International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 12, December - 2013

3-D Geometric Transformations Using Cordic Algorithm

Richa Upadhyay*, Dr. Nisha Sarwade**

*(Department of Electronics and Telecommunication Engineering, SVKM's NMIMS MPSTME, Mumbai

** (Department of Electrical Engineering, V.J.T.I, Mumbai

ABSTRACT
This work deals with the application of CORDIC in 3D graphics. Multimedia applications for 3D graphics have massive computational requirements. Since the 3D operations are of a geometric nature, it is easy to express them using CORDIC-type primitives. Although CORDIC may not be the fastest technique to perform these operations, it is attractive due to the simplicity of its hardware implementation, since the same iterative algorithm could be used for all these applications using the basic shift-add operations. It only needs the use of 2 shifter and 3 adder modules. Hence here the very well known CORDIC is being used to make the basic building blocks required for various operations of the 3D geometric transformation. The CORDIC architecture implemented in this paper is purely combinational since only the speed of the part of geometric processor designed is being considered. Hence, this paper aims to develop a high speed geometric processor.

***Keywords:*CORDIC algorithm, Geometric transformation, Lighting, Pipeline architecture,Perspective division, Rendering, Vector Normalization**

I. INTRODUCTION

3-d Graphics is an integral part of multimedia applications such as computer animation, video games, medical imaging, scientific visualization and simulation, virtual reality, CAD tools etc. 3-d graphics studies methods for digitally synthesizing and manipulating visual content. In this paper the focus is on the 3 d graphics rendering pipeline which converts 3-d model to 2-d image. This pipeline consists of three distinct stages: Application, geometric transformation and rasterization, where each of these stages is a pipeline in itself. In this paper the emphasis is only on Geomertic Transformation .Such systems require quality, interactivity and simulation of physical effect which calls for the design of high-performance graphics subsystems. The main design criterion of such system is strong support of arithmetic functionality i.e. 3 –d graphics require tremendous computational requirement. To cope up with the future scaling of performance, it is accept that introduction of arithmetic modules which provide high throughput for a variety of different arithmetic function will be necessary. The CORDIC algorithm might be a good candidate to satisfy the simultaneous demand of high throughput and diversity of arithmetic functions.

In this work, we present the formulation of high throughput representative 3D computer graphics operations in terms of CORDIC type primitives. In this paper we describe a fixed-point implementation of the vertex processor operations using the COordinate Rotation DIgital Computer (CORDIC).A CORDIC-based solution for vertex processing exhibits a number of advantages over classical Multiply-and-Accumulate solutions. First, since a single primitive is used to describe the computation, the code can easily be vectorized and multithreaded and second, since a CORDIC iteration consists of only a shift operation followed by an addition, the computation may be deeply pipelined. Third, the CORDIC algorithm produces one bit of accuracy per iteration. Thus, all CORDIC-based rotations will have the same latency for a given precision, which allows trade-offs to be made between precision and latency at run-time.

The remainder of this paper is organized as follows: Next section briefs about the basic CORDIC algorithm. The section III describes the 3-D graphics pipeline and use of CORDIC algorithm in its different operations. Section IV presents the design of the CORDIC modules and its architecture along with experimental results. The section V summarizes the contribution of this paper and all the figures and tables have been accumulated in section VI.

II. CORDIC ALGORITHM

COordinate Rotation DIgital Computer is abbreviated as CORDIC. The key concept of CORDIC arithmetic is based on the simple and ancient principles of two-dimensional geometry. But the iterative formulation of a computational algorithm for its implementation was first described in 1959 by Jack E. Volder [1], [2] for the computation of trigonometric functions, multiplication and division.

The CORDIC algorithm performs a planar rotation as shown in Fig 1. Graphically, planar rotation means transforming a vector (Xi, Yi) into a new vector (Xj, Yj).
Using a matrix form, a planar rotation for a vector of (Xi, Yi) is defined as-

not write "Magnetization (A/m) × 1000" because the reader would not know whether the top axis label in Fig. 1 meant 16000 A/m or 0.016 A/m. Figure labels should be legible, approximately 8 to 12 point type.

$$\begin{bmatrix} X_j \\ Y_j \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \qquad (1)$$

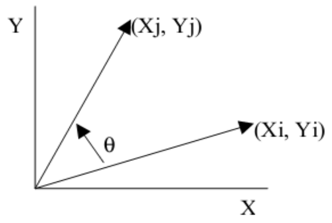


Figure 1: Plannar rotation of a vector

The θ angle rotation can be executed in several steps, using an iterative process. Each step completes a small part of the rotation. Many steps will compose one planar rotation. The angle for each step is given by equation 2

$$\theta_n = \arctan\left(\frac{1}{2^n}\right) \qquad (2)$$

All iteration-angles summed must equal the rotation angle θ.

$$\sum_{n=0}^{\infty} S_n\theta_n = \theta (3)$$

Where S = {+1,-1}

After the required manipulations in the basic equation, the CORDIC equations derived are -

$$X_{n+1} = X_n - S_n 2^{-n} Y_n \qquad (4)$$

$$Y_{n+1} = Y_n + S_n 2^{-n} X_n$$

$$Z_{n+1} = Z_n - S_n \arctan\left(\frac{1}{2^n}\right)$$

Thus, thereexisttwomodalitiesofCORDICalgorithm,**VECTORING** and**ROTATION**mode.Invectoring mode,coordinates(X_n , Y_n)arerotated until Y_n convergestozero.In rotationmode,initialvector(X_n ,Y_n)startsalignedwiththe*x*axisan disrotatedbyanangleofθ*i*everycycle,so after*n*iterations,θ*n*is the obtainedangle.

Also,

$$S_n = \begin{bmatrix} sign(Z_n) for rotation mode \\ -sign(Y_n) for vectoring mode \end{bmatrix}$$

Hence, in ROTATION MODE

$$S_n = \begin{bmatrix} 1 & ; & Z_n > 0 \\ -1 & ; & Z_n \le 0 \end{bmatrix}$$

And in VECTORING MODE

$$S_n = \begin{bmatrix} -1 & ; & Y_n > 0 \\ 1 & ; & Y_n \le 0 \end{bmatrix}$$

So, the CORDIC method evaluates elementary functions merely by table-look-up, shift and add operations. A small number (of the order of n, where n bits of precision is required in the evaluation of the functions) of pre-calculated fixed constants is all that is required to be stored in the look-up table. The CORDIC algorithm has nice geometrical interpretations: trigonometric, exponential, multiply functions are evaluated via rotations in the circular, hyperbolic and linear coordinate systems, respectively. Their inverses (i.e., inverse trigonometric functions, logarithm and division) can be implemented in a "vectoring" mode in the appropriate coordinate system.

III. 3-D GRAPHICS PIPELINE

The process of converting the geometric description of a 3D model to a 2D image to be displayed on a monitor is referred to as 3D graphics rendering. The 3D graphics pipeline is thus the underlying tool for this real-time process. The 3D graphics pipeline itself consists of three distinct stages: Application, geometric transformation and rasterization, where each of these stages is a pipeline in itself. Rasterization and Application is not of interest for this work; therefore, it is not discussed any longer. The sequence of steps involved in geometric transformation is presented in Figure 2, and consists of the following stages:

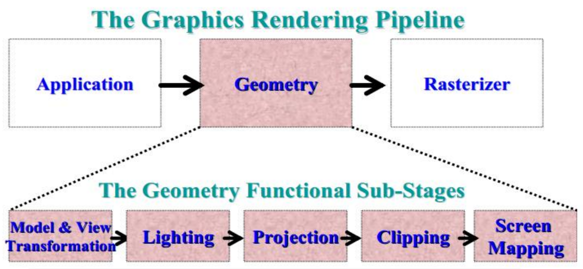

The Graphics Rendering Pipeline
Application
Geometry
Rasterizer
The Geometry Functional Sub-Stages
Model & View Transformation
Lighting
Projection
Clipping
Screen Mapping

Figure 2: 3-d graphics rendering pipeline

My work deals with some of the most important data-intensive 3D graphics geometric transformation which includes lighting, exponentiation, vector normalization, perspectivedivision etc. A detail description of them as well as their design using CORDIC algorithm is as follows:

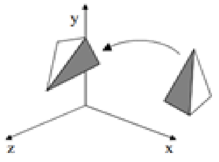
1) MODELING AND VIEWING TRANSFORM:

Model transforms link object coordinates to world co-ordinates and Viewing Transform, transforms the objects from World Co-ordinates to Camera co-ordinates (eye space). These include the Euclidean transformations, that do not change lengths and angle measures (translation and rotation) followed by Affine Transformations (scaling and shearing). Of all these operation only rotation operation can be implemented with the help of CORDIC algorithm.

Rotation in space is more complex because in space we rotate about either x- axis, or y-axis or z-axis. When rotation about z axis only the x and y co-ordinates will change, z coordinate will remain the same. In effect it is exactly a rotation about origin in the xy plane. Here we find that this is an application of CORDIC algorithm as this rotation matrix is very similar to the CORDIC matrix. Therefore the rotation equations are:

Rotation about x-axis

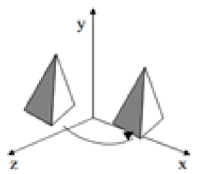
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\mathbf{P'} = \mathbf{R}_x(\theta) \cdot \mathbf{P}$$

Rotation about y-axis

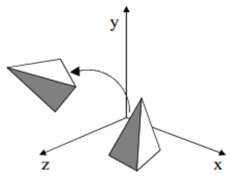
$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\mathbf{P' = R_y(\theta) \cdot P}$$

Rotation about z-axis

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\mathbf{P' = R_z(\theta) \cdot P}$$

We find that the rotation matrix is quiet similar to the basic CORDIC rotation matrix. The co-ordinate of the axis about which the rotation has to take place remains unchanged while the other two are fed into the rotation mode CORDIC block diagram i.e. the sine/cosine module.

LIGHTING: Lighting is another compute-intensive task of the graphics pipeline. Here, we illustrate the implementation using CORDIC primitives for a per-vertex lighting calculation. The model computes the intensity in a vertex for a single light as follows:

$$\boldsymbol{I = K_a\sqrt{1-(l\cdot t)^2} + K_s(\sqrt{1-(l\cdot t)^2}\sqrt{1-(v\cdot t)^2} - (l\cdot t)(v\cdot t)^s\,)}$$

Since the dot product of unit vectors corresponds to the cosine of the angle between the vectors, the computation of the intensity requires the computation of the sine and cosine of the angle between the pair of vectors {l, t} and {v, t}. To allow an efficient implementation with CORDIC primitives, we express the above equation
as follows:

$$I = K_a + c((K_d + K_s \sin(v,t))\sin(l,t) - K_s \exp[s\,ln(\cos(v,t))]\cos(l,t))$$

K_a, K_D, and K_s are the ambient, diffuse, and specular coefficients, s is the surface shininess, and l, n, v, and t are the light, normal, view, and tangent unit vectors on the surface.

During lighting stage the sine and the cosine of angles θ between 3D unit vectors are to be computed as it is explained in the above equations. Assuming v1=[x1 y1 z1] and v2=[x2 y2 z2] are two units vectors, from the computation of sine and cosine angles for the equation. First, two CORDIC rotations are carried out to align one of the vectors with the x axis. This gives cosθ. Then, three CORDIC rotations align the second vector with the yOz plane. This gives sinθ. As in the equation for calculating light intensity a powering is also required so the powering a^b is computed as exp (b ln (a)). The exponential and logarithm functions can be implemented with CORDIC in hyperbolic coordinates.

PROJECTION TRANSFORMATION

This transforms the view volume to a normalize view volume and then does the orthogonal projection. It includes the following steps:

i) *VECTOR NORMALIZATION:* Some data intensive graphics operations such as the computation of the effects of a light source and bump mapping require the specification of normalized vectors. We now discuss the implementation of 3D normalization using the same CORDIC modules as for rotation. Normalizing a vector consists of obtaining a unit vector in the same direction. In a Cartesian coordinate system, the operation is performed by computing the magnitude of the vector and dividing its components by the magnitude (or multiplying by its reciprocal).That is-

$$V' = (x'y'z'\,0)^T = \frac{V}{|V|} = \left({}^{x}/_{w}\,{}^{y}/_{w}\,{}^{z}/_{w}\;1\right)^T / \sqrt{x^2+1}$$

A pure software implementation requires three multiplications, two additions, one square root and one division. The CORDIC approach follows as- First, two CORDIC vectoring rotations are needed to calculate the angles with axes y and z. Then, a rotation of a unit vector along the x axis with the same angles will generate a unit vector (that is, normalized) having the same direction as the initial vector.

ii) PERSPECTIVE DIVISION : A projection, in terms of the rendering pipeline is a way to transform a world from one dimensionality to another . Our initial world is three dimensional , and therefore, the rendering pipeline defines a projection from this 3D world into the 2D one that we see. The basic perspective projection function is simple. It can be noticed that the scal ing can be expressed as a division operation (multiplying by the reciprocal). And the difference between clip space and normalized device coordinate space is a division by the W coordinate

$$V' = (x'y'z'w')^T = \frac{V}{w} = \frac{\left({}^{x}/_{w}\,{}^{y}/_{w}\,{}^{z}/_{w}\;1\right)^T}{\sqrt{x^2+y^2+z^2}}$$

CLIPPING: removes the objects that are outside the viewable area. It requires 6 comparisons per vertex.

SCREEN MAPPING: Screen mapping is a translation followed by a scaling that affects the x and y coordinates of the primitives (objects), but not their z coordinates. Screen coordinates plus z∈ [-1, 1] are passed to the rasterizer stage of the pipeline.

It should be noted that the CLIPPING and SCREEN MAPPING do not include any CORDIC application.

The table 1 below gives a brief detail of the different CORDIC modules used in 3-D geometric rendering pipeline. It can be derived that nearly 60% of the operations of geometric transformations canbe performed with the help of CORDIC algorithm which helps in development of architectures for high-performance and low-cost hardware solutions of those applications.

IV. MODULE DESIGN

All the 3-D geometric transformations which can be performed using CORDIC are being implemented using only the two basic CORDIC modules i.e. the Rotation mode CORDIC (used for sine/cosine) and the Vectoring mode CORDIC (used for arctangent). In this section we will discuss the design and implementation of these modules.

Table 1: Use of CORDIC modules in 3-d operations

S.No.	GEOMETRIC PIPELINE	OPERATIONS	DESIGN MODULES
1.	Modelling and Viewing Tranformations	3-D Rotation	Linear Rotation mode CORDIC
2.	Lighting	Sine and Cosine of angles exponentiation	Rotation mode CORDIC, Hyperbolic Vectoring mode CORDIC
3.	Projection Transformation	Division and Squareroot	Linear Vectoring mode CORDIC

Low latency and **high throughput** CORDIC modules have been designed along with **constant scaling**. Since the number of shifts to be performed by the shifters at different stages is fixed (shift-operation through i-bit positions is performed at the ith stage) in case of pipelined CORDIC the shift operations could be hardwired with adders; and therefore shifters are eliminated in the cascaded implementation and this actually leads to a reduction in the latency of circuit. Every CORDIC processor utilizes an individual arctan value that can also be hardwired to the input of every angle accumulator in the absence of a state machine which provides simplicity to this type of architecture. For high-throughput applications, efficient pipelined-architectures with multiple-CORDIC units have been developed to take the advantage of pipelineability of CORDIC, because the digitalhardware is getting cheaper along with the progressive device scaling.

A detailed description of **sine/cosine module** using CORDIC is explained in [8], here the same design is used and further calculations are made.

The HDL code for a sine/cosine module [9] has been generated in Simulink, Figure 3 and then the code is implemented in Xilinx12.1. Figure 4 shows the detailed pipelined architecture of the module.

Figure 5 shows the simulation result of the sine/cosine module on ISIM, the angle input is 30^0 i.e. 1555 (hex) and results obtained are- sin = 3FFC (hex) and cos = 6EDD (hex)

The **arctangent**, ɵ =Atan(y/x), is directly computed using the vectoring mode CORDIC rotator if the angle accumulator is initialized with zero. The argument must be provided as a ratio expressed as a vector (x, y). Presenting the argument as a ratio has the advantage of being able to represent infinity (by setting x=0). Since the arctangent result is taken from the angle accumulator, the CORDIC rotator growth does not affect the result.

$$Z_n = Z_0 + \tan^{-1}[{}^{Y_0}/_{X_0}]$$

The vectoring mode CORDIC rotator produces the magnitude of the input vector as a by-product of computing the arctangent. The Figure 6 shows the block diagram of the Arctangent module using CORDIC in simulink. The pipelined structure of the same code is shown in figure 7.

V. CONCLUSION AND FUTURE SCOPE

In this work, a pipelined architecture of the CORDIC algorithm is designed which is used for various trigonometric operations, division operation and conversion of polar to rectangular co-ordinates and further used for the geometric transformations of 3 D graphics.

In this work all the CORDIC based hardware primitives needed for 3-D graphics operations have been designed and all those 3-D operations which require those CORDIC based modules have also been designed. Besides these operations all the other operations (like translation, scaling, clipping etc) require only matrix multiplication, these have also been designed so that it completes a full geometric processor. All the designing has been done in simulink block set and HDL code of them have been generated, so that FPGA implementation of the code can be done.

It has been noticed that in the intended application with an aim to design a minimum hardware and maximum performance solution, the architecture fulfills both the criteria.

As future work to this project an analog design of the same can also be done. This can be implemented on FPAA by using SIM2SPICE tool which compiles the simulink code to spice netlist, ready for simulation. The next step in the process chain is to compile the SPICE netlist onto the analog hardware. The GRASPER tool is used to place-and-route the netlist onto the FPAA.

VI. FIGURES AND TABLES

In this section all the figures and tables referred in the above sections are assembled.

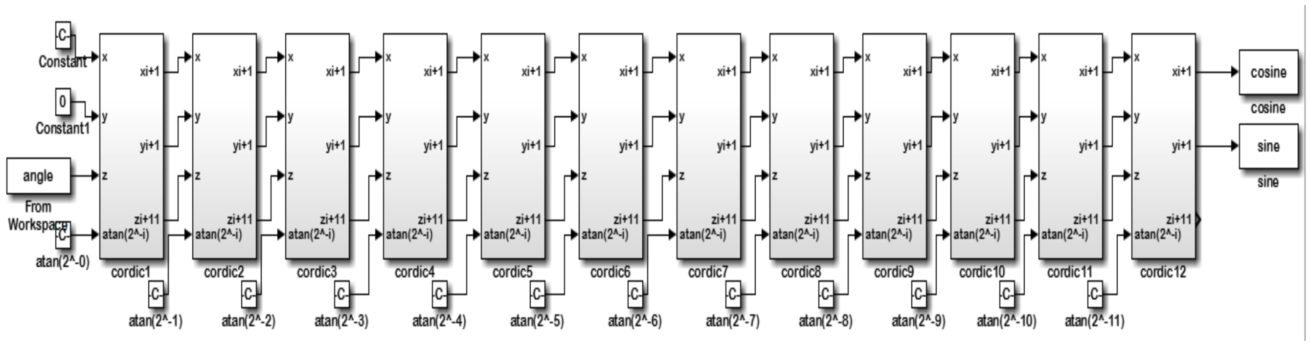


Figure 3: Pipeline architecture of CORDIC for sine and cosine

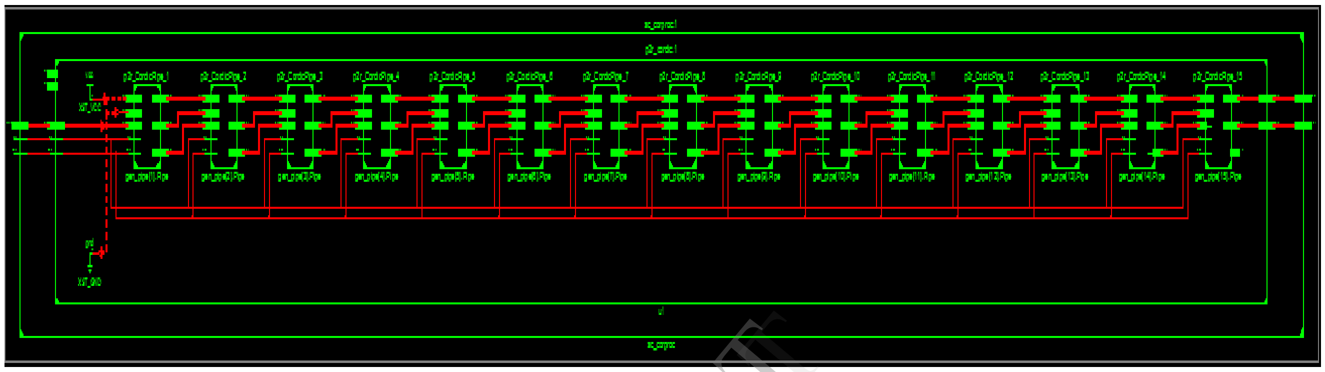


Figure 4: Pipelined stages inside CORDIC core for sine and cosine module

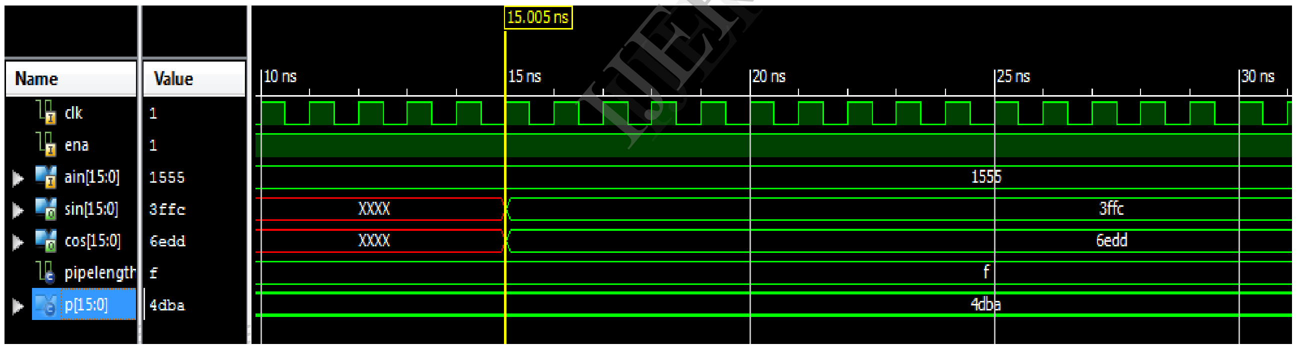


Figure 5: ISIM simulation result for sin/cosine block

Table 2: Synthesis results of sine/cosine pipelined CORDIC module

FAMILY	DEVICE	RESOURCE USAGE	MAX. CLOCK	DELAY
Spartan 3	xc3s50	605Slices 78%utilization	180.364MHz	5.544ns (49.3% logic, 50.7% route)
Virtex4	xc4vlx15	367 slices 5% utilization	399.637MHz	2.502ns (59.0% logic, 41.0% route)
Virtex5	xc5vlx30	690 slices 3% utilization	511.587MHz	1.955ns (71.9% logic, 28.1% route)
Virtex6	xc6vlx75t	723 slices 1% utilization	710.158MHz	1.408ns (71.2% logic, 28.8% route)

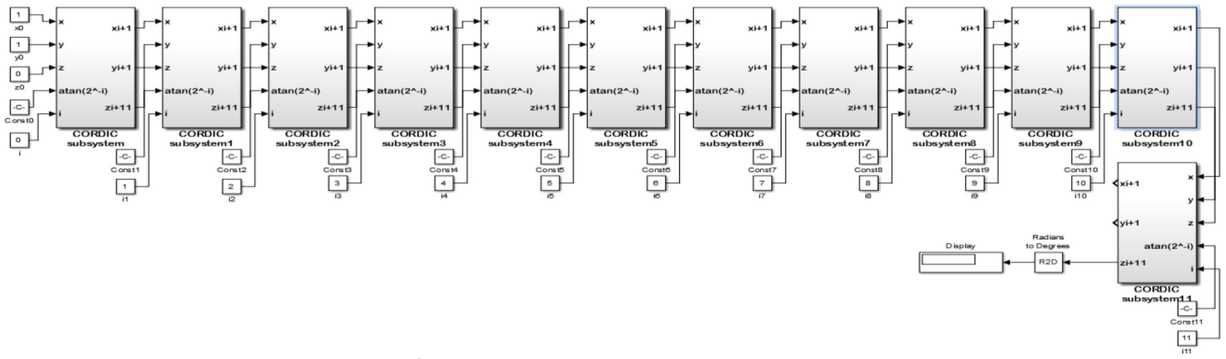


Figure 6: CORDIC implementation of Arctangent module in Simulink

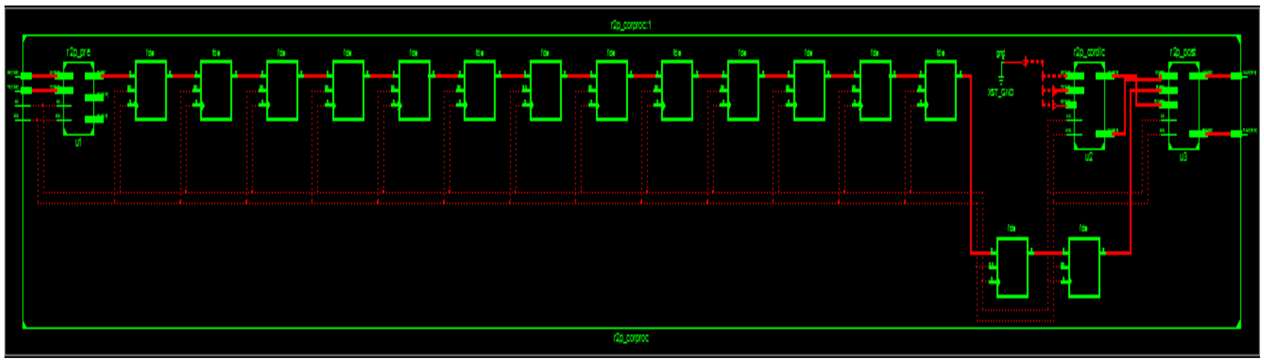


Figure 7: Pipelined stages inside CORDIC core for arctangent module

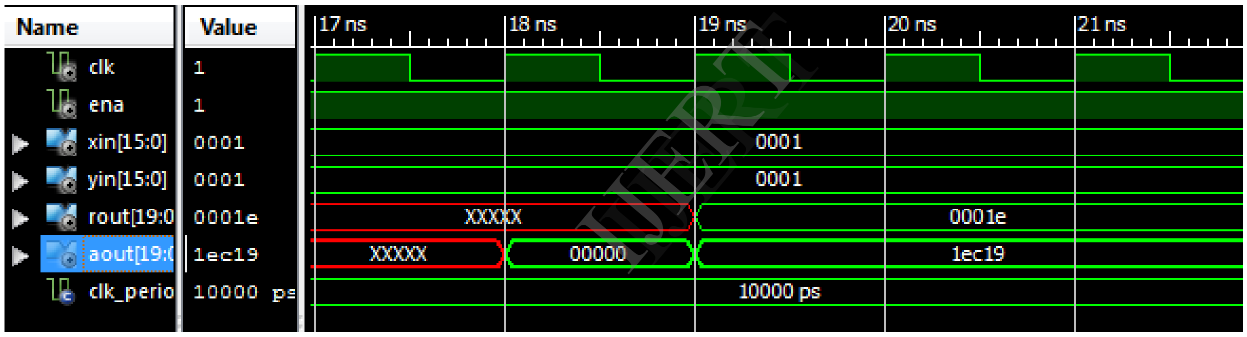


Figure 8: ISIM simulation result for arctangent block

Table 3: Synthesis results of arctangent pipelined CORDIC module

FAMILY	DEVICE	RESOURCE USAGE	MAX. CLOCK	DELAY
Spartan 3	xc3s50	605Slices 78% utilization	152.36MHz	6.563ns (49.3% logic, 50.7% route)
Virtex4	xc4vlx15	603 slices 9% utilization	318.388MHz	4.584ns (55.4% logic, 44.6% route)
Virtex5	xc5vlx30	1031 slices 5% utilization	393.811MHz	2.539ns (53.5% logic, 46.5% route)
Virtex6	xc6vlx75t	1019 slices 1%utilization	579.791MHz	1.725ns (55.9% logic, 44.1% route)

REFERENCE

[1] J. Volder, "The CORDIC Trigonometric Computing Technique," IRE Transactions on Electronic Computers, vol. EC-8, no. 3, pp. 330-334, 1959.

[2] J.Walther, "A Unified Algorithm for Elementary Functions," Proceedings of Spring Joint Computer Conference, vol. 38, pp. 379-385, 1971.

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

[4] P. Meher, J. Valls, T. Juang, K. Sridharan 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.

[5] T. Lang and E. Antelo, "High-throughput CORDIC-based geometry operations for 3D computer graphics," *IEEE Trans. Computers*, vol. 54, no. 3, pp. 347–361, Mar. 2005.

[6] J. Euh, J. Chittamuru, and W. Burleson, "CORDIC vector interpolator for power-aware 3D computer graphics," in IEEE Workshop on SignalProcess. Syst., SIPS'02, Oct. 2002, pp. 240–245.

[7] Mihai Simaa, Daniel Iancu and John Glossner "Software-Based Geometry Operations for 3D Computer Graphics", in Proceedings of the 3rd Workshop on Applications Specific Processors (WASP'04), pp. 53–58, (Stockholm, Sweden), September 2004.

[8] Richa Upadhyay, Dr. Nisha Sarwade, Shrugal Varde "Simulink design of pipelined CORDIC for generation of sine and cosine values", International Journal of Computational Engineering Research, Vol-3, Issue-3, pp 312-316, March-2013