Master's Thesis Report

Design of FFTs using CORDIC and Parabolic Synthesis as an alternative to Twiddle Factor Rotations

By

Muhammad Waqas Shafiq
Nauman Hafeez

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Abstract

Fast Fourier Transform (FFT) processor is an important signal processing block widely adopted in various disciplines, such as within the computer area, computer graphics, digital signal processing, communication systems, robotics, navigational systems, etc. Advanced and complex algorithms in these disciplines need higher computation performance and low power utilization processes. Due to advancements and down scaling of hardware technologies, it becomes possible to fulfill these higher demands from the most advanced algorithms with higher clock rates on the chips. The improvement in hardware technology performance has moved the interest more over to the hardware implementation of algorithms.

The scope of this thesis is to investigate different algorithms to compute the rotations, often done by complex twiddle factor multiplications in in run time, in FFT processor architectures. The FFT processor is commonly implemented with a complex multiplier but due to demand of higher point FFTs in the applications, the size of ROM in the multiplier based implementation for the twiddle factors becomes the matter of concern with larger chip area. The aim of this thesis is to design an FFT with un-rolled CORDIC and Parabolic Synthesis algorithms in order to replace complex multipliers and ROMs.

The outcome of the presented thesis is comparison results for accuracy, power consumption, performance and area of these algorithms in the STM 65nm CMOS technology with different transistor technologies and power supplies. It is noted that implementation with Parabolic synthesis algorithm gives better results as it automatically gives a high degree of parallelism that gives a shorter critical path and thereby fast computation. The structure of the methodology will also assure an area efficient hardware implementation if higher point FFTs are implemented.

Acknowledgments

It is a pleasure to thank the many people who made this thesis possible.

It is difficult to overstate our gratitude to our supervisor, Dr. Peter Nilsson. With his enthusiasm, his inspiration, and his great efforts to explain things clearly and simply, he helped to make this thesis fun for us. Throughout our thesis work, he provided encouragement, sound advice, good teaching, good company, and lots of good ideas. We would have been lost without him. We are also grateful to Erik Hertz who helped us to understand his research work.
We would like to thank the many people who have taught us throughout the MSc especially Joachim Rodrigues, Viktor Öwall, Markus Törmänen, Pietro Andreani and all Teacher assistants for their kind assistance with our courses and lab works especially Yasser Sherazi, Chenxin Zhang, Reza Meraji, Johan Löfgren, Stefan Molund, and Jonas Lindstrand.

We wish to thank our best friends Minna, Waqas Khan, Umair, Farhan, Muaz, Salman, Naveed, Adnan, Shoaib, Ammar and Naeem for providing a stimulating and fun environment in which I learnt and grow.

We wish to thank our entire families for providing a loving environment for us. My brothers, my sister Kayynat were particularly very supportive and had faith in me.

Lastly, and most importantly, we wish to thank our parents. They bore us, raised us, supported us, taught us, and loved us. To them I dedicate this thesis.

Table of Contents

Preface

This thesis is submitted in partial fulfillment of the requirements of Master's degree in System on Chip at LTH, Lund University. This thesis has been made solely by the authors, however most of the text is based on the research of others, we tried to provide references where necessary.

In this thesis work, initial literature review have been done together with discussions and note taking to proceed with development process. Nauman Hafeez has done design of 14 stage un-rolled CORDIC and Parabolic Synthesis design in VHDL, tested and verified their simulation with MATLAB design. Waqas Shafiq has done design of 16-point FFT, test and verified with MATLAB results. He has also done MATLAB reference model design. Result extractions of area, speed and power after synthesis process have been done together.

CHAPTER 1

1. Introduction

1.1 Overview

In past few years the advancements in VLSI technologies have opened many windows for designing real time applications, using efficient algorithms with custom chips. The use of special arithmetic techniques, parallel and pipelined architectures have led the designs far away from basic computers.

Nowadays the world of information technology has lined up many areas together such as communication systems, computer graphics, robotics, navigations, astrophysics etc. and all applications demands high performance with minimum possible power consumption. Many digital signal processor designers and manufacturers are facing one big challenge, that is, how to perform complex mathematical function calculations more efficiently. To gain efficiency, it is needed to dig inside our complete designing cycle which contains algorithms, architectures, hardware technology, power supply etc.

Fast Fourier Transform (FFT) is widely used transform in digital applications especially in communication systems. An FFT is an important processing block in these systems, which takes most of the hardware complexity in a digital baseband transceiver for instance. Due to demand in higher data rates in communication systems, large-point FFTs are required for multiple carrier modulation, such as 1024/2048/8192 etc. An FFT is commonly implemented with complex multiplier; a complex multiplier is equivalent to four real multipliers and two real adders, and a ROM to store the twiddle factors. The ROM in this type of implementation takes most of the chip area, consumes more power and degrades the speed because of ROM read operation ROM size increased with large-point FFTs. Hence poor performance of the FFT in terms of power, speed, and area can be seen.

The main objective of the thesis is to investigate algorithms which can be used to calculate trigonometric functions and perform complex multiplications. An un-rolled CORDIC architecture is widely used instead of complex multiplier and ROM based architectures. Power consumption, speed and accuracy are issues in this architecture implementation. A newly invented “Parabolic Synthesis” algorithm is also investigated that could improve the results with respect to CORDIC based implementation of the FFT. Both algorithms are implemented in VHDL, synthesized and compared in terms of power, area, accuracy and timing. Both algorithms are tested to calculate the rotations and to perform complex multiplications for a 16-Point FFT processor.

Traditional methodologies such as use of complex multipliers with pre-computed twiddling factors (stored in ROM/Registers) for FFT processors are also being compared with CORDIC and Parabolic Synthesis.

1.2 Organization of Thesis

Chapter 2 explains the basic theory and fundamental equations of CORDIC algorithm along with the hardware architecture.

Chapter 3 contains the brief description of Parabolic Synthesis and covers the basic theory, mathematical equations and hardware architectures used to implement this algorithm for FFT processors.

Chapter 4 is a review of FFT and explanation of butterfly architecture is provided, and Single-path Delay Feedback (SDF) architecture for implementing FFT is also discussed and implemented in hardware.

Chapter 5 contains the synthesis results of tested algorithms and gives the guideline to the reader about the selection of algorithm, the detail comparison between power, area and speed using different transistor technologies and supply voltages, is presented.

Chapter 6 summarizes the results achieved in the project.

Chapter 7 discusses the future work in order to improve performance.

Chapter 2

2. Fast Fourier Transform

The Fast Fourier Transform (FFT) architecture was invented in 1965 by Cooley and Tukey [8] and is not a different algorithm from DFT. It is architecture for efficient computation of DFT.

2.1 Discrete Fourier Transform

DFT is the most widely used transform of all the available transforms in digital signal processing. The DFT maps the input sequence $x(n)$ into frequency domain.

The Discrete Fourier Transform (DFT) of $x(n)$ input sequence is defined as in (2.1), which is an N-point sequence [9].

$$X(k) = \sum_{n=0}^{N-1} x(n)\ W_N^{kn} \qquad k = 0,1,\dots,N-1 \tag{2.1}$$

Where $x(n)$ and $X(k)$ are complex in general and the indices n and k are integers.

The term $W_N$is twiddle factors, which is described in (2.2)

$$W_N = e^{\frac{-2\pi j}{N}} = \cos\left(\frac{2\pi}{N}\right) - jsin\left(\frac{2\pi}{N}\right) \tag{2.2}$$

The DFT is complex valued but in hardware, the real (Re) and imaginary (Im) parts are separated in two real parts as in (2.3)

$$X_{Re}(k) = \sum_{n=0}^{N-1} \left(x_{Re} \cos\frac{2\pi kn}{N} - x_{im} \sin\frac{2\pi kn}{N}\right)$$
$$X_{Im}(k) = \sum_{n=0}^{N-1} \left(x_{Re} \sin\frac{2\pi kn}{N} + x_{im} \cos\frac{2\pi kn}{N}\right) \tag{2.3}$$

2.2 FFT Algorithms

Fast Fourier Transform (FFT) [8] is an efficient algorithm for computing the DFT. Its principle is based on decomposing the computation of the discrete Fourier transform with sequence of length N into successively smaller discrete Fourier transforms. Computation complexity is reduced from $O(N^2)$ to O(Nlog(N)) after

introducing the FFT architecture to calculate a DFT. There are different algorithms of FFT based on different decomposition schemes. Some of the algorithms are described next.

2.2.1 Decimation in Time (DIT) algorithm

The algorithm which decomposes the sequence $x(n)$ into subsequently smaller sequences is called decimation in time algorithm [10]. In order to explain it, a radix-2 DIT algorithm is explained to demonstrate the decomposition.

In (2.1), the input sequence is divided into even and odd numbered sequences. In this case, the DFT is described as

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n)\, W_{\frac{N}{2}}^{kn} + W_N^{kn} \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\, W_{\frac{N}{2}}^{kn}, k = 0,1,\ldots,\frac{N}{2}-1 \quad (2.4)$$

The input sequence $x(n)$ is split into even numbered $x(2n)$ sequence and odd numbered $x(2n+1)$ sequence.

For larger N, the complexity is reduced from $O(N^2)$ to nearly half by dividing the input sequence into even and odd numbered sequences. The complexity can be further reduced by using the symmetry and periodicity of twiddle factors.

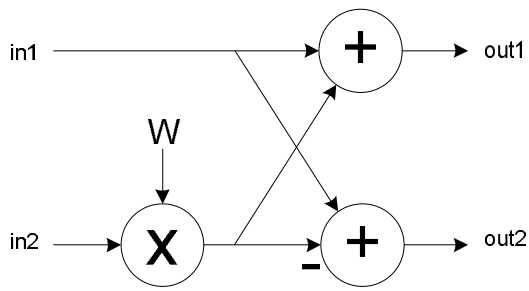

Figure 2.1 A Radix-2 DIT Butterfly

Due to the even and odd numbered sequence split, the twiddle factor is also simplified. The architecture of radix-2 DIT butterfly only needs one complex multiplier and two complex adders instead of two complex multipliers.

2.2.2 Decimation in Frequency (DIF) algorithm

In this algorithm, the output of the DFT is divided into smaller subsequent sequences. DFT main equation is recalled as in (2.1).

Even numbered output sequence can be described in (2.5)

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} x(n)\, W_N^{2kn} + \sum_{\frac{N}{2}}^{N-1} x(n)\, W_N^{2kn}, k = 0,1,..,\frac{N}{2}-1 \quad (2.5)$$

By changing the second summation from $\frac{N}{2}$ to N into the summation from 0 to $\frac{N}{2}$ and by using the property of W_N^{2kn}, (2.5) can be rewritten as in (2.6)

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} \left(x(n) + x\left(n+\frac{N}{2}\right)\right) W_{\frac{N}{2}}^{2kn}, k = 0,1,..,\frac{N}{2}-1 \quad (2.6)$$

In the same way, odd numbered sequence can be expressed as in (2.7)

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} \left(x(n) - x\left(n+\frac{N}{2}\right)\right) W_N^n\, W_{\frac{N}{2}}^{2kn}, k = 0,1,..,\frac{N}{2}-1 \quad (2.7)$$

(2.6) and (2.7) are combined to get the first order DIF derivation. First step is to compute $x(n) + x\left(n+\frac{N}{2}\right)$ and $x(n) - x\left(n+\frac{N}{2}\right)$ and then multiply the latter term withW_N^n. Fig. 2.2 shows a simple radix-2 DIF butterfly.

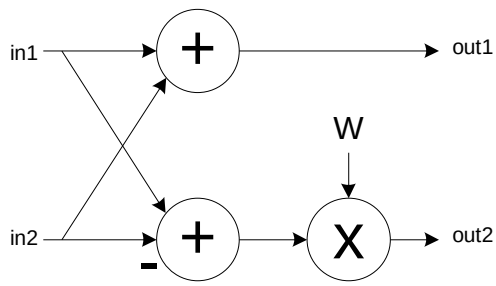

Figure 2.2 A Radix-2 DIF Butterfly

A 16-point radix-2 DIF FFT architecture is chosen to be used in this project. The signal flow graph of the architecture is presented in (2.3).

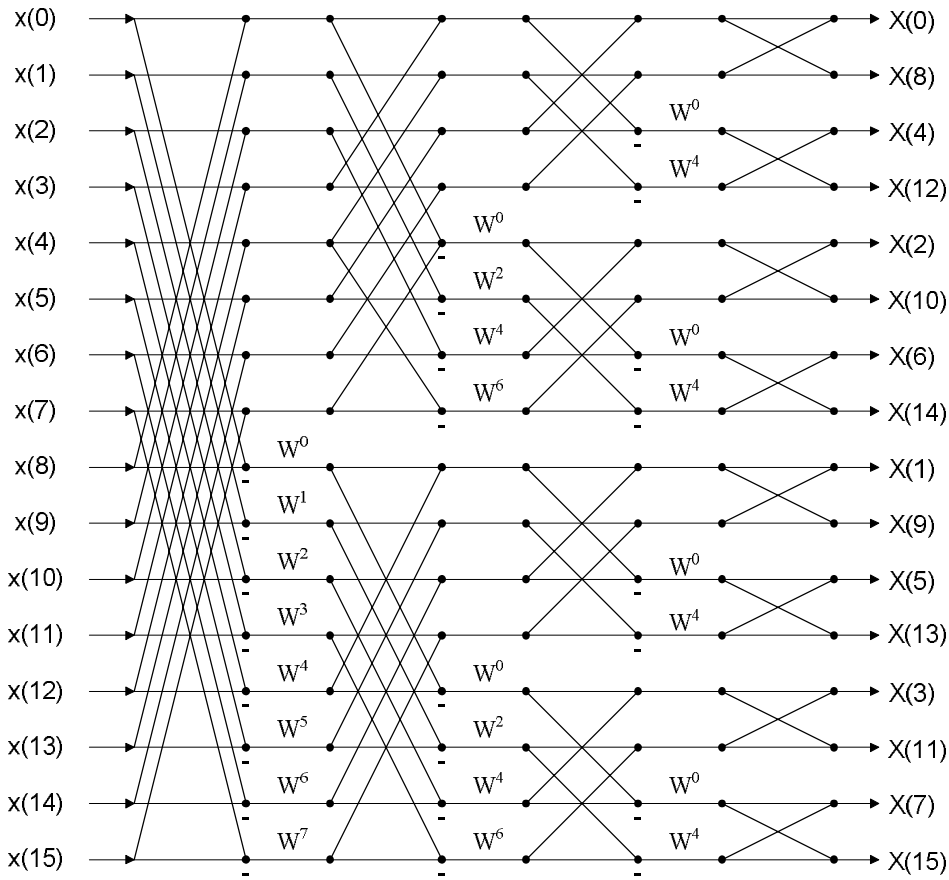

Figure 2.3 A 16-point radix-2 DIF FFT algorithm

Fig. 2.3 shows the signal flow graph for a 16-point FFT. Two structures as the one in the figure are needed, one for the real part and one for the imaginary part. The output is in bit-reversal order i.e., the bits are flipped at the output.

The signal flow graph in Fig. 2.4 shows 4 stages of butterflies. In between a multiplication with the twiddle factor W is shown according to (2.7).

2.3 Hardware Architecture

Figure 2.4 describes a Single-path Delay Feedback (SDF) architecture of a 16-point FFT using multipliers. The pre-computed twiddle factors are stored in an array or ROM if large-point FFTs are to be designed. A controller is used to generate the addresses to access the stored twiddle factors and to control multiplexers in every clock cycle.

Fixed point 2's complement data format is used. In 2's complement, first bit of a positive number is always 0 and rest of bits are ordinary binary values. For negative numbers, invert its standard binary value and add one to it.

Complex data (real and imaginary) has been used for both twiddle factors and input sequence. Both real and imaginary values are 15 bits each; 1 sign bit, 4 integer bits and 10 fractional bits to get 3 decimal places of accuracy in the fractional part. Data scaling and truncation between different stages of the architecture due to multiplication is done according to precision requirements.

Fig. 2.4 shows an architecture for a 16-point FFT architecture with multipliers. In the first phase, 8 input samples are, coming serially, are stored in the 8 registers of 15 bits each during the first stage.

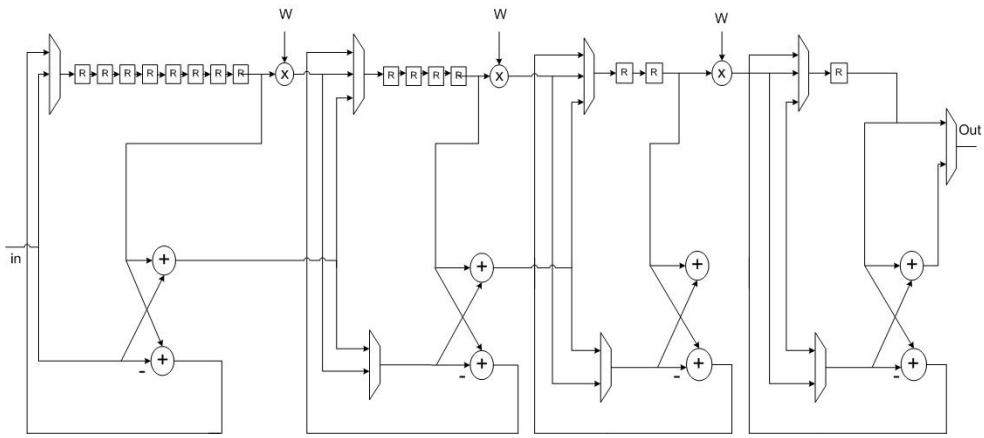

Figure 2.4 A 16-point FFT architecture using multipliers

In the second phase, the first 8 samples, stored in registers, are processed, in the butterfly, with the remaining 8 samples. . The reason behind is that the 1st sample, x(0) has to be processed with the 9th sample, x(8). The second sample, x(1) should be processed with 10th sample, x(9). The butterfly will generate the sum and difference of two input samples. The result of difference of input samples will eventually be stored in the register and multiplied with twiddle factor. In contrast, the addition of two input samples is sent to the next stage without any multiplication with a twiddle factor. The MUXes in this architecture are controlled by a controller, which keep track of the values that are sent to the next stage in the specific clock cycle. In the second stage, the same operations are done but in two parts. The first 4 results from the first stage are stored in the 4 registers in the second stage. These 4 samples are processed with the upcoming 4 results in the butterfly. The same procedure is done with the next results from the first stage. This architecture generates the first end result in the 17th clock cycle. It produces all the 16 results in 32 clock cycles.

Twiddling factors approximation and complex multiplication can be done by a coefficient ROM and a multiplier, by using CORDIC algorithm, and by Parabolic Synthesis as described in Fig. 2.5.

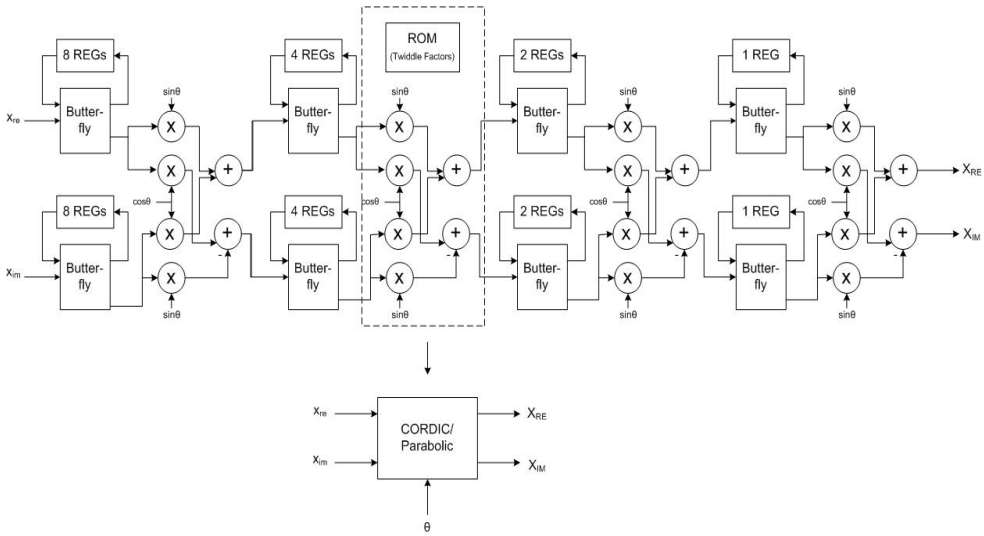

Figure 2.5 A 16-point FFT architecture

Figure 2.5 shows a complete architecture of a 16-point FFT with both real and imaginary parts. Complex multiplier (comprising of 4 real multipliers and 2 real adders) and ROM can be replaced with CORDIC and Parabolic Synthesis algorithms to calculate real and imaginary values for next stage in the FFT. An angle "θ" and previous stage real and imaginary values are provided to the CORDIC algorithm, it will generate the final X_{re} and X_{im} values at the output for the next stage of the FFT. In the same manner, an angle "θ" and previous stage real and imaginary values are provided to the Parabolic Synthesis algorithm, which will generate the rotations, with the final X_{re} and X_{im} values at the output according to (2.3).

CHAPTER 3

3. CORDIC

The CORDIC (**CO**ordinate **R**otation **DI**gital **C**omputer) is a simple and efficient "shift-add" algorithm to calculate the wide range of functions including trigonometric, logarithmic, hyperbolic and linear. It is commonly used when there is no hardware multiplier is available.

The CORDIC algorithm was first described by Jack E.Volder in 1959. It was developed to provide the digital solution for the real-time navigation problems [1].

Two basic CORDIC modes are well known for the computation of different functions, the rotation mode and vector mode. The CORDIC algorithm can be taken in as iterative structure of additions, subtractions and binary shift operations, which can perform fixed angle rotation (also known as micro-rotations).

The CORDIC algorithm is well suited for VLSI implementations due to the simplicity of the involved operations. The CORDIC does not provide the perfect rotation when it is involved with the complex multiplication because the rotated vector gets scaled. To achieve perfect angle rotation, the scaling factor can be easily corrected. The idea behind the CORDIC for rotating the complex number by successive constant angles is highly useful and needs to be compared with other algorithms, which are claiming the similar functionality.

3.1 Rotation

As the nature of CORDIC algorithm is based on simplicity, and the basic observation of unit-length vector can give the foundation of CORDIC algorithm.

The point **z** in Fig. 3.1 at coordinates (x, y) = (1, 0) is rotated by the angle θ and the new point **z'** will be at (x, y) = ($\cos\theta$,$\sin\theta$). Thus, the trigonometric functions $\cos\theta$ and $\sin\theta$ can be computed by finding the co-ordinates of point **z'**, which is a rotated vector of angle θ [2].

The rotation of the vector in a rectangular coordinate system from one point to another point can be performed by simply multiplying the vector by a complex number. Let the initial vector is

$$z = x + jy \tag{3.1}$$

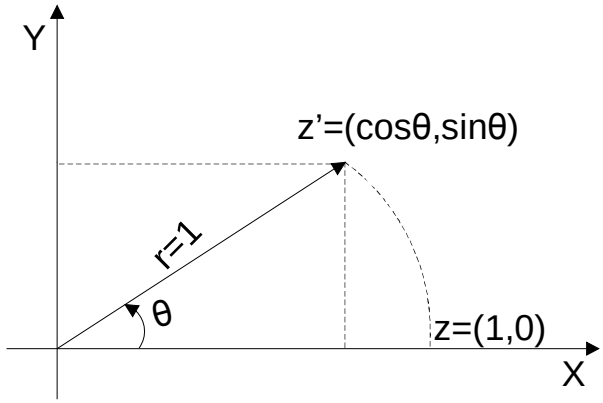

Figure 3.1 Unit-length vector in Cartesian coordinate system

And the desire vector is z' which is rotated by angle θ

$$z' = x' + jy' \tag{3.2}$$

z' can easily be computed by multiplying z with $e^{j\theta}$

$$z' = z\, e^{j\theta} \tag{3.3}$$

$$e^{j\theta} = \cos\theta + j\sin\theta \tag{3.4}$$

$$z' = (x + jy)(\cos\theta + j\sin\theta) \tag{3.5}$$

$$z' = x\cos\theta + jx\sin\theta + jy\cos\theta - y\sin\theta \tag{3.6}$$

$$z' = (x\cos\theta - y\sin\theta) + j(x\sin\theta + y\cos\theta) \tag{3.7}$$

Equation (3.7) gives us the new rotated points of the vector z', as shown in Fig. 3.2.

The complex number rotations in 2-dimensions are commutative [3], unlike in higher level of dimensions. By observing the equation (3.5), whenever two complex numbers gets multiplied with each other, their phase is added up and the magnitude gets multiplied. Thus, during the multiplication with the complex number with unity magnitude we are not affecting the magnitude but only rotating the vector by the desired angle. Such a rotation is known as a Real rotation as shown in Fig. 3.2.

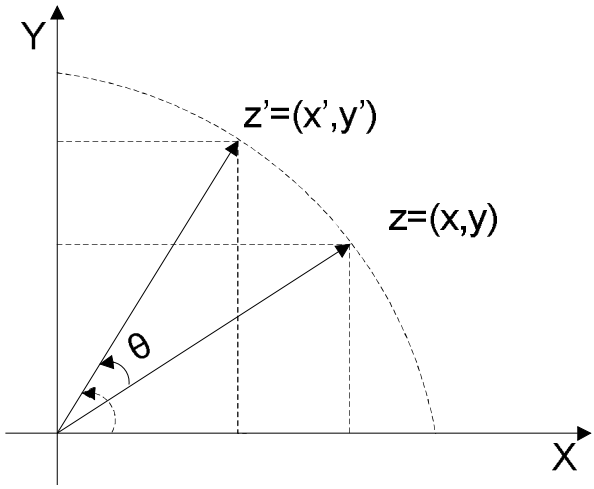

Figure 3.2 Real Rotation of the vector

$$x' = x \cos\theta - y \sin\theta \tag{3.8}$$

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

Let us take (3.8) and (2.9) one step further and make our way towards the CORDIC

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

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

$$x' = \cos\theta\,(x - y \tan\theta) \tag{3.10}$$

$$y' = \cos\theta\,(y + x \tan\theta) \tag{3.11}$$

In (3.10) and (3.11) the factor $\cos\theta$ provides the scaling in rotated angle and causes the reduction in magnitude due to that fact that $\cos\theta \leq 1$, but the term $\tan\theta$ brings ease in computation because it is simply a shift operation.

To further investigate the (3.10) and (3.11) and neglecting the scaling factor $\cos\theta$ from the above equation

$$x^p = x - y \tan\theta \tag{3.12}$$

$$y^p = y + x \tan\theta \tag{3.13}$$

Equation (3.12) and (3.13) are the coordinates of the rotated vector without any scaling factor, the absence of scaling factor can affect the results during the rotations but at the end of total computation this factor can be multiplied to balance the effect. The new rotated values x^p and y^p are shown in Fig. 3.3, such rotation is known as Pseudo Rotation. Fig. 3.3 shows the different between the real rotation and pseudo rotation.

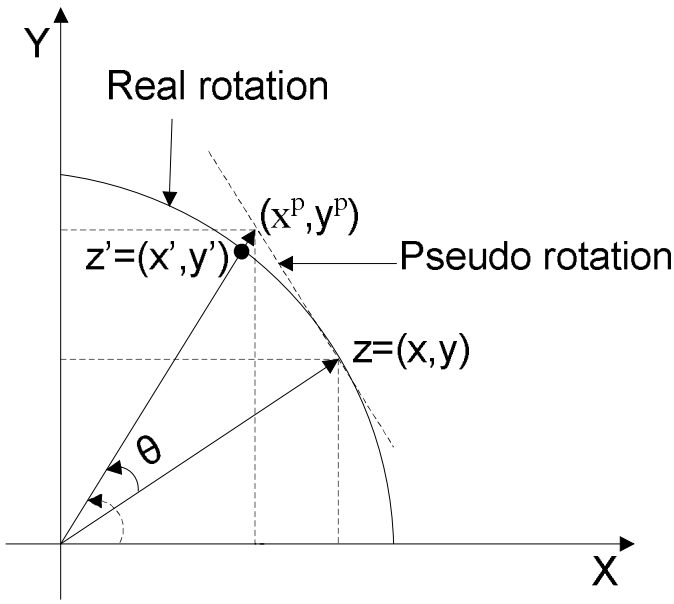

Figure 3.3 Real and Pseudo Rotation of the vector

3.2 CORDIC Formulation

Jack E.Volder has proposed the idea to break the rotation angle θ into the series of smaller angles $\boldsymbol{\alpha_i}$ [4] such that the resultant series could utilize the property of tangent function

$$\tan\alpha_i = \mathbf{2}^{-i} \tag{3.14}$$

The resultant series of angles is shown below in Table 3.1

Table 3.3.1 Pre-Computed angle set

i	$\alpha_i$deg	$\tan\alpha_i$
0	45	1
1	26.5	0.5
2	13.25	0.25
3	7.125	0.125
4	3.576	0.0625
5	1.789	0.03125
6	0.895	0.015625
7	0.447	0.00781
8	0.223	0.00390
9	0.115	0.001953125
10	0.055	0.0009765625

Using (3.14) in (3.12) and (3.13), the foundation of the CORDIC equation is verified as, which is

$$x_{i+1} = x_i - y_i 2^{-i} \tag{3.15}$$

$$y_{i+1} = y_i + x_i 2^{-i} \tag{3.16}$$

The rotations in each stage around the desired angle could be clock-wise or anti-clock-wise, and with the sequential values of **i** the angle α_i and $\tan\alpha_i$ goes on decreasing as shown in following equations.

$$\theta = \pm\theta_1 \pm\theta_2 \pm\theta_3 \pm\theta_4 \pm\theta_5 \pm\theta_6 \pm\theta_7 \ldots\ldots..$$

$$\theta = \tan^{-1}2^0 + \tan^{-1}2^{-1} + \tan^{-1}2^{-2} + \tan^{-1}2^{-3} + \tan^{-1}2^{-4} \ldots\ldots.$$

During the angle rotation, the resultant angles either perform addition or subtraction depending on the *operation deciding factor* $\boldsymbol{d_i}$. Generally, the CORDIC algorithm for other mathematical functions depends on $\boldsymbol{d_i}$ as-well.

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

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

Example: To compute the angle θ = 30º, using the rotation algorithm, is done by breaking the angles into smaller parts

$$30^o \approx 45^o - 26.6^o + 14^o - 7.1^o + 3.6^o + 1.8^o - 0.9^o + 0.4^o$$

All the angles in series against the angle 30º can be taken from the Table 3.1. These angle values can easily be computed using shift, add and subtract operations.

3.3 Unrolled CORDIC ARCHITECTURE

An eleven stage CORDIC is shown in Fig. 3.4. In the eleven adders at the top, the remaining angle is computed for each stage. The input variable θ is the angle for the cosine and sine that is searched for. At the top, the fixed coefficient angle values α_i, for each rotation are provided. These are added or subtracted from θ in each stage. They are fixed coefficients (hardware wired) but they can be stored in a ROM as well.

In the middle and the lower adder rows the approximation of the real and imaginary signals, $X_{re}(n)$ and $X_{im}(n)$, are computed and provided to the right. The initial vector values, $x_{re}(1)$ and $x_{im}(1)$, are provided to the left. The input signals are taken from previous real and imaginary butterflies (in the case of FFT). Furthermore, the output signals are provided to next real and imaginary butterflies. In each stage new vectors are determined, in order to converge towards the vector that approximates the vector that represents the angle θ. In each vector rotation stage there is a crosswise addition or subtraction of the vector coordinates. The vector rotation depend on the sign bit, *sgn*, in each stage of the upper row of adders. That is, the sign bits determine if it should be an addition or subtraction. There are also divisions of the vector coordinates by a factor corresponding to 2^i where i's are the integers {1, 2, 3, 4, 5, 6 … n}. That is, division by {1, 2, 4, 8, 16, 32 … 2n}. This corresponds to the right shifts as discussed in the CORDIC algorithm, which are done by shifting the busses between the stages. There is thus no extra cost in hardware for the divisions.

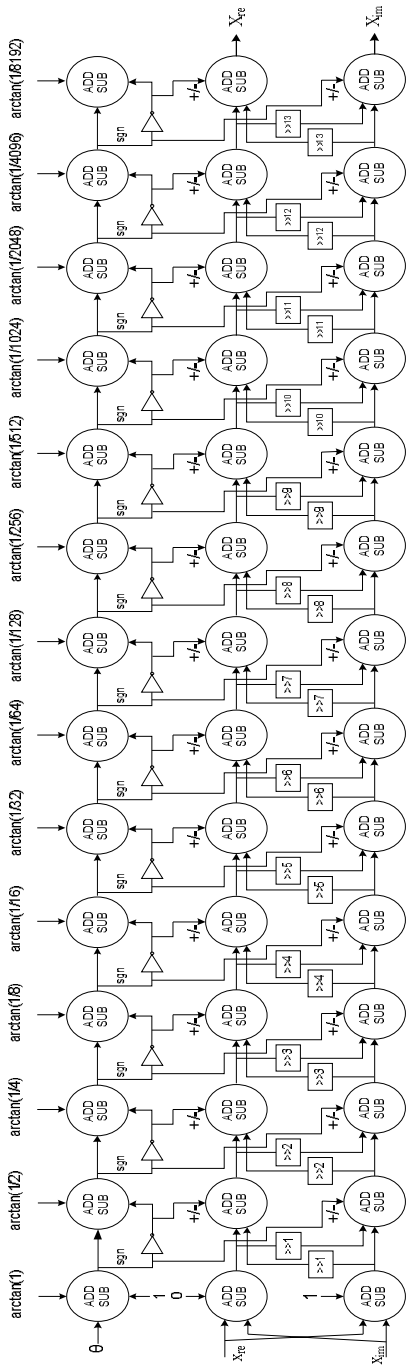

Figure 3.4 An un-rolled 14-stage CORDIC architecture

3.4 CORDIC Scaling Factor

The above CORDIC algorithm introduces the constant scaling factor (K) with the resulted rotated vector, and this factor needs to be multiplied. The property $\cos\theta = \cos(-\theta)$ brings more ease to pre-compute the scaling factor because, the direction of rotation does not matter in case of cosine function

$$K = cos\left(tan^{-1}(1)\right) \times cos\left(tan^{-1}\left(\frac{1}{2}\right)\right) \times cos\left(tan^{-1}\left(\frac{1}{4}\right)\right)\ldots.$$

$$K = 0.707 \text{ x } 0.894 \text{ x } 0.970 \text{ x } 0.992 \text{ x } 0.998 \ldots.$$

$$K \approx 0.60725$$

Now, constant scaling factors which can be further designed with shifts and add operations to eliminate the need of multiplier, as shown in Fig. 3.5.

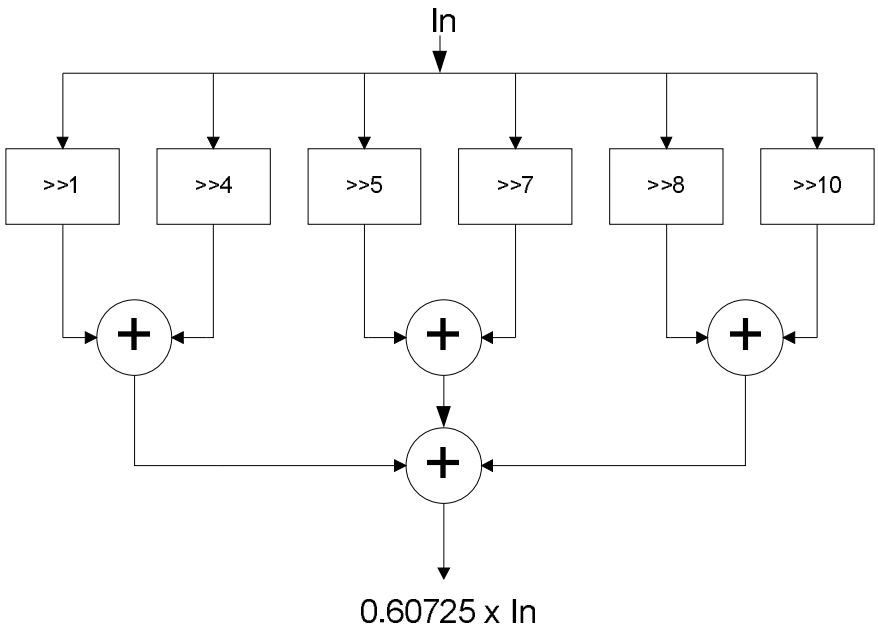

Figure 3.5 Hardware Multiplication of scaling factor.

3.5 Generalize CORDIC

Generalized CORDIC covers the equations for the linear, circular and hyperbolic systems and the (3.17) and (3.18) can be modified to accommodate these functions as well

$$x_{i+1} = x_i - \mu\, d_i y_i\, 2^{-i} \tag{3.19}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \tag{3.20}$$

$$z_{i+1} = z_i + d_i e_i \tag{3.21}$$

The parameters for generalized CORDIC algorithm are listed in Table 3.2 with the rotation type

Table 3.3.2 Parameters for Generalized CORDIC

Linear Rotation	$e_i = 2^{-i}$	μ=0
Circular Rotation	$e_i = \tan^{-1} 2^{-i}$	μ=1
Hyperbolic Rotation	$e_i = \tanh^{-1} 2^{-i}$	μ=-1

CHAPTER 4

4. Parabolic Synthesis

Parabolic synthesis methodology [6] is devised to implement approximation of different unary functions e.g. trigonometric, logarithm, square root and division functions that are extremely important in the field of astronomy, digital signal processing, image processing, robotics, navigation systems etc. High speed applications in these fields need hardware implementation where software solutions in most of the cases are not sufficient.

Parabolic synthesis methodology performs an approximation of original unary function by developing other sub-function and help functions. The architecture of the proposed methodology itself parallel in nature and take great advantage of parallelism to reduce the execution time. Only low complexity operations e.g. multiplications, shifts, additions are used that is easily implemented in the hardware.

Parabolic synthesis methodology has been devised in order to generate better results in terms of speed, area, power and precision as compared to algorithms employed previously for the approximations of unary functions. Pre-processing and post-processing are two steps besides the most important approximation part that are used to normalize and transformation respectively. Therefore, the implementation is divided into three parts, normalizing, approximation and transforming.

4.1 The Methodology

Pre-processing is the first step; normalization, approximation is the next one in which sub-function and help functions are developed and post-processing is the last step to transform the result according to the requirement.

4.1.1 Normalization

Normalization is done to facilitate the hardware implementation by restricting the numerical range. The purpose of this step is to satisfy that the values should be in the interval of $0 \leq x < 1$on the x-axis and $0 \leq x < 1$on the y-axis keeping the coordinates of starting point at (0,0) and ending point having coordinates smaller than (1,1). Furthermore the function must be strictly concave or convex during the interval.

4.1.2 Hardware Architecture Development

Low complexity operations are used when developing hardware architecture that approximates an original function. These operations include additions, multiplications and shifts, which are efficient for hardware implementation. The emergence of efficient multiplier architecture and downscaling of semiconductor

technologies has made the multiplication operation efficient in hardware implementation.

The proposed methodology is based on decomposition of basic functions. It is based on second order parabolic functions since these functions can easily be implemented with low complexity operations. The recombination process is done with multiplication.

The methodology is devised in terms of second order parabolic functions called sub-functions $s_n(x)$, these sub-functions when recombined give the approximation of the original function $f_{org}(x)$ as shown in (4.1). The accuracy of the results depends on the number of sub-functions used.

$$f_{org}(x) = s_1(x) \cdot s_2(x) \cdot \ldots \cdot s_\infty(x) \tag{4.1}$$

The first help function $f_1(x)$ is generated by dividing the original function $f_{org}(x)$ with the first sub-function $s_1(x)$. This help function will be a parabolic looking function as shown in (4.2).

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} \tag{4.2}$$

The subsequent help functions will be generated according to (4.3). The sub-function $s_n(x)$ should be chosen to be feasible for hardware implementation.

$$f_{n+1}(x) = \frac{f_n(x)}{s_{n+1}(x)} \tag{4.3}$$

4.1.3 Methodology for developing sub-functions

The sub-functions are developed by decomposing the original function $f_{org}(x)$ according to second order parabolic functions in the interval $0 \leq x < 1$ and the sub-interval within the interval. The second order parabolic function is the decomposition function and it is easily implemented in hardware with low complexity functions such as multiplication and addition.

The first sub-function

The methodology to develop the first sub-function $s_1(x)$ is by dividing the original function $f_{org}(x)$ with x as a first order approximation. There are two possibilities as a result of this division, one when $f(x) > 1$ and one when $f(x) < 1$. The first sub-function $s_1(x)$ as given by (4.4). The expression $1 + \left(c_1(1-x)\right)$ is utilized to get approximation of these functions. The first sub-function $s_1(x)$ results in a second order parabolic function as in (4.4).

$$s_1(x) = x \cdot \left(1 + \left(c_1(1-x)\right)\right) = x + \left(c_1 \cdot (x - x^2)\right) \tag{4.4}$$

In equation (4.4), the coefficient c_1 is determined according to (4.5).

$$c_1 = \lim_{x \to 0} \frac{f_{org}(x)}{x} - 1 \tag{4.5}$$

The second sub-function

The first help-function $f_1(x)$, is important in developing the second sub-function. The first help function $f_1(x)$, is calculated according to (4.2) and dividing two continuously concave or convex functions having the same starting and ending point which will give rise to another function similar to a parabolic function. The second sub-function $s_2(x)$, is developed according to second order parabolic function as shown in (4.6).

$$s_2(x) = 1 + \left(c_2 \cdot (x - x^2)\right) \tag{4.6}$$

The coefficient c_2 in (4.6), is to satisfy the condition that the quotient between the first help-function and second sub-function is equal to 1 when x is set to 0.5 in (4.7).

$$c_2 = 4 \cdot \left(f_1\left(\frac{1}{2}\right) - 1\right) \tag{4.7}$$

The second help-function $f_2(x)$ is developed in such a way that it can be divided into a pair of functions that look like parabolic functions, where the interval of first function is from $0 \leq x < 0.5$ and for second, interval is from $0.5 \leq x < 1$.

Sub-function when n>2

For help functions $f_n(x)$ when n>2, there are one or more pairs of parabolic looking functions. Each pair of parabolic looking function is divided into two parabolic help-functions in order to develop higher order sub-functions. A parabolic sub-function is developed as an approximation of the help function $f_n(x)$ in the sub-interval. Sub-sub-functions are associated with a specific sub-interval, and for this the subscript index is increased with the index m in sub-help-function in $f_{n,m}(x)$. The numbers of sub-help-functions are doubled in every order of n>1. The corresponding sub-sub-functions are developed from these help functions.

4.2 Hardware Implementation

An implementation for calculating real and imaginary values during the twiddle factor multiplication in the FFT using the proposed methodology of parabolic synthesis is described in this section.

The hardware implementation is divided into three parts; pre-processing, processing and post-processing. Fixed point 2's complement representation is used.

4.2.1 Pre-Processing

Pre-processing is done to satisfy that the value of incoming angle θ is in the interval $0 \leq \theta < 1$ and transfer angles from quadrant 2,3 and 4 into quadrant 1, to achieve this operand in radians is multiplied with $\frac{2}{\pi}$ as shown in Fig. 4.2.

Original angle=θ

Normalized angle = $\theta_f = \left(\frac{2}{\pi}\right) \cdot \theta$ = (integer part)(fractional part) (4.8)

The integer part $\theta_1\theta_0$ represents the quadrant of the original angle and it will be used for 2's complement conversion at the output and as MUX select bits. Quadrant 1, 2, 3 and 4 is represented by 0, 1, 2, and 3 respectively in binary representation as shown in Fig. 4.1.

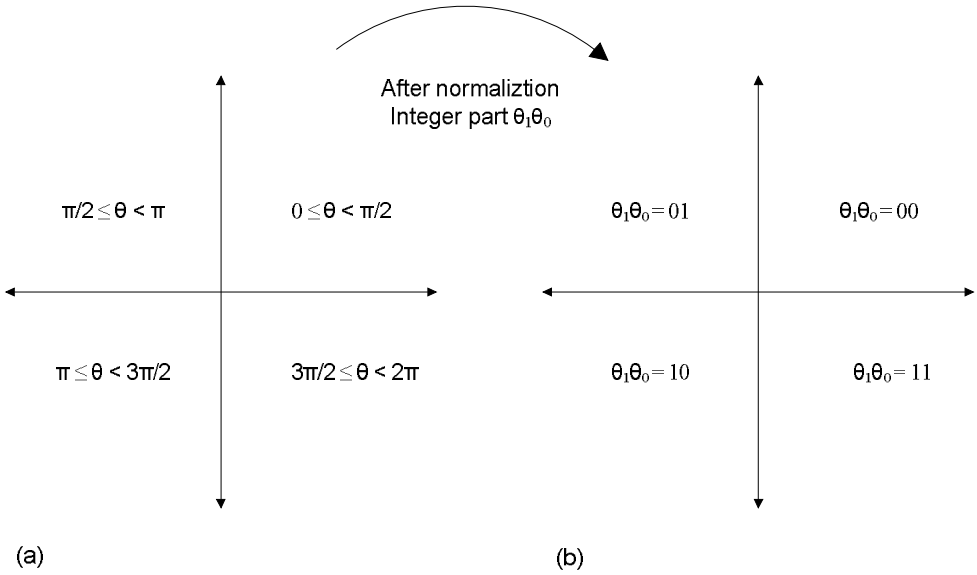

Figure 4.1 Angle transformation from quadrant 2-4 to quadrant 1

The word length of the input angle is fifteen bits; one sign bit, four integer bits and ten fractional bits to get three fractional digit precision. It is multiplied with the factor $\frac{2}{\pi}$ to get the normalized angle.

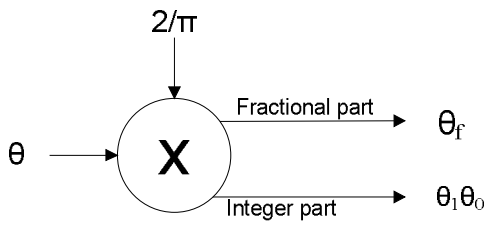

Figure 4.2 Hardware architecture for pre-processing

4.2.2 Processing

The Discrete Fourier Transform (DFT) is defined as in (4.9) and (4.10)

$$X(k) = \sum_{n=0}^{N-1} x(n)\ W_N^{kn} \tag{4.9}$$

$$W_N = e^{\frac{-2\pi j}{N}} \tag{4.10}$$

The DFT is complex valued but in hardware, the real (Re) and imaginary (Im) parts are separated in two real parts as shown in (4.11) and (4.12)

$$X_{Re}(k)=\sum_{n=0}^{N-1}\left(x_{Re}\cos\frac{2\pi kn}{N}-x_{im}\sin\frac{2\pi kn}{N}\right) \tag{4.11}$$

$$X_{Im}(k)=\sum_{n=0}^{N-1}\left(x_{Re}\sin\frac{2\pi kn}{N}+x_{im}\cos\frac{2\pi kn}{N}\right) \tag{4.12}$$

(4.11) and (4.12) can also be presented for simplicity as in (4.13) and (4.14).

$$X_{Re} = x_{Re} \times \cos(\theta) - x_{im}\sin(\theta) \tag{4.13}$$

$$X_{Im} = x_{Re} \times \sin(\theta) + x_{im}\cos(\theta) \tag{4.14}$$

The sine and cosine values must be determined by approximation. The approximated sine and cosine functions according to first and second sub-functions (4.4) and (4.6) are in (4.15) and (4.16) respectively.

$$\begin{aligned} s_1(\theta_f) &= \theta_f + c_1 \times (\theta_f - \theta_f^2) \\ s_2(\theta_f) &= 1 + c_2 \times (\theta_f - \theta_f^2) \\ sin(\theta_f) &= s_1(\theta_f) \times s_2(\theta_f) \end{aligned} \tag{4.15}$$

$$\begin{aligned} s_1(\theta_f) &= 1 - \theta_f + c_1 \times (\theta_f - \theta_f^2) \\ s_2(\theta_f) &= 1 + c_2 \times (\theta_f - \theta_f^2) \\ cos(\theta_f) &= s_1(\theta_f) \times s_2(\theta_f) \end{aligned} \tag{4.16}$$

These equations are developed according to [7] with coefficients $c_1 = 0.571$ and $c_2 = 0.401$. Only two sub-functions are used as it will be equivalent to precision CORDIC algorithm can deliver with 11 stages. The angle θ_f is the normalized fractional part of θ. It is only s_1 that differs from the sin and cosine functions. Since there is a connection between the sine and cosine terms in all 4 quadrants, the $1 - \theta_f$ term is used instead of θ_f in first sub-function in the cosine equation.

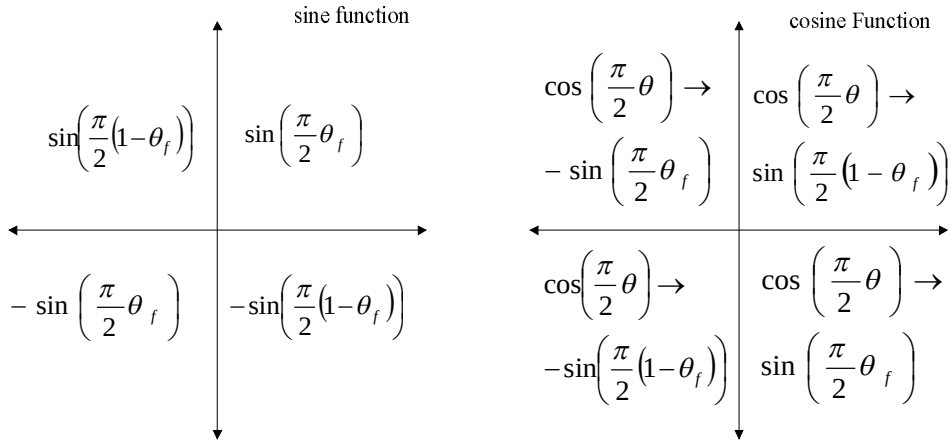

Figure 4.3 Input Transformation from quadrant 2-4 to quadrant 1

In figure (4.4), both sine and cosine functions are approximated and multiplied with previous real $x_{Re}$and x_{Im} values in the FFT to get new real and imaginary values. The normalized angle θ_f from pre-processing block is multiplied with itself to get θ_f^2 and in the next step θ_f^2 is subtracted from θ_f to get θ_f-θ_f^2. The result from θ_f-θ_f^2 is multiplied with coefficients c_1 and c_2 with simple shifts and add operation in Multiple Constant Multiplication (MCM) block as shown in Fig. 4.5.

There is one input θ_f-θ_f^2 and it will generate two results $c_1(\theta_f$-$\theta_f^2)$ and $c_2(\theta_f$-$\theta_f^2)$ with shifts and adders. It will considerably reduce the hardware, compared to two multipliers.

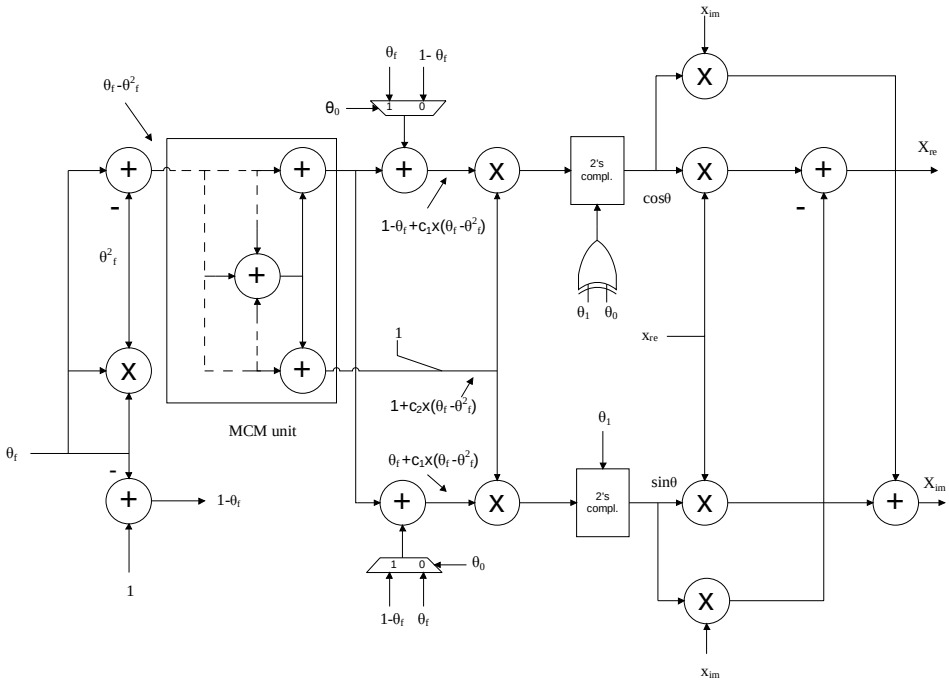

Figure 4.4 Architecture for the multiplication of complex numbers with approximated trigonometric functions (sine and cosine)

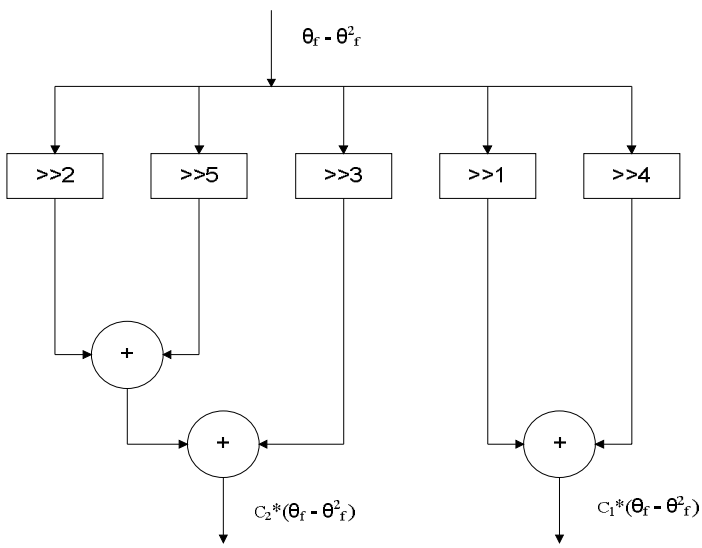

Figure 4.5 Architecture for Multiple Constant Multiplication

In case of adding "1" to $c_2(\theta_f\text{-}\theta_f^2)$ term, there is no need of an adder. Since c_2 is positive, and $c_2(\theta_f\text{-}\theta_f^2)$ always will be less than "1". The fractional part can thus be merged with "1" directly with the wiring as shown in Fig. 4.6.

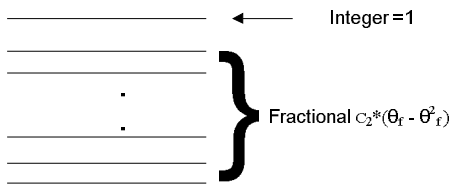

Figure 4.6 The fractional bus with an added integer "1"

This wiring will give the second sub-functions for both sine and cosine functions as shown in Fig. 4.4.

Table 4.1 shows when the input transformations are needed according to quadrant of original angle θ. The integer value θ_0 is used to select the MUXes as shown in Fig. 4.4.

Table 4.4.1 Input Transformations

	1st Quadrant	2nd Quadrant	3rd Quadrant	4th Quadrant
Sine	θ_f	1- θ_f	θ_f	1-θ_f
Cosine	1- θ_f	θ_f	1- θ_f	θ_f

The upper adder with a MUX will generate first sub-function $1 - \theta_f + c_1 \times (\theta_f - \theta_f^2)$ for cosine function and lower adder with MUX will give $\theta_f + c_1 \times (\theta_f - \theta_f^2)$ for sine function. The following multipliers will produce the approximated sine and cosine values.

Since all the computation is done in the first quadrant, the output is to be transformed back again. Table 4.2 shows the output transformations necessary at the output.

Table 4.4.2 Output Transformations

	1st Quadrant	2nd Quadrant	3rd Quadrant	4th Quadrant
Sine	+	+	-	-
Cosine	+	-	-	+

For 2's complement conversion, the architecture in Fig. 4.7 is used with half adders and XOR gates. A control signal θ_1 or θ_1 XOR $\theta_0$is used to select when the conversion is to be done according to table 4.2.

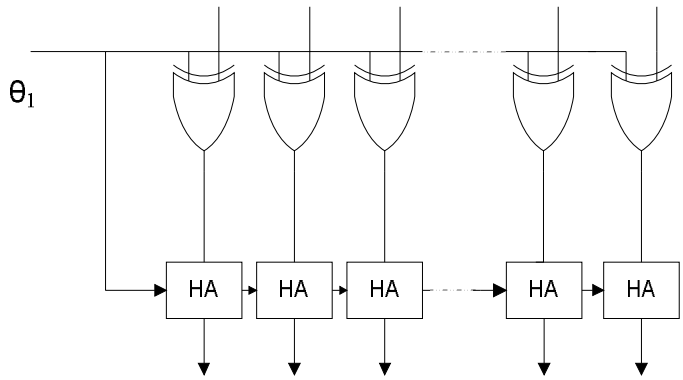

Figure 4.7 Architecture for 2's complement conversion

After post-processing, sine and cosine, the approximated values are multiplied with real x_{Re} and imaginary x_{im} parts from previous stage of the FFT. The new real X_{Re} and imaginary X_{Im} values are calculated according to (4.13) and (4.14).

CHAPTER 5

5. Results

The methodology for extracting results of area, timing and power constraints in both the designs is shown in Fig. 5.1. The HDL description of the designs is done in *VHDL*. The target design is synthesized into a gate-level netlist using an ASIC design synthesis tool, *Synopsys Design Compiler* with a standard cell library such as in our case LPLVT (Low Power Low Threshold Voltage) and LPHVT (Low Power High Threshold Voltage). The results are taken on different voltages (1V, 1.1V, 1.2V) on each library. Area and timing information for both the designs are gathered after the synthesis process. The synthesis process generates a Gate-level netlist of the design and a SDC file, which contains area and timing design constraints in the *Synopsys* Design Constraint format. *MentorGraphics ModelSim* is used to generate toggle information of the target design in VCD (Value Change Dump) file by using the Gate-level netlist and the SDC file from Design Compiler. The power analysis tool, *Synopsys PrimeTime* estimates the static and dynamic power dissipation in the target design using the design netlist and the VCD file.

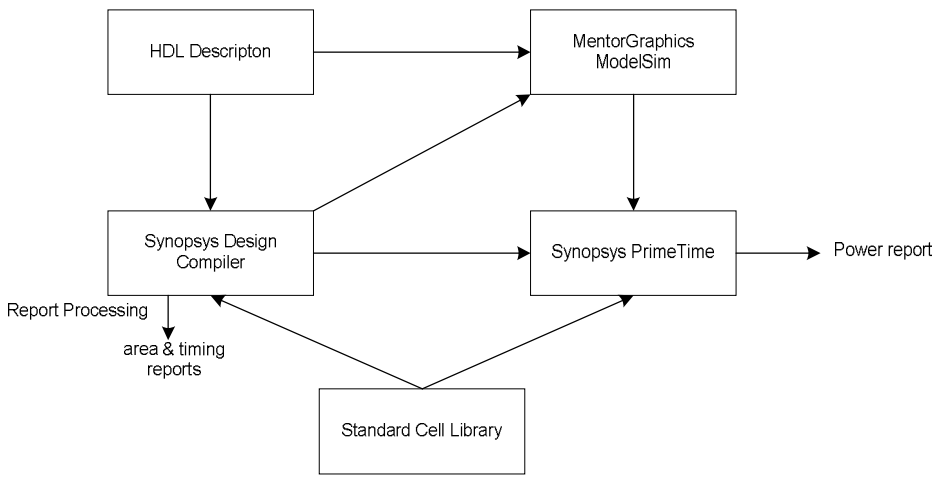

Figure 5.1 ASIC flow for results extraction

5.1 Area

The CORDIC and Parabolic Synthesis algorithm designs are synthesized in *Synopsys Design Compiler* using STMicroelectronics 65-nm LPLVT and LPHVT CMOS libraries. The synthesis is done using different supply voltages i.e., 1V, 1.1V and 1.2V. Fig. 5.2 and Fig. 5.3 show total area of the CORDIC and Parabolic Synthesis algorithm designs respectively, in LPLVT and LPHVT CMOS libraries with different supply voltages. It is obvious that parabolic synthesis design is more area efficient than 14 stage unrolled CORDIC design.

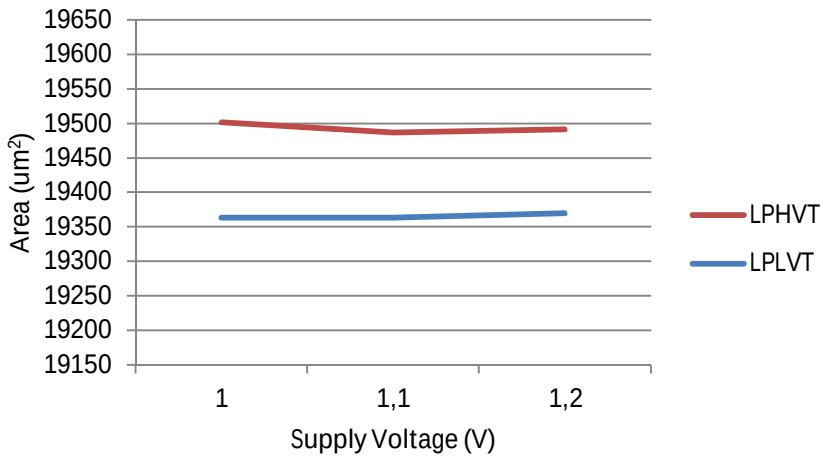

Figure 5.2 Total Area analysis of CORDIC design in $LPLV_t$ and $LPHV_t$

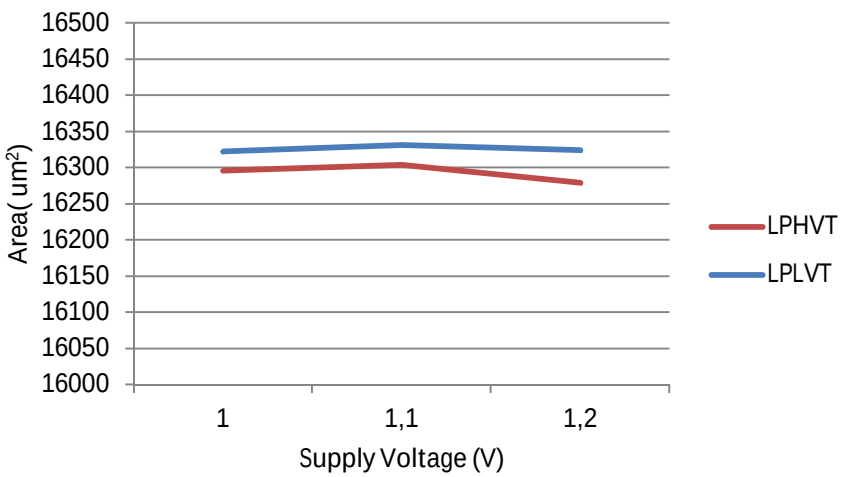

Figure 5.3 Total Area analysis of Parabolic Synthesis design in $LPLV_t$ and $LPHV_t$

Table 5.1 Total area of CORDIC and Parabilic Synthesis in $LPLV_t$ and $LPHV_t$

Architecture	CORDIC			Parabolic Synthesis		
Supply Voltage (V)	1V	1.1V	1.2V	1V	1.1V	1.2V
Area (um^2) LPLVT	19303	19363	19370	16322	16331	16324
Area (um^2) LPHVT	19502	19487	19491	16296	16304	16279

5.2 Timing

Fig. 5.4 and Fig. 5.5 show the critical path propagation delay timing for the CORDIC and Parabolic Synthesis designs respectively. It can be seen that the propagation delay is less in parabolic synthesis compared to the 14-stage un-rolled CORDIC design. The reason for that is the shorter critical path in the Parabolic Synthesis design due to the high degree of parallelism. There is also a difference in delay in two different libraries LPLVT and LPHVT. The delay is smaller in LPLVT, that is, it is almost half of that in LPHVT.

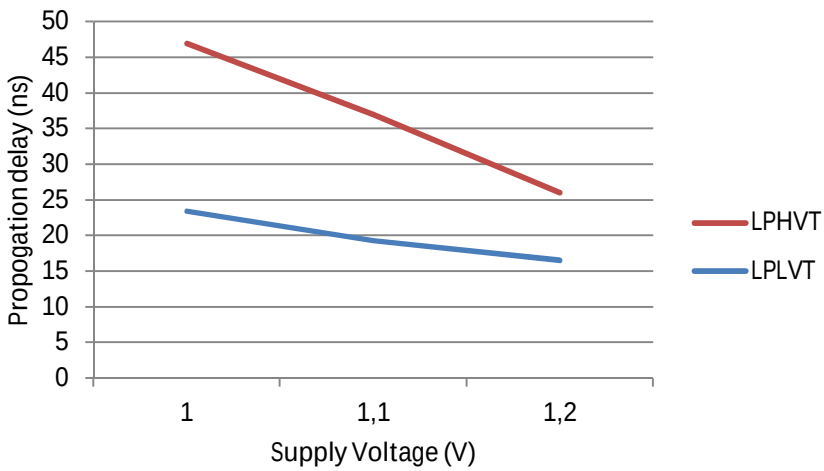

Figure 5.4 Propagation delay of CORDIC design in $LPLV_t$ and $LPHV_t$

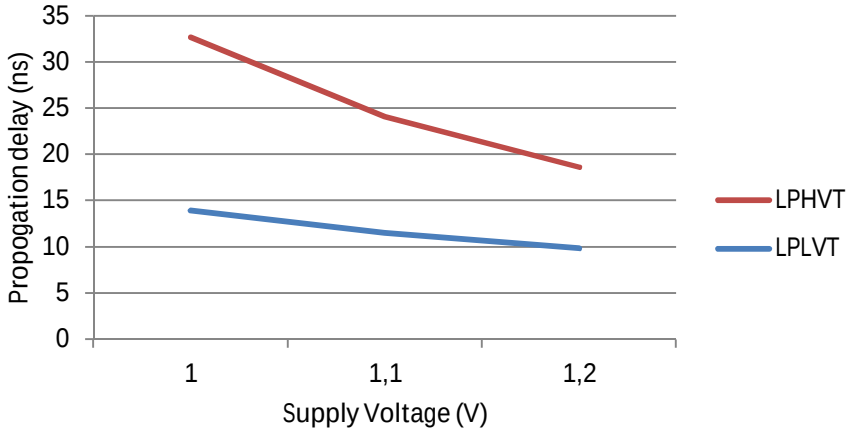

Figure 5.5 Propagation delay of Parabolic Synthesis design in $LPLV_t$ and $LPHV_t$

Table 5.2 Total Propagation Delay of CORDIC and Parabilic Synthesis in $LPLV_t$ and $LPHV_t$

Architecture	CORDIC			Parabolic Synthesis		
Voltage Supply (V)	1V	1.1V	1.2V	1V	1.1V	1.2V
Propagation delay (ns) LPLVT	23.4	19.13	16.25	13.95	11.5	9.85
Propagation delay (ns) LPHVT	47	37	26	32.7	24.1	18.66

5.3 Static Power

Fig. 5.6 and Fig. 5.7 show the static power dissipation in the CORDIC and Parabolic Synthesis designs respectively, using LPLVT and LPHVT CMOS libraries. The static power dissipation is almost zero with respect to LPLVT transistor technology. The static power dissipation in the Parabolic Synthesis is less than in the CORDIC algorithm.

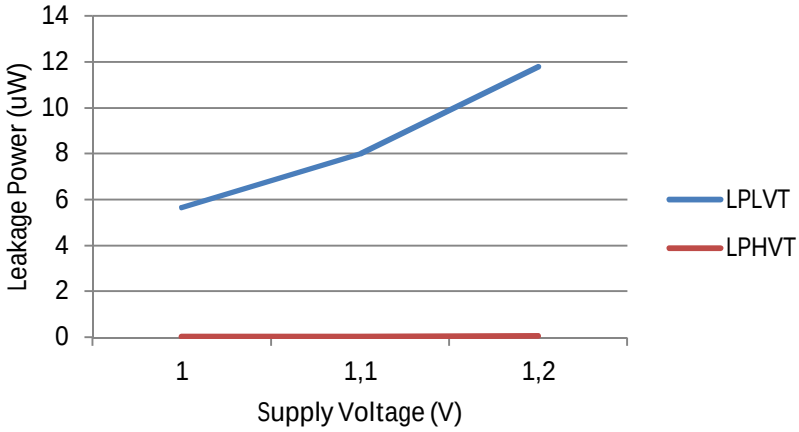

Figure 5.6 Static Power (Leakage) of CORDIC design in LPLVt_t and LPHV$_t$

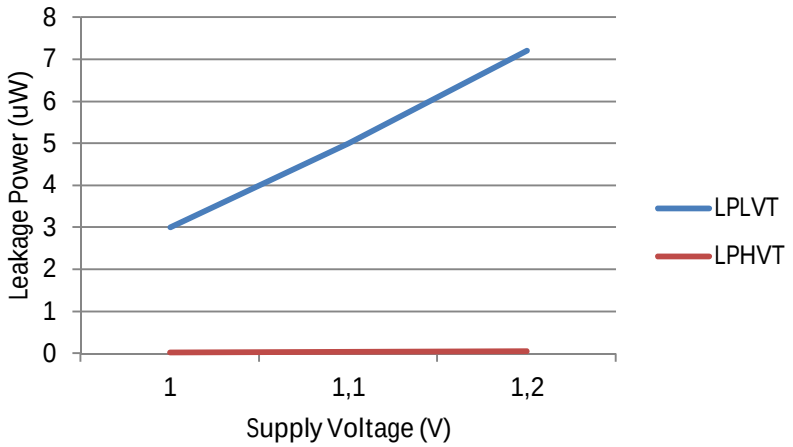

Figure 5.7 Static Power (Leakage) of Parabolic Synthesis design in LPLV$_t$ and LPHV$_t$

Table 5.3 Static Power (Leakage) of CORDIC and Parabolic Synthesis design in $LPLV_t$ and $LPHV_t$

Architecture	CORDIC			Parabolic Synthesis		
Supply Voltage (V)	1	1.1	1.2	1	1.1	1.2
Static Power (uW) LPLVT	5.67	8	11.8	3	5	7.21
Static Power (uW) LPLVT	0.03	0.043	0.001	0.021	0.031	0.045

5.4 Dynamic Power

Fig. 5.8 and Fig. 5.9 describes the total dynamic power (both cell internal power and Net Switching power) in the unrolled-CORDIC and Parabolic Synthesis designs respectively, using LPLVT and LPHVT design libraries. The architectures are tested on maximum frequencies.

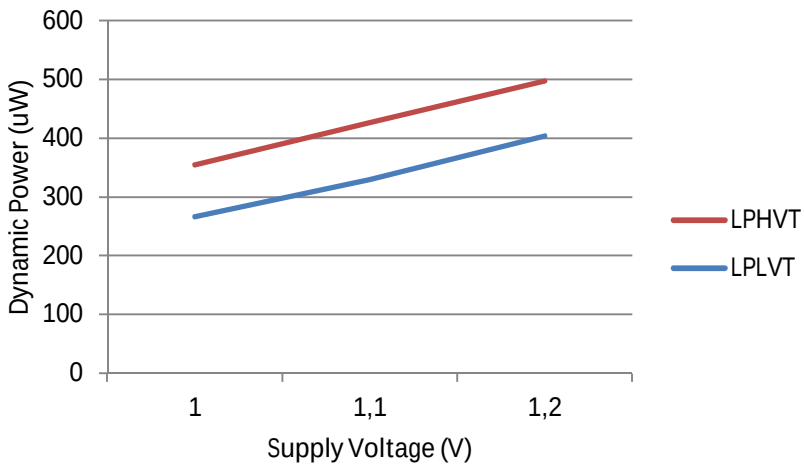

Figure 5.8 Total Dynamic Power of CORDIC design in $LPLV_t$ and $LPHV_t$

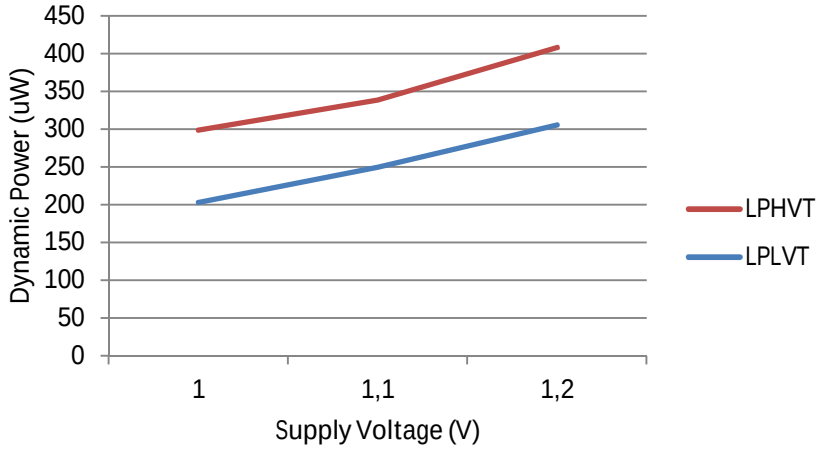

Figure 5.9 Total Dynamic Power of Parabolic Synthesis design in $LPLV_t$ and $LPHV_t$

Table 5.4 Dynamic Power of CORDIC and Parabolic Synthesis design in $LPLV_t$ and $LPHV_t$

Architecture	CORDIC			Parabolic Synthesis		
Voltage Supply (V)	1V	1.1V	1.2V	1V	1.1V	1.2V
Dynamic Power (uW) LPLVT	267	330	404	203	250	306
Dynamic Power (uW) LPHVT	355	427	498	299	339	409

5.5 Energy

Fig. 5.8 and Fig 5.9 for dynamic power comparison gives us total dynamic power consumed by Parabolic Synthesis and Cordic, but due to different design approaches, both architectures have different frequencies, as shown in Fig. 5.4 and fig. 5.5. Total energy consumed by both architectures at their maximum frequencies is shown below in Fig. 5.10 and Fig. 5.11. It is clearly seen that Parabolic synthesis is about 50% more energy efficient than Cordic.

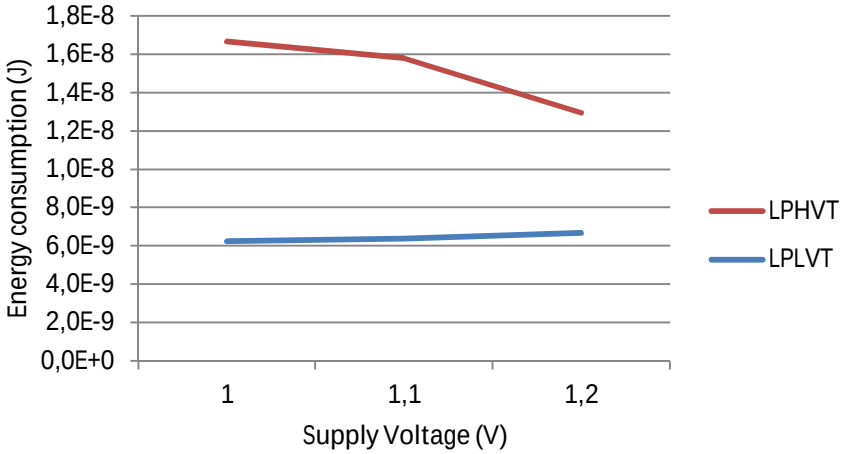

Figure 5.10 Energy Consumption in CORDIC design in $LPLV_t$ and $LPHV_t$

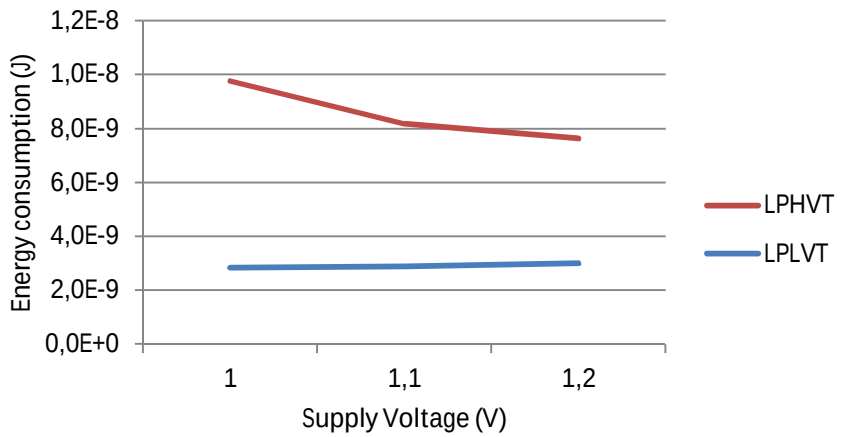

Figure 5.11 Energy Consumption in Parabolic Synthesis design in $LPLV_t$ and $LPHV_t$

Table 5.5 Energy Consumption in CORDIC and Parabolic Synthesis design

Architecture	CORDIC			Parabolic Synthesis		
Voltage Supply (V)	1	1.1	1.2	1	1.1	1.2
Energy consumption (nJ) LPLVT	6.24	6.37	6.67	2.83	2.87	3.01
Energy consumption (nJ) LPHVT	16.68	15.79	12.94	9.77	8.16	7.63

Table 5.6.1 required memories, registers and multipliers for FFT points

No of FFT points	No of register/Rom cells required to store the Twiddling factors (Real + Imaginary)	Real Multipliers
16	16	12
32	32	16
64	64	20
128	128	24
256	256	28
512	512	32
1024	1024	36
2028	2048	40

CHAPTER 6

6. Conclusions

A comparison of two different algorithms, un-rolled CORDIC and Parabolic Synthesis, has been made for area, speed and energy. These algorithms are utilized to approximate real and imaginary values in an FFT. Comparisons have been done on different transistor technologies at different voltages. The results show that the parabolic synthesis algorithm is better in energy consumption, speed and area. The area utilization in the Parabolic Synthesis is 16% less than in the un-rolled CORDIC for the specific precision and energy consumption, on the other hand, is 50% less than the CORDIC. In the same manner, the timing of the parabolic synthesis is about 1.4 times better than the un-rolled CORDIC algorithm. It is also concluded that, when using a LPLVT transistor technology, the static power is higher but less dynamic power with comparison of LPHVT transistor technology. Since the cell size in both the technologies is almost same, the area utilization comes to be very close to each other. Speed is better when using LPLVT transistor technology as expected due to less parasitic effects.

An investigation of the rotations, using twiddle factor multiplications, is not included here. However, a rough estimate on how the number of register/ROM cells, to store the twiddle factors, increase with the number of FFT points, can be done as shown in Table 5.1

It can be noted that the number of twiddle factor cells are increasing exponentially. For small FFTs, registers can be beneficial but for large FFTs, A ROM will be preferred. But the size of memory increases exponentially with increase in FFT points, whereas on other hand the Parabolic synthesis and the CORDIC algorithms computes rotations in the real time, hence, there is no need to store twiddle factors.

CHAPTER 7

7. Future Work

The comparison of Un-rolled CORDIC and Parabolic Synthesis algorithms for twiddle factors can also be carried out with multiplier based FFT architecture which needs ROM to store larger number of twiddle factors. The multiplier based FFT architecture is area efficient for the low number of FFT points. However, when the number of points is increasing, it utilizes an exponentially larger chip area. Parabolic Synthesis algorithm can also be improved with respect to wordlength optimization between the architecture.

References

[1] Ray Andraka, "A survey of CORDIC algorithms in FPGA based computers", Andraka Consulting Group, Inc. North Kingstown, USA.

[2] Y. H Hu and S. Naganathan, "An angle recording method for CORDIC algorithm implementation", *IEEE Transaction on Computers*, vol. 42, issue 1, pp. 99-102, 1993.

[3] Axler, Sheldon "*Linear Algebra Done Right, 2e"*, Springer Press, ISBN 978-0-387-98259-5.

[4] Jack E. Volder, "The CORDIC Trigonometric Computing Technique", *IRE transactions on Electronic Computers*, vol. EC-8, issue 3, pp. 330-334, 1959.

[5] E. Hertz, P. Nilsson, "A Methodology for Parabolic Synthesis of Unary Functions for Hardware Implementation," *Proc. of the 2nd International Conference on Signals, Circuits and Systems*, ISBN-13: 978-1-4244-2628-7, pp 1-6, Hammamet, Tunisia, Nov. 2008.

[6] Erik Hertz and Peter Nilsson, "Parabolic Synthesis Methodology", In the proceedings of the 2010 GigaHertz Symposium, pp 35, March 9-10, 2010, Lund, Sweden.

[7] E. Hertz and P. Nilsson, "Parabolic Synthesis Methodology Implemented on the Sine Function", in Proceedings of the *2009 IEEE International Symposium on Circuits and Systems*, pp 253-256,Taipei, May 24-27, 2009.

[8] J. W. Cooley and J. W. Tukey, An algorithm for machine calculation of complex Fourier series, Math. Comp., 19 (1965), 297-301.

[9] Kamisetty Rao, Do Nyeon Kim, Jae Jeong Hwang, *Fast Fourier Transform: Algorithms and Application*, Springer Press, ISBN 976-1-40206628-3.

[10] Alan V. Oppenheim, Ronald W. Schafer, *Discrete-time signal processing*, Prentice Hall, ISBN 978-0-13198842-2, the University of Michigan, Michigan, USA.

List of Figures

List of Tables