Summer Project Report

Implementation of CORDIC Algorithms in FPGA

Sidharth Thomas
Suyash Mahar

under the guidance of
Dr. Bishnu Prasad Das

May 2017

Department of Electronics and Communication Engineering
Indian Institute of Technology, Roorkee

Contents

Acknowledgement
We would like to take this opportunity to thank Dr. Bishnu Prasad Das for giving us the opportunity to do a project on *Implementation of CORDIC algorithms in FPGA*.

We would also like to thank IIT Roorkee IEEE students branch for reviewing our report.

Introduction

In this project we have implemented CORDIC algorithms on Spartan 3E FPGA.

This project has been done under the guidance of Dr. Bishnu Prasad Das of **Electronics and Communication Engineering Department, IIT Roorkee**. The project was started on **May 2, 2017** and was continued till **May 28, 2017**. During the course of the project, following things were done:

- Python and MatLab description of CORDIC algorithms (link for Jupyter implmentation : https://github.com/suyashmahar/cordic-algorithm-python/blob/master/cordic_implementation.ipynb)
- Writing complete hardware description of CORDIC algorithms in Verilog
- Implementation and verification of CORDIC algorithms on Spartan 3E FPGA

1. CORDIC Algorithms

1.1 Introduction

CORDIC ,**CO**ordinate **R**otation **DI**gital **C**omputer, also known popularly as Volder's algorithm, is an efficient algorithm to compute popular linear, circular as well as hyperbolic functions, at a lower hardware cost. CORDIC algorithms take advantage of the properties of rotation of vectors.

The algorithm, as in present form was first described by Volder in 1959 in his paper `The CORDIC Trigonometric Computing Technique`[3]. They were first developed to replace analog resolver in B-58 bombers with digital solution. CORDIC algorithms can even be extended to compute various transcendental functions to an appreciable amount of accuracy.

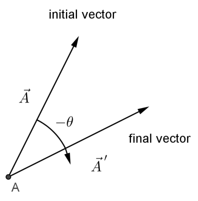

Figure 1.1: Rotation of Vector.

The popularity of CORDIC comes from its possibilities when no hardware multipliers are available. It requires just simple additions or subtractions and bit-shifts. A look-up table is also required. CORDIC algorithms typically converge with one bit per iteration [1] and belong to the class of `digit by digit algorithms`.

1.2 Working

CORDIC algorithms works by rotating a vector in space. All trigonometric functions can essentially be derived from a series of vector rotations.
Suppose x and y are the initial co-ordinates of a vector in 2-dimensional space. Let the vector denoted by these co-ordinates be rotated through an

angle ϕ. Then the new position coordinates of the vector is given by the formulae:

$$x' = x\cos(\phi) - y\sin(\phi) \tag{1.1}$$

$$y' = x\sin(\phi) + y\cos(\phi) \tag{1.2}$$

The above given formulae forms the basis for Volder's[3] calculations. These formulae can be further simplified as follows:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{1.3}$$

In other words:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{1}{\sqrt{1+\tan^2(\phi)}} \begin{bmatrix} 1 & -\tan(\phi) \\ \tan(\phi) & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{1.4}$$

This equation has so far not reduced the hardware cost in any amount. But if we limit the value of the angle ϕ [1] such that the tangent of the angle is always of the form $\pm 2^{-i}$, then the multiplications involved in eqn(1.4) can be reduced to bit shift operations. This is a much cheaper solution in terms of the hardware cost and complexity. So after applying the above mentioned conditions, our *modified* CORDIC algorithm would take the form:

$$x' = K(x \pm 2^{-i}y) \tag{1.5}$$

$$y' = K(y \pm 2^{-i}x) \tag{1.6}$$

The $\pm$ arithmetic operator can be handled by using a new decision variable d such that d $\in$ {-1,1} for all the iterations.

If i denotes the iteration count, then the values of x and y after i iterations can be given as:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = K_i \begin{bmatrix} 1 & -\tan(\phi) \\ \tan(\phi) & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \tag{1.7}$$

$$K_i = \prod_{n=0}^{i-1} \frac{1}{\sqrt{1+2^{-n}}} \tag{1.8}$$

This value $K_i \to 0.6073$ as $i \to \infty$.[1]. Fig. 1.2 shows variation of system gain with iteration.

Ignoring the value of K_i will be the final step to convert the algorithm to only additions, subtractions and bit shifts. Then, instead of pure rotations, we are performing *pseudo-rotations*. K_i can then be treated as the gain of the circuit[1] or the inputs can be adjusted in order to scale the gain down to unity.

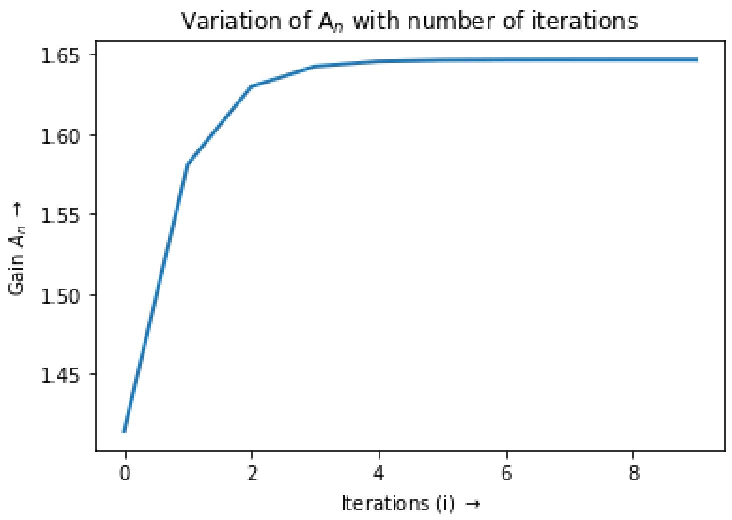

Figure 1.2: Variation of system gain for Circular rotation mode with each iteration

Suppose the vector was initially located at an angle z to the positive x-axis. In each iteration, we are incrementing the angle by $d_i \cdot tan^{-1}(2^{-i})$.So after i number of iterations, the angle θ_i would be given by[4] :

$$z_i = z - \sum_{n=0}^{i-1} d \tan^{-1}(2^{-i}) \tag{1.9}$$

The value of the decision variable d depends on the mode of operation of CORDIC algorithm, which will be explained in a later section. Also, a look-up table can be used in order to obtain the values of the inverse tangents.[1]

Based on the required transcendental function, CORDIC algorithms broadly works in two modes:

- Rotation Operation Mode
- Vector Operation Mode

Of these, *rotation operation mode* basically deals with each iteration trying to bring an original vector close to a given vector, while the *vector operation mode* deals with the initial vector, trying to approach the positive x-axis after each subsequent iteration.

Furthermore, each of these modes can further operate in *three* different coordinate systems, that can result in calculating the values of more transcendental functions. They are:

- Linear Coordinate System

- Circular Coordinate System
- Hyperbolic Coordinate System

The calculations mentioned in this section are mostly general, but some of the specifics like the value of K_i has been given, keeping circular mode in mind. A detailed description of the other modes will be given in the coming sections. Also the CORDIC algorithms have been found to operate within a certain convergence range[4].

1.3 CORDIC Operation Modes

The two operation modes of CORDIC is fundamentally just the principle of vector rotation. But both these modes operate in a different way that provides the results of different transcendental functions.

1.3.1 Rotation Operation Mode

In rotation mode, basically we rotate the given vector such that it tries to approach a certain angle, after each iteration. The inputs to be given in rotation mode is given in Table 1.1.

Co-ordinate System	x_{input}	y_{input}	z_{input}
Linear	x	0	z
Circular	0.6073	0	z
Hyperbolic	1.2075	0	z

Table 1.1: Table Containing the Inputs for Rotation Mode

What is striking with these inputs is that the value of y is always zero. This suggests a vector lying along the x-axis. The length of the vector, would then be equal to its x co-ordinate value. This x value is actually the scaled values, which have been taken into account considering the gain of these circuits which arise due to the *pseudo-rotations.* The values of z should be within the *range*[4]. The condition for the direction variable d would depend on the difference between the required angle and the current angle.
The outputs produced by the algorithm in rotation mode is given in Table 1.2.

Co-ordinate System	x_{output}	y_{output}	z_{output}
Linear	x	x*z	0
Circular	$cos(z)$	$sin(z)$	0
Hyperbolic	$cosh(z)$	$sinh(z)$	0

Table 1.2: Table Containing the Outputs of Rotation Mode

The update equation for d_i in rotation mode is given by:

$$d_i = \begin{cases} -1 & \text{if } z_i < 0 \\ 1 & \text{if } z_i \geq 0 \end{cases}$$

1.3.2 Vector Operation Mode

In vector mode, we rotate a given vector at any arbitrary angle, lying within the prescribed range, such that it approaches the positive x-axis. The inputs to be given in vector mode are given in the Table 1.3.

Co-ordinate System	x_{input}	y_{input}	z_{input}
Linear	x	y	0
Circular	x	y	0
Hyperbolic	x	y	0

Table 1.3: Table Containing the Inputs for Vector Mode

On performing the algorithm, we reduce the input y to zero. This is actually the parameter that dictates the decision variable d. The computed values are given in Table 1.4.

Co-ordinate System	x_{output}	y_{output}	z_{output}
Linear	x	0	$\frac{y}{x}$
Circular	$\sqrt{x^2 + y^2}$	0	$\tan^{-}1\frac{y}{x}$
Hyperbolic	$\sqrt{x^2 - y^2}$	0	$\tanh^{-}1\frac{y}{x}$

Table 1.4: Table Containing the Outputs of Vector Mode

The update equation for d_i in vector mode is given by:

$$d_i = \begin{cases} 1 & \text{if } y_i < 0 \\ -1 & \text{if } y_i \geq 0 \end{cases}$$

1.4 CORDIC Co-ordinate Systems

The update rules of x, y and z in the *pseudo-rotation* equations can be modified such that it would result in three different results. This can be called as the co-ordinate system of operation. These three modes can be brought together by using a unified theory for CORDIC algorithms [4] by using a new variable μ. The new equations are as follows.

$$X_{i+1} = X_i - \mu \cdot d_i \cdot Y_i \cdot 2^{-i} \tag{1.10}$$

$$Y_{i+1} = Y_i + d_i X_i 2^{-i} \tag{1.11}$$

$$Z_{i+1} = Z_i - d_i E \tag{1.12}$$

μ and E are defined as in Table 1.5.

Co-ordinate System	μ	E
Linear	0	2^{-i}
Circular	1	$\tan^{-1}(2^{-i})$
Hyperbolic	-1	$\tanh^{-1}(2^{-i})$

Table 1.5: Table Containing the Parameters for Updating z

The interesting thing with hyperbolic mode is that for the algorithm to converge, we need to repeat certain iterations[4]. It has been found that repeating every $3k + 1$ iteration, *i.e.* the iterations 4, 13, 40 *etc.* would provide the value of hyperbolic functions.

1.5 Other Applications of CORDIC

The rotation mode of CORDIC can be used to convert *Cartesian* system co-ordinates to *Polar* system co-ordinates. Vector mode can be used for generating the opposite result. Moreover, if we cascade the different co-ordinate systems, we can obtain many more transcendental functions like the remaining circular and hyperbolic functions,their inverse functions, logarithmic and even exponential functions.An example:

$$\sin^{-1}(x) = \tan^{-1}\left(\frac{x}{\sqrt{1 - x^2}}\right) \tag{1.13}$$

2. Variation and Accuracy of Algorithms

2.1 Domain of Convergence

2.1.1 Circular Co-ordinate System

The reason why CORDIC algorithms work efficiently is because of the convergence of the rotating vector to the fixed angle. This is possible only because

$$|z[i]| \leq \sum_{n=i}^{\infty} \tan^{-1} 2^{-n} \tag{2.1}$$

Then, the maximum value to which z would get summed up would give the domain of convergence.This value could be obtained by taking all the decision variable d_i to be 1. This is given by:

$$\theta_{max} = \sum_{i=0}^{\infty} \tan^{-1} 2^{-i} \approx 1.7429 \tag{2.2}$$

2.1.2 Other Co-ordinate Systems

The convergence for linear mode follows the rules stated above and come out to be

$$\theta_{max} = \sum_{i=0}^{\infty} 2^{-i} = 2 - 2^{-n} \tag{2.3}$$

But hyperbolic mode does not converge that easily. The reason for it is that an equation similar to eqn(2.1) does not generally hold for hyperbolic arc tangents. In *some* cases,

$$\sum_{n=i}^{\infty} \tanh^{-1} 2^{-n} < |z[i]| \tag{2.4}$$

This can be fixed by repeating every $3k + 1$ iteration, *i.e.*, the iterations 4,13,40,81 etc. [1]In other words:

$$\sum_{i=j+1}^{\infty} \tanh^{-1} 2^{-i} < \tanh^{-1} 2^{-j} < \sum_{i=j+1}^{\infty} \tanh^{-1} 2^{-i} + \tanh^{-1}(3j+1) \quad (2.5)$$

If we add up the values of z_i, we will get:

$$\theta_{max} = 1.11817 \quad (2.6)$$

2.2 Variation of values with iterations

CORDIC algorithms converges to true value with each iteration, typical accuracy is of 1 bit per iteration. But this value is an approximation since variation is not uniform. Typically the convergence is $\tilde{1}$ bit per iteration.

For an example case CORDIC algorithms were run on `Circular coordinate system` and `rotation mode` for 20 iterations. Using the initial values given in Table 2.1. On each iteration value of x was compared with the expected value and most significant bit with error was calculated. Variation of error bit thus obtained with iterations is plotted in Fig. 2.1.

After 20 iterations X was 0.500039 while the expected value is $5 \cdot 10^{-1}$. This gives a percentage error of 0.0078%.

Example:

Table 2.1: Initial values in circular Coordinate system and rotation mode

X_0	Y_0	Z_0
1/a = 0.6073	0	60^o

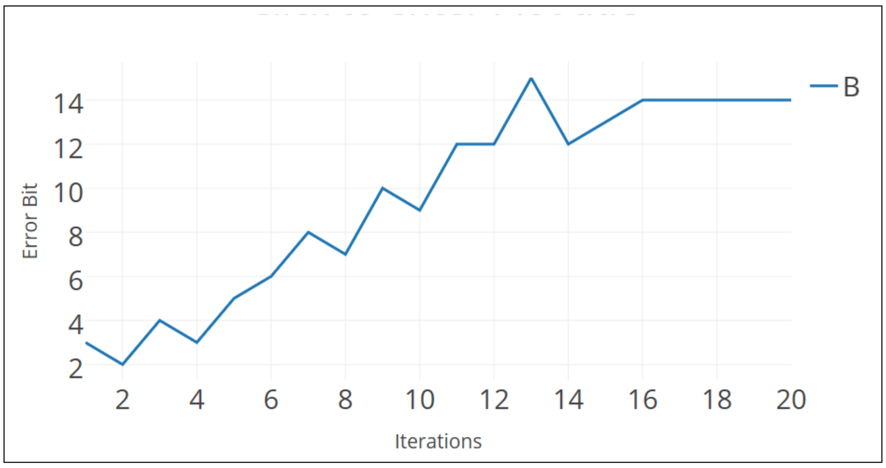

Figure 2.1: Variation of error bit in calculation of X for initial values given in Table 2.1

Iteration	Error Bit	X Value
1	3	0.607300
2	2	0.303650
3	4	0.531387
4	3	0.427007
5	5	0.483349
6	6	0.510685
7	8	0.497253
8	7	0.504031
9	10	0.500657
10	9	0.498967
11	12	0.499813
12	12	0.500236
13	15	0.500024
14	12	0.500130
15	13	0.500077
16	14	0.500051
17	14	0.500038
18	14	0.500044
19	14	0.500041
20	14	0.500039

Table 2.2: Variation of X and error bit with iteration for initial values given in Table 2.1. With each iteration most significant bit with error in difference of expected value and output of algorithm was noted. Note that the error bit was calculated after converting output to fixed point unsigned binary number.

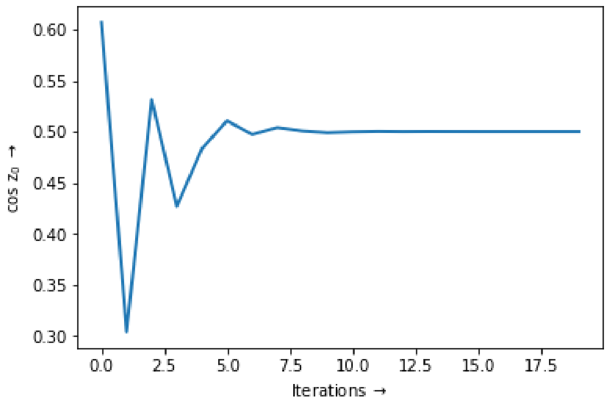

Figure 2.2: Variation of $Cos(Z_0)$ with each iteration

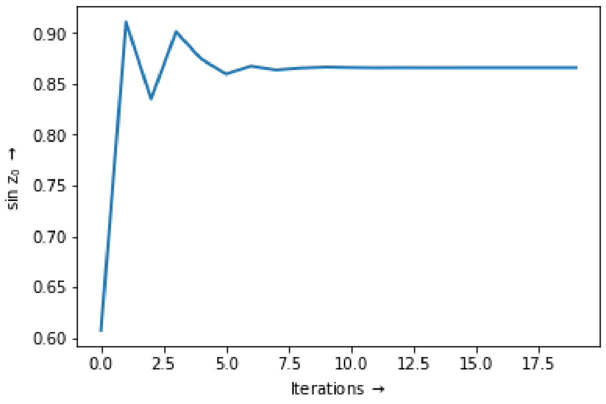

Figure 2.3: Variation of $Sin(Z_0)$ with each iteration

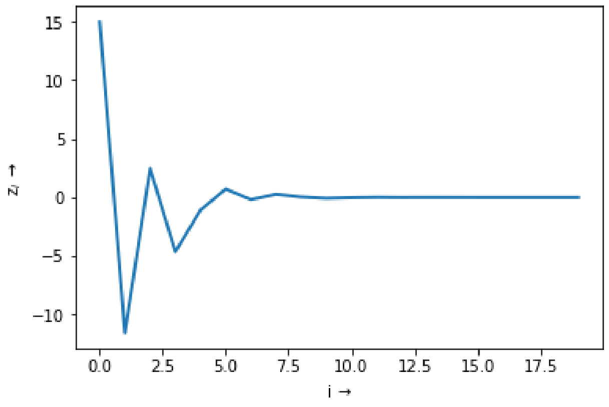

Figure 2.4: Variation of Z with each iteration

Figure 2.5: Plots representing variation of x, y and z with $X_0 = 1/\text{a}$, $Y_0 = 0$ and $Z_0 = 30^o$, when CORDIC algorithm is operated for 20 iteration in rotation mode of circular coordinate system

3. FPGA Implementation

3.1 Introduction

The FPGA implementation of CORDIC algorithm has been done with the help of Xilinx ISE software package. The process involved the following steps:

1. HDL description of the algorithm using Verilog
2. Verification and simulation of synthesized design in Xilinx ISE
3. Verification of implemented hardware

Architecture used for implementation and project structure is given in Fig 3.1 and Fig. 3.2 respectively

3.2 HDL Description

The HDL description of CORDIC algorithm has been written completely using the `Verilog 2001` language. IEEE-754 single precision floating point representation has been used in order to represent real numbers.

Short description of each of the modules (structure given in Fig. 3.2 used in our code is given below:

- **top.v** top module establishes communication between CORDIC module and LCD module. It also controls the clocks used and provides initial values for calculation.
- **cordic_top.v** cordic_top handles all the logic related to calculation of results using the algorithm.
- **floating_point_shifter** module used to perform arbitrary bit wise shift on a floating point number.
- **ROM_lookup** ROM_lookup module provides the values required during the calculation i.e. $tan^{-1}(z)$, $tanh^{-1}(z)$ and 2^{-z}. Calculating these values without using a look up table would increase the hardware cost tremendously.

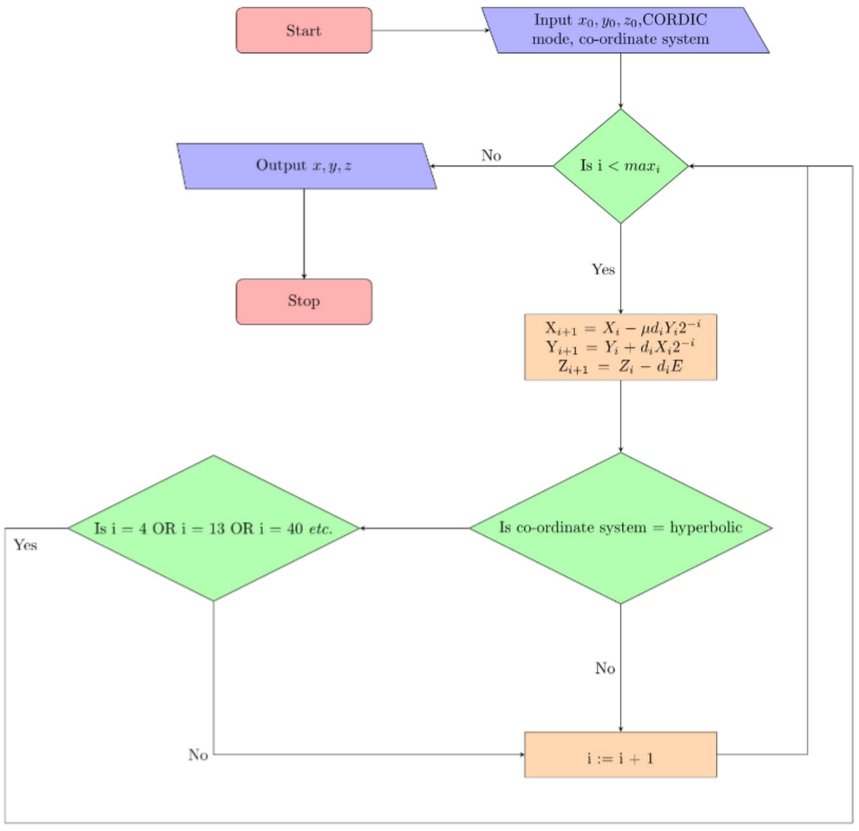
Start
Input x_0, y_0, z_0,CORDIC mode, co-ordinate system
Is i < max_i
No
Output x, y, z
Stop
Yes
$X_{i+1} = X_i - \mu d_i Y_i 2^{-i}$
$Y_{i+1} = Y_i + d_i X_i 2^{-i}$
$Z_{i+1} = Z_i - d_i E$
Is co-ordinate system = hyperbolic
Is i = 4 OR i = 13 OR i = 40 *etc.*
Yes
No
No
i := i + 1

Figure 3.1: Flowchart describing CORDIC Algorithm

- **floating_point_add_sub**[5] This is an IP for add-sub on floating points.

3.3 Simulation and verification of algorithm

Simulation of HDL description has been performed using LSim Software. Verification of the results have been performed. Details of hardware generated on Spartan 3E device is given in Table 3.1.

Logic Utilization	# used	available	utilization
Slice Flip Flops	283	9,312	3%
4-input LUT	1,906	9,312	20%
RAMB16s	1	16	5%

Table 3.1: Logic utilization on Spartan 3E xc3s500e

```
top - top
├─ LCD_instance - LCD
└─ cordic_instance - cordic_top
   ├─ ordic_top
   ├─ shifter_x
   ├─ shifter_y
   ├─ x_add_sub
   ├─ y_add_sub
   ├─ z_add_sub
   └─ rom_out
```

Figure 3.2: Project Structure

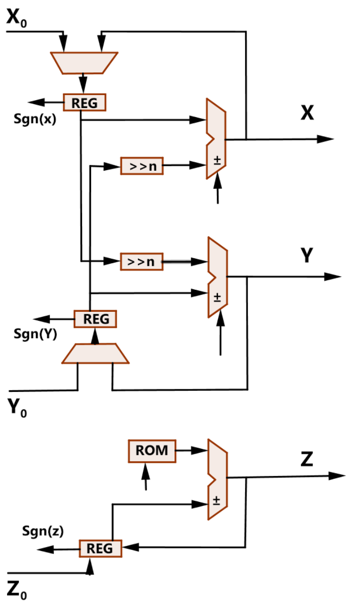

Figure 3.3: CORDIC Architecture[1]

3.3.1 Verification on FPGA

Verification on FPGA was done by displaying the data (value of x, y and z) using an LCD[2] as ASCII characters, the characters displayed were converted back to binary and then to decimal numbers. These values were then verified to be correct.

4. Our Learning Experience

This chapter deals with the experience and knowledge that we gained while working on this project.

4.1 New Knowledge

This project has been very helpful in introducing us to the field of research. It has taught us a lot about how research works, how to read research papers and much more. Our knowledge of Verilog has quadrupled in this short span of time. We have now a deeper insight on how HDL are different from programming languages, this further helped in understanding how FPGA's work, how the HDL code gets synthesized to LUT's and MUXs. Moreover, it has helped us in understanding how to provide hardware solutions by cutting down the hardware cost without trading off performance.

4.2 Challenges Faced

Our knowledge in Verilog was initially rather limited. Moreover, we knew nothing about using FPGA's. Obtaining the expected result in the LCD screen was rather challenging. We found the hyperbolic mode of CORDIC rather challenging which took considerably longer to implement since details of that is rarely available readily.

Bibliography

[1] Ray Andraka. A survey of cordic algorithms for fpga based computers. In *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, pages 191–200. ACM, 1998.

[2] M. Nguyen. Controlling the spartan-3e lcd.

[3] Jack Volder. The cordic computing technique. In *Papers presented at the the March 3-5, 1959, western joint computer conference*, pages 257–261. ACM, 1959.

[4] John S Walther. A unified algorithm for elementary functions. In *Proceedings of the May 18-20, 1971, spring joint computer conference*, pages 379–385. ACM, 1971.

[5] Xilinx Inc. LogiCORE IP Floating-Point Operator v5.0. https://www.xilinx.com/support/documentation/ip_documentation/floa ting_point_ds335.pdf, 2011. Online.