FPGA Implementation of the Trigonometric Functions Using the CORDIC Algorithm

A Project report Submitted in partial fulfillment of the requirements for the award of the Degree of

Bachelor of Technology

in

Electronics and Communication Engineering

By

NIMMAPANDU POOJITHA (203R1A0455)

MUNAGALA MYDHILI (203R1A0444)

GUDARI KALYANI (203R1A0428)

BANDLA DILEEP (203R1A0416)

Under the Guidance of

Mrs. K. SREE LAKSHMI M.Tech

Associate professor

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

RAMIREDDY SUBBARAMIREDDY ENGINEERING COLLEGE

(Approved by AICTE and Affiliated to JNTUA, Ananthapuramu, Accredited by NAAC with 'A' Grade)

NH-16, KADANUTHALA, KAVALI, SPSR Nellore (Dt.), A.P – 524142

(2023-2024)

Department of Electronics & Communication Engineering,

RAMIREDDY SUBBARAMIREDDY ENGINEERING COLLEGE

(Affiliated to JNTUA, Ananthapuramu, Approved by AICTE, Accredited by NAAC with 'A' Grade)

NH-16, KADANUTHALA, KAVALI, SPSR Nellore (Dt.), A.P – 524142

CERTIFICATE

This is to certify that the project report titled **"FPGA Implementation of the Trigonometric Functions Using the CORDIC Algorithm" ,**

being submitted by

NIMMAPANDU POOJITHA (203R1A0455)

MUNAGALA MYDHILI (203R1A0444)

GUDARI KALYANI (203R1A0428)

BANDLA DILEEP (203R1A0416)

In partial fulfillment of the requirements for the award of the degree of ***Bachelor of Technology in Electronics and Communication Engineering,*** to the Jawaharlal Nehru Technological University Anantapur, Ananthapuramu is a record of bonafied work carried out by them under my guidance and supervision.

Internal Guide	**Head of the Department**
Mrs. K. Sree Lakshmi,	**Mr. V. Suneel Reddy,**
Associate Professor,	**Associate Professor,**
Dept. of ECE,	**Dept. of ECE,**

External Viva Voce conducted on_____

Internal Examiner **External Examiner**

DECLARATION

We hereby declare that the project entitled, **"FPGA Implementation of the Trigonometric Functions Using the CORDIC Algorithm"** completed and written by us, has not been previously submitted elsewhere for the award of any degree or diploma.

Place:

Date:

NIMMAPANDU POOJITHA (203R1A0455)
MUNAGALA MYDHILI (203R1A0444)
GUDARI KALYANI (203R1A0428)
BANDLA DILEEP (203R1A0416)

ACKNOWLEDGEMENTS

We consider it our duty to express our gratitude to all those who guided, inspired, and helped us in the completion of this project work.

We acknowledge, with a profound sense of gratitude, the guidance and support of our guide, ***Mrs. K. Sree Lakshmi,*** *Associate Professor*, Department of Electronics and Communication Engineering, RSR Engineering College, Kadanuthala, Kavali. Her timely suggestions and cooperation, both professionally and personally, have greatly contributed to bringing out the project successfully.

We express our heartfelt thanks to ***Mr. V. Suneel Reddy***, Associate Professor and Head of the Department of Electronics and Communication Engineering, RSR Engineering College, Kadanuthala, Kavali, for his kind encouragement and for providing us with all required facilities for the completion of the project work.

We also express our gratitude to the Principal, ***Dr. P. V. N. Reddy***, for providing the necessary infrastructure & an ambient atmosphere to complete our project work.

We also express our gratitude to the Director, ***Dr. K. Raja Reddy***, Professor, for being a source of inspiration throughout our study in this college.

We extend our sincere appreciation to the Chairman, ***Mr. R. Pratap Kumar Reddy***, for his visionary leadership and support in all our academic endeavors.

We are indeed indebted to all our teachers who have guided us throughout our B. Tech course for the past four years and have imparted sufficient knowledge and inspiration to take us forward in our career.

Finally, we thank each and everyone who has helped us directly and indirectly in the completion of the project work.

NIMMAPANDU POOJITHA (203R1A0455)
MUNAGALA MYDHILI (203R1A0444)
GUDARI KALYANI (203R1A0428)
BANDLA DILEEP (203R1A0416)

ABSTRACT

The project presents the design principle and FPGA implementation of various trigonometric functions such as Sine, Cosine, Exponential, Inverse Exponential, Arc Tangent, Logarithm, and Polar to Rectangular conversion using the standard coordinate rotation digital computer (CORDIC) algorithm. Traditions implementation of these functions on a FPGA consumes a lot of area and the results of these functions are floating point which is difficult to design. Hence CORDIC algorithm with IEEE 32-bit floating-point representation is used in this project for implementation. CORDIC is an iterative algorithm which can perform the complex functions using the shift and add approach. The serial and pipelined CORDIC architectures configured on a Cyclone IV E Device are compared in terms of Area, Delay, and Power dissipation. Serial CORDIC architecture design has low area whereas; pipeline CORDIC architecture has low latency. It finds the application in graphic processors, digital synchronizer, Real time image processing, and scientific calculators and so on.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

The advances in the very large scale integration (VLSI)technology and the advent of ZAADS electronic design automation (EDA)tools have been directing the current research in the are as of digital signal processing (DSP), communications, etc. in terms of the style of fast-speed VLSI architectures for real-time algorithms and systems which have applications in the above mentioned areas. The development rate of VLSI technology was predicted by Gordon Moore and since 1965 newer technologies have been developed by the industry fitting his predicted curve, which was introduced as the so called Moore's law. These advances have provided momentum to the designers for transforming algorithm into architecture. Many DSP algorithms use elementary functions like logarithmic, trigonometric, exponential, division and multiplication. Two of the ways of implementing these functions are by using table lookup method and through polynomial expansions. The above mentioned methods require large number of multiplications/divisions and additions/subtractions. CO-ordinate Rotation Digital Computer (CORDIC), a special purpose computer to compute many non-linear and transcendental functions, was proposed by Volder in 1959.

The functions that can be computed using a CORDIC computer include trigonometric, logarithmic, exponential, hyperbolic, multiplication, division, square root, etc. Though it initially served the purpose of navigation systems, it later became a popular tool to implement several digital systems especially in the areas of digital signal processing, communications, computer graphics, etc. The simplicity of CORDIC is that it can compute any of the above mentioned functions using shifts and additions which are of the form $x \pm 2^{-i} * y$. The operating mode and the coordinate system chosen are two key factors to compute the desired functions in the CORDIC. Many signal processing and communication systems operate CORDIC in circular coordinate system and in either of rotation or vectoring modes.

1.2 History

CORDIC was first introduced by Jack E Volder for his navigation applications in 1959. Then introduced CORDIC was applicable for evaluating the trigonometric identities that involved in plane coordinate rotation and transformation between polar and rectangular coordinates.

Later, in 1971, J.S. Walther unified the CORDIC that was earlier introduced by Volder for evaluating multiple elementary functions based on round, straight line, and hyperbolic organize systems based on which function is to be computed. Over the last 54 years, the architecture of CORDIC has seen multiple changes and multiple variants of the initially proposed algorithm have emerged Architectures such as low-latency pipelined CORDIC, mixed-scaling-rotation (MSR)CORDIC, scaling- free CORDIC have seen much usage in modern digital systems.

1.3 CORDIC Applications

Applications of CORDIC can mainly be seen in the following areas:

1. Matrix decomposition
2. Signal and image processing
3. Communications
4. Computer graphics
5. Robotics

In matrix decomposition, CORDIC Is used to compute QR decomposition (QRD), singular value decomposition (SVD), and Eigen value estimation. In signal and image processing, CORDIC Is used in fixed/adaptive filtering, computing discrete transforms of Fourier basis, image enhancement operation, etc. In communications, CORDIC is mostly used in direct digital synthesis, digital and analog modulation, envelope detection, etc... In computer graphics and robotics, CORDIC is used in solving direct and inverse kinematics for robot manipulators, 3D vector rotation in computer graphics, etc.

1.4 Advantages

1. Components need and cost of CORDIC processer is less as only move signs up adders and look-up desk (ROM) are required

2. Variety of gateways needed in hardware execution, such as on an FPGA, is lowest as hardware complexness is decreased in comparison to other processer chips such as DSP multipliers

3. It is relatively easy in design

4. No multiplication and only inclusion, subtraction and bit-shifting function guarantees easy VLSI execution.

1.5 Disadvantages

1. Large amount of versions needed for precise outcomes and thus the rate is low and time delay is high.

2. Energy intake is great in some structure types.

3. Whenever a hardware multiplier is available, e.g. in a DSP micro-processor, desk lookup techniques and good old-fashioned power sequence techniques are usually faster than this CORDIC criteria.

1.6 Motivation and Problem Statement

Thus, we can realize that CORDIC is one of the most determining arithmetic techniques that has found far-reaching applications in digital systems. Area efficient and power efficient systems are always the preferred choice of any designer. Thus, as CORDIC based systems are both area-efficient and power-efficient, one can design architectures for emerging digital systems based on CORDIC.

The problems that have been addressed here are the analysis of CORDIC unit for small data-width (up to16 bits and sometimes 20bits) and to develop a scale free CORDIC unit with the mapping mechanism that maps the angle to entire 360^0, efficient FPGA design of an 8-point DFT core, FPGA design of a direct digital synthesizer (DDS), and FPGA design of a folded pipelined radix-2^2 complex FFT core. The problem also addresses the application specific integrated circuit (ASIC) design of DDS and FFT core.

CHAPTER 2
LITERATURE OF SURVEY

2.1 CORDIC Background

Coordinate Rotation Digital Computer (CORDIC) unit has become an essential and inevitable hardware block in modern engineering and scientific applications. It serves many applications such as solving trigonometric and transcendental equations, in digital signal processing (DSP) for Fourier basis based orthogonal transforms, in computer technology for 3D graphics, in digital communication systems for modulation and demodulation of the signals, etc. The conventional CORDIC was first implemented by Volder, in 1959.CORDIC perform by spinning the organize program through continuous prefixed perspectives until the position is decreased to zero. Figure 2.1 shows the coordinate rotation of a vector in 2D circular coordinate system.

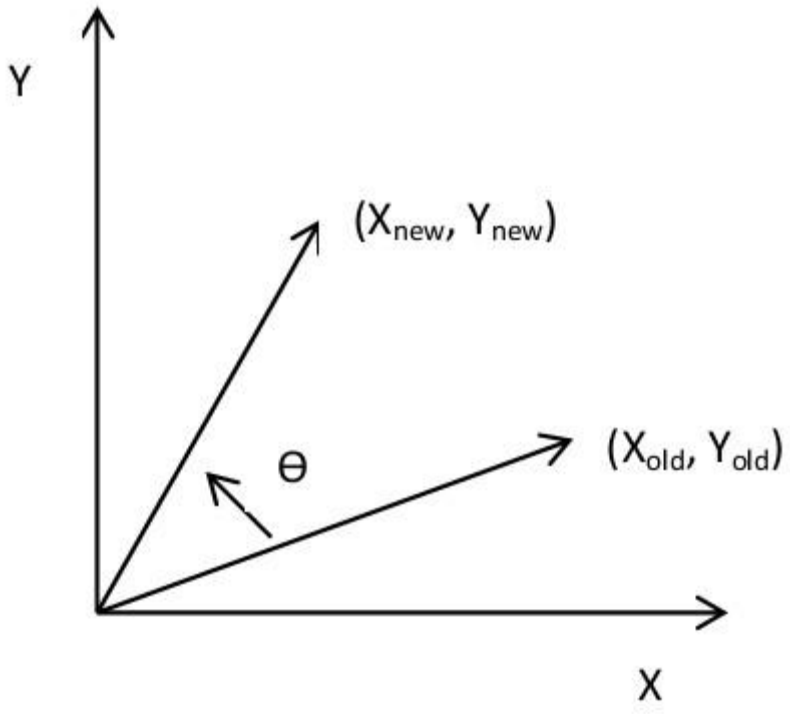


Figure 2.1- Rotation of a vector by an angle in 2D circular coordinate system

Frequency synthesizers, sometimes also called oscillators, are an essential unit of many communication systems. With the maturity of digital systems, communication systems are employing digital subsystems in their units, thus making the usage of digital system more ubiquitous. Direct digital synthesizers (DDS) area class of frequency synthesizers in digital domain, which generate waveforms of

desired frequencies. Sometimes also called numerically controlled oscillators (NCO), these generate waveforms like sine, cosine, triangular, square or rectangular, saw tooth, etc. As mentioned earlier, these have wide applications in satellite communication systems, RF signal processing etc. Many communication systems require quadrature inputs, for example both sine and cosine, for their systems thus bringing in the need of design of DDS which can generate quadrature outputs.

2.2 Overview of CORDIC

CORDIC acronym of CO-ordinate Rotation Digital Computer, which is also known as Volder's algorithm and digit-by-digit method, is a simple and efficient method to calculate many functions including trigonometric, hyperbolic, logarithmic, etc. Its main advantage lies in its less hardware over head and reduced latency. It performs complex multiplications using simple shifts and additions. Jack E. Volder described the modern CORDIC algorithm in 1959 for his navigation applications, which consisted of vector rotations. It can be used in building low complexity finite state CPUs. Further in 1971, J.S. Walther generalized the algorithm by allowing it to calculate functions such as hyperbolic, exponential, logarithms, multiplications, divisions and square-roots. The key usage of CORDIC lies in selecting it's one of the operating modes, rotation or vectoring, and one of the coordinate systems, which being either circular, or linear or hyperbolic.

Usage of CORDIC in circular coordinate system results in direct outputs like and through which we can calculate other trigonometric identities. Implementing CORDIC in hyperbolic coordinate system results in functions such as and from which we can get other hyperbolic, logarithmic functions. CORDIC implementation in linear coordinate system results in calculating functions like multiplication, division. Several variants of CORDIC, both scaled and un-scaled, have been designed and among them, scale-free CORDIC architectures have gained popularity in terms of scale-factor compensation. CO-ordinate Rotation Digital Computer (CORDIC) is an entire-transfer computer, containing a unique sequential mathematics unit made up of three shift signs up, three adder-sub tractors, and unique inter connections. It is mostly used to solve the trigonometric relationships that are involved in plane organize spinning and conversion from rectangle-shaped to complete harmonizes and vice versa.

Using a pre-determined set of depending improvements or subtractions, the CORDIC arithmetic can be controlled for solving either set of the equations given below.

$$X_{NEW}=K(X_{old}\cos\theta - Y_{old}\sin\theta) \tag{2.1}$$

$$Y_{NEW}=KX_{old}\sin\theta - Y_{old}\cos\theta) \tag{2.2}$$

$$R = K\sqrt{X^2_{old} + Y^2_{old}} \tag{2.3}$$

In above set of equations, K is an invariable constant. The above set of equations can be used in calculating the coordinates of the vector (X_{new}, Y_{new}) from the vector (X_{old}, Y_{old}) which is rotated by an angle of θ in a 2D circular coordinate system. This is shown in figure 2.1. The first set of equations given in equation (2.3) can be written in matrix form as shown below:

$$\begin{bmatrix} X_{NEW} \\ Y_{NEW} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} Xold \\ Yold \end{bmatrix} \tag{2.4}$$

Re Writing the Equation (2.4) Using Notations We get,

$$V_{new} = R.\, V_{old}$$

$$V_{new} = \begin{bmatrix} X_{new,} \\ Y_{new} \end{bmatrix}, \quad R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \quad V_{old} = \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} \tag{2.5}$$

Modifying the matrix in equation (2.5) as shown below results in,

$$R = \cos\theta \begin{pmatrix} 1 & \tan\theta \\ -\tan\theta & 1 \end{pmatrix} \quad (2.6)$$

Now Rewriting Equation (2.6) We get

$$R = K R_p \quad (2.7)$$

Where, K = $\cos\theta$ and R_p is called pseudo-rotation matrix.

The rotation through the angle can be performed as a sequence of incremental rotations, in this context fixed rotations, in such a way that after N rotations, the sum of the incremental rotation angles nearly equals the required rotation. The residue that remains after N rotations can be ignored as it does not affect the value of the angle that has to be rotated.

The total scale factor, K, can be given as the product of the incremental scale factors obtained at each incremental rotation, as shown below

$$K = \prod_{i=0}^{N-1} Ki \approx 0.60725 \quad (2.8)$$

2.3 LOOKUP TABLE

2.3.1 Elementary Functions

A function is an elementary function if it can be constructed from a finite combination of constant functions, field operations (addition, subtraction, multiplication, or division), algebraic (*x n*), exponential (*e n*), logarithmic (log*nx*) functions and their inverses. Some of the most common elementary functions are the trigonometric (sin*x*, cos*x*, and tan*x*) and the hyperbolic functions (sinh*x*, cosh*x*, and tanh*x*) and their inverses.

Many current and future applications depend upon the accurate calculation of elementary functions. Trigonometric functions, such as sine and cosine, are especially important. Whether it is the ENIAC calculating shell trajectory tables, the HP 35 hand held calculator, Singular Value Decomposition (SVD) algorithms, computer graphics applications, Digital Signal Processing (DSP)systems, digital communications,

adaptive filters, or robotic movement, trigonometric functions are repeatedly required during the course of their operation.

In order to correctly calculate their results, each of these applications requires accurate trigonometric values for every possible angle. Whether the functions are implemented in software or hardware, the calculation of trigonometric values is a complex and time consuming process. Often, the computation time for obtaining the trigonometric values can dominate the execution time of the algorithm.

System performance can also be degraded if the calculations following the trigonometric functions the value it returns. If this happens, the system will not be able to proceed until the trigonometric calculation is complete. Table 2.1 shows the latency and error for some of the elementary functions in Intel's IA-64 processor. The cube root(cbrt), exponential(exp), and natural logarithm(ln) all have large latencies, but the latencies of the trigonometric functions (sin, cos, tan, and arc tan) have the highest latencies. This underscores the fact that reducing the latency of trigonometric calculations will significantly improve the performance of any system that calculates these functions.

Table 2.1- Latency and Error of LA-64 Elementary Functions

FUNCTION NAME	LATENCY (cycles)	ERROR (*ulps*)
cbrt	60	0.51
exp	60	0.51
ln	52	0.53
sin	70	0.51
cos	70	0.51
tan	72	0.51
Arc tan	66	0.51

2.3.2 Table Look-up CORDIC

The new CORDIC algorithm utilizes look-up tables and standard microprocessor arithmetic functional units to perform the calculations. The lookup tables implement either the traditional CORDIC or the new Parallel Arc Tangent

Radix (ATR). Each entry in the traditional CORDIC ATR combines multiple CORDIC iterations into a single effective rotation. Combining these rotations divides the angle domain into separate partitions between the critical angles. All of the angles in an individual partition are rotated in the same direction in all of the iterations represented in the table.

The Parallel ATR improves up on the traditional CORDIC ATR by rotating the vector by the exact values of the angle. This provides several significant benefits for a designer implementing the Table Look-up CORDIC algorithm. First, the complexity of the ROM decoder is greatly simplified. Second, all ROM look-up tables can be accessed simultaneously without intermediate computations to determine residual angles. And finally, the number of computations required to obtain the final answer is reduced.

The Table Look-up CORDIC (TLC) algorithm is shown to be correct through the development of a mathematical proof utilizing the polar form of the CORDIC iteration equations. The TLC algorithm and other versions of the CORDIC algorithm are implemented in MATLAB and simulated. The results of these simulations are compared to verify the new algorithm's operation. The same CORDIC algorithms are then modeled in Verilog.

The Verilog models are then synthesized into gates, routed, and statically timed. The auto place and route of these circuits allowed area estimates to be obtained for the different algorithms. The auto place and route allows silicon areas to be compared while the static timing analysis allows the worst-case path.

2.3.3 Algorithm classes

There are a large number of algorithms that can be used to calculate the various trigonometric functions. These algorithms can be classified by the manner in which they perform their computations. Four classes of algorithms will be discussed in the following sections. These classes are the polynomial approximation algorithms, rational approximation algorithms, linear convergence algorithms, and quadratic convergence algorithms.

Although this dissertation only examines variations of the CORDIC algorithm, a linear convergence algorithm, it is important to understand the implementation and computation time of the other classes of algorithms so that valid performance and architectural comparisons can be made between the algorithms.

2.3.4 Polynomial Approximation

A polynomial approximation, P(x), is a degree-n polynomial of the form shown in Equation (2.9). This polynomial is used to approximate a function over the interval of interest. The degree of the polynomial, n, depends upon the amount of error that can be allowed in the calculation. Polynomials of higher degrees generate less error, but they obtain this precision at the expense of computation time.

$$f(x) = P(x) = p_n x^n + p_{n-1} x^{n-1} + \cdots + p_1 x + p_0 \quad (2.9)$$

The coefficients of each term of the polynomial are selected to minimize the average error between the polynomial and the actual function. Normally, a standard least squares or Chebyshev approximation is used to calculate the coefficients of the polynomial. Sometime the function being approximated has regions where there are sharp changes in its slope. These regions are more difficult to accurately model than regions with small changes or no change in the slope. In order to model these intervals without significantly increasing the degree of the polynomial, weighting functions are used to ensure a more precise match. Two of the more common weighting functions in use are Legendre and Jacobi functions.

The polynomial approximation class of algorithms is one of the easiest to implement with in a digital system. Once the degree and coefficients of the polynomial have been selected, the values of the coefficients are stored in a ROM. These coefficients are used to calculate the given function anytime it is needed.

It is possible to reduce the required order of the polynomial by breaking the function into several intervals. Although this will increase the number of ROM tables needed to hold the coefficients, It will reduce the total number of calculations that must be performed. The increased speed of the implementation is obtained at the expense of increased silicon area.

Even with intelligently subdivided intervals, the approximation will still be a degree polynomial, where mz≤n . It is possible to rearrange the polynomial to minimize the number of multiplications that must be performed. Applying Horner's scheme to the polynomial, P(x), it can be re-written as shown in Equation (2.10).

Using Horner's scheme, a degree-n polynomial requires n multiplications and additions. The total computation time required by this calculation is n. t Mult +n . t add where n is the degree of the polynomial, t Mult is the time to perform a multiplication, and t add is the time to perform an addition.

$$P(x) \approx ((((p_n x + p_{n\_1})x + p_{n\_2}\text{K})x + p_1)x + p_0 \tag{2.10}$$

In addition to Horner's scheme, there are several other computation minimization techniques that can be applied to polynomial approximations. Two of these techniques are the E-Method and Estrin's Method. Some techniques are only useful for polynomials of specific degrees or polynomials that only contain even or odd powers.

Koren and Zenati researched the degrees and coefficients required by rational approximations in order to achieve errors less than 1ulp, unit in the last position, in 32-bit binary numbers. Even though there search has not been published on the requirements for polynomial approximations, it is possible to draw some conclusions by examining papers that have been published. AMD's K5 microprocessor implements the elementary functions through the use of polynomial approximations stored in ROM tables. Unfortunately, no mention is made of the degree of these polynomials or their coefficients.

Intel's IA-64 architecture also uses polynomial approximations for calculating elementary functions. The number of polynomials and the degrees of the polynomials are given within the descriptions of each of the functions presented. The calculation of sine or cosine requires the calculation of two polynomials, one with a degree of eight and the other with a degree of nine. These two numbers are multiplied by separate coefficients and then combined using a formula that requires several additions.

2.3.5 Linear Convergence

A linear convergence algorithm is a family of iteration equations where the next value for each variable in the equation is based upon the current value of the variables. A single iteration through the family of equations refines the accuracy of the variables as the equations linearly converge upon the correct answer. An example of a family of iterations can be seen in Equations (2.10), (2.11) and (2.12). At least two independent equations are required.

$$X_{i+1} = f(x_i, y_i, z_i) \tag{2.11}$$

$$Y_{i+1} = g(x_i, y_i, z_i) \tag{2.12}$$

$$Z_{i+1} = h(x_i, y_i, z_i) \tag{2.13}$$

The CORDIC algorithm is an example of a linear convergence algorithm. In order to obtain an accurate answer for an n-bit binary number, n iterations of the equations must be performed. Even though this might not sound very time consuming, it can have a significant performance impact on an algorithm that requires the trigonometric results. Consider any program that requires the use of double precision floating point numbers such as a drafting tool for high precision parts, a simulation of high-energy physics, or even a high definition computer animation for a feature film. Each of these programs uses one or more trigonometric functions to model a chamfer, particle trajectory, or lighting effect.

The format of the double precision floating-point number that the program uses is defined by IEEE specification 754. This specification dictates that fifty-two bits will be used to represent the mantissa of the double precision floating-point number. These are the fifty-two bits following the first binary 1 in the number. This means that there are actually fifty-three bits in the mantissa of a double precision floating-point number, fifty-two bits that are stored and the one hidden bit that is understood to be there.

Because there are fifty-three bits in the mantissa, the CORDIC algorithm requires fifty-three iterations to obtain an accurate answer. In reality, several more iterations are required to ensure that the final answer is bit correct after rounding. From this example, it is obvious that if many trigonometric functions are required, a large number of iterations and time will be required to obtain the correct answer.

2.3.6 Q*uadratic Convergence*

Just like a linear convergence algorithm, a quadratic convergence algorithm is a family of iteration equations where the next value for each variable in the equation is based upon the current value of the variables. An example of a quadratic family of iterations can be seen in Equations (2.14), (2.15) and (2.16).

$$X_{i+1}=j(x_i,y_i,z_i) \tag{2.14}$$

$$Y_{i+1}=k(x_i,y_i,z_i) \tag{2.15}$$

$$Z_{i+1}=l(x_i,y_i,z_i) \tag{2.16}$$

The difference between a linear convergence algorithm and a quadratic convergence algorithm is the speed with which they converge upon the answer. As the name suggests, this category of algorithms converges upon the correct answer quadratically. The time to compute the correct answer for this category of algorithm is a logarithmic function of the number of bits of precision required by the digital system. Even though quadratic convergence equations only require log2n iterations to converge upon the correct answer, this can still represent a significant number of iterations if n is large.

Unfortunately, many quadratic convergence equations are made up of complex operations that require significant amounts of computation time to calculate. In 1976, Richard Brent published a paper describing quadratic convergence algorithms for many different elementary functions.

Brent's algorithm requires two shift operations, nine additions, five multiplications, six divisions, and three square roots to be performed during iteration of this arc tangent algorithm. In addition to these operations, another shift, two additions, one division, and one square root have to be performed during the initialization of the algorithm. The calculation of the final answer also requires two more additions, one multiplication, one division, and one logarithm. This makes Brent's quadratic convergence algorithm for arctangent a very costly implementation in terms of system resources and time. Even using a ROM table to provide an initial approximation to the answer in order to reduce the number of iterations required for convergence, the algorithm still requires significant computation time to converge upon the final answer.

CHAPTER 3
CLASSIC CORDIC ALGORITHM

3.1 Introduction

The CO-ordinate Rotations Digital Computer (CORDIC) algorithm originally developed to replace the limited accuracy analog driven navigation system of the B-58 bomber. This replacement was necessary because the analog methods could not provide accurate results for flights near the North Pole and were too slow in providing solutions for star fixing and radar ground sightings. The trigonometric algorithms being used by the analog system were too slow
to meet the B-58's real-time requirements.

The CORDIC algorithm was developed to provide a purely digital solution to these navigation problems. The CORDIC algorithm is an iterative family of equations that is used to calculate vectors or angles, depending on the mode in which they are used. The CORDIC algorithm is classified as a linear convergence algorithm, requiring n-iterations for n-bits of accuracy.

Obtaining the correct value of a trigonometric function is always important. When dealing with navigation, it is imperative to obtain the correct answer. Because of the distances that can be involved, even an error of only a single ulp of a trigonometric function can cause catastrophic errors in positioning. The best way to prevent an error in calculating a trigonometric function is to have a ROM table with entries for all possible angles. Each entry in the table is bit correct to less than 0.5 ulp. If a small number of angles are required, the trigonometric function can be implemented as a ROM table. Due to the number of angles required by navigation systems, a ROM table for all of the angles is not practical using today's technology.

In 1959, Jack Volder published the definitive paper on the CO-ordinate Rotation Digital Computer (CORDIC) algorithm. The CORDIC algorithm allows for the calculation of the sine and cosine functions using its rotation mode. These trigonometric functions, as well as others, can be precisely calculated to any bit length that is required as long as the equations are iterated through enough times and the adder is wide enough to provide a guard band for correct rounding.

Volder's original paper on the CORDIC algorithm explains its operation and high lights several of the major design decisions that were required to make the

algorithm possible. Volder's explanation of the algorithms original development or a detailed mathematical derivation of the CORDIC algorithm, this chapter attempts to show its development as a series of logical design tradeoffs.

3.2 The Unit Circle

A vector is a line segment that represents a magnitude and a direction. If the magnitude of the vector is equal to a unit length, it is known as a unit vector. The unit vector is used in many areas of science, but its most common use is to define coordinate systems. With in the field of mathematics, one of its uses is defining the unit circle. If the tail of the unit vector is located at the origin of the x-y plane and the unit vector is rotated through every angle from -pi to +pi, the path of the head of the unit vector inscribes the unit circle.

The unit circle is used to define the trigonometric or circular functions over all real numbers. A unit vector from the origin (0,0) to the point (1,0) is defined to have a rotation angle of zero. All positive angles are found by rotating the unit vector counter clockwise, while all negative angles are found by rotating the unit vector clockwise. A full revolution in either direction requires a rotation of 2pi. paper on the birth of CORDIC emphasizes the importance of the selection of the appropriate Arc Tangent Radix that makes it possible. Even though neither of Volder's papers provides the full

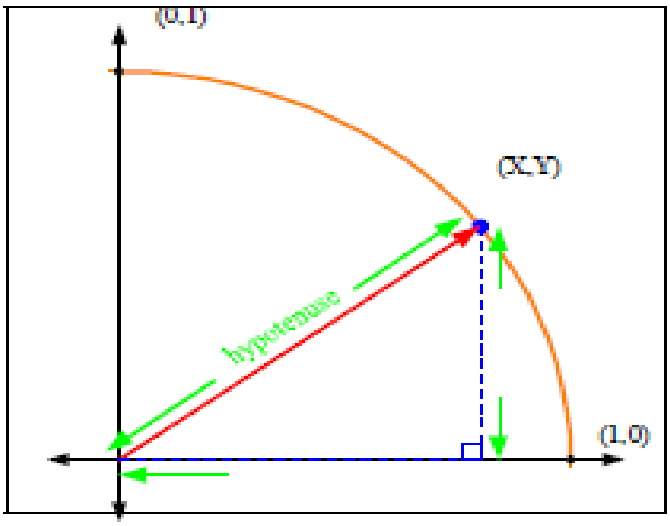


Figure 3.1- The Unit Vector

Examining a generic rotation can show the utility of using the unit vector and unit circle for calculating trigonometric functions. A unit vector with a rotation of θ radians. The head of the unit vector intersects the unit circle at point (x, y). Using the unit vector as the hypotenuse, a right triangle can be constructed inside the unit circle. A line parallel to the y–axis from the point (x, y) to the x–axis creates one side of the

right triangle. This side of the right triangle is called the opposite side because it is opposite the rotation angle θ.

A line on top of the *x*–axis from the origin (0,0) to the location where the opposite side intersects the *x*-axis creates the other side of the right triangle. This side is known as the adjacent side because it is adjacent to the rotation angle, θ. Using, the well-known formulas for cosine and sine can be developed. The cosine of angle θ is defined as the ratio of the adjacent side to hypotenuse while the sine of angle θ is defined as the ration of the opposite side to the hypotenuse. Because the hypotenuse of this right triangle is the unit vector, the length of the hypotenuse is one. Using this identity in the ratios simplifies the definitions of cosine and sine as shown in Equations (3.1) and (3.2).

$$\text{Cos}\,(\theta) = \frac{adjascent}{hypotenuse} = \frac{adjascent}{1} = \text{adjascent} \tag{3.1}$$

$$\sin(\theta) = \frac{opposite}{hypotenuse} = \frac{opposite}{1} = opposite \tag{3.2}$$

Equations (3.1) and (3.2) show that when using the unit circle, the cosine of an arbitrary angle θ is the length of the adjacent side while the sine of the arbitrary angle θ is the length of the opposite side. If the lengths of the sides of the right triangle are known , then values of the sine and cosine of angle θ are also known. Because the unit vector intersects the unit circle at point (*x*, *y*), it can be shown that the length of the adjacent side of the right angle is *x* and that the length of the opposite side of the right triangle is *y*. Substituting these values into Equations (3.1) and (3.2) produces the identities of cos (θ)= *x* and sin(θ)= *y* for the unit circle.

3.3 Calculation by Rotations

Using the unit vector and unit circle as a model, the equation for a generic rotation can be developed. Figure3.1 shows a random unit vector that has a n initial rotation of α. The tail of the unit vector is located at the origin, while the head is located at point (*Xi*, *Yi*). Using a unit circle and the well known identities for cosine and sine derived in the previous section, the position of the head of the unit vector can be expressed in terms of cos(α) as shown in Equation (3.3).

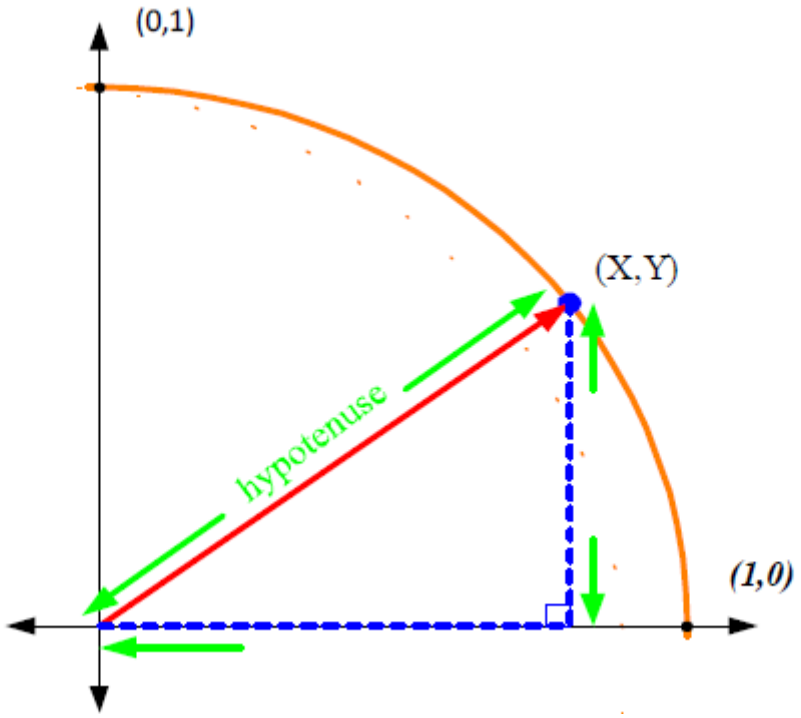


Figure 3.2- Generic Unit Vector Rotatation

Even though Figure (3.1) shows the unit vector at angle θ in the first quadrant of the Cartesian coordinate system, it can be located in any quadrant of the unit circle.

$$\text{Cos}\,(\alpha) = \frac{X}{1} = X_I \tag{3.3}$$

Rotating the unit vector by angle θ will move the head of the vector to a new location on the unit circle. If the unit vector is rotated in a positive direction, the location of the head of the vector will be at the summation of angles α and θ. If the unit vector is rotated in a negative direction, the location of the head of the unit vector will be at the difference of angles α and θ. In order to develop a generalized equation, the possibility of rotating in both directions must be taken into account, as shown in Equations (3.4) and (3.5).

$$\sin(\alpha \pm \theta) = \frac{y_{i+1}}{1} = y_{i+1} \tag{3.4}$$

$$\cos(\alpha \pm \theta) = \frac{x_{i+1}}{1} = x_{i+1} \tag{3.5}$$

Using the additive angle formulas for cosine and sine, Equations (3.4) and (3.5) can be rewritten as Equations (3.6) and (3.7) respectively. These forms also allow for the possibility of rotating the unit vector in either direction.

$$x_{i+1} = \cos(\alpha \pm \theta) = \cos(\alpha)\cos(\theta) \pm \sin(\alpha)\sin(\theta) \quad (3.6)$$

$$y_{i+1} = \sin(\alpha \pm \theta) = \sin(\alpha)\cos(\theta) \pm \cos(\alpha)\sin(\theta) \quad (3.7)$$

Utilizing the identities for xi and Yi found in Equations (3.3) and (3.4), Equations (3.6) and (3.7) can be simplified as shown in Equations (3.8) and (3.9).

$$x_{i+1} = \cos(\theta)x_i \pm \sin(\theta)y \quad (3.8)$$

$$y_{i+1} = \cos(\theta)y_i \pm \sin(\theta)x_i \quad (3.9)$$

This means that it does not matter what path around the unit circle a unit vector takes, as long as the head of the unit vector ends up at the correct location. Taking this to the next logical step, an angle can be calculated by performing a series of rotations that place the head of the unit vector at its final location. Figure (3.3) provides a graphical example of performing an effective rotation of angle θ by rotating the unit vector by angles θ 1, θ 2, and θ 3. Even though the three angles shown in this example are positive, the angles can be positive or negative, as long as their summation is equivalent to the effective rotation angle of θ.

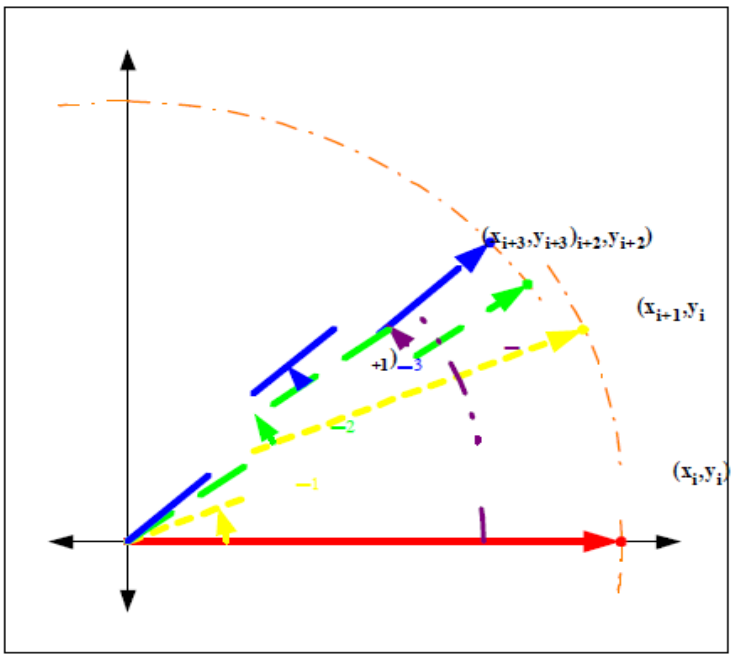


Figure 3.3 -Multiple Unit Vector Rotations.

Using this property of unit circle rotations, the calculation of the trigonometric functions can be accomplished by performing a series of rotations of the unit vector.

3.4 Iteration Equations

The CORDIC algorithm uses the system of equations shown in Equations (3.10), (3.11) and (3.12) to iteratively calculate the vector or angle in question. As the equations demonstrate, the multiplication is by a power of two and can be accomplished through a simple shift operation rather than a real multiplication. The only arithmetic operation required to calculate each new value of these equations is a simple addition or subtraction

$$X_{i+1} = X \; -_i \sigma_i 2^{-i} Y_i \tag{3.10}$$

$$Y_{i+1} = Y_i + \sigma_i 2^{-I} x_i \tag{3.11}$$

$$Z_{i+1} = Z_i - \sigma_i \tan^{-1}\left(2^{-i}\right) \tag{3.12}$$

The additions cannot be performed until the value of σ *i* has been determined for each residual angle. Table4.1 shows how σ *i* is selected during each iteration of the equations. If the angle is positive, the unit vector is rotated in a negative direction, the X variable is reduced by a fraction of the y variable, and Y variable is incremented by a fraction of the X variable. If the angle Is negative, the opposite operation is performed for each variable. Because the sign of the next residual angle cannot be determined until the current operation has been performed, the CORDIC iteration equations are inherently serial in nature.

Table 3.1 Classic CORDIC Sign Bit Selection

ANGLE	SIGN
$Z_i \geq 0$	$\sigma_i = 1$
$Z_i < 0$	$\sigma_i = -1$

3.5 Previous Work

Many variations of the CORDIC algorithm have been developed to improve the algorithm's performance. Variations that allow the sign of the angle, $\sigma\ i$, to take on the value of zero at the cost of a variable scale factor K, or correct the scale factor in parallel with the selection have been developed. The CORDIC iteration equations have been implemented after applying a House holder transform in order to improve performance. Multiplier recoding techniques have been applied to the iteration equations to improve performance and reduce latency.

Due to the number and types of variations of CORDIC algorithms available, the b set of Hardware reduction has been achieved by carefully pairing rotation iterations to reduce the number of shifters required. Control theory has been applied to the equations to eliminate the over damped response of the Classic CORDIC algorithms, the algorithms that will be examined, implemented, evaluated and compared in this dissertation needs to be selected carefully.

Because normalization and the method by which it is implemented can have a large impact on the performance of a CORDIC algorithm, only algorithms with a constant scale factor K, will be considered. In addition, CORDIC algorithms that maintain a constant scale factor through the use of additional correction rotations will not be considered in order to keep comparisons equivalent. The CORDIC algorithms that were selected and will be discussed in the following sections are the Unified, Step Branching, Double Step Branching, and Hybrid CORDIC algorithms.

3.6 Unified CORDIC

Not only can the Classic CORDIC algorithm calculate trigonometric functions, but there are also variations of the algorithm that can compute hyperbolic functions, exponentials, logarithms, and multiplication and division. The Unified CORDIC algorithm, developed by J.S. Walther in 1971, merges all of these functions into a single algorithm. This algorithm consists of a set of iteration equations that can calculate trigonometric, hyperbolic functions, exponentials, logarithms, multiplications and divisions using the same hardware and simply setting a single bit to choose the mode of operation.

3.6.1 Iteration Equations

The Unified CORDIC algorithm iteration equations are shown in Equations (3.13), (3.14), (3.15). There are minimal differences between the Classic CORDIC and

the Unified CORDIC iteration equations. The first difference is the insertion of the α parameter in to the $Xi+1$ equation. The parameter determines whether the hardware will perform trigonometric, hyperbolic, or linear functions. The allowable values of σ and their corresponding operational modes are shown in Table 3.2.

Equations (3.13), (3.14) and (3.15) are shown above.

$$X_{i+1} = X_i - \propto \sigma_i 2^{-i} Y_i \quad (3.13)$$

$$Y_{i+1} = Y_i + \sigma_i 2^{-i} X_i \quad (3.14)$$

$$Z_{i+1} = Z_i - \sigma_i e(i) \quad (3.15)$$

Vector by arc $\tan 2^{-i}$, the generalized function *e(I)* is used. The function *e(i)* is used because the rotation function is different for each of the algorithms modes of operation. A list of the operation a l modes and its rotation function, *e(i)*, is shown in Table 3.3. The selection of the sign bit, σ *i*, remains the same as the Classic CORDIC algorithm and can be found in Table (3.1).

Table 3.2- Unified CORDIC Operational Modes

∝	**ROTATIONTYPE**
1	Circular Rotations (sin, cos, etc.)
0	Linear Rotations (multiplication, division)
-1	Hyperbolic Rotations (sinh, cosh, etc.)

Table 3.3- Unified CORDIC Rotation Functions

ROTATIONTYPE	***e(i)***
Circular Rotations	$\tan^{-1}(2^{-i})$
Linear Rotations	2^{-i}
Hyperbolic Rotations	$\tanh^{-1}(2^{-i})$

3.7 Step-Branching CORDIC

The Step-Branching CORDIC algorithm, developed by Duprat and Muller in 1993, improves the performance of the algorithm by using the Binary Signed Digit (BSD) number system for representing all of the equation variables. The BSD number system, first studied by Avizienis in 1961, is a redundant number system that uses the digit set (-1,0,1). The BSD number system has the beneficial property of very short carry chains. The sum of the bit in position only depends on results from bit position i -1 and bit position i - 2.

3.7.1 Iteration Equations

The iteration equations used by the Step-Branching CORDIC algorithm are the same as those used by the Unified CORDIC algorithm. The only difference between the two algorithms is that the additions in the Step Branching CORIDC algorithm are performed using redundant binary adders and the results are stored as a redundant numbers.

Although the use of the BSD number system improves the performance of the additions, it introduces a different problem. At each iteration of the algorithm, the sign of the current angle, *Zi*, must be determined before the next iteration can begin. The sign of any BSD number is the same as the sign of its most significant non-zero digit. To determine the sign of *Zi*, the most significant non-zero digit must be found, and then its sign must be determined.

If the angle *Zi* is close to zero, almost every single digit would need to be examined. This delay could eliminate much of the gain obtained by using the BSD number system. To avoid examining every digit in *Zi*, as set of digits is examined to determine the sign of the current angle. Examining the subset can be performed in a constant i.e in order to preserve the benefits of using a redundant number system.

In 1990, Ercegovac and Lang implemented an On-Line CORDIC algorithm that calculated the scale factor in parallel with the rotations. The results are normalized once all of the rotations are complete. Even though this algorithm achieves the potential gains of carry free addition, it does so at the expense of additional complexity and computation. A multiplier for calculating the scale factor and a divider for performing the normalization must be designed and implemented.

Even though the multiplications may be computed in parallel with the rotations, the division to normalize the numbers will occur after the rotations have

completed. Due to the complexity of the division, a large portion of the speed up obtained through the use of the redundant number system may be lost. Takagi, Asada, and Yajima proposed a double rotation method in 1987 and a correcting rotation method in 1991 that use redundant number systems to speed the computations of each rotation. Both of these methods preserve a constant scale factor.

Although these methods eliminate the additional complexity required in calculating a variable scale factor, the y still require additional double rotations or correcting rotations.

Even though these operations do not require as much time as a division, these additional rotations prevent these methods from achieving the full potential speedup offered by the carry-free addition.

The Step-Branching CORDIC algorithm takes a different approach to solve this problem. If the examination of the subset of digits is sufficient to show that Zi is positive, then σi is selected to be 1. If the amination of the subset of digits is sufficient to show that Zi is negative, then σi is selected to be1. If the examination of these bits is inconclusive, then two computations are performed in parallel in two separate hardware implementations. One computation, with σi=1, is performed in the "positive" hardware branch, and the other computation, with σi= -1, is performed in the "negative" hardware branch. This covers both possibilities for the value of the residual angle *Zi*.

When this occurs, the algorithm is considered to be in branch mode. Each possible set of calculations continues in parallel until another branch is reached. Once this next branch is reached, the correct set of calculations can be determined from the signs of the angles. The appropriate branch according to +σi and -σ*i* _ is given in Table 3.4.The correct branch is reloaded into each hardware branch and the calculations split into parallel computations again. This process repeats until all of the required iterations have been performed.

Because the two calculations are performed in parallel, the full benefit of carry-free addition from the redundant number system is realized. In addition, by only allowing the sign bit, σ*i*, to take on the values of 1, the Step Branching CORDIC algorithm produces a constant scale factor, so no post-iteration normalization is required. The cost of this speed-up is the addition of three extra addition/subtraction units for the second set of branch hardware. Compared to the addition of a multiplier

and divider, the area required by the additional adders is a small price to pay for the speed-up the algorithm achieves.

Table 3.4- Step Branching Calculation Selection

σ_i^+	σ_i^-	**Correct Branch**
0	*	Positive Branch
1	*	Positive Branch
-1	0	Negative Branch
-1	1	Negative Branch

CHAPTER 4

REALIZATION OF THE CORDIC ALGORITHM

4.1 Basic Architecture

The following diagram explains the basic hardware architecture of a CORDIC processor. It shows the adders/subtractors and the shift registers.The adders/subtractors perform the addition/subtraction of binary numbers. The shift register performs the bit-shift operation in accordance with the algorithm. The constants corresponding to fixed angle values are obtained from the Look-up table implemented as a ROM.

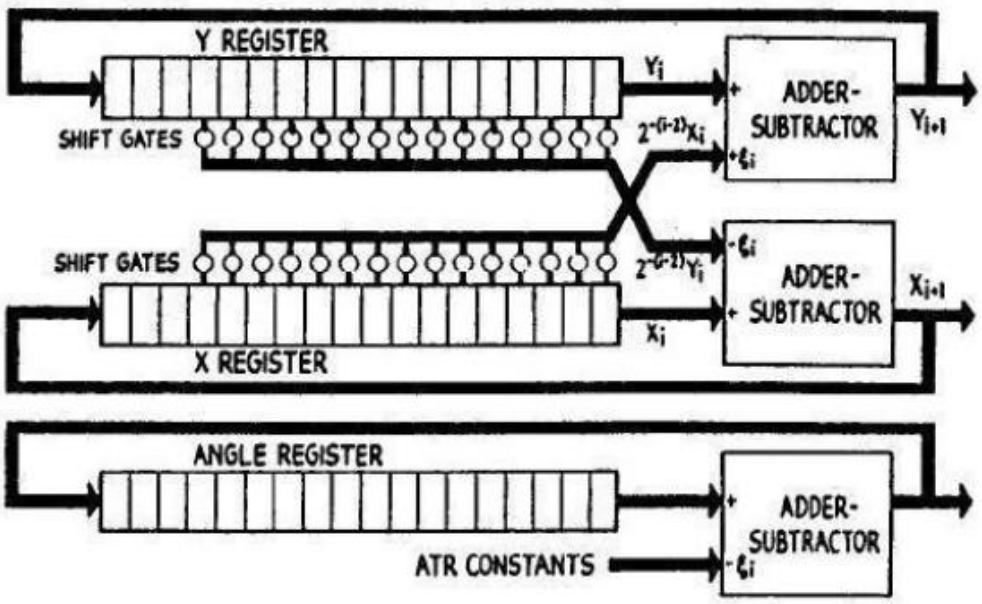


Figure 4.1- Basic CORDIC Architecture

4.2 CORDIC Architecture

CORDIC algorithm, for calculation of sine and cosine values, is of three types. Each of the types has its own advantages and disadvantages depending upon the type of use intended.

The three types are:

1. sequential or iterative
2. parallel or cascaded
3. pipelined

4.2.1 Sequential or CORDIC Iterative structure

In this type of CORDIC architecture, a single iteration process takes place in a single clock cycle. The sequential CORDIC structure is as shown below:

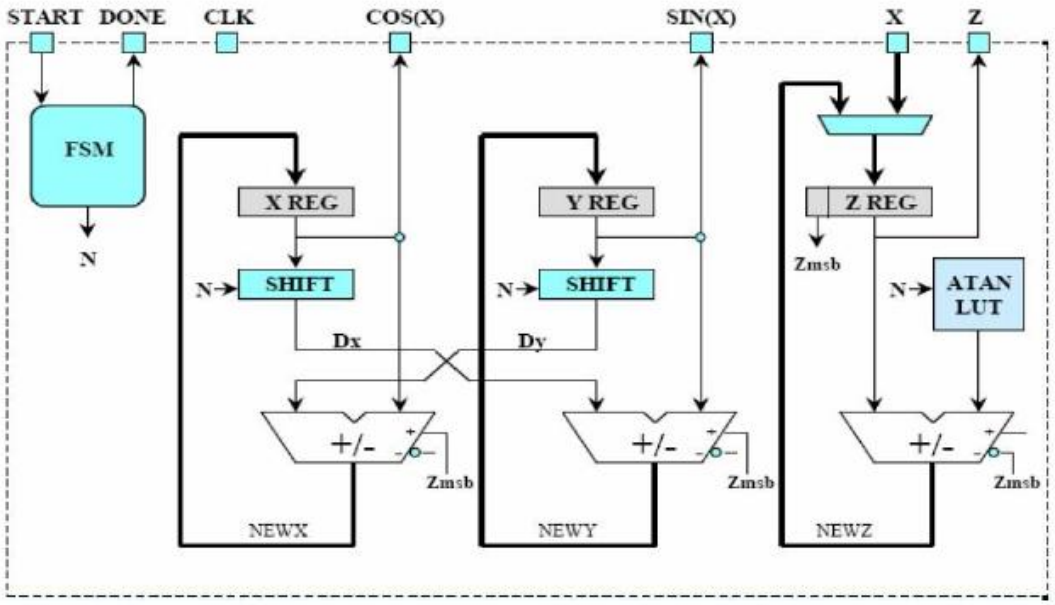


Figure 4.2- Sequential/Iterative CORDIC Structure

Advantages

1. The hardware complexity is least.
2. It has maximum number of clock cycles per iteration.
3. Power consumption is least.

Disadvantages

1.Maximum number of clock cycles are required to calculate the output, thus calculation time is large.

2. Varying shifters do not map well on certain FPGAs due to high fan-in.

4.2.2 Parallel or Cascaded CORDIC architecture:

In this type of architecture, all the iterations take place in a single clock cycle. The architecture is as shown below:

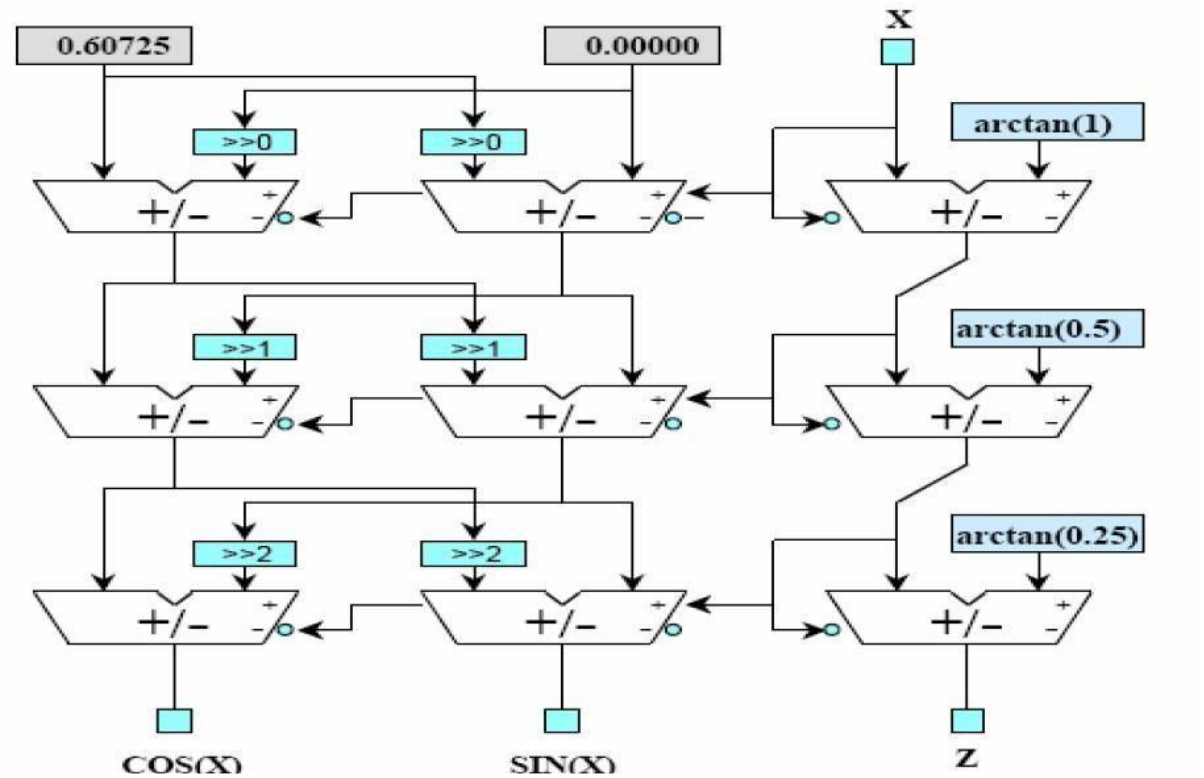


Figure 4.3- Block diagram of Parallel CORDIC architecture

Advantages

1. It has considerable wait, but handling time is reduced as in comparison to the iterative process.
2. Shifters are of fixed size and so can be implemented in the wiring.
3. Always the same can be conventional hardwired instead of demanding storage space.

Disadvantages

1. The amount of components needed is huge and the place needed is maximum.
2. Energy intake is the highest possible among the three CORDIC architectures.

4.2.3 Pipelined CORDIC Architecture

It is comparatively the most efficient CORDIC architecture. In this method multiple iterations take place in multiple clock cycles. It is implemented by inserting registers with in the different adder stages.

The architecture is given as:

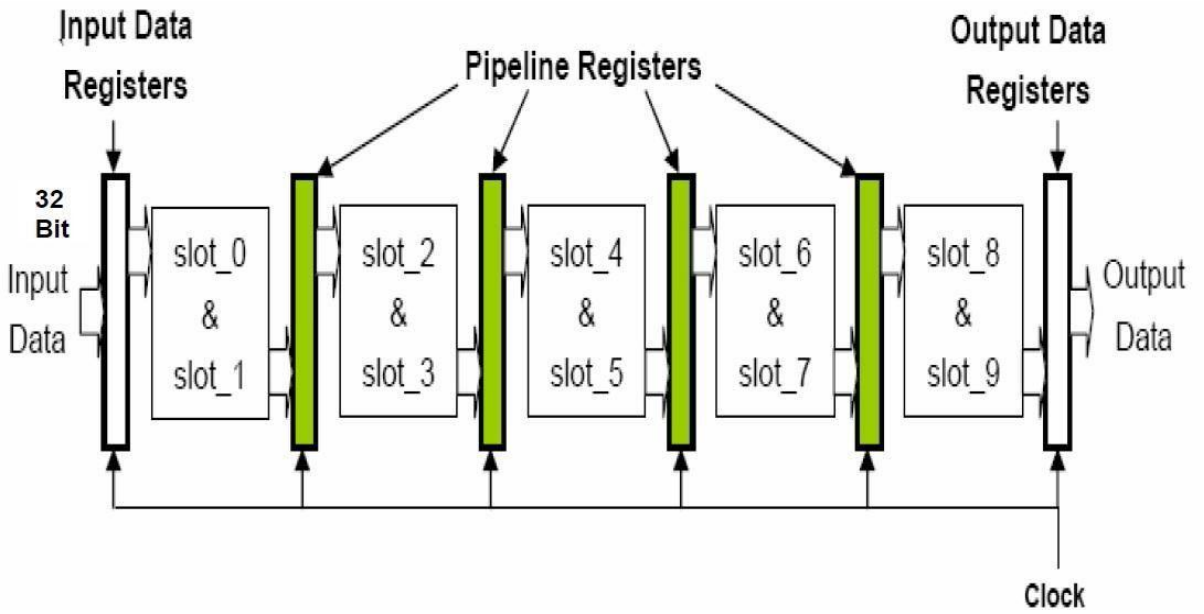


Figure 4.4-Block diagram of Pipe-lined CORDIC Architecture

Depending upon the application, CORDIC Processer is applied in number of ways. The simple structure is sequential structure involve three adder/subtractor and two shifter with a rom containing search table. Serial structure performs one small spinning for every time pattern. Outcome is acquired after n time pattern. Since sequential structure uses n time pattern for every spinning hence it is very slowly.

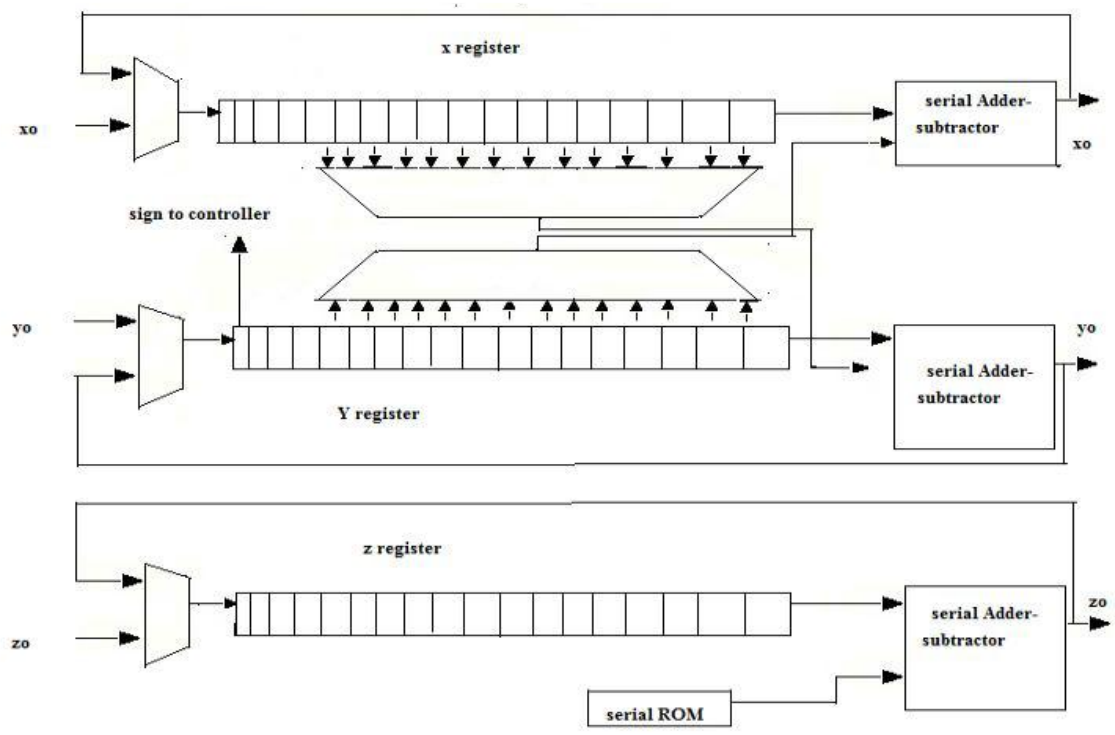


Figure 4.5- Shows Serial architecture

Pipelined structure transforms versions in to direction words. It includes n cascaded prevents. The first outcome of n stage CORDIC is acquired after n time pattern. Thereafter outcome is acquired after every time pattern. Pipelined structure having move sign-up that perform set number of changes every time. Signs up are used to store the position for a particular small rotation.

Pipelined architecture is much faster than serial architecture. Sign 'z' gives the direction of iteration at each stage. In this paper a six stage pipeline sine cosine digital wave generator is developed performing specific micro-rotations. This structure is quick than sequential structure since it doesn't need any search desk. It functions in round spinning method. Sine and Cosine conditions are given by

$$Xn = \cos\theta \tag{4.1}$$

$$Yn = \sin\theta \tag{4.2}$$

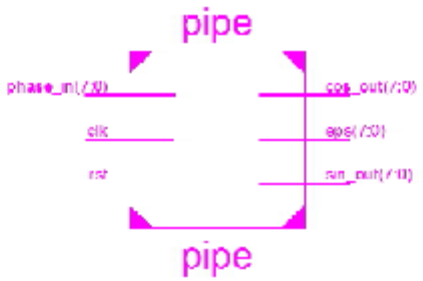


Figure 4.6- RTL View of Pipelined CORDIC

For any angle as an input three outputs are generated sine function, cosine function and eps. eps gives the error proximity to required angle.

Advantages

1. FPGA execution is easy, as signs up are already available, thus demanding no extra components.

2. Number of versions after which the program gives precise result can be made, considering time regularity of the program.

3. When working at higher time period power consumption in later levels reduces due to more compact modifying action in every time period.

Disadvantages

1. Hardware complexity as well as area required is more than sequential architecture
2. Power consumption is lower than parallel but higher than sequential structure.

4.3 FPGA

FPGA or Field Programmable Gate Arrays can be programmed or configured by the user or designer after manufacturing and during implementation. Hence they are otherwise known as On-Site programmable. Unlike a Programmable Array Logic (PAL) or other programmable device, their structure is similar to that of a gate-array or an ASIC. Thus, they are used to rapidly prototype ASICs, or as a substitute for places where an ASIC will eventually be used. This is done when it is important to get the design to the market first. Later on, when the ASIC is produced in bulk to reduce the NRE cost, it can replace the FPGA.

The programming of the FPGA is done using a logic circuit diagram or a source code using a Hardware Description Language (HDL) to specify how the chip should work. FPGAs have automated reasoning elements known as logic blocks, and a hierarchy or reconfigurable interconnects which facilitate the ,wiring of the blocks together. The programmable logic blocks are called configurable logic blocks and reconfigurable interconnects are called switch boxes. Logic blocks (CLBs) can be programmed to perform complex combinational functions, or easy reasoning gateways like AND and XOR. In most FPGAs the reasoning prevents also consist of storage components, which can be as simple as a flip-flop or as complex as complete blocks of memory.

4.3.1 FPGA Architecture

FPGA architecture depends on its vendor, but they are usually variation of that shown in the figure. The structure consists of Configurable Reasoning Prevents, Configurable I/O blocks and Automated Interconnects. It also homes a time circuits to drive time alerts to each logic prevent. Additional logic sources like ALUs, Decoders and memory may be available. Static Ram and anti-fuses are the two primary kinds of automated components for an FPGA. The number of CLBs and I/O s required can easily be determined from the design but the number of routing tracks is different even within the designs employing the same amount of logic.

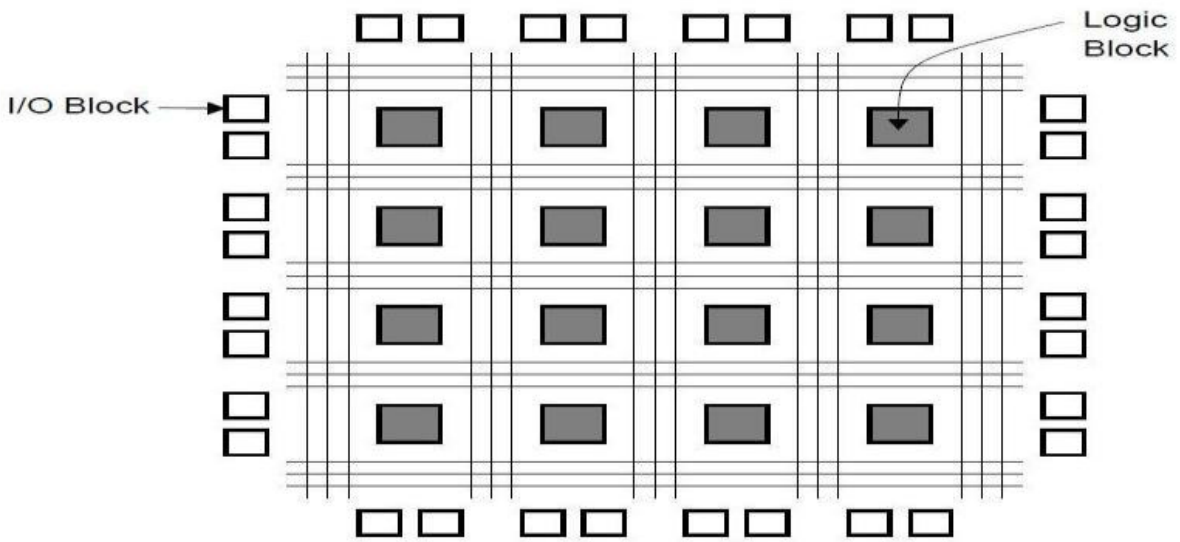


Figure 4 .7- FPGA Architecture

1. *Configurable Logic Blocks*

They contain the logic for the FPGA. CLBs contain RAM for developing irrelevant combinatorial reasoning features. It also has flip-flops for clocked space for storage components, and multiplexers that route the logic within the block to/from external resources.

2.ConfigurableI/O Blocks

Configurable I/O block is used to route signal towards and away from the chip. It comprises input buffer, output buffer with three states and open collector output controls. Pull-up and Pull-down resistors may also be present at the output. The output polarity is programmable for active high or active low output.

3. Programmable interconnect

FPGA interconnect is similar to that of a gate array ASIC and different from a CPLD. There are long lines that interconnect critical CLBs located physically far from each other without introducing much delay.

They also serve as buses within the chip. Short lines that interconnect CLBs present close to each other are also present. Switch matrices that connect these long and short lines in a specific way are also present. Programmable Switches connect CLBs to interconnect collections and interconnect collections to each other and the switch matrix. Three- state buffers connect multiple CLBs to a lengthy range developing a bus. Specially designed long lines called Global Clock lines are present that provide low impedance and fast propagation times.

4. *Clock circuitry*

Special I/O blocks having special high-drive clock buffers, called clock drivers, are distributed throughout the chip. The buffers are connected to clock I/P pads. They drive time alerts onto the International Clock liens described above. The clock lines have been designed for fast propagation time and less skew time.

4.4 Advantages of FPGA

FPGAs have become very popular in the recent years owing to the following advantages that they offer:

1. ***Fast prototyping and turn-around time***- Prototyping is the defined as the building of an real routine to a theoretical style to confirm for its operating, and to offer a actual physical system for debugging the core if it doesn't. Turnaround is the total time between expired between the submission of a process and its completion. On FPGAs interconnects are already present and the designer only needs to fuse these programmable interconnects to get the desired output logic. This reduces the time taken as compared to ASICs or full-custom design.

2.NRE cost is zero- Non-Recurring Engineering refers to the one-time cost of researching, developing, designing and testing a new product. Since FPGAs are reprogrammable and they can be used without any loss of quality every time, the NRE cost is not present. This significantly reduces the initial cost of manufacturing the ICs since the program can be implemented and tested on FPGAs free of cost.

3.High-Speed- Since FPGA technology is primarily based on referring to the look-up Tables the time taken to execute is much less compared to ASIC technology. Due to the above mentioned advantages of FPGAs in IC technology and DCT in mapping of images, implementation of DCT in FPGA can give us a clearer idea about the advantages and limitations of using DCT as the mapping function. This can help in forming better image compression and restoration techniques.

4.5 Implementation of exponential and logarithmic functions

4.5.1 Powering Function:

The complexity of the powering function, x_y (where x is the base and y the exponent), makes very difficult to implement an efficient and accurate operator in a direct way without any range reduction. However it can be reduced to a combination of other operations and calculated straight forward with the transformation.

A direct implementation of this approach with three sub-operators (a logarithm, a multiplier and an exponential) presents three main problems that have to be effectively handled:

1. The enormous complexity of both exponential and logarithm functions. However, the use of table driven methods in combination with range reduction algorithms makes possible their implementation.

2. The computation with a negative base results in Not a Number even though the powering function is defined for negative bases and integer exponents.

3. Equation (7.1) can lead to a large error in the result. Although the sub-operators were almost exact the relative error from each sub-operator spreads through the equation generating the final large relative error. Extending the precision of the partial results in an effective way to minimize these relative errors.

To the best of our knowledge there are only two previous works focused on the exponential function and only one for the logarithm function (from same authors of).

The first one employs an algorithm that does not exploit the FPGA characteristics, and consequently presents poor performance. The other two implementations are part of a common work and are designed suiting with FPGA flexibility (using internal tailored fixed arithmetic and exploiting the parallelism features of the FPGA) achieving much better results.

They are parameterizable implementations that, additionally to single floating point format, also allow smaller exponent and mantissa bit-widths and are both based on input range reduction and table-driven methods to calculate the function in the reduced range.

Our e^x and ln x units, based on these units, include the following innovative features:

Single precision floating point. arithmetic extensions were designed considering only normalized numbers, not denormalized. Additional logic has been introduced to handle denormalized numbers at the output of *ex* and the input of ln *x*.

1. Redesign of units to deal only with single precision. The feature of bit-width configurability of the base designs has been removed. Thus, the resources needed have been reduced because specific units, just for single precision, have been developed to

Simplification of constant multiplications. As suggested in conventional multipliers have been removed where the multiplications involved constant coefficients, improving performance and reducing size.

2. Unsigned arithmetic. In internal fixed arithmetic with sign is used. However, some

operations (like the ones involving range reduction and calculation of the exponent for the result in *ex*) are consecutive and related, and the sign of the result can be inferred from the input sign. For such operations signed arithmetic has been replaced by unsigned arithmetic with the corresponding logic reduction.

3. Improved pipelining. The speed is enhanced by systematically introducing pipeline stages to the data path of the exponential and logarithm units and their subunits.

The paper explains about the implementation of power and log function based on a simple modification of power series expansion of Taylor series. In power function implementation, the paper aims at reducing the exponent number to a smaller value. It requires a large amount of block ram and hardware multipliers as well. It becomes platform dependent and the clock frequency may vary from vendor to vendor. The degradation in throughput rate is due to the use of 18 X 18 embedded multipliers in it. The powering unit also requires more number of stages which may be reduced further.

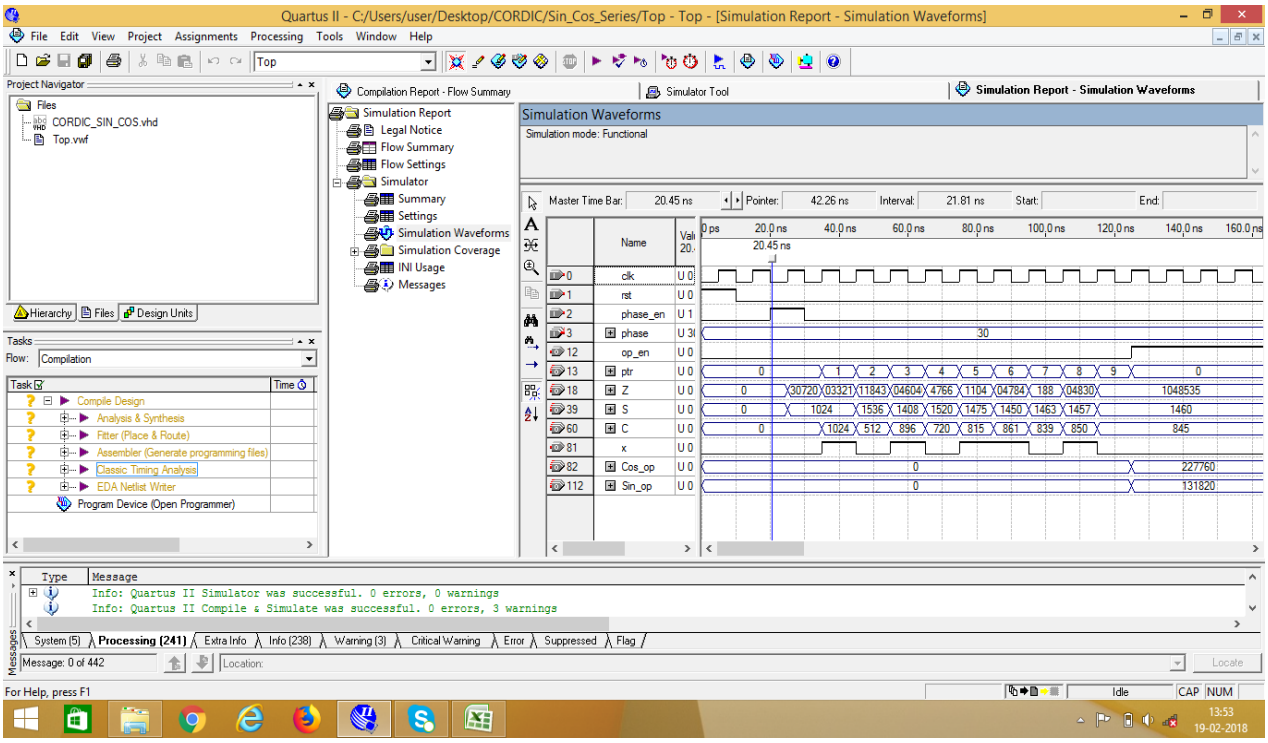
In the proposed method, we are going to reduce delay and improve the throughput rate by avoiding the embedded multipliers and block RAMs. In this paper, we are not completely avoid look up tables, but any value of logarithm or exponential can be calculated, by adjusting the look up table values to the desired number.

CHAPTER 5
SIMULATION RESULTS

For proving our algorithm, we are using Stratix II device and Quartus 9.1 editions from ALTERA software. In logarithm implementation, the input is to be multiplied by 65536 and the output we get has to be divided by 65536 in order to get the actual value. The output values of our algorithm are compared with the MATLAB.

In the context of FPGA implementation of trigonometric functions using the Cordic algorithm, the serial version of the sine and cosine series refers to the sequential computation of trigonometric values using a series expansion approach. This method involves calculating trigonometric functions such as sine and cosine by summing up terms in their respective Taylor series expansions.

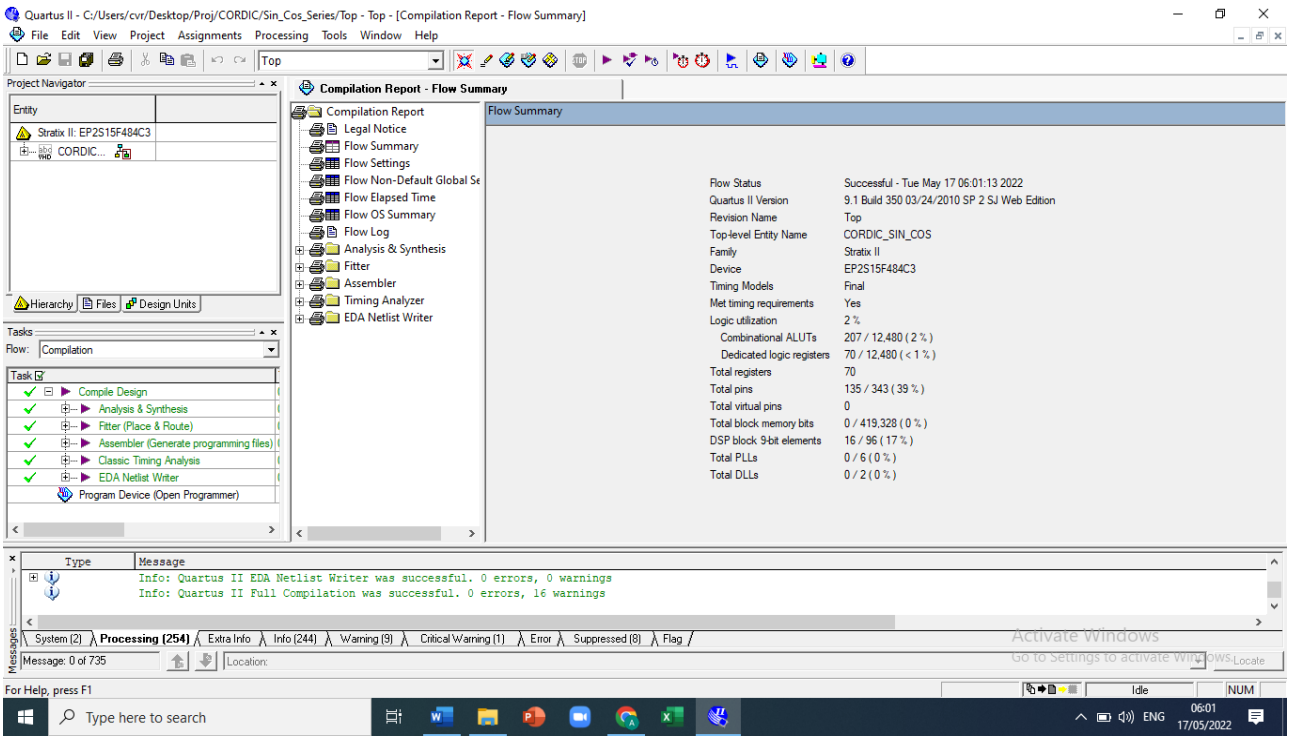
In the serial version, each term of the series is computed sequentially, leading to a slower but potentially more accurate computation compared to parallel methods. The Cordic algorithm, however, offers a more efficient alternative by using iterative rotations and shifts to approximate trigonometric functions, making it well-suited for FPGA implementations due to its simplicity and resource efficiency.



Serial version of Sine Cosine series

In the implementation results of the serial version for sine and cosine on FPGA using the Cordic algorithm.

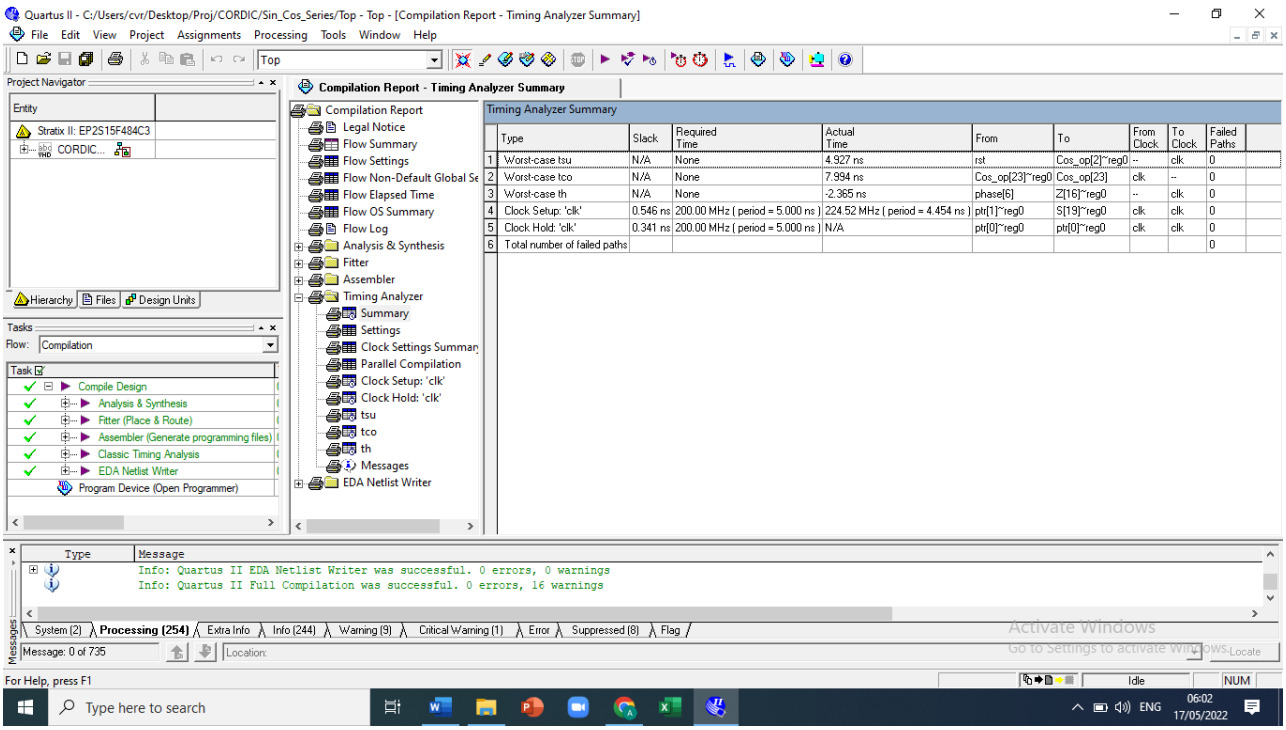
Overall, the implementation results would provide insights into the feasibility and efficiency of using the Cordic algorithm for computing sine and cosine functions on FPGA platforms in a serial manner.



Implementation results for Sine Cosine serial version

In the context of FPGA implementation of trigonometric functions using the Cordic algorithm, a timing analyzer for the sine and cosine serial version would examine the critical paths and timing constraints of the design. It helps ensure that the implemented circuit operates correctly within the specified clock frequency.

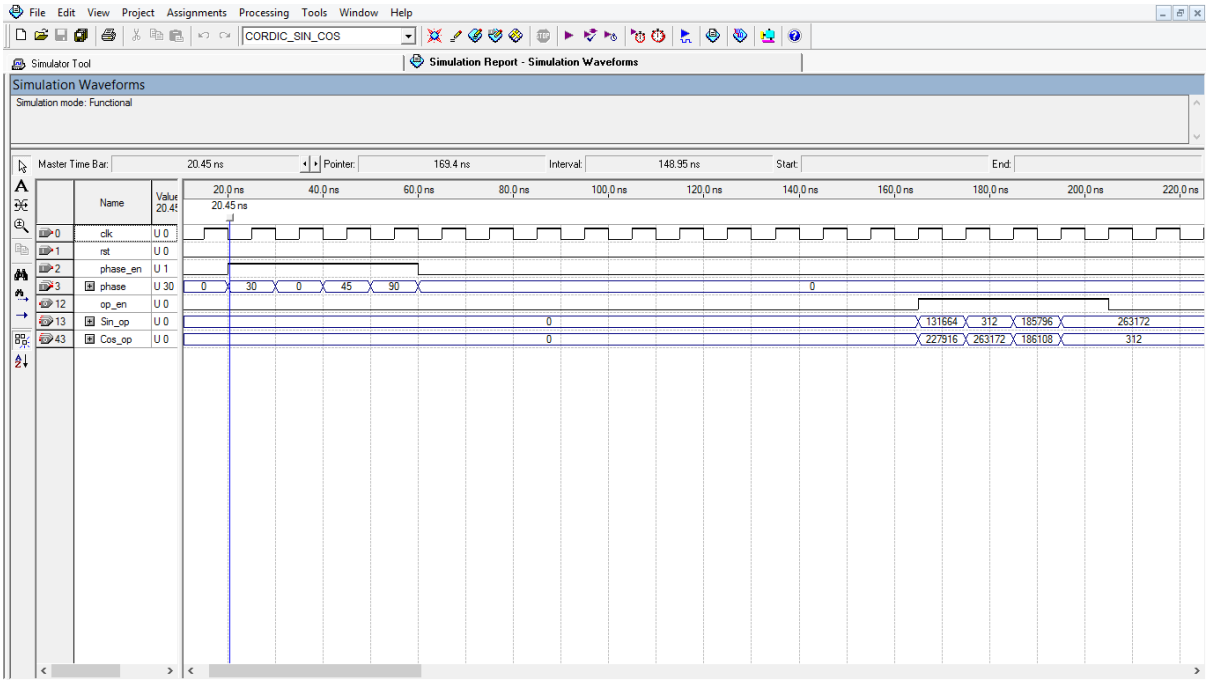
Overall, the timing analyzer plays a crucial role in optimizing the performance and reliability of the FPGA implementation of trigonometric functions using the Cordic algorithm in the serial version, ensuring that the design meets its timing requirements.



Timing analyzer for the Sine Cosine serial version

In the context of FPGA implementation of trigonometric functions using the Cordic algorithm, a pipelined version of the sine and cosine series involves breaking down the computation into sequential stages or pipeline segments. Each stage performs a portion of the calculation, allowing multiple calculations to be in progress simultaneously, thereby increasing throughput and reducing latency.

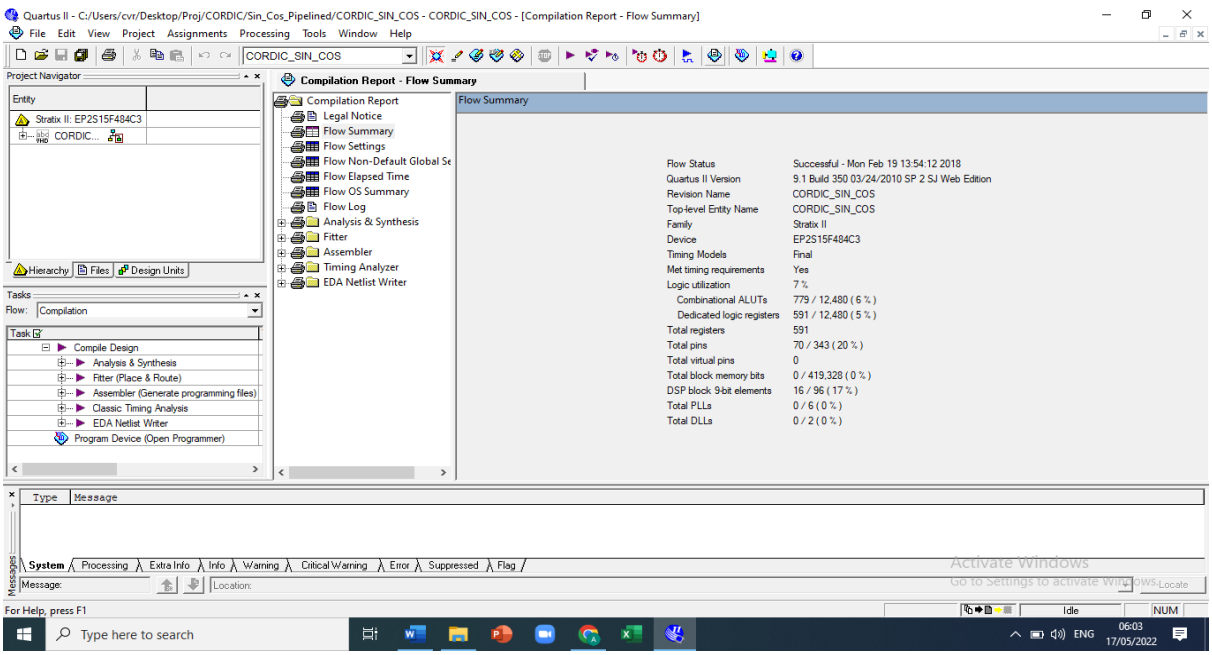
Overall, the pipelined version of the sine and cosine series offers improved performance and efficiency in FPGA implementations by leveraging parallelism and overlapping computations.



Pipelined version of Sine Cosine series

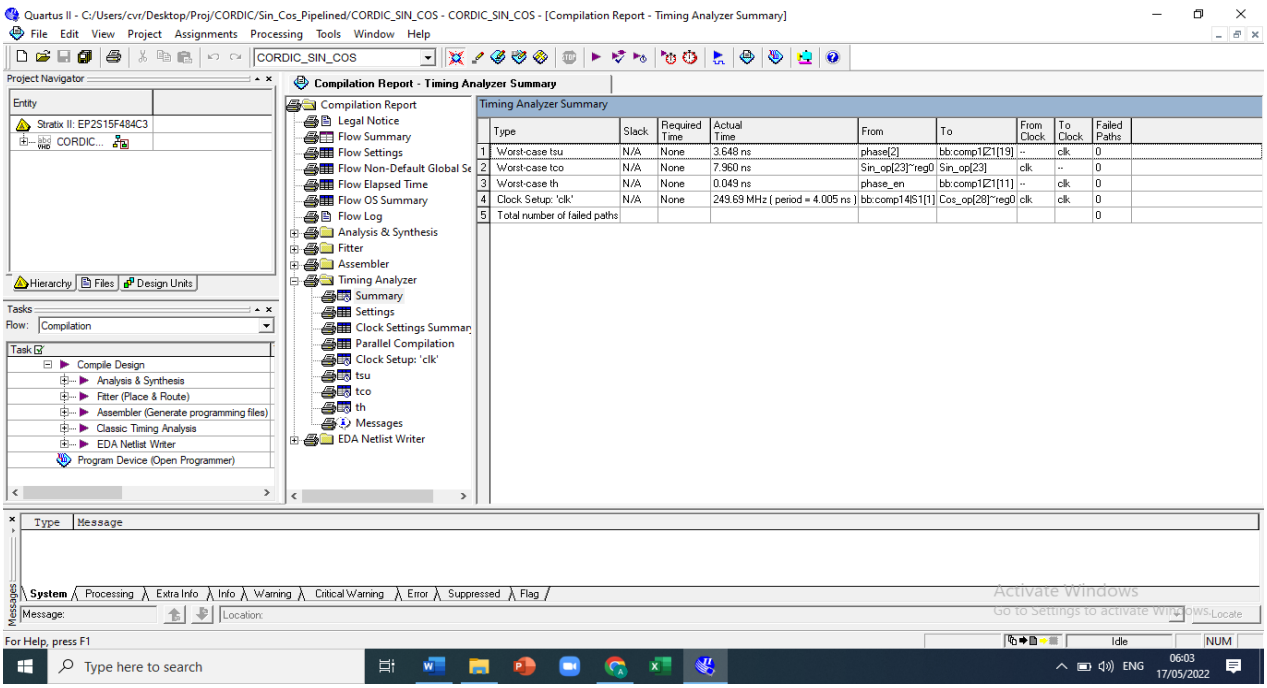
A compilation report for the pipelined version of the sine and cosine realization in FPGA implementation using the Cordic algorithm would provide detailed information about the synthesis and implementation process.

Overall, the compilation report provides a comprehensive overview of the pipelined version of the sine and cosine realization, detailing resource utilization, timing characteristics, optimization strategies, and verification results to evaluate the effectiveness and efficiency of the FPGA implementation using the Cordic algorithm.



Compilation report for the pipeline version of the Sine Cosine realization.

In the context of FPGA implementation of trigonometric functions using the Cordic algorithm, a timing analyzer for the pipelined version of the sine and cosine serial realization would focus on analyzing the timing characteristics of the pipelined design. The timing analyzer plays a crucial role in optimizing the performance and reliability of the pipelined version of the sine and cosine realization on FPGA, ensuring that the design meets its timing requirements and operates efficiently using the Cordic algorithm.



Timing analyzer report for the pipelined version of the Sine Cosine serial realization.

CHAPTER 6
CONCLUSION

The advantage of the design proposed in this paper is that no DSP or multiplication blocks are used. As 16 bit precision is used, the accuracy of the design is high. The only disadvantage is that the number of iterations required is slightly more. This block can be used in few decoding algorithms in communication systems. The design will be used in LDPC decoder sum product algorithm, IMAGE ENHANCEMENT ALGORITHMS the parallel architecture has high throughput (i.e. speed) as compared to serial architecture.

While implementing exponential algorithm, we need to multiply input by 65536 to ensure the floating point number converts to a fixed point number. Here we are going to truncate the value to the nearest value. This paper aims at implementing the exp (x) where x value varies from 0 to ±50. But it can be extended further by increasing the bit length required to store the data. But it requires more hardware at the expense of a little delay.

REFERENCES

[1] J.E. Volder, “The CORDIC trigonometric computing technique,” IRE Transactions on Electronic Computers, vol. EC- 8, pp. 330–334, Sept. 1959.

[2] B. Gisuthan and T. Srikanthan, “Pipelining flat CORDIC based trigonometric function generators,” Microelectronics Journal, volume 33, Pp.77–89, 2002.

[3] E. Deprettere, P. Dewilde, and R. Udo, “Pipelined CORDIC architectures for fast VLSI filtering and array processing,” in IEEE International Conference on Acoustic, Speech, Signal Processing, ICASSP’84, March 1984, volume 9, pp.250–253.

[4] C. C. Doss and R. L. Riley, \FPGA-Based execution of a effective IEEE-754 rapid device," in IEEE Field-Programmable Custom Computing Machines, 2004, pp.229{238.

[5] J. Detrey and F. Dinechin, \A parameterized floating-point exponential function for FPGAs," in IEEE International Conference Field-Programmable Technology, 2005, pp.27{34.

[6] A parameterized °oating-point logarithm operator for FPGAs," in Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference,2005, pp.
1186{1190.

[7] Pedro Echeverra, Marisa Lopez-Vallejo,”An FPGA Implementation of the Powering function with Single Precision Floating-Point Arithmetic*”*.

APPENDIX OVERVIEW OF TOOL

1.For creating new project:

a. File -> new project wizard -> Introduction (next) ->

b. In the directory page, specify the directory name and name of the project (name of the project and top level design entity should be same)

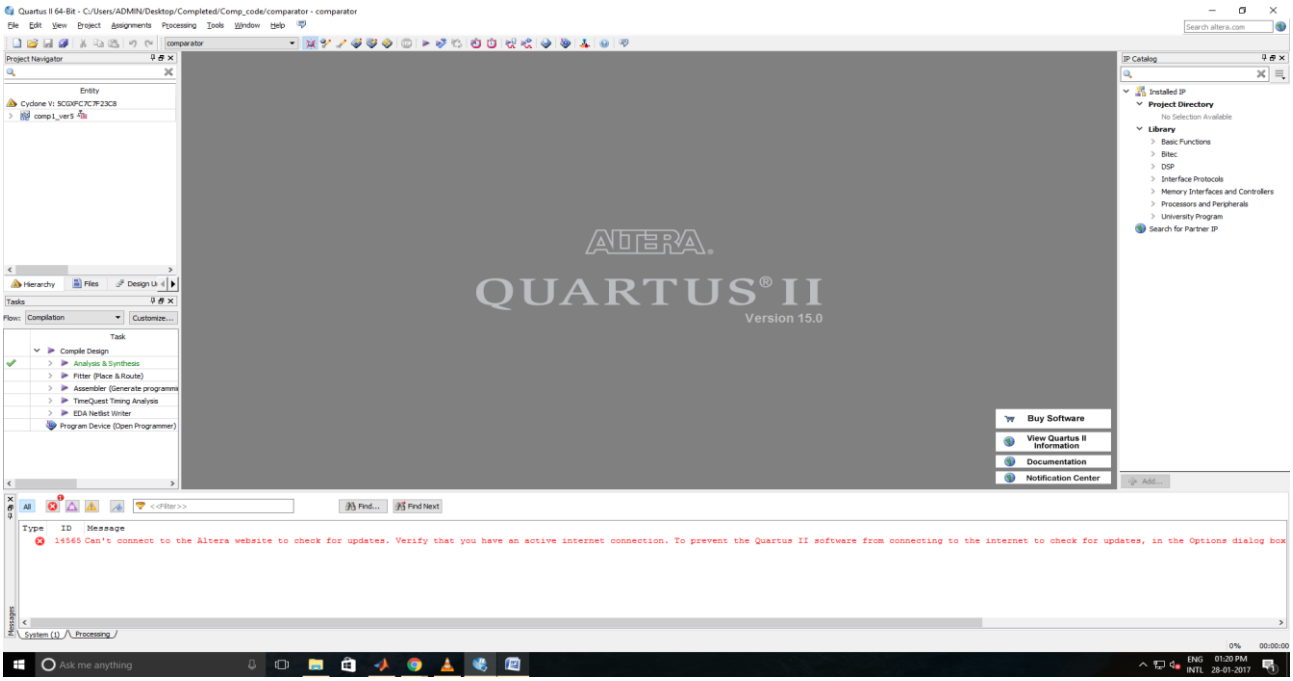
c. In the family & device settings, specify the family name, device name, package, pin count and speed grade.

d. Select the device name which is installed (For a particular device you can have a look at the core voltage, ALM, IO, memory and PLL)

e. In the EDA toll settings, select VHDL/verilog in the simulation format.

2. If you have already created a project, select the qpf file from the file -> open project (go to the directory and select the .qpf file from the corresponding directory)

3. In the files select new and choose VHDL or Verilog HDL file. The name of the entity and file name should be same.



4. In the entity box, there are three sub menus (hierarchy, Files and design units). The added file should be in this submenu else we need to add the file. Right click on the file in the file submenu and select the file from the corresponding directory.

5. Right click on the file name and set as top level entity.

6. Once the code is written, the code should be compiled. In the processing menu-> start compilation option is there. (ctrl + L short cut key for start compilation)

7. If there are errors, look for the errors in the message box located at the bottom. Rectify the errors and compile once again.
8. Once everything is okay, in the file menu select new and choose the waveform editor and save the waveform file.
9. Add the nodes to the waveform editor
10. You have the option for adding clock, constant signal, counter, random generator to the signals.
11. In the processing menu, select the simulator tool and add the waveform file . Select the simulation type as functional / timing.
12. Start simulation.