Hardware Implementation of FFT Algorithm

A DISSERTATION

Submitted in partial fulfilment of the

requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

(With Specialization in Microelectronics and VLSI)

Submitted by

WADKAR SUMIT SATYAVIJAY

Under the guidance of

Dr. Bishnu Prasad Das

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

ROORKEE - 247667 (INDIA)

MAY, 2016

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being reported in this dissertation on,"**Hardware Implementation of FFT Algorithm"**, being submitted in the partial fulfilment of the requirements forthe award of the degree of **Master of Technology** in **Microelectronics & VLSI**, submitted in the Department of Electronics and Communication Engineering, Indian Institute of Technology Roorkee, Roorkee, India, is an authentic record of my own work carried out from May 2015 to May 2016 under the guidance and supervision of **Dr. Bishnu Prasad Das**, Assistant Professor, Department of Electronics and Communication Engineering, Indian Institute of Technology Roorkee, Roorkee.

The matter embodied in the dissertation to the best of my knowledge has not been submitted for the award of any other degree elsewhere.

Dated:

Place: Roorkee

_____

(Wadkar Sumit Satyavijay)

CERTIFICATE

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

_____

Dr. Bishnu Prasad Das

Assistant Professor,
Indian Institute of Technology Roorkee,
Roorkee.

ACKNOWLEDGEMENT

I would like to express my gratitude to my supervisor **Dr. Bishnu Prasad Das**, Assistant Professor, Department of Electronics and Communication Engineering, Indian Institute of Technology Roorkee for helping me to learn different aspects of my research work. I would like to thank him for assisting me to successfully complete this work.

I would like to thank the research scholars of Microelectronics and VLSI group especially Mr. Sannena Govinda for his constant guidance and support throughout my work to successfully complete this work. I would like to express my thanks to Miss. Priyamwada Sharma and Miss. Poorvi Jain for helping me in various stages of my work.

I would like to thank Dr. Sudeb Dasgupta, Associate Professor, Dr. Brajesh Kumar Kaushik, Associate Professor, Dr. Sanjeev Manhas, Associate Professor, Dr. Anand Bulusu, Associate Professor, Dr. Arnab Datta, Assistant Professor, and Dr. Brijesh Kumar, Assistant Professor, Department of Electronics and Communication Engineering for their help and support.

My special gratitude and love to my parents and friends for supporting and encouraging me throughout this work.

Wadkar Sumit Satyavijay

M. Tech (Microelectronics and VLSI)

Enrolment No: 14534016

ABSTRACT

The study investigates hardware implementation of Fast Fourier Transform (FFT), considering hardware complexity and computational accuracy. The direct computation of FFT involves complex multiplications. Thus, hardware implementation of FFT results in high hardware complexity. This problem can be addressed by exploiting CORDIC (COrdinate Rotation Digital Computer) algorithm in FFT implementation. In FFT computation, the CORDIC algorithm is used to perform complex multiplications. This converts the complex multiplications into shift and adds operations which are very easy to implement in hardware. Hence, this approach reduces the hardware complexity of FFT implementation. Various CORDIC algorithms available in literature are discussed in detail. Two new CORDIC architectures are proposed. Both the proposed CORDIC architectures are suitable for pipelined implementation. The CORDIC architecture proposed in chapter 2 uses unique angle set at each pipelined stage which results in latency of only 7 clock cycles. Further, suitable approximation of sine and cosine function using the terms with only negative power of two $\left(2^{-i}\right)$ makes the architecture completely scaling-free. On the other hand, the proposed CORDIC architecture in chapter 3 has the very low latency of only 5 clock cycles. This low latency can be achieved due to the use of efficient mixture of rotation angles at each pipelined stage. Additionally, the suitable use of Taylor series approximation of sine and cosine functions makes the architecture completely scaling-free. Performance of two proposed architectures is compared the other CORDIC architecture present in literature. These architectures are coded in VerilogHDL, synthesised in Xilinx ISE14.7 and mapped onto SPARTAN 3E XC3S500E-FG320-5 FPGA device. The comparison shows that proposed CORDIC architecture in chapter 2 has better accuracy, while the architecture proposed in chapter 3 has best performance in terms of slice-delay product. Finally, radix-2 four-point DIT-FFT is implemented in Virtex-4 XC4VLX25-FF668 FPGA device by exploiting two proposed CORDIC architectures. The comparison of these two implementations is performed by considering hardware utilization and error in computation of FFT. The outputs of FFT implementations are analysed and verified using Xilinx ChipScope Pro Analyser.

Table of Contents

List of Figures

List of Tables

CHAPTER-1

INTRODUCTION

1.1 Importance of FFT Algorithms and Applications

Among all the discrete transforms, Discrete Fourier Transform (DFT) is most widely used in digital signal processing. It allows representation of discrete time domain signal into frequency domain signal. DFT has widespread applications in spectral analysis of systems, LTI systems, calculation of convolution of signal, noise removal, multiplication of large polynomials etc. However, direct computation of N-point DFT has arithmetic complexity of $O(N^2)$and which takes considerable time. The complexity of computation of DFT can be reduced by exploiting Fast Fourier Transform (FFT) algorithms. In general FFT implementation of N-point DFT requires $(\mathrm{N}\log_2 \mathrm{N})$complex arithmetic operations. Hence FFT algorithms have contributed to the DFT implementation by VLSI chips.

1.2 Evolution of CORDIC algorithm and its use in FFT computation

It is often required to calculate trigonometric functions in signal processors, robotics and linear systems. Computation techniques such as lookup tables (LUTs) can be used for calculating trigonometric functions. The LUTs are faster but requires memory for storing the intermediate points and has limited precision. A Coordinate Rotation Digital Computer (CORDIC) algorithm in [1] efficiently calculates trigonometric functions by using vector rotation. It decomposes the target angle into small elementary angles and rotates the input vector through these elementary angles. Each rotation by an elementary angle is called micro-rotation, which consists of only shift-add operations. Hence, the CORDIC algorithm is very efficient method for calculation of trigonometric functions in hardware. This makes the CORDIC algorithm popular in areas such as computation of trigonometric functions, calculation of fast Fourier transform (FFT), discrete Hartley transform (DHT), discrete cosine transform (DCT), discrete sine transform (DST), digital signal processing (DSP), image processing and communication systems [2], [3], [5-9]. However, the major limitations of CORDIC algorithm are slow speed, scaling factor compensation and limited convergence range.

Many variations to conventional CORDIC algorithm have been available in the literature. These variations have proceeded mainly in two directions. One of these is on high speed solutions [10] while the other is on carefully handling of scale factor for high precision implementation [12]. In [11], modified vector rotational CORDIC (MVR-CORDIC) reduces the latency of algorithm by repeating and/or skipping some micro-rotations. However, it requires extensive search to find the right sequence of micro-rotations. This result in variable scale factor and hence scaling factor calculation and compensation is required. Modified virtually scaling-free adaptive CORDIC in [12] uses 2nd order Taylor Series approximation of sine and cosine functions but requires multiplication by constant scaling factor. Additionally,

because of pipelined implementation the area reduction achieved is not significant. Enhanced scaling-free CORDIC in [13] initially uses few conventional CORDIC iterations followed by scaling-free CORDIC iterations to reduce latency and area requirement as compared to [12]. However, scaling factor of an algorithm depends on the number of initial iterations of conventional CORDIC.

The work in [14] uses 3rd order approximation of Taylor series to make CORDIC completely scaling free with improved speed. Reconfigurable CORDIC in [15] can be configured for either circular or hyperbolic trajectories in rotation as well as vectoring mode. But area requirement of this algorithm is more as compared to general CORDIC algorithms. In [16] and [17], the scale factor compensation/correction techniques are introduced however they increase the latency of algorithm. ACORDIC II algorithm in [4] reduces the number of adders and decreases the latency of a basic CORDIC algorithm. However, the output of CORDIC II has a constant scaling factor. Hence, scaling factor compensation is necessary. Also, accuracy of the algorithm is degraded because of large scaling factor of rotation stages. A brief overview of major developments in the CORDIC algorithms and architectures is presented in [19], [20].

The basic idea of exploiting CORDIC algorithm in FFT computation is using a CORDIC unit as basic processing unit (PU) which performs butterfly operation. In [22] CORDIC algorithm is used to make FFT architecture area efficient for OFDM Digital Video Broadcasting. The multimode processor in [23] uses CORDIC algorithm to improve its area and energy efficiency.

1.3 Various CORDIC algorithms

In this section, basic principles of various CORDIC algorithms existing in literature are discussed.

1.3.1 Conventional CORDIC

The CORDIC algorithm was first coined in 1959 by Jack Volder. CORDIC stands for **CO**ordinate**R**otation **DI**gital**C**omputer. The CORDIC algorithm is an iterative method to implement various trigonometric and transcendental functions in hardware. The motive behind CORDIC is that each iteration of an algorithm consists of only add and shift operations; this reduces the cost of a hardware used to implement the corresponding function.

CORDIC Theory:CORDIC is an algorithm for vector rotation. All the trigonometric function can be computed with the help of vector rotation. In Figure 2.1 it is shown that a vector (0, 1) is rotated through an angle θ.After rotation let the coordinates of a vector are (x,y). Where $x = \sin\theta and y = \cos\theta$. Now if we again rotate a vector (x,y) though an angle Φ as shown in figure 2 we get new coordinates of the resultant vector, let them be (x',y'). Then x' and y' can be calculated as

$$x' = x\cos\varphi - y\sin\varphi and$$

$$y' = y\cos\varphi + x\sin\varphi \quad (1.1)$$

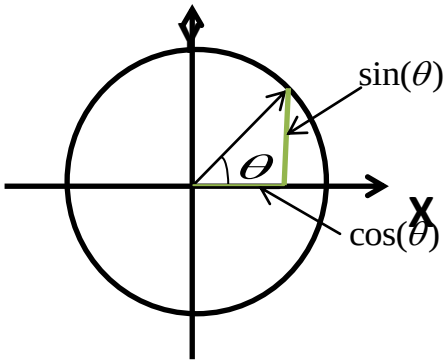

Figure 1.1 Vector rotation through an angleθ

This can be rearranged so that:

$$x' = cos\,\varphi[x - y\,tan\,\varphi] \tag{1.2}$$

$$y' = cos\,\varphi[y + x\,tan\,\varphi] \tag{1.3}$$

Now, what if we restrict the angle of rotation such that $\tan\varphi = \pm 2^{-i}$? Surprisingly this reduces the multiplication by the tangent term to simple shift operation. Hence by performing series of such elementary angle rotations we can obtain a rotation by arbitrary angles. Now there is still $\cos\varphi$term left to deal with. If in each rotation, i,we take a decision as which direction to rotate instead of whether to rotate or not the $\cos\varphi$ term becomes constant because$\cos\varphi = \cos(-\varphi)$. Hence the above equations become

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

$$y_{i+1} = K_i\left[y_i + x_i d_i 2^{-i}\right] \tag{1.4}$$

Where $K_i = \cos\left(\tan^{-1} 2^{-i}\right) = {1}\Big/{\sqrt{1 + 2^{-2i}}}$ $\quad and \quad d_i = \pm 1$

Removing the scale constant from the above iterative equations results in shift-add algorithm for vector rotation. Then each elementary rotation can be called Pseudo Rotation as shown in Figure 1.2. The product of all K_i's can be applied elsewhere in the system or can be treated as processing gain of an algorithm. For infinite iterations the product approaches a constant value of 0.6073 and hence the algorithm has processing gain of$1/0.6073 = 1.647$. However this gain value depends on number on iterations, and is given as

$$A_n = \prod_n \sqrt{1 + 2^{-2i}} \tag{1.5}$$

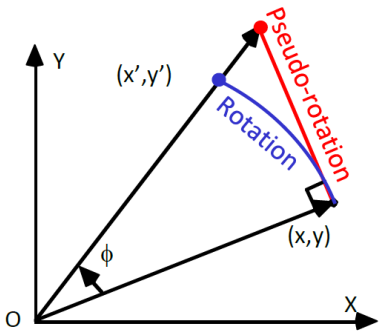

Figure 1.2 Pseudo rotation

The composite angle of all iterations depends on the sequence of directions of elementary rotations. We define a vector such that it accumulates the elementary rotation angles at each iteration. This adds a third equation to the CORDIC algorithm

$$Z_{i+1} = Z_i - d_i \tan\left(2^{-i}\right) \tag{1.6}$$

Here the elementary rotation angles can be represented in any angular unit.

Modes of CORDIC Rotation: There are two modes defined for CORDIC rotator. First mode is a rotation mode which rotates an input vector by a given angle. The vector mode rotates the input vector to the X-axis and returns the angle required to make that rotation.

1) Rotation Mode: In rotation mode the angle accumulator is initialized with the desired rotation angle. Then decision at each iteration is taken such that magnitude of a residual angle reduces after each iteration. The decision depends on the sign of a residual angle. The CORDIC equations in rotation mode are

$$\begin{aligned} X_{i+1} &= X_i - Y_i d_i 2^{-i} \\ Y_{i+1} &= Y_i + X_i d_i 2^{-i} \\ Z_{i+1} &= Z_i - d_i \tan\left(2^{-i}\right) \end{aligned} \tag{1.7}$$

Where $d_i = \begin{cases} -1 & if\ Z_i < 0 \\ +1 & otherwise \end{cases}$

After *n* number of iterations results are obtained

$$\begin{aligned} X_n &= A_n[X_0 \cos Z_0 - Y_0 \sin Z_0] \\ Y_n &= A_n[Y_0 \cos Z_0 + X_0 \sin Z_0] \\ Z_n &= 0 \\ A_n &= \prod_n \sqrt{1 + 2^{-2i}} \end{aligned} \tag{1.8}$$

2) Vector Mode: In vector mode the input vector is rotated through an angle such that it gets aligned with the X-axis. The result of vector rotation is a scaled magnitude of input vector and the angle required to align the input vector with X-axis. Decision at each iteration is taken such that Y component of a vector reduces in magnitude after each iteration. Hence the sign of a residual Y component is used to decide the direction of rotation. If the angle

accumulator is initialized with zero then it contains value of rotated angle, at the end of all iterations. In vectoring mode CORDIC equations are

$$\begin{aligned} X_{i+1} &= X_i - Y_i d_i 2^{-i} \\ Y_{i+1} &= Y_i + X_i d_i 2^{-i} \\ Z_{i+1} &= Z_i - d_i \tan\left(2^{-i}\right) \end{aligned} \tag{1.9}$$

Where $d_i = \begin{cases} +1 \; if \; Y_i < 0 \\ -1 \; otherwise \end{cases}$

The results after n iterations are

$$\begin{aligned} X_n &= A_n \sqrt{X_0^2 + Y_0^2} \\ Y_n &= 0 \\ Z_n &= Z_0 + \tan^{-1}\left({Y_0}/{X_0}\right) \\ A_n &= \prod_n \sqrt{1 + 2^{-2i}} \end{aligned} \tag{1.10}$$

The range of convergence of CORDIC algorithm for both rotation and vector mode is $\left({-\pi}/{2} \; to \; {+\pi}/{2}\right)$. This is calculated by summation of all elementary angles of each rotation. For the angles outside this range trigonometric identities are used to convert the range.

The CORDIC rotator described above can compute several trigonometric functions directly and others indirectly. By selecting initial values and mode it is possible to directly calculate sine, cosine, arctangent, vector magnitude and transformation between polar and Cartesian coordinates.

1.3.2 Scaling-free CORDIC

If the Taylor series approximation to sine and cosine function is used then processing gain of CORDIC can be eliminated completely. Assuming elementary angle of rotation $(\alpha_i)$is sufficiently small such that $\sin(\alpha_i) \cong 2^{-i}$ and $\cos(\alpha_i) = 1 - {\alpha_i^2}/{2!} = 1 - 2^{-(2i+1)}$. But the 2nd order approximation used imposes a restriction on the maximum elementary angle of rotation that can be used in CORDIC iteration. In second order approximation the largest term among the ignored terms is

$$\frac{\alpha_i^3}{3!} = \frac{\left(2^{-i}\right)^3}{3!} = 2^{-(3i+\log_2 6)} \tag{1.11}$$

The multiplication of $\alpha_i^3 / 3!$ with any quantity has no effect if $3i + \log_2 6 \geq b$.Where b is word length. This constraint gives the minimum value of 'i' that can be used in scale-free CORDIC rotator, and is

$$i \geq \frac{b - \log_2 6}{3} \text{ Where } i \text{ takes only integer values}$$

In practice lower bound of *i* can be relaxed slightly and is given as:

$$\left\lfloor \frac{b - 2.585}{3} \right\rfloor \leq i \leq b - 1 \tag{1.12}$$

In scaling-free algorithm the target angle is achieved pure summation of each elementary angle. This means vector is rotated in only one direction, hence d_i=+1 for all iterations. Assuming clockwise vector rotation, CORDIC equations in matrix form can be written as:

$$\begin{bmatrix} X_{i+1} \\ Y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 - 2^{-(2i+1)} & 2^{-i} \\ 2^{-i} & 1 - 2^{-(2i+1)} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \tag{1.13}$$
$$Z_{i+1} = Z_i - 2^{-i}$$

The drawback of this method is that range of convergence is very small. This is because the restriction imposed by tailor series approximation on maximum elementary angle value. However this algorithm provides completely scaling free output for any number of iterations.

1.3.3 Modified virtual scaling-free adaptive CORDIC rotator algorithm and architecture

The problem of range of convergence in scaling free CORDIC can solved in two steps: 1) Use of argument reduction technique to reduce the total angular range to be computed. 2) Carry out elementary rotations such the rate of convergence is enhanced and angle approximation error is reduced below a predefined limit. The main objective of an argument reduction technique is to map the results of a vector rotation by a large target angle θ to the results of vector rotation by a small target angle ∅. This can be achieved by dividing four quadrants of coordinate system into 16 equal domains each spans over an angle of π/8. Any input target angle lies in one of these 16 domains. To examine CORDIC rotator we first consider input target angle θ which lies in one of the domains of first quadrant. Domains in first quadrant are defined as shown in Figure 1.3 i.e. A ([0, π/8)), B ([π/8, π/4)), C ([π/4,3π/8)) and D ([3π/8,π/2)). In each domain angle θ is redefined in terms of an angle ∅ which bounded in interval [0, π/8], and is given as in Equation 1.14. For input vector [X Y] ^{T} CORDIC rotator is defined as in Equation 1.15 and Equation 1.16.

Using Equation 1.15 and Equation 1.16 it is possible to rotate a vector through a target angle lying in any domain in first quadrant with the help of CORDIC rotation through an angle ∅[0, π/8]. This is called Domain Folding since domain B,C and D are essentially folded back to domain A. For the target angle lying in domain B and C a multiplication by

$1/\sqrt{2}$ is required while for target angles in domain D multiplication is not required, hence the name Virtually Scaling Free. For the input target angles lying in other quadrants the domain folding technique can be extended by exploiting the symmetry of coordinate axes. This is given in Table 1.1

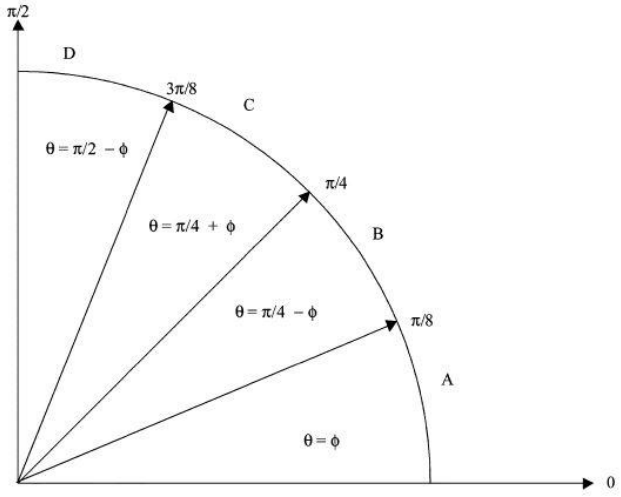

Figure 1.3 Different domains and definition of target angle ∅

$$\begin{aligned}
&\theta = \phi, \ \ in\, domain\, A \\
&\theta = \frac{\pi}{4} - \phi, \ \ in\, domain\, B \\
&\theta = \frac{\pi}{4} + \phi, \ \ in\, domain\, C \\
&\theta = \frac{\pi}{2} - \phi, \ \ in\, domain\, D
\end{aligned} \tag{1.14}$$

$$\begin{aligned}
&\begin{bmatrix} X_A \\ Y_A \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} && ,in\, domain\, A \\
&\begin{bmatrix} X_B \\ Y_B \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} (\cos\phi + \sin\phi) & (\cos\phi - \sin\phi) \\ -(\cos\phi - \sin\phi) & (\cos\phi + \sin\phi) \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} && ,in\, domain\, B \\
&\begin{bmatrix} X_C \\ Y_C \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} (\cos\phi - \sin\phi) & (\cos\phi + \sin\phi) \\ -(\cos\phi + \sin\phi) & (\cos\phi - \sin\phi) \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} && ,in\, domain\, C \\
&\begin{bmatrix} X_D \\ Y_D \end{bmatrix} = \begin{bmatrix} \sin\phi & \cos\phi \\ -\cos\phi & \sin\phi \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} && ,in\, domain\, D
\end{aligned} \tag{1.15}$$

$$
\begin{aligned}
&X_A = X_- \ , Y_A = Y_- \ , \phi = \theta \\
&X_B = \frac{1}{\sqrt{2}}\left[X_+ + Y_+\right], Y_B = \frac{1}{\sqrt{2}}\left[-X_+ + Y_+\right] \ , \phi = \frac{\pi}{4} - \theta \\
&X_C = \frac{1}{\sqrt{2}}\left[X_- + Y_-\right], Y_C = \frac{1}{\sqrt{2}}\left[-X_- + Y_-\right] \ , \phi = \theta - \frac{\pi}{4} \\
&X_D = Y_+ \ , Y_D = -X_+ \ , \phi = \frac{\pi}{2}\theta
\end{aligned}
\tag{1.16}
$$

Table 1.1 Result of CORDIC rotations for different quadrants using domain folding technique

Range of target angle	**X1**	**Y1**
[0, **π**/8)	X_A	Y_A
[**π**/8, **π**/4)	X_B	Y_B
[**π**/4, 3**π**/8)	X_C	Y_C
[3**π**/8, **π**/2)	X_D	Y_D
[**π**/2, 5**π**/8)	Y_A	$-X_A$
[5**π**/8, 3**π**/4)	Y_B	$-X_B$
[3**π**/4, 7**π**/8)	Y_C	$-X_C$
[7**π**/8, **π**)	Y_D	$-X_D$
[**π**, 9**π**/8)	$-Y_A$	X_A
[9**π**/8, 5**π**/4)	$-X_B$	$-Y_B$
[5**π**/4, 11**π**/8)	$-X_C$	$-Y_C$
[11**π**/8, 3**π**/2)	Y_D	$-X_D$
[3**π**/2, 13**π**/8)	X_A	Y_A
[13**π**/8, 7**π**/4)	$-Y_B$	X_B
[7**π**/4, 15**π**/8)	$-Y_C$	X_C
[15**π**/8, 2**π**)	X_D	$-Y_D$

With the help of domain folding technique it is sufficient to consider angle of rotation in range [0, π/8] for CORDIC rotation. But this range lies beyond the range of convergence of scaling free CORDIC rotator. The range of convergence of CORDIC rotator is increased by adaptively selecting the iteration index *i* for each iteration and this depends on the residual angle still to be computed. Hence by repeating some initial iterations the ROC of [0, π/8] can be achieved. Repeating of some elementary rotation steps does not add any scaling to rotator output.

1.3.4 Enhanced Scaling-Free CORDIC

This algorithm provides some modifications in scaling free CORDIC kernel to enhance the performance. These modifications are 1) Use of radix-4 Booth recording algorithm to reduce the number of iterations which leads to reduced pipelined stages. 2) Elimination of domain folding technique.Radix-4 booth recoding is used for Z data path (angle). The use of radix-4 booth recoding ensures that for every two consecutive bits of recoded angle only one bit can be set to 1 as much. This reduces the maximum possible

number of iterations to N/2 for N-bit input angle. The objective of domain folding elimination is to reduce the complexity of preprocessing unit. This is achieved by increasing the RoC to [-π/2, π/2] by using first four conventional CORDIC iterations. However introduction of conventional CORDIC iterations leads to use of Z datapath and a constant scale factor. The use of Z datapath can be avoided using scaling free kernel.

1.3.5 CORDIC II Algorithm

CORDIC II algorithm uses completely different set of elementary angles than that of conventional CORDIC. This set consist of three types of angles 1) Friend Angles 2) Uniformly-Scaled Redundant CORDIC 3) Nano-Rotations.

1) Friend Angles:

This is the set of all angles having same magnitude. Let the set of angles be α_i for which there exists set of coefficients $P_i = C_i + jS_i$such that $\alpha_i = \tan(S_i/C_i)$. Then all angles α_i are Friend angles if all have same magnitude i.e. $\forall\, i,j, |P_i| = |P_j|$. Figure 2.4 show the example of friend angles. The angles $\alpha_1 = 8.13^\circ\ and\ \alpha_2 = 45^\circ$ $\alpha_1 = 8.13^\circ\ and\ \alpha_2 = 45^\circ$have coefficient $P_1 = 7 + j\ and\ P_2 = 5 + j5$respectively. Their magnitude is$|P_1| = |P_2| = \sqrt{50}$. In general any angle α is friend to itself and also to $-\alpha + n\pi/2\ and\ \alpha + n\pi/2$for any value of *n*.

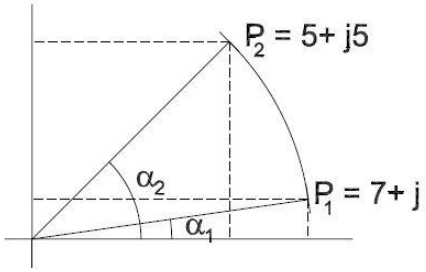

Figure 1.4 Example of friend angles

2) Uniformly-Scaled Redundant (USR) CORDIC

USR CORDIC uses the same rotation angles as the redundant CORDIC, but all the angles have similar scaling. The coefficients for the USR CORDIC are:

$$P_0 = 2^{2k-1} + 1$$
$$P_1 = 2^{2k-1} + j2^k \tag{1.17}$$

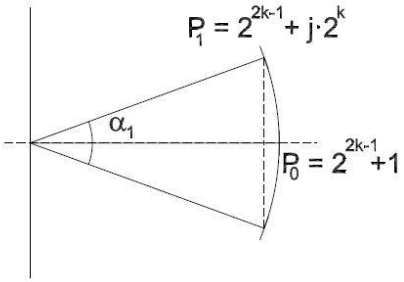

Figure 1.5 USR CORDIC: graphical representation of coefficients

Figure 1.5shows the graphical representation. It is clear that magnitude of both vectors P_0 and P_1 is almost same. Both magnitudes can be related as $|P_0|^2 = |P_1|^2 + 1$. Angles of the USR CORDIC are: $\alpha_0 = 0 \, and \, \alpha_1 = \tan^{-1}\left(2^k / 2^{2k-1}\right) = \tan^{-1}\left(2^{-k+1}\right)$

3) Nano-Rotations

This is a set of angles with the coefficients $P_k = C + jk$ such that k=0,1,…,N and C is constant. This corresponds to a set of angles $\alpha_k = \tan^{-1}\left(k/C\right)$. Here N is assumed to be much smaller than C. This leads to angle α_k to be very small and hence $\alpha_k = \tan^{-1}\left(\alpha_k\right)$ is satisfied. Since N≪C the magnitude of all the coefficients is almost same. Figure 2.6shows the kernel to calculate nano-rotations.

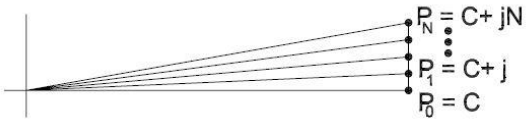

Figure 1.6 Kernel to calculate nano-rotations

The CORDIC II algorithm consists of several stages connected in series. Where each rotation stage has input range [-α_{in}, α_{in}] and output range [-α_{out}, α_{out}].In general rotation stage can have any number of rotation angles N. Each input is rotated through one of these N angles. The output angle of a rotation stage is given by:

$$\begin{aligned} \alpha_{out} &= \max_i\left(\delta_i\right) \;\; i = 1,.....,N/2 \;\; if \; N \, is \, even \\ \alpha_{out} &= \max_i\left(\delta_i\right) \;\; i = 1,.....,\left(N-1\right)/2 \, if \; N \, is \, odd \end{aligned} \tag{1.18}$$

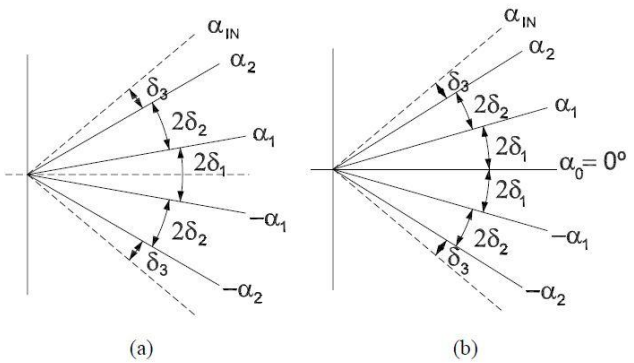

Figure 1.7 Rotation of N-rotator (a) N=4(even) (b) N=5(odd)

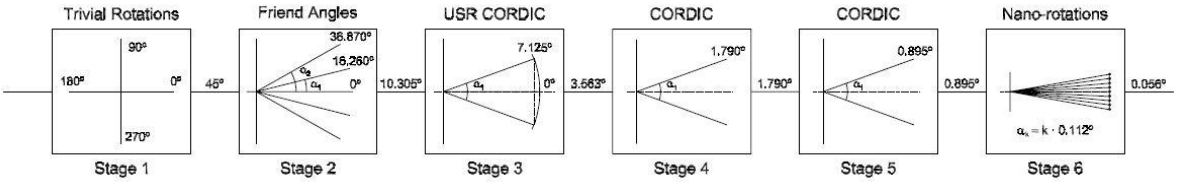

Figure 1.8 CORDIC II Architecture

Figure 1.7 shows both the cases when N is even and N is odd. Figure 2.8 shows the architecture of CORDIC II algorithm. This consists of six iteration stages connected in series. Input angle to the rotator can be in the range [0, 360]. At first stage a trivial rotations by ±180° and ±90° are performed so that the residual angle at the output of the first stage is ±45°. In 2nd stage rotation through any one of the angles from the set of friend angles is done. This stage has a kernel [25, 24+j7, 20+j15], which corresponds to angles 0°, 16.26° and 36.87° respectively. Scale factor for all the coefficients is 25.The maximum residual angle at the output of second stage is ±10.305°. The third stage uses USR CORDIC and uses a kernel [129, 128+j16]. The maximum residual angle after 3rd stage is ±3.563°. The 4th and 5th stage uses conventional CORDIC rotation by 1.79° and 0.895° respectively. The 6th stage uses nano-rotations. The kernel for this stage is $P_k = 512 + jk, k = 0,.....,8$ which leads to rotation angles of $\alpha_k = k * 0.112^\circ$. The remaining angle at the end of 6th stage is $\pm 0.056°$.

CHAPTER 2

A NEW SCALING FREE CORDIC ARCHITECTURE

2.1 Introduction

In this chapter, a new CORDIC architecture is proposed which is completely scaling free. Limitations of existing virtually scaling free algorithms [12] and CORDIC II algorithm [4] are discussed. The proposed architecture is coded in VerilogHDL, synthesised in Xilinx ISE14.7 and mapped onto Xilinx SPARTAN 3E XC3S500E-FG320-5 FPGA device. The hardware implementation results of the proposed CORDIC architecture are discussed in later part on this chapter.

Existing virtually scaling free algorithm [12] reduces the average number of iterations to 50% compared to conventional CORDIC iterations. Additionally, the output of this algorithm is virtually scaling free meaning that for some input angles (angles lying in particular domain) the result requires to be scaled down. Hence, algorithm requires a scaling unit which further increases the hardware cost during implementation. However, the average error in computation of sine and cosine functions is more than conventional CORDIC. Further, pipelined implementation of this algorithm [12] has large number of (14) pipelined stages which results in output latency of 14 clock cycles and large hardware cost.

The CORDIC II algorithm [4] overcomes the limitations of scaling free algorithm to some extend but it has its own limitations. The CORDIC II algorithm in [4] has low hardware cost as compared to virtually scaling free algorithm[12] when implemented onto FPGA board. Additionally, CORDIC II algorithm [4] has latency of only 6 clock cycles. However, CORDIC II [4] has very poor accuracy in sine and cosine function computation.

2.2 Proposed CORDIC Architecture

In literature, many algorithms [12-15] use the Taylor series approximation of sine and cosine functions for making CORDIC implementation completely or virtually scaling free. The use of Taylor series approximation puts limit on the maximum elementary angle (basic shift) that can be used in CORDIC algorithm. The CORDIC algorithm in [12] [13] uses 2^{nd} order Taylor series approximation while algorithms in [14] [15] uses 3^{rd} order Taylor series approximation of sine and cosine functions. But, for 2^{nd} and 3^{rd} order Taylor series approximation, the maximum allowable elementary angle is 3.581° and 14.32° respectively. Hence, with such small angles, the convergence to target angle lying in range of 0° to ± 360°, requires large number of iterations in case of iterative CORDIC algorithm[12] [14], and large number of rotation stages in case of pipelined CORDIC algorithm[12] [15]. This problem can be addressed by introducing few large angles in the set of elementary angles. To implement such approach, we approximate sine and cosine values of corresponding angle with only negative powers of two as shown in Table 2.1 and this approximation is used in basic

equations of CORDIC algorithm. This results in rotation of input vector through the corresponding angle without any scale factor. The advantage of this approach is that it is possible to rotate input vector through a particular angle using only shift and add operation hence the simplicity of conventional CORDIC algorithm is still maintained. Moreover, there is no restriction on the angle that can be used for vector rotation.

The proposed CORDIC architecture is suited for pipelined implementation. The architecture of the proposed CORDIC is shown in Figure2.1. The architecture has seven pipelined stages. Each pipelined stage in the architecture has three inputs (X_i, Y_i, and Zi) and three outputs (X_{i+1}, Y_{i+1} and Z_{i+1}). The X and Y datapaths corresponds to x and y coordinates of input vector to the pipelined stages respectively while the Z datapath represents the remaining angle of vector rotation. Each rotation stage has a fixed set of elementary angles. From the set of elementary angle, one angle is chosen based on the input angle to that rotation stage and input vector is rotated through the chosen angle. Table 2.1 summarises the angle set and sine and cosine function approximation used for each pipelined stage. As shown in Table 2.1 since each rotation stage uses sine and cosine approximation for only one angle the shifters for each stage become simple wire connections reducing the hardware cost of proposed architecture. Further the proposed architecture does not have any scaling factor at the output since each stage in the architecture is completely scaling free. Hence proposed architecture is completely scaling free.

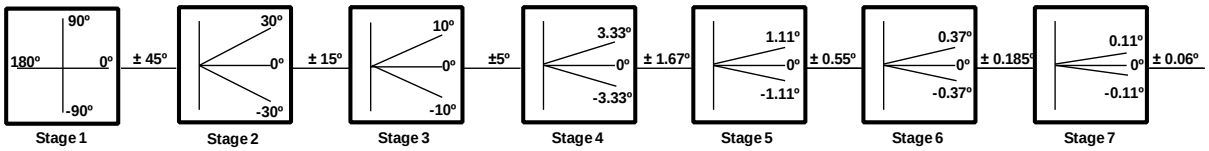

Figure 2.1 Architecture of the proposed CORDIC

Table 2.1 Summary of rotation stages of proposed architecture.

Stage	Angle Set	SINE and COSINE function approximation used	% Error in approximation	Maximum Remaining Angle
1	0°, ±90°, ±180°	-	-	±45°
2	0°, ±30°	$\sin(30^{\circ}) \cong 2^{-1}$ $\cos(30^{\circ}) \cong 2^{0} - 2^{-3} - 2^{-7} - 2^{-10}$	0 0.021	±15°
3	0°, ±10°	$\sin(10^{\circ}) \cong 2^{-3} + 2^{-4} - 2^{-6} + 2^{-9}$ $\cos(10^{\circ}) \cong 2^{0} - 2^{-6} + 2^{-11}$	0.104 5.64×10^{-3}	±5°
4	0°, ±3.33°	$\sin(3.33^{\circ}) \cong 2^{-4} - 2^{-8} - 2^{-11}$ $\cos(3.33^{\circ}) \cong 2^{0} - 2^{-9} + 2^{-12}$	0.032 0.022	±1.67°
5	0°, ±1.11°	$\sin(1.11^{\circ}) \cong 2^{-6} + 2^{-8} - 2^{-13}$ $\cos(1.11^{\circ}) \cong 2^{0} - 2^{-13} - 2^{-14}$	0.092 4.93×10^{-4}	±0.55°
6	0°, ±0.37°	$\sin(0.37^{\circ}) \cong 2^{-7} - 2^{-10} - 2^{-11}$ $\cos(0.37^{\circ}) \cong 2^{0}$	1.802 2.01×10^{-3}	±0.185°
7	0°, ±0.11°	-	-	±0.06°

The detailed description of each rotation stage is as follow:

Stage 1: Stage 1 is the first pipelined stage of proposed CORDIC architecture. The stage 1 has the elementary angle set as {-180°,-90°, 0°, +90°, +180°}. This stage rotates the input vector though any one angle from this set. The advantage of such rotation is that it does not require any shifting operation; instead only negation and addition/subtraction operations are required. Hence, approximation of sine and cosine functions is not required in stage 1 of CORDIC rotation. The rotation angle is selected based on the value of input angle to this rotation stage such that output angle of stage 1 is minimized. The input angle to stage 1 is in the range of 0° to ±360°. The maximum output angle of this stage is ±45°.

Stage 2: As output of the stage 1 is fed to stage 2, the input angle range for the stage 2 is ±45°. The elementary angle set for this rotation stage is {-30°, 0°, +30°}. Hence when the input angle to this stage is greater than +15° or less than -15° then input vector is rotated through +30° or-30° respectively, otherwise input vector is not rotated and passed to the next stage as it is. This results in the maximum output angle of ±15° at the output of stage 2. The sine and cosine function for angle 30° are approximated as shown in Table 2.1. In sine function approximation for 30° only one term with negative power of two (2^{-i}) is used while cosine function is approximated with the help of four terms with negative power of two (2^{-i}). As shown in Table 2.1 the percentage error in approximation of sine and cosine function for 30° angle is 0 and 0.021 respectively. Putting these approximated sine and cosine functions in equation 1.1 results in vector rotation by ±30°.The hardware for the vector rotation through ±30° for X and Y datapath is shown in Figure 2.2 (a) and Figure 2.2 (b) respectively. As stated earlier, as shifting value for each shifter is fixed they become simple wire connection. Hence, no shifters are required in this vector rotation. The advantage of using such method for rotation results in completely scaling free output, hence scaling factor compensation circuitry is not required in the proposed architecture.

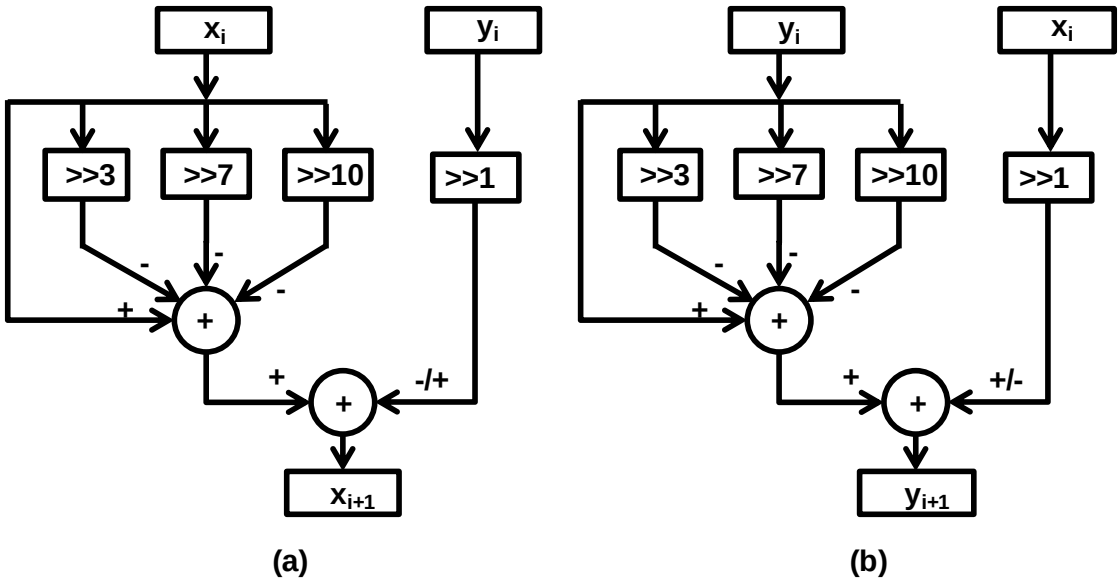

Figure 2.2 Hardware for vector rotation through ±30° (a) for X datapath (b) for Y datapath

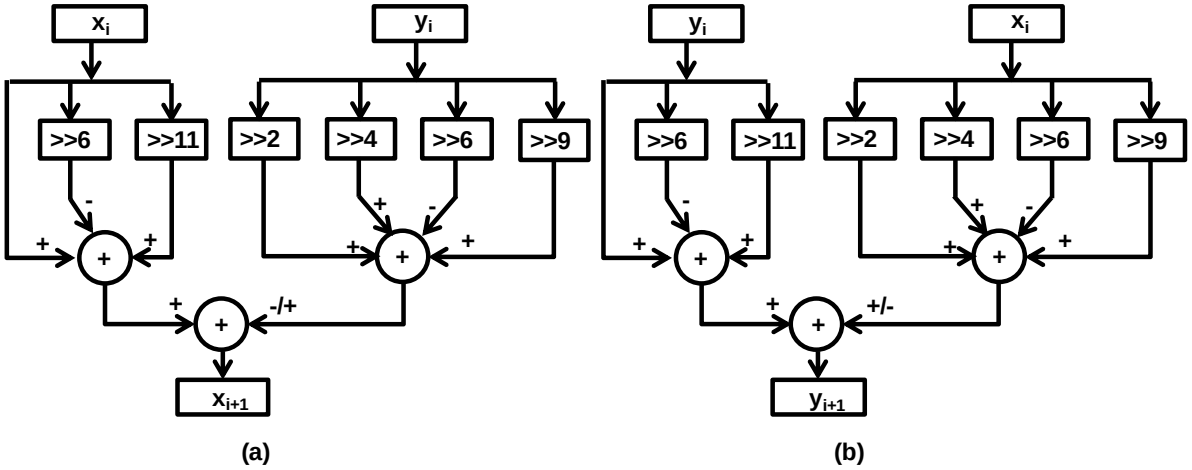

Figure 2.3 Hardware for vector rotation through ±10° (a) for X datapath (b) for Y datapath

Stage 3: The elementary angle set for stage 3 is {+10°, 0°,-10°}. The sine and cosine function values for an angle 10° are approximated as shown in Table 2.1. The percentage error in this approximation of sine and cosine functions is 0.104 and 5.64 $\times 10^{-3}$ respectively. Even though percentage error in sine function approximation is more as compared to that of cosine function it does not introduce significant error in the output of CORDIC algorithm. This is because for smaller angle values (10°) sine function has small value hence the magnitude of the error is also small. The hardware required for vector rotation through ±10° for X and Y datapath is as shown in Figure 2.3 (a) and Figure 2.3 (b) respectively. Depending on the value of the input angle any one from the elementary angle set is chosen and input vector is rotated through this angle. The rotation angle is chosen such that output angle of stage 3 is minimized. The maximum output angle of this stage is ±5° which is given as input angle to stage 4.

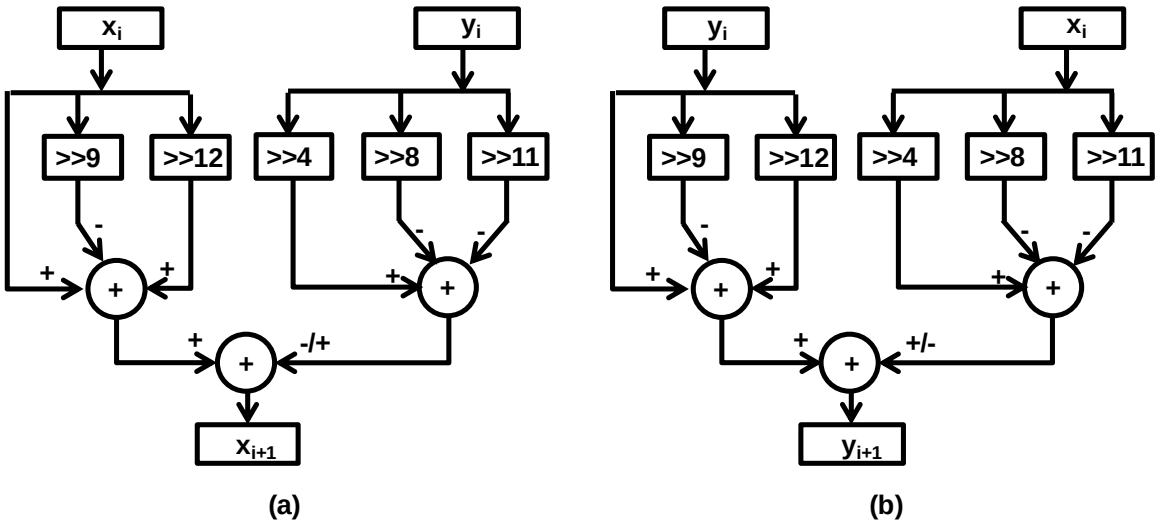

Figure 2.4 Hardware for vector rotation through ±3.33° (a) for X datapath (b) for Y datapath

Stage 4: Stage 4 has elementary angle set as {+3.33°, 0°,-3.33°}. Sine and cosine functions for angle 3.33° are approximated as shown in Table 2.1. The percentage error in

approximating sine and cosine functions for angle 3.33° is only 0.032% and 0.022% respectively. This is very small error. The input angle to stage 4 lies in the range 0 to ±5°. The rotation angle is selected depending value of the input angle to stage 4. Upon selection of rotation angle the input vector is rotated through selected angle. The maximum output angle of this stage is ±1.67° which is given as input angle to stage 5. The hardware for vector rotation through ±3.33° for X and Y datapath is as shown in Figure 2.4 (a) and Figure 2.4 (b) respectively.

Stage 5: For stage 5, ±1.11 ° angles are used along with 0° as elementary angle of rotation. When magnitude of input angle to this stage is less than or equal to +0.55° the input vector to this stage is rotated through 0°, in other words input vector is passed to next stage without any rotation. Otherwise input vector is rotated through ±1.11 ° depending on the sign of input angle to this stage. The sine and cosine functions for angle 1.11° are approximated as shown in Table 2.1.The percentage errors in approximating sine and cosine functions for angle 3.33°are only 0.092 % and 4.93×10^{-4} % respectively. The sine function approximation has three 'power of two' terms on RHS of approximation equation in Table 2.1 while in case of cosine function approximation only two 'power of two' terms are used on RHS. The hardware for vector rotation through ±1.11° for X and Y datapath is as shown in Figure 2.5 (a) and Figure 2.5 (b) respectively. The maximum output angle of this stage is ±0.55°.

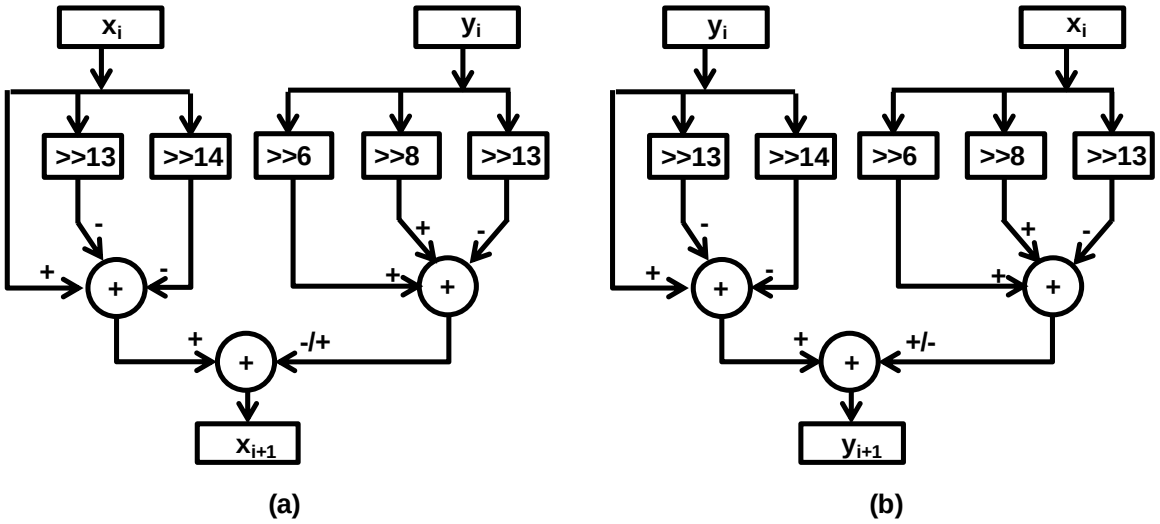

Figure 2.5 Hardware for vector rotation through ±1.33° (a) for X datapath (b) for Y datapath

Stage 6: Stage 6 has elementary angle set as {+0.37°, 0°,-0.37°}. The sine and cosine functions for angle 0.37° are approximated using 'power of two' terms as shown in Table 2.1. The percentage error in approximating sine and cosine functions for angle 3.33° is 1.802 % and 2.01×10^{-3} % respectively. As shown in Table 2.1, the sine function is approximated by using three 'power of two' terms while the cosine function is approximated by using only one 'power of two'. The maximum output angle of this stage is ±0.185° which is given as input angle to stage 7.The hardware for vector rotation through ±1.11° for X and Y datapath is as shown in Figure 2.6 (a) and Figure 2.6 (b) respectively.

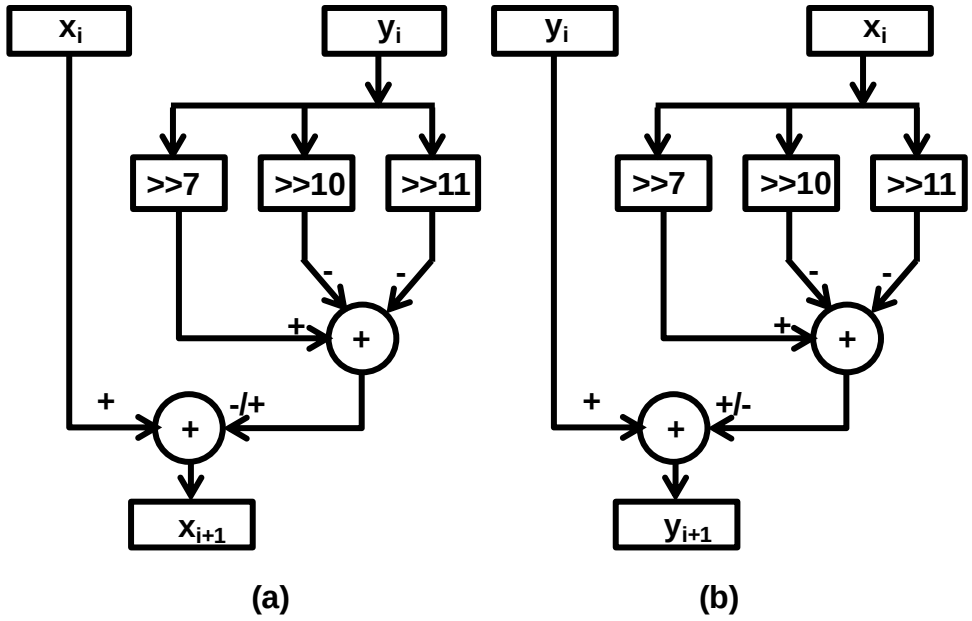

Figure 2.6 Hardware for vector rotation through ±0.37° (a) for X datapath (b) for Y datapath

Stage 7: Stage 7 employs an elementary angle from the set of conventional CORDIC along with the angle 0°. The elementary angle set is {+0.112°, 0°,-0.112°}. For implementation of this rotation stage conventional CORDIC equation with the shift value of 9 is used as shown in Equation 2.1. As the shift value is fixed, the shifting operation can be implemented using only hardwired connections and hence no actual shifter hardware is required. The input vector to this rotation stage is rotated through any one angle from the elementary angle set and the selection of this angle depends on the value of the input angle to this stage. The maximum output angle of this stage is ±0.06°. As the large shift index value (9) is used for vector rotation in stage 7, the output of stage 7 has scaling factor of approximately one. Since the maximum output angle of stage7 is ±0.06° the maximum residual angle of this algorithm is ±0.06°.

$$\mathrm{X_{i+1}} = X_i \mp Y_i(2^{-9})$$

$$\mathrm{Y_{i+1}} = Y_i \pm X_i(2^{-9}) \qquad (2.1)$$

2.3 Hardware Implementation Results

The proposed architecture along with modified virtually scaling free CORDIC algorithm [12], CORDIC II algorithm [4] are coded in VerilogHDL, synthesised in Xilinx ISE 14.7 and mapped onto XC3S500E-FG320-5 FPGA device which is from SPARTAN 3E device family. The XC3S500E device has a total of 4656 slices, 9312 four-input LUTs and 9312 slice flip-flops (FFs). The proposed architecture is compared with other CORDIC architectures. This comparison is done by considering hardware requirement of architectures, maximum frequency of operation, latency, slice-delay product and accuracy. Table 2.2 shows the comparison of CORDIC algorithms in terms of hardware cost, latency, maximum

frequency of operation and slice-delay product, whereas in Table 2.3 error performance of CORDIC algorithms is compared. The hardware requirement of CORDIC architectures when mapped onto XC3S500E device is calculated in terms of number of slices, slice FFs and four-input LUTs occupied by architecture. The proposed architecture occupies total 900 slices (19% of total number of slices available), 1612 four-input LUTs (17% of total number of LUTs available) and 544 slice flip-flops (5% of total number of slice flip-flops available). Hence, it is clear from the Table 2.2 that proposed architecture occupies less hardware as compared to modified virtually scaling free CORDIC [12] when implemented on FPGA. Further, slice-delay product of proposed architecture is lesser than modified virtually scaling free CORDIC [12] but more than that of CORDIC II algorithm [4]. However the error performance of proposed architecture is much better than the CORDIC II algorithm [4] as shown in Table 2.3. The proposed architecture has latency of 7 clock cycles as compared to that of 6 and 16 in CORDIC II [4] and modified virtually scaling free CORDIC [12] respectively.

Table 2.2: Comparison of hardware cost of CORDIC architectures

Algorithm	**Slice (A)**	**LUTs**	**Slice FFs**	**Max. Freq MHz (B)**	**Latency (C)**	**Slice-Delay Product (A*C/B)**
CORDIC II [4]	597	1078	468	85.85	7	48.58
MVSFA CORDIC [12]	1174	2144	722	68.85	14	238.71
Proposed CORDIC	900	1612	544	63.59	7	99.07

The error performance is calculated in terms of Bit Error Position (BEP) which tells the position of error measured from the most significant bit (MSB). For example, BEP of 12 means that bit error occurs on the 12^{th} bit from the MSB. Hence, for better error performance BEP should be as large as possible. In other words the error should lie as far as possible from the MSB since as we go away from the MSB by one bit position the weight of a bit gets halved.

Table 2.3 Comparison of error performance of CORDIC architectures

Algorithm	**Max. Error (BEP)**		**Avg. Error (BEP)**	
	SINE	**COSINE**	**SINE**	**COSINE**
CORDIC II [4]	6.01	5.98	6.78	6.71
MVSFA CORDIC [12]	10.97	10.97	12.60	12.57
Proposed CORDIC	9.28	9.30	12.27	12.07

The error performance shown in Table 2.3 is computed in terms of maximum and average bit error positions for sine and cosine function computation using different CORDIC architectures. When CORDIC rotator is used in rotation mode and initial values of X and Y datapaths are 1 and 0 respectively, the outputs of the CORDIC rotator are nothing but sine (Y datapath) and cosine (X datapath) values of the input angle (initial value of Z datapath). The error in computation of sine and cosine values for different input angles is calculated by

comparing the results of the CORDIC architecture to the actual values of sine and cosine functions from MATLAB. The error for different input angles using proposed architecture is plotted in Figure 2.7 and Figure 2.8 for sine and cosine functions respectively.

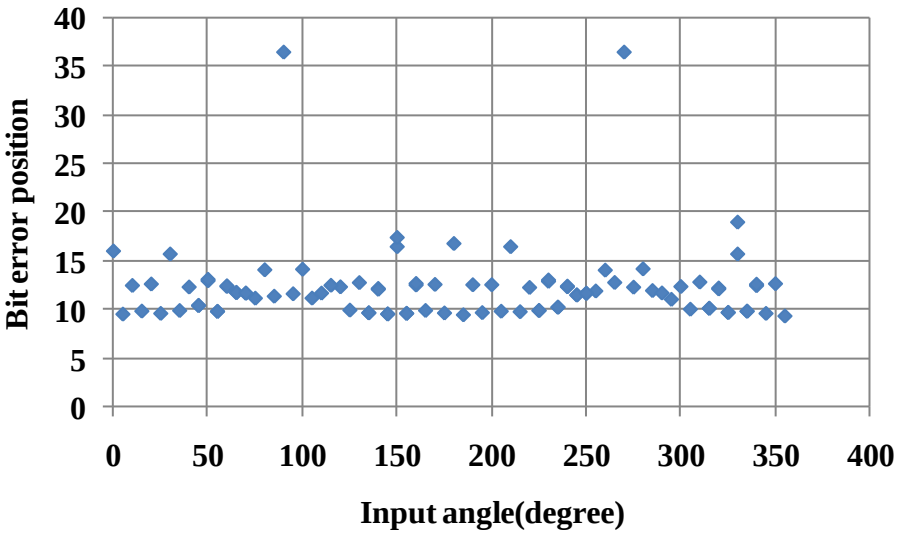

Figure 2.7 BEP in computation of SINE values of input angles

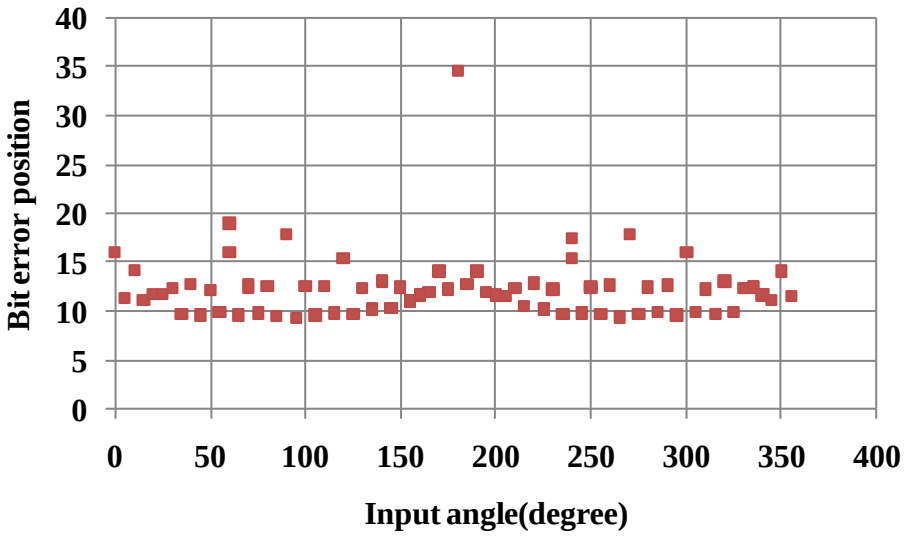

Figure 2.8 BEP in computation of COSINE values of input angles

In case of proposed architecture, for the sine and cosine function computation maximum error occurs at 9.28th and 9.30thbit position from MSB respectively. Also the average error for sine and cosine function computation occurs at 12.27th and 12.07th bit position from MSB respectively. Hence, maximum error performance of modified virtually scaling free CORDIC [12] is better than that of proposed CORDIC by only one bit position. However, the average error of both the architectures for sine and cosine computation is almost same. Further, in case of proposed CORDIC architecture maximum error and average error is improved by three and six bit positions respectively as compared to the error

performance of CORDIC II algorithm [4]. Hence, proposed architecture is most efficient when both hardware requirement and error performance are taken into account.

2.4 Summary

In this chapter, a new CORDIC architecture is proposed and discussed in detail. The proposed architecture uses a unique angle set at each pipelines stage to reduce the latency of architecture. Further, suitable approximation of sine and cosine function using the terms with only negative power of two (2^{-i}) makes the architecture completely scaling-free. The proposed architecture along with other architectures is coded in VerilogHDL, synthesised in Xilinx ISE14.7 and mapped onto XC3S500E-FG320-5 FPGA device which is from SPARTAN 3E device family. The performance of architecture is calculated in terms of hardware requirement, maximum frequency of operation, latency, slice-delay product and bit error position. This algorithm is more accurate as compared to CORDIC II [4] and requires less hardware as compared to modified virtually scaling free CORDIC [12]. Hence, proposed architecture is most efficient when both hardware requirement and error performance are taken into account.

CHAPTER 3

LOW LATENCY SCALING FREE PIPELINED CORDIC ARCHITECTURE

3.1 Introduction

In this chapter, a new CORDIC architecture is proposed which has very low latency in pipelined implementation. In literature, only Hybrid CORDIC algorithm [21] has less latency than the proposed architecture. However, Hybrid CORDIC algorithm [21] requires more calculations at each iteration and produces variable scaling factor. The outputs of the proposed CORDIC architecture do not have any scaling factor. Hence, the proposed architecture is completely scaling free. Because of the scaling free nature of proposed architecture scaling factor calculation and compensation circuitry is not required for its hardware implementation. The proposed architecture is coded in VerilogHDL and synthesised in Xilinx ISE14.7. Further it is mapped onto XC3S500E-FG320-5 Xilinx FPGA device which is from SPARTAN-3E device family. The hardware implementation results of this architecture are presented in tabular form and compared with the implementation results of some of the other pipelined architectures present in the literature. The hardware implementation results provide information about hardware requirement, maximum frequency of operation, latency, slice-delay product and accuracy of the architecture.

3.2 Proposed Architecture

The proposed CORDIC architecture is well suited for pipelined implementation with only 5 pipelined stages whereas CORDIC II [4] and modified virtually scaling free CORDIC [12] has 6 and 14 pipelined stages respectively. The architecture of the proposed CORDIC architecture is shown in Figure 3.1. Since the architecture has five pipelined stages, for each input the corresponding output is available after five clock cycles, hence the architecture has latency of only five clock cycles. Each pipelined stage in the architecture has three inputs (X_i, Y_i, and Zi) and three outputs (X_{i+1}, Y_{i+1} and Z_{i+1}). The X and Y datapaths corresponds to x and y coordinates of input vector to the pipelined stages respectively while the Z datapath represents the remaining angle of vector rotation. There is a fixed set of rotation angles for each pipelined stage in the architecture and depending on the value of the input remaining angle to the pipelined stage only one angle from this set is selected for vector rotation. The angle of rotation in each pipelined stage is selected such that at the output of each pipelined stage the remaining rotation angle has minimum value.

The modified virtually scaling free algorithm in [12] uses 2^{nd} order Taylor series approximation of sine and cosine functions for vector rotation. This results in maximum angle of rotation for a pipelined stage to only 3.58°. Hence, multiple rotations of 3.58° are required depending on the value of input angle. This results in more hardware and increased latency of the implementation. This issue can be addressed by using larger rotation angles for

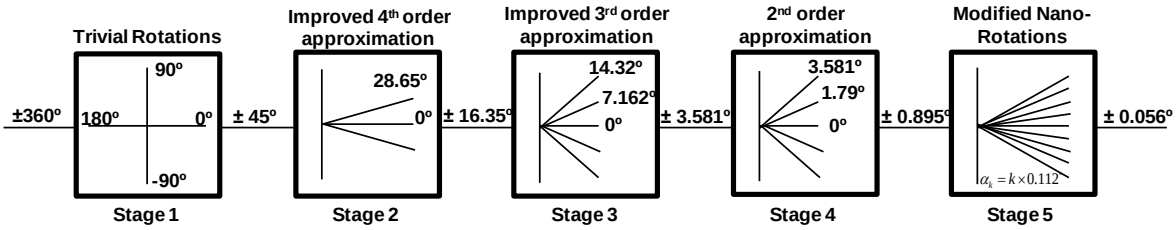

Figure 3.1 Architecture of proposed architecture.

initial pipelined stages. The higher order Taylor series approximation allows using large angles for vector rotation. Hence, the 4th and 3rd order Taylor series approximations of sine and cosine functions are also exploited in the proposed architecture. For the rotation angle of (2^{-i}), the 4th and 3rd order Taylor series approximations are shown in Equation 3.1 and Equation 3.2 respectively for sine and cosine functions. Here 'i' is the shift index.

$$\sin(2^{-i}) \cong 2^{-i} - 2^{-(3i+3)} \quad (3.1\text{ a})$$

$$\cos(2^{-i}) \cong 1 - 2^{-(2i+1)} + 2^{-(4i+5)} + 2^{-(4i+7)} + 2^{-(4i+8)} \quad (3.1\text{ b})$$

$$\sin(2^{-i}) \cong 2^{-i} - 2^{-(3i+3)} \quad (3.2\text{ a})$$

$$\cos(2^{-i}) \cong 1 - 2^{-(2i+1)} \quad (3.2\text{ b})$$

However, as in equation 3.1(a) 4th order Taylor series approximation of sine function involves only two terms. Hence, the accuracy of sine values for larger input angles degrades. As a result we add few more terms to the right hand side (RHS) of the equation 3.1(a) to improve the accuracy of the approximation. Similarly, to improve the accuracy of 3rd order Taylor series approximation of sine function one term on the RHS of the equation 3.2(a) is added. As shown in Figure 3.1 stage 2, stage 3 and stage 4 uses improved 4thorder,improved 3rd order and 2nd order Taylor series approximation of sine and cosine functions.

The details of each pipelined stage are as follows:

Stage 1: The input angle to the proposed architecture can be in the range 0° to ±360°.This input is fed to the stage 1. Stage 1 rotates the input vector though any one angle from its angle set such that the maximum output angle at the output of the stage is ±45°. The angle set for this stage is {-180°,-90°,0°,+90°,+180°}. The rotation angle is selected based on the value of the input angle to this stage. The advantage of this stage is that it does not require any shift operation instead only negation and addition/subtraction are sufficient. Hence, less hardware is requirement for implementation of this stage.

Stage2: The angle set for stage 2 is {-28.65°, 0°, +28.65°}. In Equation 3.1 the shift index i=1 results in the rotation angle of 28.65°. The 4th order Taylor series approximation of sine cosine functions shown in Equation 3.1 results in percentage error of 1.032% and 0.012% for sine and cosine functions respectively. Hence, clearly sine function approximation produces significant error in the approximation. To improve the accuracy in sine function approximation stage 2 uses improved 4th order Taylor series approximation of sine function.

This means two more terms are added on the RHS of the Equation 3.1(a). Hence, the Equation 3.1(a) now gets modified to Equation 3.3 and it has percentage error of 0.014% in the approximation. However, the approximation for cosine function is not modified and is same as shown in Equation 3.1(b). Hence, the use of two extra terms in sine function approximation improves the overall accuracy of algorithm at the cost of small hardware overhead. Since in stage 2, only rotation through 28.65° is performed by exploiting improved 4^{th} order Taylor series approximation the shifters become simple wired connections and hence reducing the hardware cost. The rotation of input vector through 0° does not require any operation to be performed. The input vector and input angle are passed to next stage without any operation.

$$\sin(2^{-1}) \cong 2^{-1} - 2^{-6} - 2^{-8} - 2^{-10} \tag{3.3}$$

To rotate the input vector through 28.65°, the approximated values of sine and cosine function are put into basic CORDIC equation. Hence, this results into a completely scaling free rotation. The hardware requirement of X and Y datapath for implementing rotation through ±28.65° is shown in Figure 3.2 and Figure 3.3 respectively. Depending on the value of input angle to stage 2 rotation angle and its direction is decided. The maximum output angle of stage 2 is ±16.35°.

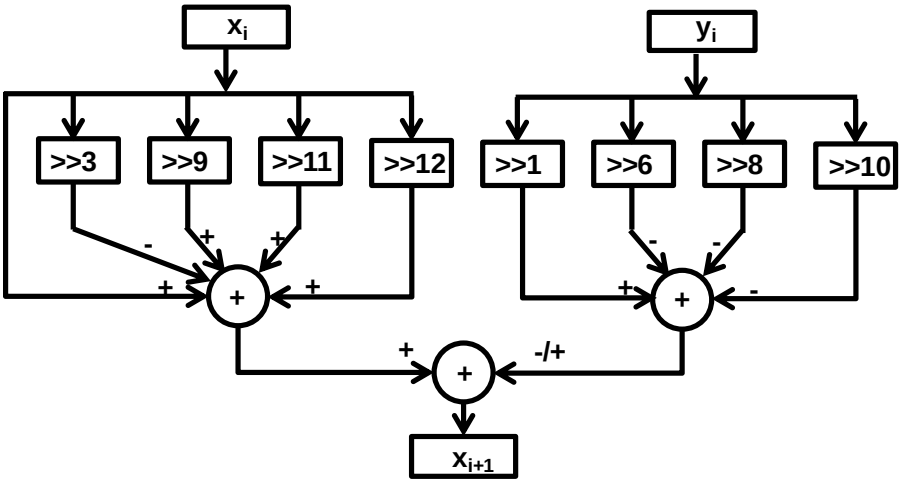

Figure 3.2 Hardware for X datapath for vector rotation through ±28.65°

Stage3: The stage 3 has five angles in its angle set and the angle set is {-14.32°,-7.162°,0°,+7.162°,+14.32°}. For this stage improved 3^{rd} order Taylor series approximation of sine and cosine functions is used. The 3^{rd} order Taylor series approximation of sine and cosine functions shown in Equation 3.2. For the angle 14.32° the 3^{rd} order approximation of sine and cosine functions results in percentage error of 0.26% and 0.017% respectively. Similarly for rotation angle 7.162° the percentage error in sine and cosine approximation is 0.07% and 0.001% respectively. Hence, clearly sine function has higher error as compared to cosine function. To improve the accuracy in approximation of sine function, one extra term is added in RHS of equation 3.2(a) and equation 3.2(a) gets modified to equation 3.3. The error in sine function approximation for angle 14.32° and 7.162° is now reduced to 0.06% and 0.016% respectively. Hence, the approximation used in stage 3 is named as improved 3^{rd}

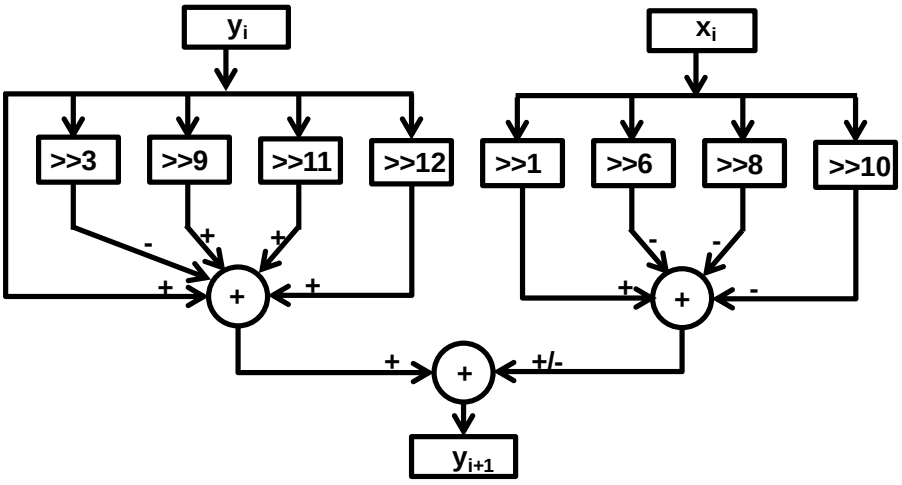

Figure 3.3 Hardware for Y datapath for vector rotation through ±28.65°

order Taylor series approximation. For cosine function approximation only Taylor series terms as shown in equation 3.2(b) are sufficient and provide sufficient accuracy. Hence, equation 3.2(b) is not modified. The addition of one more term in sine approximation adds two extra shifters and two adders in hardware implementation. However, the improvement in accuracy is more significant as compared to hardware overhead. The vector rotation through angles 14.32° and 7.162° is implemented in same hardware instead of separate ones. This hardware implementation for X and Y datapath is shown in Figure 3.4 and Figure 3.5 respectively. Because of barrel shifters used in the hardware it possible to use same hardware for the rotation through both the angles. The shift index (i) equal to 2 results in rotation angle of 14.32° while i=3 corresponds to rotation angle of 7.162°. The angle of rotation is selected based on the input angle to stage 3. The maximum output angle of this stage is ±3.581° and this is fed as input angle to stage 4.

$$\sin\left(2^{-i}\right) \cong 2^{-i} - 2^{-(3i+3)} - 2^{-(3i+5)} \tag{3.3}$$

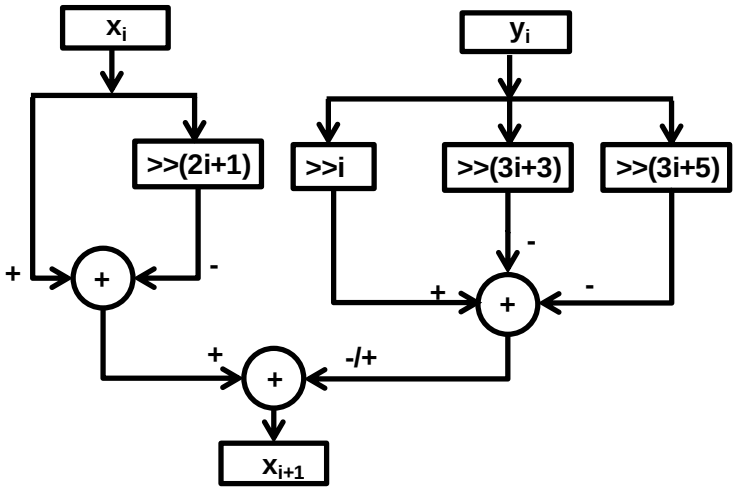

Figure 3.4 Hardware for X datapath for vector rotation through ±14.32° and ±7.162°.

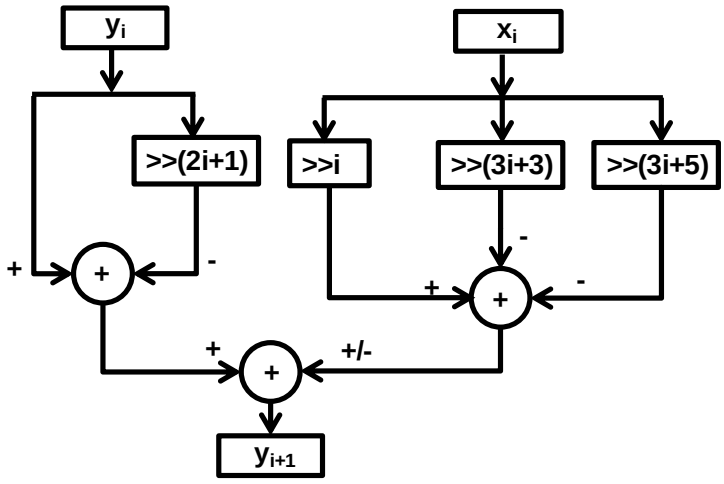

Figure 3.5 Hardware for Y datapath for vector rotation through ±14.32° and ±7.162°.

Stage4: The angle set for stage 4 consist of five angles {-3.581°,-1.79°,0°,+1.79°,+3.581°}. Stage 4 uses 2nd order Taylor series approximation of sine and cosine functions to rotate the vector through one of the angle from the angle set. Equation 3.4 shows the 2nd order Taylor series approximation of sine and cosine functions. For an angle 3.581° the percentage error in sine and cosine function calculation is 0.065% and 0.00006% respectively. While for rotation angle of 1.79° this error is further reduced to 0.016% and 3.98×10^{-8} % for sine and cosine functions respectively. Hence, 2nd order provides acceptable accuracy in sine and cosine approximations. The hardware required for stage 4 is shown in Figure 3.6. The hardware for vector rotation though ±3.581° and ±1.79° is same and requires only four shifters and four adders. The shift index of value 4 corresponds to rotation angle of ±3.581° and shift index with value 5 corresponds to rotation angle of ±1.79°. This stage requires lesser hardware as compared to stage 2 and stage 3 since less number of terms on the RHS of the equation 3.4 are used in sine and cosine function approximation. The angle of rotation is selected based on the value of the input angle to this stage. The maximum output angle of this stage is ±0.895°.

$$\sin\left(2^{-i}\right) \cong 2^{-i} \tag{3.4 a}$$

$$\cos\left(2^{-i}\right) \cong 1 - 2^{-(2i+1)} \tag{3.4 b}$$

Stage5: The angles set of stage 5 consists of linearly spaced angles which are $\alpha_k = k \times 0.112°$ where k=0, 1...., 8. We call these rotation angles as modified nano-rotations. These angles are same as that used in CORDIC II [4]. However, in stage 5 the rotation of vector through these angles differs from that in CORDIC II [4]. Stage 5 is implemented such that the output of this stage is not scaled at all. This is in contrast with the last stage of CORDIC II which has scale factor of approximately 1024. Hence, by eliminating the scaling factor of this stage the truncation error which is present in case of CORDIC II algorithm [4] is avoided in this architecture. The hardware for this stage is shown in Figure 3.7. Depending on the value of the input angle to stage 5 angle of rotation is selected such that remaining angle at the output of stage 5 is minimum. Then the input vector is rotated through rotated through the selected angle using hardware shown in Figure 3.7. Figure 3.7(a) shows the hardware for modified nano-rotator. It requires only two adders, two shifters and two multiply by 'k' units. The

hardware for multiply by ‘k’ unit is shown in Figure 3.7(b). This unit multiplies the input with the selected value of ‘k’.

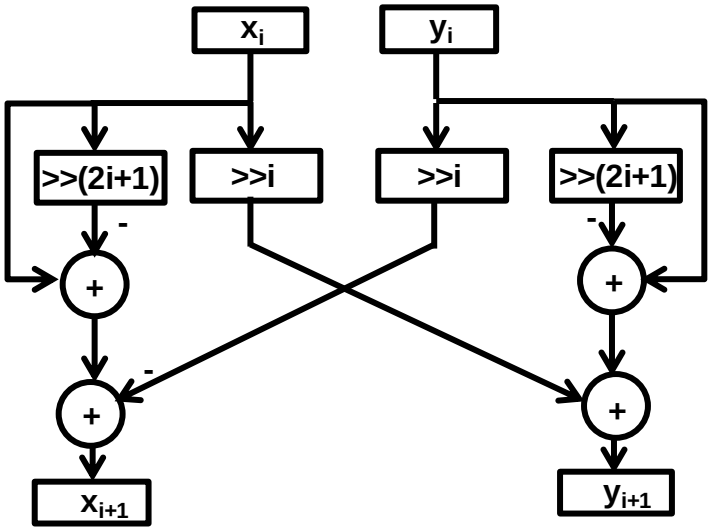

Figure 3.6 Hardware for X and Y datapath for vector rotation through ±3.581° and ±1.79°.

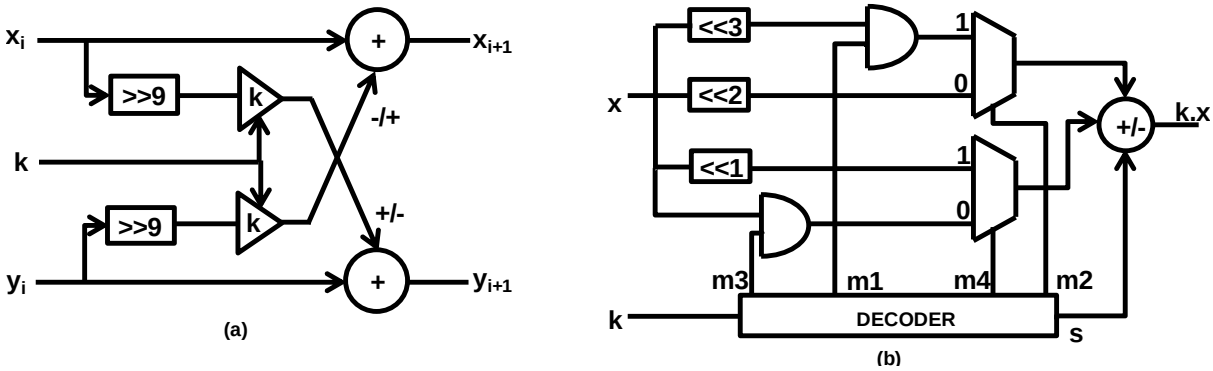

Figure 3.7 Architecture of modified nano-rotator (Stage 5). (a) Modified nano-rotator for angle set α_k= k × 0.112°, k=0, 1...., 8. (b) Multiplication by k unit.

The detail of each pipelined stage is summarised in Table3.1.

Table 3.1 Summary of rotation stages of proposed architecture.

Stage	**Angle Set**	**Maximum Remaining Angle**
1	0°, ±90°, ±180°	±45°
2	0°, ±28.65°	±16.35°
3	0°, ±7.162°, ±14.32°	±3.581°
4	0°, ±1.79°, ±3.581°	±0.895°
5	α_k=k×0.112° k=0,1....,8.	±0.056°

3.3 Hardware Implementation Results

The proposed architecture along with modified virtually scaling free CORDIC algorithm [12], CORDIC II algorithm [4] are coded in VerilogHDL, synthesised in Xilinx ISE 14.7 and mapped onto SPARTAN 3E XC3S500E-FG320-5 FPGA device. Table 3.2 and Table 3.3 show the implementation results and error performance of these algorithms respectively. The algorithms are compared in terms of hardware requirement when mapped onto FPGA device, maximum operating frequency, latency of each algorithm and accuracy of the implementation. Hardware required for implementation of the algorithm onto FPGA device is given in terms of occupied number of slices, total number of four-input LUTs and number of slice flip-flops occupied. The proposed architecture occupies 703 slices (15% of total number of slices available), 1303 four-input LUTs (13% of total number of LUTs available) and 387 slice flip-flops (4% of total number of slice flip-flops available).

Table 3.2: Comparison of hardware cost of CORDIC architectures

Algorithm	**Slice (A)**	**LUTs**	**Slice FFs**	**Max. Freq MHz (B)**	**Latency (C)**	**Slice-Delay Product (A*C/B)**
CORDIC II [4]	597	1078	468	85.85	7	48.58
MVSFA CORDIC [12]	1174	2060	722	68.85	14	238.71
Proposed CORDIC	703	1303	387	75.46	5	46.58

Table 3.3 Comparison of error performance of CORDIC architectures

Algorithm	**Max. Error**		**Avg. Error**	
	SINE	**COSINE**	**SINE**	**COSINE**
CORDIC II [4]	6.01	5.98	6.78	6.71
MVSFA CORDIC [12]	10.97	10.97	12.60	12.57
Proposed CORDIC	7.91	7.93	11.68	11.28

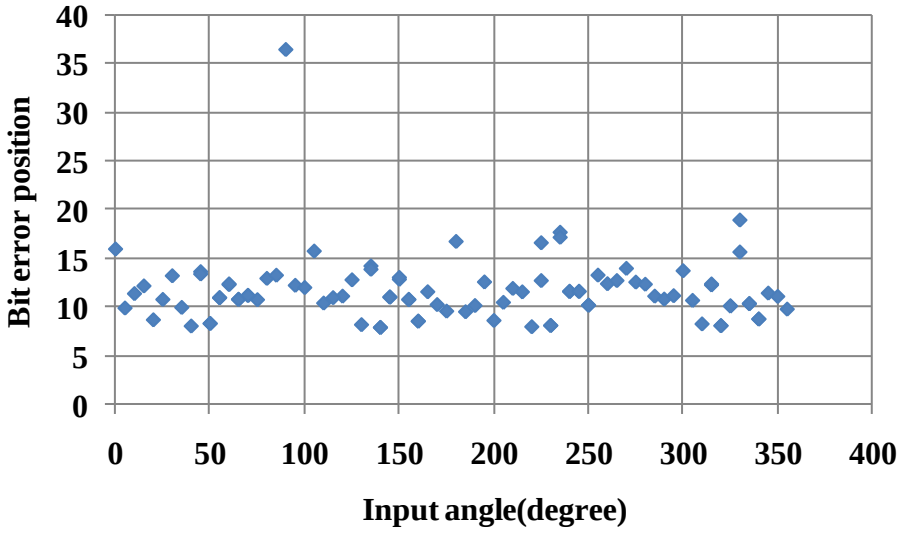

Figure 3.8 Bit error position for sine function

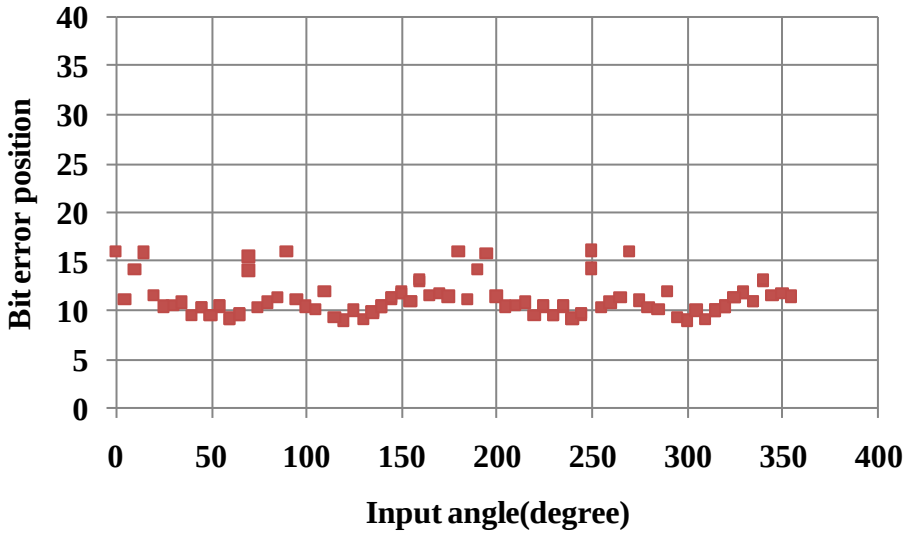

Figure 3.9 Bit error position for cosine function

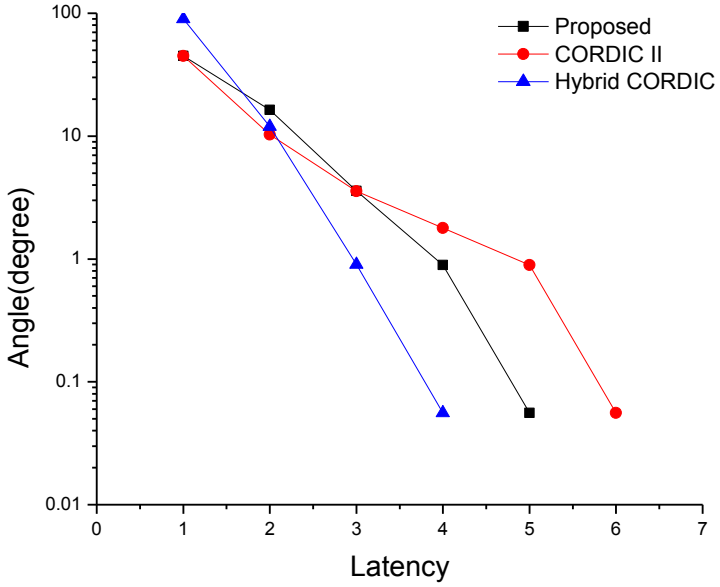

Figure 3.10 Maximum remaining angle of algorithms versus latency

In Table 3.3 error performance of proposed architecture is summarised. The error is measured in terms of Bit Error Position (BEP). For sine function datapath the algorithm has maximum error at 7.9th bit position from most significant bit (MSB) and average error at 11.68th position from MSB. Further for cosine function datapath the maximum and average error occurs at 7.93th and 11.28th bit position from MSB respectively. The error for different input angles is plotted in Figures 3.8 and Figure 3.9 for sine and cosine functions respectively. As shown in Table 3.2 slice-delay product of proposed architecture is least as compared to other architectures. Hence, proposed architecture is most efficient in terms of area- time product. Further, the proposed CORDIC architecture has much better accuracy

than CORDIC II [4]. However, the accuracy of proposed architecture is degraded by only one decimal bit position as compared to modified virtually scaling free CORDIC [12]. Hence, proposed CORDIC architecture has best performance in terms of slice-delay product without compromising on accuracy.

In Figure 3.10 maximum remaining angle versus latency of proposed CORDIC architecture is plotted along with other low latency algorithms such as CORDIC II [4] and Hybrid CORDIC [21]. Plot shows that only Hybrid CORDIC algorithm [21] beat the latency of the proposed CORDIC at the cost of more complex iterations and variable scaling factor.

3.4 Summary

In this chapter a new CORDIC architecture is proposed which has very low latency in pipelined implementation. Additionally, the proposed architecture is completely scaling-free hence scaling factor calculation and compensation is not required. In literature, only Hybrid CORDIC algorithm [21] has less latency than the proposed architecture as Hybrid CORDIC is based on high radix CORDIC. However, the Hybrid CORDIC algorithm [21] requires more calculations at each iteration and produces variable scaling factor. The proposed architecture is coded in VerilogHDL, synthesised in Xilinx ISE14.7 and mapped onto XC3S500E-FG320-5 FPGA device which is from SPARTAN 3E device family. The performance of algorithm is calculated in terms of hardware requirement, maximum frequency of operation, latency, slice-delay product and accuracy. The implementation exhibit the best performance of proposed architecture in terms of slice-delay product without affecting the accuracy.

CHAPTER-4

HARDWARE IMPLEMENTATION OF FFT ALGORITHM USING CORDIC

This chapter explains the hardware implementation of FFT algorithms using the CORDIC architecture.

4.1 Overview of FFT Algorithm

The N- point discrete Fourier transform (DFT) of N-sample signal x[n] is defined as shown in equations 4.1 and 4.2. Where $X_{DFT}[k]$ is a frequency domain signal and x[n] is a time domain signal.

$$X_{DFT}[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N}, \qquad k = 0, 1, 2, \dots\dots, N-1 \tag{4.1}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X_{DFT}[k] e^{-j2\pi nk/N}, \qquad n = 0, 1, 2, \dots\dots, N-1 \tag{4.2}$$

Clearly, the DFT computation involves complex multiplications. N-point DFT implemented by using Equation 4.1 has N^2 complex multiplications and hence has computational complexity of $O(N^2)$. However, fast and efficient computation of DFT is possible by using algorithms called as Fast Fourier Transform (FFT) algorithms. FFT algorithms calculate N-point DFT by calculating many smaller sized DFTs. This results in computational complexity of radix-2 N-point FFT of only $O(N.\log_2 N)$. Hence, FFT algorithms are fast and efficient for hardware implementation of DFT.

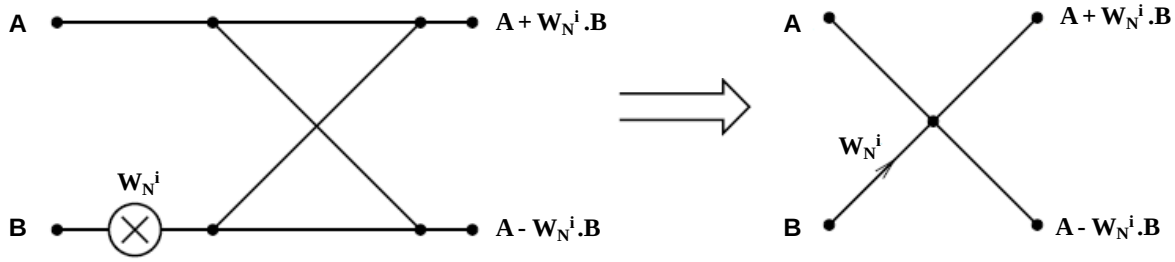

Figure 4.1 Butterfly Operation

This makes FFT a very commonly used algorithm in digital signal processing. The butterfly structure of FFT algorithm is shown in Figure 4.1. Each butterfly uses one complex multiplication and two complex additions. Such butterfly units are repeated so as to form butterfly diagram for given point FFT. Figure 4.2 shows butterfly diagram for radix- 2 four-point DIT–FFT. It requires four butterfly units. The input is decimated in time and given to FFT butterfly unit.

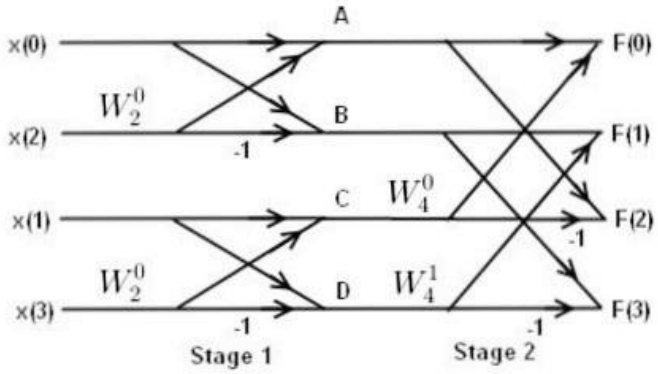

Figure4.2 Butterfly diagram for radix-2 4-point DIT-FFT

4.2 FPGA Implementation of FFT using CORDIC

CORDIC can be exploited to implement the FFT so that the hardware complexity of FFT implementation can be reduced. This is because complex multiplications in FFT computation can be performed using CORDIC algorithm. Hence, complex multiplications get reduced to simple shift and add operations. The idea behind exploitation of CORDIC in FFT computation is explained herewith. Figure 4.1 shows the single butterfly operation. The input to butterfly are A and B and $A+W_N^i.B$ and $A-W_N^i.B$ are outputs. Where W_N^i is called twiddle factor and is given as

$$W_N^i = e^{j2\pi i/N} = \cos\left(2\pi i/_N\right) + j\sin\left(2\pi i/_N\right) \tag{4.3}$$

The inputs A and B to the butterfly unit can be complex. Let $B = x_0 + jy_0$, then the product $W_N^i.B$ is given as

$$W_N^i.B = (\mathrm{x}_0\cos\theta + y_0\sin\theta) + j(-\mathrm{x}_0\sin\theta + y_0\cos\theta) \quad where\ \theta = {}^{2\pi i}/_N \tag{4.4}$$

In general, the CORDIC algorithm for inputs as $X = x_0, Y = y_0\ and\ Z = \theta$ produces the outputs as

$$\begin{bmatrix} X_n \\ Y_n \end{bmatrix} = K_n \begin{bmatrix} \mathrm{x}_0\cos\theta - y_0\sin\theta \\ \mathrm{x}_0\sin\theta + y_0\cos\theta \end{bmatrix} \tag{4.5}$$

Where K_n is the gain of the CORDIC algorithm. From Equations 4.4 and 4.5 it can be observed that if we give the input y_0 to the CORDIC as negative to that of actual input and take output Y_n as negative of the actual output, then CORDIC produces same output as that of complex multiplication operation in Equation 4.4. In this way CORDIC algorithm performs single butterfly operation of FFT algorithm. Figure 4.3 shows the implementation of single butterfly operation. By repeating such butterfly operations FFT can be implemented.

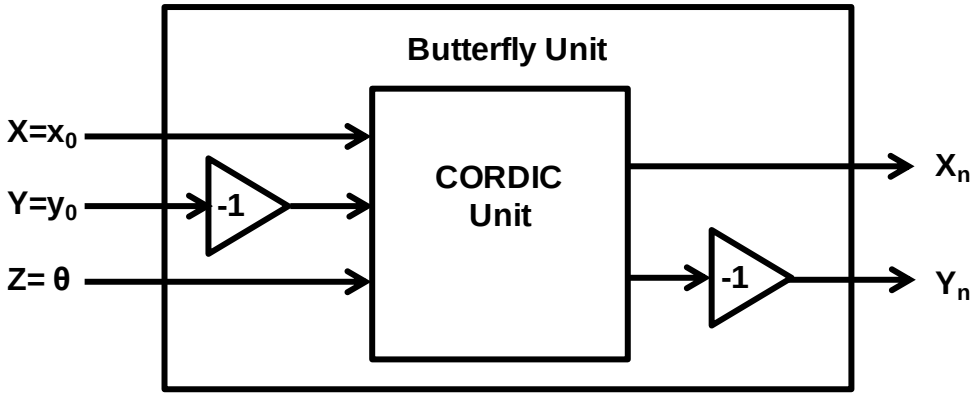

Figure 4.3 Implementation of single butterfly operation

4.3 Hardware Implementation Results

Radix-2 four-point DIT-FFT is implemented in VerilogHDL for a word-length of 16-bits.The FFT implementation is done by using two proposed CORDIC architectures in Chapter 2 and Chapter 3. These implementations are simulated and synthesized using Xilinx ISE14.7 and mapped onto Virtex-4 XC4VLX25-FF668 FPGA device. The Virtex-4 FPGA has total of 10752 slices, 21504 slice flip-flops and 21504 four-input LUTs. Hardware implementation results of these implementations are summarized in Table 4.1. Hardware complexity of particular algorithm is shown in terms of total number of slices, slice-flip-flops and four-input LUTs occupied when mapped onto FPGA. To perform the error analysis of FFT percentage error is calculated. To decide the reference for error calculation FFT function in MATLABR2011b is used. Hence, error is calculated with respect to the output obtained by MATLAB function. For a 16-bit wordlength the bit error position (BEP) in computation of FFT using different CORDIC algorithm is summarized in Table 4.2.

Table 4.1 Comparison of Hardware cost for different FFT implementations

FFT Using	Slice	4-Input LUTs	Slice FFs
Proposed CORDIC (chapter 2)	395	688	615
Proposed CORDIC (chapter 3)	464	810	741

Table 4.2 Comparison error performance for different FFT implementations

FFT Using	Bit Error Position	
	Real Part	Imaginary Part
Proposed CORDIC (chapter 2)	15.66	16.17
Proposed CORDIC (chapter 2)	15.97	16.17

As shown in Table 4.1, FFT implemented by CORDIC architecture proposed in chapter 2 occupies 395 slices (3% of total number of slices available), 688 four-input LUTs (3% of total number of LUTs available) and 615 slice flip-flops (2% of total number of slice flip-flops available) and FFT implemented by CORDIC architecture proposed in chapter 3 occupies 464 slices (4% of total number of slices available), 810 four-input LUTs (3% of total number of LUTs available) and 741 slice flip-flops (3% of total number of slice flip-

flops available). Hence, it is clear that the FFT module implemented using proposed CORDIC architectures occupies very less hardware when mapped onto Virtex-4 XC4VLX25-FF668 FPGA device.

As shown in Table 4.2 the calculated average bit position error for real and imaginary part of the output of FFT implemented using proposed CORDIC architecture in chapter 2 is 15.66 and 16.17 respectively. Moreover, FFT implementation using proposed CORDIC in chapter 3 has average bit error position at 15.97^{th} and 16.17^{th} bit for real and imaginary part respectively. As the given error performance is for 16 bit wordlength, it is clear that for radix-2 four-point FFT has almost no error in output when implemented using any of the proposed architecture. The experimental setup for FFT implementation is shown in Figure 4.4. Figure 4.5 shows the output of FFT computation which is analysed using ChipScope Pro Analyzer in the laptop. In Figure 4.5 inputs (xin0_r, xin0_im,....., xin3_r, xin3_im) are the inputs to FFT module while (X0_R, X0_IM,....., X3_R,X3_IM) are the outputs . Here decimal 1 is represented as 0100 000 000 000 000. All input/outputs are displayed in signed decimal radix.

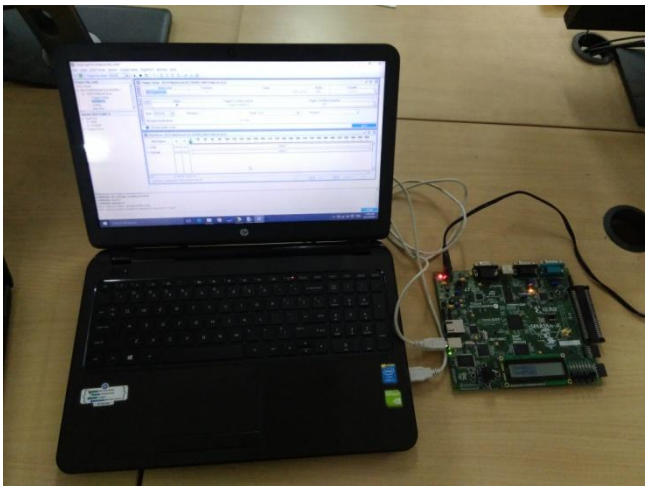

Figure4.4 Experimental Setup

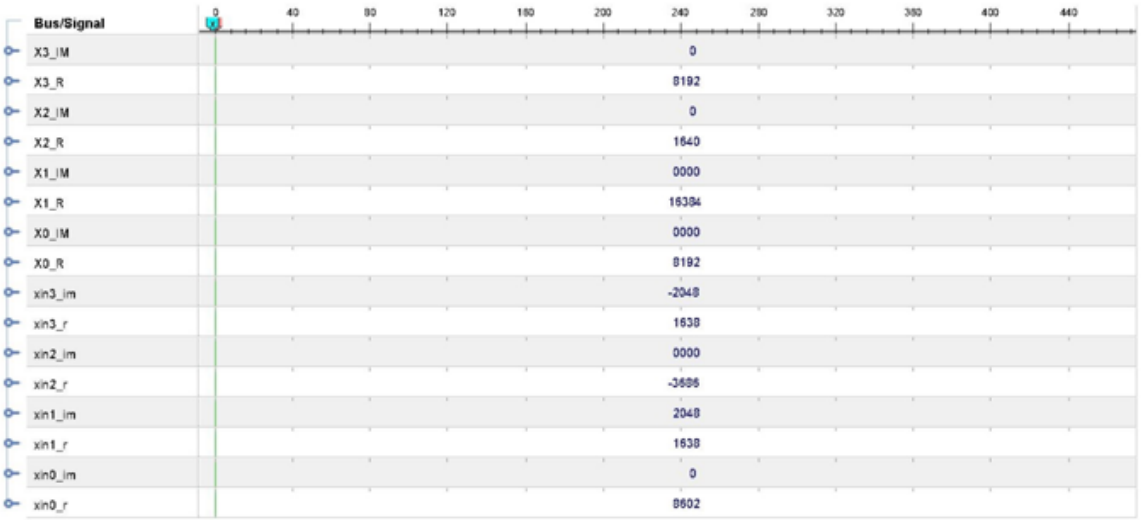

Figure 4.5 Output of FFT computation analysed using ChipScope Pro Analyze

CHAPTER-5

CONCLUSION

The conventional CORDIC algorithm and its various versions are elaborately explained. The CORDIC algorithm is very useful for trigonometric function computations in hardware, as it computes these functions using only add and shift operations. Two new CORDIC architectures are proposed. These architectures along with other CORDIC algorithms are coded in VerilogHDL, synthesised in Xilinx ISE14.7 and mapped onto SPARTAN 3E XC3S500E-FG320-5 FPGA device. Their performance in terms of hardware cost, maximum operating frequency, latency and error in computation of sine and cosine functions are compared with other existing CORDIC algorithms in literature. The proposed architecture discussed in chapter 2 employs novel approach to rotate the input vector by using a sine and cosine function approximation using negative powers of two. For 16 bit wordlength the slice-delay product of proposed CORDIC is 99.07 and average bit error position in sine and cosine function computation is 12.27^{th} and 12.07^{th} bit respectively. Hence, from the comparison with other algorithms, it is concluded that proposed architecture is most efficient when both hardware requirement and error performance are taken into account. It is shown that proposed CORDIC architecture in chapter 3 has latency of only 5 clock cycles whereas CORDIC II [4] and modified virtually scaling free CORDIC [12] has latency of 6 and 14 clock cycles respectively. Hence, proposed CORDIC architecture provides very low latency. The implementation exhibit the best performance of proposed architecture in terms of slice-delay product. Additionally, both the proposed CORDIC architectures are completely scaling free. Hence, need scaling factor calculation and/or compensation is totally avoided which required in case of other CORDIC algorithms [4] [12] [21].

For efficient hardware implementation of FFT, the CORDIC architecture is exploited to reduce the hardware complexity. The radix-2 four-point DIT-FFT is implemented using two proposed architectures. These implementations are coded in VerilogHDL, synthesised in Xilinx ISE14.7 and mapped onto Virtex-4 XC4VLX25-FF668-12 FPGA device. These two FFT implementations are compared in terms hardware cost and average bit error position in FFT computation. The FFT implementation using proposed CORDIC architecture in chapter 2 and chapter 3 occupies 395 (3% of total number of slices available) and 464 (4% of total number of slices available) slices respectively when mapped onto Virtex-4 XC4VLX25-FF668-12. Hence, it is concluded that FFT implementation using proposed CORDIC architectures in chapter 2 and chapter 3 occupies very less hardware. For 16 bit wordlength, the average bit position error in FFT computation for real and imaginary part when implemented using proposed CORDIC architecture in chapter 2 is 15.66 and 16.17 respectively. Moreover, FFT implementation using proposed CORDIC in chapter 3 has average bit error position at 15.97^{th} and 16.17^{th} bit for real and imaginary part respectively. Hence, for 16 bit wordlength the radix-2 four-point FFT has almost no error in output when implemented using proposed CORDIC architectures in chapter 2 and chapter.

References

[1] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electronic Computing,* vol. EC-8, no.3, pp. 330-334, Sep. 1959.

[2] Y. H. Hu, "CORDIC-based VLSI architectures for digital signal processing," *IEEE Signals Process. Mag.,* vol. 9, no. 3, pp. 13-35, Jul. 1992.

[3] K. Maharatna, A. S. Dhar, and S. Banerjee, "A VLSI array architecture for realization of DFT, DHT, DCT and DST," *Signal Process.,* vol. 81, pp. 1813-1822, 2001.

[4] Mario Garrido, Petter Källström, Martin Kumm and Oscar Gustafsson, "CORDIC II: A New Improved CORDIC Algorithm," *IEEE Trans. Circuits Syst. II :Exp. Briefs*, vol. PP, no. 99, pp. 1549-7747, Sept. 2015*(Early access).*

[5] M. Garrido and J. Grajal, "Efficient memoryless CORDIC for FFT computation," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing,* vol. 2, Apr. 2007, pp. II-113-116.

[6] M. C. Mandal, A. S. Dhar, and S. Banerjee, "Multiplierless array architecture for computing discrete cosine transform" *Comput. Elect. Eng.* , vol. 21, no. 4, pp. 327-333, 1994.

[7] A.S. Dhar and S. Banerjee, "An array architecture for fast computation of discrete Hartley transform", *IEEE Trans. Circuits Syst.,* vol.38, no.9, pp. 1095-1098,Sep. 1991.

[8] K. Maharatna and S. Banerjee, "CORDIC based array architecture for affine transformation of images," in *Proc. Int. Conf. Communications, Computers and Devices,* Kharagpur, India, Dec. 2000, vol. 2, pp. 645-648.

[9] A. Troya, K. Maharatna, M. Krsti, E. Grass, U. Jagdhold, and R. Kraemer, "Low-power VLSI implementation of the inner receiver for OFDM-based WLAN systems," *IEEE Trans. Circuits Syst. I, Reg. Papers,* vol. 55, no. 2, pp. 672-686, 2008.

[10] T. Jaung, S. Hsiao, and M. Tsai, "Para-CORDIC: Parallel CORDIC rotation algorithm," *IEEE Trans. Circuits Syst. I: Reg. Papers,* vol. 51, no. 8, pp. 1515-1524, Aug. 2004.

[11] C. Wu and A. Wu, "Modified vector rotational CORDIC (MVR-CORDIC) algorithm and architecture," *IEEE Trans. Circuits Syst. II, Exp. Briefs,* vol. 48, no. 6, pp. 548-561, Jun. 2001.

[12] K. Maharatna, S. Banerjee, E. Grass, M. Krstic, and A. Troya, "Modified virtually scaling-free adaptive CORDIC rotator algorithm and architecture," *IEEE Trans. Circuits Syst. Video Technol.,* vol. 11, no.11, pp.1463-1474, Nov.2005.

[13] F.J. Jaime, M. A. Sanchez, J. Hormigo, J. Villalba, and E. L. Zapata, "Enhanced scaling-free CORDIC," *IEEE Trans. Circuits Syst. I, Reg. Papers,* vol. 57, no. 7, pp. 1654-1662, Jul.2010.

[14] S. Aggarwal, P. K. Meher, and K. Khare, "Area-time efficient scaling-free CORDIC using generalized micro-rotation selection," *IEEE Trans. VLSI Syst.,* vol. 20, no. 8, pp. 1542-1546, Aug. 2012.

[15] S. Aggarwal, P. K. Meher, and K. Khare, “Concept, design, and implementation of reconfigurable CORDIC,” *IEEE Trans. VLSI Syst.*, vol. PP, no. 99, pp-1063-8210, Jul. 2015 *(Early Access).*

[16] J. Villalba, T. Lang, and E. L. Zapata, “Paraleel compensation of scale factor for the CORDIC algorithm,” *J. VLSI Signal Process. Syst.*, vol. 19, no. 3, pp. 227-241, Aug. 1998.

[17] M. G. B. Sumanasena, “A scale factor correction scheme for the CORDIC algorithm,” *IEEE Trans. Comput.*, vol. 57, no. 8, pp. 1148-1152, Aug. 2008.

[18] N. Takagi, T. Asada, and S. Yajima, “Redundant CORDIC methods with a constant scale factor for sine and cosine computation,” *IEEE Trans. Comput.*, vol. 40, no.9, pp. 989-995, Sep. 1991.

[19] P. K. Meher, J. Walls, T. B. Juang, K. Srisharan, and K. Maharatna, “50 years of CORDIC: Algorithms, architectures and applications,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 9, pp. 1893-1907, Sep.2009.

[20] R. Andraka, “A survey of CORDIC algorithms for FPGA based computers,” in *Proc. ACM/SIGDA Int. Symp. FPGAs*, Feb. 1998, pp. 191-200.

[21] R. Shukla and K. Ray, “Low latency hybrid CORDIC algorithm,” *IEEE Trans. Comput.*, vol. 63, no. 12, pp. 3066–3078, Dec 2014.

[22] R. M. Jiang, "An area-efficient FFT architecture for OFDM digital video broadcasting", *IEEE Trans. Consumer Electron.*, vol. 53, pp. 1322-1326, 2007.

[23] S.-N. Tang, C.-H. Liao and T.-Y. Chang, "An area- and energy-efficient multimode FFT processor for WPAN/WLAN/WMAN systems", *IEEE J. Solid-State Circuits*, vol. 47, no. 6, pp. 1419-1435, 2012.

[24] El-Motaz, Mohammed A.; Nasr, Omar A.; Osama, Karim "A CORDIC-friendly FFT architecture", *Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International,* On page(s): 1087 – 1092.

[25] K.R.Rao, D.N.Kim, J.J.Hwang, “Fast Fourier Transform: Algorithms and Applications,” *Springer* 2010.

Publications

[1] **Sumit S. Wadkar** and Bishnu Prasad Das, "Low Latency Scaling-free CORDIC Architecture with Efficient Area-Time Product," *IEEE Trans. Circuit Syst. II, Exp. Briefs*. (Under preparation)

[2] **Sumit S. Wadkar** and Bishnu Prasad Das, "A New Efficient Scaling-free CORDIC Architecture," *IEEE Trans. VLSI Syst.*.(Under preparation)