dspGuru™

by Iowegian International

CORDIC FAQ

CORDIC is an iterative algorithm for calculating trig functions including sine, cosine, magnitude and phase. It is particularly suited to hardware implementations because it does not require any multiplies.

1. Basics

1.1 What does "CORDIC" mean?

> **CO**ordinate **R**otation **DI**gital **C**omputer. (Doesn't help much, does it?!)

1.2 What does it do?

> It calculates the trigonometric functions of sine, cosine, magnitude and phase (arctangent) to any desired precision. It can also calculate hyperbolic functions, but we don't cover that here.

1.3 How does it work?

> CORDIC revolves around the idea of "rotating" the phase of a complex number, by multiplying it by a succession of constant values. However, the multiplies can all be powers of 2, so in binary arithmetic they can be done using just shifts and adds; no actual multiplier is needed.

1.4 How does it compare to other approaches?

> Compared to other approaches, CORDIC is a clear winner when a hardware multiplier is unavailable, e.g. in a microcontroller, or when you want to save the gates required to implement one, e.g. in an FPGA. On the other hand, when a hardware multiplier *is* available, e.g. in a DSP microprocessor, table-lookup methods and good old-fashioned power series are generally faster than CORDIC.

2.0 Examples

2.1 Do you have an example in the form of an Excel spreadsheet?

> Funny you should ask: I just happen to: cordic-xls.zip. I highly recommend you open this up and look it over a little before you ask any more questions.

2.2 Do you have an example as C code?

> Yup, it's your lucky day: cordic.zip.

3.0 Principles

3.1 What symbols will we be using?

Given a complex value: `C = Ic + jQc`

we will create a rotated value: `C' = Ic' + jQc'`

by multiplying by a rotation value: `R = Ir + jQr`

3.2 What are the basic principles?

1. Recall that when you multiply a pair of complex numbers, their phases (angles) add and their magnitudes multiply. Similarly, when you multiply one complex number by the *conjugate* of the other, the phase of the conjugated one is subtracted (though the magnitudes still multiply).Therefore:

To *add* R's phase to C:	`C' = C·R`	`Ic' = Ic·Ir − Qc·Qr` `Qc' = Qc·Ir + Ic·Qr`
To *subtract* R's phase from C:	`C' = C·R*`	`Ic' = Ic·Ir + Qc·Qr` `Qc' = Qc·Ir − Ic·Qr`

2. To rotate by +90 degrees, multiply by R = 0 + j1. Similarly, to rotate by -90 degrees, multiply by R = 0 − j1. If you go through the Algebra above, the net effect is:

To *add* 90 degrees, multiply by R = 0 + j1:	`Ic' = -Qc` `Qc' = Ic`	(negate Q, then swap)
To *subtract* 90 degrees, multiply by R = 0 − j1:	`Ic' = Qc` `Qc' = -Ic`	(negate I, then swap)

3. To rotate by phases of less than 90 degrees, we will be multiplying by numbers of the form "R = 1 +/- jK". K will be decreasing powers of two, starting with 2^0 = 1.0. Therefore, K = 1.0, 0.5, 0.25, etc. (We use they symbol "L" to designate the power of two itself: 0, -1, -2, etc.)Since the phase of a complex number "I + jQ" is atan(Q/I), the phase of "1 + jK" is atan(K). Likewise, the phase of "1 − jK" = atan(-K) = -atan(K).To *add* phases we use "R = 1 + jK"; to *subtract* phases we use "R = 1 − jK". Since the real part of this, Ir, is equal to 1, we can simplify our table of equations to add and subtract phases for the special case of CORDIC multiplications to:

To add a phase, multiply by R = 1 + jK:	`Ic' = Ic − K·Qc = Ic − (2^-L)·Qc = Ic − (Qc >> L)` `Qc' = Qc + K·Ic = Qc + (2^-L)·Ic = Qc + (Ic >> L)`
To subtract a phase, multiply by R = 1 − jK:	`Ic' = Ic + K·Qc = Ic + (2^-L)·Qc = Ic + (Qc >> L)` `Qc' = Qc − K·Ic = Qc − (2^-L)·Ic = Qc − (Ic >> L)`

4. Let's look at the phases and magnitudes of each of these multiplier values to get more of a feel for it. The table below lists values of L, starting with 0, and shows the corresponding values of K, phase, magnitude, and CORDIC Gain (described below):

L	**K = 2^-L**	**R = 1 + jK**	**Phase of R in degrees = atan(K)**	**Magnitude of R**	**CORDIC Gain**
0	1.0	1 + j1.0	45.00000	1.41421356	1.414213562
1	0.5	1 + j0.5	26.56505	1.11803399	1.581138830
2	0.25	1 + j0.25	14.03624	1.03077641	1.629800601

3	0.125	1 + j0.125	7.12502	1.00778222	1.642484066
4	0.0625	1 + j0.0625	3.57633	1.00195122	1.645688916
5	0.03125	1 + j0.031250	1.78991	1.00048816	1.646492279
6	0.015625	1 + j0.015625	0.89517	1.00012206	1.646693254
7	0.007813	1 + j0.007813	0.44761	1.00003052	1.646743507
…	…	…	…	…	…

A few observations:

- Since we're using powers of two for the K values, we can just shift and add our binary numbers. *That's why the CORDIC algorithm doesn't need any multiplies!*
- You can see that starting with a phase of 45 degrees, the phase of each successive R multiplier is a little over half of the phase of the previous R. *That's the key to understanding CORDIC:* we will be doing a "binary search" on phase by adding or subtracting successively smaller phases to reach some target phase.
- The sum of the phases in the table up to L = 3 exceeds 92 degrees, so we can rotate a complex number by +/- 90 degrees as long as we do four or more "R = 1 +/- jK" rotations. Put that together with the ability to rotate +/-90 degrees using "R = 0 +/- j1", and you can rotate a full +/-180 degrees.
- Each rotation has a magnitude greater than 1.0. That isn't desirable, but it's the price we pay for using rotations of the form 1 + jK. The "CORDIC Gain" column in the table is simply a cumulative magnitude calculated by multiplying the current magnitude by the previous magnitude. Notice that it converges to about 1.647; however, the *actual* CORDIC Gain depends on how many iterations we do. (It *doesn't* depend on whether we add or subtract phases, because the magnitudes multiply either way.)

4.0 Applications

4.1 How can I use CORDIC to calculate magnitude?

You can calculate the magnitude of a complex number C = Ic + jQc if you can rotate it to have a phase of zero; then its new Qc value would be zero, so the magnitude would be given entirely by the new Ic value.

"So how do I rotate it to zero," you ask? Well, I thought you might ask:

1. You can determine whether or not the complex number "C" has a positive phase just by looking at the sign of the Qc value: positive Qc means positive phase. As the very first step, if the phase is positive, rotate it by -90 degrees; if it's negative, rotate it by +90 degrees. To rotate by +90 degrees, just negate Qc, then swap Ic and Qc; to rotate by -90 degrees, just negate Ic, then swap. The phase of C is now less than +/- 90 degrees, so the "1 +/- jK" rotations to follow can rotate it to zero.
2. Next, do a series of iterations with successively smaller values of K, starting with K=1 (45 degrees). For each iteration, simply look at the sign of Qc to decide whether to add or subtract phase; if Qc is negative, add a phase (by multiplying by "1 + jK"); if Qc is positive, subtract a phase (by multiplying by "1 – jK"). The accuracy of the result converges with each iteration: the more iterations you do, the more accurate it becomes.[Editorial Aside: Since each phase is a little *more* than half the previous phase, this algorithm is slightly underdamped. It could be made slightly more accurate, on average, for a given number of iterations, by using "ideal" K values which would add/subtract phases of 45.0, 22.5, 11.25 degrees, etc. However, then the K values wouldn't be of the form 2^-L, they'd be 1.0, 0.414, 0.199, etc., and you couldn't multiply using just shift/add's (which would eliminate the major benefit of the algorithm). In practice, the difference in accuracy between the ideal K's and these binary K's is generally negligible; therefore, for a multiplier-less CORDIC, go ahead and use the binary Ks, and if you need more accuracy, just do more iterations.]

Now, having rotated our complex number to have a phase of zero, we end up with "C = Ic + j0". The magnitude of this complex value is just Ic, since Qc is zero. However, in the rotation process, C has been multiplied by a CORDIC Gain (cumulative magnitude) of about 1.647. Therefore, to get the *true* value of magnitude we must multiply by the reciprocal of 1.647, which is 0.607. (Remember, the exact CORDIC Gain is a function of the how many iterations you do.) Unfortunately, we can't do this gain-adjustment multiply using a simple shift/add; however, in many applications this factor can be compensated for in some other part of the system. Or, when relative magnitude is all that counts (e.g. AM demodulation), it can simply be neglected.

4.2 How can I use CORDIC to calculate phase?

To calculate phase, just rotate the complex number to have zero phase, as you did to calculate magnitude. Just a couple of details are different.

1. For each phase-addition/subtraction step, accumulate the *actual* number of degrees (or radians) you have rotated. The actuals will come from a table of "atan(K)" values like the "Phase of R" column in the table above. The phase of the complex input value is the negative of the accumulated rotation required to bring it to a phase of zero.
2. Of course, you can skip compensating for the CORDIC Gain if you are interested only in phase.

4.3 Since magnitude and phase are both calculated by rotating to a phase of zero, can I calculate both simultaneously?

Yes – you're very astute.

4.4 How can I use CORDIC to calculate sine and cosine?

You basically do the inverse of calculating magnitude/phase by adding/subtracting phases so as to "accumulate" a rotation equal to the given phase. Specifically:

1. Start with a unity-magnitude value of C = Ic + jQc. The exact value depends on the given phase. For angles greater than +90, start with C = 0 + j1 (that is, +90 degrees); for angles less than -90, start with C = 0 – j1 (that is, -90 degrees); for other angles, start with C = 1 + j0 (that is, zero degrees). Initialize an "accumulated rotation" variable to +90, -90, or 0 accordingly. (Of course, you also could do all this in terms of radians.)
2. Do a series of iterations. If the desired phase minus the accumulated rotation is less than zero, add the next angle in the table; otherwise, subtract the next angle. Do this using each value in the table.
3. The "cosine" output is in "Ic"; the "sine" output is in "Qc".

A couple of notes:

1. Again, the accuracy improves by about a factor of two with each iteration; use as many iterations as your application's accuracy requires.
2. This algorithm gives you *both* cosine (Ic) and sine (Qc). Since CORDIC uses complex values to do its magic, it's not possible to calculate sine and cosine separately.

5.0 References

5.1 Where can I learn more about CORDIC on-line?

1. A survey of CORDIC algorithms for FPGAs by Ray Andraka of Andraka Consulting Group. This very readable paper's review of CORDIC's principles is more comprehensive and rigorous than this FAQ, so it's a good place to go from here.
2. Wikipedia's CORDIC Page
3. Fixed-Point Trigonometry With CORDIC Iterations by Ken Turkowski
4. New Virtually Scaling-Free Adaptive CORDIC Rotator

5.2 What are some article references for CORDIC?

1. Jack E. Volder, *The CORDIC Trigonometric Computing Technique*, IRE Transactions on Electronic Computers, September 1959.
2. J. E. Meggitt, *Pseudo Division and Pseudo Multiplication Processes*, IBM Journal, April 1962.

5.3 What are some book references for CORDIC?

1. M. E. Frerking, *Digital Signal Processing in Communication Systems* [Fre94].
2. Henry Briggs, *Arithmetica Logarithmica*, 1624.

6.0 Acknowledgments

Thanks to Ray Andraka for helping me understand. Thanks to Rick Lyons for review. Thanks to Mark Brown and Bill Wiese for providing links.