High Performance Rotation Architectures Based on the Radix-4 CORDIC Algorithm

Elisardo Antelo, Julio Villalba, Javier D. Bruguera, and Emilio L. Zapata

Abstract—Traditionally, CORDIC algorithms have employed radix-2 in the first n/2 microrotations (n is the precision in bits) in order to preserve a constant scale factor. In this work, we will present a full radix-4 CORDIC algorithm in rotation mode and circular coordinates and its corresponding selection function, and we will propose an efficient technique for the compensation of the nonconstant scale factor. Three radix-4 CORDIC architectures are implemented: 1) a word serial architecture based on the zero skipping technique, 2) a pipelined architecture, and 3) an application specific architecture (the angles are known beforehand). The first two are general purpose implementations where redundant (carry-save) or nonredundant arithmetic can be used, whereas the last one is a simplification of the first two. The proposed architectures present a good trade-off between latency and hardware complexity when compared with already existing CORDIC architectures.

Index Terms—CORDIC algorithm, radix-4, redundant arithmetic, VLSI architectures, pipelined architectures.

1 Introduction

The CORDIC (Coordinate Rotation Digital Computer) algorithm was introduced by Volder [21] in 1959 and later generalized by Walther [22]. It is an iterative algorithm for the calculation of the rotation of a two dimensional vector in linear, circular, or hyperbolic coordinate systems. The rotation is carried out by means of a sequence of iterations in each of which one rotation over a given elementary angle (microrotation) is evaluated by means of addition and shift operations. The rotated vector is scaled by a constant factor that must be compensated. Its current applications are within the fields of computational algebra and image processing [11]. Architectures based on CORDIC have been proposed for matrix inversion, filtering, eigenvalue calculations, SVD algorithms, orthogonal transforms, etc.

Traditionally, the implementations of the CORDIC algorithm have been carried out on word serial architectures using conventional nonredundant arithmetic with radix-2 microrotations and fixed point internal format [5]. Pipelined implementations, where each iteration is carried out in a different module, have been proposed in order to increase the throughput [2], [15]. Different architectures with carry-save or signed-digit redundant arithmetic have been designed to eliminate the ripple carry of the adders and several methods have been developed in order to assure convergence of the algorithm using redundant arithmetic. On-line implementations of the CORDIC algorithm are proposed in [9], [17] using a redundant set of values in order to express the direction of each microrotation. These implementations result in a nonconstant scale factor. Several methods that maintain the scale factor constant using carry-save [6], [20] or signed digit [7], [16], [19] arithmetic have been proposed in order to reduce the complexity.

The usual implementations of the CORDIC algorithms employ fixed point [5] or mixed implementations, with the coordinates of the vector externally in floating point and the angle in fixed point [9], [14], [15]. A CORDIC processor that is completely implemented in floating point arithmetic, where the coordinates of the vector and the angle to be rotated are expressed in floating point, has recently been designed [10]. On the other hand, architectures using radix-4 microrotations have been proposed in order to reduce the number of iterations. CORDIC processors that use radix-4 in the second half of the microrotations are presented in [1], [16]. In [3], a general purpose pipelined CORDIC processor is proposed, where all the microrotations are radix-4 and the scale factor is not constant. In this algorithm, the radix-4 microrotation is obtained as the repetition of a radix-2 microrotation, and then the first half of the radix-4 microrotations have double the complexity of a radix-2 microrotation, losing the advantage of using radix-4 microrotations. Moreover, this algorithm is only suitable for a pipelined architecture.

In this work, we present a new radix-4 CORDIC algorithm for circular coordinates in rotation mode. This algorithm is an extension of the radix-2 algorithm, where we use powers of four instead powers of two. The complexity of the radix-4 microrotations is similar to that of the conventional radix-2 microrotations. We develop word serial and pipelined VLSI architectures with carry-save arithmetic (the corresponding implementations using nonredundant arithmetic can be obtained easily from the redundant architectures), both for specific applications in which the angles are known beforehand and for the general case, so as to implement the rotation over any angle. The scale factor is not constant and its compensation is more complex than in

- *E. Antelo and J.D. Bruguera are with the Dept. Electrónica y Computación, Facultad de Fisica, Univ. Santiago de Compostela, 15706 Santiago de Compostela, Spain.*
- *J. Villalba and E.L. Zapata are with the Dept. Arquitectura de Computadores, University of Malaga, E.T.S.I. Informática, Campus Teatinos, 29080 Málaga, Spain. E-mail: ezapata@atc.ctima.uma.es.*

Manuscript received 15 Mar. 1995; revised 14 May 1996.
For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number 104661.0.

full radix-2 or half radix-2 (radix-2 in the first half and radix-4 in the second half) architectures. However, the total number of microrotations is reduced by 50 percent with respect to the radix-2 algorithm and 33 percent with respect to the mixed radix algorithm, maintaining the same complexity for each microrotation. This way, the number of cycles of a word serial implementation and the total hardware in the case of a pipelined architecture is significantly reduced.

The rest of this work is structured as follows. In Section 2, we develop the radix-4 CORDIC algorithm. We prove its convergence and calculate the number of radix-4 iterations. We also develop a selection function that is valid for its implementation in conventional and carry-save redundant arithmetic and we propose a method for the compensation of the nonconstant scale factor. In Section 3, we propose different architectures based on the radix-4 CORDIC algorithm. We present word serial and pipelined general and specific purpose architectures. Finally, in Section 4, we evaluate each one of these architectures comparing them to other designs that have recently appeared in the literature.

2 Radix-4 CORDIC Algorithm

Even though, traditionally, the radix-2 CORDIC algorithm was the one used, it is possible to employ radix that are higher than two [3], [16]. When a higher radix is considered, more bits of the result are calculated each iteration, reducing the total number of iterations. In radix r, where r is a power of two, the rotation angle is decomposed into a series of elementary angles, or microrotation angles, whose values are $\alpha_i[\sigma_i] = \tan^{-1}(\sigma_i r^{-i})$. The coefficients σ_i can take values from the set $\{-r/2, ..., 0, ..., r/2\}$. This is a minimally redundant coefficient set, over which each angle may present different decompositions. For radix higher than 4, the σ_i coefficients are not integer powers of two and the complexity of each iteration increases significantly [3].

When radix-4 is used instead of radix-2, the total number of iterations of the CORDIC algorithm is halved (see Section 2.3). This leads to a significant reduction in the number of cycles for a word serial architecture and in the complexity of the VLSI implementation for a pipelined architecture. In this section, we develop the radix-4 CORDIC algorithm for the rotation of a vector in circular coordinates (rotation mode). We first extend the iterative equations of the CORDIC algorithm to radix-4. We prove the convergence of the algorithm and we develop a new selection function for the calculation of the σ_i coefficients, valid for implementations using either redundant or nonredundant arithmetic. We determine the number of iterations needed in order to achieve a given precision and present a method for the compensation of the nonconstant scale factor.

We propose the following radix-4 CORDIC equations:

$$\begin{aligned} x_{i+1} &= x_i + \sigma_i 4^{-i} y_i \\ y_{i+1} &= y_i - \sigma_i 4^{-i} x_i \\ z_{i+1} &= z_i - \alpha_i[\sigma_i] \end{aligned} \tag{1}$$

where $\sigma_i \in \{-2, -1, 0, 1, 2\}$, $\alpha_i[\sigma_i] = \tan^{-1}(\sigma_i 4^{-i})$, x_0 and y_0 are the coordinates of the initial vector, and z_0 the rotation angle. Consequently, x_{i+1} and y_{i+1} are the coordinates of the vector resulting from applying i + 1 microrotations and z_{i+1}, the angle remaining to be rotated. The final coordinates are scaled by

$$K = \prod_{i \geq 0} k_i = \prod_{i \geq 0} \left(1 + \sigma_i^2 4^{-2i}\right)^{1/2} \tag{2}$$

The scale factor is not constant as it depends on the sequence of σ_is. This scale factor must be evaluated for each rotation angle and compensated to preserve the norm of the vector. For the conventional radix-2 CORDIC algorithm, the scale factor takes the value of approximately K = 1.64. How,ever in the case of the radix-4 algorithm, this value ranges from K = 1.0 to K = 2.52. This way, the vector can be amplified by a larger factor, and then the values of x and y during the iterations can be greater than in the case of the radix-2 algorithm.

2.1 Convergence of the Radix-4 CORDIC Algorithm in Rotation Mode

In order to prove the convergence of the radix-4 CORDIC algorithm, we must prove that variable z ((1)) is bounded in each one of the iterations. In order to make this proof easier, we perform the following change of variable

$$\begin{aligned} w_i &= 4^i z_i \\ A_i[\sigma_i] &= 4^i \tan^{-1}(\sigma_i \bullet 4^{-i}) \end{aligned} \tag{3}$$

This way, we can express iteration z as

$$w_{i+1} = 4(w_i - A_i[\sigma_i]) \tag{4}$$

To prove that variable z is bounded in each iteration means that we have to select the σ_i coefficients such that (4) is also bounded in each iteration. Following a similar method to the one proposed for the radix-4 SRT-division in [8], we define

$$\begin{aligned} P_i[q] &= A_i[q] - (2/3)A_i[1] \\ B_i[q] &= A_i[q] + (2/3)A_1[1] \end{aligned} \tag{5}$$

where $q \in \{-2, -1, 0, 1, 2\}$. Both $P_i[q]$ and $B_i[q]$ are monotonous functions that verify $|P_i[q]| \leq |P_{i+1}[q]|$ and $|B_i[q]| \leq |B_{i+1}[q]|$. Then a selection criterion which guarantees the convergence of the radix-4 CORDIC algorithm is the following: We select $\sigma_i = q$ if

$$P_i[q] \leq w_i \leq B_i[q] \tag{6}$$

These intervals verify the continuity condition, that is, any value of w belongs to at least one of the selection intervals. In Fig. 1, we show the intervals for the selection of σ_i according to (6). The limits of the intervals that permit the selection of a given value of σ_i depend on i, the iteration being evaluated. Among the selection intervals, there is an overlap due to the redundancy in the set of values that σ_i can take. This overlap permits obtaining a selection function in redundant arithmetic that is easy to implement (see below). In the following theorem, we prove that the selection criterion for the σ_i values expressed in (6) guarantees the convergence of the radix-4 CORDIC algorithm.

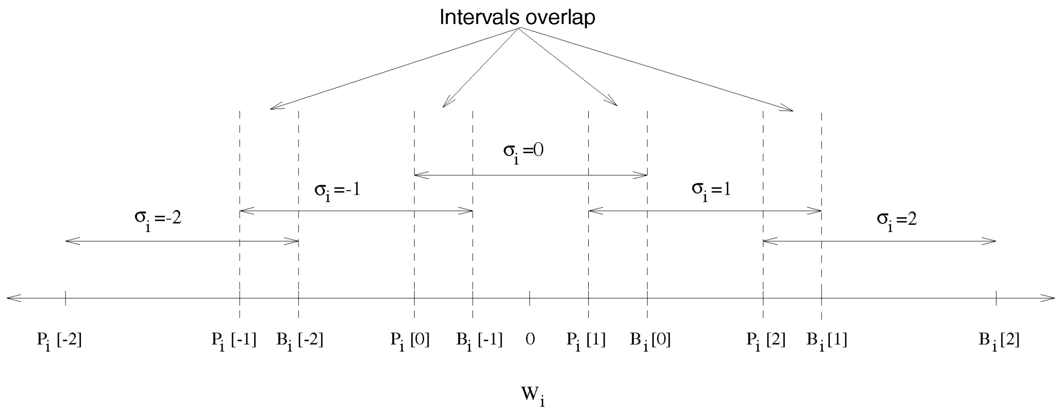


Fig. 1. Selection intervals according to (6).

THEOREM 1. *Selecting* σ_i *according to the criterion given in* (6), *we have that*

$$|w_i| \le A_i[2] + (2/3)A_i[1] \quad (7)$$

PROOF. We are going to prove the theorem by induction. Assuming that $|z_0| < \pi/2$, the theorem is verified for i = 0

$$|z_0| = |w_0| \le \pi/2 < A_0[2] + (2/3)A_0[1] \simeq 1.63 \quad (8)$$

Then, if we assume that (7) is valid for i = m – 1, we must prove that it is verified for i = m. As (6) is verified for i = m – 1, then there is some value of q such that

$$P_{m-1}[q] \le w_{m-1} \le B_{m-1}[q] \quad (9)$$

Substituting $P_{m-1}[q]$ and $B_{m-1}[q]$ by their value ((5)), subtracting $A_{m-1}[q]$, and multiplying each side of this expression by four, we obtain

$$-(8/3)A_{m-1}[1] \le 4(w_{m-1} - A_{m-1}[q]) \le (8/3)A_{m-1}[1] \quad (10)$$

Introducing (4) in (10), taking into account that

$$(8/3) \bullet \tan^{-1}(4^{-(m-1)}) \le \tan^{-1}(2 \bullet 4^{-m}) + (2/3) \bullet \tan^{-1}(4^{-m})$$

and the definition given in (3), we obtain

$$|w_m| \le (8/3)A_{m-1}[1] \le A_m[2] + (2/3)A_m[1] \quad (11)$$

Therefore, the theorem is verified for i = m. □

2.2 Selection Function

In (6), we have obtained a criterion for the selection of the coefficients. We also know that redundant arithmetic is used in the design of high speed CORDIC processors making the additions carry-free [9]. Therefore, in this section, we obtain a selection function for carry-save redundant arithmetic that can be easily implemented in hardware (this selection function can be used for nonredundant arithmetic as well).

In order to be able to obtain a selection function that is independent from the index of the iteration, it is necessary that the limits of the intervals for the selection of σ_i be the same in every iteration. Then, we have to identify a common overlapping area for all the iterations, which determines the limits of the selection intervals that are independent of i. This is possible for microrotations with i > 0. Consequently, we are going to obtain the selection function for i > 0 and i = 0 separately. However, we only discuss the case for i > 0 because the case for i = 0 is similar. In Fig. 2, we show the overlap between the selection intervals corresponding to coefficients q, –q, q + 1, and –q – 1 for different values of i (q ≥ 0).

The selection criterion for the σ_i provided by (6) can be reformulated as

$$\sigma_i = q \ \ if \ \ P[q] \le w_i \le B[q] \quad (12)$$

where $P[q] = \max\{P_i[q]\}$ and $B[q] = \min\{B_i[q]\}$. If i -> ∞, $P_\infty[q+1] < B_1[q]$ and $P_1[-q] < B_\infty[-q-1]$ and so we always have a common overlapping area. Therefore, the selection intervals [P[q], B[q]] are independent from i and overlap, due to the redundancy in the representation of the σ_i.

In order to obtain the value of σ_i, it is necessary to compare the value of w_i (see (3)) to the decision points of the selection function. However, w_i is expressed in carry-save arithmetic. In order to make the implementation of the selection function in redundant arithmetic easier, the value of σ_i must be determined by assimilating just a small number of most significant bits of w_i (a similar strategy was used by Ercegovac and Lang for the radix 2 CORDIC [9]). This is possible due to the overlap in the selection intervals, which permits the selection of the correct value of σ_i from an estimation of the value of w_i (see Figs. 1 and 2). The shorter overlapping interval (P[q + 1], B[q]) is obtained for q = 1 and i = ∞, and its size is $\Delta = B_1[1] - P_\infty[2] = 0.29$.

Let $\hat{w}_i$ be the assimilated value of w_i with t fractional bits. Then we have

$$\hat{w}_i \le w_i < \hat{w}_i + 2^{-t+1} \quad (13)$$

The value of σ_i is determined from the value of $\hat{w}_i$. To reduce the time and hardware cost, we must assimilate a minimum number of fractional bits. Fig. 3 shows the overlap interval (P[q + 1], B[q]) with the different allowed values for $\hat{w}_i$, in the case of a minimum value of t. The minimum value of t is achieved when the following two conditions are true:

1) It is possible to differentiate the extremes of the overlapping interval with t fractional bits, that is, the

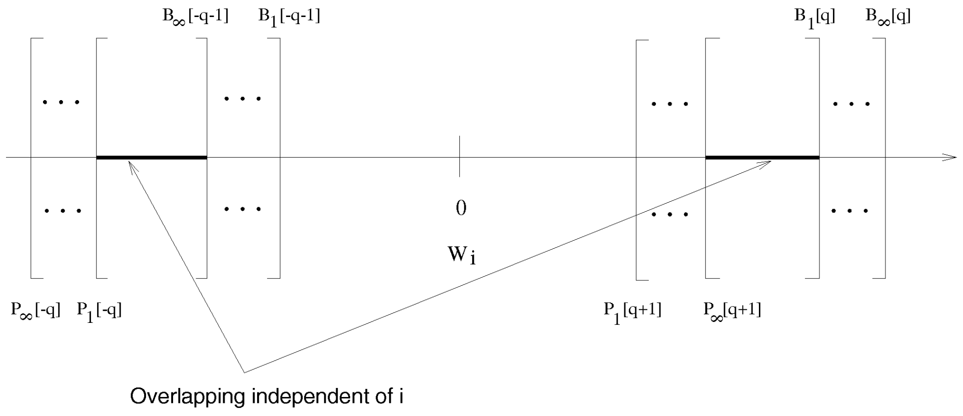


Fig. 2. Overlapping between the selection intervals for different values of i.

overlapping between intervals (Δ) must verify $\Delta > (M + 2)\,2^{-t} - (M + 1)\,2^{-t}$.

2) M is an integer number such that $P(q + 1) < (M + 1)\,2^{-t}$ and $B(q) > (M + 2)2^{-t}$.

These conditions are verified for $t \geq 3$. Consequently, it is enough to assimilate three fractional bits. To calculate the number of bits of the integer part of w_i that must be assimilated, it is necessary to take into account its maximum range. For $i > 0$, we have $|w_i| < 3$ (see (7)) and, consequently, it is necessary to assimilate three bits from the integer part including the sign. The limit in the convergence range guarantees that $-\pi/2 \leq w_0 \leq \pi/2$, and so, for i = 0, it is necessary to assimilate two bits from the integer part, including the sign. Summarizing, a total of five bits must be assimilated in order to obtain $\hat{w}_0$ and six bits for $\hat{w}_i$, with $i > 0$.

We now define $\hat{W}_{\max}[q]$ ($\hat{W}_{\min}[q+1]$) as the maximum (minimum) value of $\hat{w}_i$, for which we select $\sigma_i = q$ ($\sigma_i = q + 1$). In order to guarantee the correct selection, we must have that (see Fig. 3)

$$\begin{aligned} \hat{W}_{\max}[q] &\leq B[q] - 2^{-t+1} \\ \hat{W}_{\min}[q+1] &= \hat{W}_{\max}[q] + 2^{-t} \\ \hat{W}_{\min}[q+1] &\geq P[q+1] \end{aligned} \tag{14}$$

Then we can deduce that

$$P[q+1] - 2^{-t} \leq \hat{W}_{\max}[q] \leq B[q] - 2^{-t+1} \tag{15}$$

being $\hat{W}_{\max}[q] = M\ 2^{-t}$, with M an integer number. For each value of q, we can obtain the upper limit of each selection interval ($\hat{W}_{\max}[q]$) from (15). Taking into account that $\hat{W}_{\min}[q+1] = \hat{W}_{\max}[q] + 2^{-t}$, it is possible to calculate the lower limit of each selection interval. This way, we obtain the selection function for i = 0,

$$\sigma_0 = \begin{cases} +2 & \text{if} \quad 5/8 \leq \hat{w}_0 \\ +1 & \text{if} \quad 3/8 \leq \hat{w}_0 < 5/8 \\ 0 & \text{if} \quad -1/2 \leq \hat{w}_0 < 3/8 \\ -1 & \text{if} \quad -7/8 \leq \hat{w}_0 < -1/2 \\ -2 & \text{if} \quad \hat{w}_0 < -7/8 \end{cases} \tag{16}$$

and, for i > 0,

$$\sigma_i = \begin{cases} +2 & \text{if} \quad \hat{w}_i \geq 3/2 \\ +1 & \text{if} \quad 1/2 \leq \hat{w}_i < 3/2 \\ 0 & \text{if} \quad -1/2 \leq \hat{w}_i < 1/2 \\ -1 & \text{if} \quad -3/2 \leq \hat{w}_i < -1/2 \\ -2 & \text{if} \quad \hat{w}_i < -3/2 \end{cases} \tag{17}$$

Observe that the comparison points are independent from the input operand and, thus, the selection function is easy to implement in hardware. In addition, the selection function described in (16)-(17) is valid both for carry-save redundant arithmetic and nonredundant arithmetic. This is due to the fact that when nonredundant arithmetic is used, the value of w_i is precisely known. The comparison is carried out over the exact values of w_i, and not over estimations.

2.3 Number of Radix-4 Iterations

Once the selection function is obtained, it is necessary to determine the number of iterations needed in order to achieve n bit precision. In this subsection, we demonstrate that the number of iterations is halved with respect to the conventional radix-2 CORDIC algorithm. In the following lemma, we establish the maximum value of w_i in each iteration.

LEMMA 1. *Using the selection functions* (16)-(17) *for* $i > 0$, *we have that*

$$|w_{i+1}| < 2 + (4/3)4^{-2i} \tag{18}$$

PROOF.

1) First, it is necessary to find a bound for w_1. We are going to prove that $|w_1| \leq 2$. For i = 0, we use the selection function given by (16). Even though we

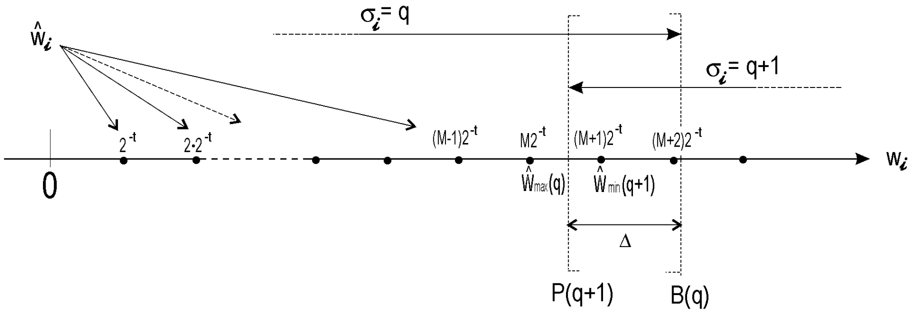


Fig. 3. Conditions to select the minimum value of t.

have several different cases, we are only going to analyze one of them:

- If $3/8 \le \hat{w}_0 < 5/8$ then $\sigma_0 = 1$. Taking into account the error of assimilation for t = 3, $3/8 \le w_0 < 6/8$. Subtracting $A_0[1]$, multiplying times 4 each side and substituting in (4)

$$-2 < -1.64 \le w_1 < -0.14 < 2 \qquad (19)$$

- In the same way, we can prove that $|w_1| \le 2$ for $\sigma_0 = 0, -1, -2y + 2$.

2) We are going to prove (18) by induction. For i = 1, we use the selection function defined by (17). Continuing in the same way as in 1) and taking into account that $|w_1| \le 2$, we can prove the basic condition of the theorem, that is, $|w_2| < 2 + (4/3)4^{-2}$.

3) We assume that $|w_i| < 2 + (4/3)4^{-2(i-1)}$, with $i \ge 2$, and we have to prove that (18) is verified.
As in 1), we will have several different cases, although we are going to analyze just one of them.

- If $-3/2 \le \hat{w}_i < -1/2$, then $\sigma_i = -1$. Taking into account the error of assimilation for t = 3, $-3/2 \le w_i < -3/8$. Subtracting $A_i[-1]$, multiplying times 4 in each side, and substituting in (4), we obtain

$$-4(3/2 + A_i[-1]) < w_{i+1} < -4(3/8 + A_i[-1]) \qquad (20)$$

Taking into account the Taylor series expansion of $\tan^{-1}(q4^{-i})$, we have that $|A_i[q]| > |q - (q^3/3)4^{-2i}|$. On the other hand, as $|A_i[q]| < |q|$, we obtain

$$\begin{aligned}-(2 + (4/3)4^{-2i}) < -4(3/2 + A_i[-1]) < w_{i+1} \\ < -4(3/8 + A_i[-1]) < 2\end{aligned} \qquad (21)$$

- In the same manner, we can prove that (18) is verified for $\sigma_i = 0, 1, -2$, and 2. □

COROLLARY. *In order to achieve n bit precision with the selection function* (16) *and* (17), *it is enough to perform* n/2 *radix-4 iterations.*

PROOF. Particularizing (18) for i = n/2 − 1 and substituting variable w by variable z, we obtain

$$|z_{n/2-1}| < 2^{-n+1} + (4/3)2^{-3n+4} \simeq \tan^{-1}(2^{-n+1}) \qquad (22)$$

□

Comparing with the radix-4 CORDIC presented in [3], the number of iterations needed to carry out a rotation with *n* bits of precision is the same as ours. Nevertheless, the equations associated with the algorithm proposed in [3] have two additions instead of one in each microrotation. Therefore, the complexity of the basic iteration in [3] is double (two adders for the implementation of each data path, and a longer cycle time).

2.4 Scale Factor

Unlike the conventional radix-2 CORDIC algorithm, the scale factor generated in the radix-4 CORDIC algorithm ((2)) is not constant. It depends on the rotation angle through coefficients σ_i and must be calculated for each angle. We do not use the traditional scaling iterations technique as in radix-2 CORDIC processors, but carry out a direct multiplication. In what follows, we describe the procedure to calculate the scale factor.

The scale factor is determined by the n/4 + 1 first microrotations because, in the rest of the microrotations, the scale factor can be considered as one for n bit precision. The scale factor can be obtained in parallel with the evaluation of the microrotations. One solution consists in storing all the possible scale factors in a table as Lin and Sips used in the design of a radix-2 on-line CORDIC processor [17]. As σ_i^2 can present three different values, the number of possible scale factors is $3^{n/4+1}$ (see (2)).

In principle, it would be necessary to have a $(3^{n/4+1} x n)$ bit table. However, we can reduce the size of the table to $(3^{\lfloor n/12+1 \rfloor} x n)$ bits if we analyze the Taylor series expansion of the scale factor:

$$k_i^{-1} = (1 + \sigma_i^2 4^{-2i})^{-1/2} \simeq 1 - \frac{1}{2}\sigma_i^2 4^{-2i} + \frac{3}{8}\sigma_i^4 4^{-4i} + \ldots \qquad (23)$$

For i > n/4, k_i can be approximated by one and we do not have to compensate it. For $i \ge \lfloor n/8 + 1 \rfloor$, k_i can be approximated by the first two terms of its series expansion in n bit precision. For $i \ge \lfloor n/12 + 1 \rfloor$, k_i can be approximated by the first three terms. This way, the size of the table is reduced to

($3^{\lfloor n/12+1 \rfloor} x n$) bits because we only have to store in the table the scale factors generated by the first $\lfloor n/12 + 1 \rfloor$ microrotations. Thus, for example, for n = 32, the size of the table is only (27 × 32) bits. Then the table should have five input bits and 32 output bits. For the case $i \geq \lfloor n/12 + 1 \rfloor$, we can calculate the factor generated in each microrotation by means of addition and shift operations over the scale factor obtained from the table and controlled by the value of σ_i.

In the next section, we will analyze the hardware requirements to compute the scale factor.

3 Architectures

In this section, we present three architectures based on the radix-4 CORDIC algorithm. The first two are general purpose architectures. The first one is word serial and the second unfolded (pipelined or nonpipelined). The third type of architecture is application specific (known angles) and corresponds to a simplification of the previous ones. Although we present architectures with carry-save arithmetic, the corresponding architectures using nonredundant arithmetic can be obtained easily.

3.1 Word Serial Architecture

The conventional word serial radix-2 CORDIC architecture is implemented in nonredundant arithmetic [5], [14]. This way, the duration of the cycle has a complexity of O(n), and the total computation time has a complexity of O(n^2) (if carry ripple adders are used). Faster implementations are obtained if fast carry propagate adders or if redundant arithmetic (carry-free additions) is used. In the last case, the basic cycle has a complexity of O(1) and the total computation time is given by O(n) [7], [16], [19]. In this section, we develop a word serial architecture that implements the radix-4 CORDIC algorithm in carry-save redundant arithmetic. The radix-4 architecture with fast carry propagate adders can be obtained from the carry-save architecture since the hardware blocks are functionally equivalent in both architectures. The compensation of the scale factor is carried out by means of the radix-4 CORDIC algorithm in linear coordinates and rotation mode (multiplication). We must take into account that in the radix-4 CORDIC algorithm in linear coordinates, the scale factor is one. For the selection of the σ_i coefficients, we use the selection function presented in Section 2.2. We incorporate the zero skipping technique [18] to the w coordinate to reduce the number of iterations and the total computation time.

3.1.1 Zero Skipping Technique

The zero skipping technique was initially developed for the division algorithm [18]. This technique can be applied to the radix-4 CORDIC algorithm in the rotation mode. Since iteration w is independent of iterations x and y, the coefficient σ_{i+1} can be evaluated while microrotation i (over x and y) is performed. If $\sigma_{i+1} = 0$, then $w_{i+2} = 4 \cdot w_{i+1}$ and the value of σ_{i+2} can be directly obtained from w_{i+1}. Then, if, in microrotation i, we obtain $\sigma_{i+1} = 0$, we can obtain σ_{i+2} in parallel and the next microrotation would be i + 2 instead of i + 1. With this technique, the number of microrotations is not constant, as it depends on each rotation angle. With a large number of data items, the computation time can be reduced by an important factor. Applying this technique, we do not perform the cycle corresponding to microrotations with $\sigma_{i+1} = 0$. However, if σ_{i+1} and σ_{i+2} are zero, the cycle corresponding to microrotation i + 2 will necessarily have to be carried out. This is because, at most, it is only possible to obtain two coefficients in parallel each cycle.

The zero skipping technique is applied both in the rotation (circular CORDIC) and in the compensation of the scale factor (linear CORDIC). Without this technique, the total number of iterations for n bits of precision is n (n/2 microrotations in circular radix-4 CORDIC plus n/2 microrotations in linear radix-4 CORDIC). In Fig. 4, we present the distribution of the total number of iterations (microrotations and compensation of the scale factor) for 16, 24, and 32 bit precision. These results were obtained from w_0 uniformly distributed over all possible patterns for each of the precision considered. In this graph, we compare the results obtained applying the zero skipping technique to the number of iterations obtained by just counting iterations with σ_i different from zero (optimal number of iterations). As we see in this graph, the zero skipping technique is efficient with results that are close to the optimum. The average number of iterations is 12.16 (11.6 in the optimal case) for 16 bits, 18.58 (17.5) for 24 bits, and 24.97 (23.49) for 32 bits. Therefore, we estimate that we achieve an average reduction of 20 percent in the total number of iterations (4n/5 iterations).

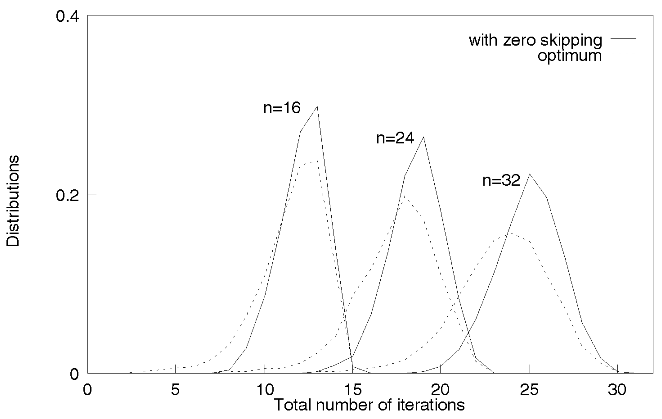


Fig. 4. Distribution of the total number of iterations for 16, 24, and 32 bit precision.

In Fig. 5, we present the hardware implementation of the selection functions containing the zero skipping technique. Initially, for the assimilation process, we would have to use two six bit CLAs (Carry Lookahead Adders), one to obtain σ_{i+1} (the six most significant bits of w_{i+1}, see Section 2.2) and another to obtain σ_{i+2} (the next six bits after the second most significant bit of w_{i+1}). Therefore, a total of eight bits are assimilated. In order to optimize the hardware requirements, we use three four bit CLAs together with the conditional addition technique. The assimilated values are passed on to the selection tables TM (selection of σ_{i+1}) and TL (possible selection of σ_{i+2}), which operate in parallel and carry out the comparison between the assimilated values

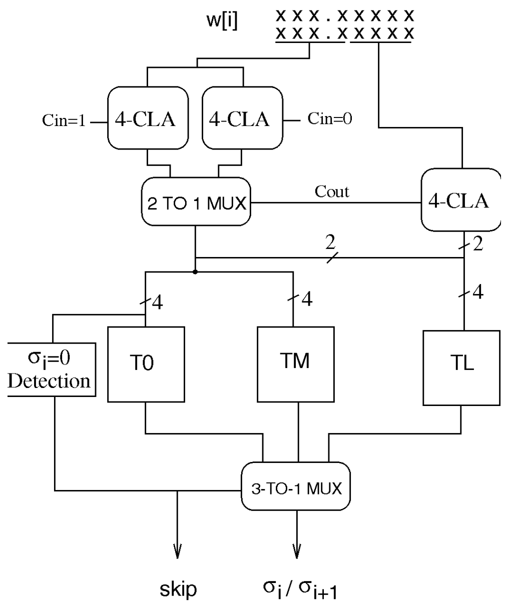


Fig. 5. Hardware implementation of the zero skipping technique.

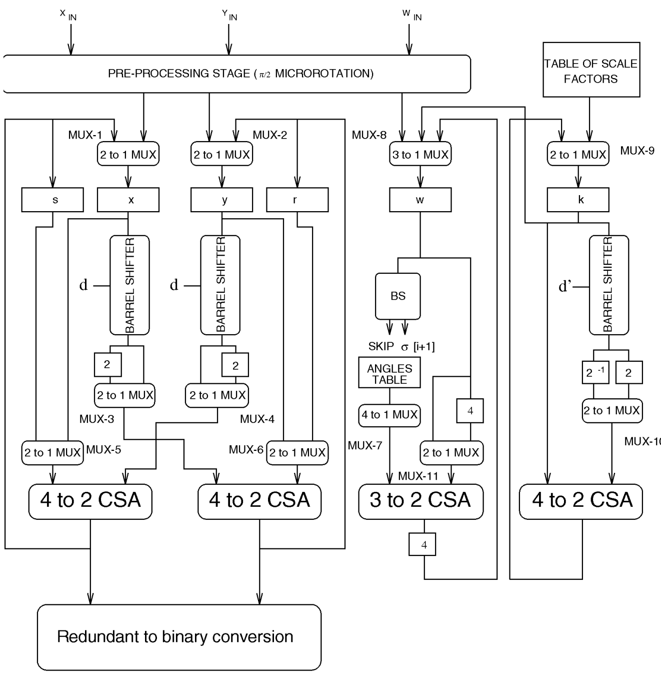


Fig. 6. Architecture of the word serial radix-4 CORDIC processor.

and the comparison points (±1/2 and ±3/2) of the selection function given by expression (17). Table T0 implements the selection function given by expression (16) for the first microrotation. The hardware cost of the skipping technique is approximately the right-hand side data path of Fig. 5.

3.1.2 Structure of the Processor

In Fig. 6, we show the architecture of the word serial radix-4 CORDIC processor. Basically, it consists of a preprocessing unit where an initial $\pi/2$ rotation can be carried out in order to obtain a convergence range of $[-\pi, \pi]$, three processing paths for the x, y, and w coordinates, and a section where the inverse of the scale factor (K^{-1}) is computed.

In order to explain the processor structure and data flow, we are going to follow an example. It consists of a full rotation in circular coordinates for 16 bit precision. The time diagram is shown in Fig. 7. Basically, we program the processor in circular mode for the unscaled rotation computation (cycles 1 through 7) and in linear mode for the scale factor compensation (cycles 8 through 14). Note that, in order to obtain a suitable synchronization of the σ_i computation, the w coordinate is one cycle ahead of the x and y coordinates.

During cycle 0, the preprocessing unit carries out a rotation of $\pi/2$ and, by means of MUX 1, 2, and 8, the registers x, y, and w are loaded with the output values x_0, y_0, and w_0.

Without losing generality, we assume that $\sigma_0 \neq 0$; during cycle 1, σ_0 is obtained from w_0 (see (4)). In order to do this, the BS block (see Fig. 5) obtains σ_0 and controls MUX 7, 11, and 8 to load register w with the addition of w_0 and $A_1(\sigma_0)$ (contained in the angle table).

σ_1, w_2, x_1, y_1 are obtained in cycle 2. σ_1 and w_2 are produced in a similar manner to the previous cycle; in order to obtain x_1 and y_1 ((1), with i = 0), the barrel shifters carry out a shift d (d = 2i), which is 0 in this case (factor 4^0 in (1)), and it will be incremented in every cycle. MUX3 and MUX4 are in charge of performing the multiplication of coefficient $|\sigma_0|$ obtained in the last cycle and each one of the x and y coordinates. This multiplication is carried out by means of the selection of the coordinate ($|\sigma_0| = 1$) or the value of the coordinate shifted one position to the left in a hardwired manner ($|\sigma_0| = 2$). If $\sigma_0 = 0$, the loading of the registers x and y is not activated. MUX-5 and 6 are ready to effect (1) and MUX-1 and 2 carry out the update of the registers x and y with x_1 and y_1.

Cycle 3 is similar to the last one, whereas, in cycle 4, the value σ_3 obtained by BS block is 0 and, therefore, BS block generates σ_4 in parallel (skipping technique) and, moreover, controls MUX-11 and 7 in order to select $4w_3$ and $A_5(\sigma_4)$ like the inputs to the 3 to 2 CSA. The value of shift d is now incremented by two units instead of one unit, and will be used in the barrel shifters on the next cycle. Cycles 5, 6, and 7 are similar to cycle 2.

For the word serial architecture and a moderate precision, it is more efficient to perform the approximation of the scale factor with just two terms as, this way, the control is simplified. Consequently, the scale factor table has a size of $3^3 \times 16$ ($3^{\lfloor n/8+1 \rfloor} \times n$, see Section 2.4). However, for n > 32, the table would be too large, and, then, the approximation with three terms should be used. Assuming the case of a linear approximation with two terms, the scale factor obtained from the table is introduced in an addition and shift unit, where the linear approximation of the factor generated between microrotations $\lfloor n/8 + 1 \rfloor$ and $n/4$ is carried out, and where $d' = 4i$ (see Fig. 6 and (23)). MUX-10 has similar

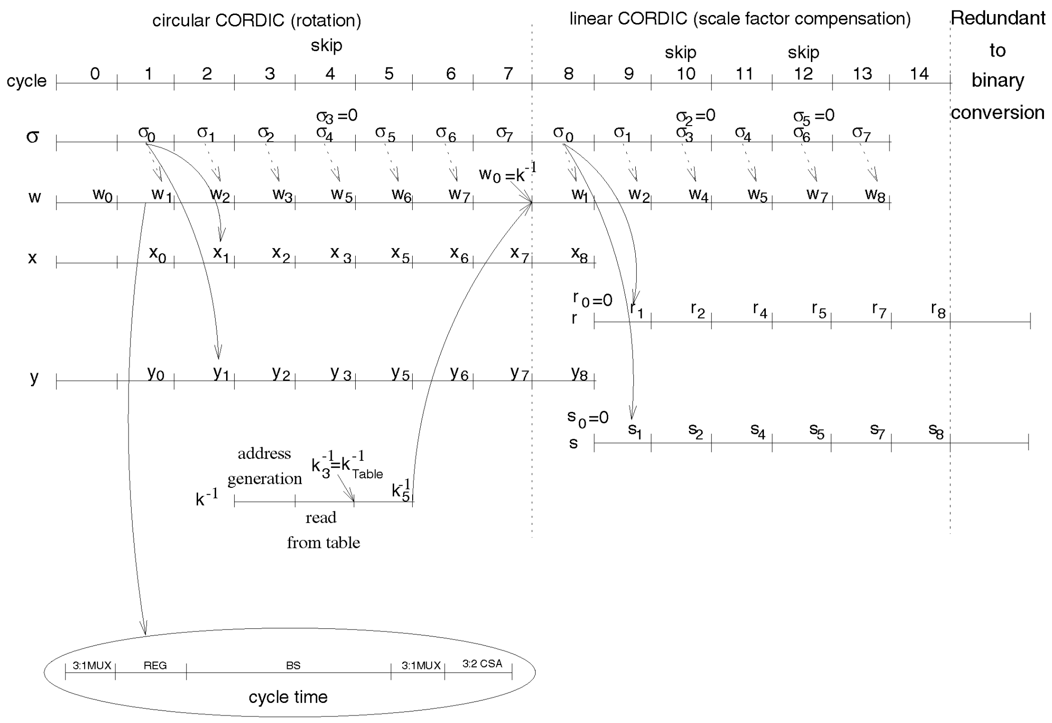


Fig. 7. Time diagram for the word serial architecture.

function to MUX-3 and 4. In our case, the scale factor obtained from the table is addressed and loaded in register k during cycle 4. In cycle 5, the scale factor obtained from the table is introduced in the addition and shift unit where the linear approximation of the factor ((23)) generated in microrotation 4 is performed. During this cycle shift, d′ = 8 and is obtained from shift d = 4. The value is now loaded onto register k (MUX-9), and will be transferred to register w during cycle 7 (MUX-8). Now the system is ready to carry out the scale factor compensation.

Once the rotation has been evaluated, in the next iterations, the radix-4 CORDIC algorithm is performed in linear coordinates in order to evaluate the multiplication of the scale factor times coordinates x and y. It is necessary to carry out two parallel multiplications, corresponding to $K^{-1}\cdot x$ and $K^{-1}\cdot y$. The iteration can be described as follows

$$\begin{aligned} r_{i+1} &= r_i + \sigma_i 4^{-i} x \\ s_{i+1} &= s_i + \sigma_i 4^{-i} y \\ w_{i+1} &= 4(w_i - \sigma_i) \end{aligned} \tag{24}$$

where $w_0 = K^{-1}$, $r_0 = 0$, $s_0 = 0$, and x and y are the values in registers x and y after the rotation has been evaluated. Now we use two auxiliary registers s and r to support (24). In order to carry out this iteration, MUX-8 selects the scale factor that was calculated and loads it into register w. Registers x and y are enabled to store the value generated in the rotation. MUX-5 and MUX-6 select the output of registers r and s, respectively. The selection function used is the same as for the case of circular coordinates and i > 0. Therefore, the procedure for cycles 8 to 14 is similar to the initial cycles (1 through 7) described before.

The operands stored in registers r and s are in redundant arithmetic. Then, to perform the final conversion to nonredundant arithmetic, we include two fast carry propagate adders.

In Fig. 7, we display the composition of the cycle time. The total computation time is obtained by multiplying the average number of cycles times the duration of each cycle and adding the delay associated to the final arithmetic conversion. The total average number of iterations is $4n/5$, taking into account the average 20 percent reduction when applying the zero skipping technique.

An important reduction in area (reducing also time performance) can be obtained if nonredundant arithmetic and fast carry propagate adders are used (see Section 4.1). The resultant architecture is similar to Fig. 6, but uses a fast carry propagate adder (i.e., Cascaded Carry Lookahead adders CCLA), instead of 4-2 CSAs, and eliminates the redundant to binary conversion module. The hardware to obtain the selection function is simplified by eliminating each 4-CLA of Fig. 5.

3.2 Unfolded and Pipelined Architecture

The pipelined CORDIC architecture consists of a linear array of modules, in each one of which a microrotation is carried out. The shifts can be hardwired in, as each module always performs the same microrotation. The architecture we present uses carry-save arithmetic. However, from this redundant architecture, it is easy to obtain an implementation with nonredundant adders. For the selection of the σ_i values, we combine two different techniques. For $0 \le i < \lceil n/6 \rceil$, we use iteration w, together with the selection function presented in Section 2.2. For $i \ge \lceil n/6 \rceil$, we can eliminate iteration w, as it is possible to perform a parallel prediction

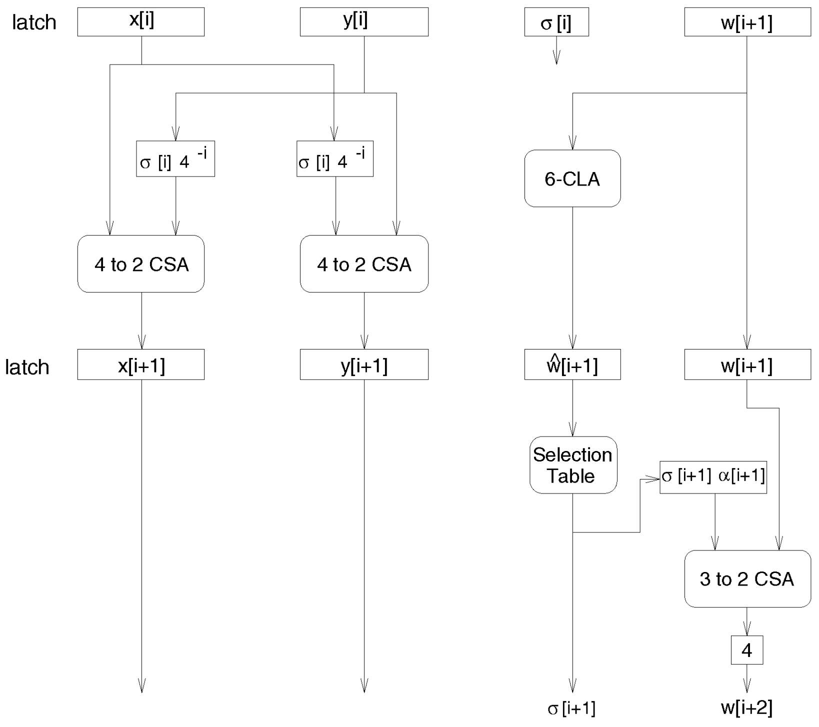


Fig. 8. Architecture of a microrotation in the pipelined design.

of the σ_i values from the value of w_p ($p = \lceil n/6 \rceil$), because $\tan^{-1}(\sigma_i 4^{-i}) \approx \sigma_i 4^{-i}$. If we express w_p in signed digit radix-4, we obtain

$$w_p = \sum_{j=0}^{n/2-p-1} c_j 4^{-j} = \sum_{j=0}^{n/2-p-1} 4^p \tan^{-1}(\sigma_j 4^{-p-j}) \simeq \sum_{j=0}^{n/2-p-1} \sigma_{j+p} 4^{-j} \quad (25)$$

with $c_i \in \{-2, -1, 0, 1, 2\}$. Once w_p is known, the selection function is simply $\sigma_{p+j} = c_j$ with $j \geq 0$. As coordinate w is expressed in carry-save redundant arithmetic, we will convert it to signed digit radix-4 arithmetic. A recoder of this type is described in [8]. This procedure permits obtaining all the σ_{p+j} for the microrotations with $j \geq 0$ in parallel, and it is not necessary to implement iteration w in these microrotations. The hardware required to carry out this conversion is similar to a 4-to-2 CSA (Carry Save Adder). This way, the delay of the stages is not affected, as we introduce parallel operations.

In Fig. 8, we present the architecture corresponding to a microrotation that uses the selection functions in iteration w. These microrotations are pipelined in two stages in order to increase the throughput. The pipelining is carried out so as to make the critical path that of coordinates x/y. In a first stage, the assimilation of the six most significant bits of w_{i+1}, as well as microrotation i over x_i and y_i, are carried out in parallel. In the second stage, the values of x_i and y_i are not modified, while, in iteration w, we calculate the value for σ_{i+1} from the bits that were assimilated in the previous stage and obtain w_{i+2}. This pipelining only affects microrotations with $i < p$. For microrotations with $i \geq p$, iteration w is not implemented and the double pipelining of each microrotation is not necessary.

In order to calculate the scale factor generated, we use the method proposed in Section 2.4. In Fig. 9, we present the architecture for the calculation of the scale factor for 32 bit precision. For this case, the scale factor table must store (27 × 32) bit words. In cells A and B, we approximate, using three terms of the series expansion. Note that cells A and B are different because the scale factor obtained from the table is expressed in conventional nonredundant arithmetic, while the input to cell B is expressed in carry-save form. After this, we have four type C cells in which we perform the linear approximation. Finally, we carry out a conversion from carry-save representation to radix-4 signed digit representation using a recoder like the one described in [8] in order to obtain the final coded scale factor for the multiplication.

In Fig. 10, we present the whole architecture of a 32 bit processor. The first p (p = 6 for 32 bits) microrotations are pipelined into two blocks and implement iteration w with the selection function. The remaining microrotations only implement coordinates x and y. The calculation of the scale factor is carried out in parallel with the microrotations. When the last microrotation ends, the scale factor has already been calculated and is coded in order to carry out the final multiplication over the x and y coordinates.

3.3 Application Specific Architecture

There are a large number of applications, such as orthogonal transforms [12], [13], image processing [4], in which it is only necessary to evaluate the rotation of vectors over angles that are known beforehand. In this case, it is possible to use a radix-4 CORDIC processor with less hardware com-

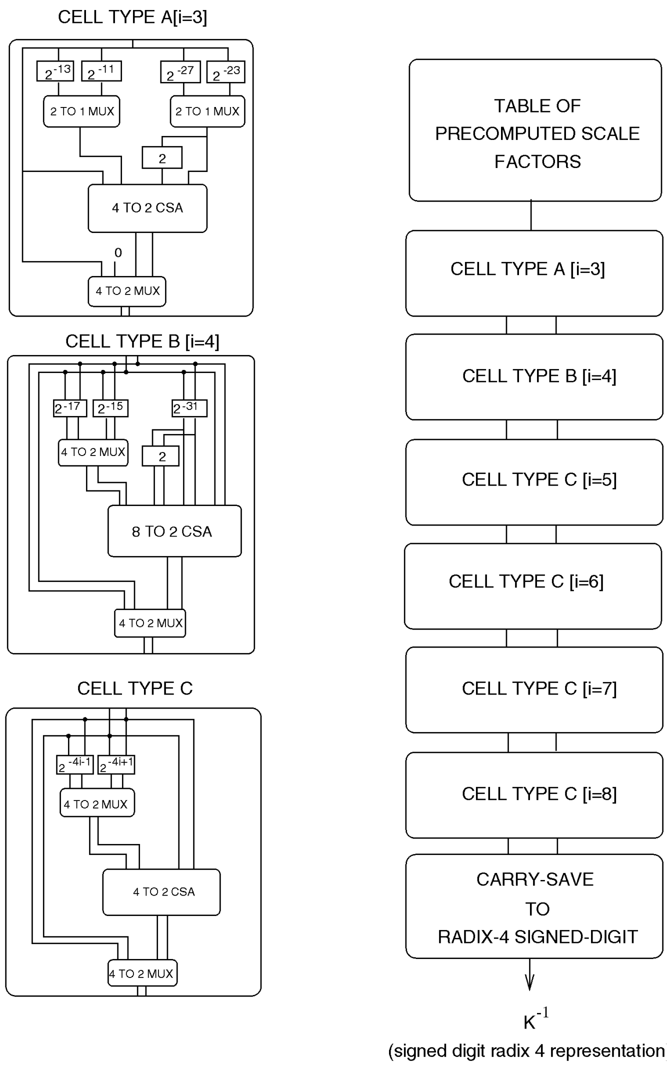


Fig. 9. Architecture for the computation of the scale factor for 32 bit precision.

plexity and greater speed than the general purpose radix-4 CORDIC processor.

If the rotation angles are known beforehand, then the σ_i values can be precomputed, and stored in a look-up table, that is, it is not necessary to implement iteration w ((4)) in order to obtain the sequence of σ_is, saving the hardware associated with the double segmentation of the first p microrotations in a pipelined architecture. In the word serial architecture, coordinate w is not necessary either, but we can reduce the number of microrotations using the zero skipping technique. Finally, it is possible to increase the precision by one bit (n + 1 bits for n/2 radix-4 iterations) if we decompose the angular interval [0, π/2) into two subintervals. If the angular interval is [0, π/4), we can choose the elementary angles as $\alpha_i(\sigma_i) = \tan^{-1}(\sigma_i 2^{-1} 4^{-i})$, transforming the equation for *x* and *y* in (1) into

$$x_{i+1} = x_i + \frac{1}{2}\sigma_i y_i 4^{-i}$$
$$y_{i+1} = y_i - \frac{1}{2}\sigma_i x_i 4^{-i} \qquad (26)$$

When the angles are in the interval [π/4, π/2), (26) transforms into

$$x_{i+1} = y_i + \frac{1}{2}\sigma_i x_i 4^{-i}$$
$$y_{i+1} = -x_i + \frac{1}{2}\sigma_i y_i 4^{-i} \qquad (27)$$

It can be easily seen that (27) is obtained from (26) using the trigonometric equivalences for the complementary angles.

The hardware implementation of the microrotation associated with (26) and (27) needs a multiplexor added to each input of coordinates *x* and *y*, respectively. This multiplexer selects (26) or (27) as a function of a new bit that should be added to the coding of each angle. The factor of ½ in (26) and (27) is contained in the variable shifters.

LEMMA 2. *In the convergence range* [0, π/2), *we need* n/2 *radix-4 iterations in order to achieve* n + 1 *bit precision.*

PROOF. Taking the partition of the interval [0, π/2) into two subintervals into account and using the definition of elementary angles, if, in (5), we substitute the term (2/3)$A_i(1)$ by $A_i(½)$ and follow the steps of Theorem 1 in Section 2, we conclude that $|w_{n/2}| < A_{n/2}(2)$, and, therefore,

$$|z_{n/2-1}| \leq \tan^{-1}(2^{-n}) \qquad (28)$$

□

In application specific architectures, the decomposed angles are stored as a sequence of σ_is. This way, the word serial architecture only requires performing the iterations corresponding to the values of σ_i that are not zero (zero skipping). In order to reduce the number of iterations of the decomposition of the angles, the σ_i coefficients must have the largest possible number of zeroes. In fact, the optimal decompositions are those with the largest number of coefficients set to zero. These decompositions can be found by means of an intensive search over all possible decompositions. However, the problem becomes very complex as the value of n increases. A simple alternative is to use selection functions (16) and (17), but we would obtain nonoptimal decompositions. Next, we propose an algorithm that obtains a quasi optimal decomposition of the angles:

```
Initialization: $w(0) = \theta$, $\{\sigma_i = 0, 0 \leq i \leq ½n - 1\}$, $k = 0$;
While $|w(k)| \geq A_{n-1}(1)$ Do
1. Choose $j, 0 \leq j \leq ½n - 1$ such that
       $||w(k)| - A_j(\mu)| = \min||w(k)| - A_i(\mu)|$, with
       $j\_old[k-1] \leq i \leq ½n - 1$ and $1 \leq \mu \leq 2$
2. if $j = j\_old(k-1)$ and $k \geq 1$ then
       {
           while $[j\_old(k-2) = j - 1$ and $|\sigma_{j\_old[k-2]}| = 2$
           and $k > 1]$ Do $\{\sigma_j = 0, k--, j--\}$
           $\sigma_j = 0, k--, j--, \mu = 1$;
       }
3. $w(k+1) = 4(w(k) - A_j(\sigma_j))$, where $\sigma_j = \mathrm{sign}(w(k))\mu$
```

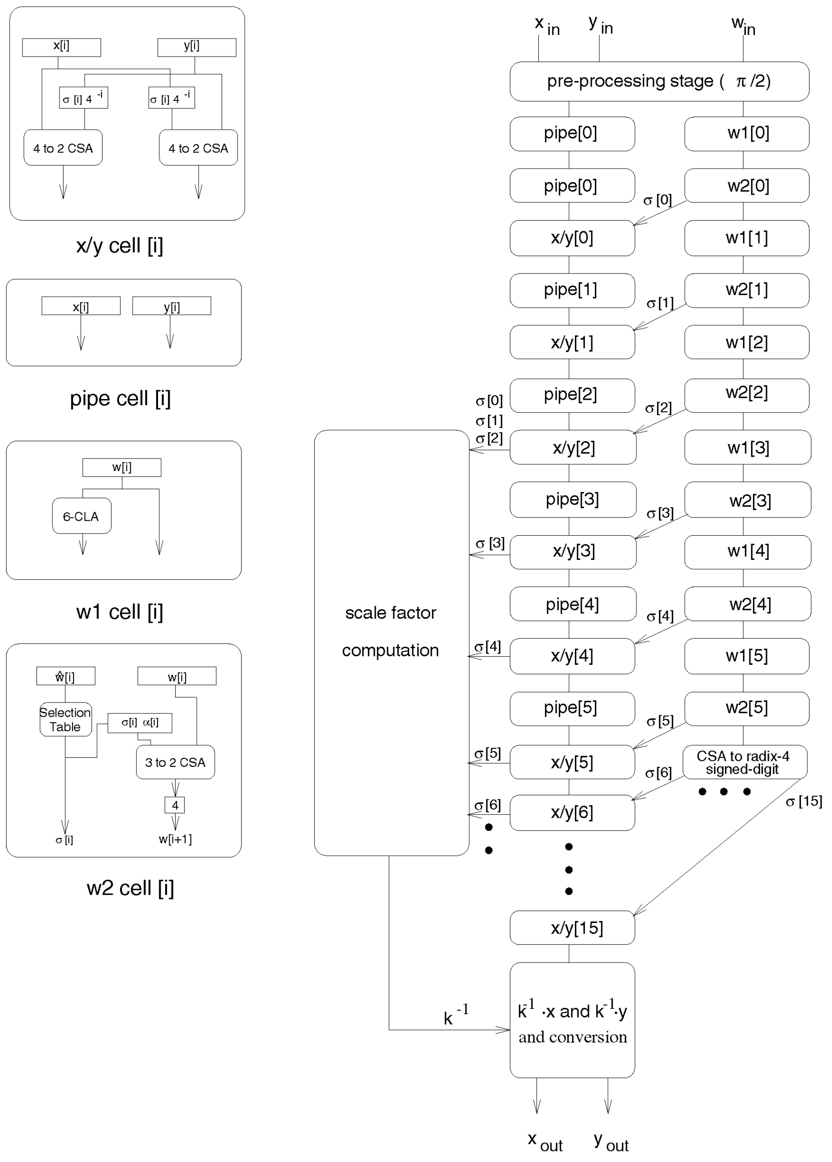


Fig. 10. Architecture of the pipelined CORDIC processor for 32 bit precision.

j_old[k] = j
k++.

This algorithm is similar to the one presented by Hu and Naganathan [12] for the radix-2 case.

After the pertinent initializations, step 1 of the algorithm seeks the elementary angle $A_j(\mu)$ that most closely approximates the angle that still has to be rotated, that is w(k). In the second step, it tests if the selected iteration *j* coincides with the iteration that has just been calculated in the previous cycle *j_old*(*k* – 1) (note that each radix-4 iteration admits two different shifts). If this is so, we must go back and select the next elementary angle of the previous cycle. Summarizing, this step prevents the repetition of microrotations. Step 3 updates the angle that still has to be rotated. The algorithm ends when the angle remaining to be rotated is smaller than the precision.

In order to prove the convergence of the algorithm, we will see that the sequence w(k) verifies that $|w(k+1)| < 4|w(k)|$. Steps 1 and 2 of the algorithm guarantee that the rotation angle of an iteration is larger than that of the next (j > j_old(k – 1)). From step 3, we can claim that

$$|w(k+1)| = |w(k) - A_j(\sigma_j)| = ||w(k)| - A_j(|\sigma_j|)|$$

Taking into account steps 1 and 2 and that $|w(k)| \geq A_{n-1}(1) > 0$, we have

$$|w(k+1)| \leq 4||w(k)| - A_{n-1}(1)| < 4|w(k)|$$

Summarizing, the design of an application specific radix-4 CORDIC architecture is an efficient alternative because it reduces hardware, decreases the latency of the pipelined design, increases the speed of the word serial design, and improves the precision of the results. In addition, we provide an effective algorithm for the calculation of the coefficients that must be stored in the look-up table.

4 Evaluation

We have proposed several new architectures based on a new radix-4 CORDIC algorithm. In this section, we perform a comparative analysis with equivalent solutions proposed in the literature. We compare the word serial architectures, presented in Section 3.1, to the conventional word serial architecture (with carry propagate adders) and redundant architectures [7], [16], [19]. After this, we compare the general purpose unfolded architecture with carry-save arithmetic of Section 3.2 to those proposed in [6] and [20]. We also compare the radix-4 unfolded architecture with a radix-2 unfolded architecture, both with nonredundant arithmetic, using fast CCLA adders. Finally, in order to evaluate the application specific design of Section 3.3, we compare the algorithm for obtaining the σ_i values to the angle recording method proposed in [12].

We would like to emphasize that a true comparison between different implementations is possible only if actual implementation is considered and circuit level simulations are carried out. Therefore, we present a rough and first order approximation comparison based on the gate count for the hardware complexity and the number of gate levels for the delays. Nevertheless, it can express the general trend among the different designs.

For this first order comparison, we will constrain to a model that considers *x-input* gates (and, or, nand, nor) to have a delay of x/2 gate units and a hardware complexity of x/2 gates, *x-to-1* multiplexers and *x-input* xor/xnor gates to have x unit delays and a hardware complexity of 1.5x gates, a buffer attaching *x* gates to have a delay of 1 + x/32 gate units, barrel shifters of n bits to have a complexity of n^2 gates and a delay of $2 \log_2 n$ units, an n-bit register to have 4n gates and six unit delays, a 4-CLA to have 38 gates and 7.5 unit delays, and a CCLA of n bits (adder/subtracter with block size of four) to have 14n gates and 17, 23, or 25 unit delays if n = 16, $16 < n \le 32$ or $32 < n \le 64$, respectively. For the comparison, we have assumed standard logic implementations for the different hardware blocks, although other alternatives with optimized blocks exist. For n iterations, we have considered a datapath of n bits to simplify the comparison. Actual implementations should incorporate overflow and guard bits.

4.1 Word Serial Architecture

For the radix-4 redundant architecture, taking into account our model of delays, for practical precision, the critical path corresponds to iteration w. The resulting cycle time is (see Fig. 6):

$$T_{radix\text{-}4} = t_{reg} + t_{4\text{-}CLA} + t_{2-1mux} + t_{TM} + t_{3-1mux} + t_{buffer} + t_{mux\text{-}7} + t_{3-2CSA} + t_{mux\text{-}8} \quad (29)$$

where the four first terms following the delay of the register correspond to the module BS of Fig. 6, t_{TM} being the delay of the corresponding selection table (see Fig. 5). t_{buffer} is the delay of the buffers which attach to MUX-7. In a nonredundant implementation, the term $t_{4\text{-}CLA}$ is eliminated and t_{3-2CSA} is substituted by t_{CCLA}. This matches the delay of the w data path and the x, y data path, for precision between n = 16 and n = 64. The average number of iterations is 4n/5, taking into account the average reduction of 20 percent in the number of iterations due to the skipping the zeroes technique. The worst case number of iterations is n.

In [19], Takagi et al. present a design based on the correcting microrotation method for the implementation of the rotation mode. The σ_i values can be obtained from the assimilation of a reduced number of bits in each iteration. If t + 3 bits are assimilated, then a repetition is needed every t microrotations to assure convergence. For comparison purposes, we have considered t = 5 (largest value of t that makes x/y the critical path). Furthermore, we have assumed that no correcting microrotations are needed for i > n/2 [16]. Basically, the *x* and *y* data paths are the same as in a standard CORDIC (except obvious changes by using redundant arithmetic) and the angle data path adds a multiplexer to support the correcting iterations. The critical path for this design is:

$$T_{cor\text{-}micr} = t_{reg} + t_{buffer} + t_{shifter} + t_{3:1mux} + t_{4:2CSA} + t_{2:1mux} \quad (30)$$

The number of iterations is n basic microrotations, n/10 correcting microrotations and iterations for compensate the scale factor. The number of scalings to compensate the scale factor can be approximated by n/3. However, when repetitions are introduced, some simplifications are introduced in the scaling operation. Then, for the correcting microrotation method, we assume n/4 scaling iterations.

In [7], Duprat and Muller present a design of a CORDIC processor in redundant arithmetic based on performing two rotations in parallel in two CORDIC units (branching CORDIC method). Only four bits are assimilated in each iteration. This method does not use any additional iteration to work in redundant arithmetic as [16] does. The critical path for the design presented in [7] is:

$$T_{branching} = t_{reg} + t_{buffer} + t_{shifter} + t_{3:1mux} + t_{4:2CSA} + t_{3:1mux} \quad (31)$$

The number of iterations is n basic microrotations, and n/3 scaling iterations.

We also consider, for the comparison, the conventional radix-2 CORDIC architecture with fast carry propagate adders (CCLA). For this case, the number of iterations is n basic microrotations and n/3 scaling iterations.

Fig. 11 shows the average latency (gate delays) and complexity (gate count) of these designs for n = 16 through n = 64. For the hardware complexity, we have not taken into account the tables for storing the microrotation angles, or the table to store the scale factors, since its implementation depends heavily on the technology. However, the hardware complexity of the tables should not add too much complexity comparing with other very costly hardware elements like the barrel shifters. The top graphs (Figs. 11a and 11b) correspond to architectures with scale factor compensation, and the bottom graphs (Figs. 11c and 11d) corresponds to the case where only the latency of the microrotations have been taken into account (no scale factor compensation). Table 1 shows the ratios for the gate count and number of gate levels (referred to the carry-save radix-4 design) for n = 32 bits. We can see that the radix-4 CORDIC design has the lower average latency, while the radix-2 CORDIC design with CCLA has the higher latency and the minimal complexity.

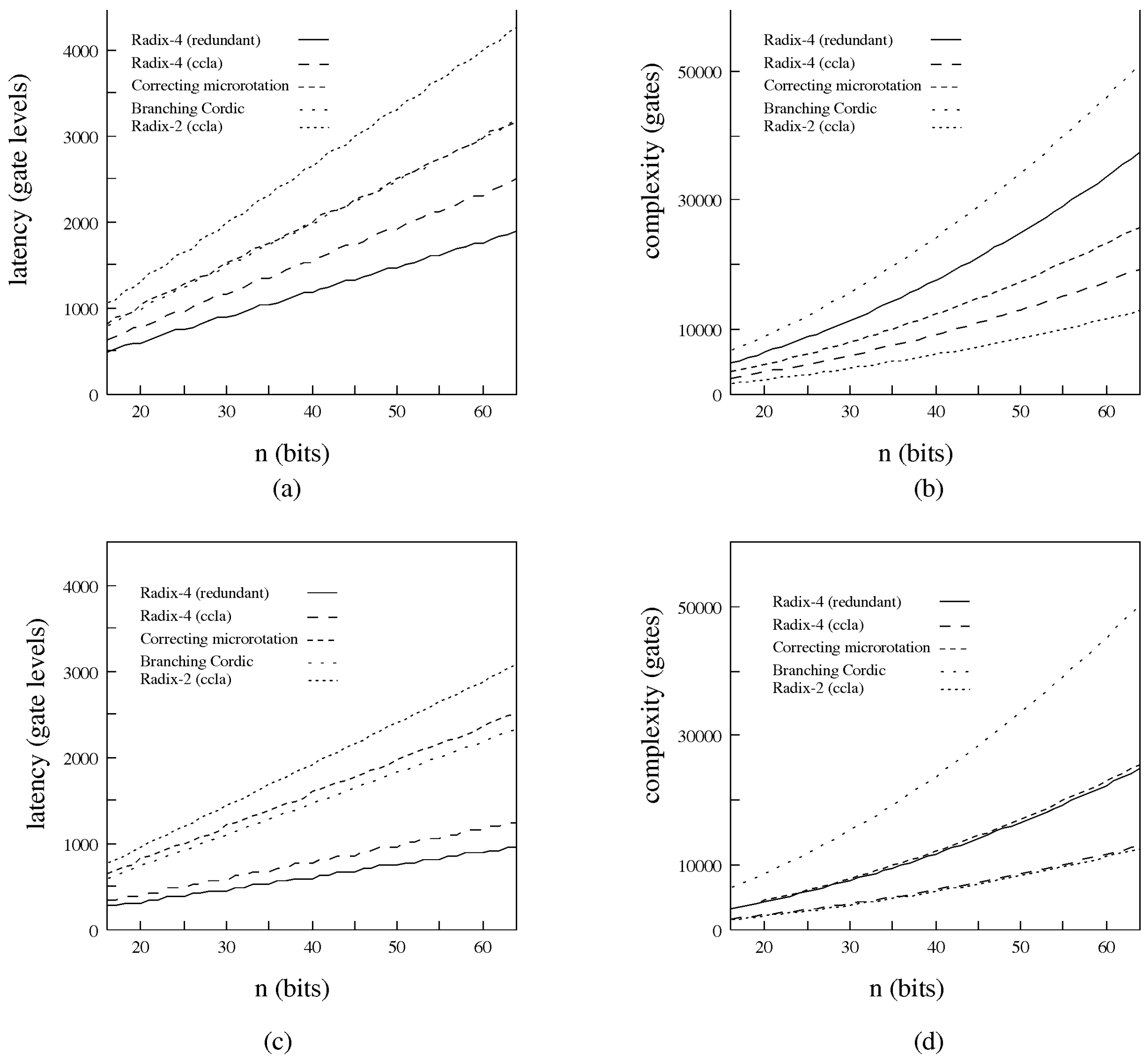


Fig. 11. Comparison of word serial architectures.

TABLE 1
GATE COUNT AND GATE LEVELS RATIOS FOR N = 32

Design	Duprat & Muller [7]	Takagi et al. [19]	Radix-2 (CCLA)	Radix-4 (CCLA)	Radix-4 (CSA)
Gate count	1.4	0.7	0.3	0.5	1
Gate levels	1.6	1.7	2.2	1.3	1

Comparing the nonredundant based designs, we can see that the radix-4 design is twice as fast as the radix-2 design without multiplying by two the complexity (similar complexity if scale factor compensation is not considered, see Fig. 11d).

Comparing the redundant arithmetic designs, the complexity of the proposed carry-save radix-4 architecture is greater than the correcting microrotation based design and less than the branching CORDIC design, but with the lowest latency (see Figs. 11a and 11b). The complexity of the radix-4 design is similar to the correcting microrotation based design if no scaling factor compensation is considered (see Fig. 11d).

Comparing the redundant based designs with nonredundant designs, we can see that the radix-4 architecture with CCLA adders has a lower complexity than each one of the redundant arithmetic based designs, maintaining a lower latency than branching and correcting based architectures, although with more latency than the carry-save radix-4 design. The above comparison establishes the trade-off between hardware complexity and latency between the redundant and nonredundant architecture with fast carry propagate adders.

4.2 Unfolded and Pipelined Architectures

In this subsection, we consider two unfolded architectures that have been proposed for the rotation mode and

TABLE 2
COMPARISON OF PARALLEL NONPIPELINED DESIGNS

Design	Radix-2 (CCLA)	Timmerm.[20]	Radix-4 (CCLA)	Radix-4 (CSA)
Redundant	No	Yes	No	Yes
Latency (t_g)	17n (n = 16) 23n (16 < n ≤ 32) 25n (32 < n ≤ 64)	$7n + t_{conv}$* + $10(\log_2(n) - 1)$	11n (n = 16) 14n (16 < n ≤ 32) 15n (32 < n ≤ 64)	$8n + t_{conv}$
Crossed shifts (x/y)	2n	5n/4 + 2	n	n
Additions in w	n	15	n/6	n/6
Other hardware		2·W(n/2+4)**	n/4 (shift&add) Table $3^{\lfloor n/12+1 \rfloor}$ *x* n	n/4 (shift&add) Table $3^{\lfloor n/12+1 \rfloor}$ *x* n

* $t_{conv} = 16t_g$ $(n = 16)$, $t_{conv} = 22t_g$ $(16 < n \leq 32)$, $t_{conv} = 24t_g$ $(32 < n \leq 64)$
** *W(r) means addition of r operands in a tree of 4-to-2 CSA adders*

TABLE 3
COMPARISON OF PIPELINED DESIGNS

Design	Radix-2 (CCLA)	Dawid [6]	Radix-4 (CCLA)	Radix-4 (CSA)
Redundant	No	Yes	No	Yes
Latency (stages)	n	3n + 2	n/2 + 1	2n/3 + 4
Cycle (tg)	26 (n = 16) 32 (16 < n ≤ 32) 34 (32 < n ≤ 64)	13	31 (n = 16) 37 (16 < n ≤ 32) 39 (32 < n ≤ 64)	21
Crossed shifts (x/y)	2n	2n	n	n
Additions in w	n	n	n/6	n/6
Other hardware		2.5n rows of registers	n/4 (shift&add) Table $3^{\lfloor n/12+1 \rfloor}$ *x* n	n/4 (shift&add) Table $3^{\lfloor n/12+1 \rfloor}$ *x* n

carry-save redundant arithmetic [6], [20], and a conventional nonredundant radix-2 unfolded architecture. We have considered two types of architectures, unfolded pipelined and unfolded nonpipelined. For our algorithm, we have considered two different architectures, one with redundant adders and another one with conventional nonredundant fast adders. We perform the comparison of redundant architectures and nonredundant architectures separately. However, the numbers presented in the tables could be useful for designers to establish a first order tradeoff between redundant and nonredundant architectures. We have not considered in these architectures the cost in terms of time and hardware of the scale factor compensation, which should be similar for all architectures, although some simplifications can be made in the case of the algorithms with constant scale factor. However, we have considered the necessary hardware to compute the scale factor when a nonconstant scale factor is generated.

In Table 2, we present the results corresponding to the comparison of the unfolded nonpipelined architectures. In Table 3, we present the latency, cycle time, and significant hardware associated with the unfolded pipelined designs as compared to our design. For the evaluation of the hardware complexity, we have considered the number of crossed shifts in the x/y datapath, the number of additions in w, and other representative hardware specific to each architecture. We have not considered a comparison based on a gate count since, for the unfolded architectures, the complexity of the design is dominated by the routing area due to the hardwired shifts. This effect is even more important when redundant arithmetic is used. Although the comparison based on these terms is very rough, the number of crossed shifts in the x/y datapath is a good first order approximation to compare the hardware complexity of these architectures. Note that each crossing shift has also associated an adder. Furthermore, the hardware elements of the architectures with redundant adders should have double complexity as compared to those corresponding to nonredundant architectures.

The architecture proposed by Timmermann et al. [20] is unfolded but not pipelined. The latency of the processor is reduced by an important factor at the cost of an irregular design. It is difficult to perform a $\pi/2$ initial rotation or the rotation of index $i = 0$ for circular coordinates, as it would force a conversion from redundant to conventional arithmetic for coordinate z just after the first microrotation, which is very costly in area or in time. The hardware and time cost of this architecture is shown in Table 2. The latency was evaluated considering that path x/y was slower than path z. This implies that the elements for conversion between redundant and conventional arithmetic have to be fast, making their cost, in terms of area, high.

Comparing with our design with redundant adders, according to the data presented in Table 2, the hardware associated to iterations z, x, and y is higher than in our design. In addition, the hardware we use for calculating the scale factor is less than that used in the design by Timmermann et al. to keep it constant. For example, when $n = 32$ in our architecture, we have a complexity of 32 crossing shift operations over x/y, whereas, in the case of Timmermann et al.'s architecture, it is 42. Both architectures present approximately the same latency (see Table 2). Furthermore, the range of convergence in the angle of our design is $[-\pi, \pi]$ and in Timmermann et al.'s it is only $[-1, +1]$. The architecture presented in [20] is not adequate for its pipelined implementation due to the intermediate conversions from redundant to conventional arithmetic, and it is very irregular because of the use of tree structures for the parallelization of the additions.

Comparing the radix-2 conventional design to our architecture with fast nonredundant adders (CCLA) adders), the latency of our design is significantly reduced and also the number of crossing shifts. On the other hand, our architecture needs a table to store partial scale factors and $n/4$ shift and addition operations to complete the computation of the scale factor. Note that we compare with a baseline radix-2 conventional architecture. This architecture permits certain improvements like the termination algorithms [20], reducing hardware, and latency. In this case, we estimate a reduction of $n/2$ crossing shifts for our architecture.

For the redundant pipelined architecture that we propose, the critical path of the stage delay corresponds to the sum of the delays of a register, a buffer, a 3 to 1 multiplexer, and 4 to 2 carry-save adder/subtractor. Based on the model of delays used, this corresponds to a delay of 21 t_g. The total latency is $2n/3 + 3$ stages, including two stages for the final conversion from redundant to nonredundant representation.

The architecture proposed by Dawid and Meyr [6] is pipelined and uses carry-save arithmetic. The pipelining is carried out at the full adder level, allowing for high operation speeds. In Table 3, we present the data corresponding to the hardware and time complexity. According to this data, the complexity in the number of crossing shift operations of our architecture (redundant) is n as compared to the 2n complexity of Dawid and Meyr's design. This implies significant hardware savings as hardwired shifts are very costly in terms of area. On the other hand, in our case, the complexity in the z coordinate is $\lceil n/6 \rceil$ with respect to a complexity of n in their architecture. Although, in our case, we needed a table for storing the scale factors and approximately $n/4$ shift and addition operations to compute the scale factor, in Dawid and Meyr's architecture, there is an area cost for registers because of the pipelining at the full adder level and of the n initial register rows used for performing the skew of the input data.

The stage delay of the processor proposed by Dawid and Meyr (13 t_g, that is, the adding of the delays of the register load, 3 to 2 CSA, and an absolute value computation slice) is 38 percent less than that of the architecture we propose (21 t_g) (see Table 3). This is due to the high pipelining level. However, the latency of the processor is very high. For example, for $n = 32$, Dawid and Meyr's architecture has a pipeline filling time of 1,274 t_g. In our architecture, this time is only 546 t_g. On the other hand, we could also introduce a higher level of pipelining with fewer rows of registers.

Comparing our radix-4 pipelined architecture with nonredundant fast adders to the radix-2 pipelined architecture, the cycle time of our design is slightly larger than for the radix-2 case. However, the radix-4 approach reduces the latency by approximately 50 percent. For $n = 32$, in the radix-2 case, the pipeline filling time is 1,024 t_g. In the radix-4 case, this time is 629 t_g. The comparison of the hardware complexity is similar to the unfolded nonpipelined case.

4.3 Application Specific Architecture

In specific purpose processors, it is necessary to precompute the coefficients corresponding to the angles they will handle, which will be stored in a look-up table. In Section 3.3, we propose an algorithm that is an adaptation for our radix-4 CORDIC algorithm of the radix-2 CORDIC algorithm used by Hu and Naganathan [12] and which permits obtaining quasi optimal decompositions. The complexity is similar to that of Hu's algorithm, being $O(n^2)$ for 95 percent of the cases and $O(n^3)$ for the rest (only when the condition of step 2 is verified in the algorithm proposed in Section 3.3).

We have carried out a statistical study in the same line as in [12], with 8,000 angles uniformly distributed between 0 and $\pi/2$, for 16 and 32 bit precision, in order to test the goodness of the algorithm that we have proposed in Section 3.3. The average number of iterations (4.9 and 10.24 for 16 and 32 bits, respectively) is larger than the optimal case by just five percent. The average number of iterations for our algorithm is lower than [12] by 0.5 iterations because, in [12], on average, one more iteration for the angular interval $[\pi/4, \pi/2)$ is needed.

The pipelined implementation of the radix-4 algorithm we propose requires $n/2$ stages (microrotations) for $n + 1$ bit precision, where the shifts in each stage are hardwired in. An implementation with the same number of stages, based on the method proposed by Hu and Naganathan, requires variable shifters in each stage. This results in Hu et al.-based designs needing more complex hardware per stage and longer cycle time. Another alternative for implementing a pipelined version of [12] would be to use n stages with fixed shifts. In this case, the hardware of the pipelined radix-4 CORDIC would be approximately half of that required by Hu and Naganathan.

Finally, the procedure for increasing the precision of the specific purpose radix-4 CORDIC by one bit, described in Section 3.3, can be immediately extended to the radix-2 CORDIC, reducing by one the number of radix-2 iterations needed for achieving a given precision in the range $[0, \pi/2)$.

5 Conclusions

In this work, we have developed a new radix-4 CORDIC algorithm in rotation mode. We have proposed redundant architectures that implement this algorithm. The corresponding nonredundant architectures can be obtained with minor modifications. Since the scale factor is not a constant, we have developed a technique to evaluate it for each rotation angle. The proposed algorithm halves the number of microrotations with respect to the standard radix-2 CORDIC algorithm. Furthermore, special methods to assure convergence, like in the case of the radix-2 redundant CORDIC algorithm with constant scale factor (correcting microrotation method [19], branching CORDIC method [7], etc.), are not needed in our radix-4 algorithm. New VLSI CORDIC architectures based on the radix-4 algorithm have been proposed: word serial and pipelined architectures for specific application (known angles) and for general purpose. The word serial architecture we propose incorporates redundant carry-save arithmetic and skipping of the iterations. We have compared the proposed word serial and unfolded architectures (redundant and nonredundant with fast adders) to other functionally equivalent redundant and nonredundant architectures. These first order comparisons indicate that our architectures may have a better trade-off between delay and hardware complexity than previously proposed CORDIC architectures. Finally, the architectures we have designed can be the kernel of VLSI application specific processors for high speed applications. The choice of the architecture (word serial or pipelined, for general or specific application purpose, with redundant or nonredundant fast adders) depends of the application.

ACKNOWLEDGMENT

This work was supported by the Ministry of Education and Science (CICYT) of Spain under project TIC-92-0942.

REFERENCES

[1] E. Antelo, J.D. Bruguera, and E.L. Zapata, "Unified Mixed Radix-2-4 Redundant CORDIC Processor," *IEEE Trans. Computers*, vol. 45, no. 9, pp. 1,086-1,073, Sept. 1996.

[2] P.W. Baker, "Suggestion for a Fast Binary Sine/Cosine Generator," *IEEE Trans. Computers*, pp. 1,134-1,136, 1976.

[3] J.D. Bruguera, E. Antelo, and E.L. Zapata, "Design of a Pipelined Radix-4 CORDIC Processor," *J. Parallel Computing*, vol. 19, no. 7, pp. 729-744, 1993.

[4] J.D. Bruguera, N. Guil, T. Lang, J. Villalba, and E.L. Zapata, "CORDIC Based Parallel/Pipelined Architecture for the Hough Transform," *J. VLSI Signal Processing*, vol. 12, no. 3, pp. 207-221, 1996.

[5] J.R. Cavallaro and F.T. Luk, "CORDIC Arithmetic for an SVD Processor," *J. Parallel and Distributed Computing*, no. 5, pp. 271-290, 1988.

[6] H. Dawid and H. Meyr, "The Differential CORDIC Algorithm: Constant Scale Factor Redundant Implementation without Correcting Iterations," *IEEE Trans. Computers*, vol. 45, no. 3, pp. 307-318, Mar. 1996.

[7] J. Duprat and J.-M. Muller, "The CORDIC Algorithm: New Results for Fast VLSI Implementation," *IEEE Trans. Computers*, vol. 42, no 2, pp. 168-178, Feb. 1993.

[8] M.D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, 1994.

[9] M.D. Ercegovac and T. Lang, "Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD," *IEEE Trans. Computers*, vol. 39, no. 6, pp. 725-740, June 1990.

[10] G.J. Hekstra and E.F. Deprettere, "Floating Point CORDIC," *Proc. 11th Symp. Computer Arithmetic*, pp. 130-137, 1993.

[11] Y.H. Hu, "CORDIC-Based VLSI Architecture for Digital Signal Processing," *IEEE Signal Processing Magazine*, no. 7, pp. 16-35, July 1992.

[12] Y.H. Hu and S. Naganathan, "An Angle Recoding Method for CORDIC Algorithm Implementation," *IEEE Trans. Computers*, vol. 42, no. 1, pp. 99-102, Jan. 1993.

[13] Y.H. Hu, "A Forward Angle Recoding CORDIC Algorithm and Pipelined Processor Array Structure for Digital Signal Processing," *Digital Signal Processing*, vol. 3 no. 1, pp. 2-15, Jan. 1993.

[14] K. Kota and J.R. Cavallaro, "Numerical Accuracy and Hardware Tradeoffs for CORDIC Arithmetic for Special Purpose Processors," *IEEE Trans. Computers*, vol. 42, no. 7, pp. 769-779, July 1993.

[15] A.A. de Lange and E.F. Deprettere, "Design and Implementation of a Floating-Point Quasi Systolic General Purpose CORDIC Rotator for High-Rate Parallel Data and Signal Processing," *Proc. 10th Symp. Computer Arithmetic*, pp. 272-281, 1991.

[16] J.-A. Lee and T. Lang, "Constant-Factor Redundant CORDIC for Angle Calculation and Rotation," *IEEE Trans. Computers*, vol. 41, no. 8, pp. 1,016-1,025, Aug. 1992.

[17] H.X. Lin and H.J. Sips, "On-Line CORDIC Algorithms," *IEEE Trans. Computers*, vol. 39, no. 8, pp. 1,038-1,052, Aug. 1990.

[18] P. Montuschi and L. Ciminiera, "Reducing Iteration Time when Result Digit Is Zero for Radix-2 SRT-Division and Square Root with Redundant Remainders," *IEEE Trans. Computers*, vol. 42, no 2, pp. 239-246, Feb. 1993.

[19] N. Takagi, T. Asada, and S. Yajima, "Redundant CORDIC Methods with a Constant Scale Factor for Sine and Cosine Computation," *IEEE Trans. Computers*, vol. 40, no. 9, pp. 989-995, Sept. 1991.

[20] D. Timmermann, H. Hahn, and B.J. Hosticka, "Low Latency Time CORDIC Algorithms," *IEEE Trans. Computers*, vol. 41, no. 8, pp. 1,010-1,015, Aug. 1992.

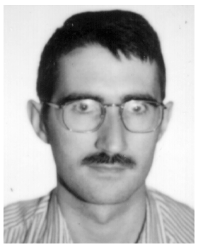
[21] J.E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. Electronic Computers*, vol. 8, pp. 330-334, Sept. 1959.

[22] J.S. Walther, "A Unified Algorithm for Elementary Functions," *Proc. Spring. Joint Computer Conf.*, pp. 379-385, 1971.

Elisardo Antelo received the BS degree in physics in 1991 and the PhD in 1995, both from the Universidade de Santiago de Compostela, Spain.

In 1992, he joined the Departamento de Electronica e Computacion of the Universidade de Santiago de Compostela. From 1992 to March 1996, he was an assistant professor. He has been an associate professor since March 1996.

Dr. Antelo's primary research interests are in computer arithmetic, VLSI design, and high performance architectures for real time applications.



Julio Villalba recieved the BS degree in physics from the University of Granada in 1986 and the PhD degree in computer engineering from the University of Malaga in 1995. During 1986-1991, he worked in the R&D Department of Fujitsu-Spain and was an associate professor at the University of Malaga. Since 1992, he has been a full-time associate professor in the Department of Computer Architecture at the University of Malaga. His research interests include computer arithmetic, parallel architectures, and VLSI design.

Javier D. Bruguera received the BS degree in physics from the University of Santiago de Compostela in 1984 and the PhD degree from the University of Santiago de Compostela in 1989. Currently, he is a professor with the Department of Electronic and Computation at the University of Santiago de Compostela. His research interests are in the areas of VLSI digital signal and image processing, computer arithmetic, and parallel algorithms.

Emilio L. Zapata received the BS degree in physics from the University of Granada in 1978 and the PhD degree from the University of Santiago de Compostela in 1983. During 1978-1981, he was an assistant professor at the University of Granada, and, during 1982-1991, he successively was an assistant, associate, and full professor at the University of Santiago de Compostela. Currently, he is a professor with the Department of Computer Architecture at the University of Malaga. His research interests are in the areas of parallel computer architecture, parallel algorithms, numerical algorithms for dense and sparse matrices, and VLSI digital signal processing. In these areas, he has published more than 40 papers in refereed international journals and about 50 refereed international conference proceedings.