E-ISSN 2581 - 7957
P-ISSN 2277 - 3916

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1910

FPGA Implementation of CORDIC – I using Redundant Arithmetic

Niharika Chaudhary[1] and T. Subha Sri Lakshmi[2]
[1]PG Scholar, CVR College of Engineering/ECE Department, Hyderabad, India
Email: niharikaodf@gmail.com
[2]Asst. Professor, CVR College of Engineering/ECE Department, Hyderabad, India
Email: subha.sri@cvr.ac.in

Abstract: **This paper provides high speed, low power consumption, and less area utilization of the Coordinate Rotation Digital Computer (CORDIC) Algorithm for digital signal processing applications. Here methodology is built on a multiplexer-based, that is used to accomplish the fast and efficient hardware on FPGA for sine and cosine values. A 6-stage CORDIC is calculated by four arrangements scheduled i.e., Unrolled CORDIC and MUXes based CORDIC with and without pipelining up to three stages. The proposed architecture has adders, subtractors, and shifters. Shifters are replaced with multiplexers up to 3-stages. All remaining adders and subtractors are traded with Redundant Arithmetic. A 16-bit CORDIC algorithm is designed to achieve the sine and cosine function values by using VIVADO 20.1. Comparisons are performed between Unrolled CORDIC structure and MUXes based CORDIC structure for sine and cosine values in terms of timing and power consumed. MUXes based CORDIC structure attains high operating frequency, less area utilization, and low power consumption for hardware implementation.**

Index Terms: **CORDIC Algorithm, Rotation Mode, Multiplexer, Pipelining, Redundant Arithmetic, Unrolled CORDIC**

I. Introduction

In 1956 the CORDIC algorithm was presented by Jack Volder while building a real-time navigator [1]. CORDIC is simple and efficient because requires only addition, subtraction, shifting of bits, and a lookup table. Sine and Cosine can be derived from any complex functions. These functions are used in a wide range of applications such as DSP, wireless communication, biometrics, robotics, etc. CORDIC method is castoff to generate hardware that performs sine and cosine calculations. CORDIC is used in a wide variety of elementary transcendental functions involving exponentials, logarithms, and square roots. Thereafter it was polished by Walter and others. In this algorithm, the angle is broken into the sum of angles and micro rotated by predefined angles. The CORDIC algorithm is implemented in the FPGA platform because the speed and computational power of ASICs are merged with the resilience of microprocessors [3]. CORDIC algorithm is castoff in various applications such as calculators, mathematical coprocessor units, clock recovery circuits, waveform generators. The iterative architecture provides hardware implementation with minimum size and throughput as a trade-off, while parallel and pipelined CORDIC architecture offers high – speed and high-throughput computation.

CORDIC algorithm comprises diverse architectures for mapping into hardware. These are grouped as folded and unfolded (as shown in figure 1). These operations are done by the knowledge of three iterative calculations [14]. First method architecture of folded implemented by copying every difference equation of CORDIC into time multiplexing and hardware. In a single functional unit, all these operations are done, so it will provide a trading area for speed [16]. It can be categorized into two kinds; folded bit-serial and folded word-serial architectures. It will check the functional unit implementation logic for every one bit or word of CORDIC in each iteration

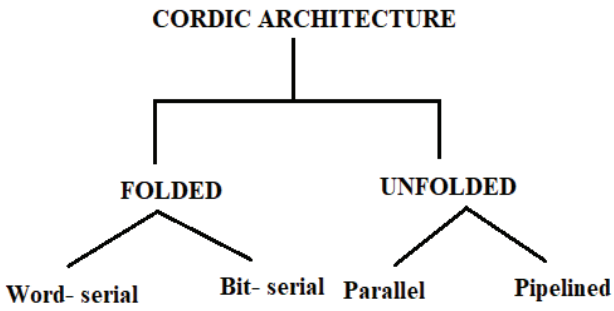

Figure 1. Nomenclature of CORDIC

Cordic is an iterative method of performing the microrotations for arbitrary angles using shifts and addition [1]. This MUXes based approach shows high speed and hardware efficient CORDIC, which can be used for DSP applications. A reconfigurable CORDIC can be operated in both rotation and vectoring mode. In this paper, only the rotation mode is discussed.

Nevertheless, the number of micro rotations is a serious downside of the critical path delay, the micro-rotation increases propagation delay In this paper, proposed CORDIC architecture the adders, subtractors, and shifters are replaced by multiplexers and redundant Arithmetic. CORDIC is designed in Unrolled CORDIC, pipelined Unrolled CORDIC, and MUXes based CORDIC up to 3 stages with and without pipelining. Replacement of the multiplexer and Redundant Arithmetic in the place of adders and shifters minimize the area of utilization on an FPGA, which intern reduces the power consumption, and surges the speed of operation.

This paper provides a prototype for implementing unrolled and pipelined architectures.

E-ISSN 2581 - 7957
P-ISSN 2277 - 3916

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1910

II. CORDIC ALGORITHM

The CORDIC programs on an iterative process that executes vector rotations by representing them as arbitrary angles by the application of shift and add operations [3]. The generalized rotation transform from the results of the Volder algorithm is given below [4].

$$f \sin \beta + r \cos \beta = r' \quad (1)$$

$$f \cos \beta - r \sin \beta = f' \quad (2)$$

This Cartesian plane switches by the angle β, as publicized in figure 2.

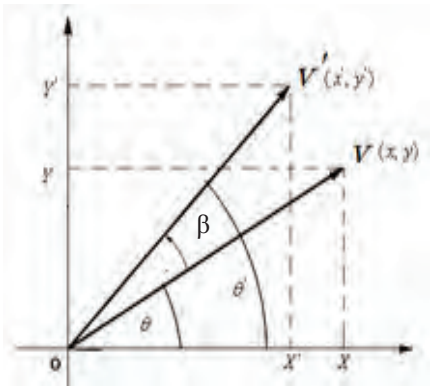

Figure 2. CORDIC angle rotation by φ.

The above equations can readapt as:

$$[\, r + f \tan \beta \,] \cos \beta = r' \quad (3)$$

$$[\, f - r \tan \beta \,] \cos \beta = f' \quad (4)$$

These rotations of angles are forced such that $\tan \beta = \pm 2^{-i}$. This will decrease tangent multiplication by the simple shift operation. By the execution of elementary rotations alternative rotation of arbitrary angles is managed. Through a sequence of micro-rotations of elementary angles (φ_i) CORDIC rotations are achieved. By decomposing β into elementary rotations in a sequence style the micro-rotations is achieved:

$$\beta = \Sigma \varphi_i \quad (5)$$

By using these we can form simple iterative rotations

$$[r_i + f_i \tan \varphi_i \,] \cos \varphi_i = r_{i+1} \quad (6)$$

$$[f_i - r_i \tan \varphi_i \,] \cos \varphi_i = f_{i+1} \quad (7)$$

From trigonometric identities we have:

$$\cos \varphi_i = 1/ (1+\tan^2 \varphi_i)^{1/2} \quad (8)$$

Substituting equation (8) in equation (6) and (7) we have:

$$r_{i+1} = [\, r_i + f_i \tan \varphi_i \,] / (\, 1 + \tan^2 \varphi_i \,)^{1/2} \quad (9)$$

$$f_{i+1} = [\, f_i - r_i \tan \varphi_i \,] / (1 + \tan^2 \varphi_i \,)^{1/2} \quad (10)$$

To guarantee that tangent multiplication reduced to a small shifting operation, the rotation angles obtained from the following relation:

$$\text{Tan } \varphi_i = \pm 2^{-i} \quad (11)$$

Here i represent the total number of iterations. And $\tan \varphi_i$ is traded in above equation by substituting equation (12) in (9) and (10), we have:

$$r_{i+1} = [\, r_i + f_i \,.k_i.\, 2^{-i}].J_i \quad (12)$$

$$f_{i+1} = [\, f_i - r_i \,.k_i.\, 2^{-i}].J_i \quad (13)$$

Here,

$1/ (1+2^{-2i})^{1/2} = J_i$, denoted as scale of constant.

$k_i = \pm 1$, where k_i is decision function.

The constant scale value from obtained equations of vector rotation is eradicated by a shift-add algorithm. In the system, the product term J_i's is applied. Through the use of iterations in infinite times this product value has a range up to 0.6073. Rotation algorithm gain is represented as A_n where,

$$A_n = \Pi [1+2^{-2i}]^{1/2} \quad (14)$$

The gain is unevenly equal to 1.647 for infinite iterations. However, this gain is calculated with the support of the total number of iterations, and also it depends on the compound rotation angle is discrete by a sequence of elementary rotations in the way of the direction. These sequences are denoted as decision vectors. And all these vectors, are used for the angular calculation system, depending on the values of binary arctangents. Translations are completed utilizing a look-up table among the angular system and any other systems. For each iteration, an enhanced conversion technique is employed by adding subtractor and adder unit to it for elementary rotation angles. The above angles are expressed by an appropriate angular unit. A small lookup table supplies the angular values, and also we can use hardwired, depending on the suitable implementation.

Accumulator angle is added to the third value of difference equation to CORDIC algorithm:

$$z_i - \varphi_i = Z_{i+1}$$

$$Z_{i+1} = z_i - k_i \tan^{-1} (2^{-i})$$

E-ISSN 2581 - 7957
P-ISSN 2277 - 3916

DOI: 10.32377/cvrjst 1910

Hence, the CORDIC micro-rotation equations are inscribed as:

$$x_{i+1} = x_i - y_i . 2^{-I}.k_i \quad (15)$$

$$y_{i+1} = y_i + x_i . 2^{-I}.k_i \quad (16)$$

$$z_{i+1} = z_i - k_i \tan^{-1} (2^{-i}) \quad (17)$$

III . CORDIC MODES

The CORDIC algorithm is divided into two modes. Rotation mode is the one [15], in which the input vector value is rotated by a specified angle. The second one is Vectoring mode where the input vector is rotated to the x-axis.

A. Rotation Mode

In this mode, some favored rotation angle is set by the angle accumulator. The trigonometric, hyperbolic, or some other transcendental values are found through these rotation angles as the argument. Angle accumulators will take care of the rotation decision. Here, the decision is made and evaluated at every rotation.

CORDIC equations in this model are written as:

$$l_i - h_i . 2^{-I}.k_i = l_{i+1}$$

$$h_i + l_i . 2^{-I}.k_i = h_{i+1}$$

$$t_i - k_i \tan^{-1} (2^{-i}) = t_{i+1}$$

Here,

$$k_i = -1 \quad \text{if } t_i < 0 \text{ +1, else}$$

n iterations after it produces the following results:

$$t_n = 0$$

$$[y_1 \cos z_1 + x_0 \sin z_1] An = H_n$$

$$[x_1 \cos z_1 - y_1 \sin z_1] An = L_n$$

B. Vectoting Mode

In vectoring mode, the input vector of the CORDIC rotator is circled through whichever angle is essential to align the resultant vector with the *x*-axis. The outcome of the vectoring operation is a scaled magnitude of the original vector (the *x* component of the result) and rotation angle. At every rotation, the vectoring function works by minimizing the *y* component of the residual vector. The sign of the residual *y* component is used to determine which direction to rotate next. If the angle accumulator is initialized to zero, it holds the traversed angle at the end of the iterations.

A CORDIC equation in vectoring mode follows:

$$q_{j+1} = q_j - m_j . w_j . 2^{-j}$$

$$m_{j+1} = m_j + q_j . w_j . 2^{-j}$$

$$f_{j+1} = f_j - w_j \tan^{-1} (2^{-j})$$

Here,

$$w_j = +1 \quad \text{if} \quad m_j < 0 \text{ -1, else}$$

n iterations after it produces following results:

$$q_n = A_n (q_0^2+m_0^2)^{1/2}$$

$$f_n = \tan^{-1}(m_0 / q_0) + F_0$$

$$m_n = 0$$

IV. REDUNDANT ARITHMETIC

In implementations as the computations always start from the most significant bit (MSB). Redundant number systemAdders play a major role in CORDIC and due to carry propagation in adders the delay increases rapidly and slows down the speed of operation so that, move on to Redundant Arithmetic to decrease the delay and increase the speed of operation. The conservative tasks like subtraction, multiplication, and addition produce carry-propagation chains. A redundant number scheme was announced to resolve this problem [10]. The redundant number scheme improves the arithmetic operations speed. This method is used for sign processing and additional applications. When the reconversion and conversion circuitry shares the information among all the function units, this method also saves the area in VLSI and also power dissipation, due to these two reasons system will become more effective. Redundant number systems (RNS) suitable for numerically intensive applications. RNS can prevent or captures the carry propagation, by generating parallel adders with the delay of constant value; it won't depend on the operand word-length. This will be produced in an RNS format by using low latency results. RNS can improve performance in mathematically intensive applications. However, the implementation of an arithmetic circuit is expensive because for each symbol multiple bits are required. These circuits will eliminate carry propagation, by giving near-constant addition delay, regardless of the operand width. The Redundant number system (RNS) has a unique property of "carry-free" addition which makes them beneficial in implementations as the computations always start from the most significant bit (MSB).

A. Carry- Free Addition Radix-2

Redundant number representations limit the carry propagation to a few bit positions and are usually independent of word length W. This carry free propagation feature enables fast addition

The logic implementation is varied because the algorithm for signed binary digit addition is not unique. By using two binary unsigned numbers, it can perform the radix-2 operation, one bit is negative and another bit is positive and

E-ISSN 2581 - 7957
P-ISSN 2277 - 3916

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1910

it can be represented as [11]

$$X = X^{+} - X^{-} \quad (18)$$

x_i^+ and x_i^- are both negative as well as positive numbers these bit values are 0 and 1, x_i should vary {1,0,1}, all these values are given in Table I.

TABLE I.
REDUNDANT NUMBER SYSTEM OF RADIX-2

x^+	x^-	X	$X = x^+ x^-$
0	0	0	00
0	1	-1	01
1	1	1	10
1	1	0	11

B. Hybrid addition Radix-2

In this hybrid operation, the 2 input operands are a redundant signed-digit representation and conventional unsigned number. The output operand obtained is in redundant signed-digit representation. For The signed-digit number addition $X_{<2.1>}$ is considered which is a radix-2 operation, where 2 indicates the radix-2 job, and 1 indicates the largest digit value and an unsigned conventional number Y.

$$X_{<2.1>} + Y = S_{<2.1>} \quad (19)$$

In 2 steps we can get an added value. Here 1st step all the bits are in parallel positions i ($0 \le i \le W-1$), W being the word length. The intermediary sum $p_i = x_i + y_i$ is calculated, it ranges between {1, 0, 1, 2}. This addition can be

$$x_i + y_i = p_i = 2t_i + u_i, \quad (20)$$

Table II summarizes hybrid radix-2 addition, in that table t_i denotes transfer digit and it varies value from 0 or 1, and it is also represented as t_i^+ and u_i denotes interim added sum and it varies the values either 1 or 0, and it is also represented as u_i^-.

TABLE II.
SUMMARIZES THE DIGIT SETS INVOLVED IN HYBRID RADIX-2 ADDITION

Digit	Binary Code	Radix 2 Digit Set
x_i	$x_i^+ - x_i^-$	{1,0,1}
y_i	y_i^+	{0,1}
$p_i = x_i + y_i$	.....	{1,0,1,2}
u_i	$-u_i^-$	{1,0}
t_i	t_i^+	{0,1}
$s_i = u_i + t_{i-1}$	$s_i^+ - s_i^-$	{1,0,1}

The most significant interim sum digit u_w has a value zero, the same as the least significant transfer digit t_{-1}.

The digit sum s_i is designed by linking t^+_{i-1} and also u_i^-, which is one of the single-digit in the second step:

$$s_i = t^+_{i-1} - u_i^-. \quad (21)$$

Replacing the corresponding binary codes from Table 2 in (3a) we get:

$$x_i^+ - x_i^- + y_i^+ = 2t_i^+ + u_i^- \quad (22)$$

These all operations are performed by using type-1 full adder [12], it is nothing but plus-plus-minus adder (PPM) [13] as shown in figure 3. The four-digit hybrid radix-2 adder is shown in figure 4.

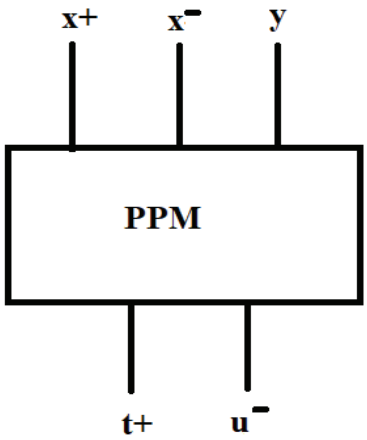

Figure 3. Hybrid Radix 2 PPM Adder

E-ISSN 2581 - 7957
P-ISSN 2277 - 3916

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1910

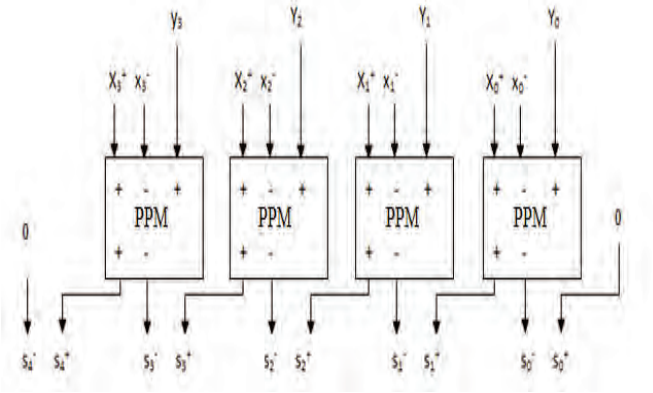

Figure 4. Four Digit Hybrid Radix-2 Adder

C. Hybrid Radix-2 subtraction

MMP subtractor does subtraction on a redundant binary signed digit number system. To draw high-speed systems this subtractor shown in Figure 5 is beneficial as it permits "borrow free" subtraction. It carries out subtraction of a redundant number x where, x = x- and x+ to an unsigned binary number y, resulting in another redundant number expressed by an interim sum b^- and a transfer digit t^+. the input bits are defined as x+, x-, y ε {0,1}, and the output bits are $b^-, t^+ \varepsilon$ {0,1}.

The following operations are performed by subtractor:

$$x - y = x^+ - x^- - y \Rightarrow$$

$$b^- - 2t^+ \quad (23)$$

where x is a redundant number expressed as

$$x = x^+ - x^-$$

Therefore,

$$x^+ - x^- - y = b^- - 2t^+ \quad (24)$$

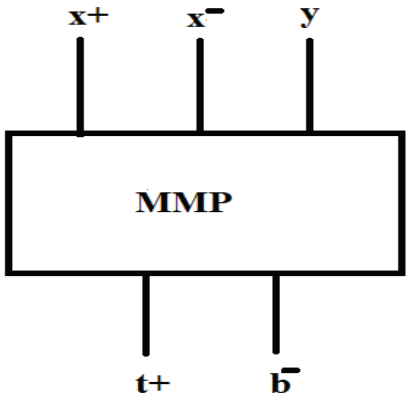

Figure 5. Hybrid Radix 2 MMP Subtractor

The interim sum b^- and the transfer digit t^+ is stated by the following Boolean expression using the Truth table:

$$b^- = (x^+)'(x^-)y' + (x^+)'(x^-)y + (x^+)(x^-)'y' + (x^+)(x^-)y$$

$$t^+ = (x^+)'(x^-)y' + (x^+)(x^-)'y' + (x^+)'(x^-)y + (x^+)(x^-)y$$

After simplification of the above equations, a new the equation for the interim sum b^- and the transfer digit t^+ is

$$b^- = x^+ \overline{\oplus}\, x^- \overline{\oplus}\, y$$

$$t^+ = x^+ \cdot (x^+ \oplus x^-) + y(x^+ \overline{\oplus}\, x^-) \quad (25)$$

V. Unrolled Cordic Algorithm

CORDIC algorithm calculates the sine and cosine values of input angles concurrently in rotation mode. Figure 6 shows the unrolled CORDIC. It carries redundant adders, and subtractors, and shifters respectively. The subtraction or addition of angle succeeded based on the MSB of the previous angle in every rotation of the vector. The right shifts for division are executed by shift registers. Initially, for sine and cosine angles x_i=1 and y_i= 0. These initial values are shifted by i bits, where i= {1,2,3,4,5,6} which is divided by 1,2,4,6,8,16,32 at each stage. Discrete sine and cosine values range from -1 to 1 [18].

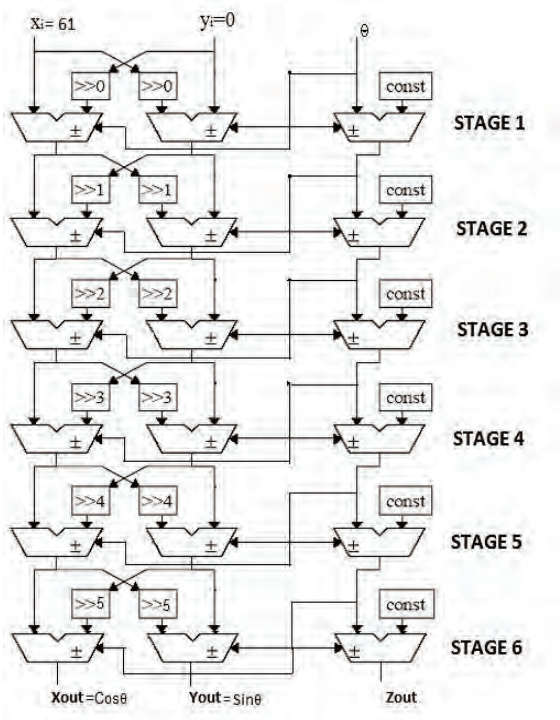

Figure 6. Construction of general unrolled CORDIC

As initial conditions has the outcome of discrete sine and cosine values varying from -1 to 1, so the fractional values are realized in FPGA by -100 to 100. z_i is varied for every clock pulse to generate sine and cosine values. The rest values are computed by using the quadrature symmetry property of sine and cosine waves.

E-ISSN 2581 - 7957
P-ISSN 2277 - 3916

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1910

VI. Pipelined Unrolled Cordic Algorithm

Pipelining is an implementation technique where a bundle of data processing instructions is overlapped. These instructions are given in a series. The pipeline process maximum frequency of operation in CORDIC. The architecture of the pipelined unrolled CORDIC is shown in Figure 7 [19,20].

Enlarged area and N-clock delays are the disadvantages of pipeline architecture. Hence, several pipelined registers and their positions are computed repetitively.

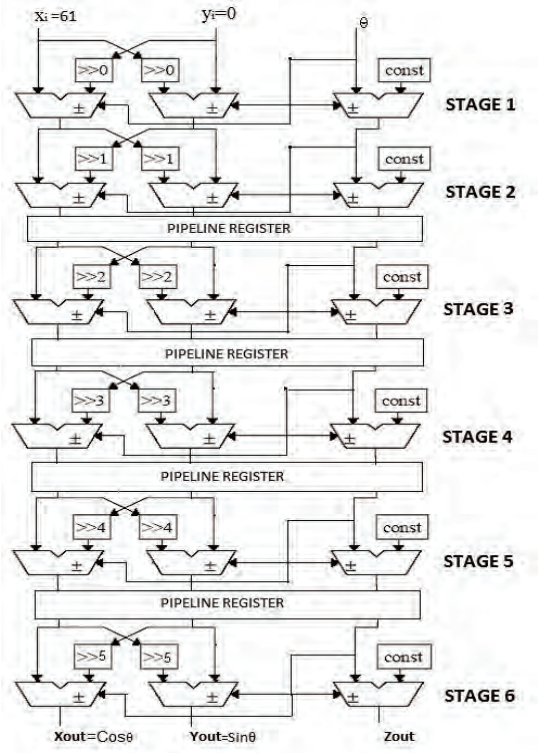

Figure 7. The architecture of Pipelined unrolled CORDIC

Four intermediate stages of pipelined registers are used to get optimized output.

VII. Multiplexer Based Unrolled Cordic Algorithm

CORDIC architecture's area is reduced via multiplexer [20]. A multiplexer is used in the place of three stages in general unrolled CORDIC. The output is equal to X_i as Y_i=0 in the first stage in original unrolled CORDIC architecture so the 1st stage's output is given as:

$$Y_1 = X_i = 61$$
$$X_1 = X_i = 61$$
$$Z_1 = Z_i - 45 \quad (26)$$

In the first iteration stage, Z_1 is calculated by subtraction since Z_i is always positive as it varies from 0 to 90.

If Z_1 is positive, then the second stage output is described as

$$Y_2 = Y_1 - (X_1/2) = (X_1/2) = 31$$
$$X_2 = X_1 + (Y_1/2) = 3X_1/2 = 91 \quad (27)$$

If Z_1 is negative, then the second stage output is

$$Y_2 = Y_1 + (X_1/2) = 3X_1/2 = 91$$
$$X_2 = X_1 - (Y_1/2) = X_1/2 = 31 \quad (28)$$

Figure 8 shows two multiplexers used for the second stage.

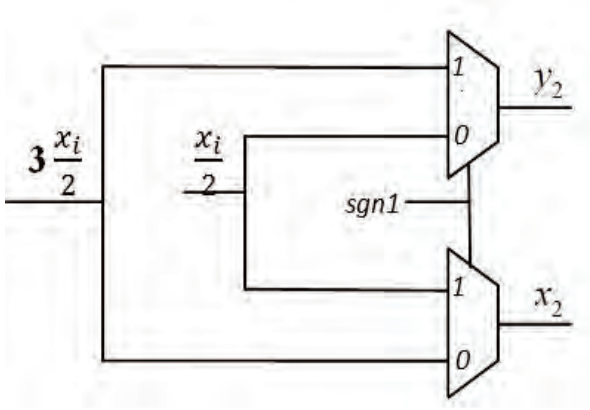

Figure 8. 2 multiplexers used for the second stage

Equivalently, by using 4 multiplexers the third stage is implemented with the following equation. Z_2 is computed by the formula

$$Z_{n+1} = \theta - \sum_{i=0}^{n}\theta_i = \theta - \sum_{i=0}^{n} arc\ tan\left(\frac{1}{2^n}\right) \quad (29)$$

For the third stage multiplexers, the above equation is used as the selection line input of the multiplexer.

For Z_1 = positive , Z_2 = positive

$$Y_3 = Y_2 + (X_2/4) = (3X_1/2) + (X_1/8) = 13X_1/8 = 99$$
$$X_3 = X_2 + (Y_2/4) = (3X_1/2) - (X_1/8) = X_1/8 = 7 \quad (30)$$

For Z_1 = negative , Z_2 = positive

$$Y_3 = Y_2 + (X_2/4) = (X_1/2) + (3X_1/8) = 7X_1/8 = 99$$
$$X_3 = X_2 + (Y_2/4) = (3X_1/2) - (X_1/8) = 11X_1/8 = 83 \quad (31)$$

For Z_1 = positive , Z_2 = negative

$$Y_3 = Y_2 - (X_2/4) = (3X_1/2) - (X_1/8) = 11X_1/8 = 83$$
$$X_3 = X_2 + (Y_2/4) = (3X_1/2) + (X_1/8) = 7X_1/8 = 53 \quad (32)$$

For Z_1 = negative, Z_2 = negative

$$Y_3 = Y_2 - (X_2/4) = (X_1/2) + (3X_1/8) = X_1/8 = 7$$
$$X_3 = X_2 + (Y_2/4) = (3X_1/2) + (X_1/8) = 13X_1/8 = 99 \quad (33)$$

The area is minimized as adders with 2:1 multiplexers are swapped up to the 3rd stage. There is an exponential increase in multiplexers i.e, 6,14,30 multiplexers for 3, 4, and 5stages as adders and shifters are replaced with mux. The fixed values expand due to the growth in multiplexers. The deletion of the 5th stage needs thirty muxes with sixteen

E-ISSN 2581 - 7957
P-ISSN 2277 - 3916

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1910

fixed values. On that occasion, utilizing a ROM is more effective.

The CORDIC-I algorithm runs on rotation mode whose input is Y_i=0 and X_i=1. The equation to used is

$$f_{i+1} = k_i . [f_i - r_i . d_i . 2^{-i}]$$

$$r_{i+1} = k_i . [r_i + f_i . d_i . 2^{-i}]$$

At this time, k_i is the scaling factor with 0.611 is multiplied with the input X_i= 1 Discrete sine and cosine values are varied from -100 to 100 for FPGA realization.

Equations 30-33 are solved by taking input X_i= 61 and Y= 0.

The obtained values are

- P = x_i /8 = 8
- Q = 11 x_i /8 = 84
- R = 7 x_i /8 = 53
- S = 13 x_i /8 = 99

Figure 9 shows the architecture of unrolled CORDIC based on 2:1 mux.

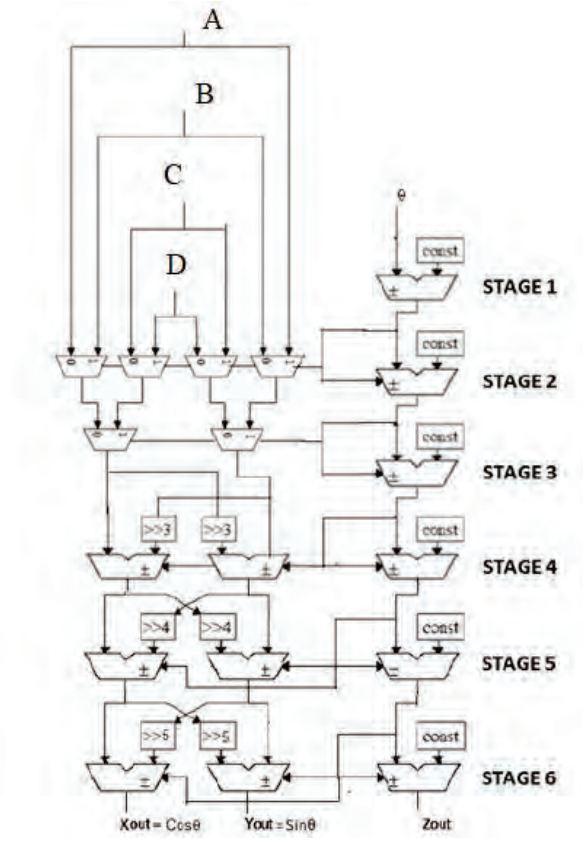

Figure 9. The architecture of unrolled CORDIC with MUX

VIII. Multiplexer Based Pipelined Unrolled Cordic Algorithm

The multiplexer based pipeline CORDIC utilizes the same computation as used for unrolled CORDIC with a multiplexer. Architecture is shown in Figure 10.

Here, subtractors, adders, and shifters are swapped up to 3 stages with a multiplexer.

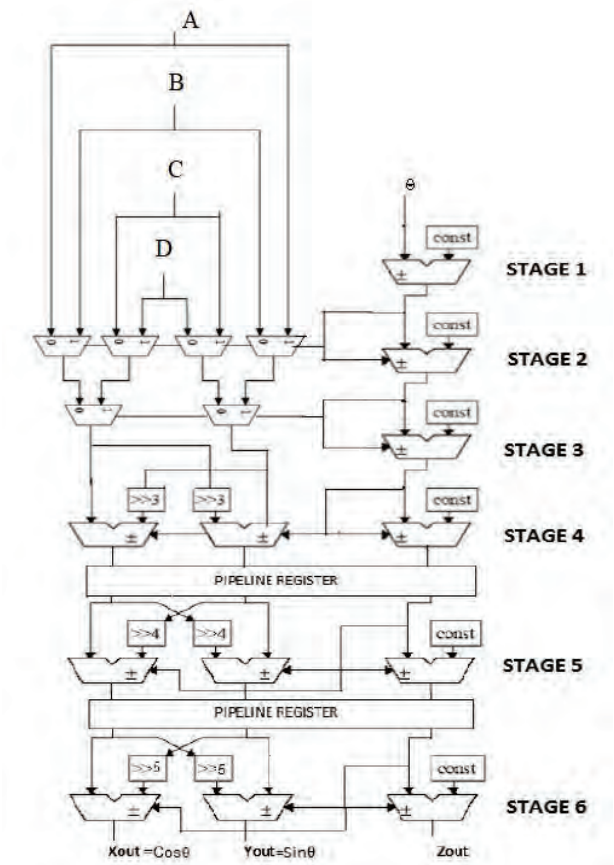

Figure 10. The architecture of Pipelined unrolled CORDIC with MUX

IX. Results

An 8-bit CORDIC for constructing sine and cosine function with and without pipelining based on Unrolled and multiplexer based CORDIC. The initial design entry is finished using VERILOG. The core is implemented with the following synthesis description:

- Design Entry: VERILOG
- Synthesis and Simulation: VIVADO 18.1
- Platform: FPGA
- Family: Zynq-7000
- Target board: xc7z020clg484-1
- Optimization area: Power

Figure 11 is the implemented result of the Unrolled CORDIC architecture schematic obtained from the tool.

In Figure 12 “clk” is the input signal along with x, y, and z is the angle which is 60. “sin” is discrete sine values, “cos” is the discrete cosine values and “zout” is the microrotation

E-ISSN 2581 - 7957
P-ISSN 2277 - 3916

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1910

angle obtained as output. The addresses are the stored predefined values of the CORDIC angles.

Calculation of zout : $60^0 - 45^0 = 15^0$
$15^0 - 27^0 = -12^0$
$12^0 + 14^0 = 2^0$
$2^0 - 7^0 = -5^0$
$-5^0 + 4^0 = -1^0$
$-1^0 + 2^0 = 1^0$

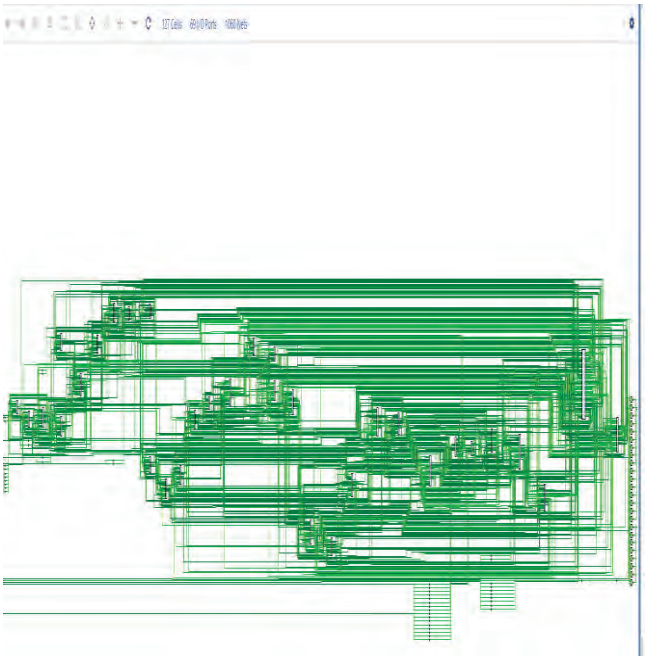

Figure 11. Schematic of Unrolled CORDIC

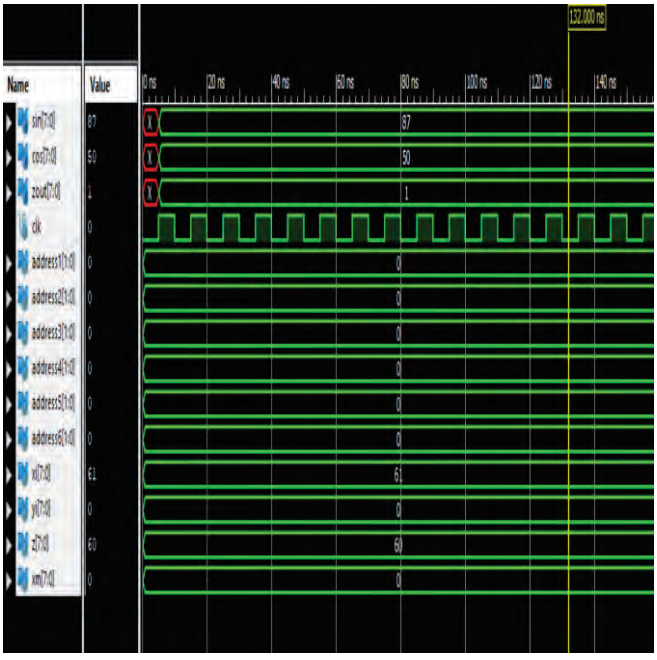

Figure 12. Simulation Result of Unrolled CORDIC

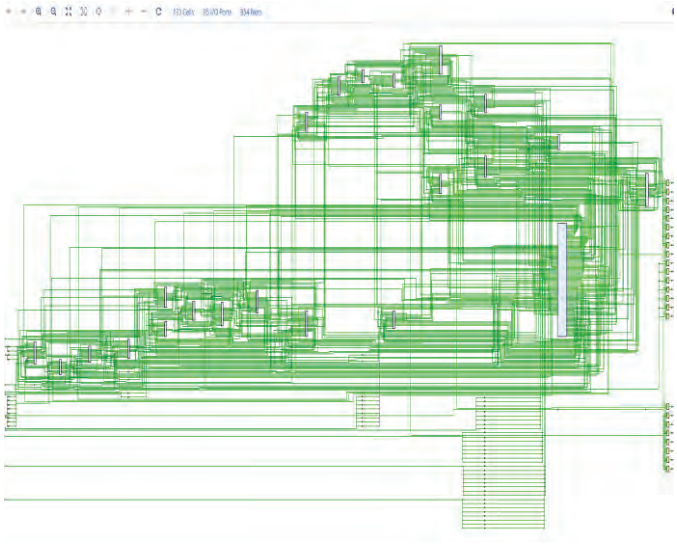

Figure 13. Schematic of Pipelined Unrolled CORDIC

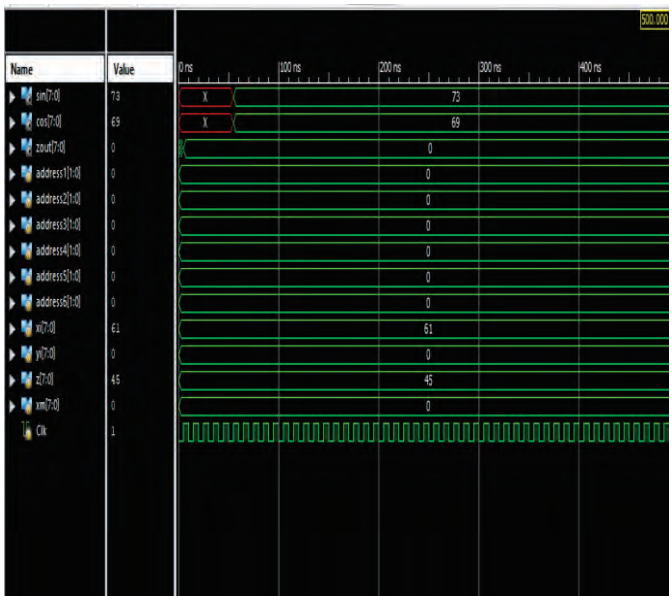

Figure 14. Simulation Result of Pipelined Unrolled CORDIC

Figure 13 is the implemented result of the pipelined unrolled CORDIC architecture schematic obtained from the tool.

In Figure 14 “clk” is the input signal along with x, y, and z is the angle which is 45. “sin” is discrete sine values, “cos” is the discrete cosine values and “zout” is the microrotation angle obtained as output.

The addresses are the stored predefined values of the CORDIC angles.

Figure 15 is the implemented result of unrolled CORDIC using MUXes schematic obtained from the tool.

In Figure 16 “clk” is the input signal along with x, y, and z is the angle which is 45. “sin” is discrete sine values, “cos” is the discrete cosine values and “zout” is the microrotation angle obtained as output.

The addresses are the stored predefined values of the CORDIC angles.

E-ISSN 2581 - 7957
P-ISSN 2277 - 3916

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1910

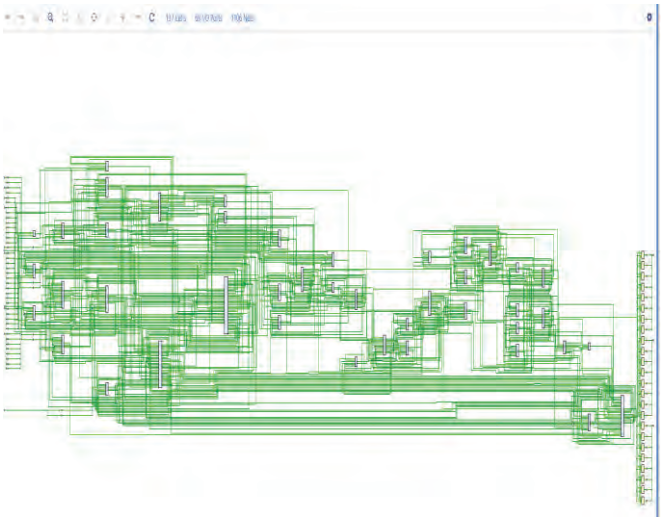

Figure 15. Schematic of Unrolled CORDIC using MUXes

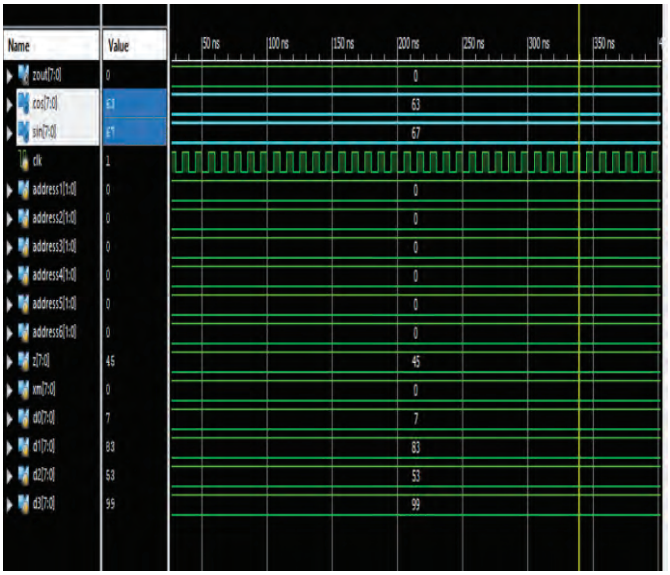

Figure 16. Simulation Result of Unrolled CORDIC using MUXes

Figure 17. Schematic of Pipelined Unrolled CORDIC using MUXes

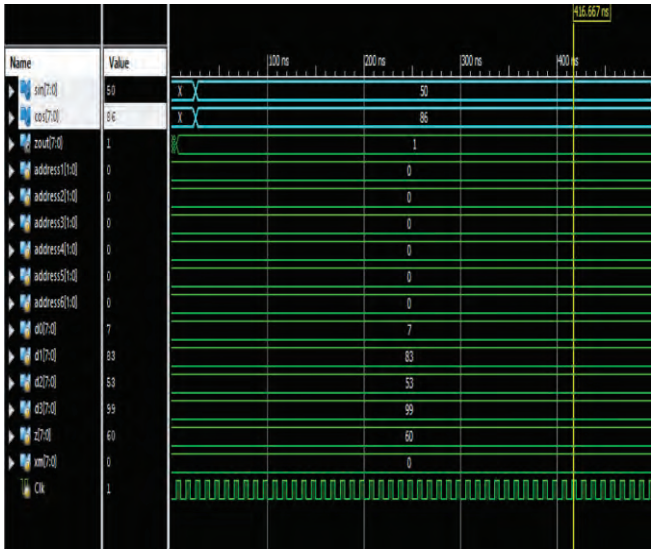

Figure 18. Simulation Result of Pipelined Unrolled CORDIC using MUXes

Figure 17 is the implemented result of pipelined unrolled CORDIC using with MUXes schematic obtained from the tool. In Figure 18 “clk” is the input signal along with x, y, and z is the angle which is 60. “sin” is discrete sine values, “cos” is the discrete cosine values and “zout” is the microrotation angle obtained as output.

The addresses are the stored predefined values of the CORDIC angles.

TABLE III.
COMPARISON OF FOUR SCHEMES BASED ON RESULTS OBTAINED IN VIVADO IMPLEMENTATION

	SIN 60	COS 60	SIN 45	COS 45	No.of Slices used (Area)	Power (W)	Off set Timing (nS)
Unrolled CORDIC Without Mux	50	87	73	69	580	48.6	60.21
Unrolled CORDIC With Mux	50	86	63	67	427	43.1	53.42
Pipelined CORDIC Without Mux	50	87	69	73	594	42	53.59
Pipelined CORDIC With Mux	50	86	63	67	414	38.7	53.51

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1910

In this paper proposed algorithm achieves high speed, less hardware implementation, and less power consumption on FPGA. Table III shows a comparison of all four architectures in terms of area and power. The area and power are reduced because the adders, subtractors, and shifters are replaced with the multiplexer up to 3 stages. This lessens the complexity of architecture by which it runs faster and efficiently. Thus, the decrease in area is shown in Table III as the number of slices.

X. Conclusions

This paper discusses 8-bit CORDIC using the Unrolled and multiplexer-based architectures with and without pipelining for generating the sine and cosine values. A 6-stages CORDIC is implemented with unrolled CORDIC, pipelined CORDIC general, and Multiplexer based architecture up to 3 stages. From the results observations, it is found that the multiplexer-based approach operates on only 11% of the total area when compared with without MUXes which uses 16%. Therefore, 25% of the circuitry can be detached when 3 stages are eliminated. In terms of Power, the pipelined Unrolled CORDIC MUXes based utilizes only 39% of the total power available on FPGA. The pipelined Unrolled CORDIC based on MUXes has a maximum frequency of 88.75MHz which is relatively good as compared with others. As shown in Table III, the pipelined Multiplexer built CORDIC algorithm decreases equally area and power but increases the speed of operation. Swapping the multiplexer in the place of adders, subtractors, and shifters up to 3 stages and replacing the adders and subtractors of all 6-stages with Redundant Arithmetic reduces area utilization and power consumption on FPGA and increase the speed of operation. Henceforth, built on a user-defined application, any one of the 4 methodologies can be selected.

References

[1] B. Khurshid, G.M.Rather and N.Hakim, “Performance Comparison of Non-redundant and Redundant FPGA based Unfolded CORDIC Architectures, “in International Journal of Electronics and Communication Technology, vol. 3, issue 1 pp 85-89, March 2012.

[2] Volder J. E., “The CORDIC trigonometric computing technique”, IRE Trans. Electronic Computing, Volume EC-8, pp 330 - 334, 1959.

[3] Walther J. S., “A unified algorithm for elementary functions,” in Proceedings of the AFIPS Spring Joint Computer Conference, pp. 379–385, May 1971.

[4] Walther J. S., “The story of Unified CORDIC,” Journal of VLSI Signal Processing, vol. 25, no. 2, pp. 107–112, 2000.

[5] Andraka R, “A survey of CORDIC algorithms for FPGA based computers,” FPGA ’98, ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp 191-200.

[6] De Lange A. A. J., Van der Hoeven A. J., Deprettere E. F., and Bu J., “Optimal floating-point pipeline CMOS CORDIC processor,” Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS ’88), vol. 3, pp. 2043–2047, June 1988.

[7] Despain A. M., “Fourier transform computers using CORDIC iterations,” IEEE Transactions on Computers, vol. 23, no. 10, pp. 993–1001, 1974.

[8] Ahmed H. M., Delosme J. M., and Morf M., “Highly concurrent computing structures for matrix arithmetic and signal processing,” Computer, vol. 15, no. 1, pp. 65–82, 1982.

[9] Cavallaro J. R. and Luk F. T., “CORDIC arithmetic for an SVD processor,” Journal of Parallel and Distributed Computing, vol. 5, no. 3, pp. 271–290, 1988.

[10] Lee J. A. and Lang T., “SVD by constant-factor-redundant-CORDIC,” Proceedings of the 10th IEEE Symposium on Computer Arithmetic, pp. 264–271, June 1991.

[11] Meyer A., Watzel R., Meyer U., and Foo S., “A parallel CORDIC architecture dedicated to compute the Gaussian potential function in neural networks,” Engineering Applications of Artificial Intelligence, vol. 16, no. 7-8, pp. 595– 605, 2003.

[12] Kang C. Y. and Swartzlander E. E., “Digit-pipelined direct digital frequency synthesis based on differential CORDIC,” IEEE Transactions on Circuits and Systems I, vol. 53, no. 5, pp. 1035–1044, 2006.

[13] Antelo E., Lang T. and Bruguera J. D., “Very-High Radix CORDIC Rotation Based on Selection by Rounding,” Journal of VLSI Signal Processing, Kluwer Academic Publishers, Netherlands, Vol.25, 141.153, 2000.

[14] Delosme M. J.,Lau C. Y. and Hsiao S. F., “Redundant Constant-Factor Implementation of Multi-Dimensional CORDIC and Its Application to Complex SVD,” Journal of VLSI Signal Processing, Kluwer Academic Publishers, Netherlands, Volume 25, pp 155.166, 2000.

[15] Choi J. H., Kwak J. H. and Swartzlander, Journal of VLSI Signal Processing, Kluwer Academic Publishers, Netherlands, Volume 25, 2000.

[16] Meggitt J. E., “Pseudo division and pseudo multiplication processes” IBM Journal, vol. 6, no. 2, pp. 210–226, 1962.

[17] Deprettere E., Dewilde P., and Udo R., "Pipelined CORDIC Architecture for Fast VLSI Filtering and Array Processing," Proc. ICASSP'84, 1984, pp. 41.A.6.1- 41.A.6.4.

[18] M. Chinnathambi, N. Bharanidharan, and S. Rajaram, "FPGA implementation of fast and area efficient CORDIC algorithm," 2014 International Conference on Communication and Network Technologies, Sivakasi, 2014, pp. 228-232.

[19] P. Nilsson, "Complexity reductions in unrolled CORDIC architectures," 2009 16th IEEE International Conference on Electronics, Circuits, and Systems - (ICECS 2009), YasmineHammamet, 2009, pp. 868-871.

[20] V. Naresh, B. Venkataramani and R. Raja, "An area efficient multiplexer based CORDIC," 2013 International Conference on Computer Communication and Informatics, Coimbatore, 2013,pp.1-5.