Effect of various CORDIC algorithms on DFT implementation

A Project Report

submitted by

PRANIT J. THAKUR

in partial fulfilment of the requirements

for the award of the degree of

DUAL DEGREE IN ELECTRICAL ENGINEERING(B.TECH AND M.TECH)

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Effect of various CORDIC algorithms on DFT implementation**, submitted by **Pranit J. Thakur**, to the Indian Institute of Technology, Madras, for the award of the degree of **Dual degree**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr.K.SRIDHARAN
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 12th May 2019

ACKNOWLEDGEMENTS

I would like to thank Prof.Dr.K.SRIDHARAN for being an excellent mentor and a teacher. This project has given me a chance to experience a significant side of academic research. I am sure it will be helpful in my further endeavours.

I would like to thank IIT Madras for giving this great opportunity. It was a good learning experience to have worked on this project.

Finally, I would like to thank my family and friends who have been supportive all along

ABSTRACT

KEYWORDS: FPGA; CORDIC; Discrete Fourier Transform.

The DFT is central to digital signal processing. It is one of the most important tools to analyze signals and systems in the digital world and is a corner stone of the language that we use to describe and manipulate signals. The resources for processing and the timing are both a constraint. When the input data becomes to large, there is a need of different methods in order to eliminate the complexity.In this thesis, we have tried to solve this above mentioned problem.

Different methods of CORDIC and DFT are synthesized and implemented in order to get the data on resource utilization. The experimental validation was done using Quartus simulation tool, where the numerical synthesis and the post and route described in Verilog, was realized using ISE Design Suite 14.7.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

LUT	Lookup Table
FPGA	Field Programmable Gate Array
MUX	Multiplexer
FFT	Fast Fourier Transform
CORDIC	COordinate Rotation DIgital Computer
DFT	Discrete Fourier Transform
PLD	Programmable Logic Device
GCLK	Global Clock
ACLK	Asynchronous Clock
BUFG	Global Clock Buffer
FF	Flipflops

CHAPTER 1

INTRODUCTION

1.1 FPGA and LUT's

A field-programmable gate array (FPGA) is an integrated circuit (IC) that can be programmed in the field after manufacture. FPGAs are similar in principle to, but have vastly wider potential application than, programmable read-only memory (PROM) chips. FPGAs are used by engineers in the design of specialized ICs that can later be produced hard-wired in large quantities for distribution to computer manufacturers and end users. Ultimately, FPGAs might allow computer users to tailor microprocessors to meet their own individual needs.

A LUT, which stands for LookUp Table, in general terms is basically a table that determines what the output is for any given input(s). In the context of combinational logic, it is the truth table. This truth table effectively defines how your combinatorial logic behaves.

In other words, whatever behavior you get by interconnecting any number of gates (like AND, NOR, etc.), without feedback paths (to ensure it is state-less), can be implemented by a LUT.

The way FPGAs typically implement combinatorial logic is with LUTs, and when the FPGA gets configured, it just fills in the table output values, which are called the "LUT-Mask", and is physically composed of SRAM bits. So the same physical LUT can implement Y=AB and Y=AB', but the LUT-Mask is different, since the truth table is different. We can also create your own lookup tables. For example, we could build a table for a complex mathematical function, which would work much faster than actually calculating the value by following an algorithm. This table would be stored in RAM or ROM.

LUT is built out of SRAM bits to hold the comfiguration memory(CRAM) LUT mask and a set of multiplexer to select the bits of CRAM that is to drive the output.A k input LUT can implement k input function with 2^kSRAM bits and 2^k to 1 multiplexer:-

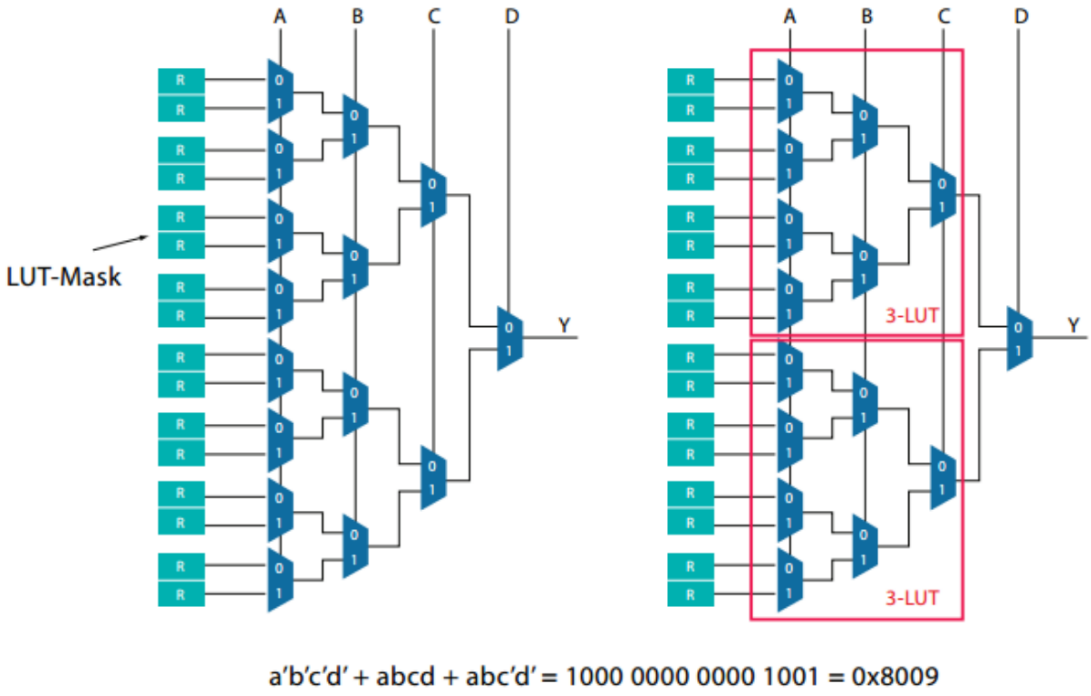

Figure 1.1: 4 Input LUT

1.2 Problem statement

FPGA have a limited amount of LUT's and also the number of inputs to these LUT's is fixed. If the function is small(less no. of input) then that function can easily be implemented on the FPGA but as the number of input increases the function becomes more and more complex to be implemented on FPGA. Here we are implementing CORDIC algorithm in different ways and observing its effect on DFT in terms of resource utilization and timing constraints. Each method can be divided into modules with which we implement them. The difference between them is we are either replacing these module with something else or in some case removing it all together. The effect of which is shown in later chapters.

CHAPTER 2

Literature review

2.1 Cooley Tukey FFT algorithm

The DFT of a finite length sequence of length N:-

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}$$

where k=0 to N-1 and the inverse of DFT is:-

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X[k] e^{j\frac{2\pi}{N}kn}$$

,where n=0 to N-1.

The cooley tukey algorithm exploits the symmetry and periodicity of complex exponential $e^{j\frac{2\pi}{N}kn}$. This can easily be shown for $N = 2^k$.Since N is an even integer, X[k] can be computed by separating x[n] into N/2 point sequences consisting of the even numbered points in x[n] and odd numbered points in x[n].

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}$$

, where k=0 to N-1 and separating x[n] into its even and odd numbered points, we get:-

$$X[k] = \sum_{neven} x[n] W_N^{kn} + \sum_{nodd} x[n] W_N^{kn}$$

Substituting of variables n=2r for n=even and n=2r+1 for n=odd,

$$X[k] = \sum_{r=0}^{N/2-1} x[2r](W_N^2)^{rk} + W_N^k \sum_{r=0}^{N/2-1} x[2r+1](W_N^2)^{rk}$$

and as $W_N^2 = W_{N/2}$,it can be rewritten as

$$X[k] = \sum_{r=0}^{n/2-1} x[2r]W_{N/2}^{rk} + W_N^K \sum_{r=0}^{N/2-1} x[2r+1]W_{N/2}^{rk} \tag{2.1}$$
$$= G[k] + W_N^k H[K]$$

each of the sums in 2.1 is N/2 point DFT,the first sum N/2 DFT of even numbered points of the original sequence and second being N/2 point DFT of odd numbered points od original sequence.G[k] and H[k] are each periodic in k with period N/2.After the DFT are calculated, they are combined to form DFT X[k] of original signal.

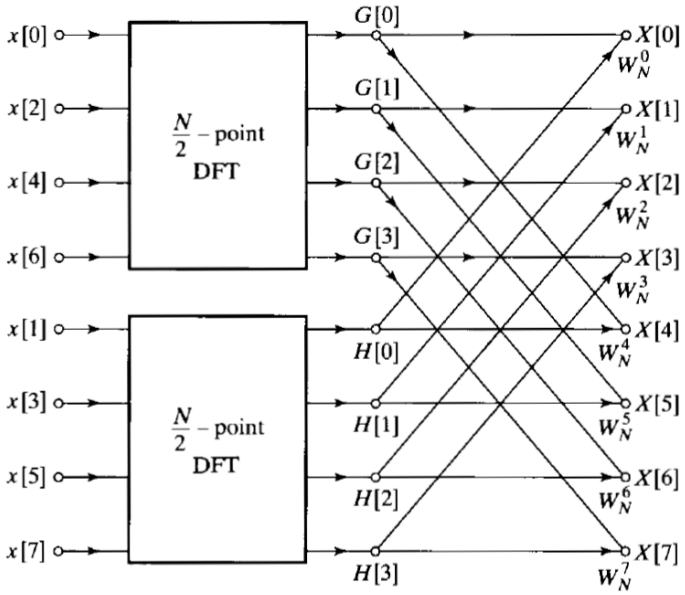

Figure 2.1: Flow graph of the decimation in time decomposition of an N point DFT computation into two N/2 point DFT computation

This N/2 DFT can further be broken down into 2 N/4 DFT in the same way as shown below:-

Similarly the N/4 DFT can be decomposed further leading to just multiplication with constant 1 and -1 (in case N=8).

So at the end we only need N/2 twiddle factor which can be stored in ROM block instead of computing them at each stage and this will reduce execution time by a lot.

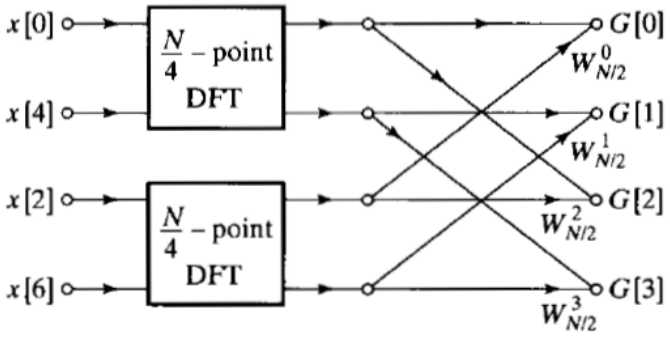

Figure 2.2: Flow graph of the decimation in time decomposition of an N/2 point DFT computation into 2 N/4 point DFT computation

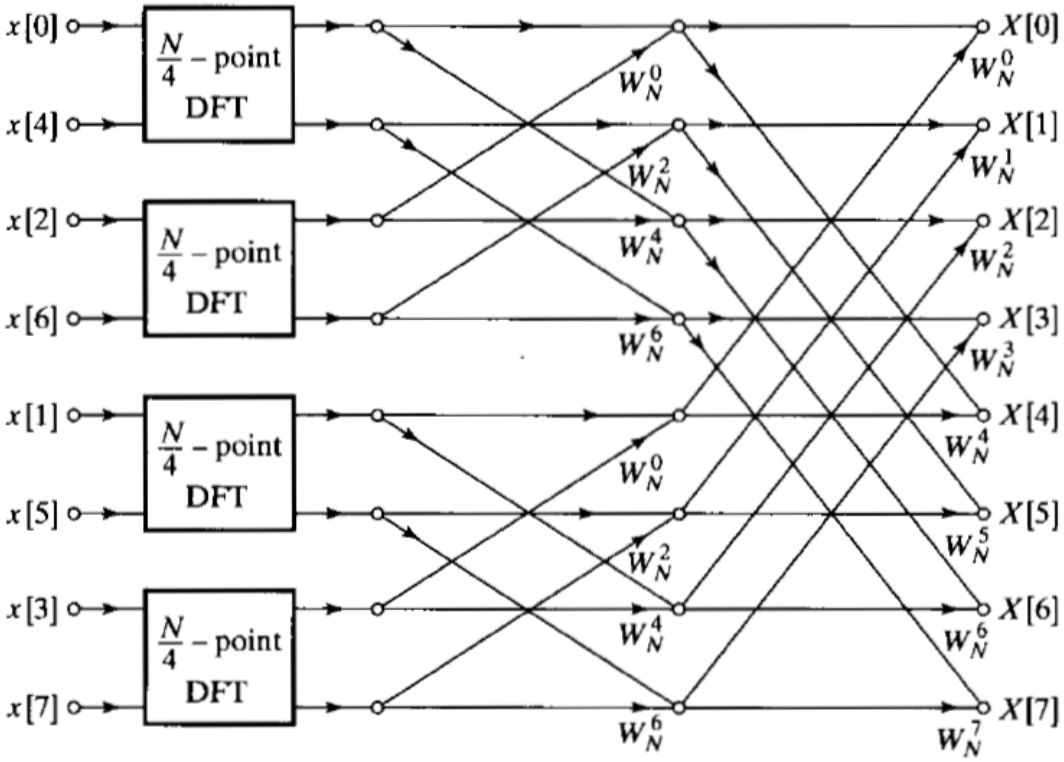

Figure 2.3: Result of substituting structure of figure 2.3 in figure 2.1

2.2 FPGA:-SPARTAN 6

We are using **XC6SLX150** to implement our code. This section will elaborate the resources this board has and how an algorithm will be analyzed on the basis of utilization of these resources:-

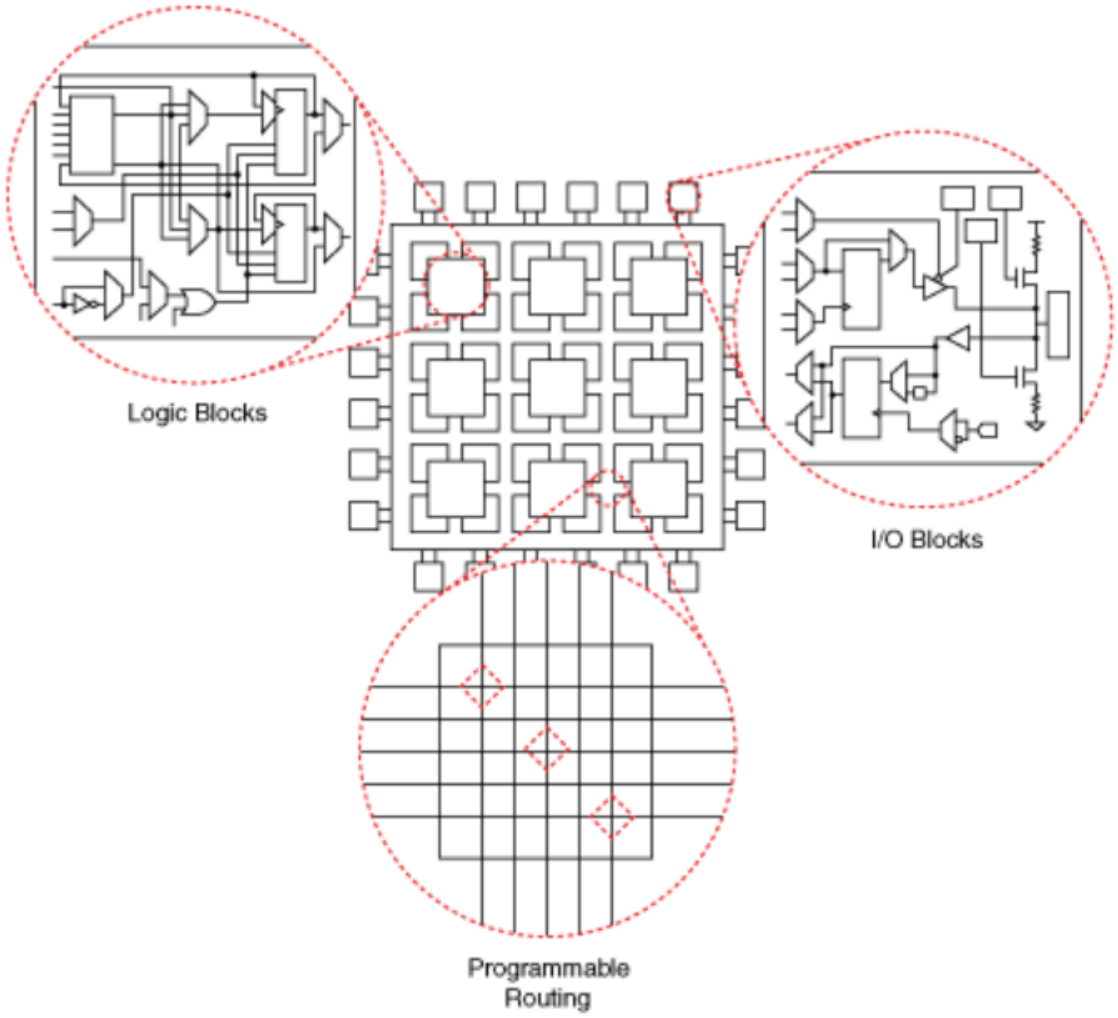

Figure 2.4: A basic FPGA

1. Slices are the basic unit in a FPGA. These are the configurable blocks which are programmed to give the result we want. Each of them contain LUT's, registers, multiplxers, switches (are programmed to make or break a connection within a slice and also outside it and are usually implemented using MOSFET's).

2. Slice register:- As mentioned above,every slice contains a specific numbers of register. SPARTAN XC6SLX150 board has 184304 registers.A register is a group of flip-flops that stores a bit pattern. A register on the FPGA has a clock, input data, output data, and enable signal port. Every clock cycle, the input data is latched, stored internally, and the output data is updated to match the internally stored data. FPGA VIs use registers to perform the following functions:-

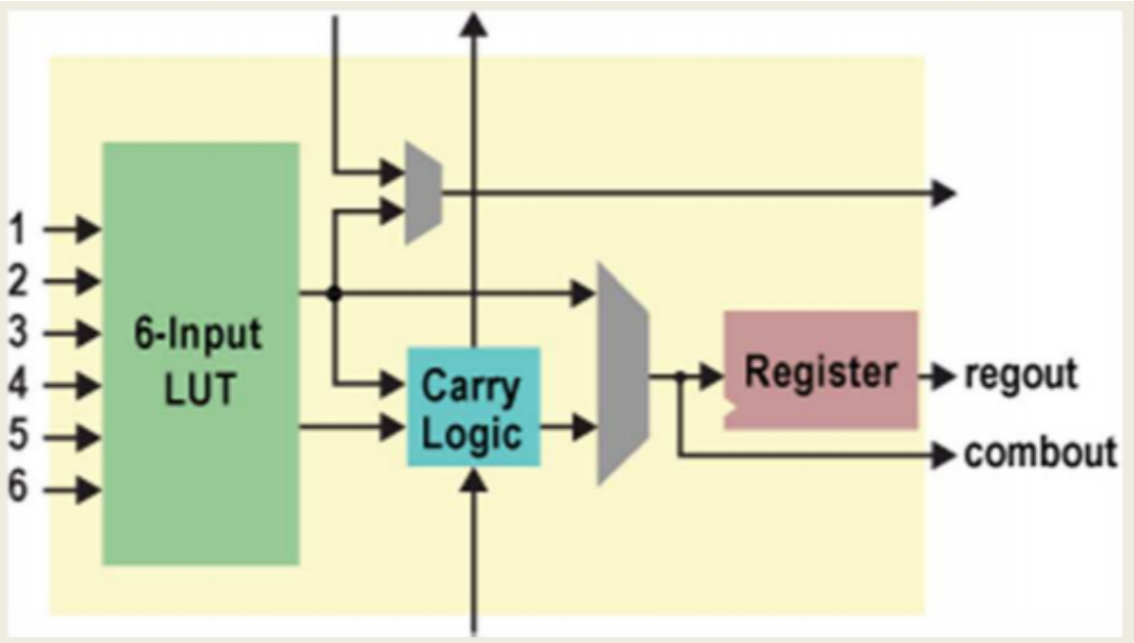

Figure 2.5: A basic slice diagram

(a) Holding state between iterations of a loop

(b) I/O synchronization

(c) Handshaking data between clock domains

(d) Pipelining

3. Slice LUT:-This board has 92152 LUTs.LUTs store a predefined list of outputs for every combination of inputs and provide a fast way to retrieve the output of a logic operation. LUTs have constant delay, regardless of implemented function (LUTs delay can be small but routing delay can be large).

4. LUT FF pair:-This tells us how much, For every LUT in the design you have also used the adjacent flip-flop within that slice. Normally designs will have some logic that uses only the LUT of the pair, for example when you have multiple logic levels between registers. It can also use only the flip-flop of the pair, for example when you have multiple pipeline stages with no combinatorial logic in between.

5. Bonded IOBs:-An IOB is an Input/Output Buffer. In effect it is the number of pins you are using on your device.

6. DSP48A1:-This is the digital signal processing element in the FPGA.The DSP48A1 slices are organized as vertical DSP columns. Within the DSP column, a single DSP slice is combined with extra logic and routing. The vertical column configuration is ideal to connect a DSP slice to its two adjacent neighboring slices, hence cascading those blocks and facilitating the implementation of systolic DSP algorithms.

7. BUFG/BUFGCTRLs:- An architecture-independent global buffer, distributes high fan-out clock signals throughout a PLD device. The Xilinx implementation software converts each BUFG to an appropriate type of global buffer for the target

PLD device.e.g For an XC3000 design, you can use a maximum of two BUFG symbols (assuming that no specific GCLK or ACLK buffer is specified).

CHAPTER 3

DFT with CORDIC

3.1 Overview of CORDIC algorithm

The CORDIC algorithm is an iterative method of performing vector rotations by arbitrary angles using shifts and adds. In the rotation mode, CORDIC may be used for converting a vector in polar form to rectangular form. In the vector mode, it converts a vector in rectangular form to polar form. Both the modes are derived from the general rotation transform.

$$x_{new} = xcos(\theta) - ysin(\theta) \tag{3.1}$$

$$y_{new} = xsin(\theta) + ycos(\theta) \tag{3.2}$$

which rotates a vector in a cartesian plane by angle θ. These can be rearranged so that:-

$$x_{new} = cos(\theta)(x - ytan(\theta)) \tag{3.3}$$

$$y_{new} = cos(\theta)(y + xtan(\theta)) \tag{3.4}$$

In order to accommodate this algorithm in hardware the rotation is restricted so that tan(θ)=$\pm 2^i$, the multiplication by tangent term is reduced to shift operation.Arbitary angles of rotation are obtained by performing a aeries of smaller rotation. If the decision at each iteration i, is which direction to rotate rather than whether or not to rotate, then cos(θ) term becomes constant.

$$x_{i+1} = k_i[x_i - y_i d_i 2^- i] \tag{3.5}$$

$$y_{i+1} = k_i[y_i + x_i d_i 2^- i] \tag{3.6}$$

where,

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

$$d_i = \pm 1$$

d_i is depended on the direction in which angle rotation is taking place at each stage. This K_i becomes equal to 0.6073 as iterations go to infinity. The iterations are done till the angle Z_i reaches 0. So,

$$z_{i+1} = z_i - d_i tan^{-i}(2^{-i}) \tag{3.7}$$

3.2 Implementation

The algorithm is implemented using a verilog module which is called from the fft module and each stage is performed at the positive edge of the clock.Here we are calculating a 8 point DFT so at each calculation we need 4 twiddle factor and the remaining 4 are the negative of first 4.For the tan^{-1} a table is stored in ROM for 8 stages.

Given $x[n]$ as input function for n=0 to 7(N=8), the DFT is

$$X[k] = \sum_{n=0}^{7} x[n] e^{j\frac{2\pi}{N}kn} \tag{3.8}$$

here twiddle factors areW_N^{kn}, and we are storing only 4 of these values i.e W_N^0,W_N^1,W_N^2 and W_N^3 and using those and CORDIC, we calculate the remaining ones.
The angles received as an input to is first brought in the 1^{st} and 4^{th} quadrant, only then the CORDIC will work properly.The correction in angle and the input values is shown below:-
In 2^{nd} quadrant $x_0 = -y_{in}$, $y_0 = x_{in}$ and $z_0 = angle - \pi/2$ and for third quadrant, $x_0 = y_{in}$, $y_0 = -x_{in}$ and $z_0 = angle + \pi/2$.After these corrections, we can use above mentioned algorithm properly.

The input in fixed point notation Q1.7 is given serially and stored in a 2D array.As we are doing 8 point FFT the no. of W_N^{kn} term we need to calculate is 4 and multiplied

with respective input index and the remaining 4 exponential values are negative of first 4.

At each DFT e.g X[1],

$$X[1] = \sum_{n=0}^{N-1} x[n]W_N^n$$

$$X[1] = x[0]W_N^0 + x[1]W_N^1 + x[2]W_N^2 + x[3]W_N^3 + x[4]W_N^4 + x[5]W_N^5 + x[6]W_N^6 + x[7]W_N^7$$

and as $W_N^{r+N/2} = -W_N^r$

$$X[1] = x[0]W_N^0 + x[1]W_N^1 + x[2]W_N^2 + x[3]W_N^3 - x[4]W_N^0 - x[5]W_N^1 - x[6]W_N^2 - x[7]W_N^3$$

These 4 values are called from main FFT module 4 times for each different angle in order to calculate the $cos(\theta)$(real part) and $sin(\theta)$(imaginary part) of the corresponding angle and as we are using 8 iterations for completion of CORDIC(to get an accuracy upto first decimal point), it takes 8 clock cycle for it to finish and hence we have to slow down the FFT calculation clock(minimum 8 times slower). **This is the problem that is solved in the next chapter where the sequential CORDIC calculating module is transformed into combinatorial circuit.**

3.3 Simulations

Here we are ginving the angle of $3pi/4$ in (Q3.7) format in the CORDIC module which is run for 8 iteration. It can be seen in the figure3.1 that the output(Q2.7) cos value is $\frac{-1}{\sqrt{2}}$ and sin value is $\frac{1}{\sqrt{2}}$

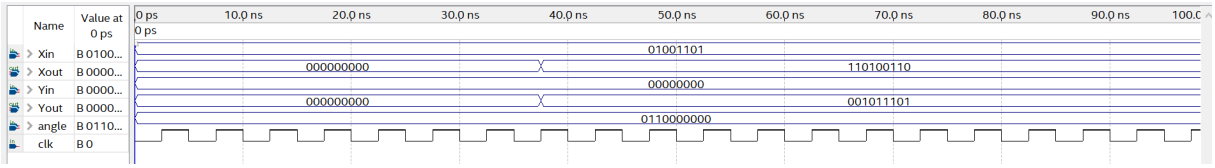

Figure 3.1: CORDIC operation for 8 iterations

The DFT(simulation) for the above mentioned algorithm is shown in 3.2:-

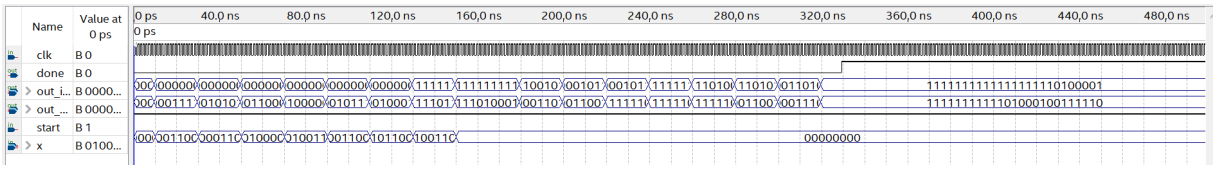

Figure 3.2: 8 point DFT

3.4 results

The synthesis and implementation of the above mentioned algorithm is done on **Spartan 6 XC6SLX75 Board**.

Table 3.1: Resource Utilization for DFT with CORDIC

Logic Utilization	**Used**	**Available**	**Utilzation**
Number of slice registers	573	93296	<1%
Number of slice LUT's	589	46648	1%
Number of fully used LUT FF pairs	496	607	81%
Number of bonded IOBs	61	280	21%
Number of BUFG/BUFGCTRLs	2	16	12%
Number of DSP48A1s	16	132	12%

The timing summary for the Table 3.1 is given below:-

1. Maximum Frequency :- 278.145 MHz.
2. Minimum input arrival time before clock:- 4.570ns.
3. Maximum output required time after clock:- 24.98 ns

Table 3.2: Resource Utilization for DFT with Twiddle factor in RAM Block

Logic Utilization	**Used**	**Available**	**Utilzation**
Number of slice registers	448	93296	<1%
Number of slice LUT's	509	46648	<1%
Number of fully used LUT FF pairs	445	512	86%
Number of bonded IOBs	73	280	26%
Number of BUFG/BUFGCTRLs	1	16	6%
Number of DSP48A1s	24	132	18%

The timing summary for the Table 3.2 is given below:-

1. Maximum Frequency :- 97.88 MHz.
2. Minimum input arrival time before clock:- 4.225ns.
3. Maximum output required time after clock:- 3.634 ns

Table 3.3: Resource Utilization for Cooley tukey Algorithm

Logic Utilization	**Used**	**Available**	**Utilzation**
Number of slice registers	422	93296	<1%
Number of slice LUT's	863	46648	4%
Number of fully used LUT FF pairs	403	865	45%
Number of bonded IOBs	37	280	13%
Number of BUFG/BUFGCTRLs	1	16	6%
Number of DSP48A1s	4	132	3%

The timing summary for the Table 3.3 is given below:-

1. Maximum Frequency :- 125 MHz.
2. Minimum input arrival time before clock:- 4.97ns.
3. Maximum output required time after clock:- 3.57 ns

NOTE:-All the above mentioned algorithms are calculating 8 point DFT of real input.

Out of Table 3.1, 3.2 and 3.3, the DFT calculation with twiddle factor in ROM block uses maximum no. of DSP48A1 as it has maximum no. of multiplications as DSP48A1 has 18×18 bit multiplier and 48 bit accumulator. LUTs are maximum in DFT with cooley tukey algorithm, it has maximum number of function(different bit adders) unlike the other 2 where we are reusing the same module more than once (at different clock edges) and hence the complexity of program to be implemented using LUT is less. DFT which uses CORDIC for twiddle factor calculation(Table 3.1) has the highest maximum allowable clock frequency.

CHAPTER 4

DFT with multiplexer based CORDIC

4.1 Algorithm for MUX based CORDIC

CORDIC here is used in the same way in previous chapters in order to calculate cosine and sine of different angles involved while calculating twiddle factor for DFT. The scheme for reducing the area of the CORDIC using multiplexer is proposed for the FPGA implementation.

The area is reduced by removing all the stages and replacing them with multiplexers.As shown in chapter 3, the calculation of cos and sin of each angle takes a minimum of 8 iteration in order to get a precision of 1 decimal point and each iteration takes place at the positive edge of clock which hinders the ability to speed up the DFT calculation upto a point.The entire sequential circuit of chapter 3 is replaced by a combinational circuit.

The first stage output of original unrolled CORDIC architecture is equal to x_i,therefore we can directly write the output of first stage as

$$y_1 = x_i \tag{4.1}$$

$$x_1 = x_i \tag{4.2}$$

If the first stage output is positive, then

$$y_2 = y_1 - x_1/2 = x_1/2 \tag{4.3}$$

$$x_2 = x_1 + y_1/2 = 3x_1/2 \tag{4.4}$$

The vector coordinates corresponding to negative output is

$$x_2 = x_1 - y_1/2 = x_1/2 \tag{4.5}$$

$$y_2 = y_1 + x_1/2 = 3x_1/2 \tag{4.6}$$

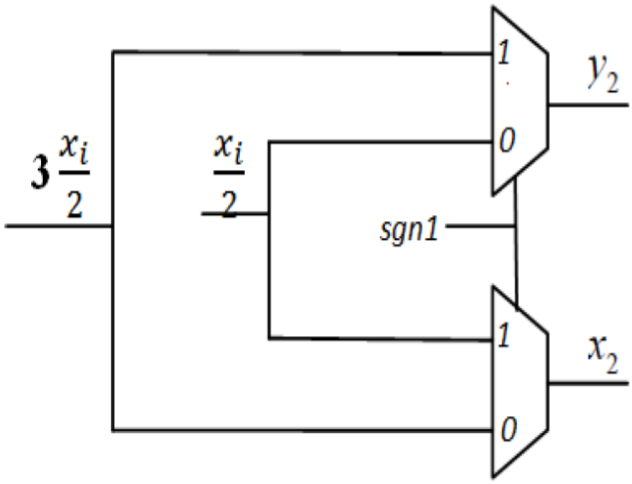

Figure 4.1: MUX operation for the first 2 stages

4.2 Calculation for further stages

Given x_{in} and y_{in}, we have calculate the constant multipliers for each multiplexer.Here we are using 2 multiplexer:-one 8 to 1 multiplexer and other 16 to 1 multiplexer.The selection input are actually the direction of rotation at each stage,so for 8 to 1 multiplexer first 3 rotation direction are selector input(0 for clockwise and 1 for anti clockwise) and for 16 to 1 next 4 direction act as selector input. **NOTE:- The constant multiplier for both the multiplexer are stored in fixed point notation of Q2.7 and output of both multiplexer is Q5.14 which is truncated to Q2.7**

The 8 to 1 multiplexer is shown below:-Input $Z_2Z_1Z_0$=

for 000,

$$X = 0.125X_{in} - 1.625Y_{in}$$

$$X = 1.625X_{in} + 0.125Y_{in}$$

for 001,

$$X1.625X_{in} + 0.125Y_{in}$$

$$X = -0.125X_{in} + 1.625Y_{in}$$

for 010,

$$X = 1.375X_{in} - 0.875Y_{in}$$

$$X = 0.875X_{in} + 1.375Y_{in}$$

for 011,

$$X = 0.875X_{in} + 1.375Y_{in}$$

$$X = -1.375X_{in} + 0.875Y_{in}$$

for 100,

$$X = 0.875X_{in} - 1.375Y_{in}$$

$$X = 1.375X_{in} + 0.875Y_{in}$$

for 101,

$$X = 1.375X_{in} + 0.875Y_{in}$$

$$X = -0.875X_{in} + 1.375Y_{in}$$

for 110,

$$X = 1.625X_{in} - 0.125Y_{in}$$

$$X = 0.125X_{in} + 1.625Y_{in}$$

for 111,

$$X = 0.125X_{in} + 1.625Y_{in}$$

$$X = -1.625X_{in} + 0.125Y_{in}$$

We can see here only for values are needed to be stored in ROM for above calculation:-

1. 0.125
2. 1.625

3. 1.375

4. 0.875

The output of this multiplexer is taken and given as input for the 16 to 1 multiplexer(Selector input are $Z_6Z_5Z_4Z_3$:- for 0000,

$$X_{out} = 0.9829X - 0.2339Y$$

$$Y_{out} = 0.2339X + 0.9829Y$$

for 0001,

$$X_{out} = 1.01X + 0.0152Y$$

$$Y_{out} = -0.0152X + 1.01Y$$

for 0010,

$$X_{out} = 1.004X - 0.109Y$$

$$Y_{out} = 0.1097X + 1.004Y$$

for 0011,

$$X_{out} = X + 0.1408Y$$

$$Y_{out} = -0.1408X + Y$$

for 0100,

$$X_{out} = 0.995X - 0.1720Y$$

$$Y_{out} = 0.172X + 0.995Y$$

for 0101,

$$X_{out} = X + 0.078Y$$

$$Y_{out} = -0.078X + Y$$

for 0110,

$$X_{out} = X - 0.046Y$$

$$Y_{out} = 0.046X + Y$$

for 0111,

$$X_{out} = 0.98X + 0.2030Y$$

$$Y_{out} = -0.2030X + 0.98Y$$

for 1000,

$$X_{out} = 0.98X - 0.2030Y$$

$$Y_{out} = 0.2030X + 0.98Y$$

for 1001,

$$X_{out} = X + 0.046Y$$

$$Y_{out} = -0.046X + Y$$

for 1010,

$$X_{out} = X - 0.078Y$$

$$Y_{out} = 0.078X + Y$$

for 1011,

$$X_{out} = 0.995X + 0.1720Y$$

$$Y_{out} = -0.172X + 0.995Y$$

for 1100,

$$X_{out} = X - 0.1408Y$$

$$Y_{out} = 0.1408X + Y$$

for 1101,

$$X_{out} = 1.004X + 0.109Y$$

$$Y_{out} = -0.1097X + 1.004Y$$

for 1110,

$$X_{out} = 1.01X - 0.0152Y$$

$$Y_{out} = 0.0152X + 1.01Y$$

for 1111,

$$X_{out} = 0.9829X + 0.2339Y$$

$$Y_{out} = -0.2339X + 0.9829Y$$

There is a pattern in the above calculations and hence the values that needed to be stored in ROM are:-

1. 0.9829 and 0.2339
2. 1.01 and 0.0152
3. 1.004 and 0.875
4. 1 and 0.1408
5. 0.995 and 0.1720
6. 1.007 and 0.078
7. 1 and 0.046
8. 0.98 and 0.2030

The FFT module gets a real input of notation Q1.7 and it stores 4 twiddle factor mentioned in 3 and for calculation of each DFT the multiplexer modules give out the first 4 W_N^{kn} value and the remaining 4 will be negative of first 4.

4.3 Simulations

Here in figure:4.2, we are giving $\pi/4$ as an input angle and the gain is 1.6, the output we got is $\frac{1}{\sqrt{2}}$in fixed point notation Q2.7.

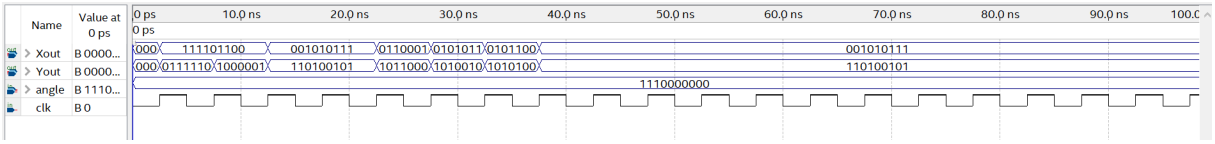

Figure 4.2: Output of MUX bases CORDIC Module

The input given is :-
x[0]=0.5, x[1]=0.2, x[2]=0.1, x[3]=0.25, x[4]=0.3, x[5]=0.2, x[6]=0.7 and x[7]=0.6,
We can see in the simulation(figure:4.3 is that the output DFT we get is serial after all serial input data is written in the memory.Output is of format Q11.14 and can be truncated.

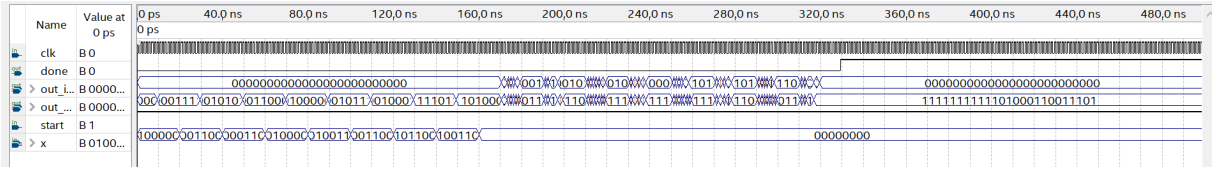

Figure 4.3: Output of FFT module with MUX based CORDIC

4.4 Results

Table 4.1: Resource Utilization for DFT with MUX based CORDIC

Logic Utilization	**Used**	**Available**	**Utilzation**
Number of slice registers	244	93296	<1%
Number of slice LUT's	3789	46648	8%
Number of fully used LUT FF pairs	184	3818	4%
Number of bonded IOBs	61	280	21%
Number of BUFG/BUFGCTRLs	2	16	12%
Number of DSP48A1s	16	132	12%

The timing summary for the Table5.1 is given below:-

1. Maximum Frequency :- 286 MHz.
2. Minimum input arrival time before clock:- 4.05ns.
3. Maximum output required time after clock:- 42 ns

We can see the MUX based CORDIC is faster than the CORDIC in chapter 3 as the entire sequential circuit is converted into combinatorial. But using a tree like structure used for decision making (which in real life circuit is realized by multiplexers) at each stage has led to high usage of Slice LUTs.Number of multiplications are almost same. The LUT here are not just used for logic but also as a shift register , carry load and a combination of 06 and 05 are also used in order to realize the complex logic unlike other 2 algorithm where most of the LUT used is O6.

CHAPTER 5

DFT with CORDIC without barrel shifter

5.1 Overview of algorithm

The objective of this algorithm is to eliminate the use of barrel shifter.The basic idea is to fix the angles by which the rotation is done at each iteration such that the need of barrel shifter is eliminated. The 2 angles were selected using trial and error method which gives the minimum error.

The 2 angles were chosed to represent all the angles in [-180,180] range, that are $tan^{-1}(2^{-1})$ and $tan^{-1}(2^{-3})$,corresponding rotations provide architecture for X and Y with merely shifters and adder/subtractors.

$$Z = k_0 tan^{-1}(2^{-1}) + k_1 tan^{-1}(2^{-3})$$

. such that $|k_0| + |k_1|$ is minimized.The following are the few values of k_0 and k_1:-

Table 5.1: Values of k_0 and k_1 with corresponding angles

Angles	k_0	k_1	**Total iterations**
1	0	0	0
2	-1	4	5
3	2	-7	9
4	2	-7	9
5	1	-3	4
-2	1	-4	5
-3	-2	7	9
-4	-2	7	9
-5	-1	3	4

As we can see for negative angle, the absolute values of k_0 and k_1 remain same, hence the number of iterations remain same but the sign of k_0 and k_1 are changed.So, we only need to store 180 values.The maximum no. of iteration **13** is for **170**$^\circ$ and **-170**$^\circ$.

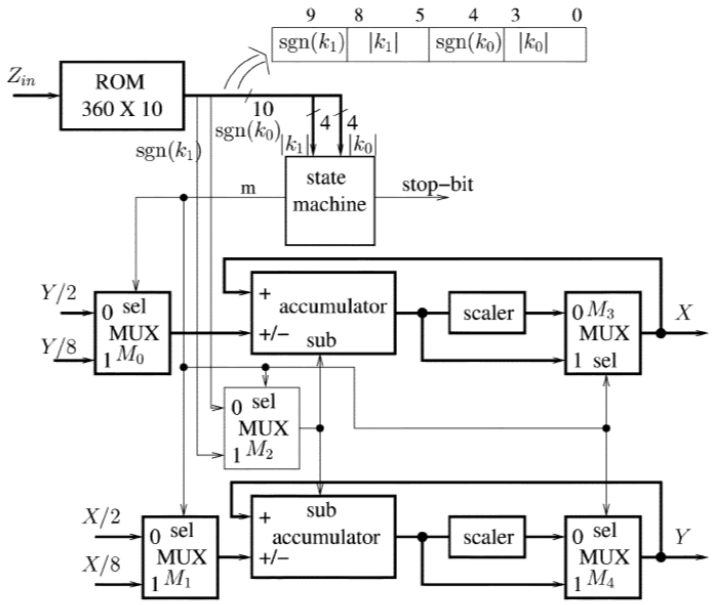

Figure 5.1: Architecture for above mentioned algorithm

The Architecture for the algorithm is shown above.

The state machine runs $|k_o|+|k_1|$ times. It has two outputs:m and stop bit.the bit m is set for 0 for k_0 times and set to 1 for k_1 times.The angle for rotation is $sgn(k_0)tan^{-1}(2^{-1})$, when m is 0 while it is $sgn(k_1)tan^{-1}(2^{-1})$, when m is 1. The values of X nad Y is chosen by the multiplexer M_0 and M_1 as shown in the 5.1.The MUX M_2 selects the direction of rotation. This operation can be represented in form of equation:-

1. If the angle of rotation is $tan^{-1}(2^{-1})$:-

$$X = K(X - (-1)^{sgn(k_0)}.Y.2^{-1})$$
$$Y = K(Y + (-1)^{sgn(k_0)}.X.2^{-1})$$

2. If the angle of rotation is $tan^{-1}(2^{-3})$:-

$$X = X - (-1)^{sgn(k_1)}.Y.2^{-3}$$
$$Y = Y + (-1)^{sgn(k_1)}.X.2^{-3}$$

Here $K = 0.8944$ is a correction in order to nullify the gain we incur in rotation from $x = 1$ and $y = 0$.

5.2 Simulations

Here in figure:5.2, we are giving $\pi/2$ as an input angle , the output we got is cos=0 and sin= 1 in fixed point notation Q2.7.

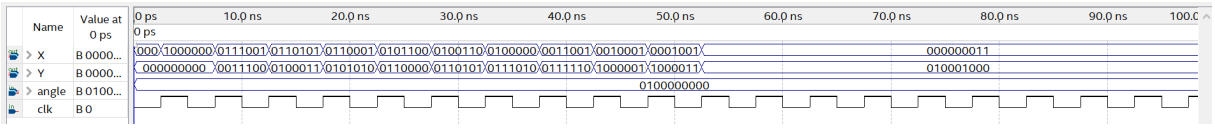

Figure 5.2: Output of CORDIC module for angle=90°

NOTE:-there is an error of 1° present for few angles

5.3 Results

Table 5.2: Resource Utilization for DFT for CORDIC without barrel shifter

Logic Utilization	**Used**	**Available**	**Utilzation**
Number of slice registers	163	93296	1%
Number of slice LUT's	373	46648	1%
Number of fully used LUT FF pairs	152	374	40%
Number of bonded IOBs	61	280	21%
Number of BUFG/BUFGCTRLs	2	16	12%
Number of DSP48A1s	16	132	12%

The timing summary for the Table5.2 is given below:-

1. Maximum Frequency :- 167.675 MHz.
2. Minimum input arrival time before clock:- 4.247ns.
3. Maximum output required time after clock:- 25.08 ns

The combinatorial logic corresponding rotation by angle $tan^{-1}(2^{-1})$ includes multiplication (for the correction of gain incurred by rotation) with shifting and adding and hence the LUT in a slice might not be enough and need another LUT. In this way the FF corresponding to the first LUT remain unused and hence the percentage of fully used LUT-FF pair is less than normal CORDIC based DFT.
But the absence of barrel shifter led to less number of LUT used.

CHAPTER 6

Conclusion

The resource allocation for the 3 different implementation of CORDIC algorithm after synthesis and implementation on Spartan 6 board is shown below:-

Table 6.1: Resource Utilization

Logic Utilization	**DFT with normal CORDIC**	**DFT with MUX based CORDIC**	**DFT with CORDIC without barrel shifter**
Slice registers	573	244	163
Slice LUT's	589	3789	373
Number of fully used LUT FF pairs	81%	4%	40%
Maximum frequency	278 MHz	286 MHz	167.675MHz
Bonded IOBs	61	61	61
BUFG/BUFGCTRLs	2	2	2
DSP48A1s	16	16	16

The following are the noticeable points in the above shown data:-

1. **LUT used**:-MUX based CORDIC uses most LUTs as tree like logic is used to implement it and that logic is implemented in a form of 2 multiplexer (one 8 to 1 MUX and other 16 to 1 MUX).
   The no. of LUT in *CORDIC without barrel shifter* is lesser than *normal CORDIC* as there is no shifting more than 3 bits.

2. **Fully used LUT-FF pair**:- MUX based CORDIC has the least percentage as most of the logic is combinatorial not sequential and hence only LUT are used not the FF in the slices.
   Normal CORDIC has the highest percentage as the every calculation is done at positive edge of clock so there is a proper proportion of combinatorial logic and the Flipflops.

3. **Slice Registers**:- These are used to implement state machines in form of flipflops.

4. **Maximum frequency**:-It can be seen that CORDIC without barrel shifter is slower compared to other 2 as the average no. of iterations are higher than other 2 (but no. of LUTs and registers are significantly less because use of barrel shifter is eliminated).
   The maximum frequency of *normal CORDIC* and *MUX based CORDIC* are same as in both we have to wait 7 iteration in order to get the direction of rotation at each iteration. **The speed of MUX based CORDIC can be significantly increased by eliminating the sequential logic used for calculation of direction**

of rotation and actually saving the direction for each angle(total 360) in Block RAM as we did in third algorithm but this can't be done for *normal CORDIC* as the remaining logic for calculation of cos and sine value is sequential unlike *MUX based CORDIC* which is purely combinatorial.

REFERENCES

1. Leena Vachhani, K. Sridharan, and Pramod K. Meher Efficient CORDIC Algorithms and Architectures for Low Area and High Throughput Implementation *IEEE Transactions on circuits and systems*,January 2009

2. V.Naresh , B.Venkataramani and R.Raja An area efficient multiplexer based CORDIC *2013 International Conference on Computer Communication and Informatics (ICCCI -2013)*,Jan 04, 2013.

3. Renu Bala, Shamim Aktar Fast Fourier Transformation Realization with Distributed Arithmetic *International Journal of Computer Applications*Volume 102 No.15, September 2014

4. Muzhir Shaban Mohammed, Santiago Lorenzo Matilla and LUISNozal Fast 2D convolution filter based on look up table FFT

5. Robert J Francis A tutorial on logic synthesis for lookup table

6. G. William Slade, The Fast Fourier Transform in Hardware ,20 May 2014