

CORDIC Rotation README

Peimin Chi

Block Functionality

The CORDIC Rotation block takes real and imaginary components of a complex number and an angle in the range of $-\pi$ and π , rotates the input vector according to the input angle. (Counterclockwise angles are positive.)

Block Interface

Input Word Length: Total number of bits of inputs. (X and Y are assumed to have the same format.)

Binary Point: Number of fraction bits for inputs.

Input Angle Word Length: Total number of bits of input angle

Input Angle Binary Point: Number of fraction bits for input angle

Number of stages: User can choose number of rotation stages.

Scaling Factor

Since CORDIC does pseudo-rotation instead of ideal rotation, the resulting X and Y components are scaled by a constant factor. The factor is

$$K = \prod_{i=1}^{N_{\text{stage}}} \sqrt{1 + \tan^2(\text{angle}_i)} = \prod_{i=1}^{N_{\text{stage}}} \sqrt{1 + 2^{2(1-i)}}$$

For the available choices of stages, the factors are listed in Table 1.

Stages	Scaling Factor
2	1.5811
3	1.6298
4	1.6425
5	1.6457
6	1.6465
7	1.6467

8	1.6467
---	--------

Table 1 Scaling Factors

Theory of Operations

If we have a vector $x + jy = Ae^{j\alpha}$ in the complex plane, where

$$A = \sqrt{x^2 + y^2}$$

$$\alpha = \tan^{-1} \frac{y}{x}$$

and we want to rotate the vector by an angle β , therefore the resultant vector would be

$Ae^{j(\alpha+\beta)}$ and in rectangular coordinates, we have

$$\begin{aligned} x' &= A(\cos \alpha \cos \beta - \sin \alpha \sin \beta) \\ &= x \cos \beta - y \sin \beta \\ y' &= A(\sin \alpha \cos \beta + \cos \alpha \sin \beta) \\ &= y \cos \beta + x \sin \beta \end{aligned} \quad (1)$$

With trigonometry identities and some manipulation, we can rewrite (1) as

$$\begin{aligned} x' &= \frac{1}{\sqrt{1 + \tan^2 \beta}} (x - y \tan \beta) \\ y' &= \frac{1}{\sqrt{1 + \tan^2 \beta}} (y + x \tan \beta) \end{aligned} \quad (2)$$

If we ignore the factor $\frac{1}{\sqrt{1 + \tan^2 \beta}}$, then the new X and Y coordinates can be put into a

simple form:

$$\begin{aligned} x_{new} &= x - y \tan \beta \\ y_{new} &= y + x \tan \beta \end{aligned} \quad (3)$$

and the amplitude of the resultant vector will be bigger than that of the original vector by

a factor of $\sqrt{1 + \tan^2 \beta}$. Therefore, we have done pseudo-rotation rather than ideal

rotation as shown in Figure 1.

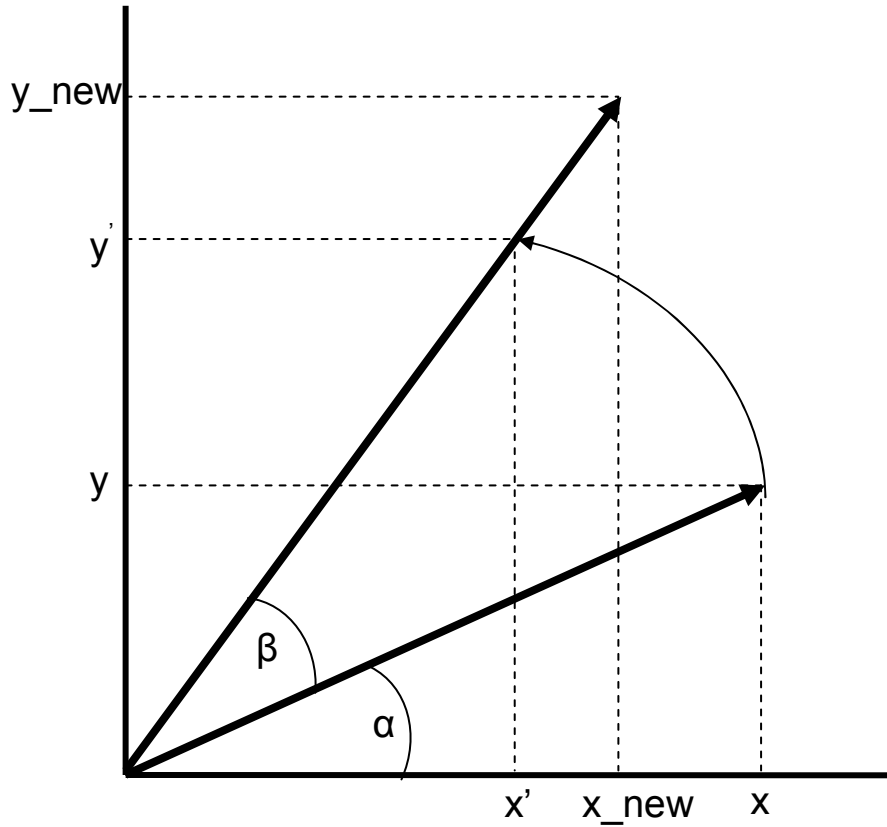


Figure 1 Pseudo-rotation and Rotation of a Vector

Choice of Angles and Iterative Rotation

From (3), it is clear that in order to calculate the x_{new} and y_{new} , it is still necessary to multiply $\tan \beta$. To make implementation easier, we can choose special angles such that their tangents are powers of 2 and approximate the angle β as sum (or difference) of such angles. This implies an iterative method to rotate an angle β . Table 2 lists the special angles for the first 8 iterations.

Iteration i	Angle _i	tan(angle _i) = 2 ¹⁻ⁱ
1	45°	2 ⁰
2	26.6°	2 ⁻¹
3	14.0°	2 ⁻²
4	7.1°	2 ⁻³
5	3.6°	2 ⁻⁴
6	1.8°	2 ⁻⁵
7	0.9°	2 ⁻⁶
8	0.4°	2 ⁻⁷

Table 2 Iterative Angles for CORDIC Rotation

Any angles between 99.7° and -99.7° can be represented as a sum or difference of the angles in Table 2. For example, 70° can be approximated as:

$$45^\circ + 26.6^\circ - 14.0^\circ + 7.1^\circ + 3.6^\circ + 1.8^\circ - 0.9^\circ + 0.4^\circ + 0.2^\circ + 0.1^\circ \approx 70^\circ$$

Now we can use the iterative algorithm to rotate a vector by any angle between 99.7° and -99.7°.

```

Initialize x0 to be the x-coordinate of original vector
Initialize y0 to be the y-coordinate of original vector
Initialize z0 to be the desired angle of rotation
for i = 1: Number of Stages
    di = sign(zi-1)
    xi = xi-1 - yi-1 * (di * 21-i)
    yi = yi-1 + xi-1 * (di * 21-i)
    zi = zi-1 - di * anglei
end

```

(4.6)

If the desired angle of rotation is more than 99.7° and less than 180°, then we just simply modify the initialization process.

```

Initialize x0 to be the NEGATIVE x-coordinate of original vector
Initialize y0 to be the NEGATIVE y-coordinate of original vector
Initialize z0 = desired angle of rotation - 180°

```

Similarly, for rotation angles of less than -99.7° and more than -180°, we can do

Initialize x_0 to be the NEGATIVE x-coordinate of original vector
 Initialize y_0 to be the NEGATIVE y-coordinate of original vector
 Initialize $z_0 = \text{desired angle of rotation} + 180^\circ$

After all the iterations, z will be forced to almost zero. Since each of the iterations performs a pseudo-rotation rather than an ideal rotation, after all the iterations, x and y will be coordinates of the ideal results multiplied by a factor. If the total number of iterations is N_{stage} , then the amplifying factor is

$$K = \prod_{i=1}^{N_{\text{stage}}} \sqrt{1 + \tan^2(\text{angle}_i)} = \prod_{i=1}^{N_{\text{stage}}} \sqrt{1 + 2^{2(1-i)}} . \text{ For large number of stages, this factor}$$

K converges to 1.6467.

Shared Implementation

Since all CORDIC stages are almost identical, we can also upsample the input signal by the number of stages used and require the hardware to finish one stage of operation per cycle. Three registers are required at the end for intermediate output to be latched back and three muxes are needed to pass either the input or the intermediate output to the shared stage as shown in Figure 2.

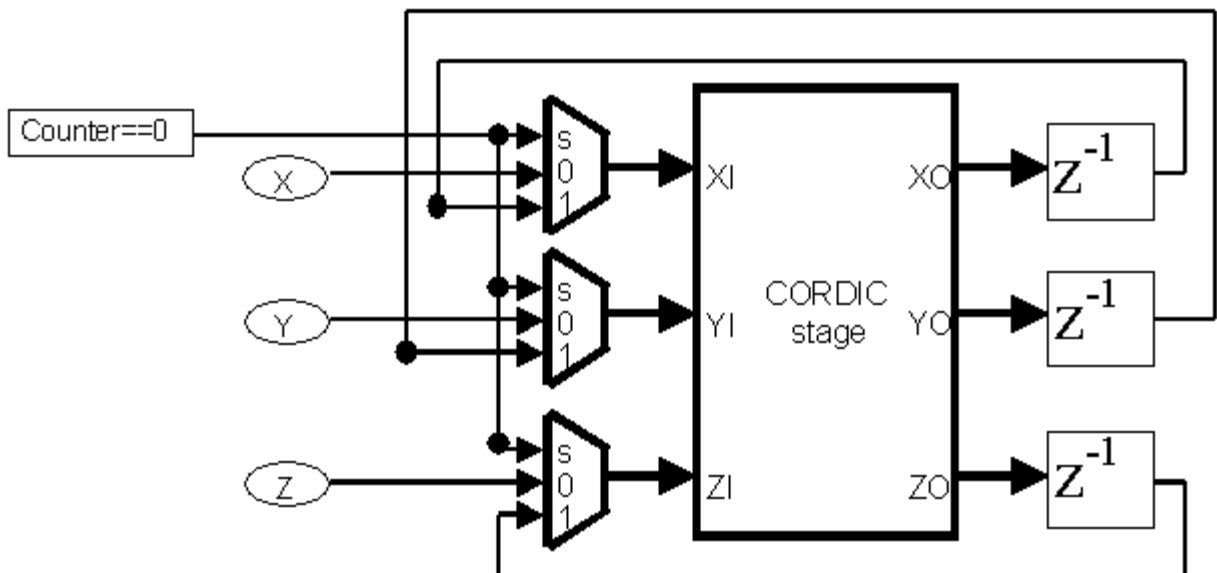


Figure 2 Shared Implementation of CORDIC

Shared implementation will result hardware savings for large number of stages since the number of adders and negates no longer increases linearly with the number of stages and the only incremental costs are the registers and muxes.