Low Latency Pipelined Circular CORDIC*

Elisardo Antelo
Dept. of Electronic and Computer Eng
University of Santiago. SPAIN
elisardo@dec.usc.es

Julio Villalba
Dept. of Computer Architecture
University of Málaga. SPAIN
julio@ac.uma.es

Abstract

The pipelined CORDIC with linear approximation to rotation has been proposed to achieve reductions in delay, power and area; however, the schemes for rotation (multiplication) and vectoring (division) complicate implementation in a single unit. In this work, we improve the linear approximation scheme, leading to a unified implementation for rotation and vectoring where fully parallel tree multipliers are used instead of the second half of CORDIC iterations. We also combine the linear approximation to rotation with the scale factor compensation so that the compensation is performed concurrently with the rotation process. Comparison with other designs is also provided.

1. Introduction

The CORDIC algorithm is an arithmetic method to perform 2D vector rotations. The rotations are performed as a sequence of elementary rotations with a decreasing angle in a convergent linear process. In fact, to use only adders and shifters, the elementary rotations are implemented as similarities. Therefore, the vectors are scaled by a constant during the rotation process. The algorithm has two operating modes: rotation and vectoring.

Current applications include digital signal processing, 3D graphics, reconfigurable computing, speech and music synthesizers, and communication devices (OFDM, CDMA, etc). CORDIC modules are offered by core vendors, specially for FPGA. It is also being used in an FPGA–based Supercomputer [10].

To achieve high performance the algorithm is unfolded and pipelined. For small angle a linear approximation to the rotation can be used requiring multiplications and addition.This approach has a very significant effect on the latency of the conventional pipelined CORDIC since about half of the stages (serially organized) with a delay of about one carry–propagate adder each, are changed by a fast tree–like structure of carry–free counters with only one final carry–propagate addition. The drawback is that this method can only be applied to the rotation mode. The linear approximation in the vectoring mode leads to a division operation. Thus, a fully operational pipelined CORDIC (rotation and vectoring modes) cannot be efficiently implemented using this approach.

In this work we extend the final multiplication approach to the vectoring mode. Our approach is based on the concurrent computation of a reciprocal with the first half of the CORDIC stages. We also present an architecture that implements both modes of operation with final multiplication and with a concurrent compensation for the scale factor, so that further reductions of latency are obtained with respect to the conventional pipelined CORDIC.

*E. Antelo has been partially supported by Xunta de Galicia under project PGIDT03TIC10502PR and J. Villalba has been supported by the Ministry of Educatinal & Science of Spain under project TIC2003–006623

2. CORDIC algorithm

In this section we present a brief description of the CORDIC algorithm. For more details and references see [5]. The algorithm consists in the following steps:

1.-**Initialization**: $x[0] = x_0 \in [1/2, 1)$, $y[0] = y_0 \in (-1, 1)$ and $z[0] = \theta \in [-\pi/2, \pi/2]$ (rotation) or $z[0] = 0$ (vectoring)

2.-**Iteration**. For $j = 0$ to $n-1$: $\sigma_j = sign(z[j])$ (rotation) or $\sigma_j = sign(y[j])$ (vectoring),

$$\begin{aligned} x[j+1] &= x[j] + \sigma_j 2^{-j} y[j] \\ y[j+1] &= y[j] - \sigma_j 2^{-j} x[j], \\ z[j+1] &= z[j] - \sigma_j \tan^{-1}(2^{-j}) \end{aligned} \tag{1}$$

3.-**Scale factor** ($K_n = \prod_{j=0}^{n-1} \cos(\tan^{-1}(2^{-j})) \approx 0.607...$): $x_f = K_n\, x[n]$, $y_f = K_n\, y[n]$, $z_f = z[n]$.

For the rotation mode (x_f, y_f) are the new coordinates of the rotated vector and for the vectoring mode x_f gives the modulus of the vector and z_f is the rotated angle.

It can be shown that using the sign of $z[j]$ (rotation mode) or $y[j]$ (vectoring mode) to obtain the direction of

each elementary rotation (σ_j), the following conditions are verified:

- Rotation mode $|z[j]| \leq \tan^{-1}(2^{-(j-1)})$
- Vectoring mode

$$|\tan^{-1}(|y[j]|/x[j])| \leq \tan^{-1}(2^{-(j-1)}) \quad (2)$$

which results in

$$|y[j]|/x[j] \leq 2^{-(j-1)} \quad (3)$$

where $x[j]$ is bounded by the initial modulus of the input vector (M) scaled by a factor K_j, that is

$$x[j] \leq M/K_j = \sqrt{x[0]^2 + y[0]^2}/K_j \quad (4)$$

In this work we concentrate on a unfolded (parallel) pipelined implementation of the algorithm with carry propagate adders. It consist of n hardware stages implementing Equations (1) and a final constant multiplication by K_n. Since each iteration of the algorithm is implemented in a separate hardware, the shifters are actually hardwired. Registers are introduced to achieve the desired clock cycle resulting in a pipelined system.

The accuracy of the algorithm is determined by many parameters [7] [5]. To have a simpler presentation we assume the following: the approximation to the rotation angle is of the order of the last elementary rotation angle, which is $\tan^{-1}(2^{-(n-1)})$ for n iterations; the input operands have $n-1$ fractional bits. The width of the data–path is roughly $b = 3+(n-1)+m$ bits ($m = \log_2(n)$), including guard and overflow bits; the scale factor is rounded to $n-1+m$ fractional bits. All the approximations that follow in this work have an error less than $2^{-(n-1)}$ and the resultant accuracy should be of $O(2^{-(n-1)})$.

From iteration $j = \lceil n/3 \rceil + 1 = t$, the elementary rotation angles can be approximated to within n bits of precision by $\sigma_j \ \tan^{-1}(2^{-j}) = \sigma_j \ 2^{-j}$. Therefore for $j \geq t$, after a recoding, it is not necessary to implement the z recurrence for both rotation and vectoring.

2.1. Reducing latency through the linear approximation to rotation

Termination schemes perform linear approximations for the rotation when the remaining angle is small enough [1] [12] [13]. These linear approximations lead to a final multiplication (rotation) or division (vectoring) to complete the rotation. To have this approximation correct to within n bits of precision for both rotation and vectoring, it is necessary for the remaining rotation angle α to be bounded by $\alpha < 2^{-n/2}$. Therefore, it is possible to perform this simplified rotation after iteration $j = n/2$ [1] [12] [13]. The implications of this approximation are the following:

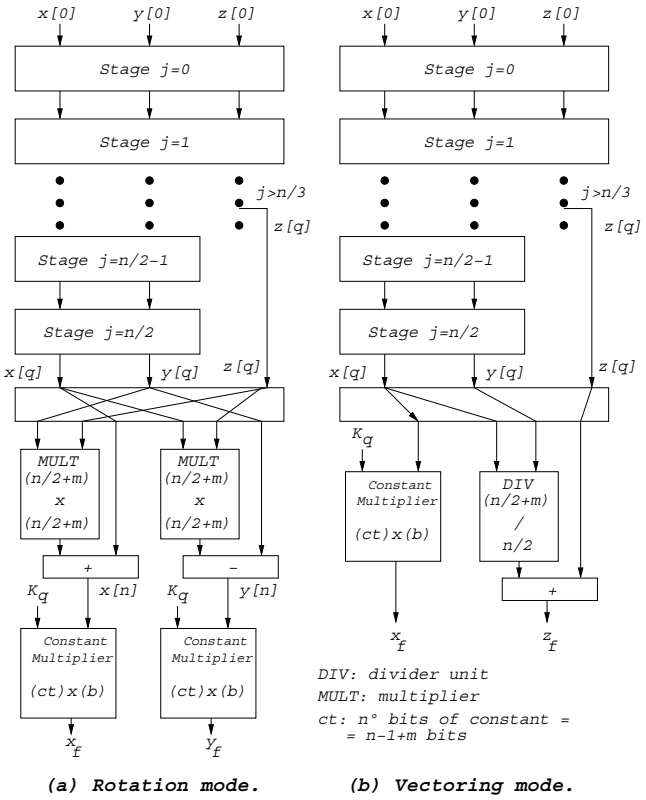


Figure 1. Architectures with a linear approximation to rotation.

- **Rotation**: after CORDIC iteration $j = n/2$ the x and y coordinates ($x[q] = x[n/2+1]$, $y[q] = y[n/2+1]$) are rotated by the remaining angle $z[q] = z[n/2+1]$ as follows

$$xf = K_q(x[q] + z[q]\ y[q]), yf = K_q(y[q] - z[q]\ x[q])$$

Since $|z[q]| < 2^{-n/2}$, the multiplication by z is of about $n/2+m$ bits and can be performed with a tree of counters with logarithmic delay. The multiplication by K_q is performed after the linear approximation to rotation. This scheme is illustrated in Figure 1(a).

- **Vectoring**: in this mode the modulus and the angle between the vector and the x axis are computed. The modulus and the angle are obtained by solving the following equations $xf = K_q\ (x[q] + \alpha\ y[q])$, $0 = K_q(y[q] - \alpha\ x[q]), zf = z[q] + \alpha$, which, for n–bit precision, results in

$$xf = K_q\ x[q], \qquad zf = z[q] + (y[q]/x[q])$$

Since $|y[q]/x[q]| < 2^{-n/2}$, the computation required after the CORDIC iterations are a division of about

$n/2$ bits to obtain the angle, and the scale factor compensation to obtain the modulus. Both processes can be performed concurrently. This scheme is illustrated in Figure 1(b).

Since division is a sequential process, the linear approximation leads to different latency schemes for vectoring and rotation. This fact leads to inefficient unified implementations of rotation and vectoring in a pipelined CORDIC processor with a latency determined by the division process. In addition, the scale factor compensation in the rotation mode is performed after the rotation, while in vectoring the compensation of the scale factor to obtain the modulus can be performed concurrently with the linear approximation to the rotation.

In [2] a unified implementation with a linear approximation was proposed, using a radix–4 prescaled division algorithm to complete the vectoring operation. To have a unified implementation, the rotation is completed with an iterative radix–4 multiplication. As indicated, the division algorithm limits the use of fast parallel tree multipliers. We compare with this scheme in Section 6.

3 Multiplicative scheme for vectoring

In this section we show how to efficiently combine the architecture for vectoring and rotation when a linear approximation to the rotation is used. As shown in the previous section, the linear approximation to rotation leads to a multiplication scheme in the rotation mode and to a division scheme in the vectoring mode. We now show how to also use a multiplication scheme for vectoring.

For vectoring we need to compute $zf = z[q] + (y[q]/x[q])$ which requires a division and add operation. This is transformed into a multiplication operation by first computing $R = 1/x[q]$. There seems to be no apparent advantage to doing this. However, two observations need to be taken into account:

i) We know the bound $|y[q]|/x[q] < 2^{-n/2}$. Therefore, since $1/2 \leq x[q] < M/K_q$, R needs to be computed to about $n/2$ bits of precision.

ii) The $n/2$ leading bits of R can be computed from $x[j]$ with $j < q$. This is due to the fact that roughly two bits of the modulus M (stored as the x coordinate) are determined in each iteration. More specifically, from (2) we know that, at iteration j, the angle between the vector and the x axis is bounded by $\tan^{-1}(2^{-(j-1)})$. In addition, the modulus of the vector at this iteration is bounded by $\sqrt{x[j]^2 + y[j]^2} = M/K_j < M/K_q$. Therefore, the following bound results:

$$|y[j]| < \frac{M}{K_q}\sin(\tan^{-1}(2^{-(j-1)})) = \frac{M}{K_q}\frac{2^{-(j-1)}}{\sqrt{1+2^{-2(j-1)}}}$$

Then from (3) and (4), a bound for $x[j]$ is obtained

$$\frac{M}{K_q}\frac{1}{\sqrt{1+2^{-2(j-1)}}} < x[j] < \frac{M}{K_q} \tag{5}$$

Thus, the maximum difference between $x[j]$ and the scaled modulus M/K_q is

$$\frac{M}{K_q} - x[j] < \frac{M}{K_q}\left(1 - \frac{1}{\sqrt{1+2^{-2(j-1)}}}\right) \leq \tag{6}$$
$$\leq \frac{M}{K_q}(2^{-2j+1} - 2^{-4j+1}) \quad (for\ j > 1)$$

This bound decreases at a rate of 2^{-2j}, which means that approximately two bits of the modulus are determined in each iteration.

We now fully develop both observations to determine the index j from which we can obtain an approximation $\widehat{R}$ of $R = 1/x[q]$. The precision required for $\hat{R}$ is such that the error of the multiplication of $\hat{R}$ and $y[q]$ is less than $2^{-(n-1)}$, that is,

$$|R \times y[q] - \widehat{R} \times y[q]| < 2^{-(n-1)}$$

We assume that we obtain an approximation of R from $\widehat{x}[j]$ with $j < n/2$, which represents the $f+1$ leading bits of $x[j]$. We proceed by following three steps: 1) obtain a bound of $x[q] - \widehat{x}[j]$; 2) determine the error produced by the reciprocal computation of $\widehat{x}[j]$ using a specific method of computation; and finally 3) combine both items to obtain a bound on j.

3.1. Bound of $x[q] - \widehat{x}[j]$

We use the identity

$$x[q] - \widehat{x}[j] = (x[q] - x[j]) + (x[j] - \widehat{x}[j])$$

From (5) and (6) we have:

$$0 \leq x[q] - x[j] < \frac{M}{K_q} - x[j] < \frac{M}{K_q}(2^{-2j+1} - 2^{-4j+1})$$

To simplify the derivation of the bound of $x[j] - \widehat{x}[j]$ we multiply by a factor to normalize to the range $[1, 2)$. From (5) we obtain

$$1 < \frac{2}{\sqrt{1+2^{-2(j-1)}}} < \frac{2K_q}{M}x[j] < 2$$

Thus, the normalizing factor is $2K_q/M$ and then the normalized value has at most one integer bit. We scale the difference $x[j] - \widehat{x}[j]$ by this factor. Since $\widehat{x}[j]$ represents the $f+1$ leading bits of $x[j]$, the following bound results:

$$\frac{2K_q}{M}(x[j] - \widehat{x}[j]) < 2^{-f}$$

Therefore

$$x[j] - \widehat{x}[j] < 2^{-f}\ \frac{M}{2K_q}$$

Then the resultant bound for $(x[q] - \widehat{x}[j])$ is

$$x[q] - \widehat{x}[j] < \frac{M}{K_q}\left(2^{-f-1} + 2^{-2j+1} - 2^{-4j+1}\right) \qquad (7)$$

3.2. Reciprocal computation

To simplify the presentation we assume a direct table method to compute the reciprocal. The extension to other methods (say bipartite tables, linear or quadratic interpolation, or high–order methods) is straightforward.

From [3] we know the following result: for $d \in [1, 2)$, a table taking as input an estimation of d with error less than 2^{-p}, produces an approximation $a = 0.1a_1a_2....a_{p+g}$ of the reciprocal of d that satisfies

$$1 - 2^{-p}\frac{(1 + 2^{-(g+1)})}{2} < a \times d < 1 + 2^{-p}\frac{(1 + 2^{-(g+1)})}{2}$$

Therefore

$$\left|\frac{1}{d} - a\right| < \frac{2^{-p}}{d}\frac{(1 + 2^{-(g+1)})}{2}$$

This is the error obtained when an estimation of $d \in [1, 2)$ with an error bounded by 2^{-p} is used to address the table.

Since $x[q] \in [1/2, M/K_q)$ we need to normalize $x[q]$ by $2K_q/M$ to have its value within [1,2), so that we can use the above result. Thus, taking $d = 2K_qx[q]/M \in [1, 2)$, the error obtained is bounded by

$$\left|\frac{1}{d} - a\right| < \frac{M\ 2^{-p}}{2K_qx[q]}\frac{(1 + 2^{-(g+1)})}{2} \qquad (8)$$

De–normalizing this result, we obtain the bound in the error of estimation of the reciprocal of $x[q]$

$$\left|\frac{1}{x[q]} - \widehat{R}\right| < \frac{2^{-p}}{x[q]}\frac{(1 + 2^{-(g+1)})}{2} \qquad (9)$$

Since the error of estimation of d is bounded by 2^{-p}, the resultant allowed estimation error for $x[q]$, considering the de–normalizing factor, results in

$$x[q] - \widehat{x}[j] \leq \frac{M}{2K_q}2^{-p} \qquad (10)$$

3.3. Bound on j

We now determine a bound for the index j so that we can obtain an estimation of $R = 1/x[q]$ from some bits of $x[j]$ in such a way that $y[q]/x[q]$ is obtained within n bits of precision.

From (9) we obtain a bound in the error of the computation of $y[q]/x[q]$

$$|y[q]| \times \left|\frac{1}{x[q]} - \widehat{R}\right| < \frac{y[q]}{x[q]}2^{-p}\frac{(1 + 2^{-(g+1)})}{2}$$

Moreover, since $|y[q]| \leq 2^{-(n/2)}x[q]$, we have

$$|y[q]| \times \left|\frac{1}{x[q]} - \widehat{R}\right| < 2^{-(p+n/2)}\frac{(1 + 2^{-(g+1)})}{2}$$

Since the error in the computation of $y[q]/x[q]$ must be less than $2^{-(n-1)}$, we obtain the following condition for p,

$$2^{-p} < 2^{-(n/2-1)}\frac{2}{1 + 2^{-(g+1)}}$$

Since $1 < 2/(1 + 2^{-(g+1)}) < 2$ for $g \geq 0$, the following bound results

$$p \geq \frac{n}{2} - 1$$

To reduce table size we take $p = n/2 - 1$ and $g = 0$.

Therefore, from (10) the allowed error for the estimation of $x[q]$ is

$$x[q] - \widehat{x}[j] < \frac{M}{2K_q}2^{-(n/2-1)} = \frac{M}{K_q}2^{-n/2} \qquad (11)$$

A bound for the value of $x[q]-\widehat{x}[j]$ is given in (7). Therefore, from the bound of the allowed error given in (11), it is required that

$$2^{-f-1} + 2^{-2j+1} - 2^{-4j+1} < 2^{-n/2}$$

Since f determines the number of bits of $x[j]$ that input the reciprocal table, we select the minimum possible value of $f = n/2$. Then we obtain the condition on j:

$$2^{-2j+1} - 2^{-4j+1} < 2^{-n/2-1}$$

which is verified for $j \geq n/4 + 1$.

In summary, the computation of $y[q]/x[q]$ is performed as follows: (see Figure 2):

- Obtain a reciprocal $\widehat{R}$ from a table addressed by $f = n/2$ bits (it is not necessary to input the leading one into the table) of $x[j]$ with $j \geq n/4+1$. The result is of about $n/2$ bits. The index j is selected so that enough time is provided for the delay of the table. The bound of about $n/4$ CORDIC iterations seems long enough so that the reciprocal computation is not in the critical path.
- After iteration $j = n/2$, perform a $(n/2+m)\times(n/2+m)$ multiplication of $\widehat{R} \times y[q]$. Since only the leading $n/2$ bits of the multiplication are necessary, we use a truncated multiplier.

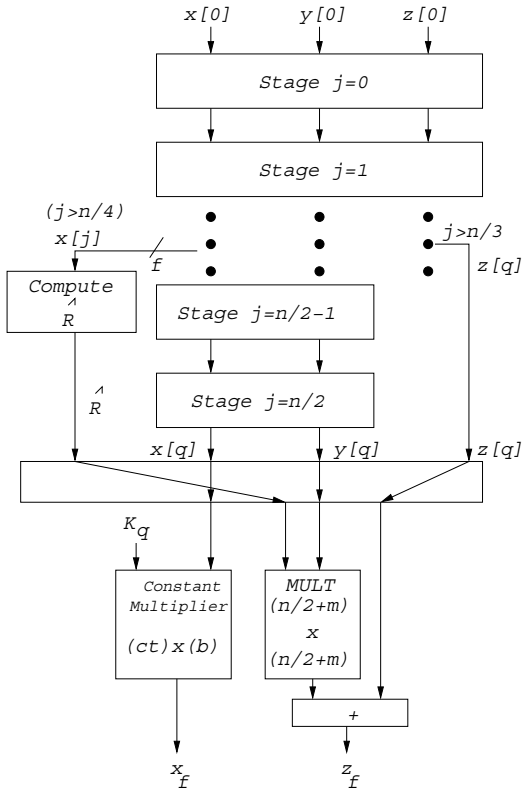


Figure 2. Proposed scheme for linear approximation to rotation (vectoring mode).

If n is large the table for reciprocal computation could be large. In this case alternative methods for reciprocal computation can be used. Depending on the precision required, we find the following among several alternatives [5]: bipartite tables, linear or quadratic interpolation, high–order polynomial methods and digit–by–digit methods.

4. Concurrent scale factor compensation

Constant scale factor compensation has been performed in the following ways for different authors [5]: Pre– or post–multiplying the x and y coordinates by the constant scale factor, decomposing the scale factor in a product of shift and add terms and performing the compensation using similar iterations such as elementary rotations, concurrent compensation [14], etc.

For a unified rotation/vectoring pipelined architecture, the most suitable method might be pre– or post–constant multiplication. Using this scheme the compensation adds latency overhead, since in the rotation mode the compensation has to be performed after or before the elementary rotations but not concurrently (see Figure 1(a)).

We now show how to perform the compensation concurrently with the linear approximation to the rotation, to reduce the latency overhead of the compensation.

After the $n/2$ elementary rotations we perform the following computation

$$xf = K_q\, x[q] + P\, y[q], \quad yf = K_q\, y[q] - P\, x[q]$$

with $P = K_q\, z[q]$. The multiplications $K_q\, x[q]$ and $K_q\, y[q]$ are constant multipliers with a multiplicand of b bits (result truncated to b bits). Since $|z[q]| < 2^{-n/2}$, it has $n/2 + m$ significant bits (including the sign). Moreover $K_q < 1$ and therefore $P < 2^{-n/2}$. Then the multiplications $P\, y[q]$ and $P\, x[q]$ are of $(n/2+m) \times (n/2+m)$ (truncated to $n-1+m$ fractional bits).

Note that P should be computed before $x[q]$ and $y[q]$ for the scheme to be effective. In Section 2 we mentioned that to reduce the complexity of the z iteration, from iteration $j = \lceil n/3 \rceil + 1 = t$, the σ_j values are obtained from a direct recoding of $z[t]$. Therefore, we obtain $z[q]$ from $z[t]$ by performing a reverse recoding (from digit set $\{-1, 1\}$ to $\{0, 1\}$ two's complement) of the digits with weights lower than $2^{-n/2}$.

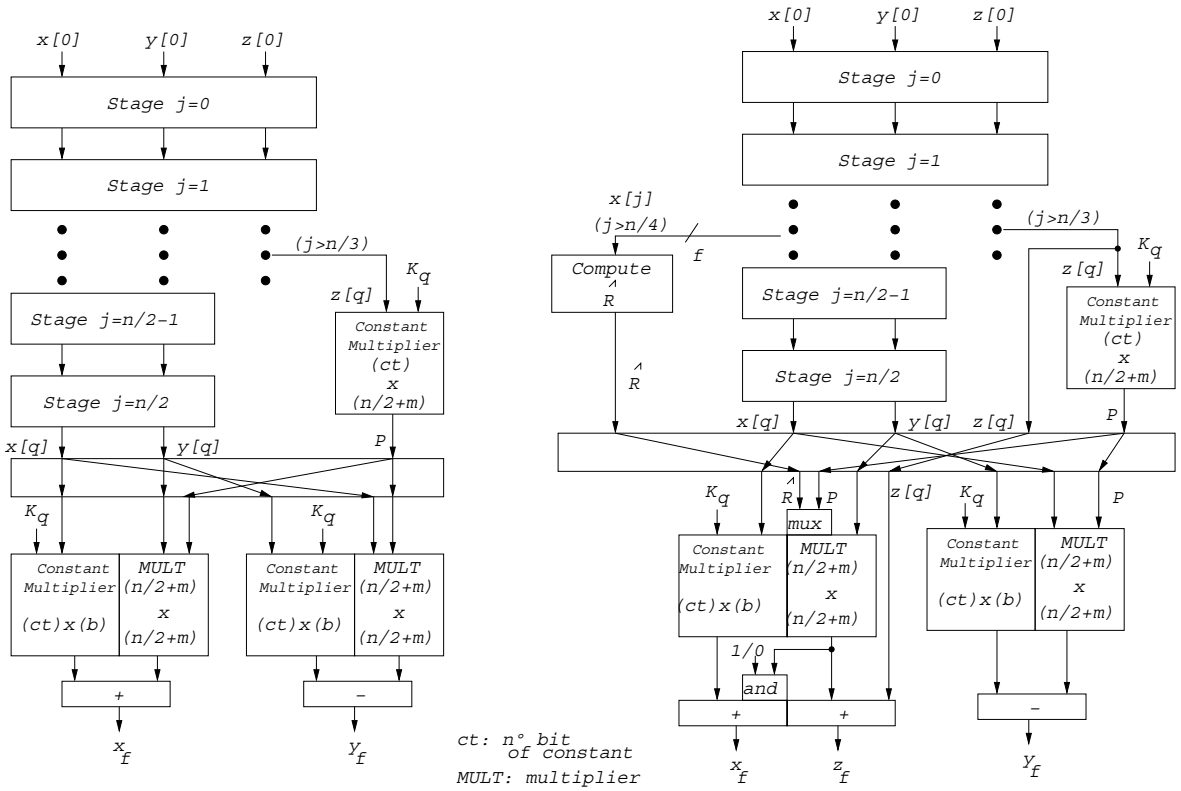
Figure 3(a) shows the implementation of the proposed method. This method introduces a latency overhead of about one constant multiplication for both the linear approximation to rotation and scale factor compensation, while the conventional scheme requires two series multiplications, a $(n/2+m) \times b$ multiplication and a constant multiplication. The effect on hardware complexity is analyzed in Section 6.

5. Combined architecture for rotation and vectoring

Figure 3(b) shows the unified architecture for vectoring and rotation incorporating the linear approximation to rotation for both operation modes and the concurrent compensation of the scale factor. The combination of both architectures is simple since the additional hardware required is only one multiplexer, one adder, and one row of AND gates.

6. Evaluation

In this section we compare our design and existing proposals in terms of hardware complexity and delay. There have been many proposals regarding variations on the CORDIC algorithm. However, we are only concerned about those schemes proposed for a CORDIC unit implementing both rotation and vectoring. Moreover, many techniques proposed to reduce CORDIC latency, such as redundant CORDIC and very–high radix CORDIC, are orthogonal to our proposal. Therefore, it is appropriate to only compare



(a) Concurrent scale factor compensation for rotation.

(b) Architecture for combined rotation and vectoring.

Figure 3. Proposed architectures.

it to the conventional radix–2 CORDIC and to a combined radix–2/radix–4 CORDIC implementation [2].

The radix–2/radix–4 proposal performs about the first half of the iterations in radix–2, and the rest of iterations in radix–4. The radix–4 iterations correspond to a multiplication operation for rotation and a division operation for vectoring with prescaling of operands in parallel to the scale factor compensation.

For the delay calculations we use a rough timing model based on logical effort [11] normalized to FO4 units (FO4 refers to the delay of a 1x inverter with a load of four 1x inverters). The effect of the interconnections in the delay was not considered.

Regarding hardware complexity, we determined the number of equivalent two–input nand gates for each design. The relative complexity of each gate with respect to a two–input nand gate was determined in terms of the size of the total active area of the transistors of the gate. This simple area–delay model provides area and delay ratios that should indicate the potential advantage of our proposal when actual implementations are considered.

Table 1 shows the data of the model corresponding to the simple gates and basic hardware elements used in the evaluation.

For the comparison we considered additions implemented with fast parallel prefix adders: for the table look–up we assumed an implementation using a multiplexer tree (of 4–to–1 multiplexers) for each output bit. This represents a fast but costly implementation for a look–up table. Assuming that the complexity of a look–up table varies according to an area $\times$ time^2 law, a slower but more area efficient implementation results by doubling the delay, reducing the hardware complexity by a factor of four.

The computation of $\widehat{R}$ corresponds to the computation of a reciprocal with about $n/2$ bits of precision. We considered three types of implementations: linear approximation [4], quadratic approximation [9] [8], and a very–high radix digit–by–digit reciprocal [6]. Figure 4 shows the estimated delay and area for the three methods for the range $16 \leq n \leq 116$. In addition, since the reciprocal should be computed in parallel to about $n/4$ CORDIC iterations, we show in Figure 4(a) the bound of delay corresponding to $n/4$ CORDIC iterations. In our design we used the method with minimum area and a delay less than the delay of $n/4$

Table 1. Delay equations and relative area for basic components.

Hardware module	Delay equation FO4 units*	Input load # inverter loads	relative area # nand-2
nand-2	$0.4 + 0.2L$	4/3	1.0
nor-2	$0.4 + 0.2L$	5/3	1.3
21AOI	$0.5 + 0.2L$	(2,2,5/3)	2.1
21OAI	$0.5 + 0.2L$	(4/3,2,2)	2.1
XOR	$1.5 + 0.2L$	7/3	3.8
2–1 mux (control)	$1.5 + 0.2L$	7/3	3.4
2–1 mux (data)	$1.1 + 0.2L$	4/3	3.4
4–1 mux (decoded)	$1.6 + 0.2L$	4/3	7.0
Full–adder	$4.0 + 0.2L$	13/3	11.0
4-to-2 adder	$6.0 + 0.2L$	5	23.0
recoder (binary to radix–4)	$2.2 + 0.2L$	17/3	10/digit
buffering tree (L_{in} *to* L)	$0.72 \ln(L/L_{in})$	L_{in}	L
i inputs look–up table (fast)	$1.75i - 4.35 + 0.2L$	4	$2.3(2^{i-1} - 1)$/output bit
i inputs look–up table (slow)	$2\,(1.75i - 4.35) + 0.2L$	4	$0.6(2^{i-1} - 1)$/output bit

* L: total load capacitance normalized to inverter capacitance.

CORDIC iterations. Figure 4(b) shows three regions A, B and C, each one corresponding to the method used for the corresponding range of n. For lower precisions (regions A and B) the linear or quadratic approximation are used. However, for higher precisions the very–high radix approach is more convenient due to better scalability[1].

In the delay and area estimations we only considered the combinational elements, since the contributions due to registers depend on the cycle time requirements. We performed the comparisons for a range of $16 \leq n \leq 116$.

Figure 5 shows the delay and hardware complexity for the compared designs. Figure 6 shows the corresponding delay and hardware complexity ratios taking as a reference the proposed design.

The conventional CORDIC presents 1.3 to 1.5 more area and 1.7 to 2.0 more delay compared to our proposal. The radix–2/radix–4 CORDIC presents about 1.2 more area and 1.5 more delay.

We conclude that our approach might lead to more efficient CORDIC modules when rotation and vectoring are implemented in the same unit. The reduction in delay can be "converted" into a reduction in dynamic power consumption through voltage scaling (increasing the delay). Specifically, if s is the speedup factor among two designs with the same voltage, through voltage scaling of the faster design to have the same delay in both designs, the factor of reduction in dynamic power consumption is roughly $[(0.3s+0.7)/s]^2$, provided that both designs present similar activity factors and active capacitance. Since our design has less hardware complexity than the other designs we have compared it with, we can expect factors of dynamic power reduction of about 0.4-0.5 with respect to conventional CORDIC, and 0.6 with respect to radix–2/radix–4 CORDIC.

[1]For higher precisions high–order polynomial approximations could also be considered.

7. Conclusions

In this work we extend the approach of final multiplication to the vectoring mode of the CORDIC algorithm, by computing a reciprocal concurrently to the first iterations and a final multiplication using parallel tree multipliers. This is in contrast to previous proposals where the implementation of both modes of operation in the same architecture was constrained by a division operation, preventing the use of fast parallel tree multipliers. We also combined the linear approximation scheme with the scale factor compensation, thus further reducing the delay. A comparison using a rough area–time model indicates that the proposed scheme may achieve significant delay and/or dynamic power reductions with no increase in area in actual implementations.

References

[1] H. Ahmed. Efficient elementary functions generation with multipliers. *Proc. 9th IEEE Symposium on Computer Arithmetic*, pages 52–59, 1990.

[2] E. Antelo, J. Bruguera, and E. Zapata. Unified mixed radix 2–4 redundant cordic processor. *IEEE Transactions on Computers*, 43(9):227–241, Sept. 1996.

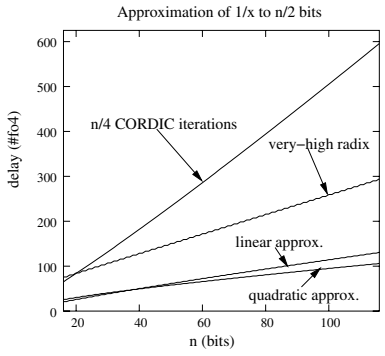
[3] D. DasSarma and D. Matula. Measuring the accuracy of rom reciprocal tables. *IEEE Transactions on Computers*, 43(8):932–940, Aug. 1994.

[4] D. DasSarma and D. Matula. Faithful interpolation in reciprocal tables. *Proc. 13th IEEE Symposium on Computer Arithmetic*, pages 82–91, 1997.

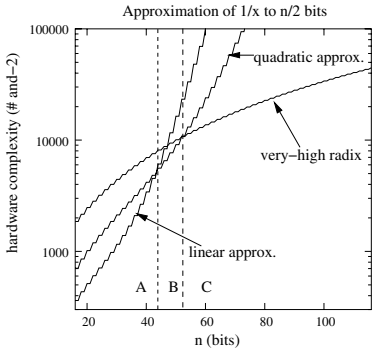
[5] M. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[6] M. Ercegovac, T. Lang, and P. Montuschi. Very-high radix division with prescaling and selection by rounding. *IEEE Transactions on Computers*, 43(8):909–918, Aug. 1994.

[7] Y. Hu. The quantization effects of the cordic algorithm. *IEE Transactions on Signal Processing*, 40(4):834–844, 1992.

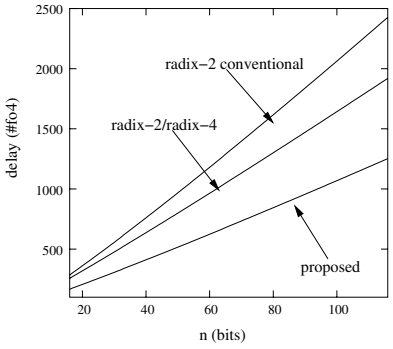


(a) Delay.

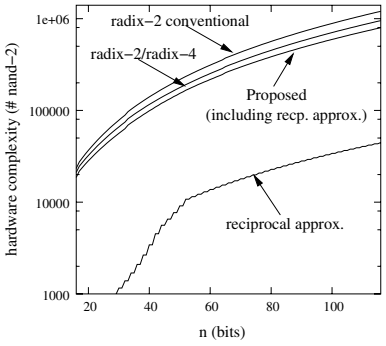


(b) Hardware complexity.

Figure 4. Complexity of the methods used to compute a reciprocal of $n/2$ bits.



(a) Delay.



(b) Hardware complexity

Figure 5. Complexity of the compared designs.

[8] A. Piñeiro, J. Bruguera, and J. Muller. Faithful powering computation using table look-up and a fused accumulation tree. *Proceedings. 15th IEEE Symposium on Computer Arithmetic*, pages 40–47, June 2001.

[9] M. Schulte and E. Swartzlander. Hardware designs for exactly rounded elementary functions. *IEEE Transactions on Computers*, 43(8):964–973, Aug. 1994.

[10] O. Storaasli. Computing faster without cpus: Scientific applications on a reconfigurable, fpga-based hypercomputer. *Available at http://acmb.larc.nasa.gov/acmbexternal/ Personnel/Storaasli/PDF/1Olaf.pdf*.

[11] I. Sutherland and al. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann Publishers, 1999.

[12] D. Timmermann and al. A modified cordic algorithm with reduced iterations. *Elect. Letters*, 25(15):950–951, 1989.

[13] D. Timmermann, H. Hahn, and B. Hosticka. Low latency time cordic algorithms. *IEEE Transactions on Computers*, 41(8):1010–1015, Aug. 1992.

[14] J. Villalba, T. Lang, and E. Zapata. Parallel compensation of scale factor for the cordic algorithm. *Journal of VLSI Signal Processing*, 19(3):227–241, Aug. 1998.

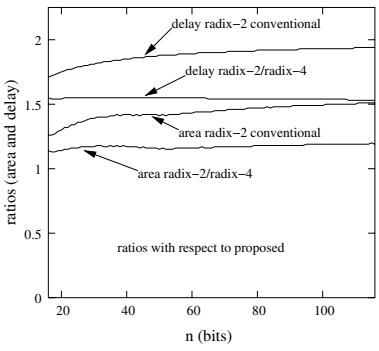


Figure 6. Ratios of complexity.