

## Appendix III (extracted from Peimin Chi's MS thesis)

### CORDIC

Due to frequency offset, the maximum correlation value is  $Ne^{j2\pi\Delta f T}$  when there is no noise, as shown in (2.11). To estimate the frequency offset, we need to estimate the phase of the max correlation value based on its real and imaginary components and this is done by the CORDIC module that only uses basic elements such as adders and shifters.

#### 4.5.1 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  $\beta$ , therefore the resultant vector would be

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

$$\begin{aligned}x' &= A(\cos \alpha \cos \beta - \sin \alpha \sin \beta) \\ &= x \cos \beta - y \sin \beta \\ y' &= A(\sin \alpha \cos \beta + \cos \alpha \sin \beta) \\ &= y \cos \beta + x \sin \beta\end{aligned}\tag{4.3}$$

With trigonometry identities and some manipulation, we can rewrite (4.3) as

$$\begin{aligned}x' &= \frac{1}{\sqrt{1 + \tan^2 \beta}} (x - y \tan \beta) \\ y' &= \frac{1}{\sqrt{1 + \tan^2 \beta}} (y + x \tan \beta)\end{aligned}\tag{4.4}$$

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:

$$\begin{aligned} x_{new} &= x - y \tan \beta \\ y_{new} &= y + x \tan \beta \end{aligned} \tag{4.5}$$

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 4.3.

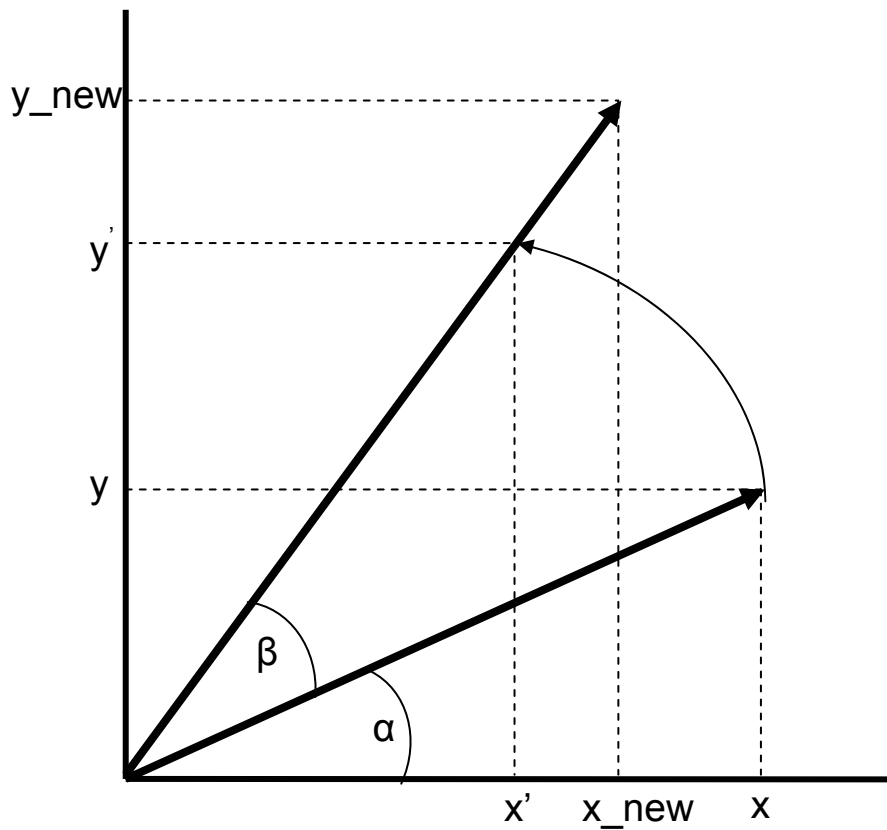


Figure 4.3 Pseudo-rotation and Rotation of a Vector

#### 4.5.2 Choice of Angles and Iterative Rotation

From (4.5), it is clear that in order to calculate the  $x_{\text{new}}$  and  $y_{\text{new}}$ , 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  $\beta$  as sum (or difference) of such angles. This implies an iterative method to rotate an angle  $\beta$ . Table 4.6 lists the special angles for the first 10 iterations.

Iteration $i$	Angle $_i$	$\tan(\text{angle}_i) = 2^{1-i}$
1	$45^\circ$	$2^0$
2	$26.6^\circ$	$2^{-1}$
3	$14.0^\circ$	$2^{-2}$
4	$7.1^\circ$	$2^{-3}$
5	$3.6^\circ$	$2^{-4}$
6	$1.8^\circ$	$2^{-5}$
7	$0.9^\circ$	$2^{-6}$
8	$0.4^\circ$	$2^{-7}$
9	$0.2^\circ$	$2^{-8}$
10	$0.1^\circ$	$2^{-9}$

Table 4.6 Iterative Angles for CORDIC Rotation

Any angles between  $99.7^\circ$  and  $-99.7^\circ$  can be represented as a sum or difference of the angles in Table 4.6. For example,  $70^\circ$  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  $99.7^\circ$  and  $-99.7^\circ$ .

```

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 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°

```

After all the iterations, z will be forced to almost zero. Since each of the iterations performs a pseudo-rotation rather than a 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<sub>stage</sub>, then the amplifying factor is 
$$K = \prod_{i=1}^{N_{\text{stage}}} \sqrt{1 + \tan^2(\text{angle}_i)} = \prod_{i=1}^{N_{\text{stage}}} \sqrt{1 + 2^{2(1-i)}}.$$

For large number of stages, this factor K converges to 1.6467.

### 4.5.3 CORDIC in Vector Mode

In addition to rotate a given vector by an angle, CORDIC in vector mode is able to take an input vector (the x and y coordinates) and estimate the angle of the input [17].

The procedures are almost identical to (4.6) with a few minor changes. The operation

behaves like this. At each stage, the rotate angle is chosen such that the resultant y-coordinate is forced toward zero. When all iterations are done, summing over all the intermediate rotation angles produces an approximation of the angle of the input vector. Figure 4.4 plots the normalized error of the estimation for different number of stages.

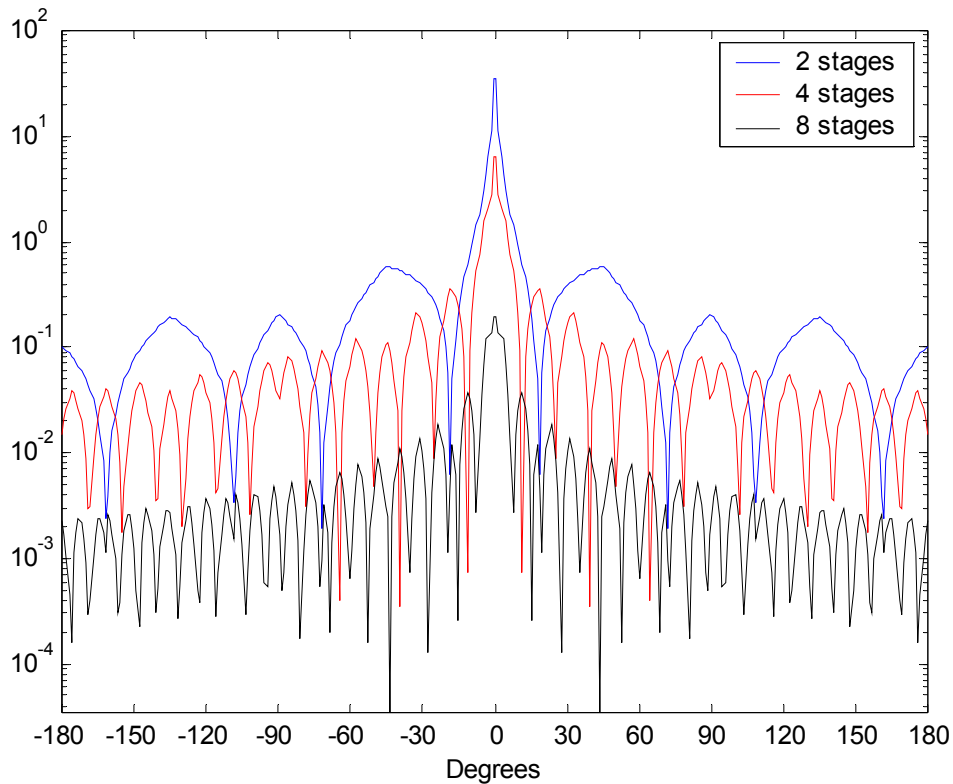


Figure 4.4 Normalized Angle Estimation Error for CORDIC

#### 4.5.4 CORDIC as an Amplitude Estimator

When CORDIC is used in the vector mode, the output x value is the original input amplitude scaled by the factor  $K$ . Since we know exactly the value of  $K$  based on the number of stages, CORDIC in vector mode can also be used as an amplitude estimator

for free. Figure 4.5 plots the normalized error for amplitude estimation using CORDIC. The simulation uses unit-norm vectors with different phases as inputs. It is clear from Figure 4.5 that at some particular angles, the normalized error becomes biggest. For example, for a CORDIC with only one stage, large error occurs at  $-180^\circ$ ,  $-90^\circ$ ,  $0^\circ$ ,  $90^\circ$ ,  $180^\circ$ . This observation can be explained with the following reasoning. With one stage CORDIC amplitude estimator, the input vector has to be rotated by either  $45^\circ$  or  $-45^\circ$ . For a vector whose phase is one of the above angles, this one time rotation produces a resultant vector that has the greatest Euclidean distance from the original vector. For instance, if the input is  $[1 \ 0]^T$ , we could estimate the amplitude by just reading the x value without doing any rotation. However, CORDIC blindly rotates the input to

$$\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}^T \text{ (after scaling correction), therefore the greatest error. It is also interesting to}$$

see that for certain input angles, the estimation is perfect due to the same reason explained above. For a one stage CORDIC, the angles of zero estimation error are  $-135^\circ$ ,  $-45^\circ$ ,  $45^\circ$ ,  $135^\circ$ .

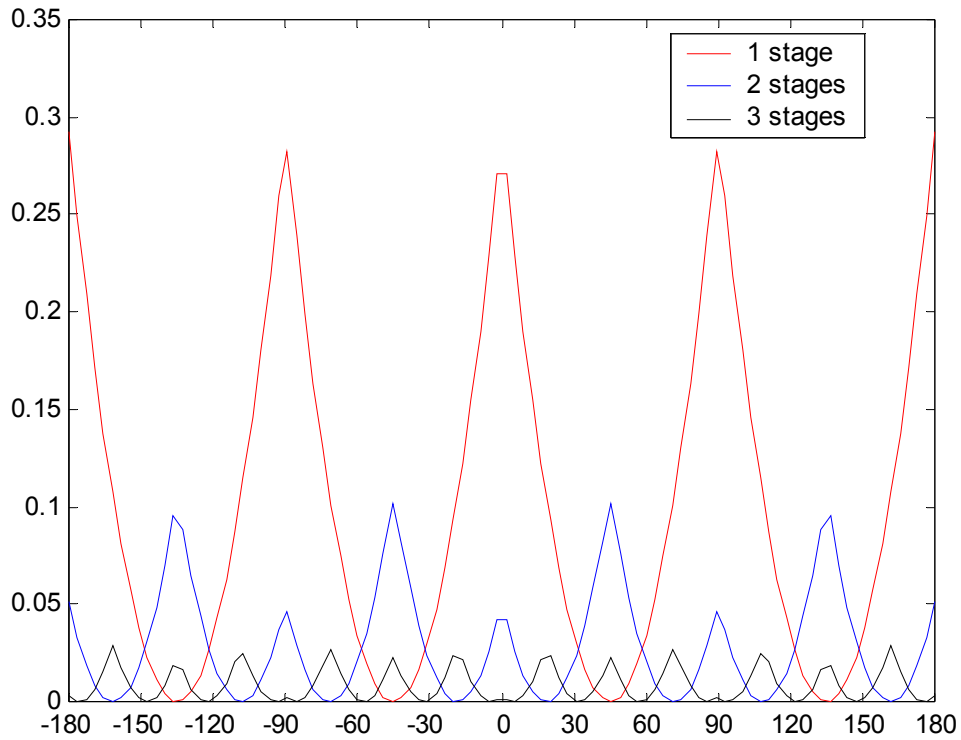


Figure 4.5 Normalized Amplitude Estimation Error for CORDIC

#### 4.5.5 Hardware Cost

The core component of the CORDIC module is the different stages. If we have parallel implementation of all the different stages, then the hardware cost is directly proportional to the number of stages. From (4.6), we see they are only 4 operations to be performed at each stage. Taking the sign of  $z_{i-1}$  can be thought of as hardware cost free since the sign information is stored in the most significant bit of the input  $z_{i-1}$  if we have 2's complement representation. Updating x coordinates requires an arithmetic shifter, an adder, a mux whose select signal is  $d_i$ , and a negate block. This is also true for updating y. For the  $z_i$  equation, a shifter is no longer needed, instead we need a constant to represent angle $_i$ .

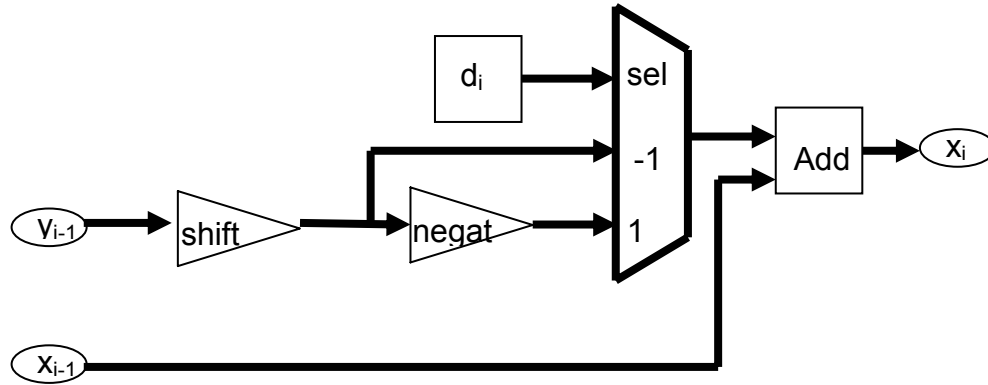


Figure 4.6 Implementation for Updaing x-coordinates

The total number of hardware components per stage is listed in Table 4.7.

Elementary Components Per Stage	
Adders	3
Shifters	2
Negate	3
Multiplexers	3

Table 4.7 Number of Components Per Stage of CORDIC

Since all 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 4.7.

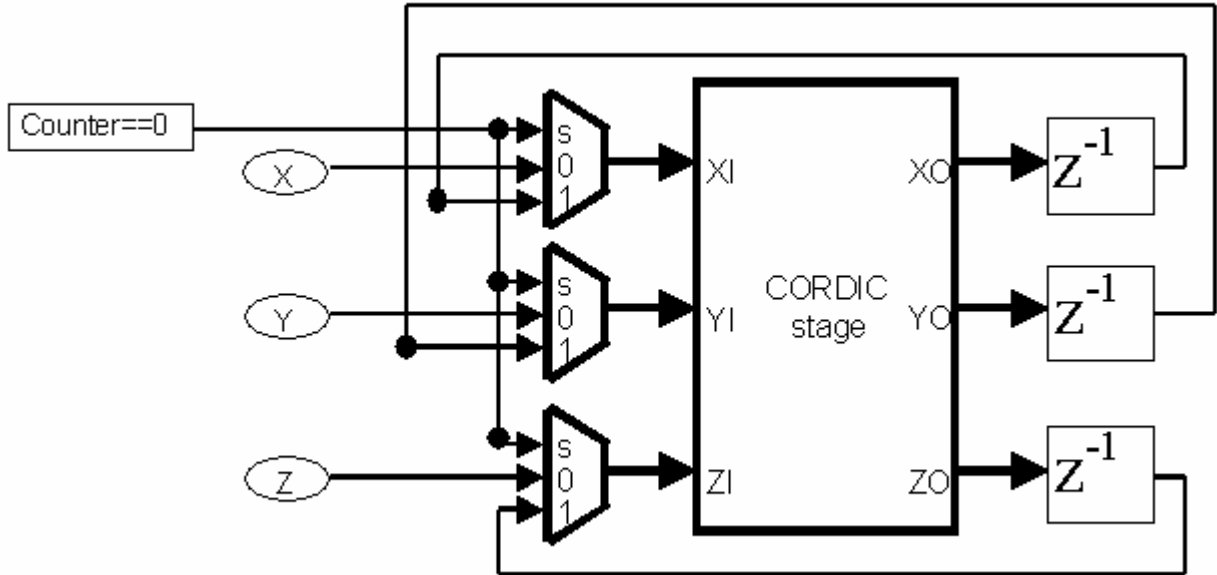


Figure 4.7 Shared Implementation of CORDIC

Shared implementation like that shown in Figure 4.7 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. However, power consumption will increase since all hardware components are operating at higher rate and power is linearly proportional to frequency.

#### 4.5.6 Module Compiler Implementation

Table 4.8 shows the area and critical path delay of the MC implementation of a 12-stage CORDIC. The input I and Q components are 10 bits wide and output angle is also 10 bits.

<b>Adder Type</b>	<b>Area (<math>\mu\text{m}^2</math>)</b>	<b>Critical Path Delay (ns)</b>
<b>Ripple</b>	11982	13.94
<b>CLSA</b>	24338	11.20
<b>CSA</b>	17739	11.89
<b>CLA</b>	20784	11.13
<b>FASTCLA</b>	23772	11.12

Table 4.8 MC Estimates for 12-stage CORDIC Optimized for Speed