DESIGNING OF CORDIC PROCESSOR IN VERILOG USING XILINX ISE SIMULATOR

Swati Sharma[1], Mohit Bansal[2]

[1]*Department of Electrical & Electronics Engineering, J.P. Institute of Engineering & Technology, Meerut, UP, India*
[2]*Department of Electrical & Electronics Engineering, Ideal institute of Technology, Ghaziabad, UP, India*

Abstract

In this paper, designing of CORDIC Processor in Verilog to determine the sine and cosine of a given argument, and extending this code to determine the Cartesian co-ordinates of a complex number represented in Euler's form. The inputs given are the Cartesian vector and the input angle in 17- bit signed number representation. The outputs obtained are sine and cosine of the input angle. To determine the Cartesian co-ordinates of the complex number, the magnitude of the complex number is given as input along with the input angle. The language used for the designing of CORDIC Processor is Verilog. The software used for the simulation is Xilinx ISE Simulator.

Keywords: *CORDIC Processor, Verilog, Cartesian co-ordinates, and Cartesian co-ordinates, Simulator etc.*

----------***----------

1. INTRODUCTION

CORDIC stands for COordinate Rotation DIgital Computer. The key concept of CORDIC arithmetic is based on the simple and ancient principles of two-dimensional geometry. But the iterative formulation of a computational algorithm for its implementation was first described in 1959 by Jack E. Volder [1], [2] for the computation of trigonometric functions, multiplication and division. Not only a wide variety of applications of CORDIC have been suggested over the time, but also a lot of progress has taken place in the area of algorithm design and development of architectures for high performance and low-cost hardware solutions [3]–[12]. Not only a wide variety of applications of CORDIC have emerged in the last 50 years, but also a lot of progress has been made in the area of algorithm design and development of architectures for high-performance and low-cost hardware solutions of those applications. CORDIC-based computing received increased attention in 1971, when John Walther [3], [4] showed that, by varying a few simple parameters, it could be used as a single algorithm for unified implementation of a wide range of elementary transcendental functions involving logarithms, exponentials, and square roots along with those suggested by Volder [1]. During the same time, Cochran [5] benchmarked various algorithms, and showed that CORDIC technique is a better choice for scientific calculator applications. The popularity of CORDIC was very much enhanced thereafter primarily due to its potential for efficient and low-cost implementation of a large class of applications which include: the generation of trigonometric, logarithmic and transcendental elementary functions; complex number multiplication, eigenvalue computation, matrix inversion, solution of linear systems and singular value decomposition (SVD) for signal processing, image processing, and general scientific computation. Some other popular and upcoming applications are

i. Direct frequency synthesis, digital modulation and coding for speech/music synthesis and communication;
ii. Direct and inverse kinematics computation for robot manipulation; and
iii. Planar and 3-dimensional vector rotation for graphics and animation.

Although CORDIC may not be the fastest technique to perform these operations, it is attractive due to the simplicity of its hardware implementation, since the same iterative algorithm could be used for all these applications using the basic shift-add operations of the form $a \pm b.2^{-i}$. Key points regarding CORDIC may elaborated as

i. Introduced in 1959 by Jack Volder
ii. Performs vector rotations of arbitrary angles using only shifts and add.
iii. An iterative algorithm
iv. Computation involves addition, subtraction, compares, and shifts
v. Calculates a wide variety of functions like sine, cosine, arc tangent, square root
vi. Idea is to rotate a vector in Cartesian Plane by some angle
vii. Mostly used when no hardware multiplier is available
viii. Calculations are performed through a number of micro-rotations

CORDIC architectures have been successfully employed for waveform generation [7], [8], implementation of digital filters

[9], transform computation [10], [11], matrix calculations [12] etc. In spite of its simplicity and low computational complexity, CORDIC algorithm suffers from major bottlenecks like either high latency or large overheads of scale-factor compensation, when an optimized set of micro-rotations are used to reduce the latency. Parallel CORDIC architectures have been suggested in [13] and [14] to reduce the latency but at the cost of additional hardware and time to implement the scale-factor compensation. The redundant iterations are eliminated by greedy search in [15]–[17], but the hardware savings achieved by this approach are counter balanced by variable scale-factor compensation circuits. Various scale-factor compensating techniques have been suggested in the literature [18]–[20], but these techniques either lead to large area overheads or otherwise affect throughput or latency. The Taylor series expansion offers a low complexity solution for the design of scale-free CORDIC. Scaling-Free CORDIC [21]–[23] indemnify the scale-factor limitations to certain extent. Various optimization efforts in the above CORDIC algorithms are targeted for circular CORDIC, while hyperbolic CORDIC still needs to be explored for improvements. The number of efficient CORDIC designs for hyperbolic trajectory is far less as compared to circular trajectory regardless, inspite of its wide scope in artificial neural networks [24]–[26], adaptive filtering [27] and for computing logarithm and exponential function [28]. In [29], the authors improve the range of convergence of conventional CORDIC algorithm in hyperbolic trajectory by using additional iterations which allow negative iteration indices as well. Though it increases the RoC of the hyperbolic CORDIC algorithm, it significantly adds to the latency of the processor.

The main contributions of this paper are: (i) Applications and principle and mode operations of CORDIC (ii) implementation of LUTs, Xilinx ISE, Verilog Hardware, (iii) designing of CORDIC processor (iv) simulation and results, and (v) conclusion.

2. CORDIC's, APPLICATIONS, ARCHITECTURE MODE OF OPERATIONS

2.1 CORDIC's, Applications

2.1.1 Hardware

CORDIC is generally faster than other approaches when a hardware multiplier is unavailable (e.g., in a microcontroller based system), or when the number of gates required to implement the functions it supports should be minimized (e.g., in an FPGA).On the other hand, when a hardware multiplier is available (e.g., in a DSP microprocessor) as shown in Fig. 1.

2.1.2 Software

Many older systems with integer only CPUs have implemented CORDIC to varying extents as part of their IEEE Floating Point libraries. As most modern general purpose CPUs have floating point registers with common operations such as add, subtract, multiply, divide, sin, cos, square root, log10, natural log, the need to implement CORDIC in them with software is nearly non-existent. Only microcontroller or special safety and time constraint software applications would need to consider using CORDIC.

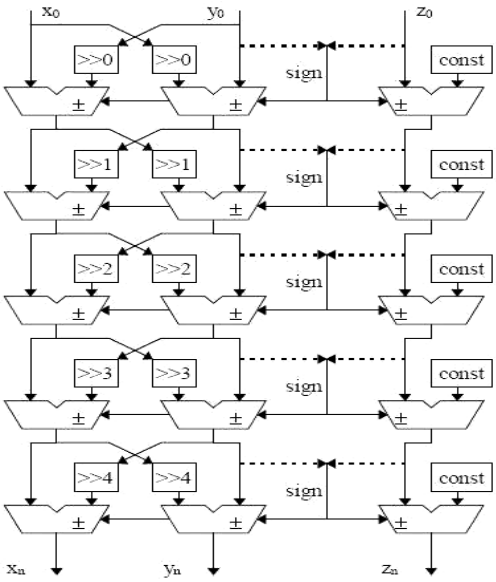


Fig.-1: CORDIC Architecture

2.2 CORDIC's, Applications

The CORDIC core can be realized in one of three methods:

i. **ITERATE:** This option builds a single ROTATOR. The user provides the arguments and gives the core ITERATIONS clock cycles to get the result. A signal named init is instantiated to load the input values. It uses the least amount of LUTs.

ii. **PIPELINE:** This option can take a new input on every clock and gives results ITERATIONS clock cycles later. It uses the most amount of LUTs.

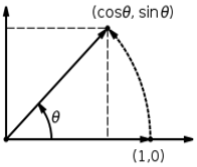
iii. **COMBINATORIAL:** This option gives result in a single clock cycle at the expense of very deep logic levels.

The combinatorial implementation runs at about 10 MHz while the iterative ones run at about 125 in Lattice ECP2 device.

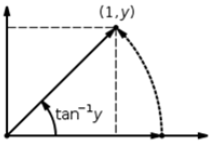
2.3 Mode of Operations

A Survey of CORDIC Algorithms for FPGAs‖ [7] CORDIC can be used in one of the two modes of operation. (i) Rotation and vector mode as shown in Fig. 2 (a-b) respectively. The basic idea of CORDIC lies

Rotate (1,0) by φ degrees to get (x,y) : x=cos(φ), y=sin(φ)



Rotation Mode
[Start at (1, 0) Rotate by θ (cosθ, sinθ)]



Vector mode
[Start at (1, y) Rotate until y = 0 The rotation is tan−1y]

(a) (b)

Fig.-2: Rotation of a vector by an angle θ

$$x' = x\cos\phi - y\sin\phi$$
$$y' = y\cos\phi + x\sin\phi$$

Rearrange as

$$x' = \cos\phi \cdot [x - y\tan\phi]$$
$$y' = \cos\phi \cdot [y + x\tan\phi]$$

Can compute rotation φ in steps where each step is of size

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

The ith iteration results in

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

where:

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

Zi is introduced to keep track of the angle that has been rotated (z0 = φ)

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

2.3 Architecture of LUTs, Xilinx ISE, Verilog Hardware

2.3.1 LUTs

An overview of how LUTs are built helps describe the key innovations in the ALM. A LUT is typically built out of SRAM bits to hold the configuration memory (CRAM) LUT mask and a set of multiplexers to select the bit of CRAM that is to drive the output. To implement a k-input LUT (k-LUT) a LUT that can implement any function of k inputs—2k SRAM bits and a 2k:1 multiplexer are needed. Fig. 3 shows a 4 LUT, which consists of 16 bits of SRAM and a 16:1 multiplexer implemented as a tree of 2:1 multiplexers. The 4 LUT can implement any function of 4 inputs (A, B, C, D) by setting the appropriate value in the LUT-mask. To simplify the 4-LUT, it can also be built from two 3-LUTs connected by a 2:1 multiplexer.

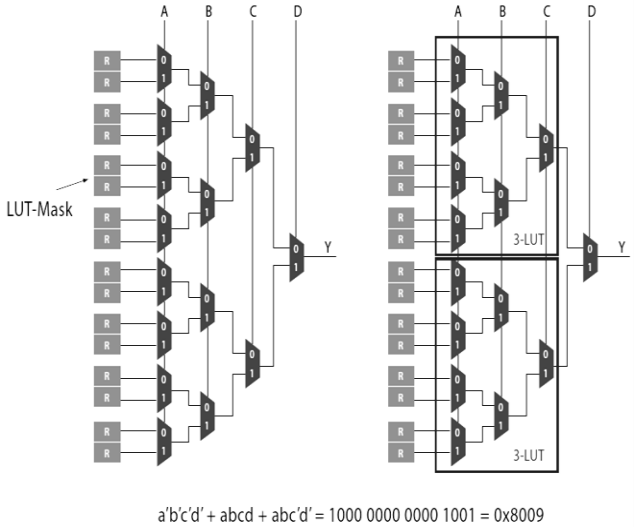


Fig.-3: Building a LUT

2.3.2 Verilog

Verilog HDL is most commonly used in the design, verification, and implementation of digital logic chips at the register transfer level (RTL) of abstraction. There are two assignment operators: a blocking assignment (=) and a non-blocking (<=) assignment. Verilog modules that conform to a synthesizable coding-style, known as RTL, can be physically realized by synthesis software. Synthesis-software algorithmically transforms the Verilog source into a netlist, a logically-equivalent description consisting only of elementary logic primitives (AND, OR, NOT, flip-flops, etc.) that are available in a specific FPGA or VLSI technology. The function of non- blocking (<=) assignment operator in Verilog is that its action doesn't register until the next clock cycle. This means that the order of the assignments is irrelevant and will produce the same result. The other assignment operator is referred to as a blocking (=) assignment. When "=" assignment is used, for the purposes of logic, the target variable is updated immediately. There are two separate ways of declaring a Verilog process. These are the always and the initial keywords. The always keyword indicates a free-running process. The initial keyword indicates a process executes exactly once. Both constructs begin execution at simulator time 0, and both execute until the end of the block. Once an always block has reached its end, it is rescheduled (again).

2.3.3 Xilinx ISE

Xilinx ISE is a software tool produced by Xilinx for synthesis and analysis of HDL designs, which enables the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

-Select File, then New Project.
-Select a project location and name.
-Select the device family, device, package, and speed grade.
-Click New Source.
-Select Verilog Module and enter the file name.
-Specify the inputs and outputs for the decoder. These will be inserted into an automatically generated template for the Verilog file.

In case, there are no existing sources, click Next. If there are any, click Add-Source.

Project Navigator now shows a summary of the project

Click on the ―filename.v‖ tab below the summary window in the top left―Sources‖ pane.

Before the design can be synthesized, we need to specify what pins on the FPGA the inputs and outputs are connected to. Compile the design (Check Syntax) to check for errors. Make a Test Bench Waveform using New Source. Associate the Test Bench Waveform with the Project filename. Perform Behavioral Simulation using Xilinx ISE Simulator. View the RTL Schematic and the Synthesis Report. Fig. 4 outlined the Flow of building a Xilinx.

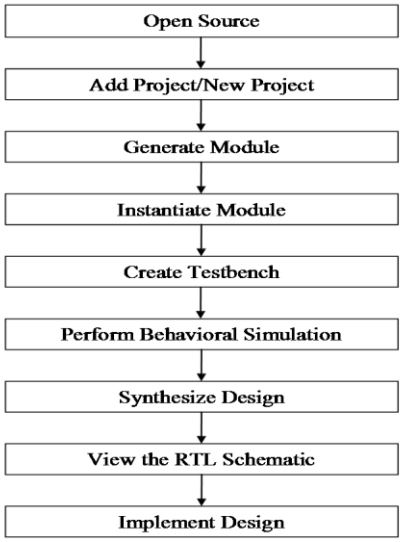


Fig.-4: Building a Xilinx Design Flow

3. DESIGNING OF CORDIC PROCESSOR

3.1 Design Specifications

The code is designed for first quadrant

i. Mode of operation chosen is rotation mode
ii. Input Cartesian vector(x, y) is (0.603,0)
iii. Rotation mode is realized through combinatorial mode.
iv. Number of iteration are 16 which is equal to 2^iteration bits.

The various design specifications are as follows: The core can operate in either radian or degree mode. The core uses 16+sign (17 bit) numbers for x,y, and theta, and iterates 16 times in the algorithm. There are two arctan function tables, one for radian and one for degree mode. The core will operate in ROTATION mode in which X and Y Cartesian vector and an angle are given. The CORDIC rotator reduces the angle to zero by rotating the vector. To compute the cos and sin of the angle, set the inputs as follows:

y_i = 0
x_i = 0.603 =17'd19896
theta_i = the input angle

On completion:
y_o = sin theta_i x_o = cos theta_i

The CORDIC can work with the angle expressed in radians or degrees as demonstrated in Table 1.

RADIAN_16 uses 16 bit values (+ sign bit for 17 bit accuracy). Angle information is in the format U (1, 15) where bit 16 is the sign bit, bit 15 is the whole number part and bits [14:0] are the fractional parts.

DEGREE_8_8 uses U(8,8) + a sign bit where bit 16 = the sign bit, [15:8] = the whole number part and [7:0] = the fractional.

The X and Y values are computed using a `XY_BITS + sign bit accuracy. The format is assumed to be U(1,15) + sign bit

Going to a higher number of bits would allow more iteration, thus, improving accuracy. Iteration Accuracy is the number of times the algorithm will iterate. Number of iterations <= the number of bits used in the angles The code is further used to

determine the Cartesian co-ordinates of a complex number represented in Euler's form.

Euler's form : $re^{i\theta}$

where r = magnitude of the complex number & θ = angle

The Cartesian coordinates, thus, obtained are given as

$$x\_cor = r*\cos\theta$$

$$y\_cor = r*\sin\theta$$

Table -1: Representation of angles in 17-bit signed decimal numbers

Angle (degrees)	0	1	2	3	4	5	6	7
Angle(radians,17'd)	0	571	1143	1715	2287	2859	3431	4003
Angle(degrees)	8	9	10	11	12	13	14	15
Angle(radians, 17'd)	4575	5147	5719	6291	6862	7434	8006	8578
Angle(degrees)	16	17	18	19	20	21	22	23
Angle(radians, 17'd)	9150	9722	10294	10866	11438	12010	12582	13153
Angle(degrees)	24	25	26	27	28	29	30	31
Angle(radians, 17'd)	13725	14297	14869	15441	16013	16585	17157	17729
Angle(degrees)	32	33	34	35	36	37	38	39
Angle(radians, 17'd)	18301	18873	19444	20016	20588	21160	21732	22304
Angle(degrees)	40	41	42	43	44	45	46	47
Angle(radians, 17'd)	22876	23448	24020	24592	25164	25735	26307	26879
Angle(degrees)	48	49	50	51	52	53	54	55
Angle(radians, 17'd)	27451	28023	28595	29167	29739	30311	30883	31455
Angle(degrees)	56	57	58	59	60	61	62	63
Angle(radians, 17'd)	32026	32598	33170	33742	34314	34886	35458	36030
Angle(degrees)	64	65	66	67	68	69	70	71
Angle(radians, 17'd)	36602	37174	37746	38317	38889	39461	40033	40605
Angle(degrees)	72	73	74	75	76	77	78	79
Angle(radians, 17'd)	41177	41749	42321	42893	43465	44037	44608	45180
Angle(degrees)	80	81	82	83	84	85	86	87
Angle(radians, 17'd)	45752	46324	46896	47468	48040	48612	49184	49756
Angle(degrees)	88	89	90					
Angle(radians, 17'd)	50328	50899	51471					

4. SIMULATION AND SYNTHESIS

Corresponding to the specifications and methodology discussed above following are the simulation and synthesis results obtained associated with the various input/output

4.1 Testbench Waveform

The testbench waveform for the required simulation is shown in Fig. 5.

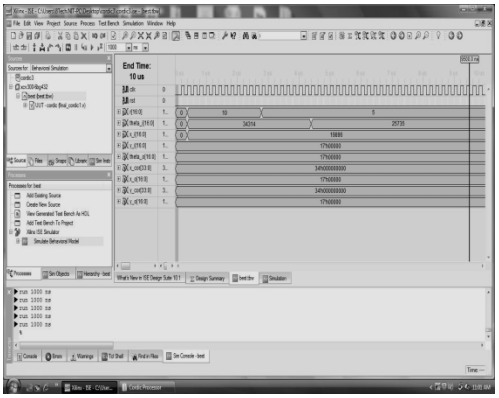


Fig.-5: Testbench waveform

Fig. 5-11 represents the simulation results. The inputs are:

theta_i: 17'd34314 → 600

17'd25735 → 450

x_i: 17'd19896 →0.603

y_i: 0

r: 10 , 5

4.2 Testbench Waveform

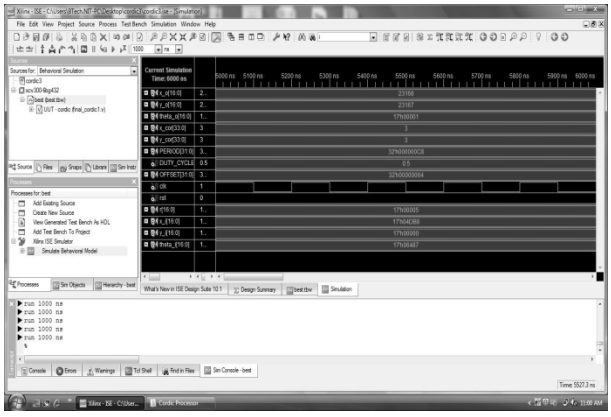


Fig.-6: Simulation (Output corresponding to θ = 45°)

Simulation results:
x_i = 0.603, y_i = 0
y_i = 0 theta_i = 45°, r = 5
x_o = 17'd23168 = 0.7070
y_o = 17'd23167 = 0.7070
theta_o = 1
x_cor = 3, y_cor = 3

Theoretical results:
x_i = 0.603,
theta_i = 45°, r = 5
x_o = 17'd23170 = 0.7071
y_o = 17'd23170 = 0.7071
theta_o = 0
x_cor = 3.54, y_cor = 3.54

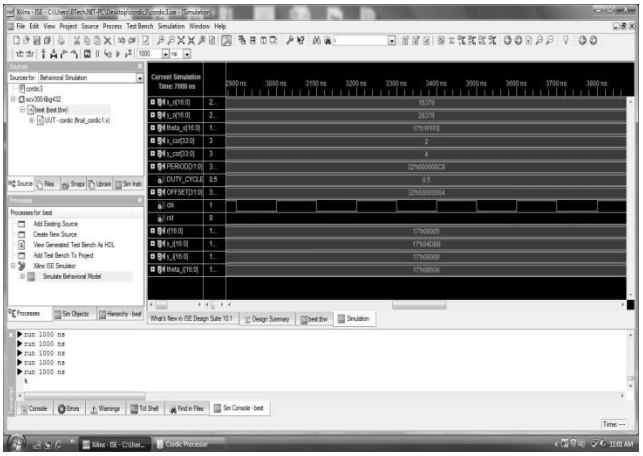


Fig.-7: Simulation (Output corresponding to θ = 60°)

Simulation results:
x_i = 0.603, y_i = 0
theta_i = 60°, r = 5
x_o = 17'd15379 = 0.4963
y_o = 17'd28378 = 0.8660
theta_o = -1
x_cor = 2, y_cor = 4

Theoretical results:
x_i = 0.603, y_i = 0
theta_i = 60°, r = 5
x_o = 17'd16384 = 0.5000
y_o = 17'd28377 = 0.8659
theta_o = 0
x_cor = 2.50, y_cor = 4.33

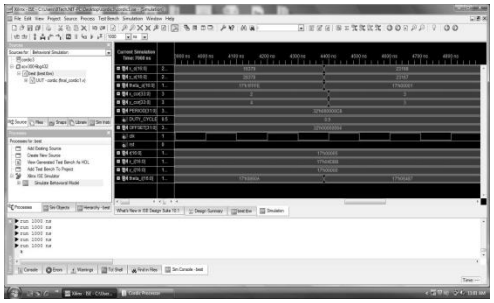


Fig.-8: Simulation (Output showing transition between θ = 45° and θ = 60°)

4.3 RTL Synthesis

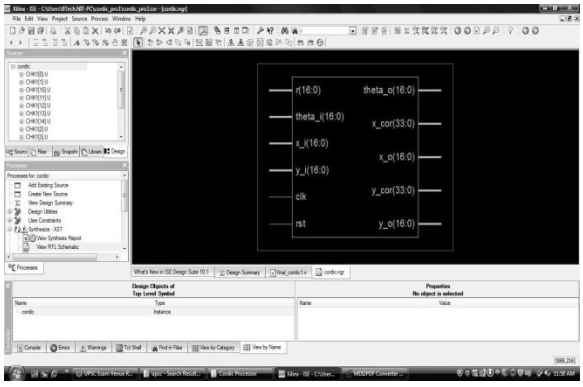


Fig.-9: Diagram showing inputs and ouputs

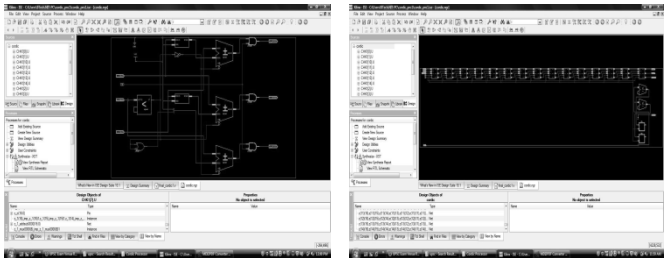


Fig.-10: Architecture of CORDIC & module CORDIC

4.3.1 RTL Synthesis Report

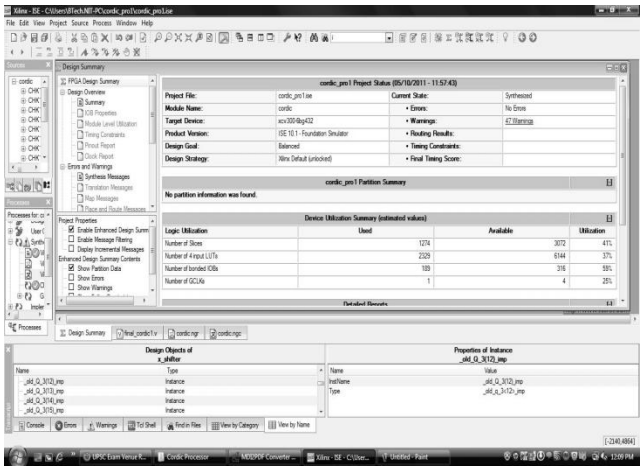


Fig.-11: Synthesis Report

5. CONCLUSIONS

This paper presents the designing of CORDIC processor in VERILOG using XILINX ISE simulator. The output of the code, designed for the CORDIC processor in Rotation mode, was obtained using different values of the magnitude(r) of the vector and the input angle (theta_i). The outputs were found to be in complete agreement with the theoretical results. The RTL Schematic and the Synthesis Report confirmed that there were no errors in the designing of the code. Hence, the code is found to be correct for any positive value of r and 00≤theta_i≤900. XILINX ISE simulator is useful tool for designing of various efficient CORDIC processors with different configurations and can further extend with latest advance technology.

REFERENCES

[1] CORDIC, http://en.wikipedia.org/wiki/CORDIC, accessed on March 2014.

[2] J. E. Volder, "The Birth of CORDIC", J. VLSI Signal Processing, Vol. 25, No. 101 , 2000, also available on http://dx.doi.org/10.1023/A:1008110704586, accessed on March 2014.

[3] Schmid Hermann, "Decimal computation", New York, Wiley, 1974

[4] Andraka Ray, "A survey of CORDIC algorithms for FPGA based computers" also available on http://www.andraka.com/files/crdcsrvy.pdf, accessed on March 2014.

[5] Verilog, http://en.wikipedia.org/wiki/Verilog, accessed on March 2014.

[6] Ayan Banerjee and Anindya Sundar Dhar "FPGA realization of a CORDIC based FFT processor for biomedical signal processing", Journal of Microprocessors and Microsystems, Vol. 25, No.3, pp. 131–142, 2001.

[7] D.G. Steer and S.R. Penstone, "Digital Hardware for Sine-Cosine Function", IEEE Transactions on Computers, Vol. C-26, No. 12, pp.1283-1286, 1977.

[8] G.L. Haviland and A.A. Tuszynski, "A CORDIC Arithmetic Processor Chip", IEEE Journal of Solid-State Circuits, Vol.15, No.1,pp.4-15,1980.

[9] T.W. Curtis, P. Allison and J.A. Howard, "A Cordic Processor for Laser Trimming", IEEE Transaction on Micro, Vol.6, No.3, pp. 61.71,1986.

[10] Y.H. Hu and S. Naganathan, "A novel implementation of a chirp Z-transform using a CORDIC processor", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.38, No.2, pp.352-354, 1990.

[11] M.B. Yeary, R.J. Fink, H. Sundaresan and D.W. Guidry, "IEEE Transactions on Instrumentation and Measurement", Vol. 51, No.4, pp.804-809, 2002.

[12] S. Aggarwal, P.K. Meher and K. Khare, "Scale-Free Hyperbolic CORDIC Processor and Its Application to Waveform Generation", IEEE Transactions on Circuits and Systems I: Regular Papers, Vol.60, No.2, pp. 314-326, 2013.

[13] A. Troya, K. Maharatna, M. Krstic, E. Grass, U. Jagdhold and R. Kraemer, "Low-Power VLSI Implementation of the Inner Receiver for OFDM-Based WLAN Systems", IEEE Transactions on Circuits and Systems I: Regular Papers, Vol.55, No.2, pp.672-686, 2008.

[14] S. Aggarwal, P.K. Meher and K. Khare, "Area-Time Efficient Scaling-Free CORDIC Using Generalized Micro-Rotation Selection", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 20, No.8, pp.1542 – 1546, 2012.

[15] T. Y. Sung and H.C Hsin, "Design and simulation of reusable IP CORDIC core for special-purpose processors", IET Computers & Digital Techniques, Vol.1, No.5, pp. 581 – 589, 2007.

[16] T. Y. Sung, "Memory-efficient and high-speed split-radix FFT/IFFT processor based on pipelined CORDIC rotations", IEE Proceedings Vision, Image and Signal Processing, Vol.153, No.4, pp. 405 – 410, 2006.

[17] J. Granado, A. Torralba, J. Chavez and V. Baena-Lecuyer, "Design of an efficient CORDIC-based architecture for synchronization in OFDM", IEEE Transactions on Consumer Electronics, Vol.52, No. 3, pp. 774 – 782, 2006.

[18] K. Maharatna, A. Troya, S. Banerjee and E. Grass, "Virtually scaling-free adaptive CORDIC rotator", IEE Proceedings Computers and Digital Techniques, Vol. 151, No. 6, pp. 448 – 456, 2004

[19] M.B. Yeary, R.J. Fink, H. Sundaresan and D.W. Guidry, "Design of a CORDIC processor for mixed-signal A/D conversion", IEEE Transactions on Instrumentation and Measurement, Vol.51, No.4, pp. 804 – 809, 2002.

[20] Durga Prasad, "SIMD based baseband processor for CORDIC algorithms", ProQuest, UMI Dissertations, The University of Texas at Dallas, 2010.

[21] Terence Keith, "Adaptive CORDIC: Using parallel angle recoding to accelerate CORDIC rotations", ProQuest, UMI Dissertation, The University of Texas at Austin, 2007.

[22] P. K. Meher, J. Valls, T-B Juang, K. Sridharan, and K. Maharatna, "50 Years of CORDIC: Algorithms, Architectures and Applications", IEEE Transactions on Circuits & Systems-I: Regular Papers, Vol.56, No.9, pp.1893- 1907, 2009.

[23] P. K. Meher and S.Y. Park, "CORDIC Designs for Fixed Angle of Rotation", IEEE Transactions on VLSI Systems, Vol.21, No.2, pp.217-228, 2013.

[24] B. Lakshmi and A. S. Dhar, "CORDIC Architectures: A Survey", Journal: VLSI Design, 2010.

[25] Jack E. Volder, "The CORDIC Trigonometric Computing Technique", IRE Transactions on Electronic Computers, pp.330-334, 1959.

BIOGRAPHIES

Swati Sharma received B.E. degree in Instrumentation Engineering from Sant Longowal Institute of Engineering and Technology Longowal Punjab in 2009. He is currently pursuing Masters of Technology in VLSI from Mewar University Mewar, India. Her areas of interest are VLSI and Digital Electronics.

M. Bansal is currently working as an Assitant Professor in Ideal Institute of Technology Ghaziabad, UP. He has completed his doctoral research work in the area of renewable energy systems from Alternate Hydro Energy Center, Indian Institute of Technology Roorkee in 2014. Earlier, He did his masters in Energy and Environmental Management from Center for Energy Studies, Indian Institute of Technology Delhi. He completed his graduation in Electrical Engineering from Aligarh Muslim University, Aligarh. His research interests include Renewable Energy Systems, Rural Electricafication, Emebeded System and VLSI.