

Pre-print version

S. Wang, V. Piuri, "A unified view of
CORDIC processor design", in
Application specific processors, E.E.
Swartzlander Jr. (ed.), Kluwer, pp. 121-
160, 1997. ISBN: 0-792-39792-4

A UNIFIED VIEW OF CORDIC PROCESSOR DESIGN

Shaoyun Wang and Vincenzo Piuri*

*Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas 78712*

*Crystal Semiconductor Corporation
4210 S. Industrial Dr.
Austin, TX 78744*

**Department of Electronics and Information
Politecnico di Milano
piazza L. da Vinci 32
20133 Milano, Italy*

ABSTRACT

The COordinate Rotation DIgital Computer (CORDIC) algorithm is a well-known and widely studied method for plane vector manipulations. It uses a sequence of partial vector rotations to approximate the expected one. Under different operating modes, this algorithm can be used either to do Givens transformation for vector rotation and vectoring or to evaluate more than a dozen of elementary, trigonometric, and hyperbolic functions such as multiplication, division, square root, sine, cosine, inverse tangent, hyperbolic sine, hyperbolic cosine, and inverse hyperbolic tangent. CORDIC processors are therefore powerful computing systems for applications involving a large amount of rotation operations and mathematical functions mentioned above.

CORDIC computation adopts only simple primitive arithmetic operations (additions, subtractions, and shiftings) instead of multiplications though the algorithm achieves linear convergence only. This has a great impact on the hardware characteristics especially when circuit complexity is concerned. As a consequence, the CORDIC algorithm is become a widely used approach for elementary function evaluation whenever the silicon area is a primary constraint in circuit design.

The main drawback is the intrinsic low performance due to the iterative computational approach. It is quite difficult to increase the performance and the computing capacity massively. In particular, it is not easy to exploit the computational parallelism since

each CORDIC iteration has to select the rotation direction by analyzing the results of the previous one.

In this chapter, a unified view of the CORDIC architecture design is presented. Our goal is to provide a wide spectrum of architectures, a coordinated and comprehensive design methodology, and the main figures of merit characterizing architectures' performance and complexity. This methodology contains the basic guidelines for designers to choose an approach with respect to specific requirements and constraints of the application.

1 INTRODUCTION

The COordinate Rotation DIgital Computer (CORDIC) algorithm is a well-known and widely-studied iterative technique (e.g., see [1] [2]) for planary vector rotation/vectoring and for evaluating some basic arithmetic operations and several mathematical functions. Examples are multiplication, division, square root, sine, cosine, inverse tangent, hyperbolic sine, hyperbolic cosine, and inverse hyperbolic tangent. The result of these functions can then be exploited to generate other transcendental functions such as tangent, hyperbolic tangent, logarithms, and exponentials. All the above mentioned functions are widely used in many massive-computing applications (e.g., dynamic system modeling, control, robotics, computer graphics, digital signal processing, image processing, document processing, imaging, simulation, virtual reality, *etc.*). Navigation and guidance processing with the CORDIC algorithm [1] dates back to 1959. Currently, the application areas have been expanded to deal with several DSP problems: for example, filtering [61] [60], equalization [59], FFT [58] [41] [37], Chirp Z-transform [36], Hough transform [57], and QR decomposition [7] [56] [55] [29]. The use in image processing draws much attention. Singular Value Decomposition (SVD) has several applications in image processing; SVD for complex matrices has been developed [54]. J. R. Cavallaro and F. Luk proposed four different architectures in [53] [52]. An efficient implementation of the CORDIC algorithm is a key point for the effective realization of dedicated and embedded systems in these application areas.

Since the first description of the CORDIC algorithm by Volder in 1959 [1], several researchers have examined different aspects of the algorithm. They cover a wide area from the theoretical generalization to the hardware implementation.

Based on the convergence proof of the CORDIC algorithm [3], J.-M. Muller generalized the CORDIC algorithm and developed several other algorithms using

only additions and shifts to compute elementary functions [4]. The CORDIC algorithm has also been extended to higher dimensions to deal with the alignment of a given vector to a certain direction [5]. In [6], the inverse algorithm is extended to evaluate the vector's coordinates in a multidimensional space, leading to a Householder algorithm. Studies continue to exploit the CORDIC algorithm for additional elementary functions: C. Mazenc, X. Merrheim, and J.-M. Muller have modified the algorithm for $\cos^{-1}$, $\sin^{-1}$, $\sqrt{1-t^2}$, $\sqrt{1+t^2}$, $\cosh^{-1}$, and $\sinh^{-1}$ [50].

The quantization effects have also been studied in detail in order to evaluate the precision of the results generated by this technique. The error bound for the inverse tangent is computed in [51]. The error bounds for approximation error of the angle, the rounding error, and the overall quantization error for the other elementary functions are derived in [35].

The CORDIC algorithm can be exploited to realize mathematical coprocessors having better accuracy than many other approaches. Implementations are nowadays available in many real systems, in special purpose chips and in general-purpose CORDIC chips. Some of the first commercial products are the HP 35 calculator [32] and the Intel 8087 mathematical coprocessor. A laser trimming system [46], which is used to correct the position of a micro-circuit on a laser trimming platform, gives an example for the CORDIC algorithm performing high precision computations. FELIN [45] is another CORDIC processor designed as a mathematical coprocessor. The pipelined CORDIC processor developed at the University of Warwick is the first floating-point processor for QR decomposition [44] [43]. Another fixed point CORDIC processor reported is a programmable CORDIC chip [42]; it is monolithic, fully parallel and very suited for digital signal processing since it can emulate other algorithms by programming.

Floating point implementation of the CORDIC algorithm has two main drawbacks. First, accuracy becomes unacceptable when computing angles close to or smaller than the angle resolution. Besides, linear convergence slows down the computation especially for high-precision floating point operands. For these reasons, I. Koren and O. Zinaty recommended rational approximation for elementary functions rather than the CORDIC algorithm in the implementation of a high precision numerical coprocessor [49]. However, researchers are still working on floating point CORDIC [48] and complex CORDIC [47].

Several approaches to design advanced CORDIC processors have been studied in order to enhance the performance or to reduce the circuit complexity (e.g., see [15]- [31]). General digital design techniques allowed either to reduce the

system latency or to increase the throughput. By cascading the stages required to perform all CORDIC iterations, the resulting purely combinatoric structure minimize the latency [21] [27], while pipelining the cascade of stages allows for throughput enhancement [26] [31]. Reference [27] introduced a technique to anticipate the direction of each CORDIC rotation without having completed all the previous iterations, but with some possible limited error in the function evaluation. The rotation directions for an initial group of iterations are determined in parallel directly from the rotation angle, while the rotations are then applied sequentially. This makes it possible to design an efficient architecture based on carry-save adders since it is not necessary to wait for the complete carry propagation for each CORDIC iteration in order to evaluate the next direction in the group of iterations. This is a "semi-parallel architecture" since additions must be done serially, even if a decrease in the latency can be achieved due to carry-save operations. Then, a second group of rotation directions is determined in parallel by analyzing the residue angle generated of the previous iteration; the rotations are again applied sequentially on the vector. In the last about $\frac{2N}{3}$ stages, the bits of the residue rotation angle generated by the previous iteration control the rotation directions of the next iteration; the exact number of these stages has been determined empirically. A similar approach has been presented in [30]: some additional rotations are introduced among the nominal ones in order to correct the possible error induced by prediction. The reasoning in [27] and [30], leads to a very low latency architecture, although the complexity may be too high for many implementations.

Redundant arithmetic has been proved an interesting solution to enhance the throughput. However, in general, it is not able to guarantee a constant scale factor, independent from the rotations actually applied. Though time consuming, several modified versions have been proposed to avoid this drawback [7]- [11]. In [9], additional rotations are introduced to preserve the total number of rotations and, as a consequence, the scale factor even when void rotations are considered. The branching CORDIC method is similar to rotation direction select CORDIC [12], since it uses an approach like the carry select adder. The CORDIC algorithm can be used to convert redundant numbers into their conventional representations [13].

Another technique to increase the throughput is based on on-line implementation of the CORDIC algorithm [14]. In this case, determination of the rotation direction is difficult since it is based on the sign of the previous rotation error, which is not directly available. The only solution is to allow zero rotations. Several methods have been published about the on-the-fly computation of the scale factor and about scaling the results so that the final scale factor multi-

plication is easy and fast. In [8] [62], the on-line CORDIC algorithm is applied to the SVD problem.

The use of Nonconstant Scale Factor CORDIC (NSF-CORDIC) algorithms has been considered in some specific applications, e.g., FFT [41] [37] and Chirp Z-transform [36], even if no general research has been yet published on the general characteristics of this approach. Rotation angles are fixed: by knowing these angles, a greedy algorithm can be adopted to minimize the number of iteration steps [37]. This optimization is called angle recoding due to its resemblance to multiplier recoding. The greedy algorithm could also be applied to the scale factor to eliminate some iterations. However, since the algorithm has a computation complexity of $O(n^2)$, it is not practical for most of the real-time applications.

The wide variety of solutions and architectures proposed in the literature is difficult to be explored by the designer in order to identify the most suited approach for a given application. One of the main problems is in fact the number of quite differentiated structures and the non-homogeneous analysis of the possible solutions.

In this chapter, a comprehensive and coordinated view of CORDIC processors is proposed. The goal is to provide a continuous and homogeneous spectrum of solutions to the designer, with the related figures of merit as guidelines for choosing the architectural structure best fitting the application constraints and requirements. All the known and possible solutions are placed in a reference space. Two basic transformations rules on mapping the CORDIC algorithm onto a hardware dedicated architecture are identified to transform one solution into the adjacent ones: namely, *combining* and *unrolling*. The first one merges more CORDIC steps into only one operational clock cycle. Unrolling re-maps the operations performed by an iterative CORDIC architecture, implemented by a sequential machine, onto a combinatoric circuit. In the reference space, each of these transformations is associated to one dimension; the values characterizing their application are used as measures in the corresponding dimension. This continuous set of solutions includes not only the known architectures, but also a number of new intermediate structures that could better match the application constraints and requirements on circuit complexity, latency, and throughput. While the first two dimensions of the reference space are concerned with the processor architecture, a third dimension is added to take into account the variants for the internal structure of the individual components (e.g., adders, shifters).

By abstracting from a specific architecture and from a single optimization goal, we can achieve a more general knowledge of the arithmetic behavior of the CORDIC systems and, as a consequence, we can deduce different families of new architectures supporting the CORDIC computation at different level of cost and performance. In particular, we provide the designer with a wide spectrum of alternative solutions so that the best trade-off between performance and cost can be selected for each specific application.

In Section 2, the basic characteristics of the CORDIC algorithms are recalled. Section 3 introduces the first "processor architecture" dimension of the reference space mentioned above: *combining* is the transformation ruling the CORDIC mapping onto sequential architectures. The resulting structures constitute the family of *combined* architectures. In Section 4, the *unrolling* rule is presented and applied to the combined structures to generate their counterpart in the second "processor architecture" dimension: the *pipelined* architectures are so obtained. In Each Section, the alternative internal structures for the components are discussed to generate the processor solutions having the same general architecture along the third "component architecture" dimension. The evaluation of circuit complexity, latency, and throughput is developed to provide the guidelines for the optimal architectural choice. The analysis is carried on in a way independent from the implementation technology by using traditional gate count techniques. Convergence and accuracy are also addressed. In Section 5, the comparisons of these figures of merit for all the architectural strategies is given, while the general designed guidelines are derived in Section 6.

2 THE CORDIC ALGORITHM

2.1 The Basic CORDIC Algorithm

The COordinate Rotation DIgital Computer (CORDIC) technique was originally described by J. E. Volder for real-time airborne computations in 1959 [1]. Since then, this technique evolves from a simple plane coordinate rotator to an algorithm that performs Givens transformation and evaluates more than a dozen of elementary, trigonometric, and hyperbolic functions, directly or indirectly.

The basic idea of the algorithm remains the same for all these extensions and applications. Let's consider the computation of the Givens rotation of a vector

$\mathbf{R}_0(x_0, y_0)$ by an angle θ . Let's divide θ into two parts: θ_1 and θ_2 . The required transformation can be performed by rotating the vector first by the angle θ_1 and, then, by rotating the resulting vector by the angle θ_2 :

$$\begin{aligned}
 \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} &= \begin{bmatrix} \cos \theta_1 & -\sin \theta_1 \\ \sin \theta_1 & \cos \theta_1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \\
 \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} &= \begin{bmatrix} \sin \theta_2 & -\sin \theta_2 \\ \sin \theta_2 & \cos \theta_2 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \\
 &= \begin{bmatrix} \cos \theta_2 & -\sin \theta_2 \\ \sin \theta_2 & \cos \theta_2 \end{bmatrix} \begin{bmatrix} \cos \theta_1 & -\sin \theta_1 \\ \sin \theta_1 & \cos \theta_1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \\
 &= \cos \theta_2 \begin{bmatrix} 1 & -\tan \theta_2 \\ \tan \theta_2 & 1 \end{bmatrix} \cos \theta_1 \begin{bmatrix} 1 & -\tan \theta_1 \\ \tan \theta_1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \\
 &= \cos \theta_1 \cos \theta_2 \begin{bmatrix} 1 & -\tan \theta_2 \\ \tan \theta_2 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\tan \theta_1 \\ \tan \theta_1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}
 \end{aligned}$$

Rotation decomposition is the basic idea underlying the CORDIC algorithm. If the tangents of θ_1 and θ_2 are powers of two (i.e., it is $\tan \theta_1 = 2^{-j}$ and $\tan \theta_2 = 2^{-i}$, being i and j integers), the matrix multiplications are simplified to shiftings.

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \cos \theta_1 \cos \theta_2 \begin{bmatrix} 1 & -2^{-i} \\ 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} 1 & -2^{-j} \\ 2^{-j} & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (5.1)$$

The Givens transformation by the angle θ that can be decomposed in such a way can be reduced to rotations which can be easily performed. The factor $\cos \theta_1 \cos \theta_2$ is treated as a scale factor; it can be considered as a multiplicative constant since it can be a priori known when the angles θ_1 and θ_2 are given values. Conventional rotation is performed in a single computational step by means of 4 multiplications and 2 additions, in the example above. Conversely, the total computational complexity of the decomposed rotation is: 4 additions and 4 shifts (an additional 2 multiplications are required if the scale multiplications are performed); as a consequence, the CORDIC computation has a lower complexity than the conventional case.

Matrix $\begin{bmatrix} 1 & -2^{-i} \\ 2^{-i} & 1 \end{bmatrix}$ is a forward CORDIC rotation, which rotates a vector counterclockwise. Figure 1 illustrates a general CORDIC iteration step that introduces the matrix $\begin{bmatrix} 1 & \mp 2^{-i} \\ \pm 2^{-i} & 1 \end{bmatrix}$ to describe both forward and backward rotations. Vector $\mathbf{R}_i(x_i, y_i)$ is the result of i -th iteration. For $(i+1)$ -th iteration, the possible rotation direction is either $\sigma_{i+1} = -1$ or $\sigma_{i+1} = 1$; as a

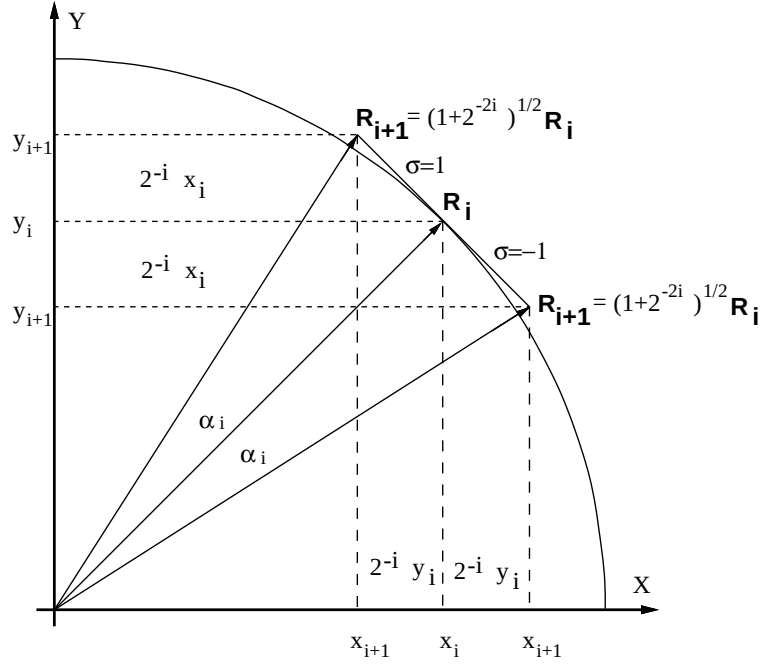


Figure 1 A Rotation of the CORDIC Algorithm

consequence, the new vector $\mathbf{R}_{i+1}(x_{i+1}, y_{i+1})$ has two possible positions. The vector length is increased by the same value after the iteration regardless of the rotation direction.

For general cases, J. E. Volder showed that an arbitrary angle $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ can be decomposed into a set of angles $\{\theta_i, i = 0, 1, 2, \dots, N\}$.

$$\theta = \sum_{i=0}^N \sigma_i \theta_i$$

where $\sigma_i = \{-1, 1\}$. By using the trigonometric identities, it is:

$$\begin{bmatrix} x_N \\ y_N \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (5.2)$$

$$= \begin{bmatrix} \cos(\sum_{i=0}^N \sigma_i \theta_i) & -\sin(\sum_{i=0}^N \sigma_i \theta_i) \\ \sin(\sum_{i=0}^N \sigma_i \theta_i) & \cos(\sum_{i=0}^N \sigma_i \theta_i) \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (5.3)$$

$$= \prod_{i=0}^N \cos \sigma_i \theta_i \begin{bmatrix} 1 & -\tan \sigma_i \theta_i \\ \tan \sigma_i \theta_i & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (5.4)$$

$$= \prod_{i=0}^N \cos \theta_i \begin{bmatrix} 1 & -\sigma_i \tan \theta_i \\ \sigma_i \tan \theta_i & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (5.5)$$

If each matrix multiplication has to be implemented by using simple arithmetic operations (additions and subtractions) and shifting only, the angle set $\{\theta_i, i = 0, 1, 2, \dots, N\}$ has to satisfy the following constraint [1]:

$$\{\theta_i = \arctan 2^{-i}, \forall i = 0, 1, 2, \dots, N\}$$

$$\begin{bmatrix} x_N \\ y_N \end{bmatrix} = \prod_{i=1}^N \cos \theta_i \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (5.6)$$

The above angles' set is called Arc Tangent Radix (ATR).

2.2 The Unified CORDIC Algorithm

In the general case, the CORDIC algorithm is a bit-recursive implementation of the forward and backward Givens transformation [1] [2] (also called *rotation* and *vectoring*, respectively).

Consider the vector $\mathbf{v}$ in the plane xy , extending from the origin to x^* and y^* . Let z^* be the angle of the vector with respect to the x axis. The basic CORDIC iteration equations at the i -th step are [2]:

$$\begin{cases} x_{i+1} = x_i - m \sigma_i 2^{-S(m,i)} y_i \\ y_{i+1} = y_i + \sigma_i 2^{-S(m,i)} x_i \\ z_{i+1} = z_i - \sigma_i \alpha_{m,i} \end{cases} \quad (5.7)$$

where the iterations are repeated $\forall i, i = 0, 1, 2, \dots, N-1$. The *coordinate parameter* m identifies the coordinate system type

$$m = \begin{cases} 1 & \text{circular coordinate system} \\ 0 & \text{linear coordinate system} \\ -1 & \text{hyperbolic coordinate system} \end{cases} \quad (5.8)$$

The *rotation direction* σ_i is defined as:

$$\sigma_i = \begin{cases} \text{sign}(z_i) & \text{for rotation mode} \\ \text{sign}(x_i \cdot y_i) & \text{for vectoring mode} \end{cases} \quad (5.9)$$

The *shift sequence* $S(m, i)$ (i.e., the exponent of the rotation coefficient)

$$S(m, i) = \begin{cases} 0, 1, 2, 3, 4, 5, \dots & m = 1 \\ 1, 2, 3, 4, 5, 6, \dots & m = 0 \\ 1, 2, 3, 4, 4, 5, \dots & m = -1 \end{cases} \quad \text{repeated at } \frac{3^{i+2}-1}{2} \quad (5.10)$$

The *rotation angle* $\alpha_{m,i}$ is given by

$$\alpha_{m,i} = \frac{1}{\sqrt{m}} \arctan(\sqrt{m}2^{-S(m,i)}) \quad (5.11)$$

The *scale factor* $k_{m,i}$ corrects the distortion introduced by the linearized rotation in the x and y coordinates. With the above assumptions, $k_{m,i} = \sqrt{1 + m\sigma_i^2 2^{-2S(m,i)}}$ for the i -th iteration. After N iterations, the total scale factor is

$$K_m = \prod_{i=0}^{N-1} k_{m,i} = \prod_{i=0}^{N-1} \sqrt{1 + m\sigma_i^2 2^{-2S(m,i)}} \quad (5.12)$$

For rotation of the vector $\mathbf{v}$ by the angle θ , we set $x_0 = x^*$, $y_0 = y^*$, and $z_0 = \theta$; at the end of the iterations, the vertex of the rotated vector $\mathbf{v}'$ has coordinates (x_N, y_N) . Conversely, for vectoring, we set $x_0 = x^*$, $y_0 = y^*$, and $z_0 = 0$; at the end of the iterations, the required angle z^* is equal to z_N , while the modulus $|\mathbf{v}|$ is given by x_N . In this chapter, we consider only rotation since we are concerned with function generation.

The functions directly computed by these CORDIC iteration equations, according to the selected value of the coordinate parameter, are summarized in Table 2 and Table 3. With the appropriate initial values of x^* and y^* , we can compute the value of the trigonometric and transcendental elementary functions.

In traditional Constant Scale Factor CORDIC, the rotation directions are restricted to $\sigma_i = \{-1, 1\}$. In this case, the scale factor becomes constant with respect to the rotation coefficients, and depends only on the word length N and coordinate parameter m . A constant scale factor simplifies the correction since the factor is applied only once (either to the final result or to the initial operands.) Since in function generators, the initial values of x^* and y^* are constant numbers that must be set before starting the CORDIC iterations, we choose to incorporate in these constants the global scale factor so that no multiplication will be required at the end of the iterations themselves.

To enhance the convergence speed of the algorithm, some authors (e.g., [8] [29]) also use void rotations, i.e., they consider 0 as an acceptable value for σ_i .

Coordinate System	rotation $z_n \rightarrow 0$	vectoring $y_n \rightarrow 0$
Trigonometric $m = 1$	$x_n = K_1(x \cos z - y \sin z)$ $y_n = K_1(y \cos z + x \sin z)$	$x_n = K_1 \sqrt{x^2 + y^2}$ $z_n = z - \tan^{-1}(\frac{y}{x})$
Linear $m = 0$	$x_n = x$ $y_n = y + x \times z$	$x_n = x$ $z_n = z + \frac{y}{x}$
Hyperbolic $m = -1$	$x_n = K_{-1}(x \cosh z + y \sinh z)$ $y_n = K_{-1}(y \cosh z + x \sinh z)$	$x_n = K_{-1} \sqrt{x^2 - y^2}$ $z_n = z + \tanh^{-1}(\frac{y}{x})$

Table 1 Outputs of the CORDIC Algorithm

Coordinate System	rotation ($z_N \rightarrow 0$)			
	x_0^*	y_0^*	z_0^*	Result
Trigonometric $m = 1$	$\frac{1}{K_1}$	0	θ	$x_N = \cos \theta$ $y_N = \sin \theta$
Linear $m = 0$	a	0	b	$y_N = a \times b$
Hyperbolic $m = -1$	$\frac{1}{K_{-1}}$	0	θ	$x_N = \cosh \theta$ $y_N = \sinh \theta$
Hyperbolic $m = -1$	$\frac{1}{K_{-1}}$	$\frac{1}{K_{-1}}$	θ	$y_N = e^\theta$

Table 2 Functions Generated by the CORDIC Algorithm (Rotation Mode)

However, this produces a variable scale factor that depends on the specific sequence of rotations actually applied. If void rotation steps are obtained by using micro-rotations [9], a constant scale factor is preserved, but the number of basic micro-iterations is twice the number of traditional iterations (i.e., the number of additions and shifts is doubled).

3 COMBINED ARCHITECTURES

The first approach to design CORDIC processors was based on direct mapping of the iterative operations of the CORDIC algorithm onto a sequential digital

Coordinate System	vectoring ($y_N \rightarrow 0$)			
	x_0^*	y_0^*	z_0^*	Result
Trigonometric $m = 1$	1	θ	0	$z_N = \arctan \theta$
Linear $m = 0$	a	b	0	$z_N = \frac{b}{a}$
Hyperbolic $m = -1$	1	θ	0	$z_N = \operatorname{arctanh} \theta$
Hyperbolic $m = -1$	$a + 1$	$a - 1$	0	$z_N = \frac{1}{2} \ln a$
Hyperbolic $m = -1$	$\frac{a-0.25}{K_{-1}^2}$	$\frac{a-0.25}{K_{-1}^2}$	0	$z_N = \sqrt{a}$

Table 3 Functions Generated by the CORDIC Algorithm (Vectoring Mode)

machine. The architecture emulated exactly the sequencing of the algorithm steps. This allowed for realizing very compact structures, but the latency required to complete one run was high since operations are strictly serialized.

Combining is a transformation rule that can be applied to the nominal CORDIC algorithm to reorganize its operations onto a dedicated hardware structure. This rule merges more steps into the same computational cycle in order to save some latency by removing some storing operation of intermediate results. As a consequence, the throughput is increased, despite of a circuit complexity increase.

In this Section, we show how the CORDIC steps can be progressively merged. We start from the architecture directly mapping the traditional algorithm: we arrive to design a structure in which all CORDIC steps are collapsed into the same clock cycle.

3.1 The Traditional CORDIC Architecture

Direct mapping of these operations on a sequential machine produce the structure shown in Figure 2. The operation of the parallel adder/subtractors is controlled by the rotation direction. The adder/subtractor is implemented by using a traditional two-input adder for two's-complement integers: the first in-

put is the first addend, while the second input may be either the nominal value of the second operand or its one's complement, according with the current value of the rotation direction (a multiplexer is used to select between these two values). The carry input of the whole adder is 0 for addition and 1 for subtraction, again according with the current value of the rotation direction.

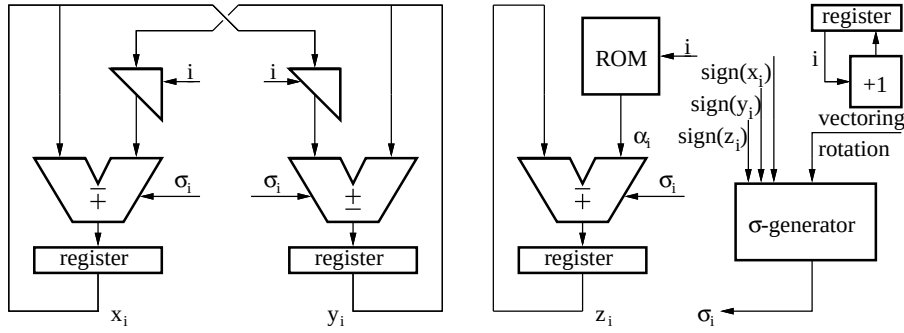


Figure 2 The Traditional (First-Order Combined) CORDIC Architecture

For the generation of the coordinates x and y , N -bit N -position shifters are required to prepare the second operands of the adder/subtractors, controlled by the number of the current iteration directly. For the generation of the coordinate z , the angle corresponding to the current iteration must be provided by using a look-up table. A dedicated control circuit (σ -generator) is used to compute the rotation direction for the current iteration, according with the current values of the accumulators X , Y , and Z , and with the selected CORDIC operation.

A great simplification of this architecture (as well as for all the other here presented) can be achieved when only rotation must be implemented and all directions are a priori known. This allows for avoiding the σ -generator. However, since we are concerned only with the complete CORDIC processor, we do not further consider here this specific case that can be anyway derived from the general analysis. This architecture occupies the origin of the bidimensional reference space defined by the two dimensions concerned with the processor architecture.

The analysis of the characteristic figures of circuit complexity, latency, and throughput should take into account the specific structural choices for the basic building blocks composing the processor architecture, namely the adders, the shifters, and the σ -generator. The structure of this last circuit is quite simple

and is dependent from the operations performed in the processor. Since there are basically no major alternative design strategies affecting its complexity and its performances, we do not further consider this component as one of the possible causes of variants of the whole processor.

Shifters may be implemented by using barrel shifters or switches. However, since the interconnection flexibility of both of these approaches is identical, we consider only the case of the barrel shifters. In fact, for the same topology, they have a smaller circuit complexity and a lower latency than a network of switches.

As in [22], adders may be implemented by using: ripple-carry adders, carry-save adders, carry-look-ahead (CLA) adders, conditional-sum adders, cascaded carry-look-ahead (CCLA) adders (i.e., array of CLA adders between which the carry is propagated in a ripple way), or other structures. As examples, in this chapter, the processor internal structure is built by using only ripple-carry adders or CCLA adders.

As a first-approximation analysis, the circuit complexity is evaluated by using the traditional gate count in order to give an idea of the complexity independently from the specific realization technology, in order to allow for an architectural-level selection of the most suited approach for the specific constraints. To have a good estimation of the transistor count and, as a consequence, to have a rough relative evaluation of the silicon area occupied by the circuits, we used only two-input gates in designing and evaluating the prototype structures.

For each of the adder structures mentioned above, the circuit complexity $C_1^{c,t}$ of the whole CORDIC processor is given by:

$$\begin{aligned}
 C_1^{c,t} &= 2C_{\text{shift}}(N, N) + 3C_{\text{addsub}}(N) + 3C_{\text{acc}}(N) + C_{\text{rom}}(N, N) + \\
 &\quad + C_{\text{reg}}(\lceil \log_2 N \rceil) + C_{\text{inc}}(\lceil \log_2 N \rceil) + C_{\sigma} \\
 C_1^{c,r} &= 3N^2 + 75N + 10 + (0.5N + 17)\lceil \log_2 N \rceil + 0.5\lceil \log_2 N \rceil^2 \\
 C_1^{c,l} &= 3N^2 + 36N + 10 + 252\lfloor \frac{N-1}{4} \rfloor + 3\beta(\text{mod}_4 N) + 0.5\lceil \log_2 N \rceil^2 + \\
 &\quad + (0.5N + 11)\lceil \log_2 N \rceil + 66\lfloor \frac{\lceil \log_2 N \rceil - 1}{4} \rfloor + \beta'(\text{mod}_4 \lceil \log_2 N \rceil)
 \end{aligned}$$

where t is the type of the adder used in the adder/subtractor ($t = r$ for ripple-carry, $t = l$ for CCLA); $C_{\text{shift}}(a, b)$ is the circuit complexity of the a-bit b-position shifter; $C_{\text{addsub}}(a)$ is the one of the a-bit adder/subtractor; $C_{\text{acc}}(a)$ is the one of the a-bit accumulator; $C_{\text{rom}}(a, b)$ is the one of the a-bit b-word

ROM holding the CORDIC angles; $C_{\text{reg}}(a)$ is the one of the a-bit register holding the current iteration; $C_{\text{inc}}(a)$ is the one of the a-bit incrementer; C_{σ} is the one of the σ -generator; $\beta(d)$ is equal to 64, 12, 26, and 43, for d equal to 0, 1, 2, and 3, respectively; $\text{mod}_4(d)$ is the residue modulus 4 of d; $\beta'(d)$ is equal to 52, 9, 20, and 34, for d equal to 0, 1, 2, and 3, respectively. The result is not normalized in any of these architectures since it is identical in all the alternative approaches discussed in the chapter. We include in the circuit complexity also the output latches to have an homogeneous comparison with the other architectures.

By using the same evaluation technique based on the (two-input) gate count, the clock cycle time $\tau_1^{c,t}$ for driving these sequential machines is given by:

$$\begin{aligned}\tau_1^{c,t} &= \max\{\tau_{\sigma}, \tau_{\text{shift}}(N, N), \tau_{\text{rom}}(N, N)\} + \tau_{\text{addsub}}(N) + \tau_{\text{acc}}(N) \\ \tau_1^{c,r} &= 6N + 13 \\ \tau_1^{c,l} &= 13\lfloor \frac{N-1}{4} \rfloor + \gamma(\text{mod}_4 N) + 11\end{aligned}$$

where τ_{σ} is the latency of the σ -generator; $\tau_{\text{shift}}(a, b)$ is the one of the shifter; $\tau_{\text{rom}}(a, b)$ is the one of the ROM holding the CORDIC angles; $\tau_{\text{inc}}(a)$ is the one of the incrementer; $\tau_{\text{addsub}}(a)$ is the one of the adder/subtractor; $\tau_{\text{acc}}(a)$ is the one of the accumulator; $\gamma(d)$ is equal to 12, 8, 10, and 12, for d equal to 0, 1, 2, and 3, respectively. These expressions give the number of elementary two-input gate delays (being the gate delay defined as the time required to generate a steady output after presentation of steady inputs at the digital gate) which are required to complete a clock cycle.

Therefore, the resulting latency $L_1^{c,t}$ of the architectures are given by:

$$\begin{aligned}L_1^{c,t} &= N\tau_1^{c,t} \\ L_1^{c,r} &= 6N^2 + 13N \\ L_1^{c,l} &= 13N\lfloor \frac{N-1}{4} \rfloor + N[\gamma(\text{mod}_4 N) + 11]\end{aligned}$$

being the latency expressed as the number of two-input gate delays that are required to generate the final CORDIC result after presentation of the primary inputs.

The throughput $T_1^{c,t}$ is the number of final CORDIC results produced in one time unit. It is therefore equal to the inverse of the elapsed time between generation of two subsequent final CORDIC at the processor output. By taking into account the operations performed by the processor architecture discussed

in this Section, one new final result is generated only after the whole latency time is passed. As a consequence, it is:

$$\begin{aligned} T_1^{c,t} &= \frac{1}{L_1^{c,t}} \\ T_1^{c,r} &= \frac{1}{6N^2 + 13N} \\ T_1^{c,l} &= \frac{1}{13N \lfloor \frac{N-1}{4} \rfloor + N[\gamma(\text{mod}_4 N) + 1]} \end{aligned}$$

being the throughput evaluated as the number of final CORDIC results that are generated in one time unit (which has been assumed equal to the two-input gate delay).

3.2 The Second-Order Combined Architecture

Merging of two iterations of the algorithm described in Section 3.1 into the same clock cycle allows for removing every other storing operation. We define the order of a combined architecture as the number of CORDIC iterations that are performed in a single clock cycle. The traditional structure of Section 3.1 is therefore a 1st-order combined architecture, while the one discussed in this Section is 2nd-order.

In combined architectures, even if the clock cycle time increases, the time per CORDIC iteration is reduced. As a consequence, the latency decreases and the throughput is enhanced, by increasing the circuit complexity.

To obtain merging, different approaches can be considered. We can fuse completely each pair of subsequent CORDIC iterations (sequential fusion). Consider, in fact, the iteration of Eq. 5.7 and the subsequent one given by:

$$\begin{cases} x_{i+2} = x_{i+1} - m\sigma_{i+1}2^{-S(m,i+1)}y_{i+1} \\ y_{i+2} = y_{i+1} + \sigma_{i+1}2^{-S(m,i+1)}x_{i+1} \\ z_{i+2} = z_{i+1} - \sigma_{i+1}\alpha_{m,i+1} \end{cases} \quad (5.13)$$

By substitution, it is:

$$\begin{cases} x_{i+2} &= [1 - m\sigma_i\sigma_{i+1}2^{-S(m,i)-S(m,i+1)}]x_i - \\ &\quad -m[\sigma_i2^{-S(m,i)} + \sigma_{i+1}2^{-S(m,i+1)}]y_i \\ y_{i+2} &= [1 - m\sigma_i\sigma_{i+1}2^{-S(m,i)-S(m,i+1)}]y_i + \\ &\quad +[\sigma_i2^{-S(m,i)} + \sigma_{i+1}2^{-S(m,i+1)}]x_i \\ z_{i+2} &= z_i - [\sigma_i\alpha_{m,i} + \sigma_{i+1}\alpha_{m,i+1}] \end{cases} \quad (5.14)$$

By removing the terms $m\sigma_i\sigma_{i+1}2^{-S(m,i)-S(m,i+1)}$, i.e., by assuming that they can be neglected with respect to the other term, we obtain an expression which is very similar to the traditional CORDIC iteration (see Eq. 5.7), except for the fact that it has two shifted contributions for each coordinate.

If we analyze the error introduced by each sequentially-fused step into the computation of the vector $\mathbf{v}$ (see [20]), it is $\|\Delta A_i\| = |\sigma_i||\sigma_{i+1}|2^{-S(m,i)-S(m,i+1)} = 2^{-2i-1}$. This is obviously not acceptable since it is too high during the first iterations and cannot be recovered in any way, e.g., by adding some extra bits and extra iterations. Therefore, the sequential fusion cannot be considered for combining.

A second approach is the symmetric fusion [28]. The i -th iteration of Eq. 5.7 is fused with the iteration $N - 1 - i$. The resulting merged iteration is:

$$\begin{cases} x_{i+1} &= [1 - m\sigma_i\sigma_{N-i-1}2^{-S(m,i)-S(m,N-1-i)}]x_i - \\ &\quad -m[\sigma_i2^{-S(m,i)} + \sigma_{N-i-1}2^{-S(m,N-1-i)}]y_i \\ y_{i+1} &= [1 - m\sigma_i\sigma_{N-i-1}2^{-S(m,i)-S(m,N-1-i)}]y_i + \\ &\quad +[\sigma_i2^{-S(m,i)} + \sigma_{N-i-1}2^{-S(m,N-1-i)}]x_i \\ z_{i+1} &= z_i - [\sigma_i\alpha_{m,i} + \sigma_{N-i-1}\alpha_{m,N-1-i}] \end{cases} \quad (5.15)$$

By removing the term $m\sigma_i\sigma_{N-i-1}2^{-S(m,i)-S(m,N-1-i)}$, we still obtain an expression similar to the traditional one but with two shifted contribution in each coordinate. However, in this case, the error analysis produces an iteration error given by $\|\Delta A_i\| = |\sigma_i||\sigma_{i+1}|2^{-S(m,i)-S(m,N-i-1)} = 2^{-N+1}$. This error is constant for each iteration and is acceptable since it affects only the least-significant bits; besides, it is possible to remove it by adding few extra bits to the initial operands and by performing few extra iterations. However, these extra iterations may greatly reduce the time saving due to fusion.

The third approach is cascaded fusion. Two subsequent operation are squeezed into the same clock cycle, as in sequential fusion, but no modification to the nominal operations is introduced. In particular, no simplification of the coefficients is performed. Operands are used exactly as they are in the traditional

approach of Section 3.1: only storing between the CORDIC iterations of the same cycle is avoided. No additional error is therefore introduced with respect to the traditional architecture.

The resulting architecture is shown in Figure 3. The area is increased since the adders and the σ -generator are doubled. Each shifter moves an N -bit word in $\frac{N}{2}$ different positions. Since these positions are skewed by two bits each, the total circuit complexity of the shifting matrices is identical to the traditional case; only the shifting decoders driving the operation of the barrel shifter may be marginally simplified. As in Section 3.1, the circuit complexity $C_2^{c,t}$ is given by:

$$\begin{aligned}
 C_2^{c,t} &= 4C_{\text{shift}}(N, \lceil \frac{N}{2} \rceil) + 6C_{\text{addsub}}(N) + 3C_{\text{acc}}(N) + \\
 &\quad + 2C_{\text{rom}}(N, \lceil \frac{N}{2} \rceil) + C_{\text{reg}}(\lceil \log_2 \lceil \frac{N}{2} \rceil \rceil) + C_{\text{inc}}(\lceil \log_2 \lceil \frac{N}{2} \rceil \rceil) + 2C_{\sigma} \\
 C_2^{c,r} &= 3N^2 + 114N + 20 + \lceil \log_2 \lceil \frac{N}{2} \rceil \rceil^2 + (\lceil \frac{N}{2} \rceil + 16)\lceil \log_2 \lceil \frac{N}{2} \rceil \rceil \\
 C_2^{c,l} &= 3N^2 + 36N + 20 + 504\lfloor \frac{N-1}{4} \rfloor + 6\beta(\text{mod}_4 N) + \lceil \log_2 \lceil \frac{N}{2} \rceil \rceil^2 + \\
 &\quad + (\lceil \frac{N}{2} \rceil + 16)\lceil \log_2 \lceil \frac{N}{2} \rceil \rceil
 \end{aligned}$$

The clock cycle time $\tau_2^{c,t}$ is increased since two CORDIC iterations are accommodated in the same cycle:

$$\begin{aligned}
 \tau_2^{c,t} &= 2\max\{\tau_{\sigma}, \tau_{\text{shift}}(N, \lceil \frac{N}{2} \rceil), \tau_{\text{rom}}(N, \lceil \frac{N}{2} \rceil)\} + 2\tau_{\text{addsub}}(N) + \tau_{\text{acc}}(N) \\
 \tau_2^{c,r} &= 12N + 20 \\
 \tau_2^{c,l} &= 26\lfloor \frac{N-1}{4} \rfloor + 2\gamma(\text{mod}_4 N) + 16
 \end{aligned}$$

The latency $L_2^{c,t}$ and the throughput $T_2^{c,t}$ become, respectively:

$$\begin{aligned}
 L_2^{c,t} &= \tau_2^{c,t} \lceil \frac{N}{2} \rceil \\
 L_2^{c,r} &= 12N \lceil \frac{N}{2} \rceil + 20 \lceil \frac{N}{2} \rceil \\
 L_2^{c,l} &= 26\lfloor \frac{N-1}{4} \rfloor \lceil \frac{N}{2} \rceil + 2\lceil \frac{N}{2} \rceil \gamma(\text{mod}_4 N) + 16\lceil \frac{N}{2} \rceil \\
 T_2^{c,t} &= \frac{1}{L_2^{c,t}}
 \end{aligned}$$

$$T_2^{c,r} = \frac{1}{12N \lceil \frac{N}{2} \rceil + 20 \lceil \frac{N}{2} \rceil}$$

$$T_2^{c,l} = \frac{1}{26 \lfloor \frac{N-1}{4} \rfloor \lceil \frac{N}{2} \rceil + 2 \lceil \frac{N}{2} \rceil \gamma(\text{mod}_4 N) + 16 \lceil \frac{N}{2} \rceil}$$

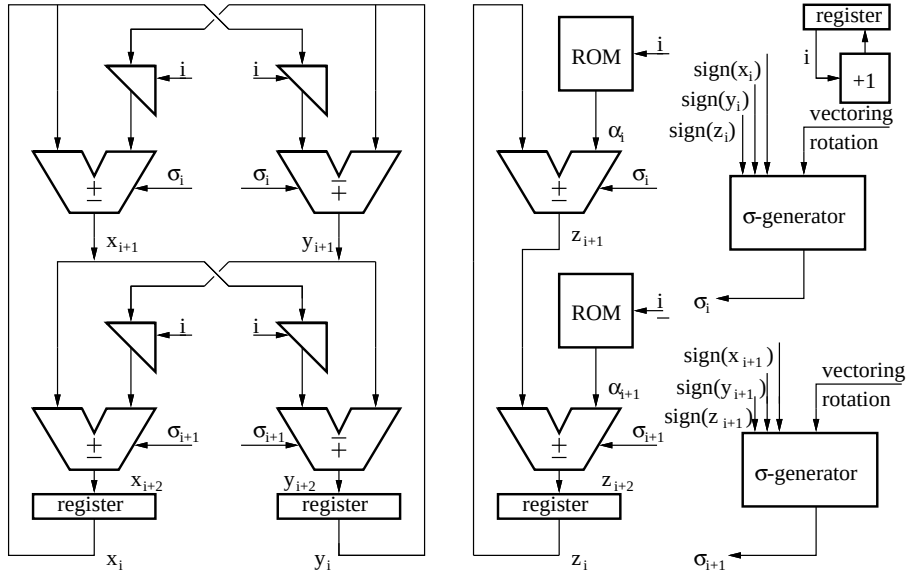


Figure 3 The Second-Order Combined Architecture

3.3 Higher-Order Combined Architectures

The combining transformation can be applied also at higher degrees to achieve a further reduction of the number of storing operations required to implement the CORDIC algorithm and, as a consequence, to reduce the latency and to increase the throughput, despite of the circuit complexity increase.

The use of symmetric fusion is in this case not feasible since the induced error becomes too high and cannot be removed by few additional operand bits and CORDIC iterations. For example, in the case of 4th-order merging, we fuse the iterations i , $\frac{N}{2} - i - 1$, $\frac{N}{2} + i$, $N - 1 - i$; it can be easily shown that the error is $2^{-\frac{N}{2}+2}$.

Cascaded fusion can be effectively used to create higher-order combined structures without introducing additional errors with respect to the traditional architecture of Section 3.1. The application of this technique is performed exactly as in Section 3.2 for the 2nd-order case. The resulting structure for the 4th-order case is given in Figure 4. Shifters treat N -bit words and are able to move them in $\frac{N}{4}$ different positions; also in this case the total circuit complexity of the shifting matrices is identical to the traditional case, but the decoder is simpler. The circuit complexity for adders and σ -generator is four times the traditional case. The latency is decreased since three out of four storing operations are avoided.

The structure for the k th-order case is similar. The circuit complexity $C_k^{c,t}$ is given by:

$$\begin{aligned}
C_k^{c,t} &= 2kC_{\text{shift}}(N, \lceil \frac{N}{k} \rceil) + 3kC_{\text{addsub}}(N) + 3C_{\text{acc}}(N) + kC_{\text{rom}}(N, \lceil \frac{N}{k} \rceil) + \\
&\quad + C_{\text{reg}}(\lceil \log_2 \lceil \frac{N}{k} \rceil \rceil) + C_{\text{inc}}(\lceil \log_2 \lceil \frac{N}{k} \rceil \rceil) + kC_{\sigma} \\
C_k^{c,r} &= 3N^2 + 36N + 10k + 39kN + 0.5k \lceil \log_2 \lceil \frac{N}{k} \rceil \rceil^2 + (\frac{N}{2} - k + 18) \lceil \log_2 \lceil \frac{N}{k} \rceil \rceil \\
C_k^{c,l} &= 3N^2 + 36N + 10k + 252k \lfloor \frac{N-1}{4} \rfloor + 3k\beta(\text{mod}_4 N) + \\
&\quad + 0.5k \lceil \log_2 \lceil \frac{N}{k} \rceil \rceil^2 + (\lceil \frac{N}{2} \rceil - k + 18) \lceil \log_2 \lceil \frac{N}{k} \rceil \rceil
\end{aligned}$$

The clock cycle time $\tau_k^{c,t}$, the latency $L_k^{c,t}$ and the throughput $T_k^{c,t}$ become, respectively:

$$\begin{aligned}
\tau_k^{c,t} &= k \max\{\tau_{\sigma}, \tau_{\text{shift}}(N, \lceil \frac{N}{k} \rceil), \tau_{\text{rom}}(N, \lceil \frac{N}{k} \rceil)\} + k\tau_{\text{addsub}}(N) + \tau_{\text{acc}}(N) \\
\tau_k^{c,r} &= 6Nk + 7k + 6 \\
\tau_k^{c,l} &= 13k \lfloor \frac{N-1}{4} \rfloor + k[\gamma(\text{mod}_4 N) + 5] + 6 \\
L_k^{c,t} &= \tau_k^{c,t} \lceil \frac{N}{k} \rceil \\
L_k^{c,r} &= 6Nk \lceil \frac{N}{k} \rceil + 7k \lceil \frac{N}{k} \rceil + 6 \lceil \frac{N}{k} \rceil \\
L_k^{c,l} &= 13k \lfloor \frac{N-1}{4} \rfloor \lceil \frac{N}{k} \rceil + k[5 + \gamma(\text{mod}_4 N)] \lceil \frac{N}{k} \rceil + 6 \lceil \frac{N}{k} \rceil \\
T_k^{c,t} &= \frac{1}{L_k^{c,t}}
\end{aligned}$$

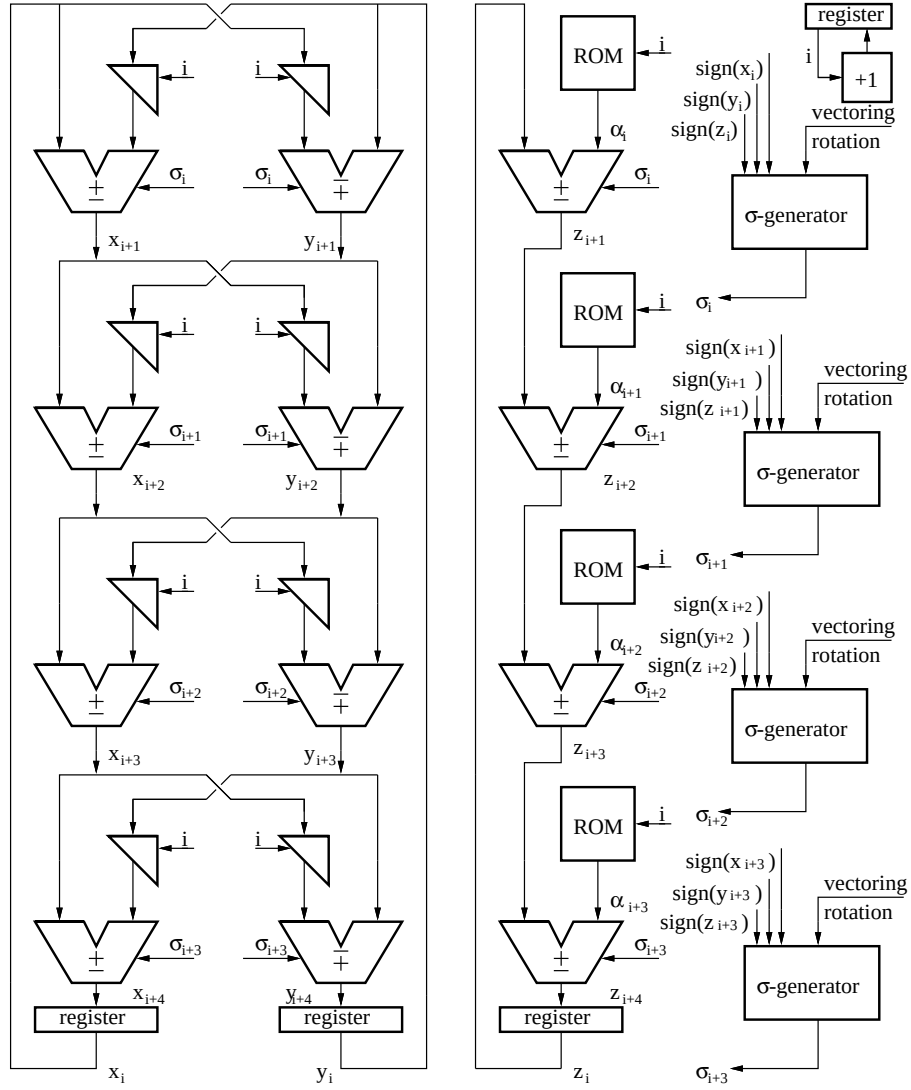


Figure 4 The Fourth-Order Combined Architecture

$$T_k^{c,r} = \frac{1}{6Nk \lceil \frac{N}{k} \rceil + 7k \lceil \frac{N}{k} \rceil + 6 \lceil \frac{N}{k} \rceil}$$

$$T_k^{c,l} = \frac{1}{13k \lfloor \frac{N-1}{4} \rfloor \lceil \frac{N}{k} \rceil + k[5 + \gamma(\text{mod}_4 N)] \lceil \frac{N}{k} \rceil + 6 \lceil \frac{N}{k} \rceil}$$

Figure 5 shows the maximum-order case, i.e., the one having order equal to N . All CORDIC iterations are completely cascaded and mapped onto separate circuits; this is practically the architecture presented in [27]. There are as many adders for each coordinate and σ -generators as the number Nof CORDIC iterations. The system becomes therefore a purely combinatoric circuit. No accumulator is necessary. Besides, no shifter is required since each shifter should move the operand into only one fixed position, i.e., it can be hardwired. The increase of circuit complexity and the clock cycle are maximum or quite-maximum (due to possible savings mentioned above); latency is minimum, while throughput becomes maximum among all combined architectures. The circuit complexity $C_N^{c,t}$, the clock cycle time $\tau_N^{c,t}$, the latency $L_N^{c,t}$ and the throughput $T_N^{c,t}$ become, respectively:

$$\begin{aligned} C_N^{c,t} &= 3NC_{\text{addsub}}(N) + 3C_{\text{acc}}(N) + NC_{\sigma} \\ C_N^{c,r} &= 39N^2 + 46N \\ C_N^{c,l} &= 252N \lfloor \frac{N-1}{4} \rfloor + 3N\beta(\text{mod}_4 N) + 46N \end{aligned}$$

$$\begin{aligned} \tau_N^{c,t} &= N\tau_{\sigma} + N\tau_{\text{addsub}}(N) + \tau_{\text{acc}}(N) \\ \tau_N^{c,r} &= 6N^2 + 7N + 6 \\ \tau_N^{c,l} &= 13N \lfloor \frac{N-1}{4} \rfloor + N[\gamma(\text{mod}_4 N) + 5] + 6 \end{aligned}$$

$$\begin{aligned} L_N^{c,t} &= \tau_N^{c,t} \\ L_N^{c,r} &= 6N^2 + 7N + 6 \\ L_N^{c,l} &= 13N \lfloor \frac{N-1}{4} \rfloor + N[\gamma(\text{mod}_4 N) + 5] + 6 \end{aligned}$$

$$\begin{aligned} T_N^{c,t} &= \frac{1}{L_N^{c,t}} \\ T_N^{c,r} &= \frac{1}{6N^2 + 7N + 6} \\ T_N^{c,l} &= \frac{1}{13N \lfloor \frac{N-1}{4} \rfloor + N[\gamma(\text{mod}_4 N) + 5] + 6} \end{aligned}$$

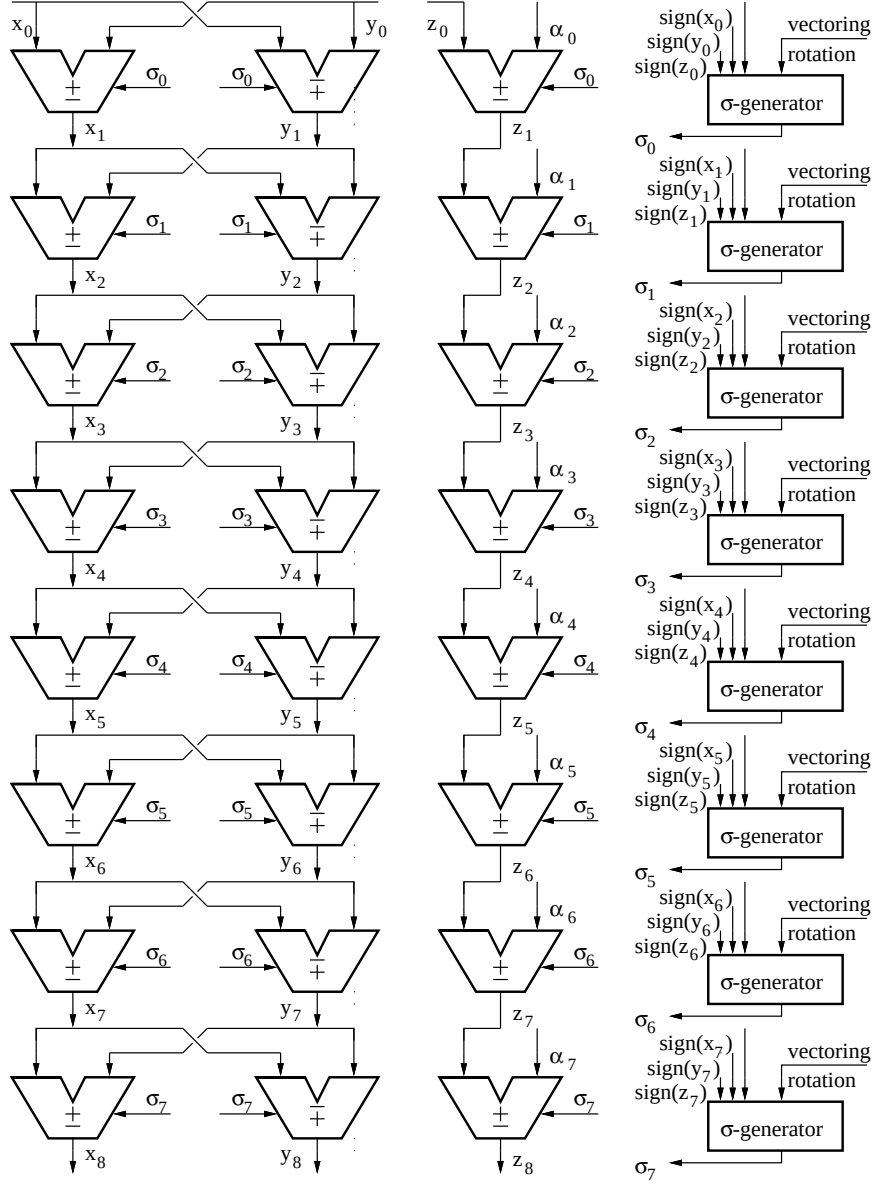


Figure 5 The Maximum-Order Combined Architecture

4 PIPELINED ARCHITECTURES

Unrolling is a transformation rule that can be applied to each sequential architecture described in Section 3 to reorganize its operations onto a combinatoric

structure. This rule maps each operation performed by the considered combined architecture onto a dedicated unit, so that no reusing of the same unit is performed during execution of the complete CORDIC algorithm as in the combined case. Since each unrolled structure contains a register where the combined counterpart applies a storing operation into the accumulator, it is automatically a pipelined architecture. The circuit complexity is highly increased since no time-multiplexing of components is exploited. Conversely, the throughput is highly increased since it becomes always equal to the inverse of the clock cycle time. In this Section, we show how the combined architectures can be unrolled by starting from the traditional CORDIC structure to the case in which all CORDIC steps are collapsed into the same pipeline stage.

4.1 First-Order Pipelined Architectures

Unrolling the traditional 1st-order combined architecture of Section 3.1 leads to design a pipelined structure having only one CORDIC iteration in each pipeline stage, i.e., having the maximum granularity of pipelining. We define the order of an unrolled architecture as the number of CORDIC iterations that are performed in a single pipeline stage.

The 1st-order pipelined architecture is shown in Figure 6. The circuit complexity is very high (maximum among the pipelined architectures) since not only all arithmetic operators are mapped onto a dedicated unit as in the maximum-order combined structure, but also there are as many pipeline (master-slave) registers in each coordinate as the number of CORDIC iterations. As in the maximum-order combined case, the shift operations may be hardwired since only one (a priori known and fixed) shifting is required within each pipeline stage. Therefore, the circuit complexity $C_1^{p,t}$, is:

$$\begin{aligned} C_1^{p,t} &= 3NC_{\text{addsub}}(N) + 3NC_{\text{acc}}(N) + NC_{\sigma} \\ C_1^{p,r} &= 75N^2 + 10N \\ C_1^{p,l} &= 36N^2 + 252N \left\lfloor \frac{N-1}{4} \right\rfloor + 3N\beta(\text{mod}_4 N) + 10N \end{aligned}$$

The clock cycle time $\tau_1^{p,t}$ required to drive the pipeline stages is:

$$\begin{aligned} \tau_1^{p,t} &= \tau_{\sigma} + \tau_{\text{addsub}}(N) + \tau_{\text{acc}}(N) \\ \tau_1^{p,r} &= 6N + 13 \\ \tau_1^{p,l} &= 13 \left\lfloor \frac{N-1}{4} \right\rfloor + \gamma(\text{mod}_4 N) + 11 \end{aligned}$$

It may be lightly smaller than the clock cycle time $\tau_1^{c,t}$ of the corresponding combined case due to the hardwired shifters, if the shifters of this latter case are slower than the σ -generator. The latency $L_1^{p,t}$ is:

$$\begin{aligned} L_1^{p,t} &= \tau_1^{p,t} N \\ L_1^{p,r} &= 6N^2 + 13N \\ L_1^{p,l} &= 13N \lfloor \frac{N-1}{4} \rfloor + N[\gamma(\text{mod}_4 N) + 11] \end{aligned}$$

In the pipelined structure, one new final CORDIC result is generated per each clock, i.e., the time elapsed between two subsequent final results is equal to the clock cycle time. As a consequence, according to the definition given in Section 3.1, the throughput $T_1^{p,t}$ becomes:

$$\begin{aligned} T_1^{p,t} &= \frac{1}{\tau_1^{p,t}} \\ T_1^{p,r} &= \frac{1}{6N + 13} \\ T_1^{p,l} &= \frac{1}{13N \lfloor \frac{N-1}{4} \rfloor + N[\gamma(\text{mod}_4 N) + 11]} \end{aligned}$$

Since the clock drives only one CORDIC operation per cycle, the throughput $T_1^{p,t}$ is maximum among all the possible solutions.

Since no additional approximation of the operations defined in the traditional algorithm are performed, the results have the same precision of the ones obtained by the 1st-order combined architecture.

4.2 Second-Order Pipelined Architectures

Unrolling the 2nd-order combined architecture we derive the 2nd-order pipelined structure, as it shown in Figure 7. Some circuit complexity is saved with respect to the 1st-order pipelined case since every other pipeline register is removed, while the shifters are hardwired with respect to the 2nd-order combined case. The circuit complexity $C_2^{p,t}$ is:

$$\begin{aligned} C_2^{p,t} &= 3NC_{\text{addsub}}(N) + 3\lceil \frac{N}{2} \rceil C_{\text{acc}}(N) + NC_{\sigma} \\ C_2^{p,r} &= 39N^2 + 36N \lceil \frac{N}{2} \rceil + 10N \\ C_2^{p,l} &= 36N \lceil \frac{N}{2} \rceil + 252N \lfloor \frac{N-1}{4} \rfloor + 3N\beta(\text{mod}_4 N) + 10N \end{aligned}$$

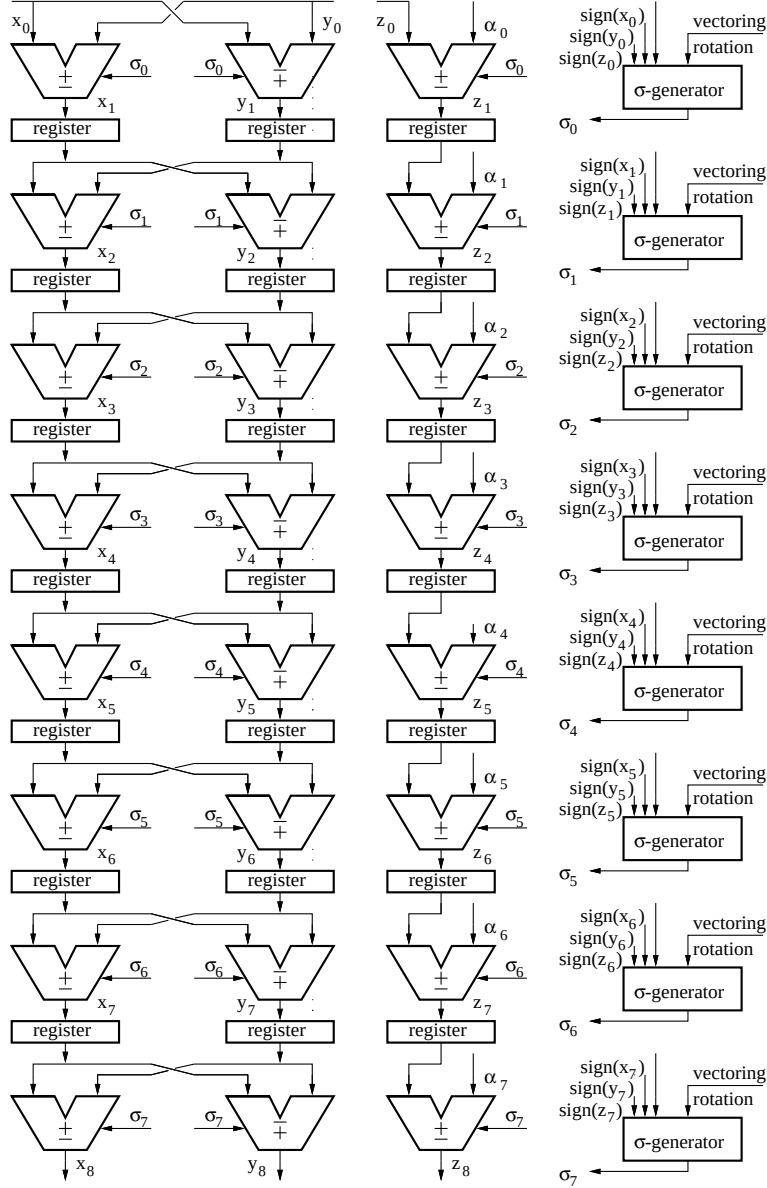


Figure 6 The First-Order Pipelined Architecture

As a consequence of the reduced granularity of pipelining, the clock cycle time $\tau_2^{p,t}$ is greater than the 1st-order case:

$$\tau_2^{p,t} = 2\tau_\sigma + 2\tau_{\text{addsub}}(N) + \tau_{\text{acc}}(N)$$

$$\begin{aligned}\tau_2^{p,r} &= 12N + 20 \\ \tau_2^{p,l} &= 26 \lfloor \frac{N-1}{4} \rfloor + 2\gamma(\text{mod}_4 N) + 16\end{aligned}$$

However, the latency $L_2^{p,t}$, is reduced with respect to the 1st-order pipelined case since a smaller number of storing operations is performed. It is:

$$\begin{aligned}L_2^{p,t} &= \tau_2^{p,t} \lceil \frac{N}{2} \rceil \\ L_2^{p,r} &= 12N \lceil \frac{N}{2} \rceil + 20 \lceil \frac{N}{2} \rceil \\ L_2^{p,l} &= 26 \lfloor \frac{N-1}{4} \rfloor \lceil \frac{N}{2} \rceil + 2 \lceil \frac{N}{2} \rceil \gamma(\text{mod}_4 N) + 16 \lceil \frac{N}{2} \rceil\end{aligned}$$

Conversely, the throughput $T_2^{p,t}$ is worse than in the previous architecture due to the inverse proportionality with respect to the clock cycle $\tau_2^{p,t}$:

$$\begin{aligned}T_2^{p,t} &= \frac{1}{\tau_2^{p,t}} \\ T_2^{p,r} &= \frac{1}{12N + 20} \\ T_2^{p,l} &= \frac{1}{26 \lfloor \frac{N-1}{4} \rfloor \lceil \frac{N}{2} \rceil + 2 \lceil \frac{N}{2} \rceil \gamma(\text{mod}_4 N) + 16 \lceil \frac{N}{2} \rceil}\end{aligned}$$

4.3 Higher-Order Pipelined Architectures

When k CORDIC iterations are compressed into the same pipeline stage, i.e., when the k th-order combined structure is unrolled, we obtain the k th-order pipelined architecture. In Figure 8, the case of the 4th-order pipelined structure is given. The figures of merit (circuit complexity $C_k^{p,t}$, clock cycle time $\tau_k^{p,t}$, latency $L_k^{p,t}$, and throughput $T_k^{p,t}$) become:

$$\begin{aligned}C_k^{p,t} &= 3NC_{\text{addsub}}(N) + 3 \lceil \frac{N}{k} \rceil C_{\text{acc}}(N) + NC_{\sigma} \\ C_k^{p,r} &= 39N^2 + 36N \lceil \frac{N}{k} \rceil + 10N \\ C_k^{p,l} &= 36N \lceil \frac{N}{k} \rceil + 252N \lfloor \frac{N-1}{4} \rfloor + 3N\beta(\text{mod}_4 N) + 10N \\ \tau_k^{p,t} &= k\tau_{\sigma} + k\tau_{\text{addsub}}(N) + \tau_{\text{acc}}(N)\end{aligned}$$

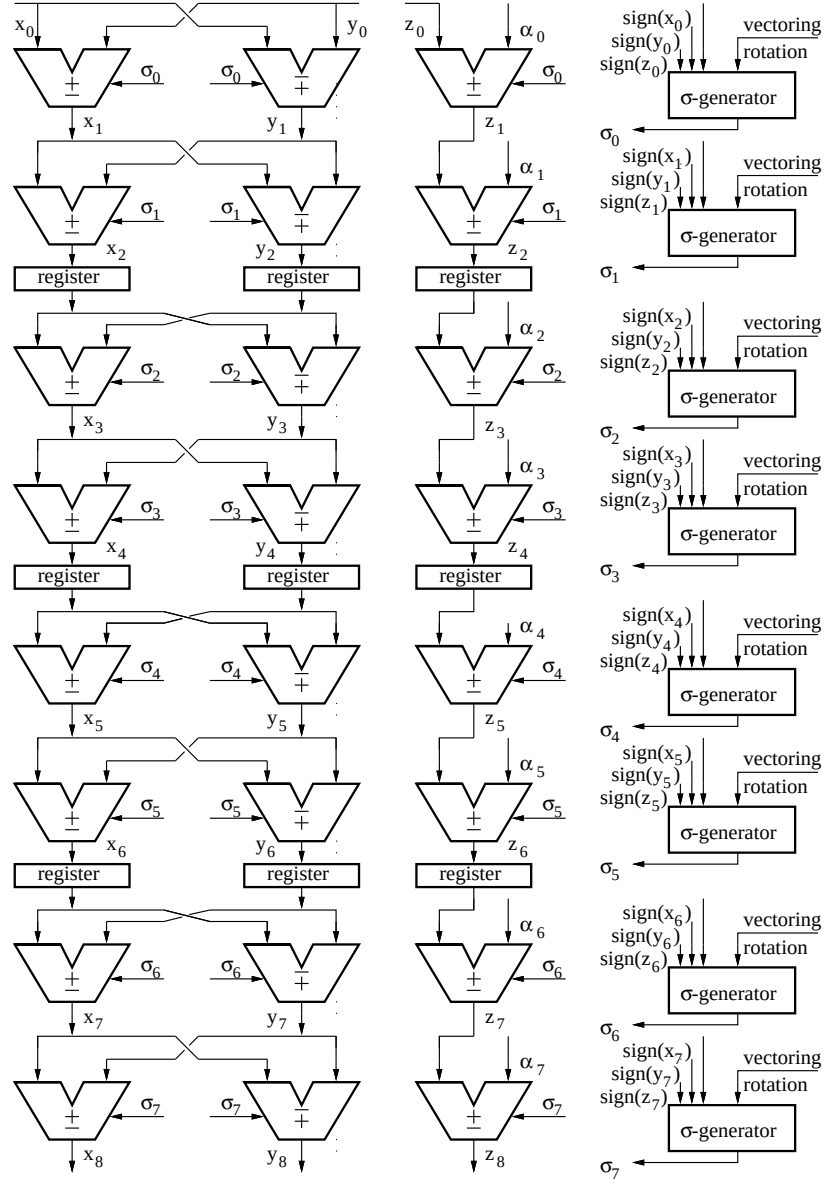


Figure 7 The Second-Order Pipelined Architecture

$$\tau_k^{p,r} = 6Nk + 7k + 6$$

$$\tau_k^{p,l} = 13k \lfloor \frac{N-1}{4} \rfloor + k[\gamma(\text{mod}_4 N) + 5] + 6$$

$$L_k^{p,t} = \tau_k^{p,t} \lceil \frac{N}{k} \rceil$$

$$L_k^{p,r} = 6Nk \lceil \frac{N}{k} \rceil + 7k \lceil \frac{N}{k} \rceil + 6 \lceil \frac{N}{k} \rceil$$

$$L_k^{p,l} = 13k \lfloor \frac{N-1}{4} \rfloor \lceil \frac{N}{k} \rceil + k[5 + \gamma(\text{mod}_4 N)] \lceil \frac{N}{k} \rceil + 6 \lceil \frac{N}{k} \rceil$$

$$T_k^{p,t} = \frac{1}{\tau_k^{p,t}}$$

$$T_k^{p,r} = \frac{1}{6Nk + 7k + 6}$$

$$T_k^{p,l} = \frac{1}{13k \lfloor \frac{N-1}{4} \rfloor \lceil \frac{N}{k} \rceil + k[\gamma(\text{mod}_4 N) + 5] \lceil \frac{N}{k} \rceil + 6 \lceil \frac{N}{k} \rceil}$$

As the order k increases, the circuit complexity decreases since registers are progressively removed. The clock cycle time increases since more CORDIC iterations are merged in the same pipeline stage. The latency decreases progressively since less storing operations are performed. The throughput decreases too since the pipeline clock time increases.

The extreme conditions are achieved when the order becomes maximum, i.e., equal to N . In this case, all CORDIC iterations are executed within the same pipeline stage: the pipeline granularity is minimum. Operations are completely cascaded and performed by different hardware components, no pipeline register is contained in the structure to separate groups of iterations, and all shifters are hardwired: therefore, this architecture coincides exactly with the maximum-order combined structure presented in Section 3.3.

5 ARCHITECTURAL EVALUATION

The evaluation of the architectural approaches presented in the previous Sections can be based on the figures of merit introduced there, namely, the circuit complexity, the clock cycle time, the latency, and the throughput. The use of high-level estimations during the initial stages of the design process concerned with the architectural design allows for abstracting from the specific implementation technologies that could be adopted for the physical realization.

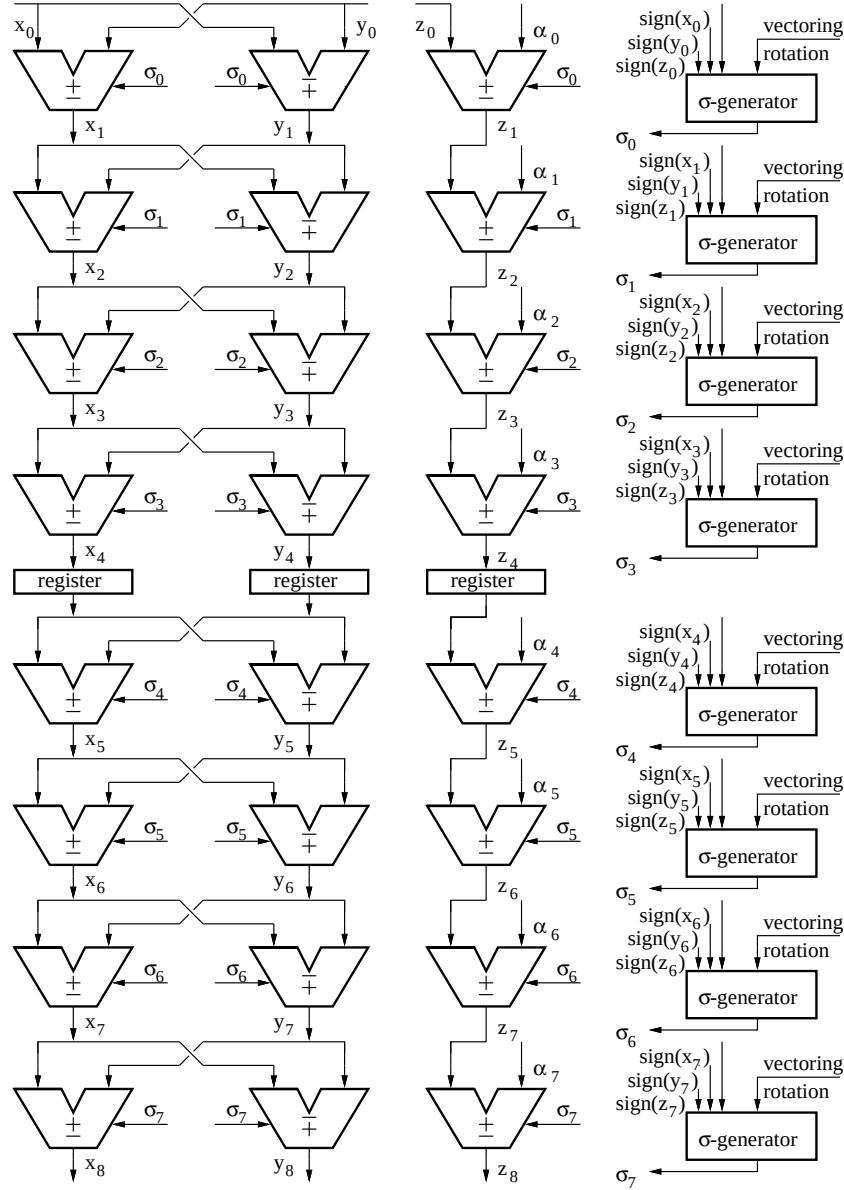


Figure 8 The Fourth-Order Pipelined Architecture

The summary of the analysis performed in the previous sections is graphically shown in Figure ??, for the case of 16-bit operands and, as a consequence, of

$N = 16$ CORDIC iterations; similar results can be achieved for different values of N .

As the circuit complexity is concerned (Fig. 8a), the combined architectures have an increasing complexity as the order increases; conversely, the size of the pipelined ones progressively decreases till coinciding at the maximum order with the combined case.

The clock cycle time and the latency (Figs. 8b and 8c, respectively) are identical since the pipelined case is simply the unrolled version of the combined one, being the latency of the σ -generator usually higher than the other components working in parallel with it.

For both the architectural approaches, the clock cycle time increases as the order increases since more CORDIC iterations must be accommodated in the same clock cycle.

The latency has a non-monotonic behavior with respect to the order since additional void CORDIC iteration must be introduced if N is not a multiple of the order. Let's consider the case of the combined architecture having order k (with N not multiple of k), the final result - expected at the nominal step N - is not available at the output of the k -th stage (i.e., at the input of the accumulator register) after $\lceil \frac{N}{k} \rceil$ clock cycles (this happens only in the case of N multiple of k). The expected result is available during the $\lceil \frac{N}{k} \rceil$ -th clock cycle at the output of the $\text{mod}_k(N)$ -th stage (being $\text{mod}_k(N)$ the residue of N modulo k). This implies that we need to wait the propagation of the result through the subsequent $k - \text{mod}_k(N)$ stages; since propagation through the extra CORDIC iterations must not modify the result, the rotation angles of these iterations must be zero and the multiplexers in the related adder/subtractors must be lightly modified in order to impose that zero is added to the propagated result during the last clock cycle. This leads to waste part of the time during the last clock cycle without doing any meaningful CORDIC iteration. We obtain a full exploitation of the hardware only for N multiple of k ; by considering only these cases, the latency is lightly decreasing with the increase of the order since some storing operations in the accumulator register are avoided (see the dashed-line in Fig. 8c).

Usually, we cannot extract the expected final result directly from the output of the stage where it is generated since the architecture is a sequential machine and it is used by the host computing system in such a way (i.e., by assuming a regular clock cycle time and the availability of the result only at the end of

the proper clock cycle even if it is ready before the end of such a cycle). An alternative solution consists of extracting the final result directly from where it is generated and of adopting an irregular clocking scheme in which all cycles have the same length (the one given in Section 3) except the last one (in which the cycle time is sufficient to execute only the meaningful CORDIC iterations). This approach leads obviously to irregularities and to increase the complexity of the design of the other component of the host computing system.

A similar problem occurs also in the case of pipelined architectures. The only difference is that, in the pipelined cases, we can avoid to introduce CORDIC stages for the void extra CORDIC iterations. However, the last pipeline stage is not identical to the previous ones if N is not a multiple of k . In this case, the dashed line of Fig. 8c gives the theoretic latency, i.e., the minimum time after which the final result is steady when the count is started from the presentation of the primary inputs. Also in this case, there are two alternatives for the designer of the host computing system in which the CORDIC processor is used. In the simplest and safest case, he can consider a regular clock scheme having period equal to the clock cycle time given in Section 4; this leads to a regular generation of pipeline clock signal, but implies to waste some time in the last pipeline stage even if the result is already available, i.e., by considering the actual latency given by the solid line in Fig. 8c. The second solution is based on the use of an irregular clocking scheme, in which the pipeline stages but the last one are driven by the same clock signal defined as in Section 4, while the output of last pipeline stage is used as soon as it becomes steady (i.e., before the completion of a clock cycle); in this case, the actual latency coincides with the theoretic one, but an accurate timing of the input presentation and result extraction must be adopted to obtain a correct generation and use of the CORDIC results.

The behavior of the throughput is shown in Fig. 8d. In the case of the pipelined architectures, whatever latency is considered, one final CORDIC result is generated at each clock cycle. This implies that the throughput is decreasing as the order increases. In the case of the combined structures, the throughput has a non-monotonic behavior (as pointed out by the logarithmic scale of Fig. 8d) since it suffers from the same problem discussed about the latency. When the clock cycle is fully exploited (i.e., N multiple of k), the throughput lightly increases since some storing time is saved by increasing the number of CORDIC iteration performed in the same clock cycle (see the dashed line in Fig. 8d).

6 DESIGN GUIDELINES AND CONCLUSIONS

The choice of the optimum architecture for a CORDIC processor is a complex and time-consuming task involving the analysis of different figures of merit for a number of structures proposed in the literature. To avoid the design of several prototypes in order to identify the characteristics and the performances of each of them, a high-level analysis must be carried on.

In this chapter, we presented a unified view of the architectural solutions as a continuous wide spectrum of possible alternatives. Structures proposed in the literature are in this spectrum or can be easily viewed as possible variants of the presented structures. Also new intermediate solutions are presented and evaluated to support the optimal choice of the designer by taking into account the application constraints and requirements on precision, circuit complexity, latency, and throughput, contemporaneously.

For the given application, the designer can derive the minimum throughput which is sufficient to deliver the results for subsequent operations, in particular when massive-computing applications are envisioned. He can also evaluate the maximum latency, which is relevant in several control and robotics applications. When integration of the architecture in a VLSI/ULSI/WSI device is constrained by the size of the processor or by the power consumed, the maximum circuit complexity may be identified as a high-level indicator of the circuit silicon area or of the power consumption. The clock cycle time may be lower bounded by the specific integration technology, according to the characteristic behavior of transistors and transmission delays.

With the actual values of these constraints, in the three-dimensional architectures' space introduced in this chapter, the designer can identify the subsets of solutions satisfying the constraints. Among these solutions (if existing), he can chose the one which optimizes the most relevant figure of merit for the specific application, or the one that best balances two or more of these figures.

If the maximum circuit complexity is quite low, only some combined structures can be considered; if it is high enough, both combined and pipelined solutions may be used. Similarly, when the minimum throughput is not too high, both combined and pipelined approaches are available, but only pipelined structures can provide the higher throughputs. When a constraint on the maximum latency is given, there are solutions with the same order both in the combined

part and in the pipelined one, since practically both of these classes have the same latency. Similarly for the clock cycle time.

APPENDIX A

DETAILED EVALUATIONS

As a reference, we include in this appendix the detailed evaluations of the figures of merit for the CORDIC processors. Table A.1 contains the circuit complexity and the latency of the basic blocks composing the architectures (namely, adder/subtractors in all different variations, pipeline registers, and σ -generators). Table A.2 summarizes the order (with respect to N and k) of circuit complexity, clock cycle time, latency, and throughput for the different kinds of adder/subtractors.

Fig. 8 - Evaluation of the CORDIC processor designs: circuit complexity (a), clock cycle time (b), latency (c), and throughput (d).

REFERENCES

- [1] Volder, J., "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, Vol. EC-8, pp. 330-334, 1959.
- [2] Walther, J., "A Unified Algorithm For Elementary Functions," *Spring Joint Computer Conference Proceedings*, Vol. 38, pp. 379-385, 1971.
- [3] Shelin, C., "Calculator Function Approximation," *Amer. Math. Monthly*, Vol. 90, No. 5, May 1983,
- [4] Muller, J., "Discrete Basis and Computation of Elementary Functions," *IEEE Transactions on Computers*, Vol. C-34, pp. 857-862, 1985.
- [5] Delosme, J., "CORDIC Algorithms: Theory and Extensions," *Proc. SPIE*, Vol. 1152, pp. 131-145, 1989.

Table A.1 Circuit complexity and latency of the basic blocks

basic block	circuit complexity	latency
accumulator/pipeline register	$12N$	6
iteration index register	$12\lceil\log_2 N\rceil$	6
σ -generator	10	5
shifter	N^2	$\lceil\log_2 \lceil\log_2 N\rceil\rceil + 2$
adder/subtractor ripple carry	$13N$	$6N + 2$
adder/subtractor CCLA	$84\lfloor\frac{N-1}{4}\rfloor + \beta(\text{mod}_4 N)$ being $\beta(d)$ equal to 64, 12, 26, and 43, for d equal to 0, 1, 2, and 3.	$13\lfloor\frac{N-1}{4}\rfloor + \gamma(\text{mod}_4 N)$ being $\gamma(d)$ equal to 12, 8, 10, and 12, for d equal to 0, 1, 2, and 3.
iteration index incrementer ripple carry	$6\lceil\log_2 N\rceil$	$N + 5$
iteration index incrementer CCLA	$66\lfloor\frac{\lceil\log_2 N\rceil - 1}{4}\rfloor + \beta'(\text{mod}_4 \lceil\log_2 N\rceil)$ being $\beta'(d)$ equal to 52, 9, 20, and 34, for d equal to 0, 1, 2, and 3.	$13\lfloor\frac{N-1}{4}\rfloor + \gamma(\text{mod}_4 N)$ being $\gamma(d)$ equal to 12, 8, 10, and 12, for d equal to 0, 1, 2, and 3.
rotation angles ROM	$N^2 + \lceil\log_2 N\rceil(\frac{\lceil\log_2 N\rceil}{2} + \frac{N}{2} - 1)$	$\lceil\log_2 \lceil\log_2 N\rceil\rceil + 2$

- [6] Hsiao, H. and Delosme, J., "The CORDIC Householder Algorithm," *Proceedings of the 10th Symposium on Computer Arithmetic*, pp. 256-263, 1991.
- [7] Lee, J. and Lang, T., "Floating Point Implementation of Redundant CORDIC for QR Decomposition," Technical Report #CSD-890044, Department of Computer Science, UCLA, 1989.
- [8] Ercegovac, M., and Lang, T., "Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD," *IEEE Transactions on*

Table A.2 Magnitude order of circuit complexity, cycle time, latency, and throughput for the CORDIC processors

processor type	circuit complexity	cycle time	latency	throughput
combined ripple	N^2, Nk	Nk	N^2	N^{-2}
combined CCLA	N^2, Nk	Nk	N^2	N^{-2}
pipelined ripple	$N^2, \frac{N^2}{k}$	Nk	N^2	$N^{-1}k^{-1}$
pipelined CCLA	$N^2, \frac{N^2}{k}$	Nk	N^2	$N^{-1}k^{-1}$

Computers, Vol. 39, 1990, pp. 725-740.

- [9] Takagi, N., Asada, T., and Yajima, S., "Redundant CORDIC Methods with a Constant Scale Factor for Sine and Cosine Computation," *IEEE Transactions on Computers*, Vol. 40, 1991, pp.989-995.
- [10] Lee, J. and Lang, T., "SVD by Constant Factor-Redundant-CORDIC," *Proceedings 10th Symposium on Computer Arithmetic*, Grenoble, France, pp. 264-271, 1991.
- [11] Lee, J. and Lang, T., "Constant-Factor Redundant CORDIC for Angle Calculation and Rotation," *IEEE Transactions on Computers*, Vol. 41, pp. 1016-1025, 1992.
- [12] Duprat, J. and Muller, J., "The CORDIC Algorithm: New Results for Fast VLSI Implementation," *IEEE Transactions on Computers*, Vol. 42, pp. 168-178, 1993.
- [13] Ercegovic, M. and Lang, T., "On the Fly Conversions of Redundant into Conventional Representations," *IEEE Transactions on Computers*, Vol. C-36, pp. 895-897, 1987.
- [14] Lin, H. and Sips, H., "On-Line CORDIC Algorithms," *IEEE Transactions on Computers*, Vol. 39, pp. 1030-1037, 1990.
- [15] Andrews, M. and Eggerding, D., "A Pipelined Computer Architecture for Unified Elementary Function Evaluation," *Computer Electronic Engineering*, Vol. 5, 1978, pp. 189-202.
- [16] Delosme, J., "VLSI Implementation of Rotations in Pseudo Euclidean Spaces," *Proceedings of the IEEE International Conference on Acoustic, Speech, and Signal Processing*, Vol. 2, 1983, pp. 927-930.

- [17] Sung, T., Parng, T., Hu, Y., and Chou, P., "Design and Implementation of a VLSI CORDIC Processor," *Proceedings of the 1986 International Symposium on Circuits and Systems*, Vol. 3, 1986, pp. 934-935.
- [18] Cavallaro, J. and Luk, F., "CORDIC Arithmetic for an SVD Processor," *Proceeding of the 8th Symposium on Circuits and Systems*, 1988, pp. 2043-2047.
- [19] De Lange, A., Van der Hoeven, A., Deprettere, E., and Bu, J., "An Optimal Floating-Point Pipeline CMOS CORDIC Processor," *Proceeding of the 1988 IEEE International Symposium on Circuits and Systems*, 1989, pp. 2043-2047.
- [20] Harber, R., Li, J., Xu, X., and Bass, S., "Bit-Serial CORDIC Circuits for Use in a VLSI Silicon Compiler," *Proceedings of the 1989 IEEE International Symposium on Circuits and Systems*, Vol. 1, 1989, pp.154-157.
- [21] Kundmund, R., and et al., "CORDIC Processor with Carry Save Architecture," *Proceeding of the 1990 European Solid State Circuits Conference*, Grenoble, Sept. 1990, pp. 193-196.
- [22] De Lange, A., and Deprettere, E., "Design and Implementation of a Floating-Point Quasi-Systolic General Purpose CORDIC Rotator for High-Rate Parallel Data and Signal Processing," *Proceeding of the 10th Symposium on Computer Arithmetic*, 1991, pp. 272-281.
- [23] Lee, J. and Lang, T., "SVD by Constant Factor-Redundant-CORDIC," *Proceeding of the 10th Symposium on Computer Arithmetic*, Grenoble, France, pp. 264-271, June 1991.
- [24] Delosme, J. and Hsiao, S., "CORDIC Algorithms in Four Dimensions," *Proceedings of SPIE – The International Society for Optical Engineering*, Vol. 1348, San Diego, CA, pp. 349-360, July 1990.
- [25] Lee, J. and Lang, T., "Constant-Factor Redundant CORDIC for Angle Calculation and Rotation," *IEEE Transactions on Computers*, Vol. 41, 1992, pp. 1016-1025.
- [26] Deprettere, E., Dewilde, P., and Udo, U., "Pipelined CORDIC Architectures for Fast VLSI Filtering and Array Processing," *IEEE Transactions on Signal Processing*, Vol. 40, 1992, pp. 834-844.
- [27] Timmermann, D., Hahn, H., and Hosticka, B., "Low Latency Time CORDIC Algorithms," *IEEE Transactions on Computers*, Vol. 41, 1992, pp. 1010-1015.

- [28] Wang, S. and Swartzlander, E., "Merged CORDIC Algorithm," *Proceedings of the IEEE 1995 International Symposium on Circuits and Systems*, Seattle, WA, pp. 1988-1991, April 1995.
- [29] Wang, S. and Swartzlander, E., "Critically Damped CORDIC Algorithm", *Proceedings of the 37th Midwest Symposium on Circuits and Systems*, Lafayette, LA, pp. 253-256, August 1994.
- [30] Antelo, E., Bruguera, J., Villalba, J., and Zapata, E., "Redundant CORDIC Rotator Based on Parallel Prediction," *Proceedings of the 12th International Symposium on Computer Arithmetic*, Bath, UK, pp. 172-1791, July 1995.
- [31] Dawid, H. and Meyr, H., "High Speed Bit-level Pipelined Architectures for Redundant CORDIC Implementation," *Proceedings of the 1992 International Conference on Application-Specific Array Processors*, Berkeley, CA, pp. 358-372, 1992.
- [32] Cochran, D., "Algorithm and Accuracy in the HP35," *Hewlett Packard Journal*, Vol. 23, 1972, pp. 10-11.
- [33] Haviland, G., and Tuszynski, A., "A CORDIC Arithmetic Processor Chip," *IEEE Transactions on Computers*, Vol. C-29, 1980, pp. 68-79.
- [34] Williams, F., "The CORDIC Algorithm—Cast in Silicon," *Electronic Engineering*, Vol. 61, pp. 47-50, 1989.
- [35] Hu, Y., "The Quantization Effects of the CORDIC Algorithm," *IEEE Transactions on Signal Processing*, Vol. 40, 1992, pp. 834-844.
- [36] Hu, Y. and Naganathan, S., "A Novel Implementation of Chirp Z-Transformation Using a CORDIC Processor," *IEEE Transactions on ASSP*, Vol. 38, pp. 352-354, 1990.
- [37] Hu, Y. and Naganathan, S., "An Angle Recoding Method for CORDIC Algorithm Implementation," *IEEE Transactions on Computers*, Vol. 42, pp. 99-102, 1993.
- [38] Wang, S., Piuri, V., and Swartzlander, E., "A Unified View of CORDIC Processor Design," Department of Electronics and Information, Politecnico di Milano, 20133 Milano, Italy, Int. Rep. No. 95-042, September 1995.
- [39] Wang, S., Piuri, V., and Swartzlander, E., "Granularly-Pipelined CORDIC Processor for Sine and Cosine Generators," *1996 IEEE International Conference on Acoustics, Speech and Signal Processing*, Atlanta, Georgia, May 1996.

- [40] Wang, S., Piuri, V., and Swartzlander, E., "The Hybrid CORDIC Algorithm," Submitted to IEEE Transactions on Computer.
- [41] Despain, A., "Very Fast Fourier Transform Algorithm for Hardware Implementation," *IEEE Transactions on Computers*, Vol. C-28, pp. 333-341, 1979.
- [42] Timmermann, D., Hahn, H., and Hosticka, B., "A programmable CORDIC chip for digital signal processing applications," *IEEE Journal of Solid-State Circuits*, Vol. 26, pp. 1317-1321, 1991.
- [43] Chown, P., "VLSI Design of a Pipelined CORDIC Processor," Research Report #164, Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK, 1990.
- [44] Chown, P., "Notes on the Design of a Barrel Shifter for the Warwick Pipelined CORDIC," Research Report #161, Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK, 1990.
- [45] Cosnard, M., Guyot, A., Hochet, B., Muller, J., Ouauicha, H., Paul, P., and Aysman, E., "The FELIN Arithmetic Processor Chip", *Proceedings of the 8th Symposium on Computer Arithmetic*, pp. 107-112, 1987.
- [46] Curtis, T., Allison, P., and Howard, J., "A CORDIC Processor for Laser Trimming," *IEEE Micro*, Vol. 6, pp. 61-71, June 1986.
- [47] Hemkumar, N. and Cavallaro, J., "Efficient Complex Matrix Transformations with CORDIC," *Proceedings of the 11th Symposium on Computer Arithmetic*, pp. 122-129, 1993.
- [48] Hekstra, G. and Deprettere, E., "Floating Point CORDIC," *Proceedings of the 11th Symposium on Computer Arithmetic*, pp. 130-137, 1993.
- [49] Koren, I. and Zinaty, O., "Evaluating Elementary Functions in a Numerical Coprocessor Based on Rational Approximations," *IEEE Transactions on Computers*, Vol. 39, pp. 1030-1037, 1990.
- [50] Mazenc, C., Merrheim, X., and Muller, J., "Computing Functions arccos and arcsin Using CORDIC," *IEEE Transactions on Computers*, Vol. 42, pp. 99-102, 1993.
- [51] Kota, K. and Cavallaro, J., "Numerical Accuracy and Hardware Tradeoffs for CORDIC Arithmetic for Special-Purpose Processor," *IEEE Transactions on Computers*, Vol. 42, pp. 769-779, 1993.

- [52] Cavallaro, J. and Luk, F., "CORDIC Arithmetic for an SVD Processor," *Proceeding of the 8th Symposium on Computer Arithmetic*, Como, Italy, pp. 113-120, 1987 and *Journal of Parallel and Distributed Computing*, Vol. 5, pp. 271-290, 1988.
- [53] Cavallaro, J. and Luk, F., "Architectures for a CORDIC SVD Processor," *Proc. SPIE, Real Time Signal Processing IX*, Vol. 698, pp. 45-53, 1986.
- [54] Cavallaro, J. and Elster, A., "Complex Matrix Factorizations with CORDIC Arithmetic," Technical Report 89-1071, Department of Computer Science, Cornell University, 1989.
- [55] Jones, K., "Parallel DFT Computation on Bit-serial Systolic Processor Arrays," *IEE Proceedings. Part E, Computers and Digital Techniques*, Vol. 140, pp. 10-18, 1993.
- [56] Chang, L. and Lee, S., "Systolic Arrays for the Discrete Hartley transform," *IEEE Transactions on Signal Processing*, Vol. 39, pp. 2411-2418, 1991.
- [57] Timmermann, D., Hahn, H., and Hosticka, B., "Hough Transform Using CORDIC Method," *Electronics Letters*, Vol. 25, pp. 105-206, 1989.
- [58] Despain, A., "Fourier Transform Computations Using CORDIC Iterations," *IEEE Transactions on Computers*, Vol. C-23, pp.993-1001, 1974.
- [59] Hahn, H., Hosticka, B., and Timmermann, D., "Alternative Signal Processor Arithmetic for Modified Implementation of a Normalised Adaptive Channel Equaliser," *IEE Proceedings. Part F, Radar and Signal Processing*, Vol. 139, pp. 36-42, 1992.
- [60] Regalia, P. and Loubaton, P., "Rational Subspace Estimation Using Adaptive Lossless Filters," *IEEE Transactions on Signal Processing*, Vol. 40, pp. 2392-2405, 1992.
- [61] Hu Y. and Lian, H., "CALF: a CORDIC Adaptive Lattice Filter," *IEEE Transactions on Signal Processing*, Vol. 40, pp. 990-993, 1992.
- [62] Tu, P. and Ercegovic, M., "Application of On-Line Arithmetic Algorithms to the SVD Computation: Preliminary Results," *Proceedings of the 10th Symposium on Computer Arithmetic*, pp. 246-255, 1991.