# CORDIC COPROCESSOR

PRELIMINARY PROJECT REPORT

Submitted by

**ABHISHEK K**
**TCR18EC002**
**ASHWIN RAJESH**
**TCR18EC019**
**AKIN MARY**
**TCR18EC007**
**HARITH MANOJ**
**TCR18EC028**

to

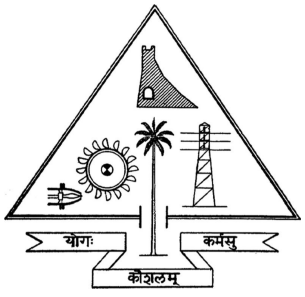
The APJ Abdul Kalam Technological University

of

Bachelor of Technology

In

*Electronics and Communication Engineering*




**Department of Electronics and Communication Engineering**

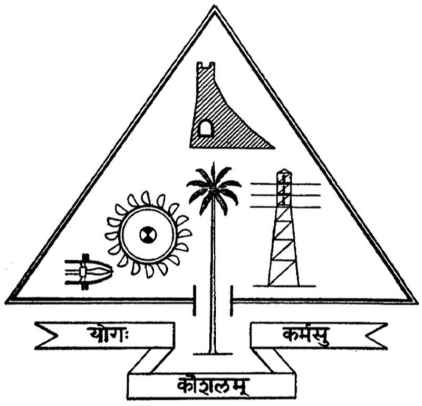
Government Engineering College

Thrissur

January 2022

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**GOVERNMENT ENGINEERING COLLEGE, THRISSUR**




# CERTIFICATE

This is to certify that the report entitled **'CORDIC coprocessor'** submitted by **Abhishek K, Ashwin Rajesh, Akin Mary, Harith Manoj** to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Electronics and Communication Engineering is a bonafide record of the project work carried out by them under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

**Internal Supervisor**

Prof. Mohammed Salih K.K
Assistant Professor,
Dept of Electronics and Communication Engineering,
Government Engineering College, Thrissur

**Seminar Coordinators**

Dr. Mohanan KP
Assistant Professor,
Dept of Electronics and Communication Engineering,
Government Engineering College, Thrissur

**Head of the department**

Dr. Jayan AR
Professor and head,
Dept of Electronics and Communication Engineering,
Government Engineering College, Thrissur

Prof. Sunny TD
Assistant Professor,
Dept of Electronics and Communication Engineering,
Government Engineering College, Thrissur

# DEPARTMENT VISION

To become a nationally acclaimed Department of higher learning and research that will serve as a source of knowledge and expertise in Electronics & Communication Engineering.

# DEPARTMENT MISSION

To provide quality education in the area of Electronics and Communication Engineering, for producing innovative and ethically driven professionals adept at dealing with a globally competitive environment, for the welfare of the nation.

To inculcate inquisitiveness in young graduates thereby persuading them to undertake research in emerging areas of Electronics and Communication Engineering.

# ACKNOWLEDGMENT

We offer our sincere gratitude and appreciation to all the faculty at the Department of Electronics and Communication Engineering, Government Engineering College, Thirssur for their support and guidance throughout this work.

We are thankful to our internal supervisor **Prof. Mohammed Salih K. K** ,Assistant professor, Department of Electronics and Communication engineering, Government engineering college, Thrissur, for his help, advice and guidance. We are thankful to the project coordinators **Dr. Mohanan KP**, Assistant Professor,Department of Electronics and Communication Engineering, Government Engineering College, Thrissur, and **Prof. Sunny TD**, Assistant Professor,Department of Electronics and Communication Engineering, Government Engineering College, Thrissur, for their full fledged support, advice and timely coordination.

We sincerely thank **Dr. A. R. Jayan**, Head Of Department, Department of Electronics and Communication Engineering,Govt.Engineering College,Thrissur, for his guidance, advice, support and suggestions. We are also indebted to all the teaching and non-teaching staff at the Department of Electronics & Communication Engineering,Govt.Engineering College,Thrissur, for their timely assistance in matters pertaining to this seminar work.

Finally, we would like to acknowledge my deep sense of gratitude to all our well-wishers , friends and classmates who helped me directly and indirectly to complete this work.

**Abhishek K**

**Akin Mary**

**Ashwin Rajesh**

**Harith Manoj**

# ABSTRACT

Transcendental functions are functions that cannot be represented by finite polynomials. Typically, they are calculated in computers using taylor series approximation, LUTs or other approximations. This is not fast enough for many use cases. Most modern processors have hardware for computing popular transcendental functions like sinusoids, exponential functions and logarithmic functions.

CORDIC is a very versatile algorithm that can be used for computing such functions. It has three modes, circular, hyperbolic and linear and

Our work aims to develop a CORDIC coprocessor that will be able to compute these functions and that can be interfaced by a main processor using standard bus protocols. It is meant to be used by SoC designers for designing low-power SoCs that need to compute such functions without a floating point unit. It will be able to use the multiple modes of CORDIC and will have some degree of parameterization for customization of the IP by an SoC designer.

We will also explore the trade-offs associated with the various hardware versions of CORDIC, namely the iterative, pipelined, unrolled models.

# CONTENTS

# LIST OF FIGURES



# LIST OF EQUATIONS

# ABBREVIATIONS

| | |
|---|---|
| ASIC | Application Specific Integrated Circuit |
| AXI | Advanced eXtensible Interface |
| CORDIC | COrdinate Rotation DIgital Computer |
| FPGA | Field Programmable Gate Array |
| HDL | Hardware Description Language |
| RTL | Register Transfer Level |
| SoC | System On a Chip |
| GPL | GNU General Public License |

# CHAPTER 1

# OBJECTIVE AND RELEVANCE

Transcendental functions are functions that cannot be computed using finite polynomial equations. This includes the trigonometric, hyperbolic, exponential and logarithmic functions among others. These functions are very important in engineering and computing. Consequently, there are a large number of algorithms that are used to compute such functions. This includes finite polynomial approximations using the taylor series or rational approximations. One of these is an iterative linear convergence algorithm called CORDIC. It can be used to compute all of the aforementioned classes of functions using only shifts and adds, eliminating the need for power and resource hungry multipliers.

Today, among the rise of SoC designs and low power embedded product development on adaptive platforms like FPGAs, it is possible to create hardware accelerators for specific computing tasks. Due to the essential nature of the aforementioned class of functions in various domains, the need for accelerators that can compute these functions is evident. There has been work done by companies on such co-processors[7, 8], but they are closed source and are not meant to be distributed as IP.

The objective of the project is to design, develop and test a coprocessor for computing transcendental functions using the CORDIC algorithm. This involves the research and development of a CORDIC computing core and its interfacing and control logic. This core should be able to operate in both rotation and vectoring modes and be able to generate using the circular and hyperbolic extensions of CORDIC.

It will also be a research into the effectiveness of different ways to implement the CORDIC algorithm and various optimizations that can be used to improve power consumption, resource utilization and frequency of operation.

Our work aims to produce a solution to be used by system designers for integrating transcendental function computation with low hardware and energy costs. Computation of such functions is crucial in many domains including

- Digital Signal Processing

- Power Electronics
- Robotics
- Communication
- Machine Learning

Our solution will be able to improve the performance of an SoC for such use-cases without consuming significant power or resources. This is relevant in the current market situation, where adaptive computing platforms using FPGAs are becoming popular, and custom computing platforms can be designed to solve issues with hardware acceleration modules for performance improvements for specialized use-cases.

To enable easy re-use and integration into systems, our system will have a standard bus protocol interface for control. This will enable use of the coprocessor as a memory mapped device.

# CHAPTER 2

# LITERATURE SURVEY

CORDIC was conceived in 1956 by Jack E. Volder [1] at the aeroelectronics department of Convair out of necessity to replace the analog resolver in the B-58 bomber's navigation computer with a more accurate and faster real-time digital solution

## 2.1) The CORDIC algorithm

CORDIC can be used to derive all trigonometric functions using vector rotations. Vector rotations can be used for polar to rectangular, and rectangular to polar conversions. CORDIC provides an iterative method of performing vector rotations by arbitrary angles using shifts and adds. The algorithm is derived from the general rotation transform,

$$x' = x\,cos(\phi) - y\,sin(\phi)$$

$$y' = x\,sin(\phi) + y\,cos(\phi)$$

Equation 2.1 : General rotation transform

This can be rearranged into :

$$x' = cos(\phi)\,[x - y\,tan(\phi)]$$

$$y' = cos(\phi)\,[y + x\,tan(\phi)]$$

Equation 2.2 : Rearranged equation for the rotation transform

The innovation in CORDIC is to restrict $\phi$ to be such that $tan(\phi) = \pm 2^{i}$. Multiplication by a power of two is equivalent to a simple right shift in binary logic. While multipliers are very expensive in terms of power and resources, shift operations can be very easily implemented. Arbitrary rotations are done by successive rotations of progressively smaller angles, $\phi$. The iterative rotation can now be represented by :

$$x_{i+1} = K_i [x_i - y_i \cdot d_i \cdot 2^{-i}]$$

$$y_{i+1} = K_i [y_i + x_i \cdot d_i \cdot 2^{-i}]$$

Equation 2.3 : Iterative equation of a CORDIC rotation

where

$$K_i = cos(tan^{-1}(2^{-i})) = 1 / \sqrt{1 + 2^{-2i}}$$

$$d_i = \pm 1$$

Removing the scale constant, $K_i$ yields a shift-add algorithm. $K_i$ is applied at the end. The total scaling factor approaches 0.673 as the number of iterations approaches infinity. The total rotation angle sequence is determined by the sequence of directions of elementary rotations. Another adder-subtractor unit is used to maintain the remainder of the angle to be rotated to decide the rotation angle in each iteration. The remainder angle is represented using $z_i$.

$$z_{i+1} = z_i - d_i \cdot tan^{-1}(2^{-i})$$

Equation 2.4 : Iterative angle updation in CORDIC

The arctangent values are provided by a small look-up table.

## 2.2) CORDIC rotation and vectoring modes

The CORDIC rotator can operate in two modes, the first called rotation rotates the input vector by a specified angle. The second called vectoring rotates the input vector to the x axis while recording the angle made to do so, effectively measuring the angle made by the initial vector with the x axis.

In rotation mode, the angle accumulator, $z$is loaded with the angle to rotate the vector $(x, y)$. The rotation decision at each iteration, represented by $d_i$ is made to reduce $z_i$ to zero. For rotation mode, the CORDIC equations are given by

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})$$
$$d_i = -1 \; if \; z_i < 0; \; +1 \; otherwise$$

Equation 2.5 : CORDIC equations in rotation mode

The results after $n$ iterations is :

$$x_n = A_n [x_0 \cos(z_0) - y_0 \sin(z_0)]$$
$$y_n = A_n [y_0 \cos(z_0) + x_0 \sin(z_0)]$$
$$z_n = 0$$
$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

Equation 2.6 : Results of CORDIC in rotation mode

In the vectoring mode, the CORDIC algorithm rotates the vector to bring it to the x axis. The result is the magnitude and angle with the x axis of the initial vector,$(x_0, y_0)$ . It seeks to minimize the y component at each iteration and take it to 0. :

$$x_{i+1} = K_i [x_i - y_i \cdot d_i \cdot 2^{-i}]$$
$$y_{i+1} = K_i [y_i + x_i \cdot d_i \cdot 2^{-i}]$$
$$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})$$
$$d_i = +1 \; if \; y_i < 0; \; -1 \; otherwise$$

Equation 2.7 : CORDIC equations in vectoring mode

The result after n iterations is :

$$x_n = A_n \sqrt{x_0^2 + y_0^2}$$

$$y_n = 0$$

$$z_n = z_0 + \tan^{-1}(y_0 / x_0)$$

$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

Equation 2.6 : Results of CORDIC in vectoring mode

The vectoring and rotation modes both are limited to angles between $-\pi/2$ and $\pi/2$. This is because the tangent is limited to $2^0$ in the first iteration. For rotations larger than $\pi/2$ additional rotations are needed.

$$x' = -d \cdot y$$

$$y' = d \cdot x$$

$$z' = z + d \cdot \pi/2$$

$$d = +1 \; if \; y < 0 \; ; \; -1 \; otherwise$$

Equation 2.7 : Equations for initial rotation by 90 degrees

$$x' = d \cdot x$$

$$y' = d \cdot y$$

$$z' = z \; if \; d = 1 \; ; \; z - \pi \; if \; d = -1$$

$$d = -1 \; if \; x < 0 \; ; \; +1 \; otherwise$$

Equation 2.8 : Equations for initial rotation by 180 degrees

## 2.3) Computing trigonometric functions using CORDIC

The CORDIC rotator described in the earlier section is usable to compute several trigonometric functions directly and others indirectly. Judicious choice of initial values and modes permits direct computation of sine, cosine, arctangent, vector magnitude and transformations between polar and Cartesian coordinates.

### 2.3.1) Sine and Cosine

The rotational mode CORDIC operation can simultaneously compute the sine and cosine of the input angle. Setting the y component of the input vector to zero in Equation 2.5 reduces the rotation mode result to

$$x_n = A_n x_0 \cos(z_0)$$
$$y_n = A_n x_0 \sin(z_0)$$

Equation 2.9 : Result of sine and cosine computation in CORDIC

### 2.3.2) Polar to Cartesian Transformation

Polar to cartesian transformation can be achieved by selecting the rotation mode with $x_0$ = polar magnitude, $z_0$ = polar phase, and $y_0$ = 0. This reduces Equation 2.5 to

$$x = A_n r \cos(\theta)$$
$$y = A_n r \sin(\theta)$$

Equation 2.10: Result of polar to cartesian transform in CORDIC

### 2.3.3) General vector rotation

The rotation mode CORDIC rotator is also useful for performing general vector rotations, as are often encountered in motion correction and control systems. For general rotation, the 2 dimensional input vector is presented to the rotator inputs. The rotator rotates the vector through the desired angle.

$$x_n = A_n [x_0 \cos(z_0) - y_0 \sin(z_0)]$$
$$y_n = A_n [y_0 \cos(z_0) + x_0 \sin(z_0)]$$

Equation 2.11 : Result of general vector rotation in CORDIC

### 2.3.4) Inverse of Sine and Cosine

The Arcsine can be computed by starting with a unit vector on the positive x axis, then rotating it so that its y component is equal to the input argument. The arcsine is then the angle subtended to cause the y component of the rotated vector to match the argument. The

decision function in this case is the result of a comparison between the input value and the y component of the rotated vector at each iteration, for input argument c, :

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

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

$$z_{i+1} = z_i - d_i \cdot tan^{-1}(2^{-i})$$

$$d_i = +1 \; if \; y_i < c; \; -1 \; otherwise$$

Equation 2.12 : CORDIC equation for computing arcsine

$$x_n = \sqrt{(A_n . x_0)^2 - c^2}$$

$$y_n = c$$

$$z_n = z_0 + sin^{-1}(c / A_n . x_0)$$

$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

Equation 2.13 : Results for computing arcsine in CORDIC

The Arccosine computation is similar, except the difference between the x component and the input is used as the decision function. The Arccosine could also be computed by using the arcsine function and subtracting π/2 from the result, followed by an angular reduction if the result is in the fourth quadrant.

## 2.3.5) Cartesian to Polar Transformation

The Cartesian to Polar transformation consists of finding the magnitude ( $r = \sqrt{x^2 + y^2}$) and phase angle ($\phi = tan^{-1}(y / x)$) of the input vector, (x, y). This can be done with the vectoring mode of the rotator by setting the initial value of z to zero.

$$x_n = A_n \sqrt{x_0^2 + y_0^2}$$

$$z_n = z_0 + tan^{-1}(y_0 / x_0)$$

Equation 2.14 : Results for cartesian to polar transform in CORDIC

## 2.4) Extension to linear functions

A simple modification to the CORDIC equation permits the computation of linear functions:

$$x_{i+1} = x_i$$

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

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

$$d_i = -1 \; if \; z_i < 0 \; ; \; -1 \; otherwise$$

Equation 2.15 : CORDIC equations in rotation mode of linear extension

$$x_n = x_0$$

$$y_n = y_0 + x_o \cdot z_o$$

$$z_n = 0$$

Equation 2.16 : CORDIC result in rotation mode of linear extension

This operation is similar to the shift-add implementation of a multiplier. However, it is not deemed to be useful for our requirement which is to implement transcendental functions. The vectoring mode for the equation above results in :

$$x_n = x_0$$

$$y_n = 0$$

$$z_n = z_0 - y_0 / x_0$$

Equation 2.17 : CORDIC result in vectoring mode of linear extension

The vectoring mode too, is also not particularly interesting for our application since it is similar to a shift and subtract implementation of a divider.

## 2.5) Extension to hyperbolic functions

The close relationship between the trigonometric and hyperbolic functions suggests the same architecture can be used to compute the hyperbolic functions. While there is early mention of using the CORDIC structure for hyperbolic coordinate transforms the first description of the algorithm is that by Walther [3].

The CORDIC equations for hyperbolic rotations are derived using the same manipulations as those used to derive the rotation in the circular coordinate system. For rotation mode these are :

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

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

$$z_{i+1} = z_i - d_i \cdot \tanh^{-1}(2^{-i})$$

$$d_i = -1 \; if \; z_i < 0 \; ; \; +1 \; otherwise$$

Equation 2.18 : CORDIC equations in rotation mode of hyperbolic extension

$$x_n = A_n [x_0 \cosh(z_0) + y_0 \sinh(z_0)]$$

$$y_n = A_n [y_0 \cosh(z_0) + x_0 \sinh(z_0)]$$

$$z_n = 0$$

$$A_n = \prod_n \sqrt{1 - 2^{-2i}}$$

Equation 2.19 : CORDIC results in rotation mode of hyperbolic extension

In the vectoring mode,

$$x_n = A_n \sqrt{x_0^2 - y_0^2}$$

$$y_n = 0$$

$$z_n = z_0 + \tanh^{-1}(y_0 / x_0)$$

$$A_n = \prod_n \sqrt{1 - 2^{-2i}}$$

Equation 2.20 : CORDIC results in vectoring mode of hyperbolic extension

The hyperbolic equivalents of all the functions discussed for the circular coordinate system can be computed by the hyperbolic extension.

## 2.6) Deriving other additional functions

CORDIC can be used to directly compute the following functions :

- Circular rotation mode : $sin(\theta)$ , $cos(\theta)$
- Hyperbolic rotation mode : $sinh(\theta)$ , $cosh(\theta)$
- Circular vectoring mode : $tan^{-1}(y/x)$ , $\sqrt{x^2 + y^2}$
- Hyperbolic vectoring mode : $tanh^{-1}(y/x)$ , $\sqrt{x^2 - y^2}$
- Circular vectoring mode : $sin^{-1}(\theta)$, $cos^{-1}(\theta)$
- Hyperbolic vectoring mode : $sinh^{-1}(\theta)$, $cosh^{-1}(\theta)$

The following functions can be derived from the above functions :

$$tan(\theta) = sin(\theta) / cos(\theta)$$

$$tanh(\theta) = sinh(\theta) / cosh(\theta)$$

$$e^{\theta} = sinh(\theta) + cosh(\theta)$$

$$ln(\theta) = 2\, tanh^{-1}(y/x); x = \theta + 1,\ y = \theta - 1$$

$$\sqrt{\theta} = \sqrt{x^2 - y^2}; x = \theta + 1/4,\ y = \theta - 1/4$$

Equation 2.21 : Derivation of function from CORDIC functions

## 2.7) Unified CORDIC equations

The similarity in equations of all the 3 extensions, that is, the circular, linear and hyperbolic extensions, motivate us to form a unified equation for CORDIC rotation.

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

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

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

$$d_i = -1\ if\ z_i < 0;\ +1\ otherwise$$

Equation 2.22 : Unified CORDIC equations

Here, $m$takes on values 1, 0 and -1 for circular, linear and hyperbolic modes. In the angle update equation, $e_i$is the elementary angle of rotation for iteration i in the selected coordinate system. $e_i = tan^{-1}(2^{-1})$for $m = 1$, $e_i = 2^{-1}$for $m = 0$ and $e_i = tanh^{-1}(2^{-1})$for $m = -1$. This allows us to build a general purpose CORDIC processor. Since the linear mode is not useful for our requirements, we use only the circular and hyperbolic modes.

## 2.8) Hardware implementations Of CORDIC

Hardware implementation for CORDIC arithmetic requires three registers for x, y and z, two shifters to supply the terms $2^{-i}$ x and $2^{-i}$ y to the adder/subs tractor units and a lookup table to store the values of $\alpha_i=\tan^{-1}2^{-i}$ . The $d_i$ factor (-1 and 1) selects the shift operand or its complement. The initial inputs to the architectures are $X_0=1$, $Y_0=0$. The structure requires a preprocessing unit to converge the input angles to the desired range and a post processing unit to fix the sign of outputs depending on the initial angle quadrants.

The pre-processing unit takes in angles of any range and converges it to the interval [-π/2, π/2]. It keeps record of the quadrant of the input angle which may be used in the postprocessing unit to fix the sign of outputs. These two blocks are inevitable for any application as the input range cannot always be predicted.

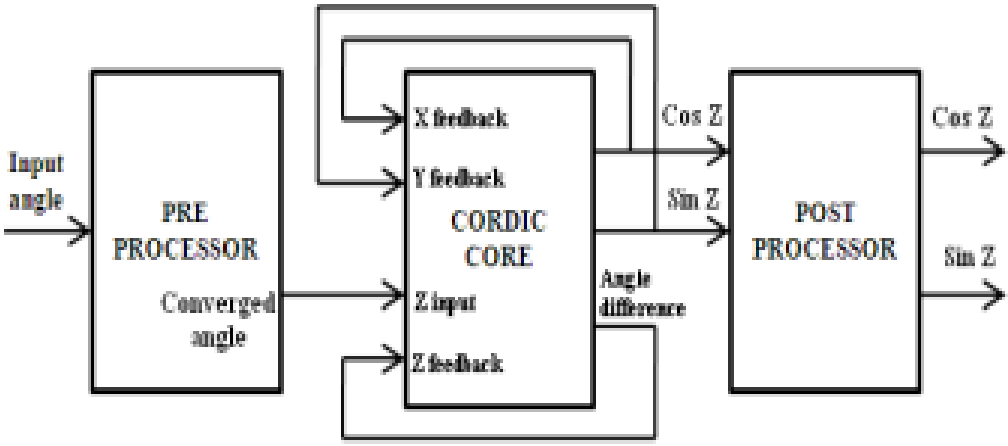



Fig 2.1.block diagram of CORDIC

### 2.8.1) Sequential /Iterative Architecture

The CORDIC algorithm requires approximately one shift add/sub operation for each bit of accuracy. A CORDIC core implemented with sequential architectural configuration, implements these shift-add/sub operations serially, using a single shift-add/sub stage and feeding back the output. An iterative CORDIC core with N bit width has a minimum latency of N cycles. It takes at least N cycles to produce new output. The implementation size is directly proportional to the internal precision. This architecture finds major application in pocket calculators, since even a delay of thousands of clock cycles constitutes a small fraction of a second for a human user. To obtain sine and cosine values of a given angle $z_0$,

iterative structure takes the value of $(x_0,y_0)$ as (1,0) in the first clock cycle. From the next clock cycle onwards it takes the feedback values and the operation continues till the required output is obtained. The control signal for the input registers is provided by a state-machine designed for the purpose. To get an N bit precise output, the structure requires iterating at least N times. Hence, it requires a minimum of N clock cycles for required output.

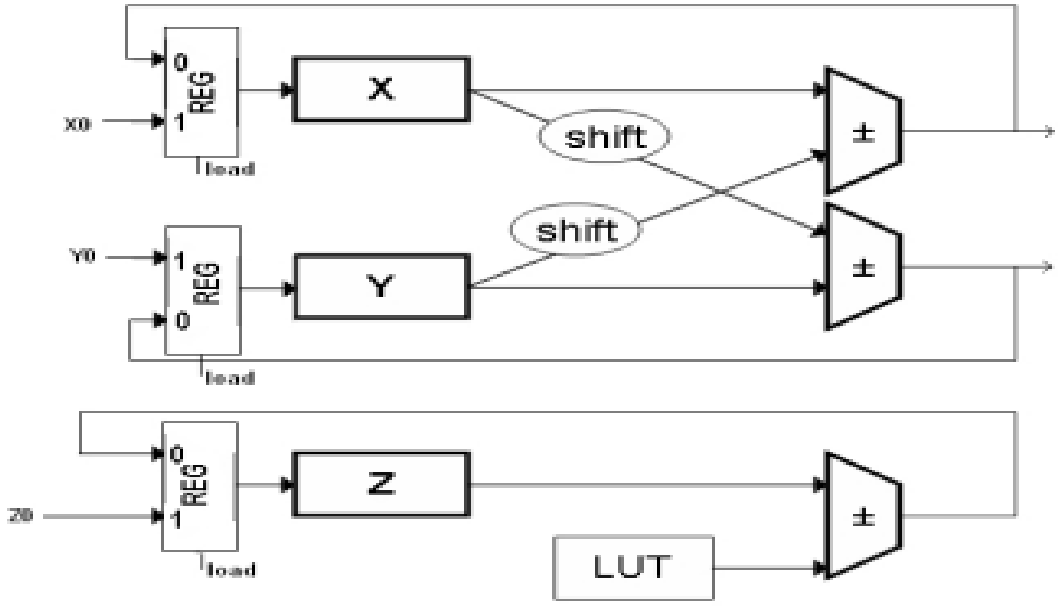



Fig 2.2.Iterative CORDIC

## 2.8.2) Parallel/Cascaded CORDIC Architecture

This architecture uses multiple instances of Iterative CORDIC structure. A CORDIC core with parallel architectural configuration implements the shift-add/sub operations in parallel using an array of shift-add/sub stages.A parallel CORDIC core with N bit output has a latency of one clock cycle. The implementation size of a parallel CORDIC core is directly proportional to the internal precision times the number of iterations. Instantiation of blocks must be done N times for an N bit precise output. Unlike in iterative CORDIC, all iterations are done parallelly and hence need not wait for N clock cycles. But, the latency of each block has an inevitable role in fixing the clock frequency. The frequency of operation for Parallel CORDIC cores will be lesser than the frequency of operation of iterative CORDIC. But this is the case with a single iteration. While dealing with a chain of inputs, the parallel structure proves to be a more efficient one since the throughput of parallel structure is much greater

than that of iterative. The shifters used in this structure are constant shifters, which can be implemented in the wiring, so that the hardware can be reduced.

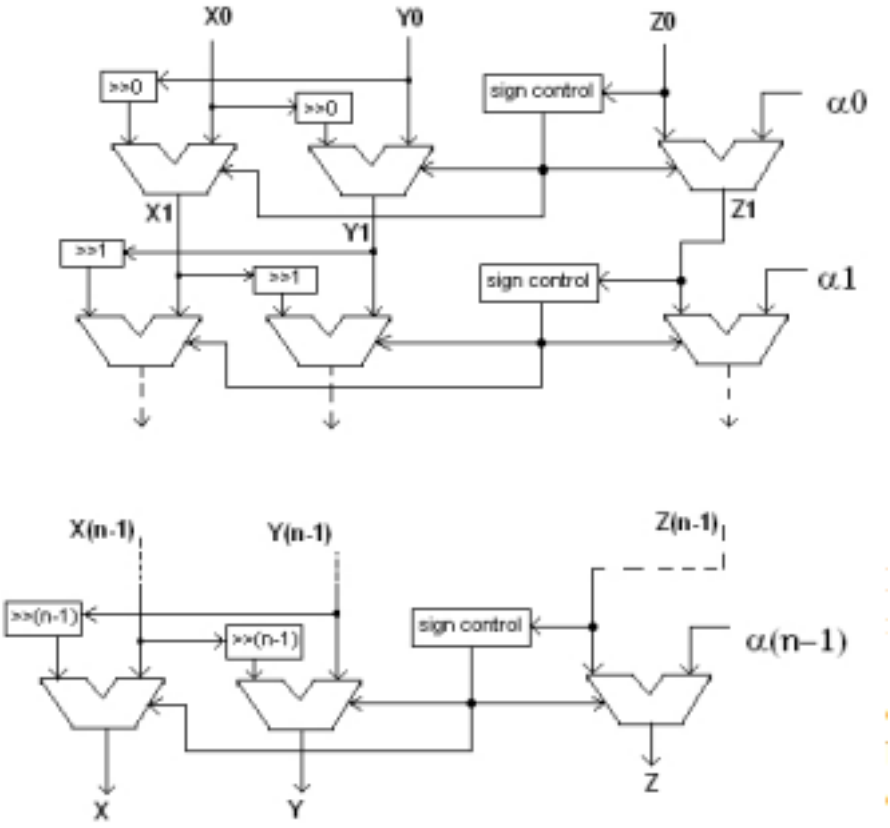



Fig.2.3.Parallel CORDIC

## 2.8.3) Pipelined CORDIC Architecture

Pipelined architecture uses a structure similar to that of a Parallel CORDIC. It uses pipeline registers in between each iteration phase as shown in Fig 2.4. Pipelined CORDIC proves to be advantageous with continuous input values. For an N bit data CORDIC core, N stage pipeline can give maximum result. The first output of an N-stage pipelined CORDIC core is obtained after N clock cycles. Thereafter, outputs will be generated during every clock cycle. The advantage of pipelined CORDIC cores over parallel and iterative CORDIC cores is its frequency of operation which is much higher when compared to the latter two structures. Pipeline realizes the same throughput as that of parallel core with improved frequency of operation. This feature of pipelined structure makes it the best possible option for high frequency satellite communication and other communication systems. A drawback of

pipelined structure is the increase in area introduced by the registers. Hence, there is a trade-off between parallel and pipelined cores based on frequency and area.

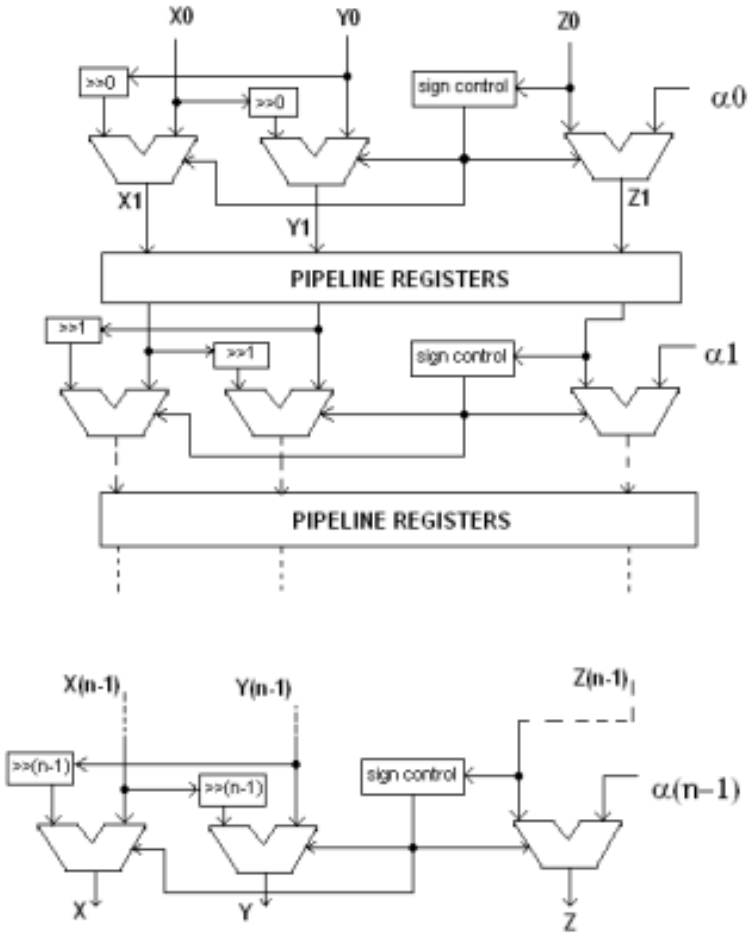



Fig 2.4. Pipelined CORDIC

# CHAPTER 3

# METHODOLOGY

The various methods of implementing the CORDIC algorithm in hardware have been discussed in section 2.8. Their relative performance and resource utilization have been studied extensively by researchers [4]. The iterative design is expected to be the most suited for our application since it has lower hardware consumption and good performance. The pipelined version is preferable only when high bandwidth is required.

Apart from these, there are more advanced versions of CORDIC with varying levels of complexity that have been proposed, like Step branching cordic, Doubt step branching cordic, Hybrid cordic, look up table cordic[5]. First, these alternatives will be studied in more detail to evaluate the improvements they make and their costs. If deemed acceptable, we will move to implement and test them

## 3.1) Work Plan

The team will first begin by studying the different CORDIC alternatives in detail and expand on the literature survey done till now.

At the same time, selected candidate algorithms including the base unified CORDIC algorithm will be tested as non-synthesizable functional models and their performance will be tested in terms of parameters like the number of cycles required and error v/s number of iterations.

Then, if deemed satisfactory, these algorithms will be implemented as synthesizable RTL code in verilog.

Then, verification of these modules will be performed, using simulation and constrained random verification by comparison with the reference models that were developed earlier. Formal verification of selected components may also be performed.

After verification of the CORDIC core, its surrounding control logic, pre and post processing logic and bus interface will be designed and developed. The blueprint for this should already have been ready when functional modeling was done

At the same time, the CORDIC cores will be synthesized for FPGA and ASIC targets. If there are multiple candidates at this stage, their resource utilization will be analyzed and reported. When ready, their supporting logic will also be synthesized.

Then, verification of the whole processor including all its individual components will be performed. This would use constrained random methodology and some parts of the design like the bus interface might be formally verified if possible.

## 3.2) Timeline

## 3.3) RTL design methodology

The RTL Design will be done using the verilog HDL[9]. Various naming conventions and standards will be put in place to ensure uniformity and clean reusable code. The modules used will be highly parameterized and can be reused in other projects. The source code will be open sourced under the GPL open source license [10].

## 3.4) Design verification methodology

Design verification will be done using formal methods and constrained random verification. System verilog will be used for verification. The earlier functional modeling phase is also planned to be done using System verilog[8].

In constrained random verification, the design will be fed with random but valid inputs and the functional models will be used to generate expected outputs. They will be compared with the actual output of the system to detect bugs in the design.

In formal verification, properties of the design are provided as assertions and assumptions. Formal tools like symbiyosys[13] then exhaustively search the state space of the design to find errors. It provides a formal proof that our design adheres to the provided properties.

The system as a whole cannot be verified using formal methods due to high computational complexity. So, it will be used to verify only crucial parts of the design while constrained random verification finds bugs in the design as a whole.

## 3.5) Synthesis methodology

The design will be synthesized at multiple stages in the project to check for unexpected bugs that cause high resource utilization or failure to synthesize. Synthesis will be tested for FPGAs using the Xilinx Vivado toolchain and for ASICs using open source PDKs like the skywater PDK or FreePDK45.

The synthesis results give valuable insights into resource utilization and maximum performance of the coprocessor. This will be recorded and reported.

## 3.6) Integration methodology

The co-processor needs to be integrated with a conventional processing core to control it and give inputs and receive outputs. The co-processor will act as a slave device on a standard bus protocol like wishbone[11] or AXI[12]. It will have memory mapped registers for the different operations to be performed and inputs to the core and outputs from it.

Currently, the microblaze soft core and picoRV32, a RISC-V core are seen as the candidates for integration testing. Programs will be written testing the interfacing with the co-processor and its use for accelerated function computation.

Further, integration with the open source SoC ecosystem adding support with the fuseSoC IP management system [6] will enable easier development of systems using the coprocessor.

## 3.7) Performance Characterization

The performance of the system will be characterized in terms of latency (number of cycles needed for computation) and maximum clock speed and the error in the results. The number of cycles and error can be found in the simulation stage, but maximum clock speed depends on the target architecture and requires the design to be synthesized. The final report will contain figures on performance of the core according to these parameters.

# CHAPTER 4

# SCOPE AND CONCLUSION

Our work will be focused on design exploration of CORDIC architectures for acceleration of transcendental functions in low power computing environments and design and development of a CORDIC coprocessor with a memory mapped register interface with the support of a standard bus protocol like AXI/wishbone.

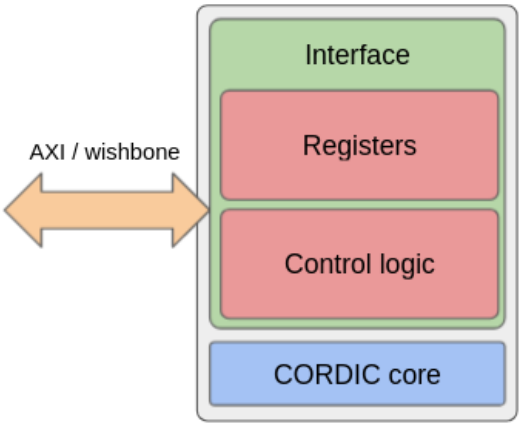



Figure 4.1 : High level architecture of the CORDIC coprocessor

It will involve the analysis of the performance, power consumption, resource utilization and maximum frequency of operation of synthesized versions of the core targeting FPGA and ASICs.

Verification will be done with constrained random verification techniques and formal techniques on parts of the design.

Further, integration with a processor into an SoC synthesized on an FPGA will also be done. Software required for testing this will also be written. If possible, integration into a standard IP ecosystem like fuseSoC[6] will also be done.

The IP developed for this project will be open sourced under the GPL license.

# REFERENCES


[1.] **Jack E Volder**, "*The CORDIC Trigonometric Computing Technique*", IRE TRANSACTIONS ON ELECTRONIC COMPUTERS, Vol EC-8, pp330-334 Sept 1959

[2.] **Ray Andraka**, "*A survey of CORDIC algorithms for FPGA based computers*", Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, March 1998

[3.] **JS Walther**, "*A unified algorithm for elementary functions*" Proceedings of the May 18-20, 1971, spring joint computer conference, May 1971

[4.] **Ramesh Bhakthavatchalu; M.S. Sinith; Parvathi Nair; K. Jismi,**"*A comparison of pipelined parallel and iterative CORDIC design on FPGA",* 5th International Conference on Industrial and Information Systems, 2010

[5.] **Jason Todd Arbaugh**, "*Table Look-up CORDIC : Effective Rotations Through Angle Partitioning*" PhD dissertation, The University of Texas at Austing, December 2004

[6.] **Olof Kindgren**, "*A Scalable Approach to IP Management with FuseSoC"*, Workshop on Open-Source Design Automation (OSDA), 29 March 2019

[7.] **STMicroelectronics**, "*STM32H7- CORDIC coprocessor*", Revision 1.0 Product training presentation

[8.] **Accellera**, "SystemVerilog 3.1a Language Reference Manual Accellera's Extensions to Verilog"

[9.] **IEEE Computer Society**, "IEEE Standard for Verilog Hardware Description Language", IEEE Std 1364™-2005 (Revision of IEEE Std 1364-2001)

[10.] **Free Software Foundation, Inc**, "*GNU GENERAL PUBLIC LICENSE*", Version 3, 29 June 2007

[11.] **OpenCores.org**, "*WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*", Revision: B.3, September 7, 2002

[12.] **Arm Limited**, "*AMBA AXI and ACE Protocol Specification Version E*", Online documentation/manual

[13.] **Yosys HQ**, "SymbiYosys (sby) Documentation", Online documentation/manual

[14.] **R. Timothy Edwards**, "*Google/SkyWater and the Promise of the Open PDK*", Workshop on Open-Source EDA Technology (WOSET), November, 2021