Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 794891, 19 pages
doi:10.1155/2010/794891

Review Article

CORDIC Architectures: A Survey

B. Lakshmi and A. S. Dhar

Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology, Kharagpur, West Bengal 721302, India

Correspondence should be addressed to B. Lakshmi, lkodali93@gmail.com

Received 6 October 2009; Accepted 10 January 2010

Academic Editor: Kiyoung Choi

In the last decade, CORDIC algorithm has drawn wide attention from academia and industry for various applications such as DSP, biomedical signal processing, software defined radio, neural networks, and MIMO systems to mention just a few. It is an iterative algorithm, requiring simple shift and addition operations, for hardware realization of basic elementary functions. Since CORDIC is used as a building block in various single chip solutions, the critical aspects to be considered are high speed, low power, and low area, for achieving reasonable overall performance. In this paper, we first classify the CORDIC algorithm based on the number system and discuss its importance in the implementation of CORDIC algorithm. Then, we present systematic and comprehensive taxonomy of rotational CORDIC algorithms, which are subsequently discussed in depth. Special attention has been devoted to the higher radix and flat techniques proposed in the literature for reducing the latency. Finally, detailed comparison of various algorithms is presented, which can provide a first-order information to designers looking for either further improvement of performance or selection of rotational CORDIC for a specific application.

1. Introduction

The current research in the design of high speed VLSI architectures for real-time digital signal processing (DSP) algorithms has been directed by the advances in the VLSI technology, which have provided the designers with significant impetus for porting algorithm into architecture. Many of the algorithms used in DSP and matrix arithmetic require elementary functions such as trigonometric, inverse trigonometric, logarithm, exponential, multiplication, and division functions. The commonly used software solutions for the digital implementation of these functions are table lookup method and polynomial expansions, requiring number of multiplication and additions/subtractions. However, digit-by-digit methods exist for the evaluation of these elementary functions, which compute faster than software solutions.

Some of the digit-by-digit methods for the computation of the above mentioned elementary functions were described by Henry Briggs in 1624 in “Arithmetica Logarithmica” [1, 2]. These are iterative pseudo division and pseudo multiplication processes, which resemble repeated-addition multiplication and repeated-subtraction division. In 1959, Volder has proposed a special purpose digital computing unit known as COordinate Rotation DIgital Computer (CORDIC), while building a real time navigational computer for use in an aircraft [3, 4]. This algorithm was initially developed for trigonometric functions which were expressed in terms of basic plane rotations.

The conventional method of implementation of 2D vector rotation shown in Figure 1 using Givens rotation transform is represented by the equations

$$\begin{aligned} x_{\text{out}} &= x_{\text{in}} \cos\theta - y_{\text{in}} \sin\theta, \\ y_{\text{out}} &= x_{\text{in}} \sin\theta + y_{\text{in}} \cos\theta, \end{aligned} \tag{1}$$

where $(x_{\text{in}}, y_{\text{in}})$ and $(x_{\text{out}}, y_{\text{out}})$ are the initial and final coordinates of the vector, respectively. The hardware realization of these equations require four multiplications, two additions/subtractions and accessing the table stored in memory for trigonometric coefficients. The CORDIC algorithm computes 2D rotation using iterative equations employing shift and add operations. The versatility of CORDIC is enhanced by developing algorithms on the same basis to convert between binary to binary coded decimal

(BCD) number representation by Daggett in 1959 [5]. These iterative methods were described using decimal radix for the design of powerful small machines by Meggitt in 1962 [6]. Subsequently, Walther in 1971 [7, 8] has proposed a unified algorithm to compute rotation in circular, linear, and hyperbolic coordinate systems using the same CORDIC algorithm, embedding coordinate systems as a parameter.

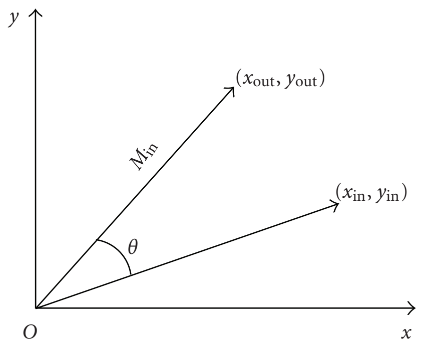
During the last 50 years of the CORDIC algorithm a wide variety of applications have emerged. The CORDIC algorithm has received increased attention after an unified approach is proposed for its implementation [7]. Thereafter, CORDIC based computing has been the choice for scientific calculator applications and HP-2152A co-processor, HP-9100 desktop calculator, HP-35 calculator are a few such devices based on the CORDIC algorithm [1, 8]. The CORDIC arithmetic processor chip is designed and implemented to perform various functions possible in rotation and vectoring mode of circular, linear, and hyperbolic coordinate systems [9]. Since then, CORDIC technique has been used in many applications [10], such as single chip CORDIC processor for DSP applications [11–15], linear transformations [16–21], digital filters [17], [22–24], and matrix based signal processing algorithms [25, 26]. More recently, the advances in the VLSI technology and the advent of EDA tools have extended the application of CORDIC algorithm to the field of biomedical signal processing [27], neural networks [28], software defined radio [29], and MIMO systems [30] to mention a few.

Although CORDIC may not be the fastest technique to perform these operations, it is attractive due to its potential for efficient and low cost implementation of a large class of applications. Several modifications have been proposed in the literature for the CORDIC algorithm during the last two decades to provide high performance and low cost hardware solutions for real time computation of a two dimensional vector rotation and transcendental functions.

A new type of arithmetic operation called fast rotations or orthonormal μ-rotations over a set of fixed angles is proposed [31]. These orthonormal μ-rotations are based on the idea of CORDIC and share the property that performing the rotation requires a minimal number of shift-add operations. These fast rotations methods form a viable low cost alternative to the CORDIC arithmetic for certain applications such as FIR filter banks for image processing, the generation of spherical sample rays in 3D graphics, and the computation of eigenvalue decomposition and singular value decomposition.

We have carried out the critical study of different architectures proposed in the literature for 2D rotational CORDIC in circular coordinate system, to initiate the work for further latency reduction or throughput improvement. In this paper, we will review the architectures proposed for rotational CORDIC. Specifically, we focus on redundant unfolded architectures, employing techniques suitable to increase throughput and reduce latency.

The rest of the paper is organized as follows. In Section 2, the basics of redundant arithmetic are presented. In Section 3, we present a review of generalized CORDIC algorithm, radix-2 and radix-4 CORDIC algorithms. In Section 4, general architectures being employed in literature for the implementation of the CORDIC algorithm are discussed. In Section 5, the complete taxonomy of rotational CORDIC algorithms is presented. Section 6 presents the low latency nonredundant CORDIC algorithm. Sections 7–9 provide different redundant CORDIC algorithms along with the architectures being proposed in the literature for the rotational CORDIC, followed by the comparison of different methods in Section 10. Finally, conclusions are presented in Section 11.



y
$(x_{\text{out}}, y_{\text{out}})$
M_{in}
$(x_{\text{in}}, y_{\text{in}})$
θ
O
x

Figure 1: Two dimensional vector rotation.

2. Redundant Arithmetic [32, 33]

A nonredundant radix-ρ number system has the set $\{0, 1, \ldots, \rho-1\}$ and all numbers can be uniquely represented. To avoid carry propagation delay in addition, redundant binary number system is employed. The two common redundant number systems employed in CORDIC arithmetic are the signed-digit (SD) [34–37] and the carry-save (CS) [38] number systems. In a SD number system for radix ρ, the numbers are represented with digit set $\{-\beta, -\beta+1, \ldots, -1, 0, +1, \ldots, \alpha\}$, where $\alpha \leq (\rho-1)$ and $(1 \leq \beta \leq (\rho-1))$. For symmetric digit set, $\alpha = \beta$, and each digit s of SD number system is represented as (s^+, s^-) by (p, n) encoding such that $(s^+ - s^- = s)$. In the radix-2 SD number system, numbers are represented with digits $\{-1, 0, 1\}$. In the carry-save number system, numbers are represented with digit set $\{0, 1, 2\}$. It may be observed that, in both SD and CS number systems each number can be represented in multiple ways. The redundancy in SD and CS number representation limits the carry propagation from each stage to its immediate more significant bit position only. In both the SD/CS adders, all sum bits are generated with two full adder delay independent of the word length. Hence, the application of redundant arithmetic can accelerate the additions/subtractions due to carry-free or limited carry-propagation.

3. CORDIC Algorithm

The CORDIC algorithm involves rotation of a vector v on the XY-plane in circular, linear and hyperbolic coordinate systems depending on the function to be evaluated. Trajectories for the vector v_i for successive CORDIC iterations are shown in Figure 2. This is an iterative convergence algorithm that performs a rotation iteratively using a series of specific incremental rotation angles selected so that each iteration is performed by shift and add operation. The norm of a vector in these coordinate systems is defined as $\sqrt{x^2+my^2}$, where $m \in \{1,0,-1\}$ represents a circular, linear or hyperbolic coordinate system respectively. The norm preserving rotation trajectory is a circle defined by $x^2+y^2=1$ in the circular coordinate system. Similarly, the norm preserving rotation trajectory in the hyperbolic and linear coordinate systems is defined by the function $x^2-y^2=1$ and $x=1$, respectively. The CORDIC method can be employed in two different modes, namely, the rotation mode and the vectoring mode. The rotation mode is used to perform the general rotation by a given angle θ. The vectoring mode computes unknown angle θ of a vector by performing a finite number of microrotations.

3.1. Generalized CORDIC Algorithm. The generalized equations of the CORDIC algorithm for an iteration can be written as [7]

$$\begin{aligned} x_{i+1} &= x_i - m\sigma_i y_i \rho^{-S_{m,i}}, \\ y_{i+1} &= \sigma_i x_i \rho^{-S_{m,i}} + y_i, \\ z_{i+1} &= z_i - \sigma_i \alpha_{m,i}, \end{aligned} \tag{2}$$

where σ_i represents either clockwise or counter clockwise direction of rotation, ρ represents the radix of the number system, m steers the choice of circular ($m=1$), linear ($m=0$) or hyperbolic ($m=-1$) coordinate systems, $S_{m,i}$ is the nondecreasing integer shift sequence, and $\alpha_{m,i}$ is the elementary rotation angle. The latter directly depends on $S_{m,i}$ through the relation

$$\alpha_{m,i} = \frac{1}{\sqrt{m}}\tan^{-1}\left(\sqrt{m}\rho^{-S_{m,i}}\right). \tag{3}$$

The shift sequence $S_{m,i}$ depends on the coordinate system and the radix of number system. $S_{m,i}$ affects the convergence of the algorithm and n affects the accuracy of the final result. A detailed discussion on these is presented later. The value of σ_i depends on the radix of the number system and is determined by the following equation assuming that vector is either in the first or in the fourth quadrant:

$$\sigma_i = \begin{cases} \operatorname{sign}(z_i), & \text{for rotation mode,} \\ -\operatorname{sign}(y_i), & \text{for vectoring mode,} \end{cases} \tag{4}$$

where z and y are the steering variables in rotation and vectoring mode respectively. The required microrotations are not perfect and increase the length of the vector. In order to maintain a constant vector length, the obtained results have to be scaled by the scale factor

$$\begin{aligned} K &= \prod_i k_i, \\ k_i &= \sqrt{1+m\sigma_i^2\rho^{-2S_{m,i}}}, \end{aligned} \tag{5}$$

where k_i denotes the elementary scaling factor of the ith iteration, and K is the resultant scaling factor after n iterations. The computation of scale factor and its compensation increases the computational overhead and hardware depending on the number system employed in the CORDIC arithmetic.

With the appropriate initial values of x, y, and z, both rotation and vectoring modes can be used to compute commonly used elementary functions [39] given in Table 1.

3.2. CORDIC Algorithm for Circular Coordinate System. We present in this section the detailed description of 2D plane rotation in circular coordinate system, since this is used in many applications. The CORDIC algorithm calculates trigonometric functions, rotation of a vector and angle of a vector by realizing two dimensional vector rotation in circular coordinate systems. Figure 3 shows the rotation of a vector with length M_{in} by a sequence of microrotations through the elementary angles α_i. Equation (2) represents the iterative rotation by an angle α_i in circular coordinate system for $m=1$ and is given by

$$\begin{aligned} x_{i+1} &= x_i - \sigma_i y_i \rho^{-i}, \\ y_{i+1} &= \sigma_i x_i \rho^{-i} + y_i, \\ z_{i+1} &= z_i - \sigma_i \alpha_i\ . \end{aligned} \tag{6}$$

The values of α_i are chosen such that $\tan(\alpha_i) = \rho^{-i}$ and the multiplication of tangent term is reduced to simple shift operation. It may observed that the norm of vector in $(i+1)$th iteration is extended compared to that in ith rotation, that is $M_{i+1} = M_i\sqrt{1+\tan^2\alpha}$. The increase in magnitude of the vector in every iteration depends on the radix of the number system and number of iterations and is represented by the scale factor K. The direction of iterative rotation is determined using z_i or y_i depending on rotation mode or vectoring mode respectively. The number of microrotations to be performed in both the modes depends on the desired computing accuracy and can be constant for a particular computer of finite word length. The number of microrotations in turn decides the number of elementary angles. The iterative equations of the CORDIC algorithm for radix-2 and radix-4 number systems will be presented in the following sections.

3.2.1. Rotation Mode. In rotation mode, the input angle θ will be decomposed using a finite number of elementary angles [3]

$$\theta = \sigma_0\alpha_0 + \sigma_1\alpha_1 + \cdots + \sigma_{n-1}\alpha_{n-1}, \tag{7}$$

where n indicates the number of microrotations, α_i is the elementary angle for ith iteration and σ_i is the direction of ith microrotation. In rotation mode, z_0 is the angle accumulator initialized with the input rotation angle. The direction of vector in every iteration must be determined to reduce the magnitude of the residual angle in the angle accumulator. Therefore, the direction of rotation in any

Table 1: Realization of some functions using CORDIC Algorithm.

m	Mode	Initialization	Output
1 (Circular)	Rotation	$x_0 = x_{\text{in}}$ $y_0 = y_{\text{in}}$ $z_0 = \theta$	$x_n = K_m \cdot (x_{\text{in}} \cos\theta - y_{\text{in}} \sin\theta)$ $y_n = K_m \cdot (y_{\text{in}} \cos\theta + x_{\text{in}} \sin\theta)$ $z_n = 0$
		$x_0 = 1/K_m$ $y_0 = 0$ $z_0 = \theta$	$x_n = \cos\theta$ $y_n = \sin\theta$ $z_n = 0$
		$x_0 = 1$ $y_0 = a$ $z_0 = \pi/2$	$x_n = \sqrt{1+a^2}$ $y_n = \sin\theta$ $z_n = 0$
1 (Circular)	Vectoring	$x_0 = x_{\text{in}}$ $y_0 = y_{\text{in}}$ $z_0 = 0$	$x_n = K_m \cdot \text{sign}(x_0) \cdot (x_{\text{in}}^2 + y_{\text{in}}^2)^{1/2}$ $y_n = 0$ $z_n = \tan^{-1}(y_{\text{in}}/x_{\text{in}})$
0 (Linear)	Rotation	$x_0 = x_{\text{in}}$ $y_0 = y_{\text{in}}$ $z_0 = z$	$x_n = x_{\text{in}}$ $y_n = y_{\text{in}} + x_{\text{in}} \cdot z$ $z_n = 0$
0 (Linear)	Vectoring	$x_0 = x_{\text{in}}$ $y_0 = y_{\text{in}}$ $z_0 = z$	$x_n = x_{\text{in}}$ $y_n = 0$ $z_n = z + y_{\text{in}}/x_{\text{in}}$
-1 (Hyperbolic)	Rotation	$x_0 = x_{\text{in}}$ $y_0 = y_{\text{in}}$ $z_0 = \theta$	$x_n = K_m \cdot (x_{\text{in}} \cosh\theta + y_{\text{in}} \sinh\theta)$ $y_n = K_m \cdot (y_{\text{in}} \cosh\theta + x_{\text{in}} \sinh\theta)$ $z_n = 0$
		$x_0 = 1/K_m$ $y_0 = 0,\ z_0 = \theta$	$x_n = \cosh\theta$ $z_n = 0,\ y_n = \sinh\theta$
		$x_0 = a$ $y_0 = a,\ z_0 = \theta$	$x_n = ae^{\theta}$ $z_n = 0,\ y_n = ae^{\theta}$
-1 (Hyperbolic)	Vectoring	$x_0 = x_{\text{in}}$ $y_0 = y_{\text{in}}$	$x_n = K_m \cdot \text{sign}(x_0) \cdot (x_{\text{in}}^2 - y_{\text{in}}^2)^{1/2}$ $y_n = 0,\ z_n = \theta + \tanh^{-1}(y_{\text{in}}/x_{\text{in}})$
		$x_0 = a$ $y_0 = 0$	$x_n = \sqrt{a^2 - 1}$ $y_n = 0,\ z_n = \coth^{-1} a$
		$x_0 = a + 1$ $y_0 = a - 1$	$x_n = 2\sqrt{a}$ $y_n = 0,\ z_n = 0.5 \ln(a)$
		$x_0 = a + b$ $y_0 = a - b$	$x_n = 2\sqrt{ab}$ $y_n = 0,\ z_n = 0.5 \ln(a/b)$

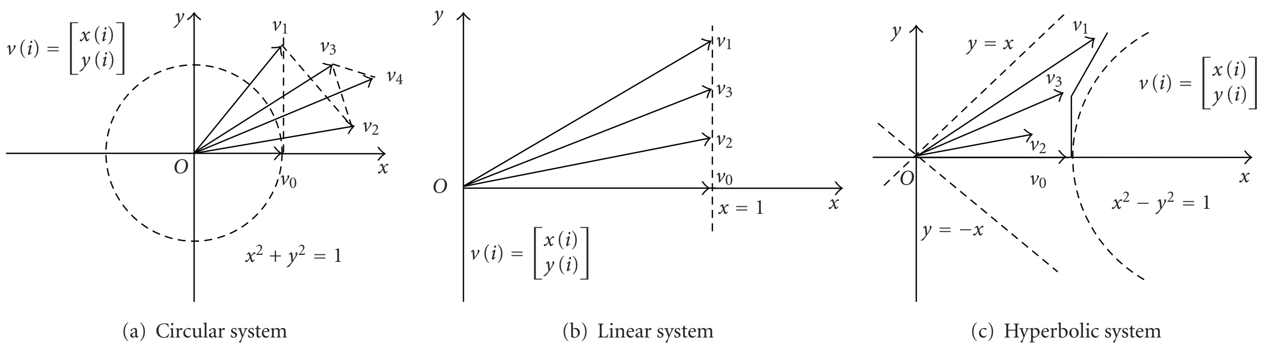


Figure 2: Rotation in various coordinate systems.

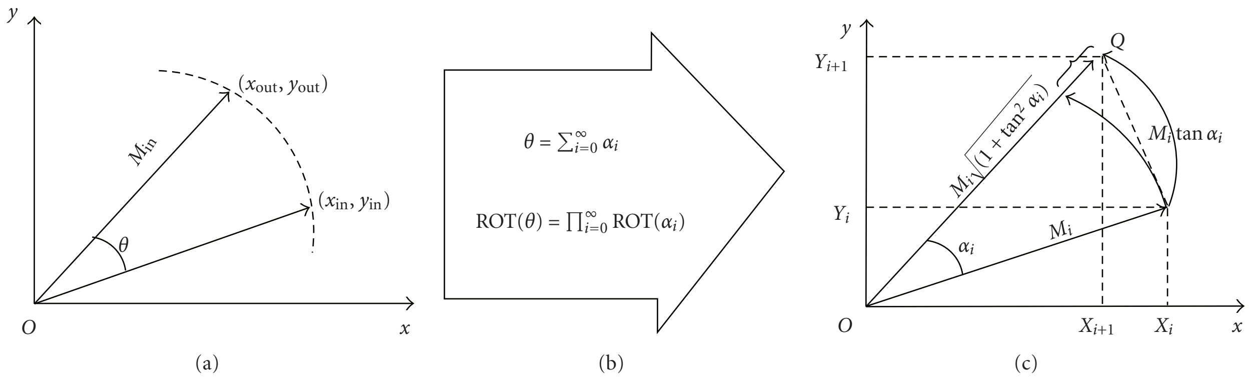

$(x_{\text{out}}, y_{\text{out}})$
$(x_{\text{in}}, y_{\text{in}})$
M_{in}
θ
O x y
(a)
$\theta = \sum_{i=0}^{\infty} \alpha_i$
$\text{ROT}(\theta) = \prod_{i=0}^{\infty} \text{ROT}(\alpha_i)$
(b)
Q
Y_{i+1}
Y_i
$M_i\sqrt{(1+\tan^2\alpha_i)}$
$M_i \tan\alpha_i$
M_i
α_i
X_{i+1} X_i
(c)

Figure 3: CORDIC algorithm based 2D vector rotation.

iteration is determined using the sign of the residual angle obtained in the previous iteration. The coordinates of a vector obtained after n microrotations are

$$\begin{aligned} x_n &= K(x_{\text{in}}\cos\theta - y_{\text{in}}\sin\theta), \\ y_n &= K(x_{\text{in}}\sin\theta + y_{\text{in}}\cos\theta), \\ z_n &\longrightarrow 0. \end{aligned} \tag{8}$$

3.2.2. Vectoring Mode. In vectoring mode, the unknown angle of a vector is determined by performing a finite number of microrotations satisfying the relation [3]

$$-\theta = \sigma_0\alpha_0 + \sigma_1\alpha_1 + \cdots + \sigma_{n-1}\alpha_{n-1}. \tag{9}$$

The vectoring mode rotates the input vector through a predetermined set of n elementary angles so as to reduce the y coordinate of the final vector to zero as closely as possible. Therefore, the direction of rotation in every iteration must be determined based on the sign of residual y coordinate obtained in the previous iteration. The coordinates obtained in vectoring mode after n iterations are given by

$$\begin{aligned} x_n &= K\sqrt{x_{\text{in}}^2 + y_{\text{in}}^2}, \\ y_n &\longrightarrow 0, \\ z_n &= \tan^{-1}\left(\frac{y_{\text{in}}}{x_{\text{in}}}\right). \end{aligned} \tag{10}$$

3.2.3. Radix-2 CORDIC Algorithm. The iteration equations of the radix-2 CORDIC algorithm [7] in rotation mode of circular coordinate system at the $(i+1)$th step are obtained by using $\rho = 2$ in (6) and are given by

$$\begin{aligned} x_{i+1} &= x_i - \sigma_i 2^{-i} y_i, \\ y_{i+1} &= \sigma_i 2^{-i} x_i + y_i, \\ z_{i+1} &= z_i - \sigma_i\alpha_i, \end{aligned} \tag{11}$$

where $\alpha_i = \tan^{-1}(2^{-i})$ and

$$\sigma_i = \begin{cases} -1, & \text{for } z_i < 0, \\ 1, & \text{otherwise.} \end{cases} \tag{12}$$

In order to maintain a constant vector length, the obtained results have to be scaled by the scale factor K given by

$$K = \prod_{i=0}^{n-1} \sqrt{1 + 2^{-2i}}. \tag{13}$$

For radix-2 CORDIC, $K \approx 1.65$. The major drawback of the conventional CORDIC algorithm is its relatively high latency and low throughput due to the sequential nature of the iteration process with carry propagate addition and variable shifting in every iteration. To overcome these drawbacks, pipelined implementations are proposed [40, 41]. However, the carry propagate addition remained a bottleneck for further throughput improvement. Two major methodologies have been employed in the literature to increase the speed of CORDIC implementation. One reduces the delay of each iteration by adopting redundant arithmetic to radix-2 CORDIC [42] to eliminate carry propagate addition. The other technique involves reducing the number of iterations by increasing the radix employed for the implementation of CORDIC algorithm [43].

The redundant radix-2 CORDIC [42] is proposed by employing redundant arithmetic. The direction of rotations σ_i, are selected from the set $\{-1, 0, 1\}$ in contrast to $\{-1, 1\}$ employed in the conventional CORDIC. These σ_i values are computed by evaluating a few most significant digits of z_i, since the determination of sign of a redundant number takes long time. This redundant CORDIC algorithm performs no rotation extension for $\sigma_i = 0$ and affects the value of scaling factor K, thus making it data-dependent. Therefore, K has to be calculated for each microrotation. This calculation and correction increases the computation time and hardware.

3.2.4. Redundant Radix-4 CORDIC Algorithm. As mentioned above, the speed of CORDIC algorithm implementation can be improved by reducing the number of iterations. The iteration equations for the radix-4 CORDIC algorithm in rotation mode derived at the $(i+1)$th step by using $\rho = 4$ in (6) and are given by

$$\begin{aligned} x_{i+1} &= x_i - \sigma_i 4^{-i} y_i, \\ y_{i+1} &= \sigma_i 4^{-i} x_i + y_i, \\ w_{i+1} &= w_i - \tan^{-1}\left(\sigma_i 4^{-i}\right), \end{aligned} \tag{14}$$

where $\sigma_i \in \{-2, -1, 0, 1, 2\}$. The final x and y coordinates are scaled by

$$K = \prod_{i\geq 0} k_i = \prod_{i\geq 0}\left(1 + \sigma_i^2 4^{-2i}\right)^{1/2}. \tag{15}$$

Here, the scale factor K depends on the values of σ_i and hence, has to be computed in every iteration. The range of K is $(1, 2.52)$ for radix-4 CORDIC. In this CORDIC, the direction of rotation is computed based on the estimated value of w_i [43]. The w path involves the computation of estimated w_i and evaluation of selection function to determine σ_i resulting in increase of the iteration delay compared to that of radix-2. However, the number of iterations required for radix-2 CORDIC can be halved by employing the radix-4 CORDIC algorithm.

The Scale factor computation and compensation, CORDIC algorithm convergence and accuracy aspects are presented in following sections.

3.2.5. Scale Factor Computation. The CORDIC rotation steps change the length of the vector in every iteration resulting in the distortion of the norm of the vector as shown in Figure 3 and is given by (5). In nonredundant radix-2 CORDIC, K is constant since $\sigma = \pm 1$. However, K is no longer constant for nonredundant radix higher than 2, and redundant number system. For radix-2, the scale factor needs to be computed for $n/2$ iterations as $k_i = \sqrt{1 + 2^{-2i}}$ becomes unity for $i > n/2 + 1$. In redundant radix-4 CORDIC [43], scale factor (15) is not constant. In addition, it is sufficient to compute K for $n/4$ iterations as $k_i = \sqrt{1 + 4^{-2i}}$ becomes unity thereafter.

3.2.6. Scale Factor Compensation. The scale factor compensation technique involves scaling of the final coordinates (x_n, y_n) with $1/K$. The most direct method for scaling operation is the multiplication of (x_n, y_n) by $1/K$ using the CORDIC module in linear mode [7]. This can realized using the CORDIC module in linear mode [7]. However, this method requires n shift and add operations which are comparable to the computational effort of the CORDIC algorithm itself. Since K^{-1} is constant for radix-2, the computational overhead can be reduced by using CSD recoded multiplier. On an average, the number of nonzero digits can be reduced to $n/3$ using CSD representation [32] and hence, the effort for multiplication using CSD recoded multiplier is approximately one third that required using conventional multiplier. Further, scaling can also be implemented using a Wallace tree by fully parallelizing multiplication and is preferred for applications aiming for low latency at the expense of more silicon area [44].

Scaling may be done by extending the sequence of CORDIC iterations [9, 16, 17] to avoid additional hardware required in the direct method. A comparison of several scale factor compensation techniques proposed in the literature along with two additional methods, additive and multiplicative decomposition approaches, for radix-2 CORDIC is presented in [44]. It is observed from the presented results that additive technique offers a low latency solution and multiplicative technique offers an area economical solution for applications of CORDIC employing array and pipelined architectures. An algorithm is proposed [45] to performs scale factor compensation in parallel with the CORDIC rotation using nonredundant and redundant arithmetic, thereby, eliminating the final multiplication [3] or additional scaling iterations [9, 16, 17].

3.2.7. Convergence. The CORDIC algorithm involves the rotation of a vector to reduce the z or y coordinate of the final vector as closely as possible to zero for rotation or vectoring mode respectively. The maximum value of rotation angle by which the vector can be rotated depends on the shift sequence [7]. The expected results of the CORDIC algorithm can be obtained if the z or y coordinate is driven sufficiently close to zero. In addition, it can be guaranteed to drive z or y to zero, if the initial values of a vector $(x_{\text{in}}, y_{\text{in}}, z_{\text{in}})$ or $(x_{\text{in}}, y_{\text{in}})$ lies within the permissible range. These ranges define the domain of convergence of the CORDIC algorithm.

For n-bit precision, the given rotation angle can be decomposed as

$$\theta = \sum_{i=0}^{n-1} \sigma_i \alpha_i + \varphi, \tag{16}$$

where φ is an angle approximation error such that $|\varphi| < \alpha_{n-1}$ and is negligible in practical computation [7]. This angle approximation error in rotation and vectoring mode can be computed as

$$\varphi(\text{rotation}) = \tan^{-1}(z_n), \qquad \varphi(\text{vectoring}) = \tan^{-1}\left(\frac{y_n}{x_n}\right). \tag{17}$$

The magnitude of elementary angle for the given shift sequence may be predetermined using

$$\alpha_i = \tan^{-1}\rho^{-S_{m,i}}, \tag{18}$$

where ρ is the radix of the number system. The direction of rotation σ_i must be selected to drive z or y towards zero for rotation or vectoring respectively. The range of σ_i depends on the radix and digit set being used for the number system. Since the number of iterations and elementary angles to be traversed by the vector during these iterations are predetermined, the range of θ for which CORDIC algorithm can be used, called domain of convergence, is given by [7]

$$|\theta| = \sum_{i=0}^{n-1}\alpha_i + \alpha_{n-1}. \tag{19}$$

The convergence range of CORDIC algorithm can be defined for rotation mode as

$$z_{\text{in}} \le \sum_{i=0}^{n-1}\alpha_i + \alpha_{n-1} \tag{20}$$

and for vectoring mode as

$$\tan^{-1}\left(\frac{y_{\text{in}}}{x_{\text{in}}}\right) \le \sum_{i=0}^{n-1}\alpha_i + \alpha_{n-1}. \tag{21}$$

The expected final results cannot be obtained, if the given initial values x_{in}, y_{in} and z_{in} do not satisfy these convergence values. The range of convergence of the CORDIC algorithm can be extended from $\pm\pi/2$ to $\pm\pi$ using preprocessing techniques [7, 27, 46].

3.3. Accuracy. The accuracy of the CORDIC algorithm is affected by two primary sources of error, namely, angle approximation and rounding error. The error bounds for these two sources of error are derived by performing the detailed numerical analysis of the CORDIC algorithm [47]. The approximation error and the rounding error derived are combined to yield the overall quantization error in the CORDIC computation. The overall quantization error can be assured to be within the range by considering an additional $\log_2 n$ guard bits in the implementation of the CORDIC algorithm [7].

3.3.1. Angle Approximation Error. Theoretically, the rotation angle θ is decomposed into infinite number of elementary angles as shown in Figure 3. For practical implementation, a finite number of microrotations n are considered. Hence, the input rotation angle θ can only be approximated resulting in an angle approximation error φ

$$|\varphi| < \alpha_{n-1}, \tag{22}$$

where α_{n-1} is the residual angle after n microrotations. Hence, the accuracy of the output of the nth iteration is principally limited by the magnitude of the last rotation angle.

3.3.2. Rounding Error. The second type of error called rounding error is due to the truncation of CORDIC internal variables by the finite length of storage elements. In addition scale factor compensation also contributes to this error. In a binary code, the truncation of intermediate results after every iteration introduces maximum rounding error of $\log_2 n$ bits. To achieve a final accuracy of 1 bit in n bits, an additional $\log_2 n$ guard bits must be considered in implementation of this algorithm [7].

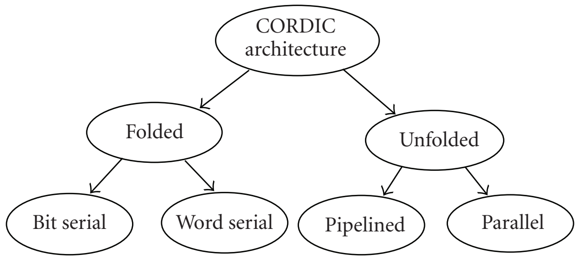


FIGURE 4: Taxonomy of CORDIC architectures.

4. CORDIC Architectures

In this section, a few architectures for mapping the CORDIC algorithm into hardware are presented. In general, the architectures can be broadly classified as folded and unfolded as shown in Figure 4, based upon the realization of the three iterative equations (6). Folded architectures are obtained by duplicating each of the difference equations of the CORDIC algorithm into hardware and time multiplexing all the iterations into a single functional unit. Folding provides a means for trading area for time in signal processing architectures. The folded architectures can be categorized into bit-serial and word-serial architectures depending on whether the functional unit implements the logic for one bit or one word of each iteration of the CORDIC algorithm.

The CORDIC algorithm has traditionally been implemented using bit serial architecture with all iterations executed in the same hardware [3]. This slows down the computational device and hence, is not suitable for high speed implementation. The word serial architecture [7, 48] is an iterative CORDIC architecture obtained by realizing the iteration equations (6). In this architecture, the shifters are modified in each iteration to cause the desired shift for the iteration. The appropriate elementary angles, α_i are accessed from a lookup table. The most dominating speed factors during the iterations of word serial architecture are carry/borrow propagate addition/subtraction and variable shifting operations, rendering the conventional CORDIC [7] implementation slow for high speed applications. These drawbacks were overcome by unfolding the iteration process [41, 48], so that each of the processing elements always perform the same iteration as shown in Figure 5. The main advantage of the unfolded pipelined architecture compared to folded architecture is high throughput due to the hard-wired shifts rather than time and area consuming barrel shifters and elimination of ROM. It may be noted that the pipelined architecture offers throughput improvement by a factor of n for n-bit precision at the expense of increasing the hardware by a factor less than n.

5. CORDIC Taxonomy

The implementation of CORDIC algorithm has evolved over the years to suit varying requirements of applications from conventional nonredundant to redundant nature. The unfolded implementation with redundant arithmetic initiated the efforts to address high latency in conventional CORDIC. Subsequently, several modifications have been proposed for redundant CORDIC algorithm to achieve reduction in iteration delay, latency, area and power. The evolution of the unfolded rotational CORDIC algorithms is shown in Figure 6. As this taxonomy is fairly rich, the remainder of the review presents taxonomy in top-down approach.

CORDIC is broadly classified as nonredundant CORDIC and redundant CORDIC based on the number system being employed. The major drawback of the conventional CORDIC algorithm [3, 7] was low throughput and high latency due to the carry propagate adder used for the implementation of iterative equations. This contradicted the simplicity and novelty of the CORDIC algorithm attracting the attention of several researchers to device methods to increase the speed of execution. The obvious solution is to reduce the time for each iteration or the number of iterations or both. The redundant arithmetic has been employed to reduce the time for each iteration of the conventional CORDIC. We have analyzed and presented in the following Sections, features of different pipelined and nonpipelined unfolded implementations of the rotational CORDIC.

6. Low Latency Nonredundant Radix-2 CORDIC [49]

A significant improvement for the conventional rotational CORDIC algorithm in circular coordinate system is proposed [50], employing linear approximation to the rotation when the remaining angle is small. This remaining angle is chosen such that a first order Taylor series approximation of $\sin\theta_r$ and $\cos\theta_r$, calling θ_r the remaining angle, may be employed as $\sin\theta_r \approx \theta_r$ and $\cos\theta_r \approx 1$. The architecture for the implementation of this algorithm using nonredundant arithmetic is presented in [49]. The iteration equations of this algorithm for the first $n/2 + 1$ microrotations are same as those for the conventional CORDIC algorithm (11). The σ_i values for the first $n/3$ iterations are determined iteratively using the sign of angle accumulator z_i. The rotation directions from iteration $n/3 + 1$ onwards can be generated in parallel, since the conventional circular arc tangent radix values approach the radix-2 coefficients progressively for increasing values of CORDIC iteration index as evident from the expression

$$\lim_{k\to\infty} \frac{\tan\left(2^{-k}\right)}{2^{-k}} = 1. \tag{23}$$

For the range of iterations $(n/3 + 1) \le i \le (n/2 + 1)$, all σ_i values are determined from the recoded representation of remaining angle $z_{(n/3+1)}$. These σ_i values are used to obtain $z_{(n/2+1)}$ from $z_{(n/3+1)}$. For $i > (n/2 + 1)$, the CORDIC microrotations are replaced by a single rotation using the remaining angle $z_{(n/2+1)}$. Thus, (11) is modified as

$$\begin{aligned} x_f &= x_{(n/2+2)} = k_{(n/2+1)}\left(x_{(n/2+1)} - \theta_r y_{(n/2+1)}\right), \\ y_f &= y_{(n/2+2)} = k_{(n/2+1)}\left(\theta_r x_{(n/2+1)} + y_{(n/2+1)}\right), \end{aligned} \tag{24}$$

where $\theta_r = z_{(n/2+1)}$, $k_{(n/2+1)}$ is the scale factor in the $(n/2+1)$th iteration and (x_f, y_f) are the scaled final coordinates.

Scale Factor. The low latency nonredundant radix-2 CORDIC algorithm achieves constant scale factor since $\sigma_i \in \{-1, 1\}$ and performs the scale factor compensation concurrently with the computation of x and y coordinates, using two multipliers in parallel [49]. This is in contrast to two series multiplications required in the algorithm [50].

7. Constant Scale Factor Redundant Radix-2 CORDIC

Redundant radix-2 CORDIC methods can be classified as variable and constant scale factor methods based on the dependence of scale factor on the input angle. In redundant radix-2 CORDIC [42], $\sigma_i \in \{-1, 0, 1\}$ and hence scale factor K is data-dependent. Therefore, K has to be calculated for each microrotation. This calculation and correction increases the computation time and hardware. Several redundant CORDIC algorithms with constant scale factor are available in the literature [51–53] to address data dependency of the scale factor as shown in Figure 7. In these methods, the iterative rotations of a point around the origin on the XY-plane are considered (see Figure 1). The direction of each rotation depends on the sign of steering variable z_i, which represents the remaining angle of rotation. Since the computation of the sign of redundant number requires more time, estimated value of z_i ($\hat{z}_i$) is used to determine the direction of rotation. The estimated value is computed based on the value of the three most significant digits of z_i. Constant scale factor is achieved by restricting σ_i to the set $\{-1, 1\}$, thus facilitating a faster implementation. The constant scale factor methods can be classified based on the arithmetic employed as redundant radix-2 CORDIC with signed digit arithmetic and carry save arithmetic (see Figure 7).

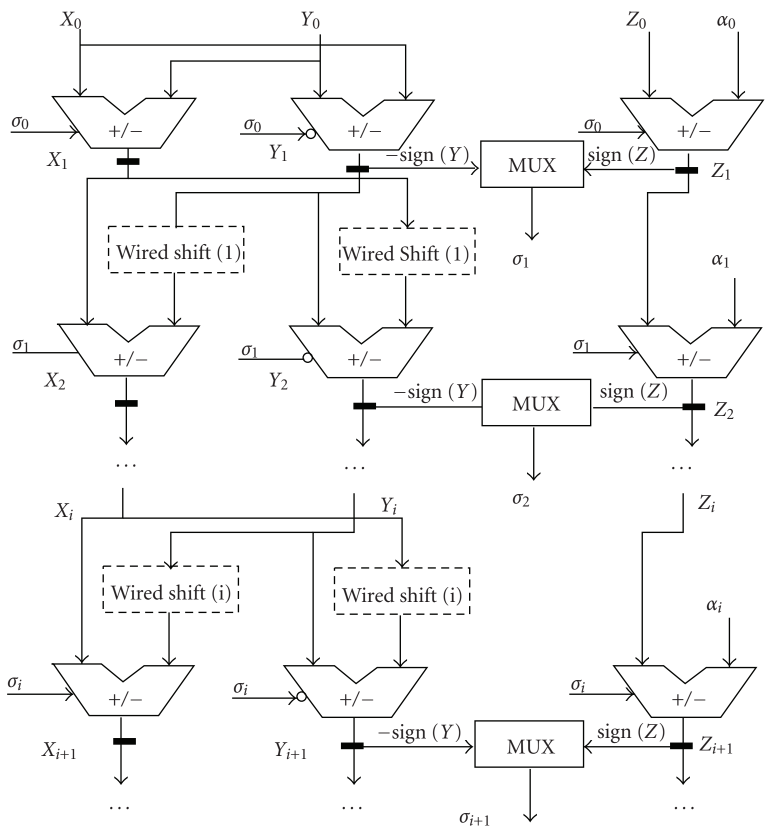


FIGURE 5: Unfolded pipelined CORDIC architecture.

Scale Factor. The scale factor need not be computed for the implementation of all the constant scale factor techniques discussed in this section. In these methods, no specific scale factor compensation technique is considered. It may be noted that a specific compensation technique can be considered depending on the application.

7.1. Constant Scale Factor Redundant CORDIC Using SD Arithmetic. The redundant radix-2 CORDIC using SD arithmetic can be further classified based on the technique employed to achieve constant scale factor (see Figure 7). These methods are implemented using the basic CORDIC iteration recurrences (11) with necessary transformations.

7.1.1. Double Rotation Method [51]. The double rotation method performs two rotation-extensions for each elementary angle during the first $n/2$ iterations for n bit precision to achieve constant scale factor independent of the operand. One rotation extension is performed for every elementary angle for iterations greater than $n/2$. A negative rotation is performed by two negative subrotations, and a positive rotation by two positive subrotations. A nonrotation is performed by one negative and one positive subrotation. Hence, 50% additional iterations are required compared to the redundant CORDIC [42].

7.1.2. Correcting Rotation Method [51]. This is another method proposed to achieve constant scale factor for the computation of sine and cosine functions. This method avoids rotation corresponding to $\sigma_i = 0$ and performs one rotation extension in every iteration depending on the $\hat{z}_i$. Further, extra rotation extensions are performed at fixed intervals for correcting the error introduced by avoiding $\sigma_i = 0$ and to assure convergence. If b fractional bits are used to estimate z_i, the interval between correcting iterations should be less than or equal to $(b - 2)$ [54]. This method also requires 50% additional iterations, if three or four most significant digits are used for sign estimation. The increase in latency of rotational CORDIC due to these double rotation and correcting iteration methods is reduced using branching algorithm [52].

7.1.3. Branching Method [52]. This method implements CORDIC algorithm using SD arithmetic, restricting the direction of rotations σ_i to ± 1, without the need for extra rotations. This requires two modules in parallel to perform two conventional CORDIC iterations, such that, the correct result is retained at the end of each iteration. Two modules perform the rotation in the same direction if the sign of corresponding z_i can be determined. Otherwise, branching is performed by making one CORDIC module (z^+) perform rotation with $\sigma_i = +1$ and another module (z^-) perform

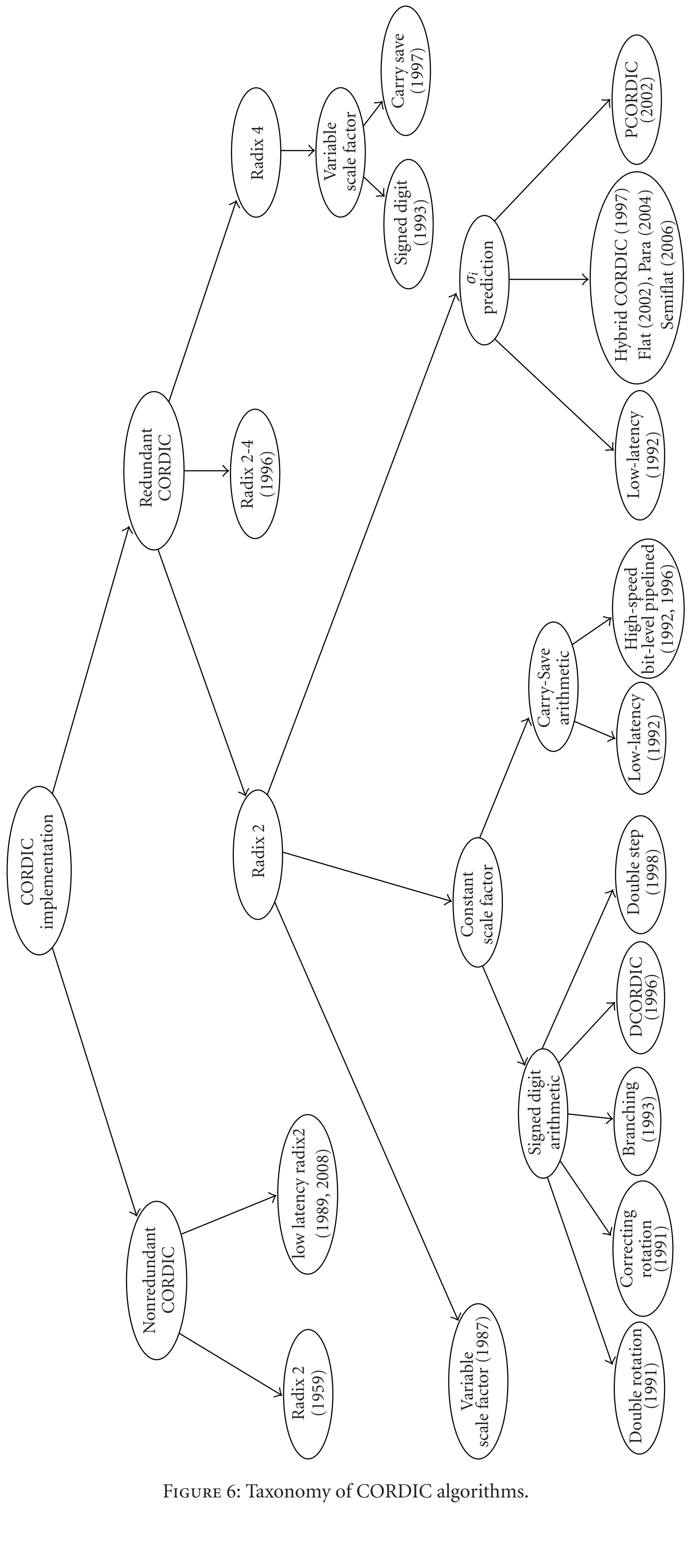


Figure 6: Taxonomy of CORDIC algorithms.

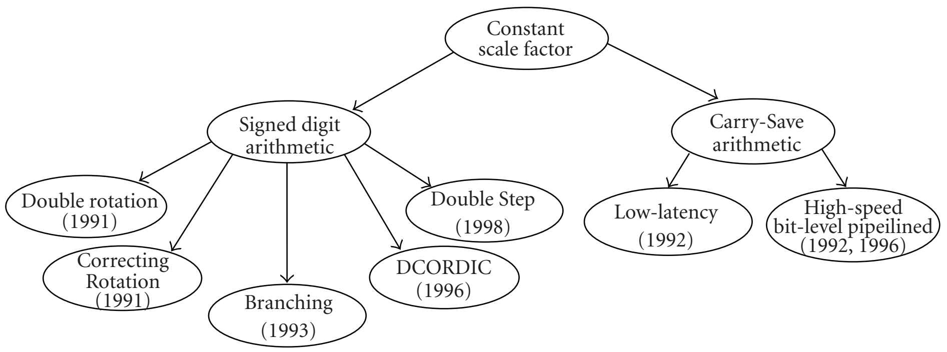


Figure 7: Taxonomy of constant scale factor redundant radix-2 CORDIC methods.

rotation with $\sigma_i = -1$. The direction of rotation in the next subsequent rotation is decided by the sign of that z_i module whose value is small. In every iteration i, angle accumulator (z^+ or z^-) computes the remaining angle (z_i^+ or z_i^-) to determine the direction of rotation for the next iteration. The direction of rotation is determined by examining window of three digits of z_i^+or z_i^-.

The disadvantage of branching method is the necessity of performing two conventional CORDIC iterations in parallel which requires almost two fold effort in terms of implementation complexity. In addition, one of the modules will not be utilized when branching does not take place. However, this method offers faster implementation than double and correcting rotation methods [51], since, it does not require additional iterations to achieve constant scale factor.

7.1.4. Double Step Branching Method [53]. The performance of branching algorithm is enhanced by the double step branching method to improve utilization of hardware. This method involves determining two distinct σ_i values in each step with some additional hardware compared to the branching method, where the two modules do different computations only when branching takes place. Double step branching method determines the two direction of rotations by examining the six most significant digits to do a double step. These six digits are divided into two subgroups of three digits each, and each subgroup is handled in parallel, to generate the required σ_i using zeroing modules (z path). Although double stepping method introduces a small hardware overhead compared to the branching method, it is better than the latter since it increases the utilization of x/y rotator modules.

7.2. Constant Scale Factor Redundant CORDIC Using CS Arithmetic. It is worth discussing here one more classification related to constant scale factor redundant radix-2 CORDIC (see Figure 7). The implementation of redundant CORDIC with constant scale factor using signed arithmetic results in an increase in the chip area [51–53] and latency [51] by at least 50% compared to redundant radix-2 CORDIC [42]. Low latency CORDIC algorithm [55] and differential CORDIC algorithm [56, 57] with constant scale factor using CS arithmetic have been proposed to reduce this overhead, the details of which are discussed below.

7.2.1. Low Latency Redundant CORDIC [55]. This algorithm is proposed to reduce the latency of redundant CORDIC [51] by subdividing the n iterations into different groups and using different techniques for each of these groups. For all the iterations, if $\sigma_i = \pm 1$, conventional iteration equations (11) are used. This method avoids $\sigma_i = 0$ for iterations between $0 \le i \le (n-3)/4$ and employs correcting rotation method [51]. For iterations $(n-3)/4 < i \le (n+1)/2$, $\sigma_i = 0$ is considered as a valid choice. Since for this group of iterations $k_i = \sqrt{1+2^{-2i}} = 1 + 2^{-2i-1}$ holds within n-bit precision, vector is not rotated for $\sigma_i = 0$. However, the length of the vector is increased by the scale factor for that iteration, as the final coordinates are scaled assuming constant scale factor. For the iterations $i > (n+1)/2$, no correcting factor is required as the scale factor becomes unity.

7.2.2. DCORDIC [56]. In the sign estimation methods [51–53], half of the computational effort in the $x/y/z$ data paths of rotational CORDIC is required to allow for the correction of possible errors, as the sign estimation is not entirely perfect. This problem is reduced by high speed bit-level pipelining technique with CS arithmetic proposed in [57]. This algorithm involves the transformation of the conventional CORDIC iteration equations (11) into partially fixed iteration equations, given by

$$\begin{aligned} |\widehat{z_{i+1}}| &= \left| |\widehat{z}_i| - \alpha_i \right|, \\ x_{i+1} &= x_i - \operatorname{sign}(z_i) 2^{-i} y_i, \\ y_{i+1} &= \operatorname{sign}(z_i) 2^{-i} x_i + y_i. \end{aligned} \tag{25}$$

It is clear from these expressions that the computation of x and y requires the actual sign of z_i, while the angle accumulator requires only the absolute value of $\widehat{z}_i$. The actual sign of z_i (σ_i) can be determined by taking into account the initial sign of z_0 and providing information about sign changes during the absolute value computation of $\widehat{z}_i$. Similarly, all σ_i values are computed recursively. Later this technique is implemented with SD arithmetic and proposed as Differential CORDIC (DCORDIC) algorithm [56]. Since

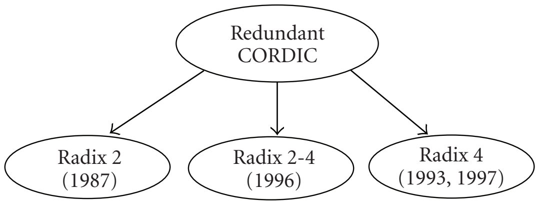


FIGURE 8: Classification of CORDIC algorithms based on the radix.

the sign calculation of steering variable ($\hat{z}_i$) during absolute value computation takes long time, most significant digit first absolute value technique is employed. This technique replaces the word level sign dependence by a bit level dependence, reducing the overall computation time. The bit level pipelined architecture is proposed to implement these transformed iteration sequences, thus allowing high operational speed.

8. Higher Radix Redundant CORDIC

As mentioned earlier, throughput and latency are important performance attributes in CORDIC based systems. The various radix-2 CORDIC algorithms presented so far may be used to reduce the iteration delay, thereby improving the throughput, with constant scale factor. Higher radix CORDIC algorithms using SD arithmetic [54, 58] and CS arithmetic [43, 59] are proposed to address latency reduction. This is possible, since higher radix representation reduces the number of iterations. The classification of redundant CORDIC algorithms proposed in the literature based on the radix of the number system is shown in Figure 8. The application of radix-4 rotations in the CORDIC algorithm was initially proposed in [54] to accelerate the radix-2 algorithm.

Scale factor need not be computed for the constant scale factor algorithms to be discussed in this section. Since no specific scale factor compensation technique is considered for these methods, a compensation technique can be considered depending on the application.

8.1. Pipelined Radix-4 CORDIC [58]. The generalized CORDIC algorithm for any radix in three coordinate systems and implementation of the same in rotation mode of circular coordinate system using radix-4 pipelined CORDIC processor is presented in [58]. This algorithm performs two successive radix-2 microrotations with the same microrotation angle using the iteration equations

$$\begin{aligned} x_{i+1} &= x_i - (\sigma_{i,1} + \sigma_{i,2})4^{-i}y_i - \sigma_{i,1}\sigma_{i,2}4^{-2i}x_i, \\ y_{i+1} &= (\sigma_{i,1} + \sigma_{i,2})4^{-i}x_i + y_i - \sigma_{i,1}\sigma_{i,2}4^{-2i}y_i, \\ z_{i+1} &= z_i - (\sigma_{i,1} + \sigma_{i,2})\alpha_i, \end{aligned} \tag{26}$$

where $\sigma_{i,1}$ and $\sigma_{i,2}$ are two redundant radix-2 coefficients to decompose radix-4 coefficient $\sigma_i \in \{-2, -1, 0, +1, +2\}$ satisfying the relation ($\sigma_i = \sigma_{i,1} + \sigma_{i,2}$). The value of α_i is selected as $\alpha_0 = 2^{-1}$ and $\alpha_i = 4^{-i}$ for $1 \le i \le n-1$. The selection function for σ_i is determined using the five most significant digits of z-coordinate, ensuring the convergence of this algorithm. This algorithm is designed using SD arithmetic and requires two adders/subtractors for each stage of x/y data path in contrast to one adder/subtractor required in radix-2 CORDIC [42], for $i < n/4$. However, the number of additions required are reduced during the last $n/4$ stages.

Scale Factor Computation. The scale factor K in radix-4 CORDIC algorithm is variable, since σ_i takes values from the digit set $\{-2, -1, 0, +1, +2\}$. K is computed in each iteration using the combinational circuit by realizing the expression

$$K = \prod_{i=0}^{n/2-1} k_i = \prod_{i=0}^{n/2-1} \left(1 + |\sigma_{i,1}| 4^{-2i}\right)^{1/2} \left(1 + |\sigma_{i,2}| 4^{-2i}\right)^{1/2}. \tag{27}$$

8.2. Redundant Radix 2-4 CORDIC [59]. The number of rotations in a redundant radix-2 CORDIC rotation unit is reduced by about 25% by expressing the direction of rotations in radix-2 and radix-4 [54]. This algorithm employs different modified CORDIC algorithms using CS arithmetic for different subsets of iterations. For the iterations $1 \le i < n/4$, nonredundant radix-2 CORDIC algorithm with $\sigma_i = \{-1, 1\}$ is employed. For $n/4 \le i \le (n/2 + 1)$, correcting iteration method [51] is employed. For $i > (n/2 + 1)$, redundant radix-4 CORDIC algorithm is employed, thus, halving the number of iterations. An unified architecture is proposed for the implementation of this algorithm to operate in rotation/vectoring mode of circular and hyperbolic coordinate systems.

Scale Factor Computation. This algorithm achieves constant scale factor, since the rotation corresponding to $\sigma = 0$ is avoided for $i \le n/2 + 1$. For$i > n/2 + 1$ scale factor need not be computed as $k_i = \sqrt{1 + 4^{-2i}} \sim 1$.

8.3. Radix-4 CORDIC [43]. A redundant radix-4 CORDIC algorithm is proposed using CS arithmetic, to reduce the latency compared to redundant radix-2 CORDIC [42]. This algorithm (14) computes σ_i values using two different techniques. For the microrotations in the range $0 \le i < (n/6)$, σ_i is determined sequentially using angle accumulator. For the microrotations in the range $i \ge (n/6)$, the σ_i values are predicted from the the remaining angle after the first $n/6$ [60]. Thus, the complexity of the w path is $n/6$, compared to n in the other architectures [42–53] presented in the previous sections. For the range $0 \le i < (n/6)$, microrotations are pipelined in two stages to increase the throughput. A 32-bit pipelined architecture is proposed for the implementation of the radix-4 CORDIC algorithm using CS arithmetic.

Scale Factor Computation. The possible scale factors are precomputed and stored in a ROM. The number of possible scale factors for $\sigma_i^2 \in \{0, 1, 4\}$ is $3^{n/4+1}$. The size of ROM and access time increases with n. Hence, the scale factors for some iterations are stored in ROM and these values are used to

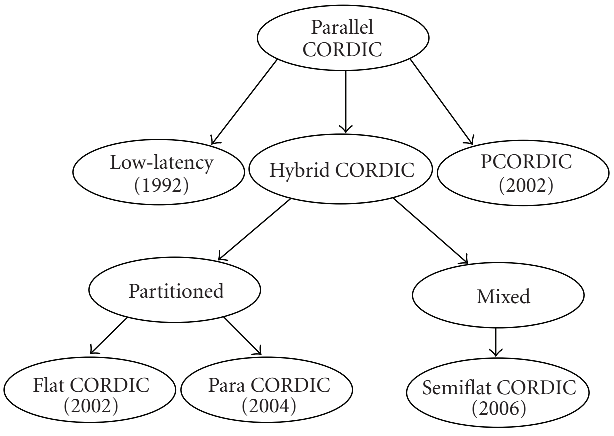


Figure 9: Taxonomy of direction prediction based CORDIC algorithms.

compute the scale factor for remaining iterations with the combinational logic. This is designed by realizing the first few terms of Taylor series expansion of scale factor. For this redundant radix-4 implementation, the number of iterations are reduced at the expense of adding hardware for computing the scale factor.

9. Parallel CORDIC Algorithms

The CORDIC algorithms discussed so far have represented θ using a set of elementary angles α_i called arc tangent radix set [3]

$$\theta = \sigma_0\alpha_0 + \sigma_1\alpha_1 + \cdots + \sigma_{n-1}\alpha_{n-1}, \quad (28)$$

where $\alpha_i = \tan^{-1}(2^{-i})$ and $\sigma_i \in \{-1, 1\}$, satisfying the convergence theorem [7]

$$\alpha_i - \sum_{j=i+1}^{n-1} \alpha_j < \alpha_{n-1} \quad (29)$$

in contrast to the representation using a normal radix

$$\theta = \sigma_0 2^0 + \sigma_1 2^{-1} + \cdots + \sigma_{n-1} 2^{-n+1}. \quad (30)$$

The direction of rotation σ_i for the ith iteration is determined after computing the $(i-1)$ iterations sequentially. It is evident from this sequential dependence of the radix system that the speed of CORDIC algorithm can be improved by avoiding the sequential behavior in the computation of σ_i values or x/y coordinates. The various redundant CORDIC algorithms proposed in the literature employing either one or both these techniques are shown in Figure 9 and are discussed in the following sections.

9.1. Low Latency Radix-2 CORDIC [55]. The low latency parallel radix-2 CORDIC architecture presented for the rotation mode [55] predicts σ_i's by eliminating sequential dependency of the z path. In order to minimize the prediction error, directions are predicted for a group of iterations at a time rather than for all iterations together. This architecture does not allow rotation for index $i = 0$. Hence, the convergence range of this architecture is less than $(-\pi/2, +\pi/2)$. On the other hand, the requirement of redundant to binary conversions of intermediate results in the z path restricts the pipelined implementation of this architecture. In order to reduce the latency of this parallelizing scheme further, termination algorithm and booth encoding method have been proposed.

9.2. P-CORDIC [61]. The sequential procedure in the computation of direction of rotations of the CORDIC algorithm is eliminated by the P-CORDIC algorithm, while maintaining a constant scale factor. This algorithm precomputes the direction of microrotations before the actual CORDIC rotation starts iteratively in the x/y path. This is obtained by deriving a relation between the constructed binary representation of direction of rotations d, and rotation angle θ [40, 62] given by

$$\sigma = 0.5\theta + 0.5c_1 + \text{sign}(\theta)\epsilon_0 + \delta, \quad (31)$$

where $c_1 = 2 - \sum_{i=0}^{\infty}(2^{-i} - \tan^{-1}(2^{-i}))$, $\delta = \sum_{i=1}^{n/3}(\sigma_i\epsilon_i)$, $\epsilon_0 = 1 - \tan^{-1}(1)$, and $\epsilon_i = 2^{-i} - \tan^{-1}(2^{-i})$. Here, δ is computed using the partial offset ϵ_i and the corresponding direction bit σ_i for the first $n/3$ iterations, since the value of ϵ_i decreases by a factor of 8 beyond $n/3$ iterations. The direction of rotations for any input angle θ in binary form are obtained by realizing this expression taking a variable offset δ from ROM. The unfolded architecture proposed for the implementation of this algorithm eliminates the z path and reduces the area of the implementation. This architecture achieves latency and hardware reduction over the radix-2 unfolded parallel architecture [55].

Scale Factor. The scale factor in the implementation of P-CORDIC algorithm remains constant, as $\sigma_i \in \{-1, 1\}$ being generated for the implementation of x/y path. The scale factor compensation is implemented using constant factor multiplication technique as discussed in Section 3.2.6.

9.3. Hybrid CORDIC Algorithm. For n-bit fixed point CORDIC processor in circular coordinate system, nearly $n/3$ iterations must be computed sequentially. This is true for both generation of direction and rotation without affecting accuracy [60]. The subsequent rotation directions for the last $2n/3$ iterations can be generated in parallel since the conventional circular ATR values approach the radix-2 coefficients progressively with increasing iteration index, that is,

$$\lim_{k \to +\infty} \frac{\tan 2^{-k}}{2^{-k}} = 1. \quad (32)$$

This behavior is exploited by introducing the hybrid CORDIC algorithms to speed up the conventional CORDIC rotator. This algorithm involves partitioning θ into θ_H and θ_L. The rotation by θ_H are performed as in the conventional CORDIC algorithm and the iterations related to θ_L can be simplified as in linear coordinate system. This algorithm led to the development of several parallel CORDIC algorithms [63–65]. These can be categorized broadly as mixed-hybrid CORDIC and partitioned-hybrid CORDIC algorithms. In mixed-hybrid CORDIC algorithms [65], the input angle θ and initial coordinates $(x_{\text{in}}, y_{\text{in}})$ are used to compute the rotations for the first $n/3$ iterations as in the conventional CORDIC. The remaining angle after these first $n/3$ iteration is used for computing directions for the last $2n/3$ iterations. The implementation is designed to keep the fast timing characteristics of redundant arithmetic in the x/y path of the CORDIC processing. In the partitioned-hybrid CORDIC [63, 64], the first $n/3$ direction of rotations are generated using the first $n/3$ bits of θ and last $2n/3$ direction of rotations are predicted using the $2n/3$ least significant bits of θ.

9.3.1. Flat CORDIC [63]. The flat CORDIC algorithm is proposed to eliminate iterative nature in the x/y path for reducing the total computation time. This algorithm transforms x/y recurrences (11) of the conventional CORDIC into a parallelized version by successive substitution to express the final vectors in terms of the initial vectors, resulting in a single equation for n-bit precision. The expressions for final coordinates of 16-bit sine/cosine generator are

$$\begin{aligned} x_{16} = \big[1 - \{&(\sigma_1\sigma_2 2^{-1}2^{-2} - \cdots - \sigma_1\sigma_{23}2^{-1}2^{-23} \\ &- \sigma_2\sigma_3 2^{-2}2^{-3} - \cdots - \sigma_9\sigma_{10}2^{-9}2^{-10}) \\ &+ (\sigma_1\sigma_2\sigma_3\sigma_4 2^{-1}2^{-2}2^{-3}2^{-4} + \cdots \\ &+ \sigma_2\sigma_3\sigma_4\sigma_5 2^{-2}2^{-3}2^{-4}2^{-5} + \cdots \\ &+ \sigma_3\sigma_4\sigma_6\sigma_7 2^{-3}2^{-4}2^{-6}2^{-7}) + E_{C-X}\}\big], \\ y_{16} = \big[&\sigma_1 2^{-1} + \sigma_2 2^{-2} + \cdots + \sigma_{16}2^{-16} \\ &- (\sigma_1\sigma_2\sigma_3 2^{-1}2^{-2}2^{-3} - \cdots - \sigma_5\sigma_7\sigma_8 2^{-5}2^{-7}2^{-8}) \\ &+ (\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5 2^{-1}2^{-2}2^{-3}2^{-4}2^{-5} + \cdots \\ &+ \sigma_2\sigma_3\sigma_4\sigma_5\sigma_6 2^{-2}2^{-3}2^{-4}2^{-5}2^{-6}) + E_{C-Y}\big], \end{aligned} \tag{33}$$

where E_{C-X} and E_{C-Y} are the error compensation factors in x_{16} and y_{16}, respectively. x_{in} and y_{in} are initialized with $1/K$ and 0 respectively. The 16 sign digits $(\sigma_1, \sigma_2, \ldots, \sigma_{15}, \sigma_{16})$ for 16-bit precision represents the polarity of 16 microrotations required to achieve the target angle. These equations demonstrate the complete parallelization of the conventional CORDIC algorithm. This technique precomputes σ_i which takes values from the set $\{-1, 1\}$ to achieve constant scale factor. The σ_i's for the first $n/3$ iterations are precomputed employing a technique, called Split Decomposition Algorithm (SDA), which limits the input angle range to $(0, \pi/4)$ [66]. The last $2n/3$ number of σ_i's are predicted from the remaining angle of $n/3$ iterations. The internal word length of the architecture proposed for this technique is considered as $(n + \log_2 n)$ for n-bit external accuracy [47]. It may be noted that the complete parallelization of x/y iterations lead to the exponential increase of terms to be flattened, affecting the circuit complexity. In addition, the implementation of flat CORDIC needs complex combinational hardware blocks with poor scalability.

Scale Factor. The scale factor in the implementation of the flat CORDIC algorithm is maintained constant, since $\sigma_i \in \{-1, 1\}$. The scale factor compensation is implemented using a multiplier designed with CS adder tree.

9.3.2. Para-CORDIC [64]. The Para-CORDIC parallelizes the generation of direction of rotations σ from the binary value of the input angle θ by employing binary to bipolar representation (BBR) and microrotation angle recoding (MAR) techniques. This algorithm computes x/y coordinates iteratively while eliminating iterative z path completely. The input angle θ is divided into the higher part θ_H and lower part θ_L. The two's complement binary representation of input angle θ is

$$\theta = (-d_0) + \sum_{i=1}^{l-1} d_i 2^{-i} + \sum_{i=l}^{n} d_i 2^{-i}, \tag{34}$$

where $d_i \in \{0, 1\}$ and $l = (n - \log_2 3)/3$. The $(l - 1)$ bits of input angle are converted into BBR, and MAR technique is employed to determine the direction of rotations σ_1 to σ_{l-1}. Since $\tan^{-1}2^{-i} \neq 2^{-i}$, this method performs additional microrotations for every iteration depending on each positional binary weight 2^{-i} for $i = 1, 2, \ldots, l - 1$. The remaining angle after the first $(l-1)$ rotations is added to θ_L. The values of σ_l to σ_{n+1} are obtained from BBR of the corrected θ_L. This method eliminates ROM for storing the predetermined direction of rotations. However, it requires additional x/y stages for the repetition of a certain microrotations and array of adders to compute the corrected θ_L.

9.3.3. Semi-Flat CORDIC [65]. The iterative nature in the implementation of the conventional CORDIC algorithm is partially eliminated by semi flat algorithm. This is designed for the semi parallelization of the $x/y/z$ recurrences, to improve the speed of a rotational unfolded CORDIC without increasing the area requirements. The internal precision is taken higher than the required external precision in order to reduce the quantization error encountered in the CORDIC algorithm as discussed in Section 3.2.6. For the first λ bits of σ_i, x/y recurrences are computed iteratively using the double rotation method [51] resulting in $x_{\lambda-1}/y_{\lambda-1}$. Then, x_{n-1}/y_{n-1} can be expressed in terms of these $x_{\lambda-1}/y_{\lambda-1}$, if all σ_i's are predicted. The σ_i's for $(n_{\text{int}}/3 - \lambda)$ bits (n_{int} = internal precision) of input angle are precomputed and stored in ROM, which is addressed by $(n_{\text{int}}/3 - \lambda)$ bits of input angle. The remaining $(2n_{\text{int}}/3)$ number of σ_i's are predicted from rotation angle [60]. It may be noted that neither the description nor the reference is provided for split decomposition method employed to precompute $(n_{\text{int}}/3 - \lambda)$ number of σ_i's.

The computation time and area of the chip are affected by the choice of λ, which is clear from the simulation results presented in [65]. It is observed from these simulation results that the best trade-off is obtained with $\lambda = 6$ and $\lambda = 8$ for a 16-bit CORDIC (internal precision 22 bits) and 32-bit CORDIC (internal precision 39 bits) respectively. After λ iterations, all the terms of (x_n/y_n) were added using the Wallace tree, flattening the x/y path. However, this architecture has poor scalability.

Scale Factor. This algorithm achieves constant scale factor, since σ_i takes value from the set $\{-1, 1\}$.

10. Comparison

We have presented a latency estimate comparison of unfolded architectures available in the literature for 2D rotational CORDIC in Table 2. Latency is defined as sum of the delays for the computation of redundant x/y coordinates, scale factor compensation and redundant to binary conversion of final x/y coordinates. The design detail of scale factor compensation and redundant to binary conversion stages is not made available in the literature for all the architectures as discussed in the previous sections (Sections 6–9). Hence, we have compared all the CORDIC algorithms with respect to the latency required for the rotation computation, excluding the scale factor compensation and redundant to binary conversion stages. All the architectures presented in this table are implemented using redundant arithmetic except the conventional CORDIC [3] and the Low latency nonredundant CORDIC [49].

The nonpipelined and pipelined implementation of the conventional radix-2 CORDIC algorithm [3, 7] requires n iterations to compute x/y coordinates iteratively. The iteration delay depends on the fast carry propagate adder, which is the bottleneck to increase throughput and reduce latency.

The application of redundant arithmetic [42] to the conventional CORDIC makes σ_i to take values from the set $\{-1, 0, 1\}$ instead of the set $\{-1, 1\}$. The σ_i values are computed iteratively and the choice of $\sigma_i = 0$ resulted in the variability of the usually constant scale factor. The variable scale factor increases the area and delay for scale factor computation. The latency of this implementation is nt_{stage}, where t_{stage} is the iteration stage delay in terms of full adder delay t_{FA}.

The double rotation and correcting rotation redundant CORDIC methods using SD arithmetic are proposed in [51], to reduce the cost of the scale factor computation. The nonpipelined and pipelined implementation of these methods require latency of $1.5nt_{\text{stage}}$ to compute final x/y coordinates iteratively. These methods achieve constant scale factor, increasing the latency by 50% compared to [42].

Low latency CORDIC algorithm [55] reduces the latency to $((9n-3)/8)t_{\text{stage}}$ compared to that $1.5nt_{\text{stage}}$ in [51]. This algorithm computes iteratively the direction of rotations and x/y coordinates. In addition, a nonpipelined architecture is also proposed in this paper using prediction technique. The latency of this architecture is $(n + \log_3 n - 1)t_{\text{stage}}$.

Branching algorithm using signed digit arithmetic is proposed to achieve constant scale factor. The latency of nonpipelined and pipelined implementation of this algorithm is nt_{stage}. This algorithm achieves 50% latency improvement over [51] to compute final x/y coordinates iteratively. However, it requires double the hardware as two sets of $x/y/z$ modules are employed.

The direction of rotations computed using the sign estimation methods [51, 52, 55] may not be accurate, therefore, half of the computational effort is required for correction. DCORDIC algorithm is proposed to determine the direction of rotations iteratively using the sign of steering variable. However, this method requires an initial latency of nt_{FA} before the CORDIC rotation starts, to obtain the first direction of rotation. The signs are obtained for the remaining iterations with one full adder delay using bit level pipelined architecture with n stages. This implementation requires latency of $nt_{\text{stage}} + (n+1)t_{\text{FA}}$ to compute the final x/y coordinates iteratively. In addition, this method requires $2.5n$ initial register rows for skewing of input data.

All the methods presented so far reduce the latency by decreasing the iteration delay using redundant arithmetic. Since the latency reduction can also be obtained by reducing the number of iterations, the same has motivated to implement radix-4 pipelined CORDIC processor [58], which results in latency of $(3n/4 + 1)t_{\text{stage}}$.

The mixed radix CORDIC algorithm [59] is proposed using radix-2 and radix-4 rotations for designing a pipelined processor to operate in rotation and vectoring modes of circular and hyperbolic coordinate systems. The latency of this pipelined architecture requires $(3n/4 + 1)$ stages with three different stage delays (t_{stage}) as $31t_{\text{NAND}}(1 \le i < n/4)$, $34t_{\text{NAND}}(n/4 \le i \le (n/2+1))$ and $36t_{\text{NAND}}(i > (n/2+1))$. This architecture takes more stage delay as this is designed for various modes of operation.

The advantage of applying radix-4 rotations for all iteration stages is exploited in [43] with less number of adders as compared to [58]. For the microrotations in the range $0 \le i < (n/6)$, the pipelined architecture proposed for this algorithm implementation determines σ_i values sequentially using angle accumulator. For the microrotations in the range $i \ge (n/6)$, the σ_i values are determined from the remaining angle after $n/6$ iterations. The latency of this architecture to compute the final x/y coordinates iteratively is $(2n/3 + 2)t_{\text{stage}}$.

In [61], P-CORDIC algorithm is proposed to eliminate z path completely, using a linear relation between the rotation angle θ and the corresponding direction of all microrotations for rotation mode. This algorithm computes the x/y coordinates iteratively. The latency of the nonpipelined architecture proposed to implement this algorithm for n-bit precision is $(n/12 + \log_2 n + 1.75 + 2n)t_{\text{FA}}$.

The iterative nature in the $x/y/z$ path is eliminated at the cost of scalability by the flat CORDIC algorithm [63]. This algorithm transforms x/y recurrences (11) of the conventional CORDIC into a parallelized version, by successive substitution to express the final vectors in terms of the initial vectors, resulting in a single equation for n-bit precision. The direction of rotations are precomputed

TABLE 2: Comparison of various rotational CORDIC architectures, (SD: signed digit, CS: carry save).

Method (year)	Radix, ρ	Arithmetic	Latency (t_{FA})		Iterative	Iterative	Scale factor
			Nonpipelined	Pipelined	X/Y path	Z-path	K
Nonredundant (1959) [3]	2	2's compliment	n^2	n^2	$\surd$	$\surd$	Constant
Redundant (1987) [42]	2	CS	nt_{stage}	nt_{stage}	$\surd$	$\surd$	Variable
Double-rotation/	2	SD	$1.5nt_{\text{stage}}$	$1.5nt_{\text{stage}}$	$\surd$	$\surd$	Constant
Correcting (1991) [51]							
Low latency (1992) [55]	2	CS	$(n+\log_3 n-1)t_{\text{stage}}$	$((9n-3)/8)t_{\text{stage}}$	$\surd$	$\surd$	Constant
Branching (1993) [52]	2	SD	nt_{stage}	nt_{stage}	$\surd$	$\surd$	Constant
DCORDIC (1996) [56]	2	SD/CS	—	$(nt_{\text{stage}}+n+1)$	$\surd$	$\surd$	Constant
Radix-4 (1993) [58]	4	SD	—	$(3n/4+1)t_{\text{stage}}$	$\surd$	$\surd$	Variable
Radix2-4 (1996) [59]	2–4	CS	—	$(3n/4+1)t_{\text{stage}}$	$\surd$	$\surd$	Constant
Radix-4 (1997) [43]	4	CS	—	$(2n/3+1)t_{\text{stage}}$	$\surd$	$n/6$	Variable
PCORDIC (2002) [61]	2	SD	$(1.7n+1.25+\log_2 n)$	—	$\surd$	$\times$	Constant
Flat CORDIC (2002) [63]	2	SD	34 for 16-bit, 50 for 32-bit	—	combinational	$\times$	Constant
Para-CORDIC (2004) [64]	2	CS	$(2(s(n)+n/2-l+2)+\lceil\log_{1.5}n+2\rceil)$	—	$\surd$	$\times$	Constant
Semi-flat (2006) [65]	2	SD	33 for 16-bit	—	λ/combinational	λ	Constant
Nonredundant low-latency (2008) [49]	2	2's compliment	—	$(n/2+2)t_{\text{adder}}+t_{\text{multiplier}}$	$(n/2+1)$/multiplier	$n/3$	Constant

before initiating the computation of x/y coordinates. The final x/y coordinates are computed using combinational blocks with the latency of $34t_{\text{FA}}$/16-bit and $50t_{\text{FA}}$/32-bit. The expressions for x and y variables need to be derived and combinational building blocks have to be redesigned with change in precision.

In [64], Para-CORDIC algorithm is proposed to precompute the direction of rotations without using ROM, while eliminating iterative z path completely. This method uses additional x/y stages for the repetition of a certain microrotations to predict the direction of rotations in contrast to ROM employed in [61, 63, 65]. The latency of this Para-CORDIC is $((2(s(n) + n/2 - l + 2) + \lceil \log_{1.5} n + 2 \rceil))t_{\text{FA}}$, where $l = (n - \log_2 3)/3$ and $s(n)$ represents the total number of microrotations required in MAR recoding of $(l - 1)$ bits of the input angle. The values of $s(n)$ for 16/32/64-bit precision are 5, 18, 52 respectively.

The semiflat technique is proposed in [65], to partially eliminate the iterative nature in $x/y/z$ paths for the $(n - \lambda)$ iterations ($\lambda = 6$ for a 16-bit CORDIC and $\lambda = 8$ for a 32-bit CORDIC, respectively). The latency of the nonpipelined implementation of this algorithm is $33t_{\text{FA}}$/16-bit and $49t_{\text{FA}}$/32-bit, respectively. It is observed that this architecture is combinational after λ iterations and has poor scalability.

In [49], the x/y coordinates are computed iteratively for the $(n/2+1)$ iterations using $(n/2+1)$ number of fast adders. These values are used to compute the final x/y coordinates using two multipliers in parallel and one adder resulting in the latency of $((n/2+2)t_{\text{adder}} + t_{\text{multiplier}})$. The σ_i values for the first $(n/3 + 1)$ iterations are determined iteratively using the sign of angle accumulator z_i. For the range $(n/3 + 1) < i \leq (n/2 + 1)$, the rotation directions are generated in parallel.

11. Conclusions

In this paper, we have surveyed the algorithms for unfolded implementation of 2D rotational CORDIC algorithms. Special attention has been devoted to the systematic and comprehensive classification of solutions proposed in the literature. In addition to the pipelined implementation of nonredundant radix-2 CORDIC algorithm that has received wide attention in the past, we have discussed the importance of redundant and higher radix algorithms. We have also stressed the importance of prediction algorithms to precompute the directions of rotations and parallelization of x/y path. It is worth noting that the considered algorithms should not be implemented as alternatives over the others, rather they should be integrated depending on the design constraints of a specific application.

We can draw final conclusions about the different algorithms to achieve efficient implementation of application specific rotational CORDIC algorithm. As far as the application of redundant arithmetic to the pipelined implementation of the conventional radix-2 CORDIC algorithm is concerned, area is doubled with reduction in the adder delay of each stage from $(\log_2 n)t_{\text{FA}}$ to $2t_{\text{FA}}$. Similarly, the hardware and iteration delay of redundant radix-2 CORDIC can be reduced by employing prediction technique for the precomputation of direction of rotations. Further, the latency reduction of this can be achieved by integrating the prediction technique with the redundant radix-4 arithmetic trading the area for variable scale factor computation. Another important observation about the solutions proposed with fully parallelization of x/y path is that it affects the modularity and regularity of the architecture leading to a poor scalable implementation. Finally, we conclude that the solution which can allow the design of scalable architecture, employing prediction and x/y path parallelization techniques to redundant CORDIC algorithm can achieve both latency reduction and throughput improvement.

References

[1] D. S. Cochran, "Algorithms and accuracy in the HP-35," *Hewlett-Packard Journal*, vol. 23, no. 10, 1972.

[2] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, Birkhäuser, Boston, Mass, USA, 2004.

[3] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. 8, no. 3, pp. 330–334, 1959.

[4] J. E. Volder, "The birth of CORDIC," *Journal of VLSI Signal Processing*, vol. 25, no. 2, pp. 101–105, 2000.

[5] D. H. Daggett, "Decimal-binary conversions in CORDIC," *IRE Transactions on Electronic Computers*, vol. 8, pp. 335–339, 1959.

[6] J. E. Meggitt, "Pseudo division and pseudo multiplication processes," *IBM Journal*, vol. 6, no. 2, pp. 210–226, 1962.

[7] J. S. Walther, "A unified algorithm for elementary functions," in *Proceedings of the AFIPS Spring Joint Computer Conference*, pp. 379–385, May 1971.

[8] J. S. Walther, "The story of Unified CORDIC," *Journal of VLSI Signal Processing*, vol. 25, no. 2, pp. 107–112, 2000.

[9] G. L. Haviland and A. A. Tuszynski, "A CORDIC arithmetic processor chip," *IEEE Journal of Solid-State Circuits*, vol. 15, no. 1, pp. 4–15, 1980.

[10] Y. H. Hu, "CORDIC-based VLSI architectures for digital signal processing," *IEEE Signal Processing Magazine*, vol. 9, no. 3, pp. 16–35, 1992.

[11] A. A. J. de Lange, A. J. van der Hoeven, E. F. Deprettere, and J. Bu, "Optimal floating-point pipeline CMOS CORDIC processor," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '88)*, vol. 3, pp. 2043–2047, June 1988.

[12] A. A. J. de Lange, A. J. van der Hoeven, E. F. Deprettere, and P. Dewilde, "An application specific IC for digital signal processing: the floating point pipeline CORDIC processor," in *Proceedings of the European Conference on ASIC Design (ASIC '90)*, pp. 62–67, May 1990.

[13] D. E. Metafas and C. E. Goutis, "A DSP processor with a powerful set of elementary arithmetic operations based on cordic and CCM algorithms," *Microprocessing and Microprogramming*, vol. 30, no. 1–5, pp. 51–57, 1990.

[14] D. Timmermann, H. Hahn, B. J. Hosticka, and G. Schmidt, "A programmable CORDIC chip for digital signal processing applications," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 9, pp. 1317–1321, 1991.

[15] A. A. J. 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," in *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pp. 272–281, June 1991.

[16] A. M. Despain, "Fourier transform computers using CORDIC iterations," *IEEE Transactions on Computers*, vol. 23, no. 10, pp. 993–1001, 1974.

[17] H. M. Ahmed, J.-M. Delosme, and M. Morf, "Highly concurrent computing structures for matrix arithmetic and signal processing," *Computer*, vol. 15, no. 1, pp. 65–82, 1982.

[18] Y. H. Hu and S. Naganathan, "A novel implementation of a chirp Z-transform using a CORDIC processor," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 2, pp. 352–354, 1990.

[19] A. S. Dhar and S. Banerjee, "An array architecture for fast computation of discrete Hartley transform," *IEEE transactions on circuits and systems*, vol. 38, no. 9, pp. 1095–1098, 1991.

[20] K. Maharatna, A. S. Dhar, and S. Banerjee, "A VLSI array architecture for realization of DFT, DHT, DCT and DST," *Signal Processing*, vol. 81, no. 9, pp. 1813–1822, 2001.

[21] K. C. Ray and A. S. Dhar, "CORDIC-based unified VLSI architecture for implementing window functions for real time spectral analysis," *IEE Proceedings: Circuits, Devices and Systems*, vol. 153, no. 6, pp. 539–544, 2006.

[22] S. K. Rao and T. Kailath, "Orthogonal digital filters for VLSI implementation," *IEEE transactions on circuits and systems*, vol. 31, no. 11, pp. 933–945, 1984.

[23] P. P. Vaidyanathan, "A unified approach to orthogonal digital filters and wave digital filters based on LBR two pair extraction," *IEEE transactions on circuits and systems*, vol. 32, no. 7, pp. 673–686, 1985.

[24] Y. H. Hu and H. E. Liao, "CALF: a CORDIC adaptive lattice filter," *IEEE Transactions on Signal Processing*, vol. 40, no. 4, pp. 990–993, 1992.

[25] J. R. Cavallaro and F. T. Luk, "CORDIC arithmetic for an SVD processor," *Journal of Parallel and Distributed Computing*, vol. 5, no. 3, pp. 271–290, 1988.

[26] J. A. Lee and T. Lang, "SVD by constant-factor-redundant-CORDIC," in *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pp. 264–271, June 1991.

[27] A. Banerjee, A. S. Dhar, and S. Banerjee, "FPGA realization of a CORDIC based FFT processor for biomedical signal processing," *Microprocessors and Microsystems*, vol. 25, no. 3, pp. 131–142, 2001.

[28] A. Meyer-Bäse, R. Watzel, U. Meyer-Bäse, and S. Foo, "A parallel CORDIC architecture dedicated to compute the Gaussian potential function in neural networks," *Engineering Applications of Artificial Intelligence*, vol. 16, no. 7-8, pp. 595–605, 2003.

[29] C. Y. Kang and E. E. Swartzlander Jr., "Digit-pipelined direct digital frequency synthesis based on differential CORDIC," *IEEE Transactions on Circuits and Systems I*, vol. 53, no. 5, pp. 1035–1044, 2006.

[30] H. Wang, P. Leray, and J. Palicot, "Reconfigurable architecture for MIMO systems based on CORDIC operators," *Comptes Rendus Physique*, vol. 7, no. 7, pp. 735–750, 2006.

[31] G. J. Hekstra and E. F. A. Deprettere, "Fast rotations: low-cost arithmetic methods for orthonormal rotation," in *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*, pp. 116–125, October 1997.

[32] K. Hwang, *Computer Arithmetic: Principles, Architecture and Design*, John Wiley & Sons, New York, NY, USA, 1979.

[33] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley & Sons, New York, NY, USA, 1999.

[34] A. Avizienis, "Signed-digit number representation for fast parallel arithmetic," *IRE Transactions on Electronic Computers*, vol. 10, pp. 389–400, 1961.

[35] D. E. Atkins, "Introduction to the role of redundancy in computer arithmetic," *IEEE Computer Magazine*, vol. 8, no. 6, pp. 74–77, 1975.

[36] B. Parhami, "Carry-free addition of recorded binary signed-digit numbers," *IEEE Transactions on Computers*, vol. 37, no. 11, pp. 1470–1476, 1988.

[37] B. Parhami, "Generalized signed-digit number systems: a unifying framework for redundant number representations," *IEEE Transactions on Computers*, vol. 39, no. 1, pp. 89–98, 1990.

[38] T. G. Noll, "Carry-save arithmetic for high-speed digital signal processing," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 982–986, May 1990.

[39] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proceedings of the 6th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '98)*, pp. 191–200, February 1998.

[40] P. W. Baker, "Suggestion for a fast binary sine/cosine generator," *IEEE Transactions on Computers*, vol. 25, no. 11, pp. 1134–1136, 1976.

[41] Y. H. Hu, "Pipelined CORDIC architecture for the implementation of rotational based algorithm," in *Proceedings of the International Symposium on VLSI Technology, Systems and Applications*, p. 259, May 1985.

[42] M. D. Ercegovac and T. Lang, "Fast cosine/sine implementation using on-line CORIC," in *Proceedings of the 21st Asilomar Conference on Signals, Systems, and Computers*, 1987.

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

[44] D. Timmermann, H. Hahn, B. J. Hosticka, and B. Rix, "A new addition scheme and fast scaling factor compensation methods for CORDIC algorithms," *The VLSI Journal on Integration*, vol. 11, no. 1, pp. 85–100, 1991.

[45] J. Villalba, J. A. Hidalgo, E. L. Zapata, E. Antelo, and J. D. Bruguera, "CORDIC architectures with parallel compensation of the scale factor," in *Proceedings of the International Conference on Application Specific Array Processors*, pp. 258–269, Strasbourg, France, July 1995.

[46] X. Hu, R. G. Harber, and S. C. Bass, "Expanding the range of convergence of the CORDIC algorithm," *IEEE Transactions on Computers*, vol. 40, no. 1, pp. 13–21, 1991.

[47] Y. H. Hu, "The quantization effects of the CORDIC algorithm," *IEEE Transactions on Signal Processing*, vol. 40, no. 4, pp. 834–844, 1992.

[48] M. D. Erecegovac and T. Lang, *Digital Arithmetic*, Elsevier, Amsterdam, The Netherlands, 2004.

[49] E. Antelo, J. Villalba, and E. L. Zapata, "A low-latency pipelined 2D and 3D CORDIC processors," *IEEE Transactions on Computers*, vol. 57, no. 3, pp. 404–417, 2008.

[50] H. M. Ahmed, "Efficient elementary function generation with multipliers," in *Proceedings of the 9th Symposium on Computer Arithmetic*, pp. 52–59, September 1989.

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

[52] J. Duprat and J.-M. Muller, "The CORDIC algorithm: new results for fast VLSI implementation," *IEEE Transactions on Computers*, vol. 42, no. 2, pp. 168–178, 1993.

[53] D. S. Phatak, "Double step branching CORDIC: a new algorithm for fast sine and cosine generation," *IEEE Transactions on Computers*, vol. 47, no. 5, pp. 587–602, 1998.

[54] J. A. Lee and T. Lang, "Constant-factor redundant CORDIC for angle calculation and rotation," *IEEE Transactions on Computers*, vol. 41, no. 8, pp. 1016–1025, 1992.

[55] D. Timmermann, H. Hahn, and B. J. Hosticka, "Low latency time CORDIC algorithms," *IEEE Transactions on Computers*, vol. 41, no. 8, pp. 1010–1015, 1992.

[56] H. Dawid and H. Meyr, "The differential CORDIC algorithm: constant scale factor redundant implementation without correcting iterations," *IEEE Transactions on Computers*, vol. 45, no. 3, pp. 307–318, 1996.

[57] H. Dawid and H. Meyr, "High speed bit-level pipelined architectures for redundant CORDIC implementation," in *Proceedings of the International Conference on Application*, pp. 358–372, 1992.

[58] J. D. Bruguera, E. Antelo, and E. L. Zapata, "Design of a pipelined Radix 4 CORDIC processor," *Parallel Computing*, vol. 19, no. 7, pp. 729–744, 1993.

[59] E. Antelo, J. D. Bruguera, and E. L. Zapata, "Unified mixed Radix 2–4 redundant CORDIC processor," *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 1068–1073, 1996.

[60] S. Wang, V. Piuri, and E. E. Swartzlander Jr., "Hybrid CORDIC algorithms," *IEEE Transactions on Computers*, vol. 46, no. 11, pp. 1202–1207, 1997.

[61] M. Kuhlmann and K. K. Parhi, "P-CORDIC: a precomputation based rotation CORDIC algorithm," *EURASIP Journal on Applied Signal Processing*, vol. 2002, no. 9, pp. 936–943, 2002.

[62] M. Kuhlmann and K. K. Parhi, "A high-speed CORDIC algorithm and architecture for DSP applications," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS '99)*, pp. 732–741, October 1999.

[63] B. Gisuthan and T. Srikanthan, "Pipelining flat CORDIC based trigonometric function generators," *Microelectronics Journal*, vol. 33, no. 1-2, pp. 77–89, 2002.

[64] T.-B. Juang, S.-F. Hsiao, and M.-Y. Tsai, "Para-CORDIC: parallel CORDIC rotation algorithm," *IEEE Transactions on Circuits and Systems I*, vol. 51, no. 8, pp. 1515–1524, 2004.

[65] H. S. Kebbati, J. Ph. Blonde, and F. Braun, "A new semi-flat architecture for high speed and reduced area CORDIC chip," *Microelectronics Journal*, vol. 37, no. 2, pp. 181–187, 2006.

[66] T. Srikanthan and B. Gisuthan, "A novel technique for eliminating iterative based computation of polarity of micro-rotations in CORDIC based sine-cosine generators," *Microprocessors and Microsystems*, vol. 26, no. 5, pp. 243–252, 2002.