

BIT SERIAL CORDIC DDFS



By
Aqib Perwaiz
(2004-NUST-MS PhD-Com 07)

Advisor
DR. SHOAB AHMED KHAN

This thesis is submitted in the partial fulfillment of requirement
for the degree of

MASTERS IN COMPUTER ENGINEERING
Department of Computer Engineering

College of Electrical and Mechanical Engineering

National University of Sciences and Technology

Rawalpindi, Pakistan

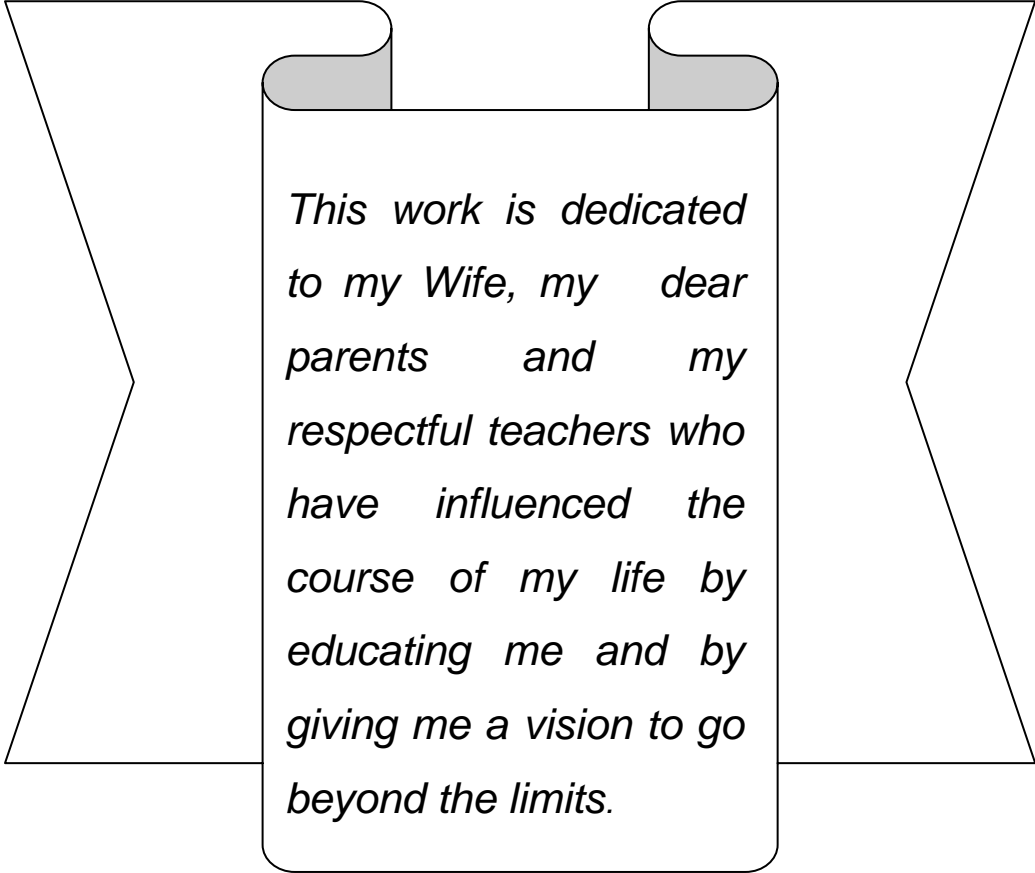
2009

Table of contents

Dedication	i
Acknowledgement	ii
Abstract	iii
List of acronyms	iv
List of tables	v
List of figures	vi
Chapter 1. Introduction	
1.1 Preamble.....	1
1.2 Bit Serial CORDIC DDFS.....	3
1.3 Thesis Objective.....	4
1.4 Methodology.....	4
1.5 Thesis Organization... ..	5
Chapter 2. Literature Review	
2.1 Background and History of CORDIC Algorithm.....	6
2.2 Basic CORDIC Iteration.....	12
Chapter 3. Computation of Sine and Cosine	
3.1 CORDIC Hardware.....	18
3.2 The CORDIC Algorithm for computing Sine and Cosine...	19
3.3 Implementation Of Various CORDIC Architectures.....	22
3.3.1 A Bit Parallel Iterative CORDIC.....	22
3.3.2 A Bit Parallel Unrolled CORDIC.....	24
Chapter 4. Bit Serial CORDIC	
4.1 Bit serial iterative CORDIC.....	27
4.2 Proposed Bit serial modified CORDIC.....	29
4.3 Flow of MATLAB Code	34

4.4 Proposed Bit serial Modified CORDIC architecture.....	35
4.5 Algorithm flow of proposed Architecture With Example...	37
Chapter. 5 Results and Block Description	
5.1 MODELSIM Simulation Results.....	41
5.2 Block Description of Verilog Code	42
5.2.1 Block CORDIC Main Module	42
5.2.2 Internal RTL Schematic of Main Module	43
5.2.3 Internal RTL Schematic of ROM	44
5.3 Error Analysis.....	47
.....5.4 Discussion.....	48
Chapter. 6 Conclusion	
6.1 Overview.....	49
6.2 Proposed Approach.....	50
6.3 Future Work.....	50
References.....	51

DEDICATION



*This work is dedicated
to my Wife, my dear
parents and my
respectful teachers who
have influenced the
course of my life by
educating me and by
giving me a vision to go
beyond the limits.*

Acknowledgement

I am grateful to Almighty ALLAH who has showered me with His valuable blessings throughout my life, given me strength and spirit to complete this research.

I was truly blessed by beings surrounded by extremely intelligent world class faculty, I have my all praises for my head of department Dr. Younus Javed whos dedication toward his profession was a source of motivation for me. I am indebted to my project advisor Dr. Shoab Ahmed khan, College of E&ME, NUST for giving me a guide line and helping me out even at odd hours to complete this project efficiently.

I have lots of appreciation for my parents and family members for their overwhelming support and prayers throughout my project research.

I would also like to thank all the committee members, who have always been a source of inspiration, for their cooperation and healthy academic environment throughout my studies at College OF E&ME Rawalpindi.

Abstract

CORDIC is an acronym for Coordinate Rotation Digital Computer. It is a class of shift adds algorithms for rotating vectors in a plane, which is usually used for the calculation of trigonometric functions, multiplication, division and conversion between binary and mixed radix number systems of DSP applications, such as Fourier Transform. The Jack E. Volder's CORDIC algorithm is derived from the general equations for vector rotation. The CORDIC algorithm has become a widely used approach to elementary function evaluation when the silicon area is a primary constraint. The implementation of CORDIC algorithm requires less complex hardware than the conventional method.

In digital communication, the straightforward evaluation of the cited functions is important, numerous matrix based adaptive signal processing algorithms require the solution of systems of linear equations, the computation of eigen values, eigenvectors or singular values. All these tasks can be efficiently implemented using processing elements performing vector rotations. The (CORDIC) offers the opportunity to calculate all the desired functions in a rather simple and elegant way. Due to the simplicity of the involved operations the CORDIC algorithm is very well suited for VLSI implementation. Verilog coding and simulation of bit serial CORDIC algorithm for sine and cosine, the comparison of resultant implementations and the specifics of the FPGA implementation has been discussed.

In this thesis, the CORDIC algorithm has been implemented in XILINX Spartan 3E FPGA kit using Verilog and is found to be accurate. It also contains bit serial implementation of CORDIC algorithm on the same FPGA kit which is actually the problem statement. Due to the high speed, low cost and greater flexibility offered by FPGAs over DSP processors the FPGA based computing is becoming the heart of all digital signal processing systems of modern era. Moreover the generation of test bench by Xilinx ISE 9.2i verifies the results.

List of Acronyms

ASICs	Application-Specific Integrated Circuits
CLBs	Configurable Logic Blocks
CORDIC	Coordinate Rotation Digital Computer
DDFS	Direct Digital Frequency Synthesizer
FPGA	Field Programmable Gate Array
LUT	Look Up Table
RAM	Random Access Memory
ROM	Read Only Memory
RTL	Register Transfer Level
SRAM	Static RAM
SVD	Singular Value Deposition
ULP	Unit in the Last Place
VHSIC	Very High Speed Integrated Circuit
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integration

List of Tables

Table Number	Caption	Page Number
2.1	8 bit CORDIC hardware.....	10
2.2	$\alpha_i = \arctan .2^{-i}$	14
2.3	Example of 30 Degree calculation.....	15
3.1	Performance and CLB usage in an XC4010E.....	25
4.1	Initialization of X and Y.....	30
4.2	Calculation of B(n) and R(n).....	37
4.3	Bit Serial CORDIC calculation.....	39
5.1	Advanced HDL synthesis Report for sine and cosine.....	47

List of Figures

Figure Number	Caption	Page Number
2.1	Angle rotation.....	8
2.2	Balance having θ at one side and Φ on the other side....	10
2.3	Inclined balance due to difference in weights of two.... side	11
2.4	First few iterations for 30 calculation.....	15
3.1	CORDIC hardware.....	18
3.2	Angular rotation.....	20
3.3	Iterative rotation.....	21
3.4	Iterative CORDIC.....	23
3.5	Unrolled CORDIC.....	26
4.1	Bit serial CORDIC.....	28
4.2a	Segment of MATLAB code.....	33
4.2b	Flow Diagram	34
4.3	Bit Serial Modified CORDIC Architecture	36
5.1	Sine and cosine values generated for an input of 30(binary value)	41
5.2	Sine and cosine values generated for an input of 30(integer value)	42
5.3-5.6	Top level RTL schematic of bit serial modified..... CORDIC	43-45
5.7	Synthesis report.....	46
5.8	Error Analysis.....	47

Chapter 1

Introduction

The **CO**ordinate **R**otation **DI**gital Computer (CORDIC) algorithm was first introduced by Jack E. Volder [1] in the year 1959 for the computation of Trigonometric functions such as Multiplication, Division, Data type conversion, Square Root and Logarithms. It is a highly efficient, low-complexity, and robust technique to compute the elementary functions. The basic Algorithm structure is actually a set of equations which iteratively converge to give the desired results. The CORDIC algorithm has found its way in various applications such as pocket calculators, numerical co-processors, to high performance radar signal processing, supersonic bomber aircraft with a digital counterpart.

Bekooij, Huisken's et.al have also explored different applications of CORDIC in the computation of the sine and cosine and its effects on the numerical accuracy and hardware size by changing the number of iterations.

1.1 Preamble

CORIDC calculates the value of trigonometric functions like sine, cosine, hyperbolic functions magnitude and phase (arctangent) to any desired precision. The CORDIC algorithm does not use calculus based methods such as polynomial or rational function approximation rather it gives approximate function values on all popular graphic calculators such as HP-48G since the hardware restriction of calculators require that the elementary functions should be computed using only additions, subtractions, digit shifts, comparisons and stored constants.

Recently CORDIC algorithm is used in Neural Network VLSI design [4], high performance vector rotation DSP applications [5], advanced circuit design, optimized low power design. CORDIC algorithm revolves around the idea of "rotating" the phase of a complex number, by multiplying it by a succession of constant values. However, the "multiplication" can all be powers of 2 so that they can be done using just shifts and adds in binary arithmetic and no such actual "multiplier" is needed. Thus it quite simple and

does not require complex hardware structure as in the case of multiplier. Earlier methods used are Table look up method [1], Polynomial approximation method [4]etc. for evaluation of trigonometric functions. The problem with table look up CORDIC is the huge sized ROM CORDIC and the disadvantage of Polynomial approximation is as the polynomial order increases the system of equations become ill contained. CORDIC is hardware efficient algorithm with no requirement of multiplier as in case of microcontroller. The drawback in CORDIC is that after completion of each iteration, there is a gain which is added to the magnitude of resulting vector which can easily be removed by multiplying the resulting magnitude with the inverse of the gain. There are two modes in CORDIC algorithm for calculation of trigonometric functions are rotation mode and vectoring mode, both of these methods initialize the angle accumulator with the desired angle value as a step one. The rotation mode, determines the right sequence as the angle accumulator approaches zero while the Vectoring mode minimizes the y component of the input vector. CORDIC is generally faster than other approaches when a hardware multiplier is unavailable (e.g. in a microcontroller), or when the number of gates required to implement are to be minimized (e.g. in an FPGA). On the other hand, when a hardware multiplier is available (e.g. in a DSP microprocessor), table-lookup methods and power series are generally faster than CORDIC. Since it is an iterative method it has the advantage over the other methods of being able to get better accuracy by doing more iteration. Where as the Taylor approximation and the Polynomial interpolation methods need to be re derived to get better results. These properties, in addition to getting a very accurate approximation is perhaps the reason why CORDIC is used in many scientific calculators today. Due to the simplicity of the involved operations the CORDIC algorithm is very well suited for VLSI implementation. However, the CORDIC iteration is not a perfect rotation which would involve multiplications with sine and cosine. Various CORDIC architectures like bit parallel iterative CORDIC, a bit parallel unrolled CORDIC, a bit-serial iterative CORDIC and bit serial modified CORDIC are discussed in this thesis and it can be seen that CORDIC is a feasible way to approximate cosine and sine. CORDIC is useful in designing computing devices. As

CORDIC was originally designed for hardware applications, there are features that make CORDIC an excellent choice for small computing devices.

1.2 Bit Serial CORDIC DDFS

Problems which involve repeated evaluation of a fixed set of nonlinear, algebraic equations appear frequently in scientific and engineering applications. Examples of such problems can be found in the robotics, engineering graphics, and signal processing areas. Evaluating complicated equation sets can be very time consuming in software, even when co-processors are used, especially when these equations contain a large number of nonlinear and transcendental functions as well as many multiplication and division operations. Both, the unrolled and the iterative bit-parallel designs, show disadvantages in terms of complexity and path delays going along with the large number of cross connections between single stages. To reduce this complexity we can change the design into a completely bit-serial iterative architecture. Bit-serial means only one bit is processed at a time and hence the cross connections become one bit-wide data paths. Clearly, the throughput becomes a function of

$$\frac{\text{Clock rate}}{\text{Number of iterations} \times \text{word width}}$$

In spite of bit serial the output rate can be almost as high as achieved with the unrolled design. The reason is the structural simplicity of a bit-serial design and the correspondingly high clock rate achievable.

Direct Digital Frequency Synthesizer (DDFS) is a digital technique for the generation of sine and cosine. In this thesis the idea is to directly generate the sine/cosine value using the angle rotation scheme as per the working of CORDIC algorithm with a little difference i.e the data is serial and the working of basic CORDIC has been modified to achieve the desired results.

1.3 Thesis objective

Based on the above discussion the thesis has following objectives:

- To study and implement CORDIC algorithm using VHDL/Verilog programming code.
- To develop efficient bit serial CORDIC which can be used as Direct Digital Frequency synthesizer(DDFS) having a small sized Read only memory(ROM) in VHDL/Verilog code and verify the results in ModelSim and MATLAB.
- To implement proposed Bit serial CORDIC algorithm on XILINX Vertex II kit.

1.4 Methodology

In this thesis, VHDL / Verilog and MATLAB programming has been used to implement CORDIC algorithm (to calculate Sine and Cosine value for a given angle). Further XILINX SPARTAN 3E kit is used for FPGA implementation of the generated HDL code.

Programming tools used for the implementations are:

- Operating system WINDOWS XP
- ModelSim SE PLUS 5.5c
- MATLAB
- XILINX 9.2i
- FPGA kit SPARTAN 3E

1.5 Thesis Organization

Chapter 2 discusses basics of CORDIC algorithm, how it came into picture, its basic equations, different implementation styles, CORDIC iteration and how it works.

Chapter 3 discusses about the calculation of sine-cosine using CORDIC algorithm, different architectures to perform CORDIC iteration and their block diagram. Chapter 4 discusses the Bit serial implementation of CORDIC algorithm for calculating sine and cosine, it also discusses the proposed bit serial architecture which is modified version. Chapter 5 contains the results of simulation using ModelSim, Matlab and XILINX. The thesis concludes in chapter 6 which also discusses future scope of work.

Chapter 2

Literature Review

In 1959, Jack E. Volder [1] described the Coordinate Rotation Digital Computer or (CORDIC) for the calculation of trigonometric functions, multiplication, division and conversion between binary and mixed radix number systems. The CORDIC algorithm provides an iterative method of performing vector rotations by arbitrary angles using only shift and add.

2.1 Background and History of CORDIC algorithm

CORDIC algorithm has found its way in many applications. The CORDIC was introduced in 1956 by Jack Volder as a highly efficient, simplicity and robust technique to compute the elementary functions. It is initially intended for navigation technology, the CORDIC algorithm has found its way in a wide range of applications, ranging from pocket calculators, numerical co-processors, to high performance radar signal processing. After invention CORDIC worked as the replacement for the analog navigation computers aboard the B-58 supersonic bomber aircraft with a digital counterpart. The CORDIC airborne navigational computer built for this purpose, outperformed conventional contemporary computers by a factor of 7, mainly due to the revolutionary development of the CORDIC algorithm. Further Steve Walther [6] continues work on CORDIC with the application of the CORDIC algorithm in the Hewlett-Packard calculators, such as the HP-9100 and the famous HP-35 in year 1972, the HP-41C in year 1980. Today's fast rotation techniques are closely related to CORDIC, to perform orthonormal rotation at a very low cost. Although fast rotations exist for certain angles only, they are sufficiently versatile, and have already been widely applied in signal processing.

Hekstra found a large range of known, and previously unknown, fast rotation methods. An overall evaluation of the methods exposes the trade-offs that exist between the angle of rotation, the accuracy in scaling and the cost of rotation. Van der Kolk, Deprettere, and Lee [7] formalized the problem of (approximate) vectoring for fast rotations in year 2000. They treated the fast and efficient selection of the appropriate fast rotation, and showed

the advantage to be gained when applied to Enhanced Versatile Disc (EVD). The selection technique works equally well for redundant arithmetic and floating-point computations. Antelo, Lang, and Bruguera[8] considers going to a higher radix than the radix-2 for the classical algorithm, so that less iterations are required. The choice of a higher radix implies that the scaling factor is no longer constant. The authors propose an on-line calculation of the algorithm of the scale factor and subsequent compensation. Hsiao, Lau, and Delosme [9] considered multi-dimensional variants of CORDIC, such as the 4-D (dimension) householder CORDIC transform, and their application to singular value decomposition (SVD). Rather than building a multi-dimensional transform out of a sequence of 2-D (dimension) CORDIC operations, they proposed to work with multi-dimensional micro-rotations, immediately at the iteration level. Their method is evaluated and benchmarked against solutions by others. Kwak, Choi, and Swartzlander [10] aimed to overcome the critical path in the iteration through sign prediction and addition.

They proposed to overlap the sign prediction with the addition, by computing the results for both outcomes of the sign, and to select the proper one at the very end of the iteration. Novel in their approach is to combine the adder logic for the computation of both results. Volder's algorithm is derived from the general equations for a vector rotation. If a vector V with coordinates (x, y) is rotated through an angle ϕ then a new vector V' can be obtained with coordinates (x', y') where x' and y' can be obtained using x, y and ϕ by the following method as shown in Figure 2.1.

Mathematically it can be written as

$$x_2 = x_1 * \cos(\phi) - y_1 * \sin(\phi) \quad (2.1)$$

$$y_2 = x_1 * \sin(\phi) + y_1 * \cos(\phi) \quad (2.2)$$

$$\text{Where } \phi = \phi_2 - \phi_1 \quad (2.3)$$

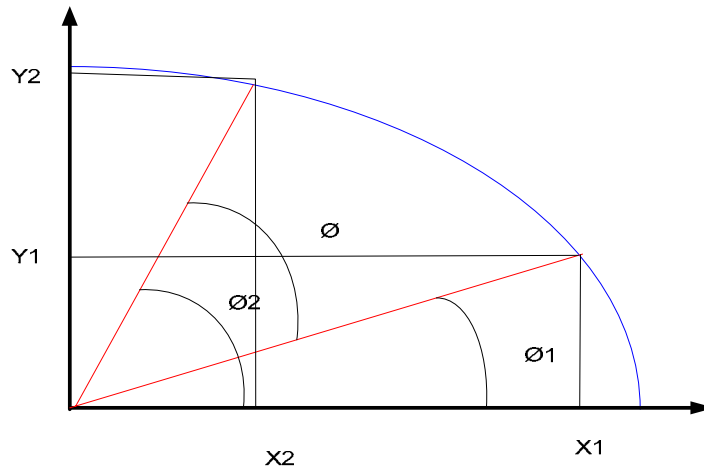


Fig. 2.1 Angle Rotation

The complete angle with required precision is executed in several iterations. In matrix form we can write the above Eq. as:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \quad (2.4)$$

The general representation of above Eq.2.4 with taking $\cos\phi$ as common from Eq. 2.4

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \cos \phi_i \begin{pmatrix} 1 & -\tan \phi_i \\ \tan \phi_i & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad (2.5)$$

$\tan(\phi) = 2^{-i}$ where $\tan(\phi)$ is a hardware shift

$$\phi = \arctan(2^{-i})$$

The multiplication by the tangent term can be avoided if the rotation angles and therefore $\tan(\phi)$ are restricted so that $\tan(\phi) = 2^{-i}$, in digital hardware this denotes a simple shift operation, if those rotations are performed iteratively and in both directions, then every value of $\tan(\phi)$ is representable. With $\phi = \arctan(2^{-i})$, the cosine term could also be simplified and since $\cos(\phi) = \cos(-\phi)$, it is constant for a fixed number of iterations. This iterative rotation can now be expressed as:

$$x_{i+1} = k_i[x_i - y_i.d_i.2^{-i}] \tag{2.6}$$

$$y_{i+1} = k_i[y_i + x_i.d_i.2^{-i}] \tag{2.7}$$

Where, i denote the number of rotation required to reach the required angle of the required vector, $k_i = \cos(\arctan(2^{-1}))$ and $d_i = \pm 1$, the product of the k_i represents the so-called K factor [6]:

$$k = \prod_{i=0}^{n-1} k_i \tag{2.8}$$

Where d_i is determined by the direction of necessary correction and

$$\prod_{i=0}^{n-1} k_i = \cos \phi_0 \cos \phi_1 \cos \phi_2 \cos \phi_3 \dots \cos \phi_{n-1} \text{ (}\phi \text{ is the angle of rotation).}$$

The above rotations requires, adding and subtracting of the different ϕ .

k_i is the gain and its value changes as the number of iterations increase. The value of k_i is approximated for 8 bit CORDIC as follows:

$$k_i = \prod_{i=0}^7 \cos \phi_i = \cos \phi_0 \cos \phi_1 \cos \phi_2 \cos \phi_3 \dots \cos \phi_7 \tag{2.9}$$

$$\cos 45^\circ . \cos 26.565^\circ \dots \cos 0.4469^\circ = 0.6073 \tag{2.10}$$

The table 2.1 shows the 8 bit CORDIC hardware. The first column is the index from 0-7, column 2 is the inverse value of 2 to the power of index in column 1, column 3 is the inverse tan of the column 2, these all are interrelated and this is basically the working of the CORDIC algorithm.

Table 2.1: 8 Bit Cordic Hardware

i	$d^{-i} = 2^{-i} = \tan \phi_i$	$\phi_i = \arctan(2^{-i})$	ϕ_i in radian
0	1	45°	0.7854
1	0.5	26.565°	0.4636
2	0.25	14.036°	0.2450
3	0.125	7.125°	0.1244
4	0.0625	3.576°	0.0624
5	0.03125	1.7876°	0.0312
6	0.015625	0.8938°	0.0156
7	0.0078125	0.4469°	0.0078

After having a look at the table 2.1 we see that the inverse tan value of index 7 is 0.4469°, which is basically the precision possible for an 8 bit cordic, here the angle ϕ_i is stored in a ROM of the hardware of CORDIC as a look up table. Now the working of cordic algorithm is explained by using the balance example as follows.

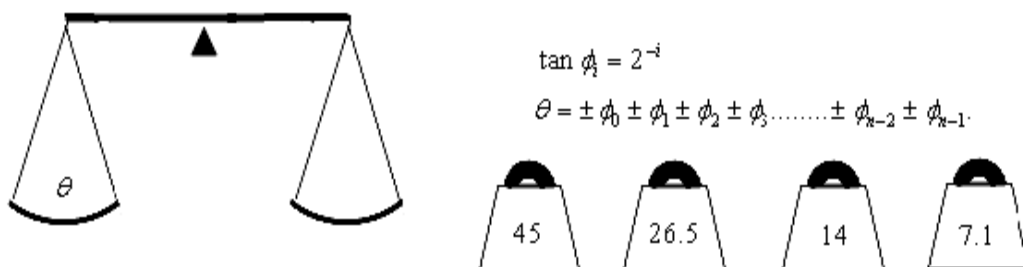


Figure 2.2: Balance having θ at one side and Φ on the other side.

In the above figure, first of all, keep the input angle θ on the left pan of balance and if the balance rotates around the anticlockwise direction then add the highest value in the table at the other side.

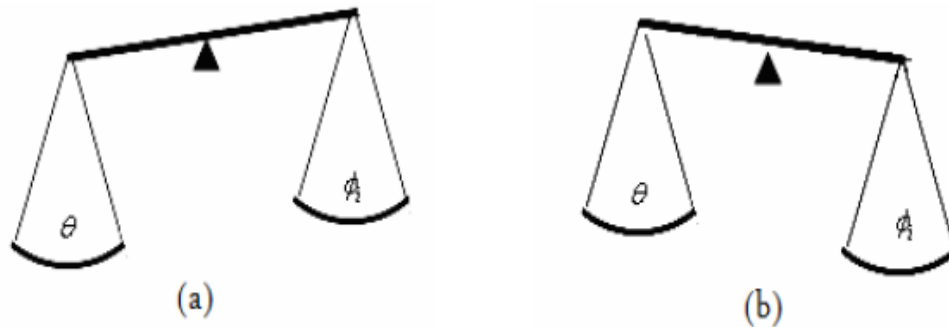


Figure 2.3: Inclined balance due to difference in weights of two sides

Then, if balance shows a left inclination as in figure 2.3 (a) then other weights are required to add in the right pan or in the term of angle if θ is greater than total ϕ_i then add other weights to reach as near to the θ as possible but in other hand if the balance shows a right inclination as in figure 2.3 (b) then a weight required to be removed from the right pan or in the term of angle if θ is less than total ϕ_i then we subtract other weights this process is repeated to reach as near to the θ as possible.

Matrix representation of the CORDIC algorithm for 8-bit hardware:

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} \cos \phi_i & \pm \sin \phi_i \\ \pm \sin \phi_i & \cos \phi_i \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad (2.11)$$

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} \cos \phi_0 & \pm \sin \phi_0 \\ \pm \sin \phi_0 & \cos \phi_0 \end{pmatrix} \begin{pmatrix} \cos \phi_1 & \pm \sin \phi_1 \\ \pm \sin \phi_1 & \cos \phi_1 \end{pmatrix} \dots \dots \dots \begin{pmatrix} \cos \phi_7 & \pm \sin \phi_7 \\ \pm \sin \phi_7 & \cos \phi_7 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad (2.12)$$

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \cos \phi_0 \cdot \cos \phi_1 \dots \dots \cos \phi_7 \begin{pmatrix} 1 & \pm \tan \phi_0 \\ \pm \tan \phi_0 & 1 \end{pmatrix} \begin{pmatrix} 1 & \pm \tan \phi_1 \\ \pm \tan \phi_1 & 1 \end{pmatrix} \dots \dots \dots \begin{pmatrix} 1 & \pm \tan \phi_7 \\ \pm \tan \phi_7 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad (2.13)$$

The scale factor = $\cos \phi_0 \cdot \cos \phi_1 \dots \dots \cos \phi_7$ (2.14)
 $= 0.6073$

Thus we can rewrite above equation as

$$\begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix} = \begin{pmatrix} 1 & \pm \tan \phi_0 \\ \pm \tan \phi_0 & 1 \end{pmatrix} \begin{pmatrix} 1 & \pm \tan \phi_1 \\ \pm \tan \phi_1 & 1 \end{pmatrix} \dots \dots \dots \begin{pmatrix} 1 & \pm \tan \phi_7 \\ \pm \tan \phi_7 & 1 \end{pmatrix} \begin{pmatrix} 0.6073 \\ 0 \end{pmatrix} \quad (2.15)$$

2.2 Basic CORDIC iterations

To simplify each rotation, picking α_i such that

$$\alpha_i = d_i \cdot 2^{-i} \quad (2.16)$$

d_i is such that it has a value either +1 or -1 depending upon the rotation .

Then we have

$$\begin{aligned}
 x_{i+1} &= x_i - d_i y_i 2^{-i} \\
 y_{i+1} &= y_i + d_i x_i 2^{-i} \\
 z_{i+1} &= z_i - d_i \tan^{-1} 2^{-i}
 \end{aligned}
 \tag{2.17}$$

The computation of x_{i+1} or y_{i+1} requires i-bit right shift and add /subtract. If the function $\tan^{-1} 2^{-i}$ is pre computed and stored in table (Table 2.1) for different values of i, a single add/subtract suffices to compute z_{i+1} . CORDIC iteration involves two shifts, a table lookup and three additions.

If the rotation is done by the same set of angles (with + or- signs), then the expansion factor K, is a constant, and can be pre computed. For example to rotate by 30 degrees, the following sequence of angles be followed that add up to ≈ 30 degree.

$$\begin{aligned}
 30.0 &\approx 45.0 - 26.6 + 14.0 - 7.1 + 3.6 + 1.8 - 0.9 + 0.4 - 0.2 + 0.1 \\
 &= 30.1
 \end{aligned}
 \tag{2.18}$$

In effect, what actually happens in CORDIC is that z is initialized to 30 degree and then, in each step, the sign of the next rotation angle is selected to try to change the sign of z; that is, $d_i = \text{sign}(z_i)$ is chosen, where the sign function is defined to be - 1 or 1 depending on whether the argument is negative or nonnegative. This is reminiscent of non restoring division. Table 2.2 shows the process of selecting the signs of the rotation angles for a desired rotation of +30 degree. Figure 3.1 depicts the first few steps in the process of forcing z to zero.

In CORDIC terminology the preceding selection rule for d_i , which makes z converge to zero, is known as rotation mode. Rewriting the CORDIC iteration, where

$$\alpha_i = \arctan .2^{-i}$$

Table 2.2: $\alpha_i = \arctan .2^{-i}$

i	α_i
0	45
1	26.6
2	14
3	7.1
4	3.6
5	1.8
6	0.9
7	0.4
8	0.2
9	0.1

$$x_{i+1} = x_i - d_i y_i 2^{-i} \tag{2.19}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \tag{2.20}$$

$$z_{i+1} = z_i - d_i \alpha_i \tag{2.21}$$

After n iterations we have

$\sum \alpha_i = z$ and the CORDIC equation becomes

$$x_n = k(x \cos z - y \sin z) \tag{2.22}$$

$$y_n = k(y \cos z + x \sin z)$$

Table 2.3: Example for 30° calculation

i	$z_i - \alpha_i$	z_{i+1}
0	+ 30.0 - 45.0	-15
1	- 15.0 + 26.6	11.6
2	+ 11.6 - 14.0	-2.4
3	- 2.4 + 7.1	4.7
4	+ 4.7 - 3.6	1.1
5	+ 1.1 - 1.8	-0.7
6	- 0.7 + 0.9	0.2
7	+ 0.2 - 0.4	-0.2
8	- 0.2 + 0.2	0
9	+ 0.0 - 0.1	-0.1

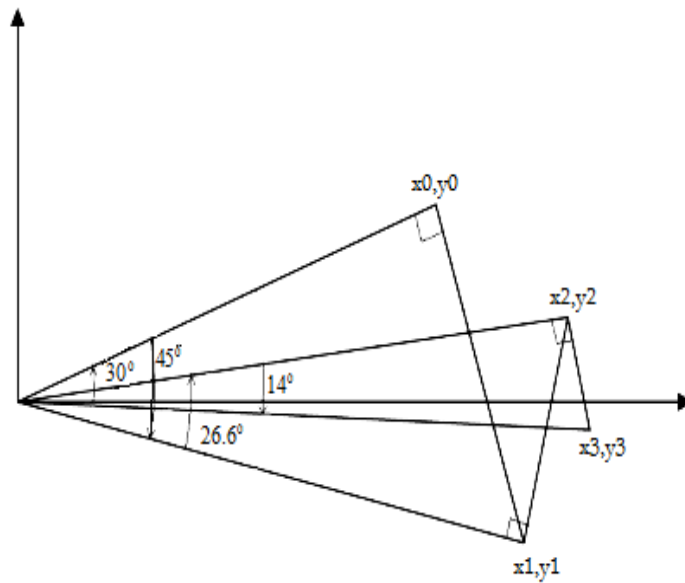


Figure 2.4: First few iterations for 30° calculation

For k bits of precision in the resulting trigonometric functions, k CORDIC iterations are needed. The reason is that for large i it can be approximated that $\tan^{-1} 2^{-i} \approx 2^{-i}$.

Chapter 3

COMPUTATION OF SINE AND COSINE

Elementary functions, especially trigonometric functions, play important roles in various digital systems, such as graphic systems, automatic control systems, and so on. The CORDIC [11], [12] is known as an efficient method for the computation of these elementary functions. Recent advances in VLSI technologies make it attractive to develop special purpose hardware such as elementary function generators. Several function generators based on the CORDIC have been developed [13]. The CORDIC can also be applied to matrix triangularization, singular value decomposition, and so on [14], [6]. In this chapter, different hardware is dealt for sine and cosine computation using CORDIC. In sine and cosine computation by the CORDIC, iterative rotations of a point around the origin on the X-Y plane are considered. In each rotation, the coordinates of the rotated point and the remaining angle to be rotated are calculated. The calculations in each iteration step are performed by shift, addition and subtraction, and recall of a prepared constant. Since the rotation is not a pure rotation but a rotation-extension, the number of rotations for each angle should be a constant independent of the operand so that the scale factor becomes a constant. When implementing a sine and cosine calculator in digital hardware, the expense of the multiplication needed for many algebraically methods, should be kept in mind. Alternative techniques are based on polynomial approximation, table-lookup [15] etc. as well as shift and add algorithms [15]. Among the various properties that are desirable, we can cite speed, accuracy or the reasonable amount of resource [15]. The architecture of FPGAs specifies suitable techniques or might even change desirable properties. Because the number of sequential cells and amount of storage area, needed for table-lookup algorithms, are limited but combinational logic in terms of LUT (Look Up Table) in the FPGA's (Field Programmable Gate Array) CLBs (Configurable Logic Blocks) is sufficiently available, shift and add algorithms fit perfectly into an FPGA.

3.1 CORDIC HARDWARE

A straight forward hardware implementation for CORDIC arithmetic is shown below in figure 3.1. It requires three registers for x, y and z, a look up table to store the values of $\alpha_i = \arctan.2^{-i}$, and two shifter to supply the terms $2^{-i}x$ and $2^{-i}y$ to the adder/subtractor units. The d_i factor (-1 and 1) is accommodated by selecting the (shift) operand or its complement.

Of, course a single adder and one shifter can be shared by three computations if a reduction in speed by a factor of 3 is acceptable. In the extreme, CORDIC iterations can be implemented in firmware (micro program) or even software using the ALU and general purpose registers of a standard microprocessor. In this case, the look up table supplying the term α_i can be stored in the control ROM or in main memory.

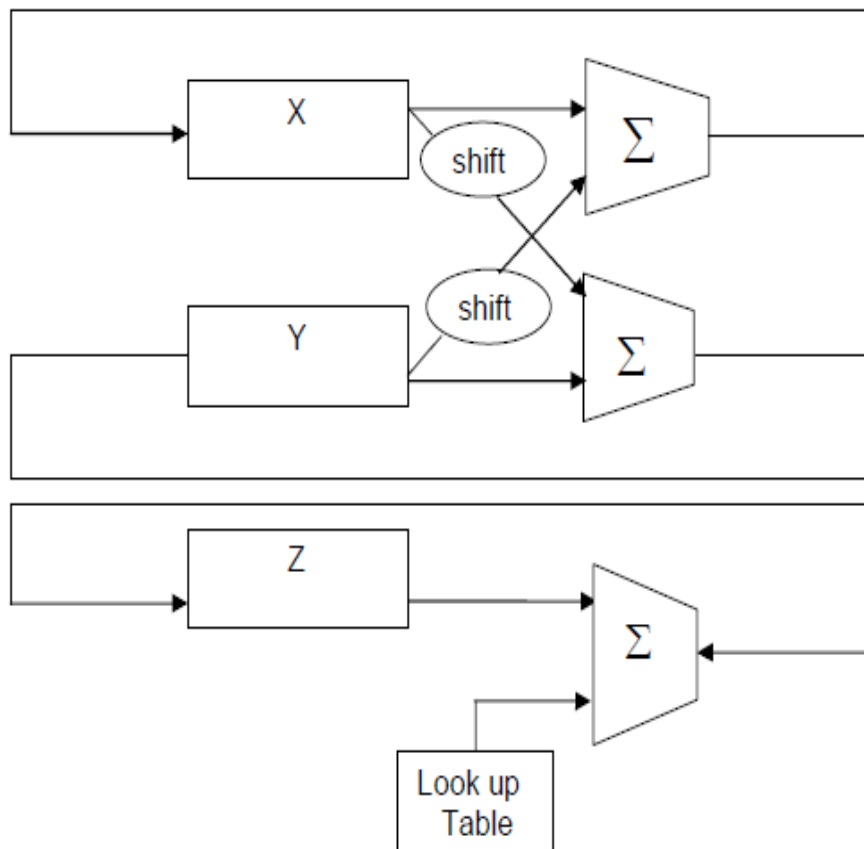


Figure 3.1: CORDIC hardware

Where high speed is not required and minimizing the hard ware cost is important (as in calculator), the adder in fig 3.1 can be a bit serial adder. Then with k bit operands, O (k) clock cycle would be required to complete the k CORDIC iterations. This is acceptable for hand handled calculators, since even a delay of tens of thousands of clock cycles constitutes a small fraction of a second and thus is hardly noticeable to a human user. Intermediate between the fully parallel and fully bit-serial realizations are a wide array of digit serial (for example decimal or radix-16) implementation that provide trade off speed versus cost.

3.2 The CORDIC algorithm for computing sine and cosine

Jack E. Volder [1] described the Coordinate Rotation Digital Computer or CORDIC for the calculation of trigonometric functions, multiplication, division and conversion between binary and mixed radix number systems. The CORDIC-algorithm provides an iterative method of performing vector rotations by arbitrary angles using only shifts and adds. Volder's algorithm is derived from the general equations for vector rotation. If a vector v with components (x, y) is to be rotated through an angle ϕ a new vector v' with components (x', y') is formed.

$$x = r \cos \theta \quad (3.1)$$

$$y = r \sin \theta$$

$$v' = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \cdot \cos(\phi) - y \cdot \sin(\phi) \\ y \cdot \cos(\phi) + x \cdot \sin(\phi) \end{pmatrix} \quad (3.2)$$

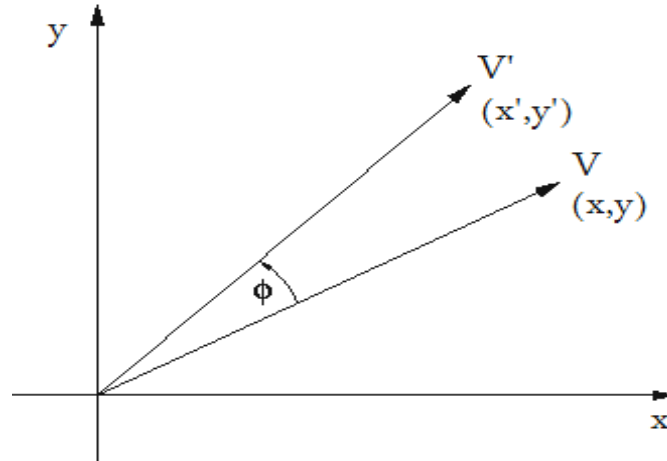


Figure 3.2: Angular rotation

$$\begin{aligned} x' &= x \cdot \cos(\phi) - y \cdot \sin(\phi) \\ y' &= y \cdot \cos(\phi) + x \cdot \sin(\phi) \end{aligned} \quad (3.3)$$

Taking $\cos(\phi)$ as common we have

$$\begin{aligned} x' &= \cos(\phi) [x - y \tan(\phi)] \\ y' &= \cos(\phi) [y + x \tan(\phi)] \end{aligned} \quad (3.4)$$

The multiplication by the tangent term can be avoided if the rotation angles and therefore $\tan(\phi)$ are restricted so that $\tan(\phi) = 2^{-i}$. In digital hardware this denotes a simple shift operation. Furthermore, if those rotations are performed iteratively and in both directions every value of $\tan(\phi)$ is representable. With $\phi = \arctan(2^{-i})$ the cosine term could also be simplified and since $\cos(\phi) = \cos(-\phi)$ it is a constant for a fixed number of iterations.

This iterative rotation can now be expressed as:

$$\begin{aligned} x_{i+1} &= k_i [x_i - y_i d_i 2^{-i}] \\ y_{i+1} &= k_i [y_i + x_i d_i 2^{-i}] \end{aligned} \quad (3.5)$$

$$k = \prod_{i=0}^{n-1} k_i \tag{3.6}$$

This k factor can be calculated in advance and applied elsewhere in the system. A good way to implement the k factor is to initialize the iterative rotation with a vector of length k which compensates the gain inherent in the CORDIC algorithm. The resulting vector v' is the unit vector as shown in Figure 3.3.

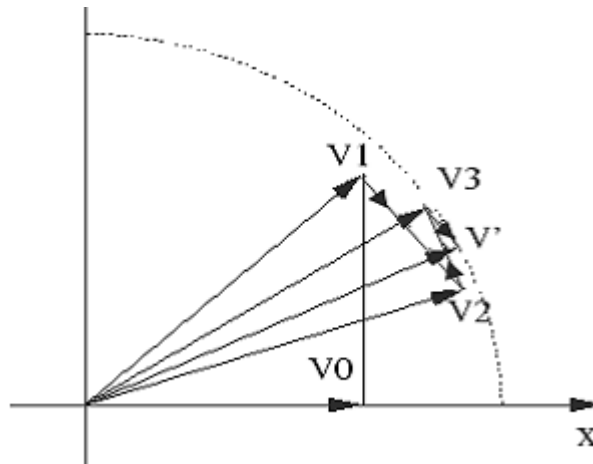


Figure 3.3: Iterative rotation

$$\begin{aligned} x_{i+1} &= [x_i - y_i d_i 2^{-i}] \\ y_{i+1} &= [y_i + x_i d_i 2^{-i}] \end{aligned} \tag{3.7}$$

The direction of each rotation is defined by d_i and the sequence of all d_i 's determines the final vector. This yields to a third equation which acts like an angle accumulator and keeps track of the angle already rotated. Each vector v can be described by both the vector length and angle or by its coordinates x and y . Following this incident, the CORDIC algorithm knows two ways of determining the direction of rotation: the *rotation mode* and the *vectoring mode*. Both methods initialize the angle accumulator with the desired angle z_0 . The *rotation mode*, determines the right sequence as the angle

accumulator approaches 0 while the *vectoring mode* minimizes the y component of the input vector.

$$z_{i+1} = z_i - d_i \cdot \arctan(2^{-i}) \quad (3.8)$$

Where the sum of an infinite number of iterative rotation angles equals the input angle [14]:

$$\phi = \sum_{i=0}^{\alpha} d_i \cdot \arctan(2^{-i}) \quad (3.9)$$

Those values $\arctan(2^{-i})$ can be stored in a small lookup table or hardwired depending on the way of implementation. Since the decision is which direction to rotate instead of whether to rotate or not, d_i is sensitive to the sign of z_i . Therefore d_i can be described as:

$$d_i = -1, +1 \quad (3.10)$$

3.3 Implementation of Various CORDIC architectures

As intended by Volder, the CORDIC algorithm only performs shift and add operations and is therefore easy to implement and resource-friendly. However, when implementing the CORDIC algorithm one can choose between various design methodologies and must balance circuit complexity with respect to performance. The most obvious methods of implementing a CORDIC, bit-serial, bit-parallel, unrolled and iterative, are described and compared in the following sections.

3.3.1 A Bit-Parallel Iterative CORDIC

The CORDIC structure as described in Figure 3.4 when directly translated into hardware. Each branch consists of an adder-subtractor combination, a shift unit and a register for buffering the output. At the beginning of a calculation initial values are fed into the register by the multiplexer where the MSB of the stored value in the z-branch determines

the operation mode for the adder-sub tractor. Signals in the x and y branch pass the shift units and are then added to or subtracted from the unshifted signal in the opposite path.

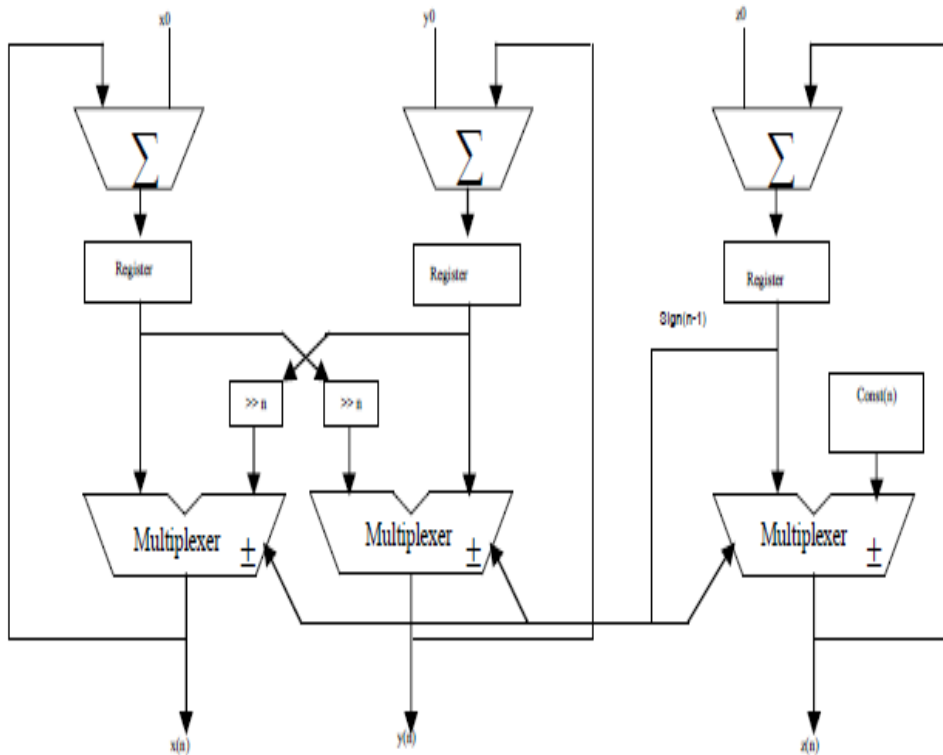


Figure 3.4: Iterative CORDIC

The z branch arithmetically combines the registers values with the values taken from a lookup table (LUT) whose address is changed accordingly to the number of iteration. For n iterations the output is mapped back to the registers before initial values are fed in again and the final sine value can be accessed at the output. A simple finite-state machine is needed to control the multiplexers, the shift distance and the addressing of the constant values. When implemented in an FPGA the initial values for the vector coordinates as well as the constant values in the LUT can be hardwired in a word wide manner. The adder and the sub tractor component are carried out separately and a multiplexer

controlled by the sign of the angle accumulator distinguishes between addition and subtraction by routing the signals as required.

shift operations as implemented change the shift distance with the number of iterations but those require a high fan in and reduce the maximum speed for the application [18]. In addition the output rate is also limited by the fact that operations are performed iteratively and therefore the maximum output rate equals $1/n$ times the clock rate.

3.3.2 A Bit-Parallel Unrolled CORDIC

Instead of buffering the output of one iteration and using the same resources again, one could simply cascade the iterative CORDIC, which means rebuilding the basic CORDIC structure for each iteration. Consequently, the output of one stage is the input of the next one, as shown in Figure 3.5, and in the face of separate stages two simplifications become possible. First, the shift operations for each step can be performed by wiring the connections between stages appropriately. Second, there is no need for changing constant values and those can therefore be hardwired as well.

The purely unrolled design only consists of combinatorial components and computes one sine value per clock cycle. Input values find their path through the architecture on their own and do not need to be controlled. Obviously the resources in an FPGA are not very suitable for this kind of architecture. As we talk about a bit-parallel unrolled design with 16 bit word length, each stage contains 48 inputs and outputs plus a great number of cross-connections between single stages. Those cross-connections from the x-path through the shift components to the y-path and vice versa make the design difficult to route in an FPGA and cause additional delay times. From table 4.1 it can be seen how performance and resource usage change with the number of iterations if implemented in an XILINX FPGA. Naturally, the area and therefore the maximum path delay increase as stages are added to the design where the path delay is an equivalent to the speed which the application could run at.

Table 3.1: Performance and CLB usage in an XC4010E [19]

No. of iterations	8	9	10	11	12	13
Complexity[CLB]	184	208	232	256	280	304
Max path delays[ns]	163.75	177.17	2.906	225.7	263.8	256.8

As described earlier, the area in FPGAs can be measured in CLBs, each of which consist of two lookup tables as well as storage cells with additional control components [20], [21]. For the purely combinatorial design the CLB's function generators perform the add and shift operations and no storage cells are used. This means registers could be inserted easily without significantly increasing the area.

However, inserting registers between stages would also reduce the maximum path delays and correspondingly a higher maximum speed can be achieved. It can be seen, that the number of CLBs stays almost the same while the maximum frequency increases as registers are inserted. The reason for that is the decreasing amount of combinatorial logic between sequential cells. Obviously, the gain of speed when inserting registers exceeds the cost of area and makes therefore the fully pipelined CORDIC a suitable solution for generating a sine wave in FPGAs. Especially if a sufficient number of CLBs is at one's disposal, as is the case in high density devices like XILINX's Virtex or ALTERA's FLEX families, this type of architecture becomes more and more attractive.

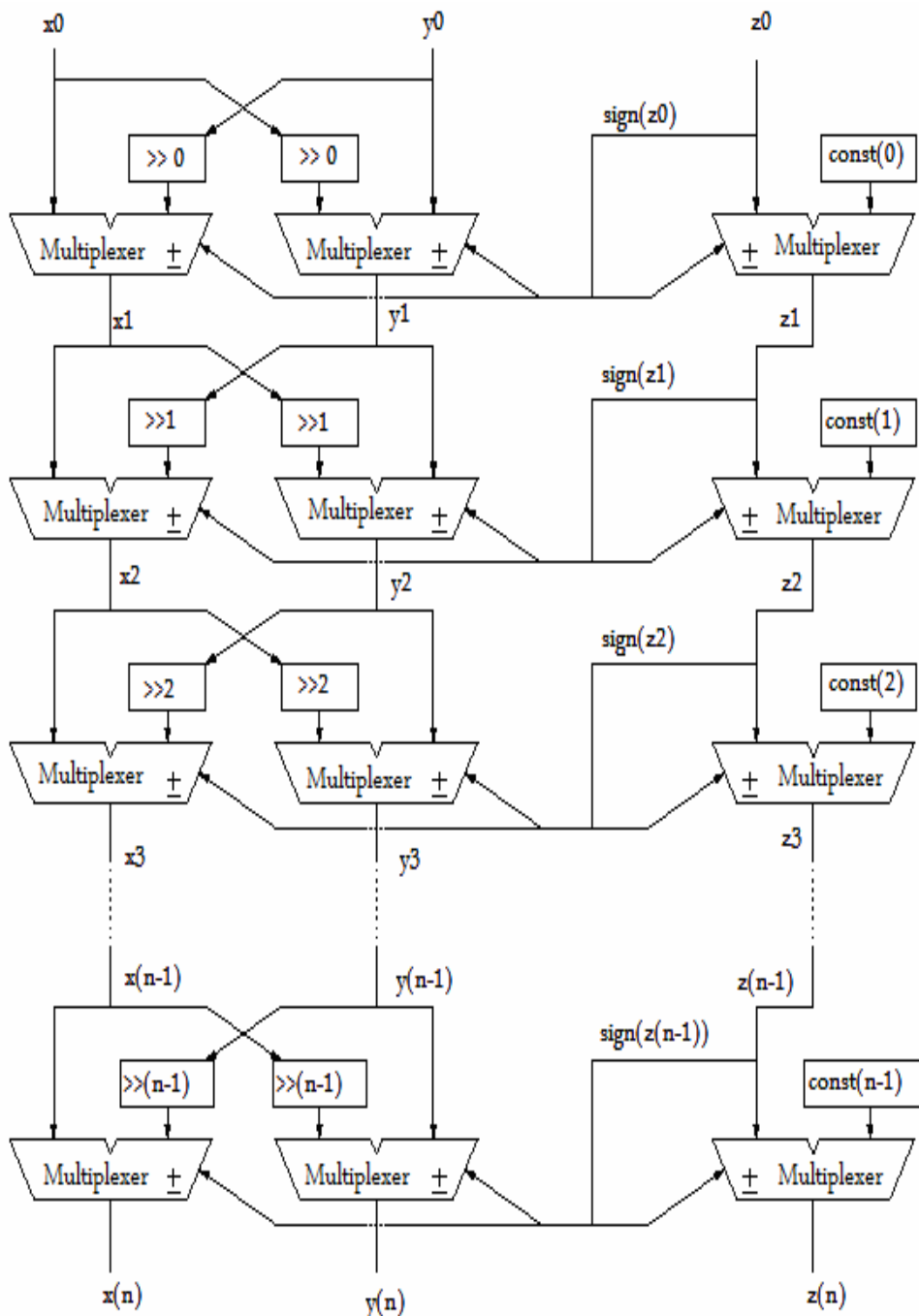


Figure 3.5: Unrolled CORDIC

Chapter 4

Bit Serial Cordic

4.1 A Bit-Serial Iterative CORDIC

Problems which involve repeated evaluation of a fixed set of nonlinear, algebraic equations appear frequently in scientific and engineering applications. Examples of such problems can be found in the robotics, engineering graphics, and signal processing areas. Evaluating complicated equation sets can be very time consuming in software, even when co-processors are used, especially when these equations contain a large number of nonlinear and transcendental functions as well as many multiplication and division operations. Both, the unrolled and the iterative bit-parallel designs, show disadvantages in terms of complexity and path delays going along with the large number of cross connections between single stages. To reduce this complexity we can change the design into a completely bit-serial iterative architecture. Bit-serial means only one bit is processed at a time and hence the cross connections become one bit-wide data paths. Clearly, the throughput becomes a function of

$$\frac{\text{Clock rate}}{\text{Number of iterations} \times \text{word width}}$$

In spite of bit serial the output rate can be almost as high as achieved with the unrolled design. The reason is the structural simplicity of a bit-serial design and the correspondingly high clock rate achievable. Figure 4.1 shows the basic architecture of the bit serial CORDIC processor as implemented in a XILINX Spartan.

In this architecture the bit-serial adder-subtractor component is implemented as a full adder where the subtraction is performed by adding the 2's complement of the actual subtrahend [22]. The subtraction is again indicated by the sign bit of the angle accumulator. A single bit of state is stored at the adder to realize the carry chain [23] which at the same time requires the LSB to be fed in first. The shift-by- i operation can be realized by reading the bit $i - 1$ from its right end in the serial shift registers. A

multiplexer can be used to change position according to the current iteration. The initial values x_0 , y_0 and z_0 are fed into the array at the left end of the serial-in serial-out register and as the data enters the adder component the multiplexer at the input switch and map back the results of the bit-serial adder into the registers.

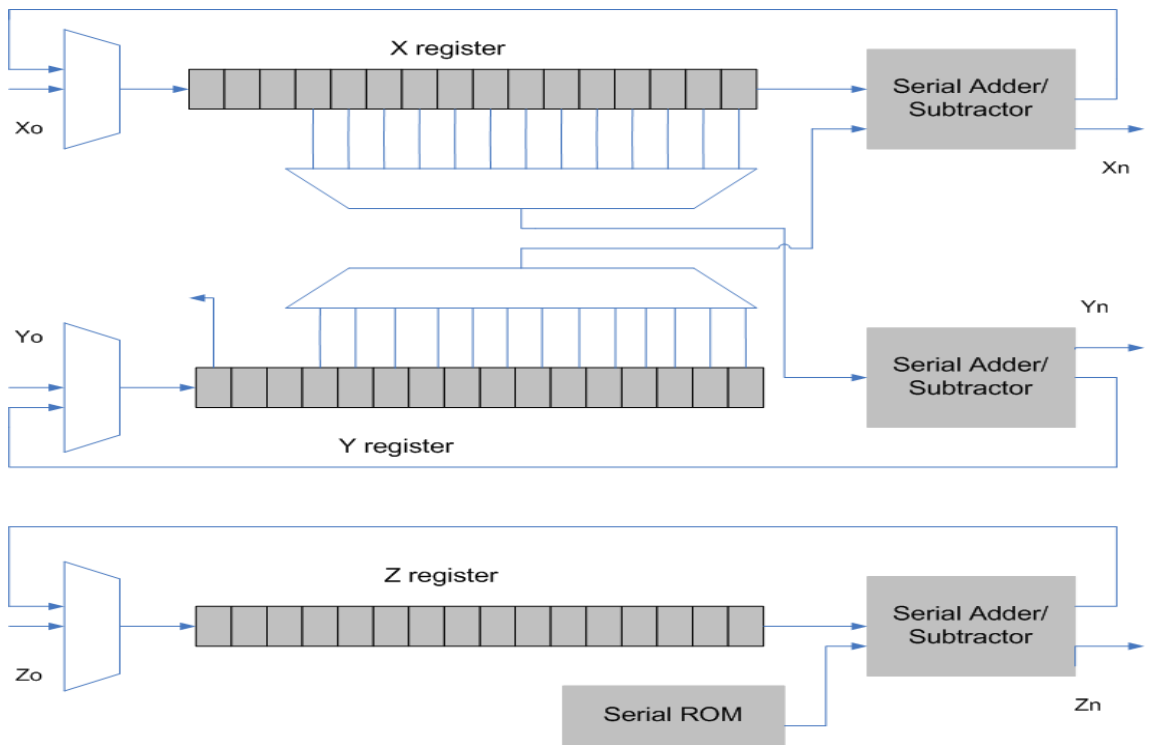


Figure 4.1: Bit serial CORDIC

The constant LUT for this design is implemented as a multiplexer with hardwired choices. Finally, when all iterations are passed the input multiplexers switch again and initial values enter the bit-serial CORDIC processor as the computed sine values exit. The design as implemented runs at a much higher speed than the bit-parallel architectures described earlier and fits easily in a XILINX SPARTAN device. The reason is the high ratio of sequential components to combinatorial components. The performance is constrained by the use of multiplexers for the shift operation and even more for the constant LUT. The latter could be replaced by a RAM or serial ROM where values are

read by simply incrementing the memory's address. This would clearly accelerate the performance.

4.2 Proposed Bit-Serial modified CORDIC

As discussed in previous chapter we have the following equations for CORDIC.

$$\begin{aligned} x_{i+1} &= [x_i - y_i d_i 2^{-i}] \\ y_{i+1} &= [y_i + x_i d_i 2^{-i}] \end{aligned} \quad (4.1)$$

$$\phi_i = \sum_{i=0}^{\infty} d_i 2^{-i} \quad (4.2)$$

We assume that in modified CORDIC d_i is either 1 or 0, In the binary notation of ϕ_i for positive rotation we will use (+1) when $d_i=1$ and for negative rotation we will use (-1) when $d_i=0$.

This idea is clarified in the example below

$$\phi_i = 101100 \quad (4.3)$$

$$d_i = (+1)(-1)(+1)(+1)(-1)(-1) \quad (4.4)$$

The pre-calculation of d_i removes the dependency of each iteration waiting for previous iteration to complete, which is a major limitation in architecting a bit serial parallel architecture. This modification has enabled the design to work in parallel a bit serial level.

$$\begin{aligned} d_i &= +1 \\ x_n &= x_{n-1} - (2^{-n})y_{n-1} \\ y_n &= y_{n-1} + (2^{-n})x_{n-1} \end{aligned} \quad (4.5)$$

$$\begin{aligned} d_i &= -1 \\ x_n &= x_{n-1} + (2^{-n})y_{n-1} \\ y_n &= y_{n-1} - (2^{-n})x_{n-1} \end{aligned} \quad (4.6)$$

Our Proposed Bit Serial CORDIC Algorithm uses Rom for the initial five values where ϕ_n varies significantly and for the remaining iterations it uses Eq. 4.5 and 4.6.

By looking at above equations we need single bit adder / subtractor along with right hardware shifts as required. Our pre computed highest value can be represented in 13 bit binary format but for the shift purposes we extend it by six bits, though the final representation is in 13 bit.

For initializing purpose an index is calculated and based on that index value the initial value i.e X5 and Y5 are selected from pre calculated values generated after extensive test and trials as shown in Table 4.1 below.

Table 4.1: Initialization of X and Y

Sr. No.	Initial X5 (decimal binary)	Initial Y5 (decimal binary)
1	8187 111111111011	256 0000100000000
2	8091 1111110011011	1274 0010011111010
3	7869 1111010111101	2273 0100011100001
4	7524 1110101100100	3237 0110010100101
5	7062 1101110010110	4149 1000000110101

6	6489 1100101011001	4998 1001110000110
7	5816 1011010111000	5768 1011010001000
8	5051 1001110111011	6448 1100100110000
9	4208 1000001110000	7027 1101101110011
10	3299 0110011100011	7497 1110101001001
11	2339 0100100100011	7850 1111010101010
12	1342 0010100111110	8080 1111100100000
13	324 0000101000100	8184 1111111110000

The initial five iterations $5x$ and $5y$ are stored in the Rom in 19 bit binary format and the value of d_i is pre calculated for the whole 14 iterations, the value of d_i is either +1 or -1, X_6 and Y_6 are calculated as under: if

$$\begin{aligned}
 d_i &= +1 & (4.7) \\
 x_n &= x_{n-1} - (2^{-n})y_{n-1} \\
 y_n &= y_{n-1} + (2^{-n})x_{n-1}
 \end{aligned}$$

We have

$$\begin{aligned}
 Cin[0] &= 0 & (4.8) \\
 x_6[1] &= x_5[1] - y_5[7] + Cin[0] \\
 x_6[2] &= x_5[2] - y_5[8] + Cin[1]
 \end{aligned}$$

$$\begin{aligned}
 x_6[3] &= x_5[3] - y_5[9] + Cin[2] \\
 x_6[4] &= x_5[4] - y_5[10] + Cin[3] \\
 x_6[5] &= x_5[5] - y_5[11] + Cin[4] \\
 x_6[6] &= x_5[6] - y_5[12] + Cin[5] \\
 x_6[7] &= x_5[7] - y_5[13] + Cin[6] \\
 x_6[8] &= x_5[8] - y_5[14] + Cin[7] \\
 x_6[9] &= x_5[9] - y_5[15] + Cin[8] \\
 x_6[10] &= x_5[10] - y_5[16] + Cin[9] \\
 x_6[11] &= x_5[11] - y_5[17] + Cin[10] \\
 x_6[12] &= x_5[12] - y_5[18] + Cin[11] \\
 x_6[13] &= x_5[13] - y_5[19] + Cin[12]
 \end{aligned}$$

It is to be noted here that for negative sign 2's compliment addition is used

Similarly for y_6 we have

$$\begin{aligned}
 Cin[0] &= 0 \\
 y_6[1] &= y_5[1] + x_5[7] + Cin[0] \\
 y_6[2] &= y_5[2] + x_5[8] + Cin[1] \\
 y_6[3] &= y_5[3] + x_5[9] + Cin[2] \\
 y_6[4] &= y_5[4] + x_5[10] + Cin[3] \\
 y_6[5] &= y_5[5] + x_5[11] + Cin[4] \\
 y_6[6] &= y_5[6] + x_5[12] + Cin[5] \\
 y_6[7] &= y_5[7] + x_5[13] + Cin[6] \\
 y_6[8] &= y_5[8] + x_5[14] + Cin[7] \\
 y_6[9] &= y_5[9] + x_5[15] + Cin[8] \\
 y_6[10] &= y_5[10] + x_5[16] + Cin[9] \\
 y_6[11] &= y_5[11] + x_5[17] + Cin[10] \\
 y_6[12] &= y_5[12] + x_5[18] + Cin[11] \\
 y_6[13] &= y_5[13] + x_5[19] + Cin[12]
 \end{aligned} \tag{4.9}$$

Similarly we can calculate X14 and Y14 by using the same technique, we require 13 iterations for each value of X and we need another 14 iterations in order to reach the final value i.e. x14 and y14 which is Sin and Cos of required angle.

The above technique is simulated in Matlab and the segment of code is as under.

```
clear all ;clc;
theta = input('theta = ');
N = 14;
tablex = [8187 8091 7869 7524 7062 6489 5816 5051 4208 3299
2339 1342 324];
tabley = [256 1275 2273 3237 4149 4998 5768 6448 7027 7497 7850
8080 8184];

actual = [sin(theta) cos(theta)]

theta = fix(theta * 2^(N-1));
for k=1:N
    b(N+1-k) = abs(rem(theta, 2));
    theta = fix(theta/2);
end

for k=1:N
    r(k) = 2*(b(k)) - 1;
end

index = b(4) + b(3)*2 + b(2)*4 + b(1)*8 + 1;

x(5) = tablex(index);
y(5) = tabley(index);

% atan table for k=6:17
for k=6:N
    x(k) = x(k-1) - r(k) * 2^(-k) * y(k-1);
    y(k) = y(k-1) + r(k) * 2^(-k) * x(k-1);
end

yk = y(k);
xk = x(k);
ddfs = [y(k) x(k)] *2^(-(N-1))
```

Fig. 4.2a Segment of MATLAB code

4.3 Flow of the MATLAB Code

The MATLAB Code developed for the calculation of Sine and cosine has the following flow diagram.

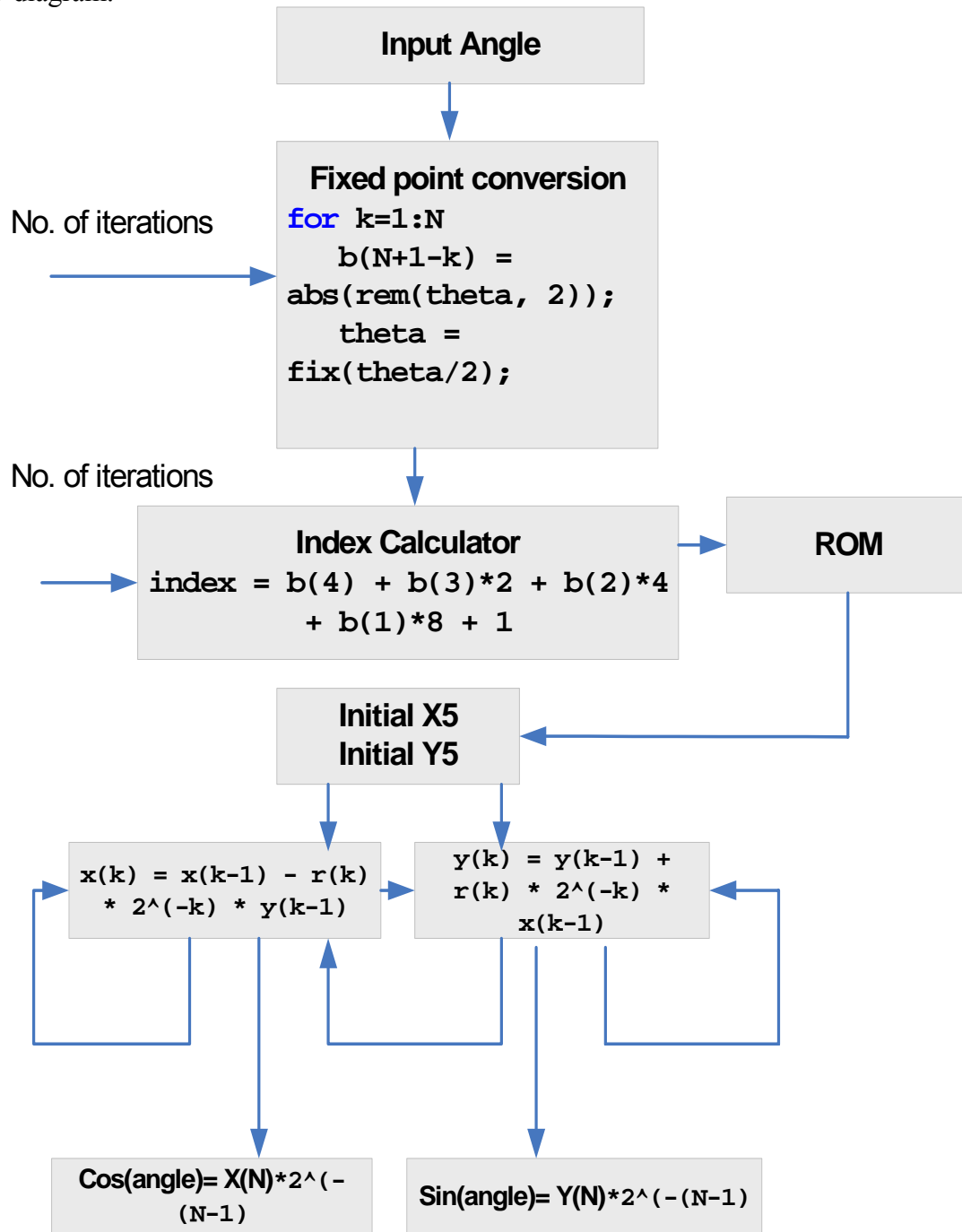


Fig. 4.2b Flow diagram of MATLAB Code

4.4 Proposed Bit-Serial modified CORDIC Architecture

The proposed bit serial architecture consists of adder / subtractor, shift registers and a ROM. There is also some gating or multiplexers to select the taps of the shift registers for the right shifted cross terms. The Bit serial CORDIC was successfully implemented as a part of a communication system, the input to the system is a 1's complement value of angle from $-\pi$ to π , the architecture is shown in Fig.4.3. The value of angle is converted in radians and then fixed point which returns a binary equivalent, after that the values goes to the index calculator which returns the address of Rom which has 13 values stored for X and 13 values stored for Y, which are used to initialize the value of X5 and Y5. The ROM has 13 values for X and 13 values for Y. The out put of index calculator initializes the value of X5 and Y5 in the shift register. When the value of the bit (LSB to MSB) in binary format for angle is '0' the value for the Rn is '-1' and when the value of angle is '1' the value of Rn is '+1'. The binary format of angle plays an important role as it directly calculates the value of Rn which is used in the calculation of sine and cosine, it means we deal with only two equation in this modified Bit serial CORDIC in stead of three equations. This means that the inter equation dependency is no more and the angle has all the information about the sine and cosine.

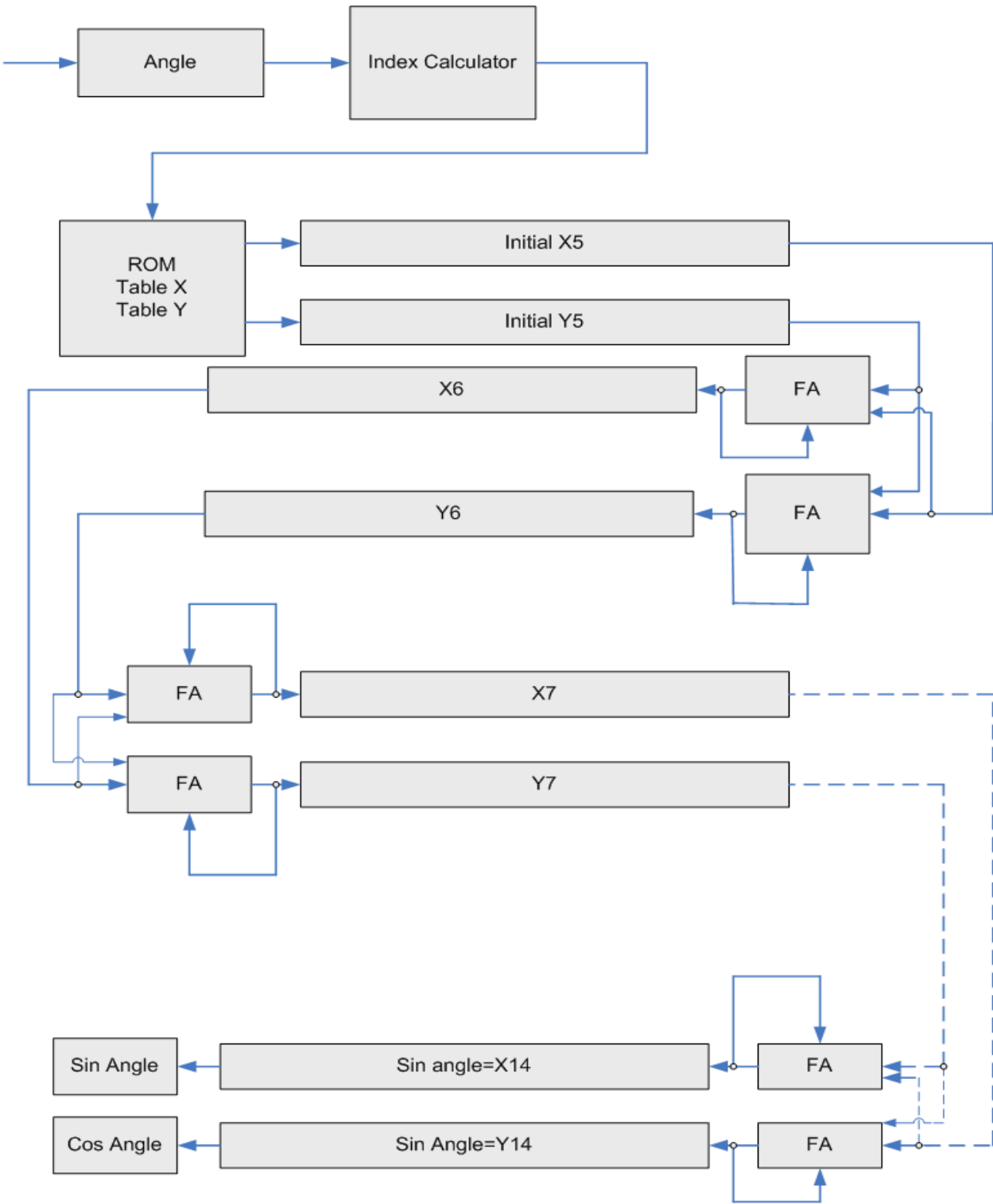


Fig. 4.3 Bit Serial modified CORDIC Architecture

Eq. 4.8 and 4.9 are used to calculate the value of X6 and y6, it takes total of (13 x14) iterations to calculate the value of Sin and Cos of the given angle. ROM has a size of 13 bit x 26 word. The value of X14 and y14 returns the Sin and Cos of desired angle.

4.4 Algorithm Flow of Proposed Architecture with Example

In order to understand the complete working of the proposed algorithm we take a example of 30° i.e 0.5236 radians.

Now $0.5236 * 2^{13} = 4289 = 01000011000001$ which is an input to the b array as per fig.

4.3.

Table 4.2: Calculation of B(n) and R(n)

Serial	B(n)	R(n)
1	0	-1
2	1	1
3	0	-1
4	0	-1
5	0	-1
6	0	-1
7	1	1
8	1	1
9	0	-1
10	0	-1
11	0	-1
12	0	-1
13	0	-1
14	1	1

The index is calculated as follows as per the MATLAB code segment in Fig. 4.2.

$$\text{Index} = B(4) + B(3)*2 + B(2)*4 + B(1)*8 + 1 \tag{5.1}$$

By using the values from table 2a the value of index comes out to be 5 there fore the initial value of X5 and Y5 are as under:

$$X5=1101110010110 \quad (5.2)$$

$$Y5=1000000110101 \quad (5.3)$$

Now we will use the following set of equations as our data is bit serial in order to get X6 and Y6, for X6 we have

$$Cin[0] = 0$$

$$x_6[1] = x_5[1] - y_5[7] + Cin[0] \quad (5.4)$$

$$x_6[2] = x_5[2] - y_5[8] + Cin[1]$$

$$x_6[3] = x_5[3] - y_5[9] + Cin[2]$$

$$x_6[4] = x_5[4] - y_5[10] + Cin[3]$$

$$x_6[5] = x_5[5] - y_5[11] + Cin[4]$$

$$x_6[6] = x_5[6] - y_5[12] + Cin[5]$$

$$x_6[7] = x_5[7] - y_5[13] + Cin[6]$$

$$x_6[8] = x_5[8] - y_5[14] + Cin[7]$$

$$x_6[9] = x_5[9] - y_5[15] + Cin[8]$$

$$x_6[10] = x_5[10] - y_5[16] + Cin[9]$$

$$x_6[11] = x_5[11] - y_5[17] + Cin[10]$$

$$x_6[12] = x_5[12] - y_5[18] + Cin[11]$$

$$x_6[13] = x_5[13] - y_5[19] + Cin[12]$$

In every cycle we get one bit out and finally we get :

$$X6=1101111010110$$

Similarly for Y6 we have following set of equations

$$Cin[0] = 0$$

$$y_6[1] = y_5[1] + x_5[7] + Cin[0]$$

$$y_6[2] = y_5[2] + x_5[8] + Cin[1]$$

$$y_6[3] = y_5[3] + x_5[9] + Cin[2]$$

$$y_6[4] = y_5[4] + x_5[10] + Cin[3]$$

$$y_6[5] = y_5[5] + x_5[11] + Cin[4]$$

$$\begin{aligned}
 y_6[6] &= y_5[6] + x_5[12] + Cin[5] \\
 y_6[7] &= y_5[7] + x_5[13] + Cin[6] \\
 y_6[8] &= y_5[8] + x_5[14] + Cin[7] \\
 y_6[9] &= y_5[9] + x_5[15] + Cin[8] \\
 y_6[10] &= y_5[10] + x_5[16] + Cin[9] \\
 y_6[11] &= y_5[11] + x_5[17] + Cin[10] \\
 y_6[12] &= y_5[12] + x_5[18] + Cin[11] \\
 y_6[13] &= y_5[13] + x_5[19] + Cin[12]
 \end{aligned}
 \tag{5.5}$$

In every cycle we get one bit out for Y6 as well

$$Y_6 = 0111111000110$$

Now by repeating the set of equations we have used for X6 and Y6 we get the values as per Table 4.3.

Table 4.3: Bit Serial CORDIC Calculation

Sr. No.	X	Y
5	1101110010110	1000000110101
6	1101111010110	0111111000110
7	1101110110111	0111111111110
8	1101110100111	1000000011010
9	1101110101111	1000000001100
10	1101110110011	1000000000101
11	1101110110101	1000000000001
12	1101110110110	1000000000000

13	1101110110110	0111111111111
14	1101110000000	1000000100000

From Table 3 we see that the value of X14 and Y14 are as under:

$$X_{14} = 1101110000000 = 7040 \tag{5.6}$$

$$Y_{14} = 1000000100000 = 4128 \tag{5.7}$$

By dividing both the values by 2^{13} we get

$$\mathbf{Cos(30) = X_{14} = 7040 / 2^{13} = 0.857395 \approx 0.866}$$

$$\mathbf{Sin(30) = Y_{14} = 4128 / 2^{13} = 0.5036 \approx 0.5}$$

We have seen that the value of B is directly calculated from the angle for which the Cos and Sin is to be determined, B is also used for the calculation of d_i and index, there fore by using this algorithm only by knowing angle we can calculate B (n) index and R_n for all the 14 iterations, and this is the reason we say that the iterations are independent from the start we know the value of d_i which is used to select the set of equations.

Chapter 5

Results and Block Description

5.1 ModelSim Simulation Results

Figure 5.1 shows the ModelSim simulation result for binary input angle 30° and binary outputs x_{new} ($\sin(30^\circ)$), y_{new} ($\cos(30^\circ)$) in the form of waveform and their corresponding magnitude respectively. Figure 5.2 consists the ModelSim simulation result for real input angle 30° and real outputs x_{new} ($\sin(30^\circ)$), y_{new} ($\cos(30^\circ)$) in the form of waveform and their corresponding magnitude respectively.

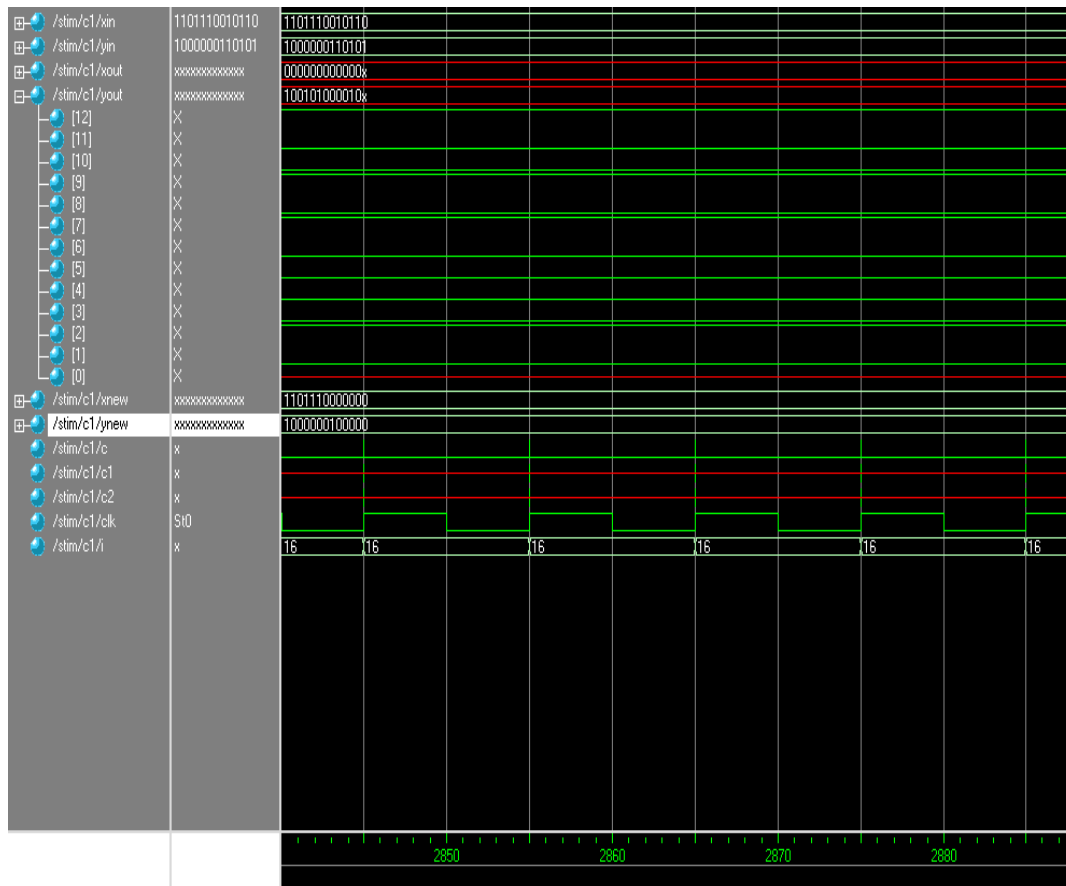


Fig 5.1 Sine and Cosine values generated for an input angle 30(binary value)

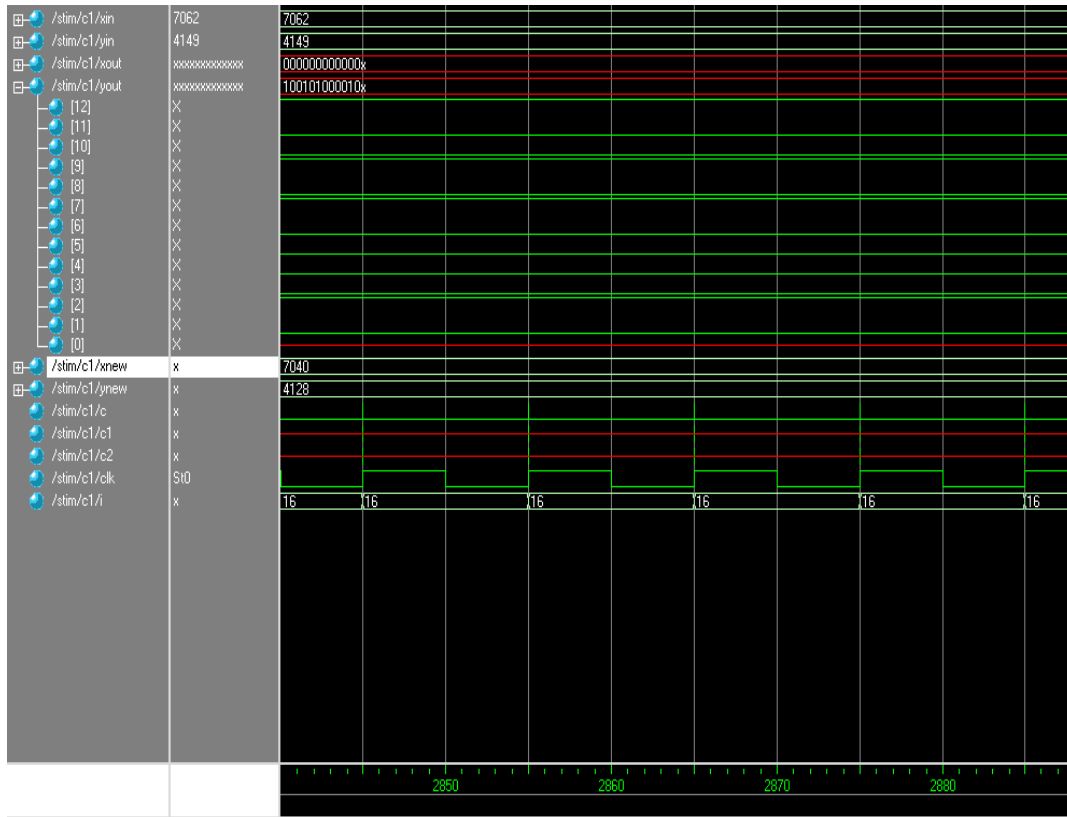


Fig. 5.2 Sine and sine value generated for 30(integer value)

5.2 Block Description of Verilog Code

A brief description of various modules used to implement the bit serial modified CORDIC is described as under:

5.2.1 Block CORDIC Main module

The name of this module is “it” in the Verilog code. Block diagram generated by XILINX 9.2i for sine-cosine using CORDIC is shown in figure 5.3. Here inputs are θ (input angle), clk (clock) and output is DDFS i.e the sine and cosine of the input angle. The module named “it” is calling the modules named “mn” and “ROM” in it.



Fig. 5.3 Main Module RTL schematic for Bit Serial modified CORDIC

5.2.2 Internal RTL schematic of Main Module

The internal RTL schematic of main module is shown as per Figure 5.4. The module name is “mn”. Here the inputs are the initial values of X and Y which is 13 bit each, the angle ta which is 16 bit with clock and reset and the out put is the sine and cosine and a 4 bit “id” which is used for index calculation of ROM. The name of this module is mn in the verilog code.



Fig. 5.4 Internal RTL schematic of Main Module for Bit Serial modified CORDIC

The detailed schematic of above module is as per figure 5.5

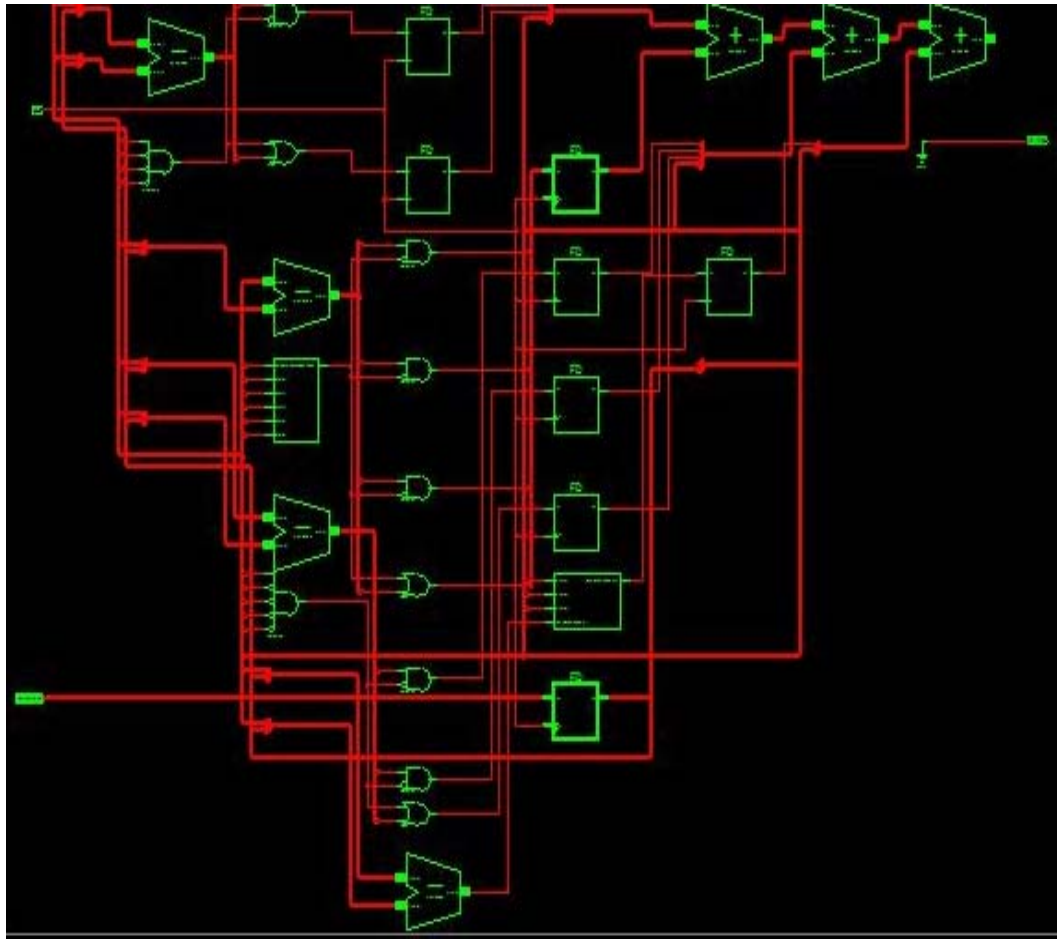


Fig. 5.5 RTL schematic for Bit Serial modified CORDIC

5.2.3 Internal RTL schematic of ROM

The module ROM has the initial values of X and Y stored in it. As per the angle the index is calculated and the value of X and Y is initialized from that index. The ROM has the initial five values and the rest are calculated iteratively in a bit serial manner. The schematic of ROM is as per Figure 5.7.

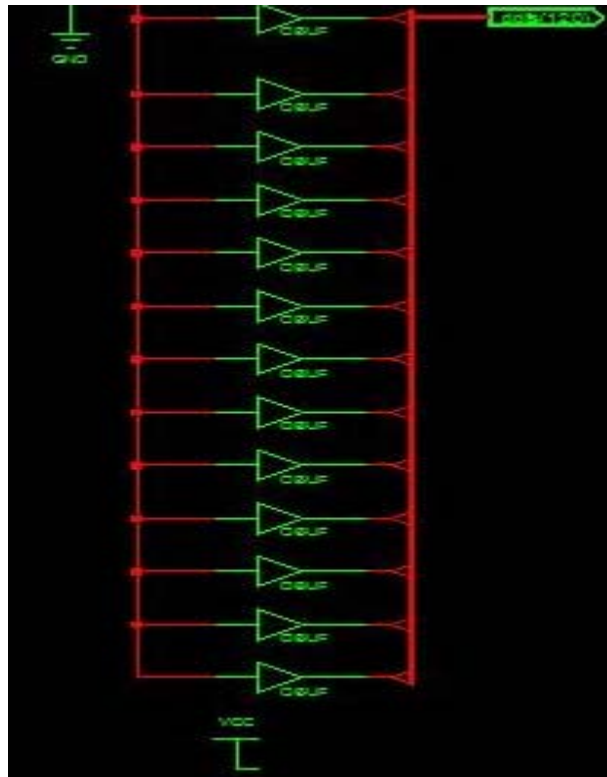


Fig. 5.6. Detailed RTL schematic of ROM

Figure 5.7 and table 5.1 shows the synthesis report generated by XILINX 9.2i showing number of multiplexers, number of adder, number of flip-flops used and timing of the chip generated respectively.

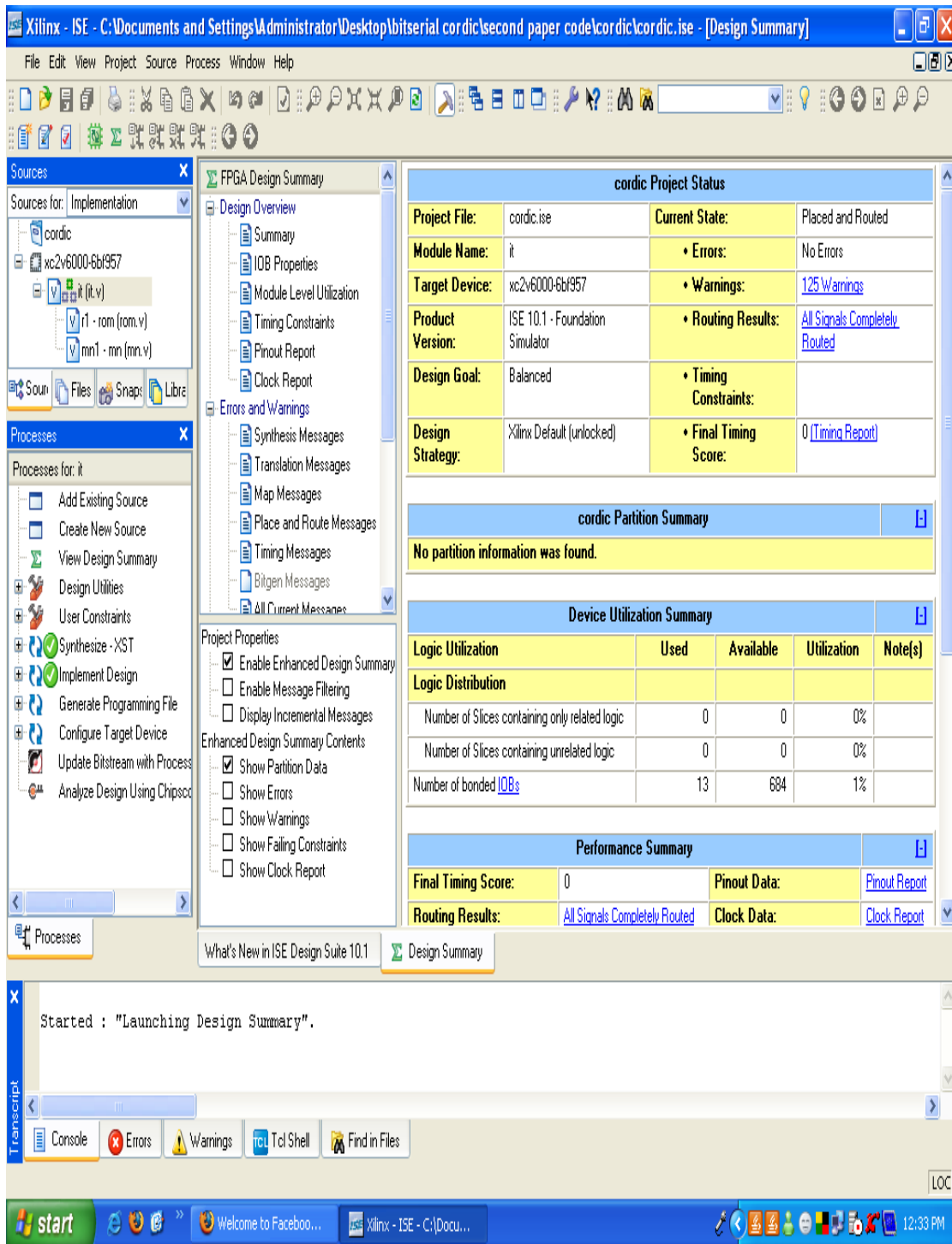


Fig. 5.7 Synthesis Report of bit serial Modified CORDIC

Table 5.1: Advanced HDL Synthesis Report for sine cosine

Macro Statistics	
Total Adders/Subtractors	: 7
16-bit adder	: 3
16-bit subtractor	: 4
Total Counters	: 1
4-bit up counter	: 1
#Total Registers	: 12
13-bit register	: 2
16-bit register	: 9
4-bit register	: 1
CPU time	:13.69 ns

5.3 Error Analysis

The Error in the values of different angles is as per the fig. 5.8, this results show that the error value is negligible as compared to the performance.

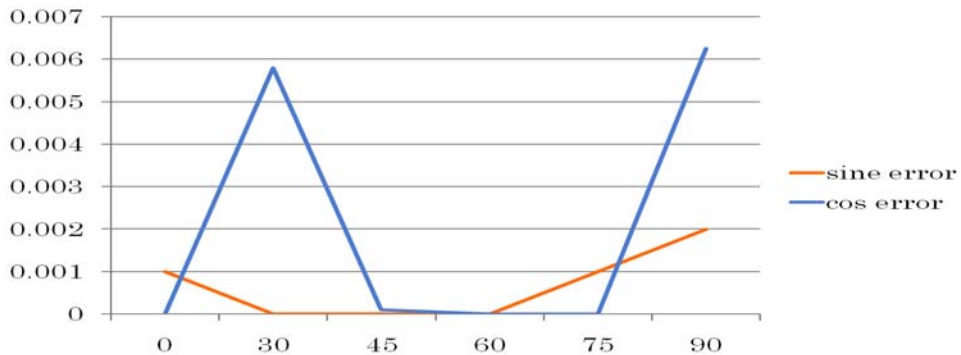


Fig.5.8 Error Analysis

5.4 Discussions

The modified bit serial CORDIC was successfully synthesized on Xilinx Vertex II and the maximum frequency of the bit serial modified CORDIC is 73 Mz on Xilinx Vertex II Xc2v6000 (6bf957).

The Architecture has a ROM in which first five values are stored maintaining a small size, these values are calculated after hit and trial and are found to be accurate. The working principal is simple as the value of desired angle is converted into binary, the iterative addition/subtraction is calculated, the binary value of the angle gives us the add / subtract strategy for iterations to be performed so as to speedup the process. This shows that the iterative process has nearly zero inter-iteration dependency, the results are accurate and the small ROM size makes the architecture efficient as compared to other models in practice with a average error value of .005852%.

Chapter 6

Conclusion

6.1 Overview

The CORDIC algorithm is a powerful and widely used tool for digital signal processing applications and can be implemented using PDPs (Programmable Digital Processors). The Jack E. Volder's CORDIC algorithm is derived from the general equations for vector rotation. The CORDIC algorithm has become a widely used approach to elementary function evaluation when the silicon area is a primary constraint. The implementation of CORDIC algorithm requires less complex hardware than the conventional method.

Bit-serial means only one bit is processed at a time and hence the cross connections become one bit-wide data paths. Normally in Bit Serial iterative CORDIC architectures the bit-serial adder- subtractor component is implemented as a full adder where the subtraction is performed by adding the 2's complement of the actual subtrahend. The subtraction is again indicated by the sign bit of the angle accumulator. A single bit of state is stored at the adder to realize the carry chain which at the same time requires the LSB to be fed in first. The shift-by- i operation can be realized by reading the bit $i - 1$ from its right end in the serial shift registers. A multiplexer can be used to change position according to the current iteration. The initial values x_0 , y_0 and z_0 are fed into the array at the left end of the serial-in serial-out register and as the data enters the adder component the multiplexer at the input switch and map back the results of the bit-serial adder into the registers. The constant LUT is implemented as a multiplexer with hardwired choices. When all iterations are passed the input multiplexers switch again and initial values enter the bit-serial CORDIC processor as the computed sine values exit. The performance is constrained by the use of multiplexers for the shift operation and even more for the constant LUT. The latter could be replaced by a RAM or serial ROM where values are read by simply incrementing the memory's address.

6.2 Proposed Approach

The problem given was to develop a bit serial CORDIC which has no inter iteration dependency and which uses a small ROM. Keeping in view the problem in this thesis the Bit serial modified CORDIC DDFS based generator is simulated using ModelSim ,then the implementation of Bit Serial CORDIC DDFS based generators is done on XILINX Vertex II Xc2v6000 (6bf957). The results are verified by test bench generated by the FPGA and MATLAB as well. This thesis shows that CORDIC is available for use in FPGA based computing machines, which are the likely basis for the next generation DSP systems. It can be concluded that the designed RTL model for bit serial modified CORDIC is accurate and can be used in real time applications.

The simplicity of the proposed bit serial design is apparent from its architecture, it is a novel idea and can be extended for any value of angle theta, for more precision we just need to increase the iterations and keeping the first five iterations in the Rom. It can be seen that the Results of our proposed bit serial CORDIC are better than other bit serial CORDIC processors. The proposed architecture is area efficient as well and is best characterized due to its speed and simplicity of design.

6.2 Future Work

Future work in can be the bit width optimization of CORDIC algorithm as a first step and then as a second step enhancing this optimization against the Black Boxes present in the FPGA's. The modern FPGAs are having optimization boxes in them for high speed applications , if we some how know the internal detail design of FPGA then by making our design to fully map onto the FPGA architecture can enhance the performance many folds.

References

- [1] Volder J. E., .The CORDIC trigonometric computing technique, IRE Trans. Electronic Computing, Volume EC-8, pp 330 - 334, 1959.
- [2] Lindlbauer N., www.cnmat.berkeley.edu/~norbert/CORDIC/node3.html
- [3] Avion J.C., <http://www.argo.es/~jcea/artic/CORDIC.htm>
- [4] Qian M., .Application of CORDIC Algorithm to Neural Networks VLSI Design., IMACS Multiconference on .Computational Engineering in Systems Applications (CESA)., Beijing, China, October 4-6, 2006.
- [5] Lin C. H. and Wu A. Y., .Mixed-Scaling-Rotation CORDIC (MSR-CORDIC) Algorithm and Architecture for High-Performance Vector Rotational DSP Applications., Volume 52, pp 2385- 2398, November 2005
- [6] Walther J.S.A., .Unified algorithm for elementary functions., Spring Joint Computer Conference, pp 379 - 385, Atlantic city, 1971.
- [7] Kolk K. J. V., Deprettere E.F.A. and Lee J. A., . A Floating Point Vectoring Algorithm Based on Fast Rotations., Journal of VLSI Signal Processing, Volume 25, pp 125.139, Kluwer Academic Publishers, Netherlands, 2000.
- [8] Antelo E., Lang T. and Bruguera J. D., .Very-High Radix CORDIC Rotation Based on Selection by Rounding., Journal of VLSI Signal Processing, Vol.25, 141.153, Kluwer Academic Publishers, Netherlands, 2000.
- [9] Delosme M. J., Lau C. Y. and Hsiao S. F., .Redundant Constant-Factor Implementation of Multi-Dimensional CORDIC and Its Application to Complex SVD., Journal of VLSI Signal Processing, Volume 25, pp 155.166, Kluwer Academic Publishers, Netherlands, 2000.
- [10] Choi J. H., Kwak J. H. and Swartzlander, Journal of VLSI Signal Processing, Volume 25, Kluwer Academic Publishers, Netherlands, 2000.
- [11] Roads C., .The Computer Music Tutorial., MIT Press, Cambridge, 1995.

- [12] Rhea T., .The Evolution of electronic musical instruments., PhD thesis, Nashville: George Peabody College for Teachers, 1972.
- [13] Goodwin M., .Frequency-domain analysis-synthesis of musical sounds. Master's thesis, CNMAT and Department of Electrical Engineering and Computer Science, UCB, 1994.
- [14] Muller J. M., .Elementary Functions - Algorithms and Implementation.Birkhauser Boston, New York, 1997.
- [15] H. M. Ahmed, J. M. Delosme, and M. Morf, .Highly concurrent com- Comput. Mag., Volume 15, no. 1, pp. 65-82, Jan. 1982.
- [16] Parhami B., .Computer Arithmetic . Algorithms and hardware designs,. Oxford University Press, New York, 2000.
- [17] Considine V., .CORDIC trigonometric function generator for DSP., IEEE- 89th, International Conference on Acoustics, Speech and Signal Processing, pp 2381 - 2384, Glasgow, Scotland, 1989.
- [18] Andraka R.A., .Survey of CORDIC algorithms for FPGA based computers., Proceedings of the 1998 ACM/SIGDA sixth international symposium on FPGAs, pp 191-200, Monterey, California, Feb.22-24, 1998.
- [19] <http://www.dspguru.com/info/faqs/CORDIC.htm>
- [20] www.xilinx.com/partinfo/#4000.pdf
- [21] Troya A., Maharatna K., Krstic M., Crass E., Kraemer R.,.OFDM synchronizer Implementation for an IEEE 802.11a Compliant Modem", Proc. IASTED International Conference on Wireless and Optical Communications, Banff, Canada,July 2002.
- [22] Andraka R., .Building a high performance bit serial processor in an FPGA., On-Chip System Design Conference, North Kingstown, 1996.
- [23] <http://comparch.doc.ic.ac.uk/publications/files/osk00jvlsisp.ps>.
- [24] Krsti M., Troyu A., Muharatnu K. and Grass E., .Optimized low-power synchronizer design for the IEEE 802.11a standard., Frankfurt (Oder), Germany, 2003.

[25] Proakis J. G., Manolakis D. G., .Digital signal processing principles, algorithms and applications., Prentice Hall, Delhi, 2008.