COMPARATIVE ANALYSIS OF VARIOUS CORDIC TECHNIQUES

Thesis submitted in partial fulfillment of the requirements

for the award of degree of

Master of Technology (VLSI design & CAD)

Submitted by

Deepika Ghai

Roll no:-600961025

Under the Guidance of

Dr. Kulbir Singh

Assistant Professor, ECE

Department of Electronics and Communications Engineering

Thapar university, Patiala-147004, India

June, 2011

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"Comparative Analysis of various CORDIC techniques",** in partial fulfillment of the requirements for the award of degree of **Master of technology in VLSI Design and CAD** at **Thapar University, Patiala** is an authentic record of my own work carried out under the supervision of **Dr. Kulbir Singh** and referred other researcher's work which are duly listed in the reference section.

The matter embodied in this thesis has not been submitted for the award of any other degree of this or any other University.

Deepika Ghai
(Deepika Ghai)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Kulbir Singh)
ECED
Thapar University,
Patiala-147004.

Dr. A.K Chatterjee
Professor & Head,
ECED
Thapar University,
Patiala-147004.

Dr. S.K Mohapatra
Dean of Academic Affairs,
Thapar University,
Patiala-147004.

ACKNOWLEDGEMENT

This report is completed with prayers of many and love of my family and friends. However, there are a few people that I would like to specially acknowledge and extend my heartfelt gratitude who have made the completion of this report possible. With the biggest contribution to this report. I would like to thank my guide Dr. Kulbir Singh who had given me his full support in guiding me with stimulating suggestions and encouragement to go ahead in all the time of thesis work.

I am also thankful to Dr. A.K. Chatterjee, Head, Electronics and Communication Engineering Department, for the providing us with adequate infrastructure in carrying the work.

I am also thankful to Dr. Alpana Aggarwal, P.G. Coordinator, Electronics and Communication Engineering Department, for the motivation and inspiration that triggered me for the thesis.

At last but not the least my gratitude towards my parents, I would also like to thank God for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

Date: (Deepika Ghai)

ABSTRACT

CORDIC is an acronym for COrdinate Rotation Digital Computer. The CORDIC method is the most versatile of all the algorithms that can be used to evaluate elementary functions. The same hardware can be used to compute trigonometric ratios (sin, cos, tan, etc.), hyperbolic ratios (sinh, cosh, tanh), multiplication, division, inverse trigonometric (arcsin, arccos) and inverse hyperbolic ratios (arcsinh, arccosh), with a slight modification it can also compute logarithms, exponentials, etc. In this work, it has been found that CORDIC algorithm requires 3 adders/subtractors and 3 registers. So it is multiplierless approach and it saves a lot of hardware and hence power dissipation is very low as compared to other methods. Due to the simplicity of the involved operations, the CORDIC algorithm is very well suited for VLSI implementation. In this work, CORDIC algorithm, pipeline CORDIC, control CORDIC and DFT (Discrete Fourier Transform) have been implemented in XILINX Spartan 3E FPGA kit using VHDL. The comparison of original CORDIC on the basis of their power, speed, area required to implement in chip designing, number of iteration etc. have been discussed. It has been found that as data rate increases, number of slices, flip-flops, LUTs, IOBs, power and delay have also increased. Look up tables have been used to compare precision for different data rates. It has been observed that resolution of CORDIC algorithm is best for 32 bit data rate.

Original CORDIC algorithm does have one drawback however, in that the algorithm exhibits linear convergence, so that N iterations are required to converge to N bits of accuracy. When used in modern computing elements which operate at a high clock frequency, this large latency has a deleterious effect on overall system performance. In this work, the latency has been calculated to be equal to 8.191 ns for 12-bit original CORDIC algorithm. Latency can be further reduced by pipeline CORDIC. It has been found that latency is reduced by 34.97% using pipeline CORDIC as compared to Original CORDIC. It has also been observed that pipeline CORDIC reduces the number of resources by 7.5% as compared to original CORDIC. In addition to it, power dissipation in pipeline CORDIC has also been reduced by 6.74% as compared to original CORDIC.

In this work, it has been found that Original CORDIC algorithm, the iteration variable (z_i) does not always converge monotonically to 0–some of the iterations may actually result in divergent micro-rotations, which do nothing to improve the convergence towards the target vector. This problem has been solved using control CORDIC method which

modifies the angle trajectory so that it now resembles a critically damped system, with no overshoot, resulting in faster convergence.

The application of CORDIC has been shown using Discrete Fourier transforms (DFT). It has been found that in DFT implementation number of multiplier, adders/subtractors, registers, comparators are 4, 6/5, 121, 5 respectively. Due to high speed, low cost and greater flexibility offered by FPGAs over DSP processors, the FPGA based computing is becoming the heart of all digital signal processing systems of modern era.

LIST OF ACRONYMS

ASICs	Application-Specific Integrated Circuits
ALUs	Arithmetic Logic Unit
B-58	Bomber's Navigation Computer
CLBs	Configurable Logic Blocks
CPLD	Complex Programmable Logic Device
CORDIC	COrdinate Rotation DIgital Computer
DFT	Discrete Fourier Transform
DHT	Discrete Hartley Transform
DCT	Discrete Cosine Transform
DST	Discrete Sine Transform
DSP	Digital Signal Processing
EVD	Enhanced Versatile Disc
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
HP	Hewlett-Packard
IOBs	Input Output Blocks
LUTs	Look Up Table
MAC	Multiply and Accumulate
NCD	Native Circuit Description
NGC	Native Generic Complier
NGD	Native information and Generic Database
PAL	Programmable Array Logic
PAR	Place and Route
RAM	Random Access memory
ROM	Read Only Memory
RTL	Register Transfer level
UCF	User Constraints File
VHSIC	Very High Speed Integrated Circuits
VHDL	VHSIC Hardware Descriptive Language
VLSI	Very Large Scale Integration

LIST OF FIGURES

LIST OF TABLES

CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Introduction

CORDIC is a special purpose digital computer for real time computation. The CORDIC algorithm was first introduced by Jack E. Volder in the year 1959 [1]. The Volder's algorithm is derived from the general equations of vector rotation. CORDIC is used for the computation of trigonometric functions (like $\sin, \cos, \tan$), magnitude and phase (arctangent), hyperbolic functions (like $\sinh, \cosh, \tanh$), multiplication, division, data type conversion, square roots, logarithms and was further extended by John Walther in 1971 [2]. It is highly efficient, low complexity and robust technique to compute the elementary functions. The CORDIC algorithm has found its various applications such as pocket (scientific) calculator, numerical co-processors, signal processing (DFT, DCT, DHT, DST), image processing applications, direct digital synthesis and analog to digital modulation etc [3].

CORDIC stands for COrdinate Rotation DIgital Computer. The CORDIC algorithm does not use calculus based method such as polynomial or rational function approximation. Earlier methods used are table look up method, polynomial approximation method etc. for evaluation of trigonometric functions. Calculation of an elementary function is often times done by using look up table. Look up tables are by far the fastest way to make computation; however the precision of the result is directly related to size of the look up table [4]. High precision look-up tables require a large amount of non-volatile memory to store the table. If the table size is reduced to save memory, precision will also be reduced. But power series may also be used to calculate these same functions without using look-up table; however these calculations have the disadvantage of being slow to converge to a desired precision. In effect, the look-up table size is being traded at the expense of computation time. Power series and Fourier series is time consuming. So we use look up table. Taylor series can be expensive to compute, especially on slow processors. But CORDIC is fast even on slow processor.

Today CORDIC algorithm is used in VLSI design, high performance vector rotation DSP applications, advanced circuit design and optimized low power design [2]. In CORDIC, the "multiplies" can be power of 2, so in binary arithmetic they can be done using just shifts and adds. No actual "multiplier" is required thus is simpler and do not require complex hardware structure as in case of multiplier

The CORDIC technique uses a one bit at a time approach to make computation to an arbitrary precision [4]. Typically, these tables only one to two entries per bit of precision. CORDIC algorithms also use only right shifts and additions, minimizing the computation time. It is hardware efficient algorithm because no multipliers are presenting in CORDIC, to save gate required implementing on FPGA. If multiplier is present, then cost and number of gates increases. The CORDIC algorithm has become a widely used approach to elementary function evaluation where the silicon area is a primary constraint. The implementations of CORDIC algorithm require less complex hardware. Hardware multiplier is unavailable in CORDIC because the strength of CORDIC algorithm is its ability to solve with vector rotation without using a multiplier and to speed up CORDIC algorithm [5]. The only operation is addition, subtraction, bit shift and lookup table. The rotated vector is also scaled making a scale factor necessary. Due to high speed, low cost and greater flexibility offered by FPGA over DSP processors, the FPGA based computing is becoming the heart of all digital signal processing systems of modern era.

There are two ways in CORDIC algorithm for calculation of trigonometric and other related functions: - rotation mode and vectoring mode [6]. Both methods are angle accumulator with desired angle value. In rotation mode, the co-ordinate component of a vector and angle of rotation are given and the coordinate component of the original vector, after rotation through a given angle are computed (z converge to zero). In vectoring mode, the co-ordinate components of a vector are given and the magnitude and angular argument of the original vector are computed (y converge to zero). CORDIC is an iterative method, it has the advantage over the other methods of being able to get better accuracy by doing more iteration, where as the Taylor approximation and polynomial approximation methods need to be re-derived to get better result [7]. These properties in addition to getting a very accurate

approximation is perhaps the reason why CORDIC is used in many scientific calculators today. Original CORDIC takes N=8 iterations, to reduce the number of iterations control CORDIC is used [8]. Control CORDIC reduces the number of iterations almost by 50%. In the original CORDIC, overshoot takes place to eliminate this in control CORDIC simple selection function is used. The simple selection function decides which angle constant should be skipped to eliminate the overshoot. In original CORDIC, z does not converges to zero. Control CORDIC has capability to converge faster.

Pipeline architectures are used in CORDIC algorithm to reduce the critical path by introducing pipeline latches; this increases the clock speed or sampling speed and reduces the power consumption at the same speed in a DSP system [9]. If an iterative implementation of CORDIC were used, the generator would take several clock cycles to build a single output sample of the wave. However, using pipeline converts iterations into pipeline phases. In this way an output is obtained at every clock cycle, after pipeline stages propagation. Each pipeline stage takes exactly one clock cycle to complete.

1.2 Thesis Objective

Based on the above discussion the thesis has following objectives:

1) To study and implement Original CORDIC and Control CORDIC algorithm using VHDL programming code.
2) To implement pipeline CORDIC algorithm using VHDL programming code.
3) To implement DFT using CORDIC algorithm in VHDL code.
4) To implement CORDIC algorithm, pipeline CORDIC and DFT on XILINX SPARTAN 3E kit.

1.3 Organization of Thesis

Chapter 2 discusses about literature survey of CORDIC.

Chapter 3 discusses about basics of CORDIC algorithm, how it came into picture, its basic equations, different mode of operation i.e. rotation mode and vectoring mode, gain factor, CORDIC iteration and how it works and their block diagram, their comparison on the basis of

their complexity, speed, area required to implement in chip designing, number of iteration required etc and discusses about the control CORDIC and pipeline CORDIC.

Chapter 4 discusses use of CORDIC algorithm for calculating DFT, DHT, calculation of DFT using FFT algorithm, basic equation used.

Chapter 5 tells about the design flow of XILINX FPGA. This chapter includes FPGA architecture, its logic blocks, technology used, placement and routing, testing and design issues.

Chapter 6 contains the results of simulation using ModelSim 6.3f and XILINX 10.1i. The thesis concludes in chapter 7.

1.4 Methodology

In this work, VHDL programming has been used to implement CORDIC algorithm (to calculate Sine and Cosine value for a given angle), control CORDIC, DFT (Digital Fourier Transform) and pipeline CORDIC. Further XILINX SPARTAN 3E kit is used for FPGA implementation of the generated VHDL code. Programming tools used for the implementations are:

1) ModelSim 6.3f
2) XILINX 10.1i
3) FPGA kit SPARTAN 3E

CHAPTER 2
LITERATURE SURVEY

CORDIC was introduced in 1959 by Jack E. Volder as a highly efficient, low complexity, and robust technique to compute the elementary functions [1]. It is initially intended for navigation technology, the CORDIC algorithm has found its way in a wide range of applications, ranging from pocket calculators, numerical co- processors, to high performance radar signal processing.

After invention CORDIC worked as the replacement for the analog navigation computers aboard the B-58 supersonic bomber aircraft with a digital counterpart. Further in 1971, Steve Walther continues work on CORDIC, with the application of the CORDIC algorithm in the Hewlett-Packard calculators, such as the HP-9100 and the famous HP-35 in year 1972, the HP-41C in year 1980 [6][10]. He told how the unified CORDIC algorithm i.e. combining rotations in the circular, hyperbolic and linear co-ordinate systems and how it was applied in the HP-2116 floating –point numerical co-processor.

In the past decade, the unprecedented advances in VLSI technology have stimulated great interests in developing special purpose, parallel processor arrays to facilitate real time digital signal processing [11]. Parallel computing systems such as systolic arrays and wave front arrays have been extensively studied. The basic arithmetic computation of these parallel VLSI arrays has often been implemented with a multiplication and accumulation (MAC) unit, because these operations arise frequently in DSP algorithms. The reduction in hardware cost also motivated the development of more sophisticated DSP algorithms to enhance the performance of modern digital signal processing systems. Many of these new algorithms require the evaluation of elementary functions, such as trigonometric, exponential, and logarithm functions, which cannot be evaluated efficiently with MAC based arithmetic units. Consequently, when DSP algorithms incorporate these elementary functions, it is not unusual to observe significant performance degradation. On the other hand, an alternative arithmetic computing algorithm known as CORDIC (Coordinate Rotation DIgital Computer) has received renewed attention, as it offers a unified iterative formulation to efficiently evaluate

each of these elementary functions. Specifically, all the evaluation tasks in CORDIC are formulated as a rotation of a 2 x 1 vector in various coordinate systems. By varying a few simple parameters, the same CORDIC processor is capable of iteratively evaluating these elementary functions using the same hardware within the same amount of time.

Today fast rotation techniques are closely related to CORDIC, to perform orthonormal rotation at a very low cost. Although fast rotations exist for certain angles only, they are sufficiently versatile, and have already been widely applied in signal processing. Hoekstra found a large range of known, and previously unknown, fast rotation methods. An overall evaluation of the methods exposes the trade-offs that exist between the angle of rotation, the accuracy in scaling and the cost of rotation. Vander kolk et al. formalized the problem of (approximate) vectoring for fast rotations in year 2000. They treated the fast and efficient selection of the appropriate fast rotation, and showed the advantage to be gained when applied to Enhanced Versatile Disc (EVD)[12]. The selection technique works equally well for arithmetic and floating point computation. Antelo et al. considers going to a higher radix than the radix-2 for the classical algorithm, so that less iterations are required. The choice of higher radix implies that the scaling factor is no longer constant.

Year 2009 marks completion of 50 years of the CORDIC algorithm. Due to rapid advances in VLSI technologies have led the way to an entirely different approach to computer-design for real –time application, using special purpose architecture with custom chips. Special–purpose architectures efficiently map the algorithm needs of the problem to the hardware [3]. Now a day's world of information interchange revolves around transmission and viewing real time images. Many of DSP applications try to closely simulate real life images. Speed, clarity and resemblance to real time object are some of the issues to be addressed in order to achieve the goal.

Trigonometric function calculation is one of the primary tasks performed in DSP applications. For long time microprocessor-based systems have been used to perform this task. Software algorithms used to processor do not meet the highly demanding needs of the entire task [6]. Using hardware system to perform these DSP tasks is a competent solution to

this problem. FPGA are often used as co- processors to perform all the high speed tasks that cannot be achieved by microprocessor. FPGA are chosen because they are on-site programmable and are highly suitable for hardware implementation. The software solutions adapted by the microprocessor to implement trigonometric functions are computer intensive. They do not suit hardware platforms because they need complex circuits to perform the mathematical operations. Hence, hardware algorithm is adopted for the calculation of trigonometric functions. Many hardware efficient algorithms exist, but these are not generally not well known due to the dominance of special software systems over the past quarter century.

Online CORDIC was developed by Ercegovac et al. for applications where input bits became available serially. Their method could also compensate for the Value of K online [13]. For applications that require increased throughput, pipelined CORDIC can be useful [14]. Pipelined CORDIC processors are used by performing more than one CORDIC iteration in a single clock cycle or in a single pipeline clock cycle, respectively. After an initial start up period, it allows a rotation to be completed every cycle, but involves heavy duplication of hardware in each pipeline stage which is wasteful of power and area [8]. In addition, the iteration count remains unchanged.

Several methods have focused on the problem of efficiently compensating for the scale factor. The scale factor can be compensated for in parallel, while the CORDIC iterations are being executed [15]. Another method is to perform additional scaling iterations which force the overall scaling factor to unity. Yet another method is repeat some of the CORDIC iterations so as to force K to be power of the machine radix, requiring only a simple shift operation at the end to get the scaled results. The scheme listed above may also be combined together to improve the performance.

There have been several attempts at trying to reduce the latency of CORDIC operations. Some have tried to use a high radix number system to perform the computations. In this case, less iteration are required to achieve a given precision at the expense of a more complex selection function as well as the cost of radix conversion. Another method by J.Arbaugh uses

ROM lookup tables to speed up the first third of the CORDIC iterations. Phatak proposed to execute two iterations in the same cycle, using dual CORDIC units. Still another method has been to use redundant arithmetic in CORDIC, which allows fast redundant adders to be used, to reduce cycle time. There has been a considerable amount of work devoted to solving problems such as sign detection that are associated with using redundant arithmetic. In contrast, there has been very little research performed on investigating the reduction in latency by skipping over some iteration. This is mainly because of the attendant incontinence of having a variable scaling factor and the need to store these in a ROM [8]. However with there being no shortage of available gates in modern chips, a ROM space is much more readily available for use that is used to be. Accordingly it makes eminent some to examine methods which may incur a variable K by jumping over iterations, if in doing so, so they reduce the number of iterations considerably. In year 2010, there have been two methods thus far which have attempted to attack the problem in this manner. Control CORDIC uses damping techniques from control theory to reduce the number of iterations. The method of Angle recoding can achieve a 50% or more reduction in the iteration count, but it is confined to static applications such as the Chirp-Z transform where the rotation angle is static.

CHAPTER 3

CORDIC ALGORITHM

CORDIC is an acronym that stands for COordinate Rotation DIgital Computer. It was proposed by Jack E. Volder in 1959 [1]. It is used for the calculation of trigonometric function, logarithmic, exponential, hyperbolic function, square root, multiplication, division, discrete Fourier transform(DFT) etc. The CORDIC algorithm provides an iterative method of performing vector rotation by arbitrary angle using shift and adds [3]. In this chapter, it is described that how CORDIC algorithm works.

3.1 Basic equation of CORDIC algorithm

Volder's algorithm [3] is derived from the general equation for a vector rotation. If a vector V with co-ordinates (x, y) is rotated through an angle ϕ then a new vector V' can be obtained with co-ordinates (x', y') where x' and y' can be obtained using x, y and ϕ by the following method.

$$X = r\cos\theta\,,\; Y = r\sin\theta \tag{3.1}$$

$$V' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x\cos\phi - y\sin\phi \\ y\cos\phi + x\sin\phi \end{bmatrix} \tag{3.2}$$

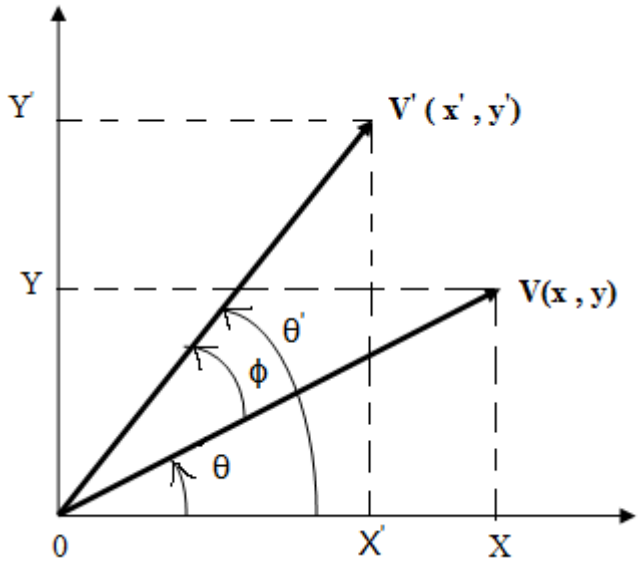


Figure 3.1: Rotation of a Vector V by the angle ϕ [3]

Let's find how equation (3.1) and (3.2) came into picture. As shown in the figure 3.1, a vector $V(x, y)$ can be restricted in two parts along the x-axis and y-axis as $r\cos\theta$ and

$r\sin\theta$ respectively. Figure 3.2, illustrates the rotation of a vector $V = \begin{bmatrix} x \\ y \end{bmatrix}$ by the angle θ.

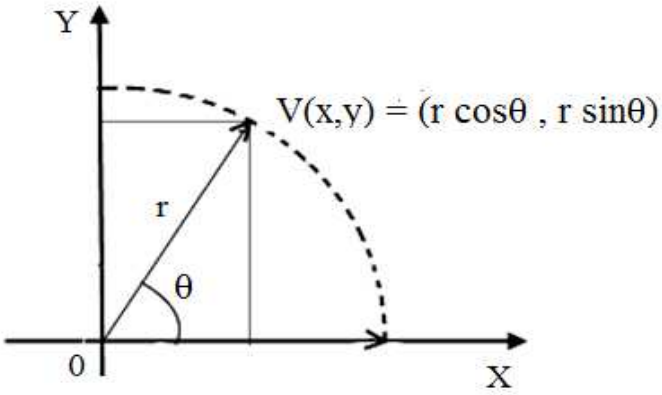


Figure 3.2: Vector V with magnitude r and phase θ

i.e. $$x = r\cos\theta$$

$$y = r\sin\theta \tag{3.3}$$

Similarly from figure 3.1 it can be seen that vector V and V' can be resolved in two parts. Let V has its magnitude and phase as r and θ respectively and V' has its magnitude and phase as r and θ' where V' came into picture after anticlockwise rotation by an angle ϕ. From figure 3.1, it can be observed

$$\theta' - \theta = \phi \tag{3.4}$$

i.e. $$\theta' = \theta + \phi \tag{3.5}$$

$$\begin{aligned} OX' = x' &= r\cos\theta' \\ &= r\cos(\theta+\phi) \\ &= r(\cos\theta.\cos\phi - \sin\theta.\sin\phi) \\ &= (r\cos\theta).\cos\phi - (r\sin\theta).\sin\phi \end{aligned} \tag{3.6}$$

Using figure 3.2 and equation 3.3, OX' can be represented as

$$OX' = x' = x\cos\phi - y\sin\phi \tag{3.7}$$

Similarly,

$$\begin{aligned} OY' = y' &= r\sin\theta' \\ &= r\sin(\theta+\phi) \\ &= r(\sin\theta.\cos\phi + \cos\theta.\sin\phi) \\ &= (r\sin\theta).\cos\phi + (r\cos\theta).\sin\phi \end{aligned}$$

$$= y\cos\phi + x\sin\phi \tag{3.8}$$

V' after anticlockwise rotation of vector V by angle ϕ is

$$x' = x\cos\phi - y\sin\phi$$

$$y' = y\cos\phi + x\sin\phi$$

Similarly, value for vector V' in the clockwise direction rotating the vector V by the angle ϕ is

$$x' = x\cos\phi + y\sin\phi \tag{3.9}$$

$$y' = y\cos\phi - x\sin\phi \tag{3.10}$$

The equation (3.7), (3.8), (3.9), (3.10) can be represented in the matrix form as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\phi & \mp\sin\phi \\ \pm\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.11}$$

V' after anticlockwise rotation of vector V by angle ϕ is

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

It is well known [4] that the rotation matrix

$$R(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

will rotate a vector $\begin{bmatrix} x \\ y \end{bmatrix}$, anticlockwise by radians in two dimensional spaces. If this rotation matrix is applied to the initial vector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, the result will be vector co-ordinates of $\begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix}$. It is easily seen that CORDIC method could be applied to calculate the functions $\sin(\phi)$ and $\cos(\phi)$ by applying successive rotations to the initial vector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

A problem arises [4] when an attempt is made to set up the rotation matrix such that all the rotations are accomplished by right shifts. Notice that if ϕ is chosen such that $\cos(\phi) = 2^{-i}$, the $\sin(\phi)$ is not necessarily a power of 2; it is not possible to choose the successive angle rotation ϕ, such that both the $\cos(\phi)$ and $\sin(\phi)$ amount to right shifts.

In working around this problem, it is possible to modify the rotation matrix by bringing a $\cos(\phi)$ term out of the matrix. Then

$$R(\phi) = \cos(\phi)\begin{bmatrix} 1 & -\tan(\phi) \\ \tan(\phi) & 1 \end{bmatrix}$$

Volder's observed that by factoring out a $\cos(\phi)$ from both sides of equation (3.7), (3.8), resulting equation be in terms of the tangent of the angle ϕ, the angle of which we want to find $\sin\theta$ and $\cos\theta$. Next if it is assumed that the angle ϕ is being an aggregate of small angles and composite angles is chosen such that their tangent are all inverse power of two, then this equation [6] can be rewritten as an iterative formula.

$$x' = \cos\phi(x - y.\tan\phi) \tag{3.12}$$

$$y' = \cos\phi(y + x.\tan\phi) \tag{3.13}$$

$z = z \pm \phi$, here ϕ is the angle of rotation ($\pm$ sign is showing the direction of rotation) and z is the angle accumulator.

The multiplication by the tangent term can be avoided if $\tan(\phi) = 2^{-i}$. In digital hardware, this denotes a simple shift operation. Furthermore, if those rotation are performed iteratively and in both directions every value of $\tan(\phi)$ is representable. With $\phi = \arctan(2^{-i})$ the cosine term could be simplified and since $\cos(\phi) = \cos(-\phi)$ is constant for a fixed number of iterations.

$$R(\phi) = \cos(\phi_i)\begin{bmatrix} 1 & -2^{-i} \\ 2^{-i} & 1 \end{bmatrix} \tag{3.14}$$

Equation (3.11) can be expressed as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \cos(\phi_i)\begin{bmatrix} 1 & -2^{-i} \\ 2^{-i} & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} \tag{3.15}$$

ϕ_i may be positive or negative depending upon whether the rotation is anticlockwise or clockwise. Equation (3.11) can be expressed as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \cos(\phi_i)\begin{bmatrix} 1 & -d_i 2^{-i} \\ d_i 2^{-i} & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} \tag{3.16}$$

This iterative rotation [6] can now be expressed as:

$$x_{i+1} = k_i(x_i - y_i d_i 2^{-i}) \tag{3.17}$$

$$y_{i+1} = k_i(y_i + x_i d_i 2^{-i}) \tag{3.18}$$

Where i denote the number of rotation required reaching the required angle of the required vector, $k_i = \cos(\arctan(2^{-i}))$ and $d_i = \pm 1$ the product of the k_i represents K-factor.

$$k = \prod_{i=0}^{n-1} k_i$$

Where $\prod_{i=0}^{n-1} k_i = \cos\phi_0.\cos\phi_1.\cos\phi_2.\cos\phi_3.....\cos\phi_{n-1}$ (ϕ is the angle of rotation here for n times rotation). The above rotation requirement and adding and subtracting of the different ϕ can be understood by the following example of a balance.

k_i is the CORDIC gain. For 8-bit hardware CORDIC approximation method, the value of k_i as

$$k_i = \prod_{i=0}^{7} \cos\phi_i = \cos\phi_0.\cos\phi_1.\cos\phi_2.\cos\phi_3.\cos\phi_4.\cos\phi_5.\cos\phi_6.\cos\phi_7$$

$$= \cos 45^o.\cos 26.565^o.\cos 14.036^o.....\cos 0.4469^o = 0.6073$$

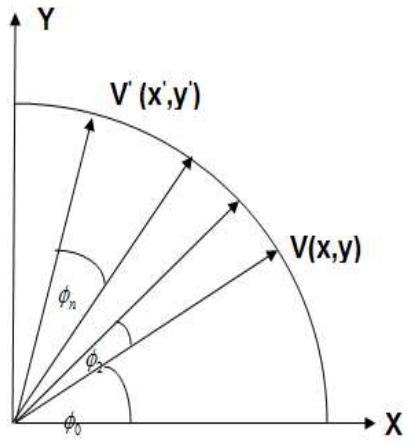


Figure 3.3: Chaining multiple micro-rotations together [8]

As shown in fig 3.3, multiple micro rotations may be chained together as the vector moves in discrete angular steps $(\phi_0, \phi_1, \phi_2, ..., \phi_n)$ from its initial position of $V(x, y)$ towards its final target position of $V'(x', y')$.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \cos\phi_o.\cos\phi_1.\cos\phi_2.....\cos\phi_n \begin{bmatrix} 1 & -\tan\phi_0 \\ \tan\phi_0 & 1 \end{bmatrix} ..... \begin{bmatrix} 1 & -\tan\phi_n \\ \tan\phi_n & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.19}$$

For 8 bit CORDIC hardware, cosine and sine of angle ϕ can be represented in matrix form by using equation (3.19)

$$\begin{bmatrix}\cos\phi\\ \sin\phi\end{bmatrix}=\begin{bmatrix}1 & -\tan\phi_0\\ \tan\phi_0 & 1\end{bmatrix}.....\begin{bmatrix}1 & -\tan\phi_7\\ \tan\phi_7 & 1\end{bmatrix}\begin{bmatrix}0.6073\\ 0\end{bmatrix} \quad (3.20)$$

For 16 bit CORDIC hardware, the value of k_i is

$$k_i=\prod_{i=0}^{15}\cos\phi_i=\cos\phi_0.\cos\phi_1.\cos\phi_2.....\cos\phi_{15}=0.6073$$

For 24 bit CORDIC hardware, the value of k_i is

$$k_i=\prod_{i=0}^{23}\cos\phi_i=\cos\phi_0.\cos\phi_1.\cos\phi_2....\cos\phi_{23}=0.6073$$

For 32 bit CORDIC hardware, the value of k_i is

$$k_i=\prod_{i=0}^{31}\cos\phi_i=\cos\phi_0.\cos\phi_1.\cos\phi_2.....\cos\phi_{31}=0.6073$$

CORDIC gain k_i is same for all the CORDIC hardware.

Table 3.1: For 8, 16, 24, 32 bit CORDIC hardware [8]

i	$2^{-i} = \tan\phi_i$	$\phi_i = \tan^{-1}(2^{-i})$	ϕ_i in radians
0	1	45^o	0.7854
1	0.5	26.565^o	0.4636
2	0.25	14.036^o	0.2450
3	0.125	7.125^o	0.1244
4	0.0625	3.576^o	0.0624
5	0.03125	1.7876^o	0.0312
6	0.015625	0.8938^o	0.0156
7	0.0078125	0.4469^o	0.0078
8	0.00390625	0.2238^o	0.0039
9	0.001953125	0.1119^o	0.0019
10	0.0009765625	0.05595^o	0.00098
11	0.00048828125	0.02798^o	0.00049
12	0.000244140625	0.01399^o	0.00024
13	0.0001220703125	0.00699^o	0.00012
14	0.00006103515625	0.00349^o	0.000061
15	0.00003051757813	0.00175^o	0.000031
16	0.00001525878906	0.000874^o	0.0000152
17	0.000007629394531	0.000437^o	0.0000076
18	0.000003814697266	0.0002186^o	0.0000038
19	0.000001907348633	0.0001093^o	0.0000019
20	0.0000009536743164	0.00005464^o	0.00000095
21	0.0000004768371582	0.0000273^o	0.00000048
22	0.0000002384185791	0.00001366^o	0.00000024
23	0.0000001192092896	0.00000683^o	0.00000012
24	0.00000005960464478	0.000003415^o	0.0000000595
25	0.00000002980232239	0.000001708^o	0.0000000298
26	0.00000001490116119	0.000000854^o	0.0000000149
27	0.000000007450580597	0.000000427^o	0.00000000745
28	0.000000003725290298	0.000000213^o	0.00000000371
29	0.000000001862645149	0.000000107^o	0.00000000187
30	0.0000000009313225746	0.0000000534^o	0.00000000093
31	0.0000000004656612873	0.0000000267^o	0.00000000047

Table 3.2: CORDIC gain k_i for 8, 16, 24, 32 bit hardware [8]

i	$2^{-i} = \tan\phi_i$	$\phi_i = \tan^{-1} 2^{-i}$	$\cos\phi_i$
0	1	45^o	0.7071
1	0.5	26.565^o	0.8944
2	0.25	14.036^o	0.9701
3	0.125	7.125^o	0.9922
4	0.0625	3.576^o	0.9980
5	0.03125	1.7876^o	0.9995
6	0.015625	0.8938^o	0.9998
7	0.0078125	0.4469^o	0.9999
8	0.00390625	0.2238^o	0.9999
9	0.001953125	0.1119^o	0.9999
10	0.0009765625	0.05595^o	0.9999
11	0.00048828125	0.02798^o	0.9999
12	0.000244140625	0.01399^o	0.9999
13	0.0001220703125	0.00699^o	0.9999
14	0.00006103515625	0.00349^o	0.9999
15	0.00003051757813	0.00175^o	0.9999
16	0.00001525878906	0.000874^o	0.9999
17	0.000007629394531	0.000437^o	1
18	0.000003814697266	0.0002186^o	1
19	0.000001907348633	0.0001093^o	1
20	0.0000009536743164	0.00005464^o	1
21	0.0000004768371582	0.0000273^o	1
22	0.0000002384185791	0.00001366^o	1
23	0.0000001192092896	0.00000683^o	1
24	0.00000005960464478	0.000003415^o	1
25	0.00000002980232239	0.000001708^o	1
26	0.00000001490116119	0.000000854^o	1
27	0.000000007450580597	0.000000427^o	1
28	0.000000003725290298	0.000000213^o	1
29	0.000000001862645149	0.000000107^o	1
30	0.000000009313225746	0.0000000534^o	1
31	0.000000004656612873	0.0000000267^o	1

From the above table it can be seen that precision up to 0.4469^{o} is possible for 8-bit CORDIC hardware. These ϕ_i are stored in ROM of the hardware of the CORDIC hardware as the look up table.

From equation (3.17), (3.18), k_i is present and an additional multiplier is required .But CORDIC works on simple shift and addition operation.

Pseudo rotation: - The pseudo rotation [3] is introduced by dropping cosine term. The pseudo rotation is introduced by performing givens rotation: the angle of rotation is correct but x and y values are scaled by $\cos(\phi)$. Note that it can make the computation of plane rotations more amenable to simple operations.

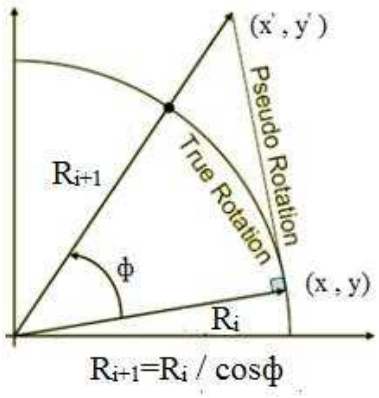


Figure 3.4: Pseudo rotation [3]

Dropping cosine term gives:

$$x' = x - y.\tan\phi \tag{3.21}$$

$$y' = y + x.\tan\phi \tag{3.22}$$

Consequence is pseudo rotation rather true rotation. Note scaling of original vector.

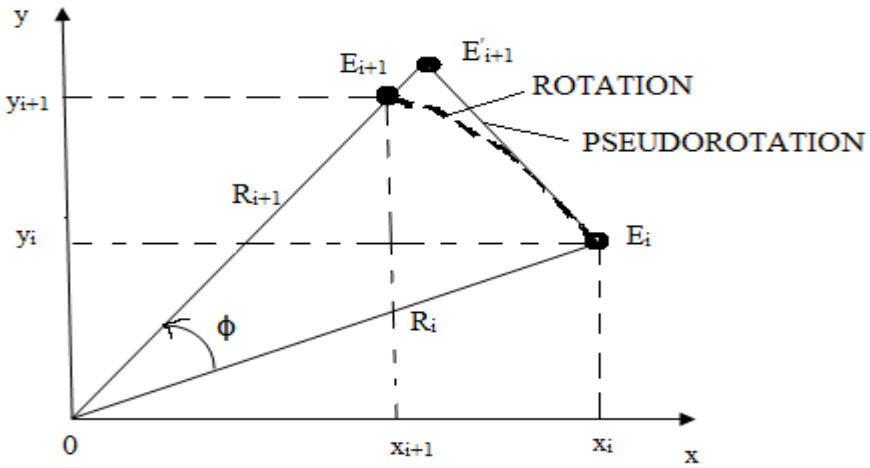


Figure 3.5: Pseudo rotation [3]

Rotate the vector OE_i with end point at (x_i, y_i) by ϕ

$$x_{i+1} = x_i.\cos\phi_i - y_i.\sin\phi_i$$

$$= \cos\phi_i (x_i - y_i.\tan\phi_i)$$

$$= \frac{(x_i - y_i \tan\phi_i)}{(1+\tan^2\phi_i)^{1/2}}$$

$$y_{i+1} = y_i.\cos\phi_i + x_i.\sin\phi_i$$

$$= \cos\phi_i (y_i + x_i.\tan\phi_i)$$

$$= \frac{(y_i + x_i.\tan\phi_i)}{(1+\tan^2\phi_i)^{1/2}}$$

Goal: eliminate the division by $(1+\tan^2\phi)^{1/2}$ and choose ϕ_i so that $\tan\phi_i$ is power of 2. Whereas a real rotation does not change the length R_i of the vector, a pseudo rotation step increases its length to:

From figure 3.5, it is seen that

$$\frac{R_i}{R_{i+1}} = \cos\phi_i$$

$$R_{i+1} = \frac{R_i}{\cos\phi_i}$$

$$R_{i+1} = R_i(1+\tan^2\phi)^{1/2}$$

The coordinates of the new end point E'_{i+1} after pseudo rotation is derived by multiplying the coordinates of E_{i+1} by the expansion factor.

$$x_{i+1} = x_i - y_i \tan\phi_i \quad (3.23)$$

$$y_{i+1} = y_i + x_i \tan\phi_i \quad (3.24)$$

Original given rotation now reduced to iterative shift add algorithm.

$$x_{i+1} = x_i - d_i y_i 2^{-i} \quad (3.25)$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \quad (3.26)$$

$$z_{i+1} = z_i - d_i \phi_i \quad (3.27)$$

$d_i = \pm 1$ (d_i is the direction of angle of rotation)

Each iteration requires :- (1) 2 shifts

(2) 1 look-up table

(3) 3 additions/subtractions

Scaling factor: - When simplifying the algorithm [3][6][15] to allow pseudo-rotations the cosine term was omitted. Applying to this iterative scheme, the scaling factor K_n can be written as

$$K_n = \prod_{i=0}^{n-1}\left(\frac{1}{\cos\phi_i}\right) = \prod_{i=0}^{n-1}\sqrt{1+\tan^2\phi_i} = \prod_{i=0}^{n-1}\sqrt{1+2^{-2i}} = 1.6467$$

As iteration number n is known, K_n can be pre-computed.

Multiplying equation (3.23), (3.24) by $\cos\phi_i$, we get

$$\cos\phi_i . x_{i+1} = \cos\phi_i (x_i - y_i \tan\phi_i)$$

$$\cos\phi_i . y_{i+1} = \cos\phi_i (y_i + x_i \tan\phi_i)$$

or

$$\cos\phi_i . x_{i+1} = (x_i \cos\phi_i - y_i \sin\phi_i)$$

$$\cos\phi_i . y_{i+1} = (y_i \cos\phi_i + x_i \sin\phi_i)$$

After n iterations, we have

$$x_n = K_n (x_0 . \cos\phi_i - y_0 . \sin\phi_i) \qquad (3.28)$$

$$y_n = K_n (y_0 . \cos\phi_i + x_0 . \sin\phi_i) \qquad (3.29)$$

To find $\cos\phi_i$ and $\sin\phi_i$:- $x_0 = 1, y_0 = 0$

$$\frac{x_n}{K_n} = \cos\phi_i \qquad \text{and} \qquad \frac{y_n}{K_n} = \sin\phi_i$$

or

$$x_{last} \times k_i = \cos\phi_i \qquad \text{and} \qquad y_{last} \times k_i = \sin\phi_i$$

By taking an example of balance it can be understood that how the CORDIC algorithm works.

In the figure (3.6), first of all, keep the angle θ on the left pan of balance and if the balance rotates around the anticlockwise direction then add the highest value in the table at the other side.

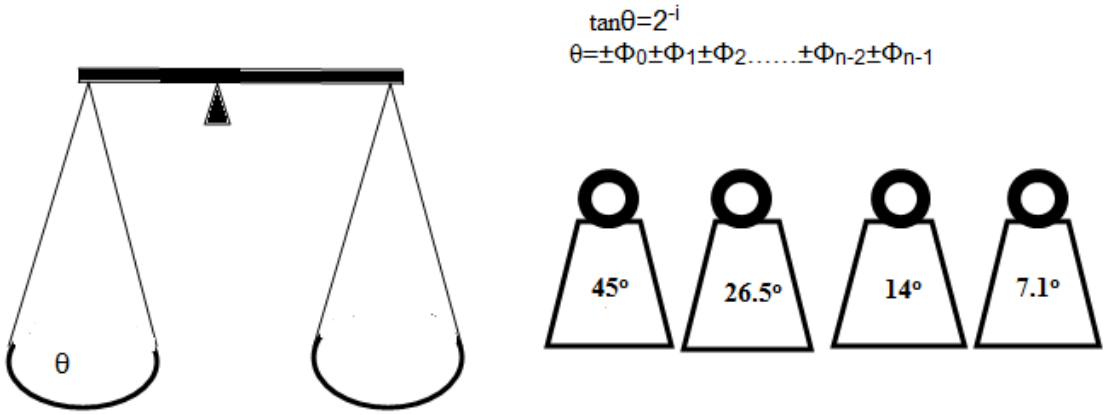


Figure 3.6: A balance having θ at one side and small weights (angle) at the other side.

If the balance show a left inclination as in figure 3.7(a) then other weights are required to add in the right pan or in the term of angle if θ is greater than total ϕ_i then add other weights to reach as near to the θ as possible but in other hand if the balance show a right inclination as in figure 3.7(b) then a weight required to be removed from the right pan or in the term of angle if θ is less than total ϕ_i then we subtract other weights this process is repeated to reach as near to the θ as possible.

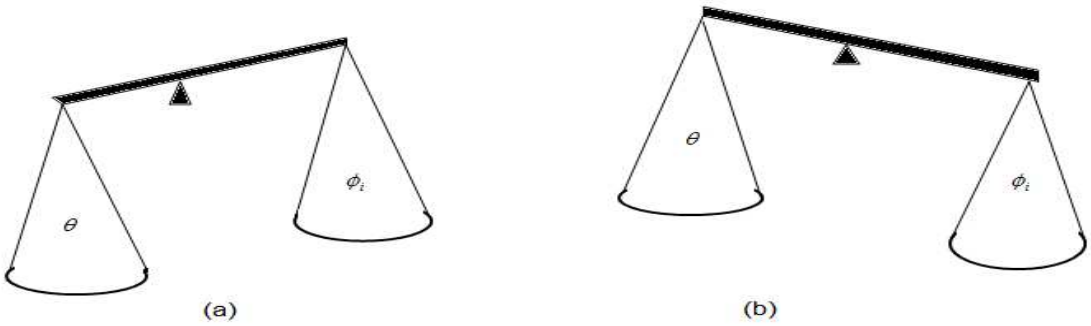


Figure 3.7: Inclined balance due to difference in weight of two sides.

For example: - $\theta = 30.0^o$

Start with $\phi_0 = 45.0^o$ $(45^o > 30.0^0)$

$45^o - 26.565^o = 18.44^o$ $(18.44^o < 30.0^o)$

$18.44^o + 14.036^o = 32.47^o$ $(32.47^o > 30.0^o)$

$32.47^o - 7.125^o = 25.346^o$ $(25.546^o < 30.0^o)$

$25.346^o + 3.576^o = 28.913^o$ $(28.913^o < 30.0^o)$

$28.913^o + 1.7876^o = 30.7^o$ $(30.7^o > 30.0^o)$

$30.7^o - 0.89^o = 29.8^o$ $(29.8^o < 30.0^o)$

$29.8^o + 0.44^o = 30.24^o$

$\theta = 30.0^o$

$= 45.0^o - 26.565^o + 14.036^o - 7.125^o + 3.576^o + 1.7876^o - 0.89^o + 0.44^o$

$= 30.2^o$

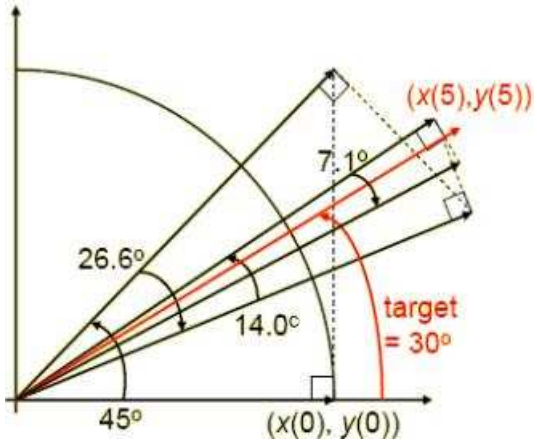


Figure 3.8: Angle rotation

3.2 Basic CORDIC iterations

The basic CORDIC iteration equations [6] are

$$x_{i+1} = x_i - d_i y_i 2^{-i} \tag{3.30}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \tag{3.31}$$

$$z_{i+1} = z_i - d_i \phi_i \tag{3.32}$$

$d_i = \pm 1$ (d_i is the direction of angle of rotation)

The computation of x_{i+1} and y_{i+1} requires an i-bit right shift and an add/subtract. If the function $\phi_i = \tan^{-1} 2^{-i}$ is pre-computed and stored in table (Table 3.1) for different value of i, a single add/subtract suffices to compute z_{i+1}. Each CORDIC iteration thus involves two shifts, a table lookup and three additions/subtractions.

If the rotation is done by the same set of angles (with + or – signs), then expansion factor or scaling factor K_n is constant and can be pre-computed.

The CORDIC method can be employed in two different modes:-

- Rotation mode
- Vectoring mode

Rotation mode: - In the rotation mode [6][16], the co-ordinate components of a vector and an angle of rotation are given and the co-ordinate components of the original vector, after rotation through a given angle, are computed.

In the CORDIC terminology, the selection rule for d_i, which make z converge to zero, is known as rotation mode.

In the rotation mode, the angle accumulator is initialized with the desired rotation angle $(z_0 = \theta)$. The rotation decision at each iteration is made to diminish the magnitude of the residual angle in the angle accumulator. The decision at each iteration is therefore based on the sign of the residual angle after each step.

For rotation mode, the CORDIC equations are

$$x_{i+1} = x_i - d_i y_i 2^{-i} \tag{3.33}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \tag{3.34}$$

$$z_{i+1} = z_i - d_i \phi_i \tag{3.35}$$

or $z_{i+1} = z_i - d_i \tan^{-1} 2^{-i}$

$d_i = \pm 1$ (d_i is the direction of angle of rotation)

$$d_i = \begin{cases} +1, & \text{if } z_i \geq 0 \\ -1, & \text{if } z < 0 \end{cases} \tag{3.36}$$

After n iteration in CORDIC mode, z_n is sufficiently close to zero, we have

$$z = \sum \phi_i \tag{3.37}$$

The output of the rotation mode x_n, y_n and z_n are given by the following expressions, x_n and y_n being the co-ordinates of the rotated (by the angle ϕ) vector.

$$x_n = K_n (x_0 . \cos z_0 - y_0 . \sin z_0) \tag{3.38}$$

$$y_n = K_n (y_0 . \cos z_0 + x_0 . \sin z_0) \tag{3.39}$$

$$z_n = 0 \tag{3.40}$$

Where $K_n = \prod_{i=0}^{n-1} (1 + 2^{-2i})^{1/2} = 1.6467$

To find $\cos\theta$ and $\sin\theta$:- The elementary functions [6][16] sine and cosine can be computed using rotation mode of the CORDIC algorithm if the initial vector is of unit length and is aligned with abscissa(x-axis).

$$x_0 = 1, y_0 = 0, z_0 = \theta$$

Put the initial condition in equations (3.38), (3.39) to get $\cos\theta$ and $\sin\theta$, we get

$$\frac{x_n}{K_n} = \cos\phi_i \quad \text{and} \quad \frac{y_n}{K_n} = \sin\phi_i \tag{3.41}$$

or

$$x_{last} \times k_i = \cos\phi_i \qquad \text{and} \qquad y_{last} \times k_i = \sin\phi_i \tag{3.42}$$

Once $\sin z_0$ and $\cos z_0$ are known, $\tan z_0$ can be computed necessary through division.

For example: - rotation mode ($\theta = 30^o$)

$$z_0 = \theta = 30^o$$

Table 3.3: Choosing the sign of the rotation angles to force z to zero [6]

i	$z_i - d_i\phi_i$	z_{i+1}	d_i	$\theta = 30^o$
0	$+30.0^o - 45.0^o$	-15^o $(30.0^o > 0)$	+1	$0 + 45.0^o = 45.0^o$
1	$-15^o + 26.565^o$	$+11.6^o$ $(-15^o < 0)$	–1	$45.0^o - 26.565^o = 18.44^o$
2	$+11.6^o - 14.0^o$	-2.4^o $(11.6^o > 0)$	+1	$18.44^o + 14.036^o = 32.47^o$
3	$-2.4^o + 7.1^o$	$+4.7^o$ $(-2.4^o < 0)$	–1	$32.47^o - 7.125^o = 25.346^o$
4	$+4.7^o - 3.6^o$	$+1.1^o$ $(4.7^o > 0)$	+1	$25.346^o + 3.567^o = 28.913^o$
5	$+1.1^o - 1.8^o$	-0.7^o $(1.1^o > 0)$	+1	$28.913^o + 1.7876^o = 30.70^o$
6	$-0.7^o + 0.9^o$	$+0.2^o$ $(-0.7^o < 0)$	–1	$30.70^o - 0.89^o = 29.8^o$
7	$+0.2 - 0.4^o$	-0.2^o $(+0.2^o > 0)$	+1	$29.8^o + 0.44^o = 30.24^o$
8	$-0.2^o + 0.2^o$	0.0^o $(-0.2^o < 0)$	–1	$30.24^o - 0.2238^o = 30.1^o$

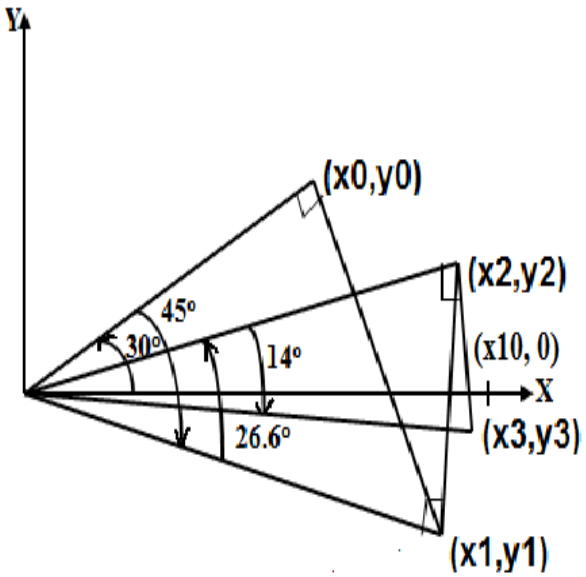


Figure 3.9: First three of 10 iteration leading from (x_0, y_0) to $(x_{10}, 0)$ in rotating by $+30^o$ by Rotation mode [17]

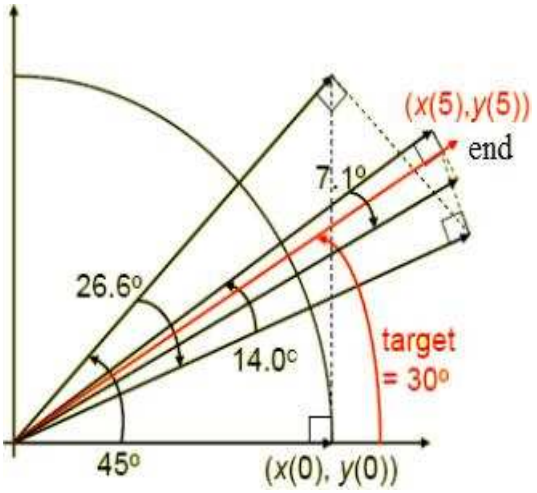


Figure 3.10: Rotation mode reached target where $z_{i+1} = 0$ [6]

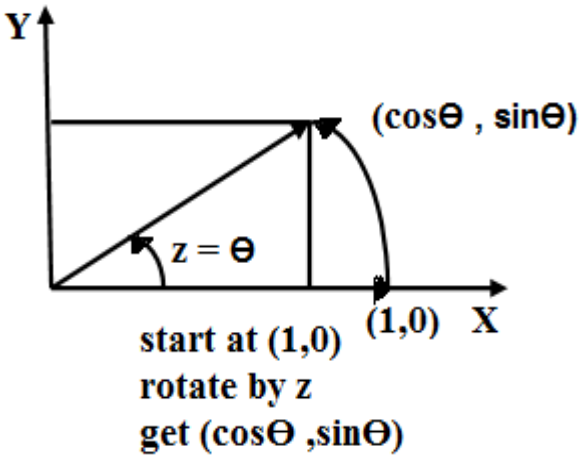


Figure 3.11: Rotation mode to compute $\sin\theta$ and $\cos\theta$

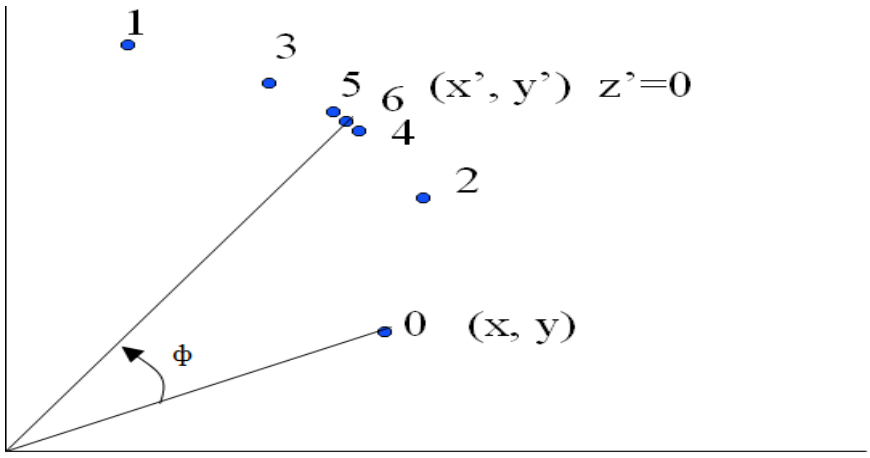


Figure 3.12: Basic rotation mode diagram

Using equations (3.33), (3.34), (3.35), we have to compute x and y

i	$z_{i+1} = z_i - d_i\phi_i$	d_i	Target ($\theta = 30^o$)	To compute x and y
0	-15^o $(30.0^o > 0)$	+1	$0 + 45.0^o = 45.0^o$	$x_1 = 1$, $y_1 = 1$
1	$+11.6^o$ $(-15^o < 0)$	–1	$45.0^o - 26.565^o = 18.44^o$	$x_2 = 1.5$, $y_2 = 0.5$
2	-2.4^o $(11.6^o > 0)$	+1	$18.44^o + 14.036^o = 32.47^o$	$x_3 = 1.375$, $y_3 = 0.875$
3	$+4.7^o$ $(-2.4^o < 0)$	–1	$32.47^o - 7.125^o = 25.346^o$	$x_4 = 1.4843$, $y_4 = 0.7032$
4	$+1.1^o$ $(4.7^o > 0)$	+1	$25.346^o + 3.567^o = 28.913^o$	$x_5 = 1.44035$, $y_5 = 0.7959$
5	-0.7^o $(1.1^o > 0)$	+1	$28.913^o + 1.7876^o = 30.70^o$	$x_6 = 1.4154$, $y_6 = 0.8409$
6	$+0.2^o$ $(-0.7^o < 0)$	–1	$30.70^o - 0.89^o = 29.8^o$	$x_7 = 1.4285$, $y_7 = 0.8187$
7	-0.2^o $(+0.2^o > 0)$	+1	$29.8^o + 0.44^o = 30.24^o$	$x_8 = 1.42210$, $y_8 = 0.8298$
8	0.0^o $(-0.2^o < 0)$	–1	$30.24^o - 0.2238^o = 30.1^o$	

After n iterations, we have

$$x_n = K_n(x_0.\cos z_0 - y_0.\sin z_0) \tag{3.43}$$

$$y_n = K_n(y_0.\cos z_0 + x_0.\sin z_0) \tag{3.44}$$

$$z_n = 0 \tag{3.45}$$

$$\text{Rule : choose } d_i \in \{-1,+1\} \text{ such that } z \to 0$$

$$\text{Where } K_n = \prod_{i=0}^{n-1}(1+2^{-2i})^{1/2} = 1.6467$$

Put $x_0 = 1, y_0 = 0, z_0 = \theta$ (using equation (3.43), (3.44))

$$\frac{x_n}{K_n} = \cos\phi_i \quad \text{and} \quad \frac{y_n}{K_n} = \sin\phi_i$$

$$x_{last} \times k_i = \cos\phi_i \quad \text{and} \quad y_{last} \times k_i = \sin\phi_i$$

$$x_{last} = 1.42210,\ y_{last} = 0.8298$$

$$\cos\theta = \frac{1.42210}{1.6467} \Rightarrow \cos\theta = 0.8636$$

$$\sin\theta = \frac{0.8298}{1.6467} \Rightarrow \sin\theta = 0.50$$

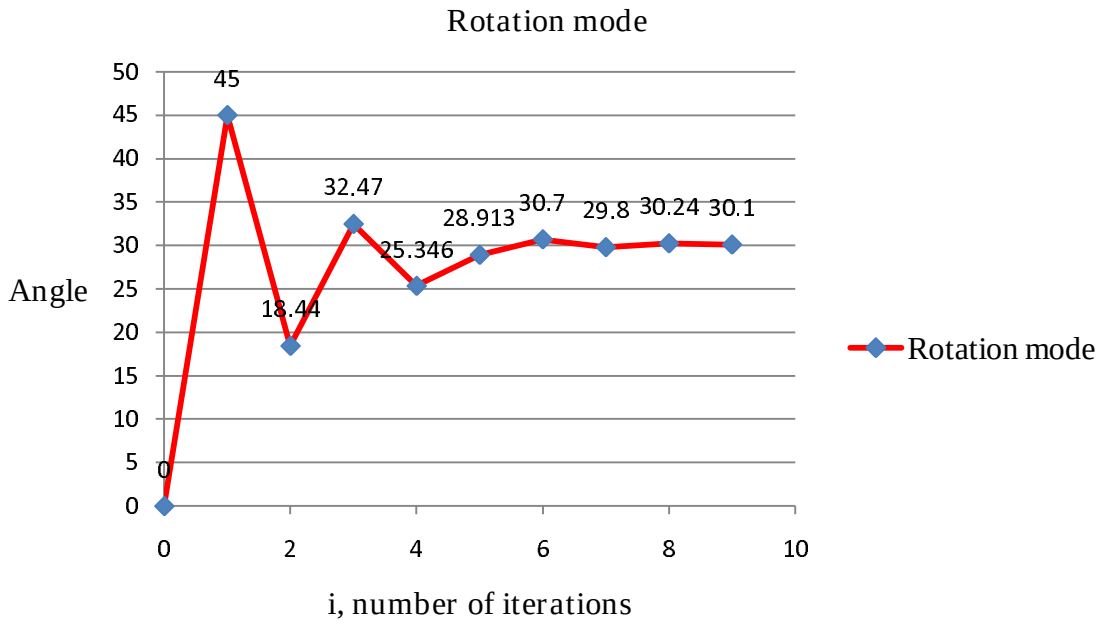


Figure 3.13: Rotation mode Graph

Vectoring mode: - In vectoring mode [6][16], the co- ordinates components of a vector are given and the magnitude and angular argument of the original vector are computed. In the CORDIC terminology, the selection rule for d_i, which make y converge to zero, is known as vectoring mode. It can be used to compute $\tan^{-1} y$, $\sin^{-1} y$ and $\cos^{-1} y$.

In the vectoring mode, the CORDIC rotator rotates the input vector through whatever angle is necessary to align the result vector with the x-axis. The result of the vectoring operation is a rotation angle and the scaled magnitude of the original vector (the x component of the result). The vectoring function works by seeking to minimize the y component of the residual vector at each rotation. The sign of the residual y component is used to determine which direction to rotate next. If the angle accumulator is initialized with zero, it will contain the transverse angle at the end of the iteration.

In vectoring mode, the CORDIC equations are

$$x_{i+1} = x_i - d_i y_i 2^{-i} \quad (3.46)$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \quad (3.47)$$

$$z_{i+1} = z_i - d_i \phi_i \quad (3.48)$$

or $z_{i+1} = z_i - d_i \tan^{-1} 2^{-i}$

$d_i = \pm 1$ (d_i is the direction of angle of rotation)

$$d_i = \begin{cases} +1, \text{ if } y_i < 0 \\ -1, \text{ if } y_i \geq 0 \end{cases} \tag{3.49}$$

In vectoring mode, y converges to zero.

$$0 = y + x.2^{-i}$$

$$-y = x.2^{-i}$$

$$2^{-i} = -\frac{y}{x}$$

$$\tan \sum \phi_i = -\frac{y}{x} \tag{3.50}$$

After n iteration, we have

$$x_n = K_n(x.\cos(\sum \phi_i) - y.\sin(\sum \phi_i))$$

$$x_n = K_n \cos(\sum \phi_i).(x - y.\tan(\sum \phi_i))$$

$$x_n = \frac{K_n(x - y.\tan(\sum \phi_i))}{(1+\tan^2(\sum \phi_i))^{1/2}}$$

$$x_n = \frac{K_n\left(x + \frac{y^2}{x}\right)}{\left(1+\frac{y^2}{x^2}\right)^{1/2}}$$

$$x_n = K_n(x^2 + y^2)^{1/2}$$

$$z_n = z - \tan^{-1}\left(-\frac{y}{x}\right)$$

or $$z_n = z + \tan^{-1}\left(\frac{y}{x}\right)$$

$$y_n = 0$$

The CORDIC equations has become

$$x_n = K_n(x^2 + y^2)^{1/2} \tag{3.51}$$

$$y_n = 0 \tag{3.52}$$

$$z_n = z + \tan^{-1}\left(\frac{y}{x}\right) \tag{3.53}$$

$$\text{Rule}: \text{choose } d_i \in \{-1,+1\} \text{such that } y \to 0$$

One can compute $\tan^{-1} y$ in vectoring mode by starting with $x = 1$ and $z = 0$.

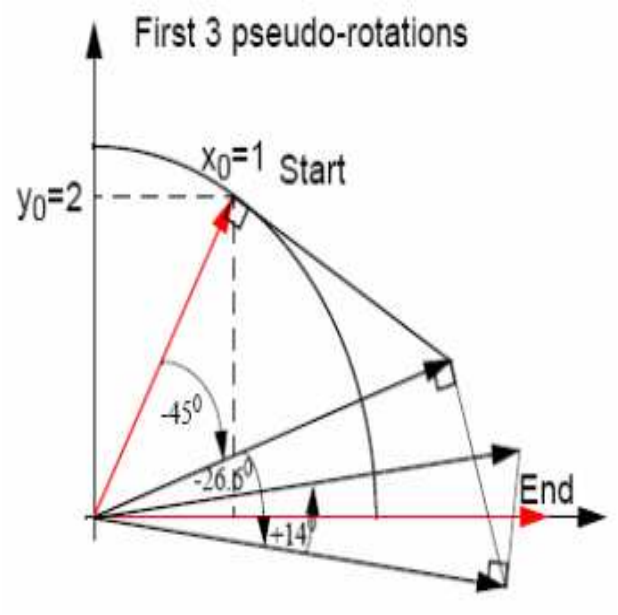


Figure 3.14: Vectoring mode y converge to zero [6]

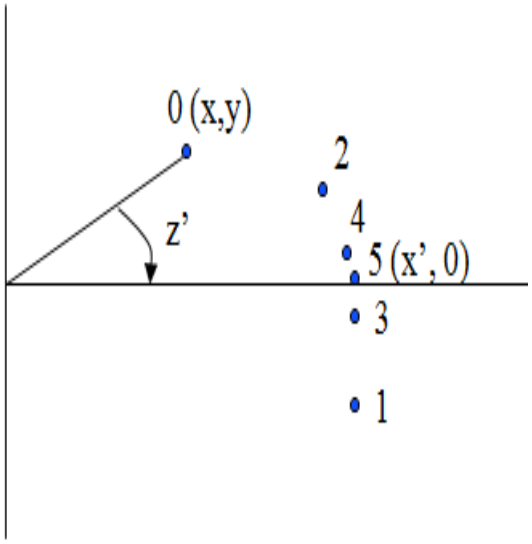


Figure 3.15: Basic vectoring mode diagram

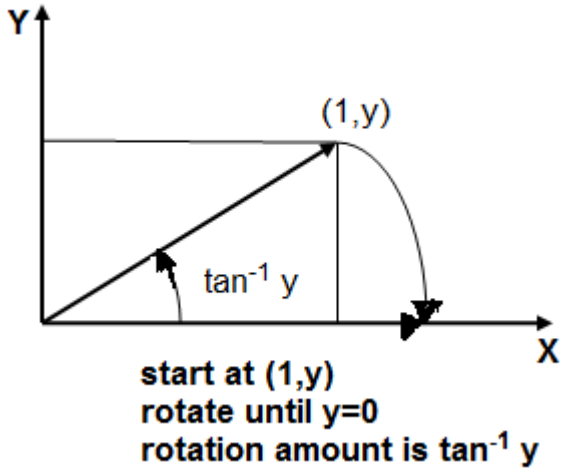


Figure 3.16: Vectoring mode to compute $\tan^{-1} y$

How to compute $\tan^{-1} y$: - angle accumulator is initialised to zero $(z_0 = 0)$.

For example: - to find angle, $x_{in} = 1.732$ and $y_{in} = 1$ are given

i	y_i	d_i	x_i	z_{i+1}
	1		1.732	0.0^o
0	– 0.732	–1 (1 > 0)	2.732	$0.0^o + 45^o = 45^o$
1	0.634	+1 (–0.732 <0)	3.098	$45^o – 26.565^o = 18.345^o$
2	–0.1405	–1 (0.634 >0)	3.2565	$18.345^o + 14.036^o = 32.471^o$
3	0.2665	+1 (–0.1405 <0)	3.27406	$32.471^o – 7.125^o = 25.346^o$
4	0.0635	–1 (0.2665 >0)	3.2907	$25.346^o + 3.125^o = 28.922^o$
5	–0.0393	–1 (0.0635 >0)	3.2926	$28.922^o + 1.7876^o = 30.70^o$
6	0.0121	+1 (–0.0393 <0)	3.2932	$30.70^o – 0.8938^o = 29.80^o$
7	–0.013	–1 (0.0121 >0)	3.2932	$29.8^o + 0.4469^o = 30.249^o$

Magnitude:- $x_R = \sqrt{x^2 + y^2}$

$$x_R = \sqrt{(1.732)^2 + (1)^2} = 1.9999$$

$$y_R = 0$$

From equation (3.43), we get

$$x_n = K_n (x_0 \cos z_0 - y_0 \sin z_0)$$

Put $x_0 = 1, y_0 = 0, z_0 = \theta$

$$\frac{x_n}{K_n} = \cos\theta$$

$$\cos\theta = \frac{3.2932}{1.6467} = 1.999$$

Therefore, $\cos\theta = x_R$

Trigonometric functions as well as exponential functions are computed in the Rotation mode. Vectoring mode is used to compute logarithm and arctangent Functions [4].

3.3 CORDIC Hardware

The basic CORDIC iterations are

$$x_{i+1} = x_i - d_i y_i 2^{-i} \tag{3.54}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \tag{3.55}$$

$$z_{i+1} = z_i - d_i \phi_i \tag{3.56}$$

or $z_{i+1} = z_i - d_i \tan^{-1} 2^{-i}$

$d_i = \pm 1$ (d_i is the direction of a ngle of rotation)

A hardware implementation for CORDIC arithmetic [11] is shown below in figure 3.17. It requires three registers for x, y, z, a look up table to store the values of $\phi_i = \tan^{-1} 2^{-i}$ and two shifter to supply the terms $2^{-i}x$ and $2^{-i}y$ to the adder/subtractor units. A Barrel shifter is a digital circuit that can shift a data word by specified numbers of bits in one cycle. It can be implemented as a sequence of multiplexers. The d_i factor (–1 and +1) is accommodated by selecting the shift operand or its complement. In the extreme, CORDIC iterations can be implemented in firmware (micro program) or software using the ALU and general purpose registers of a standard microprocessor. In this case, the look up table supplying the term ϕ_i can be stored in the control ROM or in main memory, where high speed is not required and minimizing the hardware cost is important(as in calculator), the adder in figure: 3.17 can be bit serial. Then with k operands, $O(k^2)$ clock cycle would be required to complete the k CORDIC iterations. This is acceptable for hand handled calculators.

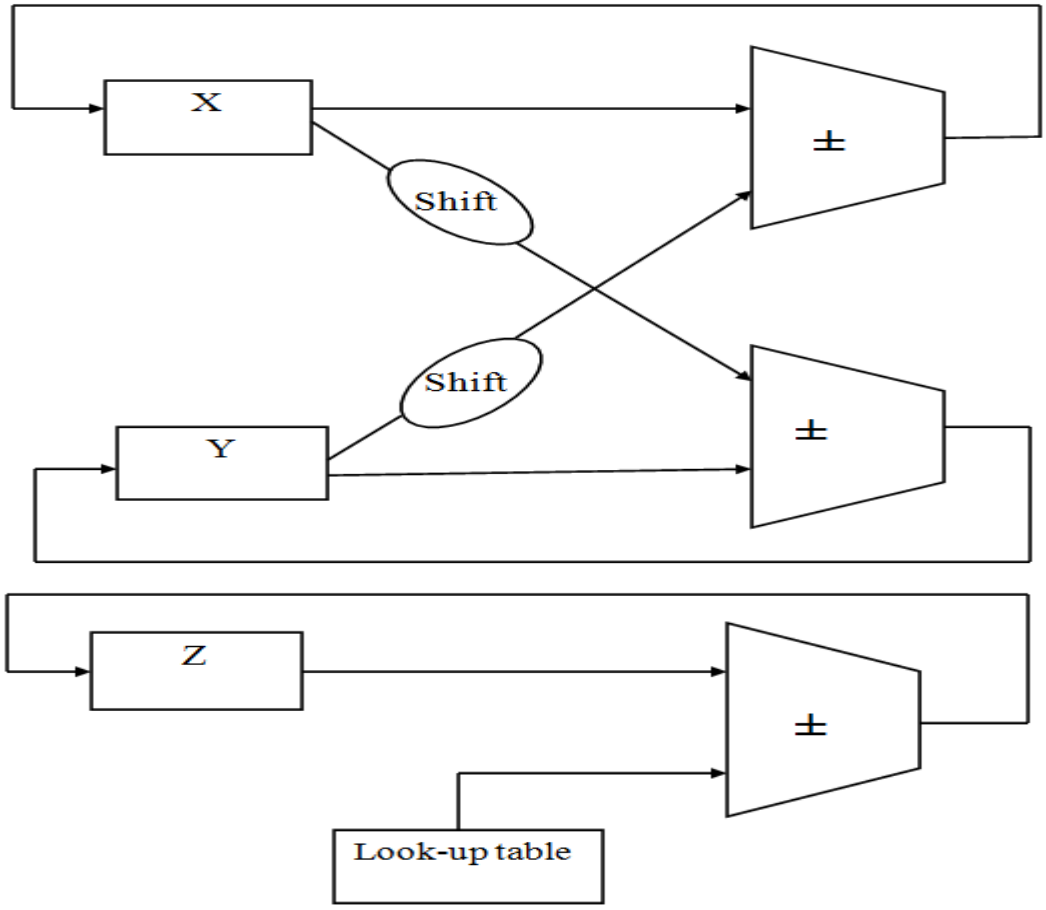


Figure 3.17: Hardware element needed for the CORDIC method [11]

3.4 Generalized CORDIC

The basic CORDIC method can be generalized [2][3][6] to provide the more powerful tool for function evaluation. Generalized CORDIC is defined as follows:

$$x_{i+1} = x_i - m.d_i y_i 2^{-i} \tag{3.57}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \tag{3.58}$$

$$z_{i+1} = z_i - d_i \phi_i \tag{3.59}$$

or $z_{i+1} = z_i - d_i \tan^{-1} 2^{-i}$

$d_i = \pm 1$ (d_i is the direction of angle of rotation)

where m is the mode of operation.

Noting that only difference with basic CORDIC is the introduction of the parameter m in the equation for x and redefinition of ϕ_i. The parameter m can assume one of three values:

$m = 1$ Circular rotation (Basic CORDIC) $\phi_i = \tan^{-1} 2^{-i}$

$m = 0$ Linear rotation $\phi_i = 2^{-i}$

$m = -1$ Hyperbolic rotation $\phi_i = \tanh^{-1} 2^{-i}$

Extension to circular function: - $m = 1$, $\phi_i = \tan^{-1} 2^{-i}$

Put $m = 1$ in equation (3.57), we get

The basic CORDIC iterations are

$$x_{i+1} = x_i - 1.d_i y_i 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

$$z_{i+1} = z_i - d_i \phi_i$$

or $$z_{i+1} = z_i - d_i \tan^{-1} 2^{-i}$$

$d_i = \pm 1$ (d_i is the direction of angle of rotation)

In Rotation mode, z converges to zero.

$$z = \sum \phi_i \tag{3.60}$$

The outputs of the rotation mode x_n, y_n and z_n are given by the following expressions, x_n and y_n being the co-ordinates of the rotated (by the angle ϕ) vector.

$$x_n = K_n (x_0 . \cos z_0 - y_0 . \sin z_0) \tag{3.61}$$

$$y_n = K_n (y_0 . \cos z_0 + x_0 . \sin z_0) \tag{3.62}$$

$$z_n = 0 \tag{3.63}$$

Rule : choose $d_i \in \{-1, +1\}$ such that $z \to 0$

Where $K_n = \prod_{i=0}^{n-1} (1 + 2^{-2i})^{1/2} = 1.6467$

In vectoring mode, y converges to zero.

$$0 = y + x.2^{-i}$$

$$-y = x.2^{-i}$$

$$2^{-i} = -\frac{y}{x}$$

$$\tan \sum \phi_i = -\frac{y}{x} \tag{3.64}$$

After n iterations, we have

$$x_n = K_n (x^2 + y^2)^{1/2} \tag{3.65}$$

$$y_n = 0 \tag{3.66}$$

$$z_n = z + \tan^{-1}\left(\frac{y}{x}\right) \tag{3.67}$$

Rule : choose $d_i \in \{-1, +1\}$ such that $y \to 0$

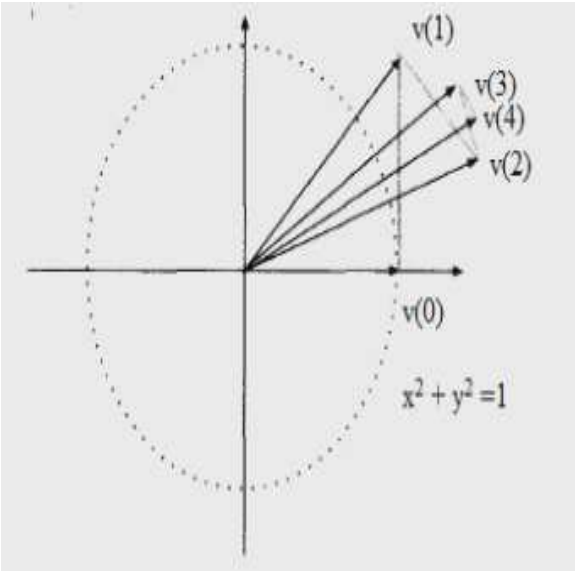


Figure 3.18: Circular rotation [11]

Extension to linear function: - $m = 0$, $\phi_i = 2^{-i}$

Put $m = 0$ in equation (3.57), we get

The basic CORDIC iterations are

$$x_{i+1} = x_i - 0.d_i y_i 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

$$z_{i+1} = z_i - d_i \phi_i$$

or $$z_{i+1} = z_i - d_i 2^{-i}$$

$d_i = \pm 1$ (d_i is the direction of angle of rotation)

In Rotation mode, z converges to zero

$$z = 2^{-i} \tag{3.68}$$

The outputs of the rotation mode x_n, y_n and z_n are given by the following expressions, x_n, y_n being the co-ordinates of the rotated (by the angle ϕ) vector.

$$x_n = x_0 \tag{3.69}$$

$$y_n = y_0 + x_0 z_0 \tag{3.70}$$

$$z_n = 0 \tag{3.71}$$

Rule : choose $d_i \in \{-1, +1\}$ such that $z \to 0$

In vectoring mode, y converges to zero.

$$0 = y + x.2^{-i}$$

$$-y = x.2^{-i}$$

$$2^{-i} = -\frac{y}{x} \tag{3.72}$$

After n iterations, we have

$$x_n = x_0 \tag{3.73}$$

$$y_n = 0 \tag{3.74}$$

$$z_n = z_0 + \frac{y_0}{x_0} \tag{3.75}$$

$$\text{Rule: choose } d_i \in \{-1, +1\} \text{ such that } y \to 0$$

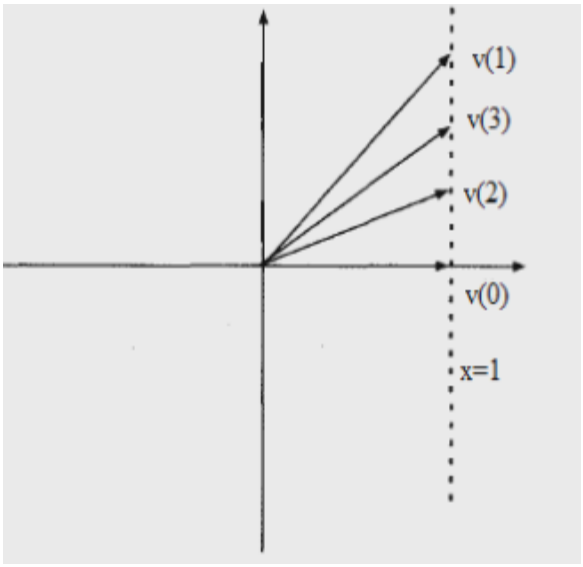


Figure 3.19: Linear Rotation [11]

Extension to Hyperbolic function:- $m = -1, \phi_i = \tanh^{-1} 2^{-i}$

Put $m = -1$ in equation (3.57), we get

The basic CORDIC iterations are

$$x_{i+1} = x_i + 1.d_i y_i 2^{-i} \tag{3.76}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \tag{3.77}$$

$$z_{i+1} = z_i - d_i \phi_i \tag{3.78}$$

or $$z_{i+1} = z_i - d_i \tanh^{-1} 2^{-i}$$

$d_i = \pm 1$ (d_i is the direction of angle of rotation)

In Rotation mode, z converge to zero

$$z = \sum \phi_i \tag{3.79}$$

The outputs of the rotation mode x_n, y_n and z_n are given by the following expressions, x_n and y_n being the co-ordinates of the rotated (by the angle ϕ) vector.

$$x_n = K_n(x_o \cosh z_0 + y_0 \sinh z_0) \tag{3.80}$$

$$y_n = K_n(y_0 \cosh z_0 + x_0 \sinh z_0) \tag{3.81}$$

$$z_n = 0 \tag{3.82}$$

$$\text{Rule : choose } d_i \in \{-1, +1\} \text{ such that } z \to 0$$

$$\text{Where } K_n = \prod_{i=0}^{n-1} (1 - 2^{-2i})^{1/2} = 0.82816$$

$$\text{Repeat as } (3^{i+2} - 1)/2$$

$$i = 1, 2, 3, ...k, (3k+1)$$

Note $\tanh^{-1} 2^0 = \infty$ (and $K_n = 0$) Necessary to begin from iteration $i = 1$

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

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

A solution: repeat some iterations [6]

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

Repeating iterations $4, 13, 40, ..., k, 3k+1, ...$ results in a convergent algorithm.

In vectoring mode, y converges to zero.

$$0 = y + x.2^{-i}$$

$$-y = x.2^{-i}$$

$$2^{-i} = -\frac{y}{x}$$

$$\tanh \sum \phi_i = -\frac{y}{x} \tag{3.83}$$

After n iteration, we have

$$x_n = K_n(x.\cosh(\sum \phi_i) + y.\sinh(\sum \phi_i))$$

$$x_n = K_n \cosh(\sum \phi_i).(x + y.\tanh(\sum \phi_i))$$

$$x_n = \frac{K_n(x + y.\tanh(\sum \phi_i))}{(1 - \tanh^2(\sum \phi_i))^{1/2}}$$

$$x_n = \frac{K_n\left(x - \frac{y^2}{x}\right)}{\left(1 - \frac{y^2}{x^2}\right)^{1/2}}$$

$$x_n = K_n(x^2 - y^2)^{1/2}$$

$$z_n = z - \tanh^{-1}\left(-\frac{y}{x}\right)$$

or $$z_n = z + \tanh^{-1}\left(\frac{y}{x}\right)$$

$$y_n = 0$$

The CORDIC equations has become

$$x_n = K_n(x^2 - y^2)^{1/2} \tag{3.84}$$

$$y_n = 0 \tag{3.85}$$

$$z_n = z + \tanh^{-1}\left(\frac{y}{x}\right) \tag{3.86}$$

$$\text{Rule: choose } d_i \in \{-1, +1\} \text{ such that } y \to 0$$

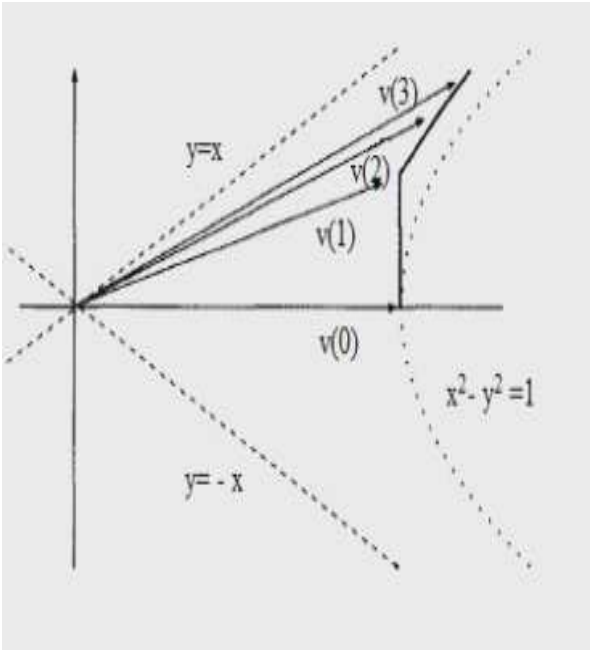
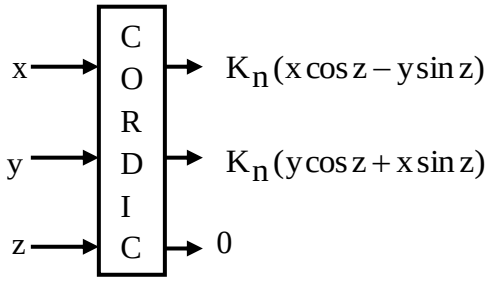
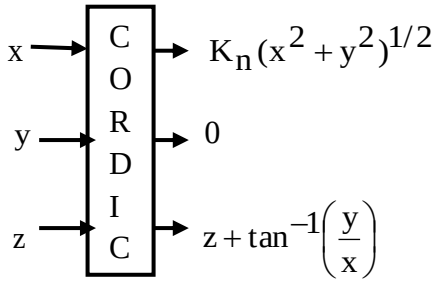
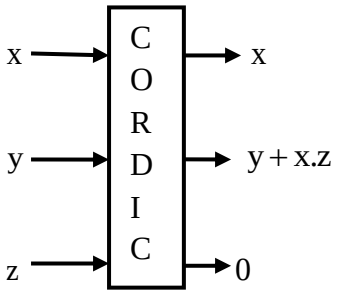
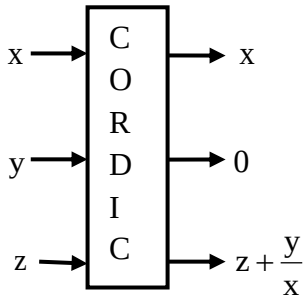
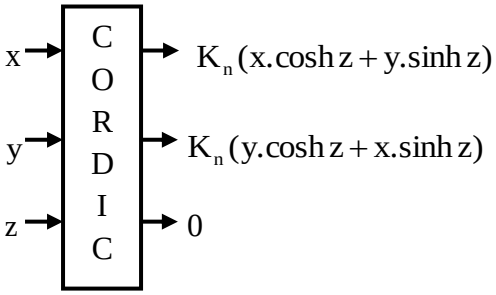
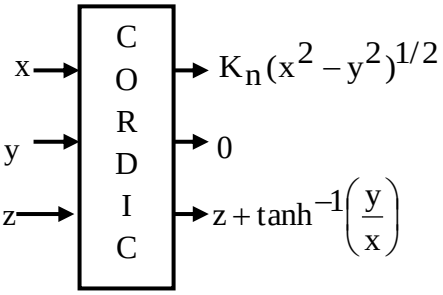


Figure 3.20: Hyperbolic rotation [11]

Table 3.4: CORDIC modes [2]

	Rotation mode $d_i = \begin{cases} +1, \text{ if } z_i \geq 0 \\ -1, \text{if } z_i < 0 \end{cases}$ $z(i) \to 0$	Vectoring mode $d_i = \begin{cases} -1, \text{if } y_i \geq 0 \\ +1, \text{if } y_i < 0 \end{cases}$ $y(i) \to 0$
$m = 1$ Circular $\phi_i = \tan^{-1} 2^{-i}$	 For cos and sin :- set $x = \frac{1}{K_n}, y = 0$	 For $\tan^{-1}$:- set $x = 1, z = 0$ $\cos^{-1} w = \tan^{-1}\left[\frac{\sqrt{1-w^2}}{w}\right]$ $\sin^{-1} w = \tan^{-1}\left[\frac{w}{\sqrt{1-w^2}}\right]$
$m = 0$ Linear $\phi_i = 2^{-i}$	 For multiplication:- set $y = 0$	 For division:- set $z = 0$
$m = 1$ Hyperbolic $\phi_i = \tanh^{-1} 2^{-i}$	 For cosh and sinh :-	 For $\tanh^{-1}$:- set $x = 1, z = 0$

	set $x = \frac{1}{K_n}, y = 0$ $\tanh z = \frac{\sinh z}{\cosh z},$ $e^z = \sinh z + \cosh z$	$\ln w = 2\tanh^{-1}\left(\frac{w-1}{w+1}\right)$ $w^{1/2} = \left[\left(w+\frac{1}{4}\right)^2 - \left(w-\frac{1}{4}\right)^2\right]^{1/2}$ $\cosh^{-1} w = \ln\left[w + \sqrt{1-w^2}\right]$ $\sinh^{-1} w = \ln\left[w + \sqrt{1+w^2}\right]$

3.5 Applications of CORDIC:-

The applications of CORDIC are:

3.5.1. CORDIC technique [6] is basically applied for the rotation of vector in circular, hyperbolic or linear co-ordinate systems, which in turn could also used for generation of sinusoidal waveform, multiplication and division operations and evaluation of angle of rotation, trigonometric functions, logarithms, exponential and square root.

CORDIC algorithm directly computes:-

- sin
- cos
- sinh
- cosh
- $\tan^{-1}$
- $\tanh^{-1}$
- multiplication(×)
- division (÷)
- $\tan^{-1}\left(\frac{y}{x}\right)$
- $y + x.z$
- $(x^2 + y^2)^{1/2}$
- $(x^2 - y^2)^{1/2}$
- $e^z = \cosh z + \sinh z$
- polar to rectangular transformations

- Cartesian to polar transformations

(1) To compute $\sin\theta$ and $\cos\theta$:- can compute in rotation mode.

$$m = 1$$

From equation (3.57), (3.58), we get

$$x_n = K_n(x_0.\cos z_0 - y_0.\sin z_0)$$

$$y_n = K_n(y_0.\cos z_0 + x_0.\sin z_0)$$

Put $x_0 = 1, y_0 = 0, z_0 = \theta$

$$\frac{x_n}{K_n} = \cos\theta \tag{3.87}$$

$$\frac{y_n}{K_n} = \sin\theta \tag{3.88}$$

(2) To compute $\sinh\theta$ and $\cosh\theta$:- can compute in rotation mode .

$$m = -1$$

From equation (3.57), (3.58), we get

$$x_n = K_n(x_0\cosh z_0 + y_0\sinh z_0)$$

$$y_n = K_n(y_0\cosh z_0 + x_0\sinh z_0)$$

Put $x_0 = 1, y_0 = 0, z_0 = \theta$

$$\frac{x_n}{K_n} = \cosh\theta \tag{3.89}$$

$$\frac{y_n}{K_n} = \sinh\theta \tag{3.90}$$

(3) To compute $\tan^{-1}$ and $\tanh^{-1}$:- can compute in vectoring mode.

To compute $\tan^{-1}$:- $m = 1$

$$x_n = K_n(x^2 + y^2)^{1/2} \tag{3.91}$$

$$y_n = 0 \tag{3.92}$$

$$z_n = z + \tan^{-1}\left(\frac{y}{x}\right) \tag{3.93}$$

To compute $\tan^{-1} y$ in vectoring mode by starting with $x = 1$ and $z = 0$

To compute $\tanh^{-1}$:- $m = -1$

$$x_n = K_n(x^2 + y^2)^{1/2} \tag{3.94}$$

$$y_n = 0 \tag{3.95}$$

$$z_n = z + \tanh^{-1}\left(\frac{y}{x}\right) \tag{3.96}$$

(4) Multiplication (×):- can compute in rotation mode.

$$m = 0$$

The basic iteration equations are

$$x_{i+1} = x_i + 0.d_i y_i 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

$$z_{i+1} = z_i - d_i \phi_i$$

$d_i = \pm 1$ (d_i is the direction of angle of rotation)

In Rotation mode, z converges to zero

$$z = 2^{-i}$$

The outputs of the rotation mode x_n, y_n and z_n are given by the following expressions, x_n and y_n being the co-ordinates of the rotated (by the angle ϕ) vector.

$$x_n = x_0$$

$$y_n = y_0 + x_0 z_0$$

$$z_n = 0$$

$$\text{Rule: choose } d_i \in \{-1, +1\} \text{ such that } z \to 0$$

Put $y_0 = 0$

$$y_n = x_0 z_0 \tag{3.97}$$

(5) Division (÷):- can be compute in vectoring mode.

$$m = 0$$

In vectoring mode, y converges to zero.

$$0 = y + x.2^{-i}$$

$$-y = x.2^{-i}$$

$$2^{-i} = -\frac{y}{x}$$

After n iterations, we have

$$x_n = x_0$$

$$y_n = 0$$

$$z_n = z_0 + \frac{y_0}{x_0}$$

$$\text{Rule}: \text{choose } d_i \in \{-1, +1\} \text{ such that } y \to 0$$

$$\text{Put } z_0 = 0$$

$$z_0 = \frac{y_0}{x_0} \tag{3.98}$$

(6) Polar to rectangular transformations: - can compute in rotation mode.

For rotation mode, the CORDIC equations are

$$x_{i+1} = x_i - d_i y_i 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

$$z_{i+1} = z_i - d_i \phi_i$$

or $$z_{i+1} = z_i - d_i \tan^{-1} 2^{-i}$$

$$d_i = \pm 1 \qquad (d_i \text{ is the direction of angle of rotation})$$

In Rotation mode, z converges to zero.

$$z = \sum \phi_i$$

The outputs of the rotation mode x_n, y_n and z_n are given by the following expressions, x_n and y_n being the co-ordinates of the rotated (by the angle ϕ) vector.

$$x_n = K_n (x_0 . \cos z_0 - y_0 . \sin z_0)$$

$$y_n = K_n (y_0 . \cos z_0 + x_0 . \sin z_0)$$

$$z_n = 0$$

$$\text{Rule}: \text{choose } d_i \in \{-1, +1\} \text{ such that } z \to 0$$

$$\text{Where } K_n = \prod_{i=0}^{n-1} (1 + 2^{-2i})^{1/2} = 1.6467$$

$$\text{Put } x_o = 1, y_0 = 0, z_0 = \theta$$

$$\frac{x_n}{K_n} = \cos\theta \tag{3.99}$$

$$\frac{y_n}{K_n} = \sin\theta \tag{3.100}$$

(7) Cartesian to polar transformations: - can compute in vectoring mode

In vectoring mode, the CORDIC equations are

$$x_{i+1} = x_i - d_i y_i 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

$$z_{i+1} = z_i - d_i \phi_i$$

or $$z_{i+1} = z_i - d_i \tan^{-1} 2^{-i}$$

$$d_i = \pm 1 \qquad (d_i \text{ is the direction of angle of rotation})$$

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

In vectoring mode, y converges to zero.

$$0 = y + x.2^{-i}$$

$$-y = x.2^{-i}$$

$$2^{-i} = -\frac{y}{x}$$

$$\tan \sum \phi_i = -\frac{y}{x}$$

The CORDIC equations has become

$$x_n = K_n (x^2 + y^2)^{1/2}$$

$$y_n = 0$$

$$z_n = z + \tan^{-1}\left(\frac{y}{x}\right)$$

$$\text{Rule}: \text{choose } d_i \in \{-1,+1\} \text{such that } y \to 0$$

$$\text{Put } z_0 = 0$$

Angle :- $$z_n = \tan^{-1}\left(\frac{y}{x}\right) \qquad (3.101)$$

Radius :- $$r = \sqrt{x^2 + y^2} \qquad (3.102)$$

Table 3.5: Calculation of different functions using CORDIC modes [6]

m	mode	Initial Values			Functions	
		x_{in}	y_{in}	z_{in}	x_R	y_R or z_R
1	Rotation	1	0	θ	$\cos\theta$	$\sin\theta$
-1	Rotation	1	0	θ	$\cosh\theta$	$\sinh\theta$
-1	Rotation	a	a	θ	ae^{θ}	ae^{θ}
1	Vectoring	1	a	$\pi/2$	$\sqrt{a^2+1}$	$\cot^{-1}a$
-1	Vectoring	a	1	0	$\sqrt{a^2-1}$	$\coth^{-1}a$
-1	Vectoring	a + 1	a - 1	0	$2\sqrt{a}$	$0.5\ln(a)$
-1	Vectoring	a + 1/4	a - 1/4	0	$\sqrt{a}$	$\ln\left(\frac{1}{4}a\right)$
-1	vectoring	a + b	a - b	0	$2\sqrt{ab}$	$0.5\ln\left(\frac{a}{b}\right)$

$$\tan z = \frac{\sin z}{\cos z}$$

$$\tanh z = \frac{\sinh z}{\cosh z}$$

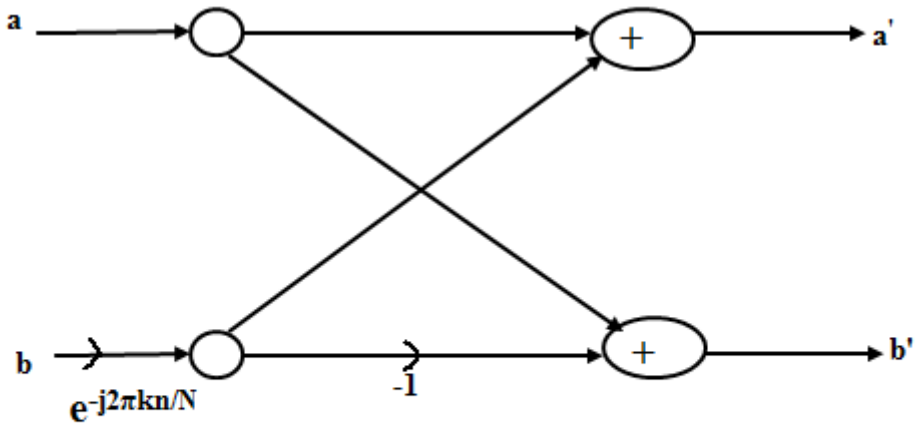
$$\ln z = 2\tanh^{-1}\left(\frac{z-1}{z+1}\right)$$

3.5.2. Signal Processing and Image processing Applications:- CORDIC techniques [3][18] have a wide range of DSP applications including the computation of discrete sinusoidal transforms such as the Discrete Fourier transform (DFT), Discrete Hartley transform (DHT), Discrete cosine transform (DCT), Discrete sine transform (DST). The DFT, DHT, DCT and DST of an N-point input sequence for $\{x(l) \text{ for } l = 0,1,2,...,N-1\}$ in general, are given by

$$X(k) = \sum_{l=0}^{N-1} C(k,l).x(l), \quad k = 0, 1, 2, ..., N-1 \tag{3.103}$$

Where the transform matrix is defined as

$$C(k,l)=\begin{cases}\cos(2\pi kl/N)-j\sin(2\pi kl/N)\text{, for DFT}\\ \cos(2\pi kl/N)+\sin(2\pi kl/N)\text{, for DHT}\\ \cos(\pi k(2l+1)/2N)\text{, for DCT}\\ \sin(\pi k(2l+1)/2N)\text{, for DST}\end{cases} \quad (3.104)$$



$$a' = a + b.e^{-j2\pi kn/N}$$
$$b' = a - b.e^{-j2\pi kn/N}$$

Figure 3.21: Butterfly unit in FFT [11].

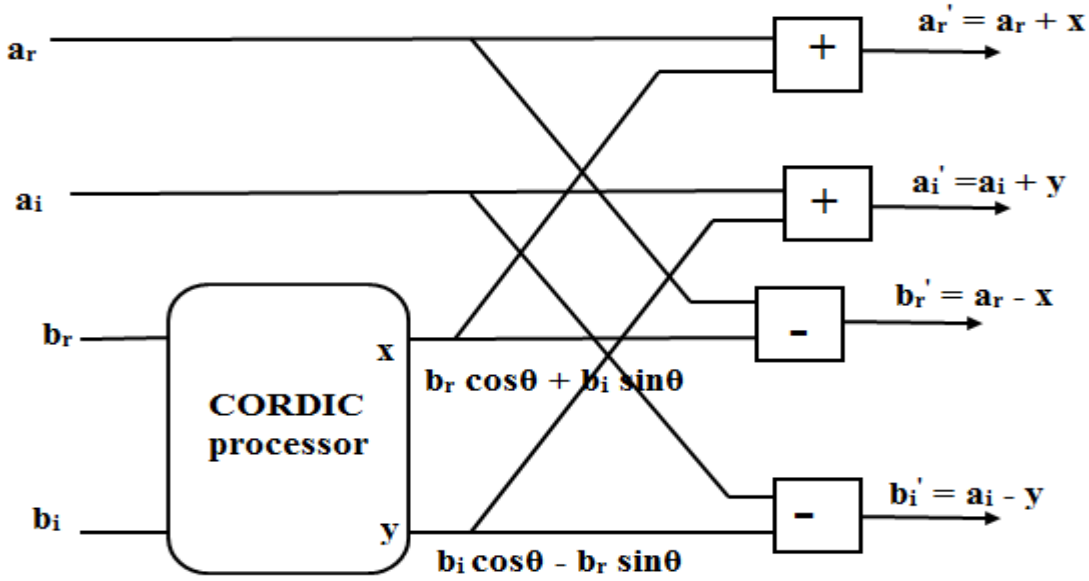


Figure 3.22: Block diagram of FFT computation based on CORDIC [18]

3.5.3. Direct digital synthesis (Application to communications):- Direct digital synthesis [3] is the process of generating sinusoidal waveforms directly in the digital domain. A direct digital synthesizer (DDS) consists of a phase accumulator and a phase-to-waveform converter the phase-generation circuit increments the phase according to $\left[\sum f_c\right]\pi$, where f_c is the normalized carrier frequency in every cycle and feeds the phase information to the phase-to-waveform converter. The phase-to-waveform converter could be realized by an RM-CORDIC .The cosine and sine waveforms are obtained, respectively, by the CORDIC outputs x_n and y_n.

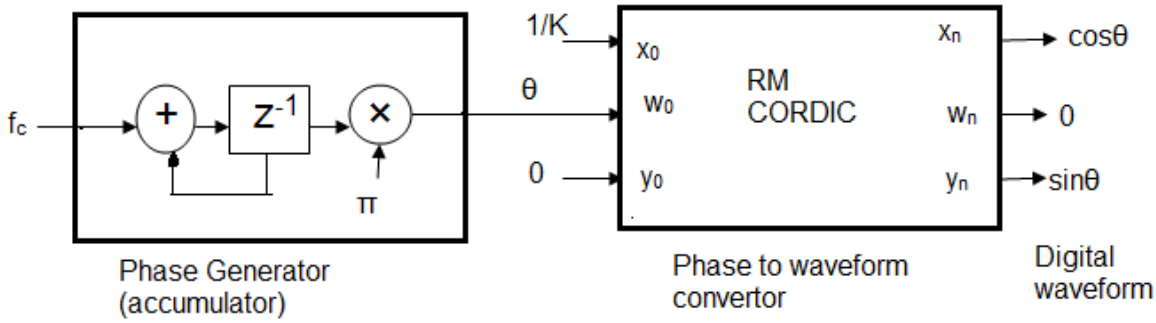


Figure3.23: CORDIC based direct digital synthesizer, $\theta = \pi.\left[\sum f_c\right]$ [3].

3.5.4. Analog and Digital Modulation: - The phase-generation unit [3] is changed to generate the phase according to $\left[\left(\sum\left(f_c + f_m\right)\right) + \phi_m\right]\pi$, for f_c and f_m being the normalized carrier and the modulating frequencies, respectively and ϕ_m is the phase of modulating component. By suitable selection of the parameters f_c, f_m and ϕ_m and the CORDIC inputs x_0 and y_0, it could be used for digital realization of analog amplitude modulation (AM), phase modulation (PM) and frequency modulation (FM), as well as the digital modulations, e.g., amplitude shift keying (ASK), phase-shift keying (PSK), and frequency- shift keying (FSK) modulators.

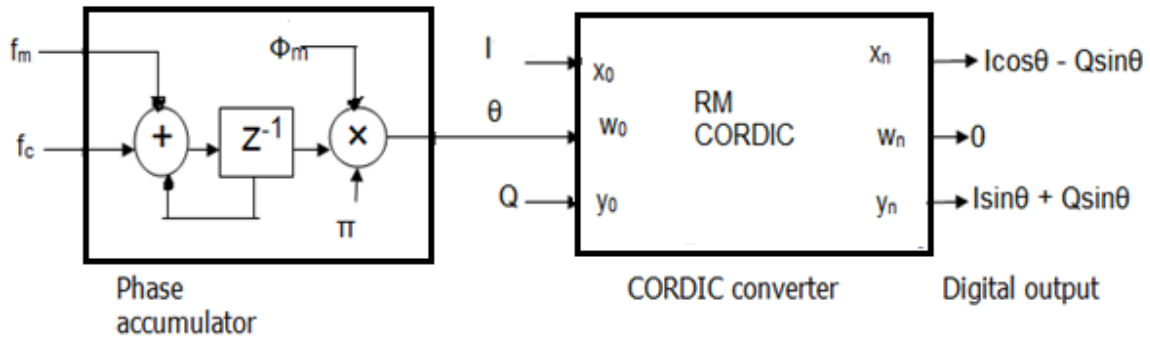


Figure 3.24: A generic scheme to use RM CORDIC for digital modulation, I and Q are respectively, the in-phase and quadrature signals to be modulated, $x_n = I.\cos\theta - Q.\sin\theta$, $y_n = I.\sin\theta + Q.\cos\theta$ and $\theta = \left[\left(\sum\left(f_c + f_m\right)\right) + \phi_m\right].\pi$ [3].

3.6 Control CORDIC

As time passed, improvements in process technology allowed the transistor count available on a chip to increase and the cost per bit to drop. The Control CORDIC method [8] was one method that took advantage of the availability of ROM space to implement a technique which reduced the number of iterations, but required a ROM to store the different scaling factors. The technique was based upon the observation that in the Original CORDIC algorithm, the iteration variable (z_i) does not always converge monotonically to 0–some of the iterations may actually result in divergent micro-rotations, which do nothing to improve the convergence towards the target vector.

The Control CORDIC method modifies the angle trajectory so that it now resembles a critically damped system, with no overshoot, resulting in faster convergence. Since divergent rotations can only occur when there is an overshoot, this method eliminates divergent micro-rotations by completely eliminating the overshoot itself. Unfortunately this also means that convergent micro rotations that overshoot the target are eliminated at the same time. The angle trajectory is modified by imposing a limit on the rotation directions. For positive angles the rotation direction (σ_i) is restricted to $\{+1, 0\}$ as given below:

$$\sigma = \begin{cases} +1, \text{ if } z_i \geq \phi_i \\ 0, \text{ otherwise} \end{cases} \tag{3.105}$$

This simple angle selection function thus prevents any overshoot of the target position by the moving vector.

$$25^o = 0 + 0 + 14.036^o + 7.125^o + 3.576^o + 0 + 0 + 0 + 0.2238^o$$

$$= 24.9608^o$$

The advantage of this method is that its angle selection function is quite simple, and is easy to implement with only a minimal effect on the cycle time, thus allowing its use in dynamic situations where the angle of rotation can take any value. However in return for the 11% reduction in iteration count, this method requires the use of a ROM to store the different scaling factors.

$$Q = \{45^o, 26.565^o, 14.036^o, 7.125^o, 3.576^o, 1.79^o, 0.895^o, 0.448^o, 0.2238^o\}$$

Rotation through original CORDIC:-

$$25^o = 45^o - 26.565^o + 14.036^o - 7.125^o - 3.576^o + 1.79^o + 0.895^o + 0.448^o + 0.2238^o$$

$$= 25.1268^o$$

Rotation through Control CORDIC:-

$$25^o = 0 + 0 + 14.036^o + 7.125^o + 3.576^o + 0 + 0 + 0 + 0.2238^o$$

$$= 24.9608^o$$

Table 3.6: Original CORDIC and control CORDIC

i	Original CORDIC	Control CORDIC
0	0^o	0^o
1	45^o	0^o
2	18.435^o	0^o
3	32.471^o	14.036^o
4	25.346^o	21.161^o
5	21.77^o	24.737^o
6	23.56^o	24.737^o
7	24.455^o	24.737^o
8	24.903^o	24.737^o
9	25.1268^o	24.9608^o

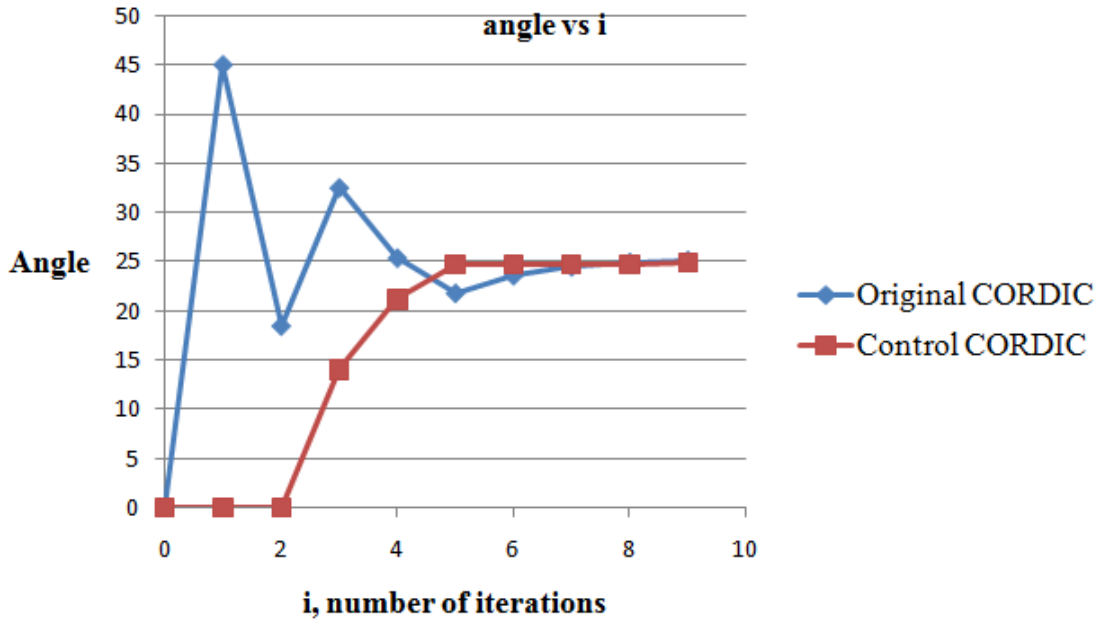


Figure 3.25: Original CORDIC and control CORDIC

3.7 Pipeline CORDIC

If an iterative implementation of CORDIC were used as discussed earlier, the generator would take several clock cycles to build a single output sample of the wave. However, using pipeline converts iterations into pipeline phases. In this way an output is obtained at every clock cycle, after pipeline stages propagation [9]. Each pipeline stage takes exactly one clock cycle to complete. In figure 3.27, register are implicit between each stage.

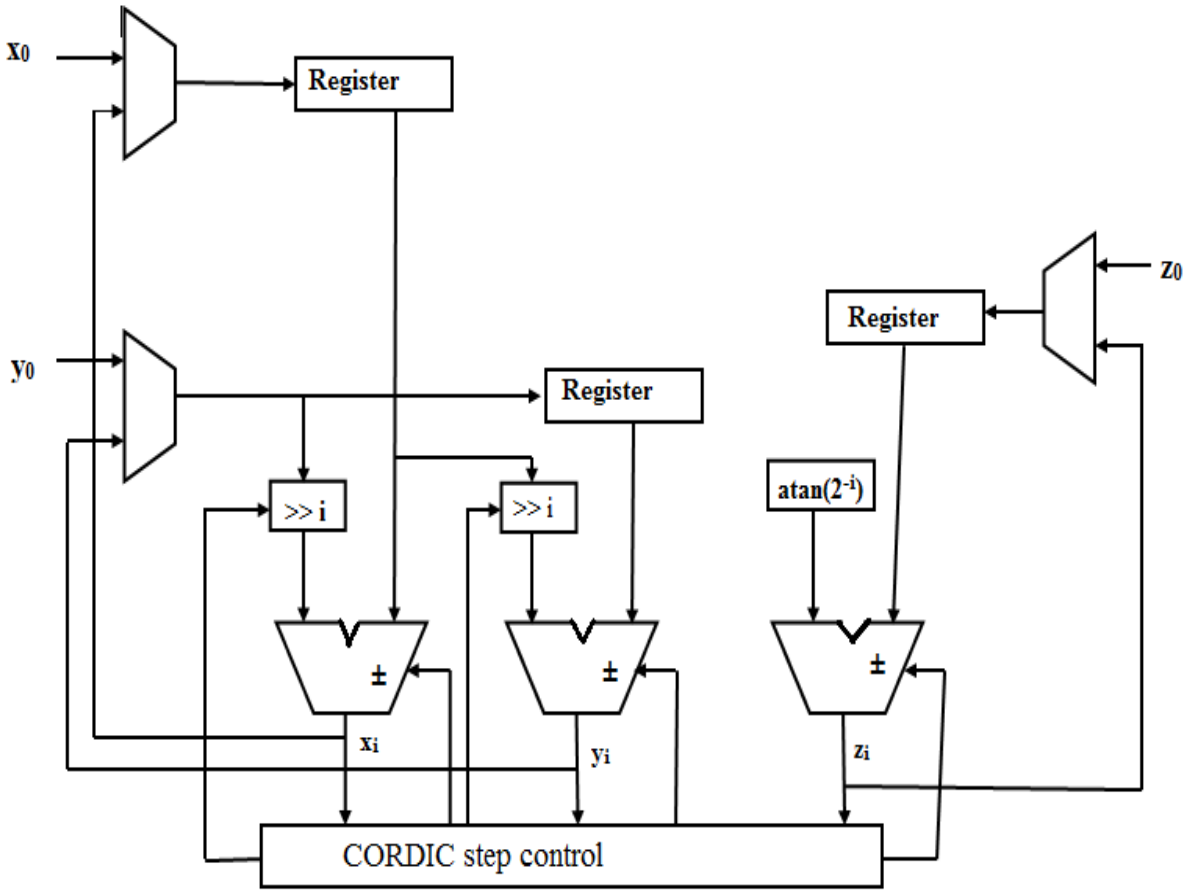


Figure 3.26: Iterative CORDIC [9]

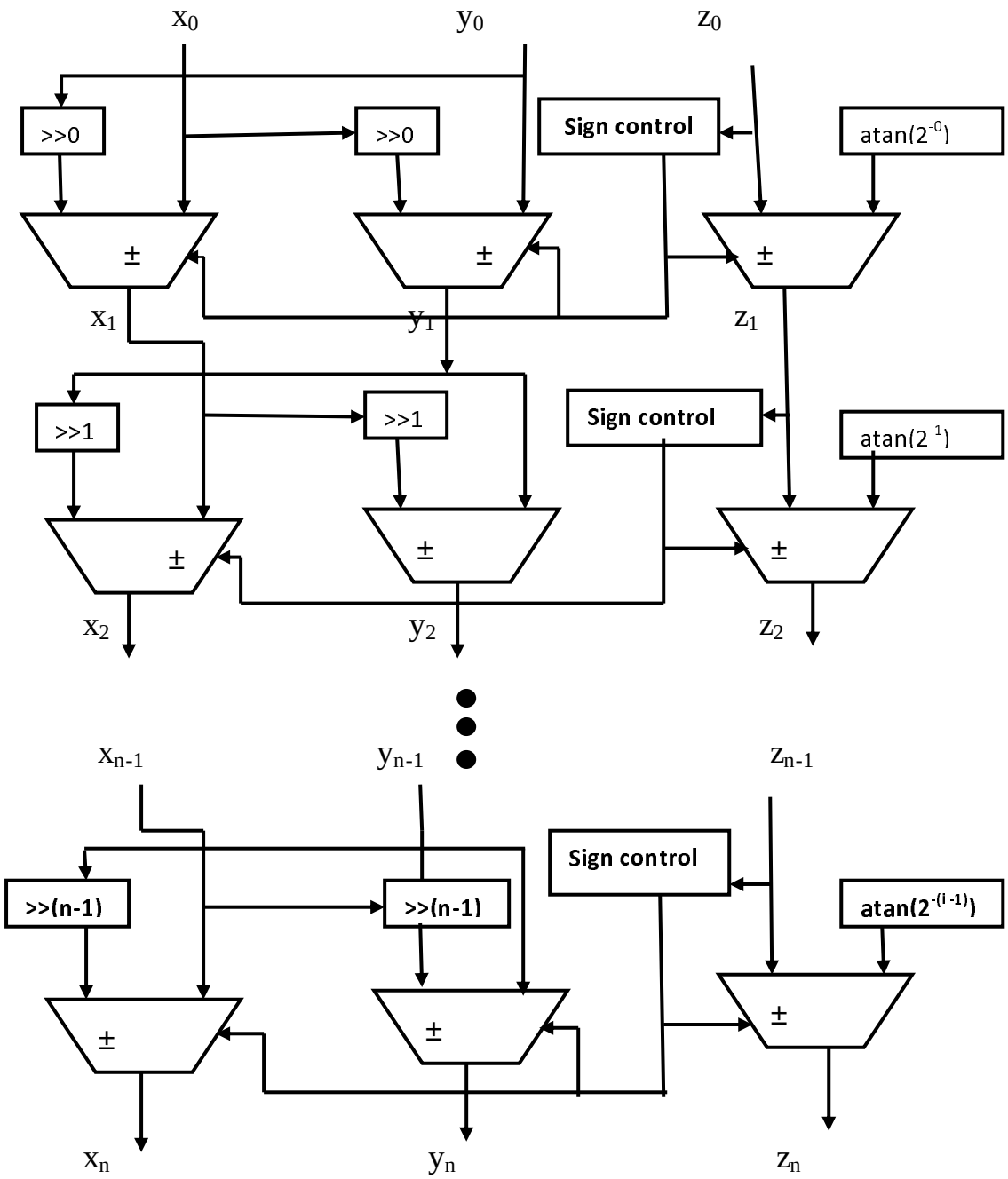


Figure 3.27: Pipelined CORDIC [9]

Pipeline architectures are used in CORDIC algorithm to reduce the critical path by introducing pipeline latches; this increases the clock speed or sampling speed and reduces the power consumption at the same speed in a DSP system.

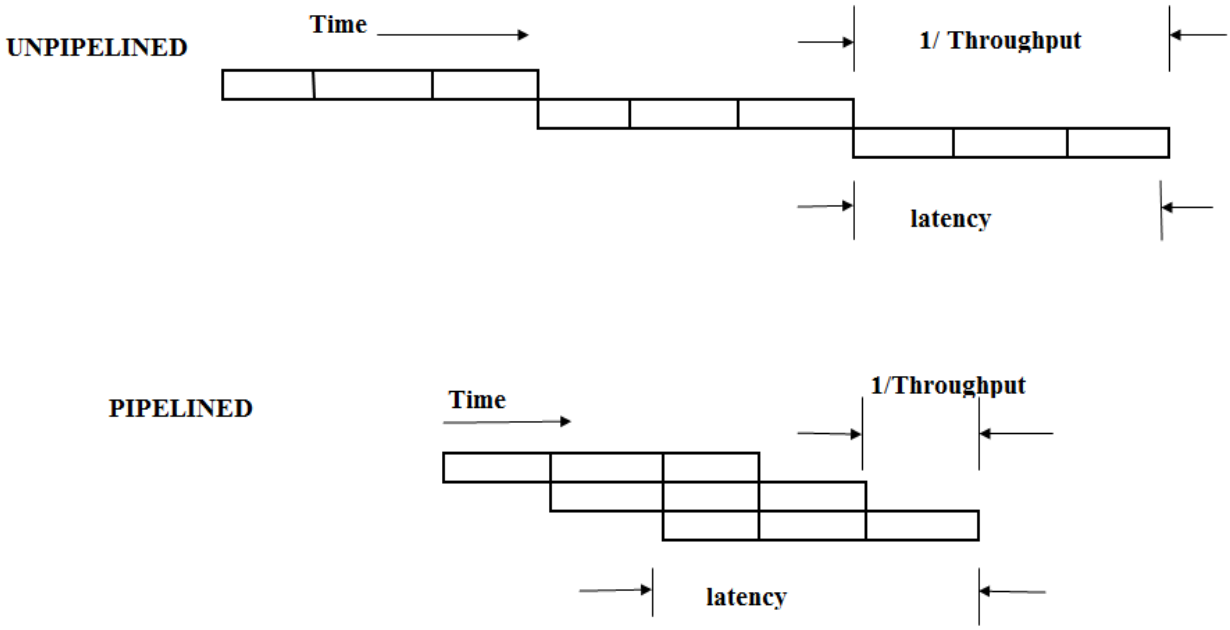


Figure 3.28: Unpipelined and pipelined architectures

In pipelined CORDIC processor, each of the n stages of the CORDIC iterations and s stages of scaling operations is realized with a separate CORDIC processor as shown in figure 3.29. By cascading these $(n+s)$ basic CORDIC processors [11], we will able to perform the entire CORDIC rotations operations, including all the CORDIC iterations and all the scaling operations in one clock cycle. Let t_0 be the time needed for performing a single CORDIC iteration or single scaling iteration. Then the total computation delay will be $(n+s)t_0$. Then the signal processing throughput rate is given by $\frac{1}{(n+s)t_0}$. To increase the throughput, one may choose to insert latch (or temporary storage digital circuit) between the successive stages of this pipelined CORDIC architecture as shown in figure 3.30. The latch is able to store the intermediate result after a CORDIC iteration or scaling operation. The computation of a CORDIC processor is isolated from the computation at the adjacent CORDIC processors. As a result, these latches can be clocked at a period of t_0 time units which is the computation delay between adjacent stages. Throughput rate is now increased $(n+s)$- fold to $\frac{1}{t_0}$. However, we note that each individual data will now take $n+s$ clock cycles to complete. Hence latency is $(n+s)t_0$ time units.

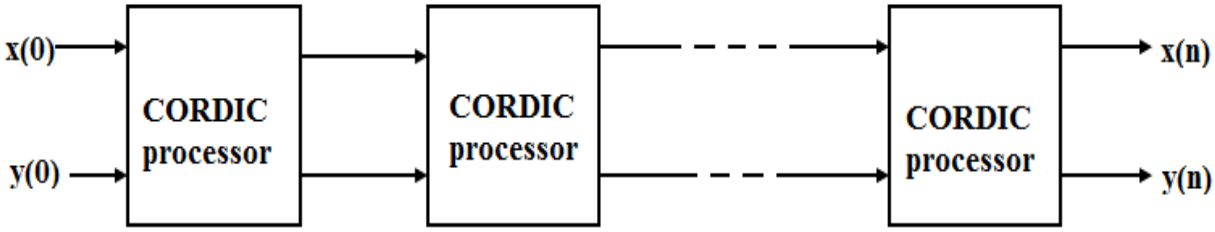


Figure 3.29: Pipelined implementation of the array [11]

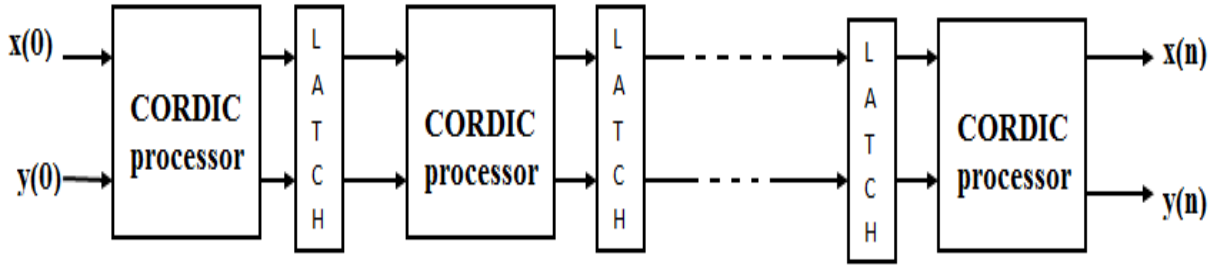


Figure 3.30: Different data are processed at different CORDIC iterations and scaling operations [11]

CHAPTER 4

CORDIC FOR DFT CALCULATION

In chapter 3, it is discussed that how sin and cos can be calculated using CORDIC algorithm and now using this algorithm how Discrete Fourier Transform (DFT) can be calculated is discussed in this chapter. A Fourier Transform is special case of wavelet transform with basis vectors defined by trigonometric functions like sin and cos. There are several ways to calculate the Discrete Fourier Transform (DFT) such as solving simultaneous linear equations or correlation method. The Fast Fourier Transform (FFT) is another method for calculating the DFT, while it produces the same result as the other approaches; it is more efficient, often reducing the computation time by hundreds. In 1965, Cooley and Tukey developed very efficient algorithm to implement FFT (also known as divide and conquer algorithm). A Fast Fourier Transform (FFT) is an efficient algorithm to compute Discrete Fourier Transform (DFT) and its inverse.

4.1 Discrete Fourier Transform (DFT)

Discrete Fourier Transform (DFT) is a finite duration discrete frequency sequence [19] which is obtained by sampling one period of Fourier transform.

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi kn/N} \tag{4.1}$$

This summation is taken for 'N' points; it is called as 'N' point DFT.

Twiddle factor [20] is given as

$$W_N = e^{-j2\pi/N} \tag{4.2}$$

Twiddle factor makes the computation of DFT a bit easy and fast.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \tag{4.3}$$

The sequence $x(n)$ is represented in the matrix form as follows:

$$x(n) = \begin{bmatrix} x(0) \\ x(1) \\ . \\ . \\ x(N-1) \end{bmatrix}_{N\times 1} \tag{4.4}$$

This is a $"N\times 1"$ matrix and n varies from 0 to $N-1$. Now the DFT of $x(n)$ is denoted by $X(k)$

$$X(k) = \begin{bmatrix} X(0) \\ X(1) \\ . \\ . \\ X(N-1) \end{bmatrix}_{N\times 1} \tag{4.5}$$

From equation (4.3), we get

$$W_N^{kn} = \begin{bmatrix} W_N^0 & W_N^0 & W_N^0 & . & W_N^0 \\ W_N^0 & W_N^1 & W_N^2 & . & W_N^{N-1} \\ W_N^0 & W_N^2 & W_N^4 & . & W_N^{2(N-1)} \\ . & . & . & . & . \\ W_N^0 & W_N^{N-1} & W_N^{2(N-1)} & . & W_N^{(N-1)^2} \end{bmatrix}_{N\times N} \tag{4.6}$$

4.1.1 Cyclic property of twiddle factor

From table 4.1, the value of W_8^0 is same as W_8^8, W_8^1 is same as W_8^9, W_8^2 is same as W_8^{10}.

Therefore,

$$\begin{aligned} W_8^0 &= W_8^8 = W_8^{16} \quad ..... \\ W_8^1 &= W_8^9 = W_8^{17} \quad ..... \\ W_8^2 &= W_8^{10} = W_8^{18} \quad ..... \\ . &\quad . \quad . \\ W_8^7 &= W_8^{15} = W_8^{23} \quad ..... \end{aligned} \tag{4.7}$$

This property of twiddle factor is called as periodicity property or cyclic property.

Table 4.1: Twiddle factor for 8 point DFT

Value of kn	$W_8^{kn} = e^{-j2\pi kn/N} = e^{-j\frac{\pi}{4}\times kn}$	Value of the phasor
0	$W_8^0 = e^0$	1
1	$W_8^1 = e^{-j\frac{\pi}{4}\times 1} = e^{-j\frac{\pi}{4}}$	$0.707 - j\,0.707$
2	$W_8^2 = e^{-j\frac{\pi}{4}\times 2} = e^{-j\frac{\pi}{2}}$	$0 - j1$
3	$W_8^3 = e^{-j\frac{\pi}{4}\times 3} = e^{-j\frac{3\pi}{4}}$	$-0.707 - j0.707$
4	$W_8^4 = e^{-j\frac{\pi}{4}\times 4} = e^{-j\pi}$	-1
5	$W_8^5 = e^{-j\frac{\pi}{4}\times 5} = e^{-j\frac{5\pi}{4}}$	$-0.707 + j0.707$
6	$W_8^6 = e^{-j\frac{\pi}{4}\times 6} = e^{-j\frac{3\pi}{2}}$	$0 + j1$
7	$W_8^7 = e^{-j\frac{\pi}{4}\times 7} = e^{-j\frac{7\pi}{4}}$	$0.707 + j0.707$
8	$W_8^8 = e^{-j\frac{\pi}{4}\times 8} = e^{-j2\pi}$	1
9	$W_8^9 = e^{-j\frac{\pi}{4}\times 9} = e^{-j\frac{9\pi}{4}}$	$0.707 - j0.707$
10	$W_8^{10} = e^{-j\frac{\pi}{4}\times 10} = e^{-j\frac{5\pi}{2}}$	$0 - j1$
11	$W_8^{11} = e^{-j\frac{\pi}{4}\times 11} = e^{-j\frac{11\pi}{4}}$	$-0.707 - j0.707$

4.1.2 Calculation of DFT, DHT, DCT and DST using CORDIC

CORDIC techniques [3][21][22] have a wide range of DSP applications including the computation of discrete sinusoidal transforms such as the Discrete Fourier transform (DFT), Discrete Hartley transform (DHT), Discrete cosine transform (DCT) and Discrete sine transform (DST). The DFT, DHT, DCT and DST of an N-point input sequence for $\{x(n) \text{ for } n = 0,1,2,...,N-1\}$ in general, are given by

$$X(k) = \sum_{n=0}^{N-1} C(k,n).x(n),\ k = 0,1,2,....,N-1 \tag{4.8}$$

Where the transform matrix is defined

$$C(k,n) = \begin{cases} \cos(2\pi kn/N) - j\sin(2\pi kn/N), \text{ for DFT} \\ \cos(2\pi kn/N) + \sin(2\pi kn/N), \text{ for DHT} \\ \cos(\pi k(2n+1)/2N), \text{ for DCT} \\ \sin(\pi k(2n+1)/2N), \text{ for DST} \end{cases} \tag{4.9}$$

As it is evident from the expressions, above transforms involve trigonometric operations on the input sample sequences. These transforms can be expressed in terms of plane rotations. In other words, all the input samples are given a vector rotation by defined angle in each of the transforms. The CORDIC unit can iteratively rotate an input vector $A = \lfloor A_x \quad A_y \rfloor$ by target angle θ through small steps of elementary angles θ_i (so that $\sum_i \theta_i$) to generate an output vector $B = \lfloor B_x \quad B_y \rfloor$. The operations can be represented mathematically as:

$$\lfloor B_x \quad B_y \rfloor = \lfloor A_x \quad A_y \rfloor \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{4.10}$$

Using sine cosine value generated by CORDIC algorithm DFT can be calculated by arranging them in matrix. It can be done by arranging sine cosine value in two different matrix one as real part and another as imaginary part then multiplying by input sampled data $x(n)$ which results in $X(k)$. Result can be stored in matrix as real and imaginary part. In the case of DHT computation, since there is no imaginary part the two matrixes can be shown in a single matrix by adding the two resulting matrix of sine and matrix of cosine in the case of 8×8 matrix. The two results in the case of DFT and one result in case of DHT can be arranged as below.

DFT: $$X(k) = \sum_{n=0}^{N-1} x(n)\left[\cos\left(\frac{2\pi kn}{N}\right) - j.\sin\left(\frac{2\pi kn}{N}\right)\right] \tag{4.11}$$

DHT: $$H(k) = \sum_{n=0}^{N-1} x(n)\left[\cos\left(\frac{2\pi kn}{N}\right) + \sin\left(\frac{2\pi kn}{N}\right)\right] \tag{4.12}$$

In case of DFT (Discrete Fourier transform):-

$$X_R(k) = \begin{bmatrix} W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 \\ W_8^0 & W_8^1 & W_8^2 & W_8^3 & W_8^4 & W_8^5 & W_8^6 & W_8^7 \\ W_8^0 & W_8^2 & W_8^4 & W_8^6 & W_8^8 & W_8^{10} & W_8^{12} & W_8^{14} \\ W_8^0 & W_8^3 & W_8^6 & W_8^9 & W_8^{12} & W_8^{15} & W_8^{18} & W_8^{21} \\ W_8^0 & W_8^4 & W_8^8 & W_8^{12} & W_8^{16} & W_8^{20} & W_8^{24} & W_8^{28} \\ W_8^0 & W_8^5 & W_8^{10} & W_8^{15} & W_8^{20} & W_8^{25} & W_8^{30} & W_8^{35} \\ W_8^0 & W_8^6 & W_8^{12} & W_8^{18} & W_8^{24} & W_8^{30} & W_8^{36} & W_8^{42} \\ W_8^0 & W_8^7 & W_8^{14} & W_8^{21} & W_8^{28} & W_8^{35} & W_8^{42} & W_8^{49} \end{bmatrix}_R \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \quad (4.13)$$

Suffix R show real part, in the above 8×8 matrix only cosine value is stored at the corresponding positions.

$$X_I(k) = \begin{bmatrix} W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 \\ W_8^0 & W_8^1 & W_8^2 & W_8^3 & W_8^4 & W_8^5 & W_8^6 & W_8^7 \\ W_8^0 & W_8^2 & W_8^4 & W_8^6 & W_8^8 & W_8^{10} & W_8^{12} & W_8^{14} \\ W_8^0 & W_8^3 & W_8^6 & W_8^9 & W_8^{12} & W_8^{15} & W_8^{18} & W_8^{21} \\ W_8^0 & W_8^4 & W_8^8 & W_8^{12} & W_8^{16} & W_8^{20} & W_8^{24} & W_8^{28} \\ W_8^0 & W_8^5 & W_8^{10} & W_8^{15} & W_8^{20} & W_8^{25} & W_8^{30} & W_8^{35} \\ W_8^0 & W_8^6 & W_8^{12} & W_8^{18} & W_8^{24} & W_8^{30} & W_8^{36} & W_8^{42} \\ W_8^0 & W_8^7 & W_8^{14} & W_8^{21} & W_8^{28} & W_8^{35} & W_8^{42} & W_8^{49} \end{bmatrix}_I \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \quad (4.14)$$

Suffix I show imaginary part, in the above 8×8 matrix only sine value is stored at the corresponding positions.

Final value is given as: $X(k) = X_R(k) + j.X_I(k)$. The two result i.e. real and imaginary part can be stored in RAM location for further use.

In case of DHT (Discrete Hartley Transform):-

There is no imaginary part the two values generated in the matrices be added together. The resulting value of $H(k)$ by using equation (4.12)

$$H(k) = \begin{bmatrix} W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 \\ W_8^0 & W_8^1 & W_8^2 & W_8^3 & W_8^4 & W_8^5 & W_8^6 & W_8^7 \\ W_8^0 & W_8^2 & W_8^4 & W_8^6 & W_8^8 & W_8^{10} & W_8^{12} & W_8^{14} \\ W_8^0 & W_8^3 & W_8^6 & W_8^9 & W_8^{12} & W_8^{15} & W_8^{18} & W_8^{21} \\ W_8^0 & W_8^4 & W_8^8 & W_8^{12} & W_8^{16} & W_8^{20} & W_8^{24} & W_8^{28} \\ W_8^0 & W_8^5 & W_8^{10} & W_8^{15} & W_8^{20} & W_8^{25} & W_8^{30} & W_8^{35} \\ W_8^0 & W_8^6 & W_8^{12} & W_8^{18} & W_8^{24} & W_8^{30} & W_8^{36} & W_8^{42} \\ W_8^0 & W_8^7 & W_8^{14} & W_8^{21} & W_8^{28} & W_8^{35} & W_8^{42} & W_8^{49} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \quad (4.15)$$

4.2 Computational complexity using DFT and FFT:-

4.2.1 Computational complexity using DFT:-

To obtain DFT of a sequence by using direct computation. DFT of $x(n)$ is defined by:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \text{ , for } k = 0,1,2,.....,N-1 \quad (4.16)$$

If the value of 'N' is large, then the number of computations will go into lakhs and more processing time is required. This proves inefficiency of direct DFT computation. To implement DFT, a very efficient algorithm is used called Fast Fourier Transform (FFT). Equation (4.16) indicates that we have to take multiplication of $x(n)$ and twiddle factor. Then we have to add all the terms. Since twiddle factor is complex; we need to perform complex multiplications and complex additions.

(1) Complex multiplication: - Equation (4.16), for one value of 'k' multiplication should be performed for all value of 'n'. The range of 'n' is from 0 to $N-1$. So for one value of 'k'; N complex multiplications are required. Now the range of 'k' is also from 0 to $N-1$. The total complex multiplications are

Complex multiplications = $N \times N = N^2$

(2) Complex additions: - Equation (4.16), for each value of k we need to add the product terms of $x(n).W_N^{kn}$. For example: $N = 4$

$$\text{for } k = 0 \Rightarrow X(0) = \sum_{n=0}^{3} x(n).W_4^0$$

$$= x(0).W_4^0 + x(1).W_4^1 + x(2).W_4^2 + x(3).W_4^3 \qquad (4.17)$$

From equation (4.17), four complex multiplications and three complex additions are required. Thus for each value of 'k', N complex multiplications and $(N-1)$ complex additions are required.

Complex additions = $N(N-1) = N^2 - N$

4.2.2 Computational complexity using FFT:-

To calculate the computation complexity [23] required for one butterfly.

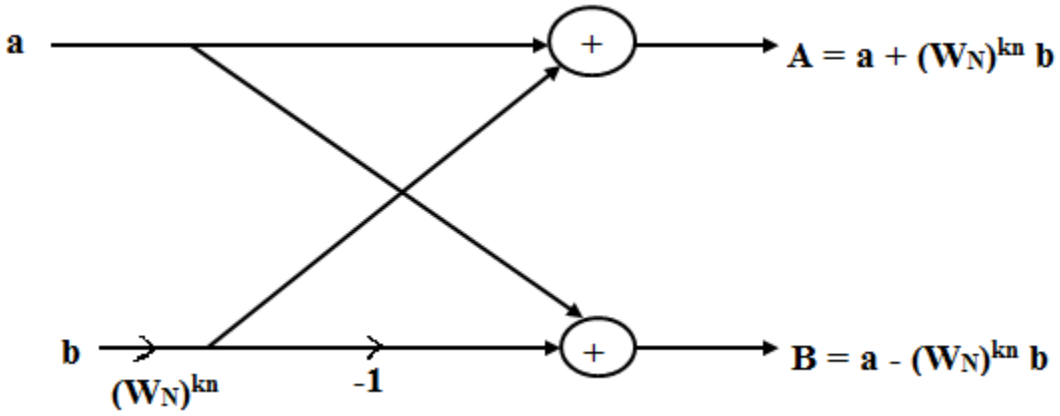


Figure 4.1 General structure of butterfly [11]

To calculate any output (A or B), we need to multiply input 'b' by twiddle factor W_N^{kn}. So one complex multiplication is required for one butterfly. To calculate output 'A', one complex addition is required, while to calculate output 'B', one complex subtraction is required. But computational complexity of addition and subtraction are same.

For 'N' point DFT, at each stage $\frac{N}{2}$ butterflies and $\log_2 N$ are required.

For 8 point DFT, 4 butterflies and 3 stages are required.

For N point DFT Butterflies = $\frac{N}{2}$, Stages = $\log_2 N$

(1) Complex multiplication: - At each stage, there are $\frac{N}{2}$ butterflies. Total numbers of stages are $\log_2 N$.

$$\boxed{\text{Complex multiplications} = \frac{N}{2}\log_2 N}$$

(2) Complex additions: - At each stage, there are $\frac{N}{2}$ butterflies. Total numbers of stages are $\log_2 N$. And for each butterfly, two complex additions are required.

$$\text{Total complex additions} = \frac{N}{2}\log_2 N$$

$$\boxed{\text{Complex additions} = N\log_2 N}$$

(3) In place computation to reduce memory size: - A butterfly calculates the value of 'A' and 'B' for inputs 'a' and 'b'. Remember that 'a' and 'b' are complex inputs. So two memory locations are required to store any one of the inputs 'a' or 'b'. One memory location is required to store real part and other memory location is used to store imaginary part. To store both 'a' and 'b'; 4 memory locations are required.

$$\begin{aligned} A &= a + W_N^{kn} b \\ B &= a - W_N^{kn} b \end{aligned} \tag{4.18}$$

Outputs 'A'and 'B' are calculated by using the values of 'a' and 'b' are stored in the memory. 'A' and 'B' are also complex numbers so 4 memory locations are required to store both the outputs A and B. Once the computation of 'A'and 'B' is done then value of 'a' and 'b' are not required. So instead of storing 'A' and 'B' at other memory locations; these values are stored at the same place where 'a' and 'b' were stored. 'A' and 'B' are stored in place of 'a' and 'b'. This is called in-place computation. In place computation reduces memory size.

Memory Requirement: - Four memory locations are required for every butterfly to store input and output values. There are $\frac{N}{2}$ butterflies per stage.

Memory location required to store inputs and outputs = $4 \times \frac{N}{2} = 2N$

One value of twiddle factor is required to compute A and B .To store one value of twiddle factor; one memory location is required for each butterfly. There are $\frac{N}{2}$ butterflies at each stage.

Memory location required to store twiddle factor = $\frac{N}{2}$

Total memory requirement = $2N + \frac{N}{2}$

Table 4.2: Comparison between direct computation (DFT) and FFT [18]

Number of Points 'N'	Direct computation (DFT)		FFT(Fast Fourier Transform)	
	Complex multiplications (N^2)	Complex additions ($N^2 - N$)	Complex multiplications $\left(\frac{N}{2}\log_2 N\right)$	Complex additions ($N\log_2 N$)
4	16	12	4	8
8	64	56	12	24
16	256	240	32	64
32	1024	992	80	160
64	4096	4032	192	484
128	16384	16256	448	896

The use of FFT algorithms, number of complex multiplications and complex additions are reduced. So there is tremendous improvement in speed.

4.3 Fast Fourier Transform (FFT)

Fast Fourier transform (FFT) is an efficient algorithm for computing the Discrete Fourier transform (DFT) [11]. It requires less multiplication than a simple approach of calculating DFT. The basic computation using FFT is called butterfly computation which is shown in figure 4.1. The choice of the CORDIC algorithm for realization the basic butterfly operation for the FFT saves a lot of hardware compared to its counterparts employing other techniques. The Decimation in time (DIT) algorithm is used for FFT algorithm. To decimate means to break into parts. The Decimation in time (DIT) indicates dividing (splitting) the sequence in time domain.

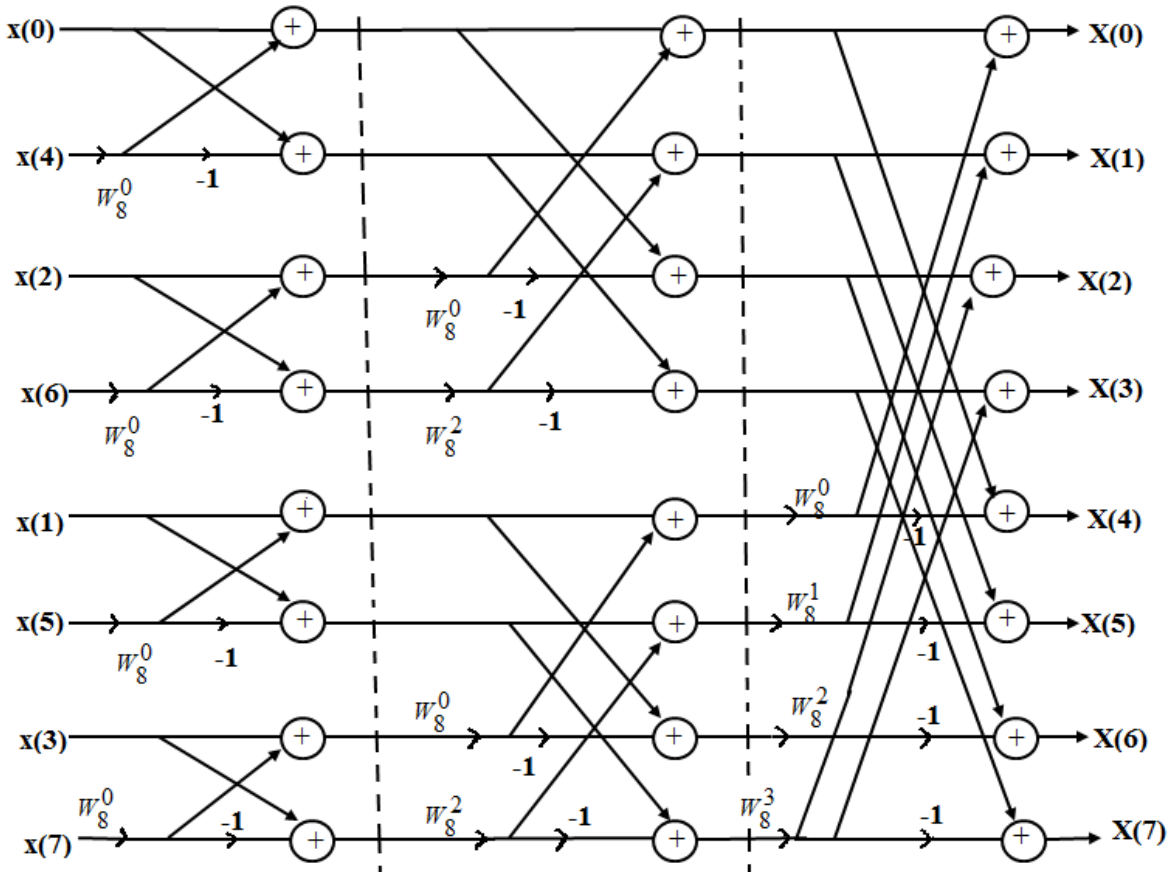


Figure 4.2: 8-point Decimation in time (DIT) FFT algorithm [11]

The steps involved in FFT (divide and conquer algorithm) method [19] can be understood by the following computations.

(1) Store the signal row-wise.

(2) Compute the N-point DFT of each column.

(3) Multiply the resulting array by the phase factors W_N^{kn}.

(4) Compute the N-point DFT of each row.

(5) Read the resulting array column wise.

Decimation in time (DIT):- To decimate means to break into parts [11][23]. The DIT indicates dividing (splitting) the sequence in time domain.

We take N point DFT and break it down into two $\frac{N}{2}$ point DFTs by splitting the input signal into odd & even numbered samples. From equation (4.1), let $x(n)$ be the given input sequence containing 'N' samples. For decimation in time, we will divide the sequence into even and odd sequences.

Let $f_1(m)$ and $f_2(m)$ be even and odd sequence respectively.

$$x(n) = f_1(m) + f_2(m)\ ,\ m = 0,1,2,....,N-1 \tag{4.19}$$

For even sequence: $$f_1(m) = x(2n)\ ,\ m = 0,1,2,.....,\left(\frac{N}{2}-1\right) \tag{4.20}$$

For odd sequence: $$f_2(m) = x(2n+1)\ ,\ m = 0,1,2,.....\left(\frac{N}{2}-1\right) \tag{4.21}$$

$$X(k) = \sum_{n\,even} x(n).W_N^{kn} + \sum_{n\,odd} x(n).W_N^{kn} \tag{4.22}$$

The first summation represents even sequence so we will put $n = 2m$ in the first summation. The second summation represents odd sequence so we will put $n = 2m+1$ in the second summation. Since even and odd sequence contain $\frac{N}{2}$ samples each.

$$X(k) = \sum_{m=0}^{\frac{N}{2}-1} x(2m).W_N^{2mk} + \sum_{m=0}^{\frac{N}{2}-1} x(2m+1).W_N^{(2m+1)k} \tag{4.23}$$

$$X(k) = \sum_{m=0}^{\frac{N}{2}-1} f_1(m).W_N^{2mk} + \sum_{m=0}^{\frac{N}{2}-1} f_2(m).W_N^{2mk}.W_N^k \quad (4.24)$$

$$X(k) = \sum_{m=0}^{\frac{N}{2}-1} f_1(m).(W_N^2)^{Km} + W_N^k \sum_{m=0}^{\frac{N}{2}-1} f_2(m).(W_N^2)^{Km} \quad (4.25)$$

We have $W_N^2 = W_{N/2}$

$$X(k) = \sum_{m=0}^{\frac{N}{2}-1} f_1(m).W_{N/2}^{km} + W_N^k \sum_{m=0}^{\frac{N}{2}-1} f_2(m).W_{N/2}^{km} \quad (4.26)$$

$$X(k) = F_1(k) + W_N^k.F_2(k) \quad (4.27)$$

$F_1(k)$ and $F_2(k)$ is $\frac{N}{2}$ point DFT of $f_1(m)$ and $f_2(m)$ respectively. For 8 point DFT, that means $F_1(k)$ and $F_2(k)$ are 4 point DFT.

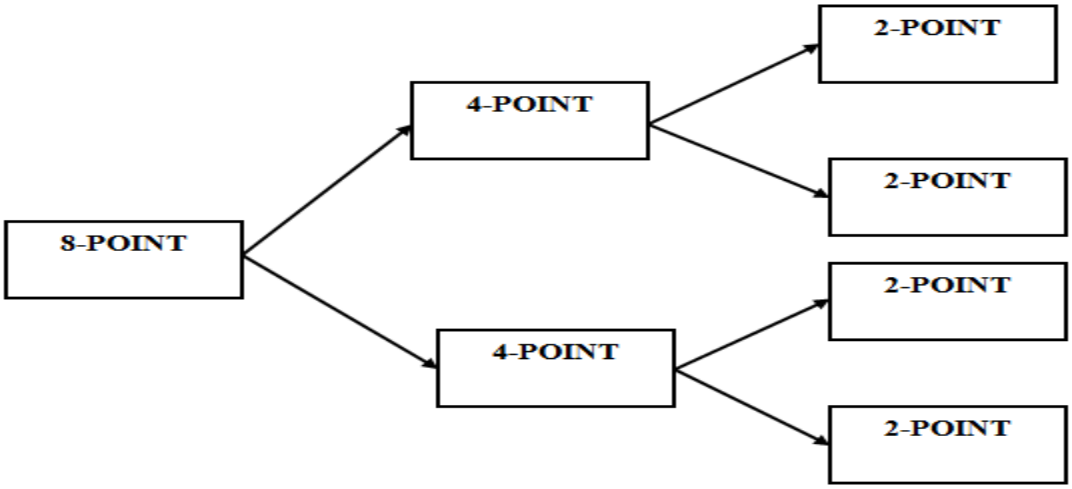


Figure 4.3: Fast Fourier Transform (FFT) [11]

CHAPTER 5

FPGA DESIGN FLOW

5.1 INTRODUCTION

FPGA stands for Field programmable gate arrays that can be configured by the customer or designer after manufacturing. An FPGA is a device that consists of thousands or even millions of transistors connected to perform logic functions [24]. They perform functions from simple addition and subtraction to complex digital filtering and error correction and detection, Aircraft, automobiles, radar, missiles and computers are just some of the systems that use FPGAs. A main benefit to using FPGAs is that design changes need not have an impact on the external hardware. Xilinx, Altera and Quick logic are just a few companies that manufacture FPGAs. FPGA provide a lot of flexibility and opportunity to make design changes quickly. Field programmable gate arrays are called this because rather than having a structure similar to a PAL or other programmable device, they are structured very much like a gate array ASIC. FPGAs are programmed using a logic circuit diagram or a source code in a hardware descriptive language (HDL) to specify how the chip will work. FPGAs contain logic components called “logic blocks” and hierarchy of reconfigurable interconnects that allow the blocks to be “wired together”. The programmable logic blocks are called configurable logic blocks (CLBs) and reconfigurable interconnects are called switch boxes [25]. Logic block (CLBs) can be configured to perform complex combinational function or simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements which may be simple flip-flops or block of memory.

5.2 FPGA Architecture

FPGA Architecture consists of three capabilities: Input/output (I/O) interfaces basic building blocks (Configurable logic block) and Programmable interconnect. Also, there will clock circuitry for driving the clock signals to each logic block and additional logic resources such as ALUs, memory, and decoders may be available. FPGA architecture is shown in figure 5.1.

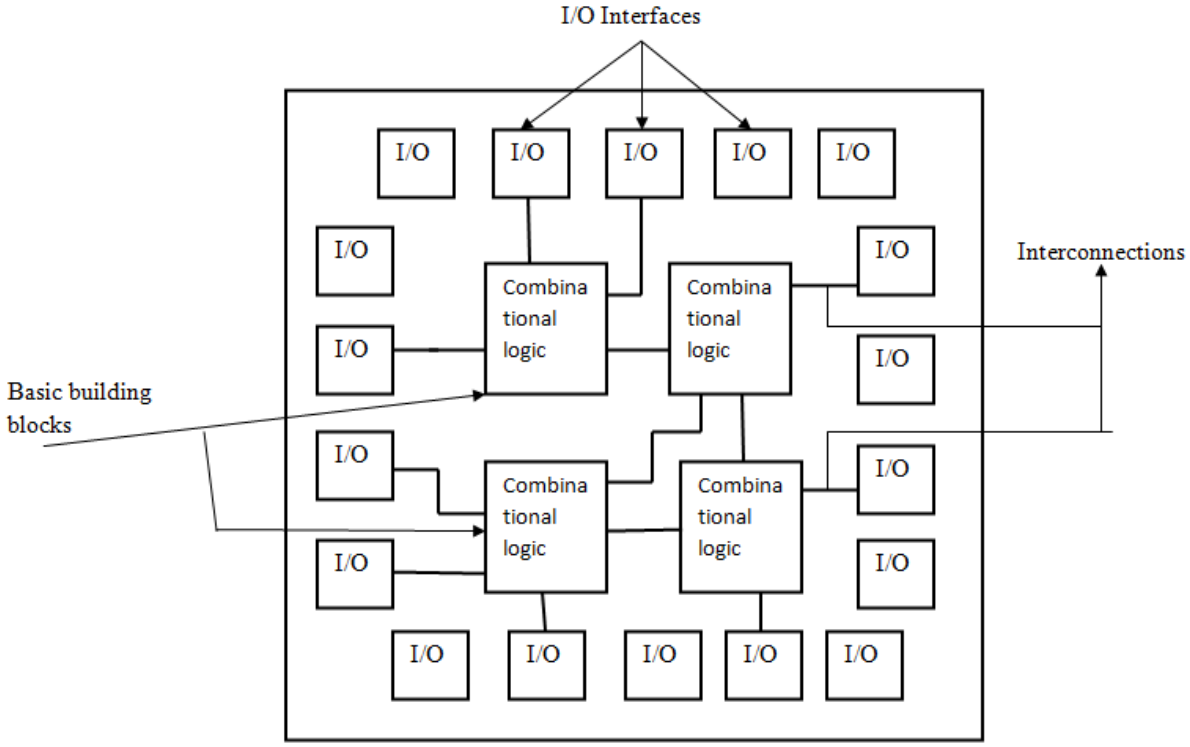


Figure 5.1: FPGA Architecture [24]

5.2.1 Input/output (I/O) interface

I/O interfaces are the mediums in which data are sent from internal logic to external sources and from which data are received from external sources. The interface signals can be unidirectional or bidirectional. At Xilinx, the I/O interfaces are called I/O blocks (IOBs). The IOBs consist of registers, internal voltage translators, and other specialized resources.

5.2.2 Configurable logic block (CLBs)

Configurable logic block (CLBs) contains the logic for FPGA. The basic logic building blocks (CLBs) are preconfigured logic and resources used to build your design [24]. Xilinx's basic building blocks are called configurable logic blocks (CLBs). Each CLB contains slices and each slice has lookup tables (LUTs). FPGAs are used in a variety of applications and systems; there is no "one device for all applications".

5.2.3 Programmable interconnect

Interconnection involves connecting the basic building blocks to perform design-specific functions as well as connecting the internal logic to I/O interfaces. The interconnect of an FPGA is very different than that of a CPLD, but is rather similar to that of a gate array ASIC. There are long lines which can be used to connect critical CLBs that are physically far from each other on the chip without inducing much delay. They can also be used as buses within the chip. There are also short lines which are used to connect individual CLBs which are located physically close to each other. There is often one or several switch matrices, like that in a CPLD, to connect these long and short lines together in specific ways as shown in figure 5.2. Programmable switches inside the chip allow the connection of CLBs to interconnect lines and interconnect lines to each other and to the switch matrix. Three-state buffers are used to connect many CLBs to a long line, creating a bus. Special long lines, called global clock lines, are specially designed for low impedance and thus fast propagation times. These are connected to the clock buffers and to each clocked element in each CLB. This is how the clocks are distributed throughout the FPGA.

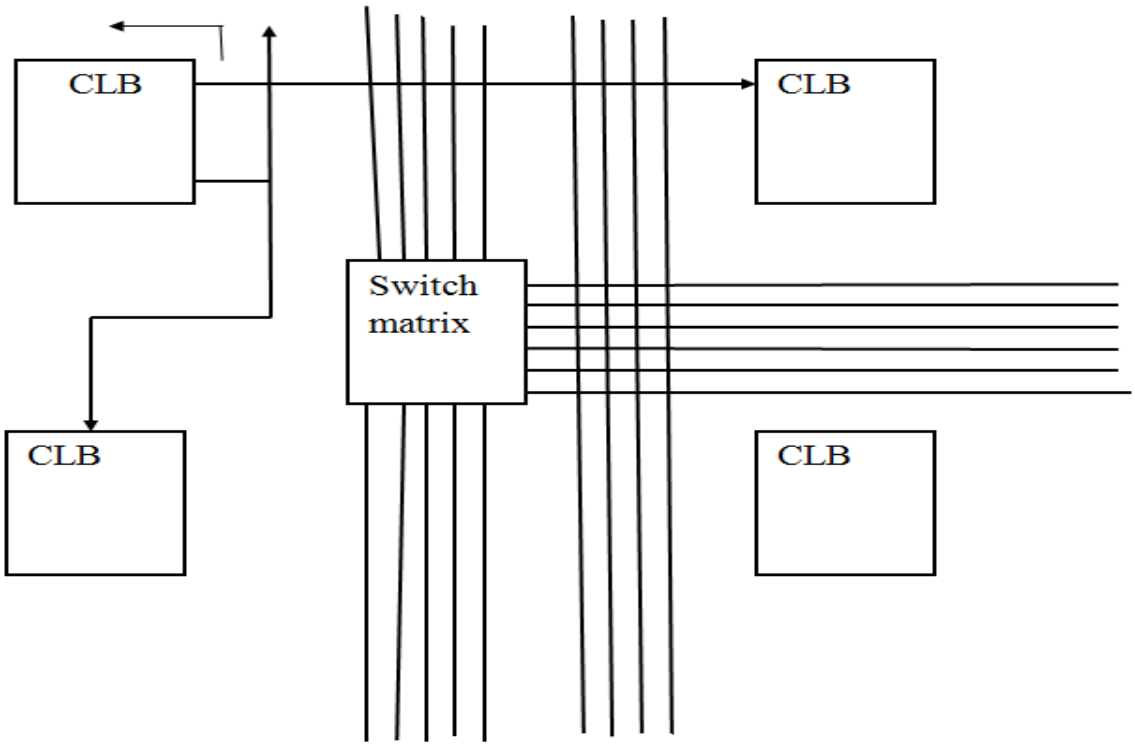


Figure 5.2: FPGA programmable interconnect

5.2.4 Clock Circuitry

Special I/O blocks with special high drive clock buffers, known as clock drivers, are distributed around the chip. These buffers are connected to clock input pads and drive the clock signals onto the global clock lines described above. These clock lines are designed for low skew times and fast propagation times.

5.3 Design Flow

The Design flow involves more than just creating a design. Many decisions must be made and the design material must be understood. The Design flow consists of various steps:

5.3.1 Writing a specification

This is especially as a guide for choosing the right technology and for making your needs known to the vendor. As specification allows each engineer to understand the entire design and his or her piece of it. It allows the engineer to design the correct interface to the rest of the pieces of the chip. It also saves time.

A specification should include the following information:

(1) An external block diagram showing how the chip fits into the system.

(2) An internal block diagram showing each major functional section.

(3) A description of the I/O pins.

(4) Setup and hold times for input pins.

(5) Propagation times for output pins.

(6) Clock cycle time.

(7) Estimated gate count.

(8) Package type.

(9) Target power consumption.

(10) Target price

(11) Test procedures

5.3.2 Choosing a technology

Once a specification has been written, it can be used to find the best vendor with a technology and price structure that best meets your requirements.

5.3.3 Choosing a Design Entry method

The basic Architecture of the system is designed in this step which is coded in hardware description language like VHDL or Verilog.

5.3.4 Choosing a synthesis tool

You must decide at this point which synthesis software you will be using if you plan to design the FPGA with an HDL. This is important since each synthesis tool has recommended or mandatory methods of designing hardware so that it can correctly perform synthesis. It will be necessary to know these methods up front so that sections of the chip will not need to be redesigned later on. At the end of this phase it is very important to have a design review. All appropriate personnel should review the decisions to be certain that the specification is correct, and that the correct technology and design entry method have been chosen.

5.3.5 Designing the chip

It is very important to follow good design practices. Top-down design approach is used for good design practices. Top-down design is the design method whereby high level functions are defined first, and the lower level implementation details are filled in later. A schematic can be viewed as a hierarchical tree as shown in Figure 5.4. The top-level block represents the entire chip. Each lower level block represents major functions of the chip.

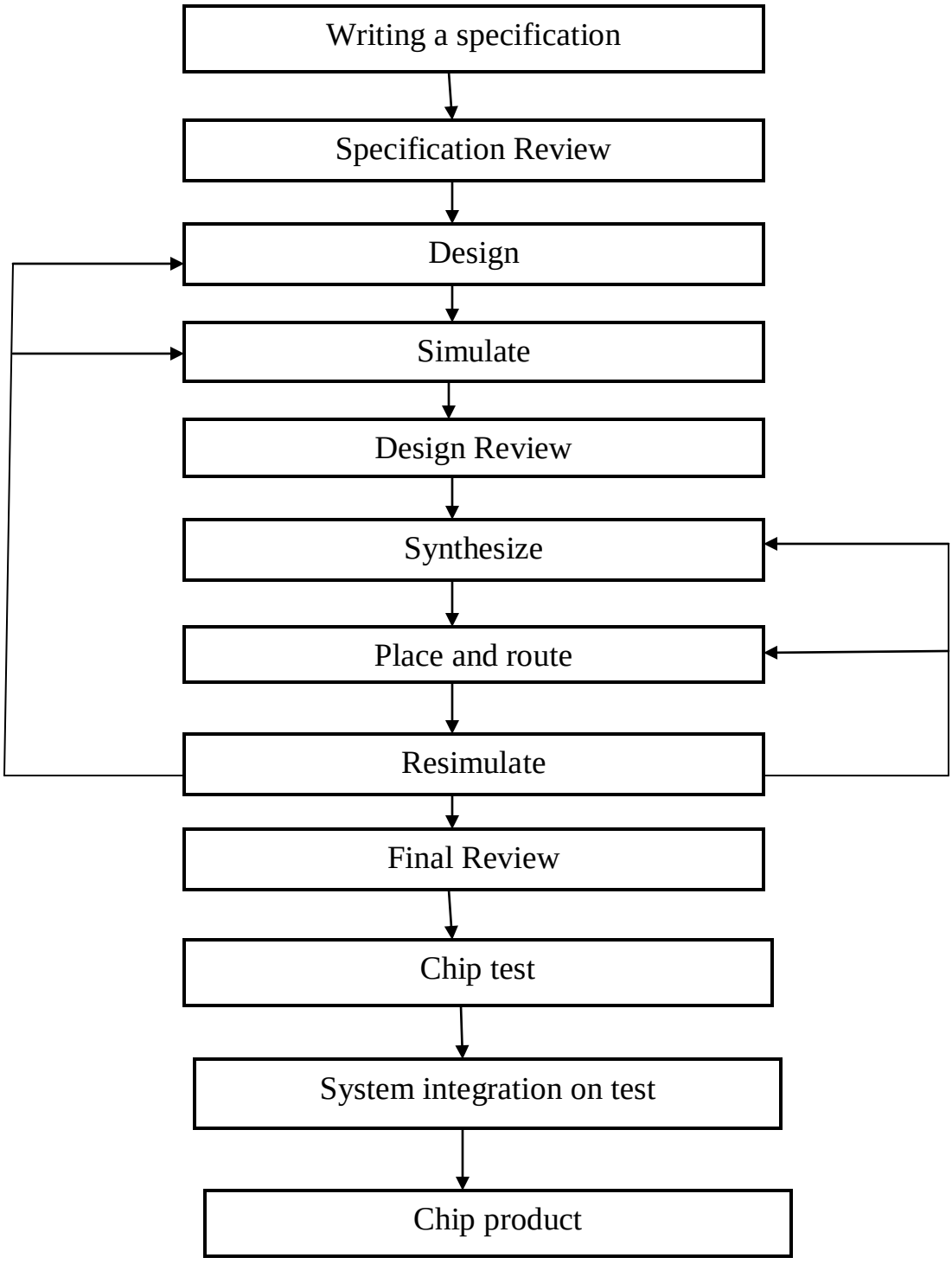


Figure 5.3: Design flow of FPGA

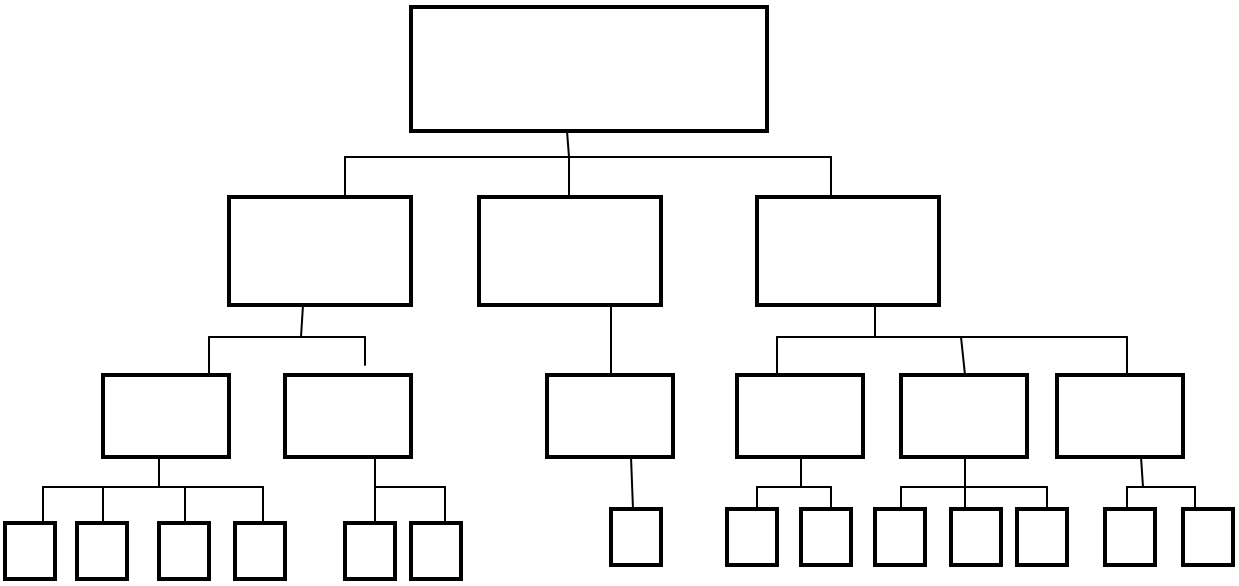


Figure 5.4: Top down design

5.3.6 Simulating – design review

Simulation is the process of applying stimulus or inputs that provide actual data to the design and observing the output. It is used to verify that the design performs the expected and required functions. There will be much iteration of design and simulation in order to get the correct functionality. This is one of the most important reviews because it is only with correct and complete simulation that you will know that your chip will work correctly in your system.

Level of simulation: Three level of simulation –register transfer level (RTL), gate level and functional level.

(1) RTL (register transfer level) performs simulation on the design phase code. Doing this prior to synthesizing allows you to troubleshoot the design for logic and syntax errors. The RTL simulation contains no timing information.

(2) Functional simulation is performed on the netlist or the code generated by the synthesis tool. It is necessary to direct the synthesis tool to generate the functional simulation netlist. The synthesis tool predicts and inserts some timing information; these are not the final timing delays. This simulation is more realistic than the RTL, but not as accurate when it comes to timing as the gate-level simulation.

(3) Gate level simulation is performed on the code or netlist generated by the implementation tool.

It may be necessary to direct the implementation tool to generate the gate-level simulation. This simulation contains actual timing information and is the most realistic representation of the FPGA design.

5.3.7 Synthesis

Design synthesis or synthesis is the process that takes the high-level design associates it with FPGA resource and reduces logic to make the design more efficient. After the correct simulation results the design is then synthesized. During synthesis the Xilinx ISE tool does the following operation.

(i) HDL Compilation: The tool require all the sub-modules in the main module if having any problem regarding module then check the syntax of the code written for the design.

(ii) Design Hierarchy Analysis: Analysis the hierarchy of the design.

(iii) HDL synthesis: Synthesizes the HDL code in terms of the components such as multiplier, adder/subtractor, counter, register, latches, comparators, XORs tri state buffers, decoders etc.

(iv) Advanced HDL synthesis: In low level synthesis, the blocks synthesized in the HDL synthesis and Advanced HDL synthesis are further defined in terms of the low level blocks such as buffers, look up tables. It also optimizes the logic entities in the design by eliminating the redundant logic, if any. The tool then generates a netlist file (NGC file) and then optimizes it. The final netlist output file has an extension of .ngc. This NGC file contains both the design data and constraints. The optimization goal can be pre defined to be the faster speed of operation or the minimum area of implementation before running this process. The level optimization effort requires larger CPU times (i.e. the design time) because multiple optimization algorithm are tried to get the best result for the target architecture.

5.3.8 Design implementation

The design implementation process consists of the following sub process:

(1) Translation: The translate process merges all of the input netlist and design constraint outputs a Xilinx NGD (Native information and Generic Database) file. The .ngd file describes the logical design reduced to the Xilinx device primitive cells. The output in the floor planner tool supplied with Xilinx ISE software. Here, defining constraints is nothing but, assigning the ports in the design to the physical elements (example. Pins, switches, buttons etc) of the targeted device and specifying time requirements of the design. This information is stored in a file named UCF (User Constraints File). Tools used to create or modify the UCF are PACE, constraint Editor etc.

(2) Mapping: The map process is run after the translate process is complete. Mapping maps the logical design described in the NGD file to the components/primitive (slices/CLBs) present on the NCD file created by the Map process to place and route the design on the target FPGA design. In this process the whole circuit is divided into sub blocks so that they can fit into FPGA logic blocks. The logic defined in the input NGD file is mapped into targeted FPGA elements i.e. CLBs and IOBs and an output NCD(native circuit description) file is generated which depicts the design mapped into the FPGA.

(3) Place and Route: After map the design is placed and routed. Place and Route (PAR) program is used for this process. The place and route process place the sub blocks from the map process into logic blocks according to the constraints and connects the logic blocks. Example if a sub block is placed in a logic block which is very near to IO pin, then it may save the time but it may affect some other constraint .So tradeoff between all the constraints is taken account by the place and route process. The PAR tool takes the mapped NCD file as input and produces a completely routed NCD file as output. Output NCD file consist the routing information. During the place process the sub blocks are placed according to the logic but these blocks do not have physical routing among them and with I/O pads also but there is only a logical connection between them which can be clearly seen using the FPGA Editor just after the place process ends. Then the Route process is run which makes physical connections between the sub blocks on FPGA. The connections are made using the switch matrices.

5.3.9 Resimulating- final review

After layout, the chip must be resimulated with the new timing numbers produced by the actual layout. If everything has gone well up to this point, the new simulation results will agree with the predicted results. Otherwise, there are two possible paths to go in the design flow. If the problems encountered here are significant, sections of the FPGA may need to be redesigned. If there are simply some marginal timing paths or the design is slightly larger than the FPGA, it may be necessary to perform another synthesis with better constraints or simply another place and route with better constraints.

5.3.10 Testing

For a programmable device, you simply program the device and immediately have your prototypes. You then have the responsibility to place these prototypes in your system and determine that the entire system actually works correctly. If you have followed the procedure up to this point, chances are very good that your system will perform correctly with only minor problems. These problems can often be worked around by modifying the system or changing the system software. These problems need to be tested and documented so that they can be fixed on the next revision of the chip. System integration and system testing is necessary at this point to insure that all parts of the system work correctly together. When the chips are put into production, it is necessary to have some sort of built-in test of your system that continually tests your system over some long amount of time. If a chip has been designed correctly, it will only fail because of electrical or mechanical problems that will usually show up with this kind of stress testing.

FPGA Overview:

The Xilinx Spartan–3E (XC3S500) FPGA kit is used for implementation of CORDIC Algorithm. The Xilinx ISE 10.1i tool is used for the design. Some specifications of Spartan – 3E kit are as following:

(1) Up to 232 user – I/O pins

(2) Over 10,000 logic cells

(3) 2 – line, 16 – character LCD screen

(4) PS/2 mouse or keyboard port

(5) VGA display port

(6) Two 9 – pin RS – 232 ports (DTE/DCE)

(7) 50 MHz clock oscillator

(8) Chip scope Soft Touch debugging port

(9) Eight discrete LEDs

(10) Four slide switches

(11) Four push-button switches

(12) Speed Grade -4

(13) FG320 package

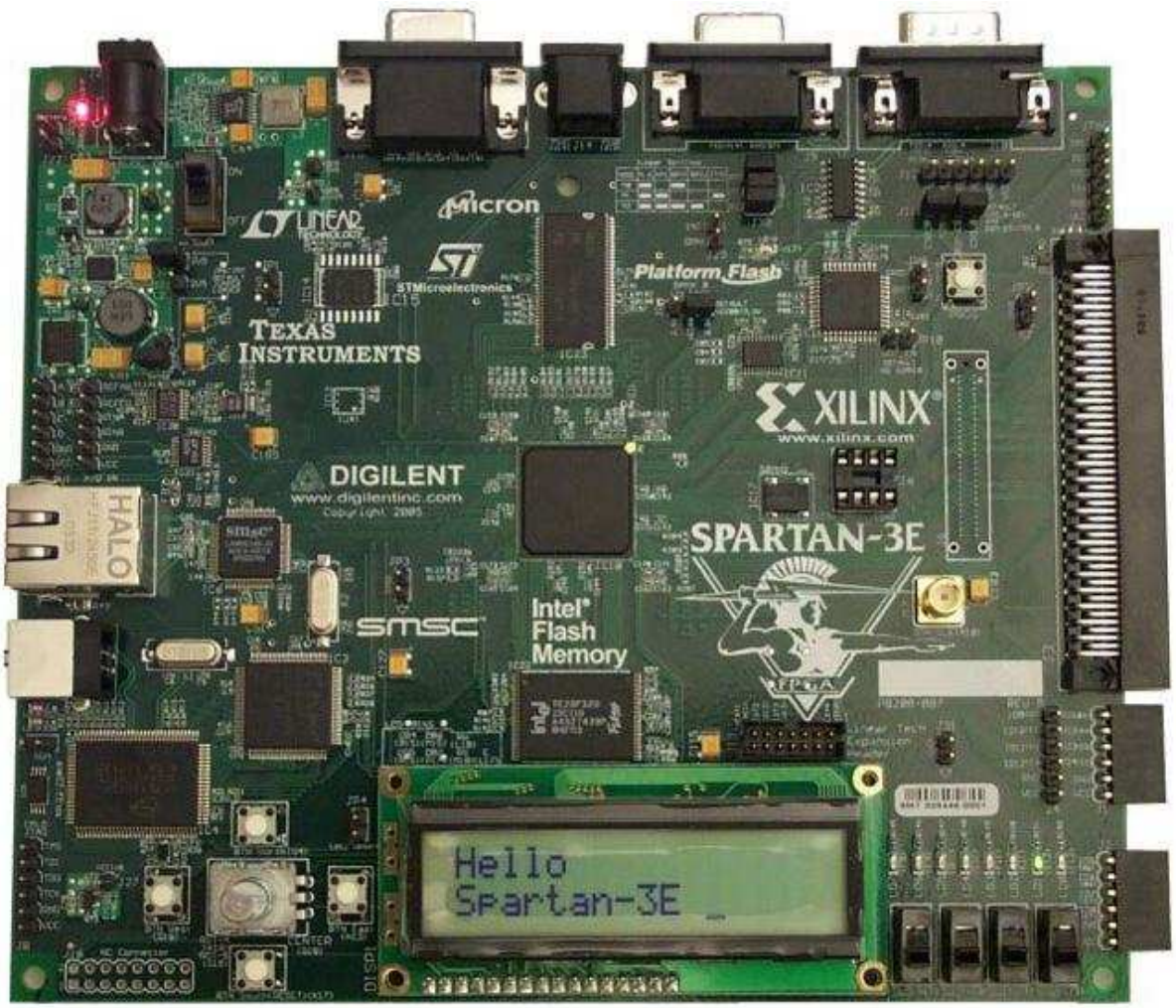


Figure 6.5: FPGA Spartan-3E kit

CHAPTER 6

RESULTS AND DISCUSSIONS

CORDIC algorithm is used to compute sinθ and cosθ by vector rotation method. Figure (6.1), (6.2) consists of Modelsim simulation result for real input and real outputs angle sinθ and cosθ in the form of the waveform for original CORDIC and control CORDIC and their corresponding magnitude respectively.

6.1 Modelsim Simulation Results:

6.1.1 For sine-cosine real input and real output of original CORDIC

$\theta = 30^o$

Sin 30^o =0.50

Cos 30^o=0.8637

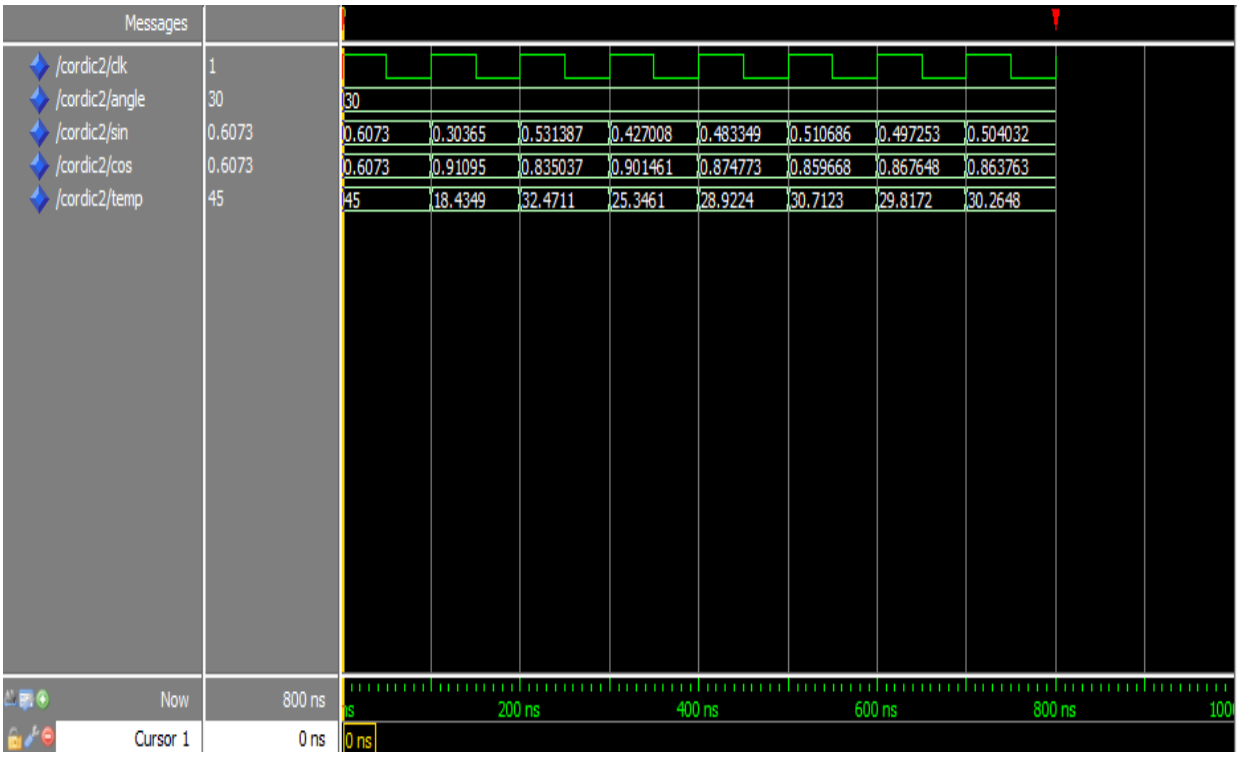


Figure 6.1: For sine and cosine real input and real output θ =30^o of original CORDIC

6.1.2 For sine-cosine real input and real output of control CORDIC

$\theta = 60^o$

Sin 60^o=0.8637

Cos 60^o= 0.50

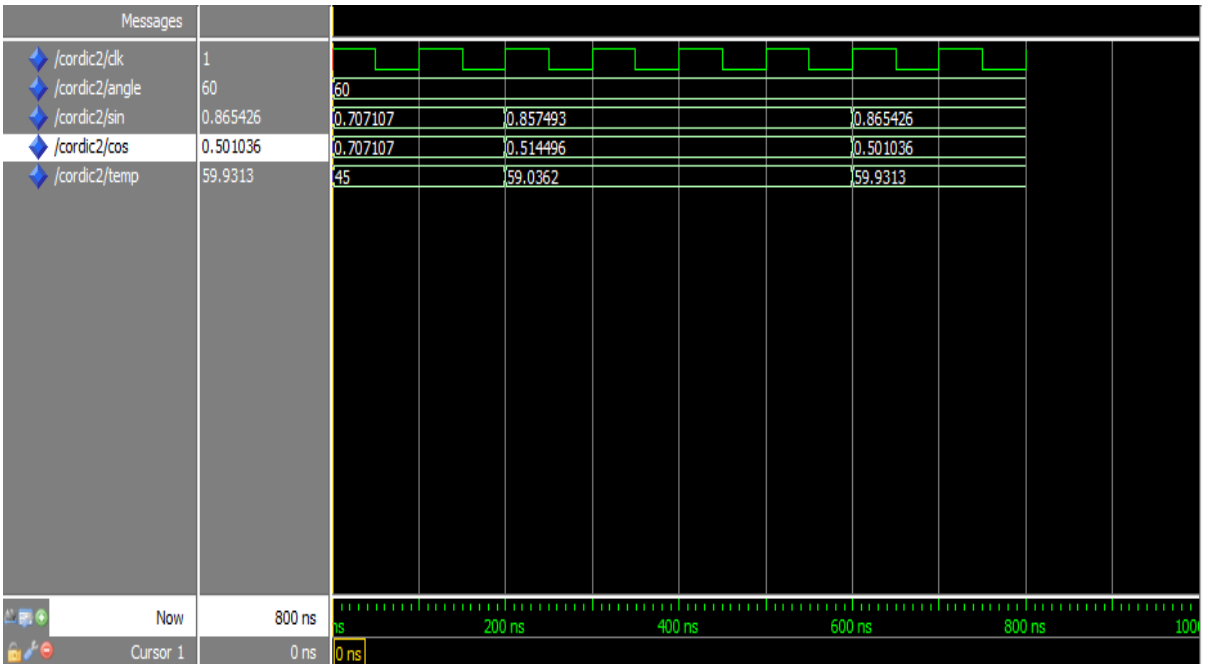


Figure 6.2: For sin and cosine real input and real output $\theta = 60^o$ of control CORDIC

6. 2 Xilinx Simulation results:

6.2.1 Simulation result for original CORDIC:

Block diagram generated by XILINX 10.1i for sine-cosine using CORDIC is shown in figure 6.4. Here inputs are angle (binary input), clk (clock), start, reset and outputs are sine (binary output), cosine (binary output), done. Figure 6.5 shows the RTL schematic of sine-cosine generator and its internal block diagram.

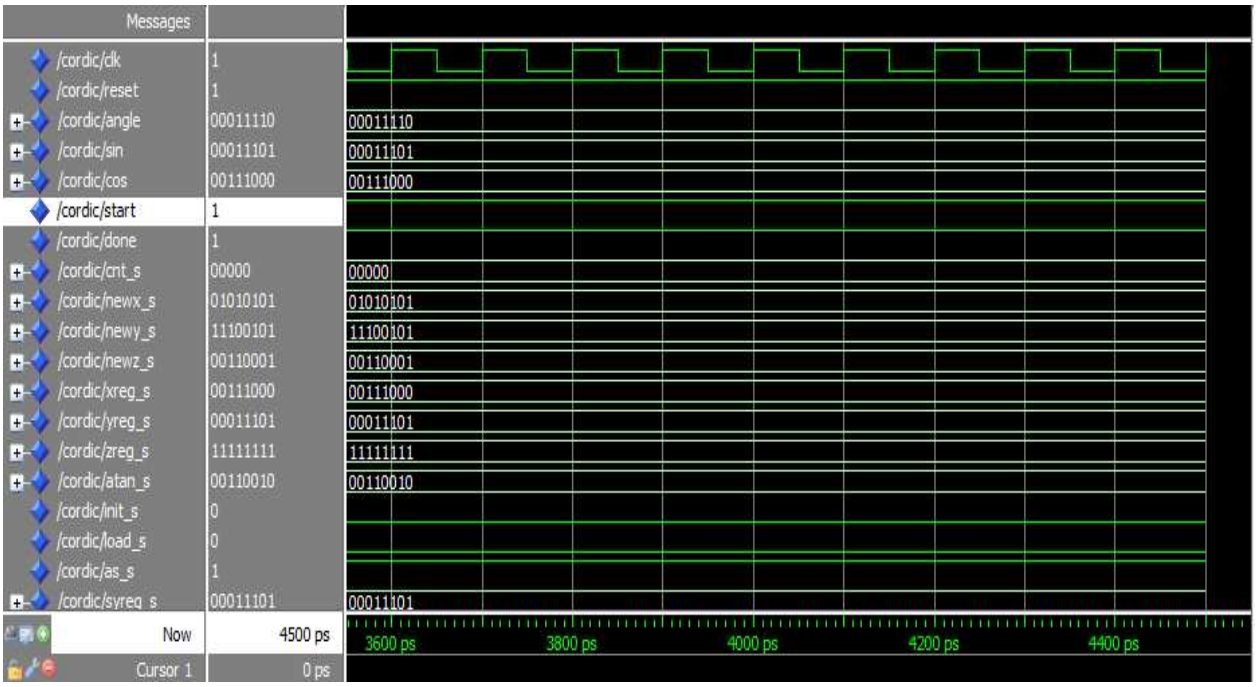


Figure 6.3: For sin and cosine binary input and binary output of original CORDIC for 8 bit

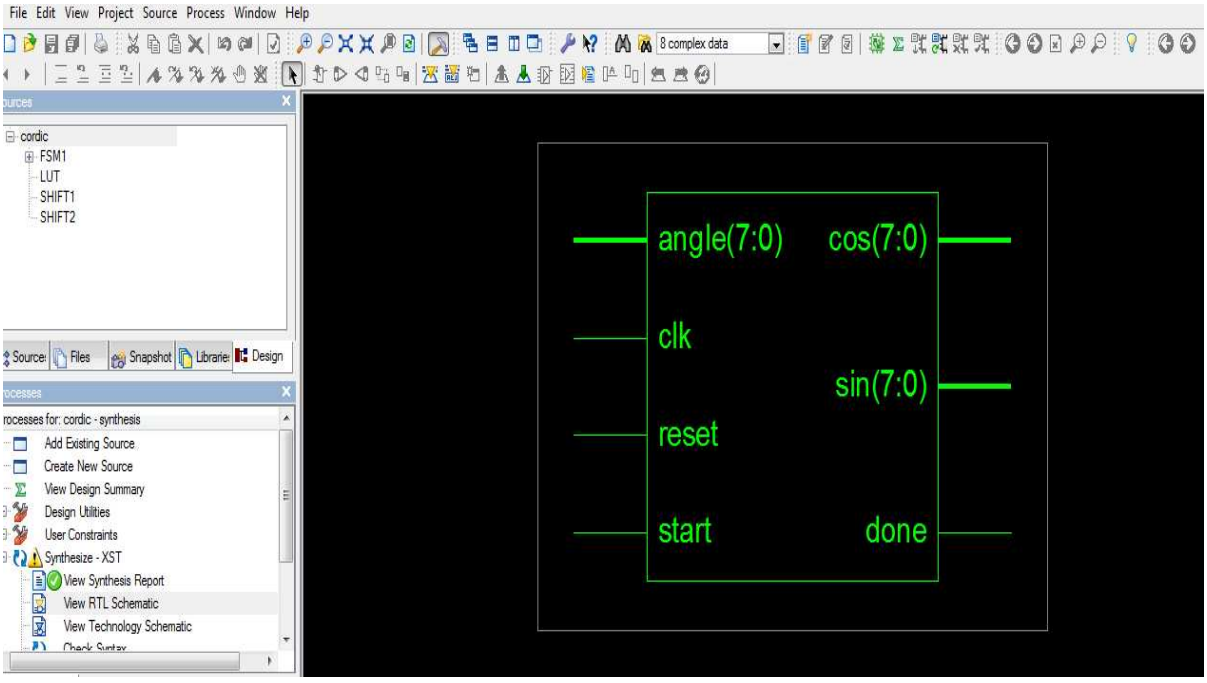


Figure 6.4: Top level RTL schematic for sine-cosine (CORDIC) for 8 bit

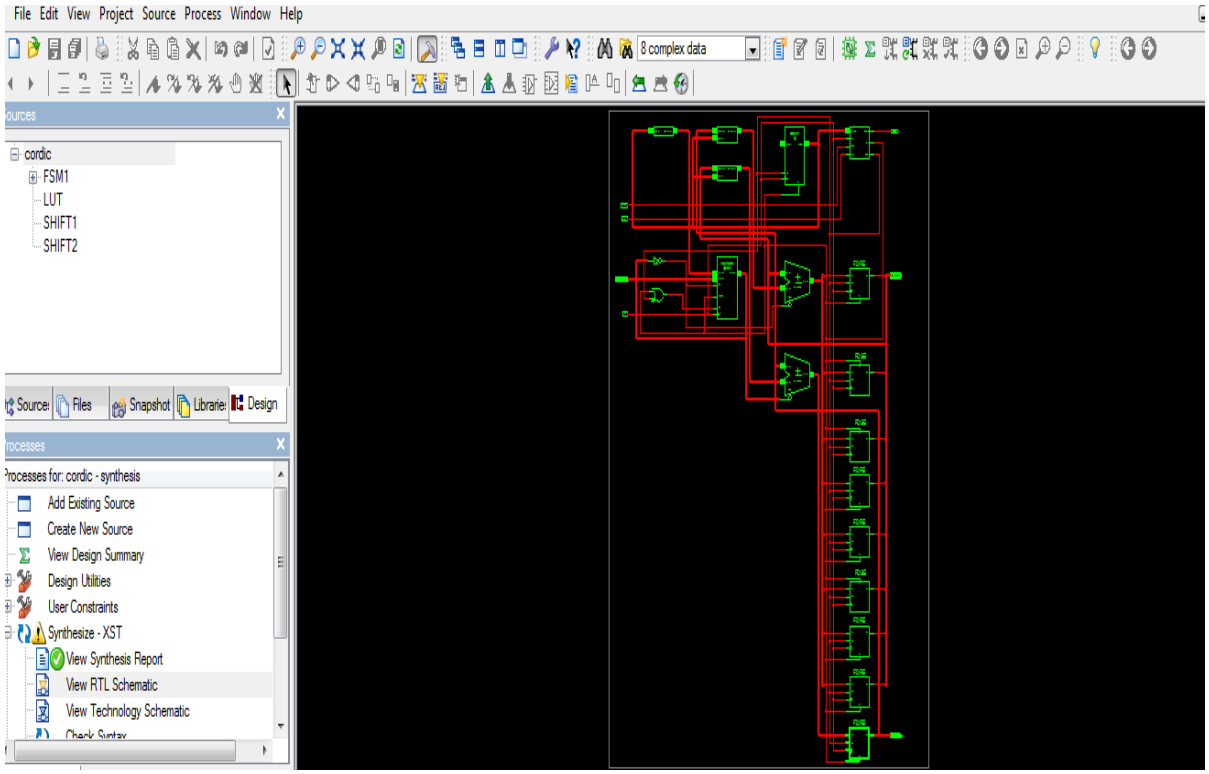


Figure 6.5: Internal RTL schematic for sine-cosine for 8 bit

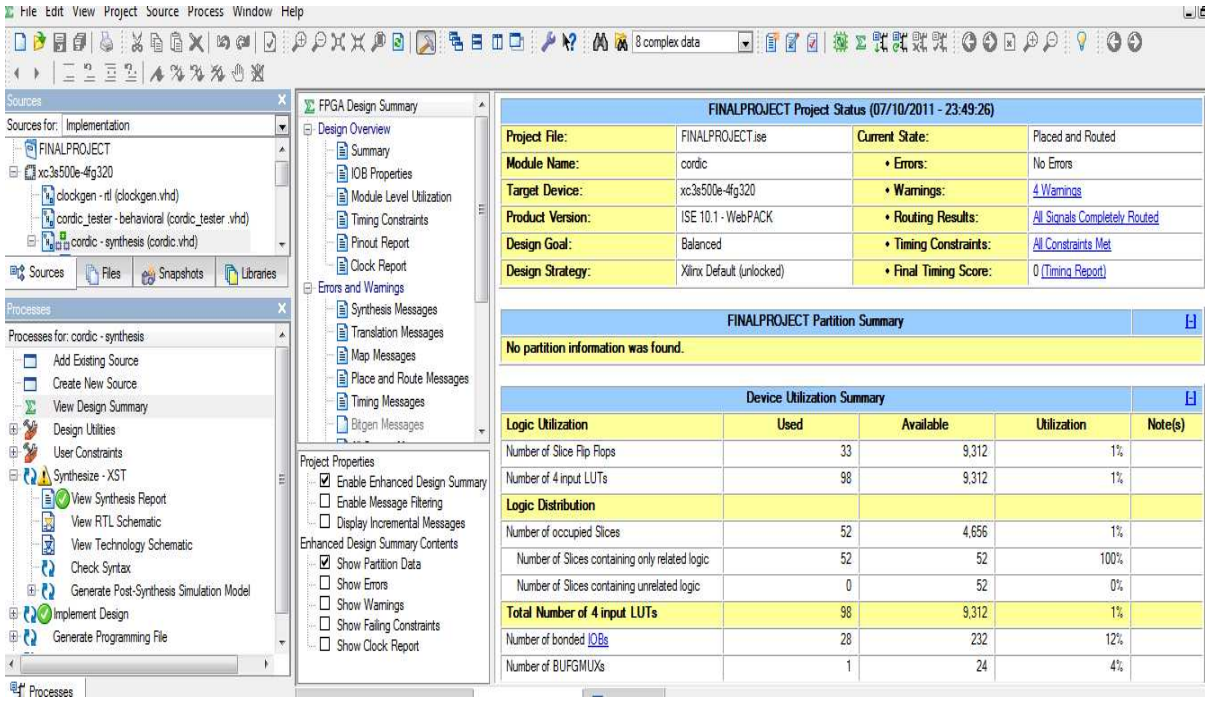


Figure 6.6: View summary report for 8 bit (Original CORDIC)

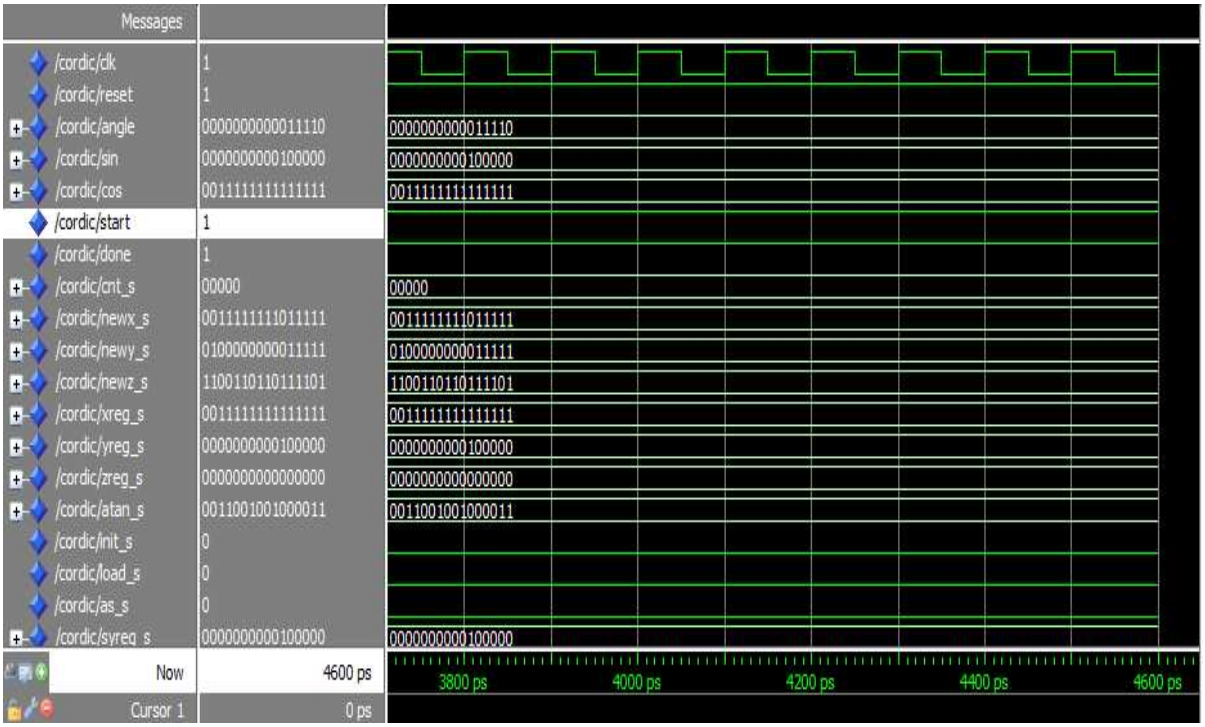


Figure 6.7: For sin and cosine binary input and binary output of original CORDIC for 16 bit

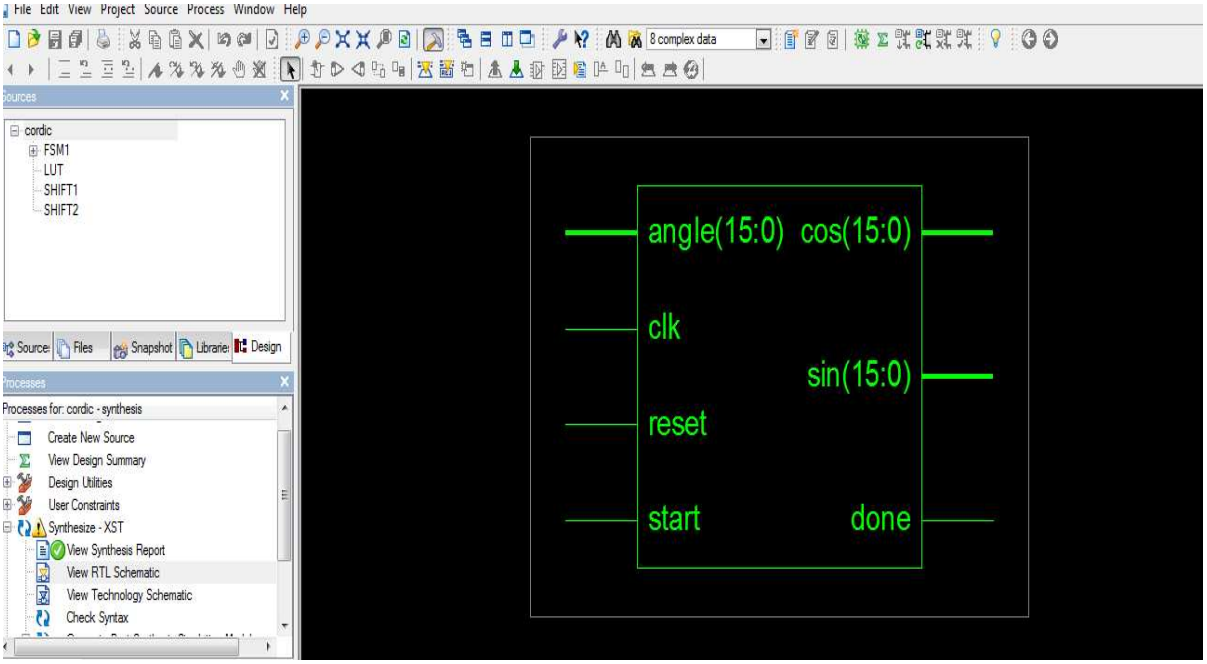


Figure 6.8: Top level RTL schematic for sine-cosine (CORDIC) for 16 bit

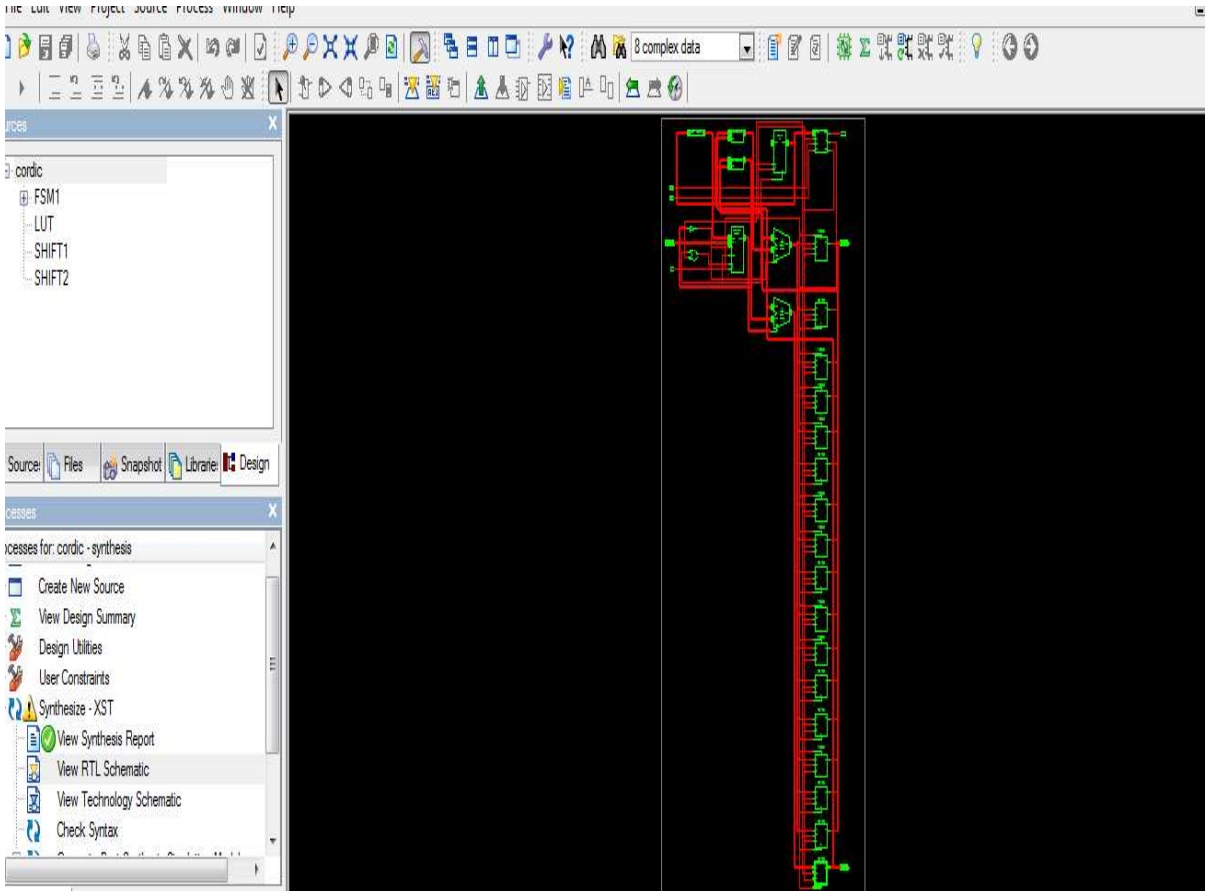


Figure 6.9: Internal RTL schematic for sine-cosine for 16 bit

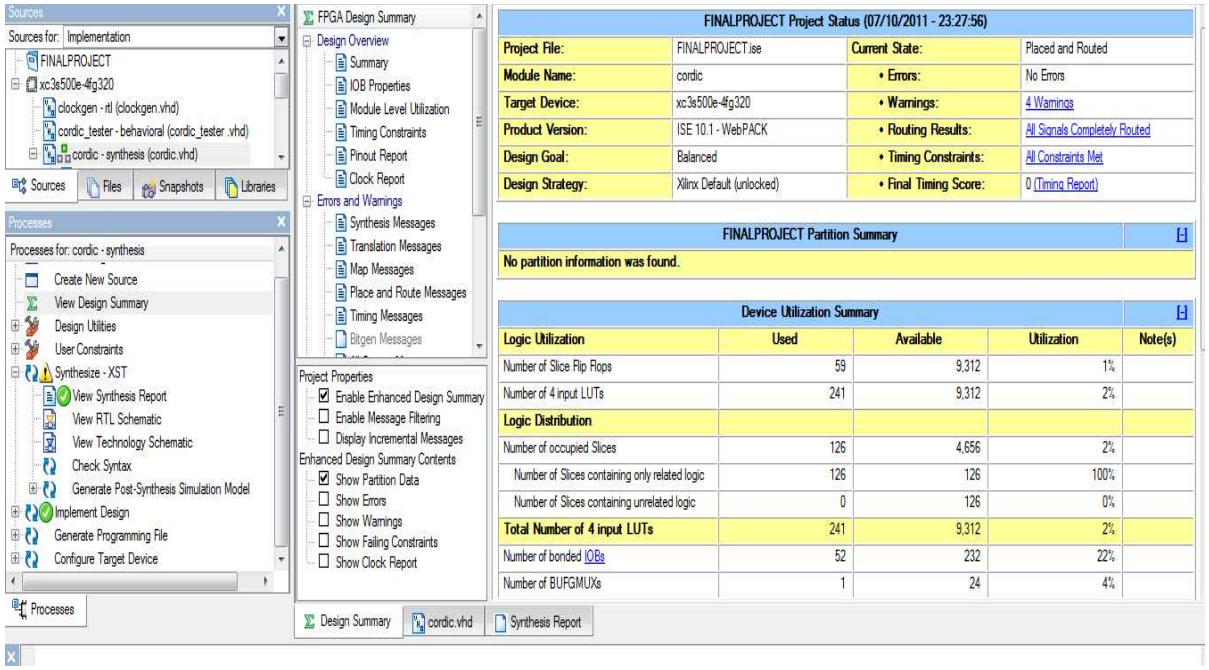


Figure 6.10: View summary report for 16 bit (Original CORDIC)

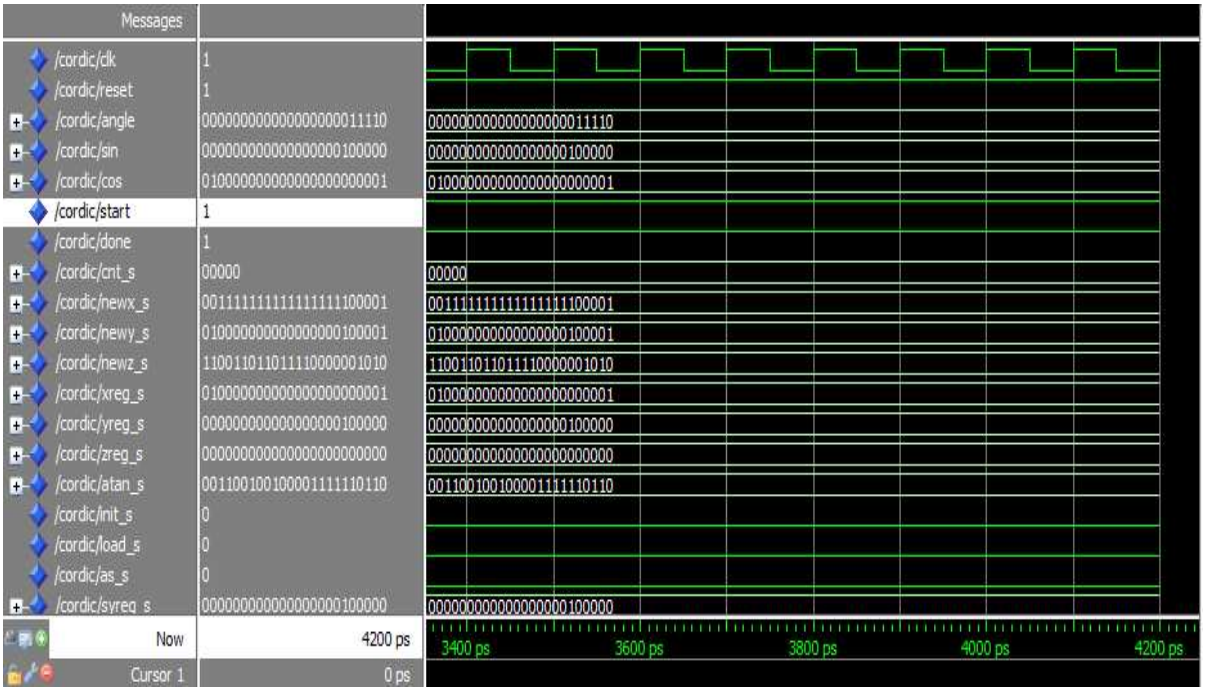


Figure 6.11: For sin and cosine binary input and binary output of original CORDIC for 24 bit

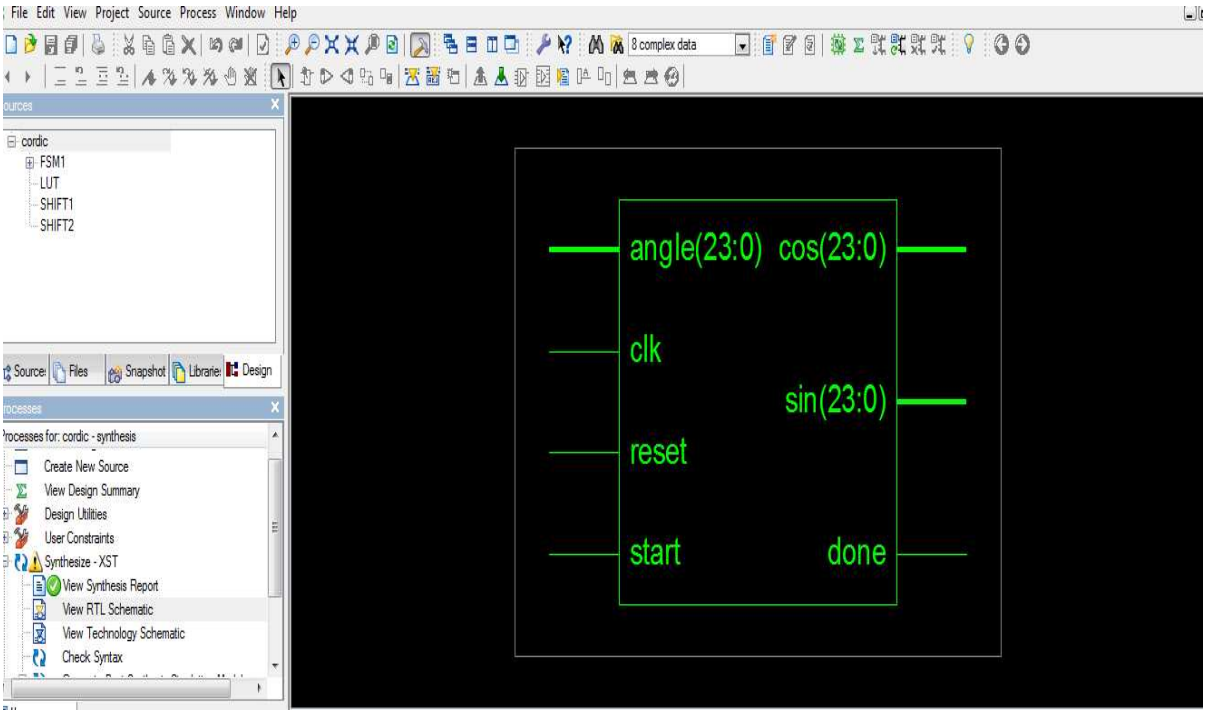


Figure 6.12: Top level RTL schematic for sine-cosine (CORDIC) for 24 bit

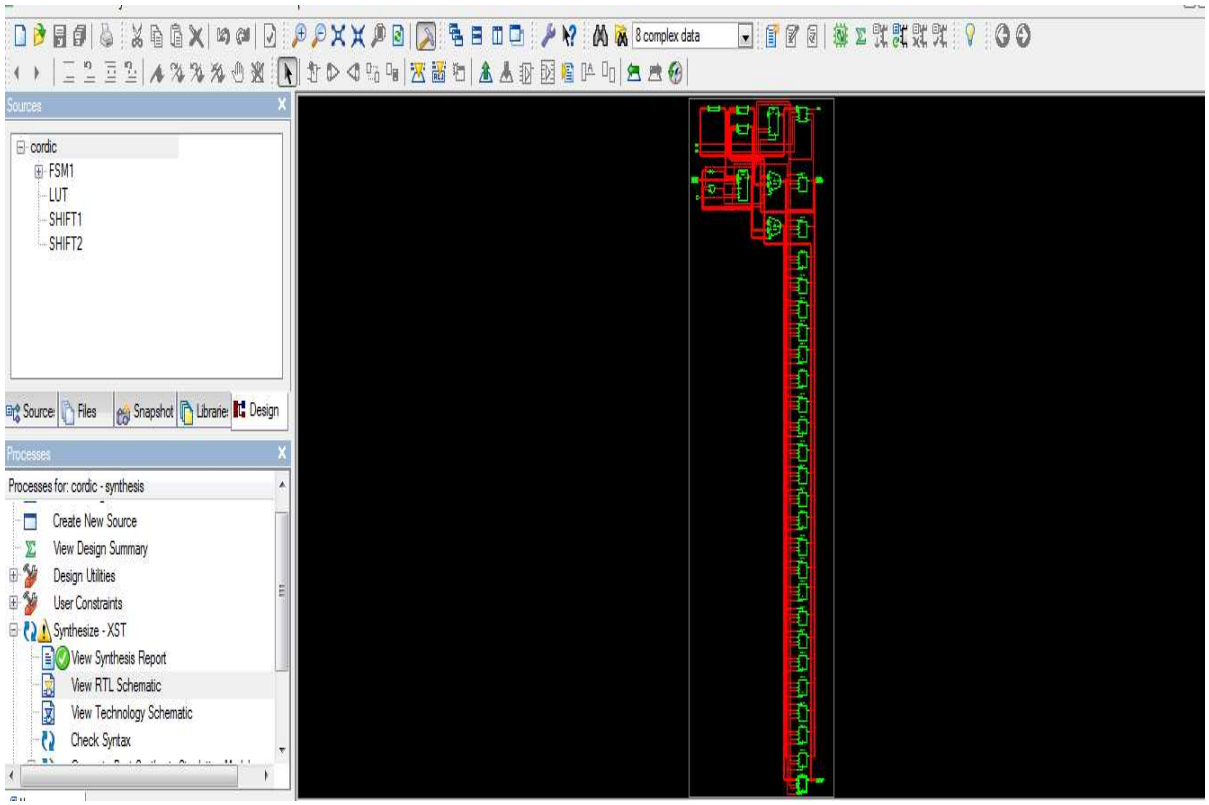


Figure 6.13: Internal RTL schematic for sine-cosine for 24 bit

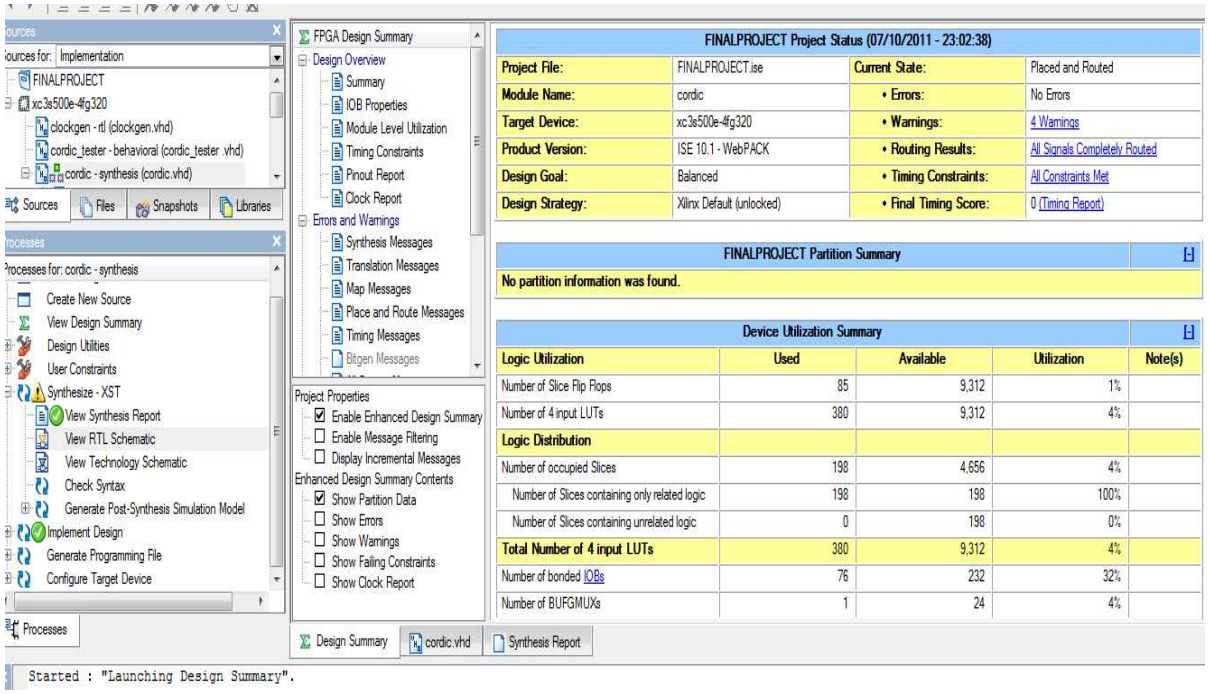


Figure 6.14: View summary report for 24 bit (Original CORDIC)

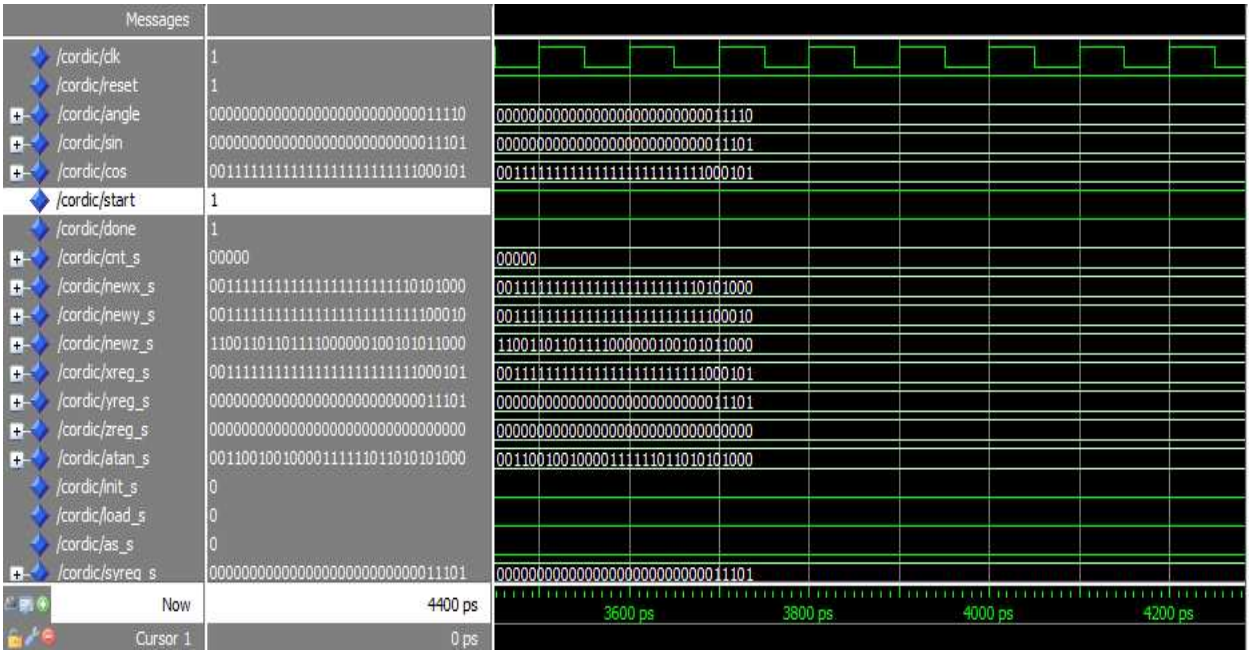


Figure 6.15: For sin and cosine binary input and binary output of original CORDIC for 32 bit

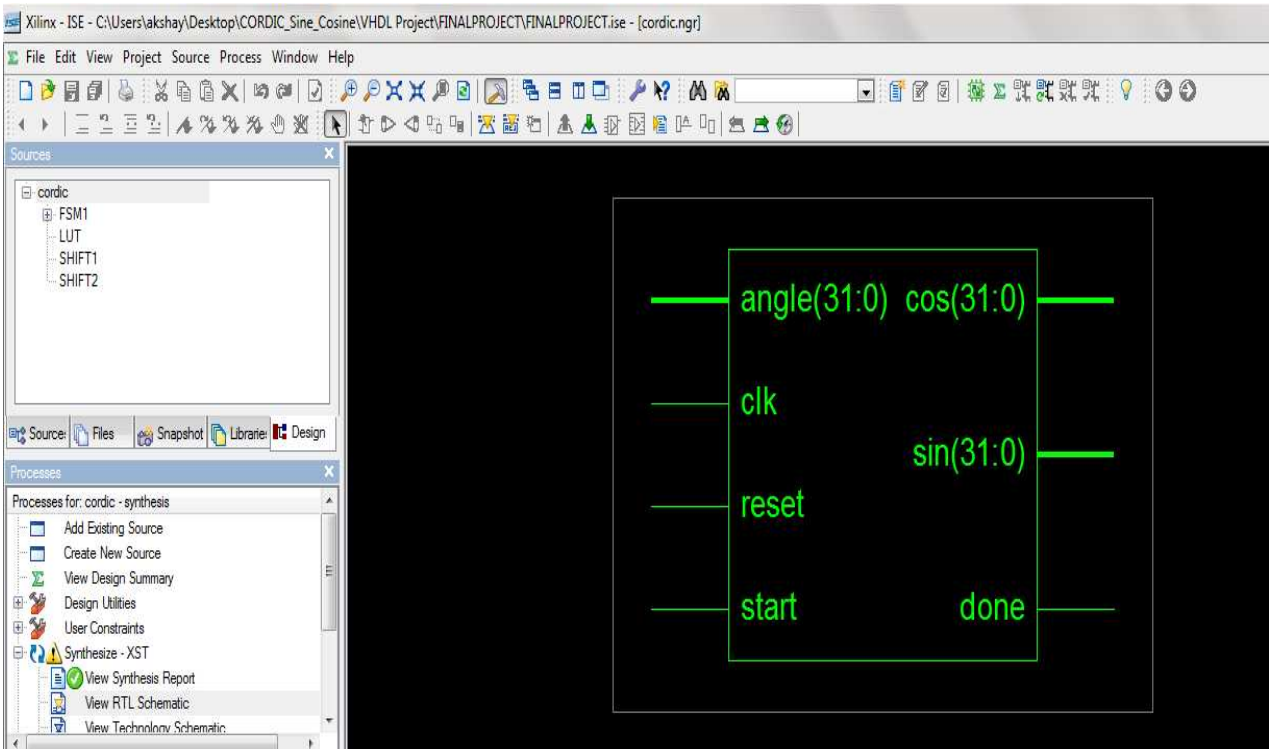


Figure 6.16: Top level RTL schematic for sine-cosine (CORDIC) for 32 bit

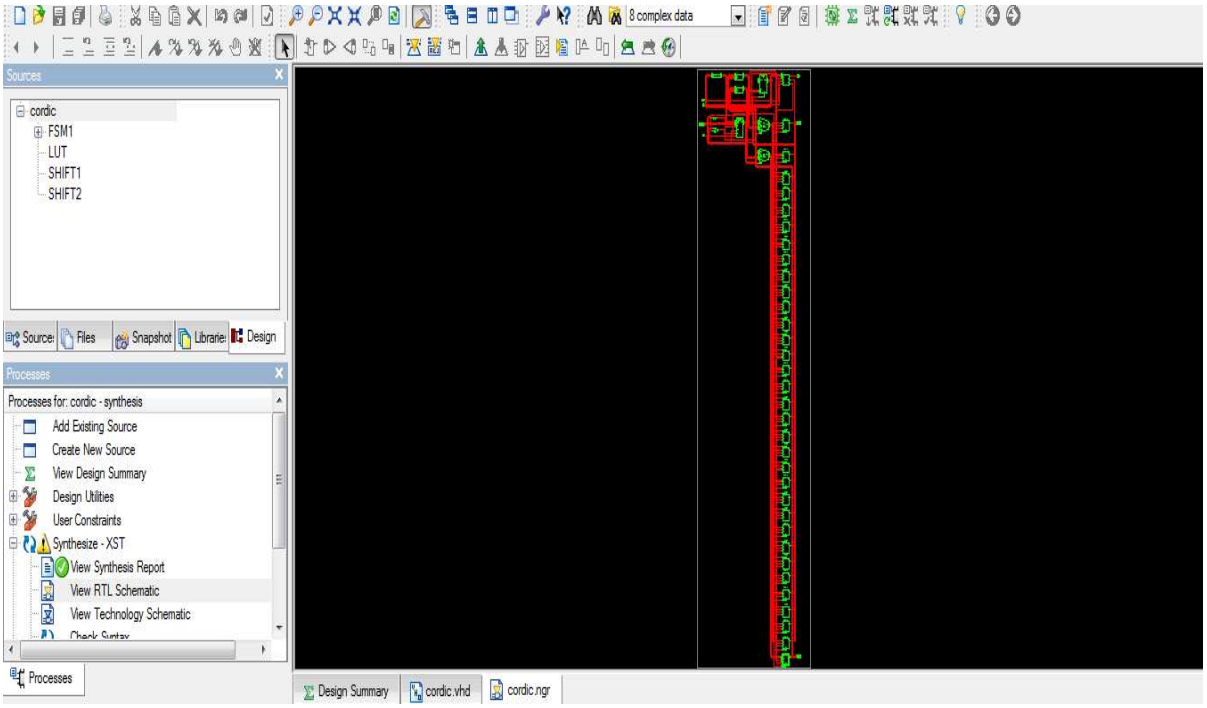


Figure 6.17: Internal RTL schematic for sine-cosine for 32 bit

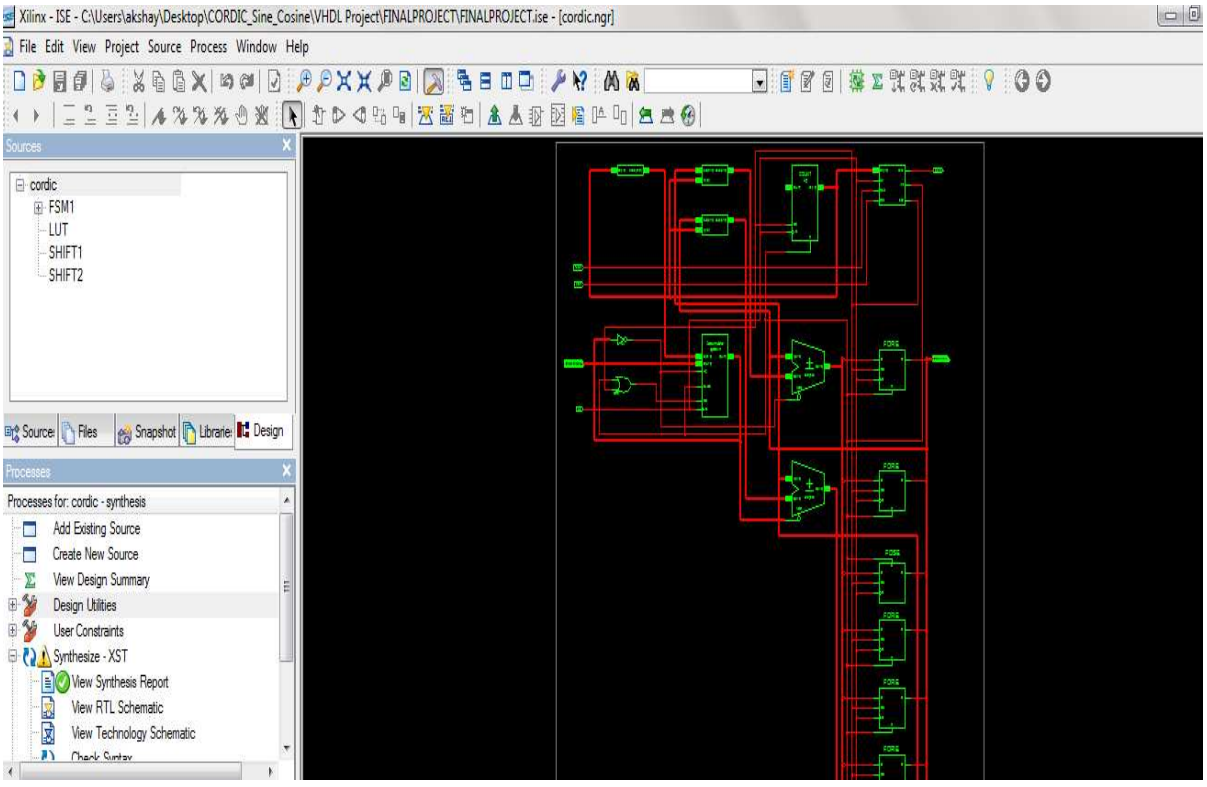
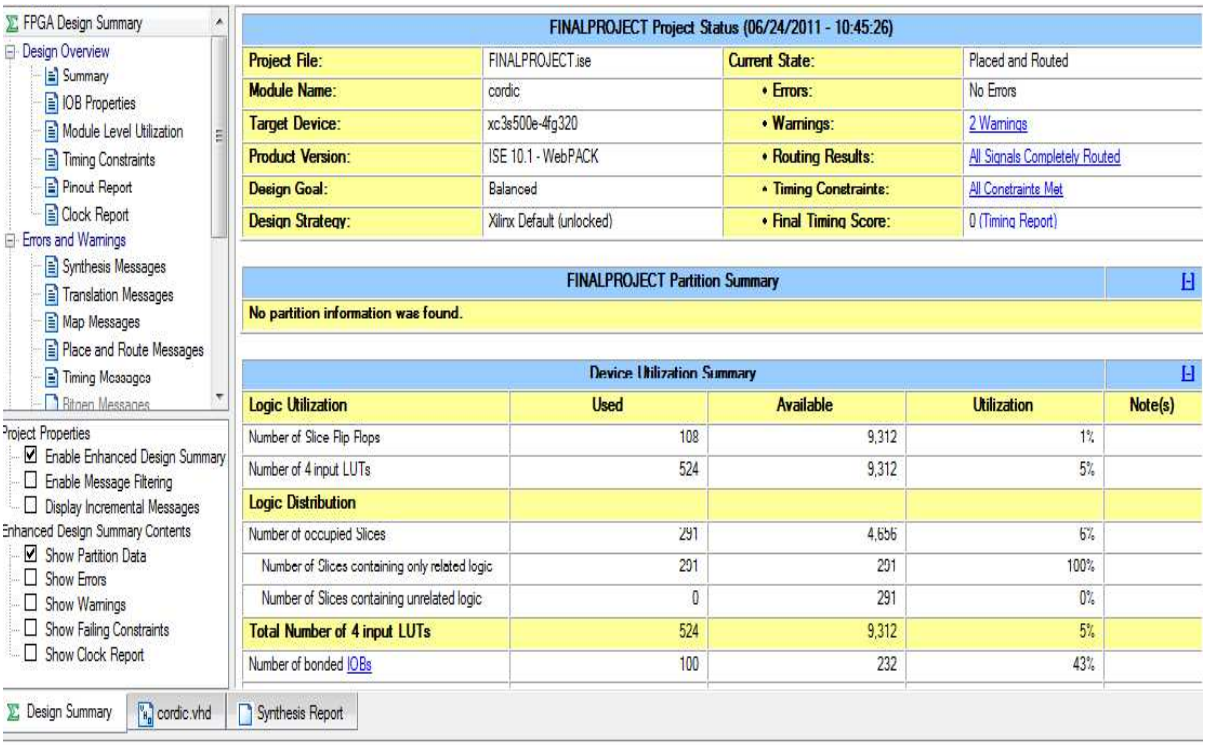


Figure 6.18: RTL schematic for sine-cosine



FINALPROJECT Project Status (06/24/2011 - 10:45:26)

Project File:	FINALPROJECT.ise	Current State:	Placed and Routed
Module Name:	cordic	• Errors:	No Errors
Target Device:	xc3s500e-4fg320	• Warnings:	2 Warnings
Product Version:	ISE 10.1 - WebPACK	• Routing Results:	All Signals Completely Routed
Design Goal:	Balanced	• Timing Constraints:	All Constraints Met
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	0 (Timing Report)

FINALPROJECT Partition Summary

No partition information was found.

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	108	9,312	1%	
Number of 4 input LUTs	524	9,312	5%	
Logic Distribution				
Number of occupied Slices	291	4,656	6%	
Number of Slices containing only related logic	291	291	100%	
Number of Slices containing unrelated logic	0	291	0%	
Total Number of 4 input LUTs	524	9,312	5%	
Number of bonded IOBs	100	232	43%	

Figure 6.19: View summary report for 32 bit (Original CORDIC)

Comparison of original CORDIC using different data rates has been shown in table 6.1

Table 6.1: Comparison between 8, 16, 24, 32 bit of Original CORDIC

	8 bit		16 bit		24 bit		32 bit	
	total	utilization	total	utilization	total	utilization	total	utilization
No. of slices	52/4656	1%	127/4656	2%	198/4656	4%	284/4656	6%
No. of slices flip-flops	33/9312	0%	59/9312	0%	85/9312	0%	108/9312	1%
No. of 4 input LUTs	98/9312	1%	241/9312	2%	383/9312	4%	526/9312	5%
No. of bonde-d IOBs	28/232	12%	52/232	22%	76/232	32%	100/232	43%
No. of clks	1/24	4%	1/24	4%	1/24	4%	1/24	4%
Power	99 mw		114 mw		129 mw		136mw	
Delay	7.645 ns		8.785 ns		9.798 ns		10.490 ns	

FPGA device Package: Xc3s500e-4fg320

Look-up tables are by far the fastest way to make the computation; however the precision of the result is directly related to size of the look-up table. High precision look-up tables require a large amount of non-volatile memory to store the table. If the table size is reduced to save memory, precision will also be reduced. Resolution of CORDIC algorithm is best for 32 bit data rates but it causes in more complexity of components and reduction in data speed.

6.2.2 Simulation results for pipeline CORDIC

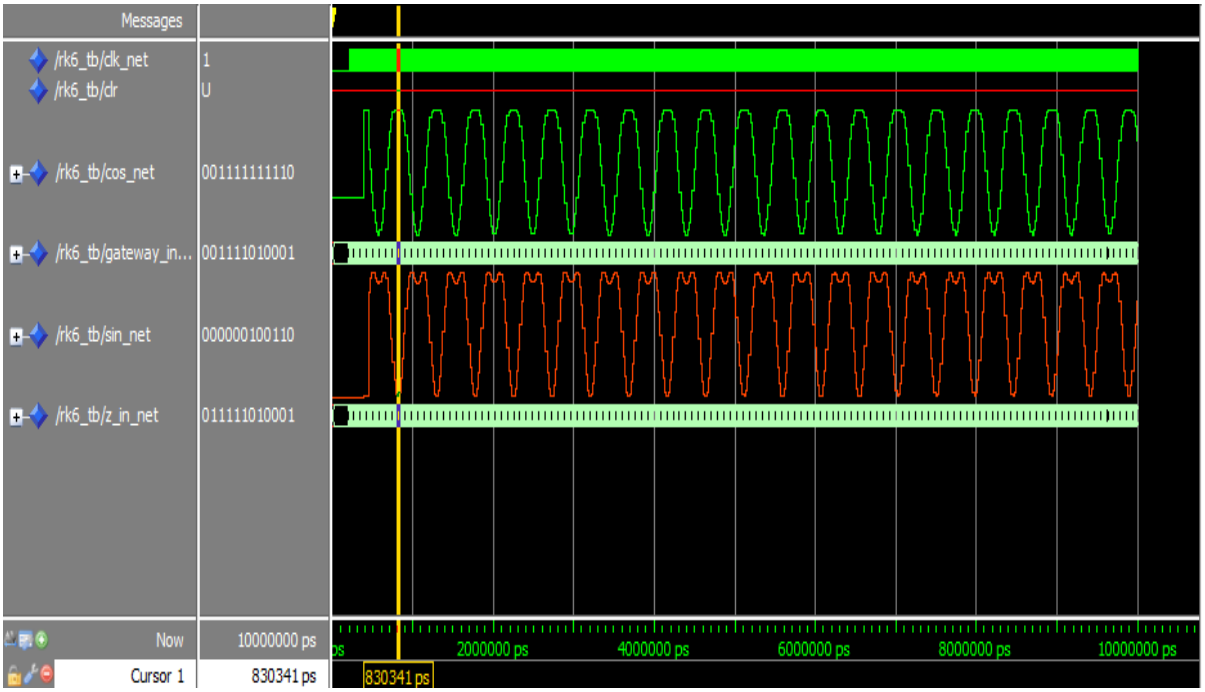


Figure 6.20: Sine and Cosine waveform using Modelsim simulation

Table 6.2: Comparison between 12 bit Pipeline CORDIC and Original CORDIC

	Pipeline CORDIC	Original CORDIC
Number of slices	196/4656 = 4%	93/4656=1%
Number of slice Flip-flops	321/9312 = 3%	49/9312=0%
Number of 4 Input LUTs	361/9312 = 3%	178/9312=1%
Number of Bonded IOBs	37/232 = 15%	40/232=17%
Number of GCLKs	1/24 = 4%	1/24=4%
Delay	5.326ns	8.191ns
Power	83mW	89mW

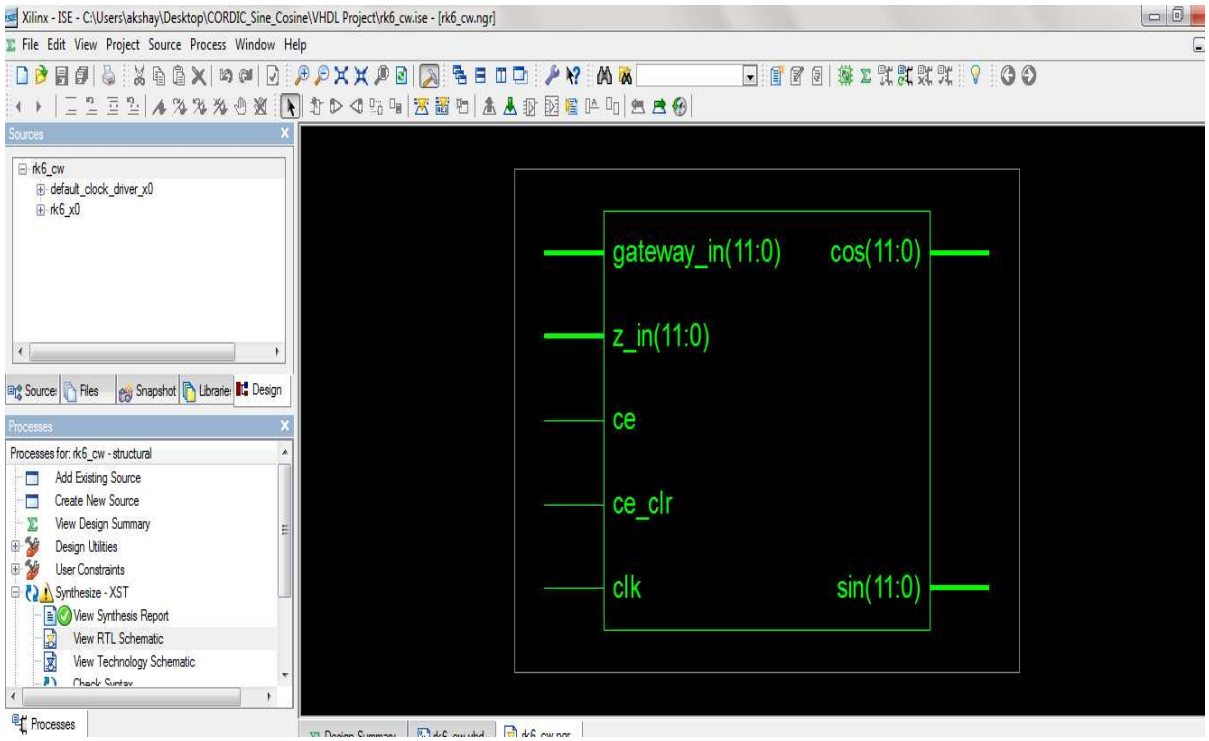


Figure 6.21: Top level RTL schematic for Pipeline CORDIC

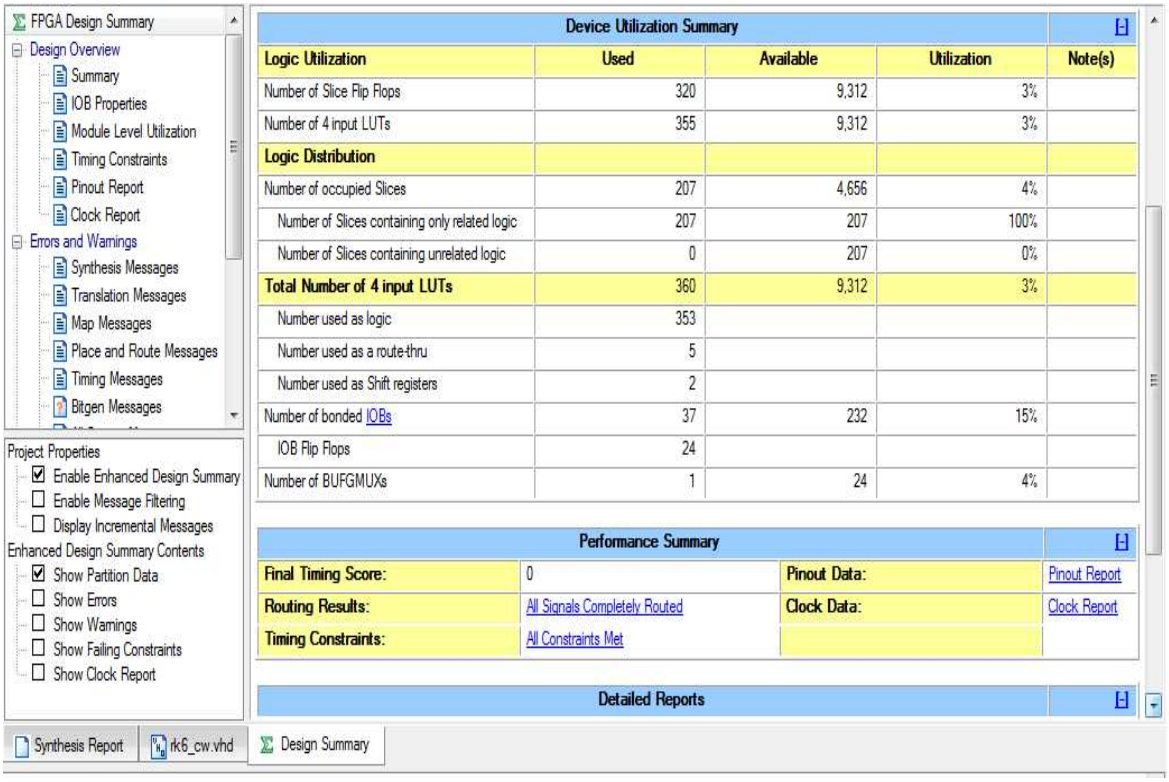


Figure 6.22: View summary report for 12 bit Pipeline CORDIC

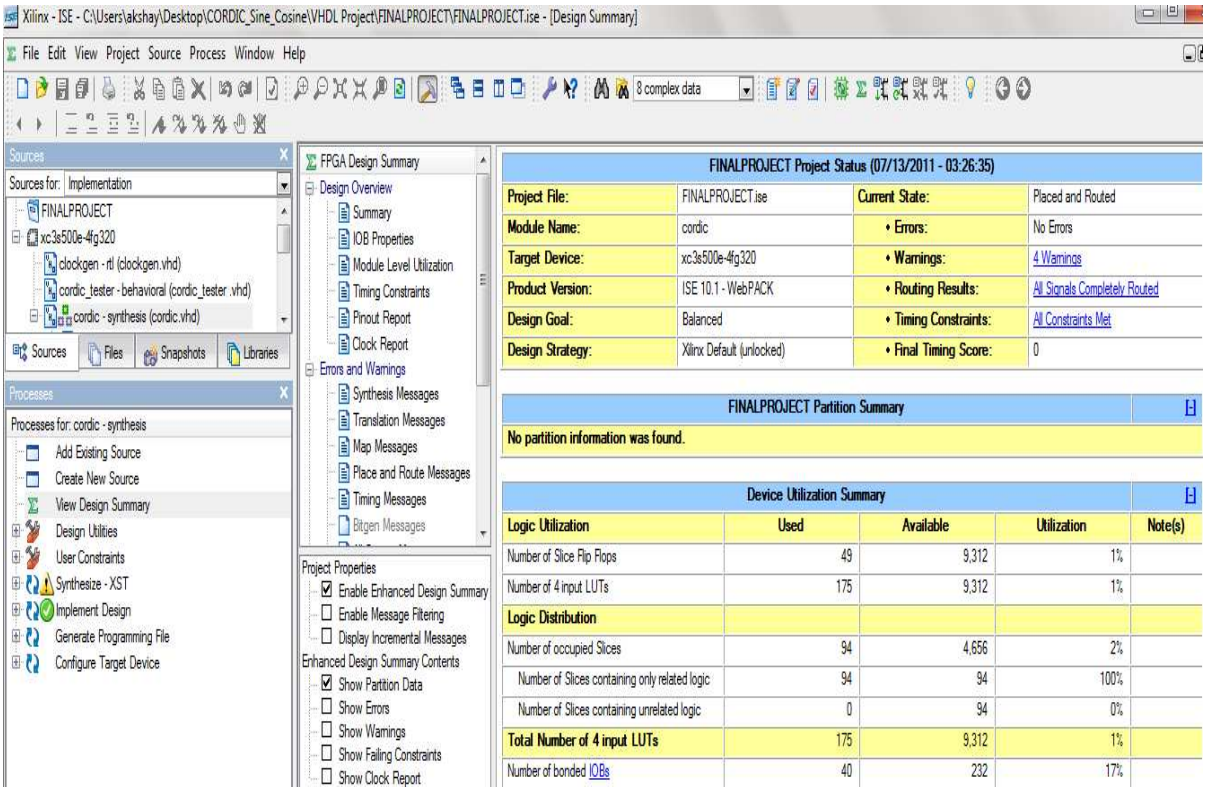


Figure 6.23: View summary report for 12 bit original CORDIC

Table 6.3 and Table 6.4 have synthesis report generated by Xilinx10.1 showing number of multiplexers, number of adders, number of subtractors, and number of flip-flops used.

Table 6.3: Original CORDIC:-HDL Synthesis Report

ROMs		1	Registers		3
	32×32 bit ROM	1		32 bit register	3
Adders/ Subtractors		3	Logic Shifters		2
	32 bit add sub	3		32 bit shifter arithmetic right	2
Counters		1			
	5 bit up counter	1			

Table 6.4: Pipeline CORDIC:-HDL Synthesis Report

Adders/ Subtractors		26
	12 bit Adder	2
	13 bit add sub	24
Registers		449
	Flip-Flops	449
Comparators		2
	12 bit comparator greater	1
	12 bit comparator less	1

6.2.3 Simulation result for Discrete Fourier Transform (DFT)

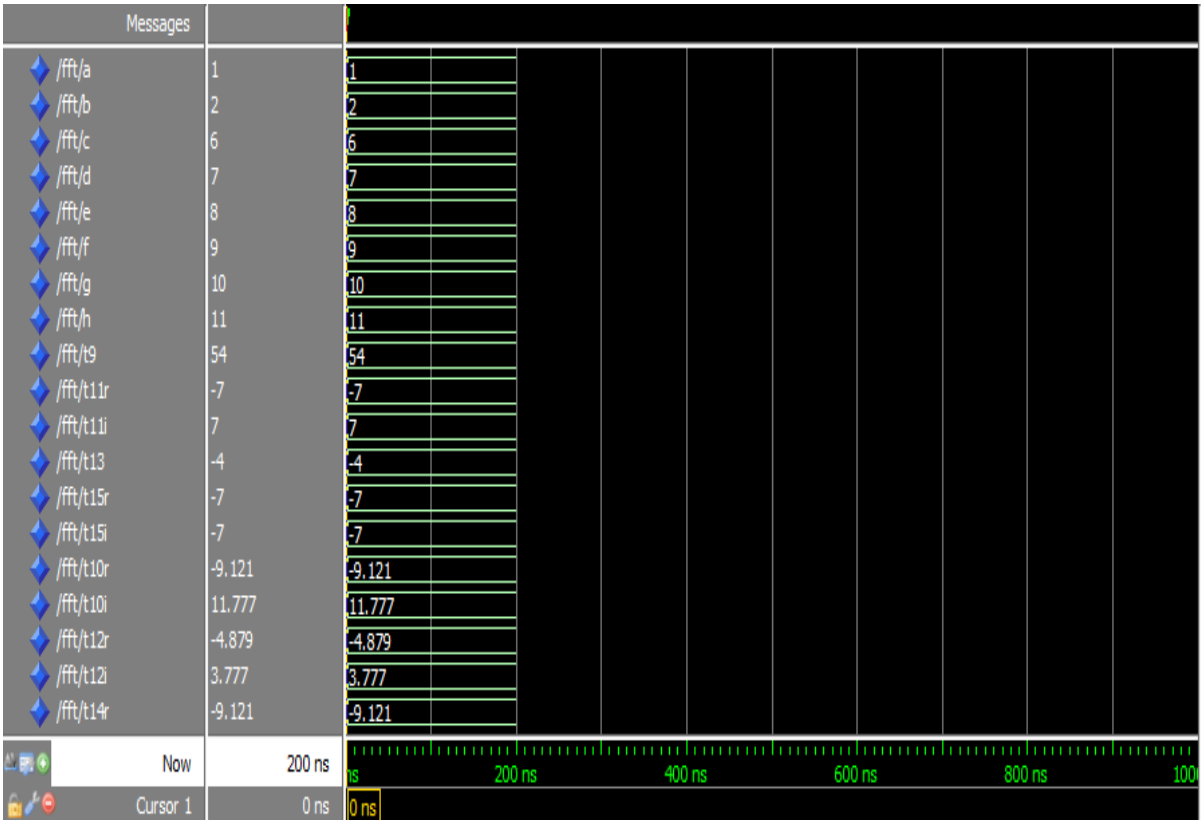


Figure 6.24: Real input/output waveforms of DFT using FFT algorithm

Table 6.5: Real input/output values of DFT using FFT algorithm

Input	Real part	Imaginary part
1	54	0
2	-9.121	11.777
6	-7	7
7	-4.879	3.777
8	-4	0
9	-9.121	-3.777
10	-7	-7
11	-9.121	-11.777

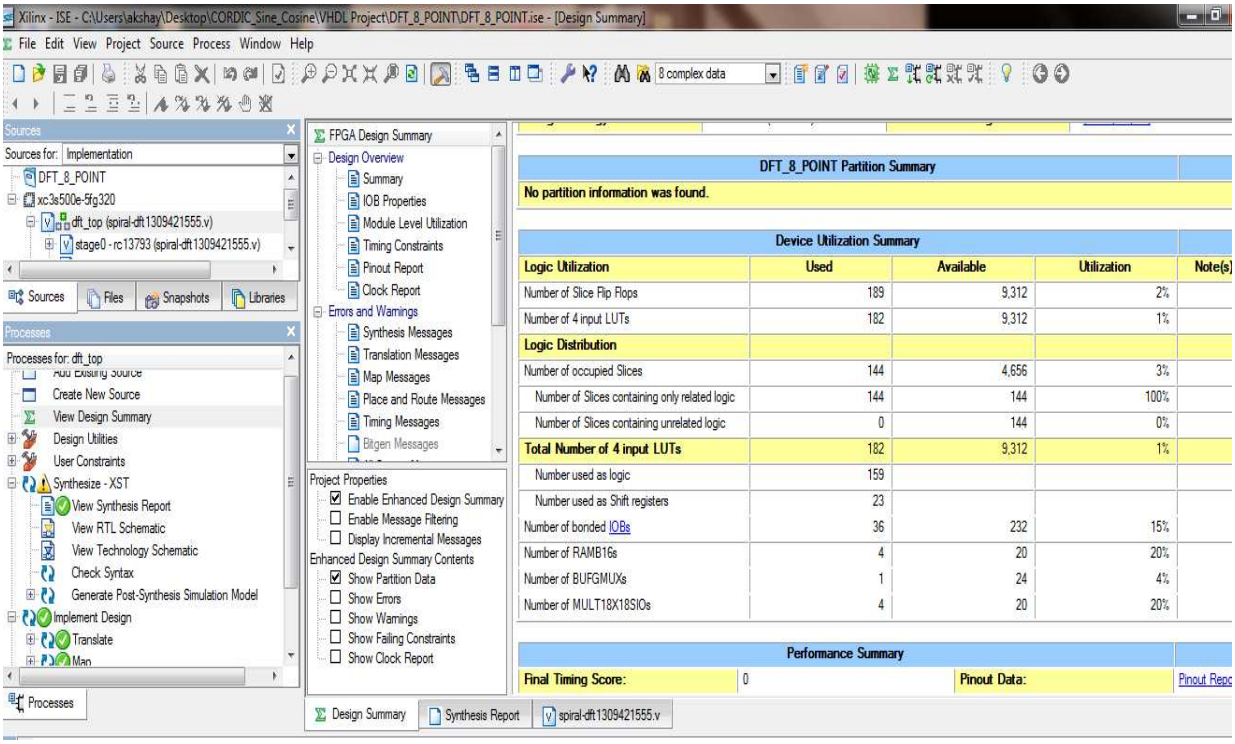


Figure 6.25: View summary report for DFT

Table 6.6: DFT:-HDL Synthesis Report

RAMs		4	Registers		121
	8x8-bit dual-port RAM	4		1-bit Register	32
ROMs		2		2-bit Register	1
	4x4-bit ROM	2		3-bit Register	4
Multipliers		4		4-bit Register	67
	4x4-bit multiplier	4		5-bit Register	1
Adders/ Subtractors		11		8-bit Register	16
	3-bit adder	1	Comparators		5
	3-bit subtractor	1		2-bit comparator less	2
	4-bit adder	4		3-bit comparator greater	1
	4-bit subtractor	4		3-bit comparator less	1
	5-bit adder	1		5-bit comparator less	1
Counters		7	Logic Shifters		1
	2-bit up counter	7		2 bit shifter logical left	1

6.3 Discussions

No multipliers used for implementation of CORDIC algorithm for sine cosine generation. Number of adders/subtractors and registers are 3 and 3 respectively. So CORDIC algorithm is multiplierless approach and it saves a lot of hardware. In case of DFT implementation number of multiplier, adders/subtractors, registers, comparators are 4, 6/5, 121, 5 respectively.

CHAPTER 7
CONCLUSION

In this thesis, comparative analysis of CORDIC algorithm has been implemented for 8 bit, 16 bit, 24 bit and 32 bit on the basis of their area, power and speed required to implement in chip designing, number of iterations required etc. The sine cosine CORDIC based generator, control CORDIC, pipeline CORDIC and DFT (Discrete Fourier Transform) have been simulated using Modelsim. Then the implementation of sine cosine CORDIC based generators, pipeline CORDIC and DFT have been done on Xilinx Spartan 3E FPGA. The area required has been measured in terms of slices, flip-flops, LUTs and IOBs. The percentage utilization for number of slices used have been found to be equal to 1%, 2%, 4% and 6% for 8-bit, 16 bit, 24 bit and 32 bit respectively. Also, in case of 8 bit, 16 bit, 24 bit and 32 bit, numbers of flip-flops required are 33, 59, 85 and 108 respectively. Number of LUTs has been calculated to be equal to 98, 241, 383 and 526 for 8 bit, 16 bit, 24 bit and 32 bit respectively. The number of IOBs has been found to be equal to 28, 52, 76 and 100 for 8 bit, 16 bit, 24 bit and 32 bit respectively. As the number of slices, IOBs and flip-flops increases, the complexity of hardware also increases.

Look-up tables are by far the fastest way to make the computation; however the precision of the result is directly related to size of the look-up table. High precision look-up tables require a large amount of non-volatile memory to store the table. If the table size is reduced to save memory, precision will also be reduced. Resolution of CORDIC algorithm is best for 32 bit data rates however it offers higher complexity as compared to lower data rate CORDIC algorithm. Also, it has been found that power dissipation increases with increase in data rate. In case of 8 bit, 16 bit, 24 bit and 32 bit, power dissipation has been found to be equal to 99mw, 114mw, 129mw and 136mw respectively. Speed has been calculated to be equal to 130.80MHz, 113.83MHz, 102.06MHz and 95.33MHz for 8 bit, 16 bit, 24 bit and 32 bit respectively. It has been found that 3 adders/subtractors and 3 registers are required and no multiplier is used in CORDIC algorithm, so it saves lot of hardware.

In control CORDIC, the latency of original CORDIC is reduced by 50% due to reduction in the number of iterations. Also, Control CORDIC shows no overshoot as compared to original CORDIC.

Original CORDIC take several clock cycles to build a single output sample of the wave. Pipeline converts iterations into pipeline phases and output is obtained at every clock cycle, after pipeline stages propagation. Pipeline architectures have been used in CORDIC algorithm to reduce the critical path by introducing pipeline latches, increases the clock speed and reduces the power consumption. In this work, number of resources has been reduced by 7.5% and delay has also been reduced by 34.97% using pipeline CORDIC as compared to original CORDIC. In addition to it, power dissipation has also been reduced by 6.7% as compared to Original CORDIC.

The application of CORDIC has been shown using Discrete Fourier transforms (FFT). It has been found that in DFT implementation 4 multiplier, 6/5 adders/subtractors, 121 registers and 5 comparators are required.

REFERENCES

[1] J. E. Volder, "The CORDIC trigonometric computing technique", Institute of Radio Engineers Transactions on Electronic Computers, Vol. EC-8, pp 330 - 334, 1959.

[2] J. Duprat and J. M. Muller, "The CORDIC Algorithm: New Results for Fast VLSI Implementation", IEEE Transactions on Computers, Vol. 42, No. 2, pp.168-178, 1993.

[3] P. K. Meher, J. Valls, T. B. Juang, K.Sridhan and K. Maharatna, "50 Years of CORDIC: Algorithms, Architectures, and Applications", IEEE Transactions on Circuits and Systems, Vol. 56, No.9, pp.1893-1907, 2009.

[4] M. Pascale, "Using CORDIC methods for Computations in micro-controllers", Dr. Dobbs The World of Software Development, No. 9, 2000.

[5] W. Han, Z. Yousi, L. Xiaokang, "A Parallel Double-Step CORDIC Algorithm for Digital down Converter", 7th Annual Communications Networks and Services Research Conference, No. 5, pp. 257-261, 2009.

[6] R. Andraka, "A survey of CORDIC algorithm for FPGA based computers", 6th International Symposium on Field Programmable Gate Arrays, No. 2, pp. 191-200, 1998.

[7] DSP Guru, http://www.dspguru.com/info/faqs/CORDIC.htm.

[8] T. K. Rodrigues and E. E. Swartzlander , "Adaptive CORDIC : Using parallel Angle Recoding to Accelerate Rotations", IEEE Transactions on Computers, Vol. 59, No.4, pp. 522-531, 2010.

[9] E. O. Garcia, R. Cumplido and M. Arias, "Pipelined CORDIC Design on FPGA for a digital sine and cosines waves Generator", 3rd IEEE International Conference on Electrical and Electronics Engineering, No.9, pp. 1-4, 2006.

[10] J.S. Walther, "Unified algorithm for elementary functions", Spring Joint Computer Conference, No.5, pp. 379 - 385, 1971.

[11] Y. H. Hu, "CORDIC-Based VLSI Architectures for Digital signal processing", IEEE Signal Processing Magazine, Vol. 9, No. 3, pp.16-35, 1992.

[12] J. A. Lee, K.J. Vander Kolk and E. F.A. Deprettere, "A low-cost floating point Vectoring Algorithm based on CORDIC", Institute of Electronics , Information and Communication Engineers Transactions Fundamentals, Vol. E83-A, No. 8, pp.1654-1662, 2000.

[13] M. Ercegovac and L. Tomas, "Redundant and On-Line CORDIC: Application to

LIST OF PUBLICATIONS

1. Deepika Ghai, Dr. Kulbir Singh, "VLSI Implementation of CORDIC Algorithm", Canadian Journal Signal Processing, Approved for publication.