VLSI PROJECT

Low Power 8 × 8 Bit Multiplier Design using Dadda Algorithm

Semi-custom design using cadence

Project done by :

Pathipati Bharath kumar - 523ec0006

Jonnalagadda Dinesh - 523ec0007

Submitted to:

Dr . P . Rangababu sir

Introduction

Multiplication is an important operation in digital circuits and is widely used in areas like Digital Signal Processing (DSP), image and video processing, communication systems, and high-performance computing. The speed and power efficiency of a digital system depends greatly on how fast and efficient its multiplier is. With growing demand for faster and low-power electronic devices, designing power-efficient and high-speed multipliers has become very important in VLSI design.

The Dadda multiplier is a fast parallel multiplier architecture used to improve speed and reduce hardware usage. It works by reducing partial products in a structured way using fewer adders, which makes it faster and more efficient than conventional array multipliers. It achieves high performance with reduced area and power consumption.

In this project, a **low-power 8×8 bit multiplier using the Dadda algorithm** is designed and implemented. The aim is to reduce power consumption and delay while maintaining accuracy. The design includes generating partial products, reducing them using Dadda stages, and finally adding the results using a fast adder. The performance of the proposed design is compared with other multiplier architectures to show that it is suitable for low-power and high-speed applications like portable devices, IoT systems, and embedded processors.

Importance / Why it is Helpful

- **Low Power Consumption**
  Modern VLSI applications demand energy-efficient arithmetic units. Dadda multipliers reduce the number of logic stages and switching activity, thereby lowering dynamic power dissipation.
- **High Speed Operation**
  Faster reduction of partial products and fewer adder levels improve the overall multiplication speed. This is crucial for real-time systems and high-performance processors.
- **Area Efficient**
  Dadda's structured reduction requires fewer hardware resources compared to other tree-based multipliers, making it suitable for compact hardware implementations like ASICs and FPGAs.
- **Scalable and Suitable for VLSI**
  The architecture can be easily extended to higher bit-width multipliers, making it ideal for scalable VLSI design in arithmetic units.
- **Applications in Modern Systems**
  Efficient multipliers are essential in image processing, cryptography, neural networks, IoT devices, and portable electronics, where performance and power efficiency are critical.

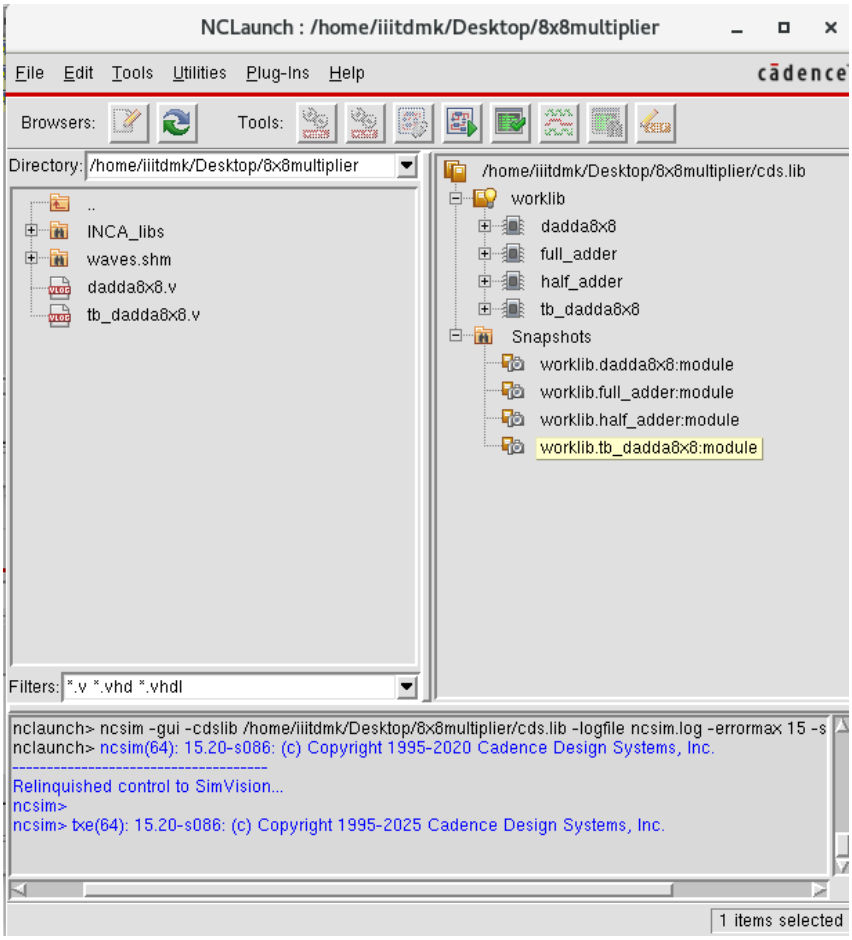
➔ Create new folder with specific name (we created with 8x8multiplier).
➔ Open terminal in that folder.
➔ Type csh → ENTER
➔ Type source /home/install/cshrc → ENTER
➔ Then it will open cadence tab in terminal , there we have to type gedit file_name.v → then it will open the text editor there we have to copy the code attached with this project .
➔ Repeat the same steps as follows. Use the attached code files.

gedit dadda8x8.v → ENTER → copy and paste the code
gedit tb_dadda8x8.v → ENTER → copy and paste the code
gedit constraints_input.sdc → ENTER → copy and paste the code
gedit run.tcl → ENTER → copy and paste the code

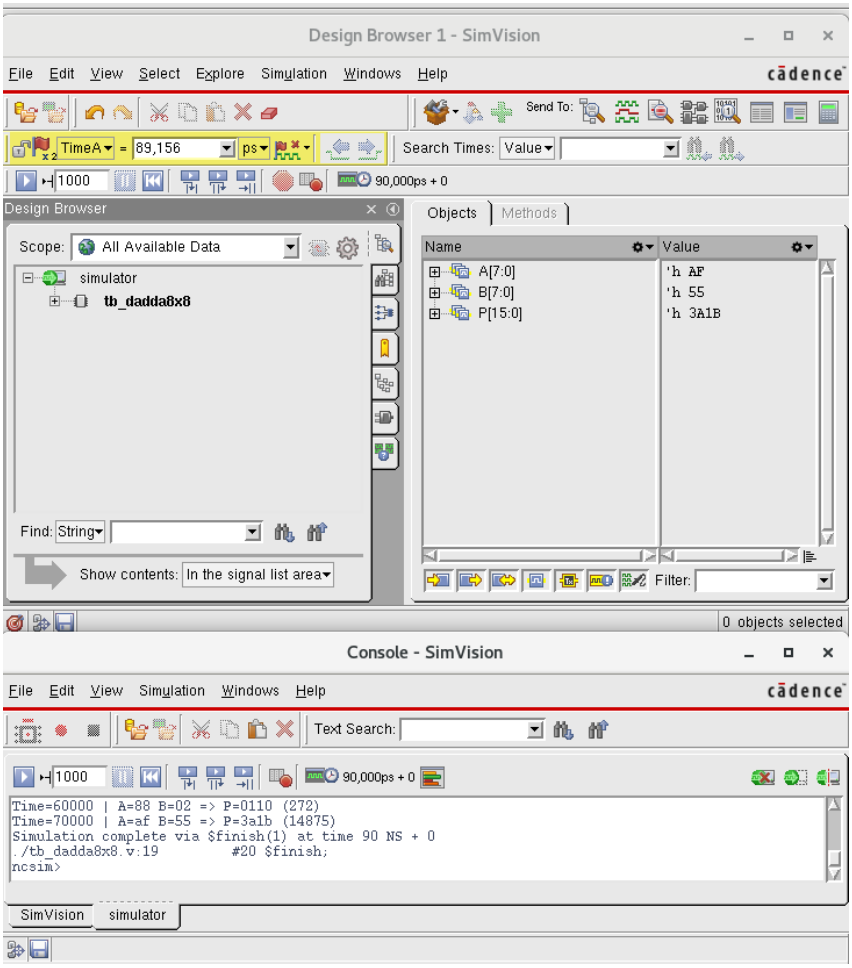
➔ Now , we have to launch the cadence tool by using **nclaunch -new**
➔ Click on options as per instructions below.

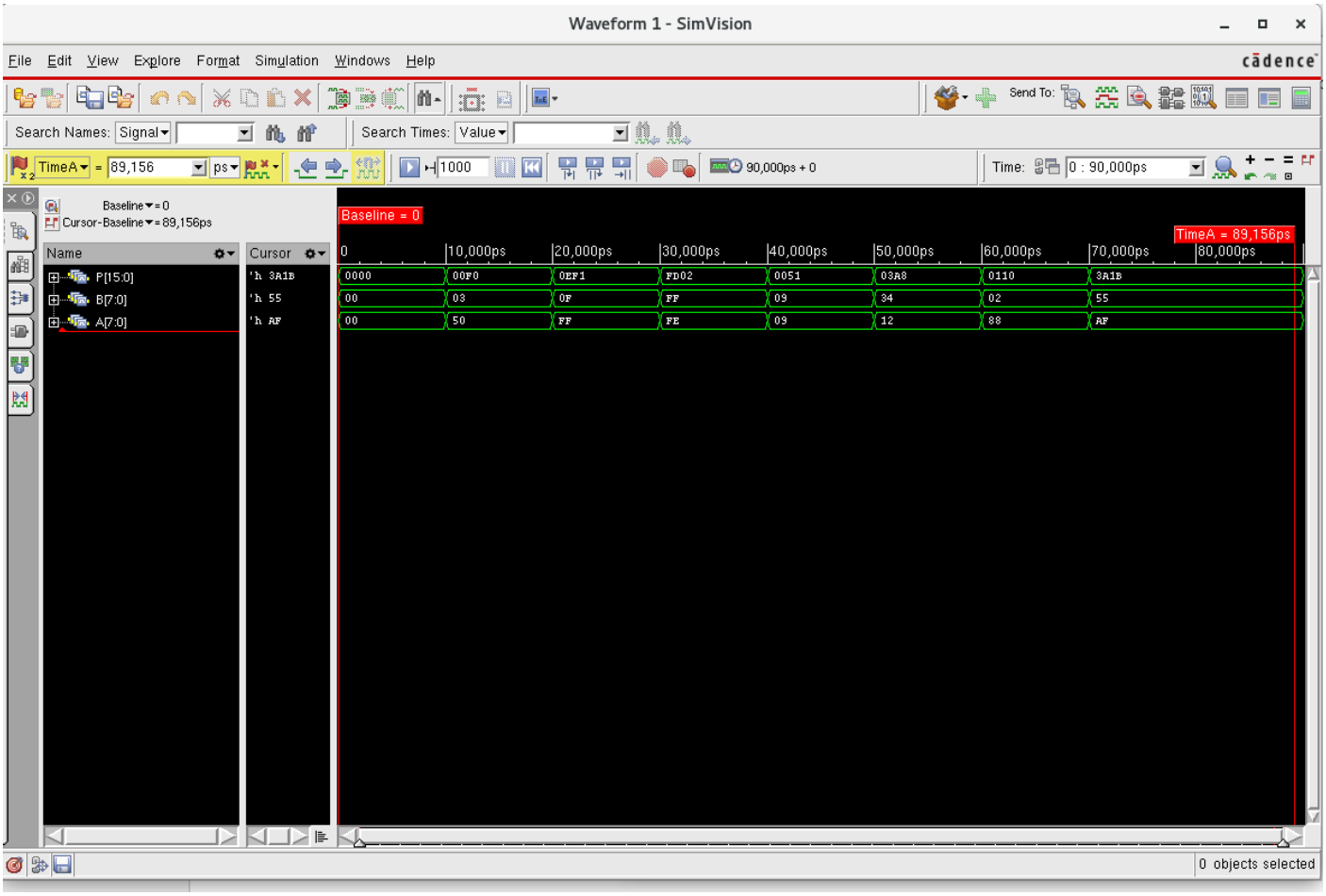
Multiple steps → Create cds.lib → don't include any lib →

➔ Based on the Libraries available and the type of RTL Code written, one of the three shown above is to be selected. Cadence tool suite provides default gpdk libraries. Here, counter RTL is of Verilog Format and hence third option is selected.
➔ A new pop-up "nclaunch" opens which will contain all the .v and .vhdl files as per the cds.lib file created.
➔ Functional Simulation using Cadence runs in 3 stages: → Compilation of Verilog/VHDL Code and/or Test Bench → Elaboration of the Code & Test Bench Compiled → Simulating the Test Bench or Top Module[in absence of Test Bench]
➔ A set of tools are shown in the nclaunch window which refer to VHDL Compiler, Verilog Compiler, Elaborator, Simulator corresponding from Left To Right.
➔ Select the .v or .vhdl files to be compiled and launch Compiler. On successful completion of compilation, on the Right hand Side, the modules appear under "Worklib" .
➔ Select the Module under Worklib and "Launch Elaborator" . On successful completion of Elaboration, "Snapshots" are generated.
➔ Select the Test Bench under snapshots and "Launch Simulator" .
➔ The above steps are depicted under following snapshosts.



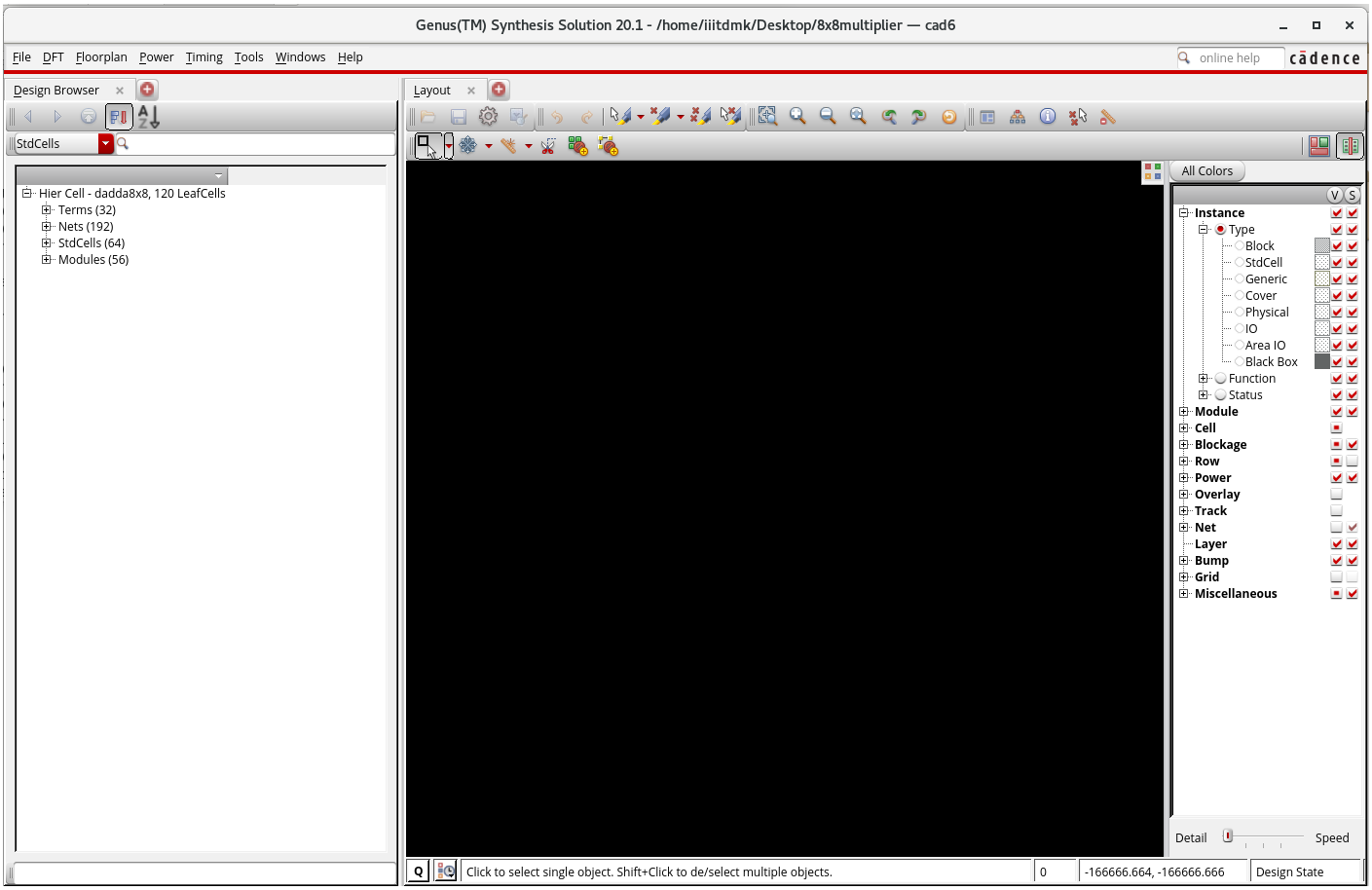
→ select all inputs and outputs →send to waveform



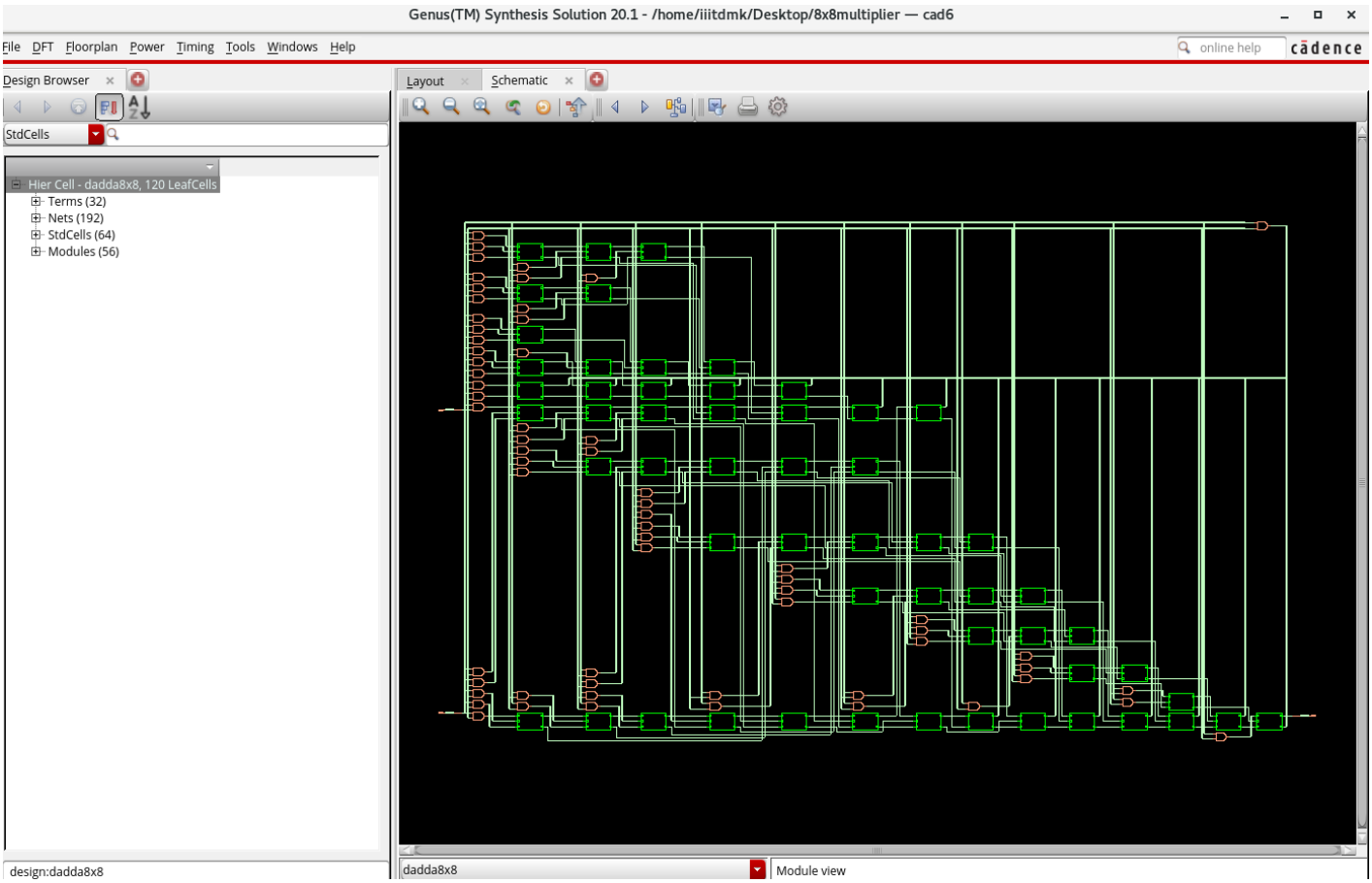


➔ Now we have to launch Genus so, we give the command in the terminal by typing **genus** in the terminal , then it will load the genus root .

Type **source run.tcl**



Double click on **Hier.cell** for **schematic view** .



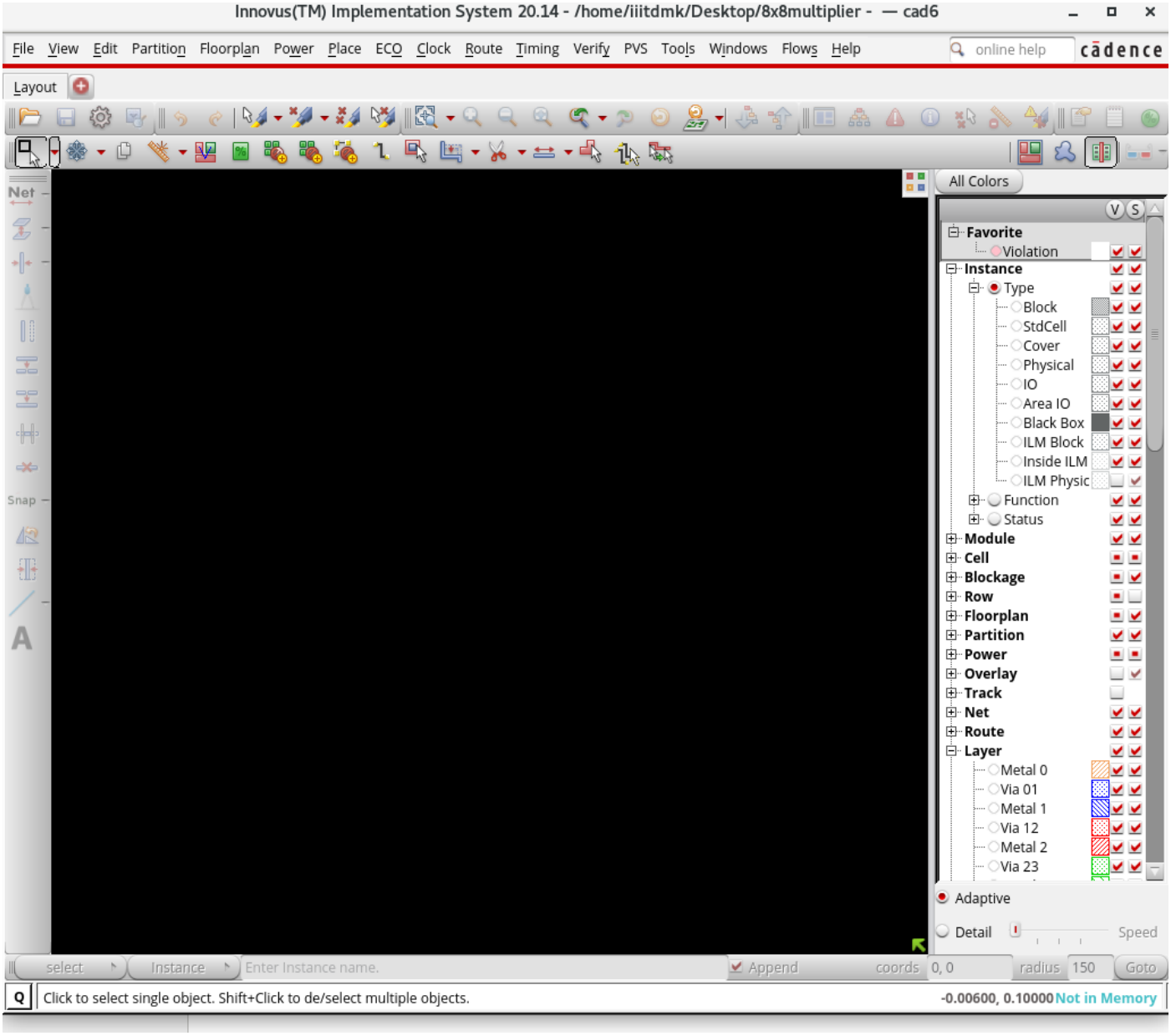
After the running the **.tcl** code as per the .tcl code all the reports so far are generated . we can check that by typing the following commands in terminal.

report_area → for area report

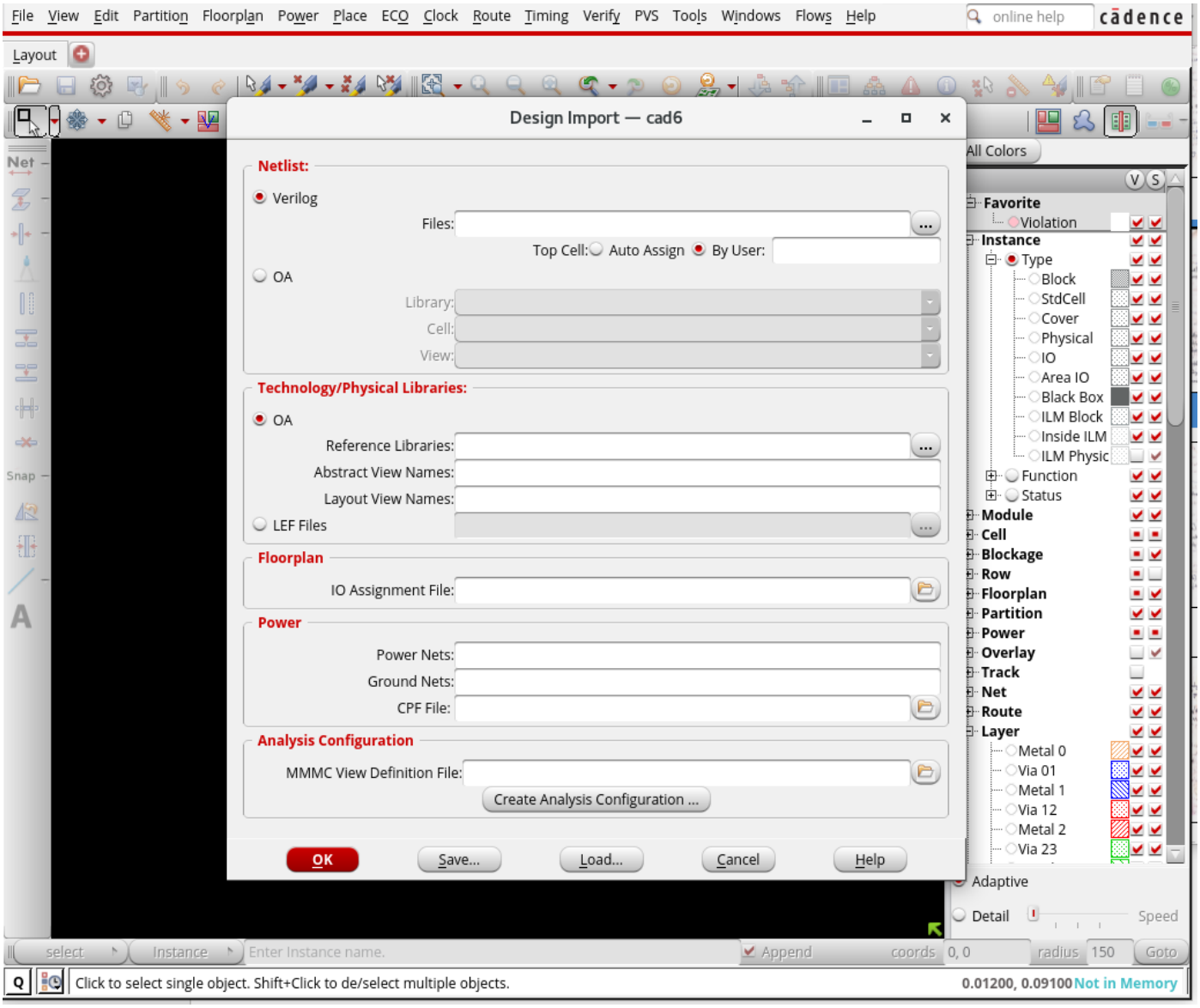
report_power → for power report

report_timing → for timing report

➔ We have attached the report (**.rpt**) files along with this project.
➔ After this we give the instruction of **innovus** in the same terminal , to open innovus tool .



➔ Now click on **file** → **import design**
➔ The design import window will pop as in below image.



➔ now follow these instruction to fill these sections.

Select auto assign

→ click on three dots

→ select dadda8x8_netlist.v file

→ ENTER

→ Select LEF files

→ click on three dots

→ /home/install/FOUNDARY/digital/90nm/dig/lef

→ select translated.lef file

→ Power nets - VDD Ground nets - VSS

→ MMMC view

→ click on create analysis

→ click on library set

→ new

→ slow

→ add (timing file)

/home/install/FOUNDRY/digital/90nm/dig/lib/slow.lib

→ click ok

→ click on library set

→ new

→ fast

→ add (timing file)

/home/install/FOUNDRY/digital/90nm/dig/lib/fast.lib

→ click ok

→ click on RC corner

→ name -- RC

→ QRC Technology

→ /home/install/FOUNDRY/digital/90nm/dig/QRC/gpdk90_91.tch

→ click ok

→ click on Delay corner

→ new

→ name = max

→ click on attributes

Library set : slow

Rc corner : Rc

→ new

→ name = max

→ click on attributes

Library set : slow

Rc corner : Rc

→ click on constraints

→ new

→ select add → add **output.sdc** (filename.sdc) file

→ click on analysis view

→ new

→ name = bc

constraints :constraint

Delay corner : min

→ new

→ name = wc

constraints :constraint

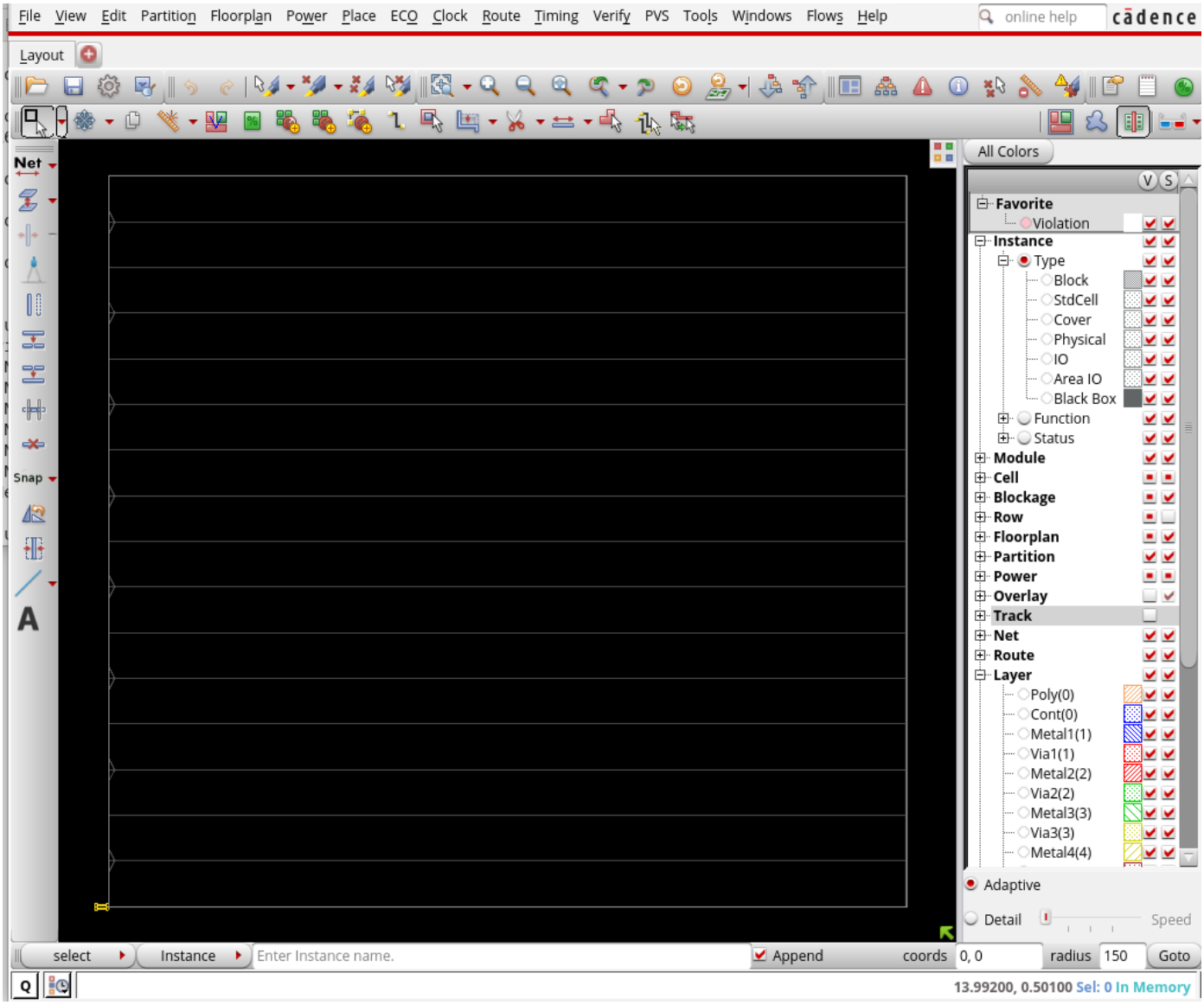
Delay corner : max

→ click on setup analysis view

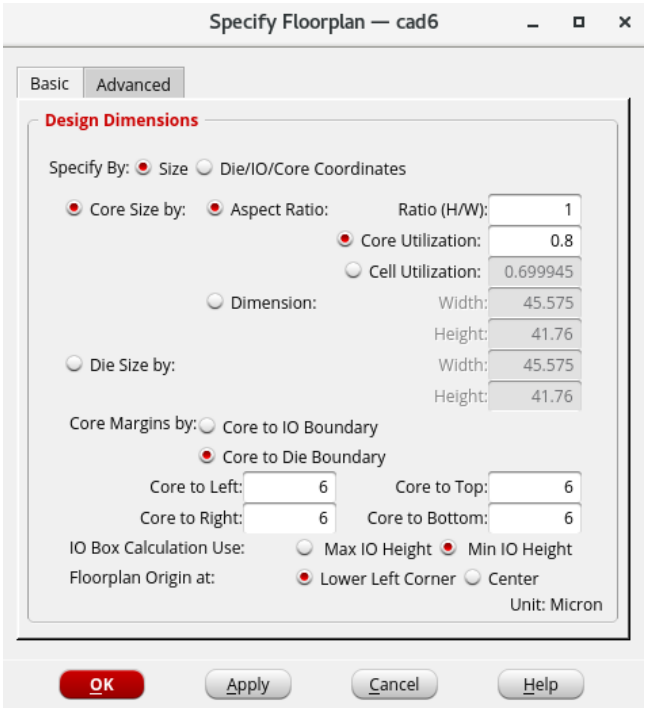
→ add → bc

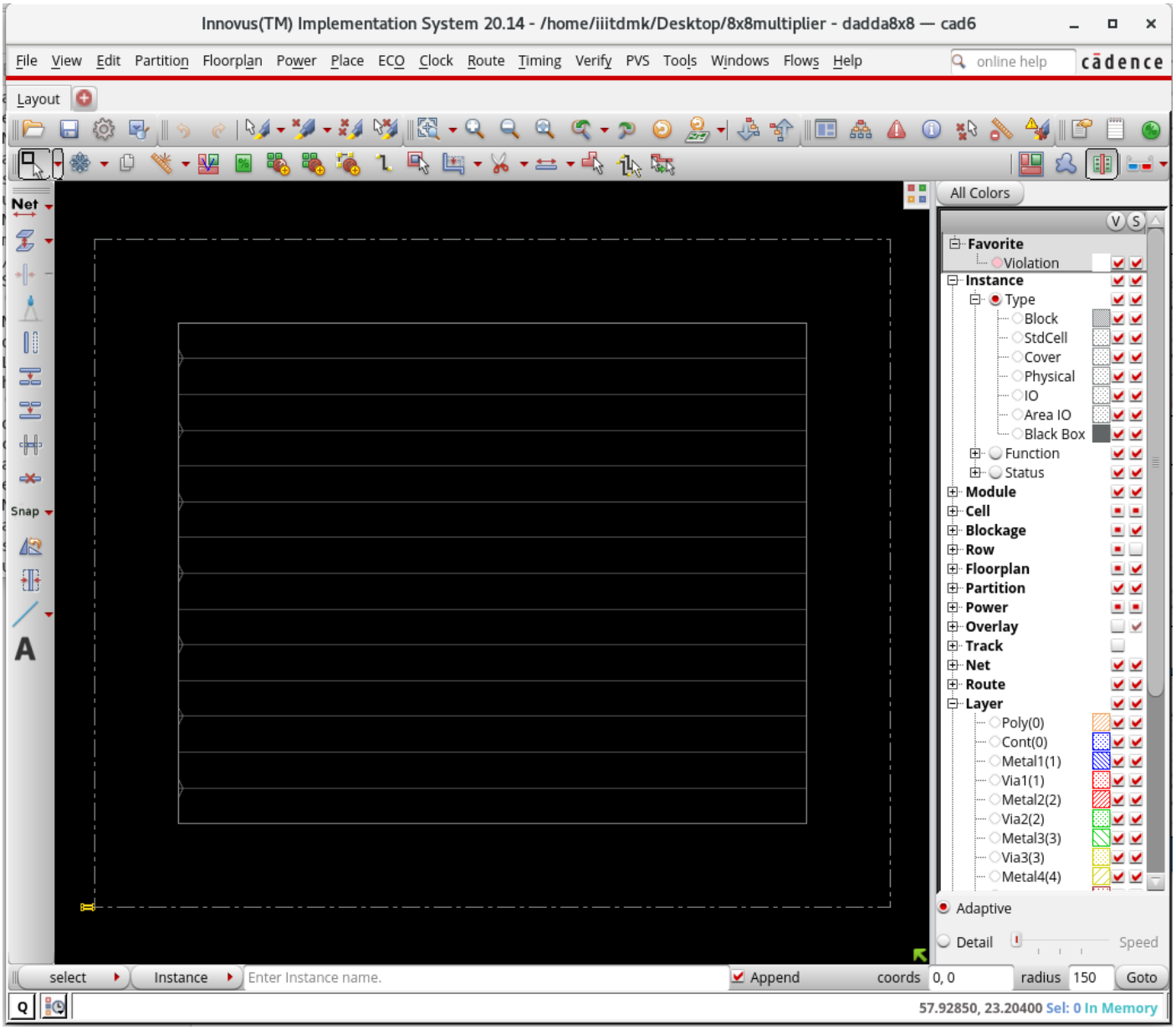
→ click on hold analysis view

→ add → wc



Now we have to do **Floor plan** and select as shown below .





→ click on **Power**

→ Global net connectors

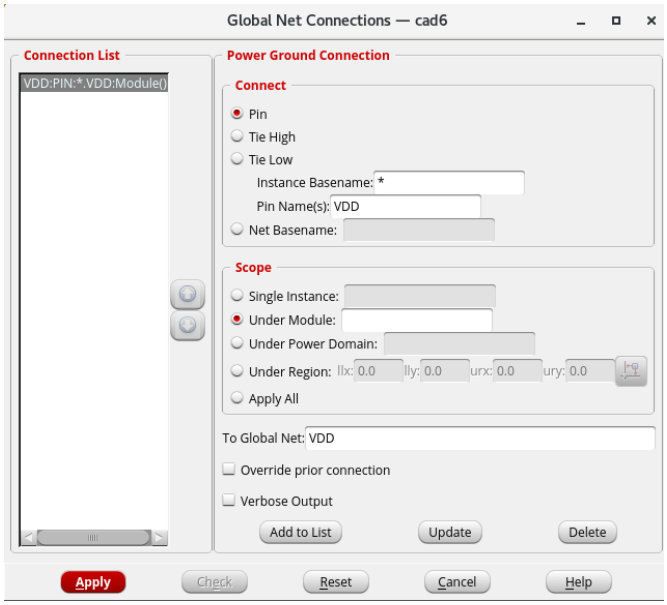
→ pin nets VDD

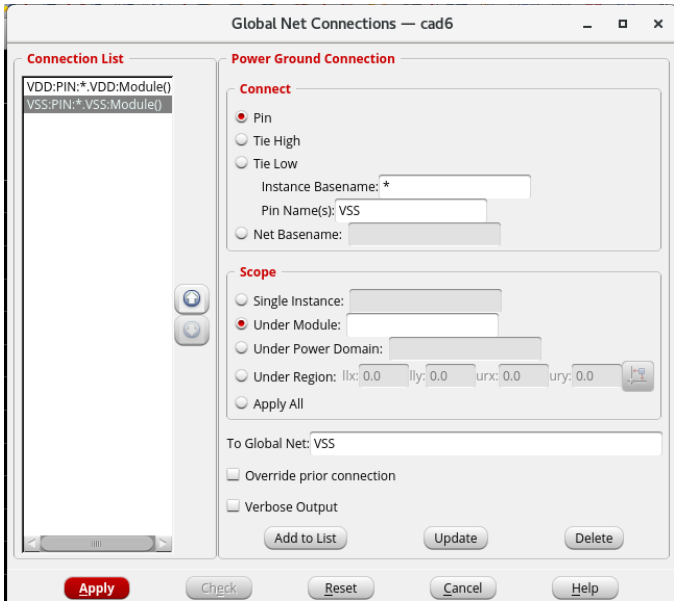
Global nets VDD then add to list

→ pin nets VSS

Global nets VSS then add to list

→ click on apply





→ click on **Power**

→ Power planning → click on net folder and add VDD , VSS → select ok

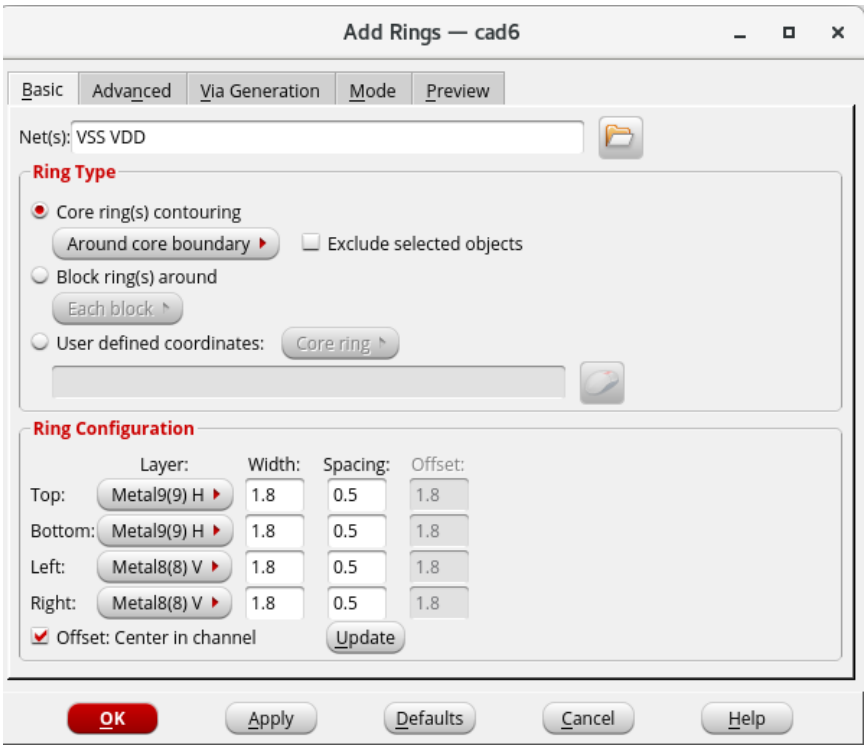
→ click on **Power**

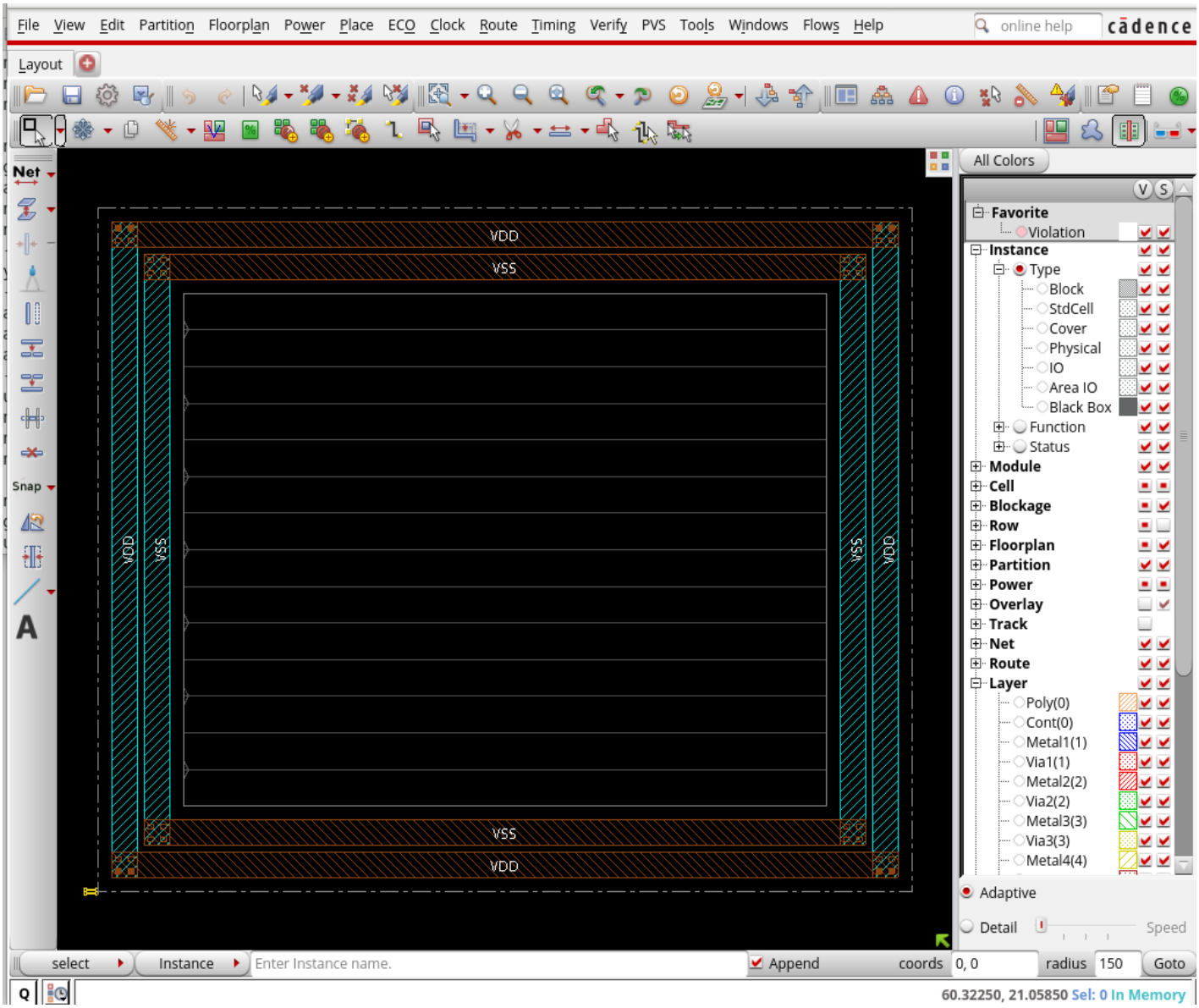
→ Power planning → click on Ring configurartion

→ Top , Bottom -- > metal9 (9) H,H

→ Left , Right -- > metal8 (8) V,V

Click offset center , then click on update , click ok





→ click on **Power**

→ Power planning → click on add stripe

→ add folder VDD , VSS

Layer -- > metal (9) Direction – Horizontal

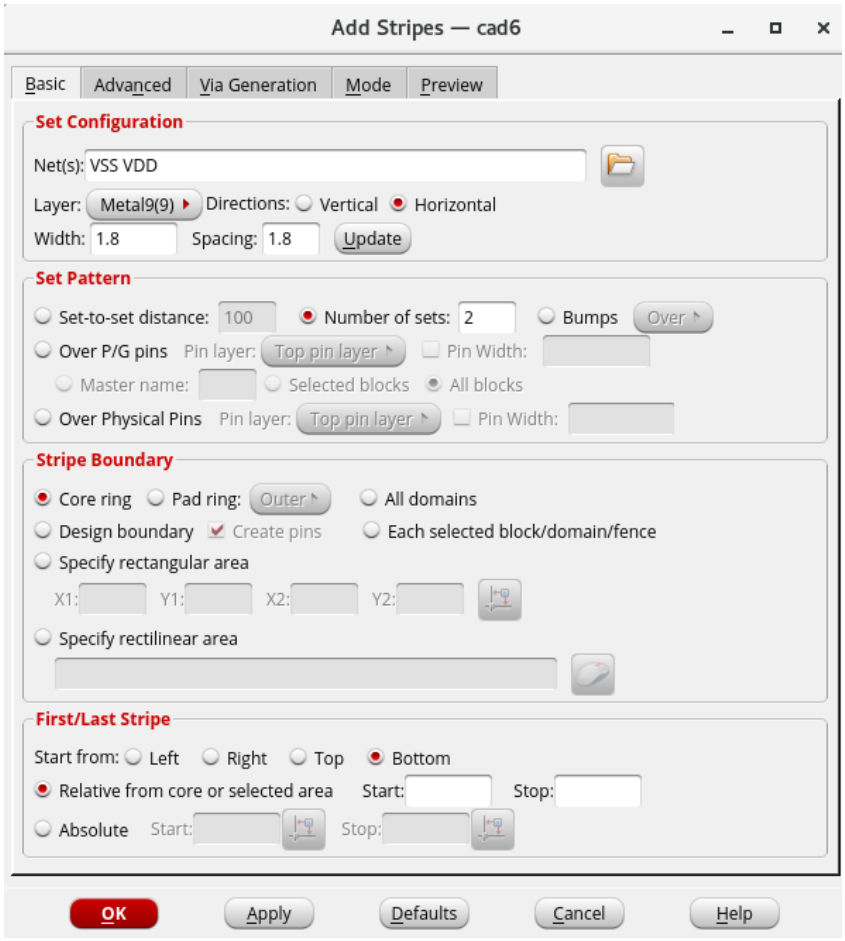
Click on no.of. sets = 2

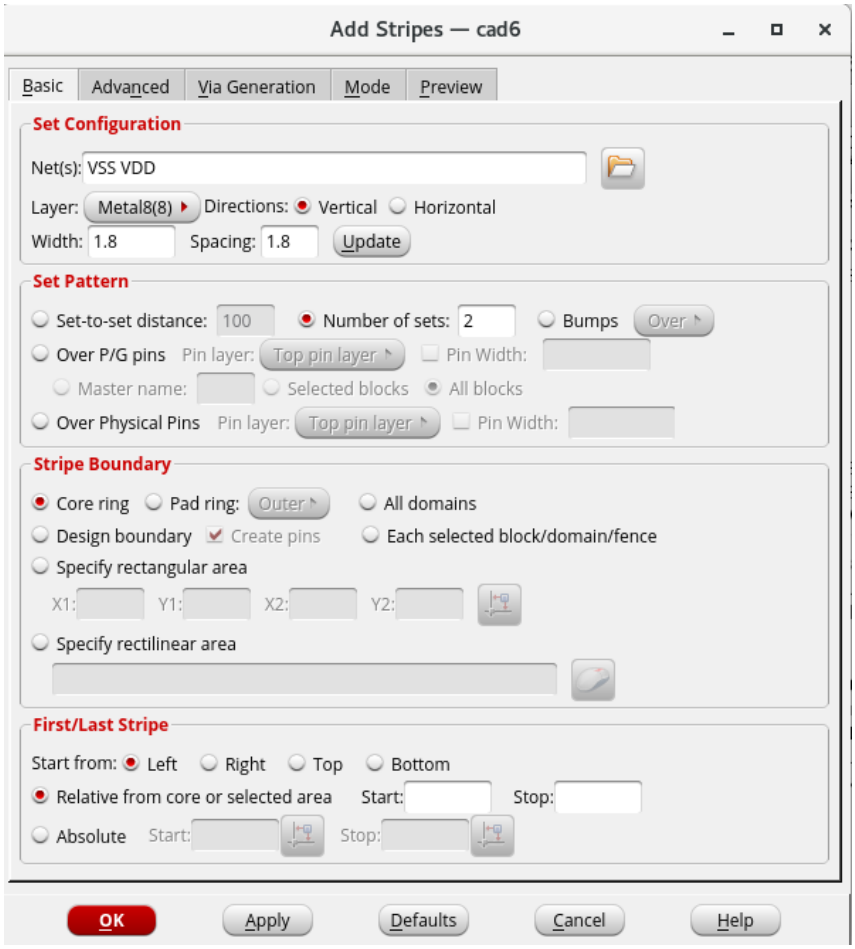
Click on apply

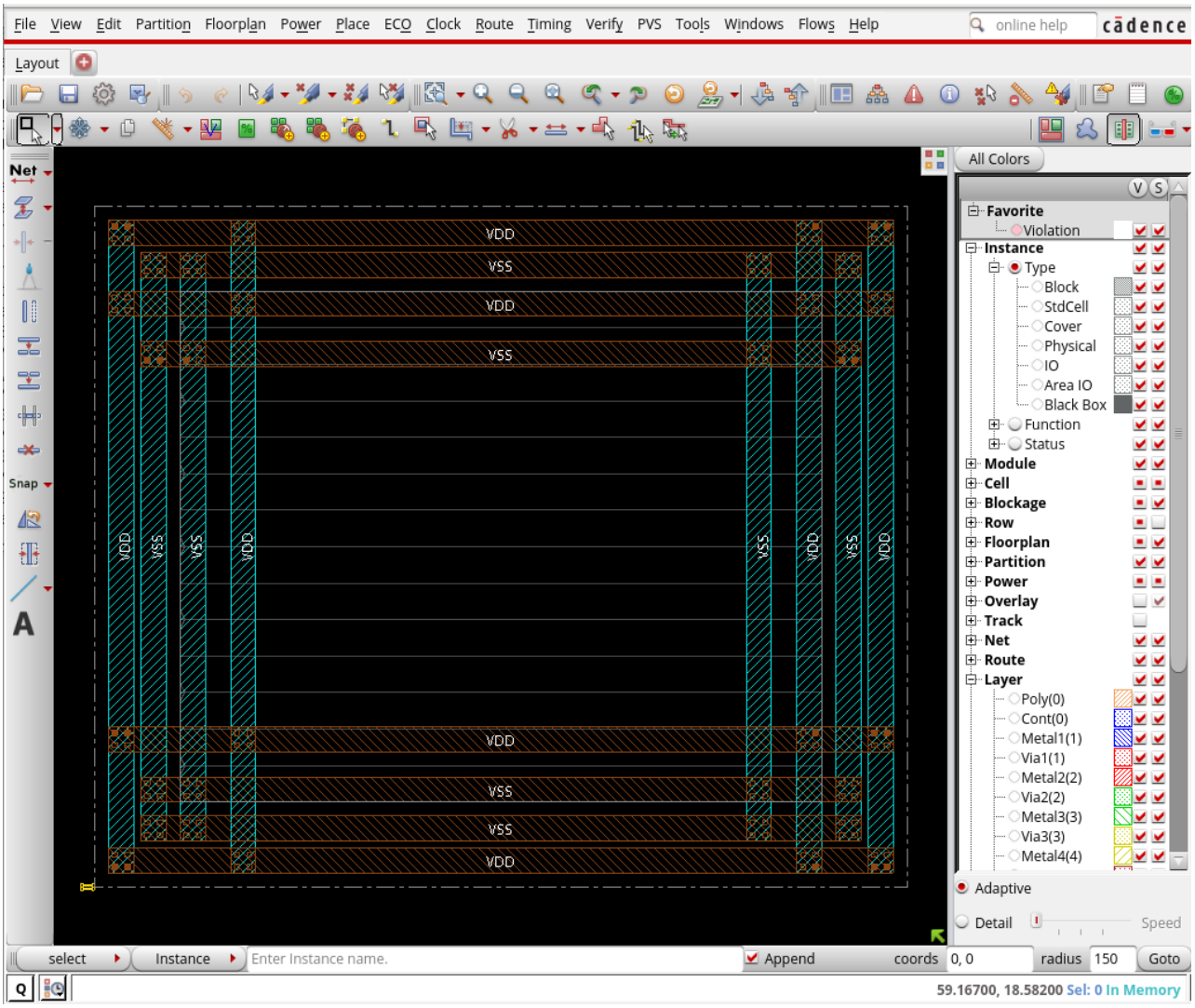
Then again click on

Layer -- > metal (8) Direction – Vertical

Then click apply .


Add Stripes — cad6
Basic
Advanced
Via Generation
Mode
Preview
Set Configuration
Net(s): VSS VDD
Layer: Metal9(9)
Directions: Vertical Horizontal
Width: 1.8
Spacing: 1.8
Update
Set Pattern
Set-to-set distance: 100
Number of sets: 2
Bumps
Over
Over P/G pins
Pin layer: Top pin layer
Pin Width:
Master name:
Selected blocks
All blocks
Over Physical Pins
Pin layer: Top pin layer
Pin Width:
Stripe Boundary
Core ring
Pad ring: Outer
All domains
Design boundary
Create pins
Each selected block/domain/fence
Specify rectangular area
X1:
Y1:
X2:
Y2:
Specify rectilinear area
First/Last Stripe
Start from: Left Right Top Bottom
Relative from core or selected area
Start:
Stop:
Absolute
Start:
Stop:
OK
Apply
Defaults
Cancel
Help


Add Stripes — cad6
Basic
Advanced
Via Generation
Mode
Preview
Set Configuration
Net(s): VSS VDD
Layer: Metal8(8)
Directions: Vertical Horizontal
Width: 1.8
Spacing: 1.8
Update
Set Pattern
Set-to-set distance: 100
Number of sets: 2
Bumps
Over
Over P/G pins
Pin layer: Top pin layer
Pin Width:
Master name:
Selected blocks
All blocks
Over Physical Pins
Pin layer: Top pin layer
Pin Width:
Stripe Boundary
Core ring
Pad ring: Outer
All domains
Design boundary
Create pins
Each selected block/domain/fence
Specify rectangular area
X1:
Y1:
X2:
Y2:
Specify rectilinear area
First/Last Stripe
Start from: Left Right Top Bottom
Relative from core or selected area
Start:
Stop:
Absolute
Start:
Stop:
OK
Apply
Defaults
Cancel
Help

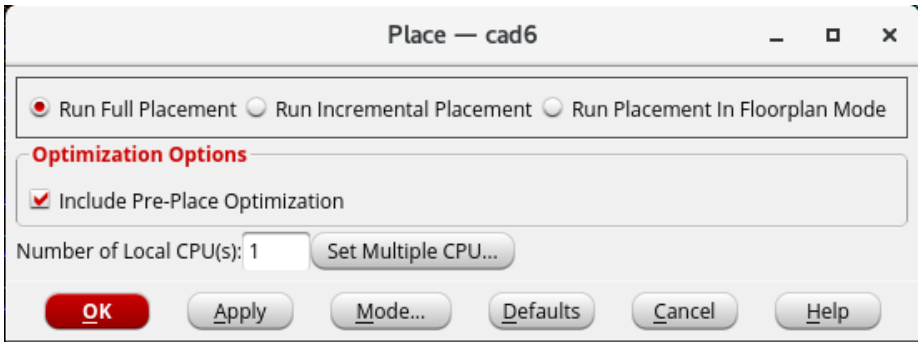


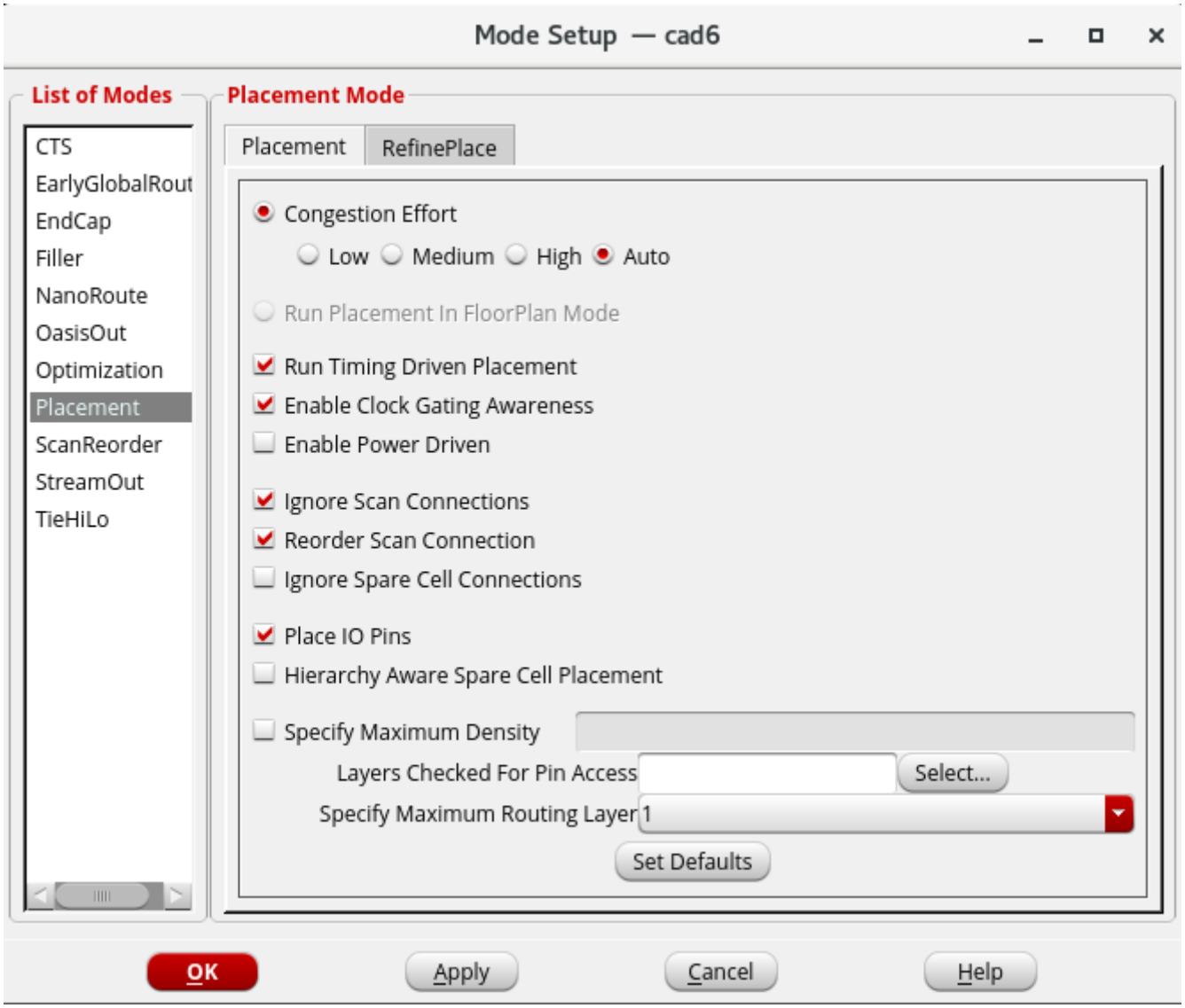
➔ select **route** , then **special route**

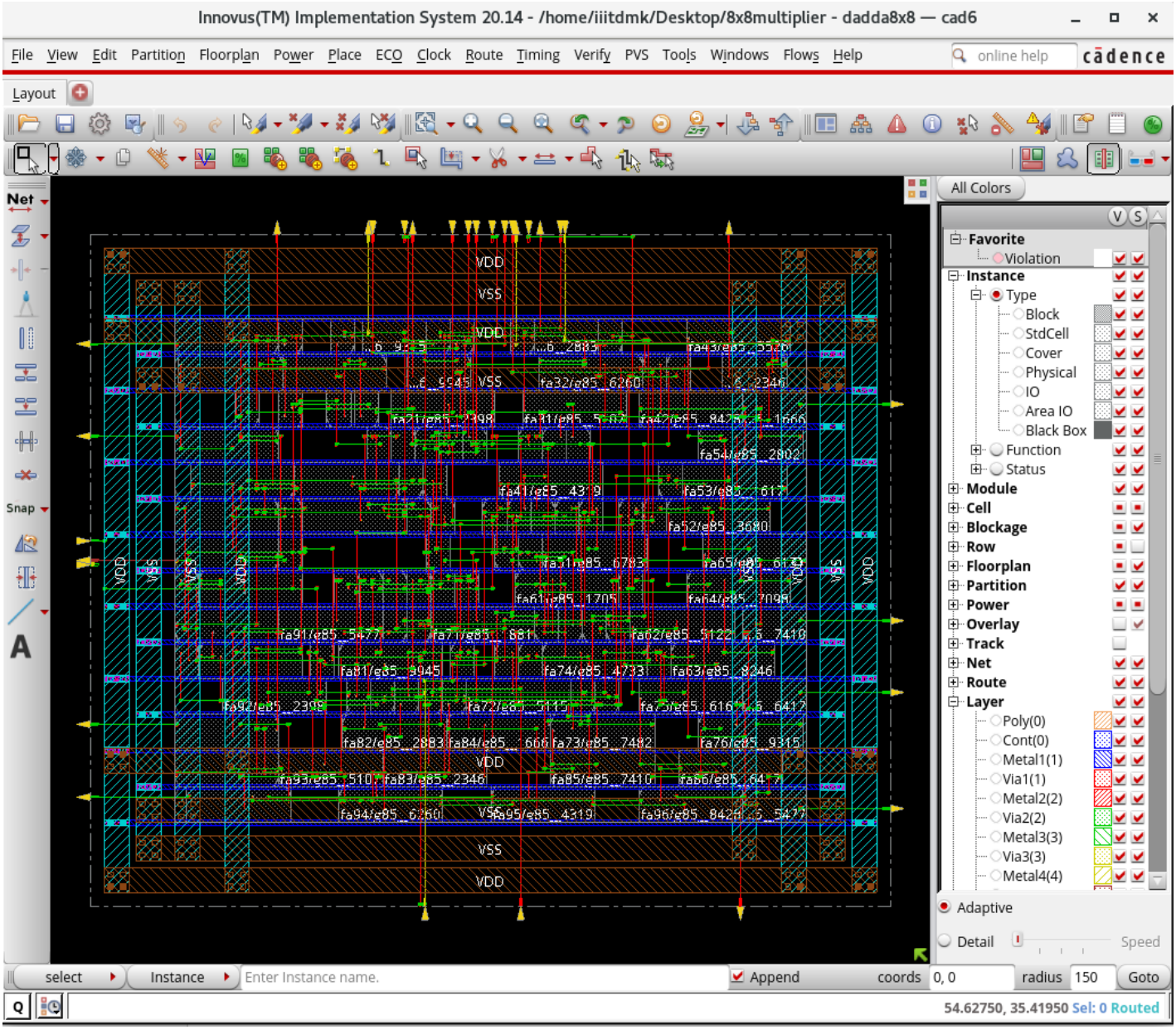
→ click on **Place**

→ place standard cell → click on mode

→ select IO pins then click ok and then apply.

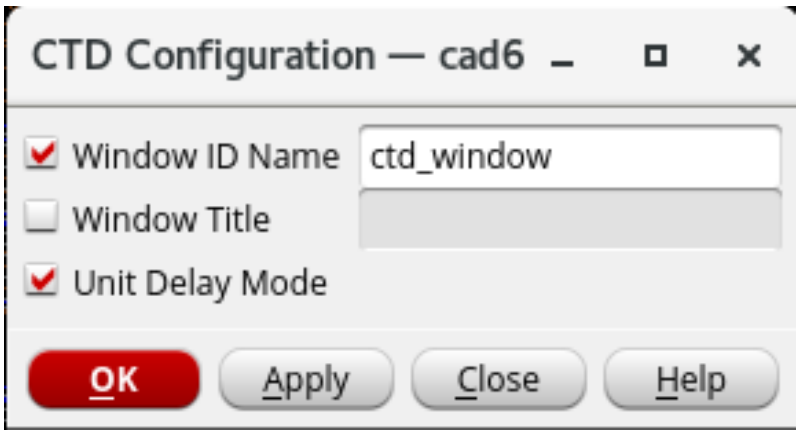






→ click on **Clock**

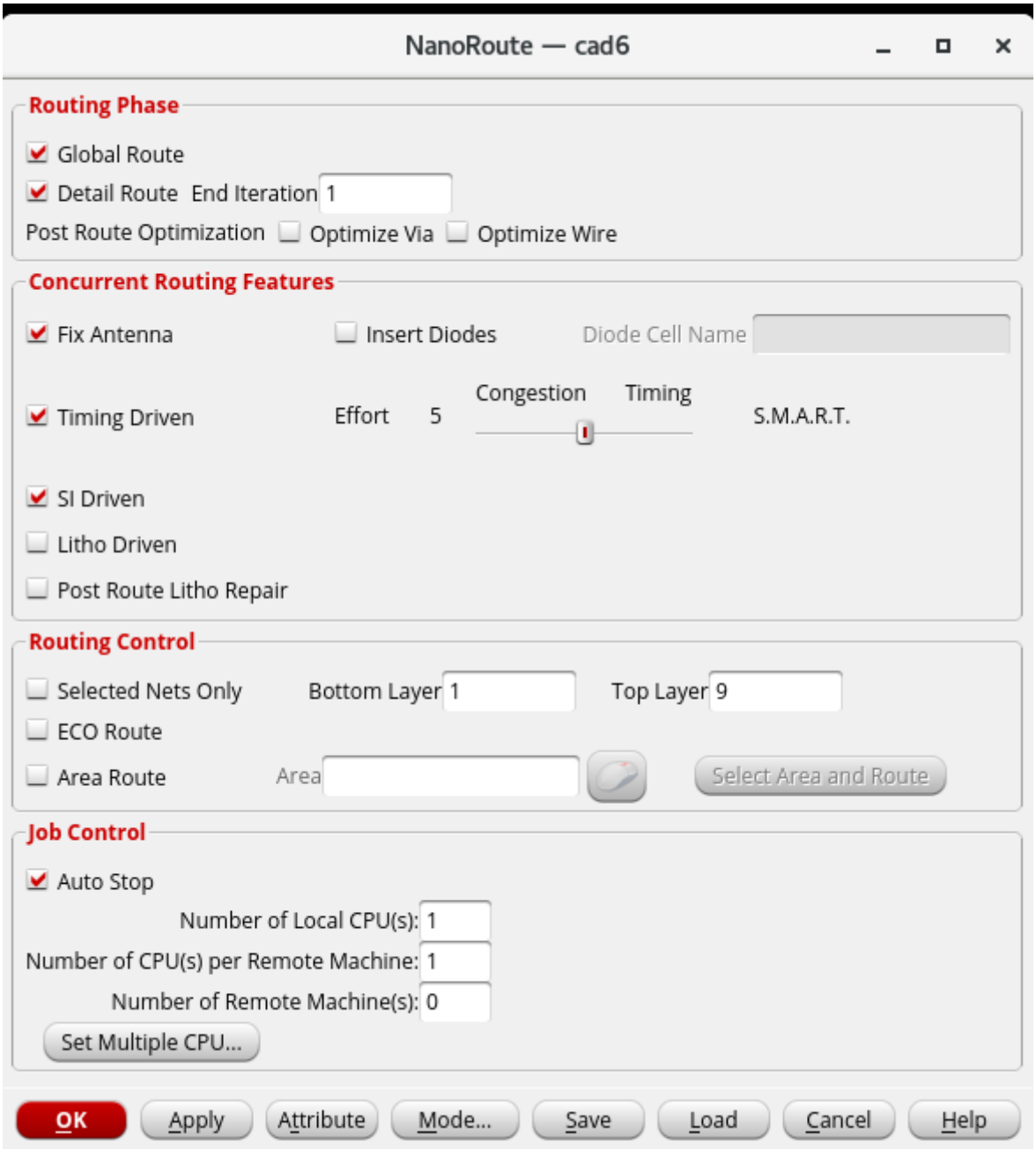
→ clock tree debug → clock delay → unit delay (select 3rd option)



→ click on **Route**

→ Nano route → select SI driven , timing driven

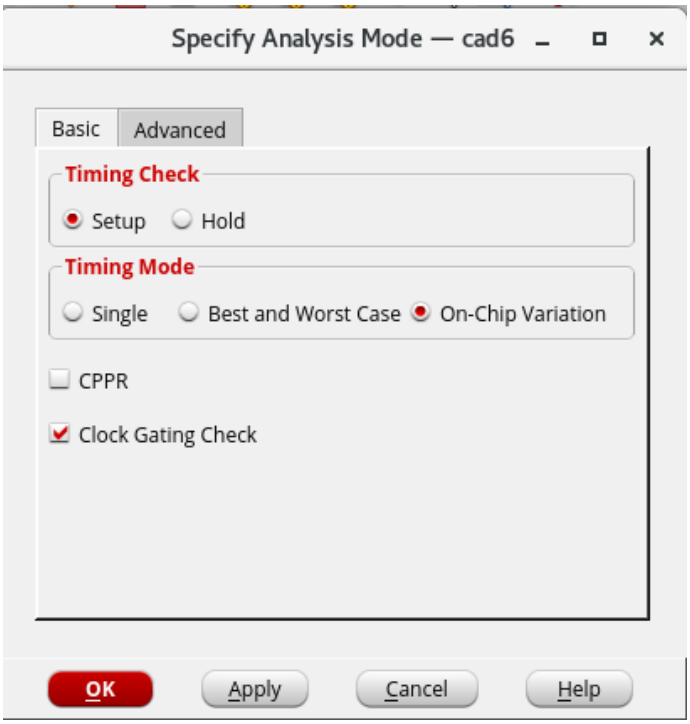
→ click apply.

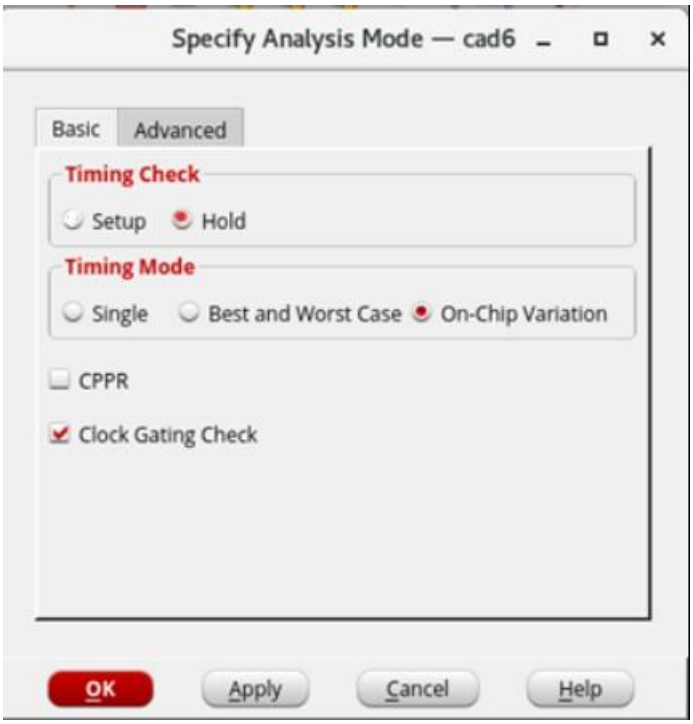


➔ Verify DRC ……then ok .
➔ Verify Connectivity ……then ok .

→ click on **Tools**

→ set mode → specify Analysis mode



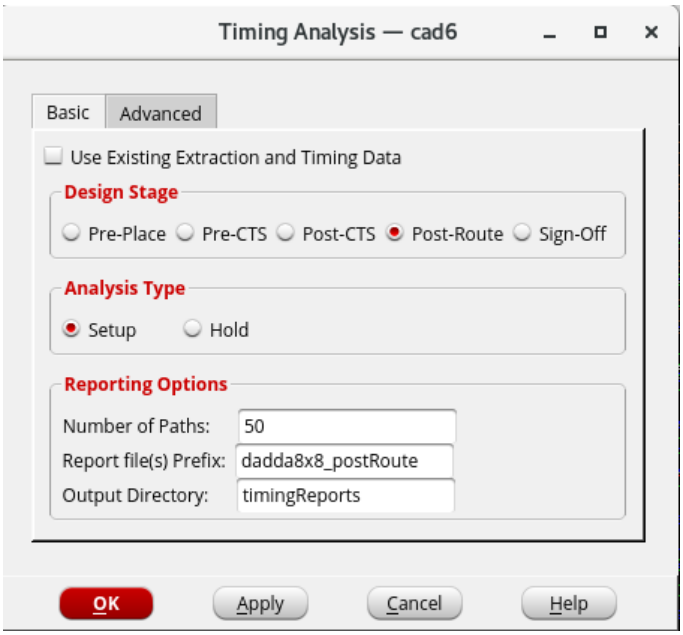


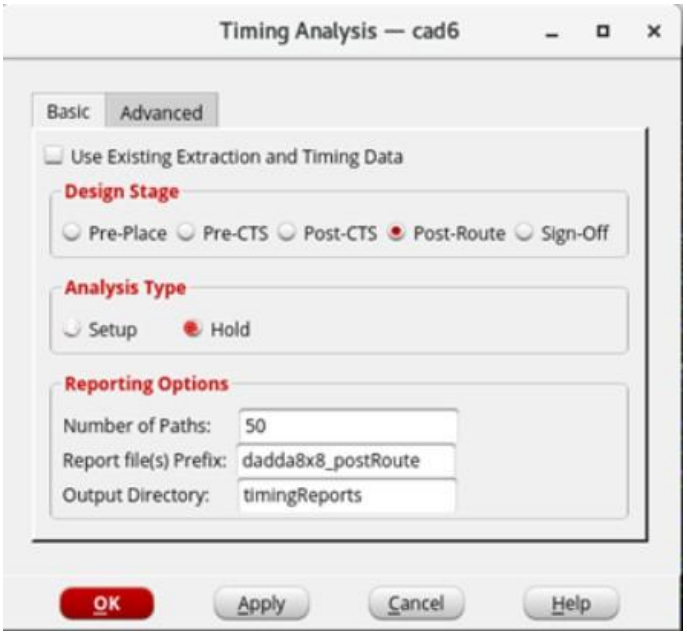
→ click on **Timing**

→ report timing

→ select Post route and setup and click apply

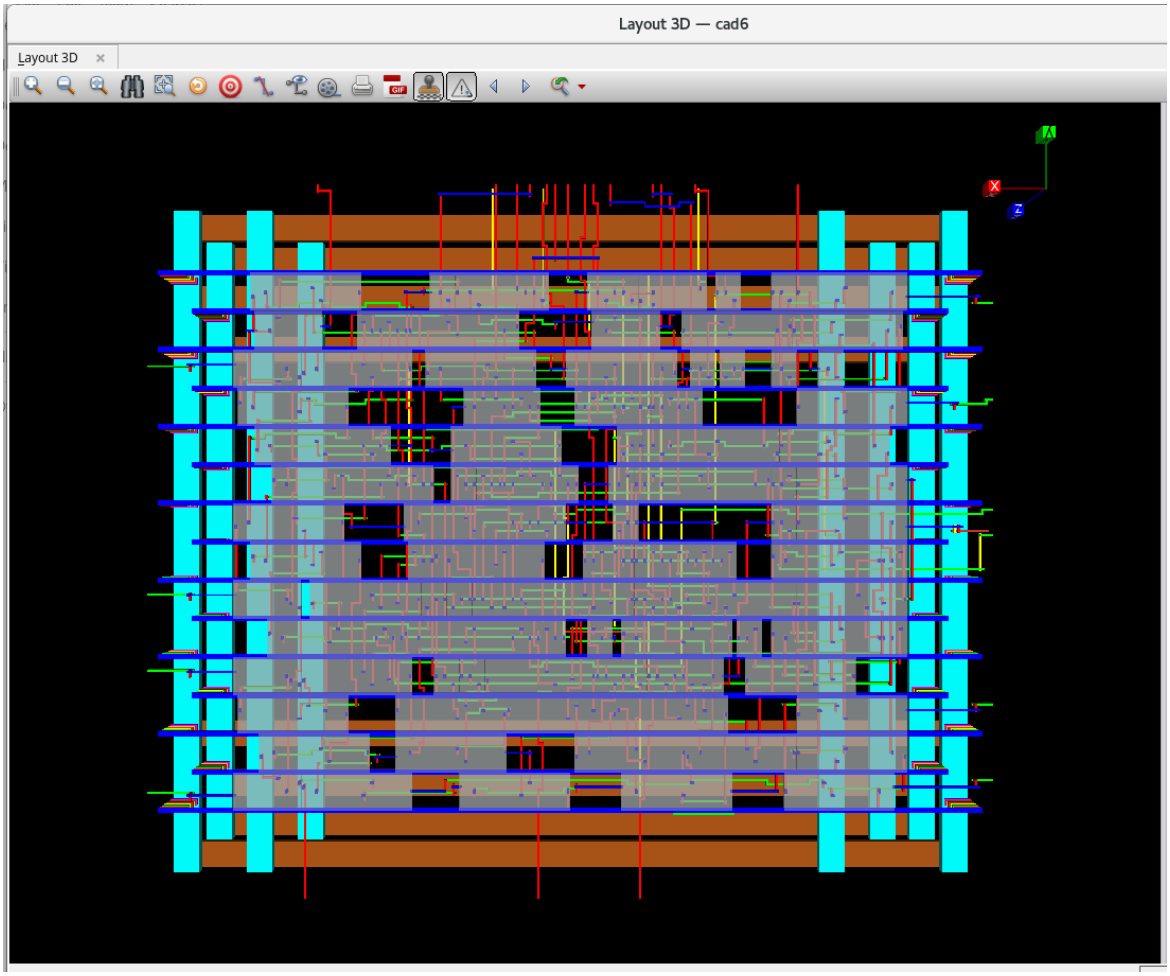
→ repeat again and select Post route and hold and click apply.



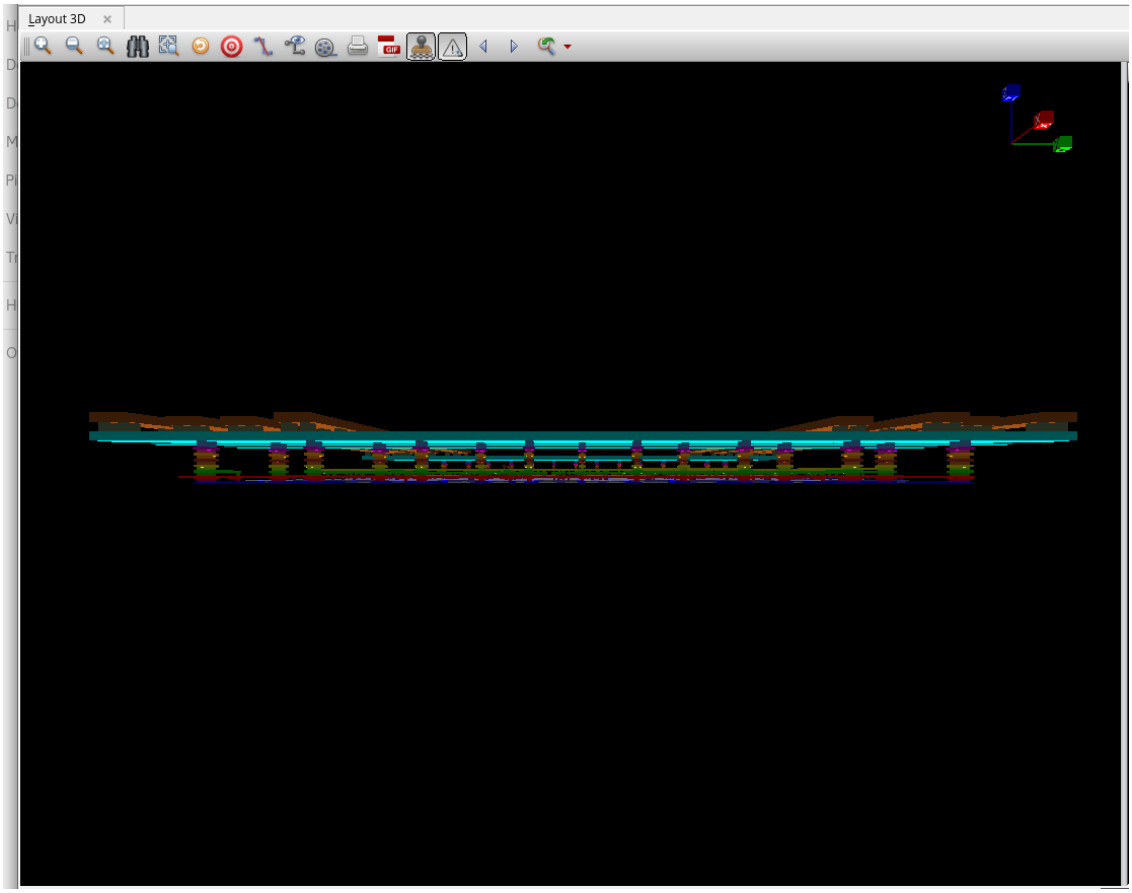


➔ For **Layout 3D viewer** , click on current view on right side of window

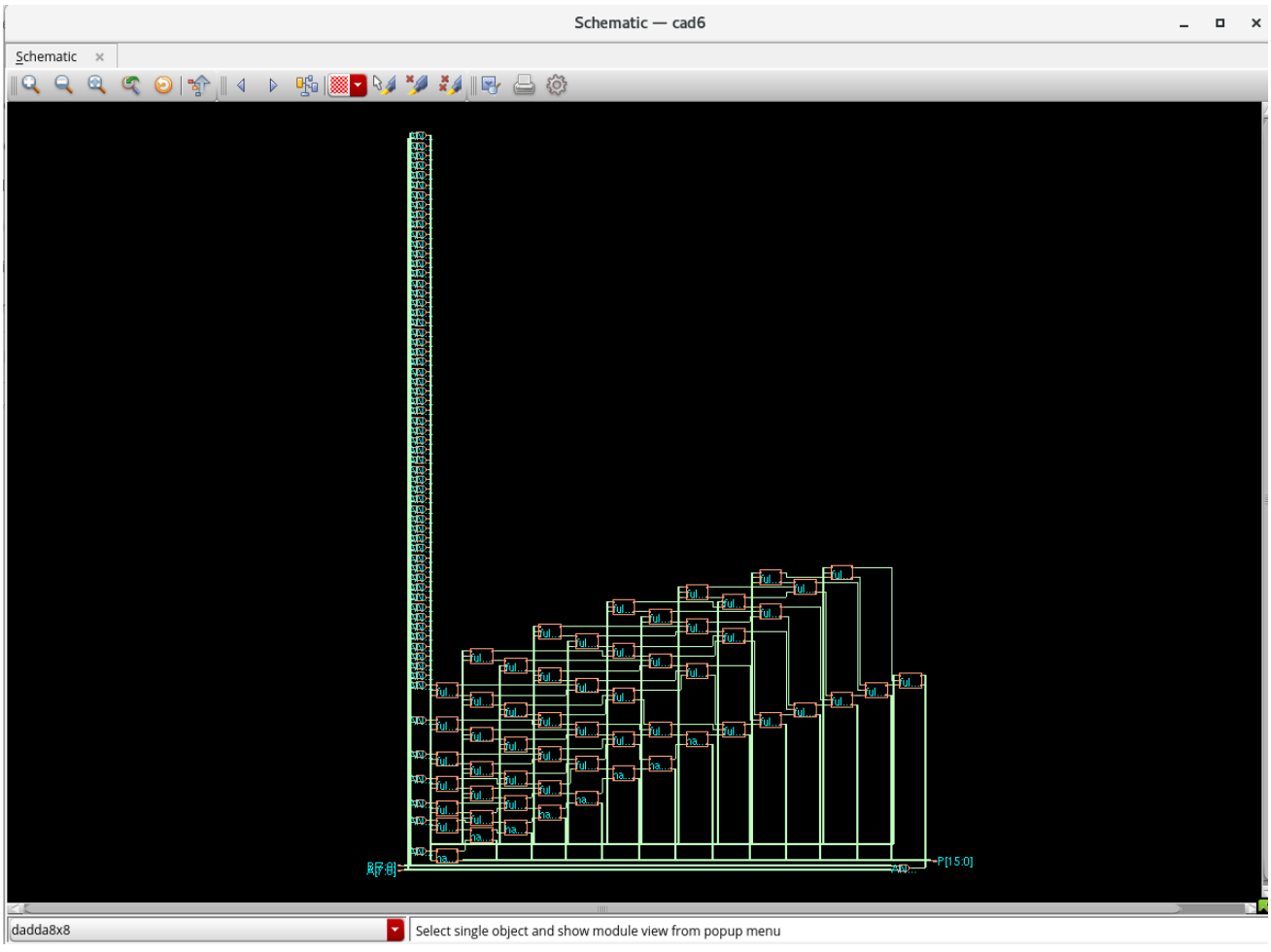
Front view :



side view :



➔ For schematic view , click on **tools** and then **schematic viewer** .



→ click on **File**

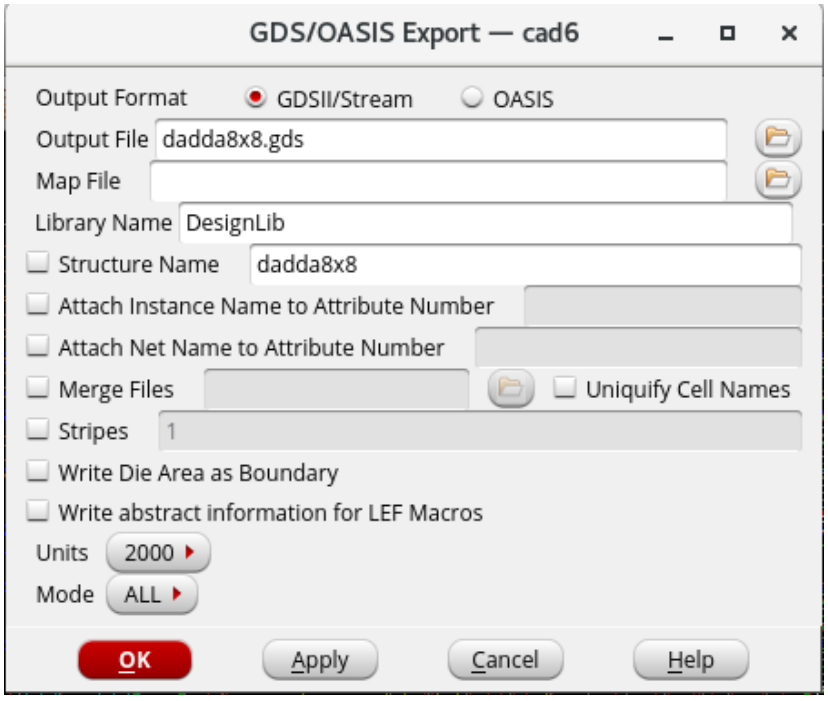
→ Save design

→ select data type as innovus and save as dadda8x8.enc (filename.enc)

→ click on **File**

→ Save

→ select GDS and save as dadda8x8.gds (filename.gds)



Finally our GDS file have saved . this could be helpful for further fabrication. we have **successfully designed our semi-custom design of 8 bit low power dadda algorithm** . successfully verified all DRC rules and connectivity rules .

Conclusion

In this project, a low-power 8×8 bit Dadda multiplier was successfully designed and implemented using the semi-custom VLSI design flow in Cadence. The complete process included schematic design, logic synthesis, netlist verification, physical design steps such as floorplanning, placement, clock tree synthesis, routing, and design rule checks (DRC) and layout versus schematic checks (LVS). Through these stages, the architecture was optimized to reduce switching activity and logic depth, resulting in improved power efficiency and performance compared to standard array multiplier structures.

The final results confirm that the Dadda-based multiplier architecture is highly suitable for low-power, high-performance VLSI systems. The design met timing and power constraints, demonstrating that the semi-custom methodology in Cadence provides efficient transistor-level control and accurate physical realization. This project strengthens the understanding of real-time ASIC development flow and shows the practical feasibility of implementing advanced arithmetic circuits in modern VLSI applications like DSP blocks, microprocessors, and low-power embedded systems.