CORDIC ALGORITHM

Implementation of CORDIC Algorithm using Verilog

Mohamed Ahmed Ezz eddin Hussien
Chipions'27

Contents

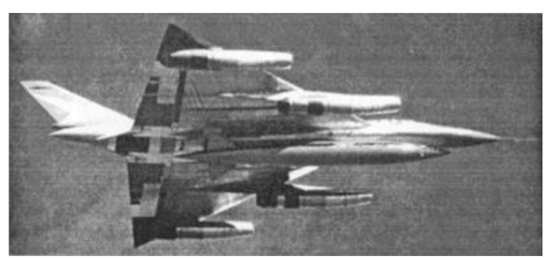
1.Introduction :

The CORDIC algorithm(COrdinate Rotation DIgital Computer) is an iterative convergence method to efficiently compute trigonometric, hyperbolic and logarithmic functions. Since it requires only addition/subtraction, bit-shifting operation and ROM look-up tables the computational complexity is extremely low, making it suitable for low cost implementation when no hardware multiplier is available. Verilog was used to implement the design and run simulations. MATLAB was used to create the testbench code and to verify and analyze the results.

1.1Piece of History :

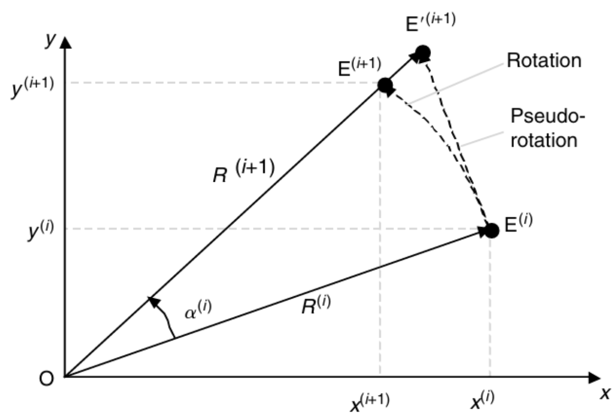
The CORDIC algorithm was introduced in 1959 by Volder. In Volder's version, CORDIC makes it possible to perform rotations (and therefore to compute sine, cosine, and arctangent functions) and to multiply or divide numbers using only shift-and-add elementary steps. In 1971, Walther generalized this algorithm to compute logarithms, exponentials, and square roots. CORDIC is not the fastest way to perform multiplications or to compute logarithms and exponentials but, since the same algorithm allows the computation of most mathematical functions using very simple basic operations, it is attractive for hardware implementations. CORDIC has been implemented in many pocket calculators since Hewlett Packard's HP 35, and in arithmetic coprocessors such as the Intel 8087. Some authors have proposed the use of CORDIC processors for signal processing applications

Jack E. Volder is a flight engineer during WW2–1956 replace analog computer of B58 bomber with digital computer



1.2 overview

The CORDIC algorithm works with a similar pattern of Binary Search, but on a circumference, and instead of fixed linear addition/subtraction of progressively smaller distances it requires trigonometric identities using fixed angle rotations.



When we rotate vector E(i) by $\alpha(i)$, the new vector E(i+1) has these coordinates (x(i+1) , y(i+1)) :

$$x^{(i+1)} = x^{(i)} \cos \alpha^{(i)} - y^{(i)} \sin \alpha^{(i)} = \frac{x^{(i)} - y^{(i)} \tan \alpha^{(i)}}{(1 + \tan^2 \alpha^{(i)})^{1/2}}$$

$$y^{(i+1)} = y^{(i)} \cos \alpha^{(i)} + x^{(i)} \sin \alpha^{(i)} = \frac{y^{(i)} + x^{(i)} \tan \alpha^{(i)}}{(1 + \tan^2 \alpha^{(i)})^{1/2}} \qquad \text{[Real rotation]}$$

$$z^{(i+1)} = z^{(i)} - \alpha_i$$

where the variable z allows us to keep track of the total rotation over several steps.

There is a complexity when we want to compute the next point after rotate ,term contains tan function , so we can multiply by this term which increase the magnitude of the vector (Pseudo rotation) .

$$R^{(i+1)} = R^{(i)} (1 + \tan^2 \alpha^{(i)})^{1/2}$$

Doing this , the calculation of the new point is easier , so all we need to compute is $\tan \alpha^{(i)}$.

$$
\begin{aligned}
x^{(i+1)} &= x^{(i)} - y^{(i)} \tan \alpha^{(i)} \\
y^{(i+1)} &= y^{(i)} + x^{(i)} \tan \alpha^{(i)} \qquad \text{[Pseudorotation]} \\
z^{(i+1)} &= z^{(i)} - \alpha^{(i)}
\end{aligned}
$$

We can recompute $\tan \alpha^{(i)}$ and load it to a ROM (Look-up table) and force it to be in range of radix-2 as shown in the following table:

$e^{(i)} = \tan^{-1} 2^{-i}$, for $0 \le i \le 9$

i	$\approx e^{(i)}$, **degrees**	$e^{(i)}$, **radians**
0	45.0	0.785 398 163 397
1	26.6	0.463 647 609 001
2	14.0	0.244 978 663 127
3	7.1	0.124 354 994 547
4	3.6	0.062 418 809 996
5	1.8	0.031 239 833 430
6	0.9	0.015 623 728 620
7	0.4	0.007 812 341 060
8	0.2	0.003 906 230 132
9	0.1	0.001 953 122 516

In CORDIC terminology , the preceding selection rule for d_i , which makes Z converge to 0 , is known as "rotation mode."

We can rewrite the CORDIC iterations as follows, where :

$$d_i = sign\left(z^{(i)}\right) \qquad d_i \in \{-1,1\}$$

$$
\begin{aligned}
x^{(i+1)} &= x^{(i)} - d_i(2^{-i} y^{(i)}) \\
y^{(i+1)} &= y^{(i)} + d_i(2^{-i} x^{(i)}) \qquad \text{[CORDIC iteration]} \\
z^{(i+1)} &= z^{(i)} - d_i e^{(i)}
\end{aligned}
$$

After m pseudo rotations

$$
\begin{aligned}
x^{(m)} &= K(x\cos z - y\sin z) \\
y^{(m)} &= K(y\cos z + x\sin z) \qquad\qquad \text{[Rotation mode]} \\
z^{(m)} &= 0 \\
\text{Rule : } & \text{ Choose } d_i \in \{-1, 1\} \text{ such that } z \to 0.
\end{aligned}
$$

The constant K in the preceding equations is K = 1.646 760 258 121 , could the expansion factor which makes the magnitude of vector after rotation increased .

$$K = \prod \sqrt{1 + (\tan\alpha^i)^2}$$

The expansion factor depends on the rotation angle . However, if we always rotate by the same angles, with positive or negative signs, then K is a constant that can be precomputed. In this case, using the simpler pseudo rotations instead of true rotations has the effect of expanding the vector coordinates and length by a known constant.

To compute cos z and sin z, one can start with x = 1/K = 0.607252935··· and y = 0.

$$X^{(m)} = K\left(\frac{1}{K}\,Cos(z)\ - (0)Sin(z)\right) = Cos(z)$$

$$Y^{(m)} = K\left((0)Cos(z)\ - \frac{1}{K}\,Sin(z)\right) = Sin(z)$$

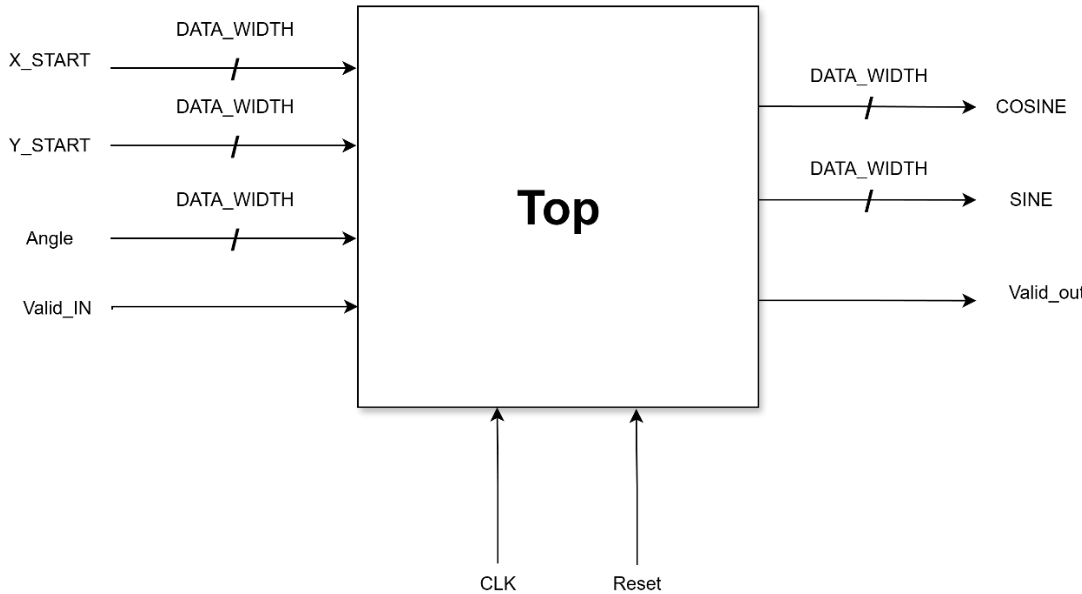
$$Z^{(m)} \approx 0$$

Then, as z(m) tends to 0 with CORDIC iterations in rotation mode , x(m) and y(m) converge to cos z and sin z, respectively. Once sin z and cos z are known, tan z can be obtained through division if desired.

For k bits of precision in the resulting trigonometric functions, k CORDIC iterations are needed.

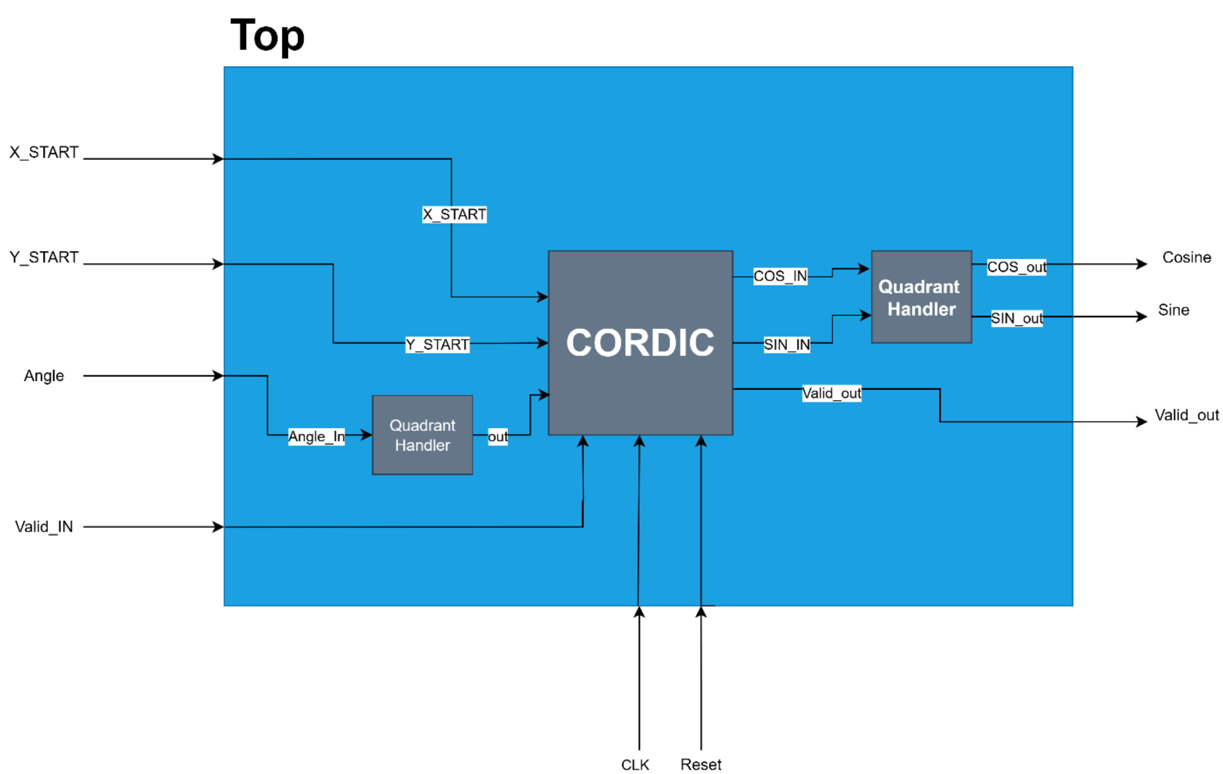
2.Design Architecture :

2.1Top-Level Hierarchy :



Signals	Direction	Description
X_START	Input	Initial value of X
Y_START	Input	Initial value of Y
Angle	Input	Angle by at the end of iterations , its sine and cosine would be calculated
Valid_in	Input	Signal Asserted if the input is right and available now
Cosine	Output	Final value of X , cosine of input angle
Sine	Output	Final value of Y , sine of input angle

Valid_out	Output	Signal Asserted if the output is right and avaliable
CLK	Input	Clock Signal
Reset	Input	Active-Low Asynchronous Reset



2.2 CORDIC :

The precomputed angles that used for microrotations are loaded to a Look-up table (ROM) and generated by MATLAB Script .

the iterations will start when valid_in input is asserted which means that the input is available now . The code is parameterized by Number of bits and number of iterations , counter is used to track the number of iteration , this counter returns to zero when counts to the desired number of iterations.

```
always @(posedge clk , negedge arst_n)
begin
    if(~arst_n) begin
        iteration_counter <= 'd0 ;
    end
    else if(valid_in)
        {x , y , z } <= {x_start , y_start , angle};
    else begin
        if(z > 0) begin
            x <= x - (y>>>iteration_counter);
            y <= y + (x>>>iteration_counter);
            z <= z - e_i ;
        end
        else begin
            x <= x + (y>>>iteration_counter);
            y <= y - (x>>>iteration_counter);
            z <= z + e_i ;
        end

        iteration_counter <= (iteration_counter == n_iterations-1) ? 'd0 : iteration_counter + 1 ;

    end
end

assign valid_out = (iteration_counter == n_iterations-1) ;
assign sine = y ;
assign cosine = x ;
```

2.3 Quadrant Handler :

Our algorithm will converge within a range of anlges from -99.7 to +99.7 (summation of all precomputed angles in the same direction either negative direction or positive direction) so we need to map angles outside this range to the target range.

By using trigonometric identities :

$$\sin(\theta \pm 2\pi k) = \sin(\theta)$$

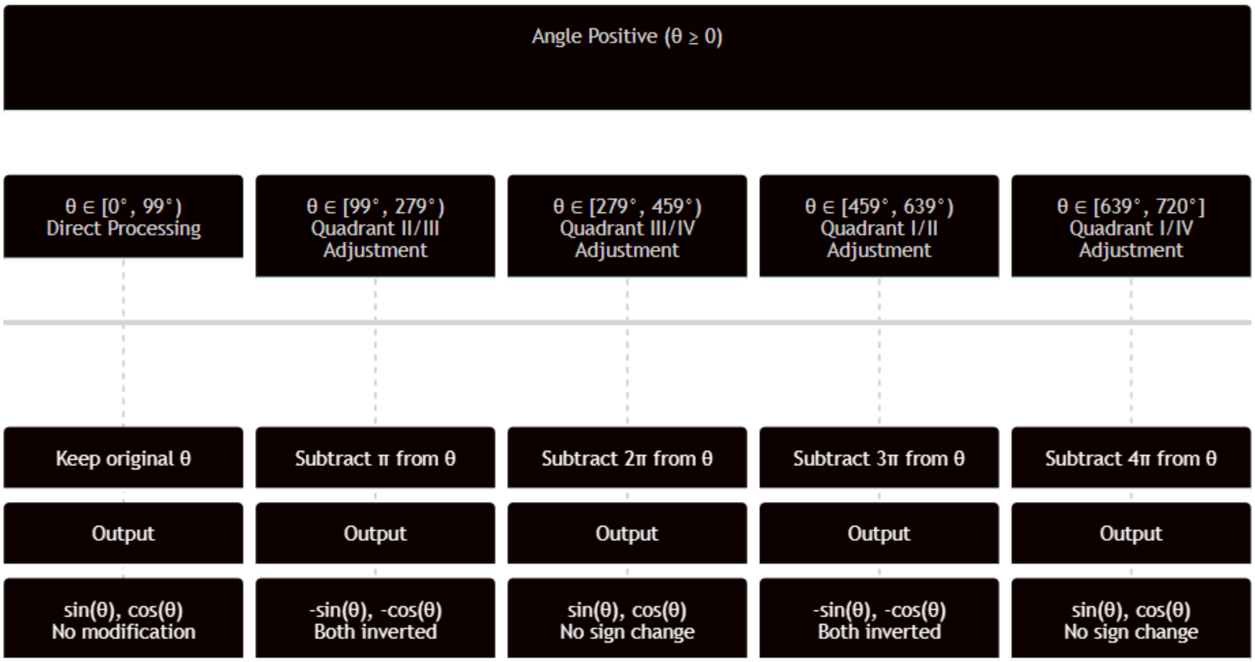
$$\cos(\theta \pm 2\pi k) = \cos(\theta)$$

$$\sin(\theta \pm \pi k) = -\sin(\theta)$$

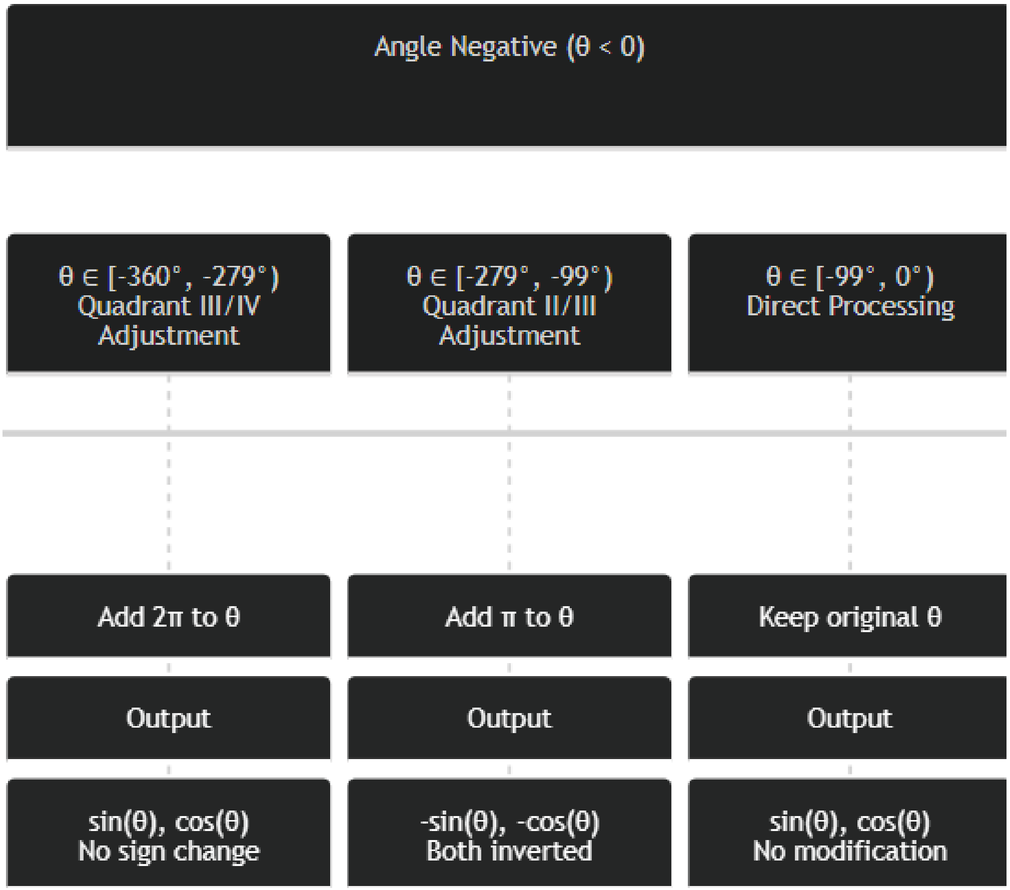
$$\sin(\theta \pm \pi k) = -\cos(\theta)$$

After we apply the trigonometric identities (Add or subtract pi or 2pi) , the angle may be still outside the range of convergence so we need to consider it by dividing our range $[-2\pi : +4\pi]$ into sub ranges each one has its own relation of Addition or subtraction .

For Positive Angles :



For Negative Angles :



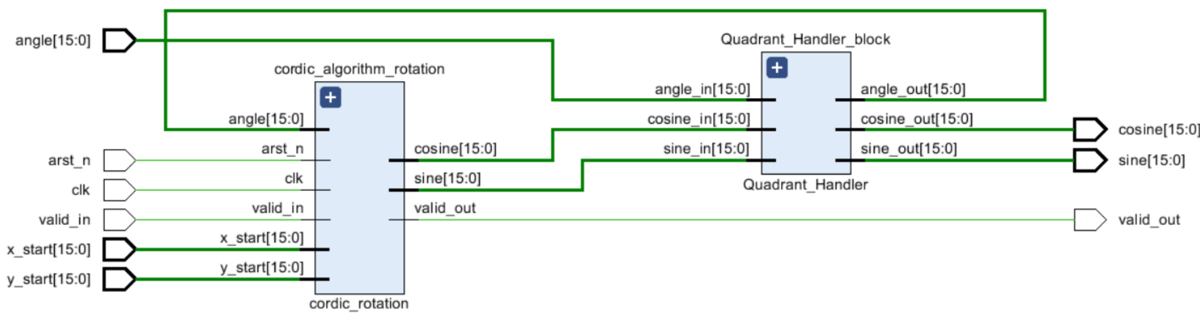
The Boundaries angles were converted and scaled to be represented in Fixed-Point binary

Using MATLAB and loaded to a ROM to access it in Verilog file . The last 4 elements is representing the Constants [PI , 2PI , 3PI , 4PI] used in the code

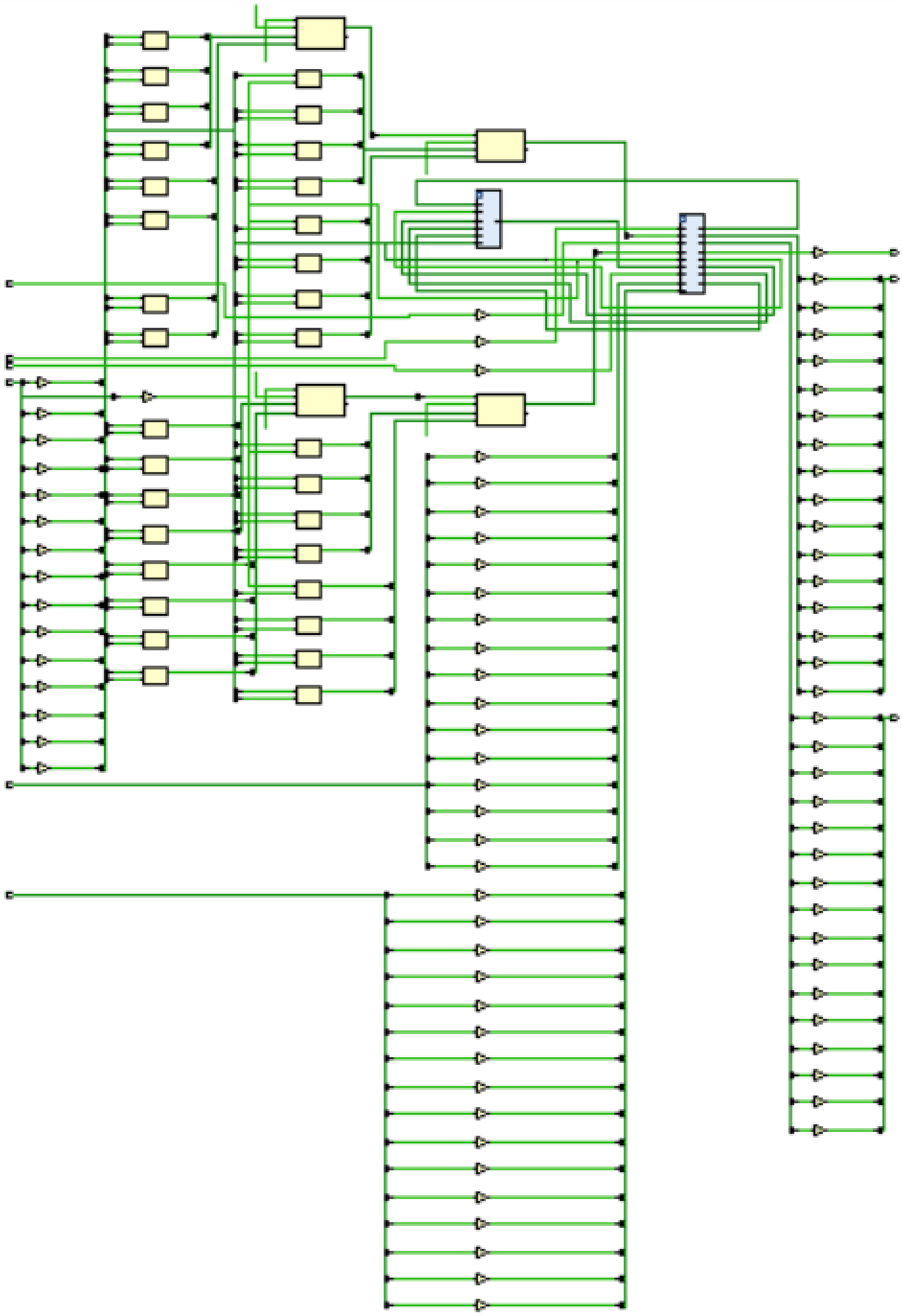
Address:	0	1	2	3	4	5	6	7	8	9	10	11
Angle:	-360°	-279°	-99°	99°	279°	459°	639°	720°	180°	360°	540°	720°

3. Reports :

3.1 RTL Analysis :



3.2 RTL synthesis netlist



3.3 STA Report Before Implementation:

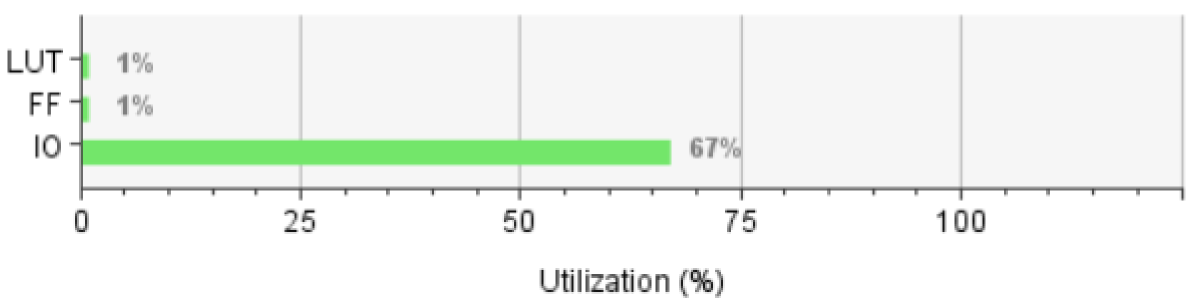
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3.959 ns	Worst Hold Slack (WHS):	0.168 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	53	Total Number of Endpoints:	53	Total Number of Endpoints:	53

All user specified timing constraints are met.

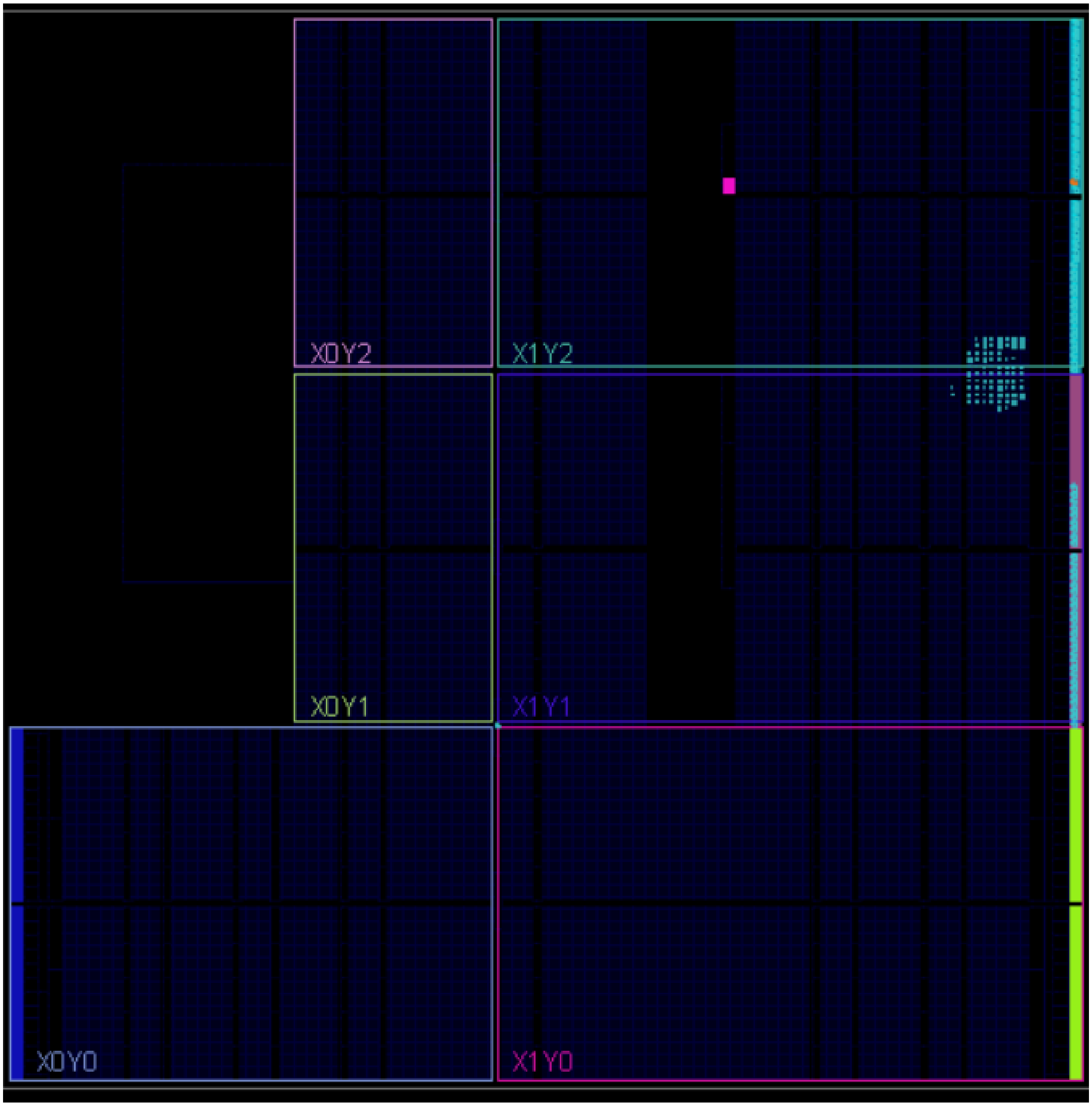
3.4 Utilization Report Before Implementation:

Name	Slice LUTs (53200)	Slice Registers (106400)	Bonded IOB (125)
top	283	53	84
cordic_algorithm_rotati...	267	53	0
Quadrant_Handler_blo...	0	0	0

Resource	Utilization	Available	Utilization %
LUT	283	53200	0.53
FF	53	106400	0.05
IO	84	125	67.20



3.5 Implementation Schematic :

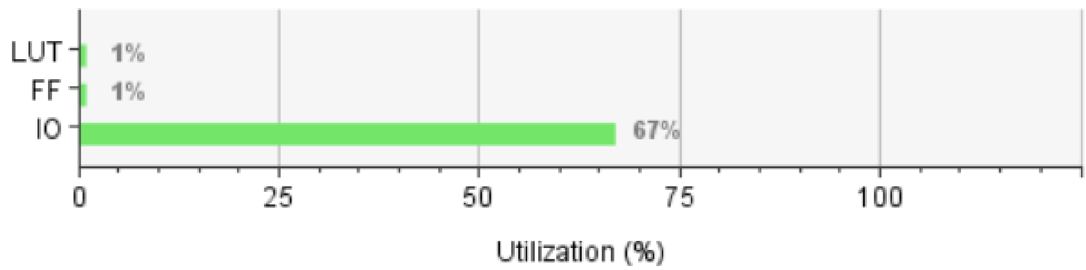


3.6 Utilization Report After Implementation :

Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	LUT Flip Flop Pairs (53200)	Bonded IOB (125)	BUFGCTRL (32)
top	279	53	77	279	43	84	1
cordic_algorithm_rotati...	263	53	73	263	43	0	0
Quadrant_Handler_blo...	0	0	4	0	0	0	0

Summary

Resource	Utilization	Available	Utilization %
LUT	279	53200	0.52
FF	53	106400	0.05
IO	84	125	67.20



3.7 STA Report After Implementation :

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3.011 ns	Worst Hold Slack (WHS):	0.142 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	53	Total Number of Endpoints:	53	Total Number of Endpoints:	54

All user specified timing constraints are met.

4. Testbench :

The Testbench covered :

- Corner Cases : [-360 , -270 , -90 , -180 , 0 , 90 , 180 , 270 , 360]
- Overflow : 720
- A large set of random angles (50 Random Angle)

The Angles were calculated and suitable to be Fixed-Point represented (Q5.11) (5 Intger , 11 Fraction Part) using MATLAB Scripts and Verilog testbench can access these files and drive it into our module.

Three Task are created :

- Drive Task : to drive the angles from the files to the module.

```
task drive;
    input reg signed [DATA_WIDTH -1:0] angle_in ;
    begin
        angle = angle_in ;
        valid_in = 1'b1 ;
        @(negedge clk) valid_in = 1'b0 ;
        repeat(n_iterations-1) @(posedge clk);
    end
endtask
```

- Error Task : to calculate the Absolute error

```
task error;
    input reg signed [DATA_WIDTH-1:0] expected , actual ;
    output real err_abs;

    begin
        err_abs = expected - actual ;
        if(err_abs<0)
            err_abs = -err_abs ;
    end
endtask
```

- Check Task : to check if the Absolute error of our module compared to the one generated by MATLAB is less than Tolerance or not .

```
task check;
    input reg signed [DATA_WIDTH-1:0] expected_sine , expected_cosine ;
    real temp_abs;
    begin
        error(expected_sine,sine, temp_abs);
        error(expected_cosine,cosine, temp_abs);

        $fdisplay(fd_abs_error_sine,"%f",temp_abs*SF);
        $fdisplay(fd_abs_error_cosine,"%f",temp_abs*SF);

        if((temp_abs *SF)> tolerance) begin
            $display("The Output is : ");
            $display("@%0d , angle(degree) : %f : angle (radian) : %f , sine : %f , cosine : %f ",$time ,$itor(angle*SF*(180/PI))  ,$itor(angle*SF)
                ,$itor(sine*SF) ,$itor(cosine*SF)) ;
            $display("But it should be : ");
            $display("sine : %f , Error : %f",$itor(expected_sine*SF) ,temp_abs*SF) ;
            $display("cosine : %f , Error : %f",$itor(expected_cosine*SF) ,temp_abs*SF) ;
            $display("------------------------------------------------------------------------------");
        end
        else
            $display("---------------------------------Test Passed---------------------------------");
    end
```

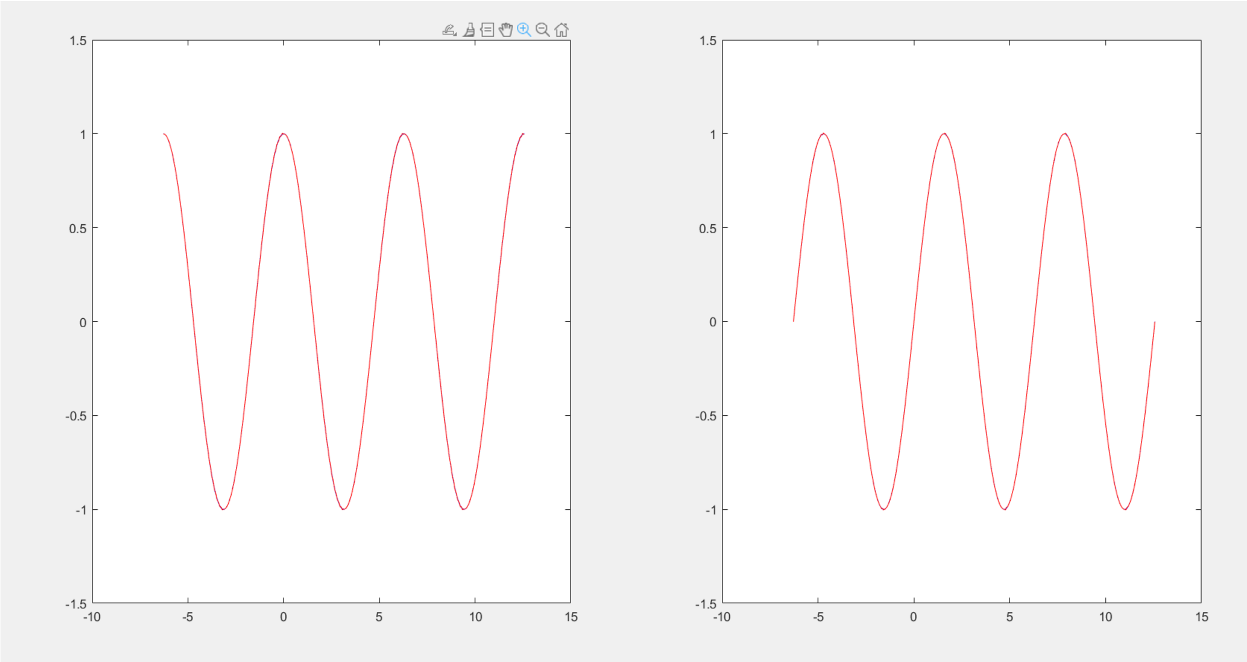
After Simulation :

The testbench stores the output of our DUT to a file and their absolute error to analyze them.

```
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# The Output is :
# @6500 , angle(degree) : -244.710050 : angle (radian) : -4.270996 , sine : 0.906250 , cosine : -0.431641
# But it should be :
# sine : 0.904297 , Error : 0.004395
# cosine : -0.427246 , Error : 0.004395
# -------------------------------------------------------------------------------
# The Output is :
# @6710 , angle(degree) : 346.068746 : angle (radian) : 6.040039 , sine : -0.240723 , cosine : 0.966309
# But it should be :
# sine : -0.240723 , Error : 0.004395
# cosine : 0.970703 , Error : 0.004395
# -------------------------------------------------------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
# ----------------------------------Test Passed----------------------------------
```

We can eliminate these errors by increasing DATA_WIDTH (e.g 32-bit : Q5.27 instead of Q5.11) and increase the number of iterations (e.g. 30 instead of 20).

when we plot the output of our DUT when the input ranges from -360 to 720 using MATLAB versus the ideal :



Most of illustrative graphs is made by Ai.

All of MATLAB Scripts and Verilog codes either for design or testbench are parameterized as much as possible.

Future work : analyze different types of error (Quantization error , precision error due to fixed-point represenation) . Implement Pipelined CORDIC and compare it with the conventional one and showing the trade-offs.

5.References :

- Computer Arithmetic: Algorithms and Hardware Designs 2nd Edition Behrooz Parhami
- CORDIC Algorithm Implementation By Ken Leonard V. Aquin : link
- The Birth of CORDIC : link