetitions.

Taking the first three terms of the Taylor series expansion of $(1+2^{-2i})^{-1/2}$, we get

$$R[j] \approx \prod_{i=j}^{n}(1 - (1/2)2^{-2i} + (3/8)2^{-4i})$$

Keeping only the terms which are cross-multiplied with 1, we get

$$R[j] \approx 1 - (1/2)2^{-2j}\sum_{k=0}^{n-j} 2^{-2k} + (3/8)2^{-4j}\sum_{k=0}^{n-j} 2^{-4k}$$

Since $\sum_{k=0}^{n-j} 2^{-4k} \approx 1$, we get

$$R[j] \approx 1 - (1/2)2^{-2j}[\sum_{k=0}^{n-j} 2^{-2k} - (3/4)2^{-2j}]$$

For the implementation form discussed above, we want to have an approximation with only the first two terms. However, since the third term has different sign than the second and has relative weight 2^{-2j}, instead of just eliminating the third term, a better approximation is

$$A[j] = 1 - (1/2)2^{-2j}\sum_{k=0}^{min(j,n/2)} 2^{-2k}$$

Moreover, with this approximation we obtain

$$B[j] = (2^{-2j-1}\sum_{k=0}^{min(j,n/2)} 2^{-2k})t$$

so that the iteration is

$$B[j+1] = \begin{cases} 2^{-2}B[j] + 2^{-5}2^{-4j}t & \text{if } j < n/2 \\ 2^{-2}B[j] & \text{otherwise} \end{cases}$$

To avoid a variable shifter we implement

$$\begin{aligned} B[j+1] &= \begin{cases} 2^{-2}B[j] + C[j] & \text{if } j < n/2 \\ 2^{-2}B[j] & \text{otherwise} \end{cases} \\ C[j+1] &= 2^{-4}C[j] \end{aligned}$$

The initial conditions are $B[0] = (1/2)t$ and $C[0] = (1/32)t$.

Now we need to determine the repetitions required for convergence. The introduction of repetitions has two contradictory effects: it aids towards convergence because it increases Q (see expression (5)), but it might go against convergence because it might make the approximation worse. The application of the following procedure shows that, for the approximation chosen, the first effect prevails.

Since for convergence at a particular iteration index j, only the repetitions with index larger than j have an effect (both on the quality of the approximation and on the value Q), we begin from $j = n-1$ and advance towards $j = 0$. For each j, we determine whether the condition of convergence is satisfied (including the effect of the repetitions already placed for $i > j$). If the condition is not satisfied, we include a repetition with index $j+1$. This will satisfy the convergence for index j and also might help for convergence for smaller values. This process produces the minimum number of repetitions. It is formalized by the following algorithm, in which the meaning of the variables is as defined before.

1. **Initial values:**
   $R[n] = (1+2^{-2n})^{-1/2}$, $F[n] = R[n]/A[n]$
   $L[n] = 0$, $Q[n] = 0$.
2. From $j = n-1$ to $j = 0$
   - **Actualizations:**

$$\begin{aligned} R[j] &= \frac{R[j+1]}{\sqrt{1+2^{-2j}}} \\ F[j] &= R[j]/A[j] \\ L[j] &= L[j+1] + \tan^{-1}(2^{-(j+1)}) \\ Q[j] &= Q[j+1] \\ E[j]_{max} &= \cos^{-1}(\min(F[j], 1/F[j])) \end{aligned}$$

Precision	16	24	32
Repetitions	1, 2, 4, 8	1, 3, 6, 12	1, 2, 4, 8, 16

Table 1. Repetitions for n=16, 24, 32.

- **Test of convergence:**
  if $E[j]_{max} > L[j] + Q[j] - \tan^{-1}(2^{-j}) + U$
  - **Repeat iteration $j+1$.** Add $j+1$ to the set of indices of repetition iterations.
  - **Actualizations:**

$$R[j] = \frac{R[j]}{\sqrt{1+2^{-2(j+1)}}}$$
$$Q[j] = Q[j] + \tan^{-1}(2^{-(j+1)})$$

As seen before, the final microrotation leads to a final error in the angle of $U = E[n]_{max} + \tan^{-1}(2^{-n})$. Thus, to reduce the number of repetitions, in the test for convergence of the previous algorithm we have considered that a remainder of $E[j]_{max} < L[j] + Q[j] - \tan^{-1}(2^{-n}) + \tan^{-1}(2^{-(n-1)})$ leads to a convergent algorithm. Then, the error in the angle is bounded by $U = \tan^{-1}(2^{-(n-1)})$.

Table 1 shows the repetitions obtained for different precisions. Figure 6 shows that the convergence is achieved with these repetitions for the case $n = 24$, since for every j the remainder is not larger than what is required for convergence. On the other hand, the Figure also shows that for the case without repetitions the algorithm does not converge since for $j < 12$ the remainder is larger than what is required for convergence.

As can be seen, the position of the repetitions follows a regular pattern. First of all, it is necessary to repeat iteration 1 because of the approximation in iteration 0. Then, repetitions appear in positions $n/2^i$. From this we conclude the following:

1. Iteration 0 can be eliminated because the rest of iterations, including the repetitions, cover the range $[0, \pi/2]$. This has been already done in the implementations of conventional CORDIC in which the scale factor compensation is performed by scaling and repetitions, as discussed in the next Section. Since we begin in iteration 1 the initial conditions are $x[1] = x_a/K$, $y[1] = y_a/K$, $z[1] = 0$, $B[1] = (5/32)t$ and $C[1] = (1/512)t$. Similarly, the value of K begins with $j = 1$.
2. The required number of repetitions is small.
3. Furthermore, as we show in the next section, there is only a very slight increase in the number of iterations with respect to the conventional CORDIC in which repetitions and scalings are used to compensate the scale factor.

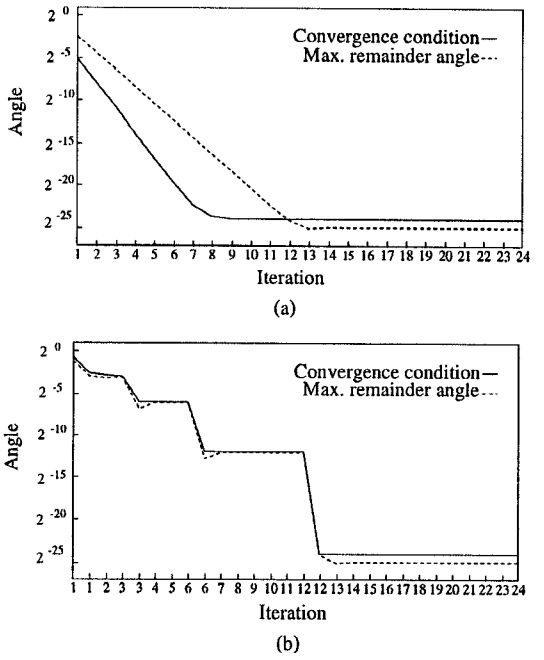

Figure 6. Convergence for n=24 (a) without repetitions (no convergence) (b) with repetitions.

3. Scaling

The initial scaling by $1/K$ is performed by scaling iterations. It is well–known that the combination of scalings and repetitions reduces the number of total iterations ([3] [5] among others). In our case, the repetitions introduced have to assure convergence, but the set of repetitions for this is not unique. Thus, we want to find a sequence of repetitions that both assures convergence and reduces the total number of iterations, including the initial scalings.

We have performed a complete search of the sets of repetitions that produce a convergent algorithm and determined for each set the minimum number of scalings. Table 2 shows the best set of repetitions and scalings obtained for precisions $n = 16$ and $n = 24$. In the same Table we show for the standard CORDIC algorithm the optimal repetitions and scalings obtained in [5], as well as the scalings required without using repetitions. Note that in the case of the extended CORDIC vectoring we need only one more iteration than in the standard CORDIC case.

4. Implementation

We now describe an implementation of the algorithm presented. This can be either word–serial or pipelined. As shown in Figure 7 it consists of the following two blocks:

Algorithm	n	Basic shift seq.	Repetitions	Scalings	Total
Standard CORDIC (no repetitions)	16	0, 1, ..., 16	-	(-,1)(+,2)(-,5)(+,8)(-,10)	22
	24	0, 1, ..., 24	-	(-,1)(+,2)(-,5)(+,8)(-,10) (+,16)(+,18)(+,19)(-,23)	34
Standard CORDIC (with repetitions)	16	0, 1, ..., 16	-	(-,1)(+,2)(-,5)(+,8)(-,10)	22
	24	1, 2, ..., 24	1,3,5,6	(-,2)(+,6)(+,17)	31
Extended CORDIC vectoring	16	1, 2, ..., 16	1,3,5,6,8	(-,2)(+,6)	23
	24	1, 2, ..., 24	1,3,5,6,12	(-,2)(+,6)(+,17)	32

Table 2. Scaling and repetitions for the standard CORDIC and for the CORDIC with extended vectoring.

- A CORDIC module for coordinates x, y, and z.
- A module to compute the direction of rotation by comparing $y[j]$ with $A[j]t$. For this module we consider two alternatives: a direct implementation and an implementation based on a residual.

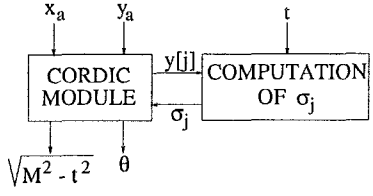

Figure 7. Block diagram of implementation.

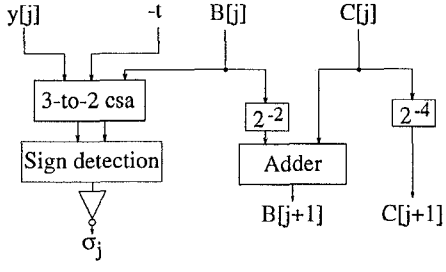

Figure 8. Direct implementation of σ_j module.

4.1. Direct implementation

This implementation is shown in Figure 8. We need to compute $A[j]t = t - B[j]$ and determine the sign of

$$y[j] - A[j]t = y[j] - t + B[j]$$

This is performed by the 3–2 carry–save adder and by the sign detector. Moreover, as described in the Section 2.1.3

$$B[j+1] = \begin{cases} 2^{-2}B[j] + C[j] & \text{if } j < n/2 \\ 2^{-2}B[j] & \text{otherwise} \end{cases}$$

Initial values: $x_a = 0.250000, y_a = 0.750000, t = 0.350000$

	x	y
Scaling: (−, 2)	0.187500	0.562500
Scaling: (+, 6)	0.190430	0.571289

j	Microrotations $(x[j], y[j], z[j])$	$B[j]$	$(T[j]\dagger, \sigma_j)$
1	(0.190430, 0.571289, 0.000000)	0.054688	(0.275977, -1)
1	(0.476074, 0.476074, -0.463648)	0.054688	(0.180762, -1)
2	(0.714111, 0.238037, -0.927295)	0.014355	(-0.097607, 1)
3	(0.654602, 0.416565, -0.682317)	0.003632	(0.070197,-1)
3	(0.706673, 0.334740, -0.806672)	0.003632	(-0.011629,1)
4	(0.664830, 0.423074, -0.682317)	0.000911	(0.073984, -1)
5	(0.691272, 0.381522, -0.744735)	0.000228	(0.031750, -1)
5	(0.703195, 0.359920, -0.775975)	0.000228	(0.010147, -1)
6	(0.714442, 0.337945, -0.807215)	0.000057	(-0.011998, 1)
6	(0.709162, 0.349108, -0.791591)	0.000057	(-0.000835, 1)
7	(0.703707, 0.360189, -0.775968)	0.000014	(0.010203, -1)
8	(0.706521, 0.354691, -0.783780)	0.000004	(0.004694, -1)
8	(0.707907, 0.351931, -0.787686)	0.000004	(0.001935, -1)
9	(0.709281, 0.349166, -0.791592)	0.000001	(-0.000833, 1)
10	(0.708599, 0.350551, -0.789639)	0.000000	(0.000551, -1)
11	(0.708942, 0.349859, -0.790616)	0.000000	(-0.000141, 1)
12	(0.708771, 0.350205, -0.790128)	0.000000	(0.000205, -1)
13	(0.708856, 0.350032, -0.790372)	0.000000	(0.000032, -1)
14	(0.708899, 0.349946, -0.790494)	0.000000	(-0.000054, 1)
15	(0.708878, 0.349989, -0.790433)	0.000000	(-0.000011, 1)
16	(0.708867, 0.350011, -0.790402)	0.000000	(0.000011, -1)

Output: $x[17] = 0.708872$, $y[17] = 0.349999$, $z[17] = -0.790417$

$\dagger\, T[j] = y[j] - (t - B[j])$

Table 3. Simulation of direct implementation (n=16).

$$C[j+1] = 2^{-4}C[j]$$

The initial conditions are $B[1] = (5/32)t$ and $C[1] = (1/512)t$. The initialization of B requires an addition. This can be done with the available adder.

An example of the execution is given in Table 3 for $n = 16$. The algorithm for the direct implementation was simulated using double precision for $n = 16$ and $n = 24$. We generated a uniform distribution of 100 initial vectors with at least one normalized component and, for each of these vectors, the algorithm was simulated for all possible

values of t.

As can be seen, this implementation is quite straightforward. The effect of this module on the cycle time is different for the word–serial and the pipelined case. In the former, the 3–2 addition and the sign detection can be overlapped with the variable shifters in the CORDIC module, so that the cycle time is similar to that of the CORDIC algorithm. On the other hand, in the pipelined implementation, since there are no variable shifters, the cycle time increases by the delay of the 3–2 adder and the sign detection. To avoid this increase, we consider the next implementation.

4.2. Implementation based on residual

We want to determine the direction of rotation immediately at the beginning of the iteration. For this, we define a residual and determine the direction from the sign of the residual. That is,

$$v[j] = 2^j(y[j] - A[j]t)$$

$$\sigma_j = \begin{cases} -1 & \text{if } v[j] \geq 0 \\ +1 & \text{if } v[j] < 0 \end{cases}$$

We now develop a recurrence for v,

$$v[j+1] - 2v[j] = 2^{j+1}(y[j+1] - y[j] - (A[j+1] - A[j])t)$$

From the CORDIC recurrence and defining

$$D[j] = 2^{j+1}(A[j] - A[j+1])t$$

we get

$$v[j+1] = 2v[j] + 2\sigma_j x[j] + D[j] \qquad (8)$$

Further details of this implementation can be found in [8].

5. Comparison with the arcsin(t) case

The calculation of arcsin(t) is a special case, which can be obtained by putting as initial value $(x_a, y_a) = (0, 1/K)$. However, it is also possible to design a special algorithm for this case, as was done in [10] and [7]. In the latter, we have used the same approach as here, but the special case has allowed for some simplifications in the algorithm resulting in fewer iterations. Moreover, for these functions it is possible to use a 2n–bit target t to achieve a precision of 2^{-n} in the whole angle range, using a datapath width similar to the standard CORDIC.

6. Conclusions

In this work we described an algorithm for the CORDIC vectoring mode with arbitrary target value. The approach assures convergence by the use of a simple approximation of the scale factor remaining after iteration j, combined with the inclusion of a few repetitions. The resulting implementation consists of two separate modules, a standard CORDIC module and a module to obtain the direction of each microrotation. We present implementations suitable for word–serial and pipelined architectures. Although we have introduced some repetitions, the number of total iterations remains practically the same as the conventional CORDIC, if repetitions and scalings are used to compensate the scale factor. Consequently, the performance of the standard CORDIC functions is not affected.

The implementation presented here introduced a significant improvement compared to the implementation described in [6] and to the extension of [10].

Acknowledgments: We thank the reviewers and Joseph Cavallaro for their useful comments.

References

[1] H. Ahmed. *Signal processing algorithms and architectures.* Ph.D. dissert., Dept. of Electrical Engineering, Stanford University, Stanford, CA, 1982.

[2] T. Chen. Automatic computation of exponentials, logarithms, ratios and square–roots. *IBM J. Res. Develop.*, 16:380–388, July 1972.

[3] J.-M. Delosme. VLSI implementation of rotations in pseudo–Euclidean spaces. In *Proc. IEEE Int. Conf. on ASSP.*, pages 927–930, 1983.

[4] R. Harber, J. Li, X. Hu, and S. Bass. The application of bit–serial CORDIC computational units to the design of inverse kinematics processors. In *Proc. IEEE Conference on Robotics and Automation*, pages 1152–1157, 1988.

[5] D. Konig and J. Bohme. Optimizing the CORDIC algorithm for processors with pipeline architecture. In *Signal Processing V: Theories and Applications*, pages 1391–1394, 1990.

[6] C. Krieger and B. Hosticka. Inverse kinematics computations with modified CORDIC iterations. *IEE Proc. Comput. Digit. Tech.*, 143(1):87–92, January 1996.

[7] T. Lang and E. Antelo. CORDIC–based computation of arcos and arcsin. Technical report, Dept. Electrical and Comput. Eng., University of California at Irvine, Irvine, USA, 1997. Available in http://www–gpaa.dec.usc.es in the 1997 reports.

[8] T. Lang and E. Antelo. CORDIC vectoring with arbitrary target value. Technical report, Dept. Electrical and Comput. Eng., University of California at Irvine, Irvine, USA, 1997. Available in http://www–gpaa.dec.usc.es in the 1997 reports.

[9] C. Lee and P. Chang. A maximum pipelined CORDIC architecture for inverse kinematic position computation. *IEEE Journal of Robotics and Automation*, 3(5):445–458, October 1987.

[10] C. Mazenc, X. Merrheim, and J. Muller. Computing functions $\cos^{-1}$ and $\sin^{-1}$ using CORDIC. *IEEE Trans. on Computers*, 42(1):118–122, January 1993.

[11] I. Walker and J. Cavallaro. Parallel VLSI architecture for real–time kinematics of redundant robots. In *Proc. IEEE Conference on Robotics and Automation*, pages 870–877, 1993.