Design and FPGA Implementation of CORDIC-based 8-point 1D DCT Processor

A Thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Electronics and Communication Engineering

by

Rohit Kumar Jain

Roll No. 107EC012

Under the supervision of

Dr. Kamala Kanta Mahapatra

Professor

Department of Electronics and Communication Engineering,

National Institute of Technology, Rourkela

Session 2010-2011

Design and FPGA Implementation of CORDIC-based 8-point 1D DCT Processor

A Thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Electronics and Communication Engineering

by

Rohit Kumar Jain

Roll No. 107EC012

Under the supervision of

Dr. Kamala Kanta Mahapatra

Professor

Department of Electronics and Communication Engineering,

National Institute of Technology, Rourkela

Session 2010-2011

National Institute of Technology, Rourkela

C E R T I F I C A T E

This is to certify that the Thesis entitled, '**Design and FPGA implementation of CORDIC-based 8-point 1D DCT processor'** submitted by **Rohit Kumar Jain** in partial fulfillment of the requirements for the award of **Bachelor of Technology Degree** in **Electronics and Communication Engineering** at the **National Institute of Technology, Rourkela** is an authentic work carried out by him under my supervision. To the best of my knowledge and belief the matter embodied in the Thesis has not been submitted by him to any other University/Institute for the award of any Degree/Diploma.

Date

Prof. Kamala Kanta Mahapatra

Dept. of Electronics and Communication Engg.,

National Institute of Technology, Rourkela

ACKNOWLEDGEMENT

This project in itself is an acknowledgement to the inspiration, drive and the technical assistance contributed to it by many people. It would have never seen the light of day without the help and guidance that it received from them.

Firstly, I would like to express my sincere thanks and deepest regards to my guide **Dr. K K Mahapatra, Professor, Department of Electronics and Communication Engineering, NIT Rourkela**, who has been the driving force behind this work. I thank him for giving me the opportunity to work under him by putting a trust in my credentials and capabilities, and helping me in exploring my potential to the fullest.

I am grateful to **Prof. S.K Patra**, Head of the Department of Electronics and Communication Engineering, for permitting me to make use of the facilities available in the department to carry out the project successfully.

I am thankful to **Mr. Ayaskant Swain** and **Mr. Vijay Sharma**, both PG students in the Department of Electronics and Communication Engineering, NIT Rourkela, for their generous help and continuous encouragement in various ways towards the completion of this project.

Last but not the least I would like to thank all my friends for their support. I am thankful to my classmates for all the thoughtful and mind stimulating discussions we had, prompting me to think beyond the obvious.

Rohit Kumar Jain

ABSTRACT

CORDIC or **CO**-ordinate **R**otation **DI**gital **C**omputer is a fast, simple, efficient and powerful algorithm used for diverse Digital Signal Processing applications. Primarily developed for real-time airborne computations, it uses a unique computing technique [7] which is especially suitable for solving the trigonometric relationships involved in plane co-ordinate rotation and conversion from rectangular to polar form. It comprises a special serial arithmetic unit having three shift registers, three adders/subtractors, Look-Up table and special interconnections. Using a prescribed sequence of conditional additions or subtractions the CORDIC arithmetic unit can be controlled to solve either of the following equations:

$$Y' = K\,(Y\cos\lambda + X\sin\lambda)$$

$$X' = K\,(X\cos\lambda - Y\sin\lambda);\quad \text{where K is a constant}$$

In this project:

- A CORDIC-based processor for sine/cosine calculation was designed using VHDL programming in Xilinx ISE 10.1. The CORDIC module was tested for its functionality and correctness by test-bench analysis. Subsequently, FPGA implementation of the CORDIC core followed by ChipScopePro analysis of the output logic waveforms was performed.
- Using this CORDIC core a DCT processor was designed to calculate the 8-point 1D DCT. The functionality and operational correctness of this processor was tested, first on the test-bench and then via ChipScopePro analysis, post FPGA implementation.

The output obtained in both the cases was compared with the actual values to test for consistency and the percentage of accuracy was established. Power consumption and FPGA resource utilization were observed. The results obtained were discussed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

For a long time the field of Digital Signal Processing has been dominated by Microprocessors. This is mainly because they provide designers with the advantages of single cycle multiply-accumulate instruction as well as special addressing modes. Although these processors are cheap and flexible they are relatively slow when it comes to performing certain demanding signal processing tasks e.g. Image Compression, Digital Communication and Video Processing. Of late, rapid advancements have been made in the field of VLSI and IC design. As a result special purpose processors with custom-architectures have come up. Higher speeds can be achieved by these customized hardware solutions at competitive costs. To add to this, various simple and hardware-efficient algorithms exist which map well onto these chips and can be used to enhance speed and flexibility while performing the desired signal processing tasks [1][2][3].

One such simple and hardware-efficient algorithm is CORDIC, an acronym for Coordinate Rotation Digital Computer, proposed by Jack E Volder [7]. CORDIC uses only Shift-and-Add arithmetic with table Look-Up to implement different functions. By making slight adjustments to the initial conditions and the LUT values, it can be used to efficiently implement Trigonometric, Hyperbolic, Exponential functions, Coordinate Transformations etc. using the same hardware. Since it uses only shift-add arithmetic, VLSI implementation of such an algorithm is easily achievable.

DCT algorithm has diverse applications and is widely used for Image compression. Implementing DCT using CORDIC algorithm reduces the number of computations during processing, increases the accuracy of reconstruction of the image, and reduces the chip area of implementation of a processor built for this purpose. This reduces the overall power consumption.

FPGA provides the hardware environment in which dedicated processors can be tested for their functionality. They perform various high-speed operations that cannot be realized by a simple microprocessor. The primary advantage that FPGA offers is On-site programmability. Thus, it forms the ideal platform to implement and test the functionality of a dedicated processor designed using CORDIC algorithm [5].

1.2 Problem Statement

The primary objective of this project is to design 8-point 1D DCT processor using CORDIC algorithm in VHDL, implement this design on a FPGA, verify and test for its functionality, and analyze its performance.

1.3 Organization of Thesis

The Thesis has been divided into five chapters including this one. Chapter 1 gives a basic introduction to the project and the motivation behind it. Chapter 2 deals with literature review of the essentials of the project i.e. CORDIC, Discrete Cosine Transform and Field Programmable Gate Arrays. The third chapter presents the different algorithms and architectures available during the design of the processor. Chapter 4 presents the results and related discussions. Conclusion and future work is proposed in Chapter 5.

Chapter 2

Literature Review

2.1 CORDIC Overview

CORDIC or Coordinate Rotation Digital Computer is a simple and hardware-efficient algorithm for the implementation of various elementary, especially trigonometric, functions. Instead of using Calculus based methods such as polynomial or rational functional approximation, it uses simple shift, add, subtract and table look-up operations to achieve this objective. The CORDIC algorithm was first proposed by Jack E Volder in 1959. It is usually implemented in either Rotation mode or Vectoring mode. In either mode, the algorithm is rotation of an angle vector by a definite angle but in variable directions. This fixed rotation in variable direction is implemented through an iterative sequence of addition/subtraction followed by bit-shift operation. The final result is obtained by appropriately scaling the result obtained after successive iterations. Owing to its simplicity the CORDIC algorithm can be easily implemented on a VLSI system.

2.1.1 Advantages

- Hardware requirement and cost of CORDIC processor is less as only shift registers, adders and look-up table (ROM) are required
- Number of gates required in hardware implementation, such as on an FPGA, is minimum as hardware complexity is greatly reduced compared to other processors such as DSP multipliers
- It is relatively simple in design
- No multiplication and only addition, subtraction and bit-shifting operation ensures simple VLSI implementation.
- Delay involved during processing is comparable to that during the implementation of a division or square-rooting operation.
- Either if there is an absence of a hardware multiplier (e.g. uC, uP) or there is a necessity to optimize the number of logic gates (e.g. FPGA) CORDIC is the preferred choice.

2.1.2 Disadvantages

- Large number of iterations required for accurate results and thus the speed is low and time delay is high
- Power consumption is high in some architecture types
- Whenever a hardware multiplier is available, e.g. in a DSP microprocessor, table look-up methods and good old-fashioned power series methods are generally quicker than this CORDIC algorithm.

2.1.3 Applications

- The algorithm was basically developed to offer digital solutions to the problems of real-time navigation in B-58 bomber [7].
- John Walther extended the basic CORDIC theory to provide solution to and implement a diverse range of functions [8].
- This algorithm finds use in 8087 Math coprocessor [11], the HP-35 calculator [15], radar signal processors [15], and robotics.
- CORDIC algorithm has also been described for the calculation of DFT [4], DHT [4], Chirp Z-transforms [12], filtering [10], Singular value decomposition [14], and solving linear systems [9].
- Most calculators especially the ones built by Texas Instruments and Hewlett-Packard use CORDIC algorithm for calculation of transcendental functions.

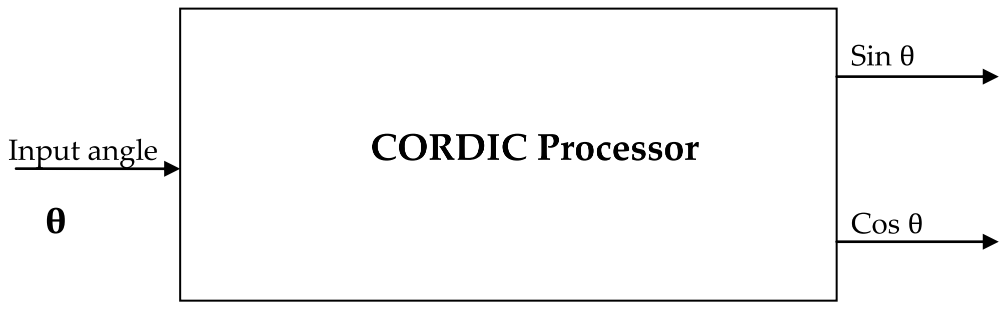


Fig. 2.1: Block Diagram of a CORDIC processor

2.2 Discrete Cosine Transformation

Discrete Cosine Transformation (DCT) is the most widely used transformation algorithm. DCT, first proposed by Ahmed [9] et al, 1974, has got more importance in recent years, especially in the fields of Image Compression and Video Compression. This chapter focuses on efficient hardware implementation of DCT by decreasing the number of computations, enhancing the accuracy of reconstruction of the original data, and decreasing chip area. As a result of which the power consumption also decreases. DCT also improves speed, as compared to other standard Image compression algorithms like JPEG.

2.2.1 One Dimensional DCT

The Discrete Cosine Transform of a one dimensional sequence of length N is defined as

$$F(u) = \alpha(u) \sum f(x) \cos[pi(2x+1)u/2N]; \text{ for } u=0,1,2\ldots N-1 \quad (2.1)$$

The Inverse DCT transformation is defined as

$$f(x) = \sum \alpha(u)\, c(u) \cos[pi(2x+1)u/2N]; \text{ for } x=0,1,2\ldots N-1 \quad (2.2)$$

In both the equations above $\alpha(u)$ is defined as

$$\alpha(u) = \text{sqrt}(1/N) \quad \text{for } u=0$$

$$\text{sqrt}(2/N) \quad \text{for } u\neq 0$$

For more than N sample points the entire sequence can be divided into sequences of length N and then DCT can be independently applied to these smaller blocks.

2.2.2 Fundamental Properties of Discrete Cosine Transformation

1) **Decorrelation**

   The neighboring pixels of an image are generally correlated. As a result of which using alternate methods of image compression, coding techniques, introduces redundancy. DCT reduces the correlation between neighboring coefficients, thus, enabling the uncorrelated transform coefficients to be encoded independently.

2) Energy Compaction

DCT provides excellent energy compaction for uncorrelated images. It packs input data into as few coefficients as possible allowing removal of coefficients with relatively less amplitudes during quantization, without introducing visual distortion in the reconstructed image. It is observed that the uncorrelated image exhibits sharper energy variations, unlike the correlated one, showing that it has got higher frequency content.

3) Separability

The DCT transform equation can also be expressed as,

$$C(u,v) = \sum \sum f(x,y) \cos[pi\ (2x+1)u/2N] \cos\ [pi(2y+1)v/2N]$$

for u,v= 0,1,2…N-1 (2.3)

This is known as the Separability property of the DCT and has the main advantage that a 2D operation can be split into two successive 1D operations on the rows and columns. During hardware design this property is utilized.

2.3 FPGA

FPGA or Field Programmable Gate Arrays can be programmed or configured by the user or designer after manufacturing and during implementation. Hence they are otherwise known as On-Site programmable. Unlike a Programmable Array Logic (PAL) or other programmable device, their structure is similar to that of a gate-array or an ASIC. Thus, they are used to rapidly prototype ASICs, or as a substitute for places where an ASIC will eventually be used [17]. This is done when it is important to get the design to the market first. Later on, when the ASIC is produced in bulk to reduce the NRE cost, it can replace the FPGA. The programming of the FPGA is done using a logic circuit diagram or a source code using a Hardware Description Language (HDL) to specify how the chip should work. FPGAs have programmable logic components called "logic blocks", and a hierarchy or reconfigurable interconnects which facilitate the "wiring" of the blocks together. The programmable logic blocks are called configurable logic blocks and reconfigurable interconnects are called switch boxes. Logic blocks (CLBs) can be programmed to perform complex combinational functions, or simple logic gates like AND and XOR. In most FPGAs the logic blocks also include memory elements, which can be as simple as a flip-flop or as complex as complete blocks of memory.

2.3.1 FPGA Architecture

FPGA architecture depends on its vendor, but they are usually variation of that shown in the figure. The architecture comprises Configurable Logic Blocks, Configurable I/O blocks and Programmable Interconnects. It also houses a clock circuitry to drive the clock signals to each logic block. Additional logic resources like ALUs, Decoders and memory may be available. Static Ram and anti-fuses are the two basic types of programmable elements for an FPGA. The number of CLBs and I/Os required can easily be determined from the design but the number of routing tracks is different even within the designs employing the same amount of logic.

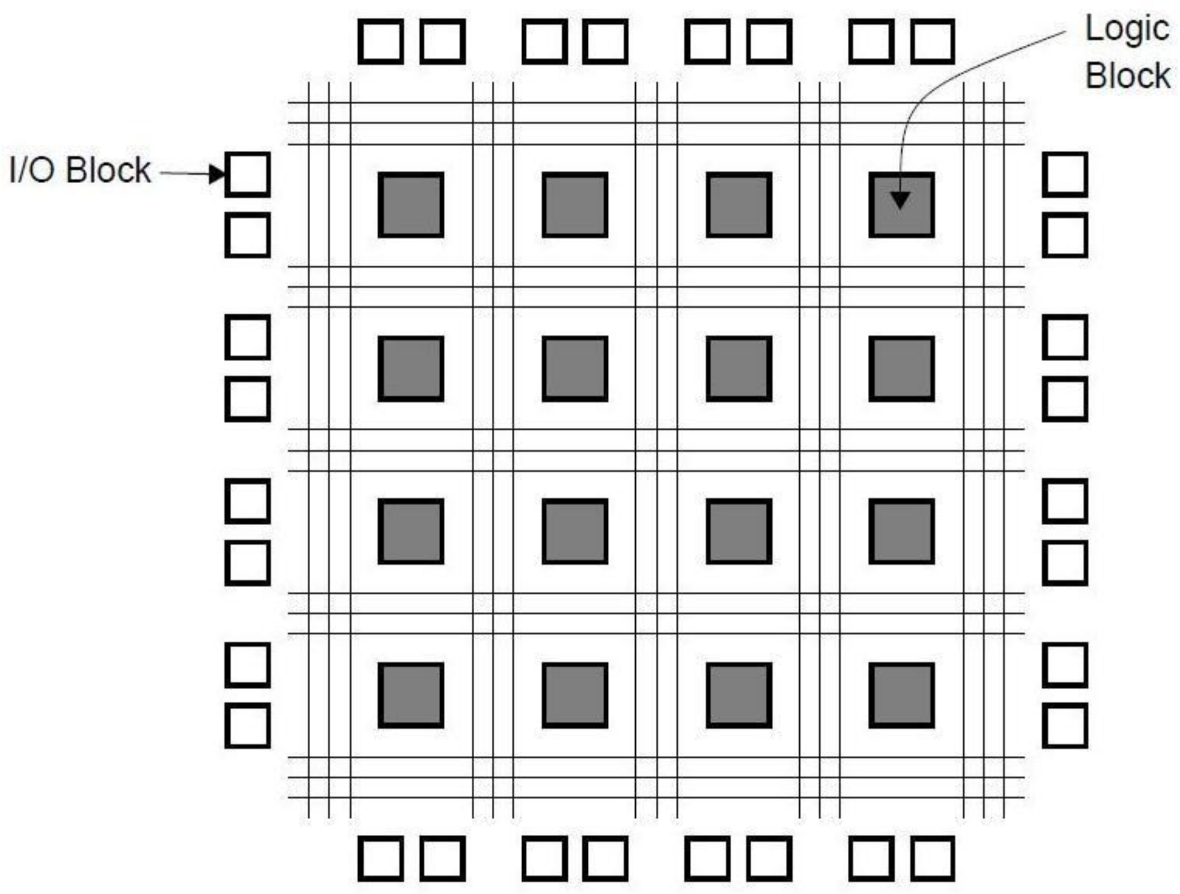


Fig. 2.2: FPGA Architecture [16]

1. Configurable Logic Blocks

They contain the logic for the FPGA. CLBs contain RAM for creating arbitrary combinatorial logic functions. It also has flip-flops for clocked storage elements, and multiplexers that route the logic within the block to/from external resources.

2. Configurable I/O Blocks

Configurable I/O block is used to route signal towards and away from the chip. It comprises input buffer, output buffer with three states and open collector output controls. Pull-up and Pull-down resistors may also be present at the output. The output polarity is programmable for active high or active low output.

3. Programmable Interconnects

FPGA interconnect is similar to that of a gate array ASIC and different from a CPLD. There are long lines that interconnect critical CLBs located physically far from each other without introducing much delay. They also serve as buses within the chip. Short lines that interconnect CLBs present close to each other are also present. Switch matrices that connect these long and short lines in a specific way are also present. Programmable Switches connect CLBs to interconnect lines and interconnect lines to each other and the switch matrix. Three-state buffers connect multiple CLBs to a long line creating a bus. Specially designed long lines called Global Clock lines are present that provide low impedance and fast propagation times.

4. Clock circuitry

Special I/O blocks having special high-drive clock buffers, called clock drivers, are distributed throughout the chip. The buffers are connected to clock I/P pads. They drive the clock signals onto the Global Clock liens described above. The clock lines have been designed for fast propagation time and less skew time.

2.3.2 FPGA Design Flow

The flow for the design using FPGA outlines the whole process of device design, and guarantees that none of the steps is overlooked. Thus, it ensures that we have the best chance of getting back a working prototype that will correctly function in the final system to be designed.

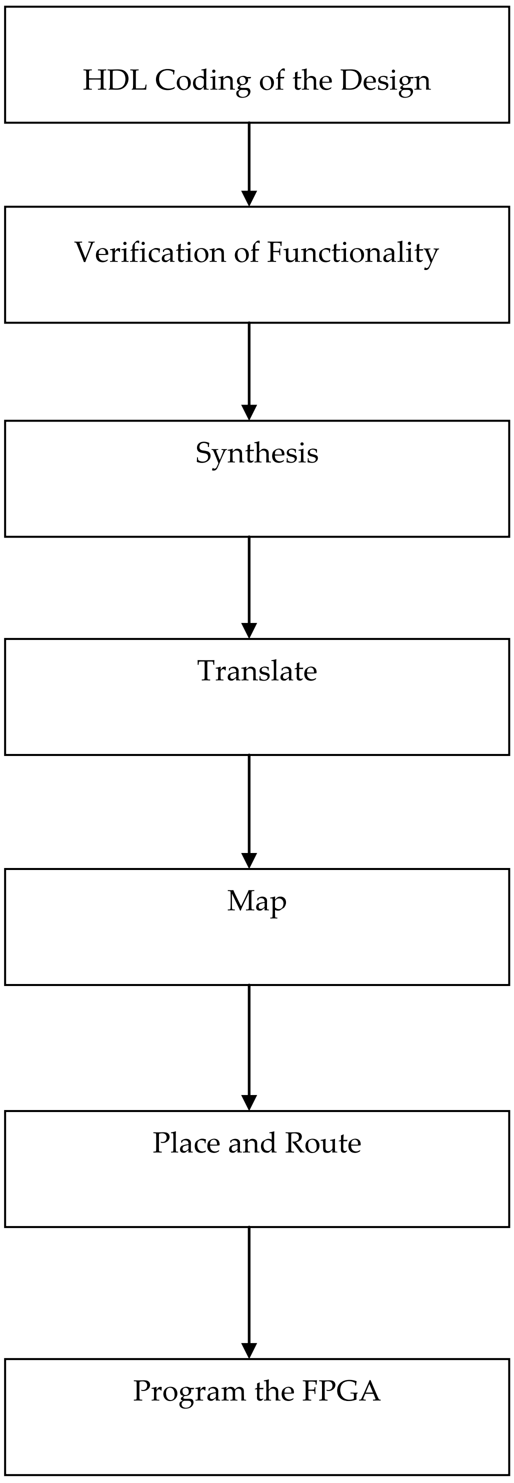


Fig. 2.3: FPGA Design Flow

2.3.3 Behavioral Simulation [2]

After HDL designing, the code is simulated and its functionality is verified using simulation software, e.g. Xilinx ISE or ModelSim simulator. The code is simulated and the output is tested for the various inputs. If the output values are consistent with the expected values then we proceed further else necessary corrections are made in the code. This is what is known as Behavioral Simulation. Simulation is a continuous process. Small sections of the design should be simulated and verified for functionality before assembling them into a large design. After several iterations of design and simulation the correct functionality is achieved. Once the design and simulation is done then another design review by some other people is done so that nothing is missed and no improper assumption made as far as the output functionality is concerned.

2.3.4 Synthesis of Design

Post the behavioral simulation the design is synthesized. During simulation following takes place:

(i) HDL Compilation

The Xilinx ISE tool compiles all the sub-modules of the main module. If any problem takes place then the syntax of the code must be checked.

(ii) HDL synthesis

Hardware components like Multiplexers, Adders, Subtractors, Counters, Registers, Latches, Comparators, XORs, Tri-State buffers, Decoders are synthesized from the HDL code.

2.3.5 Design Implementation [2]

(i) Translation

The translate process is used to merge all of the input net-lists and the design constraints. It outputs a Xilinx NGD (Native Information and Generic Database) file. The logical design reduced to Xilinx device primitive cells is described by this .ngd file. Here, User Constraints are defined by assigning the ports in the design to physical elements (e.g. pins, switches, buttons, etc) for the target device as well as specifying timing requirements. This information is stored in a UCF file which can be created using PACE or Constraint Editor.

(ii) Mapping

After the translation process is complete the logical design described in the .ngd file to the components or primitives (Slices/CLBs) present on the .ncd file is mapped onto the target FPGA design. The whole circuit is divided into smaller blocks so that they can be appropriately fit into the FPGA blocks. The mapping is done onto the CLBs and IOBs in accordance with the logic.

(iii) Placing and Routing

After the mapping process the PAR program is used to place the sub-blocks from the map process onto the logic blocks as per the constraints and then connect these blocks. Trade-off between all the constraints is taken into account during the placement and routing process. Place process places the sub-blocks according to logic but does not provide them the physical routing. On running the Route process physical connections between the sub-blocks are made using the switch-matrices.

(iv) Bit file generation

Bit-stream is used to describe the collection of binary data used to program the reconfigurable logic device. The 'Generate Programming File" process is run after the FPGA design has been completely routed. It runs BitGen, the Xilinx bit-stream generation program, to produce a .bit or .isc file for Xilinx device configuration. Using this file the device is configured for the intended design using the JTAG boundary scan method. The working is then verified for different inputs.

(v) Testing

System testing is necessary to ensure that all parts of the system correctly work together after the prototype is mapped onto the system. If the system doesn't work then the problem can be fixed by making some changes in the system or the software. The problems are documented so that on the next revision or production of the chip they are fixed. When the ICs are produced it is necessary to have some sort of burnt-in self-test mechanism such that the system gets tested regularly over a long period of time [ref].

2.3.6 Advantages of FPGA [5]

FPGAs have become very popular in the recent years owing to the following advantages that they offer:

- **Fast prototyping and turn-around time**- Prototyping is the defined as the building of an actual circuit to a theoretical design to verify for its working, and to provide a physical platform for debugging the core if it doesn't. Turnaround is the total time between expired between the submission of a process and its completion. On FPGAs interconnects are already present and the designer only needs to fuse these programmable interconnects to get the desired output logic. This reduces the time taken as compared to ASICs or full-custom design.
- **NRE cost is zero-** Non-Recurring Engineering refers to the one-time cost of researching, developing, designing and testing a new product. Since FPGAs are reprogrammable and they can be used without any loss of quality every time, the NRE cost is not present. This significantly reduces the initial cost of manufacturing the ICs since the program can be implemented and tested on FPGAs free of cost.
- **High-Speed**- Since FPGA technology is primarily based on referring to the look-up tables the time taken to execute is much less compared to ASIC technology.
- **Low cost-** FPGA is quite affordable and hence is very designer-friendly. Also the power requirement is much less as the architecture of FPGAs is based upon LUTs.

Due to the above mentioned advantages of FPGAs in IC technology and DCT in mapping of images, implementation of DCT in FPGA can give us a clearer idea about the advantages and limitations of using DCT as the mapping function. This can help in forming better image compression and restoration techniques.

2.3.7 FPGA Specifications

The FPGA used in this project has the following specifications:

- Vendor: Xilinx
- Family: Spartan 3E
- Family: XC3S500E
- Package: FG320
- Speed grade: -5
- Synthesis Tool: VHDL
- Simulator: Xilinx ISE 10.1

Chapter 3

Architectures and Algorithms

3.1 CORDIC Algorithm

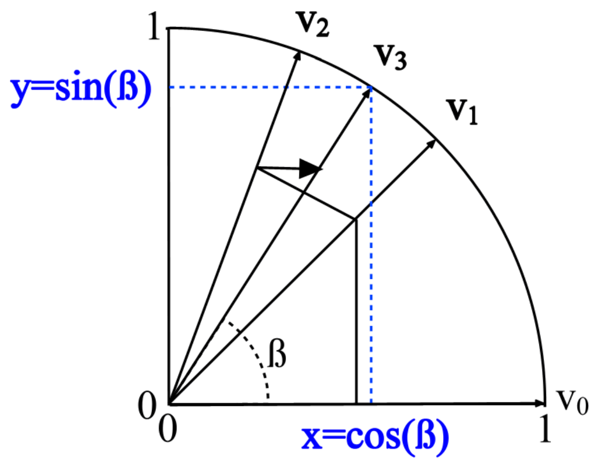


Fig.3.1: Graphical Demonstration of CORDIC Algorithm

3.1.1 Evaluation of Sine and Cosine values

This algorithm calculates the sine and cosine values of a given angle (in radians) by transforming the co-ordinates from polar representation to representation in Cartesian form. To measure the values of sine and cosine, the coordinates corresponding to the angle on a unit circle is found, the x-coordinate of which gives the cosine values while the y-coordinate gives the sine value.

We start with the vector v0 aligned with the x-axis*:*

$$v_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad (3.1)$$

Here, the x-coordinate is 1, whereas the y-coordinate is 0.

In the first iteration, this vector is rotated 45° counterclockwise to get the vector v1. Successive iterations will rotate the vector in one or the other direction by size decreasing steps, until the desired angle has been achieved. The direction of rotation of the CORDIC vector is decided by the value of β_i where β_i is the difference between the rotation value to be reached and the present angle accumulator value. CORDIC algorithm depends on the value of this parameter β_i which in turn is decided by value of co-ordinate y.

Step size is:

atan(1/(2$^{(i-1)}$)); where i= 1,2,3,…. (3.2)

Whenever one complex number is multiplied with another the magnitude of the resultant complex number is the product of the individual magnitudes and its phase is sum of the individual phases. Thus, rotation of a vector can be achieved by simply multiplying the vector with a complex number of unit magnitude. This is known as real rotation. The subsequent values of the vector v_i is given by

$$v_i = R_i v_{i-1} \qquad (3.3)$$

The rotation matrix R is given by:

$$R_i = \begin{pmatrix} \cos\gamma_i & -\sin\gamma_i \\ \sin\gamma_i & \cos\gamma_i \end{pmatrix} \qquad (3.4)$$

Using the following two trigonometric identities

$$\cos\alpha = \frac{1}{\sqrt{1+\tan^2\alpha}}$$
$$\sin\alpha = \frac{\tan\alpha}{\sqrt{1+\tan^2\alpha}}$$

The rotation matrix becomes:

$$R_i = \frac{1}{\sqrt{1+\tan^2\gamma_i}}\begin{pmatrix} 1 & -\tan\gamma_i \\ \tan\gamma_i & 1 \end{pmatrix} \qquad (3.5)$$

The expression for the rotated vector $\boldsymbol{v_i = R_i v_{i-1}}$ then becomes:

$$v_i = \frac{1}{\sqrt{1+\tan^2\gamma_i}}\begin{pmatrix} x_{i-1} & - & y_{i-1}\tan\gamma_i \\ x_{i-1}\tan\gamma_i & + & y_{i-1} \end{pmatrix} \qquad (3.6)$$

Where: x_{i-1} and y_{i-1} are the components of v_{i-1}

The resultant rotations described by the vector in the figure described below are known as Pseudo-rotations. Volder proposed that this rotation angle θ can be broken down into a series of small successive shrinking angles. Restricting the angles γ_i so that **tan** γ_i takes on the values $\pm 2^{-i}$ the multiplication with the tangent can be replaced by a division by a power of two, which is efficiently done in digital computer hardware using a bit shift. The expression then becomes:

$$v_i = K_i \begin{pmatrix} x_{i-1} & - & \sigma_i 2^{-i} y_{i-1} \\ \sigma_i 2^{-i} x_{i-1} & + & y_{i-1} \end{pmatrix} \qquad (3.7)$$

where

$$K_i = \frac{1}{\sqrt{1 + 2^{-2i}}}$$

$\boldsymbol{\sigma_i}$ can have the values of −1 or 1 and is used to determine the direction of the rotation: if the angle $\boldsymbol{\beta_i}$ is positive then σ_i is 1 otherwise it is −1.

The value of β_i is given as,

$$\beta_i = \beta_{i-1} - \sigma_i \gamma_i; \qquad (3.8)$$

Where $\gamma_i = \arctan 2^{-i}$ and the initial β value is the angle for which the sine, cosine values are to be calculated.

Table 3.1: Successive Angle Rotation Values

Iteration	**tan(α(i)))**	**α(i) (in degrees)**
0	1	45
1	0.5	26.5
2	0.25	14.03
3	0.125	7.125
4	0.0625	3.576
5	0.03125	1.7899
6	0.015625	0.895
7	0.00781	0.4476

8	0.00390	0.2238
9	0.001953125	0.1
10	0.0009765625	0.055

Thus, the new x and y coordinate values can be given as

$$X_i = X_{i-1} - 2^{-i} Y_{i-1}$$

$$Y_i = Y_{i-1} + 2^{-i} \mathbf{X}_{i-1}, \qquad \text{when } \beta_i > 0 \qquad (3.9)$$

and

$$X_i = X_{i-1} + 2^{-i} Y_{i-1}$$

$$Y_i = Y_{i-1} - 2^{-i} X_{i-1}, \qquad \text{when } \beta_i < 0 \qquad (3.10)$$

3.1.2 Scaling Factor

We can ignore K_i in the iterative process and then apply it afterward by a scaling factor:

$$K(n) = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} 1/\sqrt{1 + 2^{-2i}} \qquad (3.11)$$

which is calculated in advance and stored in a table, or as a single constant for a fixed number of iterations. This correction could also be made in advance, by scaling *v0* and hence saving a multiplication. Additionally it can be noted that

$$K = \lim_{n \to \infty} K(n) \approx 0.6072529350088812561694 \qquad (3.12)$$

Note: Here the radix 2 system is used since it avoids use of multiplications while implementing the above equation. Hence a CORDIC iteration can be realized using shifters and adders only.

3.2 Basic Architecture

The following diagram explains the basic hardware architecture of a CORDIC processor. It shows the adders/subtractors and the shift registers. The adders/subtractors perform the addition/subtraction of binary numbers. The shift register performs the bit-shift operation in accordance with the algorithm. The constants corresponding to fixed angle values are obtained from the Look-up table implemented as a ROM.

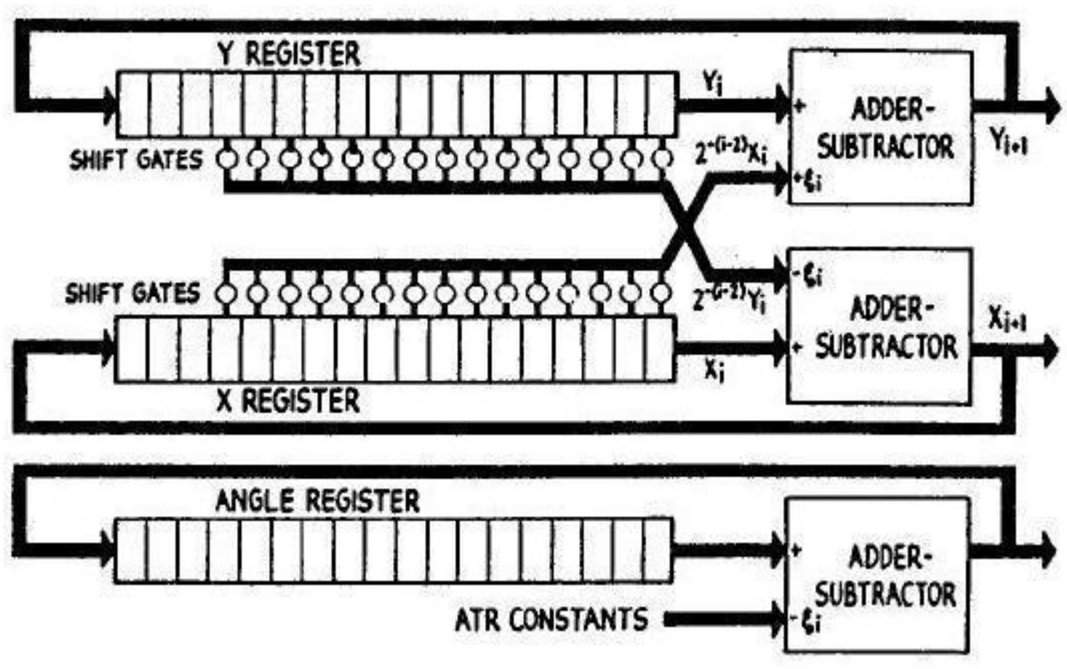


Fig. 3.2: Basic CORDIC Architecture [7]

3.3 Types of CORDIC Architecture

CORDIC algorithm, for calculation of sine and cosine values, is of three types. Each of the types has its own advantages and disadvantages depending upon the type of use intended.

The three types are: [4]

1*) Sequential or iterative*

2) Parallel or cascaded

3) Pipelined

3.3.1 Sequential or Iterative CORDIC structure:

In this type of CORDIC architecture, a single iteration process takes place in a single clock cycle. The sequential CORDIC structure is as shown below:

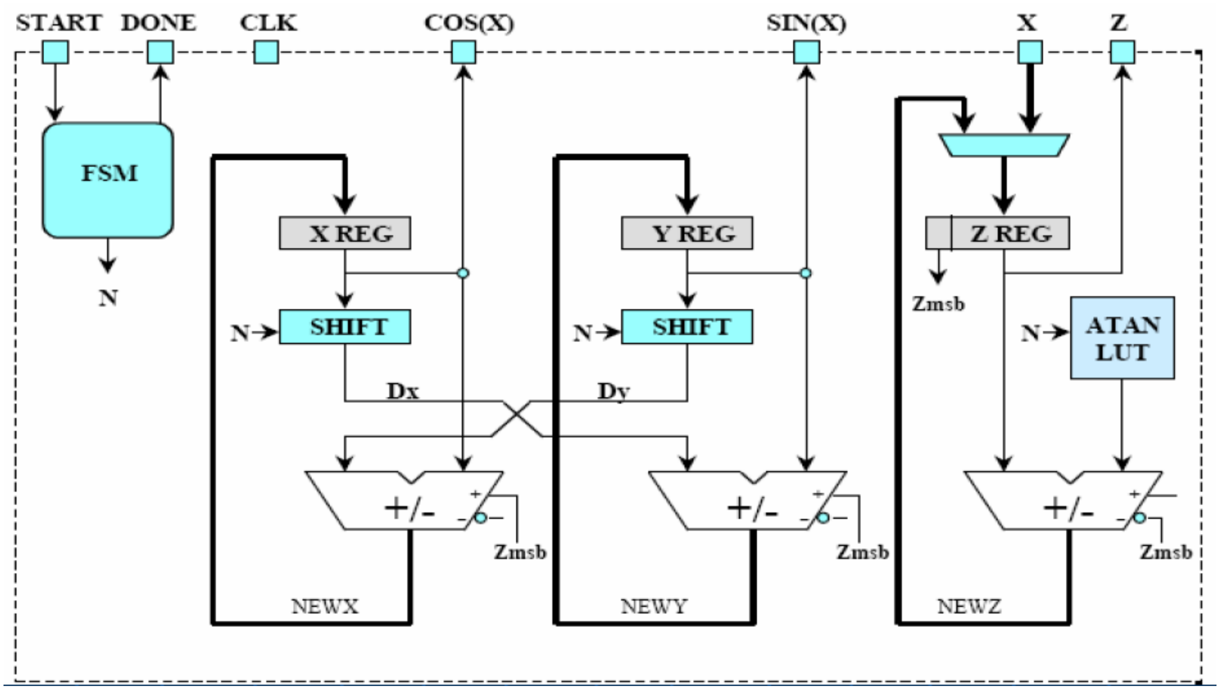


Fig.3.3: Sequential/Iterative CORDIC structure

3.3.1.1 Advantages

- The hardware complexity is least and it occupies the least area.
- It has maximum number of clock cycles per iteration.
- Power consumption is least.

3.3.1.2 Disadvantages

- Maximum number of clock cycles are required to calculate the output, thus calculation time is large.
- Variable shifters do not map well on certain FPGAs due to high fan-in.

3.3.2 Parallel or Cascaded CORDIC architecture:

In this type of architecture, all the iterations take place in a single clock cycle. The architecture is as shown below:

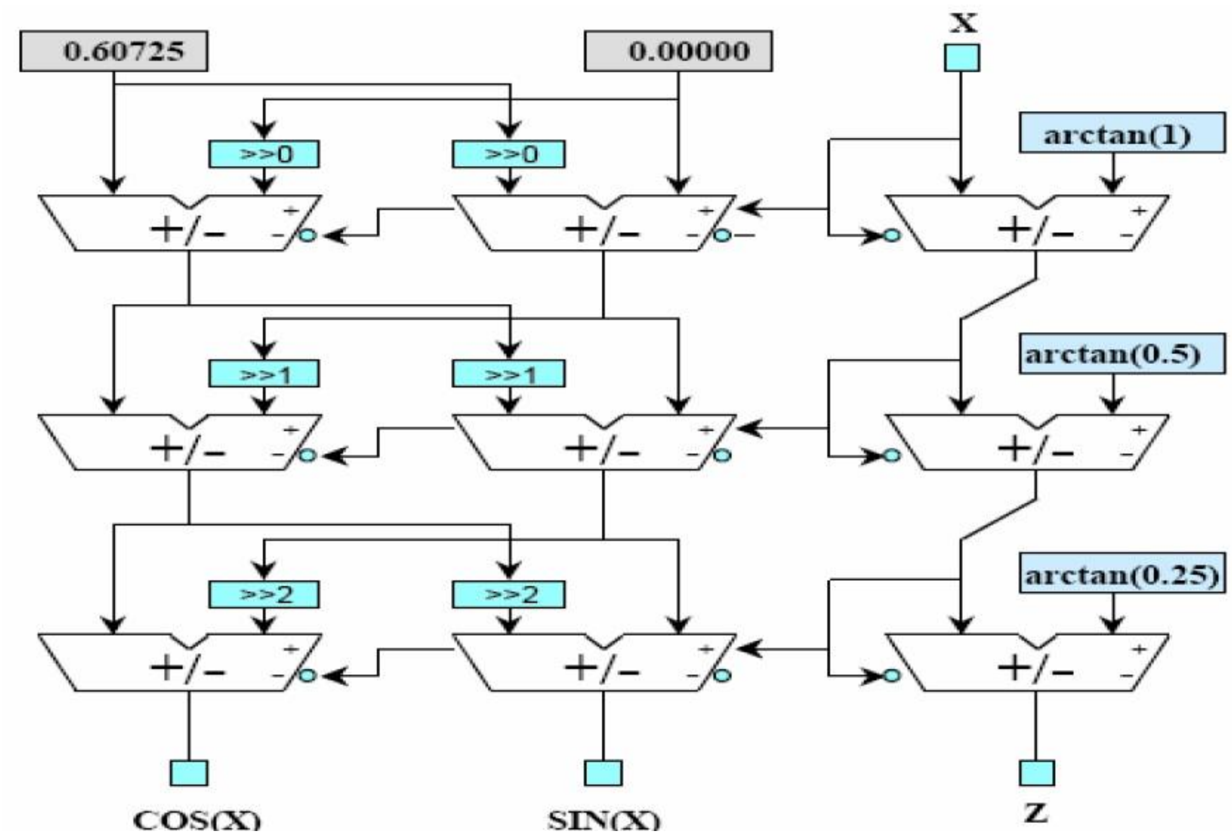


Fig.3.4: Block diagram of Parallel CORDIC architecture

3.3.2.1 Advantages

- It has considerable delay, but processing time is reduced as compared to the iterative process.
- Shifters are of fixed size and so can be implemented in the wiring.
- Constants can be hardwired instead of requiring storage space.

3.3.2.2 Disadvantages

- The amount of hardware required is large and the area required is maximum
- Power consumption is highest among the three CORDIC architectures.

3.3.3 Pipelined CORDIC architecture:

It is comparatively the most efficient CORDIC architecture. In this method multiple iterations take place in multiple clock cycles. It is implemented by inserting registers within the different adder stages.

The architecture is given as:

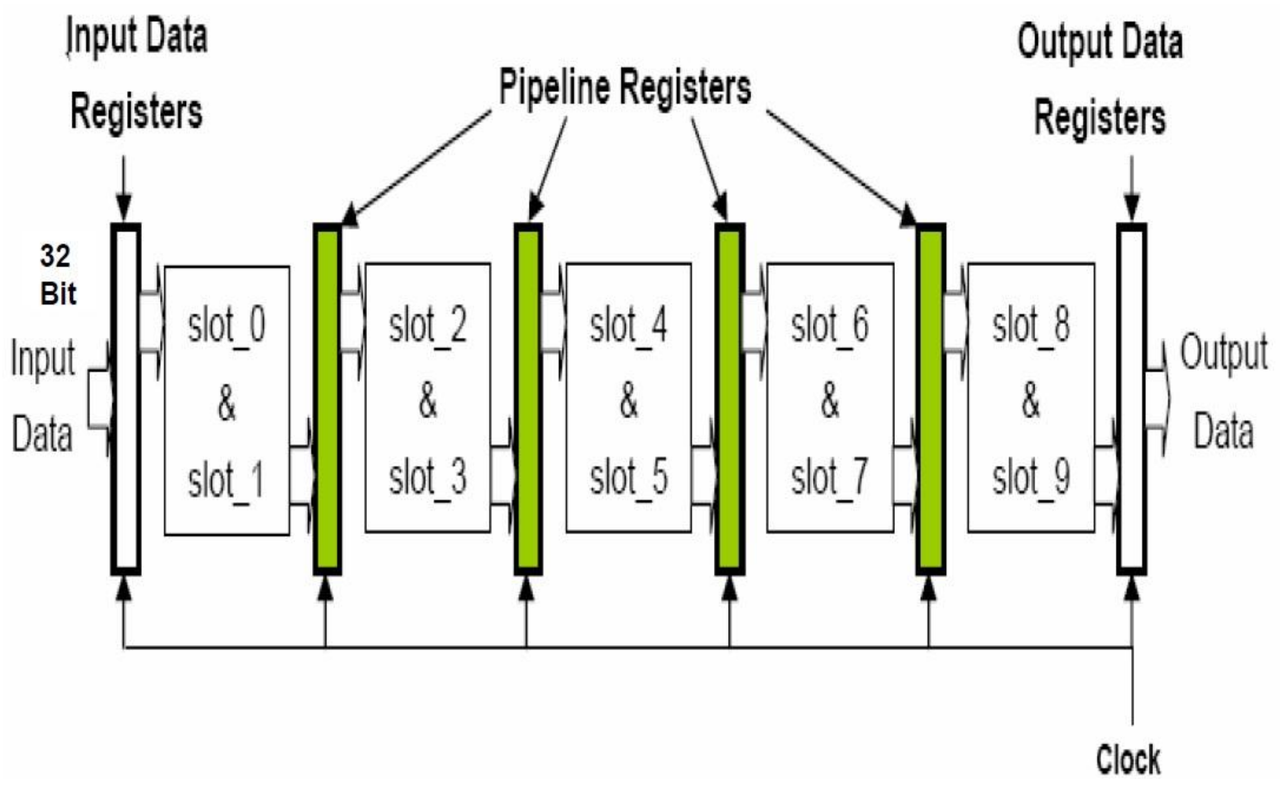


Fig.3.5: Block diagram of Pipe-lined CORDIC Architecture

3.3.3.1 Advantages

- FPGA implementation is easy, as registers are already available, thus requiring no extra hardware.
- Number of iterations after which the system gives accurate result can be modeled, considering clock frequency of the system.
- When operating at greater clock period power consumption in later stages reduces due to lesser switching activity in each clock period.

3.3.3.2 Disadvantages

- Hardware complexity as well as area required is more than sequential architecture
- Power consumption is lower than parallel but higher than sequential structure.

3.4 DCT Implementation

Implementation of DCT using a CORDIC architecture is described here:

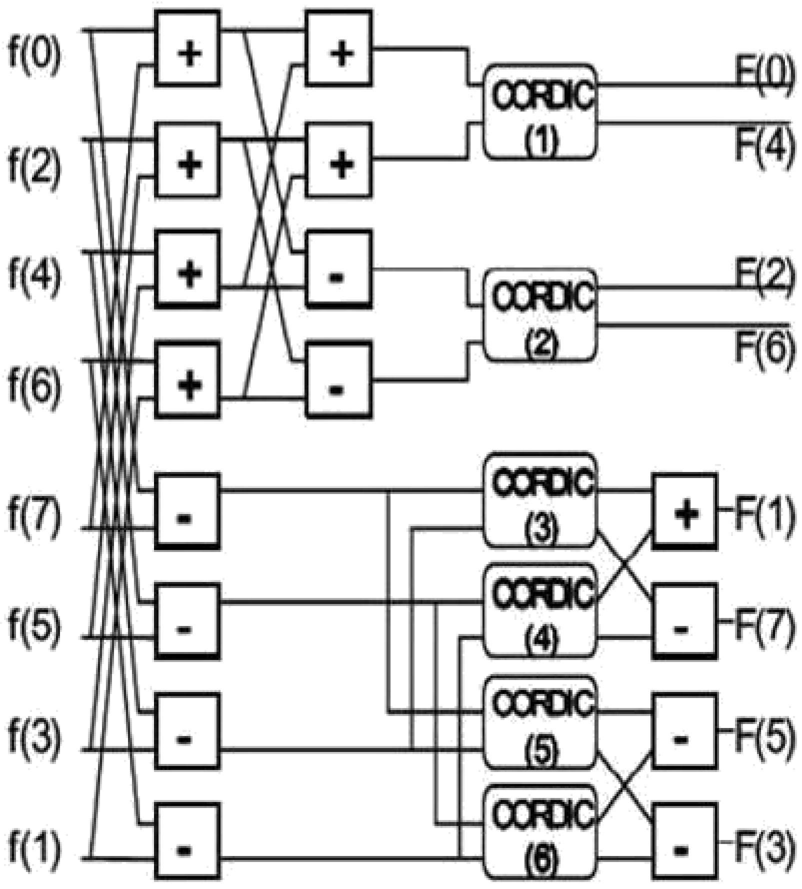


Fig.3.6: Architecture for DCT using CORDIC [3]

3.4.1 Mathematical Expression

Thus for calculating DCT using CORDIC the set of equations obtained is:

1. F(0)= 0.5 * (f(0)+ f(1) + f(2) + f(3) + f(4) + f(5) + f(6) + f(7)) * cos (pi/4)
2. F(1)= 0.5 * ((f(0)-f(7)) * cos(pi/16) + (f(1)-f(6)) * cos(3pi/16) + (f(2)-f(5)) * cos(5pi/16) + (f(3)-f(4)) * cos(7pi/16))
3. F(2)= 0.5 * ((f(0)-f(3)-f(4)+f(7)) * cos(2pi/16) + (f(1)-f(2)-f(5)+f(6)) * cos(6pi/16))
4. F(3)= 0.5 * ((f(0)-f(7)) * cos(3pi/16) + (f(6)-f(1)) * cos(7pi/16 + (f(5)-f(2)) * cos(pi/16) + (f(4)-f(3))* cos(5pi/16))
5. F(4)= 0.5 * (f(0)+f(3)+f(4)+f(7)-f(1)-f(2)-f(5)-f(6)) * cos(pi/4))
6. F(5)= 0.5 * ((f(0)-f(7)) * cos(5pi/16) + (f(6)-f(1)) * cos(pi/16) + (f(2)-f(5)) * cos(7pi/16) + (f(3)-f(4))* cos(3pi/16))
7. F(6)= 0.5 * ((f(0)-f(3)-f(4)+f(7)) * cos(6pi/16) - (f(1)-f(2)-f(5)+f(6)) * cos(2pi/16))
8. F(7)=0.5 * (f(0)-f(7)) * cos(7pi/16) + (f(6)-f(1)) * cos(5pi/16) + (f(2)-f(5)) * cos(3pi/16) + (f(4)-f(3))* cos(pi/16)

Chapter 4

Results and Discussions

The following figure shows the RTL schematic of the CORDIC processor (top) generated from Xilinx ISE simulator.

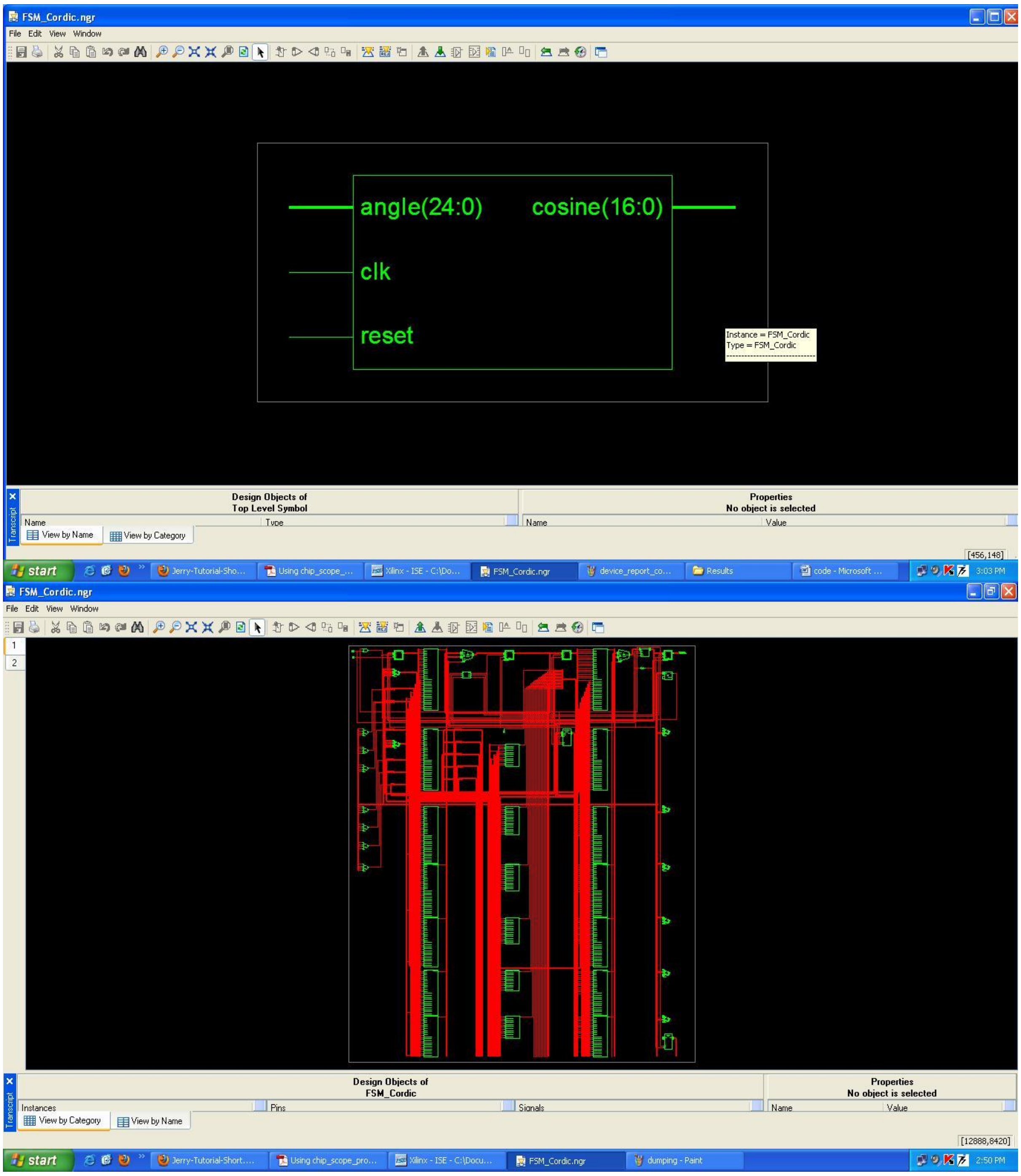


Fig.4.1: Different Views for CORDIC RTL Schematic

It has CLK, reset and angle as inputs and cosine as output. The input angle is of size 25 bits containing 1 sign-bit, 2 bits that represent the integer part and 22 bits that represent the fractional part of the input angle when expressed in radians. The cosine output is of size 17 bits with 1 sign bit and 16 bits that represent the fractional part.

The following figure is the test-bench simulation of the CORDIC processor using Xilinx ISE simulator. It shows how the output (sine and cosine values) converges to a particular value with the number of iterations n. It also shows the various intermediate signals like tmp_x, tmp_y, x, y etc. and the 4-state transitions.

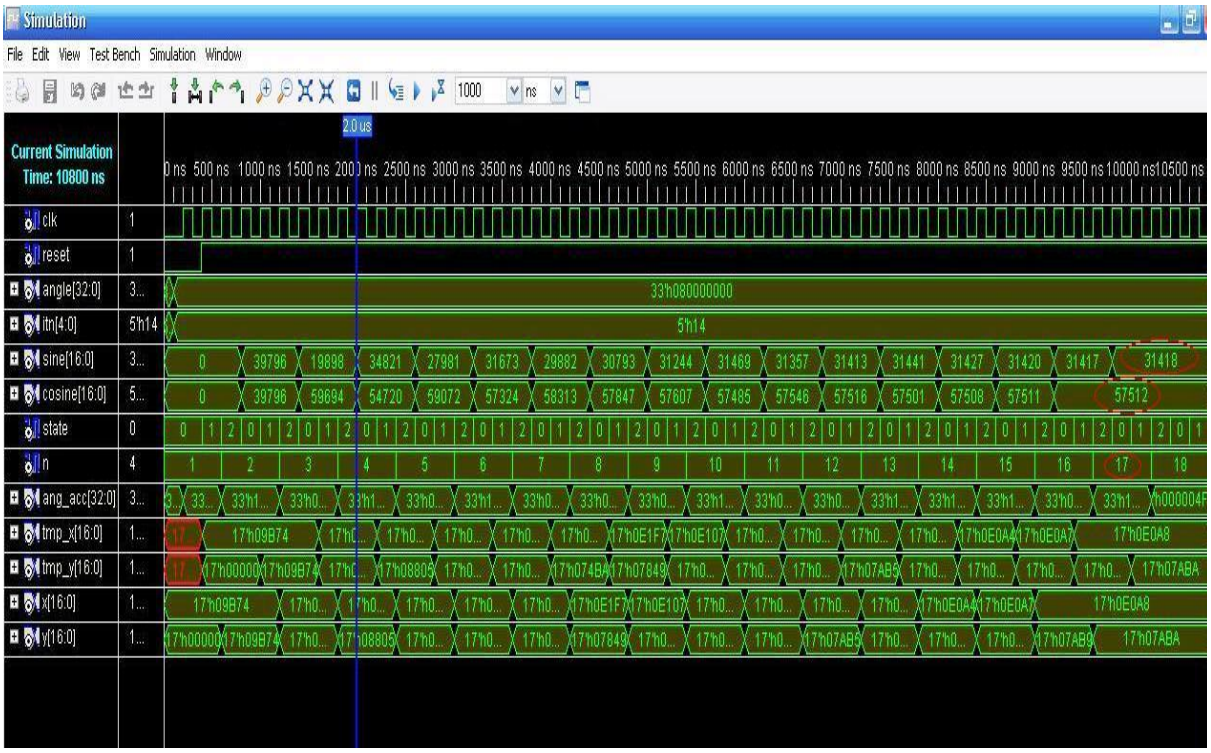


Fig.4.2: Test-Bench Waveforms obtained after the CORDIC Core Simulation

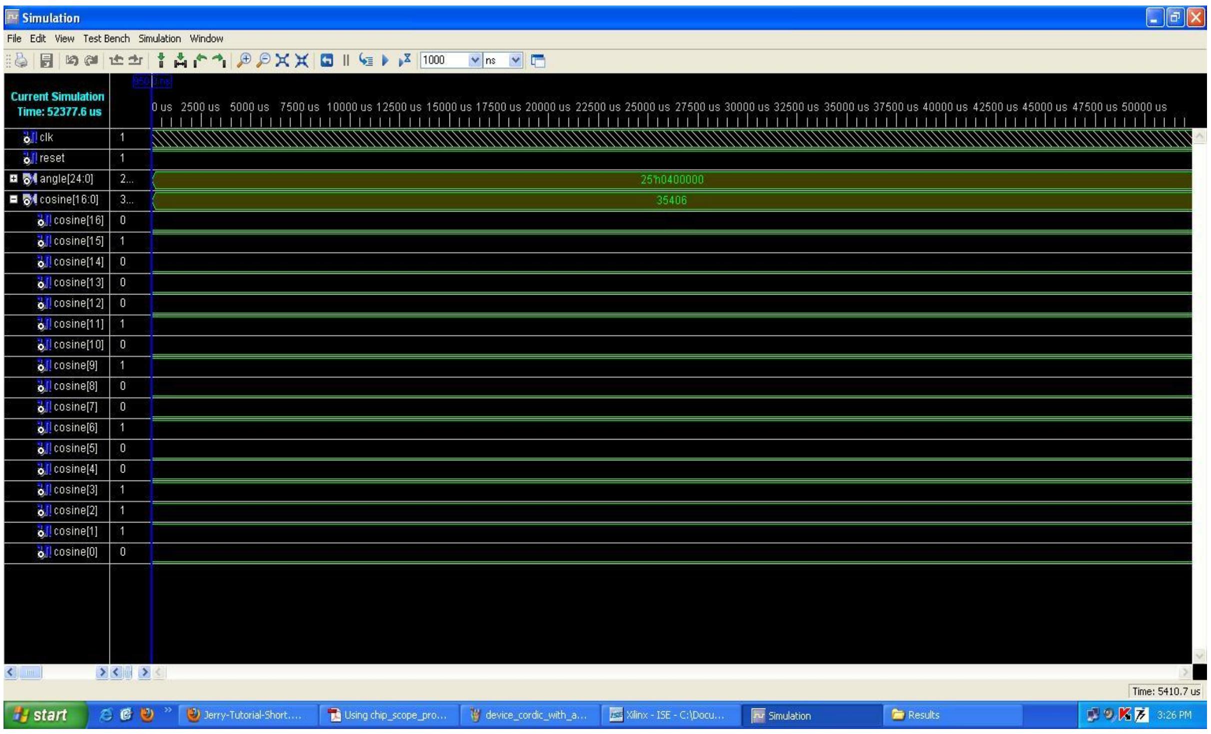
The CORDIC algorithm was implemented in Matlab as well as VHDL. The angle vector rotation values during each iteration was observed in both the cases. The following table shows the comparison between the corresponding rotation values of the ones obtained from Matlab analysis and the ones obtained from Test-Bench analysis in VHDL.

Table 4.1: Comparison of Successive Angle Rotation Values

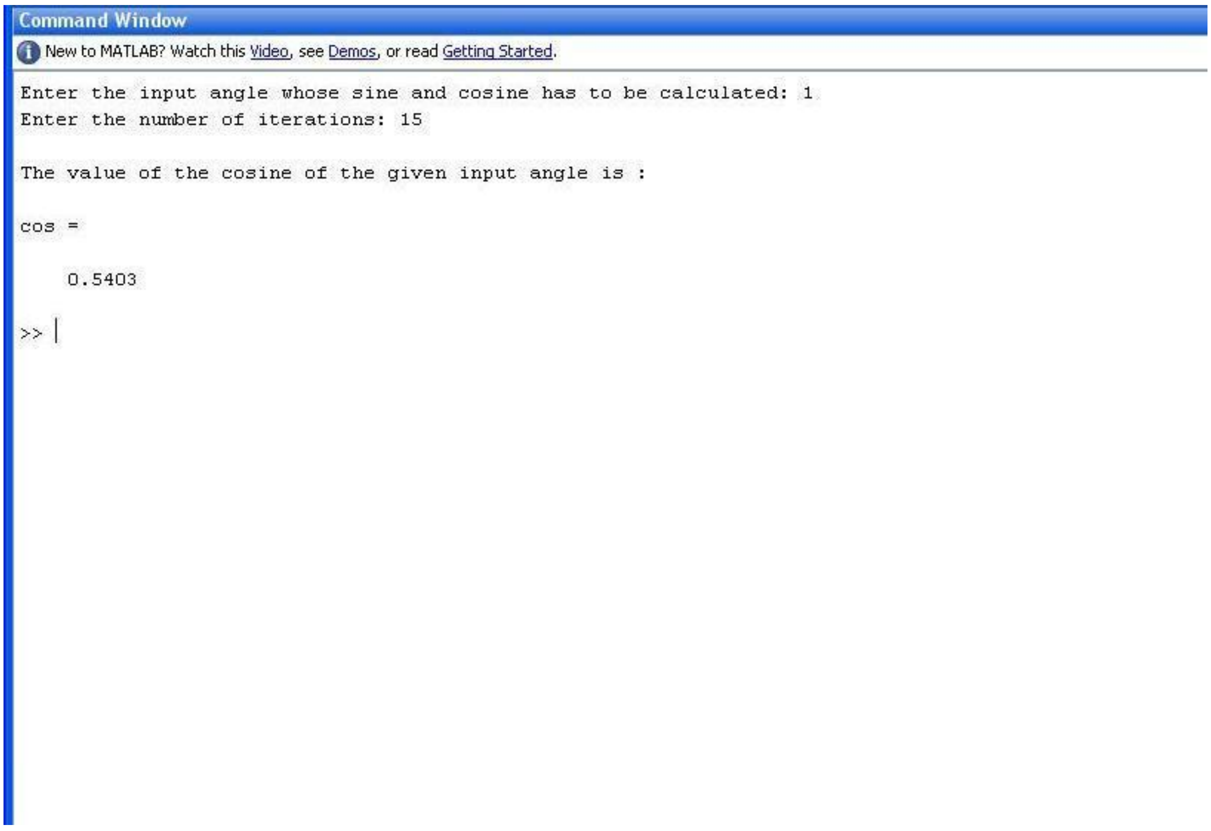
Iteration	***From Matlab***	***From VHDL Test-bench***
1	*-0.2854*	*1B6F0255E (-0.28539)*
2	*+0.1782*	*02DA1C173 (0.17824)*
3	*-0.0667*	*1EEEAD581 (-0.06672)*
4	*+0.0576*	*00EC0901B (0.05762)*
5	*-0.0048*	*1FEC5E240 (-0.00479)*
6	*+0.0264*	*006C537AE (0.02645)*
7	*+0.0108*	*002C54D03 (0.01082)*
15	*2.002 * E-5*	*000014FF9 (2.002E-5)*
16	*-1.0492 * E-5*	*1FFFF4FFA (-1.04E-5)*

Angle=1rad

Cos=35406=0.5402


Simulation
File Edit View Test Bench Simulation Window
Current Simulation Time: 52377.6 us
0 us 2500 us 5000 us 7500 us 10000 us 12500 us 15000 us 17500 us 20000 us 22500 us 25000 us 27500 us 30000 us 32500 us 35000 us 37500 us 40000 us 42500 us 45000 us 47500 us 50000 us
Time: 5410.7 us
start
Simulation
Results
3:26 PM

```
clk            1
reset          1
angle[24:0]    2...   25'h0400000
cosine[16:0]   3...   35406
cosine[16]     0
cosine[15]     1
cosine[14]     0
cosine[13]     0
cosine[12]     0
cosine[11]     1
cosine[10]     0
cosine[9]      1
cosine[8]      0
cosine[7]      0
cosine[6]      1
cosine[5]      0
cosine[4]      0
cosine[3]      1
cosine[2]      1
cosine[1]      1
cosine[0]      0
```


Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
Enter the input angle whose sine and cosine has to be calculated: 1
Enter the number of iterations: 15

The value of the cosine of the given input angle is :

cos =

    0.5403

>>
```

Fig.4.3: Comparison of Matlab and VHDL output values after CORDIC Core Simulation

The following figure shows the successful uploading and execution of the Xilinx ISE generated bit-file of the CORDIC processor to get the output in figure.

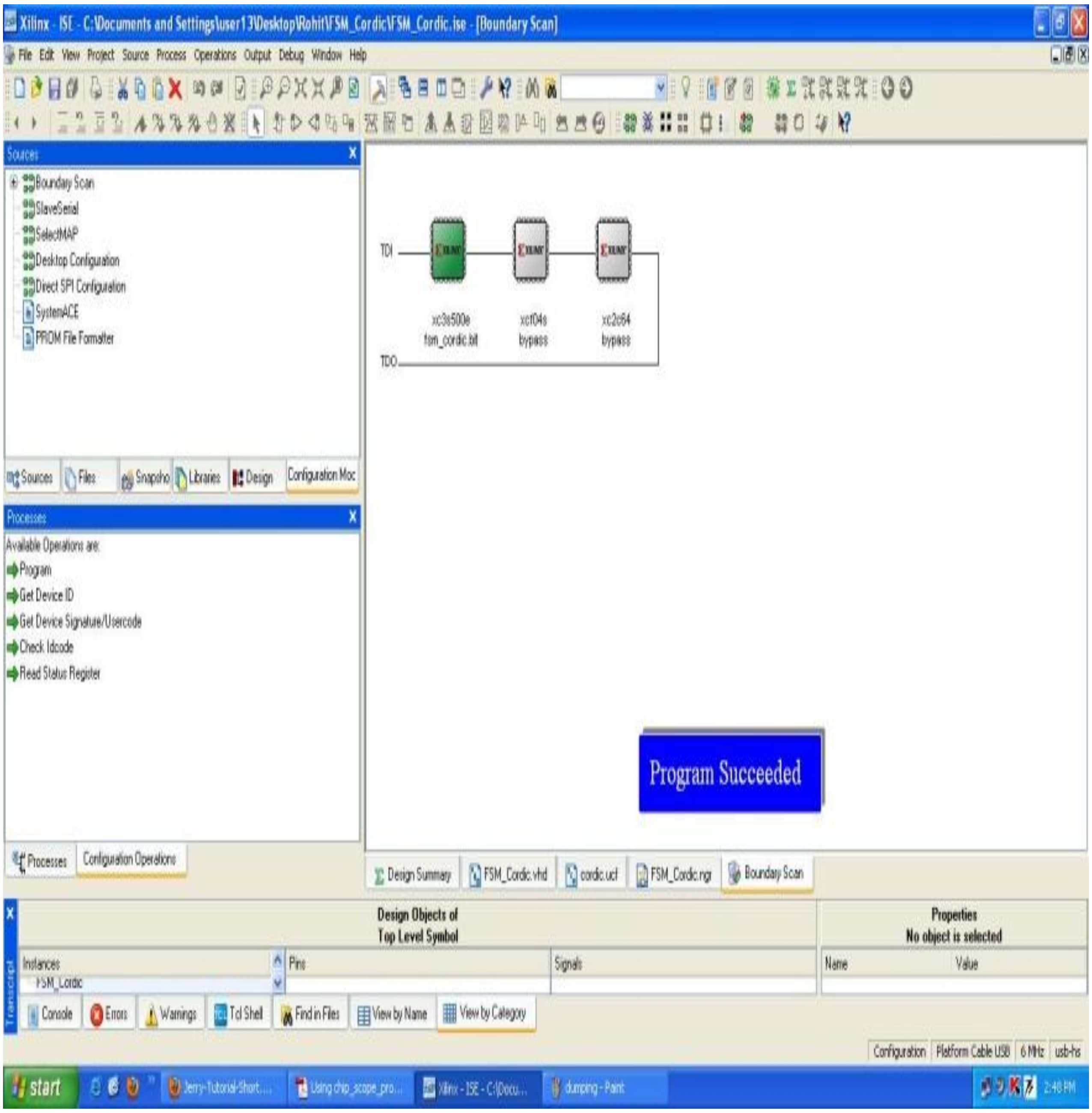


Fig.4.4: Uploading the CORDIC bit file onto FPGA

The following figure shows the Spartan 3E FPGA implementation of the CORDIC processor. The working of the design is tested by analysing only the 8 MSBs of the output at the 8 LED outputs of the FPGA. The input given are CLK, reset and angle whose Cosine value is to be calculated.

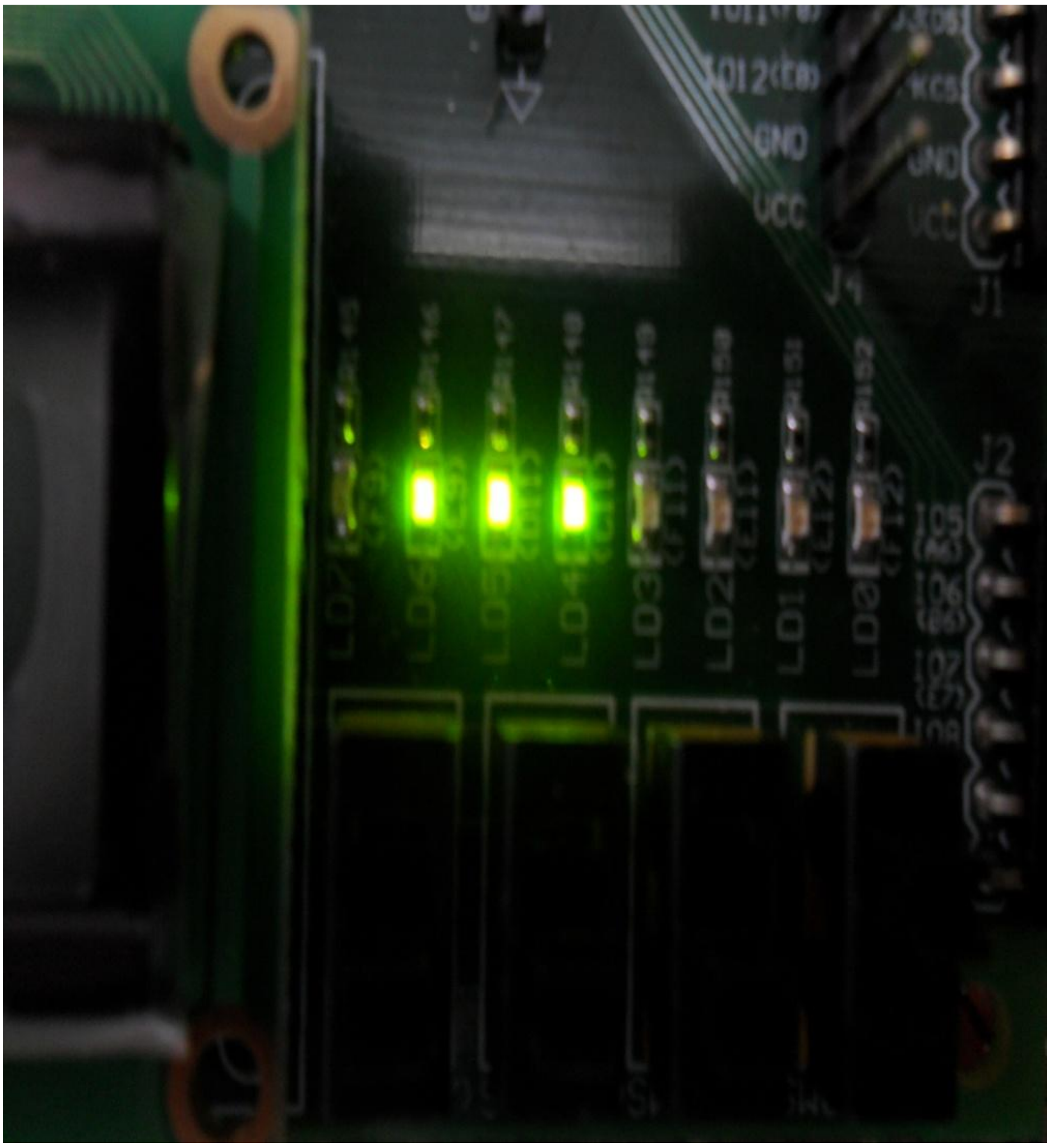
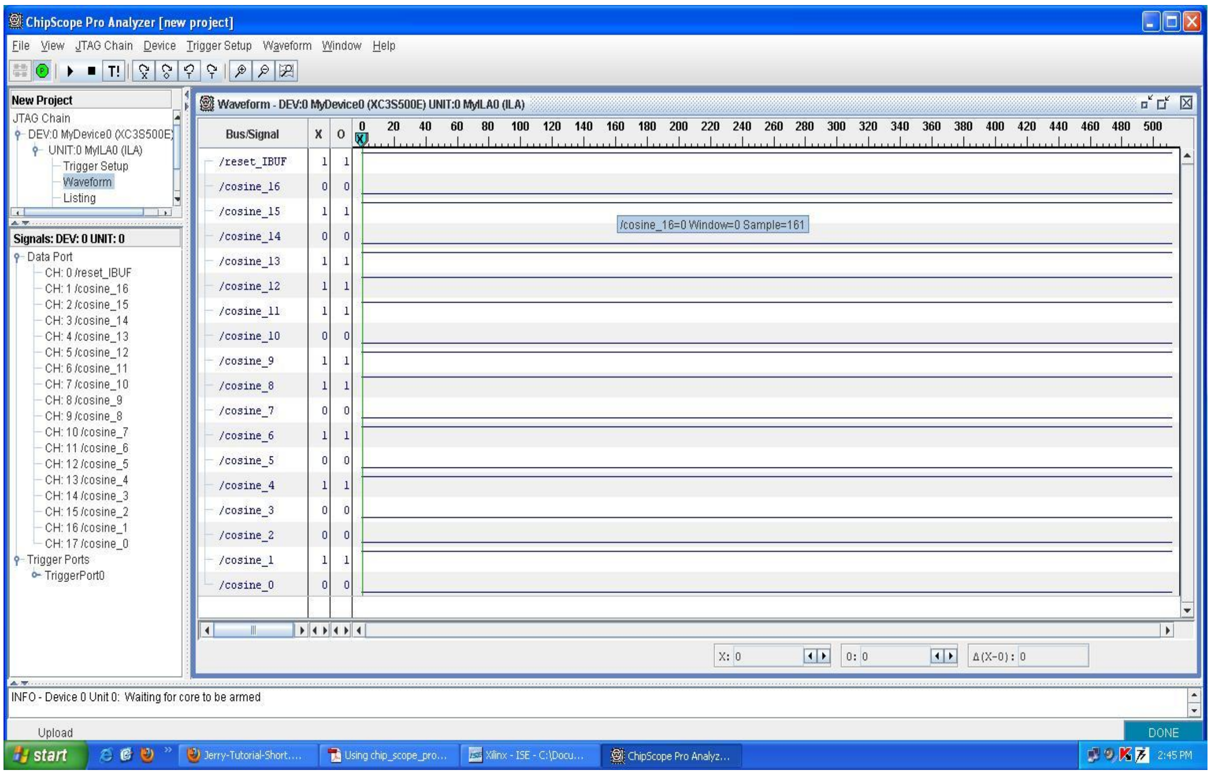


Fig. 4.5: FPGA Implementation

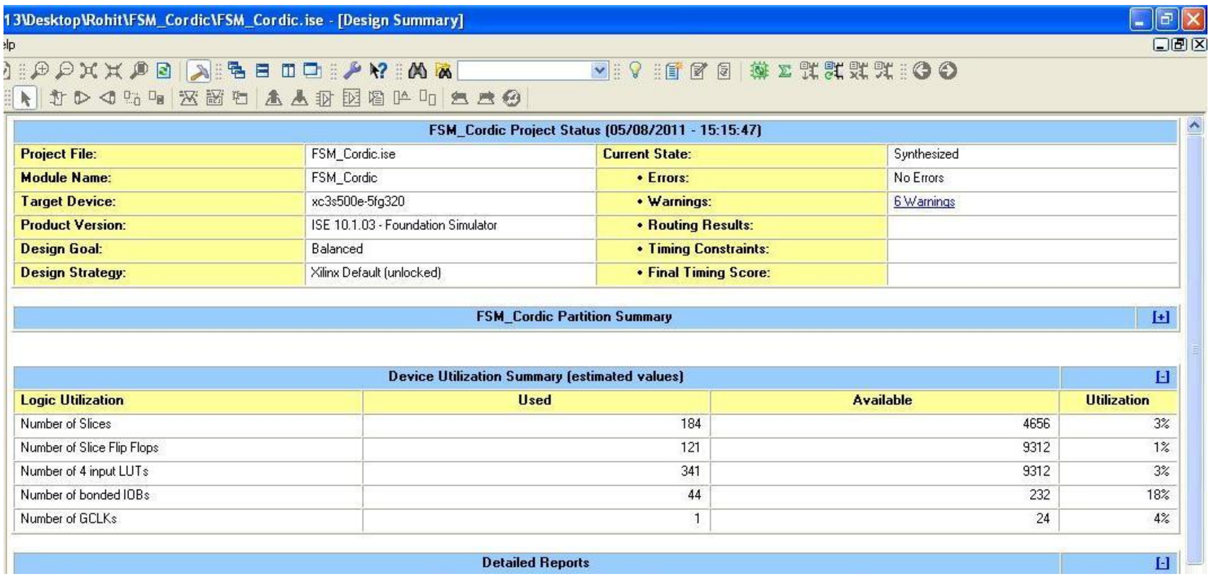
Spartan 3E FPGA implementation has the limitation of only 8-bit output. The cosine value output being generated has 17 bits. Therefore it is not possible to observe the all the 17-bits on the Spartan 3E FPGA Kit. Thus, the ChipScopePro analysis of the implemented design is carried out. ChipScopePro provides us with the provision of observing all the 17-bits of output. Using this we can also observe the intermediate signals during the deisgn implementation.


ChipScope Pro Analyzer [new project]
File View JTAG Chain Device Trigger Setup Waveform Window Help
Waveform - DEV:0 MyDevice0 (XC3S500E) UNIT:0 MyILA0 (ILA)
/cosine_16=0 Window=0 Sample=161
start

Bus/Signal	X	O
/reset_IBUF	1	1
/cosine_16	0	0
/cosine_15	1	1
/cosine_14	0	0
/cosine_13	1	1
/cosine_12	1	1
/cosine_11	1	1
/cosine_10	0	0
/cosine_9	1	1
/cosine_8	1	1
/cosine_7	0	0
/cosine_6	1	1
/cosine_5	0	0
/cosine_4	1	1
/cosine_3	0	0
/cosine_2	0	0
/cosine_1	1	1
/cosine_0	0	0

```
INFO - Device 0 Unit 0:  Waiting for core to be armed
```

Fig.4.6: ChipScopePro Analysis of Output waveform


13\Desktop\Rohit\FSM_Cordic\FSM_Cordic.ise - [Design Summary]

FSM_Cordic Project Status (05/08/2011 - 15:15:47)			
Project File:	FSM_Cordic.ise	**Current State:**	Synthesized
Module Name:	FSM_Cordic	• **Errors:**	No Errors
Target Device:	xc3s500e-5fg320	• **Warnings:**	6 Warnings
Product Version:	ISE 10.1.03 - Foundation Simulator	• **Routing Results:**	
Design Goal:	Balanced	• **Timing Constraints:**	
Design Strategy:	Xilinx Default (unlocked)	• **Final Timing Score:**	

FSM_Cordic Partition Summary [+]

Device Utilization Summary (estimated values) [-]

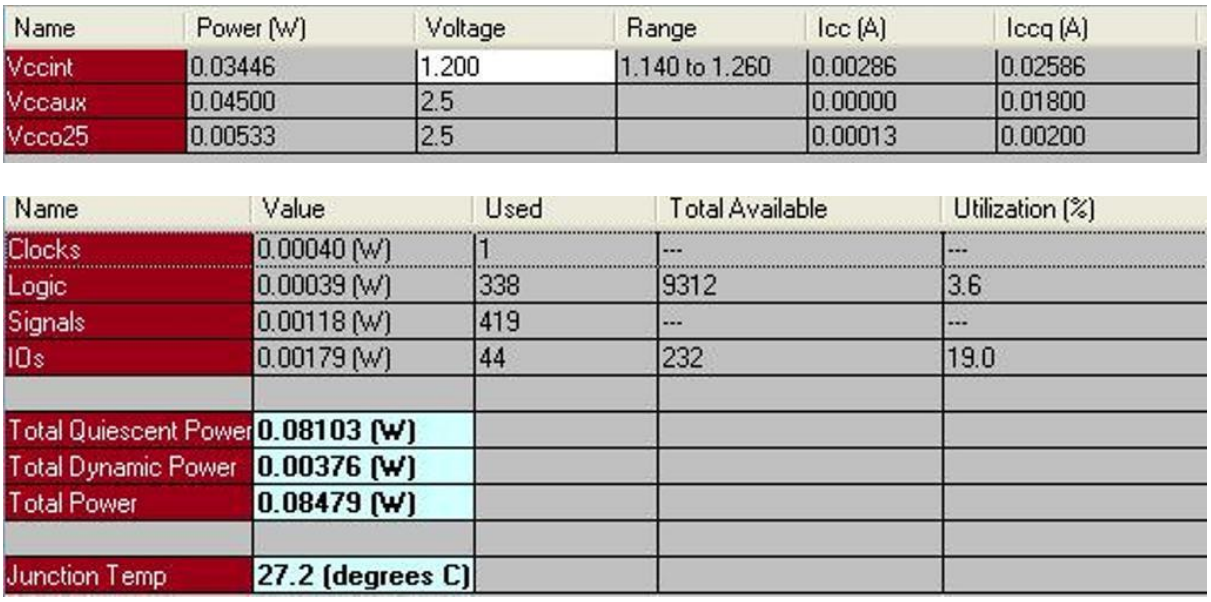
Logic Utilization	Used	Available	Utilization
Number of Slices	184	4656	3%
Number of Slice Flip Flops	121	9312	1%
Number of 4 input LUTs	341	9312	3%
Number of bonded IOBs	44	232	18%
Number of GCLKs	1	24	4%

Detailed Reports [-]

Fig. 4.7: Device Utilization Summary for CORDIC Processor

Table 4.2: Power Analysis Results for the CORDIC Core

Total power dissipated is 0.08479W.



Name	Power (W)	Voltage	Range	Icc (A)	Iccq (A)
Vccint	0.03446	1.200	1.140 to 1.260	0.00286	0.02586
Vccaux	0.04500	2.5		0.00000	0.01800
Vcco25	0.00533	2.5		0.00013	0.00200

Name	Value	Used	Total Available	Utilization (%)
Clocks	0.00040 (W)	1	---	---
Logic	0.00039 (W)	338	9312	3.6
Signals	0.00118 (W)	419	---	---
IOs	0.00179 (W)	44	232	19.0
Total Quiescent Power	**0.08103 (W)**			
Total Dynamic Power	**0.00376 (W)**			
Total Power	**0.08479 (W)**			
Junction Temp	**27.2 (degrees C)**			

The following figure shows the RTL schematic of a DCT processor. The DCT processor has a CORDIC Core inside it. As clear from the schematic it has eight 8 bit inputs and eight 17 bit outputs. Both the input and the output are in signed integer format. Along with these I/Os it has a clock and reset input to control when the output appears at the output pins.

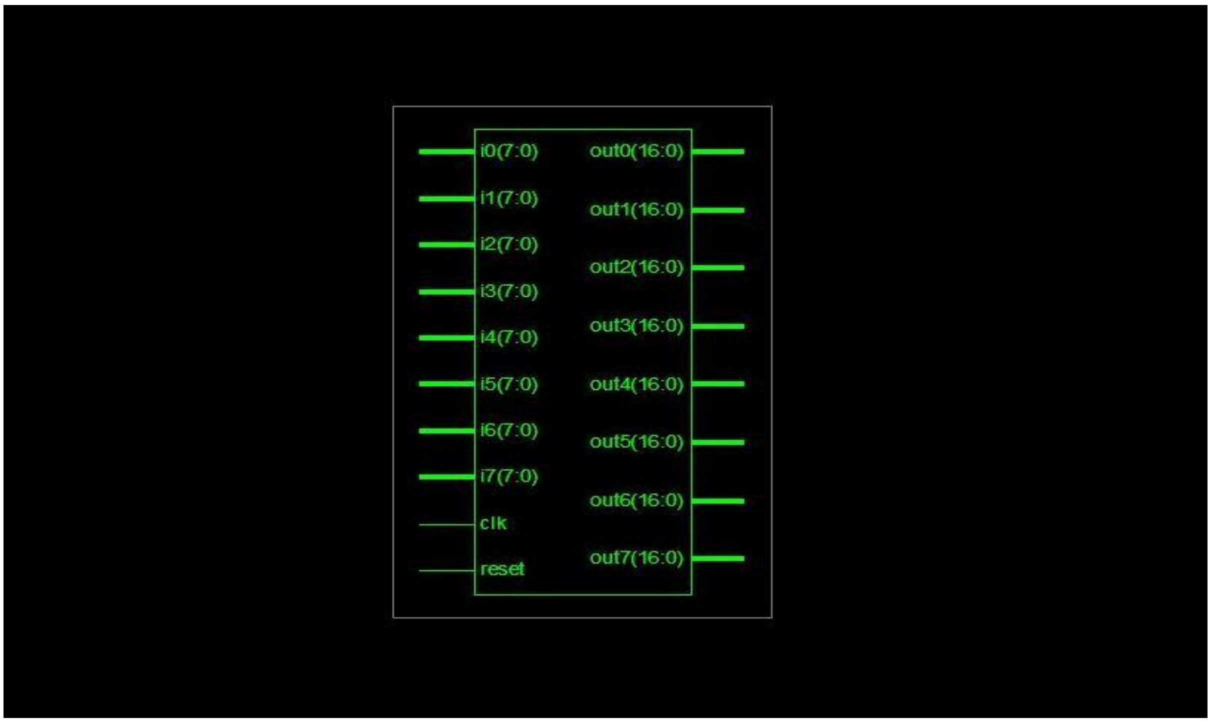
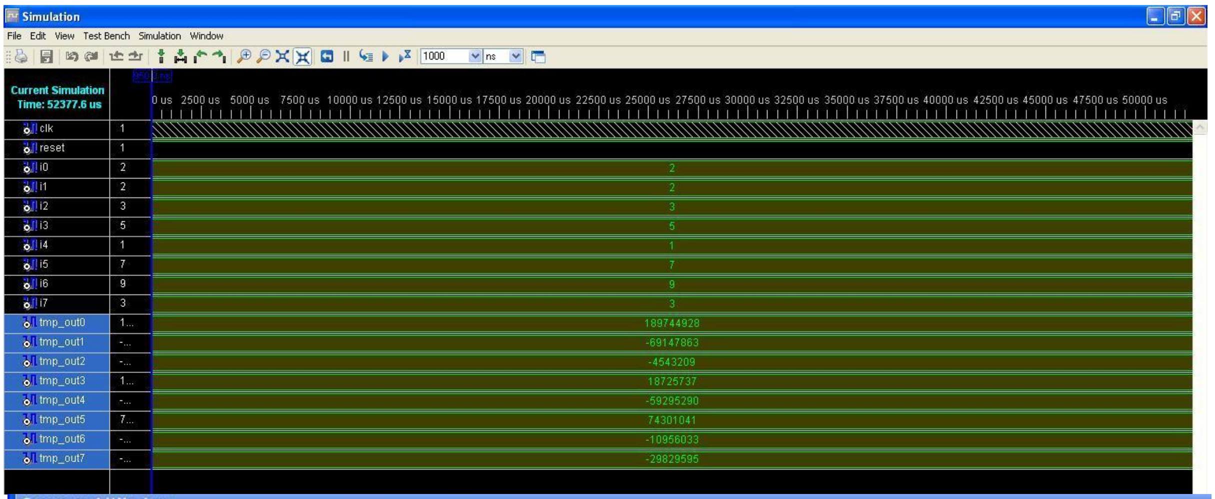
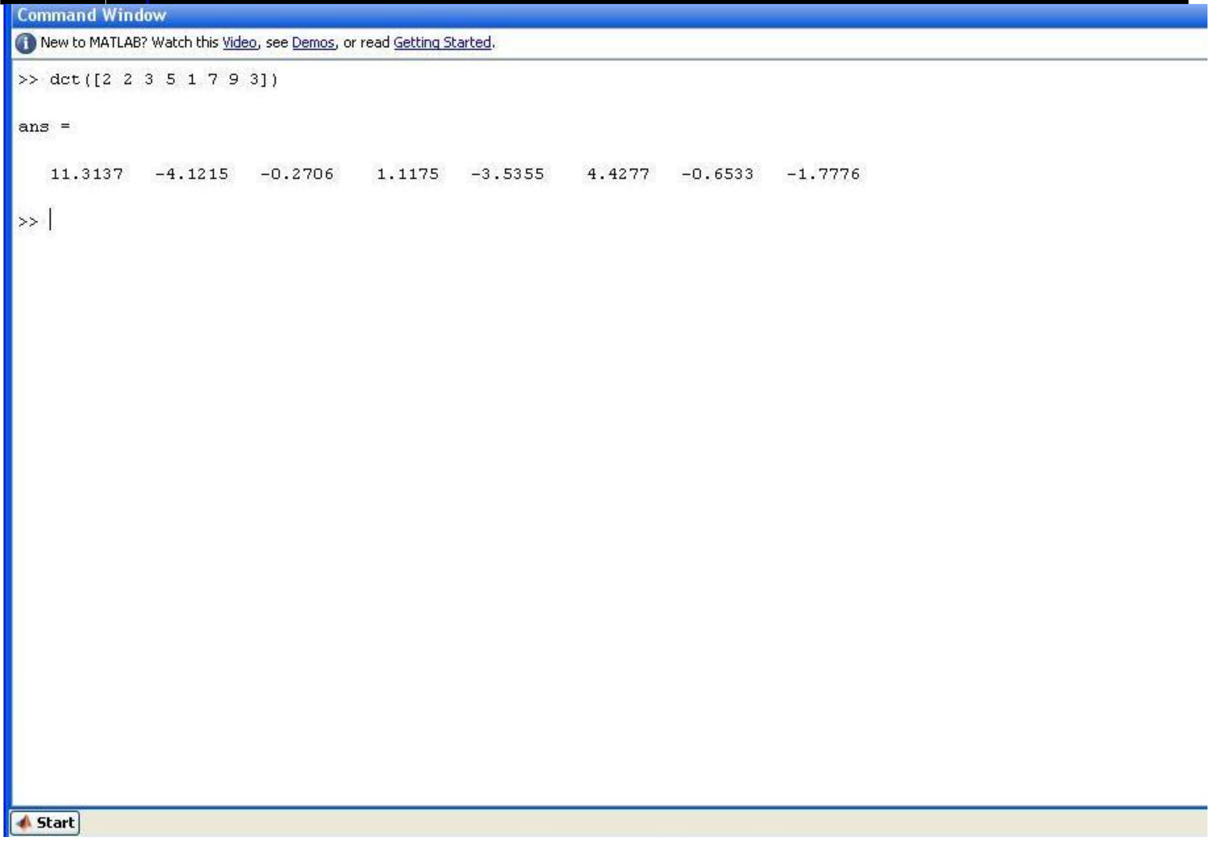


Fig. 4.8: RTL Schematic of Discrete Cosine Transform Processor

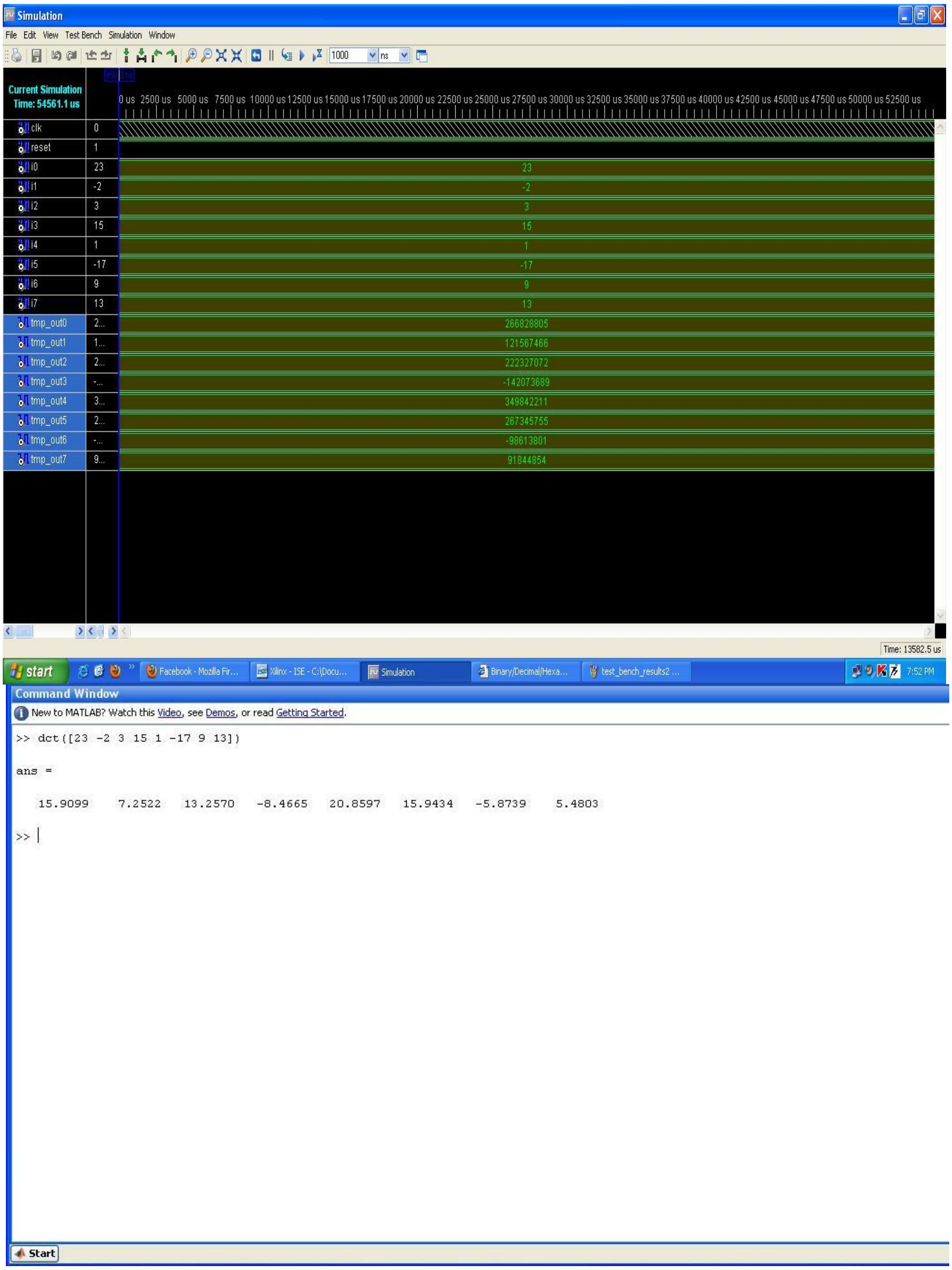

Simulation
File Edit View Test Bench Simulation Window
1000 ns
Current Simulation Time: 52377.6 us
0 us 2500 us 5000 us 7500 us 10000 us 12500 us 15000 us 17500 us 20000 us 22500 us 25000 us 27500 us 30000 us 32500 us 35000 us 37500 us 40000 us 42500 us 45000 us 47500 us 50000 us

```
clk        1
reset      1
i0         2      2
i1         2      2
i2         3      3
i3         5      5
i4         1      1
i5         7      7
i6         9      9
i7         3      3
tmp_out0   1...   189744928
tmp_out1   -...   -69147863
tmp_out2   -...   -4543209
tmp_out3   1...   18725737
tmp_out4   -...   -59295290
tmp_out5   7...   74301041
tmp_out6   -...   -10956033
tmp_out7   -...   -29829595
```


Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
Start

```
>> dct([2 2 3 5 1 7 9 3])

ans =

   11.3137   -4.1215   -0.2706    1.1175   -3.5355    4.4277   -0.6533   -1.7776

>> 
```

Fig.4.9: Comparison of DCT values obtained from Matlab and VHDL


```
Current Simulation
Time: 54561.1 us

clk        0
reset      1
i0         23          23
i1         -2          -2
i2         3           3
i3         15          15
i4         1           1
i5         -17         -17
i6         9           9
i7         13          13
tmp_out0   2...        266828805
tmp_out1   1...        121567466
tmp_out2   2...        222327072
tmp_out3   -...        -142073689
tmp_out4   3...        349842211
tmp_out5   2...        267345755
tmp_out6   -...        -98613801
tmp_out7   9...        91844854

Time: 13582.5 us
```

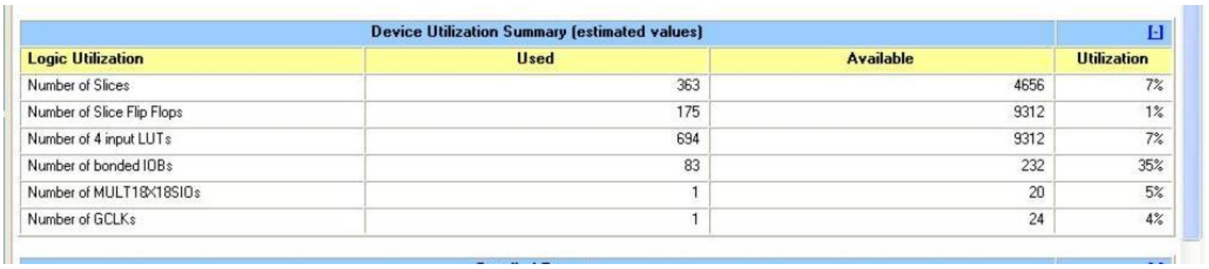
```
>> dct([23 -2 3 15 1 -17 9 13])

ans =

   15.9099    7.2522   13.2570   -8.4665   20.8597   15.9434   -5.8739    5.4803

>> |
```

Fig.4.10: Comparison of DCT values obtained from Matlab and VHDL

The following table expresses the data o/p observed in the above two figures in a tabular format. The DCT values observed from the use of dct command in Matlab and from the scaling of the Output observed on the Test-Bench are nearly equal. The error is less than 0.5%. This error arises primarily due to the rounding up of the values obtained from test-bench as well as the bit-size of the input angle signal as well as the number of CORDIC iterations. In this project the no. of CORDIC iterations were limited to 15.

Table 4.3: Table showing the comparison of Matlab and VHDL DCT outputs

INPUT	[23 -2 3 15 1 -17 9 13]	[2 2 3 5 1 7 9 3]
Matlab O/P	[15.9099 7.2522 13.2570 -8.4665 20.8597 15.9434 -5.8739 5.4803]	[11.3137 -4.1215 -0.2706 1.1175 -3.5355 4.4277 -0.6533 -1.7776]
VHDL O/P	[15.904 7.246 13.250 -8.468 20.85 15.935 -5.878 5.474]	[11.309 -4.122 -0.271 1.116 -3.534 4.429 -0.653 -1.778]

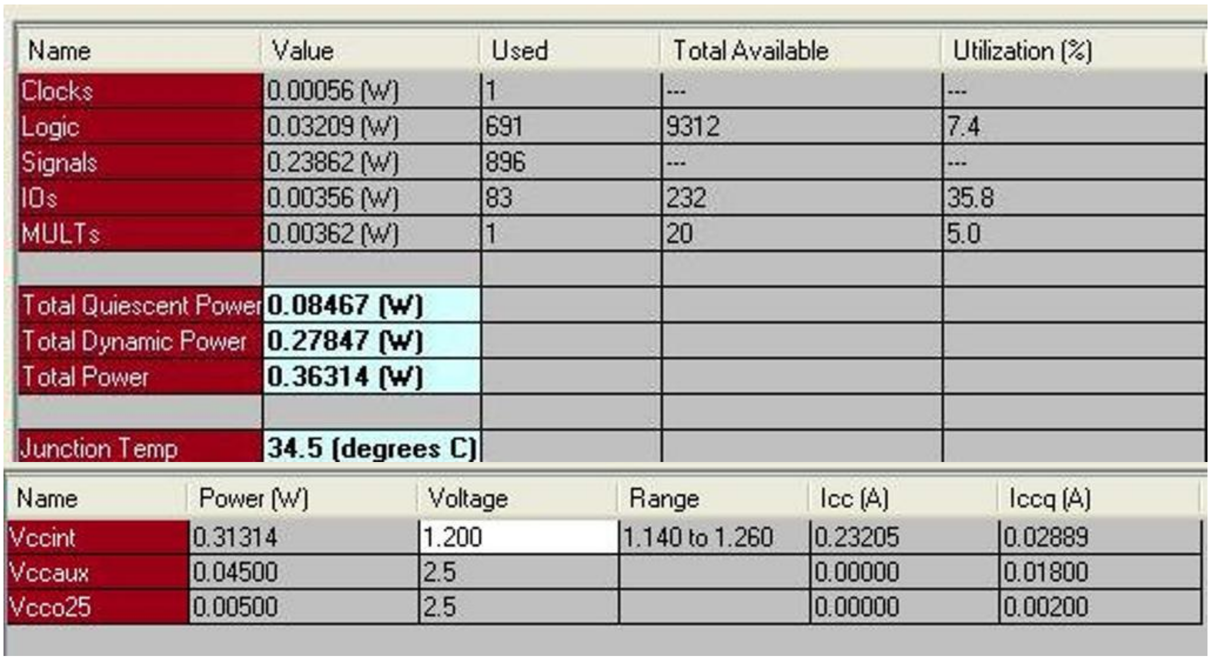


Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	363	4656	7%
Number of Slice Flip Flops	175	9312	1%
Number of 4 input LUTs	694	9312	7%
Number of bonded IOBs	83	232	35%
Number of MULT18X18SIOs	1	20	5%
Number of GCLKs	1	24	4%

Fig.4.11: Device Utilization Summary of DCT processor

Table 4.4: Power Analysis Results for the DCT Core

Total Power dissipated is 0.36314W.



Name	Value	Used	Total Available	Utilization (%)
Clocks	0.00056 (W)	1	---	---
Logic	0.03209 (W)	691	9312	7.4
Signals	0.23862 (W)	896	---	---
IOs	0.00356 (W)	83	232	35.8
MULTs	0.00362 (W)	1	20	5.0
Total Quiescent Power	**0.08467 (W)**			
Total Dynamic Power	**0.27847 (W)**			
Total Power	**0.36314 (W)**			
Junction Temp	**34.5 (degrees C)**			

Name	Power (W)	Voltage	Range	Icc (A)	Iccq (A)
Vccint	0.31314	1.200	1.140 to 1.260	0.23205	0.02889
Vccaux	0.04500	2.5		0.00000	0.01800
Vcco25	0.00500	2.5		0.00000	0.00200

Chapter 6

Conclusion and Future Work

CORDIC is a powerful algorithm, and a popular algorithm of choice when it comes to various Digital Signal Processing applications. Implementation of a CORDIC-based processor on FPGA gives us a powerful mechanism of implementing complex computations on a platform that provides a lot of resources and flexibility at a relatively lesser cost. Further, since the algorithm is simple and efficient the design and VLSI implementation of a CORDIC based processor is easily achievable.

In this project a CORDIC module is designed and simulated using Xilinx ISE using VHDL as a synthesis tool. The output of the CORDIC core is analyzed and verified on the test-bench, and compared with the actual values obtained from Matlab. This module is subsequently used for the design and simulation of 8-point 1D DCT processor. Similar analysis was performed for the DCT processor.

Finally the DCT processor was implemented on a Spartan 3E FPGA kit. The output logic waveforms were analyzed using ChipScopePro. The output values were found to be consistent with the actual values. The device utilization summary showed that minimum resources were consumed. The power analysis showed that very less power was consumed during the operation. Thus, the above processor can be used for the online calculation of 1D 8-point DCT values.

Future Scope

Although this project primarily deals with the design of 8-point 1D DCT using CORDIC algorithm, the concept and the architecture can be extended to calculate the 8-point 2D DCT. It can be further extended to calculate the higher order DCTs, thus, providing a fast, low-cost implementation of processors for Image Processing and other Digital Signal Processing Applications. The performance of this DCT processor based on CORDIC algorithm can be compared with that of a DCT processor designed using distributed arithmetic.

References

[1]Ray Andraka, FPGA '98. Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, Feb. 22-24, 1998, Monterey, CA. pp191-200

[2]Vikas Sharma,"FPGA Implementation of EEAS CORDIC based Sine and Cosine Generator",M.Tech Thesis,Dept.Electron.Comm.Eng.,Thapar Uni., Patiala, 2009.

[3]Satyasen Panda,"Performance Analysis and Design of a Discrete Cosine Transform Processor using CORDIC Algorithm",M.Tech Thesis,Dept. Electron.Comm.Eng.,NIT Rourkela, Rourkela, Orissa,2010.

[4]Kris Raj, Gaurav Doshi, Hiren Shah [online] Available: http://teal.gmu.edu/courses/ECE645/projects_S06/talks/CORDIC.pdf(URL)

[5] S.K.Pattanaik and K.K.Mahapatra,'DHT Based JPEG Image Compression Using a Novel Energy Quantization Method", IEEE International Conference on Industrial Technology,pp.2827-2832,Dec 2006.

[6] Volder, J.,"Binary Computation Algorithms for coordinate rotation and function generation,"Convair Report IAR-1 148 Aeroelectrics Group, June 1956.

[7] Volder, J.,"The CORDIC Trigonometric Computing Technique,"IRE Trans.Electronic Computing, Vol EC-8, pp330-334 Sept 1959.

[8] Walther J.S.,"A unified algorithm for elementary functions," Spring Joint Computer Conf.,1971,proc.,pp.379-385.

[9] Ahmed,H.M.,Delosme, J.M.,and Morf,M.,"Highly Concurrent Computing Structure for Matrix Arithmetic and Signal Processing," IEEE Comput. Mag,,Vol.15,1982,pp.65-82.

[10] Depreterre, E.,Dewilde, P.,and Udo, R.,'Pipelined CORDIC Architecture for Fast VLSI Filtering and Array Processing," Proc.ICASSP'84, 1984,pp.41.A.6.1-41.A.6.4.

[11] Duprat, J. and Muller, J.M.,"The CORDIC Algorithm: New Results for Fast VLSI Implementation," IEEE Transactions on Computers, Vol.42,pp.168-178,1993.

[12] Hu,Y.H.,and Naganathan, S.,"A Novel Implementation of Chirp Z-Transformation Using a CORDIC processor,"IEEE Transactions on ASSP, Vol.38, pp.352-354,1990.

[13]Hu,Y.H.,and Naganathan, S.,"An Angle Recoding Method for CORDIC Algorithm Implementation", IEEE Transactions on Computers, Vol.42, pp.99-102, January 1993.

[14]Sibul,L.H. and Fogelsanger,A.L.,"Application of Coordinate Rotation Algorithm to Singular Value Decomposition," IEEE Int. Symp.Circuits and Systems,pp.821-824,1984.

[15]Andraka.R.J.,"Building a High Performance Bit-Serial Processor in an FPGA," Proceedings of Design SuperCon '96,Jan 1996,pp.5.1-5.2.

[16]Stephen Brown and Jonathan Rose,"Architecture of FPGAs and CPLDS:A Tutorial", [online] Available: httpwww.eecg.toronto.edu~jayarpubsbrownsurvey.pdf(URL)

[17]Summanasena M.G.B "A Scale Factor Correction Scheme for CORDIC Algorithm" IEEE, August, 2008.