SINE WAVE GENERATOR USING CORDIC ALGORITHM

Md. Aamir Sohail (EE16BTECH11021),
Piyush Rajan Udhan (EE16BTECH11028)

EE5811: FPGA LAB
&
EE3025: Independent Project

Department of Electrical Engineering,
IIT HYDERABAD

1. Abstract

In this report, the authors have done the hardware implementation of sine wave generator using ***CORDIC*** *Algorithm on "Ico-Board" FPGA and demonstrated the output on the "Analog Discovery" digital Oscilloscope. Basically, CORDIC Algorithm is a 'shift and add' algorithm used for implementing countless transcendental functions like trigonometric,hyperbolic, expotential and logarithmic.*

Keywords: **CORDIC**, **RTL**, **Rotation Mode**

2. INTRODUCTION

CORDIC stand for **CO**ordinate **R**otation **D**igital **C**omputer. In 1959, Jack Volder [1] first proposed this algorithm. In his thesis, he proposed an efficient way of calculating trigonometric function. John Walther [2] and others extended the CORDIC theory to provide solutions to a wider range of functions like transcedental functions.

This paper has been divided into five parts.

1. Unified CORDIC algorithm
2. Implementation of algorithm in MATLAB & C
3. Implementation of algorithm in RTL
4. Sine wave generator implementation in RTL
5. Hardware implementation of Sine Wave Generator

3. CORDIC Algorithm

3.1. Observation

If a unit vector with co-ordinates $(x_1, y_1) = (1, 0)$ is rotated by an angle θ, its new co-ordinate will be $(x_2, y_2) = (\cos\theta, \sin\theta)$. Thus, by finding the (x_2, y_2), $\cos\theta, \sin\theta$ can easily be computed.

3.2. Pseudorotations

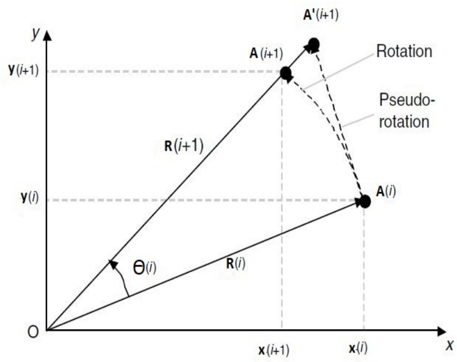


Figure 1: pseudorotation step in CORDIC [3]

Pseudorotation step increases the length of the vector $R(i)$ to

$$R(i+1) = R(i)(1 + \tan^2\theta(i))^{1/2}$$

The coordinates of the new point $A'(i+1)$ after pseudorotation are given by the set of equations:

$$x(i+1) = x(i) - y(i)\tan\theta(i) \tag{1}$$

$$y(i+1) = y(i) + x(i)\tan\theta(i) \tag{2}$$

$$\alpha(i+1) = \alpha(i) - \theta(i) \tag{3}$$

After n pseudorotations by the angle $\theta(1), \theta(2), \cdots, \theta(n)$ with $x(0) = x, y(0) = y$ and $\alpha(0) = \alpha$ we will get,

$$x(n) = \mathbf{K}\,(x\cos(\sum\theta(i)) - y\sin(\sum\theta(i)) \tag{4}$$

$$y(n) = \mathbf{K}\,(y\cos(\sum\theta(i)) + x\sin(\sum\theta(i)) \tag{5}$$

$$\alpha(n) = \alpha - \sum\theta(i) \tag{6}$$

where $\mathbf{K} = \prod(1+\tan^2\theta(i))^{1/2}$

3.3. CORDIC angle

Each pseudorotations should be choosen in such a way that,the **tan** values of these are just bit shifts (i.e divided by the power of two). **A bit shift is a much easier instruction for a CPU to deal with than full integer division.**

$$\theta(i) = \tan^{-1}(d_i\,2^{-i}),\ \ d_i \in \{+1,-1\} \tag{7}$$

RULE: Choose $d_i \in \{+1,-1\}$ such that $\alpha(n) \to 0$

3.4. CORDIC iteration

Thus eqn. 1,2,3 can be written as:

$$x(i+1) = x(i) - d_i\,y(i)\,2^{-i} \tag{8}$$

$$y(i+1) = y(i) + d_i\,x(i)\,2^{-i} \tag{9}$$

$$\alpha(i+1) = \alpha(i) - d_i\,\tan^{-1}2^{-i} \tag{10}$$

Each CORDIC iteration associates three addition,two shifts and a table lookup (it contains a list of precomputed cordic angles). If we always pseudorotate the vector by the same set of CORDIC angles either with positive or negative signs, then the value of scaling factor **K** can be pre-determine and approaches 1.646760258121 after sufficiently large number of iterations. After n pseudorotation steps, when $\alpha(n)$ is well enough close to zero, we will get $\sum\theta(i) = \alpha$.

Finally the CORDIC iterations in ROTATION MODE become:

$$x(n) = \mathbf{K}\,(x\cos\alpha - y\sin\alpha) \tag{11}$$

$$y(n) = \mathbf{K}\,(y\cos\alpha + x\sin\alpha) \tag{12}$$

$$\alpha(n) = 0 \tag{13}$$

3.5. Computation of trigonometric functions

From the eqn. 11,12,13 we observe that if we start with $x(0) = 1/K$ and $y(0) = 0$, then after 'n' pseudorotation steps as $\alpha(n) \to 0$, the $x(n) \to \cos\alpha$ and $y(n) \to \sin\alpha$ with CORDIC iterations in rotation mode.

The domain of convergence is $-99.7° \leq \alpha \leq 99.7°$.

3.6. Implementation in MATLAB & C

Circular Rotation (Basic **CORDIC**)

Algorithm 1 cordic (angle in degree α)

Input: (angle = α)
Output: $(x(n), y(n), \alpha(n)) = (\cos\alpha, \sin\alpha, 0)$
Initialisation: (x(0), y(0), α(0)) = (1/K, 0,α)
$\mu = 1$

%%% Range -pi to pi %%%
if ($\alpha < -90^o$) || ($\alpha > 90^o$) **then**
  if ($\alpha < 0^o$) **then**
    cordic (α + 180^o)
  else
    cordic (α - 180^o)
  end if
  Result $\leftarrow \begin{pmatrix} x(n) \\ y(n) \end{pmatrix}$ = - $\begin{pmatrix} x(n) \\ y(n) \end{pmatrix}$
  return Result
end if
%%% CORDIC iteration %%%
for i = [0:N-1] **do**
  $\theta(i) \leftarrow \tan^{-1}(2^{-i})$ % from lookup table
  if ($\alpha(i) < 0$) **then**
    $d_i \leftarrow (-1)$
  else
    $d_i \leftarrow (+1)$
  end if
  R $\leftarrow \begin{pmatrix} 1 & -\mu\,d_i\,2^{-i} \\ d_i\,2^{-i} & 1 \end{pmatrix}$
  $\begin{pmatrix} x(i+1) \\ y(i+1) \end{pmatrix} \leftarrow R * \begin{pmatrix} x(i) \\ y(i) \end{pmatrix}$
  $\alpha(i+1) \leftarrow \alpha(i) - d_i\,\theta(i)$
end for
Result $\leftarrow \begin{pmatrix} x(n) \\ y(n) \end{pmatrix}$
return *Result*

C-code were executed for 32 bits of precision in the resulting values, thus 32 CORDIC iterations. Trig- functions graph was simulated using MATLAB.

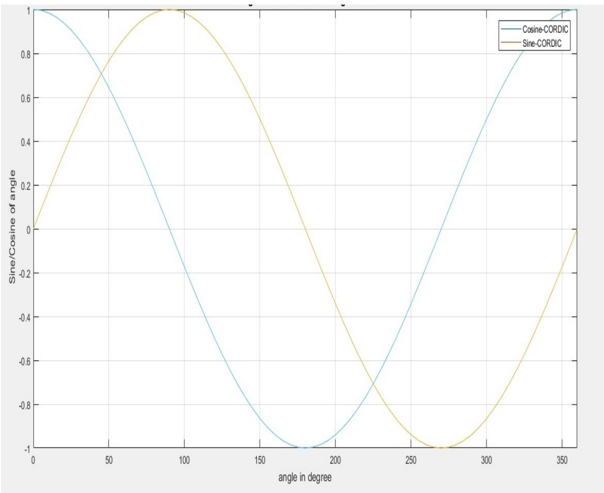


Figure 2: Trigonometric Functions using CORDIC

4. Unified CORDIC Algorithm

J.S Walther [2] improved the CORDIC iteration given by Volder by introducing a system parameter 'μ'. He proposed "Generalised CORDIC iteration" which can used to compute function belongs to three different co-ordinate system i.e Circular, Hyperbolic and Linear.

Generalised CORDIC iteration

$$x(i+1) = x(i) - \mu\, d_i\, y(i)\, 2^{-i} \tag{14}$$

$$y(i+1) = y(i) + d_i\, x(i)\, 2^{-i} \tag{15}$$

$$\alpha(i+1) = \alpha(i) - d_i\, \theta(i) \tag{16}$$

Coordinate system	μ	θ**(i)**
Circular	+1	$\tan^{-1}(2^{-i})$
Hyperbolic	-1	$\tanh^{-1}(2^{-i})$
Linear	0	2^{-i}

RULE:

$$d_i = \begin{cases} +1 & \alpha(i) \geq 0 \\ -1 & \alpha(i) < 0 \end{cases}$$

4.1. Computation of hyperbolic functions

The following equations defines the CORDIC Algorithm for hyperbolic functions:

$$x(n) = \mathbf{K}'\,(x\cosh\alpha + y\sinh\alpha) \tag{17}$$

$$y(n) = \mathbf{K}'\,(y\cosh\alpha + x\sinh\alpha) \tag{18}$$

$$\alpha(n) = 0 \tag{19}$$

Thus, if we start with x(1) = 1/$\mathbf{K}'$ and y(1) = 0 (iteration can not be start from '0', since $\tanh^{-1}(2^{-i})$ does not exist), then after 'n-1' hyperbolic CORDIC rotation steps as $\alpha(n) \to 0$, the $x(n) \to \cosh\alpha$ and $y(n) \to \sinh\alpha$ with hyperbolic CORDIC iterations in rotation mode. Thus, $\tanh\alpha$ can be computed.

4.2. Convergence of hyperbolic CORDIC iteration

Hyperbolic function does not converge with the sequence of CORDIC angles $\tanh^{-1}(2^{-i})$, since

$$\tanh^{-1}(2^{-(i+1)}) \geq 0.5\, \tanh^{-1}(2^{-i}) \tag{20}$$

does not hold in general [4]. To ensure convergence the iterations $i = 4, 13, 40, \cdots, j, 3j+1, \cdots$ must be executed twice. Thus, we get domain of convergence $|\alpha| < 1.13$. where $\mathbf{K}'$ = 0.828159361 after considering the repeated iterations.

4.3. Implementation in MATLAB & C

Hyperbolic CORDIC rotations

Algorithm 2 cordic_hyper (α)

Input: (angle = α)
Output: $(x(n), y(n), \alpha(n)) = (\cosh\alpha, \sinh\alpha, 0)$
Initialisation : (x(1), y(1), $\alpha(1)$) = (1/K$'$, 0, α)
$\mu = -1$

```
%%%     CORDIC iteration     %%%
itr =[4, 13, ··· , j, 3j + 1, ···]
for i = [1:N , itr] do
    θ(i) ← tanh^{-1}(2^{-i})     % from lookup table
    if (α(i) < 0) then
        d_i ← (−1)
    else
        d_i ← (+1)
    end if
    R ← ( 1          −μ d_i 2^{-i} )
        ( d_i 2^{-i}   1           )
    ( x(i+1) ) ← R * ( x(i) )
    ( y(i+1) )       ( y(i) )
    α(i+1) ← α(i) − d_i θ(i)
end for
Result ← y(n) / x(n)
return Result
```

MATLAB code for hyperbolic CORDIC rotation algorithm was executed for various values of iteration count and the error between the simulated and theoritical was observed.

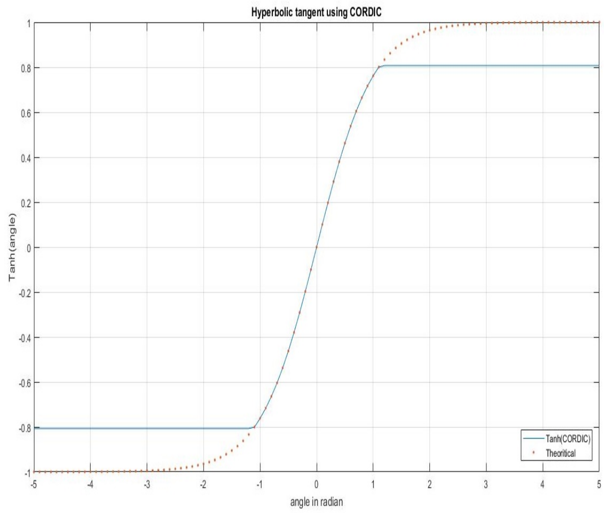


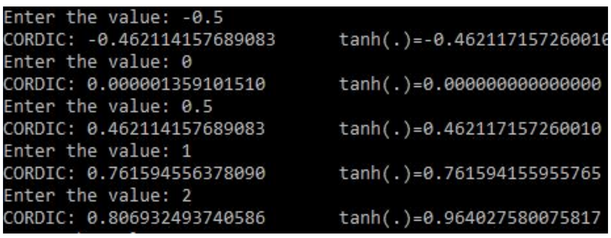
Figure 3: Simulated result i = {8,16,32,64}

From the fig. 3, after the improvement in iteration we get the domain of convergence $|\alpha| < 1.13$ (same as we see in section 4.2).

4.4. Error analysis

Iteration count	Error		
	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$
8	9.3×10^{-5}	1.4×10^{-3}	-1.3×10^{-3}
16	-1.4×10^{-5}	7.4×10^{-6}	4.0×10^{-7}
32	-6.3×10^{-10}	5.0×10^{-10}	8.8×10^{-10}
64	-8.6×10^{-10}	6.8×10^{-10}	9.8×10^{-10}

From the above table, it is clear that error almost saturates after the iteration count 32. Hence, in the **C-** code, the iteration count is taken 32. C-code for hyperbolic tangent function using CORDIC was written and executed for different values of angle and also compared with theoritical value using **math.h** library.



```
Enter the value: -0.5
CORDIC: -0.462114157689083      tanh(.)=-0.46211715726001
Enter the value: 0
CORDIC: 0.000001359101510       tanh(.)=0.000000000000000
Enter the value: 0.5
CORDIC: 0.462114157689083       tanh(.)=0.462117157260010
Enter the value: 1
CORDIC: 0.761594556378090       tanh(.)=0.761594155955765
Enter the value: 2
CORDIC: 0.806932493740586       tanh(.)=0.964027580075817
```

Figure 4: Simulated result i = 32

4.5. Improving domain of convergence

Each iterations are repeated with a same repetition factor (**R**) of an even multiple to increase the domain of convergence. MATLAB code for the improved iterations is executed and the following graph has been observed.

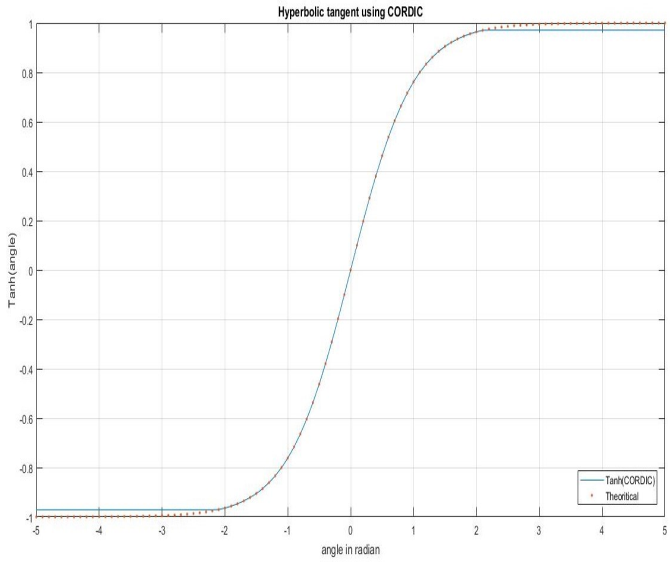


Figure 5: Simulated result i = 32 and R = 2

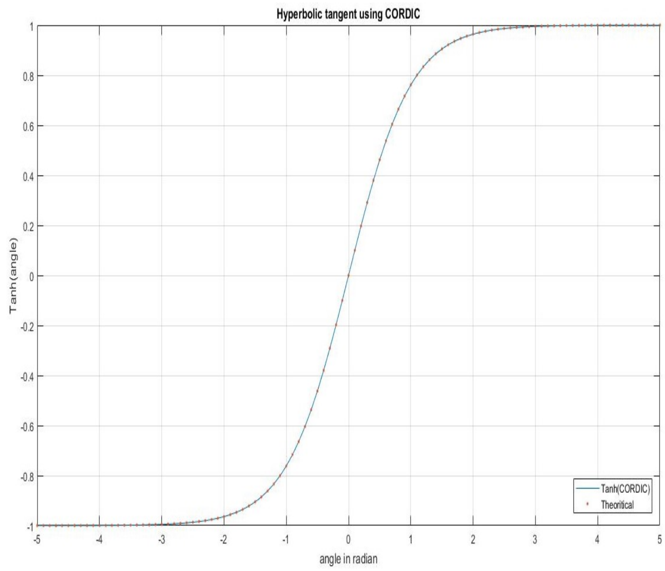


Figure 6: Simulated result i = 32 and R = 4

4.5.1. Conclusion: Error analysis.

		Error		
R	i	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$
2	16	0.0	-6.6×10^{-6}	8.2×10^{-7}
	32	0.0	-6.7×10^{-10}	1.7×10^{-10}
	64	0.0	-4.4×10^{-10}	2.8×10^{-10}
4	16	0.0	-6.6×10^{-6}	8.2×10^{-7}
	32	0.0	-6.7×10^{-10}	1.7×10^{-10}
	64	0.0	-4.4×10^{-10}	2.8×10^{-10}

From the above table, it is clear that the error remains almost same as compared to the **Table 1.0** except for α = 0. But the hyperbolic domain of convergence for 32 bits of precision in the resulting values, increases from $|\alpha| < 1.13$ (basic iterations) to $|\alpha| < 2.1$ (R = 2) and $\alpha \in \Re$ (R = 4).

5. Implementation in RTL

The Verilog platform has been used for the implementation of CORDIC Algorithm in RTL.

5.1. Circular CORDIC: Trig-functions

The digital design should be able to compute Sine and Cosine of the input angles $\in [0, 2\pi]$ including the floating point angles like 89.45^o, 29.7^o etc.
A 16-bit binary scaling system has been used to represent angles [5]. The resolution of the design is $360^o/2^{16}$ = $5.493\,164\,063 \times 10^{-3}$. The input angle is scaled to fit in a 16-bit register and user must convert the input angle to $[0, (2^{16} - 1)]$.

To convert the angle from degrees to 16-bit value, multiply the angle^o by 2^{16}, then divide it 360^o. FInally, convert the decimal value to binary. The angle is represented by 16-bit format. The upper two bits represent the quadrant [6].

1. 2b'00 = represents I quadrant i.e (0 - π/2) range
2. 2b'01 = represents II quadrant i.e (π/2 - π) range
3. 2b'10 = represents III quadrant i.e (π - 3π/2) range
4. 2b'11 = represents IV quadrant i.e (3π/2 - 2π) range

Total 8 STAGES have been used (i.e 8-bits of accuracy in the resulting values), which consist of a Pre-rotation STAGE (STAGE 1) to make sure that the rotation angle must lie within $-\pi/2$ to $\pi/2$ range (see section 3.5). The size of the Output data is 8-bits. Clock Frequency of 100MHz was used.

Testbench was written and tested for different values of input angle using Xilinx ISE Design Suite 14.7 (ISim) software. After 7-clock cycles, Output Data was displayed on the simulator screen. Ouput value is scaled by G = 75*GainFactor$_{CORDIC}$ times. Simulation Result: angle = 30^o -> 16 - bit binary = 0001010101010101, Output = 00111110 i.e 62/G = 0.5020141293).

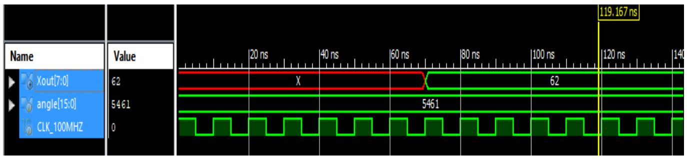


Figure 7: $\alpha = 30^o$

5.2. SINE Wave Generator

Xilinx ISE Design Suite 14.7 and ModelSim PE Student Edition 10.4a software were used to test the Verilog Code and simulate the SINE WAVE GENRATOR module from 0 to 2π respectively. The simulation result is shown below:

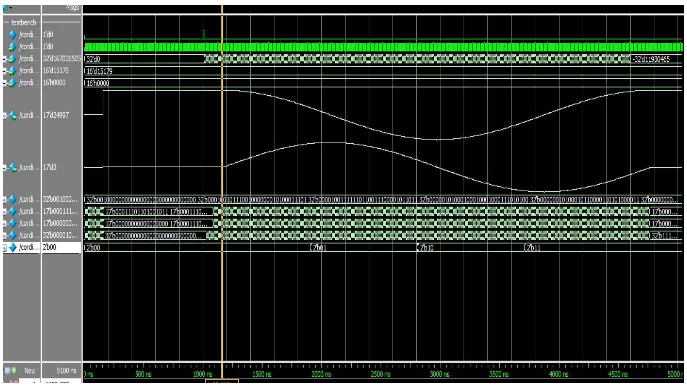


Figure 8: Simulation Result (ModelSim)

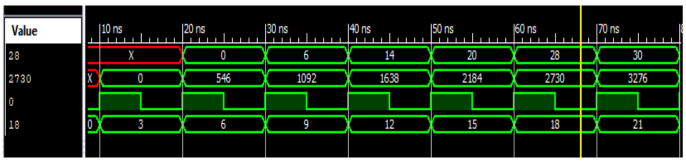


Figure 9: $\alpha \in [0, 2\pi]$

6. Hardware Implementation

The major components used for hardware implementation are as follows:

1. Ico-Board FPGA
2. Arduino MEGA 2560
3. R2R DAC
4. 'Analog Discovery'- Digital Oscilloscope

6.1. R2R DAC circuit

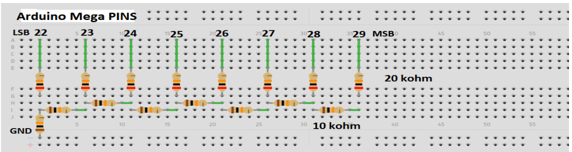


Figure 10: 8-bit 'R2R' DAC Circuit

6.2. Circuit and System setup

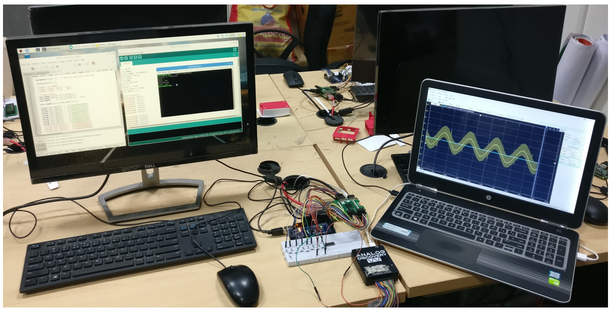


Figure 11: System setup

6.3. Digital Oscilloscope

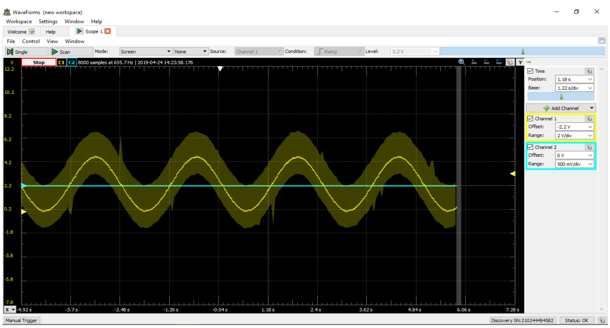


Figure 12: Output Waveform

7. Future Work

Errors in the output like presence of steps in the sine waveform can be reduced by increasing the bit size. Adding a filter in the DAC circuit will help obtain a smoother waveform. Apart from rectifying the quantisation error, we can also extend the project to generate sine waves of desired frequencies.

ACKNOWLEDGMENT

We would like to express special gratitude to our guide G.V.V. Sharma (Associate Professor, IIT Hyderabad) for giving us such a big opportunity and his meritorious guidance. We also wish to express a deep sense of gratitude to Teaching Assistants of the course for monitoring and helping us throughout the project term.

References

[1] Volder J., The CORDIC trigonometric computing technique,IRE Trans. Electronic Computing, Vol ED-8,pp330-334 Sept 1959.

[2] Walther J.S. , A unified algorithm for elementary functions, Spring Joint Computer Conf,1971 proc pp379-385.

[3] Behrooz.Parhami., COMPUTER ARITHMETIC-Algorithms and Hardware Designs 2nd edition., OXFORD UNIVERSITY PRESS, 2010, ch.22.

[4] Hsiao S.F.,The CORDIC householder algorithm, Proceedings of 10th symposium on computer arithmetic pp256-263, 1991.

[5] Mitu Raj, Floatingpoint-Numbers-BinaryLogic, CORDIC-ALGORITHM-USING-VHDL , www.instructables.com

[6] Kirk Weedman, CORDIC Design and Simulation using Verilog, www.hdlexpress.com