An Overview of the Atan2 Cordic Implementation in Digital Hardware

Document: 94
www.signal-processing.net
andreas_dsp@hotmail.com
Author: Andreas Schwarzinger
Date: November 16, 2019

For certain digital signal processing applications, it is beneficial to work with IQ sample in the polar rather than the rectangular format. To achieve the translation into the polar format, we need to compute the atan2 operation to produce the phase of the IQ sample. A very clever technique is used to calculate the angle component of a complex IQ value in digital hardware. This technique, called the cordic atan2 algorithm, is the topic of this document and it makes due without hardware multipliers or ROM tables.

The Goal of the atan2 Function

The figure below shows two points P_1 and P_2 in the coordinate plane together with their associated angles. It is the objective of the atan2 algorithm to calculate the angle of a Cartesian coordinate, which in this case represents an *IQ* sample. The atan2 function applied to the input samples 1+2*j* and -2-2*j* must yield angles 63 and -135 degrees respectively.

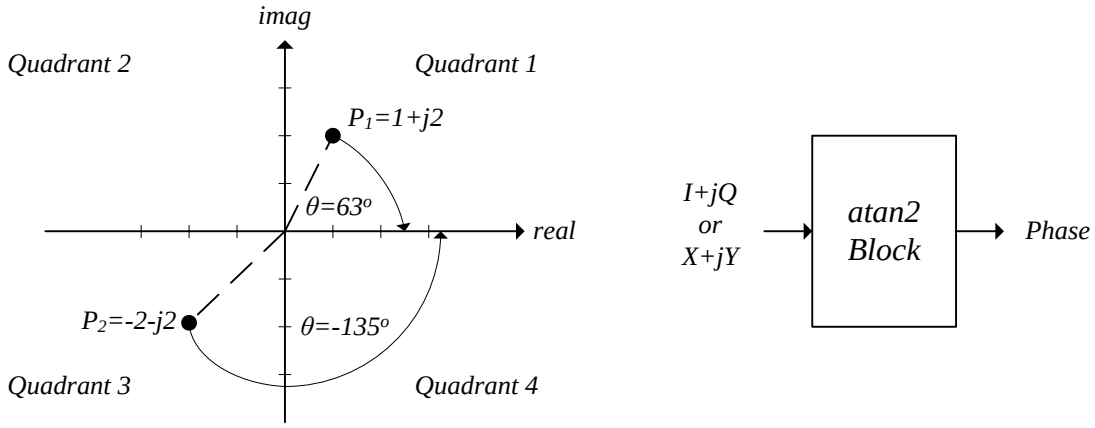

Figure 1: The Atan2 Operation finds the Angle of an IQ Sample

Whereas the hardware implementation of most trigonometric functions involve the use of potentially large ROM look up tables, the atan2 function can take advantage of the clever cordic technique, which allows us to primarily get away with simple addition and comparison operations. The figure below shows the internal function block arrangement of the cordic atan2 block.

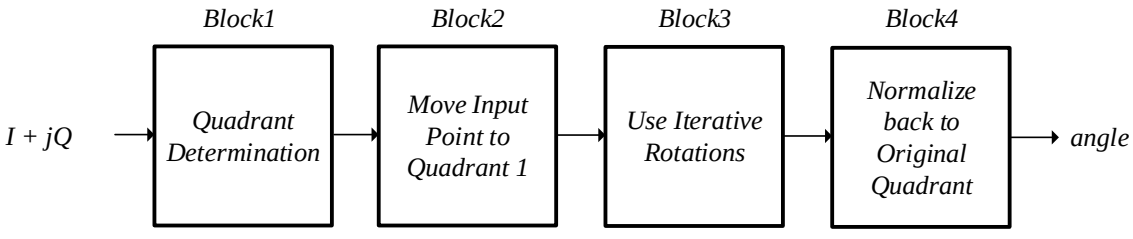

Figure 2: The Four Processing Steps of the Cordic Atan2 Algorithm

At its core, the cordic algorithm executes a series of preset rotations to determine the phase of the Cartesian coordinate. As will become clear soon, it is easiest to execute these rotations when the IQ coordinate resides in quadrant one. For this reason, the first two blocks determine where in coordinate system the sample resides and then map it to an intermediate position in the first quadrant. The angle of this intermediate position is now determined via several rotations in the third block, whereas the last block compensates for the earlier rotation to quadrant one.

<u>Computing the Angle of a Point in Quadrant One</u>

To understand why a series of rotation allows us to figure out the angle of a Cartesian coordinate in quadrant one, let's first examine what rotation looks like in terms of complex numbers. Usually, complex rotation requires the use of 4 multiplications as seen in the mathematical derivation below. One of our goals is to achieve similar rotations without using multiplication. Below, we rotate a complex number $P = X + jY$ by an arbitrary angle, θ, to a new position $P_1 = X_1 + jY_1$.

$$P_1 = P \cdot e^{j\theta}$$

$$P_1 = (X + jY) \cdot \big(\cos(\theta) + jsin(\theta)\big)$$

$$P_1 = X \cdot \cos(\theta) - Y \cdot \sin(\theta) + j\big(Y \cdot \cos(\theta) + X \cdot sin(\theta)\big) \quad (1)$$

The real and imaginary portions of P_1 can be written separately as X_1 and Y_1.

$$X_1 = X \cdot \cos(\theta) - Y \cdot \sin(\theta)$$

$$Y_1 = Y \cdot \cos(\theta) + X \cdot \sin(\theta)$$

We now factor the cosine term out of both equations to yield the following.

$$X_1 = \cos(\theta) \cdot (X - Y \cdot tan(\theta))$$

$$Y_1 = \cos(\theta) \cdot (Y + X \cdot tan(\theta))$$

$$P_1 = \cos(\theta)[(X - Y \cdot tan(\theta)) + j(Y + X \cdot \tan(\theta))]$$

As will become clear soon, the output coordinate of each individual rotation in the Cordic core does not need to produce an output coordinate with the same magnitude. To make our life easier, we simply replace the $\cos(\theta)$ expression by one to arrive at the following expression.

$$P_1 \approx X - Y \cdot tan(\theta)$$

$$+j(Y - X \cdot tan(\theta)) \quad (2)$$

Notice that compared to equation (1) we have reduced the number of multiplication to two at the expense of preserving the exact magnitude of the rotated coordinate. However, if we restrict the angles of rotation such that $tan(\theta)$ can only be ±1, ±0.5, ±0.25, ±0.125 … etc., then the remaining multiplications reduce to simple integer shifting operation and sign changes. These restrictions limit the angles we may use for the rotations to those in the following table. In block three, we cascade several of these rotation until we have managed to rotate the original point P right onto the positive X axis.

i	-2^{-i}	$Angle = atan(-2^{-i})$
0	-1.00000	-45.00
1	-0.50000	-26.565051
2	-0.25000	-14.036243
3	-0.12500	-7.125016
4	-0.06250	-3.576334
5	-0.03125	-1.789910
6	-0.015625	-0.895174
7	-.007813	-0.447614
8	-.003906	-0.223811
9	-1/512	-0.1119

Figure 3: Allowed Rotation for Cordic Pipeline with 10 Stages

Finding the Angle via Iterative Rotations

The figure below illustrates point P = [1, 2.5] featuring an angle of *atan2*(P) = 68.2 degrees as well at the iterative rotation pipeline that we employ in block three. The first stage of the pipeline looks at the point, P, and determines whether it can be rotated by -45 degrees without overshooting and producing a resulting coordinate in quadrant four. It does this by checking whether Q is larger than I. If it is, then it rotates $P=X+jY$ to $P_1=X_1+jY_1$ via the following equation.

$$X_1 = X - Y \cdot \tan(-45) = X + Y \cdot 1$$
$$Y_1 = Y + X \cdot \tan(-45) = Y - X \cdot 1$$

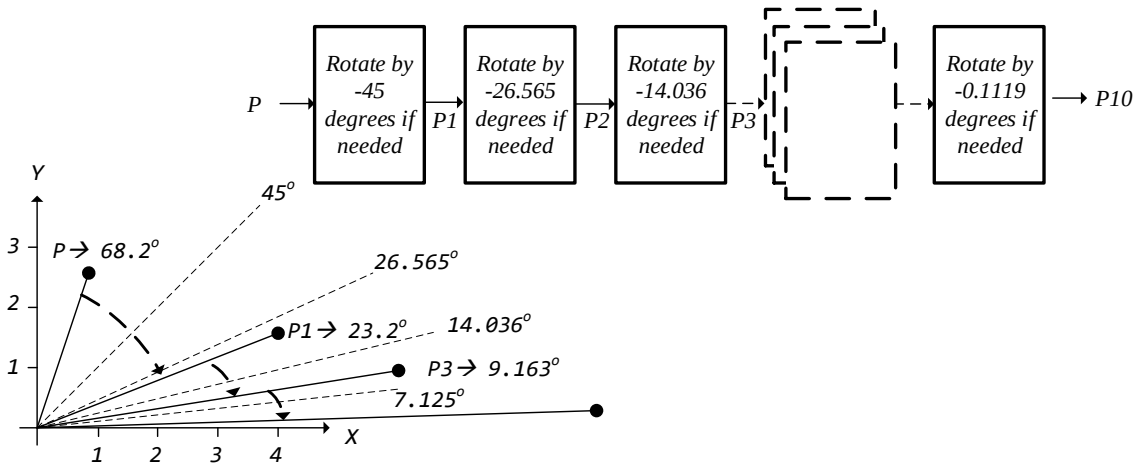

Figure 4: Example Rotation of Point P = [1, 2.5] in Block Three of The Cordic Atan2 Algorithm

The figure shows the rotation by angle A = -45 degrees that produced point P_1. The second cordic rotation block now checks to see whether the angle of P_1 is greater than 26.565 degrees and will decide to rotate P_1

if necessary. It does the check by finding out whether Y_1 is larger than $2X_1$. In this case we decide not to rotate which sets $P_2 = P_1$.

As can clearly be seen, the angle of P_1 is larger than 14.036 degrees and a rotation by -14.036 degrees is necessary. The algorithm detects that the angle of P1 is larger that 14.036 degrees by realizing that Y_1 is larger than $4X_1$.

$$X_3 = X_2 - Y_2 \cdot \tan(-14.036) \; = X_2 + Y_2 \cdot \frac{1}{4}$$

$$Y_3 = Y_2 + X_2 \cdot \tan(-14.036) \; = Y_2 - X_2 \cdot \frac{1}{4}$$

We thus proceed through as many cordic rotation elements as we need to achieve the needed accuracy. This algorithm uses 10 potential rotations to achieve a maximum phase error of .1119 degrees.

Block 1: Quadrant Determination

Since the cordic rotation pipeline of block three prefers to process points in the first quadrant, those in other quadrants must first be mapped into the first. We determine the quadrant of a point $P = X + jY$ via the following test.

```
Var1 = 0;
Var2 = 0;
if (X >= 0); Var1 = 1; end;
if (Y >= 0); Var2 = 1; end;

if(Var1 == 1 && Var2 == 1); Quadrant = 1; end;
if(Var1 == 0 && Var2 == 1); Quadrant = 2; end;
if(Var1 == 0 && Var2 == 0); Quadrant = 3; end;
if(Var1 == 1 && Var2 == 0); Quadrant = 4; end;
```

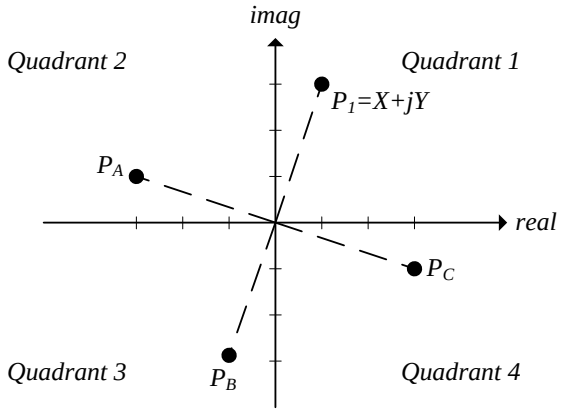

Figure 5: Mapping of Points in Quadrants 2, 3, and 4 to Quadrant1

Block 2: Mapping to Quadrant One

The figure above illustrates how IQ samples in quadrants 2, 3, and 4 are mapped to quadrant 1. Points P_A, P_B and P_C all map onto P_1. The mapping process is summarized in the next MatLab section shown below.

```
switch(Quadrant)
    case 1
        X1 = X;
        Y1 = Y;
    case 2
        X1 = Y;
        Y1 = -X;
    case 3
        X1 = -X;
        Y1 = -Y;
    case 4
        X1 = -Y;
        Y1 = X;
end
```

Block 3. Iterative Rotations

Check the final MatLab code for the iterative rotations.

Block 4. The Final Normalization Step

The last step in the cordic *atan2()* algorithm is the normalization step. This step adds the additional angle was lost during the initial mapping step to quadrant one.

```
switch(Quadrant)
    case 1
        Angle = TotalRotation;
    case 2
        Angle = TotalRotation + pi/2;
    case 3
        Angle = TotalRotation - pi;
    case 4
        Angle = TotalRotation - pi/2;
end
```

Full MatLab Code

```
function Angle = CordicAtan2(X, Y)

%% Block 1. Quadrant Determination
Var1 = 0;
Var2 = 0;
if (X >= 0); Var1 = 1; end;
if (Y >= 0); Var2 = 1; end;
if(Var1 == 1 && Var2 == 1); Quadrant = 1; end;
if(Var1 == 0 && Var2 == 1); Quadrant = 2; end;
if(Var1 == 0 && Var2 == 0); Quadrant = 3; end;
if(Var1 == 1 && Var2 == 0); Quadrant = 4; end;

%% Block 2. Mapping to Quadrant 1
switch(Quadrant)
    case 1
        X1 = X;
        Y1 = Y;
    case 2
        X1 = Y;
        Y1 = -X;
    case 3
        X1 = -X;
        Y1 = -Y;
    case 4
        X1 = -Y;
        Y1 = X;
end

%% Block 3. 10 Stage Cordic Rotation
TotalRotation = 0;

if(Y1 >= X1)                                %% Stage 1
    X2 = X1 + Y1;    Y2 = Y1 - X1;
    TotalRotation = TotalRotation + 45*pi/180;
else
    X2 = X1;    Y2 = Y1;
end

if(2*Y2 >= X2)                              %% Stage 2
    X3 = X2 + Y2/2;   Y3 = Y2 - X2/2;
    TotalRotation = TotalRotation + 26.56505*pi/180;
else
    X3 = X2;          Y3 = Y2;
end

if(4*Y3 >= X3)                              %% Stage 3
    X4 = X3 + Y3/4;    Y4 = Y3 - X3/4;
    TotalRotation = TotalRotation + 14.03624*pi/180;
else
    X4 = X3;           Y4 = Y3;
end

if(8*Y4 >= X4)                              %% Stage 4
    X5 = X4 + Y4/8;    Y5 = Y4 - X4/8;
    TotalRotation = TotalRotation + 7.12502*pi/180;
else
    X5 = X4;           Y5 = Y4;
end

if(16*Y5 >= X5)                             %% Stage 5
    X6 = X5 + Y5/16;   Y6 = Y5 - X5/16;
    TotalRotation = TotalRotation + 3.57633*pi/180;
else
```

```
if(32*Y6 >= X6)                             %% Stage 6
    X7 = X6 + Y6/32;   Y7 = Y6 - X6/32;
    TotalRotation = TotalRotation + 1.78991*pi/180;
else
    X7 = X6;           Y7 = Y6;
end

if(64*Y7 >= X7)                             %% Stage 7
    X8 = X7 + Y7/64;   Y8 = Y7 - X7/64;
    TotalRotation = TotalRotation + 0.89517*pi/180;
else
    X8 = X7;           Y8 = Y7;
end

if(128*Y8 >= X8)                            %% Stage 8
    X9 = X8 + Y8/128;  Y9 = Y8 - X8/128;
    TotalRotation = TotalRotation + 0.44761*pi/180;
else
    X9 = X8;           Y9 = Y8;
end

if(256*Y9 >= X9)                            %% Stage 9
    X10 = X9 + Y8/256;  Y10 = Y9 - X8/256;
    TotalRotation = TotalRotation + 0.22381*pi/180;
else
    X10 = X9;           Y10 = Y9;
end

if(512*Y10 >= X10)                          %% Stage 10
    TotalRotation = TotalRotation + 0.11191*pi/180;
end



%% Block 4. Normalization Step
switch(Quadrant)
    case 1;        Angle = TotalRotation;
    case 2;        Angle = TotalRotation + pi/2;
    case 3;        Angle = TotalRotation - pi;
    case 4;        Angle = TotalRotation - pi/2;
end
```