ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijareeie.com

Vol. 7, Issue 3, March 2018

Trigonometric Computation Using CORDIC Algorithm

Suresh H[1] , Roshni Oommen[2]

Graduate Associate, Dept. of ECE, Saintgits College of Engineering, Kottayam, Kearala, India[1]

Asst. Professor, Dept. of ECE, Saintgits College of Engineering, Kottayam, Kearala, India [2]

ABSTRACT*:* The CORDIC (Coordinate Rotation Digital Computer) algorithm is an iterative algorithm used for the computation of trigonometric functions, multiplication, division, data type conversion, square root and transcendental functions as those have wide range of applications. The simple way of calculating sine and cosine of an input angle using CORDIC algorithm in Verilog with fixed number of iterations is proposed. The algorithm uses predefined LUT (look-up table) for angle calculations.

KEYWORDS: CORDIC, sine, cosine, Verilog

I. INTRODUCTION

CORDIC is the full form of Co-ordinate Rotation Digital Computer. It is derived from the general equations of vector rotation. The CORDIC algorithm provides an iterative method of performing vector rotation by arbitrary angle using shift and adds. The CORDIC algorithm has become a widely used approach to elementary function evaluation when the silicon area is a primary constraint. The implementation of CORDIC algorithm requires less complex hardware than the conventional method. The CORDIC algorithm has found its way in various applications such as pocket calculators, numerical co-processors, to high performance radar signal processing, supersonic bomber aircraft with a digital counterpart, computation of the (Fast Fourier Transform) FFT, and at the effects on the numerical accuracy. CORDIC algorithm revolves around the idea of "rotating" the phase of a complex number, by multiplying it by a succession of constant.

II. CORDIC ALGORITHM

CORDIC algorithm is taken from the general equations for a vector rotation. If a vector V with co-ordinates (x, y) is resolved through an angle Ø then a new vector V' with modified coordinates (x', y') is obtained where x' and y' can be found using x, y and Ø from the subsequent method. For easy calculation only rotation in anticlockwise direction is considered. So the equations for x' and y' can be given as

$$X' = x\cos Ø - y\sin Ø$$

$$Y' = x\sin Ø + y\cos Ø$$

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Vol. 7, Issue 3, March 2018

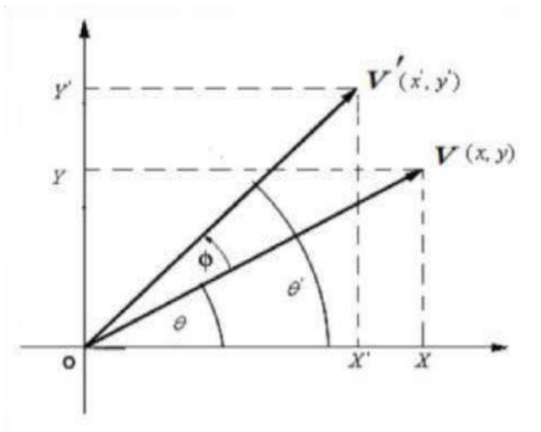


Figure 1. Rotation of Vector V(x,y)

By absorbing {cos Ø} from both sides, modified equation will be in form of the tangent of the angle Ø. Next if it is presumed that the angle Ø is being a cumulative of small angles, and composite angles is chosen such that their tangents are 2^{-i}, then this equation can be rephrased as an iterative information

$$X' = \cos Ø\,(x - y\tan Ø)$$

$$Y' = \cos Ø\,(x\tan Ø + y)$$

Where Ø is the angle of rotation.

The multiplication of tangent term can be escaped if the rotation angles and tan (Ø) are limited so that tan (Ø)=2^{-I} . In digital hardware, this signifies a simple shift process. If those rotations are phenomenon regularly.

With Ø = $\tan^{-1}(2^{-i})$

With the cosine term could also be simplified and since

$$\cos(Ø) = \cos(-Ø)$$

$$x_{i+1}=\cos(\alpha_i).[x_i-y_i.d_i.\tan(\alpha_i)]$$

$$y_{i+1}=\cos(\alpha_i).[y_i-y_i.d_i.\tan(\alpha_i)]$$

xi+1=ki[xi-yi.di.2-i]

$$y_{i+1}=k_i[y_i-x_i.d_i.2^{-i}]$$

$$k_i=\cos(\alpha_i)=\cos(\tan^{-1}(2^{-i}))$$

$$d_{i=}\pm1$$

Where 'i' denotes the number of rotations required reaching required angle of the required vector $k_i=\cos\ tan^{-1}\ (2)^{-i}$. The product of the k_i represent the k factor. k_i is the gain and its value changes as the number of rotation increases.Thus the multiplication is aggregated as

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Vol. 7, Issue 3, March 2018

$$k=\prod_{i=0}^{n} k_i \; ; \; n\to\alpha \; , \; k=0.607252935$$

i	2^i	$\Theta= tan^{-}(2)^{-i}$ in degrees	Decimal value
0	2^0	45	1048576
1	2^{-1}	26.56	619009
2	2^{-2}	14.03	327067
3	2^{-3}	7.125	166021
4	2^{-4}	3.57	82332
5	2^{-5}	1.78	41207
6	2^{-6}	0.89	20857
7	2^{-7}	0.44	10429
8	2^{-8}	0.22	5214
9	2^{-9}	0.11	2607
10	2^{-10}	0.05	1302

Table 1. Look Up Table

III. PROPOSED METHODOLOGY

A.Algorithm

The rotation-mode algorithm described above can rotate any vector (not only a unit vector aligned along the *x* axis) by an angle between –90° and +90°. Decisions on the direction of the rotation depend on being positive or negative. The vectoring-mode of operation requires a slight modification of the algorithm. It starts with a vector the *x* coordinate of which is positive and the *y* coordinate is arbitrary. Successive rotations have the goal of rotating the vector to the *x* axis (and therefore reducing the *y* coordinate to zero). At each step, the value of *y* determines the direction of the rotation. The final value of contains the total angle of rotation. The final value of *x* will be the magnitude of the original vector scaled by *K*. So, an obvious use of the vectoring mode is the transformation from rectangular to polar coordinates. Considering that z is the initial input angle. At each step, z tries to converge to '0'. The following are the steps for the algorithm

Step 1: Initialize x=0.60725, y=0, z= Ø
Step 2: For i=0 to n-1 then di = 1 when z>0 or else -1
Step 3:

- $y_{i+1}=k_i[y_i-x_i.d_i.2^{-i}]$
- $y_{i+1}=k_i[y_i-x_i.d_i.2^{-i}]$
- zi+1 = zi – di . αi

Step 4: check whether the N iterations are completed if completed
Step 5: Result: x_n=cos(Ø), y_n=sin(Ø)

B. Flow Chart

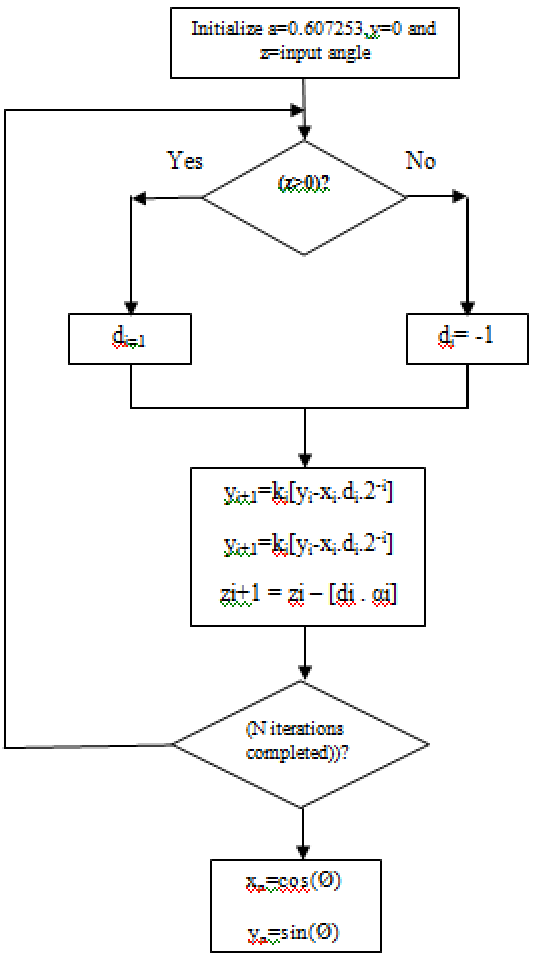

Initialize x=0.607253, y=0 and z=input angle
Yes
(z>0)?
No
$d_i=1$
$d_i= -1$
$y_{i+1}=k_i[y_i-x_i.d_i.2^{-i}]$
$y_{i+1}=k_i[y_i-x_i.d_i.2^{-i}]$
zi+1 = zi – [di . αi]
(N iterations completed))?
x_n=cos(Ø)
y_n=sin(Ø)

Figure 2. Flow chart of CORDIC algorithm

IV. SOFTWARE DESCRIPTION

Vivado Design Suite is a software suite produced by Xilinx for synthesis and analysis of HDL designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis. Vivado enables developers to synthesize (compile) their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Vivado is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors.

Components of Vivado include:

- The Vivado High-Level Synthesis compiler enables C, C++ and System programs to be directly targeted into Xilinx devices without the need to manually create RTL.
- The Vivado Simulator is a component of the Vivado Design Suite. It is a compiled- language simulator that supports mixed-language, TCL scripts, encrypted IP and enhanced verification.

- The Vivado IP Integrator allows engineers to quickly integrate and configure IP from the large Xilinx IP library. The Integrator is also tuned for Math Works Simulink designs built with Xilinx's System Generator and Vivado High-Level Synthesis.
- The Vivado TCL Store is a scripting system for developing add-ons to Vivado, and can be used to add to and modify Vivado's capabilities.

V. RESULT AND SIMULATION

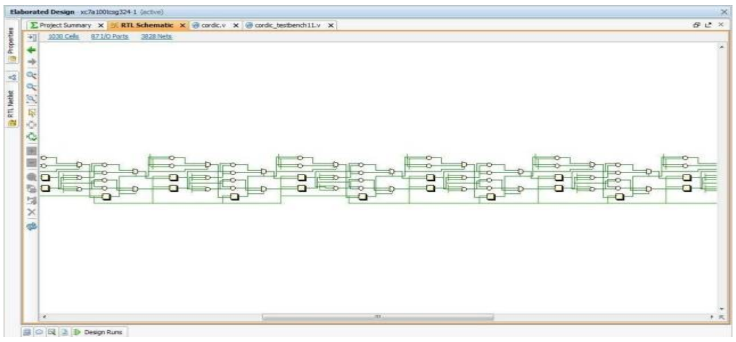
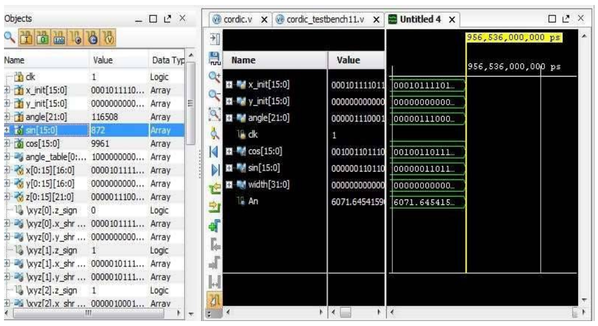


Figure 3.RTL Schematic of CORDIC Algorithm

Simulation:



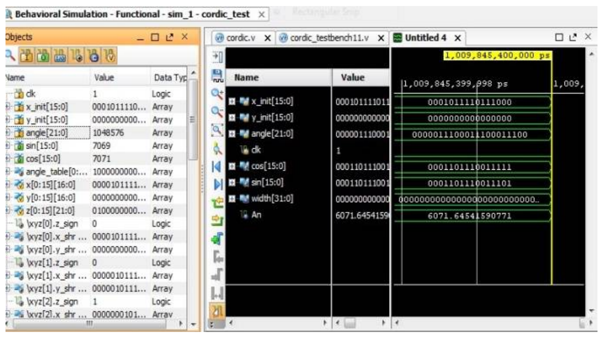
Objects

Name	Value	Data Typ
clk	1	Logic
x_init[15:0]	0001011110...	Array
y_init[15:0]	0000000000...	Array
angle[21:0]	116508	Array
sin[15:0]	872	Array
cos[15:0]	9961	Array
angle_table[0:...	1000000000...	Array
x[0:15][16:0]	0000101111...	Array
y[0:15][16:0]	0000000000...	Array
z[0:15][21:0]	0000011100...	Array
\xyz[0].z_sign	0	Logic
\xyz[0].x_shr ...	0000101111...	Array
\xyz[0].y_shr ...	0000000000...	Array
\xyz[1].z_sign	1	Logic
\xyz[1].x_shr ...	0000010111...	Array
\xyz[1].y_shr ...	0000010111...	Array
\xyz[2].z_sign	1	Logic
\xyz[2].x_shr ...	0000010001...	Array

cordic.v | cordic_testbench11.v | Untitled 4

```
956,536,000,000 ps
                                  956,536,000,000 ps
Name          Value
x_init[15:0]  000101111011   00010111101...
y_init[15:0]  000000000000   00000000000...
angle[21:0]   000001110001   00000111000...
clk           1
cos[15:0]     001001101110   00100110111...
sin[15:0]     000000110110   00000011011...
width[31:0]   000000000000   00000000000...
An            6071.6454159   6071.645415...
```

Figure 4.simulation result for Input = 5°



Behavioral Simulation - Functional - sim_1 - cordic_test

Objects

Name	Value	Data Typ
clk	1	Logic
x_init[15:0]	0001011110...	Array
y_init[15:0]	0000000000...	Array
angle[21:0]	1048576	Array
sin[15:0]	7069	Array
cos[15:0]	7071	Array
angle_table[0:...	1000000000...	Array
x[0:15][16:0]	0000101111...	Array
y[0:15][16:0]	0000000000...	Array
z[0:15][21:0]	0100000000...	Array
\xyz[0].z_sign	0	Logic
\xyz[0].x_shr ...	0000101111...	Array
\xyz[0].y_shr ...	0000000000...	Array
\xyz[1].z_sign	0	Logic
\xyz[1].x_shr ...	0000010111...	Array
\xyz[1].y_shr ...	0000010111...	Array
\xyz[2].z_sign	1	Logic
\xyz[2].x_shr ...	0000000101...	Array

cordic.v | cordic_testbench11.v | Untitled 4

```
                                 1,009,845,400,000 ps
Name          Value          1,009,845,399,998 ps          1,009,
x_init[15:0]  000101111011       0001011110111000
y_init[15:0]  000000000000       0000000000000000
angle[21:0]   000001110001     0000011100011100011100
clk           1
cos[15:0]     000110111001       0001101110011111
sin[15:0]     000110111001       0001101110011101
width[31:0]   000000000000   000000000000000000000000000...
An            6071.6454159       6071.64541590771
```

Figure 5.simulation result for Input = 45°

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Vol. 7, Issue 3, March 2018

VI. CONCLUSION

Efficient method to calculate sine and cosine is implemented using Verilog coding. Applications in several diverse areas including signal processing, image processing, communication, robotics and graphics apart from general scientific and technical computations have been explored.

REFERENCES

[1] Ranjita Naik," Sine-Cosine Computation Using CORDIC Algorithm", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 4, Issue 9, September 2015

[2] M.Chakrapani," Implementation of Cordic Algorithm for FPGA Based Computers Using Verilog", International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol. 3, Issue 8, August2014

[3] Volder, J.E. (1959)," The CORDIC Trigonometric Computation Technique" , IRE Transactions on Electronic Computers 8 (3): 330–334, retrieved 2009-06-02

[4] Walther, J.S. (1971), " A unified Algorithm for Elementary Function" (w), Proceedings of the May 18–20, 1971, spring joint computer conference: 379–385, retrieved 2009-06-02

[5] Chih-Hsiu Lin, An-Yeu Wu, "Mixed-scaling-rotation CORDIC (MSR-CORDIC) algorithm and architecture for highperformance vector rotational DSP applications", *Circuits and Systems I: Regular Papers IEEE Transactions on*, vol. 52, no. 11, pp. 2385-2396, Nov. 2005.

[6]Cheng-Shing Wu, An-Yeu Wu, "Modified vector rotational CORDIC (MVR-CORDIC) algorithm and architecture", *Circuits and Systems II: Analog and Digital Signal Processing IEEE Transactions on*, vol. 48, no. 6, pp. 548-561, Jun 2001.

[7]Xu Li, Wang Qin, "CORDIC based algorithm for frequency offset estimation", *Communication Technology (ICCT) 2010 12th IEEE International Conference on*, pp. 817-820, 11-14 Nov. 2010.

[8]M. Chinnathambi, N. Bharanidharan, S. Rajaram, "FPGA implementation of fast and area efficient CORDIC algorithm", *Communication and Network Technologies (ICCNT) 2014 International Conference on*, pp. 228-232, 18-19 Dec. 2014.

[9]Mane, D. Patil, M.S. Sutaone, A. Sadalage, "Implementation of DCT using variable iterations CORDIC algorithm on FPGA", *Computational Systems and Communications (ICCSC) 2014 First International Conference on*, pp. 379-383, 17-18 Dec. 2014.

[10]Sharma, P.N. Ravichandran, S. Kulkami, M. Vanitha, P. Lakshminarsimahan, "Implementation of Para-CORDIC Algorithm and Its Applications in Satellite Communication", *Advances in Recent Technologies in Communication and Computing 2009. ARTCom '09. International Conference on*, pp. 266-270, 27-28 Oct. 2009.

[11]. Pongyupinpanich, F.A. Samman, M. Glesner, S. Singhaniyom, "Design and evaluation of a floating-point division operator based on CORDIC algorithm", *Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTI-CON) 2012 9th International Conference on*, pp. 1-4, 16-18 May 2012.

[12]Ma Jun, K.K. Parhi, E.F. Deprettere, "Annihilation-reordering look-ahead pipelined CORDIC-based RLS adaptive filters and their application to adaptive beamforming", *Signal Processing IEEE Transactions on*, vol. 48, no. 8, pp. 2414-2431, Aug 2000.

[13]An-Yeu Wu, Cheng-Stung Wu, "A unified VIew for vector rotational CORDIC algorithms and architectures based on angle quantization approach", *Circuits and Systems I: Fundamental Theory and Applications IEEE Transactions on*, vol. 49, no. 10, pp. 1442-1456, Oct 2002.