Università degli Studi di Padova

Department of Information Engineering

Bachelor's degree in Electronic Engineering

CORDIC Algorithm and its Applications

Supervisor
Daniele Vogrig

Student
Zhi Long Zhou

Academic Year 2022/2023

To my family and friends, a special thanks to Angelo, Kuma, Federico, Mariam, Dalton, Emin and Vittorio.

Abstract

The CORDIC (Coordinate Rotation Digital Computer) algorithm is used for solving vast sets of functions such as trigonometric functions, hyperbolic functions and natural logarithms. This *thesis* is going to discuss how the algorithm works and its architecture implementation. It is also going to explore potential applications of the algorithm in digital communication systems, specifically for the realization of DDS (Direct Digital Synthesis) and digital modulation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The CORDIC algorithm (COrdinate Rotation DIgital Computer) is an iterative method to efficiently compute trigonometric, hyperbolic and logarithmic functions. Since it requires only addition/subtraction, bit-shifting operation and ROM look-up tables the computational complexity is extremely low, making it suitable for low cost implementation when no hardware multiplier is available.

1.1 The basics of CORDIC

1.1.1 Intuitive approach

Before understanding the CORDIC algorithm let's take a step back and consider the following problem:

- A straight line of fixed distance, from 0 to 100
- A target placed inside the line at an unknown distance
- We can move at fixed distances to the left or right
- The fixed distances are progressively smaller

To find the distance of the target from the origin, we can use a binary search approach. Begin from the origin and move to the midpoint of the line (50), if the target is on the left, divide the line in half and move left by half the lenght of the new line (25), otherwise take the right half of the line and move right. Repeat the process and by progressively halving the line we get closer and closer to the target until we reach the target.

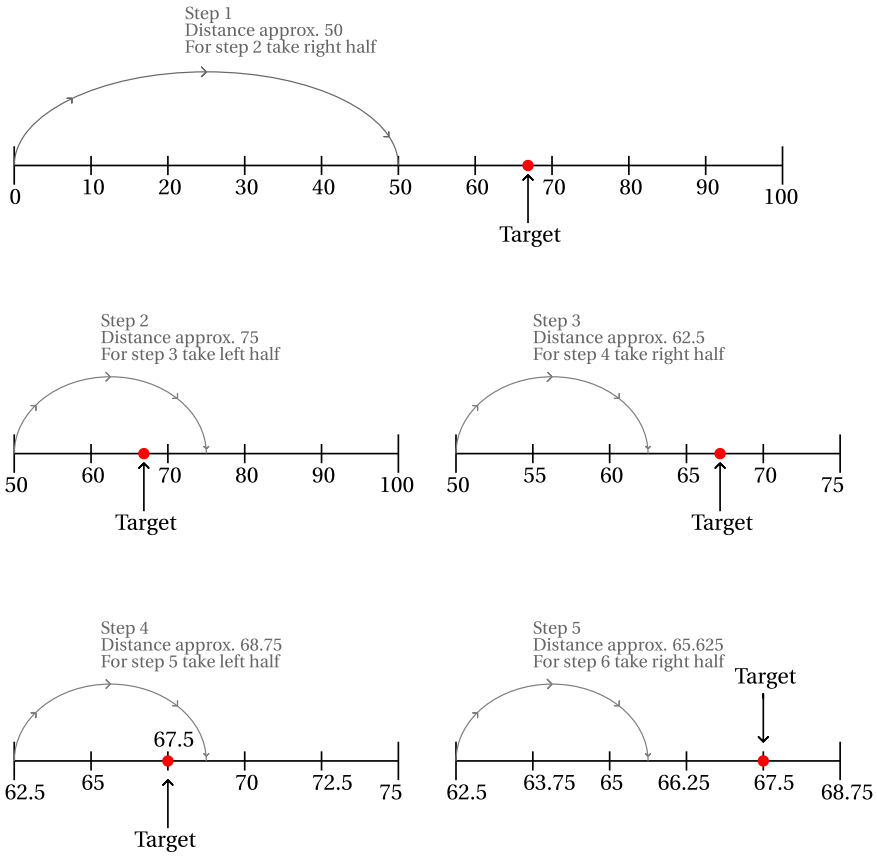


Figure 1.1: *Graphic representation of target distance search using a binary search approach*

With each step our computed distance approximation gets closer and closer until we get to the real distance value of the target.

The CORDIC algorithm works with a similar pattern, but on a circumference[1], and instead of fixed linear addition/subtraction of progressively smaller distances it requires trigonometric identities using fixed angle rotations.

[1]The CORDIC algorithm can be further generalized for hyperbolic functions, the generalized algorithm will be introduced later in the thesis

1.1.2 Functional description

Given a vector $\vec{a}$ on the Cartesian plane with magnitude R and phase θ, the new vector $\vec{b}$ corresponding to the rotation of $\vec{a}$ by an angle α about the origin can be described as:

$$\vec{a} = \begin{bmatrix} R\cos\theta \\ R\sin\theta \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} R\cos(\theta+\alpha) \\ R\sin(\theta+\alpha) \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{1.1}$$

And with these trigonometric identities in mind:

$$\begin{aligned} \cos(\alpha+\beta) &= \cos\alpha\cos\beta - \sin\alpha\sin\beta \\ \sin(\alpha+\beta) &= \cos\alpha\sin\beta + \sin\alpha\cos\beta \end{aligned} \tag{1.2}$$

The vector $\vec{b}$ can be rewritten as:

$$\begin{aligned} \vec{b} = \begin{bmatrix} R\cos(\theta+\alpha) \\ R\sin(\theta+\alpha) \end{bmatrix} &= \begin{bmatrix} R\cos\theta\cos\alpha - R\sin\theta\sin\alpha \\ R\cos\theta\sin\alpha + R\sin\theta\cos\alpha \end{bmatrix} \\ &= \begin{bmatrix} x\cos\alpha - y\sin\alpha \\ x\sin\alpha + y\cos\alpha \end{bmatrix} \\ &= \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned} \tag{1.3}$$

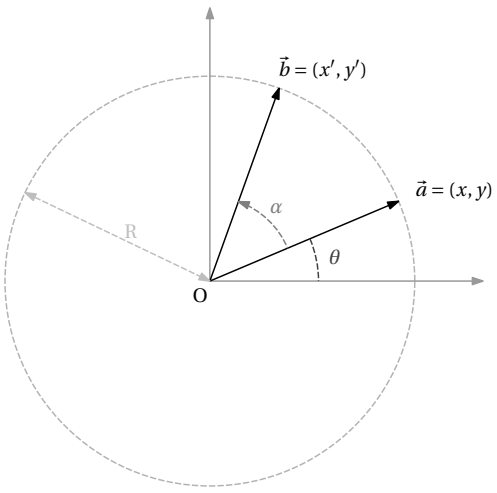


Figure 1.2: *Rotation of vector $\vec{a}$ to vector $\vec{b}$ on a Cartesian plane*

We just learned that given any vector with coordinates (x, y), to compute the rotated vector (x', y') by an angle of α about the origin we only need the pre-computed values of $\cos\alpha$, $\sin\alpha$ and basic math operations of addition/subtraction/multiplication.

We can then define a rotation matrix $R(\alpha)$ that relates the two vectors:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = R(\alpha) \begin{bmatrix} x \\ y \end{bmatrix} \tag{1.4}$$

Suppose we need to find the coordinates of the vector $(1, 0)$ after a rotation of an angle ϕ between $[0, \pi/2]$. With a binary search approach pre-compute the sine and cosine values of half the full range angle $pi/2$ which is 45°, then the quarter 22.5°, then 11.25° and so on. With each step rotation $\vec{b_1}$, $\vec{b_2}$ and $\vec{b_3}$ are computed, which are progressively closer to the value of $\vec{b} = (x_\theta, y_\theta)$.

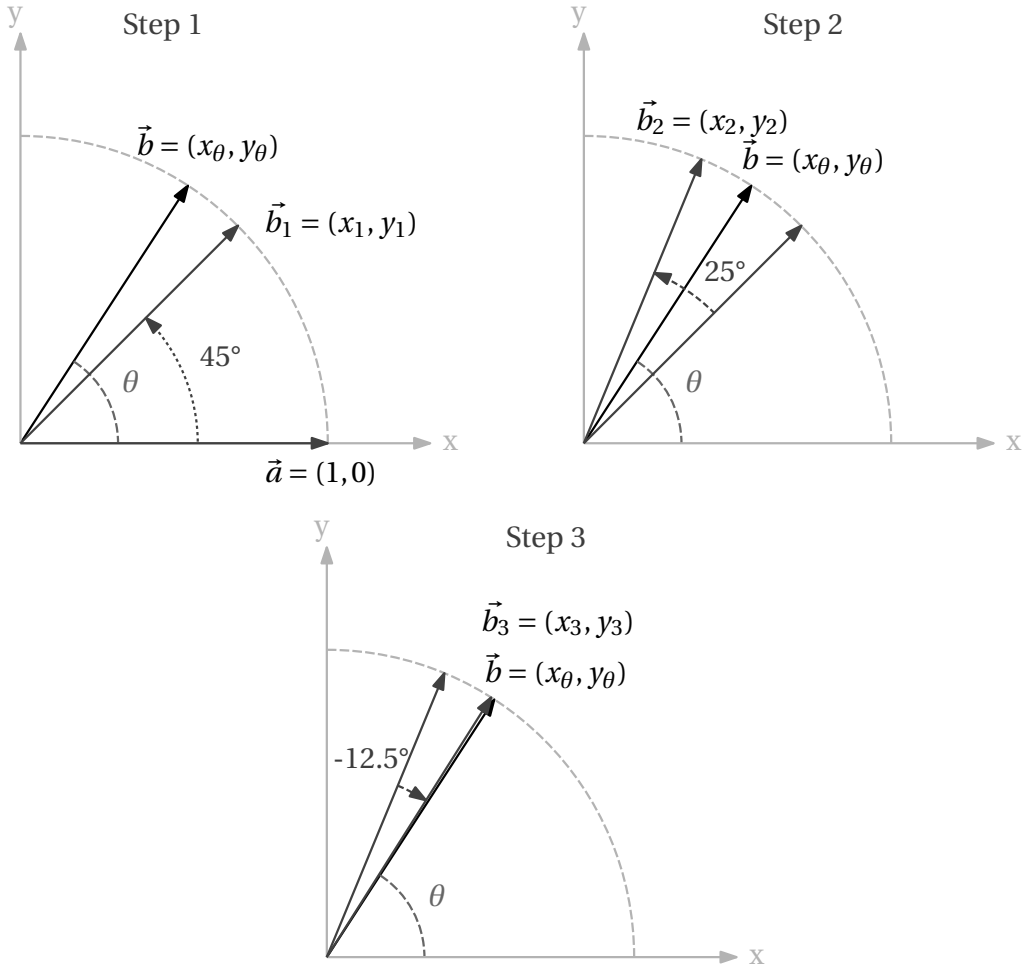


Figure 1.3: *First 3 steps rotations of vector $\vec{a}$ to vector $\vec{b_1}$ $\vec{b_2}$ and $\vec{b_3}$.*

In other words, we have decomposed the rotation θ into N smaller fixed rotations with the aid of a decision coefficient d_i that can assume either 1 or -1 depending on the current step.

$$\begin{aligned} \theta = \sum_{i=0}^{N} d_i \theta_i \qquad \text{with} \qquad & \theta_0 = 0 \\ & \theta_i = \frac{\pi}{4i} \quad \forall i > 0 \end{aligned} \tag{1.5}$$

The steps can be either finite (we can define θ with limited θ_i) or infinite; in the latter with a big enough N we can eventually converge to the real value of θ . This is an iterative algorithm with input vector $\vec{a}$ and each iteration output $\vec{b_i}$ closer or equal to $\vec{b}$. The step 0 which corresponds to a rotation of θ_0 is introduced in case the rotation is 0 degrees; the algorithm will end immediately since no rotation is needed, in fact $\vec{b} = \vec{a}$.

We basically find the vector $\vec{b}$ with rotation matrices containing progressively smaller angles. Let's analyze the first 3 iteration steps:

$$\begin{aligned} \vec{b_1} &= \begin{bmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = R\left(\frac{\pi}{4}\right)\vec{a} \\ \vec{b_2} &= R\left(\frac{\pi}{8}\right)\vec{b_1} \\ \vec{b_3} &= R\left(-\frac{\pi}{16}\right)\vec{b_2} \end{aligned} \tag{1.6}$$

To generalize this algorithm:

$$\begin{aligned} \vec{b} = \begin{bmatrix} x_\theta \\ y_\theta \end{bmatrix} &= R\left(\frac{\pi}{4}\right) R\left(\frac{\pi}{8}\right) R\left(-\frac{\pi}{16}\right) \dots \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= \prod_{i=1}^{N} R(d_i\theta_i) \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{aligned} \tag{1.7}$$

1.2 Basic CORDIC iteration

The algorithm we just covered can be implemented with minimum hardware cost with just a few optimization exploiting some binary operation properties, in particular the transformation of multiplication into shift operations. Let's normalize the rotation matrix by $\cos\alpha = K^{-1}$ and defining a new pseudorotation matrix $\mathbf{R_c}$:

$$\begin{aligned}\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \\ &= \cos\alpha \begin{bmatrix} 1 & -\tan\alpha \\ \tan\alpha & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{K}\ \mathbf{R_c}(\alpha)\ \begin{bmatrix} x \\ y \end{bmatrix}\end{aligned} \tag{1.8}$$

What if $\tan\alpha = 2^{-n}$? In the base-2 numeral system the multiplication with 2^{-n} (assuming $n > 0$) is essentially a right shift operation. The pseudo-rotation matrix is now given by :

$$\begin{aligned}R_c(\alpha) = R_c(\tan^{-1}(2^{-n})) &= \begin{bmatrix} 1 & -\tan(\tan^{-1}(2^{-n})) \\ \tan(\tan^{-1}(2^{-n})) & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & -2^{-n} \\ 2^{-n} & 1 \end{bmatrix}\end{aligned} \tag{1.9}$$

To find the coordinates of $\vec{b} = (x_\theta, y_\theta)$ of a generic vector $\vec{a} = (x_a, y_a)$ after a θ angle rotation by the origin we apply the algorithm defined in the previous section, with the only difference being that the angle decomposition equation (1.5) is now using a different set of angles:

$$\begin{aligned}\theta = \sum_{i=0}^{N} d_i\theta_i \qquad \text{with} \qquad & \theta_0 = 0 \\ & \theta_i = \arctan 2^{-i} \quad \forall i > 0\end{aligned} \tag{1.10}$$

And the coordinates of the vector $\vec{b}$ using the new pseudorotation matrix $\mathbf{R_c}$:

$$\vec{b} = \prod_{i=1}^{N} R(d_i\theta_i) \begin{bmatrix} x_a \\ y_a \end{bmatrix} = \prod_{i=1}^{N} \frac{1}{K_i}\ \mathbf{R_c}(d_i\theta_i) \begin{bmatrix} x_a \\ y_a \end{bmatrix} \tag{1.11}$$

Volder [3] introduced another simplification to the algorithm , in fact the product of the scale factors will eventually converge to a finite number, in the case of our set of angles:

$$K = \lim_{N\to\infty} \prod_{i=1}^{N} K_i \simeq 1.6467605 \tag{1.12}$$

Therefore, we can begin by computing the pseudo-rotated vector using the pseudo-rotation matrix that only requires shift/add operations and then one sigle K factor scaling at the

end, which is well known in advance:

$$\begin{aligned} \vec{b'} &= \prod_{i=1}^{N} \mathbf{R_c}(d_i\theta_i) \begin{bmatrix} x_a \\ y_a \end{bmatrix} \\ \vec{b} &= \prod_{i=1}^{N} \frac{1}{K_i}\ \vec{b'} = \frac{1}{K} \cdot \vec{b'} \end{aligned} \tag{1.13}$$

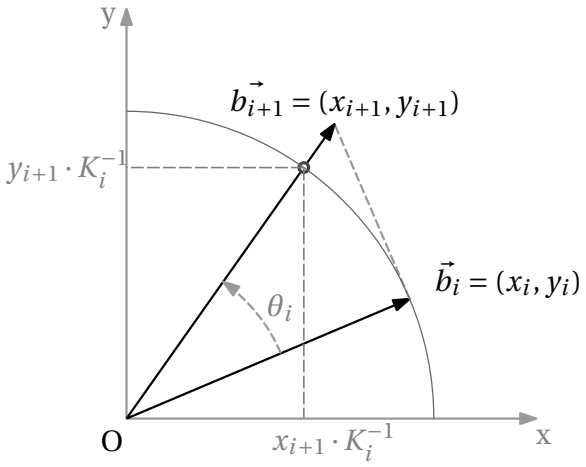


Figure 1.4: *Visualization of the pseudo-rotation after each iteration.*

To summarize, the CORDIC algorithm is an iterative algorithm which performs a full vector rotation by decomposing it into smaller microrotations using a predetermined set of angles $\theta_i = \arctan 2^{-i}$ and decision coefficient $d_i = \pm 1$ decided at the i-th iteration. Each i-th iteration has an input of the vector (x_i, y_i) and angle z_i, after the microrotation we obtain the pseudo-rotated vector (x_{i+1}, y_{i+1}) and angle z_{i+1} given by :

$$\begin{aligned} x_{i+1} &= x_i - y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} &= y_i + y_i \cdot d_i \cdot 2^{-i} \\ z_{i+1} &= z_i - d_i \cdot \theta_i \qquad \text{with} \qquad z_0 = 0 \end{aligned} \tag{1.14}$$

The CORDIC iteration step of equation 1.14 can be used in two operating modes, the rotation mode (RM) and vectoring mode (VM); what determines the decision coefficient at the i-th iteration depends on which mode the algorithm is set to work with.

1.2.1 Rotation Mode

In RM, we input the vector (x_0, y_0) we want rotated by an angle z_0. After each iteration d_i is determined by the sign of z_i: if z_i is positive then $d_i = 1$ otherwise $d_i = -1$. If $z_i = 0$ or after a fixed number of iterations we can end the algorithm and output the pseudo-rotated vector. Scale the pseudo-rotated factor by K to compute the rotated vector.
After n iterations we get the following results:

$$\begin{aligned} x_n &= K\left(x_0\ \cos(z_0) - y_0\ \sin(z_0)\right) \\ y_n &= K\left(y_0\ \cos(z_0) + x_0\ \sin(z_0)\right) \\ z_n &= 0 \end{aligned} \tag{1.15}$$

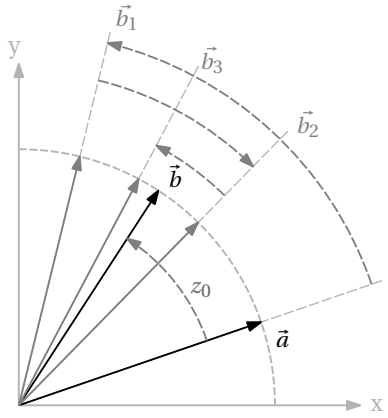


Figure 1.5: *Visualization of the rotation mode, the scale factor has been omitted.* $\vec{a} = (x_0, y_0)$ *input and* $\vec{b}$ *output*

1.2.2 Vectoring Mode

In VM, we input the vector (x_0, y_0) which will rotate towards the x-axis until the y-component reaches to zero. After each iteration if d_i is determined by the sign of y_i: if y_i is positive then $d_i = -1$ otherwise $d_i = 1$. If $y_i = 0$ or after a fixed number of iterations we can end the algorithm and output the angle of rotation z_i of the vector (x_0, y_0).
After n iterations we get the following results:

$$\begin{aligned} x_n &= K\sqrt{x_0^2 + y_0^2} \\ y_n &= 0 \\ z_n &= z_0 + \tan^- 1\left(\frac{y_0}{x_0}\right) \end{aligned} \tag{1.16}$$

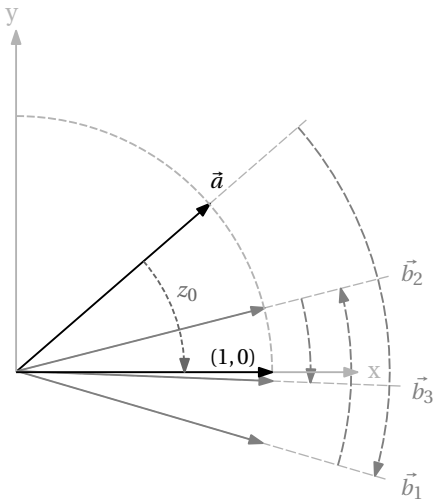


Figure 1.6: *Visualization of the vectoring mode, the scale factor has been omitted.* $\vec{a} = (x_0, y_0)$ *is the input.*

1.2.3 Generalized CORDIC algorithm

The CORDIC iterations can be generalized to not only compute trigonometric functions, but also hyperbolic functions [4]; the CORDIC algorithm has then been reformulated in to a generalized and unified algorithm capable of performing rotations in circular, hyperbolic and linear coordinates system. Consider a coordinate system parametrized in m , given a vector $P = (x, y)$ with radius R and angle A are defined as:

$$\begin{aligned} R &= \left(x^2 + my^2\right)^{1/2} \\ A &= m^{-1/2} \tan^{-1}\left(m^{1/2}\frac{y}{x}\right) \end{aligned} \tag{1.17}$$

By setting the parameter m we can decide in which coordinates system the vector will rotate. A proper angle decomposition α_i is chosen for each coordinates system in order to exploit the multiplication-to-shift property of the base-2 numeral system. Let's break down equations 1.17 to better understand how the parameter m influences the radius R and angle A.

Circular coordinates m = 1

$$\begin{aligned} R &= \left(x^2 + y^2\right)^{1/2} \\ A &= \tan^{-1}\left(\frac{y}{x}\right) = \arg(x, y) \end{aligned} \tag{1.18}$$

The equations describe all vectors on a circle of radius R and angle A. We use $\alpha_i = \tan^{-1}(2^{-i})$.

Linear coordinates m = 0

$$\begin{aligned} R &= x \\ A &= \lim_{m\to 0} m^{-1/2} \tan^{-1}\left(m^{1/2}\frac{y}{x}\right) = \frac{y}{x} \end{aligned} \tag{1.19}$$

The equations describe all vectors on a the line $x = R$, A is the inclination of the vector. We use $\alpha_i = 2^{-i}$.

Hyperbolic coordinates m = −1

With these mathematical definitions in mind:

$$\begin{aligned} j &\equiv \sqrt{-1} \\ \tan^{-1}(j \cdot a) &\equiv j \ \tanh^{-1}(a) \end{aligned} \tag{1.20}$$

The equations of the vector P are:

$$R = \left(x^2 - y^2\right)^{1/2}$$

$$\begin{aligned} A &= (-1)^{-1/2} \tan^{-1}\left((-1)^{1/2}\frac{y}{x}\right) \\ &= \frac{1}{j}\tan^{-1}\left(j\frac{y}{x}\right) = \tanh^{-1}\left(\frac{y}{x}\right) \end{aligned} \tag{1.21}$$

The equations describe all vectors on an hyperbole of radius R and hyperbolic angle A. We use $\alpha_i = \tanh^{-1}(2^{-i})$.

The pseudo-rotation of the vector $P = (x_i, y_i)$ by α_i into $P' = (x_{i+1}, y_{i+1})$ can be obtained in a similar manner as the iteration step of (1.13):

$$\begin{aligned} x_{i+1} &= x_i - m \cdot y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} &= y_i + y_i \cdot d_i \cdot 2^{-i} \\ z_{i+1} &= z_i - d_i \cdot \alpha_i \end{aligned} \tag{1.22}$$

With the decision coefficient d_i depending on the mode:

$$d_i = \begin{cases} \text{sign}(z_i) & \text{for rotation mode} \\ -\text{sign}(y_i) & \text{for vectoring mode} \end{cases} \tag{1.23}$$

We can also describe this rotation using the radius and angle:

$$\begin{aligned} A_{i+1} &= A_i - \alpha_i \\ R_{i+1} &= R_i \cdot K_i \qquad \text{with} \qquad K_i = \left(1 + m \cdot 2^{-2i}\right)^{1/2} \end{aligned} \tag{1.24}$$

Note that we are doing a pseudo-rotation by α_i. Even in different coordinates system the

products of the scaling factor K_i at the end of the algorithm still converges[2] to a finite number:

$$\begin{aligned} K_c &= \lim_{N\to\infty} \prod_{i=1}^{N} \sqrt{1+2^{-2i}} \simeq 1.6467605 \qquad \text{with} \qquad m=1 \\ K_l &= \lim_{N\to\infty} \prod_{i=1}^{N} \sqrt{1} = 1 \qquad \text{with} \qquad m=0 \\ K_h &= \lim_{N\to\infty} \prod_{i=1}^{N} \sqrt{1-2^{-2i}} \simeq 0.8281 \qquad \text{with} \qquad m=-1 \end{aligned} \tag{1.25}$$

We can summarize the input-output vectors (input is the vector $P = (x_0, y_0)$ with angle z_0, output is the vector $P_n = (x_n, y_n)$ with the angle z_n) of the generalized CORDIC algorithm after n iterations with the following table:

Table 1.1: *Generalized CORDIC Algorithm*

m	rotation mode	vectoring mode
1	$x_n = K_c\,(x_0\ \cos(z_0) - y_0\ \sin(z_0))$	$x_n = K_c\sqrt{x_0^2+y_0^2}$
	$y_n = K_c\,(y_0\ \cos(z_0) + x_0\ \sin(z_0))$	$y_n = 0$
	$z_n = 0$	$z_n = z_0 + \tan^{-1}(y_0/x_0)$
0	$x_n = x_0$	$x_n = x_0$
	$y_n = y_0 + x_0 z_0$	$y_n = 0$
	$z_n = 0$	$z_n = z_0 + (y_0/x_0)$
-1	$x_n = K_h\,(x_0\ \cosh(z_0) - y_0\ \sinh(z_0))$	$x_n = K_h\sqrt{x_0^2-y_0^2}$
	$y_n = K_h\,(y_0\ \cosh(z_0) + x_0\ \sinh(z_0))$	$y_n = 0$
	$z_n = 0$	$z_n = z_0 + \tanh^{-1}(y_0/x_0)$

[2]The hyperbolic CORDIC requires the execution of iterations $i = 4, 13, 40...$ twice in order to converge [2]

To better understand how the generalized CORDIC works, let's visualize one i-th iteration of the vector $P = (x_i, y_i)$ with radius R and angle A to the vector $P = (x_{i+1}, y_{y+1})$ with different coordinates system on the cartesian plane. Note that the vector rotation α_i on the linear and hyperbolic coordinates system is not by a geometric angle, but vector "inclination" and hyperbolic angle respectively.

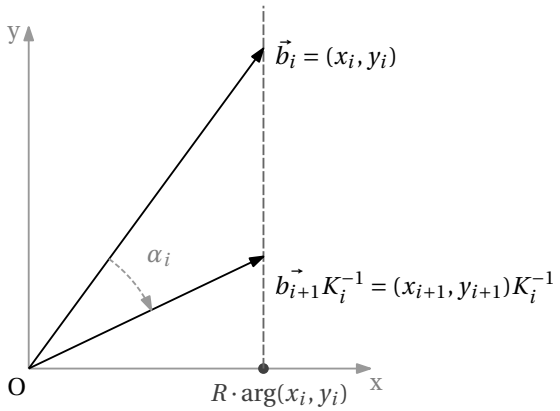


Figure 1.7: *Rotation of the vector $P = (x_i, y_i)$ with hyperbolic coordinates on cartesian plane*

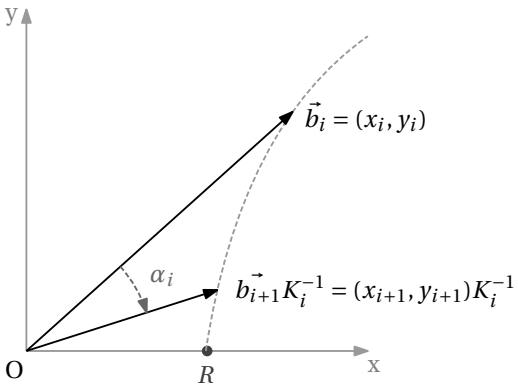


Figure 1.8: *Rotation of the vector $P = (x_i, y_i)$ with linear coordinates on cartesian plane*

1.2.4 Convergence

Returning to the CORDIC algorithm on circular coordinates, the angle decomposition using n microrotations (1.9) is limited by the satisfaction of the convergence theorem [4] :

$$\begin{aligned} \theta &= \sum_{i=0}^{n} d_i \theta_i \qquad \text{with} \qquad \theta_0 = 0;\ \theta_i = \arctan 2^{-i} \quad \forall i > 0 \\ \theta_i &\leq \theta_{n-1} + \sum_{j=i+1}^{n-1} \theta_j \qquad \forall i, i = 0, 1, 2, \ldots, n-2 \end{aligned} \tag{1.26}$$

A convergence range for z_0 with a large iteration number n is therefore obtained [5] :

$$|z_0| \leq \tan^{-1}(2^{-n}) + \sum_{i=0}^{n} \tan^{-1}(2^{-i}) \simeq 1.74329 = 99.9 \tag{1.27}$$

Generally, we would like to expand the convergence range to $\pm\pi$. To do so, an extra iteration is performed, which is a rotation through $\pm\pi/2$ and it is represented as a "negative" iteration step (since it takes place before the actual CORDIC iterations):

$$\begin{aligned} x_0 &= -d_{-i} \cdot y_{-i} \\ y_0 &= d_{-i} \cdot x_{-i} \\ z_0 &= z_{-i} - d_{-i} \cdot \alpha_{-i} \qquad \text{where} \qquad \alpha_{-i} \pm \frac{\pi}{2} \end{aligned} \tag{1.28}$$

Chapter 2

Hardware implementation

2.1 Basic CORDIC structure

Taking the basic CORDIC iteration step of (1.13) we can try to build its hardware implementation structure:

$$\begin{aligned} x_{i+1} &= x_i - y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} &= y_i + y_i \cdot d_i \cdot 2^{-i} \\ z_{i+1} &= z_i - d_i \cdot \theta_i \qquad \text{with} \qquad z_0 = 0 \end{aligned} \tag{2.1}$$

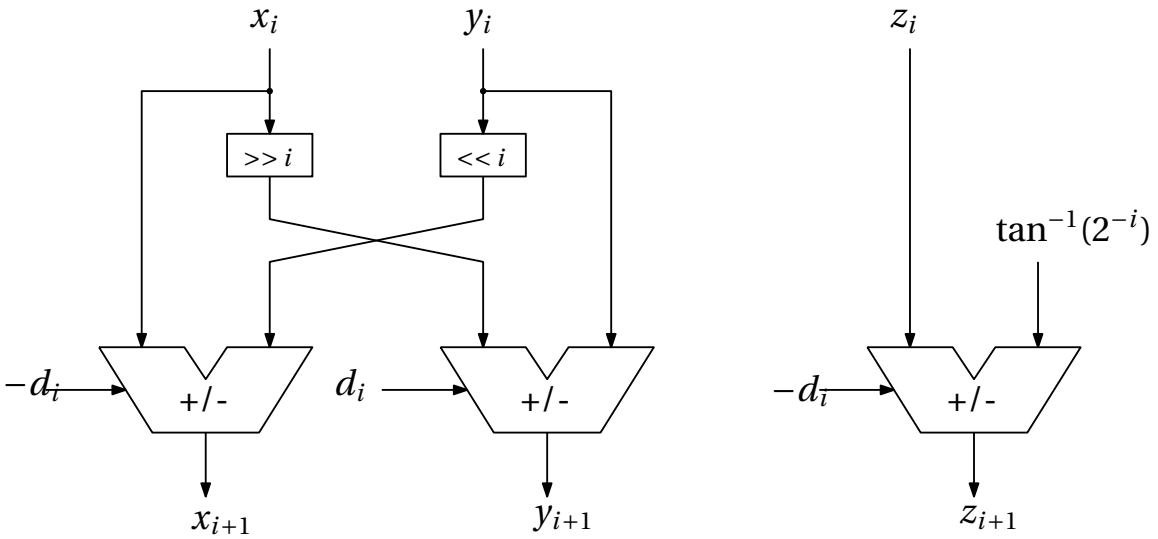


Figure 2.1: *Draft of a hardware implementation of a CORDIC iteration*

Considering that the input coordinates of the vector and the precomputed θ_i are actually n bits stored in a register we can build a more accurate hardware structure:

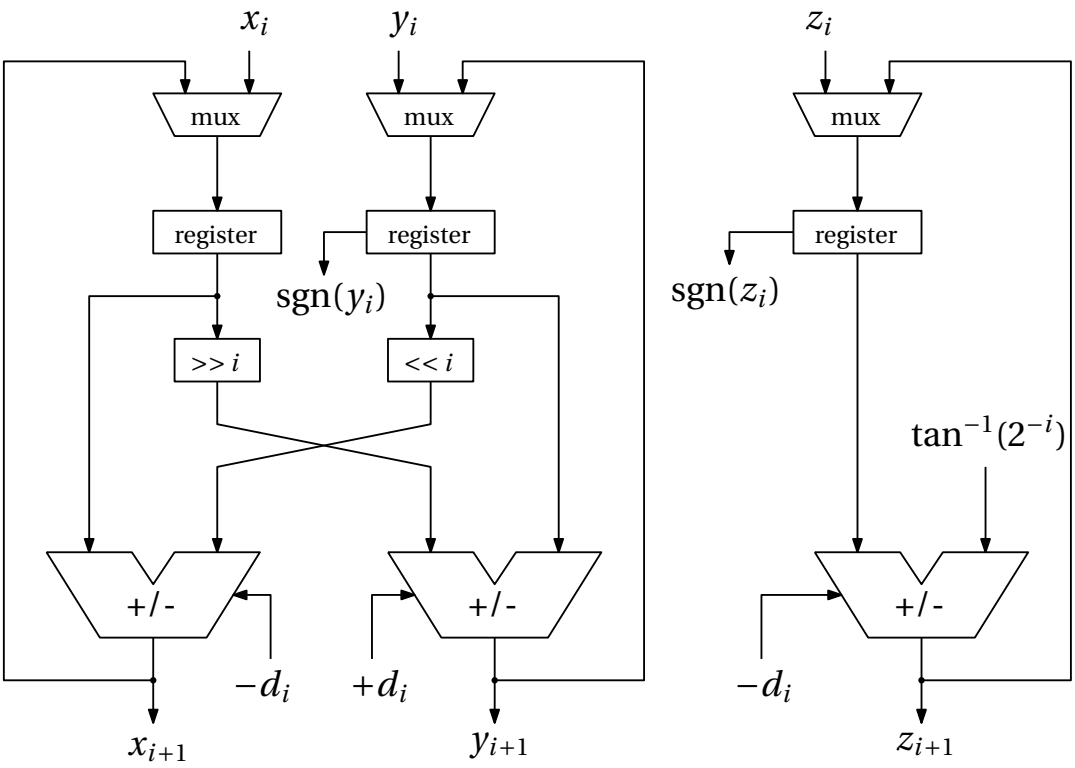


Figure 2.2: *Hardware implementation of a CORDIC iteration with circular coordinates*

The decision coefficient d_i that commands the adder/subtractor can be extracted directly from the sign bits $\mathrm{sgn}(y_i)$ or $\mathrm{sgn}(z_i)$ depending on which mode the CORDIC algorithm is operating, VM and RM respectively.
Each i-th iteration will take place in one single clock cycle (considering a clock cycle slow enough for the circuit to reach the desired state), the output will then get transferred to the input through a multiplexer, shifted by the desired shift depending on the i-th iteration and then addiction/subtraction to compute the output vector.

2.2 Reconfigurable CORDIC architecture

Taking the generalized CORDIC iteration step parametrized by m depending on the coordinates system we want to work with:

$$\begin{aligned} x_{i+1} &= x_i - m \cdot y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} &= y_i + y_i \cdot d_i \cdot 2^{-i} \\ z_{i+1} &= z_i - d_i \cdot \alpha_i \end{aligned} \tag{2.2}$$

We highlight the pseudo-rotation of the i-th iteration matrix $\mathbf{R_i}$ that relates the input-

output vectors:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & -m \cdot d_i \cdot 2^{-i} \\ d_i \cdot 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \mathbf{R_i} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{2.3}$$

In order to design a good reconfigurable architecture we need to maximize the sharing of common hardware; we note that the iteration for circular and hyperbolic coordinates system only differs by a sign factor in the pseudo-rotation matrix. So, we can repurpose the basic CORDIC iteration hardware implementation with select signals sel_y and sel_x [6] in function of the sign bits $\text{sgn}(y_i)$, $\text{sgn}(z_i)$ and parameter m; these select signals will then command the adders/subtractors:

Table 2.1: *Determination of sel_y and sel_x for reconfigurable CORDIC*

Trajectory	m	d_i	sel_x	sel_y
Circular	1	1	Sub	Add
		-1	Add	Sub
Hyperbolic	-1	1	Add	Add
		-1	Sub	Sub

In the proposed hardware implementation a new signal t is defined in order to choose the trajectory of operation, $t = 1$ for circular coordinates system and $t = 0$ for hyperbolic coordinates system. The *mode* signal is set to $mode = 1$ for vectoring mode and $mode = 0$ for rotation mode.

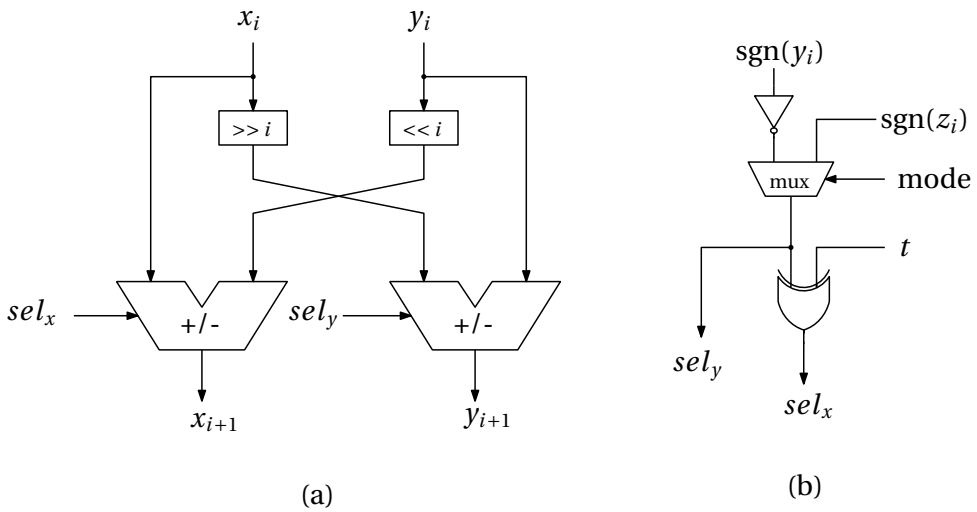


Figure 2.3: *(a) Reconfigurable coordinate computation unit. (b) Select signals generation*

The angle computation does not depend on the type of trajectory, but still depends on the mode. We need at least 2 look-up tables with the pre-computed set of angles (α_{hi} hyperbolic angles and α_{ci} circular angles) for the computation of z_i.

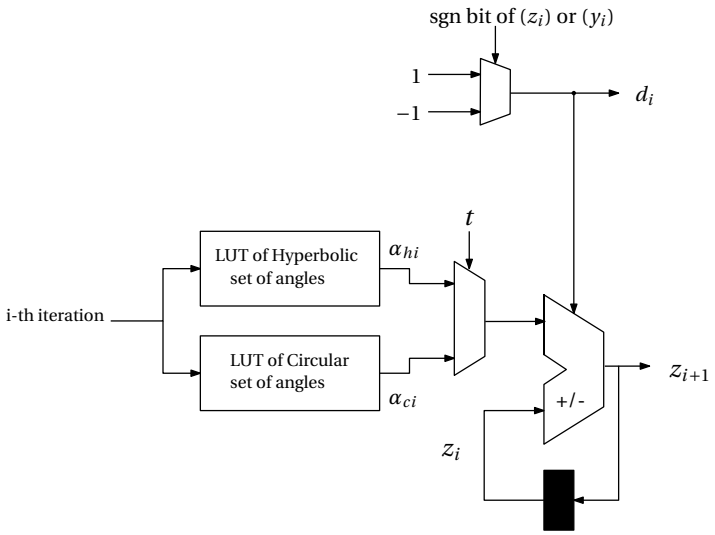


Figure 2.4: *Reconfigurable angle computation unit*

We can now build the structure of a single i-th iteration for the generalized CORDIC algorithm. Note that the sel_y corresponds to the d_i with the *mode* selected.

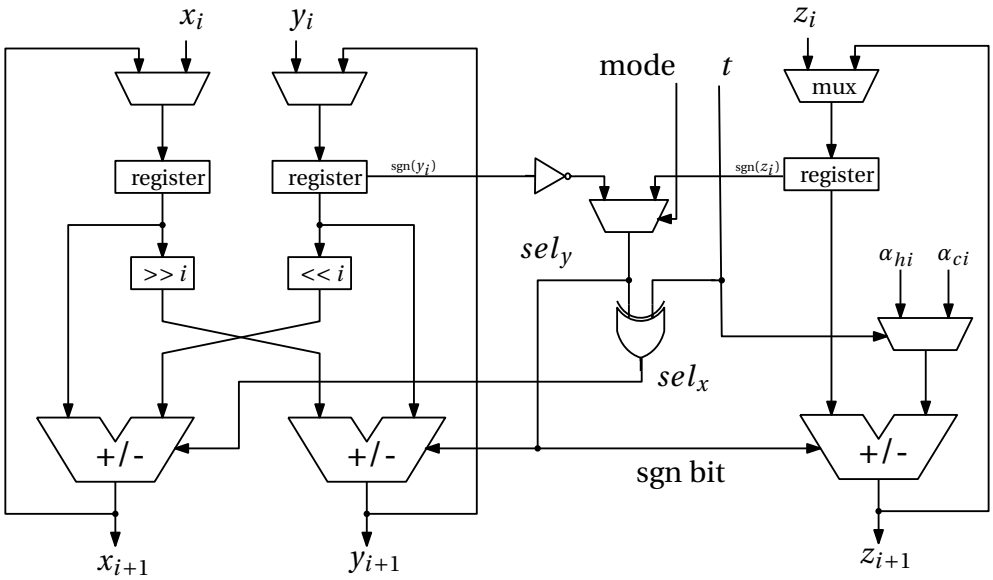


Figure 2.5: *Hardware implementation of a single iteration generalized CORDIC algorithm*

2.2.1 Pipelined CORDIC architecture

We can exploit the fact that all CORDIC iterations are identical and map the datapath with n iterations all in cascade in order to gain throughput at the cost of a higher circuit complexity. The proposed solution of Figure 2.6 is simply an n stages of single CORDIC iteration structure chained together.

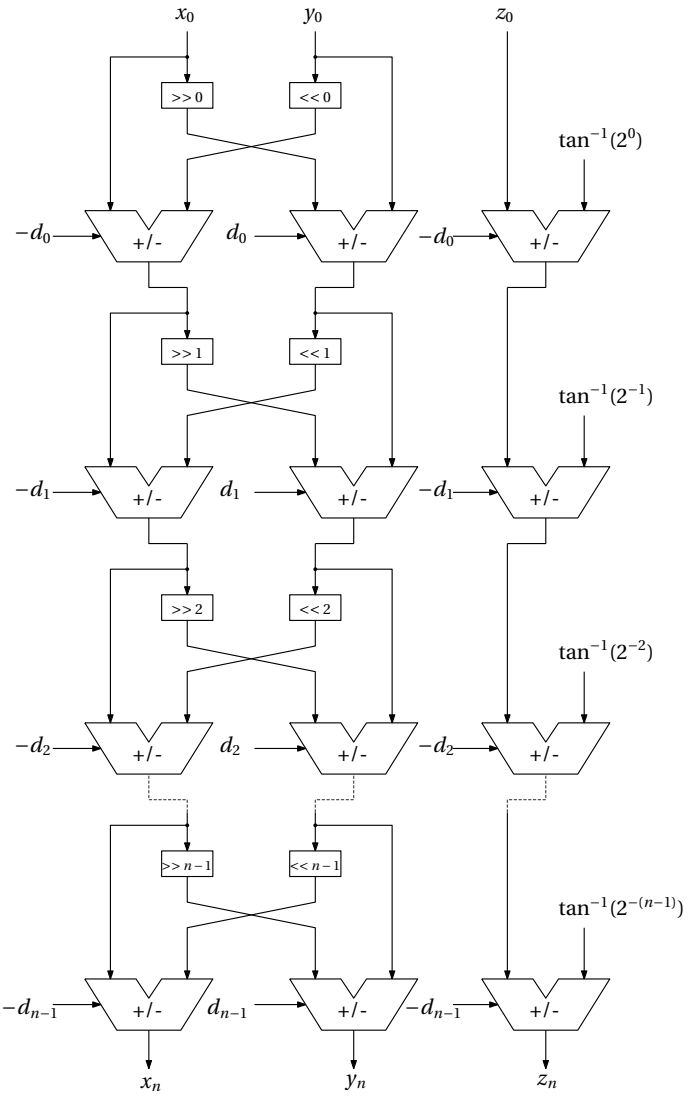


Figure 2.6: *Hardware implementation of pipelined CORDIC*

To further optimize the hardware cost we can note that the shifters at each stage always shifts a fixed amount depending on the i-th stage. The shift operation could be hardwired with adders thus removing the need of shift structures altogether.

The pipelined CORDIC architecture can be considered a fully combinational logic circuit, in order to properly function the clock period should allow enough time at each stage for the 3 adders to complete its addiction/subtraction operation. The critical-path that define the optiman $\mathrm{T_{CLK}}$ amounts to:

$$\mathrm{T_{CLK}} \simeq \mathrm{T_{ADD}} + \mathrm{T_{MUX}} + \mathrm{T_{2C}} \tag{2.4}$$

Where T_{ADD}, T_{MUX} and T_{2C} are the time required for addiction, 2:1 multiplexing and 2's complement operation respectively. The multiplexing time is due to the extra computation of the decision coefficient at each stage d_i. For known and constant angle rotations the sign could be predetermined, removing the need of multiplexing and reducing the critical-path.

The generalized CORDIC algorithm hardware implementation can also be pipelined, as shown in the figure 2.7 .

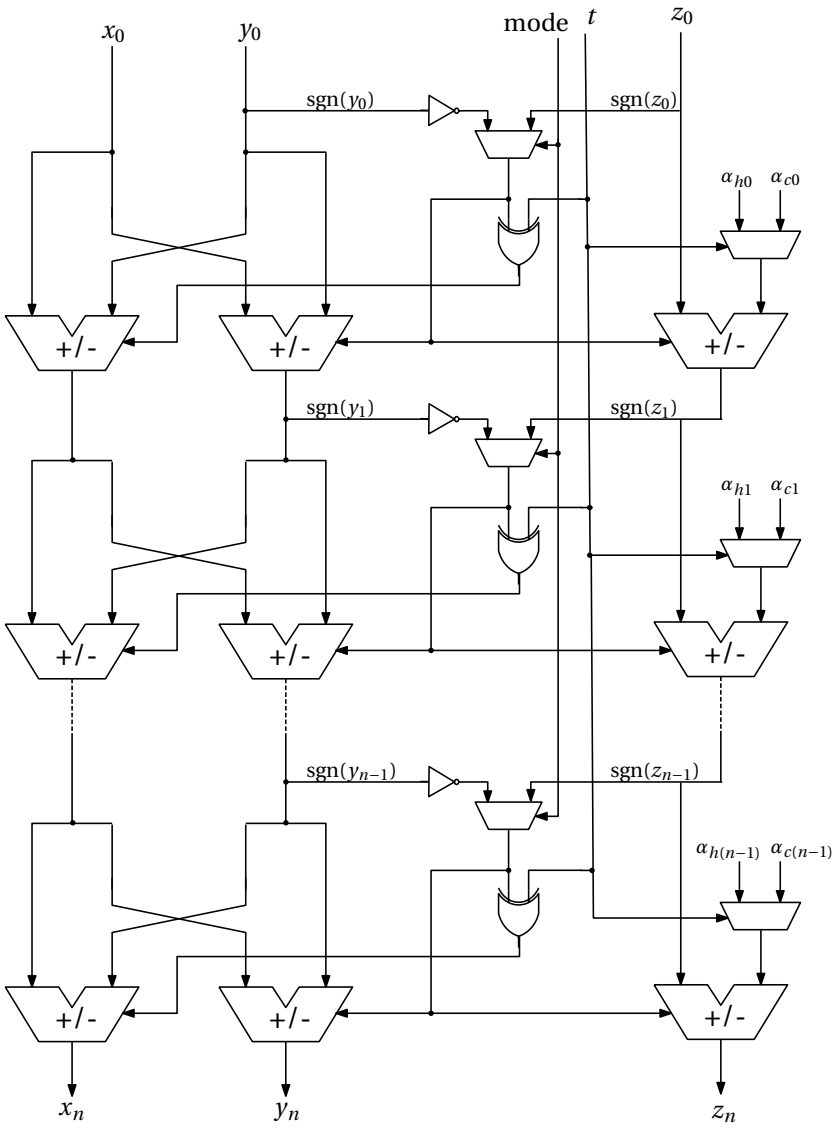


Figure 2.7: *Pipelined generalized reconfigurable CORDIC*

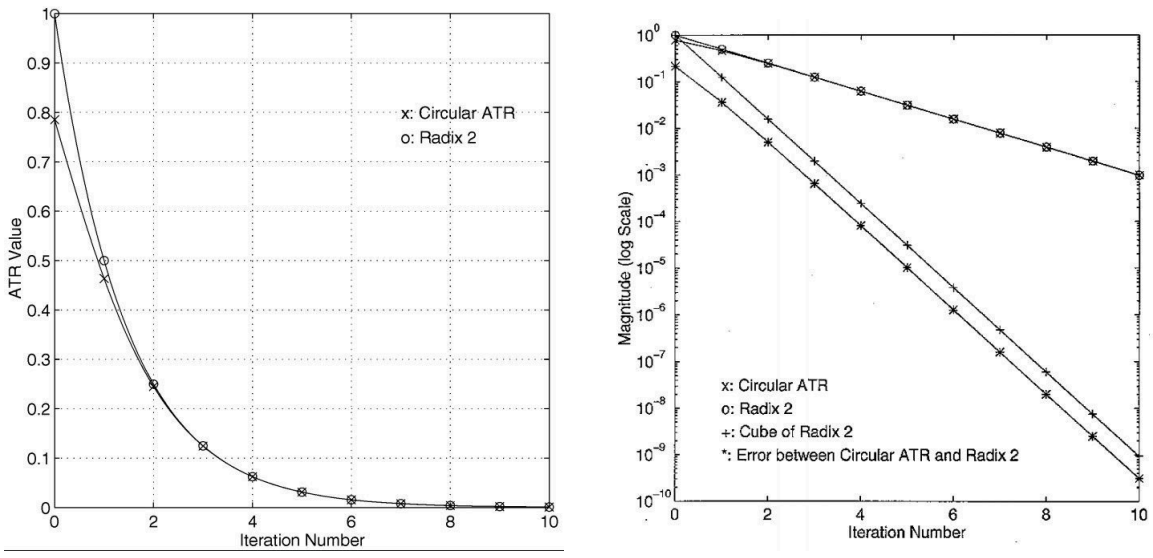
2.3 Hybrid CORDIC algorithm

The main bottleneck of the CORDIC algorithm is regarding the limited throughput caused by its sequential nature since computing the i-th decision coefficient d_i requires the (i-1)-th iteration step to be completed. In fact, knowing the decision coefficient beforehand the CORDIC algorithm could be implemented with purely combinational logic circuits [7]. The Hybrid CORDIC algorithm is a special configuration that can effectively derive 2/3 of the rotations in parallel without any error by the introduction of two arctangent radices [1].

2.3.1 Hybrid Radix Sets

In the conventional CORDIC the angle is decomposed by a set of angles that satisfy the condition $\theta_i = \tan^{-1}(2^{-i})$, this exploits the base-2 product operation of the pseudorotation matrix $\mathbf{R_c}$ and converting it into a shifting operation, discussed in the section 1.2.
When the i-th microroation angle is small enough:

$$\lim_{k\to 0} \frac{\tan^{-1}(2^{-k})}{(2^{-k})} = 1$$
$$\text{Therefore } \tan^{-1}(2^{-i}) \simeq (2^{-i}) \text{ with sufficiently large } i \tag{2.5}$$



(a) **(b)**

Figure 2.8: *(a) Radix-2 and circular ATR (classic CORDIC) computed values. (b) Computation of error between Radix-2 and circular ATR. Adapted from [1]*

What's the gain of having the angles in a radix-2 form? From the generalized CORDIC algorithm introduces in the section 1.2.3 , an angle decomposition with $\alpha_i = 2^{-i}$ is essentially the algorithm working with linear coordinates:

$$\theta = \sum_{i=0}^{n} d_i \alpha_i = 2^{-i} \qquad \text{with} \qquad d_i = \pm 1 \tag{2.6}$$

Note that we are working in the rotation mode, so our final goal is to decrease z_0 to zero. We can then rewrite the equation 2.7 considering that the angles are actually a binary sequence.

$$\theta = \sum_{i=0}^{n} z_i \alpha_i = 2^{-i} \qquad \text{with} \qquad z_i = 1, 0 \tag{2.7}$$

Therefore, having the z_0 as the input in a linear coordinates system means having all the d_i simultaneously. We introduce two typical approaches of the Hybrid algorithm angle decomposition, the Mixed-Hybrid and the Partitioned-Hybrid radix sets.

Mixed-Hybrid Circular ATR

The rotation angle θ is partitioned as:

$$\{\overbrace{\tan^{-1}(2^{-0}), \tan^{-1}(2^{-1}), \ldots, \tan^{-1}(2^{-n+1})}^{\text{most significant part}}, \underbrace{2^{-n}, \ldots, 2^{-N+1}}_{\text{least significant part}}\} \tag{2.8}$$

To increase the performance, the computation of the most significant part must be completely separated from the least significant one while preserving full accuracy [1]. We now partition the θ angle with a binary representation of $N-1$ bits:

$$\theta = \sum_{i=0}^{N-1} \theta_i 2^{-i} \tag{2.9}$$

The θ angle has been effectively decomposed into two terms θ_H and θ_L, which can be also referred as coarse and fine angular decomposition:

$$\theta_H = \sum_{i=0}^{n-1} \theta_i 2^{-i} \qquad \theta_L = \sum_{i=n}^{N-1} \theta_i 2^{-i} \tag{2.10}$$

The angle θ_H is completely independent from θ_L only if it does not change the $N-n$ significant bits of the residue rotation angle with respect to θ_H. In order to achieve this, the exact rotation of θ_H is independent from θ_L.

The first processor will use the full angle θ and generate x_n, y_n and z_n at the end of n

iterations. The second processor will compute starting from the vector (x_n, y_n) and z_n to generate (x_N, y_N) and z_N (with all the decision coefficients d_i known) at the end of the remaining $N - n$ iterations.

It has been demonstrated that $n \simeq N/3$ iterations must be computed in a strictly sequential way in order to preserve full accuracy [1]. Note that the two processors work differently, the processor I is performs conventional CORDIC operations and the processor II can perform an optimized version of CORDIC even without the look-up table or ROM since there are no precomputed angles that needs to be stored, only a shift-add processor is enough.

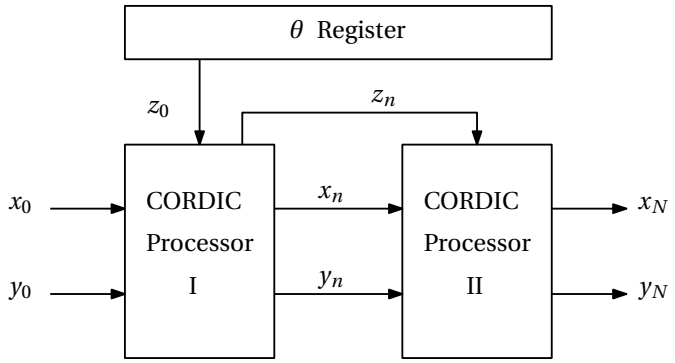


Figure 2.9: *Architecture for Mixed-Hybrid CORDIC algorithm*

The d_i prediction in the linear coordinates system can also be applied in circular and hyperbolic coordinates with the presence of a correction term. The maximum prediction error is proven to be limited [8] . Consequently, both coarse and fine rotations can then be implemented without ROM look-up tables, with the appropriate modification of the CORDIC algorithm structure.

Partitioned-Hybrid Circular ATR

In order to speed up the Hybrid-CORDIC even further we assume that θ_H rotation is performed in a single rotation; the rotation angle θ is now partitioned as:

$$\{\overbrace{\tan^{-1}(2^{-n+1})}^{\text{most significant part}}, \underbrace{2^{-n}, \ldots, 2^{-N+1}}_{\text{least significant part}}\} \tag{2.11}$$

The computation of the θ_H rotation can't be implemented as an ordinary CORDIC iteration since more than one angle is actually taken into account [1]. Processor I performs a pair of ROM look-up operations followed by addiction to realize the rotation of the coarse angle.

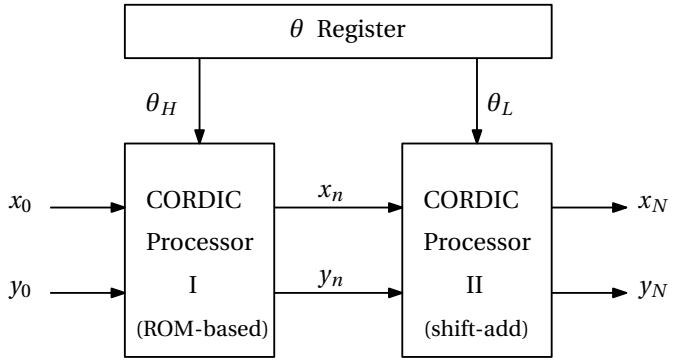


Figure 2.10: *Architecture for Partitioned-Hybrid CORDIC algorithm*

A form of Hybrid-CORDIC decomposition using a ROM based coarse operation has been proposed in [9] for a very high-precision CORDIC processor with a contained ROM size, namely the P-CORDIC.

2.4 Scaling, quantization and accuracy issues

As previously discussed, the CORDIC algorithm is powerful because the K factor converges to a finite number for large enough n iterations. Note that after 9 iterations we get pretty close to the converged value of $K = 1.6467605$.

A naive student may note that we still need at least one single multiplication at the end of the algorithm which defeats the purpose of its simplicity. Since we are scaling by a constant factor, there are efficiently implemented scaling units designed by canical signed digit (CSD)-based technique [10] and common sub-expression elimination (CSE) approach [11] [12]. Furthermore, the CORDIC algorithm has also been modified to implement online scaling [13] and even scaling-free CORDIC [14].

Table 2.2: *K_n in function of n finite iterations*

n	K_n
1	1.414213
2	1.581139
3	1.629801
4	1.645689
5	1.646492
6	1.646693
7	1.646744
8	1.646756
9	1.646759

Since we only have a fixed amount of n CORDIC iterations, and the output of the algorithm is only an approximation of the ideal value. Various papers have shown that the error are within acceptable ranges, with potential optimization through normalization of the input vector [15].

With the introduction of the pipelined CORDIC we can clearly see that there is some kind of trade-offs between throughput and hardware complexity. The trade-off for area, accuracy and latency of the algorithm depend mainly on the iteration count and its implementation [16]. In fact, to achieve an n-bit accuracy with fixed-point arithmetic, a wordlength of x and y datapath of $(n + 2 + \log 2(n))$ and for the angle z a $(n + \log 2(n))$ is required. [17]

Chapter 3

Applications of the CORDIC algorithm

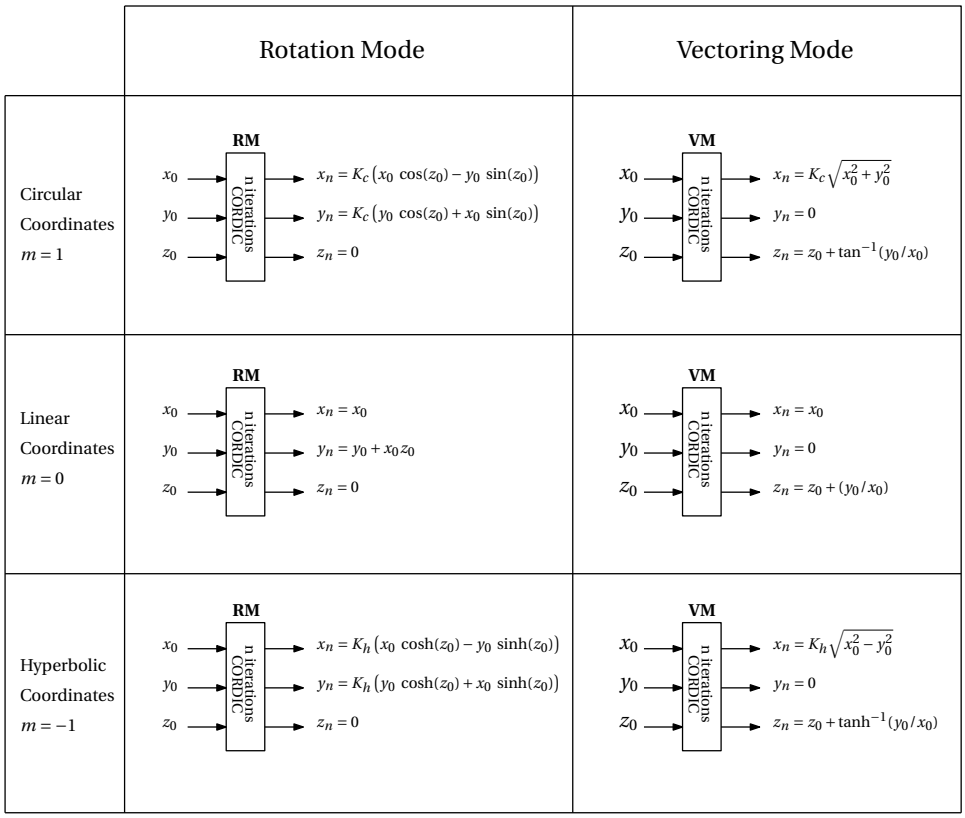


Figure 3.1: *Summary of CORDIC functions*

3.1 Computation of functions

An immediate application of the generalized CORDIC algorithm is the computation of various functions; some functions can be directly extracted from the output after n iterations while others require some post-processing computation.

3.1.1 Rotation Mode

Using the circular coordinates (CC) system CORDIC algorithm with the input vector $(1, 0)$ and angle θ we get the following output:

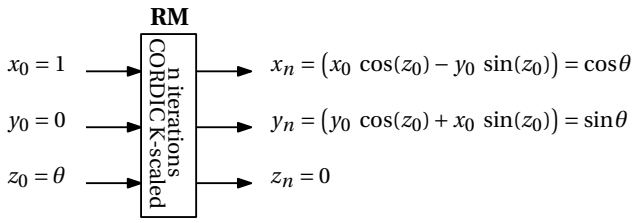


Figure 3.2: *CC-RM Cordic to compute* $\sin\theta$, $\cos\theta$ *and* $\tan\theta$

It directly computes $\sin\theta$ and $\cos\theta$; $\tan\theta$ can then be derived.

We can also do a polar to rectangular conversion operation by inputting $(R, 0)$ and the output vector will be $(R\cos\theta, R\sin\theta)$. Note that if the value of R is too big or too small the output vector error increases [15].

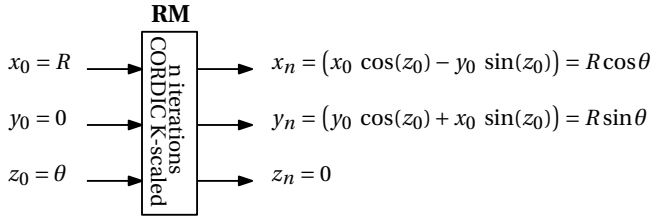


Figure 3.3: *CC-RM Cordic to compute* $\sin\theta$, $\cos\theta$ *and* $\tan\theta$

Using the hyperbolic coordinates (HC) system, with an input of $(1, 0)$ and $z_n = \theta$ we can directly compute $\sinh\theta$ and $\cosh\theta$; $\tanh\theta = (\sinh\theta / \cosh\theta)$ and $\exp(\theta) = (\sinh\theta + \cosh\theta)$ can be subsequently computed.

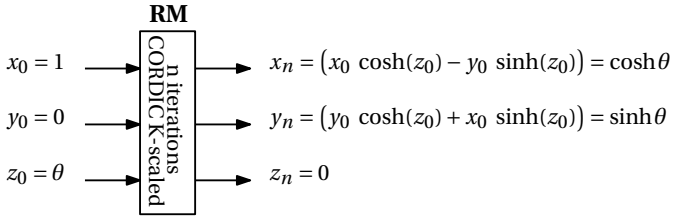


Figure 3.4: *HC-RM Cordic to compute* $\sinh\theta$, $\cosh\theta$

3.1.2 Vectoring Mode

Using the CC-CORDIC in vectoring mode with the input vector $(1, a)$ and $z_n = 0$ we can get at the output $\tan^{-1}(a)$.

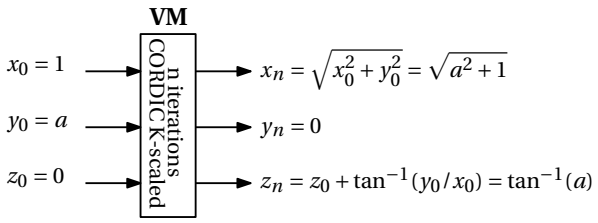


Figure 3.5: *CC-VM Cordic to compute* $\tan^{-1}(a)$

With CC-RM we computed polar-to-rectangular conversion; using CC-VM we can do the opposite conversion by inputting (b, a) and $z_0 = 0$ and the output $x_n = \sqrt{a^2 + b^2}$ and $\tan^{-1}(a/b)$.

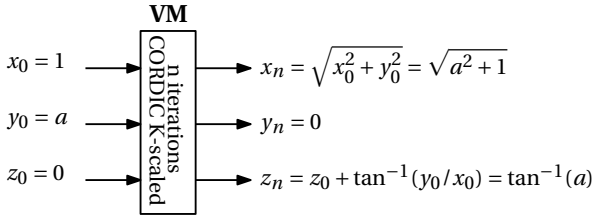


Figure 3.6: *CC-VM Cordic to compute* $\tan^{-1}(a/b)$

With HC-CORDIC in vectoring mode we can compute the square root function and natural logarithm with the input $(a + 1/4, a - 1/4)$ and $z_0 = 0$ we can get at the output $x_n = \sqrt{a}$. The computation of $\ln(a)$ can be done through this mathematical identity:

$$\ln a = 2 \tanh^{-} \left| \frac{a-1}{a+1} \right| \tag{3.1}$$

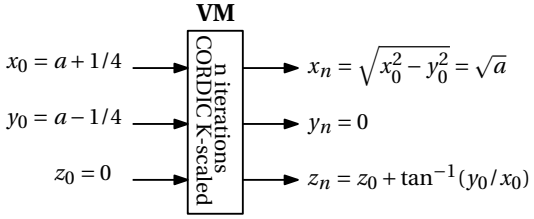


Figure 3.7: *HC-VM Cordic to compute* $\sqrt{a}$

3.1.3 Multiplication and division using LC-CORDIC

Using the linear coordinates (LC) system CORDIC algorithm multiplications and division could also be computed.

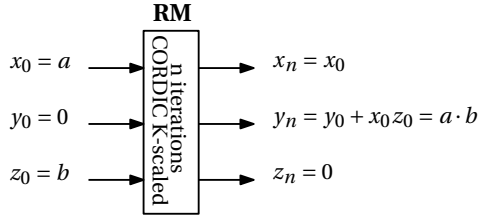


Figure 3.8: *LC-RM Cordic to compute $a \cdot b$*

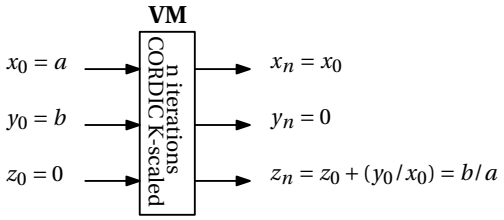


Figure 3.9: *LC-VM Cordic to compute b/a*

3.2 Applications to communication

In digital communication systems, the RM-CORDIC can be used to generate mixed signals removing the need of big ROM look-up tables that store sine/cosine values while the VM-CORDIC can be used to estimate phase and frequency parameters.

3.2.1 Direct Digital Synthesis (DDS)

The Direct Digital Synthesis is a technique used in digital communications to generate precise and stable waveforms starting from a known clock frequency f_C. The generation of any arbitrary wave requires the following components:

- A digital phase accumulator, it increments at a constant number depending on the required frequency of the wave we need to generate and the clock frequency; the output of the accumulator can be seen as a discrete sawtooth waveform which represents the phase change of the wave.
- A phase-to-amplitude converter, with the input of the phase of the wave it outputs its corresponding amplitude.
- A digital-to-analog converter (DAC), converts the outpt of the previous block to a signal in the analog domain.

Usually the DDS is employed for the generation of sine/cosine waves and the phase-to-amplitude converter is basically a ROM look-up table with all the amplitudes of the sinusoid mapped accordingly.

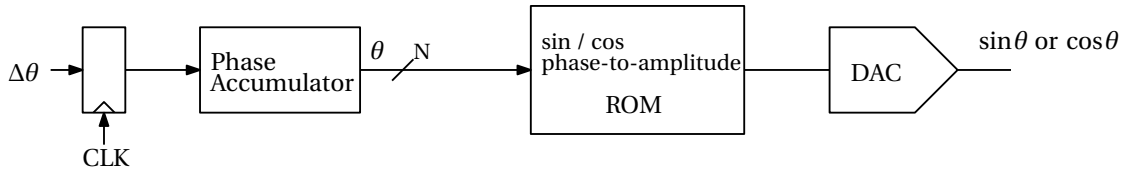


Figure 3.10: *Simplified block diagram of the direct digital synthesizer for sine/cosine waves.*

The system of the figure 3.10 represents the main blocks for the DDS. The first two blocks are also referred as a numerically controlled oscillator (NCO) since the output of the phase-to-amplitude converter is a sine/cosine wave at a controlled frequency f_{out} in function of f_{CLK} and $\Delta\theta$ given by :

$$f_{out} = \frac{f_{CLK} \cdot \Delta\theta}{2^N} \tag{3.2}$$

$\Delta\theta$ is the phase increment value and N is the length in bits of the accumulator. Observe that with increased step size of $\Delta\theta$, the frequency of the discrete sawtooth frequency increases since looping through the phase accumulator register requires less time.

The phase to amplitude converter could be switched to a CC-CORDIC in RM (Figure 3.3) in order to compute the sine value $\sin\theta$ for an input θ value. Unfortunately, despite the CORDIC structure being more area effective than the ROM look-up table the trade-offs are the lower latency and the introduction of arithmetic circuitry[1] [18].

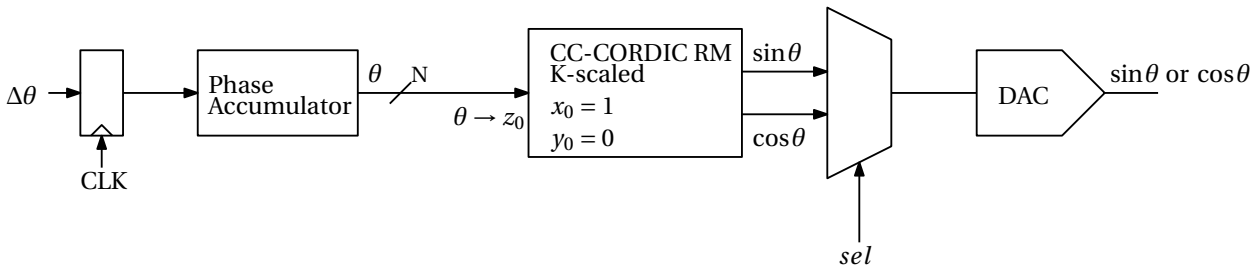


Figure 3.11: *Simplified block diagram of the DDS using the CC-CORDIC in RM*

[1]Depending on the specific CORDIC algorithm implementation further trade-offs can be observed, discussed in the section 2.4

Additionally, we can exploit the periodic nature of sinudoidal waves in particular by studying the 3 most significant bits (MSB) of the θ angle in order to determine which $\pi/2$ quadrant it corresponds, in fact the amplitude values of the sine of the other 3 quadrants are simply the first quadrant mirrored/flipped. The remaining $N-3$ are then inputted in the CC-CORDIC in RM. A control circuit is required in order to get the correct $\sin\theta$ and $\cos\theta$ values. Additionally we can get a better sine resolution or better area usage using the same CC-CORDIC since the input bits are fewer.

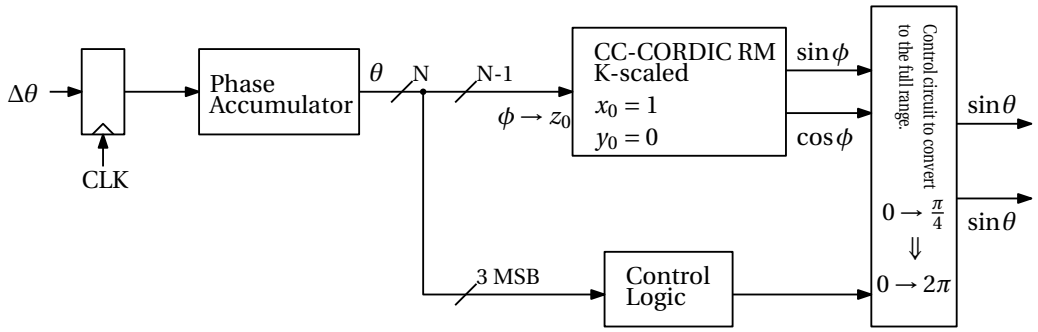


Figure 3.12: *Optimized diagram of the DDS using the CC-CORDIC in RM and additional control circuitry*

3.2.2 Analog and digital modulation

The CORDIC in RM can also be implemented for digital modulation, for instance in the computation of the in-phase and quadrature components.

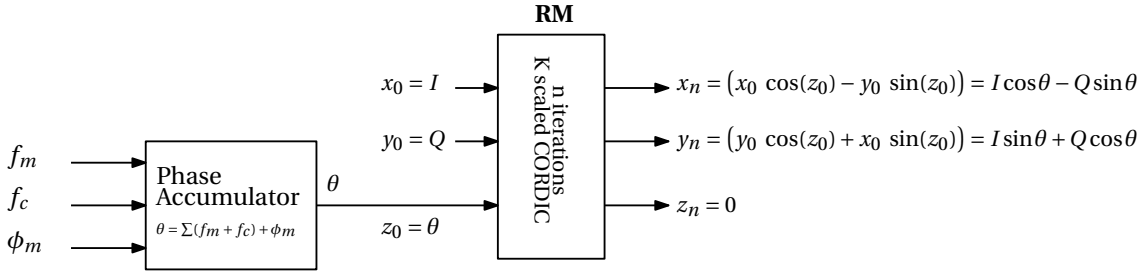


Figure 3.13: *Generic scheme to use CORDIC in RM for digital modulation. Adapted from [2]*

The figure 3.13 is a modified version of the block diagram in figure 3.10 with more freedom in parameters. $\theta = \sum(f_m + f_c) + \phi_m$ depends on the normalized carrier and modulating frequencies, f_c and f_m respectively. The modulating phase ϕ_m is also inputted in the phase accumulator. This generic scheme could be used for analog amplitude modulation (AM), phase modulation (PM), and frequency modulation (FM), as well as the digital modulations such as amplitude key shifting (ASK), phase-shift keying (PSK) and frequency-shift keying (FSK) modulators [2].

Chapter 4

Conclusion

This academic work is merely an introduction to the intricate world surrounding the CORDIC algorithm. This powerful computation technique has proven to be extremely versatile in a wide range of fields; while this thesis only scratches the surface about the applications to communications, CORDIC can also be applied in signal and image processing, matrix computations, robotics and graphics. Despite the simplicity of the CORDIC implementation initially proposed by J.E. Volder [3] and its generalized form by J.S. Walther [4], more than 60 years have passed. Significant modification to the original structure and subsequent refinements increased the throughput capabilities of the CORDIC processor while simultaneously maintaining optimal hardware resource consumption.

Bibliography

[1] S. Wang, V. Piuri, and E. Wartzlander, "Hybrid cordic algorithms," *IEEE Transactions on Computers*, vol. 46, no. 11, pp. 1202–1207, 1997.

[2] P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of cordic: Algorithms, architectures, and applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, 2009.

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

[4] J. S. Walther, "A unified algorithm for elementary functions," in *Proceedings of the May 18-20, 1971, Spring Joint Computer Conference*, ser. AFIPS '71 (Spring). New York, NY, USA: Association for Computing Machinery, 1971, p. 379–385. [Online]. Available: https://doi.org/10.1145/1478786.1478840

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

[6] S. Aggarwal and P. K. Meher, "Reconfigurable cordic architectures for multi-mode and multi-trajectory operations," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 2490–2494.

[7] B. Hosticka, D. Timmermann, and H. Hahn, "Low latency time cordic algorithms," *IEEE Transactions on Computers*, vol. 41, no. 08, pp. 1010–1015, aug 1992.

[8] C.-Y. Chen and W.-C. Liu, "Architecture for cordic algorithm realization without rom lookup tables," in *2003 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, 2003, pp. IV–IV.

[9] K. Parhi and K. Martin, "P-cordic: a precomputation based rotation cordic algorithm," *EURASIP Journal on Advances in Signal Processing*, vol. 2002, 09 2002.

[10] R. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 10, pp. 677–688, 1996.

[11] G. Gilbert, D. Al-Khalili, and C. Rozon, "Optimized distributed processing of scaling factor in cordic," in *The 3rd International IEEE-NEWCAS Conference, 2005.*, 2005, pp. 35–38.

[12] O. Gustafsson, A. Dempster, K. Johansson, M. Macleod, and L. Wanhammar, "Simplified design of constant coefficient multipliers," *Circuits Systems and Signal Processing*, vol. 25, pp. 225–251, 01 2006.

[13] H. K. Samudrala, S. Qadeer, S. Azeemuddin, and Z. Khan, "Parallel and pipelined vlsi implementation of the new radix-2 dit fft algorithm," in *2018 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS)*, 2018, pp. 21–26.

[14] J. Villalba, T. Lang, and E. Zapata, "Parallel compensation of scale factor for the cordic algorithm," *Journal of VLSI Signal Processing*, vol. 19, pp. 227–241, 08 1998.

[15] E. Antelo, J. Bruguera, T. Lang, and E. Zapata, "Error analysis and reduction for angle calculation using the cordic algorithm," *IEEE Transactions on Computers*, vol. 46, no. 11, pp. 1264–1271, 1997.

[16] K. Kota and J. Cavallaro, "Numerical accuracy and hardware tradeoffs for cordic arithmetic for special-purpose processors," *IEEE Transactions on Computers*, vol. 42, no. 7, pp. 769–779, 1993.

[17] 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.

[18] D. De Caro and A. Strollo, "High-performance direct digital frequency synthesizers using piecewise-polynomial approximation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 2, pp. 324–337, 2005.