1 Introduction

1.1 Referenced documents

[1] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *in IRE Transactions on Electronic Computers, vol. EC-8, no. 3, pp. 330-334, Sept. 1959, doi: 0.1109/TEC.1959.5222693.*

[2] E. O. Garcia, R. Cumplido and M. Arias, "Pipelined CORDIC Design on FPGA for a Digital Sine and Cosine Waves Generator," *2006 3rd International Conference on Electrical and Electronics Engineering, Veracruz, Mexico, 2006, pp. 1-4, doi: 10.1109/ICEEE.2006.251917.*

[3] Sharat, Kavya, B. V. Uma, and D. M. Sagar. "Calculation of Sine and Cosine of an Angle using the CORDIC Algorithm." *International Journal of Innovative Technology and Research (IJITR).–February–March* (2014): 891-895.

[4] Kim, Young-Man. "Error Analysis of CORDIC Processor with FPGA Implementation." *arXiv preprint arXiv:2308.01025* (2023).

1.2 Design library name

Path: /afs/iitd.ac.in/user/j/jv/jvl232241/umc65/PD_flow/Cordic

1.3 People involved in the block

Mudit Bajaj	JVL232239
Tanmay Aron	JVL232240
Madhur Aswani	JVL232241
Hasti Kasundra	JVY237559

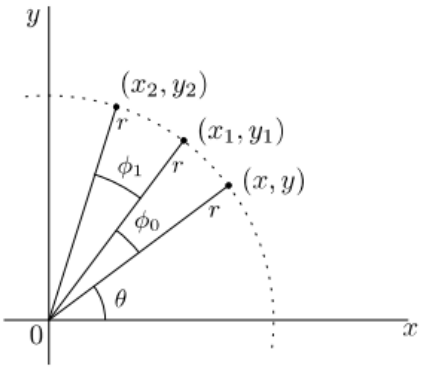
2 Function

2.1 Brief overview

- **CORDIC**: **CO**ordinate **R**otation **DI**gital **C**omputer.
- It is an iterative algorithm used for various trigonometric and other mathematical computations, particularly for calculating sine, cosine, and vector rotation.
- It operates by performing a series of rotations to gradually transform an initial value to the desired result.
- All iterations are performed in parallel, using a pipelined structure. This generally produces 1 additional bit of accuracy for each iteration.
- It is a hardware efficient algorithm as it uses only shift-add arithmetic.
- It contains the arctan table (LUT) for each iteration and the logic needed to manipulate the x, y and z values.

Basic Principle:

- Rotate (1,0) by Φ degrees to get **(x,y): $x = \cos\Phi$ and $y = \sin\Phi$.**
- Rotation of any (x,y) vector:
  $x' = x.\cos\Phi - y.\sin\Phi$
  $y' = y.\cos\Phi + x.\sin\Phi$
- Rearranging, we get:
  $x' = \cos\Phi.[x - y.\tan\Phi]$
  $y' = \cos\Phi.[y + x.\tan\Phi]$
- Can compute rotation of Φ in steps where each step size is of:
  $\tan\Phi = \pm 2^{-i}$



- Gain: K = π.cosΦ depends on rotation angle Φ1, Φ2, Φ3,….., Φn.
- Since same angles are rotated always, K is a constant which is pre-computed (K = 1.64676)

Iterative Rotations:

$$x_{i+1} = K_i\,(x_i - (y_i\,d_i\,2^{-i}))$$
$$y_{i+1} = K_i\,(y_i + (x_i\,d_i\,2^{-i}))$$

where, d_i: decision (rotation mode)

z_i is introduced to keep track of angle that has been rotated ($z_0 = \Phi$)

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

$$d_i = -1 \text{ if } z_i < 0$$
$$1 \text{ otherwise}$$

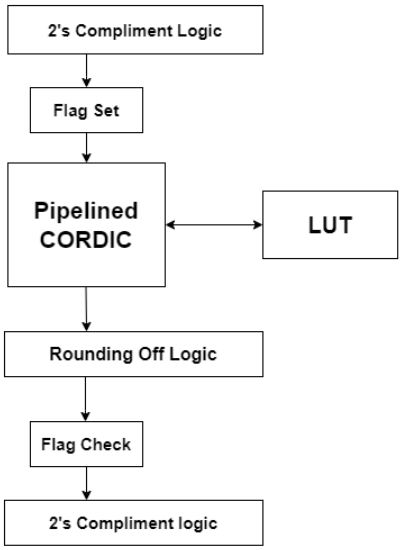
The purpose of this design is to do a RTL to GDS flow for a 8-bit dedicated processor for calculating sine and cosine of an angle using the CORDIC algorithm. The specifications are mentioned below:

Specifications	
Maximum Frequency	100MHz
Input	8-bit Angle input (in radians)
Output	8-bit Sine & Cosine
Error (%)	1%
Technology	umc65

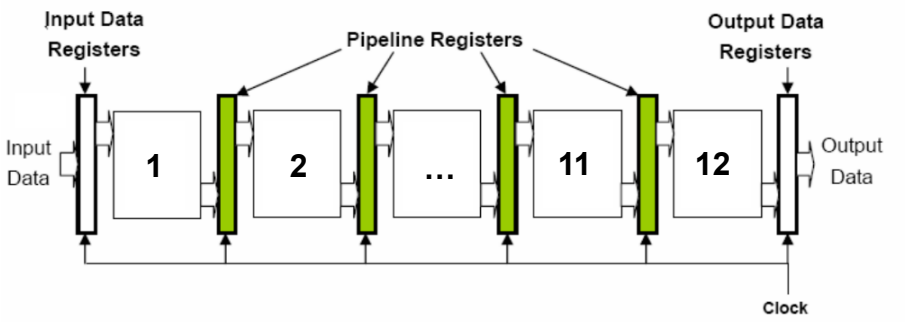
2.2 Interfaces

Signal name	I/O	Description	Logical grouping
clk	input	Clock input	-
rst	input	Reset input	-
[7:0] in	input	Signed 8-bit Angle in Radians	-
[7:0] sine	output	Signed 8-bit Sine of the given angle	-
[7:0] cosine	output	Signed 8-bit Cosine of the given angle	-

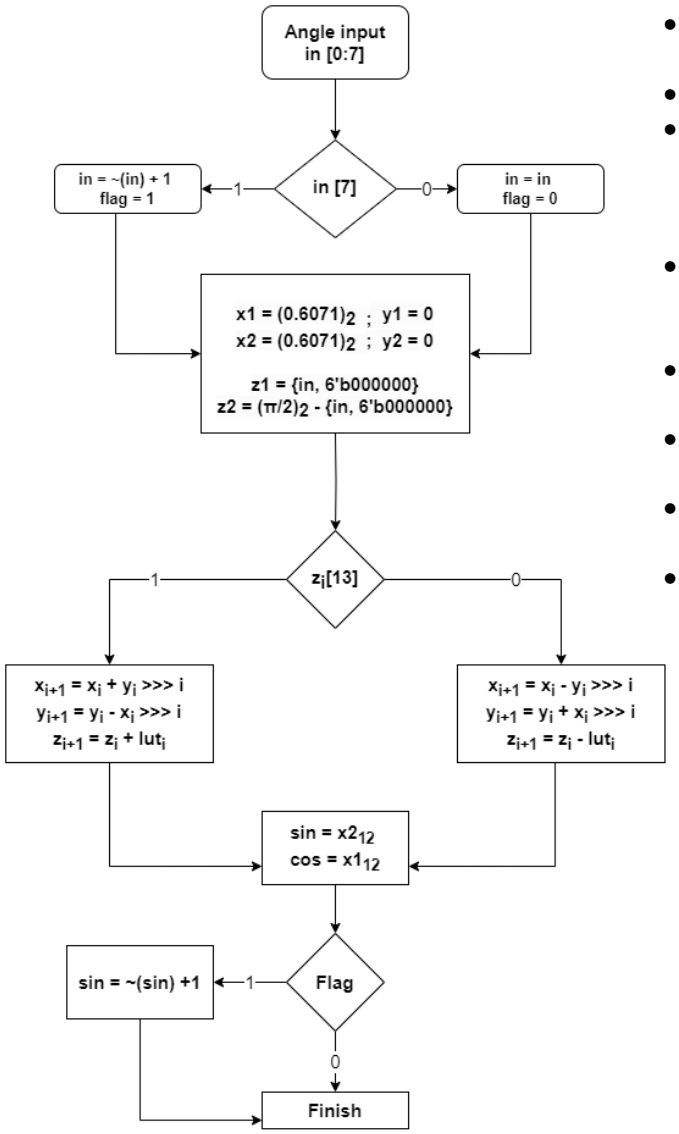
2.3 Architecture



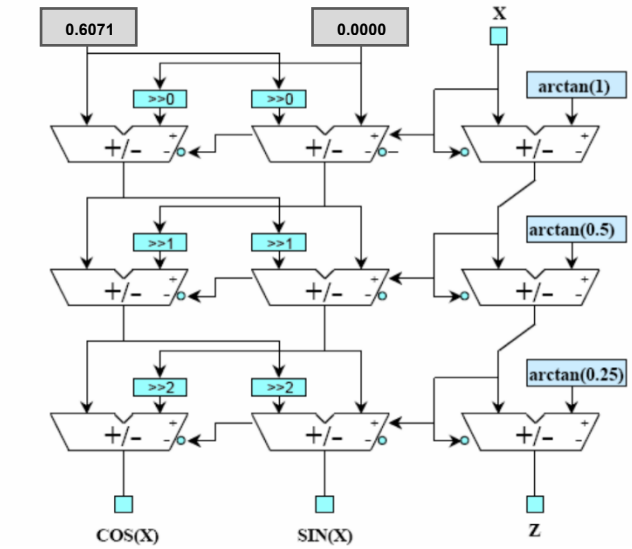
Pipelined CORDIC



2.4 Detailed functional description



- Input signal: 8-bit angle in the range: -π/2 to +π/2 radians.
- First bit of angle input is the sign bit. If sign bit =1, i.e. if the angle is in 4th quadrant (-π/2 to 0), we take 2's compliment.
- Input vectors x & y and phase angle z are initialized.
- Performs iterations for 12 pipelined stages where in each iteration, the algorithm aims to rotate the vector (x,y) towards the positive x-axis and the current angle is either added or subtracted from the phase angle z.
- Here, we consider Φ to compute cosine and (π/2 – Φ) to compute sine of the given angle Φ, in order to achieve higher accuracy.
- The rotation angles are precomputed and stored in a lookup table (LUT) which contains arctan values of
- As the iterations progress, the vector (x,y) converges towards a point that corresponds to the given phase angle z.
- The final stage input vectors represent the cosine and sine components of the input angle.
- The computed 14-bit sine and cosine are rounded off to 8 bits to get the final output.



3 Design parameters

3.1 Performance Requirements

- 8-bit dedicated Cordic processor
- Maximum Frequency: 100MHz
- Output: 8-bit Sine and Cosine for the given angle in radians
- Error(%) = 1%

3.2 Clock Distribution

- Clock period: 10ns
- Rise & Fall time: 0.1ns (1% of clock period)
- Source Latency: 0.1ns
- Clock uncertainity: 0.1ns
- Max Capacitance: 4pF
- Max Fanout: 150

3.3 Reset

- Reset is asynchronous and active low.

3.4 Timing Description

- Clock period = 10ns
- Pipeline stages = 12
- Latency = 15 clock cycles
- Post genus synthesis, maximum slack of 7.511ns is observed.
- The preCTS timing and postCTS timing is reported.

4 Verification Strategy

4.1 Objectives

- The objective is to validate the Verilog sine and cosine outputs corresponding to a specified angle in radians. This involves comparing the Verilog results with the true sine and cosine values and computing the percentage error.

4.2 Tools and Version

- Verilog (Modelsim- Intel FPGA Edition vsim 2020.1)
- Cadence Genus & Innovus

4.3 Checking mechanisms

- **Self-checking mechanism**: A testbench is developed in verilog that instantiates the CORDIC module and feeds it various input angles. This verilog generated output is subsequently compared with the actual sine and cosine values, and the error percentage is computed.

5 Functional Checklist

The basic functionality of this design is to compute sine & cosine values for the given angle input. To check this, we need to verify the following:

- If the module correctly handles positive & negative input angles in the range -π/2 to +π/2.
- LUT is correctly implemented is appropriately sized to balance accuracy and resource utilization.
- Proper scaling and rounding that is applied to convert internal representations to the final output format of 8-bits.
- The algorithm meets the required performance requirements.
- Verify the final verilog sine and cosine output and compare with the true values to fine the error %.

5 Testbench

5.1 Overview

- 2 testbenches are designed to test the functionality of CORDIC algorithm which monitors the sine and cosine outputs of the CORDIC module
  - The initial testbench focuses on functional verification of the CORDIC algorithm at critical angles such as 0, ±π/4, ±π/2, aiming to quantify the error percentage at these critical points. This testbench also verifies the congruence of input and output frequencies by observing the output of the pipelined stage at each cycle with a latency of 15 clock cycles.
  - A separate testbench has been designed to analyze the error percentage across the complete range of angles, spanning from -π/2 to +π/2.

5.2 Architecture

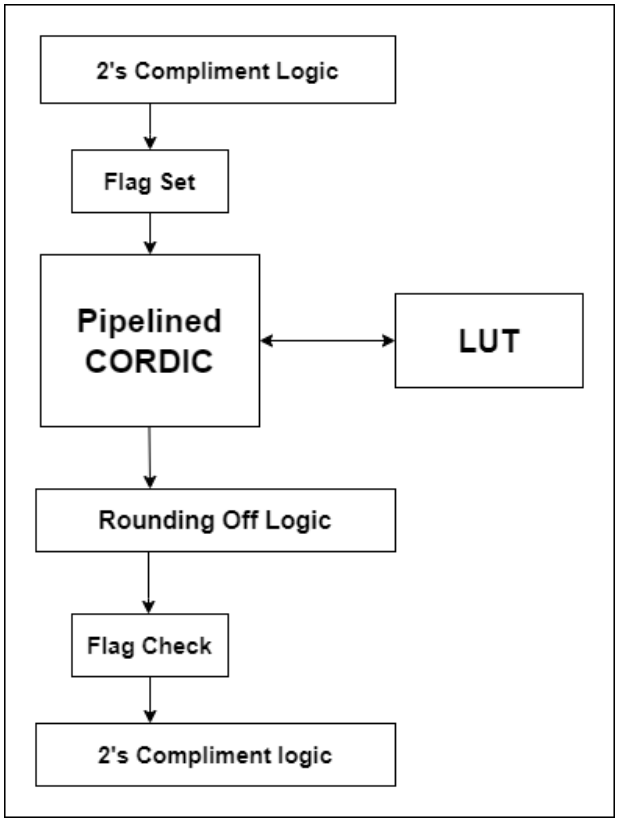
- Clock and other signals are initialized. Time period is set to 10ns.
- Sizing factor for input angle and for generating true sine & cosine values of that angle is sf = 2^{-6}.
- Sizing factor for output is o_sf = 2^{-7}.
- Information regarding cycle information, cosine, sine, input angle, and actual sine and cosine values is displayed.

6 Tests Specification

- Angle input: Testbenches are designed to check the CORDIC functionality at critical angles including 0, ±π/4, and ±π/2 and the entire spectrum of input angles ranging from -π/2 to +π/2.
- Input clock with period of 10ns and rise & fall time of 0.1ns (1% of Clock period) provided.
- The test cases and corresponding output results have been reported in section: 10.3.

7 Design Microarchitecture

7.1 Top Level Interface



7.2 Sub-Block Description

- **2's Compliment Logic:** When the sign bit is set to 1, we invert the input and pass the resulting value to BEC-1 (Binary Excess Code-1) logic. This approach allows the synthesizer to generate more efficient logic for incrementing the least significant bit (LSB) by 1 compared to employing a ripple carry adder. Conversely, when the sign bit is 0, the input value remains unaltered and is directly passed to the output.

- **Flag Set:** When the sign-bit is 1, we set the flag=1 to indicate that the angle is negative (-π/2 to 0). The flag is set to 0 in other case.

- **Pipelined CORDIC:** The CORDIC computation is divided into 12 pipelined stages, where the output of one stage feeds directly into the next stage. The input angle is fed into the pipeline. Initial values for x, y, and z are set. Each stage rotates the vector (x,y) by a precomputed angle. The rotation angle is determined based on the iteration and is part of a lookup table. The rotated values are passed to the next stage. Registers are introduced between stages to store intermediate results. The rotated angle is accumulated in z. The final values of input vectors represent the cosine and sine of the original input angle, respectively

- **LUT:** It is used to store pre-computed values of rotation angles which is looked-up and used for performing shift and add operation of Cordic algorithm. The LUT of this design stores 14-bit values for $\tan^{-1}(1)$, $\tan^{-1}(1/2)$,… upto $\tan^{-1}(1/2048)$. These binary values have been computed using python.

- **Rounding Off Logic:** Our 8-bit output comprises of 1 sign bit and 7 fractional bits. When approaching values close to 1, introducing an additional one, as done during rounding up, becomes impractical as it will change the sign of the output. The rounding process selectively applies only to values that retain their signs post-rounding. Here, the 13th bit of our 12-bit x register is used as a case to check whether to round of the output or not.

- **Flag Check:** This stage verifies whether the flag was set to 1 or 0. If the flag was set to 1, it indicates that the input angle falls within the range of -π/2 to 0. Consequently, sine values for this angle are negative. Therefore, in this scenario, the 2's complement of the sine is computed.

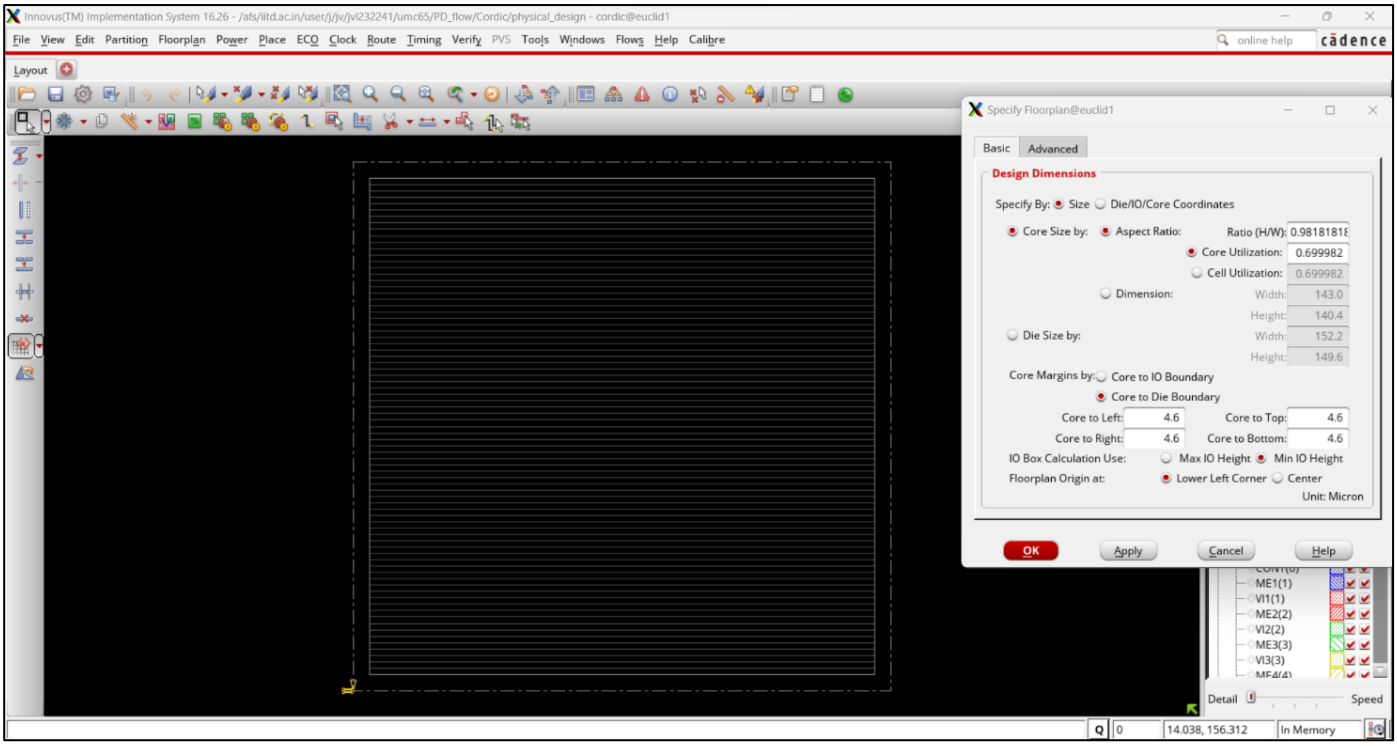
7.3 Structural Mapping Process

Cadence Genus is primarily used for synthesis, while Innovus is employed for place and route along with other physical design steps. This flow ensures a comprehensive transformation from high-level RTL description to a layout ready for manufacturing, addressing considerations such as timing, power, and physical design rules. Following steps are involved in Structural Mapping Process:

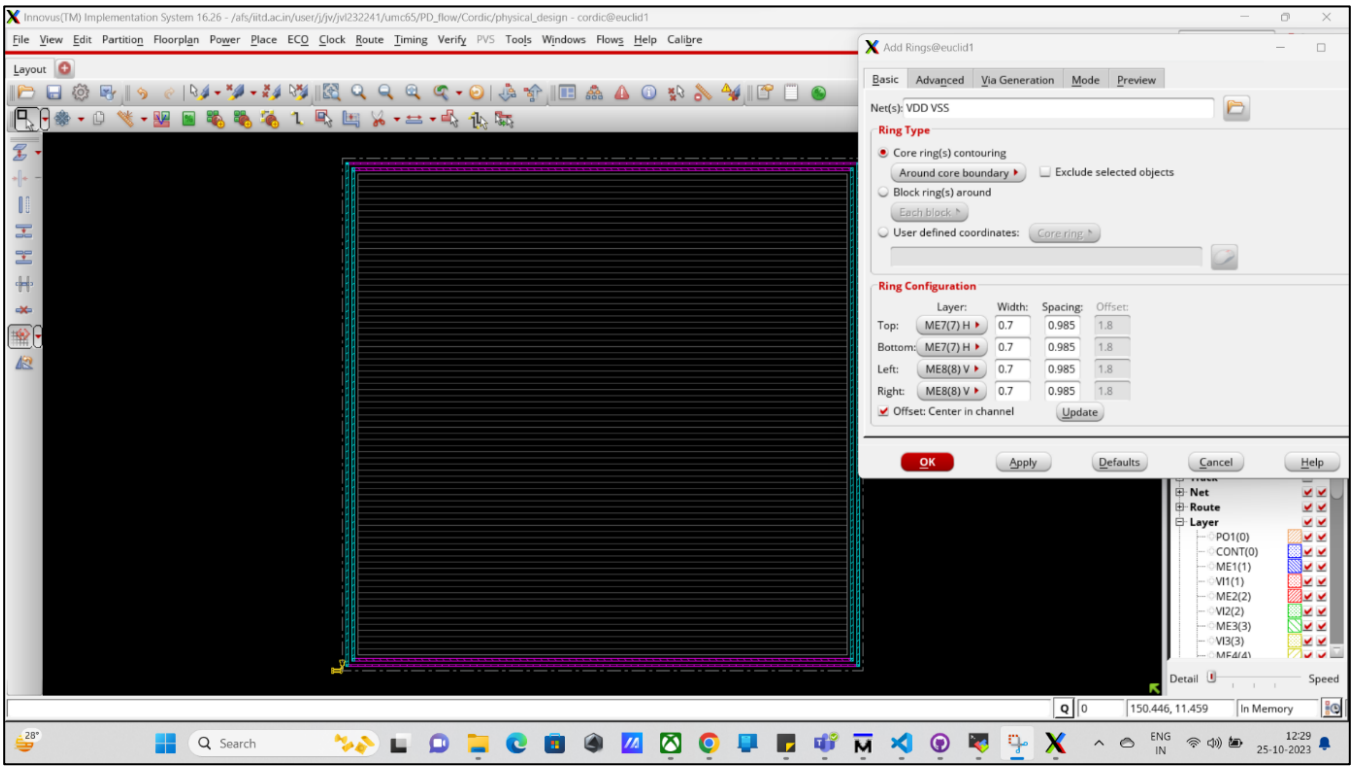
- RTL Synthesis (Genus): Translating the RTL code (Verilog) into a gate-level netlist
- Floorplanning (Innovus)
- Place and Route (Innovus)
- Clock Tree Synthesis (Innovus)
- Static Timing Analysis (Innovus)
- Physical Verification (Innovus)

8 Physical hierarchy

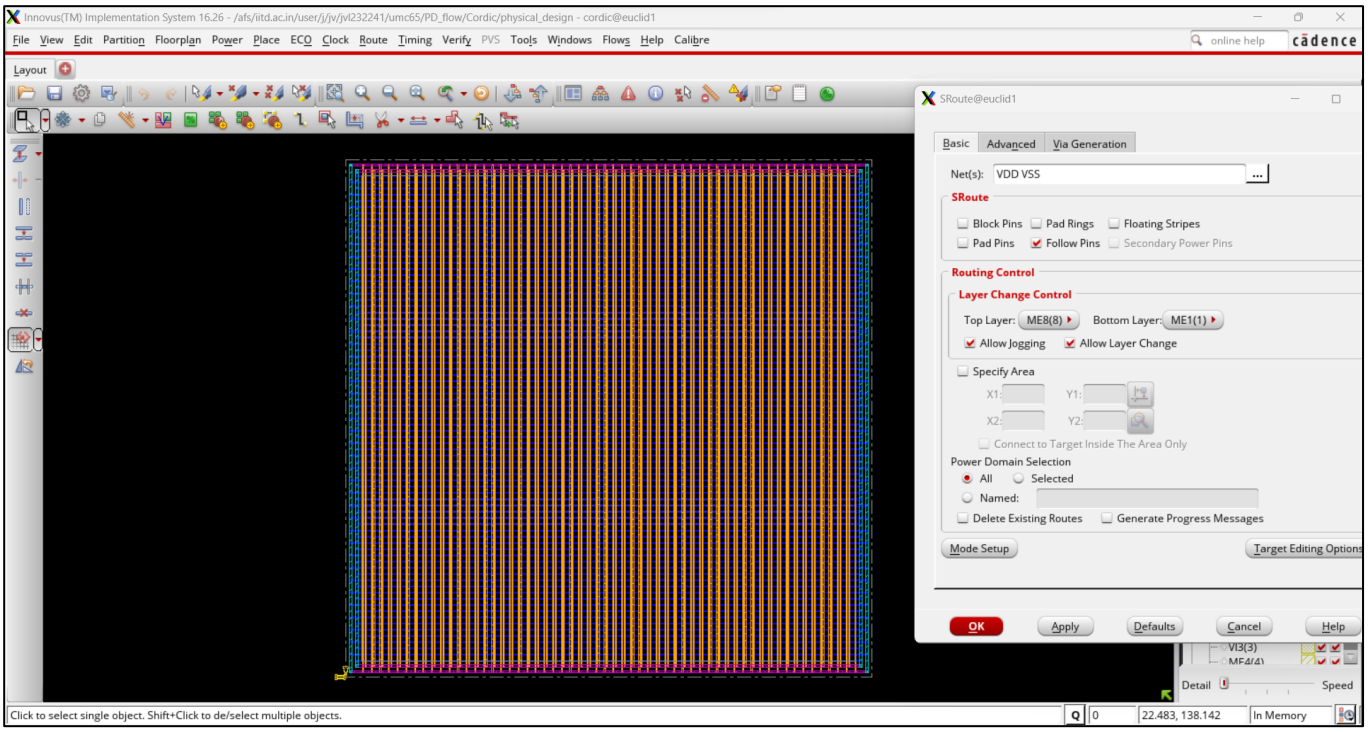
8.1 Floorplanning



Initial Floor Plan



Power Rings Added

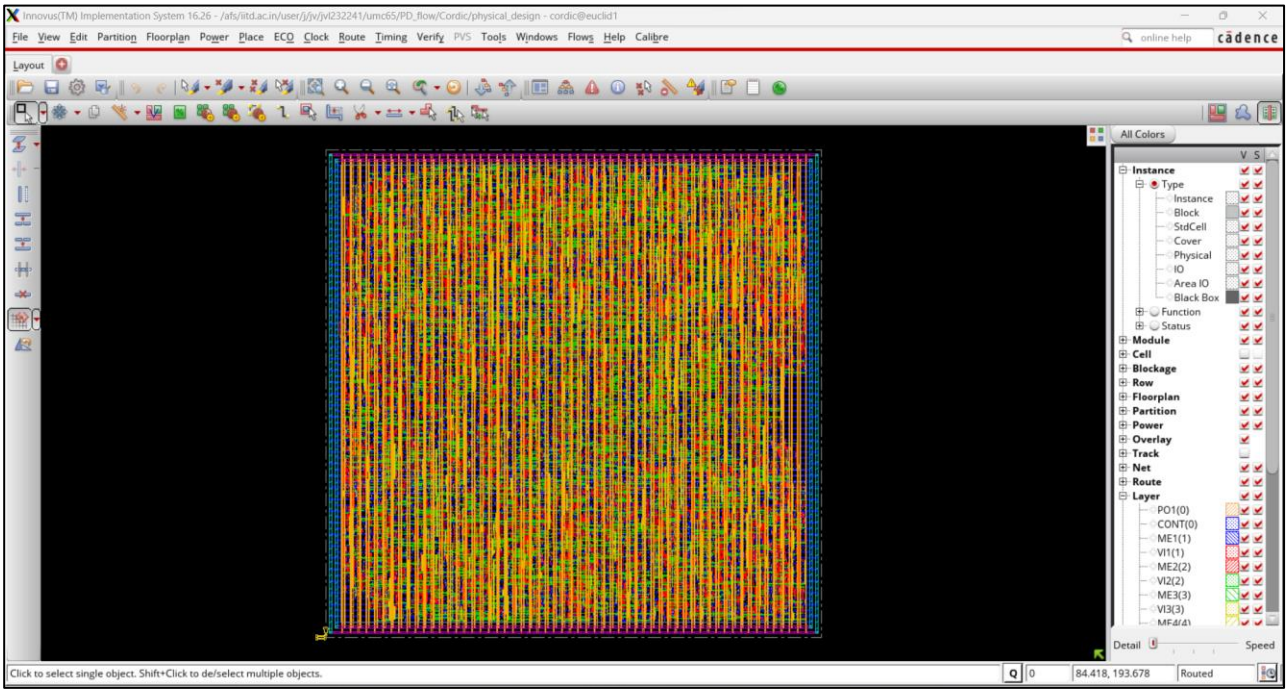


Power Planning

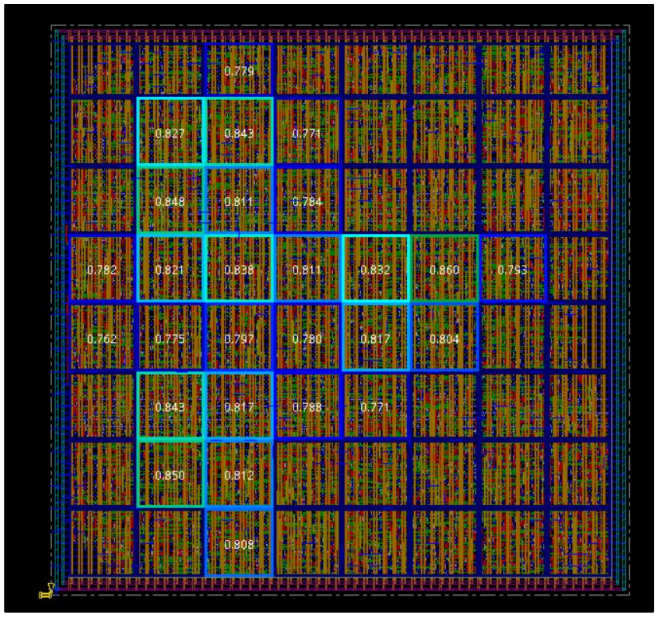
8.2 Clock-tree insertion

```
set sdc_version 1.7

set_units -capacitance 1000.0fF
set_units -time 1ns

# Set the current design
current_design cordic

create_clock -name "clk" -add -period 10 -waveform {0 5} [get_ports "clk"]

set_clock_transition -rise 0.1 [get_clocks "clk"]
set_clock_transition -fall 0.1 [get_clocks "clk"]
set_clock_latency -source 0.1 [get_clocks "clk"]

set_max_capacitance 4 [all_inputs]

set_max_fanout 150 [all_inputs]

set_clock_uncertainty 0.1 [get_clocks "clk"]

set_input_delay -clock [get_clocks "clk"] -add_delay 0.05 [get_ports "rst"]

set_wire_load_mode "top"
```

8.3 Layout Strategy



Final Layout



Hotspot Congestion

9 Results

9.1 Area

- Area = 14363.28 μm^2
- Gate Count = 13299

```
innovus 1> report_area
Depth  Name       #Inst   Area (um^2)
------------------------------------
0      cordic     3251    14363.28
1
```

```
Gate area 1.0800 um^2
[0] cordic Gates=13299 Cells=3251 Area=14363.3 um^2
*** Statistics for net list cordic ***
Number of cells      = 3251
Number of nets       = 4006
Number of tri-nets   = 0
Number of degen nets = 0
Number of pins       = 12288
Number of i/os       = 26
```

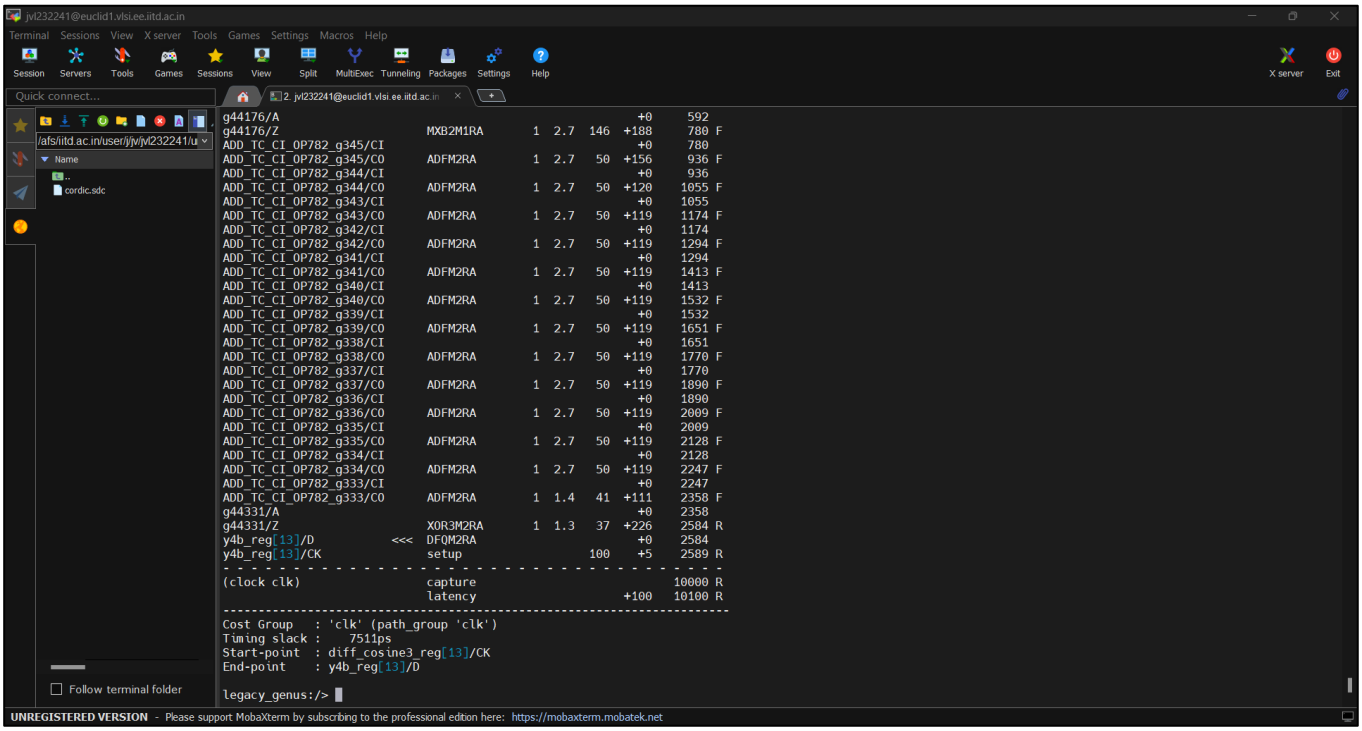
Power:

```
Total Power
-------------------------------------------------------------------------------
Total Internal Power:        0.62984682            69.7898%
Total Switching Power:       0.26986422            29.9021%
Total Leakage Power:         0.00278076             0.3081%
Total Power:                 0.90249179
-------------------------------------------------------------------------------


Group                          Internal    Switching     Leakage       Total  Percentage
                               Power       Power         Power         Power  (%)
-------------------------------------------------------------------------------
Sequential                       0.4509      0.03604    0.001137      0.4881       54.08
Macro                                 0            0           0           0           0
IO                                    0            0           0           0           0
Combinational                    0.1373      0.1414     0.00152      0.2803       31.06
Clock (Combinational)           0.04162     0.09238   0.0001237      0.1341       14.86
Clock (Sequential)                    0            0           0           0           0
-------------------------------------------------------------------------------
Total                            0.6298      0.2699    0.002781      0.9025         100
-------------------------------------------------------------------------------
```

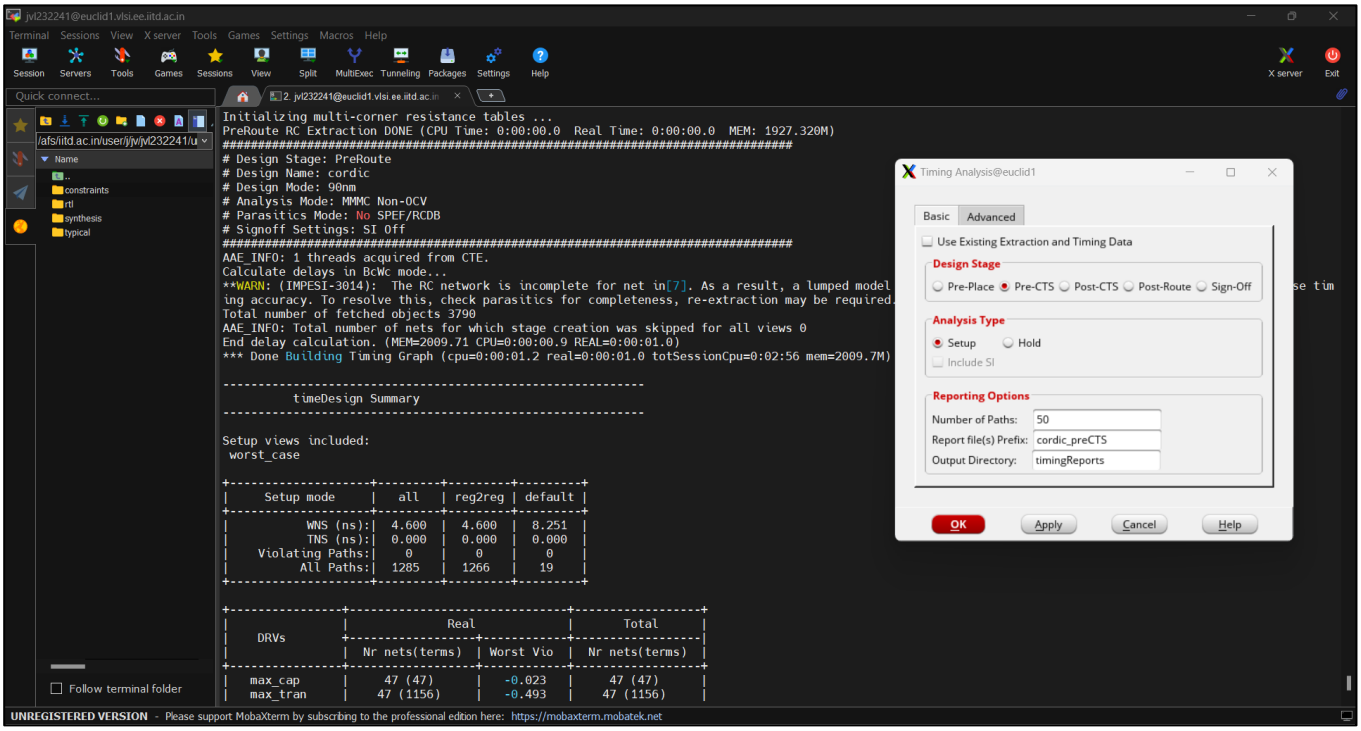
10.2 Timing

Post synthesis slack = 7.511ns

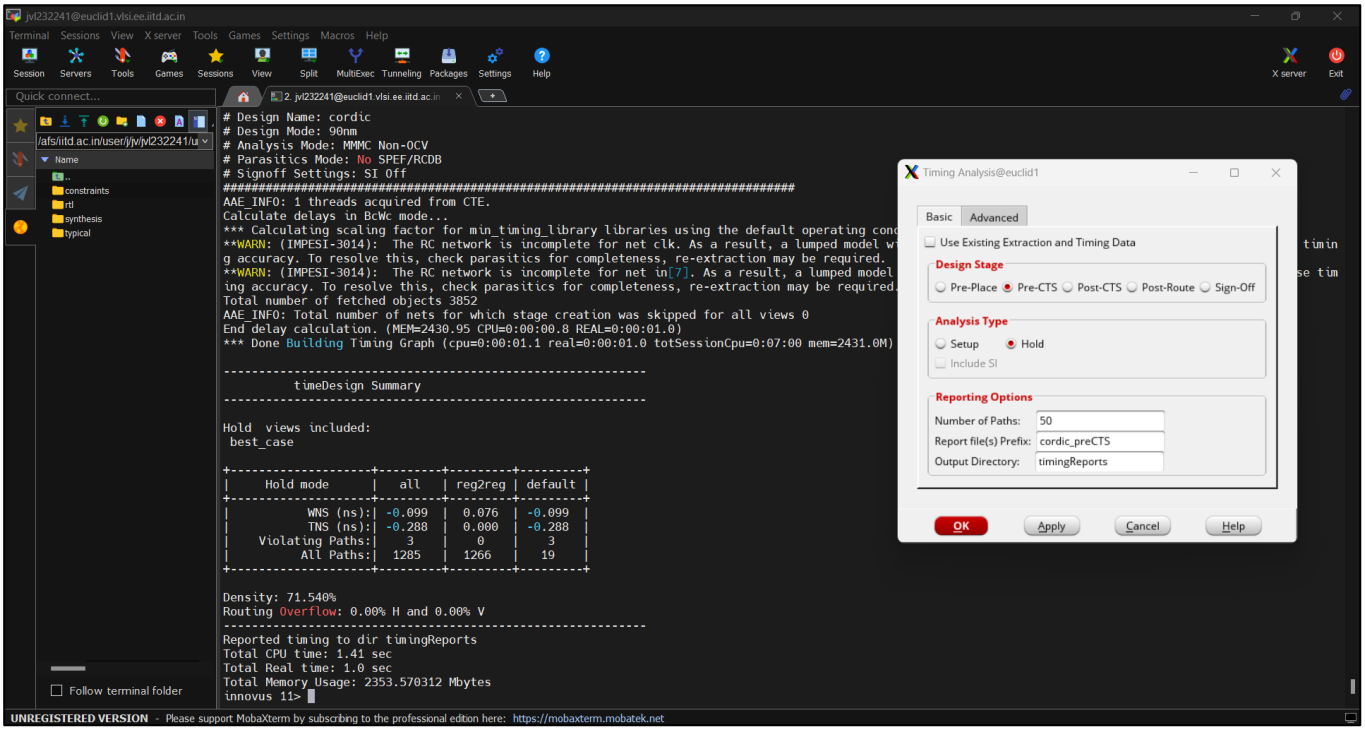
Timing	Worst Negative Slack			Total Negative Slack		
	All	Reg2Reg	Default	All	Reg2Reg	Default
Pre-CTS Setup	4.600 ns	4.600 ns	8.215 ns	0	0	0
Pre-CTS Hold	-0.099 ns	0.076 ns	-0.099 ns	-0.288 ns	0	-0.288 ns
Post-CTS Setup	5.123 ns	5.123 ns	7.547 ns	0	0	0
Post-CTS Hold	-0.126 ns	0.073 ns	-0.126 ns	-0.368 ns	0	-0.368 ns



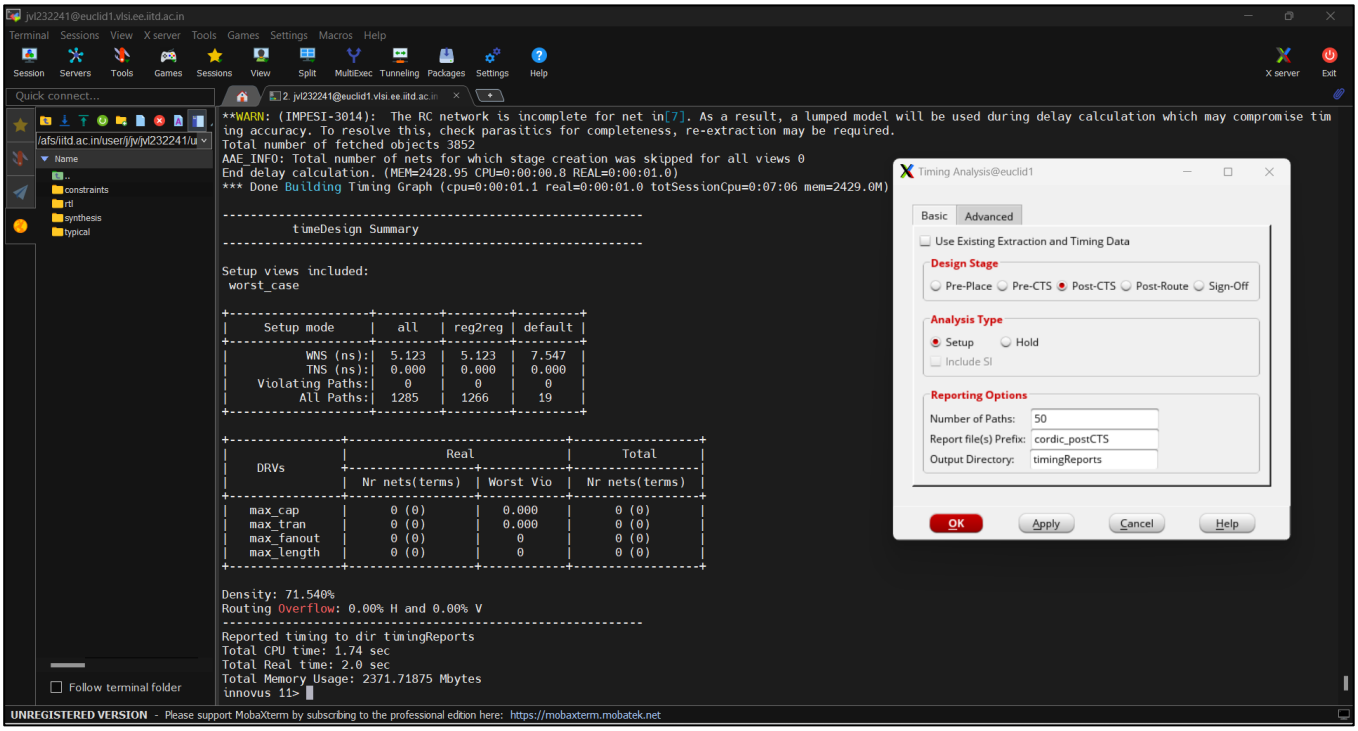
Post Synthesis Slack



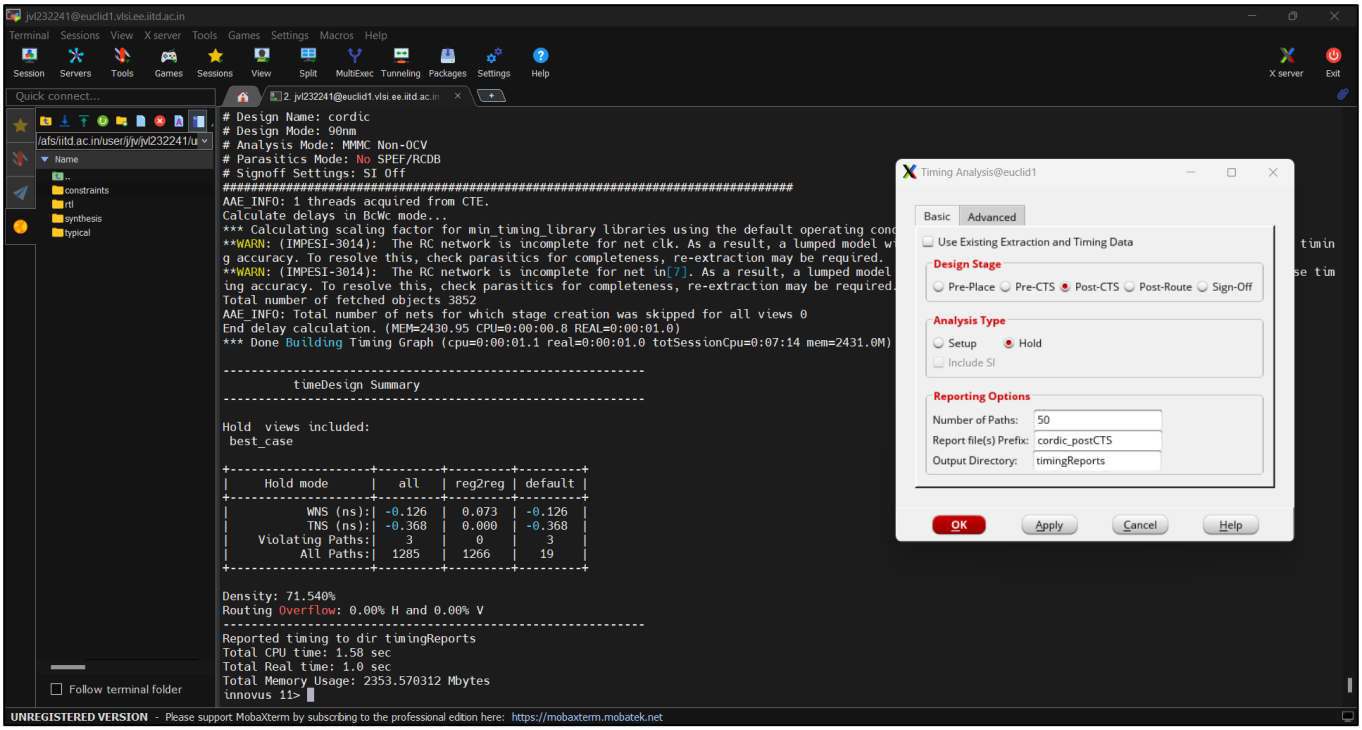
Pre-CTS Setup Time



Pre-CTS Hold Time



Post-CTS Setup Time



Post-CTS Hold Time

10.3 Testability analysis

The pipelined CORDIC has been tested with various input scenarios to validate its performance. The following table shows the outcomes of a testbench incorporating specific angle inputs, including crucial angles like 0, ±π/4, and ±π/2. It is observed that the output frequency matches the input frequency, as that the output is generated after each cycle, with a latency of 15 clock cycles.

Input	Sine			Cosine		
Angle (Radians)	Output	Actual	Error %	Output	Actual	Error %
0.218750	0.218750	0.217010	0.80%	0.968750	0.976169	0.76%
-1.531250	-0.992188	-0.999218	0.70%	0.039063	0.039536	1.19%
1.531250	0.992188	0.999218	0.70%	0.039063	0.039536	1.19%
0.000000	-0.007813	0.000000	-	0.992188	1.000000	0.78%
0.781250	0.703125	0.704168	0.14%	0.703125	0.710034	0.97%
-0.781250	-0.703125	0.704168	0.14%	0.703125	0.710034	0.97%

```
# Cycle:   0   Cosine: -0.000000   Sine: -0.000000   angle: -0.000000   actual_sine: -0.000000   actual_cosine: 1.000000
#
# Cycle:   0   Cosine: -0.000000   Sine: -0.000000   angle: 0.218750   actual_sine: 0.217010   actual_cosine: 0.976169
#
# Cycle:   1   Cosine: -0.000000   Sine: -0.000000   angle: -1.531250   actual_sine: -0.999218   actual_cosine: 0.039536
#
# Cycle:   2   Cosine: -0.000000   Sine: -0.000000   angle: 1.531250   actual_sine: 0.999218   actual_cosine: 0.039536
#
# Cycle:   3   Cosine: -0.000000   Sine: -0.000000   angle: 0.000000   actual_sine: 0.000000   actual_cosine: 1.000000
#
# Cycle:   4   Cosine: -0.000000   Sine: -0.000000   angle: 0.781250   actual_sine: 0.704168   actual_cosine: 0.710034
#
# Cycle:   5   Cosine: -0.000000   Sine: -0.000000   angle: -0.781250   actual_sine: -0.704168   actual_cosine: 0.710034
#
# Cycle:  15   Cosine: 0.968750   Sine: 0.218750   angle: -0.781250   actual_sine: -0.704168   actual_cosine: 0.710034
#
# Cycle:  16   Cosine: 0.039063   Sine: -0.992188   angle: -0.781250   actual_sine: -0.704168   actual_cosine: 0.710034
#
# Cycle:  17   Cosine: 0.039063   Sine: 0.992188   angle: -0.781250   actual_sine: -0.704168   actual_cosine: 0.710034
#
# Cycle:  18   Cosine: 0.992188   Sine: -0.007813   angle: -0.781250   actual_sine: -0.704168   actual_cosine: 0.710034
#
# Cycle:  19   Cosine: 0.703125   Sine: 0.703125   angle: -0.781250   actual_sine: -0.704168   actual_cosine: 0.710034
#
# Cycle:  20   Cosine: 0.703125   Sine: -0.703125   angle: -0.781250   actual_sine: -0.704168   actual_cosine: 0.710034
#
```

Testbench Output

Error Calculation is done using following formula:

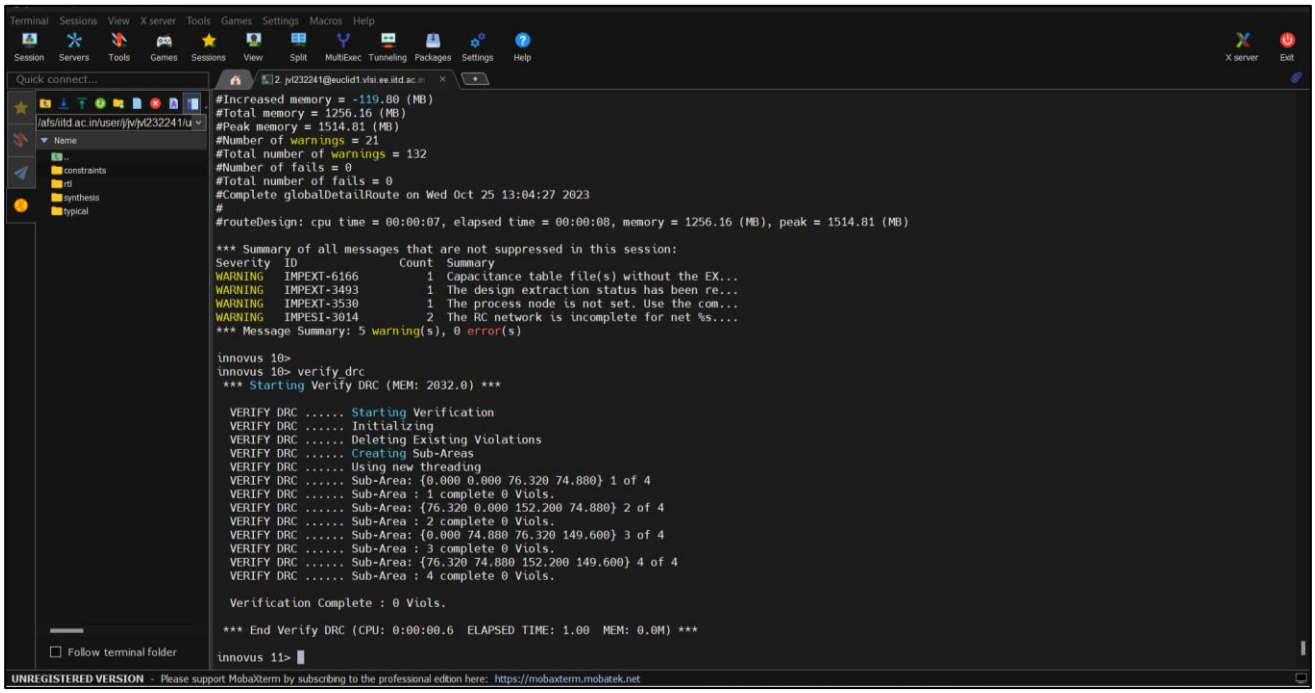
$$\%\ \text{Error (Sine)} = \left|\frac{\textit{Sine Output} - \textit{Actual Sine}}{\textit{Actual Sine}}\right| * 100\%$$

$$\%\ \text{Error (Cosine)} = \left|\frac{\textit{Cosine Output} - \textit{Actual Cosine}}{\textit{Actual Cosine}}\right| * 100\%$$

Another testbench has been developed to check the error across the complete spectrum of angle inputs, ranging from -1.53125 to +1.53125 radians. Following plot illustrates the % error between the computed sine and cosine outputs and their actual values across input angles ranging from -π/2 to +π/2.

Average Sine Error = 0.54% Average Cosine Error% = 0.51%

10.4 DRC rule violations



DRC cleared

11 Bugs known at submission date

None.