

Simulink Design Of Pipelined CORDIC For Generation of Sine and Cosine Values

Richa Upadhyay¹, Dr. Nisha Sarwade², Shrugal Varde³

^{1,2,3}Electrical Department, V.J.T.I. Mumbai

Abstract

In recent researches, there are countless applications where sine and cosine wave are used, like in Physics, Digital Signal Processing for various transforms, several modulation and demodulation techniques etc. There are numerous ways to generate digital sine and cosine waves, the use of previously calculated tables is one of the choices, but it requires excessive memory usage when good quantization level is needed. CORDIC algorithm, on the other hand, offers an excellent alternative, and its best characteristic is flexibility. Its quantization accuracy is a function of word length. In this paper a simple Pipelined CORDIC structure for generation of sine and cosine values has been implemented and verified using Simulink tool by MATLAB.

Keywords: CORDIC, Simulink, Pipeline Architecture, Shift And Add, Trigonometric, Unscaled, HDL, Hardwired.

I. Introduction

The CORDIC is hardware-efficient algorithms for computation of trigonometric and other elementary functions by only shift and adds operations. The CORDIC set of algorithms for the computation of trigonometric functions was designed by Jack E. Volder in 1959 to help building a real-time navigational system for the B-58 bomber [1]. Later, J. Walther in 1971 extended the CORDIC scheme to other functions [2]. The CORDIC method of functional computation is used by most calculators (such as the ones by Texas Instruments and HP) to approximate the normal transcendental functions.

The popularity of CORDIC was very much enhanced there-after primarily due to its potential for efficient and low-cost implementation of a large class of applications which include: the generation of trigonometric, logarithmic and transcendental elementary functions; complex number multiplication, Eigen-value computation, matrix inversion, solution of linear systems and singular value decomposition (SVD)[4] for signal processing, Fourier and related Transforms[3], image processing, and general scientific computation. Some other popular and upcoming applications are:

- [1] Direct frequency synthesis, digital modulation and coding for speech/music synthesis and communication;
- [2] Direct and inverse kinematics computation for robot manipulation;
- [3] Planar and three-dimensional vector rotation for graphics and animation

Although CORDIC may not be the fastest technique to perform these operations, it is attractive due to the simplicity of its hardware implementation, since the same iterative algorithm could be used for all these applications using the basic shift-add operations. In this paper, our purpose is to demonstrate the utility of the simulation tool Simulink by MATLAB to implement the basic CORDIC algorithm for calculating the trigonometric values. Simulink, developed by MathWorks, is a data flow graphical programming language tool for modeling, simulating and analyzing multidomain dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the MATLAB and XILINX environment and can either drive them or be scripted from them. Simulink is widely used in control theory and digital signal processing for multidomain simulation and Model-Based Design. The remainder of this paper is organized as follows. In Section II, describes the principles of CORDIC operation, covering the elementary ideas from coordinate transformation to rotation mode and vectoring model operations. In section III, the implementation of CORDIC for calculation of sine and cosine values is discussed. The CORDIC architecture applied for this paper has been described in Section IV along with its implementation in Simulink. The conclusion along with future research directions are discussed in Section V.

2. Basic Cordic Algorithm

The CORDIC algorithm performs a planar rotation as shown in Fig 1. Graphically, planar rotation means transforming a vector (X_i, Y_i) into a new vector (X_j, Y_j) . Using a matrix form, a planar rotation for a vector of (X_i, Y_i) is defined as-

$$\begin{bmatrix} X_j \\ Y_j \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \quad (1)$$

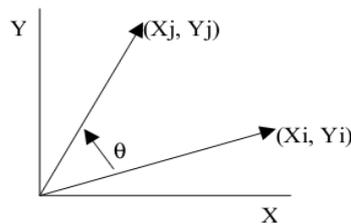


Fig-1: Planar rotation of a vector

The θ angle rotation can be executed in several steps, using an iterative process. Each step completes a small part of the rotation. Many steps will compose one planar rotation. A single step is defined by the following equation:

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \begin{bmatrix} \cos \theta_n & -\sin \theta_n \\ \sin \theta_n & \cos \theta_n \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} \quad (2)$$

Here n = no. of iterations. Equation 2 can be modified by eliminating the $\cos \theta_n$ factor.

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \cos \theta_n \begin{bmatrix} 1 & -\tan \theta_n \\ \tan \theta_n & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} \quad (3)$$

Equation 3 requires three multiplies, compared to the four needed in equation 2. Additional multipliers can be eliminated by selecting the angle steps such that the tangent of a step is a power of 2. Multiplying or dividing by a power of 2 can be implemented using a simple shift operation. The angle for each step is given by eq (4) and All iteration angles summed must equal the rotation angle θ as in eq (5).

$$\theta_n = \arctan \left(\frac{1}{2^n} \right) \quad (4)$$

$$\sum_{n=0}^{\infty} S_n \theta_n = \theta \quad (5)$$

Where $S = \{+1, -1\}$

This results in the following equation for $\tan \theta_n$

$$\tan \theta_n = S_n 2^{-n} \quad (7)$$

Combining equation 3 and 7 results in-

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \cos \theta_n \begin{bmatrix} 1 & -S_n 2^{-n} \\ S_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} \quad (8)$$

Besides for the $\cos \theta_n$ coefficient, the algorithm has been reduced to a few simple shifts and additions. The coefficient can be eliminated by pre-computing the final result. The first step is to rewrite the coefficient.

$$\cos \theta_n = \cos \left(\arctan \left(\frac{1}{2^n} \right) \right) \quad (9)$$

The second step is to compute equation 9 for all values of 'n' and multiplying the results, which we will refer to as K.

$$K = \frac{1}{P} = \prod_{n=0}^{\infty} \cos \left(\arctan \left(\frac{1}{2^n} \right) \right) \approx 0.60725 \quad (10)$$

K is constant for all initial vectors and for all values of the rotation angle; it is normally referred to as the congruence constant.

Because the coefficient K is pre-computed and taken into account at a later stage, equation 8 may be written as

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & -S_n 2^{-n} \\ S_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} \quad (11)$$

or as

$$X_{n+1} = X_n - S_n 2^{-n} Y_n, \quad Y_{n+1} = Y_n + S_n 2^{-n} X_n \quad (12)$$

At this point a new variable called 'Z' is introduced. Z represents the part of the angle θ which has not been rotated yet.

$$Z_{n+1} = \theta - \sum_{i=0}^n \theta_i \quad (13)$$

Combining equations 5 and 13 results in a system which reduces the not rotated part of angle θ to zero.

$$Z_{n+1} = Z_n - S_n \arctan \left(\frac{1}{2^n} \right) \quad (14)$$

Thus the basic CORDIC equations are –

$$X_{n+1} = X_n - S_n 2^{-n} Y_n$$

$$Y_{n+1} = Y_n + S_n 2^{-n} X_n$$

$$Z_{n+1} = Z_n - S_n \arctan \left(\frac{1}{2^n} \right)$$

So, the CORDIC method evaluates elementary functions merely by table-look-up, shift and add operations. A small number (of the order of n, where n bits of precision is required in the evaluation of the functions) of pre-calculated fixed constants is all that is required to be stored in the look-up table. The CORDIC algorithm has nice geometrical interpretations: trigonometric, exponential, multiply functions are evaluated via rotations in the circular, hyperbolic and linear coordinate systems, respectively. Their inverses (i.e., inverse trigonometric functions, logarithm and division) can be implemented in a “vectoring” mode in the appropriate coordinate system. Thus, there exist two modalities of CORDIC algorithm, **VECTORING** and **ROTATION** mode. In vectoring mode, coordinates (X_n, Y_n) are rotated until Y_n converges to zero. In rotation mode, initial vector (X_n, Y_n) starts aligned with the x axis and is rotated by an angle of θ_i every cycle, so after n iterations, θ_n is the obtained angle. Also,

$$S_n = \begin{bmatrix} \text{sign}(Z_n) & \text{for rotation mode} \\ -\text{sign}(Y_n) & \text{for vectoring mode} \end{bmatrix}$$

Hence, in ROTATION MODE

$$S_n = \begin{bmatrix} 1 & ; & Z_n > 0 \\ -1 & ; & Z_n \leq 0 \end{bmatrix}$$

And in VECTORING MODE

$$S_n = \begin{bmatrix} -1 & ; & Y_n > 0 \\ 1 & ; & Y_n \leq 0 \end{bmatrix}$$

3. CORDIC Theory for Trigonometric Functions

3.1 SINE AND COSINE

The rotation mode CORDIC operation can simultaneously compute the sine and cosine of the input angle. Setting the y component of the input vector to zero reduces the rotation mode result to:

$$x_n = \frac{1}{K} \cdot x_0 \cos z_0, \quad y_n = \frac{1}{K} \cdot x_0 \sin z_0$$

By setting x_0 to K the rotation produces unscaled sine and cosine of angle argument, z_0 . Very often sine cosine values modulate a magnitude value. Using other techniques (for example: look up table) requires a pair of multipliers to obtain the modulation. The CORDIC technique performs the multiply as a part of rotation operation, and therefore eliminates the need for a pair of explicit multipliers. The output of CORDIC rotator is scaled by the rotator gain. If the gain is not acceptable, a single multiply by the reciprocal of gain constant placed before CORDIC rotator will yield unscaled results. It is worth noting that the hardware complexity of CORDIC rotator is approximately similar to that of a single multiplier with the same word size.

Summary:

Mode: Rotation mode

Initial values: $[x, y, z] = [K, 0, \theta]$ (θ = input angle)

Direction: Reduce z to Zero

Output: $[x, y, z] = [\cos \theta, \sin \theta, 0]$

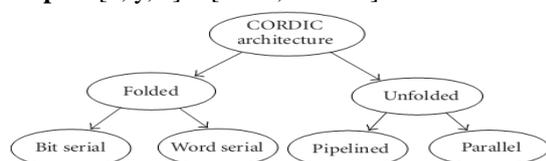


Fig-2: Taxonomy of CORDIC architectures

4. Proposed Cordic Architecture

As in fig-2, there are number of ways to implement CORDIC processor. The ideal architecture depends on the speed versus area tradeoff in the intended application. The CORDIC algorithm has traditionally been implemented using bit serial architecture with all iterations executed in the same hardware [6]. This slows down the computational device and hence, is not suitable for high speed implementation. For this paper we use a **Pipelined Architecture** [6, 10]. Instead of buffering the output of one iteration and using the same resources again, one could simply cascade the iterative CORDIC, which means rebuilding the basic CORDIC structure for each iteration. Consequently, the output of one stage is the input of the next one, as shown in Fig. 6, and in the face of separate stages two simplifications become possible. First, the shift operations for each step can be performed by wiring the connections between stages appropriately. Second, there is no need for changing constant values and those can therefore be hardwired as well. The purely unrolled design only consists of combinatorial components. Input values find their path through the architecture on their own and do not need to be controlled. While implementation a single iteration of the basic CORDIC structure like in fig-4, the Generic resource report which is obtained from the HDL code generator of Simulink is as follows:

Multipliers	0
Adders/Subtractors	6
Registers	0
RAMs	0
Multiplexers	6

Fig-3: Resource report of a Single CORDIC iteration

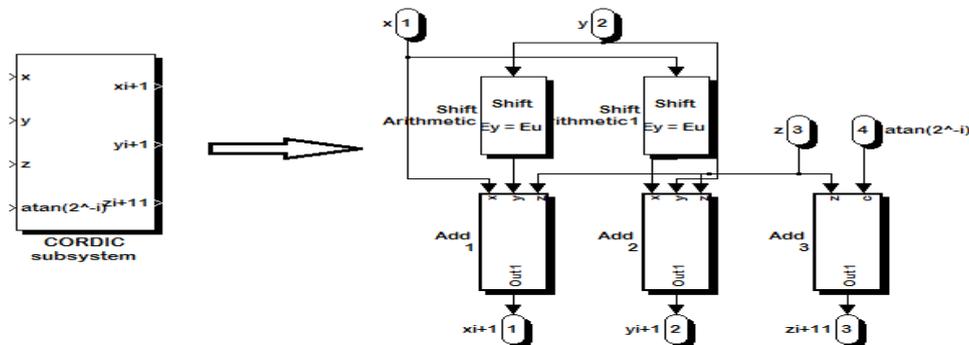


Fig-4: Hardware implementation of single CORDIC iteration

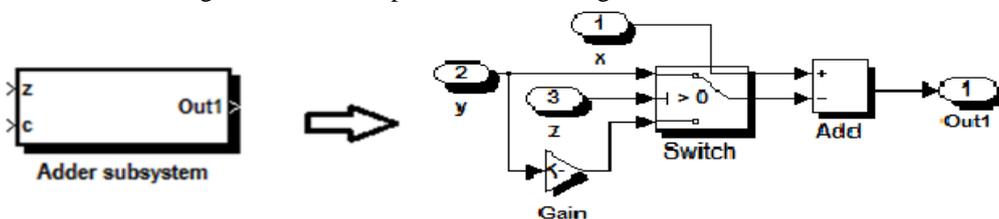


Fig-5: The Addition/Subtraction Logic Subsystem

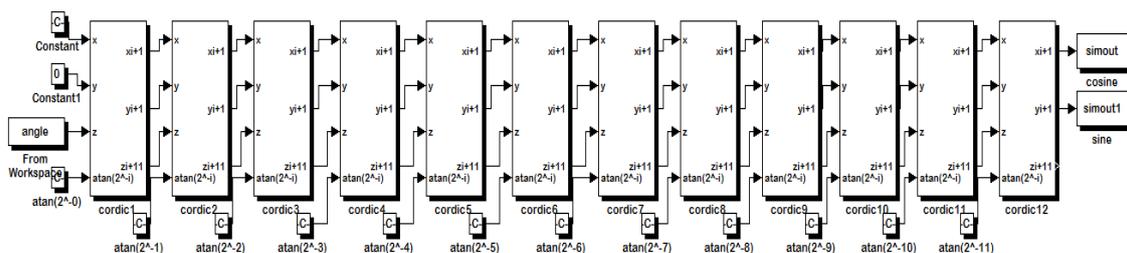
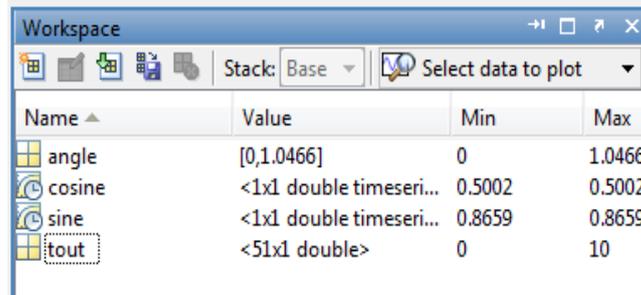


Fig-6: Pipeline architecture of CORDIC for sine and cosine.



Name	Value	Min	Max
angle	[0,1.0466]	0	1.0466
cosine	<1x1 double timeseri...	0.5002	0.5002
sine	<1x1 double timeseri...	0.8659	0.8659
tout	<51x1 double>	0	10

Fig-7: Output obtained on the workspace window

Here the input angle is 60° i.e. 1.0466 radians and

as shown in the figure above the output values attained are: $\sin 60^{\circ} = 0.8659$ and $\cos 60^{\circ} = 0.5002$.

5. Conclusion

In this paper simple calculation of sine and cosine of the input angles is done, CORDIC algorithm is implemented by using simple hardware through repeated shift-add operations and this is the feature which makes it attractive for a wide variety of applications. The code above implements a pipeline algorithm of a 12-stage CORDIC operating within the rotation mode. Here since the number of shifts to be performed by the shifters at different stages is fixed (shift-operation through i bit positions is performed at the i^{th} stage), the shift operations could be hardwired with adders.

The same is the case with the constant arctan (2^{-i}). This is to help reduce the no. of resources and also the latency of computation. In all of these ways, the CORDIC algorithm proves its merit as a simple but powerful algorithm that all FPGA designers should be aware of. Using CORDIC algorithm to generate these waveforms can, if correctly done, result in a high spurious-free dynamic range. Good SFDR performance is required for most signal-processing applications. Also, Simulink being a graphical language makes it easy to implement and understand the physical meaning of CORDIC. As future work the HDL code (can be VHDL or VERILOG) can be generated from the Simulink graphical program and can further be used on FPGA. Also the implementation of the CORDIC algorithm can be done on FPAA (Field Programmable Analog Array), the analog cousin of FPGA by using SIM2SPICE tool that automatically converts analog systems from Simulink design to Spice netlist and further the placing and routing can be done on FPAA with the help of other tools.

References

- [1] J. E. Volder, "The CORDIC Trigonometric Computing Technique," IRE Trans. on Electronic Computers, vol. EC-8, pp. 330-334, Sep. 1959.
- [2] J. S. Walther, "A unified Algorithm for Elementary Functions," in Proceedings of the 38th Spring Joint Computer Conference, pp. 379-385, 1971
- [3] A. M. Despain, "Fourier Transform Computers Using CORDIC Iterations," IEEE Transactions on Computers, vol. C-30, pp. 993-1001, Oct. 1974
- [4] J. R. Cavallaro and F. T. Luk, "CORDIC Arithmetic for a SVD processor," Journal of Parallel and Distributed Computing, vol. 5, pp. 271-290, 1988.
- [5] J. Duprat and J. Muller, "The CORDIC Algorithm: New Results for Fast VLSI Implementation," IEEE Transactions on Computers, vol. 42, no. 2, pp. 168-178, 1993.
- [6] J. P. Meher, J. Valls, T. Juang, K. Sridharan and K. Maharatna, "50 Years of CORDIC: Algorithms, Architectures, and Applications," IEEE Transactions on Circuits and Systems, vol. 56, no. 9, pp. 1893-1907, 2009.
- [7] E. O. Garcia, R. Cumplido, M. Arias, "Pipelined CORDIC Design on FPGA for a Digital Sine and Cosine Waves Generator," 3rd IEEE Transactions on Computers, vol. 59, no. 4, pp. 522-531, 2010.
- [8] R. Bhakthavathalu, M. Sinith, P. Nair and K. Jismi, "A Comparison of Pipelined Parallel and Iterative CORDIC Design on FPGA," International Conference on Industrial and Information Systems, pp. 239-243, 2010
- [9] Vladimirova, T. and Tiggler, "FPGA Implementation of Sine and Cosine Generators Using the CORDIC Algorithm", Proc. of Military and Aerospace Application of Programmable Devices and Technologies Conference (MAPLD 99), Sep. 1999, Laurel, MA, A-2, pp. 28-30.
- [10] Naveen Kumar and Amandeep Singh Sappal "Coordinate Rotation Digital Computer Algorithm: Design and Architectures", International Journal of Advanced Computer Science and Applications, Vol. 2, No. 4, 2011, pp 68-71.
- [11] "System Generator for DSP user guide", UG640 (v 12.1) April 19, 2010.