Analytical Study of Coordinate Rotation Digital Computer (CORDIC) Algorithm

Samandeep Singh Dhillon[1] Sarabdeep Singh [2]
[1]Student of M. Tech [2]Assistant Professor
[1,2]Department of Electronics & Communication Engineering
[1,2]Chandigarh Engineering College Landran Mohali India Sarabdeep Singh

Abstract— Due to rapid advances made in Very Large Scale Integration (VLSI) technologies have led the way to an entirely different approach to computer design for real time application, using special-purpose architectures with custom chips. Special purpose architectures efficiently map the algorithmic needs of the problem to hardware. Use of special arithmetic techniques for special applications and introduction of pipelining and parallelism lead to design very different from basic computers. As intended by Jack E. Volder, 1959, the CORDIC algorithm only performs shift and add operations and is therefore easy to implement and resource-friendly. In this paper analytical investigations on CORDIC algorithm have been presented.

Key words: Coordinate Rotation Digital Computer (CORDIC) Algorithm, Trigonometric Function, Vector Rotation, Vector Translation

I. INTRODUCTION

First described in 1959 [Jack E. Volder,1959], Coordinate Rotation Digital Computer (CORDIC) algorithm is an iterative algorithm, which can be used for the computation of trigonometric functions, logarithmic, complex number multiplication, matrix inversion, solution of linear systems and general scientific computation. Last half century has witnessed a lot of progress in design and development of architectures of the algorithm for high-performance and low-cost hardware solutions. CORDIC algorithm got its popularity, when [John S. Walther, 1971] showed that, by varying a few simple parameters, it could be used as a single algorithm for unified implementation of a wide range of logarithms, exponentials, and square functions.

The popularity of CORDIC was very much enhanced thereafter primarily due to its potential for efficient and low-cost implementation. With the advent of low cost, low power Field Programmable Gate Arrays (FPGAs), this algorithm has shown its potential for efficient and low-cost implementation. CORDIC algorithm can be widely used in as wireless communications, Software Defined Radio and medical imaging applications, which are heavily dependent on signal processing. Some other upcoming applications are:

- Direct frequency synthesis, digital modulation and coding for speech/music synthesis and communication
- Direct and inverse kinematics computation for robot manipulation
- Planar and three-dimensional vector rotation for graphics and animation.

Although CORDIC may not be the fastest technique to perform these operations, yet it is attractive due to the simplicity of its hardware implementation.

Keeping the requirements and constraints of different application environments in view, the development of CORDIC algorithm and architecture has taken place for achieving high throughput rate and reduction of hardware-complexity as well as the latency of implementation. Some of the typical approaches for reduced-complexity implementation are focused on minimization of the complexity of scaling operation and the complexity of barrel-shifter in the CORDIC engine. Latency of implementation is an inherent drawback of the conventional CORDIC algorithm. Parallel and pipelined CORDIC have been suggested for high-throughput computation and efficient CORDIC algorithm.

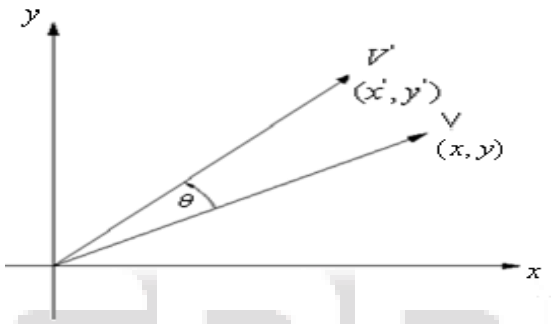

Fig. 1: Vector Rotation

CORDIC algorithm has two types of computing modes Vector rotation (Rotating mode) and vector translation (Vectoring mode). The CORDIC algorithm was initially designed to perform a vector rotation, where the vector V with components (x, y) is rotated through the angle θ yielding a new vector V' with component (x', y') shown in Figure

$$V' = [R][V] \quad (1)$$

Where R is the rotation matrix:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad (2)$$

$$V' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (3)$$

The individual equations for x' and y' can be rewritten as:

$$x' = x.\cos(\theta) - y.\sin(\theta) \quad (4)$$

$$y' = y.\cos(\theta) + x.\sin(\theta) \quad (5)$$

And rearranged so that

$$x' = \cos(\theta)[x - y.\tan(\theta)] \quad (6)$$

$$y' = \cos(\theta)[y + x.\tan(\theta)] \quad (7)$$

The multiplication by the tangent term can be avoided if the rotation angles and therefore $\tan(\theta)$ are restricted so that $\tan(\theta) = 2^{-i}$. In digital hardware this denotes a simple shift operation. Furthermore, if those rotations are performed iteratively and in both directions every value of

$\tan(\theta)$ is representable. With $\theta = \arctan(2^{-i})$ the cosine term could also be simplified and since $\cos(\theta) = \cos(-\theta)$ it is a constant for a fixed number of iterations. This iterative rotation can now be expressed as:

$$x_{i+1} = k_i[x_i - y_i.d_i.2^{-i}] \quad (8)$$

$$y_{i+1} = k_i[y_i + x_i.d_i.2^{-i}] \quad (9)$$

Where $k_i = \cos(\arctan(2^{-i}))$ and $d_i = \pm 1$. The product of the k_i's represents the so-called K factor [Shaoyun Wang and Earl E. Swartzlander, 1991]:

$$k = \prod_{i=0}^{n-1} k_i \quad (10)$$

This K factor can be calculated in advance and applied elsewhere in the system. Equations (8) and (9) can now be simplified to the basic CORDIC equations:

$$x_{i+1} = [x_i - y_i.d_i.2^{-i}] \quad (11)$$

$$y_{i+1} = [y_i + x_i.d_i.2^{-i}] \quad (12)$$

The direction of each rotation is defined by d_i and the sequence of all d_i's determines the final vector. Each vector V can be described by either the vector length or angle or by its coordinates x and y. Following this incident, the CORDIC algorithm knows two ways of determining the direction of rotation: the rotation mode and the vectoring mode. Both methods initialize the angle accumulator with the desired angle z_0. The rotation mode, determines the right sequence as the angle accumulator approaches 0 while the vectoring mode minimizes the y component of the input vector.

The angle accumulator is defined by:

$$z_{i+1} = z_i - d_i.\arctan(2^{-i}) \quad (13)$$

Where the sum of an infinite number of iterative rotation angles equals the input angle θ:

$$\theta = \sum_{i=0}^{\infty} d_i.\arctan(2^{-i}) \quad (14)$$

Those values of $\arctan(2^{-i})$ can be stored in a small lookup table or hardwired depending on the way of implementation. Since the decision is which direction to rotate instead of whether to rotate or not, d_i is sensitive to the sign of z_i. Therefore d_i can be described as:

$$d_i = \begin{cases} -1, & if \quad z_i < 0 \\ +1, & if \quad z_i \geq 0 \end{cases} \quad (15)$$

With equation (15) the CORDIC algorithm in rotation mode is described completely. Note, that the CORDIC method as described performs rotations only within $-\pi/2$ and $\pi/2$. This limitation comes from the use of 2^0 for the tangent in the first iteration. However, since a sine wave is symmetric from quadrant to quadrant, every sine value from 0 to 2π can be represented by reflecting and/or inverting the first quadrant appropriately.

In vector translation, rotates the vector V with component (X, Y) around the circle until the Y component equals zero as illustrated in Figure 2. The outputs from vector translation are the magnitude X' and phase z', of the input vector V with component (X,Y).

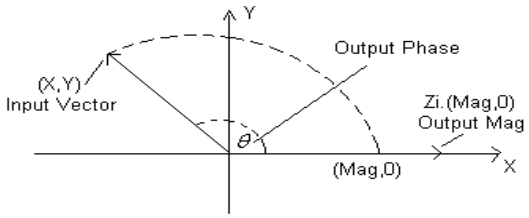

Fig. 2: Vector Translation

After vector translation, output equations are:

$$X' = k_i\sqrt{(X^2 + Y^2)} \quad (16)$$

$$Y' = 0 \quad (17)$$

$$z' = a\tan\left(\frac{Y}{X}\right) \quad (18)$$

To achieve simplicity of hardware realization of the rotation, the key ideas used in CORDIC arithmetic are to decompose the rotations into a sequence of elementary rotations through predefined angles that could be implemented with minimum hardware cost and to avoid scaling, that might involve arithmetic operation, such as square-root and division. The second idea is based on the fact the scale-factor contains only the magnitude information but no information about the angle of rotation.

II. GENERALIZED CORDIC ALGORITHM

In 1971, John S. Walther found how CORDIC iterations could be modified to compute hyperbolic functions and reformulated the CORDIC algorithm into a generalized and unified form which is suitable to perform rotations in circular, hyperbolic and linear coordinate systems. The unified formulation includes a new variable m, which is assigned different values for different coordinate systems. The generalized CORDIC is formulated as follows:

$$\begin{aligned} x_{i+1} &= x_i - m\sigma_i.2^{-i}.y_i \\ y_{i+1} &= y_i + \sigma_i.2^{-i}.x_i \\ w_{i+1} &= w_i - \sigma_i.\alpha_i \end{aligned} \quad (19)$$

Where

$$\sigma_i = \begin{cases} sign(w_i) & \text{for rotation mode} \\ -sign(w_i) & \text{for vectoring mode} \end{cases}$$

For $m = 1, 0$ or -1 and $\alpha_i = \tan^{-1}(2^{-i}), 2^{-i}$ or $\tanh^{-1}(2^{-i})$, the algorithm given by (19) works in circular, linear or hyperbolic coordinate systems, respectively. Table 1 summarizes the operations that can be performed in rotation and vectoring modes in each of these coordinate systems. The convergence range of linear and hyperbolic CORDIC are obtained, as in the case of circular coordinate, by the sum of all α_i given by $\sum_{i=0}^{\infty} \sigma_i$.

m	Rotation mode	Vectoring mode
0	$x_n = k(x_o \cos w_o - y_o \sin w_o)$	$x_n = k\sqrt{x_o^2 + y_o^2}$
	$y_n = k(x_o \sin w_o + y_o \cos w_o)$	$y_n = 0$
	$w_n = 0$	$w_n = w_0 + \tan^{-1}(y_o / x_0)$
1	$x_n = x_o$	$x_n = x_o$
	$y_n = y_o + x_o w_o$	$y_n = 0$
	$w_n = 0$	$w_n = w_0 + (y_o / x_0)$
-1	$x_n = k_h(x_o \cosh w_o - y_o \sinh w_o)$	$x_n = k_h\sqrt{x_o^2 + y_o^2}$
	$y_n = k_h(x_o \sinh w_o + y_o \cosh w_o)$	$y_n = 0$
	$w_n = 0$	$w_n = w_0 + \tanh^{-1}(y_o / x_0)$

Table 1: Generalized CORDIC Algorithm

The hyperbolic CORDIC requires to execute iterations for $i = 4,13,40.....$ twice to ensure convergence. Consequently, these repetitions must be considered while computing the scale-factor $K_h = \prod(1+2^{-2i})^{-1/2}$, which converges to 0.828

III. Applications of CORDIC Algorithm

A. Sin and Cos Function:

CORDIC can be used to compute Sin of any angle θ with little variation. The angle is given as input. A vector length 647 (CORDIC gain) along the x-axis is taken. The vector is then rotated in steps so as to reach the desired input angle θ. The x and y values are accumulated. After fixed number of iterations the final co-ordinates of the vector i.e. the x and y values give value of Cos and Sin respectively of the given angle θ.

B. Tangent:

Tangent of an input angle can be easily calculated using CORDIC by dividing the accumulated values of Y and X after rotation by the desired angle θ.

$$Y / X = \tan(\theta)$$

C. Inverse Trigonometric functions:

Give the values of x and y the inverse trigonometric functions can also be calculated. To get the inverse tangent following initial values of constants can be taken

$$\text{Set} \quad \begin{aligned} k &= 0 \\ \theta &= 0 \\ X &= 1 \end{aligned}$$

Value of Y is given as input. The vector is rotated using CORDIC iterations in steps until the final value of Y is reached equal to zero. The angle traced during this rotation is the inverse tangent of the value given to Y.

D. Phase:

To calculate phase, just rotate the vector to have zero phase, as done to calculate magnitude. The process is different by just a small bit.

- For each phase addition/subtraction step, accumulate the actual number of degrees or radians that are rotated. The values are taken from table of arctan K. The phase of the complex input value is the negative of the accumulated rotation required to bring it to a phase of zero.
- The CORDIC gain can be skipped if only the phase is to be calculated.

E. Magnitude:

Magnitude can be calculated using CORDIC algorithm. The magnitude of a complex number $C = I_C + jQ_C$ can be calculated if it is rotated to have a phase equal to zero, then its new Q_C value would be zero, so the magnitude would be given entirely by the new I_C value. The procedure to rotate the vector is given below:

- It can be determined whether or not the complex number C has a positive phase just by looking at the sign of the Q_C value. If Q_C is positive it means phase is positive. As the very first step, if the phase is positive, rotate it by -90 degrees; if it is negative, rotate it by +90 degrees. To rotate by +90 degrees, just negate Q_C, then swap I_C and Q_C, to rotate by -90 degrees, just I_C then swap. The phase of C is now less than ± 90 degrees, so the $1 \pm jK$ rotations to follow can rotate it to zero.
- Next a series of iterations is performed with successively smaller values of K, starting with K=1 (45 degrees). For each iteration, simply look at the sign of Q_C to decide whether to add or subtract phase, if Q_C is negative, add a phase (by multiplying by $1 + jK$); if Q_C is positive, subtract a phase (by multiplying by $1 - jK$). The accuracy of the result converges with each iteration. The more the iterations are performed, the more accurate results are obtained.

Since each phase is a little more than half the previous phase, this algorithm is slightly under-damped. 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, 125 degrees etc.

However, then the K values wouldn't be of the form 2^{-i}, they are supposed to be 0, 0.414, 0.199 etc and multiplication using just shift/ad operations is then not possible. In practice, the difference in accuracy between the ideal Ks and these binary Ks is generally negligible, therefore, for a multiplier-less CORDIC, use the binary Ks and if more accuracy is needed, just increase the number of iterations or increase the number of signed powers of two to get better results. So after the said rotation is performed to reduce the phase of the vector to zero, the resulting complex number obtained after this entire operation is $C = I_C + j0$.

The magnitude of this complex value is just I_C (since Q_C is zero). However, in the rotation process, C has been multiplied by a CORDIC gain of about 647. Therefore, to get the true value of magnitude we must multiply by the reciprocal of 647, which is 0.607 (the value of exact CORDIC gain is a function of the how many iterations are performed). 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. When relative magnitude is all that counts, it can simply be neglected.

Note: Since magnitude and phase are both calculated by rotating to a phase of zero, both operations can be done simultaneously.

F. Polar to Rectangular transformation:

A logical extension to the sine and cosine computer is a polar to Cartesian coordinate transformer. The transformation from polar to Cartesian space is defined by

$$x = r\cos\theta$$

$$y = r\sin\theta$$

As pointed out above, the multiplication by the magnitude comes for free using the CORDIC rotator. The transformation is accomplished by selecting the rotation mode with x_0 =polar magnitude, z_0 = polar phase and y_0 =0. The vector result represents the polar input transformed to Cartesian space. The transform has a gain equal to the rotator gain, which needs to be accounted for somewhere in the system. If the gain is unacceptable, the polar magnitude may be multiplied by the reciprocal of the rotator gain before it is presented to the CORDIC rotator.

G. Extension to Hyperbolic Functions:

The close relationship between the trigonometric and hyperbolic functions suggests the same architecture can be used to compute the hyperbolic functions. While, there is early mention of using the CORDIC structure for hyperbolic transformations. The first description of the algorithm is that by [John S. Walther, 1971]. The CORDIC equations for hyperbolic rotations are derived using the same manipulations as those used to derive the rotation in the circular coordinate system. For rotation mode these are y and z registers (these registers could also be parallel loaded to be initialized). Once loaded, the data is shifted right through the serial adder subtractors and returned to the left end of the register. At the beginning of each iteration, the control state machine reads the sign of the y.

IV. Conclusion

CORDIC algorithm only performs shift and add operations and is therefore easy to implement and resource-friendly. However, when implementing the CORDIC algorithm one can choose between various design methodologies and must balance circuit complexity with respect to performance. The CORDIC algorithm is an example of an approach that is quite different from traditional math and which is very efficient. Due to its less computational complexity, CORDIC algorithm has become a good choice for hardware designers.

References

[1] Alvin M. Despain, "Fourier transform computers using CORDIC iterations", IEEE Transactions Computers, volume 23, no. C-10, pp. 993–1001, Oct.1974

[2] B. Das and S. Banerjee, "Unified CORDIC-based chip to realise DFT/DHT/DCT/DST," Proceedings IEE Computers and Digital Techniques, volume 149, no. 4, pp. 121–127, July 2002.

[3] Bimal Gisuthan and T. Srikanthan, "Pipelining flat CORDIC based trigonometric function generators," Microelectronics Journal, volume 33, pp.77–89, 2002

[4] Chuen-Yau Chen and Wen-Chih Liu, "Architecture for CORDIC algorithm realization without ROM lookup tables," in Proceedings 2003 International Symposium on Circuits and Systems, ISCAS'03, May 2003, volume 4, pp. 544–547.

[5] Dirk Timmermann, H. Hahn, and B. J. Hosticka, "Low latency time CORDIC algorithms," IEEE Transactions on Computers, volume 41, no. 8, pp.1010–1015,August 1992.

