REDUNDANT NUMBER SYSTEM BASED CORDIC FOR FAST FOURIER TRANSFORM

Submitted in Partial Fulfillment of the Requirements for Semester III- IV of

MASTER OF TECHNOLOGY
IN
ELECTRONICS & COMMUNICATION ENGINEERING

(VLSI Design)

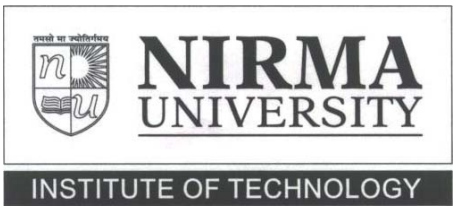
By

Vishal Bamboria (03MEC02)

Under the Guidance of

Prof. N P Gajjar
EC. Department,
Nirma University
Ahmedabad

Mr. Sujit Bhattacharya
Project Manager ASIC,
Scientist/ Engineer-'SF'
SAC (ISRO), Ahmedabad



Department of Electronics & Communication Engineering

INSTITUTE OF TECHNOLOGY,

NIRMA UNIVERSITY OF SCIENCE & TECHNOLOGY,

AHMEDABAD 382 481

Year 2004-05

Certificate

This is to certify that the Major Project Report (Part-II) entitled **"Redundant Number System Based CORDIC for Fast Fourier Transform"** submitted **by Mr. Vishal Bamboria (03MEC02)** towards the partial fulfillment of the requirements for Semester III-IV of Master of Technology (Electronics & Communication Engg.) in the field of VLSI Design of Nirma University of Science and Technology is the record of work carried out by him under our supervision and guidance. The work submitted has in our opinion reached a level required for being accepted for examination. The results embodied in this major project work to the best of our knowledge have not been submitted to any other University or Institution for award of any degree or diploma.

Date:

Dr. M D Desai	**Prof. N P Gajjar**	**Dr. H V Trivedi**
HOD Dep. of Electronics and Communication Engg.	Project Guide	Director

ABSTRACT

Digital Signal Processing (DSP) techniques find application from home appliances to Satellite applications. DSP involves complex mathematical computations and therefore, there is a need for high-speed mathematical processor. Trigonometric function calculation is one of the primary tasks performed in DSP applications. For long time microprocessor-based systems have been used to perform this task. Software algorithms used by the processor do not meet the highly demanding needs of all DSP tasks. Using hardware systems to perform these DSP task is a competent solution to this problem. Field Programmable Gate Arrays (FPGAs) are often used as co-processor to perform all the high speed tasks that can not be achieved by microprocessors. FPGAs are chosen because they are on-site programmable and are highly suitable for hardware implementations. The software solutions adopted by the microprocessor to implement trigonometric functions are computation intensive. They do not suit hardware platform because they need complex circuit to perform the mathematical operations. Among the existing hardware algorithm **CO**-ordinate **R**otation **DI**gital **C**omputer (CORDIC) is widely adopted because of its simplicity and speed efficiency.

CORDIC is one such iterative hardware efficient algorithm that performs high seed mathematical operations in linear, circular and hyperbolic coordinate system and computes various trigonometric, logarithmic and transcendental functions fast with high degree of accuracy for DSP applications.

The drawback of conventional CORDIC implementation, based on ripple carry adders or subtractions, is the internal carry propagation delay. To enhance the performance of CORDIC Redundant Arithmetic has been proposed. This arithmetic, due to its carry-free property, avoids the carry propagation from the LSB to MSB.

Nevertheless, its use involves several difficulties. It is not possible to detect the sign of a redundant number without inspecting all the digits which requires a propagation from the MSB to the LSB. Hence, the decision criteria is chosen according to several most significant digits. Since, not all digits are examined there is the possibility that the sign is not determined. In such a case either the digit set {-1,0,1} can be chosen leading to a non-constant Scale factor or an arbitrary rotation has to be performed. In the first case, the scale

factor has to be computed in parallel to the CORDIC iterations to ensure the convergence. In order to solve the problems of the redundant arithmetic based CORDIC several methods have been proposed. All of them maintain the scale factor constant. They have been classified in the three groups. Group I, which are based on an estimation of the sign like Double Rotation CORDIC, Correcting Rotation CORDIC, House holder CORDIC. Group II, Differential CORDIC algorithm. Group III, which are based on the pre-computation of the directions of the micro-rotation.

The CORDIC algorithm is used for various applications like the Digital Chirp Generation, Fast Fourier Transform, Digital Filtering, etc. The main speed limiting operation in FFT is complex multiplication. Complex multiplication is one of the most time-critical and area-consuming operations in a FPGA implementation. The CORDIC in FFT architecture eliminates the need of complex multiplier and the need to store the Sine/Cosine factors in the twiddle factor ROM.

The CORDIC based FFT architectures are well suited for the FPGAs where no memories are available to store the Sine/Cosine terms (like Actel family). To incorporate CORDIC a new butterfly is designed where required multiplication terms are given as input vector to the CORDIC block as discussed in the thesis.

This project work involves design, VHDL implementation and FPGA prototyping of Redundant arithmetic blocks, CORDIC processor, Predicting CORDIC Processor and its applications to Digital Chirp Generator and FFT. The project work has been carried out in several stages. It started with the design of the architecture and its bit-accurate implementation on MATLAB. The RTL design was carried out in VHDL. The designs were simulated in the Modelsim, synthesized using the Xilinx synthesis Tool (XST), and finally implemented using placing and routing Tool. Implemented design was downloaded on the Xilinx Vertex board along with Chipscope. All the results from MATLAB simulation and logical synthesis have been discussed.

CONTENTS

Page No.

ACKNOWLEDGEMENT (vi)
List of Figures and Tables (vii)
ABBREVIATION (x)

CHAPTER 1: Introduction 1
1.1 Overview 1
CHAPTER 2 : Review Of Literature 5
CHAPTER 3 : System Theory 7
3.1 Redundant Number System 7
3.2 Redundant Binary Representation 7
3.3 Carry Propagation Free Addition 10
3.4 Redundant to Binary Conversion
3.5 CORDIC Theory 14
3.6 CORDIC Iteration With Redundant Number 23
3.7 Predicting CORDIC Algorithm 33
3.8 Digital Chirp Generator 39
3.9 Fast Fourier Transform 47

CHAPTER 4 : System Design 82

4.1 Binary To Redundant Converter 82
4.2 Carry Free Adder 83
4.3 Carry Selector 85
4.4 S & D Selector 86
4.5 Redundant To Binary Converter 87
4.6 CORDIC implementation in an FPGA 109
4.6.1 Redundant CORDIC Implementation 115
4.7 Predicting CORDIC Processor 121

CHAPTER 5 : Conclusion and Future Work 138

REFERENCES 140
Appendix A 142
Appendix B 144
Appendix C 146

ACKNOWLEDGEMENT

Firstly I would like to express my profound gratitude to my project guide, **Mr. Sujit Bhattacharya**, Project Manager ASIC, Space Applications Center (SAC-ISRO), Ahmedabad for his outstanding support and guidance during my stay at SAC (ISRO). Mr. Sujit Bhattacharya is a fascinating tutor and a thoughtful mentor; always ready to share his expertise with his juniors. He is also a thorough, brilliant researcher and I benefited greatly from working under his guidance. His encouragement and friendship have been invaluable throughout my work at SAC (ISRO).

I also take this opportunity to thank, Mr. R. J.K. Jain, Head, HRDD who gave us wonderful opportunity to work in the nation's most premier and esteemed research institute.

Again my sincere and deep gratefulness goes to my project guide, Prof, N.P. Gajjar for his kind support and guidance throughout my project work. He is a wonderful teacher and an enthralling guide, spreading his knowledge and energy omni directionally. His academic excellence continues to be a source of inspiration, but beyond that I am especially grateful for his limitless patience and fatherly support during trying times.

I would also like to thank the professors at the Institute of technology, Nirma University, Ahmedabad especially Dr. N.M. Devashree, Prof. D.K. Kothari, Prof. A.K. Nigam, Prof. S.V. Pradhan, Prof. D.P. Dave, Prof. Y.N. Trivedi, Prof. Dhaval Pujara, and Prof. T.P. Singh for providing me with very good foundations which have proved helpful throughout my studies and project work.

Last but not the least, I express out sincere thanks to all our colleagues, friends for their continuous support and constant encouragement for friendship, patience and collegial help, which was so essential during my project work. Lastly I would I like to thank each and every person who has helped me directly or indirectly in accomplishing my project.

Finally, I would like to thank my parents for their constant love and support and for providing me with the opportunity and the encouragement to pursue my goals.

List of Figures

Figure 2.2.1 : Example of carry propagation free addition 10.
Figure2.2.1 : Conventional Redundant binary to binary converter 11
Figure2.2.2 : Logic new serial mode binary converter 12
Figure2.2.3 : Look ahead mode Redundant binary to binary converter 14
Figure 2.3.1 : Rotation in linear coordinate system. 17
Figure 2.3.2 : Vector rotation 18
Figure 2..3.3 : Rotating a vector using micro rotations 18
Figure 2.3.4 : Rotation in Hyperbolic coordinate system 22.
Figure 2.4 .11: Diagram of branching CORDIC. 25
Figure 2.4.12 : Architecture of a branching CORDIC processor 25
Figure 2.4.1 : CORDIC dependence graph for rotation mode and vectoring mode. 28
Figure 2.4.2 : Unfolded (pipelined) CORDIC signal Flow graph. 29
Figure 2.4.3 : Folded (recursive) CORDIC signal flow graph. 30
Figure 2.4.5 : Dependence graph for classical vector rotation 32
Figure 2.6.1 Traditional DCG architecture 39
Figure 2.6.4 : Modelsim result of Chirp generator.
Figure 2.6.5 : RTL view of Chirp generator. 44
Figure 2.7.1 : R2MDC(N=16) 48
Figure 2.7.2 : R2SDF (N=16) 48
Figure 2.7.3 : R4SDF 49
Figure 2.7.4 : R4SDC 49
Figure 2.7.5 : Mapping of quadrant 55
Figure 2.7.6 : CORDIC Butterfly unit 57
Figure 8.1 : Binary to redundant Converter 82
Figure 8.2 : RTL view of 1 bit adder(CARADDER) 83
Figure 8.3 : 8 bit adder (ADDER8) 84
Figure 8.4 : Carry selector(Carry_in) 85
Figure 8.5 : SDBREAKER86
Figure 8.6 : Redundant to binary conversion 87
Figure 7.1 : Binary to Redundant Converter Simulation 88
Figure 7.2 : 10 Bit Carry free adder 89
Figure 7.3 : Carry Selector 89
Figure 7.4 : S & D Selector(SDBREAKER) 90
Figure 7.5 : Redundant to Binary Converter(RENTOBIN8) 90
Figure 7.6 : TOP Level Entity(TOP) 91
Figure 6.1 : RTL view of 1 bit adder 92
Figure 6.1.1 : Gate or/ nor 92
Figure 6.1.2 : Gate or/nor 92
Figure 6.2 : RTL view of 8 bit ADDER 93

Figure 6.3.1 : Binary to redundant converter 95
Figure 6.4 : RTL view of Carry selector (Carry_in) 97
Figure 6.3 : Top level redundant to binary converter 98
Figure 6.4.1 : Internal of redundant to binary converter 98
Figure 6.5 : RTL of S & D selector (SDBREAKER) 100
Figure 6.6 : RTL of Top 101
Figure 5.1 : Sine wave(MATLAB simulation). 104
Figure 5.7 : Angle mapping for the predicting CORDIC 107
Figure 5.8 : Simulation result of predicting CORDIC 107
Figure 5.9 : Chirp values vs samples 108
Figure 5.10 : CORDIC Cosine output 108
Figure 1 : Iterative CORDIC structure. 110
Figure 2 : Bit serial iterative CORDIC 111
Figure 3 : Iterative bit serial CORDIC for Xilinx 4000E. 113
Figure 4 : Unrolled CORDIC Processor.113
Figure 5 : Two iterations of bit serial CORDIC pipeline 114
Figure 6 : Detail of pipelined bit serial adder/subtractor in Amtel 114
Figure3.1 : RTL view of TOP of CORDIC processor 118
Figure3.2 : Modelsim view of Sine and Cosine generation .118
Figure3.3 : Modelsim view of pure Sine wave 119
Figure4.1 : Proposed CORDIC architecture 116
Figure4.2 : Post processor RTL view. 116
Figure4.3 : Preprocessor RTL view 117
Figure4.4 : RTL View of CORDIC unrolled 117
Figure 4.5 : Single CORDIC Stage 119
Figure 4.6 : RTL view of Post Processor 120
Figure 9.1 : Architecture of predicting CORDIC processor 121
Figure 9.2 : Quadrant symmetry mapping of angle 123
Figure 9.3 : Shows the $\pi/4$ multiplier 124
Figure 9.6 : RTL view of Pi/4Multiplier
Figure 9.4 : Modelsim results of Sign predictor 125
Figure 9.5 : Modelsim results of predicting CORDIC 125
Figure 2.6.6 : Chirp Modelsim results
Figure 10.1 : Chipscope result of Top Redundant conversion chain 142
Figure 10.2 : Chipscope result of Top Redundant conversion chain 142
Figure 10.3 : Chipscope result of Top Redundant conversion chain 143
Figure 11.1 : Chipscope result of Sine wave by CORDIC 144
Figure 11.2 : Chipscope result of Sine wave by CORDIC 144
Figure 11.3 : Chipscope result of Cosine wave by CORDIC 145
Figure 11.4 : Chipscope result of Cosine wave by CORDIC 145
Figure 12.1 : Chipscope result of Sine wave by Chirp generator 146
Figure 12 2 : Chipscope result of Sine wave by Chirp generator 146
Figure 12 3 : Chipscope result of Cosine wave by Chirp generator 147

List of Table

Table 2.1 : Computation rules for carry propagation free addition.
Table 2.3 : For sign selection by examine 3 digit positions
Table 2.6.1 : Control signal for the out put stage
Table 2.6 : Angle ROM of CORDIC in chirp generator
Table 2.4 : For sign selection by examine 3 digit positions
Table 2.4.1 : Architecture properties for the CORDIC architecture
Table 2.6.3 : Performance summery of chirp generator
Table 3.2 : Architectural properties for three CORDIC architectures
Table 8.1 : Angle ROM of CORDIC
Table 8.2 : Performance summary

ABBREVIATIONS

CPA	:	Carry Propagation Free Addition
DSP	:	Digital Signal Processing
CORDIC	:	Coordinate Rotation Digital Computer
DCG	:	Digital Chirp Generator
FPGA	:	Field Programmable Gate Array
DFT	:	Discrete Fourier Transform
FFT	:	Fast Fourier Transform
LUT	:	Look Up Table
MUX	:	Multiplexer
GSD	:	Generalized Signed Digit Number
CS	:	Carry Save
BSD	:	Binary Signed Digit
RBC	:	Redundant to Binary Conversion
RB	:	Redundant Binary
BBR	:	Binary To Bipolar Recoding
MAR	:	Micro Rotation Angle Recoding
HDL	:	Hardware Description Language
DCT	:	Discrete Cosine Transform
FIR	:	Finite Impulse Response Filter
RAM	:	Random Access Memory
R2MDC	:	Radix-2 Multi Path Delay Commutator
R2SDF	:	Radix-2 Single Path Delay Feedback
R4SDF	:	Radix-4 Single Path Delay Feedback
R4SDC	:	Radix-4 Single Path Delay Commutator
ROM	:	Read Only Memory
RTL	:	Register Transfer Logic
VHDL	:	VHSIC Hardware Description Language
XST	:	Xilinx Synthesis Tool

CHAPTER 1

INTRODUCTION

The high capability and performance that FPGAs have achieved in last few years allow them to accelerate DSP tasks. FPGA devices have been used for implementing Custom DSPs from the beginning of the past decade. The FPGA devices have beneficed from the improvements in VLSI deep sub-micron technology, leading to higher speed and capability as well as low power consumption. On the other hand digital signal processing (DSP) algorithms exhibit an increasing need for the efficient implementation of complex arithmetic operations. The computation of trigonometric functions, coordinate transformations or rotations of complex valued phasors is almost naturally involved with modern DSP algorithms. Popular application examples are algorithms used in digital communication technology and in adaptive signal processing. While in digital communications, the straightforward evaluation of the cited functions is important, numerous matrix based adaptive signal processing algorithms require the solution of systems of linear equations, QR factorization or the computation of eigen values, eigenvectors or singular values. All these tasks can be efficiently implemented using processing elements performing vector rotations. The Coordinate Rotation Digital Computer algorithm (CORDIC) offers the opportunity to calculate all the desired functions in a rather simple and elegant way.

The Coordinate Rotational Digital Computer (CORDIC) algorithm is a well-known iterative technique to perform various basic arithmetic operations. The algorithm is very attractive for hardware implementation because it uses only elementary shift-and-add steps to perform vector rotation in a two-dimensional (2-D) plane. Hence, the CORDIC algorithm can be applied to many DSP applications where rotation-based arithmetic functions are heavily utilized, such as linear system solver, digital lattice filter, singular value problems and the fast Fourier transformation (FFT). However, the major disadvantage of the CORDIC algorithm is its slow computational speed.

The Coordinate Rotational Digital Computer (CORDIC) was introduced in 1959 by Jack Volder, for the computation of trigonometric functions; multiplication, division and data type conversion, and later on generalized to hyperbolic functions by Walther. Two basic CORDIC modes are known leading to the computation of different functions, the rotation mode and the vectoring mode. For both modes the algorithm can be realized as an iterative sequence of additions/ subtractions and shift operations, which are rotations by a fixed rotation angle (sometimes called micro rotation) but with variable rotation direction. 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. The rotated vector is also scaled making a scale factor correction necessary. The CORDIC algorithms generally produce one additional bit of accuracy for each iteration.

Several CORDIC processors have been designed and implemented onto VLSI chips. However, the CORDIC algorithm is relatively slow because each iteration requires carry propagate adders. Recently, Ercegovac and Lang have proposed using redundant signed-digit (SD) adders to replace the conventional binary adders in order to reduce the inherent carry delay . But, the redundant CORDIC destroys the nice property of constant scaling factor due to a different sign selection rule, and hence the multiplication of the scaling factor requires additional complicated hardware. Several approaches have been proposed to overcome the problems. However, all these methods required either extra correcting CORDIC iterations or additional scaling iterations.

An entirely new approach is based on parallelization of the original CORDIC algorithm by predicting all the rotation directions directly from the binary bits of the initial input angle. Unlike previous approaches that require complicated circuits or exponentially increased ROM, this algorithm provides a relatively simple prediction scheme through an efficient angle recording. Utilizing the predicted rotation directions to design an efficient multi operand carry-save addition structure also reduces the critical path delay.

With use of above algorithm there is no need to perform the Z iterations thus proved to be efficient in terms of area and as well as speed. Thus one can use carry-free arithmetic to replace the conventional binary adder with no further increase in area.

This efficient Sine/Cosine generator is used for many applications like FFTs and DCG (Digital Chirp Generator). A DCG system consists of a phase accumulator, frequency accumulator and a sine/cosine generator, Simple modifications to the phase generation circuitry produces synthesized chirps useful in radar and electronic warfare systems and in implementation of continuous-phase modulators (e.g., GMSK). The details are discussed in the latter section.

One can replace the complex multiplier operation by shift and add CORDIC algorithm. Thus ,above Sine/Cosine generator is used to implement the Fast Fourier Transform. Fast Fourier Transform (FFT) is one of the most utilized operations in Digital Signal Processing and Communications. The FFT and its inverse transform-IFFT are a key component in modern communication systems. Application Specific Integrated Circuit (ASIC) approaches have been used to achieve the high performance demands, which software

or general-purpose DSP implementations fail to deliver. Recently FPGAs has become a valid alternative as the technology has matured greatly. Nowadays FPGAs play an important role in many areas due to their direct hardware solution performance as well as their inherent reprogram ability feature. Using FPGAs for FFT processing has now become feasible in real-time applications. Development of the FFT in hardware is usually categorized into that of high throughput and that of low power.

CHAPTER 2

LITERATURE REVIEW

The basic design flow of the first module is :

- The input binary number is first converted to redundant number.
- The resultant RB numbers are added/subtracted using carry free adder/Subtractor to perform addition & subtraction.
- Finally the redundant number is converted back to binary number.

Most of the literature that deals with redundant number system is referred form:

[1] "Generalized Sign Digit number system" a unifying frame work for redundant number representations Behrooz Parhami , IEEE Transaction on computers vol:39, No.1,Jan 1990.

This paper provides enough background for redundant numbers and discussed so called signed digit numbers with radix r≥3 with digit set {-α, 0, α} where 'α 'is an arbitrary integer. Such a number system representation systems possess sufficient redundancy to allow for the annihilation of carry or borrow chains and hence result in fast propagation free addition and subtraction.

The stuff for the carry-free addition referred from :

[2] Carry Free addition of recoded binary sign digit numbers

Behrooz Parhami, IEEE Transaction on computers vol:37, No.11,Nov 1988.

and

[3] "High speed VLSI multiplication algorithm with a redundant binary addition tree".Naofumi takagi, Hiroto yasuura, IEEE Transaction on computers vol:C34, No.9,Sep 1985.

Now for converting redundant number back to binary number another IEEE paper

[4]"An efficient redundant Binary to Binary number converter

Sung Ming Yen, Chi-Sung laih, IEEE journal of solid-state circuit vol: 27,No.1, Jan 1992.

In this paper all the available technique for converting redundant number back to binary are discussed ,out of which the lookahead mode conversion algorithm is used because of its high speed conversion .

For CORDIC implementation there are large numbers of papers available out of which few IEEE papers are referred as per our requirement. Some of them are

[5]" The CORDIC Algorithm: New Results for Fast VLSI Implementation"

Jean Duprat,Jean Michel Muller,IEEE Transaction on computers vol:42, No.2,Feb 1993

[6]" Design of a unified arithmetic processor based on redundant constant factor CORDIC with merged scaling operation."

S.F.Hsiau,C.Y.Lau ,IEE, proc, comput ,digit Tech vol :147,No.4,July 2000.

Few papers on FFT with CORDIC are referred one of them is

[7]"FPGA realization of a CORDIC based FFT processor for biomedical signal processing." IITKh paper.

The stuff for the Predicting CORDIC algorithm is referred from the

[8] A Memory Efficient and High- speed Sine/Cosine Generator based on Parallel CORDIC Rotations by Shen-Fu,Yu-Hen Hu, Tso-Bing Juang ,IEEE signal processing letters Vol.11 Feb 2004 .

CHAPTER 3

SYSTEM THEORY

3.1 REDUNDANT NUMBER REPRESENTATION

A scheme using redundant number representation for fast multiplication was introduced in the late 1950's by Aviziens.A unified description for redundant number systems was given by Parhami who defined Generalized Signed Digit (GSD) number systems. A GSD number system contains the digit set $\{-\alpha,-\alpha+1,\ldots\ldots\ldots,\beta-1, \beta\}$ with $0\leq\beta,\alpha$ and $\beta+\alpha+1>r$ with r being the radix of the number system. Every suitable definition of α and β leads to a different redundant number system. The value X of a W digit integer GSD number is given by:
After N iterations, the accumulated rotation angle is

$$X = \sum_{k=0}^{W-1} r^k * x_k \ , \ x_k = \{-\alpha, -\alpha+1,\ldots,\beta-1, \beta\} \qquad (2.1)$$

An important subclass are number systems with $\alpha+\beta = r$, which are called minimal redundant", since $\alpha + \beta = r-1$ corresponds already to a conventional number system. The well-known Carry-Save (CS) number system is defined by $\alpha = 0$; $\beta = 2$;
r = 2. CS numbers are very attractive for VLSI implementation since the basic building block for arithmetic operations is a simple full adder
With $\alpha= 1$; $\beta = 1$; r = 2 the well-known Binary Signed Digit (BSD) number system results. BSD operations can be implemented using the same basic structures as for CS operations
An important advantage of CS numbers is the very simple and fast implementation of the addition operation.

3.2 REDUNDANT BINARY REPRESENTATION:

Redundant Binary Representation:

The redundant binary representation utilized is one of the SD representations proposed by Avizienis . It has a fixed radix 2 and a digit set $\{-1, 0, 1\}$ where –1 denotes –1. An n-digit redundant binary integer $y = [y_{n-1}\ldots.y_0]_{SD2}$ ($y_i \in \{-1, 0, 1\}$) has the value $\sum_{i=0}^{n-1} y_i \times 2^i$. It is similar to an unsigned binary integer except that y_i can be –1.

The redundant binary representation allows the existence of redundancy. There are several ways to represent an integer in the redundant binary representation. For example,

$[0101]_{SD2}$, $[011\text{-}1]_{SD2}$, $[1\text{-}101]_{SD2}$, $[1\text{-}11\text{-}1]_{SD2}$, and $[10\text{-}1\text{-}1]_{SD2}$ all represent "5." (However, "0" is uniquely represented). Owing to the redundancy. We can perform carry-propagation free addition, and therefore, parallel addition of two redundant binary numbers can be performed in a constant time independent of the word length of operands, as will be mentioned in the next subsection.

The negation of a redundant binary number is directly derived by changing the signs of all nonzero digits in the number. Since this computation can be performed in parallel for all digits, it requires a constant computation time independent of the word length of the number.

3.3 Carry-Propagation-Free Addition:

Carry-propagation-free addition is performed in two steps. In the first step, we determine the intermediate carry c_i ($\in\{-1, 0, 1\}$ and the intermediate sum digit s_i($\in\{-1, 0, 1\}$) at each position, satisfying the equation $x_i + y_i = 2c_i + s_i$, where x_i and y_i are the augend and addend digits, respectively. In the second step, we obtain the sum digit z_I($\in \{-1, 0, 1\}$) at each position by adding the intermediate sum digit s_i and the intermediate carry c_{i-1} from the next-lower-order position, without generating a carry.

In the first step, at each position, we determine c_i and s_i so that both s_i and c_{i-1} are not 1's, nor are they –1's. When one of x_i and y_i is 1 and the other is 0, we determine c_i and s_i as follows (note that both $[01]_{SD2}$ and $[1\text{-}1]_{SD2}$ represent "1").

1) If there is a possibility of a "1"as carry (a positive carry) from the next-lower-order position, we let $[c_i, s_i]$ be
   [1, -1].
2) If there is a possibility of a "–1" as carry (a negative carry) from the next-lower-order position, we let $[c_i, s_i]$ be [0, 1].
3) If there is no possibility of a carry from the next-lower-order position, we may let $[c_i, s_i]$ be either [1, -1] or [0, 1].

Similarly, when one of x_i and y_i is"–1" and the other is"0", we let $[c_i, s_i]$ be [0, -1] if there is a possibility of a 1-carry from the next-lower-order position, and let it be [-1, 1] if there is a possibility of a" –1" as carry. We can know the possibility of a carry from the next-lower-order position by examining the augend and the addend digits x_{i-1} and y_{i-1} at the next-lower-order position. When both x_{i-1} and y_{i-1} are 1's or one of them is "1" and the other is "0", there is a possibility of a "1"as carry. When both of them are –1's or one of them is "–1" and the

other is "0", there is a possibility of a "–1" as carry. In the other cases, there is no possibility of a carry. Therefore, c_i and s_i can be determined by examining x_i, y_i, x_{i-1} and y_{i-1}.

When we determine c_i and s_i as stated in the above, no carry is generated in the addition of s_i and c_{i-1} in the second step. Thus, each sum digit z_i can be computed from x_i, y_i,

Type	Augend Digit (x_i)	Addend Digit (y_i)	Digits at the next-lower-order position (x_{i-1}, y_{i-1})	Intermediate carry (c_i)	Intermediate Sum digit (s_i)
1	1	1	————	1	0
2	1	0	Both are nonnegative.	1	-1
	0	1	Otherwise.	0	1
3	0	0	————	0	0
4	1 -1	-1 1	————		
5	0	-1	Both are nonnegative.	0	-1
	-1	0	Otherwise.	-1	1
6	-1	-1	————	-1	0

TABLE 2.1
Computation rule for the first step in Carry-propagation free Addition

x_{i-1}, y_{i-2}, and y_{i-2}. Namely, z_i depends on only these six digits. This fact is the key to the high-speed computation.

Table-2.1 shows a computation rule in the first step. When one of x_i and y_i is 1 and the other is 0, we let $[c_i, s_i]$ be [1, -1] or [11] accordingly, as both x_{i-1} and y_{i-1} are nonnegative or not. When one of x_i and y_i is –1 and the other is 0, we let it be [0,-1] or [-1, 1] accordingly, as both x_{i-1} and y_{i-1} are non-negative or not. We assume that x_{-1} and y_{-1}, i.e., the augend and addend digits at the next-lower-order position of the least significant position are both 0's. Fig. 2.2.1 shows an example of carry-propagation-free addition in accordance with the rule. (Take notice of the computation at the second and the third least significant positions).

Thus, in the redundant binary number system, carry propagation can be eliminated from addition, and therefore, parallel addition of two numbers by a combinational circuit is performed in a constant time independent of the word length of operands. Namely, the depth of an n-digit redundant binary adder is a constant independent of n. The gate count of it is proportional to n.

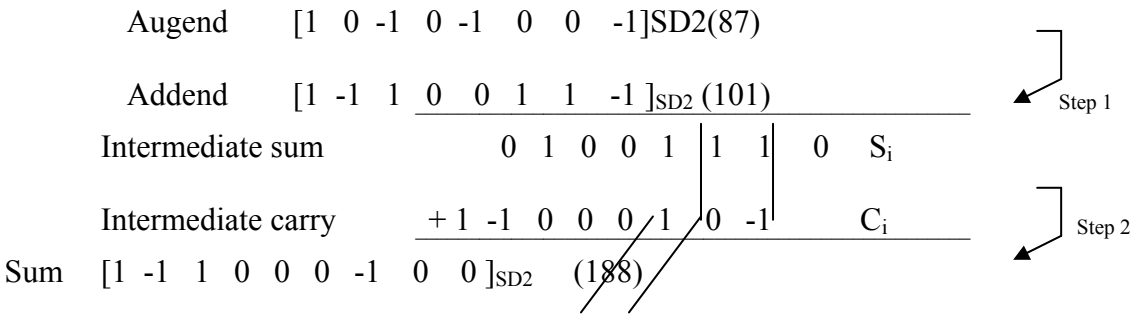


Fig 2.2. 1 Example of carry-propagation-free addition.

The addition method discussed here in an example. There are various other carry-propagation-free addition methods in the redundant binary number system.

3.4 REDUNDANT TO BINARY CONVERSION (RBC)

A scheme using redundant number representation for fast multiplication was introduced in the late 1950's by Aviziens.A computational algorithm using redundant binary number(RB) with digit set (-1,0,1) and Takagi proposed its application to multiplier & dividers. The RB adders are not popular earlier because its space complexity is twice to that of conventional adder. Recently this drawback is over come by vandemeulebroecke .

The redundant binary number(RB) representation properties:

1) Its carry propagation free addition.
2) It does not use 2's complement method to handle the negative number .
3) Require additional converter to convert the RB back to binary number.

Redundant binary number to binary number conversion:

Conversion of n digit Redundant binary number to 2's complement binary number is more complex.

There are three different approaches to perform this operation:

1) Conventional adder method.
2) Serial mode converter.
3) Look ahead mode converter.

Conventional adder method:

The RB to binary converter can be implemented by a binary adder serially as shown in the figure. In RB representation we use two binary bits(S,D) to represent an RBG digit X.In order to simplify the conversion circuit design , the representation of one digit is {(0,0),(0,1),(1,1)}={0,-1,1}.In order to guarantee correct conversion , an n digit RB number must be converted to (n+1)-bit 2's complement binary representation. Equation used is

$$Y = Xsd^{+} - Xsd^{-} \qquad (2.3.1)$$

Where Xsd^{+} and Xsd^{-} derived from input RB number Xsd.

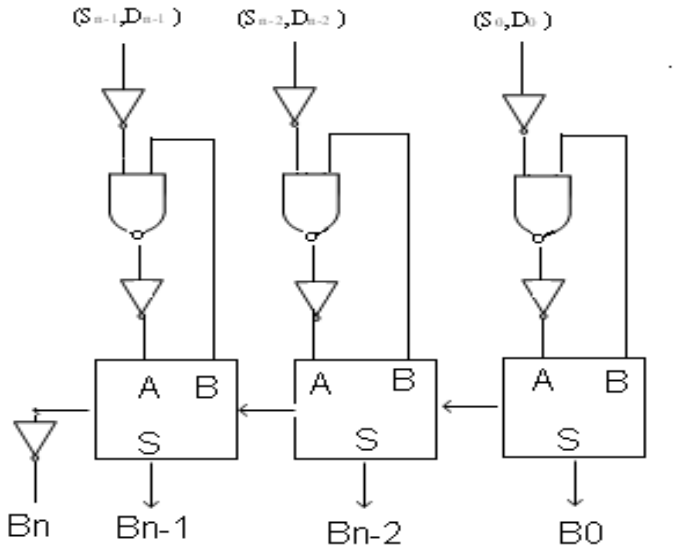


Fig.2.2.1 The conventional RB to binary convereter.

2 Serial mode converters:

In this type of converter the conversion process is achieved by selecting a new variable C_i in the ith digit position. This variable is defined as follows.

a) C_i =1 means that, for the current RB digit position I, there is at least one –1 right to this position and no +1 b/w the –1s and the current position.
b) C_i =0 otherwise.
c) C_0 =0.

The serial mode converter is shown in the figure .For each stage I, given the redundant digit (S_i, D_i) and the input variable C_i , we obtain the binary output Bi and the output variable C_{i+1}.The conversion rule is shown in the table

Table(2.3) Carry selection rule

Xi	S_i	D_i	C_i	B_i	C_{i+1}
0	0	0	1	1	1
0	0	0	0	0	0
1	0	1	1	0	0
1	0	1	0	1	0
-1	1	1	1	0	1
-1	1	1	0	1	1

From the conversion rules shown in the table we have the Boolean equations for B_i and $C_{i+1:}$

$$C_{i+1} = S_i + P_iC_i \qquad (2.3.2)$$

$$B_i = D_i \text{ xor } C_i \qquad (2.3.3)$$

Where P_i=not $D_{i.,}$ C_0=0.

The logic diagram in each stage I using only simple NAND gates is shown in the figure(2.2.2).Compare to conventional adder it takes less number of gates.

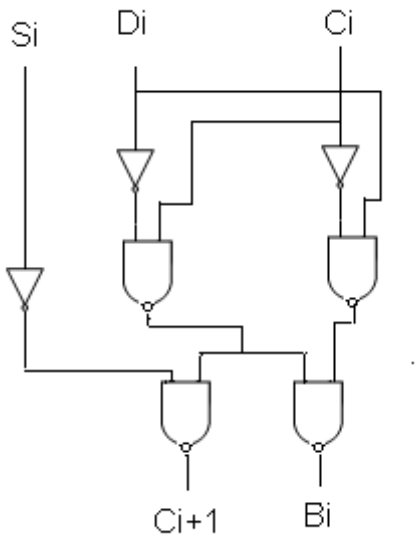


Fig 2.2.2 Logic diagram of new converter Serial mode.

Lookahead mode converter:

In serial mode converter the signal convert C_i is propagated from least significant position to most significant position digit by digit. To speed up the operation the concept of carry lookahead adder is used. The equations are:

$$C_1 = S_0 + P_0C_0 \quad (2.3.4)$$

$$C_2 = S_1 + P_1S_0 + P_1 P_0C_0 \quad (2.3.5)$$

$$C_3 = S_2 + P_2S_1 + P_1 P_2S_0 + P_2P_1 P_0C_0 \quad (2.3.6)$$

In general

$$C_{4i+4} = S_i\text{'} + Pi\text{'}C_{4i} \quad \text{where I= 0,1,2,3,4,5……} \quad (2.3.7)$$

As shown in the figure the look ahead mode converter requires only one NAND gate delay to generate the signals S and P. This method require less chip area and it also provide very fast conversion hence well suited for the application.

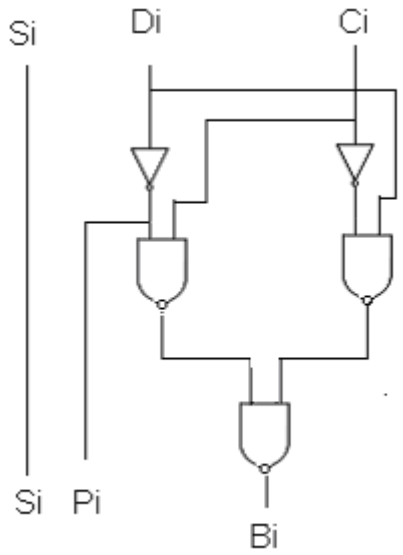


Fig.2.2.3 The logic diagram of lookahead mode converter

3.5 CORDIC THEORY

The basic concepts of the CORDIC computation is to decompose the desired rotation angle into the weighted sum of a set of predefined elementary rotation angles such that the rotation through each of them can be accomplished with simple shift-and-add operations.

All of the trigonometric functions can be computed of derived from functions using vector rotations. Vector rotation can also be used for polar conversions, vector magnitude, and as a building block in certain transforms such as the DFT and DCT. The CORDIC algorithm provides an iterative method of performing vector rotation by arbitrary angles using only shift and adds. The algorithm is derived from the general rotation transform:

$X' = x\cos(\varnothing) - y\sin(\varnothing)$ (2.3.1)

$Y' = y\cos(\varnothing) + x\sin(\varnothing)$ (2.3.2)

Which rotates a vector in Cartesian plane by the angle θ.. These can be rearranged so that:

$X' = \cos(\varnothing)[x - y\tan(\varnothing)]$

$Y' = \cos(\varnothing)[y - x\tan(\varnothing)]$

If the rotation angles are restricted so that $\tan(\varnothing) = \pm 2^{-i}$, the multiplication by the tangent term is reduced to simple shift operation. If the decision at each iteration, i, is which direction to rotate rather than whether or not to rotate, then the $\cos(\delta_i)$ term becomes a constant (because $\cos(\delta_i) = \cos(-\delta_i)$). The iterative rotation can be expressed as:

$$x_{i+1} = K_i [x_i - y_i . d_i . 2^{-i}] \quad (2.3.3)$$

$$Y_{i+1} = K_i [y_i + x_i . d_i . 2^{-i}] \quad (2.3.4)$$

Where,

$K_i = \cos(\tan^{-1}(2^{-i})) = 1/\sqrt{1 + 2^{-i}}$

$d_i = \pm 1$

Removing the scale constant from the iterative equation yields a shift-add algorithm for vector rotation. The product of the K's can be applied elsewhere in the system or treated as part of a system processing gain. That product approaches 0.6073
as the number of iterations goes to infinity. Therefore, the rotation algorithm has a gain, A of approximately 1.647. The exact gain depends on the number of iteration, and obeys the relation

$$A_n = \prod_n \sqrt{1 + 2^{-I}} \quad (2.3.5)$$

The angles of a composite rotation are uniquely defined by the sequence of the direction of the elementary rotations. That sequence of the directions of the elementary rotations. That sequence can be represented by a decision vector. The set of all possible decision vectors is an angular measurement system based on binary arctangents. Conversions between this angular system and any other can be accomplished using a look-up. A better conversion method uses an additional adder-subtractor that accumulates the elementary rotation angles at each iteration. The angle accumulator adds a third difference equation to the CORDIC algorithm:

$$Z_{i+1} = Z_i - d_i . \tan^{-1}(2^{-i}) \quad (2.3.6)$$

The CORDIC rotator is normally operated in two modes.

1. ROTATION MODE
2. VECTOR MODE

In the rotation mode, the angle accumulator is initialized with the desired rotation angle. The rotation decision at each iteration is made to diminish the magnitude of the residual angle in the angle accumulator. The decision at each iteration is therefore based on the sign of the residual angle after each step. Naturally, if the input angle is already expressed in the binary arctangent base, the angle accumulator may be eliminated. For rotation mode, the CORDIC equations are:

$$x_{i+1} = x_i - y_i . d_i .2^{-i}$$

$$y_{i+1} = y_i + x_i . d_i . 2^{-i}$$

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

Where : (2.3.7)

$d_i = -1$ if $z_i < 0$, $+1$ otherwise.

Which provides the following result :

$$x_n = A_n [x_0 \cos z_0 - y_0 \sin z_0]$$
$$y_n = A_n [y_0 \sin z_0 + x_0 \cos z_0]$$
$$z_n = 0$$
$$A_n = \prod_n \sqrt{1 + 2^{-i}}$$

In the vectoring mode, the CORDIC rotator rotates the input vector through whatever angle is necessary to align the result with the x-axis. The result of the vectoring operation is a rotation angle and the scaled magnitude of the original vector (the x component of the result). The vectoring function works by seeking to minimize the y component of the residual vector at each rotation. The sign of the residual y component is used to determine which direction to rotate next.

If the angle accumulator is initialized with zero, it will contain the traversed angle at the end of the iterations. In the vectoring mode, the CORDIC equations are:

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$
$$Z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})$$

Where (2.3.8)

$d_i = +1$ if $y_i < 0$, -1 otherwise.

Then :

$$x_n = A_n \sqrt{x_0^2 + y_0^2}$$
$$y_n = 0$$
$$z_n = z_0 + d_i \cdot \tan^{-1}(y_0 / x_0)$$
$$A_n = \prod_n \sqrt{1 + 2^{-i}}$$

The CORDIC rotation and vectoring algorithms as stated are limited to rotation angles between – pi / 2 and + pi / 2.

This limitation is due to the use of 2° for the tangent in first iteration. For composite rotation angles larger than pi /2, an additional rotation is required. This gives the correction iteration.

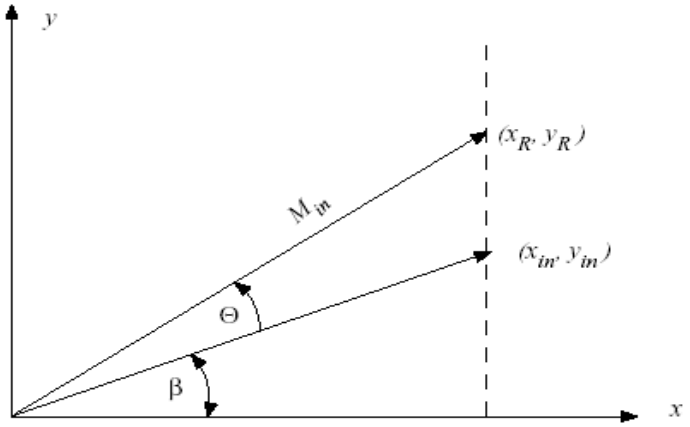


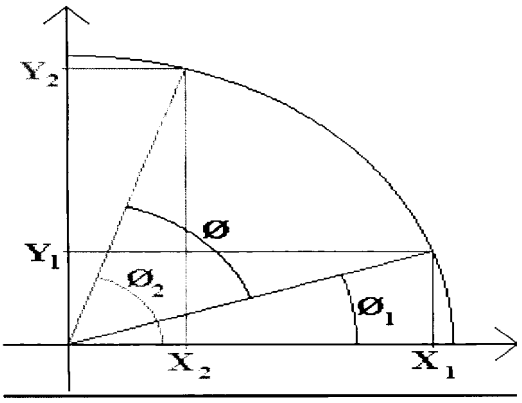
Figure (2.3.1): ROTATION IN LINEAR COORDINATE SYSTEM

The CORDIC rotator described is usable to compute several trigonometric functions directly and others indirectly. Judicious choice of initial values and modes permits direct computation of sine, cosine, arctangent, vector magnitude and transformations between polar and Cartesian coordinates.

CALCULATE THE SINE AND COSINE VALUE USING CORDIC ALGORITHM

CORDIC (COordinate Rotation Digital Calculation) finds the sine or cosine of an angle iteratively, using only simple math operations such as add, subtract, compare, shift, and table lookup.

VECTOR ROTATIONS



Fig(2.3.2)Vector rotation

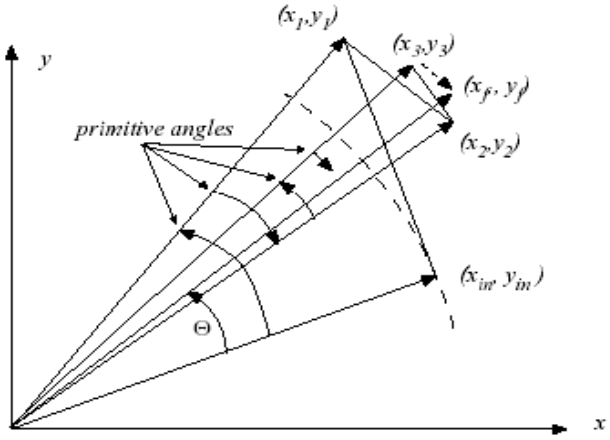


Fig (2.3.3): Rotating a vector using micro rotations

The diagonal blue line is angle ∅1 above the horizontal. The diagonal red line is the blue line rotated counter-clockwise by angle ∅. The new X and Y values are related to the old X and Y values as follows :

$$X2 = X1 * \cos(\varnothing) - y1 * \sin(\varnothing)$$

$$Y2 = X1 * \sin(\varnothing) + y1 * \cos(\varnothing) \qquad (2.3.9)$$

ITERATIVE ROTATION:

For CORDIC, the final angle ∅2 is the angle of interest; the angle whose sine or cosine we want to calculate. The initial angle ∅1 is set to a convenient value such as 0. Rather than rotating from ∅1 to ∅2 in one fell swoop, we move in steps. With careful choice of step values, the only math used is shifts and adds. The equations above can be re-written as:

$$X2 = \cos(\varnothing) * [X1 - y1 * \tan(\varnothing)]$$

$$Y2 = \cos(\varnothing) * [X1 * \tan(\varnothing) + Y1]$$

Values for ∅ are chosen such that tan (∅) is a fractional power of 2 :

tan(∅21) = 1/1 ∅21 = 45° cos(∅21) = 0.707107

tan(∅32) = ½ ∅32 = 26.5650° cos(∅32) = 0.894427

tan(∅43) = ¼ ∅43 = 14.0362° cos(∅43) = 0.970142

tan(∅54) = 1/8 ∅54 = 7.12502° cos(∅54) = 0.992278

tan(∅65) = 1/16 ∅65 = 3.57633° cos(∅65) = 0.998053

tan(∅76) = 1/32 ∅76 = 1.78991° cos(∅76) = 0.999512

tan(∅87) = 1/64 ∅87 = 0.895174° cos(∅87) = 0.999878

tan(∅98) = 1/128 ∅98 = 0.447614° cos(∅98) = 0.999969

This lets us replace the multiplication by tan(∅) with a simple, fast right-shift operation. But what about the cos(∅) factors ?

- First iteration (from X1, Y1 to X2, Y2) : rotate by angle∅21

  X2 = cos(∅21) * [X1 – Y1 * tan(∅21)]

  Y2 = cos(∅21) x [X1 * tan(∅21) + Y1]

- Second iteration (from X2, Y2 to X3, Y3) : rotate by
- X3 = cos(∅32) * [X2 – Y2 * tan(∅32)]

  Y3 = cos(∅32) * [X2 * tan(∅32) + Y2].

  angle ∅32.

- Plug in the values of X2 and Y2 from the first iteration :

  X3 = cos(∅32) * {cos(∅21) * [X1 – Y1 * tan(∅21)]

  - cos(∅21) * [X1 * tan(∅21) + y1] * tan(∅32)}

  = cos(∅32) * cos(∅21) * {[X1 – Y1 * tan(∅21)]

  [X1 * tan(∅21) + Y1] * tan(∅32)}

- The cosine factors fall out, to form an iterative product, i.e.

  Cos(∅21) * cos(∅32) * cos(∅43) … * cos(∅nn)

Expressing the ∅values in terms of inverse tangents gives the equivalent product series :-

$$\prod_{N=0}^{\infty} \frac{2^N}{\sqrt{1 + 2^{2N}}} = 0.607253 \qquad (2.3.10)$$

The value to which it converges, 0.607253, is the aggregate constant. We can ignore the cos(∅) terms and simply multiply by the aggregate constant before or after the iteration.

CORDIC EXTENSION:

A) Polar to Rectangular Transformation :

A logical extension to the sine and cosine computer is a polar to Cartesian coordinate transformer. The transformation from polar to Cartesian space is defined by :

$$x = r \cos\theta$$

$$y = r \sin\theta \qquad (2.3.11)$$

As pointed out above, the multiplication by the magnitude comes for free using the CORDIC rotator. The transformation is accomplished by selecting the rotation mode with x_0 = polar magnitude, z_0 = polar phase, and $y_0 = 0$. The vector result represents the polar input transformed to Cartesian space. The transform has a gain equal to the rotator gain, which needs to be accounted for somewhere in the system. If the gain is unacceptable, the polar magnitude may be multiplied by the reciprocal of the rotator gain before it is presented to the CORDIC rotator.

B) Arctangent :

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

$$Z_n = Z_n + \tan^{-1}(y_0/x_0).$$

C) Arcsine and Arccosine :

The Arcsine can be computed by starting with a unit vector on the positive x axis, then rotating it so that its y component is equal to the input argument. The arcsine is then the angle subtended to cause the y component of the rotated vector to match the argument. The decision function in this case is the result of a comparison between the input value and the y component of the rotated vector at each iteration :

$$x_{i+1} = x_i - y_i . d_i . 2^{-i}$$

$$y_{i+1} = y_i + x_i . d_i . 2^{-i}$$

$$z_{i+1} = z_i - d_i . \tan^{-1}(2^{-i}).$$

where,

$d_i = +1$ if $y_i < c$, -1 otherwise, and

c = input argument.

Rotation produces the following result :

$$x_n = \sqrt{(A_n \cdot x_o)^2 - c^2}$$

$$y_n = c$$

$$z_n = z_0 + \arcsin (c/An.x_0)$$

$$A_n = \prod_n \sqrt{1 + 2^{-2i}} \qquad (2.3.12)$$

The arcsine function as stated above returns correct angles for inputs $-1 < c/A_n x_0 < 1$, although the accuracy suffers as the input approaches ± 1 (the error increases rapidly for inputs larger than about 0.98). This loss of accuracy is due to the gain of the rotator. For angles near the y axis, the rotator gain causes the rotated vector to be shorter than the reference (input), so the decisions are made improperly. The gain problems can be corrected using a "double iteration algorithm" at the cost of an increase in complexity.

The Arccosine computation is similar, except the difference between the x component and the input is used as the decision function. Without modification, the arccosine algorithm works only for inputs less than $1/A_n$, making the double iteration algorithm a necessity. The Arccosine could also be computed by using the arcsine function and subtracting $\pi/2$ from the result, followed by an angular reduction if the result is in the fourth quadrant.

D) Extension to Hyperbolic Functions :

The close relationship between the trigonometric and hyperbolic functions suggests the same architecture can be used to compute the hyperbolic functions. While, there is early mention of using the CORDIC structure for hyperbolic coordinate transforms, the first description of the algorithm is that by Walther. The CORDIC equations for hyperbolic rotations are derived using the same manipulations as those used to derive the rotation in the circular coordinate system. For rotation mode these are :

$$x_{i+1} = x_i + y_i . d_i . 2^{-i}$$

$$y_{i+1} = y_i + x_i . d_i . 2^{-i}$$

$$z_{i+1} = z_i - d_i . \tanh^{-1}(2^{-i}).$$

where,

$d_i = -1$ if $z_i < 0$, $+1$ otherwise.

Then :

$$x_n = A_n[x_0 \cosh z_0 + y_0 \sinh z_0]$$

$$y_n = A_n[y_0 \cosh z_0 + x_0 \sinh z_0]$$

$$z_n = 0$$

$$A_n = \prod_n \sqrt{1-2^{-2i}} = 0.80 \qquad (2.3.13)$$

In vectoring mode ($d_i = +1$ if $y_i < 0$, -1 otherwise) the rotation

produces :

$$x_n = A_n \sqrt{x_0^2 - y_0^2}$$

$$y_n = 0$$

$$z_n = z_0 + \tanh^{-1}[y_0 / x_0]$$

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

The elemental rotations in the hyperbolic coordinate system do not coverage. However, it can be shown that convergence is achieved if certain iterations (I = 4, 13, 40, …, k, 3k + 1, …) are repeated.

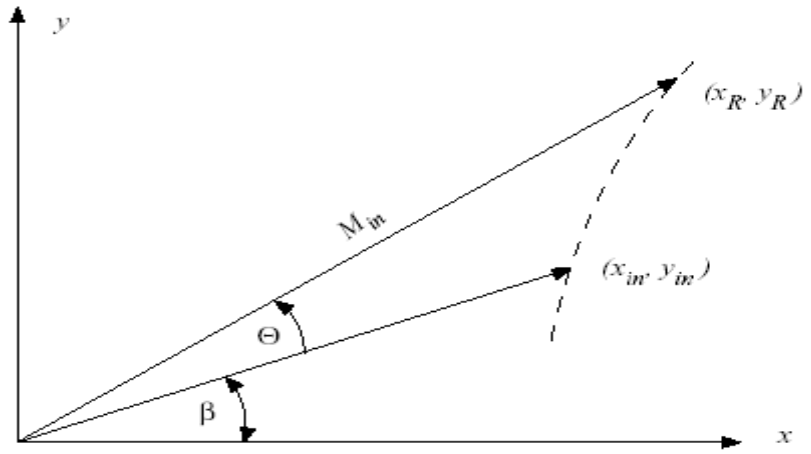


Figure (2.3.4): ROTATION IN HYPERBOLIC COORDINATE SYSTEM

The hyperbolic equivalents of all the functions discussed for the circular coordinate system can be computed in a similar fashion. Additionally, as Walther points out, the following functions can be derived from the CORDIC functions :

$\tan\alpha = \sin\alpha/\cos\alpha$

$$\tanh\alpha = \sinh\alpha/\cosh\alpha$$

$$\exp\alpha = \sinh\alpha + \cosh\alpha$$

$\ln\alpha = 2\tanh^{-1}[y/x]$ where $x = \alpha + 1$ and $y = \alpha - 1$

$(\alpha)^{½} = (x^2 - y^2)^{½}$ where $x = \alpha+1/4$ and $y = \alpha-1/4$

It is worth noting the similarities between the CORDIC equations for circular, linear, and hyperbolic systems. The selection of coordinate system can be made by introducing a mode variable that takes on values 1, 0, or –1 for circular, linear and hyperbolic system respectively. The unified CORDIC iteration equations are then :

$$x_{i+1} = x_i - m.y_i.\ d_i.\ 2^{-I}$$
$$y_{i+1} = y_i + x_i\ .\ d_i\ .\ 2^{-I}$$
$$z_{i+1} = z_i - d_i\ .\ e_i \qquad (2.3.14)$$

Where e_i is the elementary angle of rotation for iteration I in the selected coordinate system. Specifically, $e_i = \tan^{-1}(2^{-1})$ for m=1, $e_i = 2^{-I}$ for m=0, and $e_i = \tanh^{-1}(2^{-i})$ for m = -1. This unification, due to Walther, permits the design of a general purpose CORDIC processor.

3.6 CORDIC iterations with redundant number system:

In order to accelerate the CORDIC iterations , one can use redundant number systems, which enable additions without carry propagation .For redundant implementation SD number with digit set{-1,0,1} is used. With redundant number systems the main problem is the evaluation of $D_{i.}$. Assume that we are in rotation mode and the numbers are represented with m digits. In classical CORDIC, D_i is equal to the sign of Z_i .In sign digit representation this sign is the sign of most significant digit. This will require the examination of some number of digits which may be close to m. Thus choice D_i=sign(Z_i) is not satisfactory because the advantage of redundant number representation would be lost. An alternative to this is to accept D_i=0.

One examine the only p most significant digits of Z_i.The number Z_i * constituted by the these p digits is close to Z_i.The basic idea is

- If Z_i *≠0 then Z_i * and Z_i have the same sign , thus the choice D_i =sign(Z_i *) is convenient.

- If Z_i *=0, then mod(Z_i) is very small .therefore one can take D_i =0.

The main drawback of this method is that the scale factor A_n and A_n' are no longer constants. Since An is given by

$A_n = \prod 1 + 2^{-i}$

It is a constant only if the D_i's are all equal to –1 or +1 but it is no longer constant if D_i =0.

There are various approaches to solve the above problem:

1) Double rotation method.

2) Correcting rotation method.

3) Branching CORDIC .

1) Double rotation method:

The basic principle of this method is that at a step I one performs ,instead of a c-similarity of angle $D_i \arctan 2^{-i}$.

If $D_i \pm 1$, two c-similarities of angle $D_i \arctan 2^{-i-1}$.

If $D_i = 0$, a c-similarity of angle, $+\arctan 2^{-i-1}$ then a c-similarity of angle $-\arctan 2^{-i-1}$

With such a method , the scale factor is constant. However it leads to more complicated iterations.

2) Correcting rotation method.

The basic principle of this method is following:

One examines the number Z_i * constituted by the p most significant digits of Z_i .then one takes , in rotation mode , D_i = sing (Z_i *) if Z_i *≠0,

Else +1 o/w.

Sometimes an error occurs , but it is possible to correct it byrepetition of iterations p,2p,3p---- times.

3) Branching CORDIC :

In this method a sequence defined by

$Z_{i+1} = Z_i - D_i \cdot \tan^{-1}(2^{-i})$

Is builded.At each step , p digits of Z_i are examined, in order to decide the value of D_i, in general p=3.Then

If the examination of these p digits is sufficient to be sure that $Z_i > 0$, take $D_i = +1$.

If the examination of these p digits is sufficient to be sure that $Z_i < 0$, take $D_i = -1$.

If the examination of these p digits is not sufficient to know the sign of Z_i , then two computations in parallel s done :

a) former assuming $Z_i > 0$ (and therefore with $D_i = +1$).
b) Latter assuming $Z_i < 0$ (with $D_i = -1$).

This process is called Barnching.Sign selection is shown in the table (2.4)

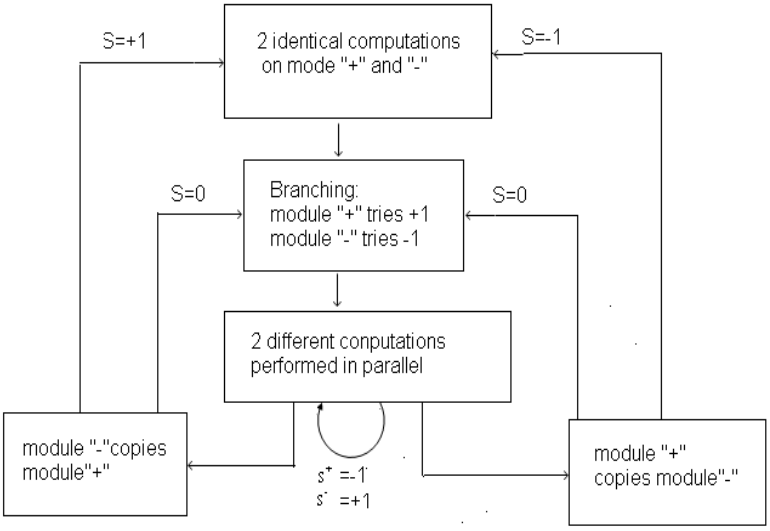


Fig (2.4.1 1) Diagram of branching CORDIC.

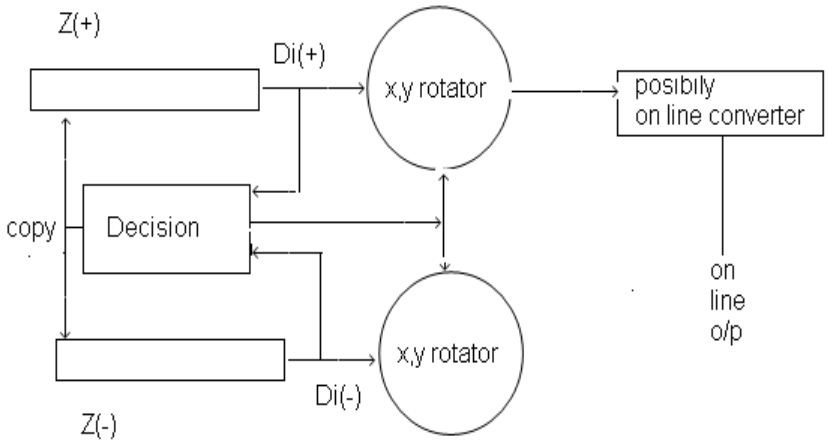


Fig (2.4.12) Architecture of a branching CORDIC processor

Fig (2.4.21) shows the global architecture for Branching CORDIC processor that implement the branching algorithm.

Table 2.4 For sign selection by examine 3 digit positions

Z_{i-2}	Z_{i-1}	Z_i	Sign	Z_{i-2}, Z_{i-1}, Z_i mod 8
-1	-1	-1	+	1
-1	-1	0	+	2
-1	-1	1	+	3
-1	0	-1	+	3
-1	0	0		4 impossible
-1	0	1	-	5
-1	1	-1	-	5
-1	1	0	-	6
-1	1	1	-	7
0	-1	-1	-	5
0	-1	0	-	6
0	-1	1	-	7
0	0	-1	-	7
0	0	0	??	0 branching
0	0	1	+	1
0	1	-1	+	1
0	1	0	+	2
0	1	1	+	3
1	-1	-1	+	1
1	-1	0	+	2
1	-1	1	+	3
1	0	-1	+	3
1	0	0		4 impossible
1	0	1	-	5
1	1	-1	-	5
1	1	0	-	6
1	1	1	-	7

This method is a very fast version of the CORDIC algorithm, which makes it possible to perform constant time elementary iteration independent of the length of the operands, with constant scale factor.

The main drawback of this method is the necessity of performing two conventional CORDIC iterations in parallel, which consumes more silicon area than classical method, but on other hand, provides fast and convenient implementation with small delay.

3.6.1 CORDIC using Redundant Number Systems

In conventional number systems, every addition or subtraction involves a carry propagation. Independent of the adder architecture the delay of the resulting carry ripple path is always a function of the wordlength. Redundant number systems offer the opportunity to implement carry-free or limited carry propagation addition and subtraction with a small delay independent of the used wordlength. Therefore they are very attractive for VLSI implementation. Redundant number systems have been in use for a long time e.g. in advanced parallel multiplier architectures (Booth, Carry{Save array and Wallace tree multipliers). However, redundant number systems offer implementation advantages for many applications containing cascaded arithmetic computations. Recent applications for dedicated VLSI architectures employing redundant number systems include finite impulse response filter (FIR) architectures, cryptography and the CORDIC algorithm. Since the CORDIC algorithm consists of a sequence of additions/subtractions the use of redundant number systems seems to be highly attractive. The main obstacle is given by the sign directed nature of the CORDIC algorithm. As will be shown below, the calculation of the sign of a redundant number is quite complicated in absolute contrast to conventional number systems where only the most significant bit has to be inspected. Nevertheless, several approaches were derived recently for the CORDIC algorithm. A brief overview of the basic ideas is given.

3.6.2 CORDIC Architectures

In this section several CORDIC architectures are presented. We start with the dependence graph for the CORDIC which shows the operational flow in the algorithm. Note that we restrict ourselves to the conventional CORDIC iteration scheme. The dependence graph for

extended CORDIC iterations can be easily derived based on the results. The nodes in the dependence graph represent operations and the arcs represent the flow of intermediate variables. Note that the dependence graph does not include any timing information, it is just a graphical representation of the algorithmic flow. The dependence graph is transformed into a signal flow graph by introducing a suitable projection and a time axis. The timed signal flow graph represents a register{transfer level (RTL) architecture. Recursive and pipelined architectures will be derived from the CORDIC dependence graph in the following. The dependence graph for a merged implementation of rotation mode and vectoring mode is shown in Fig. 3.1. The only difference for the two CORDIC modes is the way the control flags are generated for steering the adders/subtractors. The signs of all three intermediate variables are fed into a control unit which generates the control flags for the steered adders/subtractors given the used coordinate system m and a flag indicating which mode is to be applied.

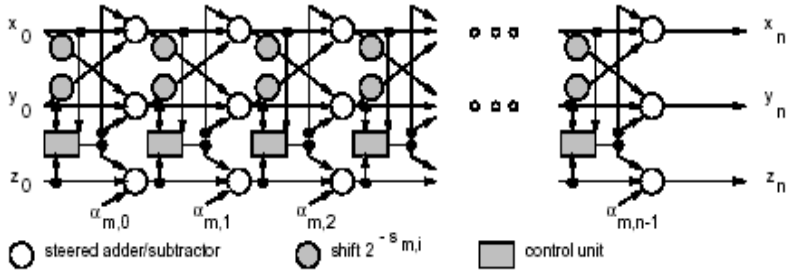


Figure 2.4.1 CORDIC dependence graph for rotation mode and vectoring mode.

In a one to one projection of the dependence graph every node is implemented by a dedicated unit in the resulting signal flow graph. In Fig. 2.4.2, the signal flow graph for this projection is shown together with the timing for the cascaded additions/subtractions (the fixed shifts are assumed to be hard wired, hence they do not represent any propagation delay). Besides having a purely combinatorial implementation, pipeline registers can be introduced between successive stages as indicated in Fig. (2.4.2).

In the following we characterize three different CORDIC architectures by their clock period T_{Clock}, throughput in rotations per second and latency in clock cycles. The delay for

calculating the rotation direction D_i is neglected due to the simplicity of this operation, as well as flip flop setup and hold times. As shown in Fig. (2.4.2) every addition/subtraction involves a carry propagation from least significant bit (LSB) to most significant bit (MSB) if conventional number systems are used. The length of this ripple path is a function of the wordlength W, e.g. $T_{Add} \sim W$ holds for a Carry{Ripple addition. The sign of the calculated sum or difference is known only after computation of the MSB. Therefore, the clock period for the unfolded architecture without pipelining is given by n * T_{Add} as shown in Fig. (2.4.2).

The throughput is equal to 1 /n*T_{add} rotations/s. The pipelined version has a latency of n clock cycles and a clock period $T_{Clock} = T_{add}$. The throughput is 1/ T_{add} rotations/s. It is obvious that the dependence graph in Fig. 3.1 can alternatively be projected in horizontal direction onto a recursive signal flow graph. Here, the successive operations are implemented sequentially on a recursive shared processing
element as shown in Fig. 2.4.3.

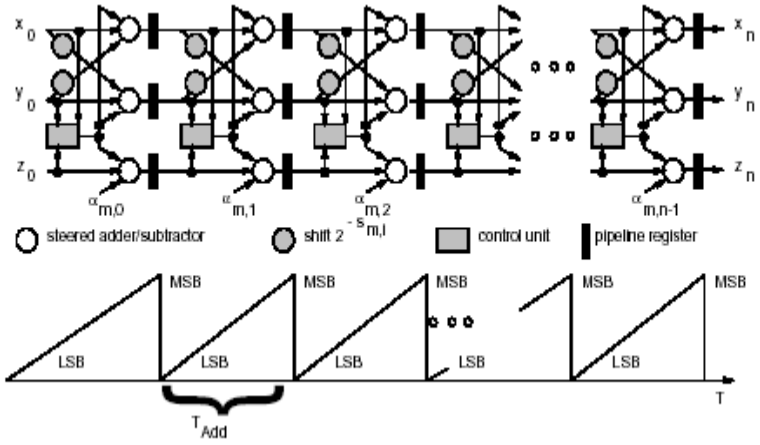


Figure 2.4.2 Unfolded (pipelined) CORDIC signal Flow graph.

Note that due to the necessity to implement a number of different shifts according to the chosen shift sequence, variable shifters (i.e. so called barrel shifters) have to be used in the recursive processing element.. The propagation delay associated with the variable shifters is comparable to the adders, hence the clock period is given by

$T_{Clock} = T_{Add} + T_{Shift}$.

The total latency for n recursive iterations is given by n clock cycles and the throughput is given by 1/ n*(T_{Add}+T_{Shift}) since new input data can be processed only every n clock cycles.

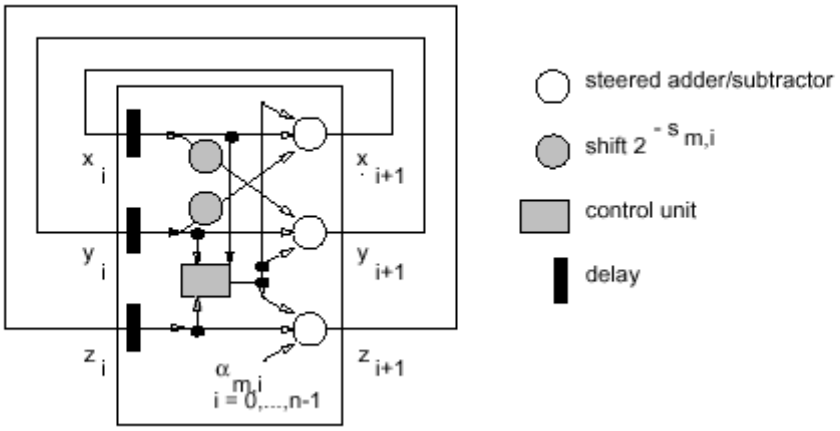


Figure 2.4.3 Folded (recursive) CORDIC signal flow graph.

The properties of the three architectures are summarized in Table 2.4.1.

Table 2.4.1 Architectural properties for three CORDIC architectures.

Architecture	Clock peroid	Throughput Rotations/s	Latency	Area
Unfolded	N* T_{add}	1/ n*T_{add}	l	3nadd,lreg
Unfolded pipelined	T_{add}	1/ T_{add}	n	3nadd,3reg
Folded recursive	T_{add} +T_{shift}	1/n(T_{add+} T_{shift})	n	3add,3+nregs 2shifters

Pipelined CORDIC architectures

In contrast to a universal CORDIC processing element the dominating motivation for a pipelined architecture is a high throughput constraint. Additionally, it is advantageous if relatively long streams of data have to be processed in the same manner since it takes a number of clock cycles to fill the pipeline and also to push the pipeline if e.g. the control flow changes (a different function is to be calculated). Although pipeline registers are usually

inserted in between the single CORDIC iterations as shown in Fig. 2.9 they can principally be placed everywhere since the unfolded algorithm is purely feed forward. A formalism to introduce pipelining is given by the well known cut{set retiming method The main advantage of pipelined CORDIC architectures compared to recursive implementations is the possibility to implement hard{wired shifts rather than area and time consuming barrel shifters. However, the shifts can be hard wired only for a single fixed shift sequence. Nevertheless, a small number of different shifts can be implemented using multiplexers which are still much faster and less area consuming than barrel shifters as necessary for the folded recursive architecture.
A similar consideration holds for the rotation angles. If only a single shift sequence is implemented the angles can be hard wired into the adders/subtractors. A small number of alternative rotation angles per stage can be implemented using a small combinatorial logic steering the selection of a particular rotation angle. ROMs or register files as necessary for the recursive CORDIC architecture are not necessary.

1.1) CORDIC Architectures for Vector Rotation

It was already noted that the CORDIC implementation of multiplication and division (m = 0) is not competitive. We further restrict consideration here to the circular mode m = 1since much more applications exist than for the hyperbolic mode (m = - 1).
Traditionally, vector rotations are realized as shown by the dependence graph given in Fig. 2.4.5. The sine and cosine values are generated by some table-lookup method (or another function evaluation approach) and the multiplications and additions are implemented using the corresponding arithmetic units as shown in Fig.2.4.5. Below, we consider high throughput applications with one rotation per clock cycle, and low throughput applications, where several clock cycles are available per rotation. High throughput applications: For high throughput applications, a one to one mapping of the dependence graph in Fig. 2.4.5 to a possibly pipelined signal flow graph is used. While only requiring a few multiplications and additions, the main drawback of this approach is the necessity to provide the sine and cosine values. A table-lookup may be implemented using ROMs or combinatorial logic. Since one ROM access is necessary per rotation, the throughput is limited by the access time of the ROMs. The throughput can not be increased beyond that point by pipelining.If even higher throughputs are needed, the ROMs have e.g. to be doubled and accessed alternatingly every other clock cycle. If on the other hand combinatorial logic is used for calculation of the sine and cosine values, pipelining is possible in principle. However, the cost for the pipelining can be very high due to the low regularity of the combinatorial logic which typically leads to a very high pipeline register count.

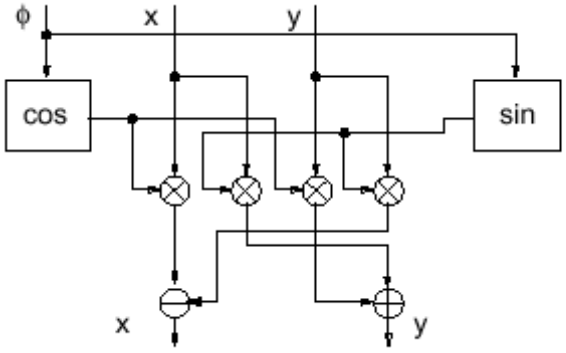


Figure 2.4.5 Dependence graph for the classical vector rotation.

Additionally, the effort for a CORDIC pipeline grows only linearly with the wordlength W and the number of stages n, hence about quadratically with the wordlength if n = W + 1 is used. In contrast the effort to implement the sine and cosine tables as necessary for the classical method grows exponentially with the required angle resolution or wordlength. Hence there is

a distinct advantage in terms of throughput and implementation complexity for the CORDIC at least for relatively large wordlengths. Due to the n pipelining stages in the CORDIC the classical solution can be advantageous in terms of latency. Low throughput applications:: A single resource shared multiplier and adder is sufficient to implement the classical method in several clock cycles as given for low throughput applications. However, at least one table shared for sine and cosine

Calculation is still necessary, occupying in the order of (2^{W-1})

*W bits of memory for a required word length of W bits for the sine and cosine values and the angle Φ In contrast, a folded sequential CORDIC architecture can be implemented using three adders, two barrel shifters and three registers.

If n = W + 1 iterations are used, the storage for the n rotation angles amounts to (W +1)*W bits only. Therefore, the CORDIC algorithm is highly competitive in terms of area consumption for low throughput applications.

The CORDIC vectoring mode can be used for fast and efficient computation of magnitude and phase of a given input vector. In many cases, only the phase of a given input vector is required, which can of course be implemented using a table lookup solution. However, the same drawbacks as already mentioned for the sine and cosine tables hold in terms of area consumption and throughput, hence the CORDIC vectoring mode represents an attractive alternative.

3.7 PREDICTING CORDIC ALGORITHM

The sine/cosine function generator is based on parallelization of the original CORDIC algorithm by predicting all the rotation directions directly from the binary bits of the initial input angle. Unlike previous approaches that require complicated circuits or exponentially increased ROM, this algorithm provides a relatively simple prediction scheme through an efficient angle recording. The critical path delay is also reduced by utilizing the predicted rotation directions to design an efficient multi operand carry-save addition structure.

One of the key components in direct frequency synthesizer (DDFS) system is the sine/cosine function generator that computes binary representation of sinθ and cosineθ to a precision of N fractional bits. CORDIC is an arithmetic algorithm developed to compute various elementary functions through a series of iterations of a unified micro rotations operation. In particular, in a circular rotation mode, N micro operations as illustrated below will be executed for i = 1, 2, … , N.

$$\begin{cases} x_{i+1} = x_i + \sigma_i 2^{-i} y_i \\ y_{i+1} = y_i - \sigma_i 2^{-i} x_i \\ z_{i+1} = z_i - \sigma_I \tan^{-1}(2^{-i}) \end{cases}$$

$\sigma_I = \text{sign}(z_i) \in \{1, -1\}.$

$= z_i - \sigma_i \alpha_I$ (2.5.1)

After N iterations, the accumulated rotation angle is

$$\theta = z_i - z_{N+1} = \sum_{i=1}^{N} \sigma_I \tan^{-1}(2^{-i}) \equiv \sum_{i=1}^{N} \sigma_i \alpha_i.$$

Using the definition of α_I, one has (note that $\cos\sigma_i\alpha_I = \cos\alpha_I$)

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & \sigma_i 2^{i} \\ -\sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

$$= \frac{1}{\cos \alpha_i} \begin{bmatrix} \cos\sigma_i\alpha_i & \sin\sigma_i\alpha_I \\ -\sin\sigma_i\alpha_i & \cos\sigma_i\alpha_i \end{bmatrix} \begin{bmatrix} Xi \\ Yi \end{bmatrix}$$

Then, it can be easily deduced that

$$\begin{bmatrix} x_{N+1} \\ y_{N+1} \end{bmatrix} = \frac{1}{K} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Where $K = \prod_{i=1}^{N} \cos\alpha_i = \prod_{i=1}^{N} (1+ 2^{-2i})^{-1\backslash 2}$ is a constant that can be precomputed in advance. Set $x_1 = K$, $y_1 = 0$, $z_1 = \theta$, then $x_{N+1} = \cos\theta$, $y_{N+1} = \sin\theta$ can be easily computed after N iterations. In conventional CORDIC, the direction $\sigma_I = \text{sign}(z_i)$ is determined sequentially since it depends on the sign of z_i calculated at the previous iteration. This dependence relation makes it difficult to execute multiple micro rotations in parallel. In this letter, we proposed a new method to quickly select the rotation directions $\{\sigma_I\}$ in order to speed up the calculation.

Angle Recording

Consider an arbitrary positive angle θ (<1 rad), which can be represented as $\theta = \sum_{k=1}^{N} b_k\theta_k$. the b_k are the bits corresponding to the (N + 1)-bit fractional binary representation of the angle θ (sign bit and N fractional bit and θk are the positional power-of-two weights. In other words, $b_k \in \{0, 1\}$ and $\theta_k = 2^{-k}$, since the angle θ constrained to be

positive, b0 = 0 does not appear in the above summation. The binary $b_k \in \{0, 1\}$ can be recoded into signed digit $r_k \in \{-1, 1\}$ such that

$$\theta = \sum_{k=1}^{N} b_k 2^{-k} = \phi_0 + \sum_{k=2}^{N+1} r_k 2^{-k} \qquad (2.5.2)$$

Where ϕ_0 is a constant. The recording can be understood intuitively if we visualize a positive sub rotation by 2^{-k} rad as the sum of the two equal but opposite half rotations by 2^{-k-1} rad and zero rotation as two equal but half rotations by 2-k-1 rad. It is quit easy to see from the binary representation of the angle θ that a binary bit $b_k = 1$ result in a rotation by 2^{-k} rad, where as a $b_k = 0$ results in zero rotation. Thus, the kth sub rotation stage consist of fixed rotation by 2-k-1 rad followed by a positive or negative rotation by 2-k-1 rad that is controlled by the bit b_k. combining all the fixed rotations, we obtained the initial fixed rotations ϕ_0 = ¼ + 1/8 + … + 1/2n+1 rad. Thus, the rotation after recoding can be represented as a fixed initial rotation ϕ_0 followed by a sequence of positive or negative rotation whose directional are controlled by the binary bits b_k. a bit $b_k = 1$ corresponding to a positive (or counterclockwise) rotation by 2^{-k-1} rad, and a bit $b_k = 0$ corresponding to a negative (or clockwise) rotation by 2^{-k-1} rad, thus $r_k = (2b_{k-1} - 1)$ for k = 2,…,N+1 . The recoding need not be performed explicitly since it corresponds to simply replacing all "0" value in the binary representation by "-1".
The recoding maintains a constant scale factor K in the data path. The scaling by K and the initial rotation by ϕ_0 are most efficient accounted for in the implementation by starting each sequence of angle rotations from the initial point $(X_0, Y_0) = (K \cos \phi_0, K \sin \phi_0)$.

One of the major benefits of recoding over CORDIC is that the direction of rotation at each stage is immediately obtained from the binary representation of the angle θ, thereby dominating the need for comparing angles at each stage. This possible because the sub angles $\theta_k = 2^{-k}$ used in recoding are different from the sub angle $\theta_k = a \tan 2^{-k}$ used in CORDIC. The penalty for using different sub angles is that the $\tan \theta_k$ multipliers used in the first few sub rotation stages cannot be implemented as simple shift-and-add operations. However, as will be shown subsequently, this limitation can be overcome by implementing these stages as a small ROM. Thus, recoding results in reducing chip area and higher operating speed.

PREDICTIONS OF ROTATION DIRECTIONS:

Binary to bipolar recoding (BBR)

The initial input angle $\theta = (-\theta_0) + \sum_{j=1}^{N} \theta_j 2^{-j}$ with $\theta_j \in \{0, 1\}$ is assumed to be in the range $|\theta| < \pi/4$ as in the application example of DDFS. It has been shown in [3] that $\tan^{-1}2^{-i}$ to N-bit precision if $i \geq m = [(N - \log_2 3)/3]$. Thus, the last 2N/3 rotation directions (from σ_m to σ_N) can be obtained in parallel after completing the first N/3 iterations. As proposed above in angle recoding, we divide the angle into two parts (the higher part and the lower part)

$$\theta = \theta_H + \theta_L = (-\theta_0) + \underbrace{\sum_{j=1}^{m-1} \theta_j 2^{-j}}_{\theta_H} + \underbrace{\sum_{j=m}^{N} \theta_j 2^{-j}}_{\theta_L}$$

The binary bits $\theta_j \in \{0, 1\}$ in the higher part θ_H can be recoded into bipolar digits as follows:

$$\theta_H = (-\theta_0) + \sum_{j=1}^{m-1} \theta_j 2^{-j}$$

$$= (-\theta_0) + \sum_{j=1}^{m-1} [2^{-j-1} + (2\theta_j - 1)\, 2^{-j-1}]$$

$$= (-\theta_0) + 2^{-1} + \sum_{j=2}^{m} r_k 2^{-k} - 2^{-m} \qquad (2.5.3)$$

where $r_k = (2\theta_k - 1) \in \{1, -1\}$.

Equation (3) is called BBR for θ_j, j = 0, 1,…, m-1.

Microrotation Angle Recoding (MAR)

Since $\tan^{-1}(2^{-i}) \neq 2^{-i}$ for I = 1,…, m-1, we decompose each positional binary weighting 2^{-i}, I = 1,…,m-1 into the combination of significant $\tan^{-1}(2^{-j})$ terms plus an error term e_i collecting all the other insignificant values of $\tan^{-1}(2^{-j})$, j > m. for simplicity, we take N = 24 as an example were $m = [(N\text{-}\log_2 3)/3] = 8$. The Microrotation angle recording from 2^{-i} to $\tan^{-1}(2^{-i})$, I = 1,…, 7 is

$$2^{-1} = \tan^{-1}(2^{-1}) + \tan^{-1}(2^{-5}) + \underbrace{\tan^{-1}(2^{-8}) + 0.000000000100111100001110_2}_{e_1}$$

$$2^{-2} = \tan^{-1}(2^{-2}) + \tan^{-1}(2^{-8}) + \underbrace{0.000000000100100100100010100_2}_{e_2}$$

$$2^{-3} = \tan^{-1}(2^{-3}) + \underbrace{0.000000000010101001000101_2}_{e_3}$$

$$2^{-4} = \tan^{-1}(2^{-4}) + \underbrace{0.000000000000010101010010_2}_{e_4}$$

$$2^{-5} = \tan^{-1}(2^{-5}) + \underbrace{0.000000000000000010101010_2}_{e_5}$$

$$2^{-6} = \tan^{-1}(2^{-6}) + \underbrace{0.000000000000000000010101_2}_{e_6}$$

$$2^{-7} = \tan^{-1}(2^{-7}) + \underbrace{0.000000000000000000000010_2}_{e_7} \qquad (2.5.4)$$

The BBR for θ_H with N = 24 is

$$\theta_H = (1-2\theta_0)2^{-1} + \sum_{k=2}^{8} r_k 2^{-k} - 2^{-8}. \qquad (2.5.5)$$

The first eight rotation directions are selected concurrently as

$\sigma_k = (1 - 2\theta_0)$

$\sigma_k = r_k = (2\theta_{k-1} - 1)$, k = 2,…,8. (2.5.6)

Then, all the signed error terms $\sigma_i e_i$, i = 1,…, 7 and the last term -2^{-8} in (5) are added to θ_l, generating the corrected lower part $\hat{\theta}_L$ represented in twos compliment format, i.e.,

$$\hat{\theta}_L = \theta_L + \sum_{i=1}^{7} \sigma_i e_i - 2^{-8}$$

$$= (-\hat{\theta}_7)2^{-7} + \sum_{k=8}^{24} \hat{\theta}_k 2^{-k},\ \hat{\theta}_L \in \{\hat{\theta}_k, 1\},\ k = 8,\ldots, 24. \qquad (2.5.7)$$

It can be shown that $|\hat{\theta}_L| < 2^{-7}$. Since $\tan^{-1}2^{-i} = 2^{-i}$, I ≥ 8 within precision of 24 fractional bits, the algorithm converges after the above selection of directions for the remaining microrotation can be derived immediately from (7) using again the BBR

$$\hat{\theta}_L = (-\hat{\theta}_7)2^{-7} + \sum_{k=8}^{24} \hat{\theta}_k 2^{-k}$$

$$= (-\hat{\theta}_7)2^{-7} + \sum_{k=9}^{25} (2\hat{\theta}_{k-1} - 1)2^{-k} + 2^{-8} - 2^{-25}$$

$$= (1 - 2\hat{\theta}_7)\, 2^{-8} + \sum_{k=9}^{25} \hat{r}_k 2^{-k} - 2^{-25}$$

$$\hat{r}_k = (2\hat{\theta}_k - 1) \in \{1, -1\}$$

(2.5.8)

leading to the parallel prediction

$$\hat{\sigma}_8 = (1 - 2\hat{\theta}_7)$$

$$\hat{\sigma}_k = \hat{r}_k = (2\hat{\theta}_k - 1),\ k = 9,\ldots,25$$

for the last 2N/3 microrotations.

3.8 **DIGITAL CHIRP GENERATOR**

Digital Chirp Generator (DCG) system are characterized by fast switching, fine frequency discrimination, low phase noise, and transient-free frequency changes. Frequency changes are phase continuous, which is describe in phase or frequency modulation. The fast frequency switching is useful in frequency-agile and spread-spectrum systems. Depending on the particular application, there exist tradeoffs among the power, IC area, size, and spectral purity of the implementation. For example, output resolution and spectral purity are of primary importance in instrumentation applications. Area and power are of considerable importance for digital mobile radio and cellular telephony. Simple modifications to the phase generation circuitry produces synthesized chirps useful in radar and electronic warfare systems and in implementation of continuous-phase modulators (e.g., GMSK).

A DCG consists of a phase accumulator, frequency accumulator and a sine/cosine generator, as shown in the shaded portion of fig.1. The phase accumulator is an overflowing M-bit accumulator whose value specifies the instantaneous phase. The M-bit value may be truncated to another L-bit value, which is fed as the argument θ to a sine/cosine generator that computes the digital $\sin\theta$ and $\cos\theta$ value to a precision of p bits.

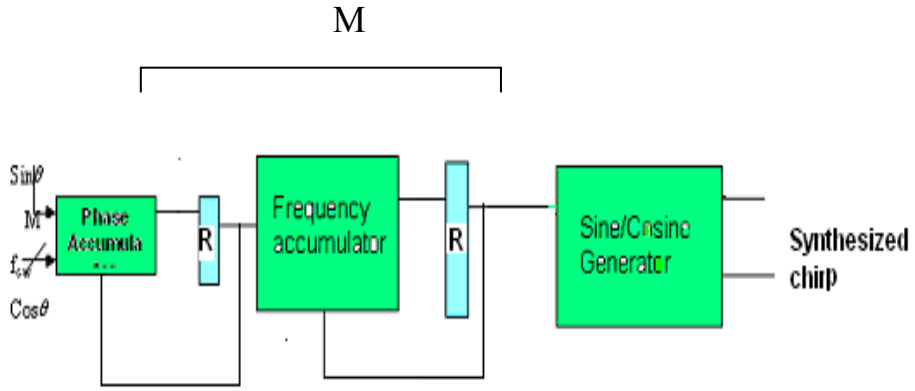


fig2.6.1. Traditional DCG architecture.

A. Output frequency

In fig.1, F_{cw} denotes an external input to the phase accumulator called the "frequency control word". At each clock cycle (clock frequency $F_{clk} = 1/T_{clk}$), the phase accumulator increments itself by the value F_{cw} until it overflows and wrap around. Each overflow of the accumulator

corresponds to cos period of a sine (or cosine) wave. Thus, F_{cw} control the rate at which the accumulator overflows, thereby controlling the frequency of the sine or cosine waveform.

The value of the M-bit accumulator is mapped to an angle (or phase value) in the interval $[-\pi, \pi]$.

Most practical DCG designs reported during the last two decades rely on some variant of the basic pioneering table-lookup algorithm, wherein the phase accumulator provides an address to a CORDIC module that generates the sine and cosine values. One of our main objectives is to describe an approach that will permit the implementation of high-speed DCG designs with high-precision spectrally pure sine and cosine output. As a result, simple tradeoffs among the power, IC area, sample rate, resolution, and spectral purity of the implementation can be preformed. Our approach is based on an architecture that evaluates the projections of a pharos rotating the unit circle onto the X and Y-axes by the CORDIC algorithm it is not based on a table-lookup architecture. It will be shown that overall architecture can be implemented as a simple multiplier less feed-forward data path , which allows for easy pipelining and limits the accumulation of round off errors. A prototype DCG has been designed, for vertex 300pq240 FPGA and tested. The design produces 10-b sine and cosine output at 99.33 MHz.

II. Sine/Cosine Generation

Let us consider the computation of the sine and cosine of an angle θ specified in new angle format.. For the given angle θ, the computation of sinθ and cosθ can be viewed as the computation of the X –axis and Y-axis coordinates (X_θ, Y_θ) of a point on the unit circle, as in circular CORDIC rotation algorithm.

The CORDIC Algorithm

A well-known iterative technique for computing rotations θ_k is the CORDIC algorithm. In the CORDIC algorithm, $\sigma_k \in \{-1, 1\}$ and θ_k at a 2^{-k}, the multiplication by tan θ_k in the above equation can be implemented as a simple sub-and-add operation, resulting in a multiplier less data path. Since cos θ = cos (-θ), the product $K = \cos \sigma_0\theta_0 \ldots \cos \sigma_N\theta_N$ become a constant, independent of the specific direction of the sub rotations. (This is true as long as a positive negative sub rotation is actually performed at each iteration. Usually, the scale factor K is simply introduced into the conditions as $(X_0, Y_0) = (K, 0)$.

The σ_k values determine whether a positive or negative sub rotation by angle θ_k is required. The σ_k values determined iteratively by the method of successive approximation. If, at the Kth iteration, the current approximation is larger (smaller) than the input angle θ, its value is adjusted by subtracting (adding) the angle θ_k. the CORDIC algorithm required hardware that a) computer σ_k and update the current approximation by the angle θ_k and b) performs the rotation by θ_k. roughly one-third of the total CRODIC hardware is required for computing the σ_k and updating the current approximation.

The throughput of the CORDIC data path can be improved by using redundant carry-free arithmetic, which eliminates the carry-propagate delay in the adders.

ARCHITECTURE FOR CHIRP GENERATOR

For frequently synthesis, the argument fed to the sine and cosine generators is derived from an overflowing two's compliment accumulator. The contents of second accumulator represent a frequency. Therefore, the normalized angle must be converted to an appropriate value θ in the interval $[-\pi/2, \pi/2]$, which is input to the sine/cosine generator. The N binary bits representing the angle that control the directions of the sub rotations that compute the sin θ and cos θ values at a precision of N bits. An output stage is required to generate the correct sin θ and cos θ from the computed sin θ and cos θ values. Thus, the overall architecture consists of the following blocks:

1) A phase accumulator that generates the normalized angle ϕ.
2) A frequency accumulator.
3) A sine/cosine generator that compute sin θ and cos θ, i.e.:(CORDIC module)
4) A preprocessor stage that processed the angle input to the CORDIC unit.
5) A delay that stores the quadrant information for the final decision.
6) A postprocessor stage that finally decide the sign based on quadrant information.

A. Phase Accumulator

The phase accumulator, shown in fig. 5, is an M-bit adder that repeatedly increments the phase angle. That is, its output increases by F_{cw} at each clock cycle. At time n, the output of the phase accumulator is $\phi = nF_{cw}/2^M$, and the sine/cosine generator must compute sin $(2\theta nF_{cw}/2^M)$ and cos $(2\theta nF_{cw}/2^M)$.

A load control signal loads the frequency control word, and a reset signal initializes the contents of the phase accumulator to zero.

B. Preprocessor

The two most significant bits of the normalized angle ϕ, MSB_1 and MSB_2 determined the quadrant occupied by ϕ. These bits determined whether this angle is in third, second, fourth or first of the quadrant. These two most significant bits are stored and used later to control an interchange/negation operation in the output stage. The xor of two bits are stored in the delay.

The value of ϕ is first modified by setting its MSB value equal to the next most significant bit value. This maps any angle in the normalized second, third to fourth and first quadrant respectively to a corresponding angle ϕ' in the fourth and first quadrant respectively, as shown in fig. 6(a). If angle lies in the second quadrant whenever delay = 1 because MSB1 MSB2="01". The sine and cosine of any angle γ above $\pi/2$ can be obtained from those of an angle equally below $\pi/2$, as shown in fig. 6(b).

C. Sine/Cosine generator

The CORDIC module does Sine/Cosine generation, which is the heart of the whole system. This module accepts 13 bit of normalized preprocessed angle and computed sin θ and cos θ values to 10 bits precession. This CORDIC module is designed using carry-free adder trees; with pipelined registers are inserted b/w adders. Here both arithmetic as well as logical shifting is done using the shifter designed specifically for this purpose.

TABLE 2.6.Angle ROM

k	θ_k	θ_k(13-bit binary)
1	45°	0010000000000
2	26.5650°	0001001011110
3	14.0362°	0000101000000
4	7.12502°	0000010100010
5	3.57633°	0000001010001
6	1.78991°	0000000101001
7	0.895174°	0000000010100
8	0.447614°	0000000001010
9	0.223807°	0000000000101
10	0.111905°	0000000000010

D. Output Stage

The output stage, show in fig. 10, maps the computed sin θ and cos θ values to the desired sin θ and cos θ values, where ϕ, lies in the correct interval within [-π/2, π/2]. As mentioned previously, this can be accomplished by simple negation and/or interchanging operations. The control signals xinvert and yinvert control the negation of cos θ = X_{N+1} and sin θ = Y_{N+1}, respectively. The control signal, derived from the two most significant bits of the normalized angle ϕ, are generated as shown in table II. The negation is assumed to occur before the interchange. Since the control signals are generated from ϕ, they must be suited delayed until the sin θ and cos θ values are computed. This delay is determined by the latency of the sine/cosine-generating datapath and can be implemented as a simple register chain.

TABLE 2.6.1
CONTROL SIGNALS FOR THE OUTPUT STAGE

MSB's of ϕ	ϕ	Delay	cos ϕ
0 0	0<ϕ<π/2	0	cosθ
0 1	π/2<ϕ<π	1	sinθ
1 0	π<ϕ<3π/2	1	-sinθ
1 1	3π/2<ϕ<2π	0	cosθ

The two most significant bits MSB are extracted from the truncated phase accumulator and used to control the conditional interchange/negation operations in the output stage. The residual angle with its one most-significant bits set equal to next MSB value represent the normalized angle ϕ', which is fed to the CORDIC..
The overall architecture is shown in fig.(2.6.2).

Since our prototype design produces 10-bit sine and cosine output, the data path consists of a cascade of ten CORDIC stages. Simulations indicate that 10-bit accuracy can be retained on the sine and cosine output.

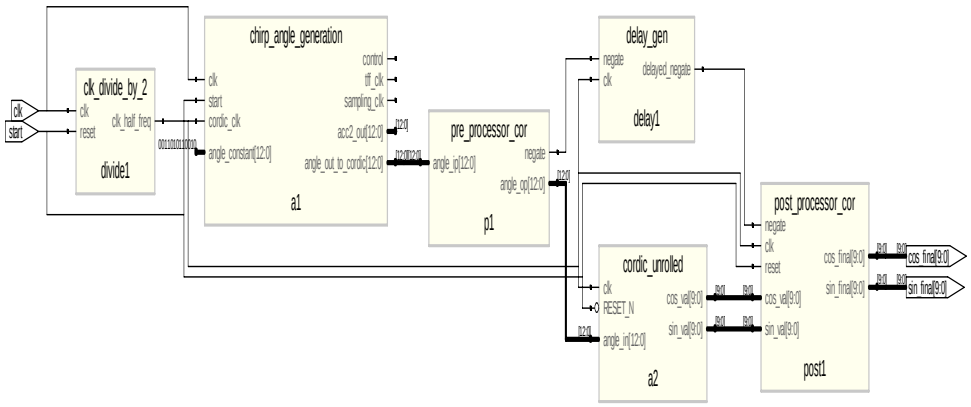
TABLE 2.6.3

PERFORMANCE SUMMARY

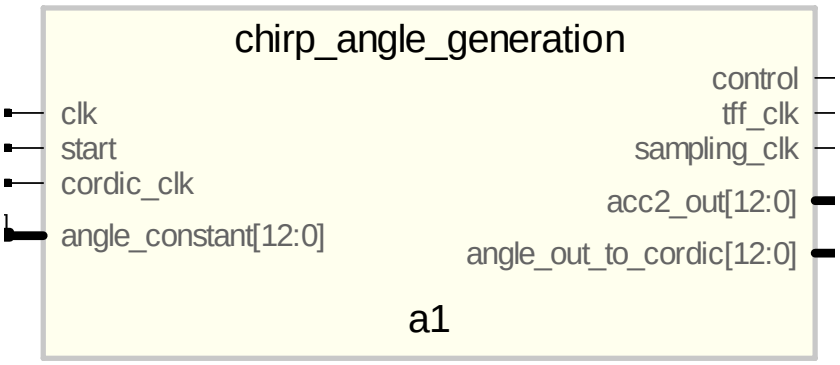
Technology	**Vertex 300**
Maximum clock frequency	99.493MHz
Output Resolution	10-bits for sine and cosine
Latency	12 clock cycle
Gate Count	13,172

Sufficient pipelining was employed to achieve this clock rate. Each stage has a pipeline register at its output. Adder stages were implemented using carry-free adders with three pipeline registers..

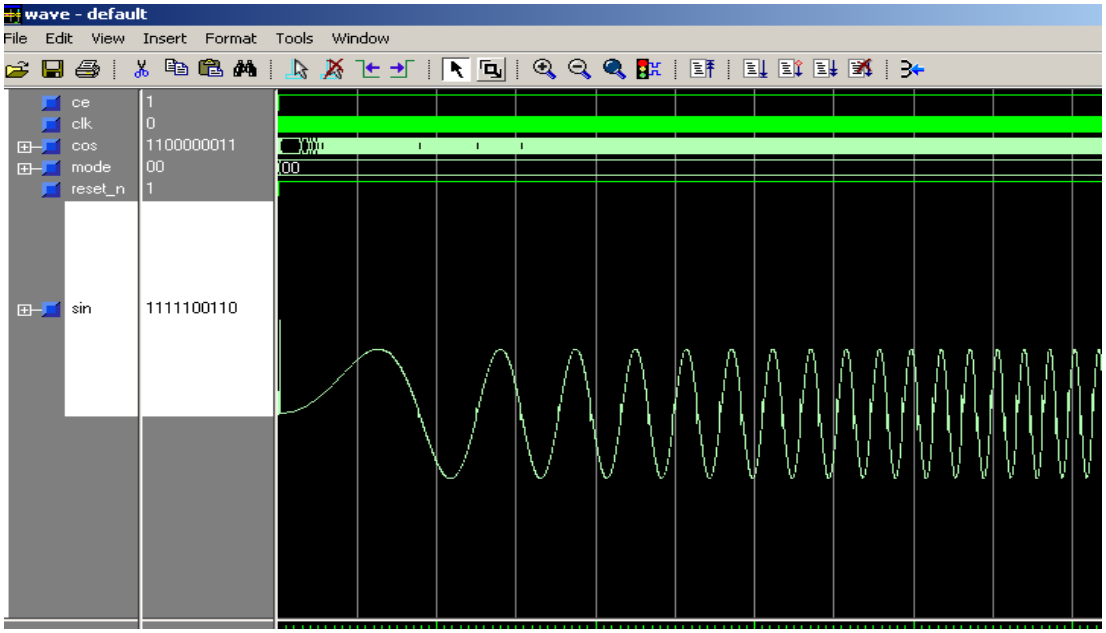
Our design testing using the chipscope tester and a customer board has found the design to be fully functional.



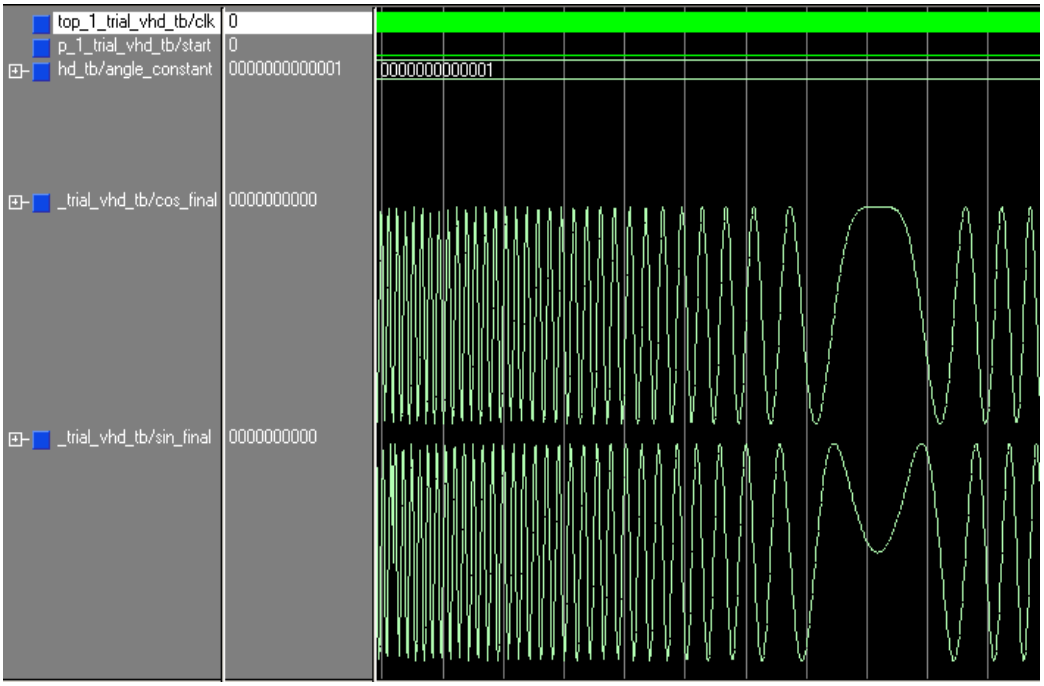
Fig(2.6.5) RTL view of chirp generator



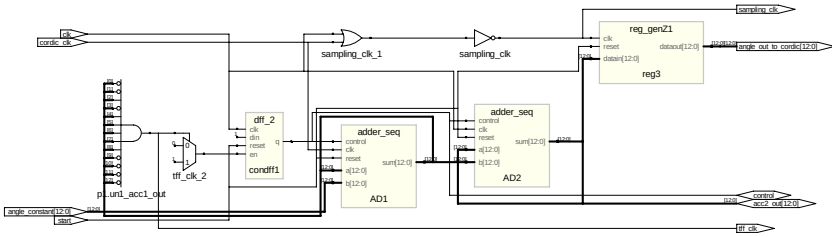
Fig(2.6.7)Chirp Angle Generation



Fig(2.6.4) ModelSim results of the Chirp generator.



Fig(2.6.6)Chirp ModelSim Results



Fig(2.6.8)RTL view of Angle generation unit

Chirp Report

Device utilization summary:

Selected Device	:	v300pq240-4
Number of Slices	:	577 out of 3072 18%
Number of Slice Flip Flops	:	529 out of 6144 8%
Number of 4 input LUTs	:	1017 out of 6144 16%
Number of bonded IOBs	:	34 out of 170 20%
Number of GCLKs	:	1 out of 4 25%

Timing Summary:

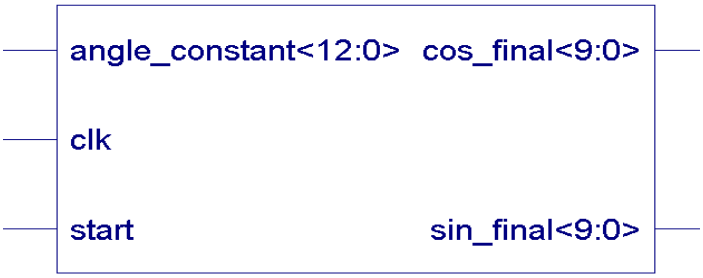
Speed Grade : -5

Minimum period: 10.051ns (Maximum Frequency: 99.493MHz)

Minimum input arrival time before clock: 5.764ns

Maximum output required time after clock: 8.289ns

Maximum combinational path delay: No path found



fig(2.6.5)Top view of Sine/Cosine generator

3.9 Fast Fourier Transform

Fast Fourier Transform(FFT)has been used in a wide range of applications, such as wide-band mobile digital communication system based on Orthogonal Frequency Division Multiplexing (OFDM) principle, where the system implementation is only feasible when the equipment complexity and power consumption are greatly reduced by utilizing a real-time FFT transformer to replace the bank of (de)modulators for each individual sub-carriers.

FFT operation has been proven to be both computational intensive, in terms of arithmetic operations, and communicational intensive, in terms of data swapping/exchanging in the storage. For real-time processing of EFT transform, O(10gN) arithmetic operations are required per sample cycle, where N is the length of the transform. High speed real-time processing can be accomplished in two different ways. In a conventional, general purpose processor approach, a single processor driven to a very high clock frequency, which is O (1og N) times the sampling frequency, is used to carry out the operation. While in an application specific approach, parallel or concur- rend pipelined processors, operating on a clock frequency close or equivalent to the sampling frequency, are used to attain the performance. Analysis has shown that the second approach is more preferable when the application environment limits power consumption, such as in mobile communication.

Pipeline FFT processor is a class of architectures for application specific real-time DIT computation utilizing fast algorithms. It is characterized by non-stopping processing on a clock frequency of the input data sampling. A lower clock frequency is a clear advantage for pipeline architectures, when either a high speed processing or a low power solution is sought. In addition, pipeline structure is highly regular, which can be easily scaled and parameterized when Hardware Description Language (HDL) is used in the design. It is also more flexible when transforms of different lengths are to be computed with the same chip.

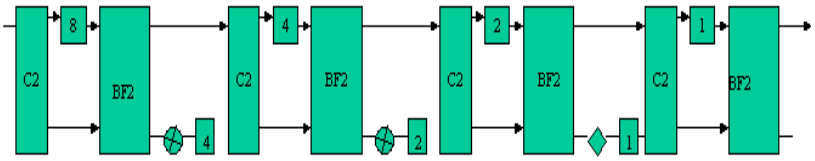
. In the following section, pipeline FFT processors are briefly reviewed.

3.9.1 PIPELINED FFT ARCHITECTURE.

The architecture design for pipeline FFT processor had been the subject of intensive research as early as in 70's when real- time processing was demanded in such applications as radar signal processing . Several architectures have been proposed over the last 2 decades. Here different approaches are put into functional blocks with unified terminology. The additive butterfly has been separated from multiplier to show the hardware requirement distinctively, as in Fig. **1.** The control and twiddle factor reading

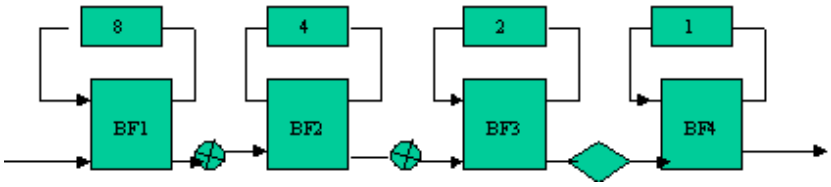
mechanism have been also omitted for clarity. All data and arithmetic operations are complex, and a constraint that N is a power of 4 applies.

R2MDC: Radix-2 Multi-path Delay Commutator was probably the most classical approach for pipeline implementation of radix-2 FFT algorithm. The input sequence has been broken into two parallel data stream flowing forward, with correct "distance" between the data elements entering the butterfly scheduled by proper delays. Both butterflies and multipliers are in 50% utilization. $Log_2 N-2$ multipliers, $log_2 N$ radix-2 butterflies and $3/2N - 2$ registers (delay elements) are required.



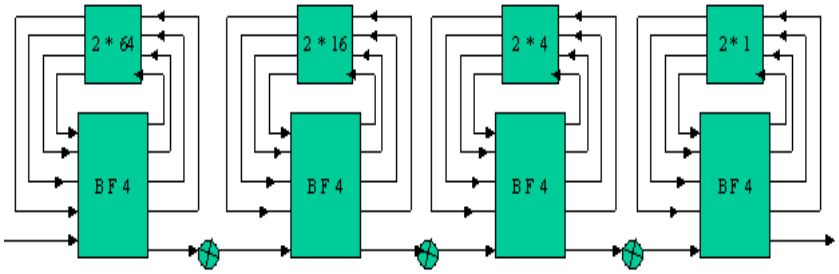
Fig(2.7.1) R2MDC(N=16)

R2SDF: Radix-2 Single-path Delay Feedback uses the registers more efficiently by storing the one butterfly out- put in feedback shift registers. A single data stream goes through the multiplier at every stage. It has same number of butterfly units and multipliers as in R2MDC approach, but with much reduced memory requirement: $(N-1)$ registers. Its memory requirement is minimal.



Fig(2.7.2) R2SDF(N=16)

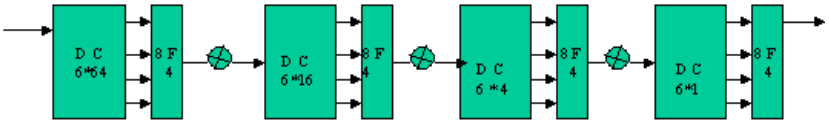
R4SDF: Radix-4 Single-path Delay Feedback was proposed as a radix-4 version of R2SDF, employing CORDIC iterations. The utilization of multipliers has been increased to 7.5% by storing 3 out of 4 radix-4 butterfly outputs. However, the utilization of the radix-4 butterfly, which is fairly complicated and contains at least 8 complex adders, is, dropped to only 25%. It requires log, N – 1multipliers, log, N full radix-4 butter- flies and storage of size $N - 1$.



Fig(2.7.3)R4SDF

R4MDC: Radix-4 Multi-path Delay Commutator is a radix-4 version of R2MDC. It has been used as the architecture for the initial VLSI implementation of pipeline FFT processor and massive wafer scale integration However, it suffers from low, 25%, utilization of all components, which can be compensated only in some special applications where four FFTs are being processed simultaneously. It requires 3 log, N multipliers, log, N full radix-4 butterflies and $5/2N - 4$ registers.

R4SDC: Radix-4 Single-path Delay Commutator uses a modified radix-4 algorithm with programmable 1/4 radix-4butterflies to achieve higher, 7.5% utilization of multipliers. A multiplexed Delay-Commutator also reduces the memory requirement to $2N - 2$ from $5/2N - 1$, that ofR4MDC. The butterfly and delay-commutator be- come relatively complicated due to programmability requirement. R4SDC has been used recently in building the largest ever single chip pipeline FFT processor for HDTV application.



Fig(2.7.4) R4SDC

A swift skimming through of the architectures listed above reveals the distinctive merits of the different approaches: First, the delay-feedback approaches are always more efficient than corresponding delay- commutator approaches in terms of memory utilization since the butterfly output share the same storage with its input. Second, radix-4 algorithm based

single-path architectures have higher multiplier utilization, however, radix-2 algorithm based architectures have simpler butterflies which are better utilized.

3.9.2 CORDIC FFT

BASIC FFT

The discrete Fourier transform (DFT) of N complex samples f (k), k = 0, 1, ..., N - 1 is defined as

$$F(r) = \sum_{k=0}^{N-1} f(k)\, W^{rk},\ r = 0, 1,2,3,\ldots ., N - 1, \qquad (2.7.1)$$

Where $W = \exp(-2\pi j/N)$.

The fast Fourier transform (FFT) is a clever efficient method of computing the DFT of N number of discrete data samples in $O(N\log_2 N)$ time as opposed to that $O(N^2)$ in the direct method. The FFT algorithm starts with splitting the input data set f(k) into odd-and even-numbered points, d(k) and e(k), respectively, as follows :

$e(k) = f(2k)$,

(2.7.2)

$d(k) = f(2k+1)$, where k = 0, 1,....N/2 - 1.

Now, Eq. (1) may be rewritten as

$$F(r) = \sum_{k=0}^{N/2-1} e(k)\, W^{2rk},\ r = 0, 1,2,3,\ldots ., N/2 - 1, \qquad (2.7.3)$$

$$F(r) = E(r) + W^r D(r), \qquad (2.7.4)$$

where

$$E(r) = \sum_{k=0}^{N/2-1} e(k)\, W^{2rk},\ r = 0, 1,2,3,\ldots ., N/2 - 1, \qquad (2.7.5)$$

and

$$D(r) = \sum_{k=0}^{N/2-1} d(k)\, W^{2rk},\ r = 0, 1,2,3,\ldots ., N/2 - 1, \qquad (2.7.5)$$

E(r) and D (r) may be viewed as the DFT of N/2 point sequences e(k) and d(k), respectively. The FFT algorithm in which the input data samples are split into odd-and even numbered ones, is called decimation in time, while in the other form of FFT, known as decimation in frequency, the discrete data samples f(k) are split two equal parts g(k) and h(k) with the first one having the first N/2 samples and the other having the last N/2 data samples such as

$$g(k) = f(k), \qquad (2.7.6)$$

and

$$h(k) = f(k=N/2), \quad k = 0, 1,\ldots.. N/2 - 1.$$

The N point DFT of f(k) as depicted in Eq. (2.7.1) may be rewritten as

$$F(r) = \sum_{K=0}^{N/2-1} \{g(k)\, W^{rk},\ h(k) w^{rk+rN/2}\}. \qquad (2.7.7)$$

If B (r) and C (r) are the even-numbered and odd numbered transform points, respectively, such as

$$B(r) = F(2r),$$

and (2.7.8)

$$C(r) = F(2r + 1),\ r = 0, 1,\ldots. N/2 - 1$$

then they can be represented by the following equations :

$$B(r) = \sum_{K=0}^{N/2-1} [g(k) + h(k)]\, W^{2rk}$$

and

$$C(r) = \sum_{K=0}^{N/2-1} [g(k) + h(k)]\, W^k\, W^{2rk}$$

B (r) and C (r) are nothing but N/2 point DFTs of the functions g(k) + h(k) and $h(k)W^k$, respective. Therefore, either in decimation in time or in decimation m frequency, a DFTs, each of which may again be computed through two N/4 point DFTs and so on. This is illustrated in the form of a signal flow graph is a two-point DFT butterfly, depicted in Fig. 3, having the following form :

$$R = P + Q, \qquad (2.7.10)$$

$$S = (P-Q)W^k.$$

Eq. (2.7.10) may be written in its expanded form in terms of the real and the imaginary parts of the signals as follows :

$$R_{re} = P_{re} + Q_{re}, \qquad (2.7.11)$$

$$R_{im} = P_{im} + Q_{im},$$

$$S_{re} = (P_{re} - Q_{re}) \cos(k\theta) + (P_{im} - Q_{im}) \sin(k\theta),$$

$$S_{im} = -(P_{re} - Q_{re}) \sin(k\theta) + (P_{im} - Q_{im}) \cos(k\theta).$$

The last two in the set of Eqs. (2.7.11) essentially represent a plane rotation operation which can be efficiently computed by applying the CORDIC algorithm.

In the CORDIC technique, the plane rotation through an angle α is achieved by decomposing the target angle into several elementary angles and carrying out rotations through each of these as follows :

$$\alpha = \sum_{i=0}^{M-1} \delta_i \theta_i, \quad \text{where } \theta_i = \tan^{-1}(2^{-i})$$

with M being the wordlength and $\delta_i = +1$ or -1,

since $\theta_i/2 < \theta_i+1 < \theta_i$, any arbitrary angle can be expressed in terms of elementary angles (θ_is) with their signs (δ_is) properly chosen. Now an elementary plane rotation in two dimension can be expressed as

$x_i+1 = x_i \cos(\theta_i) + \delta_i y_i \sin(\theta_i)$, (2.7.13)

$y_i+1 = \delta_i x_i \sin(\theta_i) + y_i \cos(\theta_i)$

with the value of δ_i deciding the direction of rotation.

Applying the condition $\tan(\theta_i) = 2^{-i}$, as stated in Eq. (2.7.12), for elementary angles, Eq. (2.7.13) may be rewritten as

$x_i+1 = \cos(\theta_i)(x_i+\delta_i y_i 2^{-i})$, (2.7.14)

$y_i+1 = \cos(\theta_i)(-\delta_i x_i 2^{-i}+y_i)$,

$x_i+1 = x_i \cos(\theta_i) + \delta_i y_i \sin(\theta_i)$,

$y_i+1 = \delta_i x_i \sin(\theta_i) + y_i \cos(\theta_i)$

with the value of δ_i deciding the direction of rotation.

Applying the condition $\tan(\theta_i) = 2^{-i}$, as stated in Eq. (2.7.12), for elementary angles, Eq. (2.7.13) may be rewritten as

$x_i+1 = \cos(\theta_i)(x_i + \delta_i y_i 2^{-i})$,

$y_i+1 = \cos(\theta_i)(-\delta_i x_i 2^{-i} + y_i)$, (2.7.14)

Let us consider a second set of iterative equations similar to Eq. (2.7.14) but dropping the cosine terms as follows :

$x'_i+1 = x'_i + di\ y'_i 2^{-i}$,

$y'_i+1 = -\delta_i\ x'_i\ 2^{-i} + y'^{i}_{i}$, (2.7.15)

If M number of iteration steps are carried out with the same starting co-ordinate $(x_0,y_0) = (x'_0,y'_0)$, then the end results provided by Eq. (2.7.14) and (2.7.15) are related as follows:

$$X'_M = \zeta_M x_M, \quad (2.7.16)$$

and

$$y'_M = \zeta_M y_{as},$$ where

$$\zeta_M = 1/\left(\prod_{i=0}^{M-1} \cos\theta_i\right)$$

The scaling factor ζ_M depends only on the word length and approaches asymptotically a constant value of 1/0.607252935. The iterations depicted in Eq. (2.7.15) can be easily implemented in digital hardware since multiplication with the term

2^{-i} is nothing but a shifting of the operand through i-bit position towards right. Thus, a plane rotation by an arbitrary angle can be accomplished by to and fro elementary rotations where the direction of next elementary rotation is determined by the sign of the present error as follows :

di - Sing (ε_i), (2.7.17)

$\varepsilon_{i+1} = \varepsilon_i - \delta_i\theta_i$ with $\varepsilon_0 = \alpha$.

Starting with co-ordinate $(P_{re} - Q_{re}, P_{im} - Q_{im})$ as (x'_0, y'_0) and target angle $\alpha = k\theta$, running iterations (2.7.15) on the same for M times yields the following results.

$y'_M = \zeta_M (P_{re} - Q_{re}) \sin(k\theta) + \zeta_M (P_{im} - Q_{im}) \cos(k\theta)$.

Expressions in Eq. (2.7.18) are identical to the last two equations in the set of Eq. (2.7.11)

depicting the butterfly operation excepting the scaling factor ζ_M which may be thought of as a constant gain factor, but in the case, the first two equations in (2.7.11) must also include the same scaling terms. Since, $\zeta_M > 1$, a further scaling is carried out in the actual implementation by shifting the result to one bit position right (which is equivalent to a division by two) to reduce the chance of overflow. With this scheme, the butterfly operation having a constant gain term becomes as follows :

$R'_{re} = \zeta'_M (P_{re} + Q_{re})$, (2.7.19)

$R'_m = \zeta'_M (P_{im} + Q_{im})$,

$S'_{re} = \zeta'_M (P_{re} - Q_{re}) \cos(k\theta) + \zeta'_M (P_{im} - Q_{im}) \sin(k\theta)$,

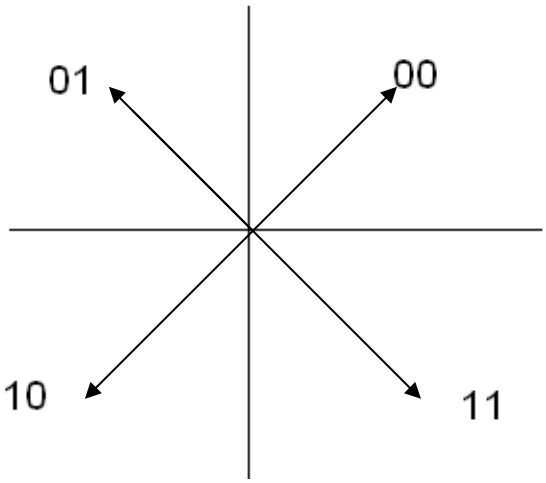
$S'_{im} = \zeta'_M (P_{re} - Q_{re}) \sin(k\theta) + \zeta'_M (P_{im} - Q_{im}) \cos(k\theta)$,

where $\zeta'_M = \zeta_M/2$.

Finally, The binary format for the representation of the error angle ε is chosen in a novel way which offers two special advantage - the first one being the ease of extending the range of rotation / vectoring from $\pm \pi/2$ to $\pm \pi$ covering all the quadrants completely, and secondly the scope of representing the rotation angle $k\theta$ for any butterfly stage terms of the multiplier k

only, thereby relieving the burden of multiplication operation for the purpose. The representation of ε may be thought of as a normalized angle in two's complement format, since the weights chosen from the MSB side are - π, π/2, π/4,...., $\pi/2^{M-1}$;M being the word length. This form of angle representation rather than in the conventional radians makes it possible to readily identify the quadrant pertaining to the amount of rotation simply by observing the first two bits from the MSB side. Since the CORDIC algorithm can easily accommodate rotation in the range ± π/2 only, so whenever the required rotation α is beyond that range, that is either $\pi/2 \leq \alpha < \pi$ (second quadrant) or $-\pi \leq \alpha < -\pi/2$ (third quadrant, as the case may be, is considered by adding/subtracting an amount equal to π from the actual angle α which can be carried out by complementing the MSB, and that value is used instead. To have proper results, the signs of the x and y components are changed while outputting their final values, to incorporate .the effect of reflections about the axes. This is illustrated in Fig. 2.7.5

For having implementation advantages, certain variations are incorporated during the realization of the actual architecture. One such modification is in choosing the directional parameter δ_i during vectoring, Instead of deriving the direction of rotation from the sign of y_i alone, an XNOR function of the signs of x_i and y_i used to accommodate vectoring in all the four quadrants as illustrated in Fig. 2.7.5



Fig(2.7.5) :Mapping of quadrant

/*Algorithm for the adders generation to compute $N=2^n$ point FFT*/

For stage=n-1 down to 0

Ncluster=2^(n-1 – stage)

Npair=2^stage

For cluster=ncluster –1 down to 0

Index=cluster * 2^(stage+1)

Rot=0

For pair=npair –1 down to 0

Call Butterfly(index,index+npair,rot)

Index=index+1

Rot=rot + ncluster

Next pair

Next cluster

Next stage

End

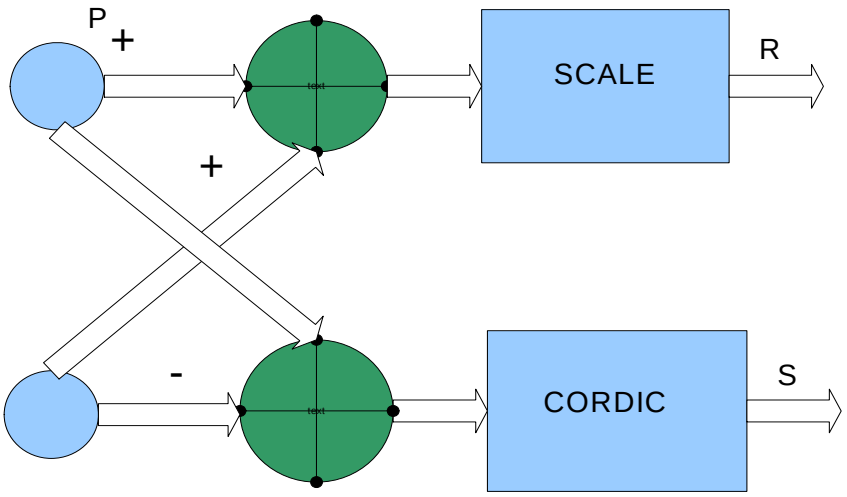
3.9.3 THE ADDRESS GENERATION UNIT

The address generation unit implements the following algorithm to realize the signal flow graph for the computation of FFT. It generates the address where from the data is to be fetched to the butterfly unit and where the result is to be written too. It also computes the angle multiplier k (under the variable name 'rot') required for the butterfly operation.

The novel angle representation scheme, i.e. choosing the weights as π, $\pi/2$, $\pi/4$,..., $\pi/2^{M-1}$ (M being the wordlength), offers the tremendous advantage of relieving the designer for the requirement of having a multiplier for computing $k\theta$ as required by the CORDIC unit to evaluate Eq. (2.7.12) from the value of the multiplier k (which is stored as the variable 'rot' in the address computation algorithm). As the value of θ equals $2\pi/N$ or $\pi/2$ n-1, it is represented by a string of zeroes except a one at the nth bit position from left hand side, while k is a n-bit number with the MSB being zero (since k ranges from 0 to N/2 -1). Therefore, multiplying k with θ yields a M bit number having the least significant n - 1 bits of k as the first n - 1 (most signifcant) bits with the rest of the bits equal to zero.

3.9.4 Butterfly unit

The butterfly unit implements Eq. (2.7.19) whose flow graph is shown in Fig. 2.7.6. The CORDIC module is the heart of the butterfly unit which accepts (P_{re}-Q_{re}) and (P_{im}-Q_{im}) as the two components of the input vector and provides the output vector after rotating the input by an amount of kθ. However, the output is scaled by a factor of 1/0.607252935 =1.646759 as indicated in Eq. (2.7.16). Instead of multiplying the result by 0.607252935 to restore the proper value of the output vector, the other output of the butterfly is also scaled by the same factor of 1.646759 to keep the uniformity which maintains the correct phase relationship between the output components but alters the magnitude of the output vector. This is permissible in digital



Fig(2.7.6):Butterfly with CORDIC

signal processing applications since relative amplitude of the components and not their absolute magnitudes are of primary concern and the scale factor appears as a fixed gain term in such context. To reduce the chances of overflow, however, a further scaling by a factor of "0.5" is carried out by shifting the result through one bit position towards right at both the output R and S. All the variables are represented in 16 bit two's complement format, therefore the CORDIC unit takes 16 clock cycles to compute the result and the block finishes its computation within that period adding on extra overhead for its operation.

First computes P - Q and passes the result to the CORDIC unit and then computes P + Q and passes the same to the unit while the CORDIC unit remains busy processing the

P - Q value.

The CORDIC block consists of ten add/subtract units implemented by carry-free adders and two shifters performing logical shift as well as arithmetic shifting along with the associated registers for implementing the iterative operation depicted in Eq. (2.7.15). The multiplexed registers can be loaded externally or can be updated with internal results. The combinatorial block is designed to serve as a lookup table ROM for storing the values of $\tan^{-1}(2^{-i})$ required for computing Eq. (2.7.12). Two extra add/subtract unit are employed as controlled two's complement negator for implementing rotations in second and third quadrants through reflections about the axes as has already been discussed in the previous section. Two extra unit one for binary to redundant conversion and other for converting redundant number back to binary 2's complement number . On completion of a butterfly rotation, it issues a termination signal which updates the parameter passing buffers at the address computation unit and also serves as an indication to itself to prepare for starting a fresh operation. The control line decides the mode of operation for the CORDIC unit with the help of two multiplexers - one governing the direction of rotation and another one driving the controller two's complementer for changing the signs of the outputs as per the tables. In the rotation mode, the direction of rotation is derived from the sign of the instantaneous error in angle computation, that is the difference between the target angle and the achieved angle. However, at the time of loading the target angle, the specially designed register copies 12-th bit as the MSB (13th bit) instead of the original value of MSB which has got an weight age equals to-π. This is carried out in order to keep the target angle a within that limit $-\pi/\leq a<\pi/2$. If the target angle is beyond that limit, i.e. if it lies either in the second or the third quadrant, then its 13th bit (MSB) and 12th bit disagree with each other and in that case, the signs of the output variables X and Y are changed to incorporate the reflections about the axes, as described in the previous section. During vectoring, the direction of rotation is determined depending upon the current position of the subject vector in terms of the quadrant which is decided by checking the signs of X and Y. Here also, as in rotation, a change in the sign of the final output is required, if the vector lies either in the second or in the third quadrant which is indicated by the sign bit of X.

CHAPTER 4

SYSTEM DESIGN

4.1 Binary to redundant conversion

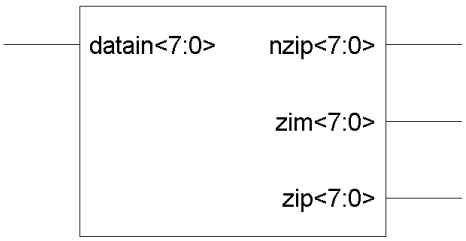
Binary to redundant conversion is simple and direct. If input number is in 2's complement form and the number is negative the only thing is to change most significant bit ie: MSB from '1' to '-1'.For a positive number no need to perform the conversion because in SD number system with digit set {-1,0,1} the value of binary as well as redundant number for a positive number is same. To realize negative numbers, coding is done. Below example shows the coding scheme for the all three possible digits of the digit set {-1,0,1}.

-1 by "01"

1 by "10"

0 by "00".

This binary to redundant conversion is performed by the entity "BINTOREND8".



Fig(8.1) BINTOREND8

Name	Type	Description
datain	In (9 down to 0) std_logic_vector	Input binary number
zip	out (9 down to 0) std_logic_vector	Redundant binary number
zim	out (9 down to 0) std_logic_vector	Redundant binary number

Table 8.1

Fig 8.1 shows the top level (RTL view) for the entity BINTOREND8 which performs binary to redundant conversion. It takes eight bit input binary number in the 2's complement form and provides three eight bit outputs, zip, zim, nzip respectively.

For eg: If datain="01000000" then

Zip="01000000".

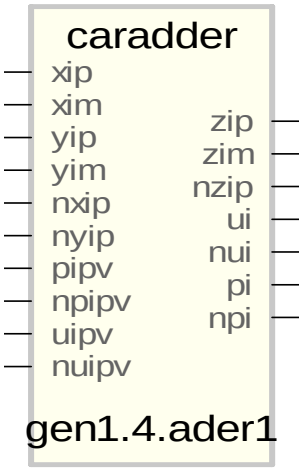
Zim="00000000".

NZip ="10111111".

Here complement of Zip is generated because of its requirement to perform carry-free-addition.Table8.1 shows the specifications of the entity "BINTOREND8".

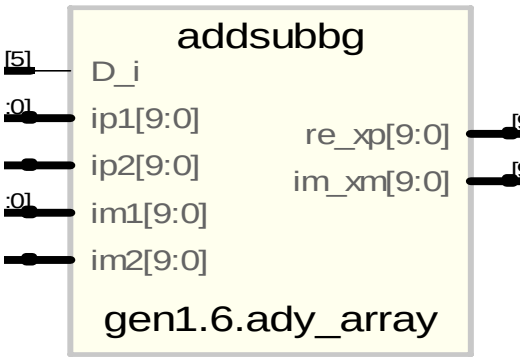
4.2 CARRY FREE ADDER:

The algorithm as explained in earlier section is used to implement the adder. First a 1 bit adder is designed under the entity name CARADDER using 4-input NOR/OR gate. Then by cascading this 1 bit adder 10 bit adder under entity name ADDER8 is designed. Fig(8.2) shows the RTL view of the one bit adder.



Fig(8.2) RTL view of 1 bit adder(CARADDER)

Here Xip, Xim, Nxip and Yip, Yim, NYip are two input redundant numbers. Initially "Pipv" and "Uipv" both are taken to be equal to '1' while their complements "npipv", "nuipv" both taken to be equal to '0'.



Fig(8.3) 10 bit adder (ADDER8)

Name	Type	Description
Xip	In (9 down to 0) std_logic_vector	I/P Redundant binary number
Xim	In (9 down to 0) std_logic_vector	I/P Redundant binary number
NXip	In (9 down to 0) std_logic_vector	I/P Redundant binary number
Yip	In (9 down to 0) std_logic_vector	I/P Redundant binary number
Yim	In (9 down to 0) Std_logic_vector	I/P Redundant binary number
NYip	In (9 down to 0) Std_logic_vector	I/P Redundant binary number
Popv	Std_logic	I/P taken to be ‘1’
Uopv	Std_logic	I/P taken to be ‘1’
NPopv	Std_logic	I/P taken to be ‘0’
NUopv	Std_logic	I/P taken to be ‘0’
Zip	Out (9 down to 0) Std_logic_vector	O/P Redundant binary number
Zim	Out (9 down to 0) Std_logic_vector	O/P Redundant binary number

NZip	Out (9 down to 0) Std_logic_vector	O/P Redundant binary number
P8	Std_logic	O/P
U8	Std_logic	O/P
NP8	Std_logic	O/P
NU8	Std_logic	O/P

4.3 CARRY SELECTOR:

Carry selector will provide carry 'Ci' while inspecting the input S &D form the previous stage "SDbreak". Algorithm has already explained in the previous section.

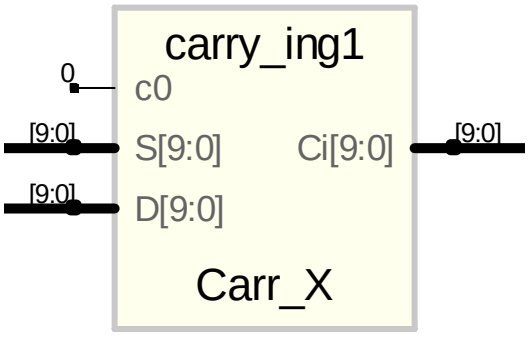


Fig (8.4) Carry_in

Fig(8.4) shows the RTL view for the entity "Carry_in" that performs the carry selection. As per the algorithm it takes two eight bit S and D along with C_0 ='0' as inputs and provides eight bit carry C_i as output. This carry C_i is used for redundant to binary conversion.

Name	Type	Description
S	In (9 down to 0) std_logic_vector	S obtained for previous stage SDbreaker
D	In (9 down to 0) std_logic_vector	D obtained for previous stage SDbreaker
C0	In std_logic	C0='0';
Ci	out (9 down to 0)	Carry Ci out goes to

	std_logic_vector	input redundant to binary converter

Table :8.4

Table :8.4 shows the specifications for the entity “Carry_in” .

4.4 S & D selector:

With given redundant number ‘Z’ represented in terms of Zip and Zim , we compute S,D form it. After extracting S ,D we can compute carry Ci, in the stage carry_in.

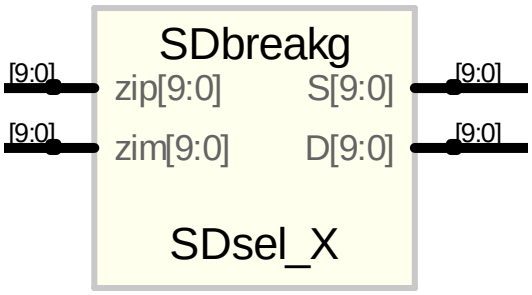


Fig (8.5) SDbreaker

Name	Type	Description
zip	In (9 down to 0) std_logic_vector	I/p redundant number
zim	In (9 down to 0) std_logic_vector	I/p redundant number
D	In (9 down to 0) std_logic_vector	O/p ,8 bit
S	In (9 down to 0) std_logic_vector	O/p ,8 bit

Table:8.5

Fig(8.5) shows the RTL view of the entity “SDBreak”. The S and D selection from the given Zip,Zim will takes place as shown below in the Table 8.5.1:

Zip	Zim	S	D
0	1	1	1
0	0	0	0
1	0	0	1

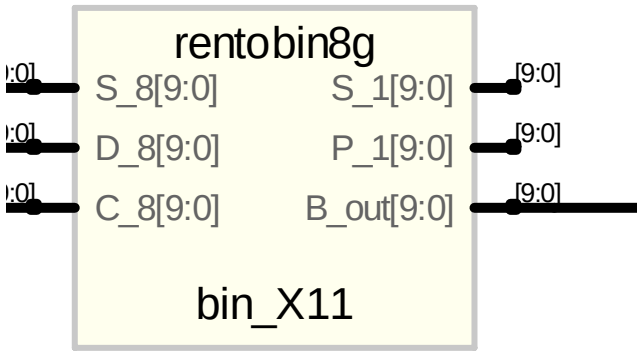
Table 8.5.1

4.5 Redundant to binary converter:

Redundant to binary converter will take C_8 (Carry_in) ,S ,D ,as input and provide eight bit binary output. The algorithm is already explained in earlier section.Fig(8.6) shows the RTL view for the entity “REND2BIN8” that performs the conversion of given redundant number to equivalent binary number.

The Binary out put B_out is

$B\_out = D\_8 \oplus C\_8.$



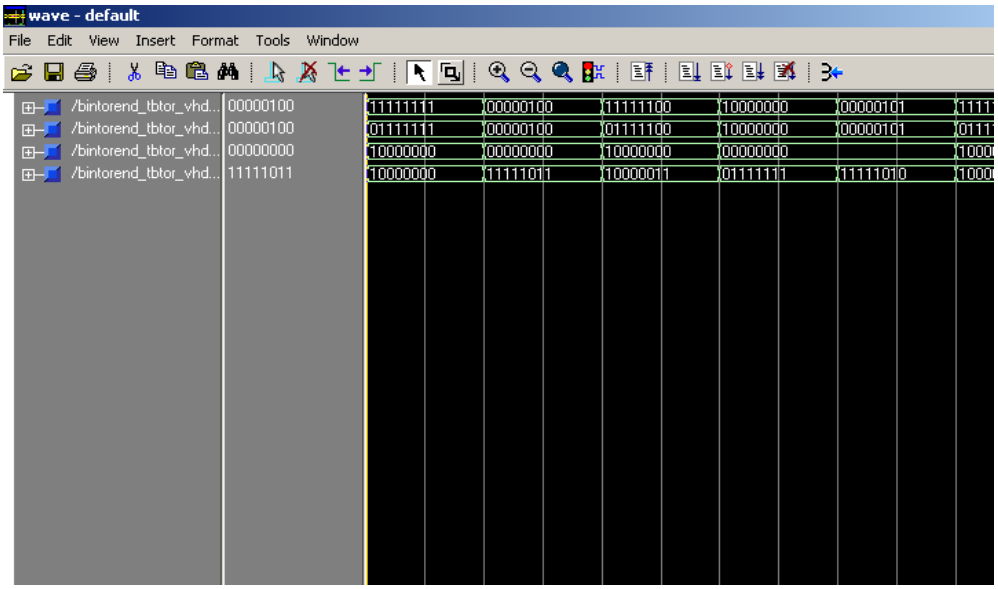
Fig(8.6) redundant to binary conversion

Name	Type	Description
C_8	In (9 down to 0) std_logic_vector	10 bit carry input
S_8	In (9 down to 0) std_logic_vector	10 bit S i/p form previous stage
D-8	In (9 down to 0) std_logic_vector	10 bit D i/p form previous stage
B_out	out (9 down to 0)	10 bit binary o/p

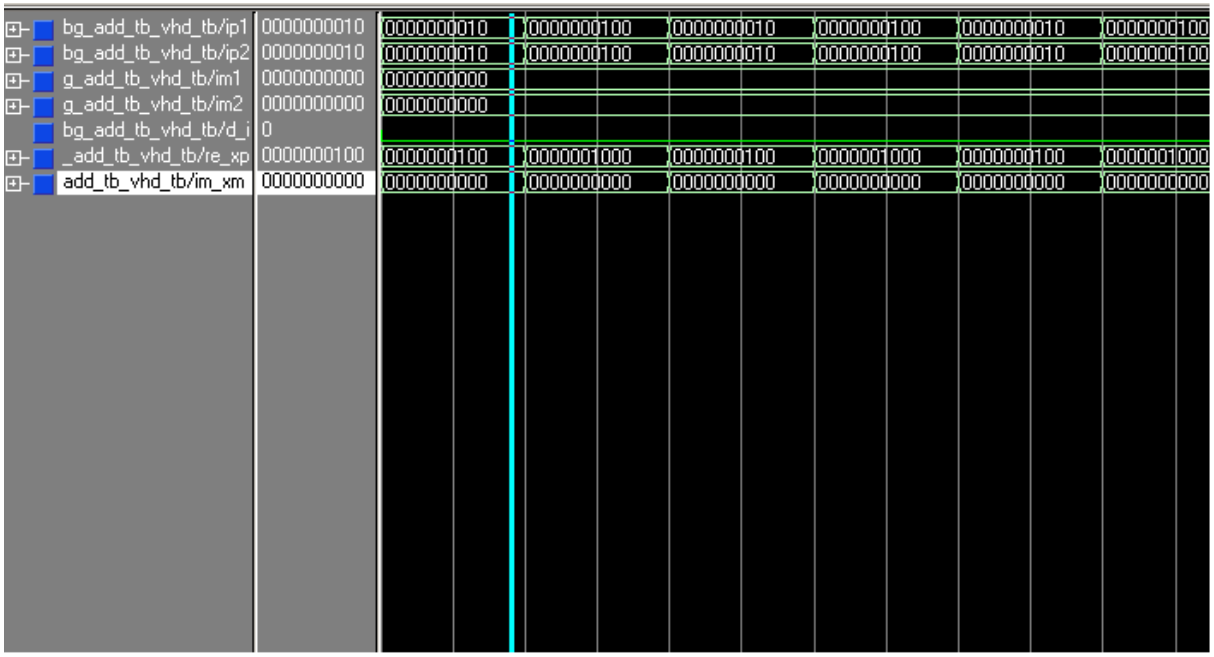
	std_logic_vector	
S_1	Out (9 down to 0) std_logic_vector	Complement of S
P_1	out (9 down to 0) std_logic_vector	Complement of D

Table:8.6

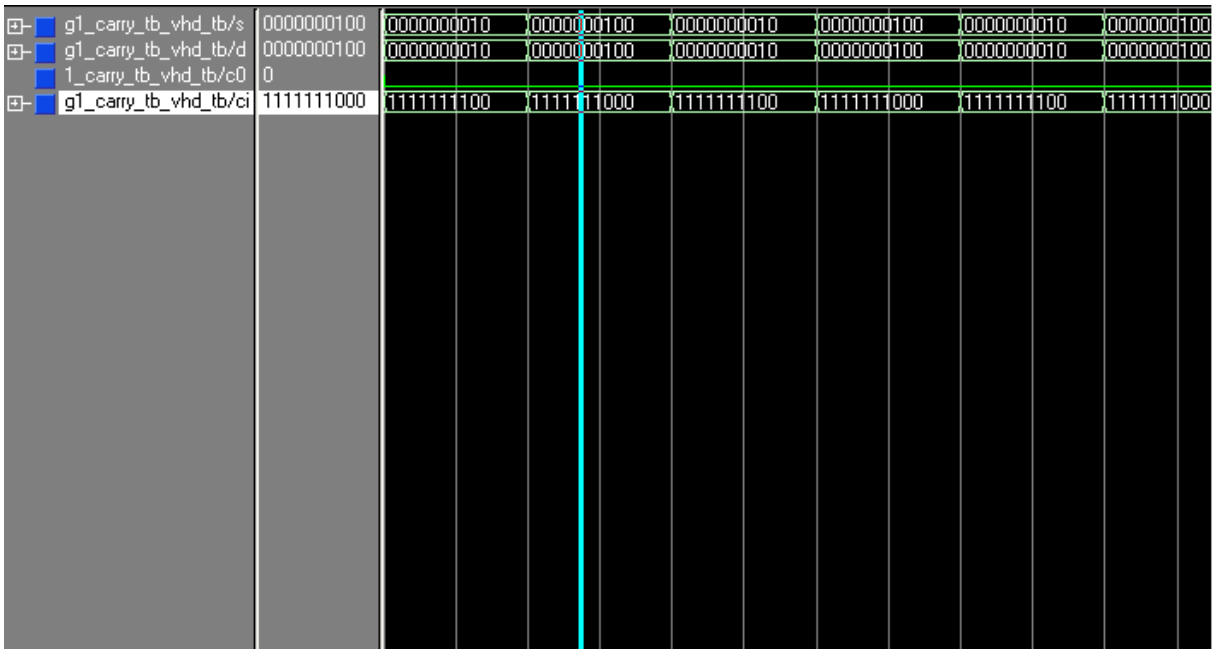
Table 8.6 shows the specifications of the entity “REND2BIN8”.



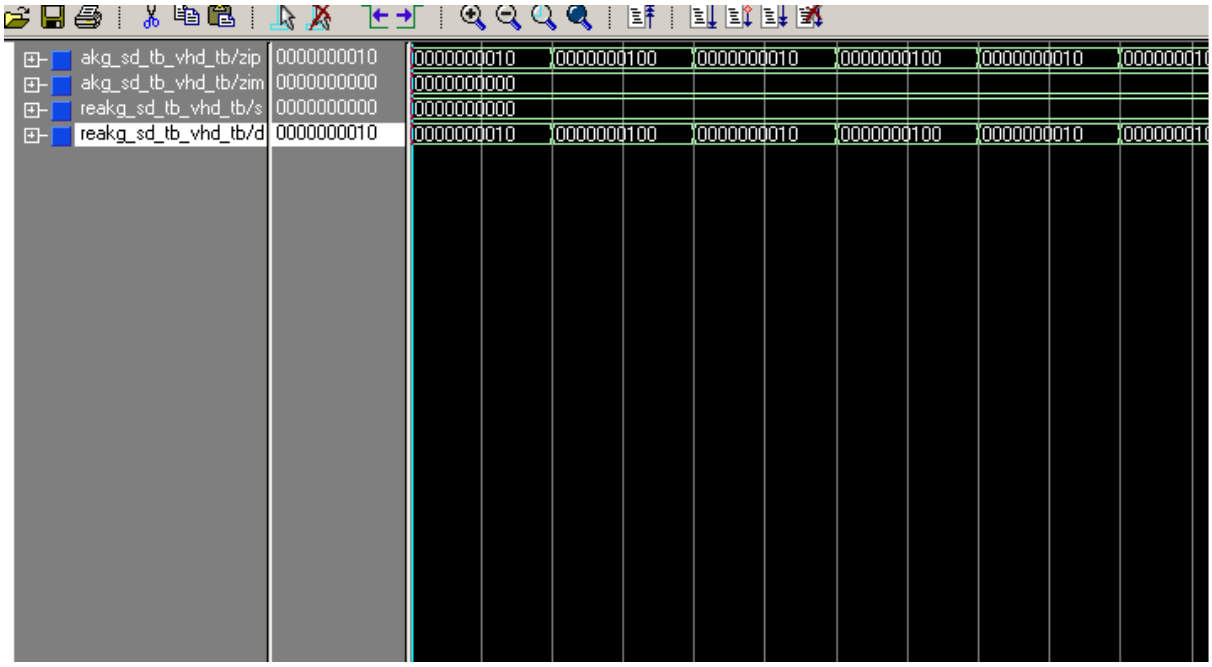
FIG(7.1)BINARY TO REDUNDANT CONVERTOR SIMULATION



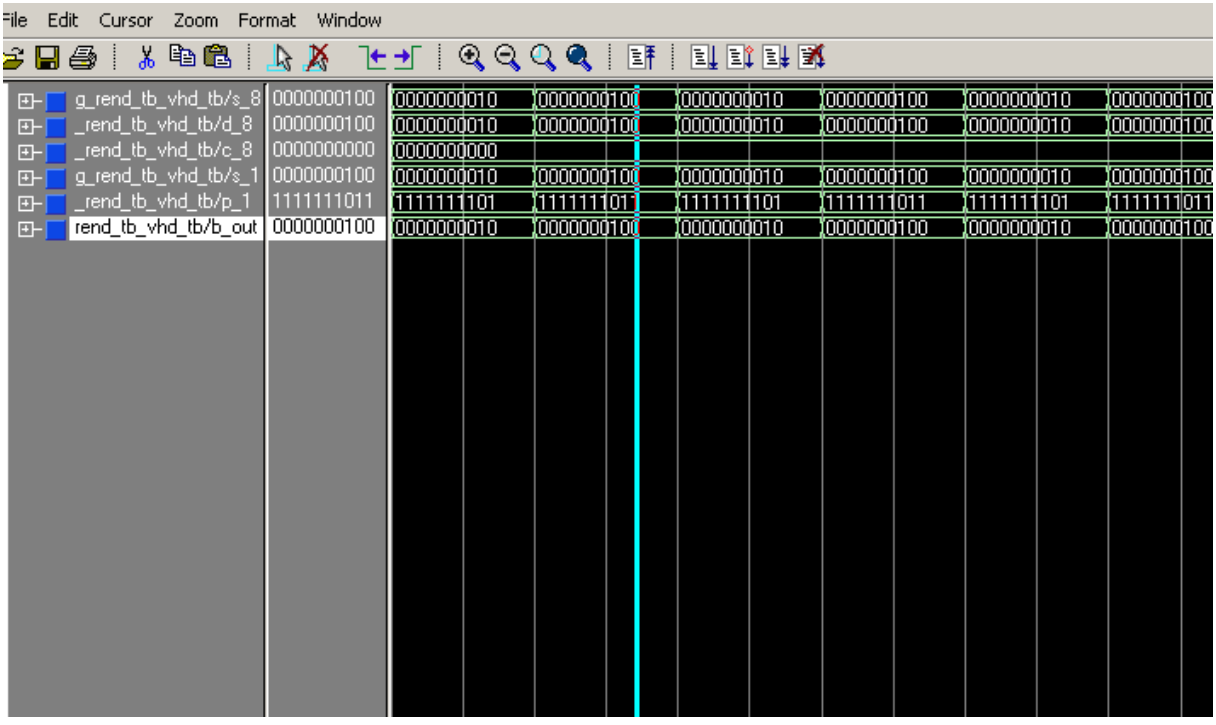
FIG(7.2)10 BIT CARRY FREE ADDER



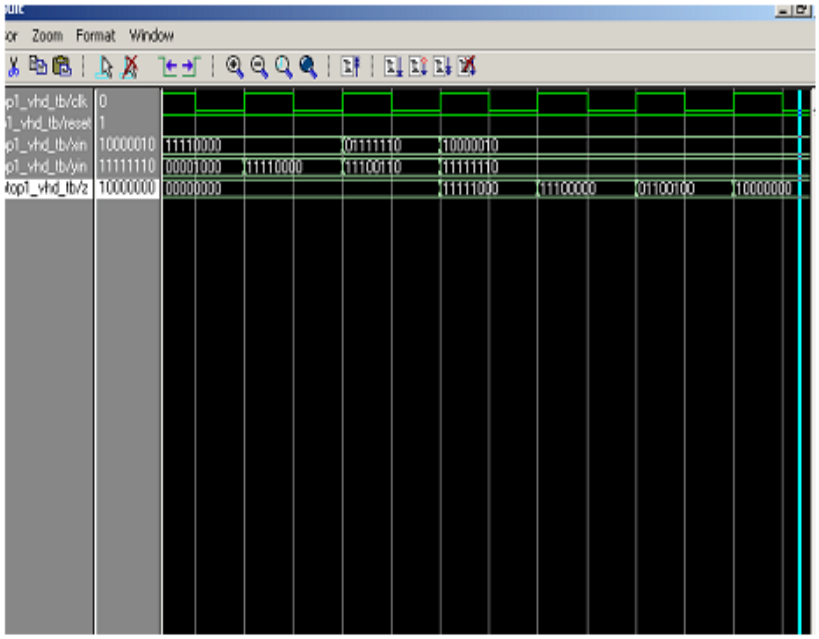
FIG(7.3)CARRY SELECTOR



FIG(7.4) S & D SELECTOR(SDBREAKER)



FIG(7.5) REDUNDANT TO BINARY CONVERTER(RENTOBIN8)



FIG(7.6) TOP LEVEL ENTITY(TOP)

1Bit Carry free adder

Entity name:CARADDER:

Device utilization summary:

Selected Device : 2v4000bf957-6

Number of Slices: 6 out of 23040 0%

Number of 4 input LUTs: 10 out of 46080 0%

Number of bonded IOBs: 17 out of 684 2%

Timing report:

Speed Grade: -6

Minimum period: No path found

Minimum input arrival time before clock: No path found

Maximum output required time after clock: No path found

Maximum combinational path delay: 7.273ns

Design Summary:

Number of errors: 0

Number of warnings: 0

Logic Utilization:

Number of 4 input LUTs: 10 out of 46,080 1%

Logic Distribution:

Number of occupied Slices: 6 out of 23,040 1%

Number of Slices containing only related logic: 6 out of 6 100%

Number of Slices containing unrelated logic: 0 out of 6 0%

Total Number 4 input LUTs: 10 out of 46,080 1%

Number of bonded IOBs: 17 out of 684 2%

Total equivalent gate count for design: 60

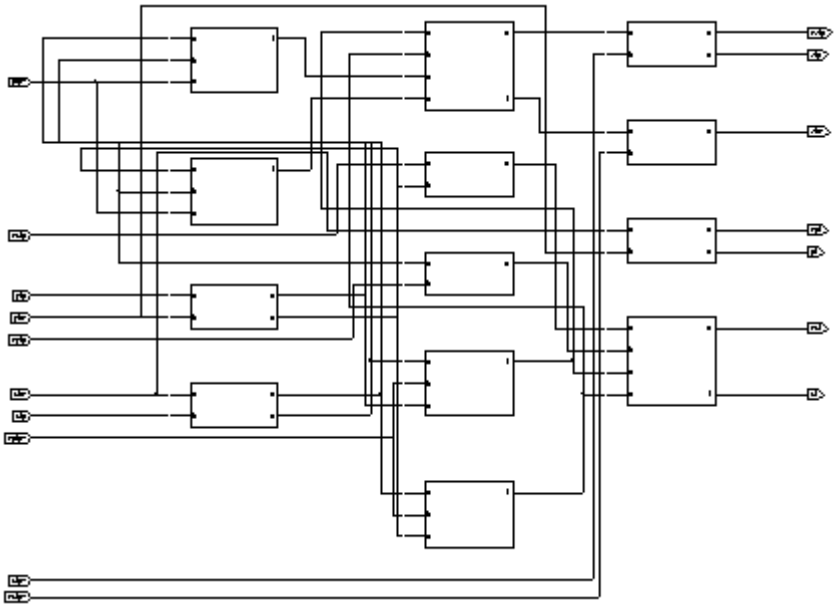


Fig (6.1):RTL of 1 bit adder

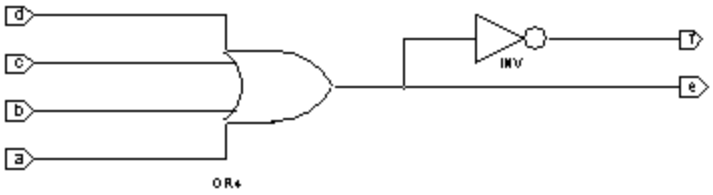


Fig (6.1.1):Gate or/ nor

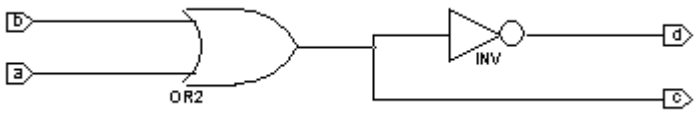


Fig (6.1.2):Gate or/nor

4.2 10 BIT CARRY FREE ADDER

Entity name :ADDER8

Device utilization summary:

Selected Device : 2v4000bf957-6

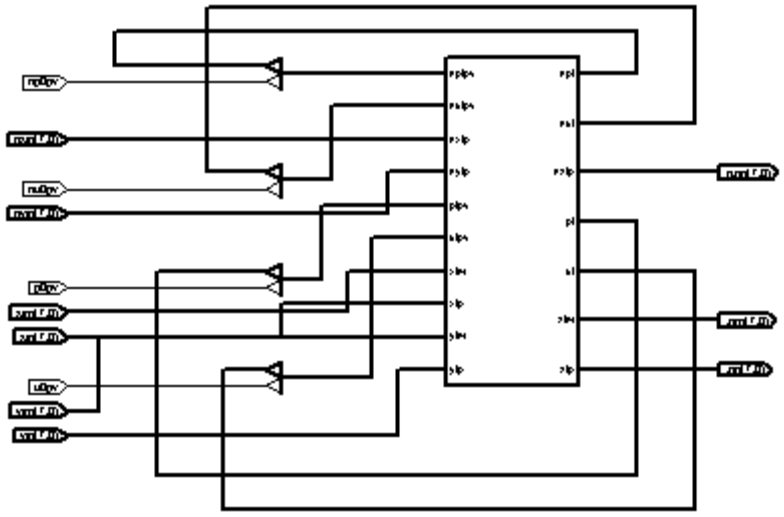
Number of Slices:	33	out of 23040	0%
Number of 4 input LUTs:	59	out of 46080	0%
Number of bonded IOBs:	80	out of 684	11%

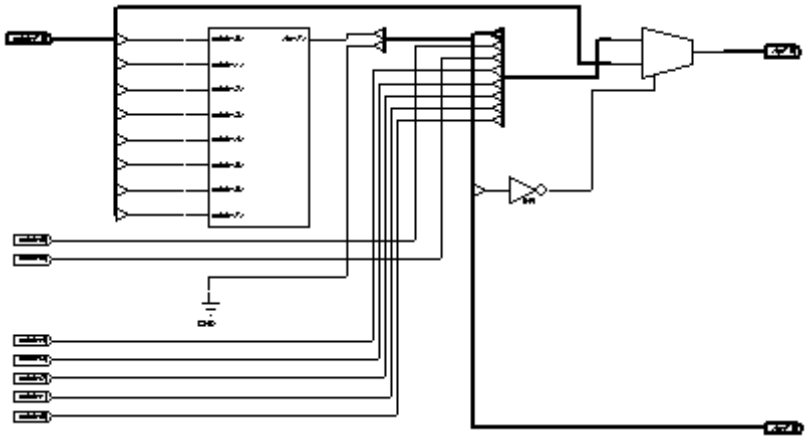
Timing report:

Speed Grade: -6

Maximum combinational path delay: 8.071ns

Fig (6.2.):RTL of 10 bit ADDER





HDL Synthesis Report

Macro Statistics

Multiplexers : 1

8-bit 2-to-1 multiplexer : 1

Device utilization summary:

Selected Device : 2v4000bf957-6

Number of Slices: 2 out of 23040 0%

Number of 4 input LUTs: 4 out of 46080 0%

Number of bonded IOBs: 24 out of 684 3%

Timing report:

Speed Grade: -6

Maximum combinational path delay: 7.239ns

BINARY TO REDUNDANT CONVERTER:

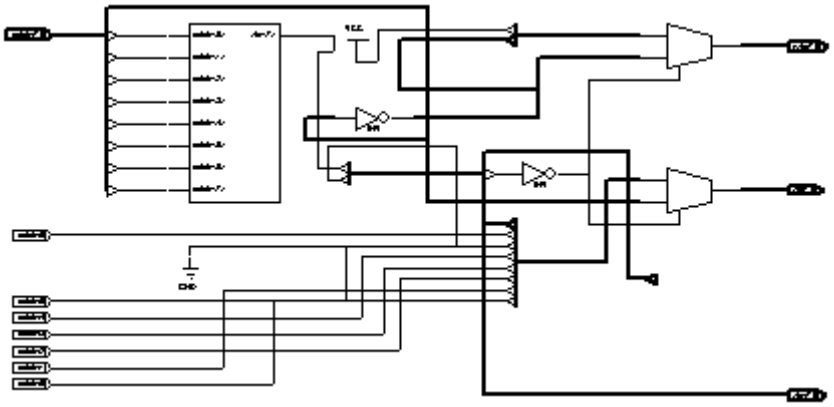
Entity name :BINTOREND

HDL Synthesis Report

Macro Statistics

Multiplexers : 2

10-bit 2-to-1 multiplexer : 2



Fig(6.3.1)Binary to redundant converter

Design Summary:

Number of errors: 0

Number of warnings: 0

Logic Utilization:

Number of 4 input LUTs: 5 out of 46,080 1%

Logic Distribution:

Number of occupied Slices: 3 out of 23,040 1%

Number of Slices containing only related logic: 3 out of 3 100%

Number of Slices containing unrelated logic: 0 out of 3 0%

Total Number 4 input LUTs: 5 out of 46,080 1%

Number of bonded IOBs: 32 out of 684 4%

Total equivalent gate count for design: 30

CARRY SELECT LOGIC:

Entity name :CARRY_IN

Selected Device : 2v4000bf957-6

Number of Slices: 6 out of 23040 0%

Number of 4 input LUTs: 10 out of 46080 0%

Number of bonded IOBs: 23 out of 684 3%

Timing report

Speed Grade: -6

Minimum period: No path found

Minimum input arrival time before clock: No path found

Maximum output required time after clock: No path found

Maximum combinational path delay: 9.004ns

Device utilization summary:

Number of External IOBs	23 out of 684	3%
Number of LOCed External IOBs	0 out of 23	0%
Number of SLICEs	6 out of 23040	1%

Design Summary:

Number of errors: 0

Number of warnings: 0

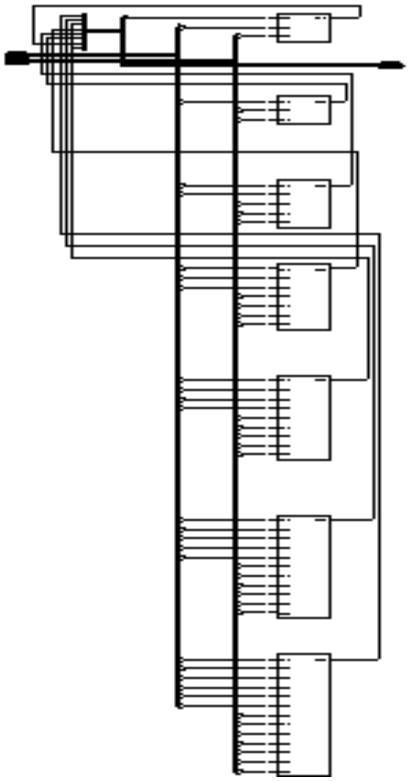
Logic Utilization:

Number of 4 input LUTs:	10 out of 46,080	1%

Logic Distribution:

Number of occupied Slices:	6 out of 23,040	1%
Number of Slices containing only related logic:	6 out of 6	100%
Number of Slices containing unrelated logic:	0 out of 6	0%
Total Number 4 input LUTs:	10 out of 46,080	1%
Number of bonded IOBs:	23 out of 684	3%

Total equivalent gate count for design: 60



Fig(6.4) RTL of Carry selector (Carry_in)

REDUNDANT TO BINARY NUMBER CONVERTER:

Entity name :RENTOBIN8

Device utilization summary:

Selected Device : 2v4000bf957-6

Number of Slices:	9	out of 23040	0%
Number of 4 input LUTs:	16	out of 46080	0%
Number of bonded IOBs:	48	out of 684	7%

TIMING REPORT

Timing Summary:

Speed Grade: -6

Minimum period: No path found

Minimum input arrival time before clock: No path found

Maximum output required time after clock: No path found

Maximum combinational path delay: 5.644ns

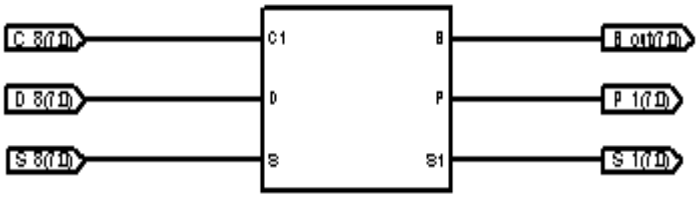
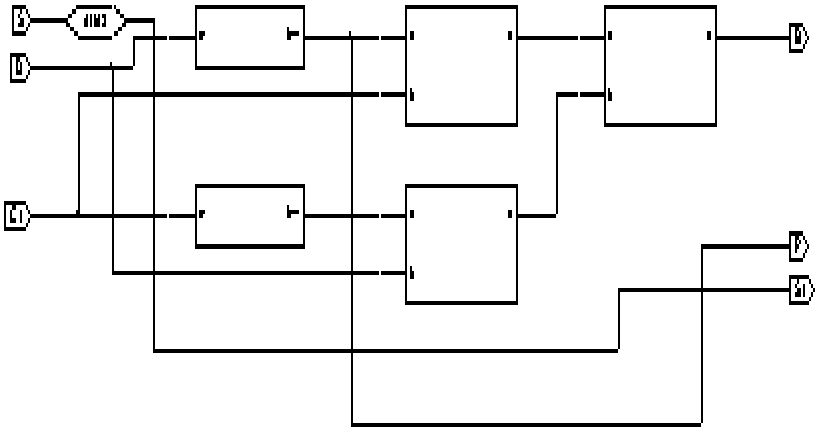
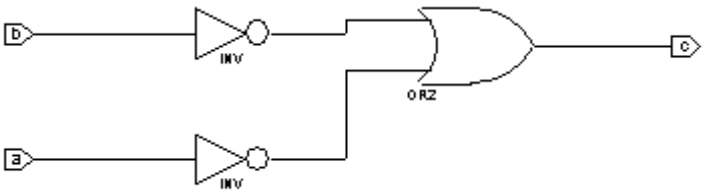


Fig (6.3):Top level redundant to binary converter



Fig(6.4.1):Internal of redundant to binary converter



Design Summary:

Number of errors: 0

Number of warnings: 0

Logic Utilization:

Number of 4 input LUTs: 8 out of 46,080 1%

Logic Distribution:

Number of occupied Slices: 8 out of 23,040 1%

Number of Slices containing only related logic: 8 out of 8 100%

Number of Slices containing unrelated logic: 0 out of 8 0%

Total Number 4 input LUTs: 8 out of 46,080 1%

Number of bonded IOBs: 48 out of 684 7%

Total equivalent gate count for design: 48

S AND D SELECTOR

Entity name :SDBREAK

HDL Synthesis Report

Macro Statistics

Multiplexers : 8

1-bit 2-to-1 multiplexer : 8

Device utilization summary:

Selected Device : 2v4000bf957-6

Number of Slices: 9 out of 23040 0%

Number of 4 input LUTs: 16 out of 46080 0%

Number of bonded IOBs: 32 out of 684 4%

Timing Summary:

Speed Grade: -6

Minimum period: No path found

Minimum input arrival time before clock: No path found

Maximum output required time after clock: No path found

Maximum combinational path delay: 5.644ns

Design Summary:

Number of errors: 0

Number of warnings: 0

Logic Utilization:

Number of 4 input LUTs: 16 out of 46,080 1%

Logic Distribution:

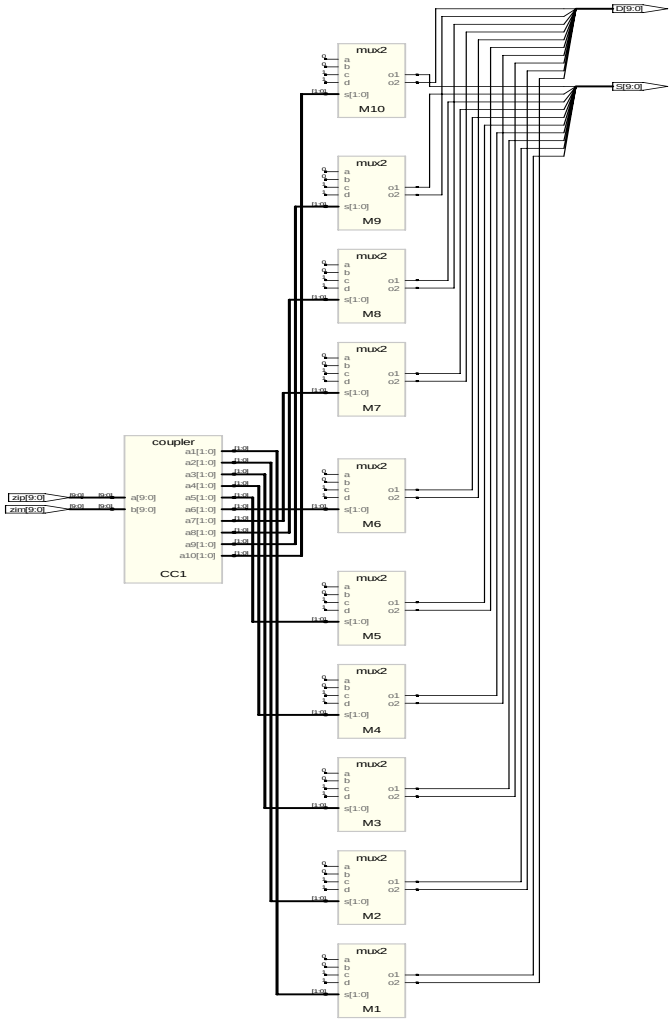
Number of occupied Slices: 8 out of 23,040 1%

Number of Slices containing only related logic: 8 out of 8 100%

Number of Slices containing unrelated logic: 0 out of 8 0%

Total Number 4 input LUTs: 16 out of 46,080 1%

Total equivalent gate count for design: 96



Fig(6.5) RTL of S & D selector (SD Breaker)

TOP LEVEL ENTITY:

Entity name :TOP

selected Device : 2v4000bf957-6

Number of Slices: 24 out of 23040 0%

Number of 4 input LUTs: 43 out of 46080 0%

Number of bonded IOBs: 44 out of 684 6%

Timing Summary:

Speed Grade: -6

Minimum period: No path found

Minimum input arrival time before clock: No path found

Maximum output required time after clock: No path found

Maximum combinational path delay: 10.820ns

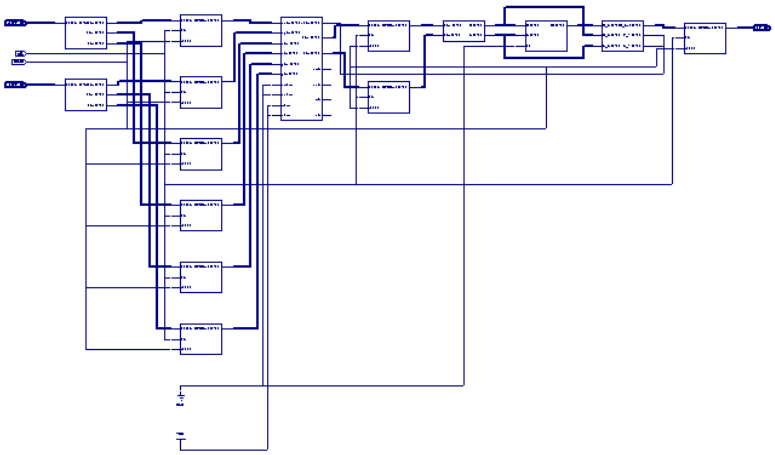


Fig (6.6) RTL of Top

Design Summary:

Number of errors: 0

Number of warnings: 0

Logic Utilization:

Number of 4 input LUTs: 43 out of 46,080 1%

Logic Distribution:

Number of occupied Slices: 23 out of 23,040 1%

Number of Slices containing only related logic: 23 out of Number of Slices containing unrelated logic: 0 out of Total Number 4 input LUTs: 43 out of 46,080 1%

Number of bonded IOBs: 44 out of 684 6%

Total equivalent gate count for design: 258

MATLAB SIMULATION RESULTS

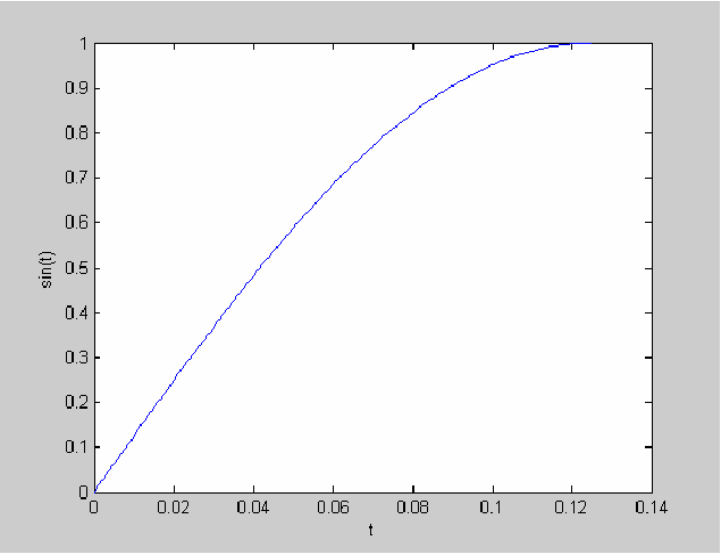
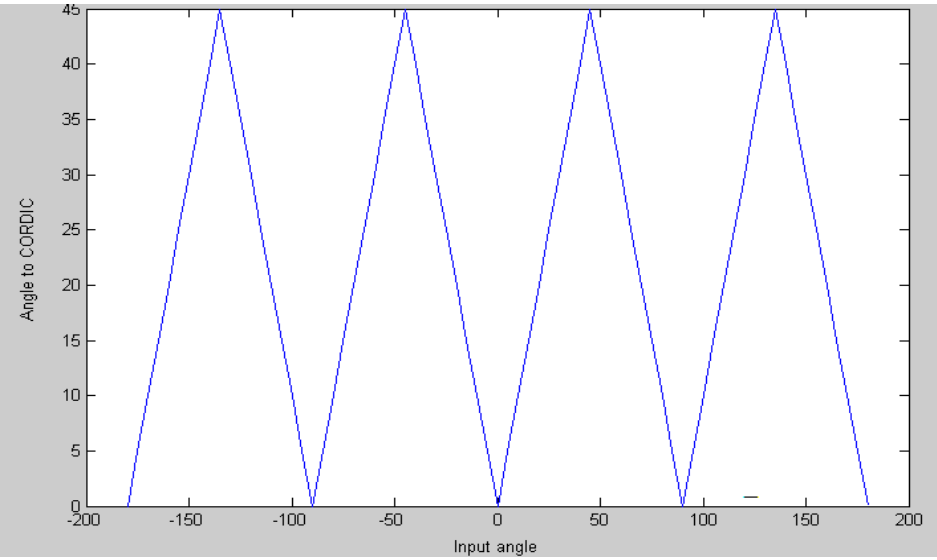
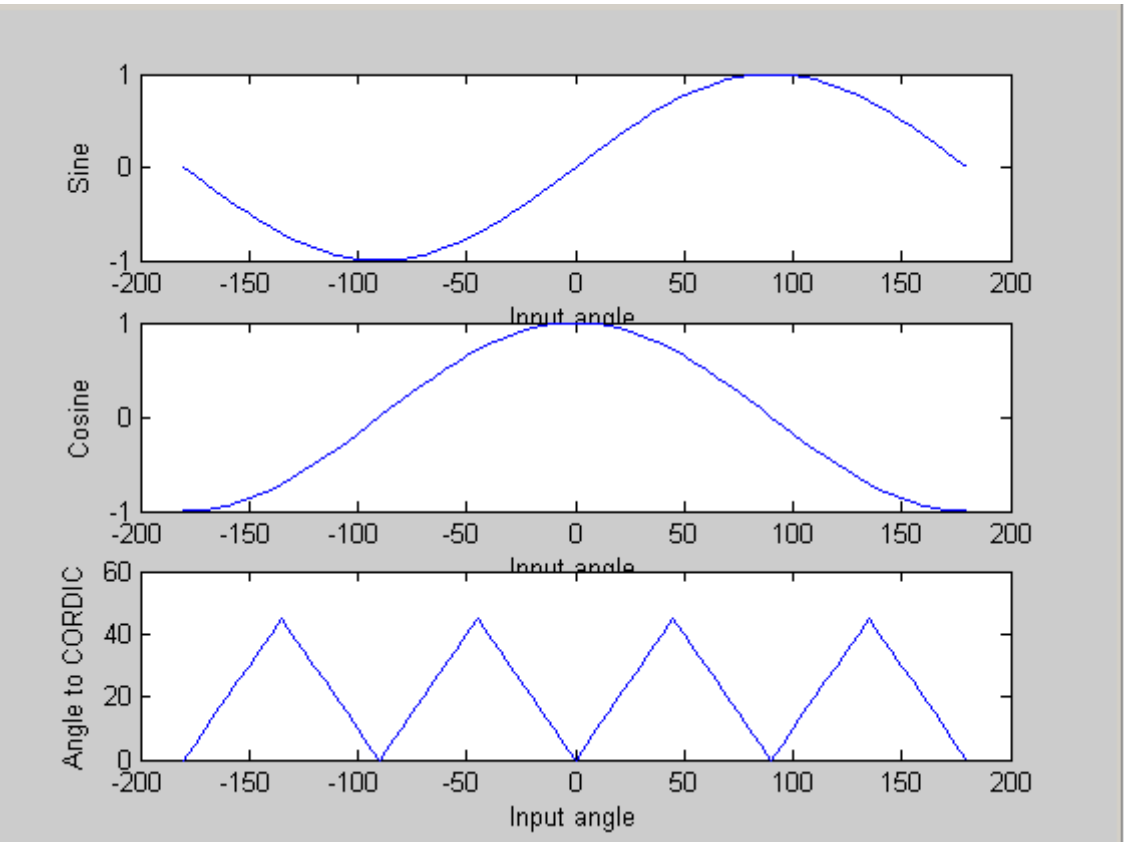


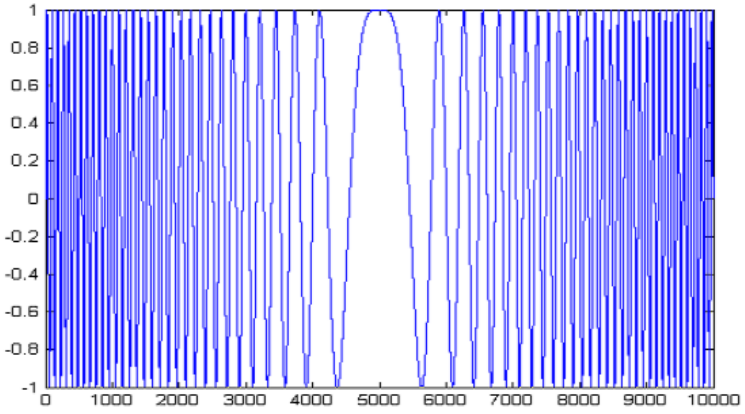
Fig (5.1) Sine wave.



Fig(5.7)Angle mapping for the predicting CORDIC algorithm.



Fig(5.8) Simulation result of predicting CORDIC



Fig(5.9)Chirp values vs samples

4.6 CORDIC IMPLEMENTATION IN AN FPGA:

There are a number of ways to implement a CORDIC processor. The ideal architecture depends on the speed versus area tradeoffs in the intended application. First we will examine an iterative architecture that is a direct translation from the CORDIC equations. From there, we will look at a minimum hardware solution and a maximum performance solution.

Iterative CORDIC Processors:

An iterative CORDIC architecture can be obtained simply by duplicating each of the three difference equations in hardware as shown in figure 1. The decision function, d_i is driven by the sign of the y or z register depending on whether it is operated in rotation or vectoring mode. In operation, the initial values are loaded via multiplexers into the x, y and z registers. Then on each of the next n clock cycles, the values from the registers are passed through the shifters and adder-subtractors and the results placed back in the registers. The shifters are modified on each iteration to cause the desired shift for the iteration. Likewise, the ROM address is incremented on each iteration so that the appropriate elementary angle value is presented to the z adder-subtractor. On the last iteration, the results are read directly from the adder-subtractors. Obviously, a simple state machine is required keep track of the current iteration, and to select the degree of shift and ROM address for each iteration.

The design depicted in Figure 1 uses word-wide data paths (called bit-parallel design). The bit-parallel variable shift shifters do not map well to FPGA architectures because of the high fan-in required. If implemented, those shifters will typically require several layers of logic (i.e., the signal will need to pass through a number of FPGA cells). The result is a slow design that uses a large number of logic cells.

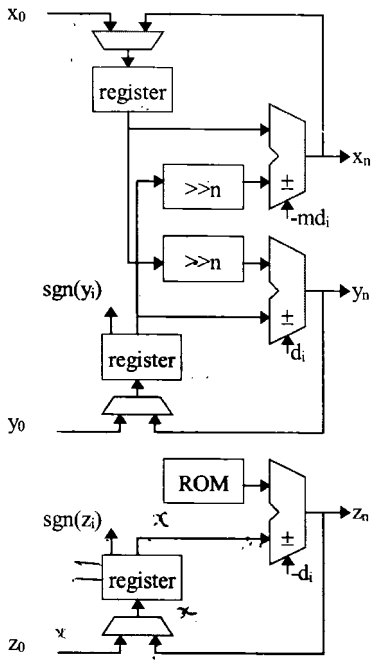


Figure 1. Iterative CORDIC structure

A considerably more compact design is possible using bit serial arithmetic. The simplified interconnect and logic in a bit serial design allows it to work at a much higher clock rate than the equivalent bit parallel design. Of course, the design also needs to clocked w times for each iteration (w is the width of the data word). The bit serial design consists of three bit serial adder-subtractors, three shift registers and a serial Read Only Memory (ROM). Each shift register has a length equal to the word width. There is also some gating or multiplexers to select taps off the shift registers for the right shifted cross terms (shifting is accomplished using bit delays in bit serial systems). The bit serial CORDIC architecture is shown in Figure 2. In this design, w clocks are required for each of the n iterations, where w is precision of the adders. In operation, the load multiplexers on the left are opened for w clock periods to initialize the x, y and z registers (these registers could also be parallel loaded to initialize). Once loaded, the data is shifted right through the serial adder-subtractors and returned to the left end of the register. Each iteration requires w clocks to return the result to the register. At the beginning of each iteration, the control state machine reads the sign of the y (or z) register and sets the add/subtract controls accordingly. The appropriate tap off the register for the cross terms is also selected at the beginning of each iteration. During the nth iteration, the results can be read from the outputs of the serial adders while the next initialization data is shifted into the registers.

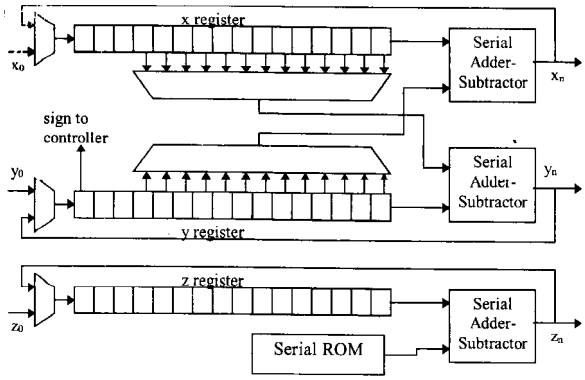


Figure 2 Bit serial iterative CORDIC

The simplicity of the bit serial designs in apparent from figure 2. Even in this case, the wiring of the shift tap multiplexers can present problems in some FPGAs (this is one place where tri-state long lines can come in handy). Even so, the interconnect is minimal and the logic between registers is simple. This combination permits bit clock rates near the maximum toggle frequency of the FPGA. The possibility of using extreme bit clock frequencies makes up for the large number of clock cycles required to complete each rotation.

Now, if the design is in a Xilinx 4000E series part, the shift registers can be implemented in the CLB RAM. The RAM emulates a shift register by incrementing the read/write address after each access. The dual port capability of the CLB RAM provides the capability to read two locations in the 16x1 RAM simultaneously. By properly sequencing the second address, the effect of the shift tap multiplexer is achieved without a physical multiplexer. The result is the shift register and multiplexer for word lengths up to 16 bits are implemented in a single CLB (plus 8 CLBs for the 2 address sequencers and iteration counter, which are shared by the three shifters). The serial ROM also uses the CLB for data storage. One CLB is required for every two iterations. The 16 bit, 8 iteration CORDIC processor shown in figure 3 uses only 21 CLBs, and will run at bit rates up to about 90 MHz (mainly limited by the RAM write cycle). This translates to about a 1.5 μS processing time, which is only about three and a half times longer than the best one could expect from the much larger bit parallel iterative solution.

ON-LINE CORDIC PROCESSORS:

The CORDIC processors discussed so far are iterative, which means the processor has to perform iterations at n times the data rate. The iteration process can unrolled so that each of n processing elements always performs the same iteration. An unrolled CORDIC processor is shown in Figure 4. Unrolling the processor results in two significant simplifications. First the shifters are each a fixed shift, which means that they can be implemented in the wiring. Second, the lookup values for the angle accumulator are distributed as constants to each adder in the angle accumulator chain. Those constants can be hardwired instead of requiring storage space. The entire CORDIC processor is reduced to an array of interconnected adder-subtractors. The need for registers is also eliminated, making the unrolled processor strictly combinatorial. The delay through the resulting circuit would be substantial, but the processing time is reduced from that required by the iterative circuit (if by nothing else than the set-up and hold times of the register). Most times, especially in an FPGA, it does not make sense to use such a large combinatorial circuit. The unrolled processor is easily pipelined by inserting registers between the adder-subtractors. In the case of most FPGA architectures there are already registers present in each logic cell, so the addition of the pipeline registers has no hardware cost.

The unrolled processor can also be converted to a bit serial design. Each adder subtractor is replaced by a serial adder-subtractor, separated by w bit shift registers. The shift registers are necessary to extract the sign of the y or z element before the first bits (1sbs) reach the next adder-subtractors.

The right shifted cross terms are taken from fixed taps in the shift registers. Some method of sign extension for the shifted terms is required too. Figure 5 shows two iterations of a bit serial CORDIC processors implemented in an Atmel 6005 or NSC Clay31 FPGA. Notice the cross term is taken from different taps off the shift register at each iteration.

This particular processor is used to compute vector magnitude. Since this is a vector mode process and the result angle is not required, there is no need for an angle accumulator. Figure 6 shows the detail of the adder-subtractor for that design.

The adder subtractor in this case includes logic to extend the sign of the shifted cross term and to reset the adder subtractor between words. The entire 7-iteration design occupies approximately 20% of the FPGA and runs at bit rates up to 125 MHz.

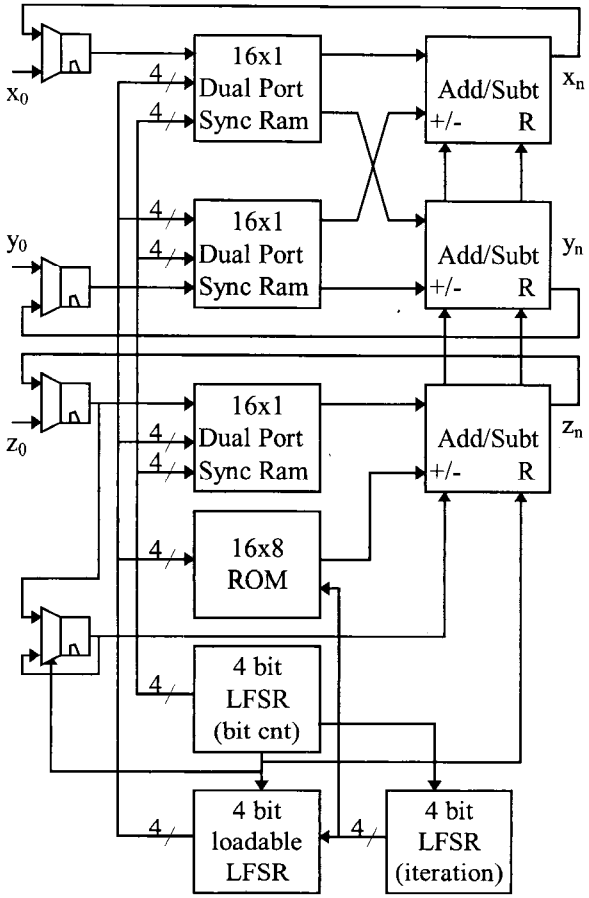


Figure 3 Iterative bit serial design for Xilinx 4000E series FPGA uses 21 CLBs

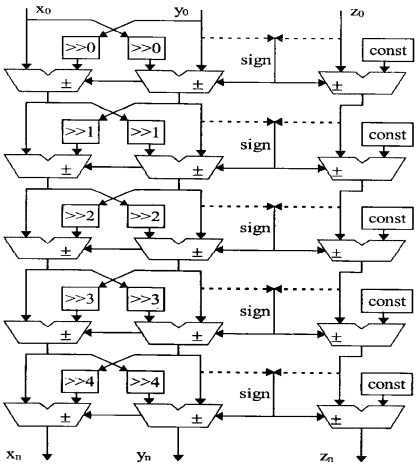


Figure 4 Unrolled CORDIC processor

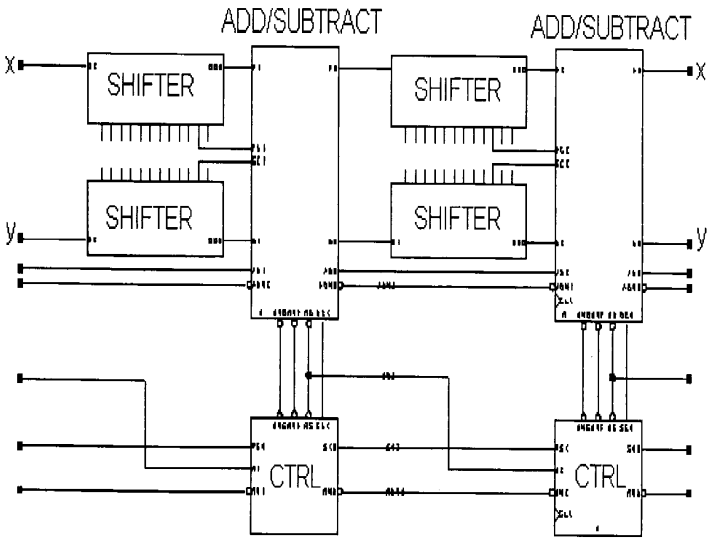


Figure 5 two iterations of bit serial CORDIC pipeline in Atmel/NSC FPGA

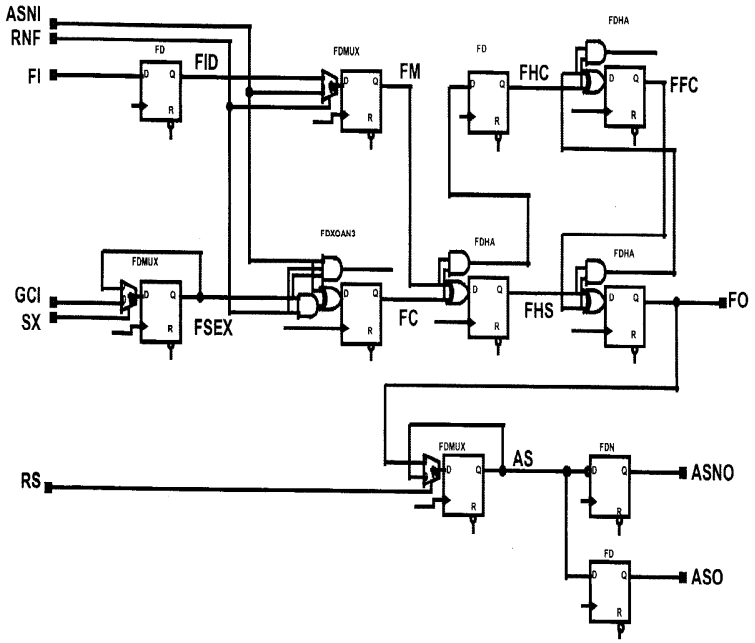


Figure 6 detail of pipelined bit serial adder-subtractor in Atmel/NSC FPGA

4.6.1 REDUNDANT CORDIC IMPLEMENTATION ON AN FPGA:

There are a number of ways to implement a CORDIC processor. The ideal architecture depends on the speed versus area tradeoffs in the intended application. We have used an iterative architecture that is a direct translation from the CORDIC equations. From there, we will look at a minimum hardware solution and a maximum performance solution by our proposed architecture.

The iteration process can unrolled so that each of n processing elements always performs the same iteration. An unrolled CORDIC processor is shown in Figure.4 .. The entire CORDIC processor is reduced to an array of interconnected adder-sub tractors.

The delay through the resulting circuit would be substantial, but the processing time is reduced from that required by the iterative circuit (if by nothing else than the set-up and hold times of the register). Most times, especially in an FPGA, it does not make sense to use such a large combinatorial circuit. Inserting registers between the adder-sub tractors easily pipelines the unrolled processor.

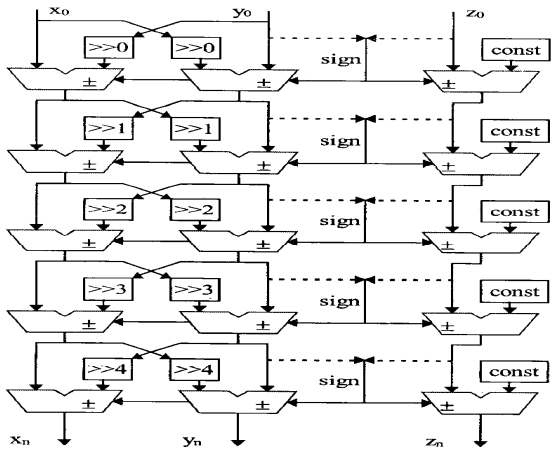
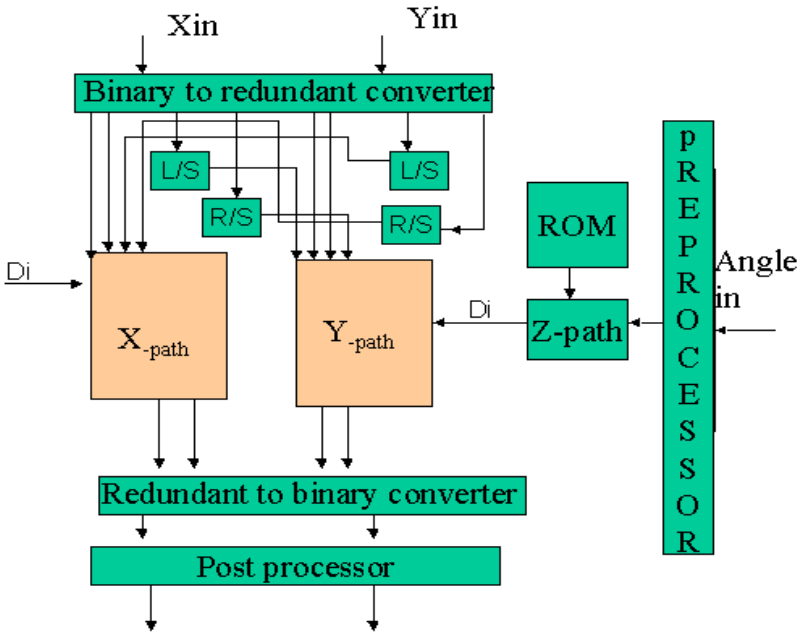


Figure 4 Unrolled CORDIC processor

Proposed Architecture of redundant CORDIC processor:

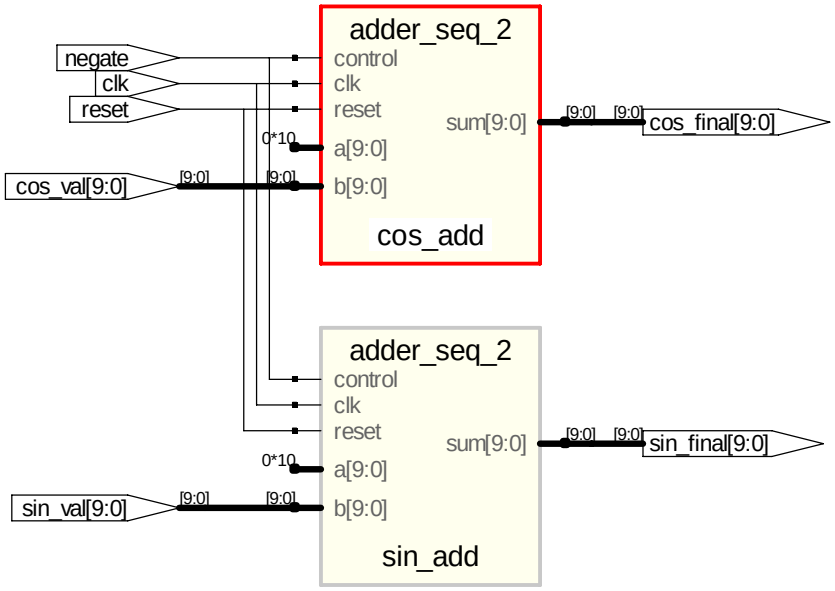


Fig(4.1) CORDIC architecture

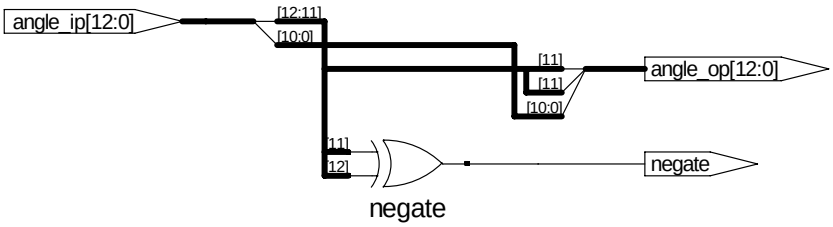
Fig(4.1) shows the architecture of the CORDIC processor having 'Xin' ,'Yin' as input binary 10 bit number. For computation of sine and cosine, value of Yin should be equal to zero and value of Xin should be equal to the scale factor K introduced during CORDIC iterations. Xin and Yin 10 bits binary number is first converted to redundant number using binary to redundant converter .The Xpath and Ypath are the standard CORDIC iterations in terms of X and Y. Here both arithmetic as well as logical shifting is done using the shifters R/S and L/S respectively.. The Xpath and Y path are implemented in similar fashion as shown in the figure 4 using Carry propagation free adders. We have designed a 10 bit CORDIC processor where Xin and Yin each are 10 bits inputs. The input angle i.e. "angle in" is taken to be of 13 bits binary number. First angle is processed using preprocessor. The Zpath performs the iteration for the third variable ie:"angle in"

Finally redundant to binary conversion is done at the last stage. Redundant to binary conversion is complex process done in three steps: firstly the S and D are extracted from given binary number, secondly the carry is decided and finally redundant to binary conversion is done as per the algorithm discussed in the earlier section. Postprocessor finally decides the sign of the final Sine and Cosine values as per the quadrant information stored in 'negation'. Postprocessor is implemented using the two 10 bits binary adder. The negation is implemented using the flip-flops that stored the quadrant information decided from the first two MSBs of input angle till the CORDIC processor gives the Sine and Cosine values.

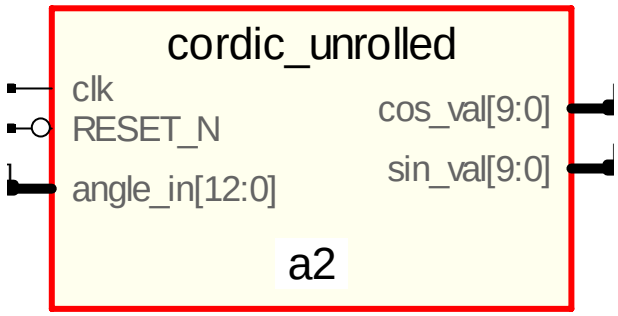
CORDIC processor takes 11 clock cycles to give the final output, so quadrant decision is done after 11 clock cycles till this value is stored in the register.



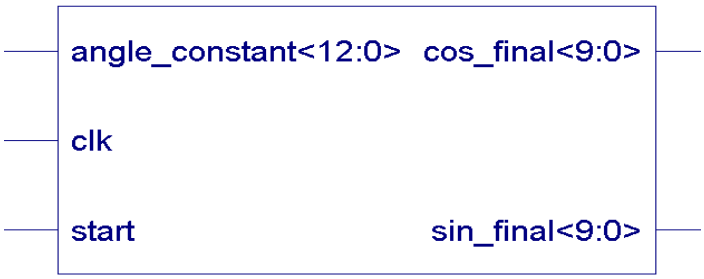
Fig(4.2) RTL view of Postprocessor



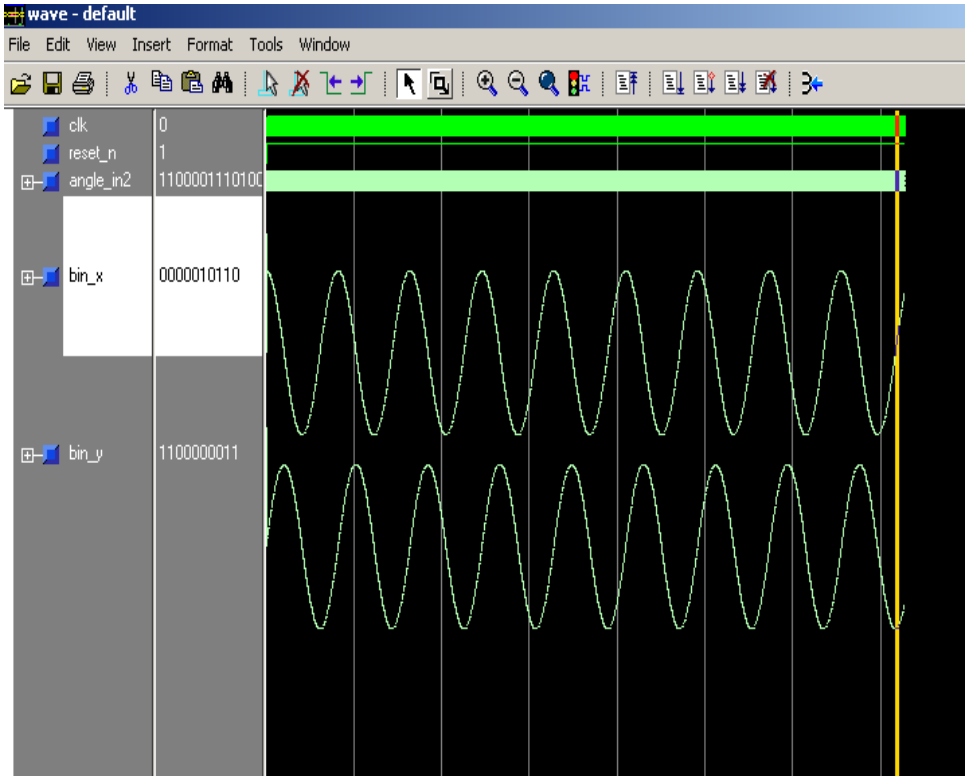
Fig(4.3):RTL view of preprocessor



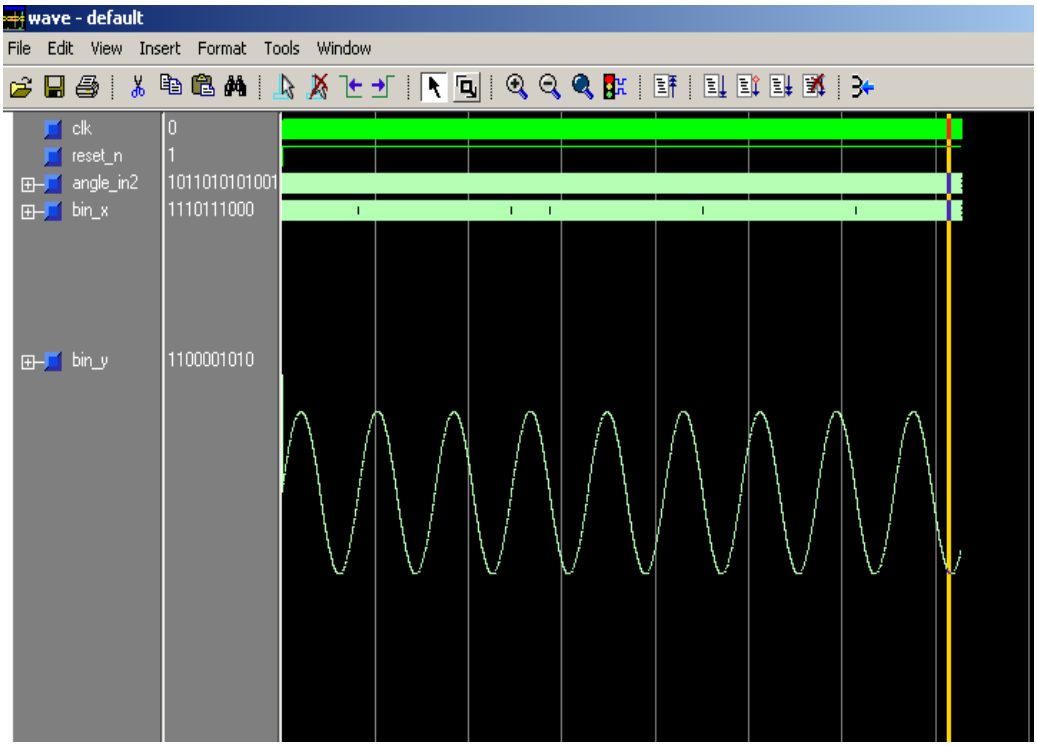
Fig(4.4):CORDIC unrolled



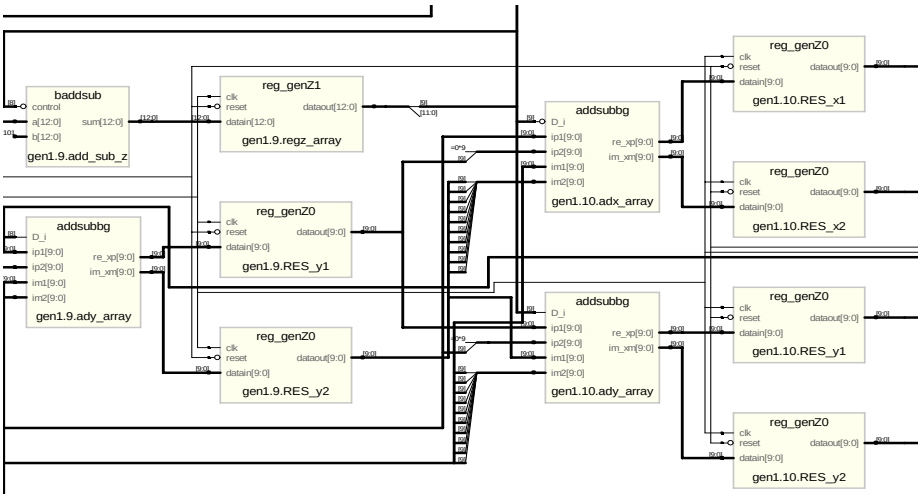
Fig(3.1) RTL Top view of CORDIC



Fig(3.2) Modelsim result shows pure Sine and Cosine wave generated by the CORDIC processor.



Fig(3.3) Modelsim result shows pure Sine wave generated by the CORDIC processor.



Fig(4.5):CORDIC one Stage

Cordic360

Device utilization summary:

Selected Device	:	v300pq240-5
Number of Slices	:	641 out of 3072 20%
Number of Slice Flip Flops	:	540 out of 6144 8%
Number of 4 input LUTs	:	1182 out of 6144 19%
Number of bonded IOBs	:	34 out of 170 20%

Number of GCLKs : 1 out of 4 25%

Speed Grade : -5

Minimum period: 11.376ns (Maximum Frequency: 87.904MHz)

Minimum input arrival time before clock: 7.264ns

Maximum output required time after clock: 7.511ns

Maximum combinational path delay: No path found

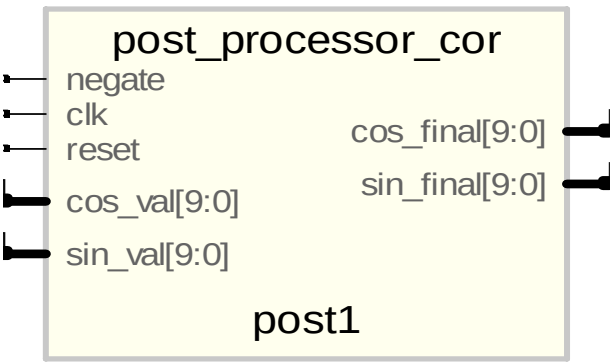
Total equivalent gate count for design: 12,221

Table (8.1) : Angle ROM of CORDIC

k	**θ_k**	**θ_k(13-bit binary)**
1	45°	0010000000000
2	26.5650°	0001001011110
3	14.0362°	0000101000000
4	7.12502°	0000010100010
5	3.57633°	0000001010001
6	1.78991°	0000000101001
7	0.895174°	0000000010100
8	0.447614°	0000000001010
9	0.223807°	0000000000101
10	0.111905°	0000000000010

Table (8.2): Performance summary

Technology	**Vertex 300**
Maximum clock frequency	87.904MHz
Output Resolution	10-bits for sine and cosine
Latency	12 clock cycle
Gate Count	12,221



Fig(4.6)RTL view of Post Processor

4.8 PREDICTING CORDIC IMPLEMENTATION

ARCHITECTURE.

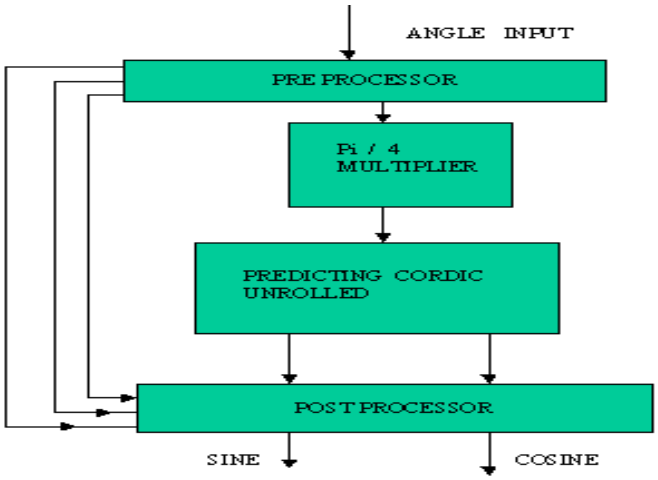
The input angle is in special binary format with weights -π,π/2, π/4---- π/2^m-1.However the sequence of rotations described in the previous section requires binary representation of an angle measured in radian. Therefore , the normalized angle must be converted to an appropriate radian value θ in the interval (-π/4, π/4).,which is the input to the Sine/Cosine generator .An output stage is required to generate the correct Sinπϕ and Cosπϕ from the computed Sinθ and Cosθ values. Thus overall architecture consist of following blocks:

1)Preprocessor

2) π/4 multiplier.

3)Sine/Cosine generator that computes Sinθ and Cosθ values.

4)Post processor that converts Sinθ and Cosθ values to correct Sinπϕ and Cosπϕ values.



Fig(9.1)Architecture of predicting CORDIC processor

Preprocessor:

The most two significant bits of the normalized angle ϕ MSB1 and MSB2 determine the quadrant occupied by πϕ. The third most significant bit MSB3 determines whether this angle is in the upper or lower half of the quadrant. These three most significant bits are stored and used later to control an interchange/negation operation in the output stage.

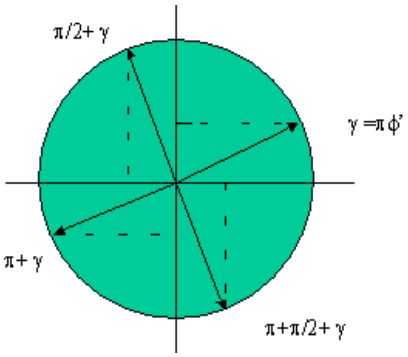
The value ϕ is first modified by setting its two most significant bits to zero. This maps any angle in the normalized second, third, fourth quadrant to a corresponding angle ϕ' in the first quadrant as shown in the figure( ).This angle lies in the upper half of the quadrant whenever

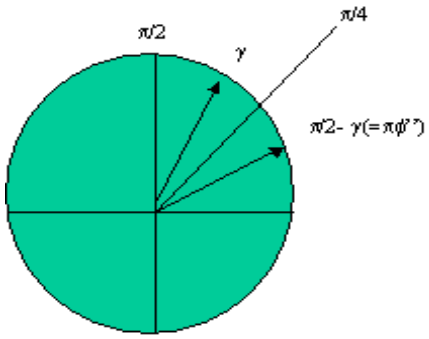
MSB3=1.The sine and cosine of angle γ above π/4 can be obtained from those of an angle equally below π/4 as shown in the figure( ),since Cos(γ)=Sin(π/2 -γ) and Sin(γ)=Cos(π/2 - γ).thus ,whenever MSB3=1 , the normalized angle below π/4 is obtained by simply replacing ϕ' by ϕ''=0.5 - ϕ'.

If MSB3=0 then ϕ''=ϕ'.The normalized angle is then converted to its value θ=πϕ'' by a multiplication by π.

Table(9.1) Control signals for the output stage

MSBs	ϕ	Cosπϕ	Sinπϕ
0 0 0	0< πϕ < π/4	Cosθ	Sinθ
0 0 1	π/4< πϕ < π/2	Sinθ	Cosθ
0 1 0	π/2< πϕ < 3π/4	-Sinθ	Cosθ
0 1 1	3π/4< πϕ < π	-Cosθ	Sinθ
1 0 0	-π < πϕ < -3π/4	-Cosθ	-Sinθ
1 0 1	-3π/4< πϕ < -π/2	-Sinθ	-Cosθ
1 1 0	-π/2< πϕ < -π/4	Sinθ	-Cosθ
1 1 1	-π/4< πϕ < 0	Cosθ	-Sinθ





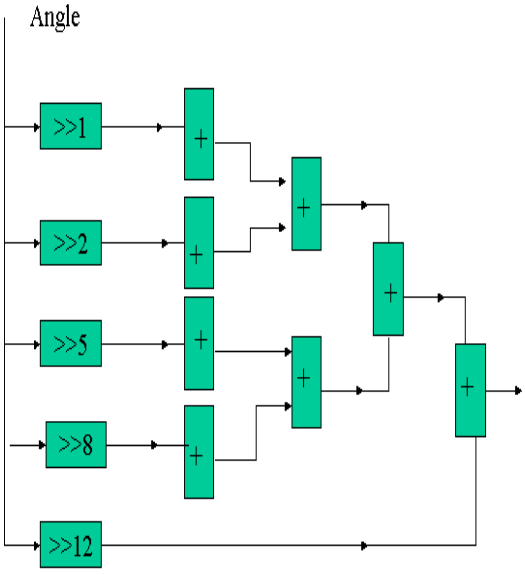
Fig(9.2) Quadrant6 symmetry maps an angle in second ,third ,fourth quadrants to an angle γ in the first quadrant.(b) π/4 mirror maps an angle γ above π/4 to an angle π/2- γ equally below π/4.

Postprocessor:

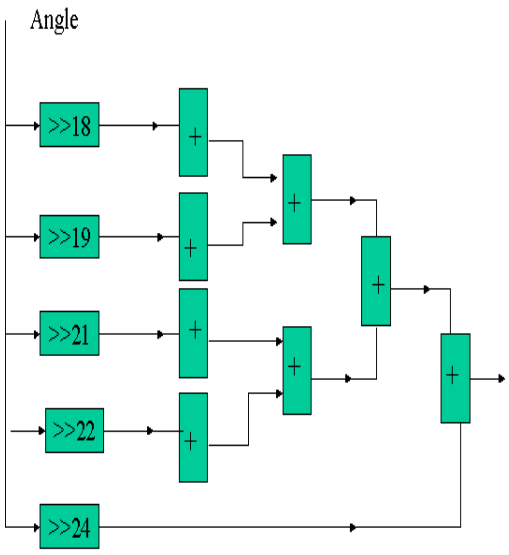
This stage maps the computed Sinθ and Cosθ values to desired Sinπϕ and Cosπϕ where πϕ lies in the correct π/4 interval within [-π π].As mentioned earlier ,this can be accomplished by simple negation and /or interchange operation. The control signals are shown in the table( ).The control signals are derived from the three most significant bits of the normalized angle.

π/4 Multiplier:

The multiplication by π/4 is achieved by simple shift and add operation. The fig( ) shows the stages of π/4 Multiplier .The input is first shifted then addition is performed.

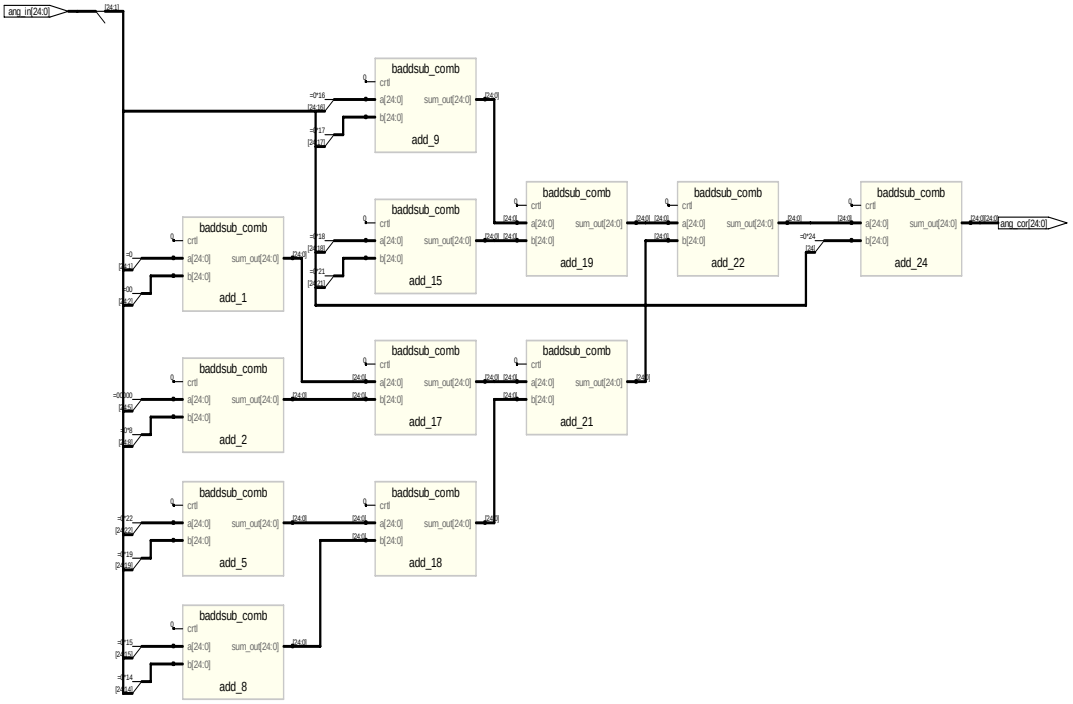


1

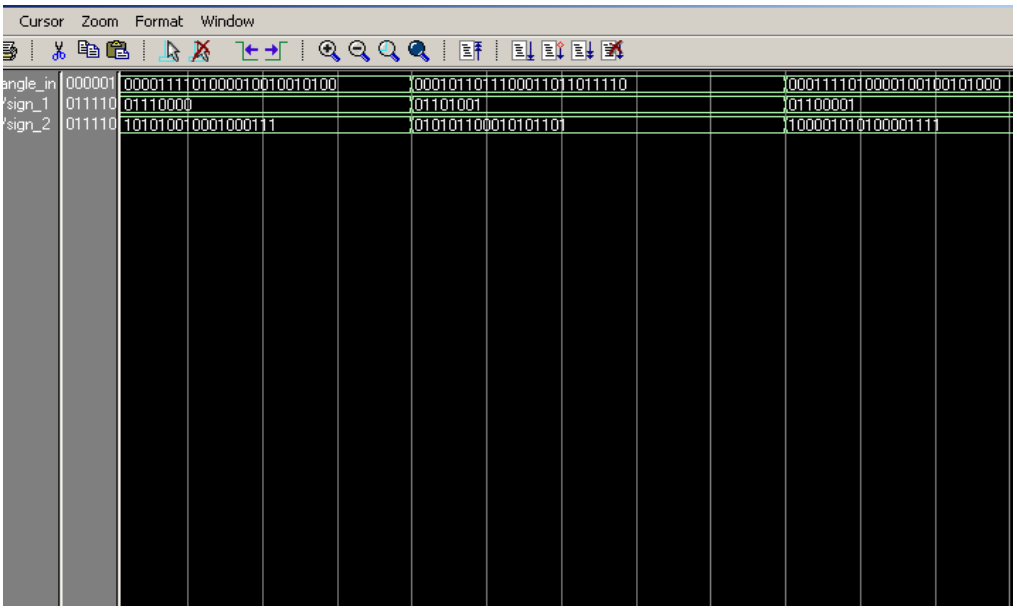


1

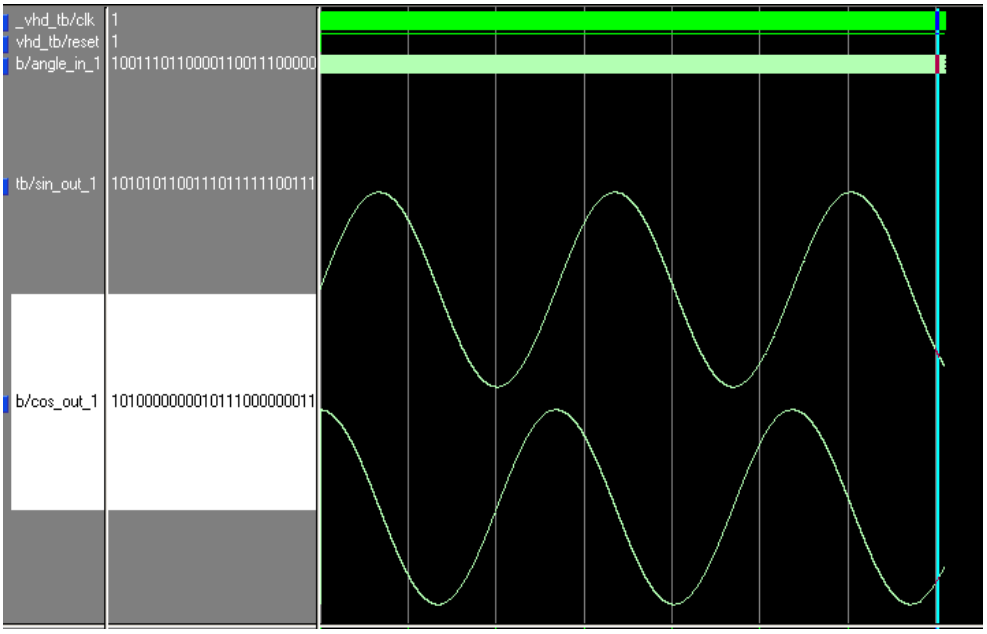
Fig(9.3)Shows the π/4 multiplier



Fig(9.6) RTL view of Pi/4Multiplier



Fig(9.4)Modelsim results of Signpredictor



Fig(9.5) Modelsim results of predicting CORDIC

CHAPTER 5

CONCLUSION

In order to accelerate the CORDIC iterations , one can use redundant number systems, which enable additions without carry propagation .For redundant implementation SD number with digit set{-1,0,1} is used. There are various other algorithm available for the carry free addition, out of which the efficient algorithm as per the digit set {-1,0,1} is used. There are few options for the conversion of redundant to binary converter out of which the look ahead mode converter, which provides fast conversion, is used.

As a result the whole conversion process i.e. form binary to redundant followed by carry free addition and then converted back to binary is operating at high frequency.

These designed blocks are used for implementation of redundant CORDIC processor, where the iterations of X and Y are performed by redundant arithmetic.At the final stage redundant sine and cosine values are converted back to binary by redundant to binary converter. The angle ROM is designed using the new angle format with weights $-\pi$, $\pi/2$, $\pi/4$,

$\pi/2^{m-1}$ .This new format helps to increase the CORDIC valid range from [$-\pi/2$ $\pi/2$] to [$-\pi$ π],thus covering all the four quadrant .The postprocessor is designed that finally decides the sign of computed sine and cosine values

With slight modification one can eliminate the Z iterations form the normal CORDIC iteration by predicting the sign for X an Y iterations directly form the given input angle.

The algorithm known as predicting CORDIC algorithm is based on parallelization of the original CORDIC algorithm by predicting all the rotation directions directly from the binary bits of the initial input angle. Unlike previous approaches that require complicated circuits or exponentially increased ROM, this algorithm provides a relatively simple prediction scheme through an efficient angle recording. This algorithm reduces the area and with use of redundant arithmetic will be the best implementation of redundant CORDIC.

Fast Fourier transform is one of the most widely use DSP algorithm. The need of complex multiplier is replace by introducing the Sine/Cosine generator. One can use radix-4 algorithm instead of radix-2 algorithm to decrease the number of multiplication. This CORDIC based FFT architecture is well suits for the FPGAs where no memory for storing the twiddle factors is available.

FUTURE WORK

Redundant number system with digit set {-1,0,1} has its own advantages, as addition performed is carry-free thus free from carry ripple phenomenon as in normal binary addition. One can use the carry save arithmetic instead of generalized sign digit number with digit set {-1,0,1}.

The predicting CORDIC processor performance can be further improved by incorporating carry-save adders in place of carry-free adders. The pi/4 multiplier designed using normal binary adder will be efficiently designed using the carry-save adder. This will add additional speed advantage.

REFERENCES

[1] Generalised Sign Digit number system, a unifying frame work for redundant number representations by Behrooz Parhami , IEEE Transaction on computers volume:39, No.1,Jan 1990

[2] Carry Free addition of recoded binary sign digit numbers
Behrooz Parhami, IEEE Transaction on computers volume:37, No.11,Nov 1988.

[3] "High speed VLSI multiplication algorithm with a redundant binary addition tree".Naofumi takagi, Hiroto yasuura, IEEE Transaction on computers vol:C34, No.9,Sep 1985.

[4] An efficient redundant Binary to Binary number converter
Sung Ming Yen, Chi-Sung laih, IEEE journal of solid-state circuit vol: 27,No.1, Jan 1992.

[5] The CORDIC Algorithm: New Results for Fast VLSI Implementation
Jean Duprat,Jean Michel Muller,IEEE Transaction on computers Volume:42, No.2,Feb 1993

[6] " Design of a unified arithmetic processor based on redundant constant factor CORDIC with merged scaling operation."
S.F.Hsiau,C.Y.Lau ,IEE, proc, comput ,digit Tech vol :147,No.4,July 2000

[7] "FPGA realization of a CORDIC based FFT processor for biomedical signal processing." IITKh paper.

[8] A CORDIC Processor for FFT Computation and its Implementation using Gallium Arsenide Technology.
Roberto Sarmiento, , Member IEEE, Felix Tobajas, Valentýn deArmas, Roberto Esper-Chaýn, Jose F.Lopez, , Member IEEE, Juan A. Montiel-Nelson, , Member, IEEE, and Antonio Nunez IEEE Transaction on very large integration (VLSI) System, Volume. 6, NO. 1, March 1998

[9] Double Step Branching CORDIC: A New Algorithm for Fast Sine & Cosine Generation.
Dhananjay S. Phatak, IEEE Transaction on Computers, Volume:. 47, NO. 5, MAY 1998

[10] Design of a unified arithmetic processor based on redundant constant factor CORDIC with merged scaling operation.

S.F.Hsiau,C.Y.Lau ,IEE, proc, computer ,digit Tech volume :147,No.4,July 2000.

[11] Comparison of CORDIC Algorithm Implementations on FPGA families.
Srikala Vadlamani , Dr. Wagdy Mahmoud, Dep. Electrical & computer engg.
Tennesse Technological University Cookeville,TN

[12] A Memory-Efficient and High –Speed Sine/Cosine Generator Based on Parallel CORDIC Rotations by Shen-Fu,Yu-Hen Hu, Tso-Bing Juang ,IEEE signal processing letters Vol.11 Feb 2004 .

[13] CORDIC Algorithms and Architectures (Chapter-24)

[14] FPGA realization of a CORDIC based FFT processor for biomedical signal processing. IITKh paper.

[15] FPGA Implementation of Sine and Cosine Generators Using the CORDIC algorithm by Tanya Vladimirova and Hans Tiggeler
Surrey Space Centre University of Surrey, Guildford, Surrey, GU2 5XH