BIRZEIT UNIVERSITY

Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 5131

Hardware design Lab Manual
Physical VLSI PART

Created and reviewed by:
Dr. Khader Mohammad, Eng. Nour Daglas, Eng. Raha Zababdi

Sep 23, 2023

Contents

Laboratory Regulations and Safety Rules

The following Regulations and Safety Rules must be observed in the laboratory:

1. Students are allowed to use only the tool provided in the experiment manual or for senior project laboratory.
2. Software installation in any computer laboratory is not allowed without the permission from the Laboratory Staff.
3. Computer games are strictly prohibited in the computer laboratory.
4. Students are not allowed to use any of the tools or shared any access to the server with anyone.
5. Smoking and drinking in the laboratory are not permitted.

Experiment No. 1 - VLSI Custom Design Flow

1.1 Objectives

In this experiment, the following objectives will be achieved:

- Learn how to install the EDA electric tool.
- How to make a transistor level schematic and simple view for simple gates like inverter, NAND, NOR gates.
- How to make the layout for simple gates like inverter, NAND, and NOR gates.
- Run timing and check delay and waveforms using LTSpice.

1.2 Equipment Required

- Electric VLSI Design System Version 9.07.

1.3 Pre-Lab

Before starting the experiment, complete the following pre-lab tasks:

- Download and run the ELECTRIC VLSI Design System Version 9.07 tool.
- For more information, you can visit the Electric VLSI website: Electric VLSI Website I PREFER TO ADD LINK TO SMALL VIDEO
- You can create a file where you download the electric tool call it:

Having issue opening jar file:

1. Go to the directory you have jar file saved
2. Create run.bat file.

3. Edit the file and add (java -jar electricBinary-9.07.jar)

<<run.bat>>

This will run your jar data can see this video too:

https://youtu.be/ErLhlCKonE0?si=CVqMl70g-aBFMblL

For more Process files:

http://ptm.asu.edu/latest.html

https://vlsiwikipedia.blogspot.com/p/technology-file.html

http://www.vlsifacts.com/getting-started-with-electric/

https://github.com/rayanfer32/electric_VLSI/blob/master/README.md

1.4 Introduction

1.4.1 Electric VLSI Design System

- VLSI Electric is a powerful software tool widely used in the field of electronic design automation (EDA). Used in designing integrated circuits.
- It facilitates design and analysis of complex circuits at various levels of abstraction.
- Key aspects of Electric VLSI:

  1. Circuit Design and Editing: Intuitive interface for creating and editing circuit designs with various entry options.

  2. Layout Design: Tools for component placement, connection routing, and optimizing physical representation.

  3. Simulation and Analysis: Built-in simulation capabilities for behavioral, timing, and circuit-level analysis.

  4. Verification and Testing: Features like design rule checking and netlist extraction for identifying and resolving errors.

- Electric VLSI Design System provides a robust environment for efficient manual design and analysis of integrated circuits. For more information about Electric VLSI, you can visit the following links.

[Official Electric VLSI Website]

[Electric VLSI User Guide]

[Electric VLSI Tutorials]

- Electric VLSI Design System provides a robust environment for VLSI designers to design and analyze integrated circuits, from small-scale designs to large-scale complex systems efficiently manually as shown in Figure 1.1.

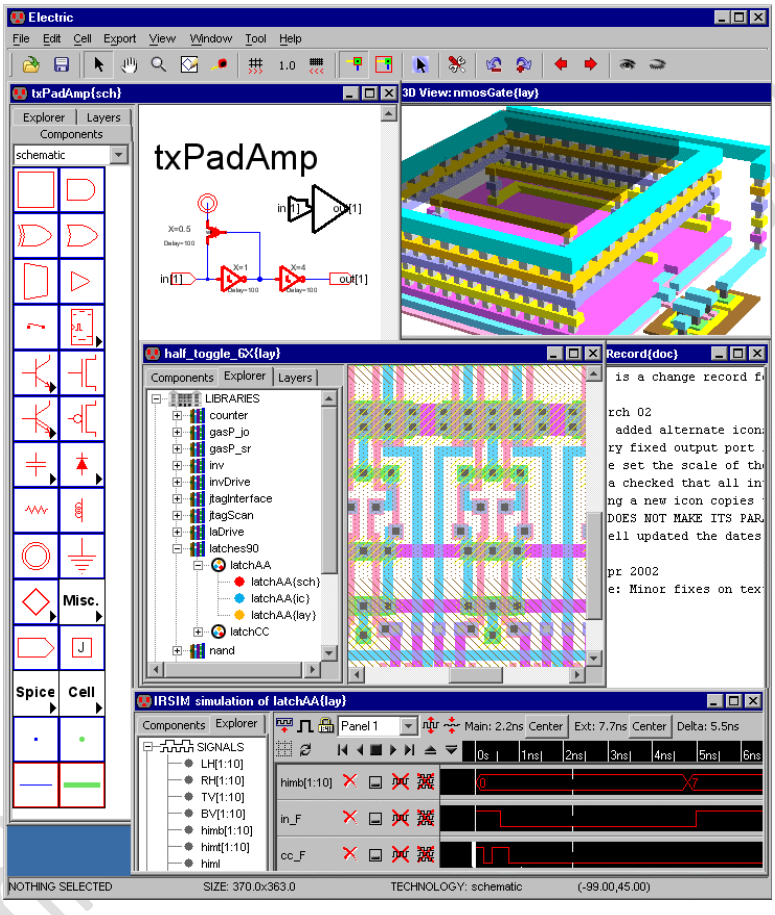

Figure 1.1: Electric VLSI Design System Version 9.07[4].

1.4.2 LTSpice Tool:

LTspice is a free electronic circuit simulation tool originally developed by Linear Technology Corporation and now owned by Analog Devices. It allows users to design, simulate, and analyze electronic circuits using a schematic capture interface. The simulation capabilities include DC, AC, transient, and noise analyses. LTspice provides a waveform viewer for analyzing simulation results, a library of component models, and supports features like Monte Carlo analysis, AC analysis, transient analysis, parametric

sweep, subcircuits, and macromodels. It is known for its accuracy, fast simulation speed, and has a large user community with ample online resources.

1.5 Procedure

1.5.1 The CMOS Inverter

An inverter can be constructed using an NMOS transistor and a PMOS transistor. The input A controls the gates of both transistors, while the output Y is connected to the drains of both transistors. The NMOS transistor's source is connected to ground (GND), and the PMOS transistor's source is connected to the power supply (VDD), which is typically process based, for now let's say it is 5 volts. Although the source and drain are symmetric, conventionally, the side closer to the rail (VDD/GND) is referred to as the source, and the side closer to the output is referred to as the drain (see Figure 1.2).

When A is 1, the NMOS transistor is turned ON, and the PMOS transistor is turned OFF. As a result, the output Y has a conducting path to GND through the NMOS transistor but no path to VDD, causing it to be pulled down to 0. Conversely, when A is 0, the PMOS transistor is turned ON, and the NMOS transistor is turned OFF. This configuration pulls the output up to 1 through the PMOS transistor. Therefore, the output Y is the logical complement of the input A (Y = ~A).

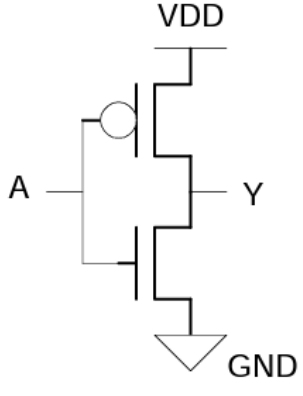

Figure 1.2: Schematic of an inverter.

Note: In this experiment we will use 300nm Technology. We can change it by changing the file we use here we will use C5_model.txt .

a) The CMOS Inverter Schematic:

To build a NOT gate using electric design software, follow these steps:

1. . Open the VLSI Electric software and create a new project (see Figure 1.3).

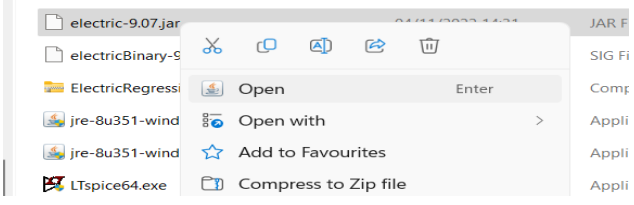

Figure 1.3: Open electric 9.07.

2. Create a new library by clicking on File >> New Library (see Figure 1.4).

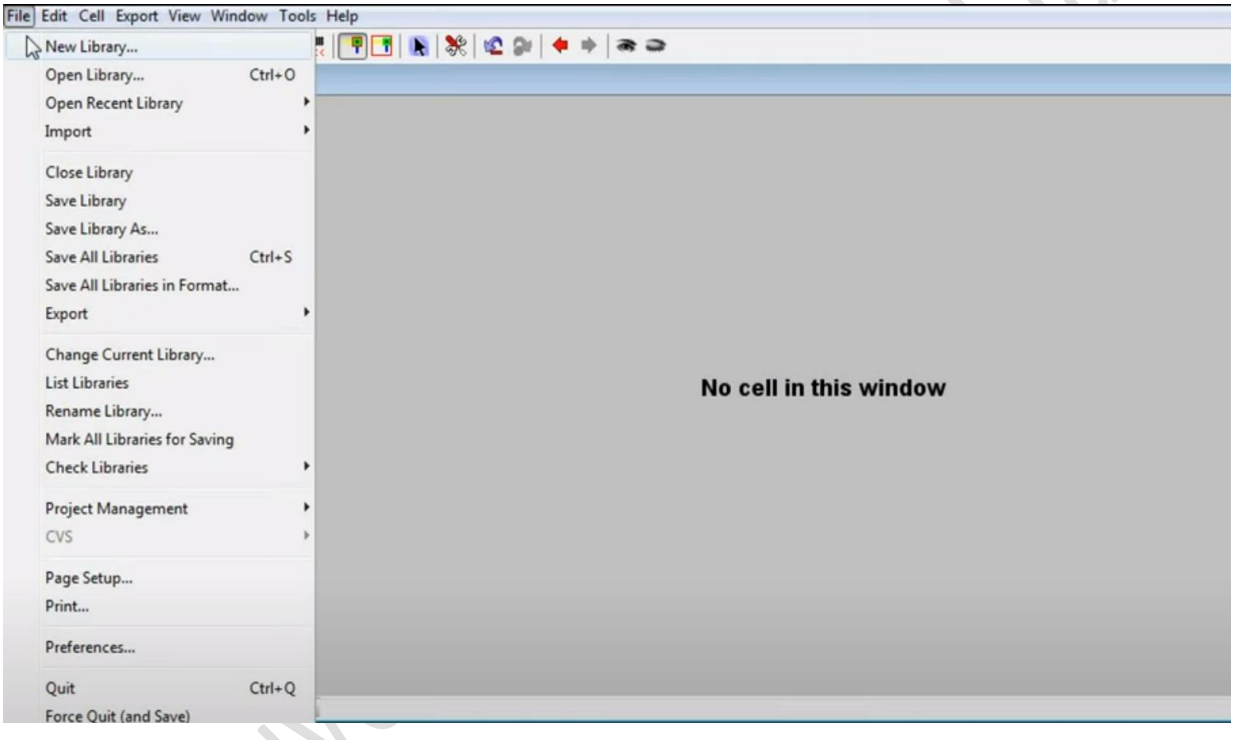

Figure 1.4: Creating a new library.

3. Set the MOS scale to 300nm by adjusting the settings and click Apply, then OK (see Figure 1.5).

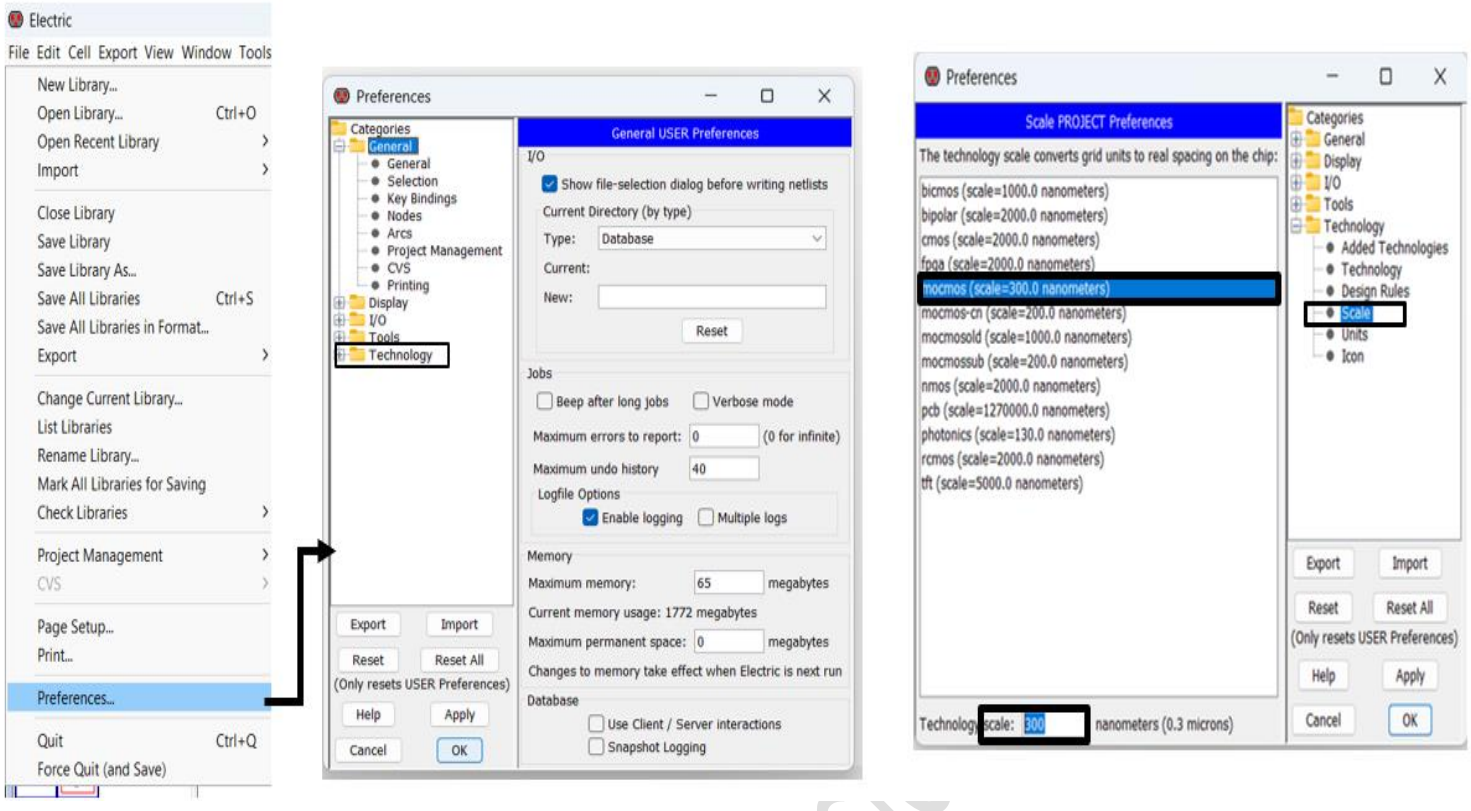

Figure 1.5: Changing the MOS scale.

4. Name your library, for example, "Inverter," and click OK (see Figure 1.6).

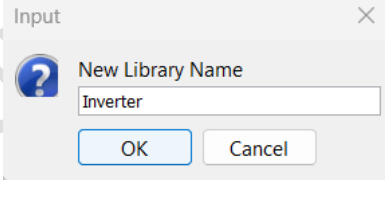

Figure 1.6: Naming the library.

5. Create a new schematic cell by clicking on Cell >> New Cell (see Figure 1.7).

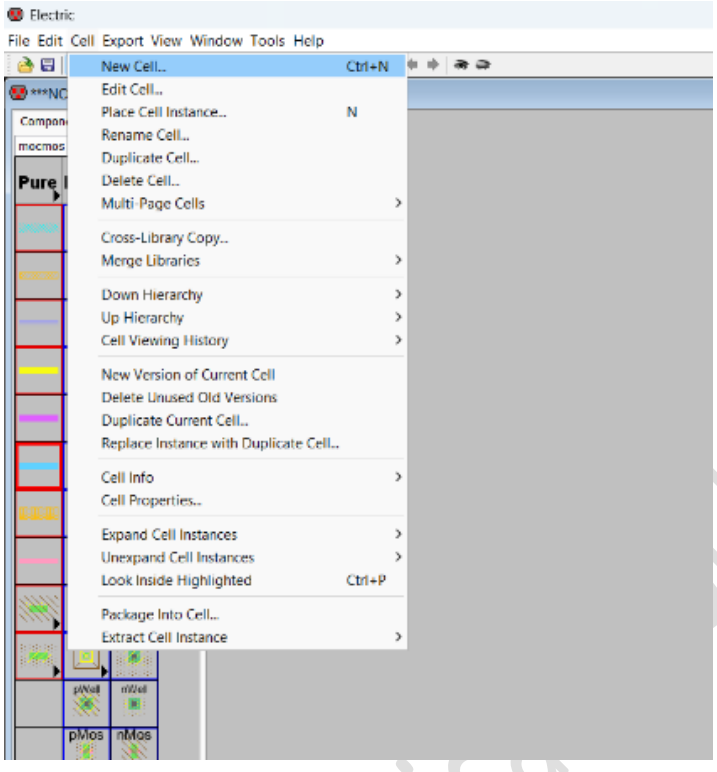

Figure 1.7: Creating a new cell.

6. Select the schematic cell type and name it, for example, "invshc," then click OK (see Figure 1.8).

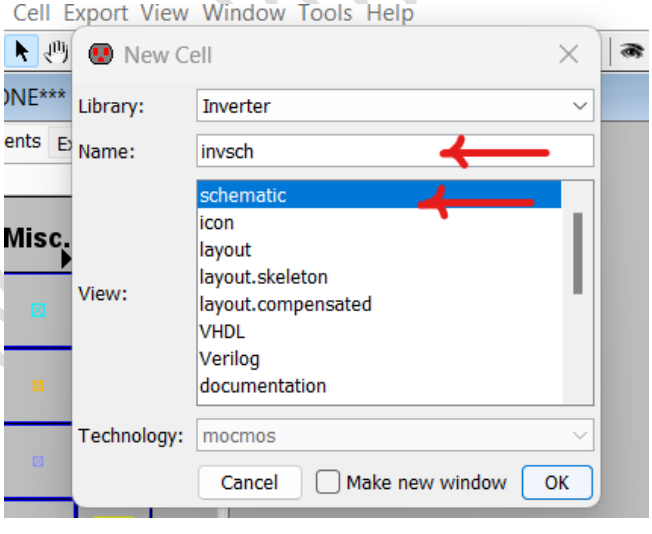

Figure 1.8: Selecting the schematic cell.

7. Select the required components, including the NMOS transistor, PMOS transistor, input, output, VDD, and GND (see Figure 1.9).

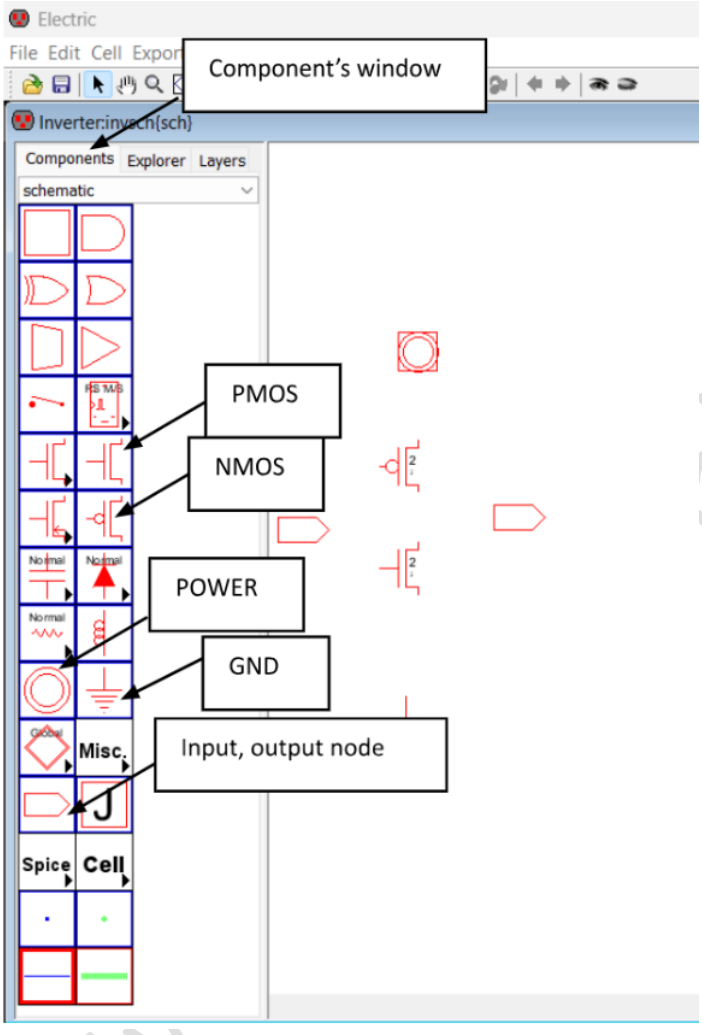

Figure 1.9: Selecting components.

8. Make connections between the components by clicking the left button of the mouse on one terminal of a component and then clicking the right button on the terminal you want to connect it to (see Figure 1.10).

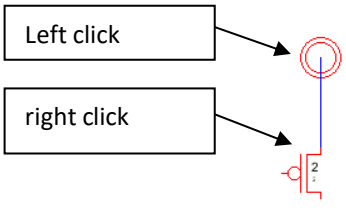

Figure 1.10: Making connections between components.

9. Connect all components as shown in Figure 1.11.

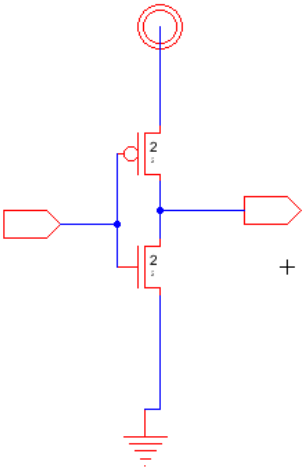

Figure 1.11: Making the connection between all components.

10. Adjust the width and length of the transistors by clicking on the component (e.g., PMOS), then selecting Edit >> Properties >> Object Properties. Change the width to 10 and the length to 2 for both NMOS and PMOS transistors. Repeat this step for both transistors (see Figure 1.12).

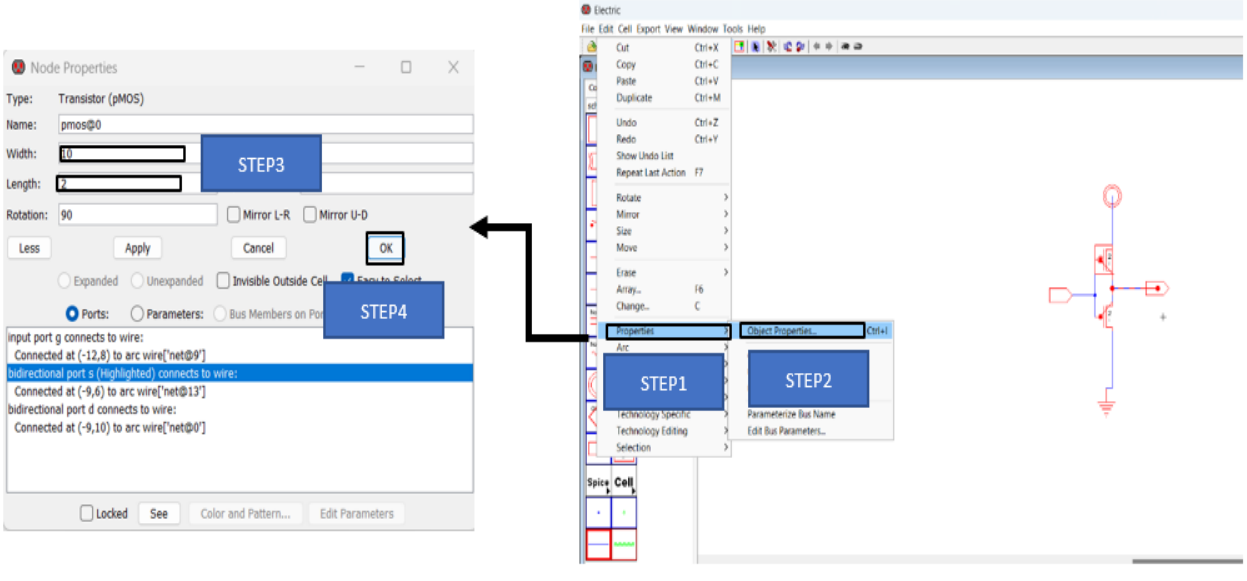

Figure 1.12: Adjusting the size of the transistors.

11. Create export for inputs and outputs: click on the (Input or output) then click on Export>>Create Export >> After that named the input or output >>then click OK. You can see the name shown in the circuit. repeat these steps for all inputs and outputs as shown in Figure 1.15.

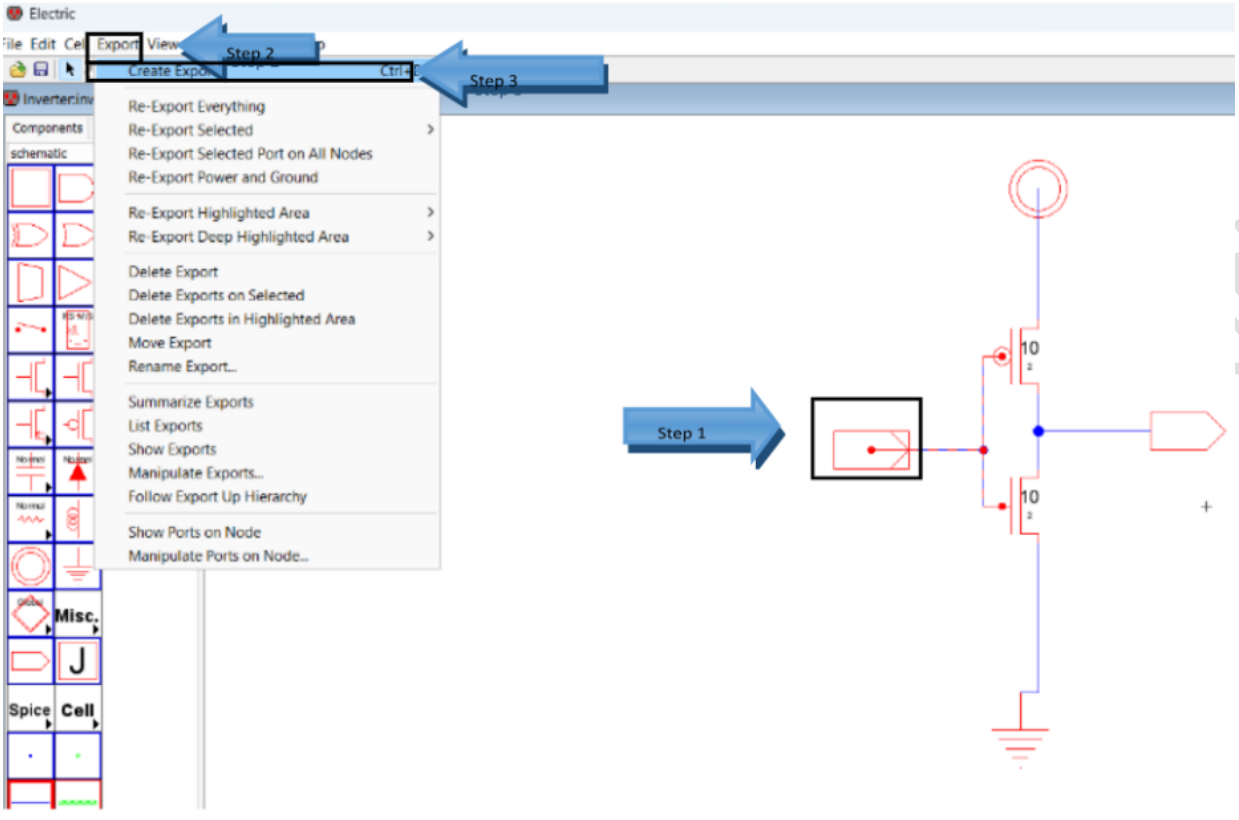

Figure 1.13: Create export for input.

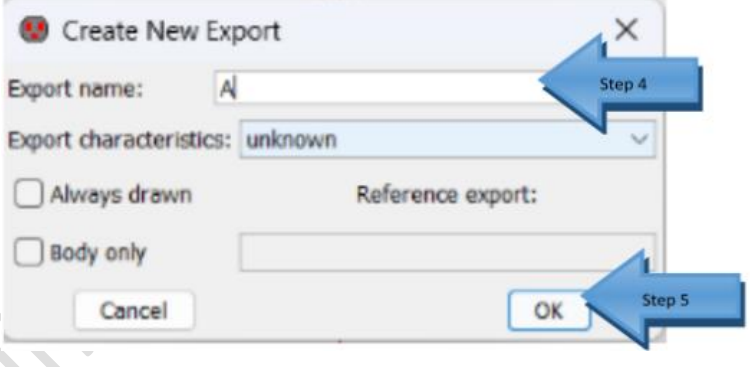

Figure 1.14: Named the input.

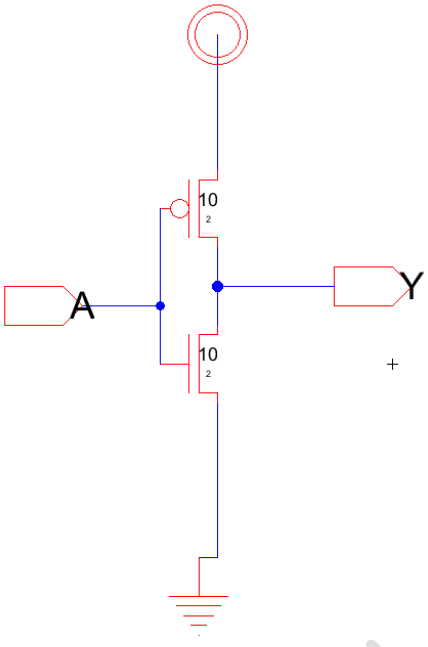

Figure 1.15: Named the input and output.

12. Now we must set the model of the NMOS AND PMOS transistors in the circuit. Click on the component (transistor)then click on tool >> Simulation (Spice)>> Set Spice Model. Then will the label be shown in the transistor as shown in Figure 1.17 (left). then we will change the label to NMOS or PMOS by clicking on the label as shown in Figure 1.17 (right). repeat these steps for all transistors as shown in Figure 1.18.

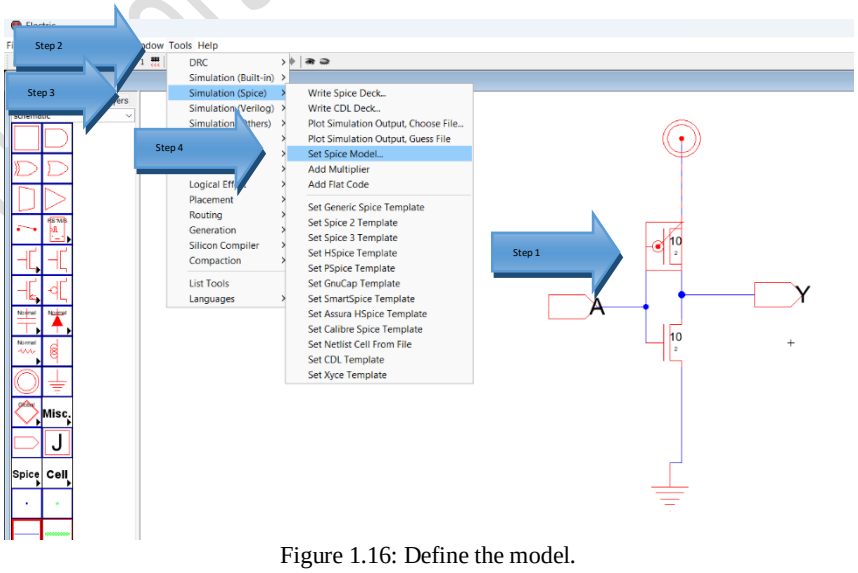

Figure 1.16: Define the model.

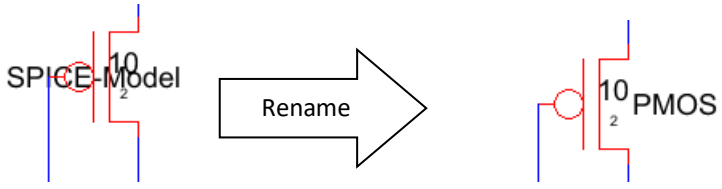

Figure 1.17: Label on the transistor.

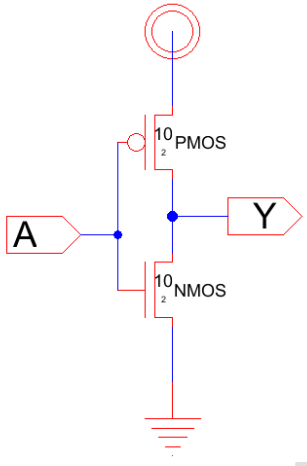

Figure 1.18: Put model for all transistors.

13. Write spice code to describe how the inputs change in waveforms. then click on the main windows. (See Figure 1.19).

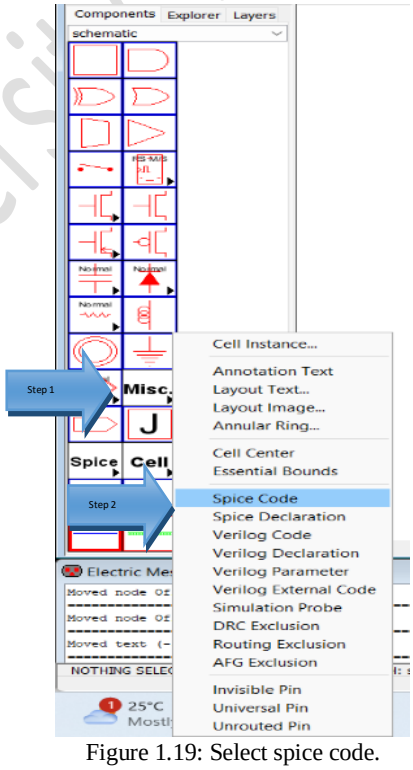

Figure 1.19: Select spice code.

14. Then write the below code by clicking on the text and then click on Edit>> Properties>>Object Properties then write the code then click OK. (See Figure 1.20).

```
vdd vdd 0 DC 5
vin A 0 pwl 10n 0 20n 5 50n 5 60n 0
cload out 0 250fF
.measure tran tf trig v(Y) val=4.5 fall=1 td=8ns trag v(Y) val=0.5 fall=1
.measure tran tr trig v(Y) val=0.5 rais=1 td=50ns trag v(Y) val=4.5 rais=1
.tran 0 0.1us
.include C:\Electric\C5_models.txt
```

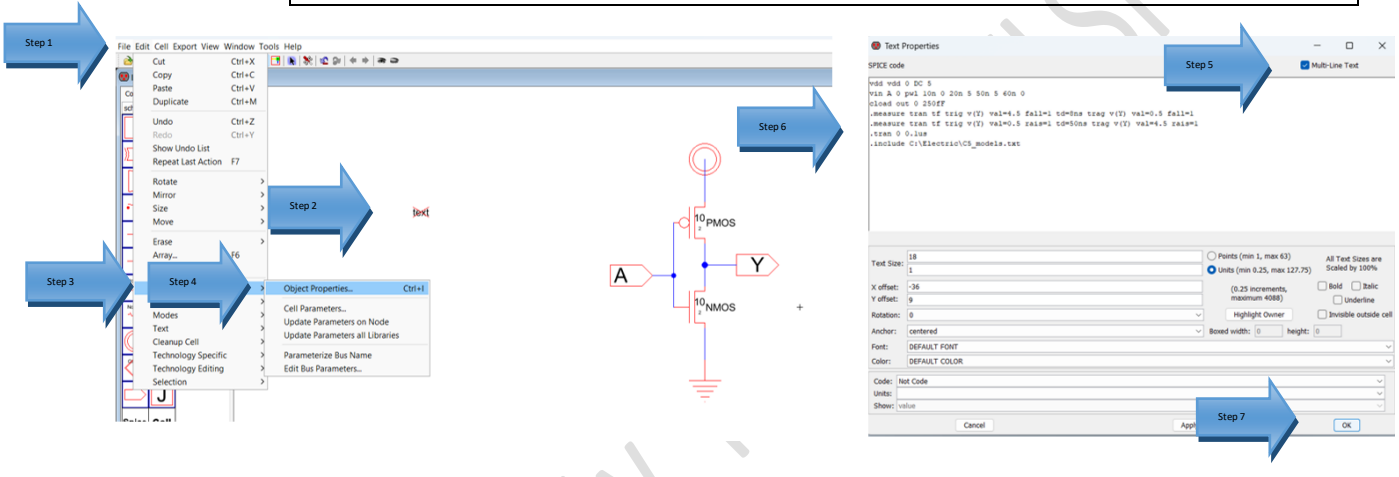

Figure 1.20: How to Write code.

15. Now, to simulate your code, click on Tools >> Simulation (Spice) >> Write Spice Deck. (See Figure 1.21).

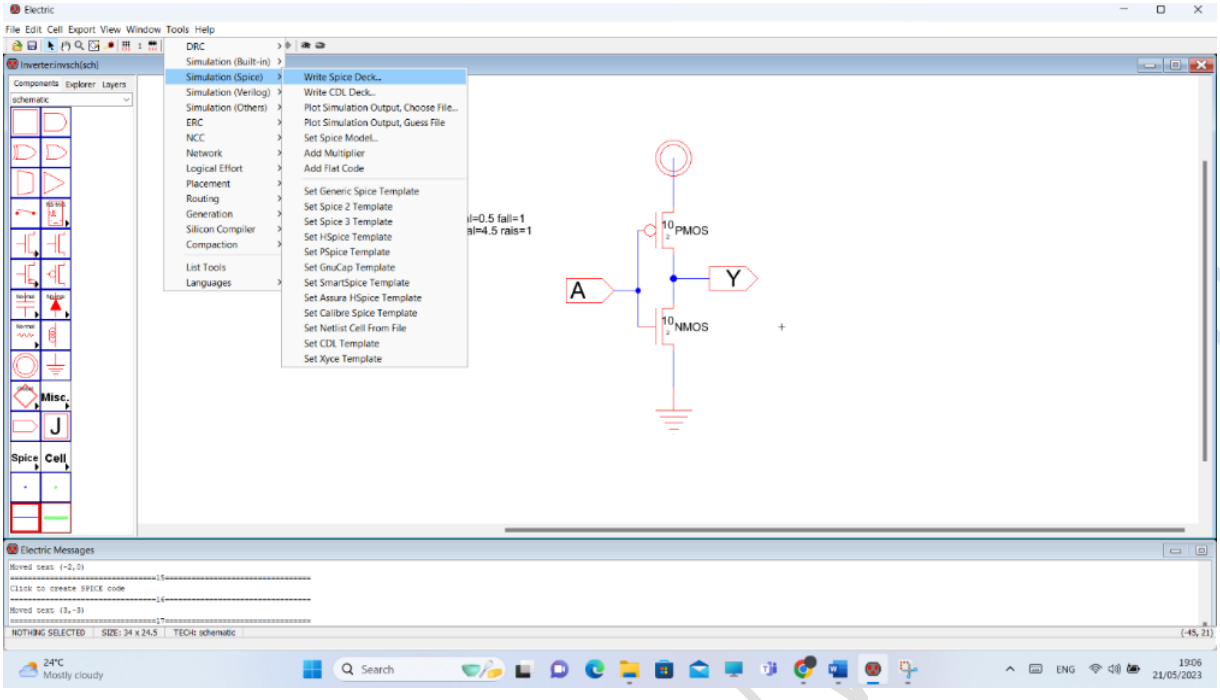

Figure 1.21 Simulation Code

16. LTspice XVII will open automatically, as Figure 1.22 shown:

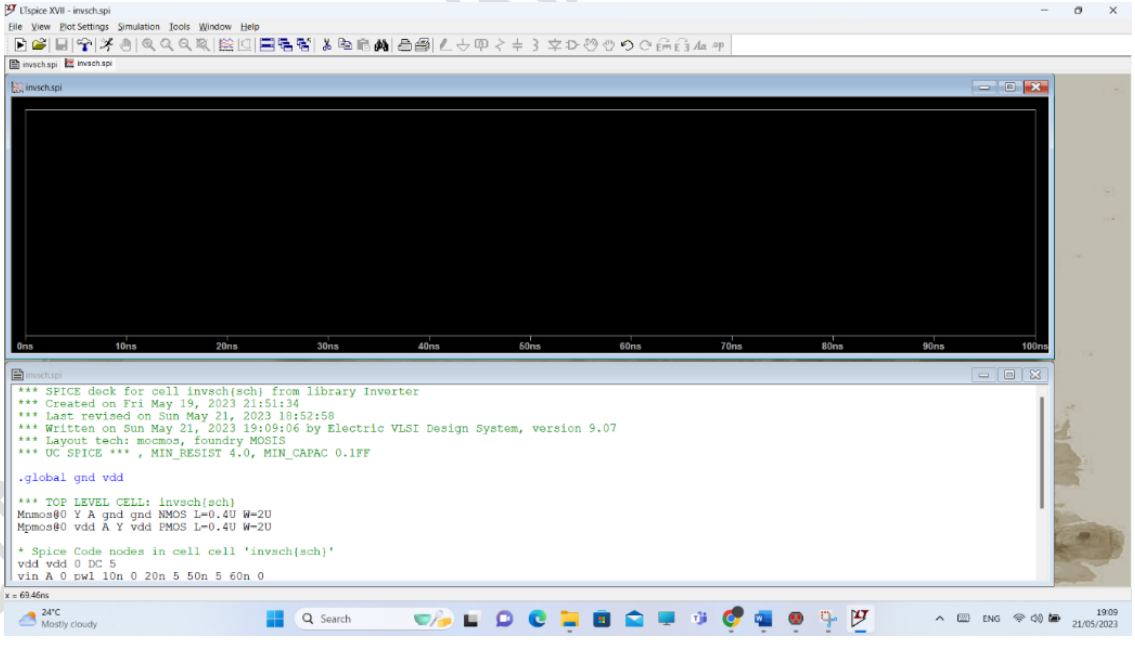

Figure 1.22: LTspice Window

17. To split your input and output b, right-click on the back window >> Add Plot Plan, as shown in Figure 1.23

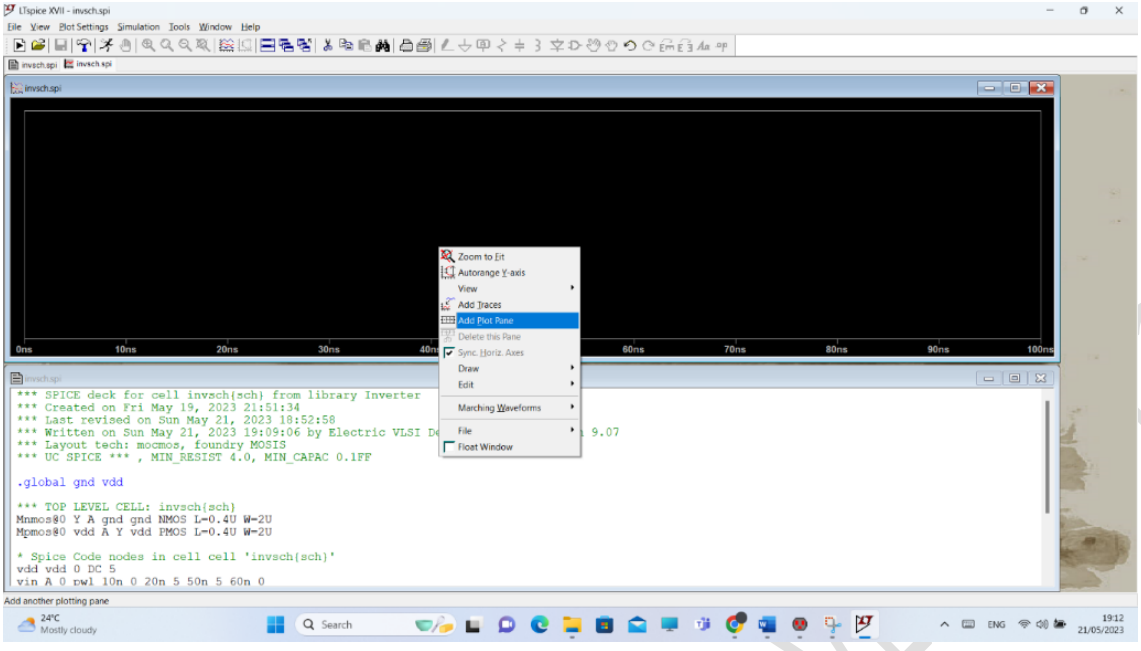

Figure 1.23: Add Plot Plan.

18. To add input & output waveforms, right-click >> on Add Traces >> select your input & output as you name it in the schematic step:

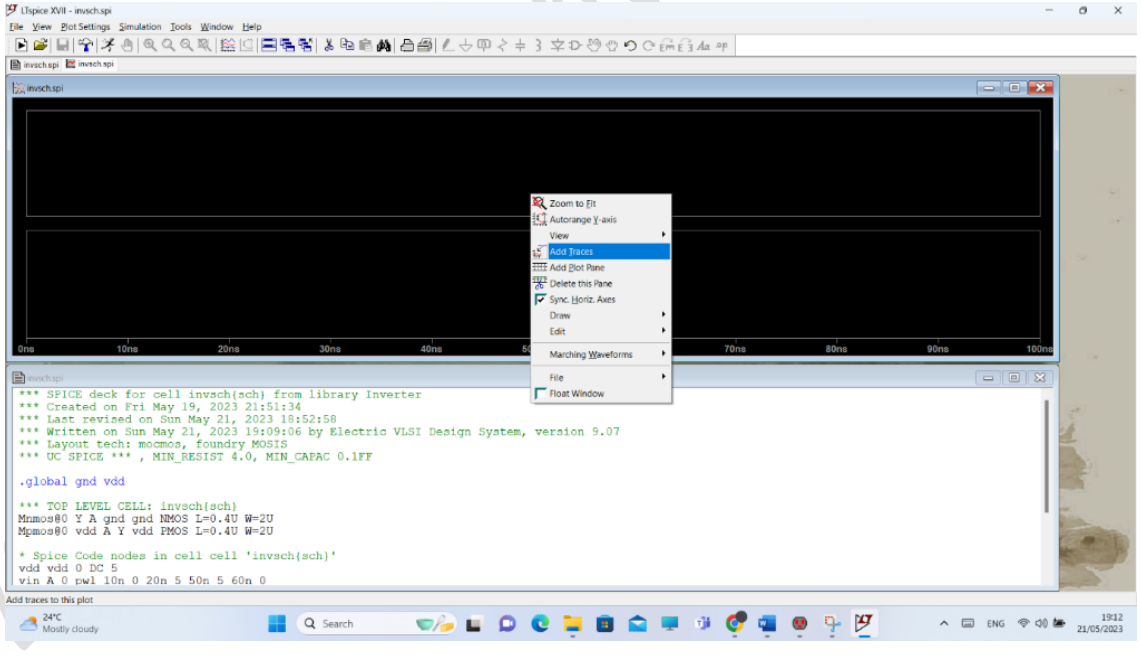

Figure 1.24: Select input & output.

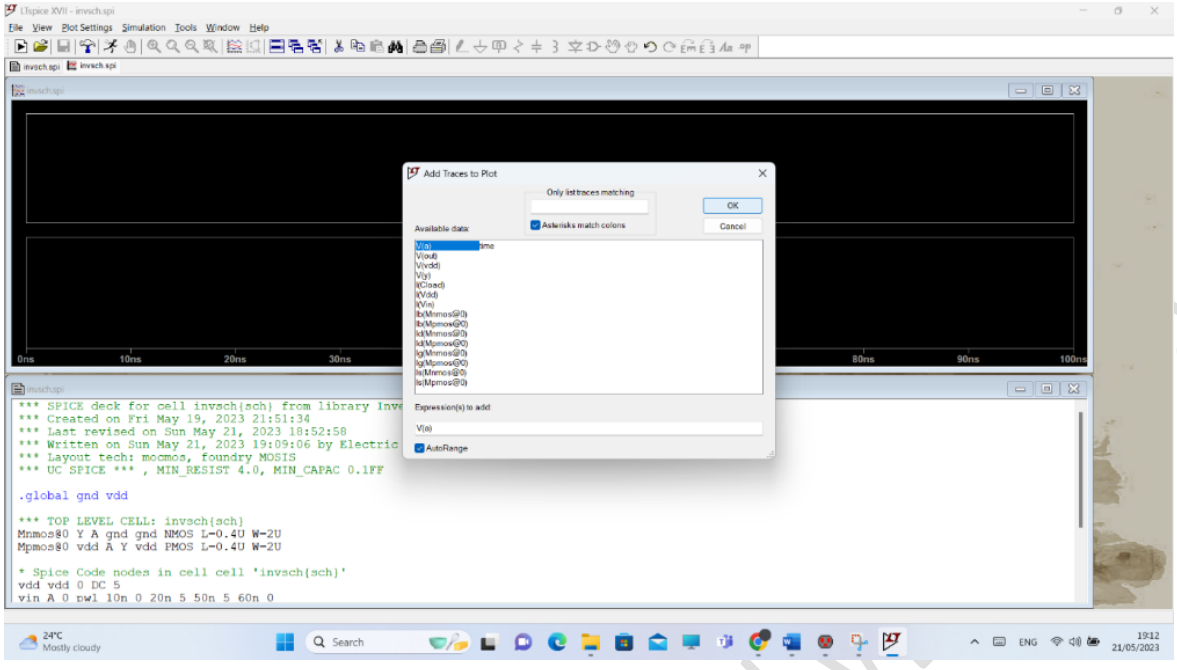

Figure 1.25: Select input in first plan.

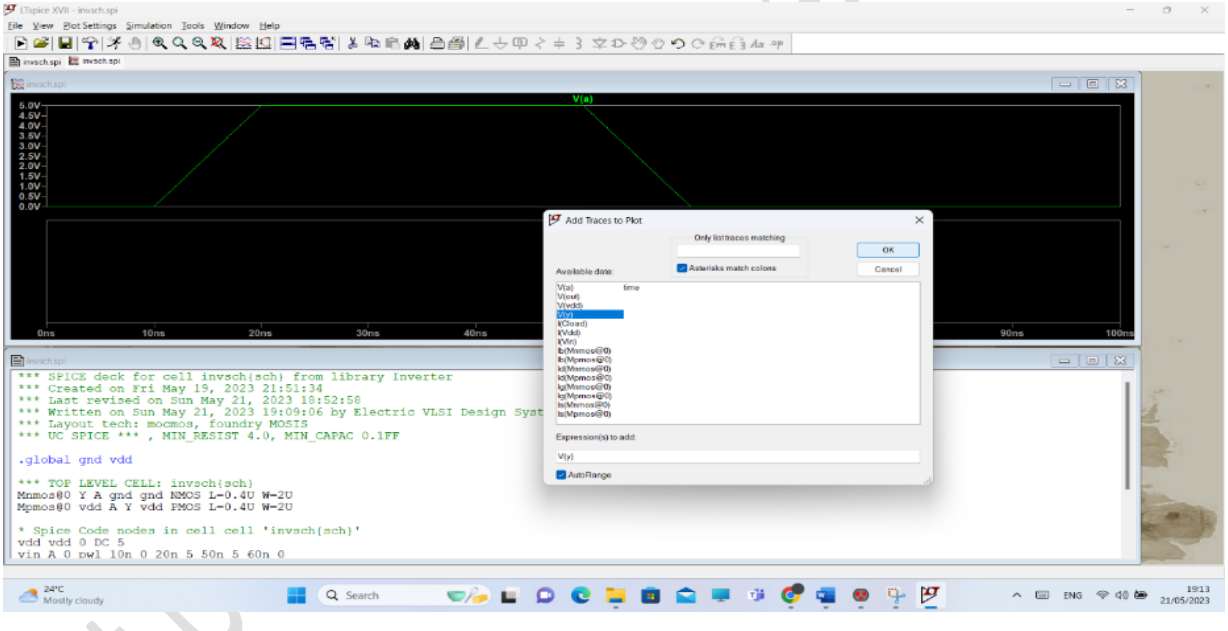

Figure 1.26: Select output in the second plan.

19. You will see the following window (see Figure 1.27):

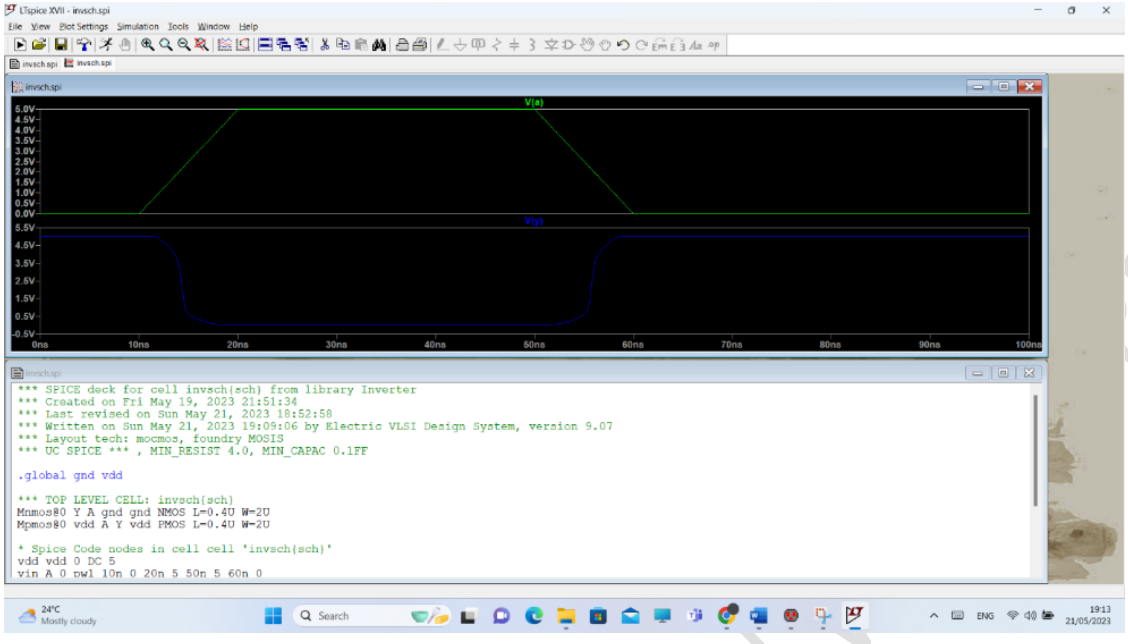

Figure 1.27: The final result.

20. To see rise and fall times for the output >> Spice Error Log:

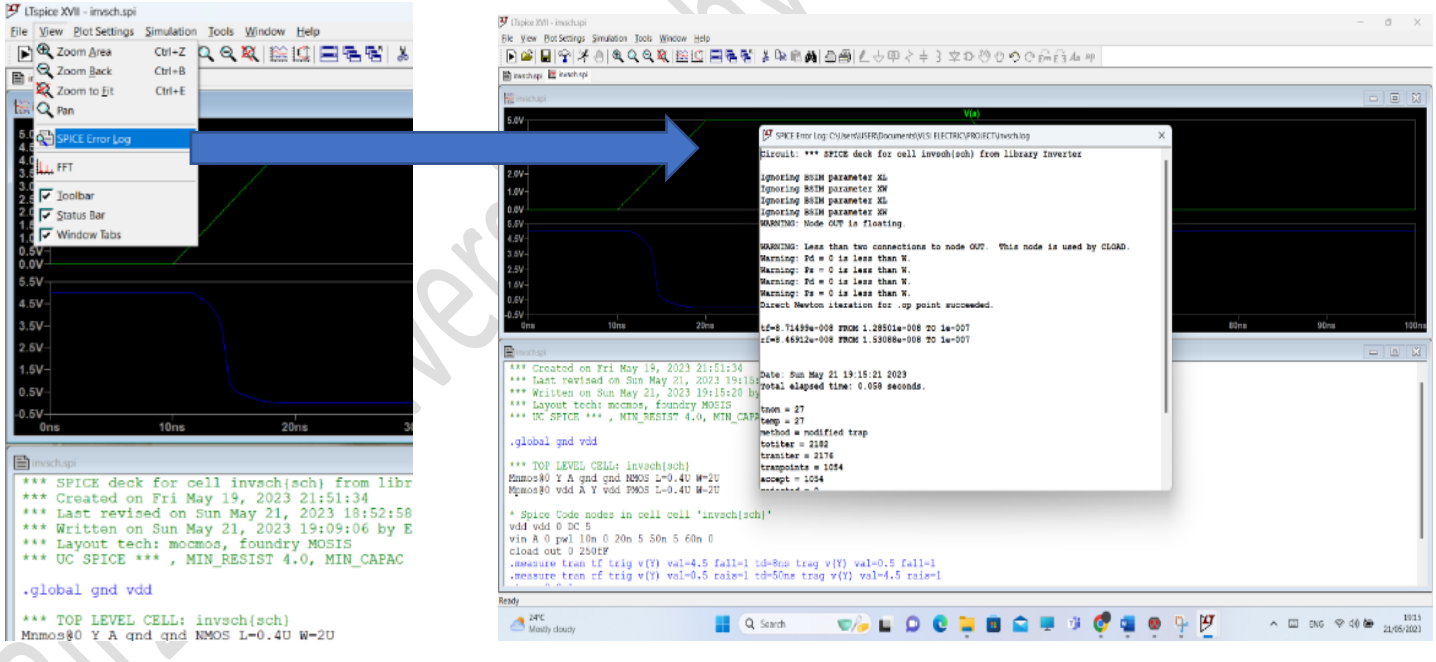

Figure 1.28: Find the fall time and rise time.

21. How to make an icon for your schematic View>> Make Icon View >> (see Figure 1.29):

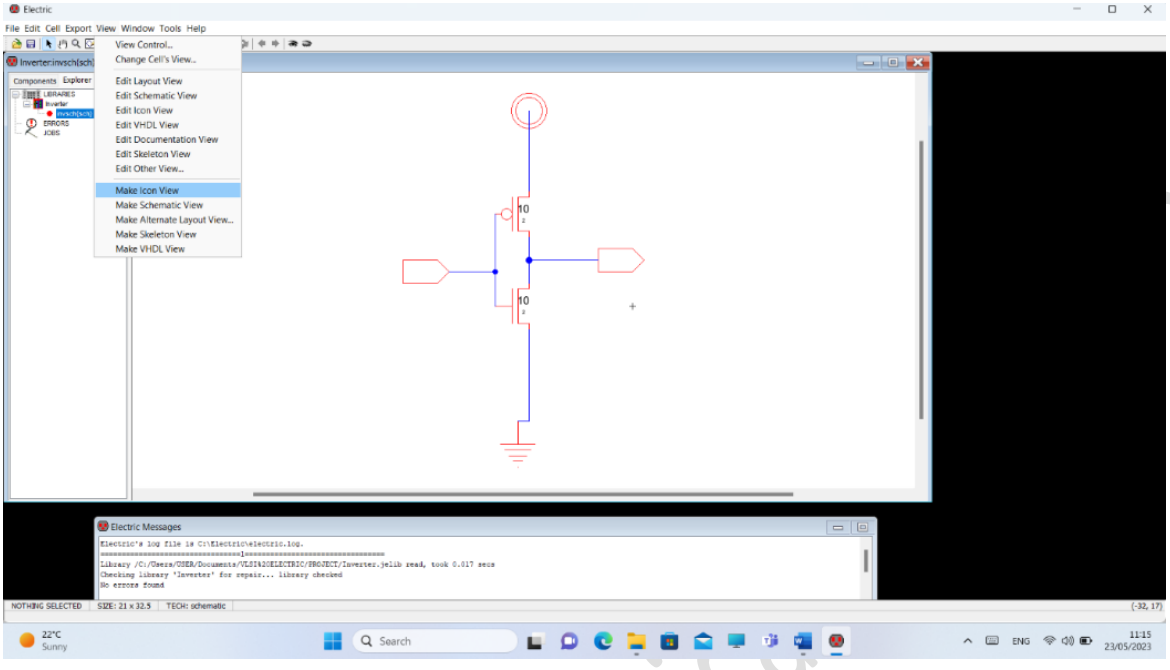

Figure 1.29:Making icon from the schematic.

22. To check if the icon has been done, go to Explorer in the left window and see if the invsch(ic) is found. If you click on it, you will see the icon as Figure 1.30 shown

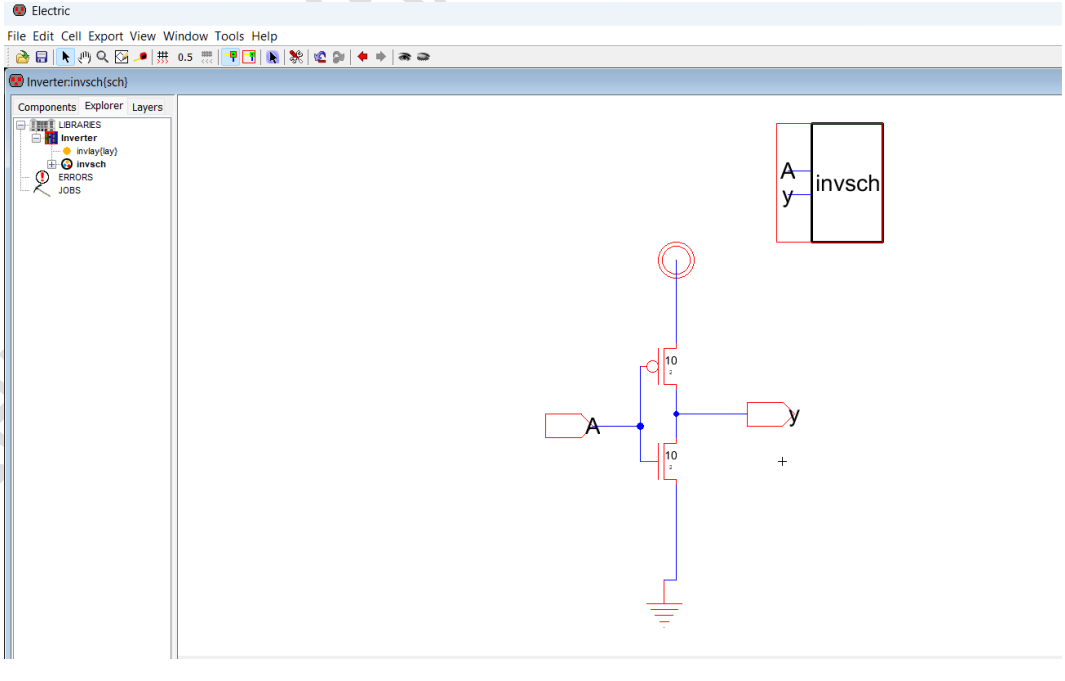

Figure 1.30: The icon from the schematic.

23. To change the icon shape, click on the component in the left-side window and choose the shape you want (see Figure 1.31).

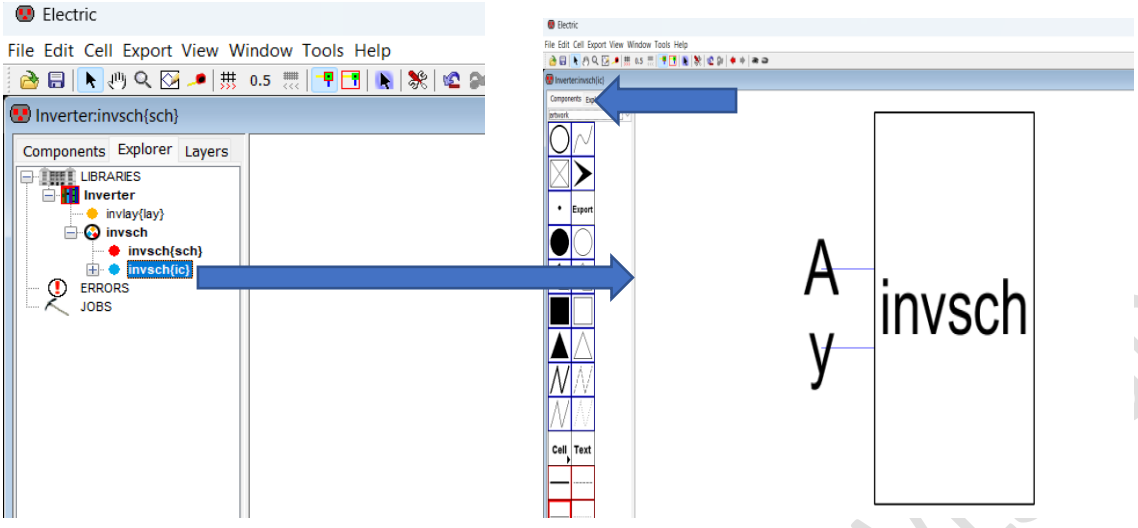

Figure 1.31: The icon file.

24. Change the shape using the existing component as shown in Figure below.

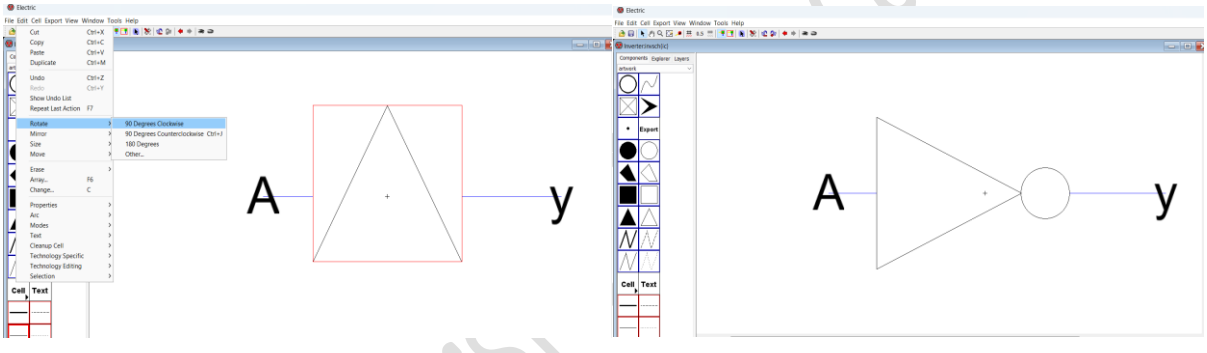

Figure 1.32:Change the inverter icon.

b) The CMOS Inverter Schematic:

1. To make a layout cell click on: Cell >> New Cell. Select the layout and name it, for example, invlay(lay) then click OK. (see Figure 1.33).

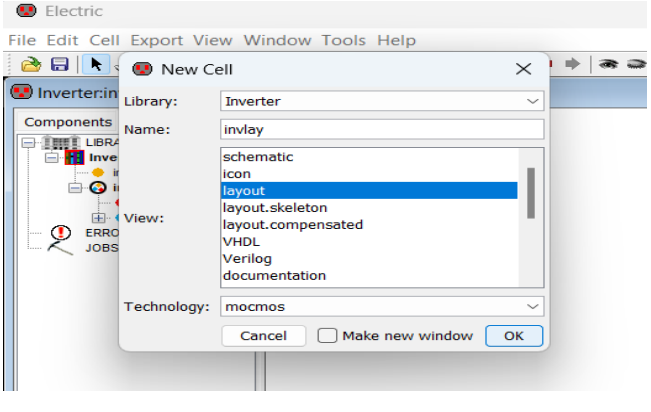

Figure 1.33: Selecting the schematic cell.

2. Now select the components needed (NMOS transistor, PMOS transistor, nWell, pWell, pAct, NAct). (see Figure 1.34).

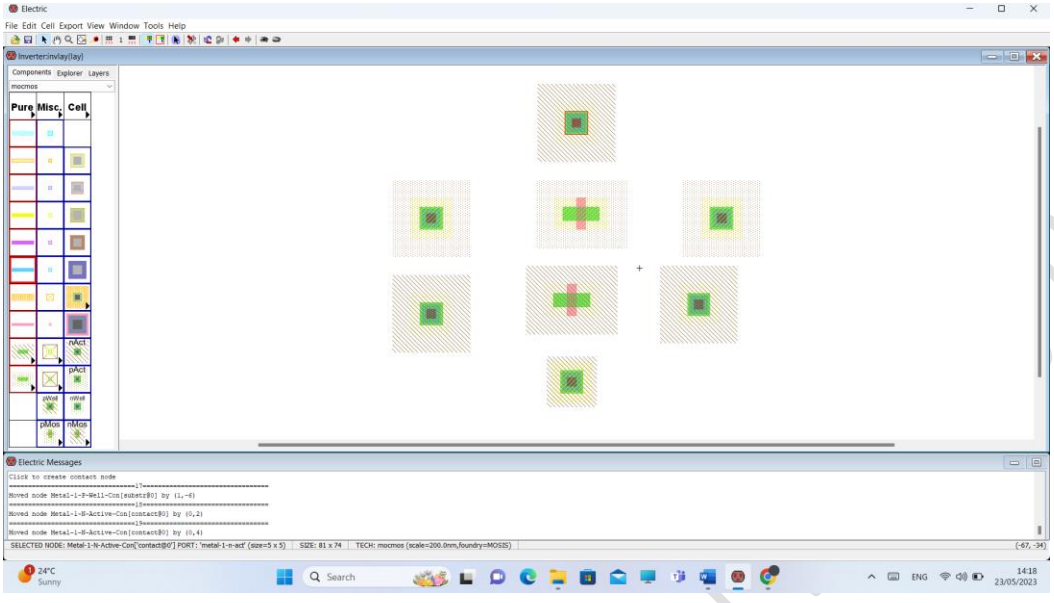

Figure 1.34: Selecting components for the layout.

3. Change the width to 10 and the length is set to 2 of the transistors NMOS and PMOS: click on the component you need to change such as PMOS then click on Edit>>properties >>Object Properties >> change the width and length >> OK. (see Figure 1.35).

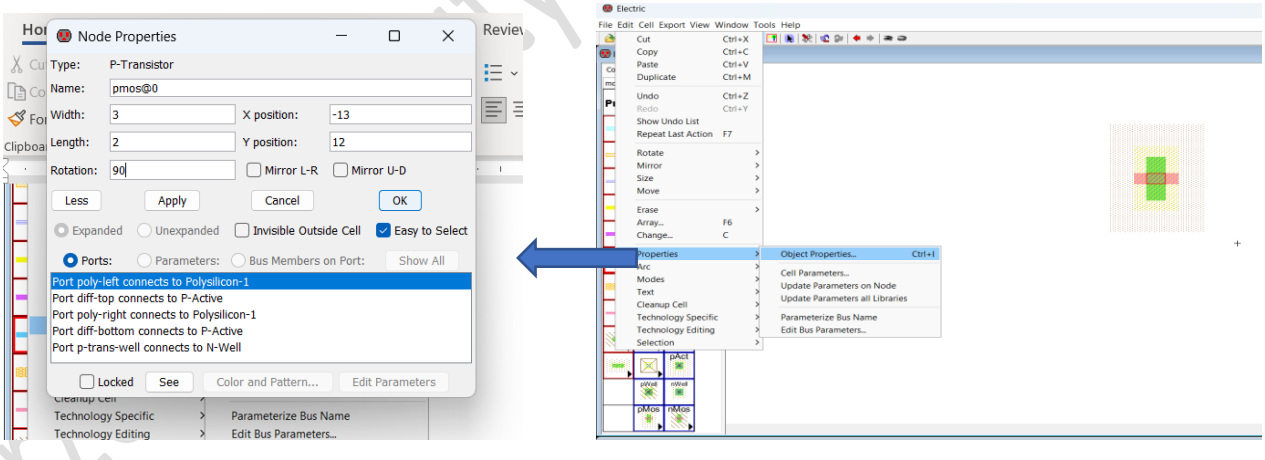

Figure 1.35: changing transistors size.

4. Change the width to 10 of pAct and nAct. click on the component you need to change such as pAct then click on Edit>>properties >>Object Properties >> change the Ysize >> OK. (see Figure 1.36).

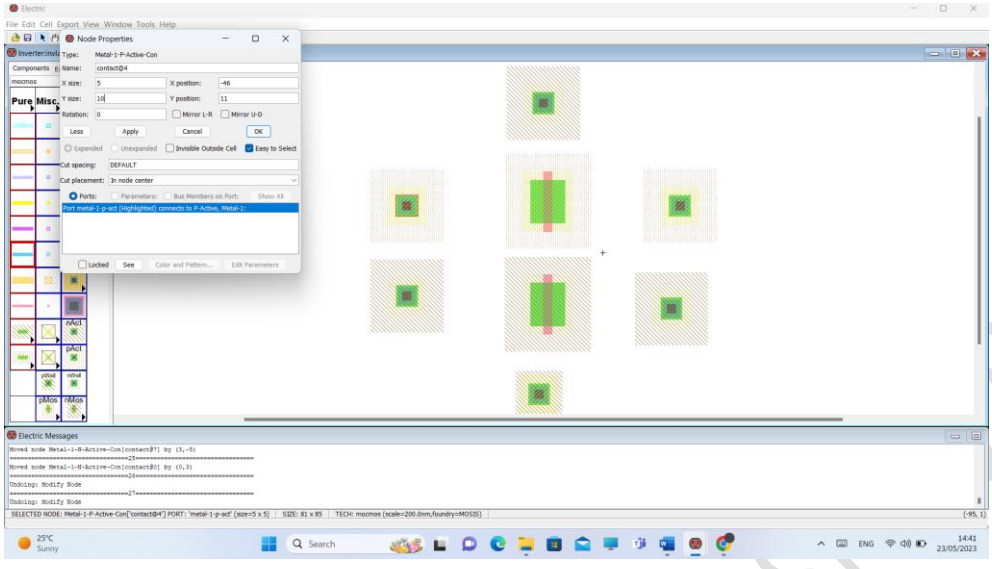

Figure 1.36: changing contact size.

5. Change the length to 20 of pwell and nwell. click on the component you need to change such as pAct then click on Edit>>properties >>Object Properties >> change the X size >> OK. (see Figure 1.37 and 1.38).

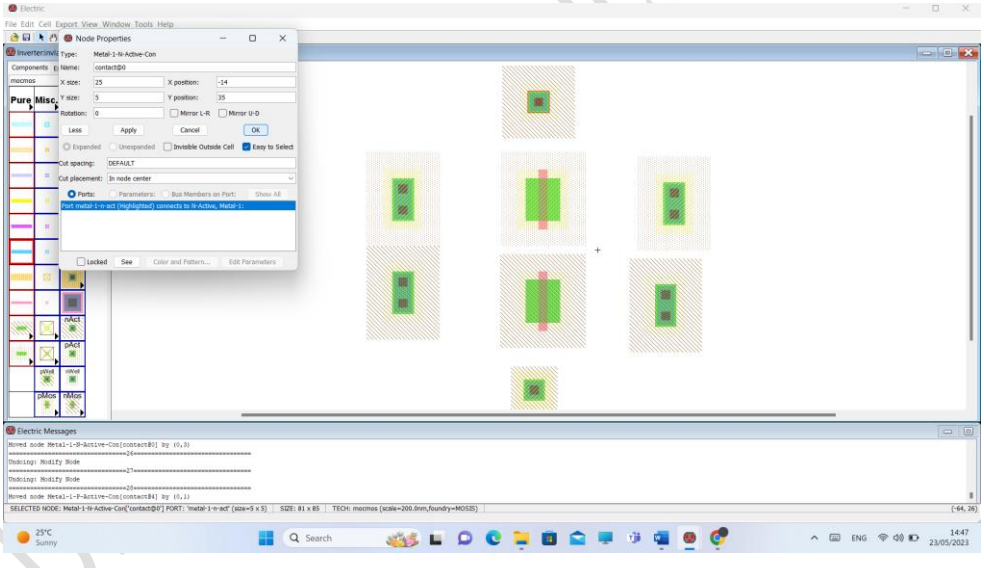

Figure 1.37: changing well size.

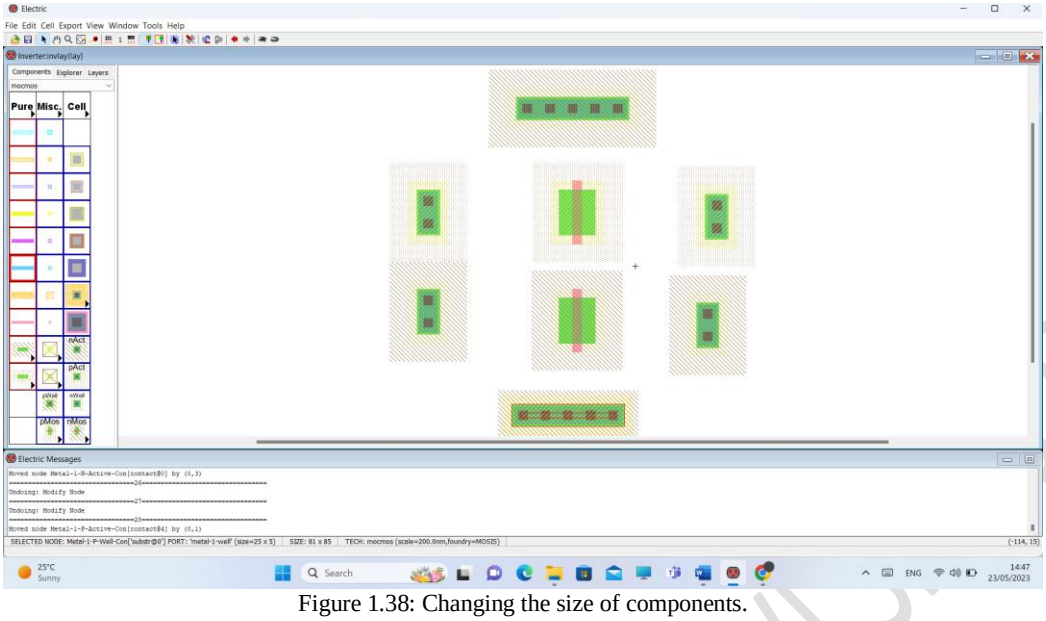

Figure 1.38: Changing the size of components.

6. Make connections between components (Click on the mouse's left button on one of the terminals of the component, then click on the terminal that you want to link with by clicking the mouse's right button). (See Figure 1.39).

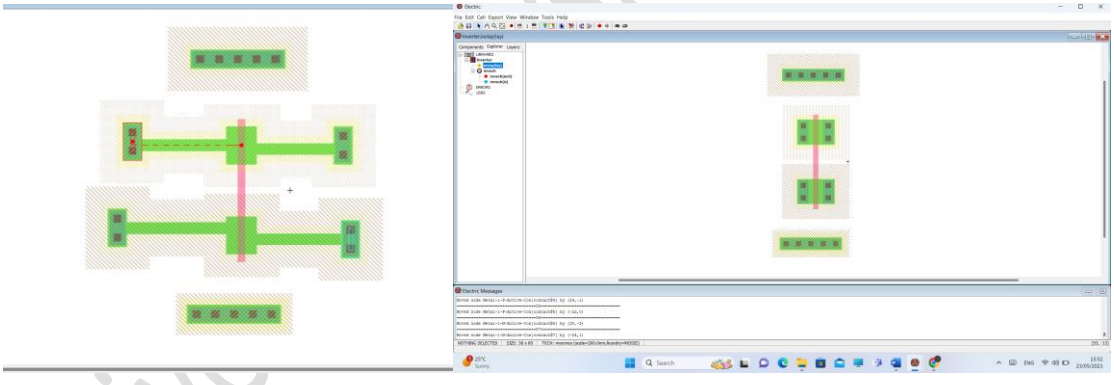

Figure 1.39: Connecting the components.

7. Connect all components as shown in Figure 1.40.

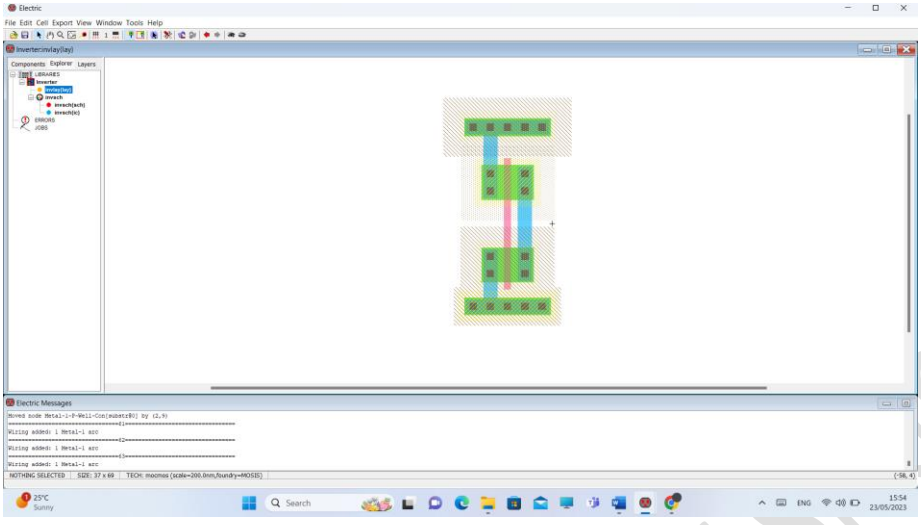

Figure 1.40:Make the connection between all components.

8. Check for error >> Tools>>DRC>>Check Hierarchically, if there are errors press **Shift + <** or **Shift + >.** (See Figure 1.41 and 1.42).

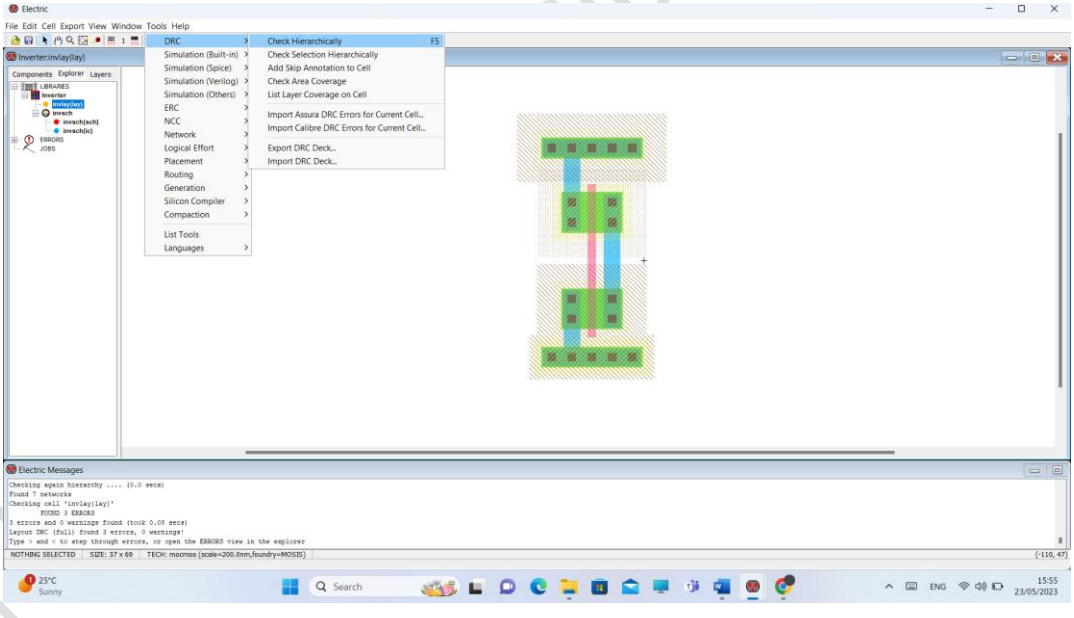

Figure 1.41: Chicking your design.

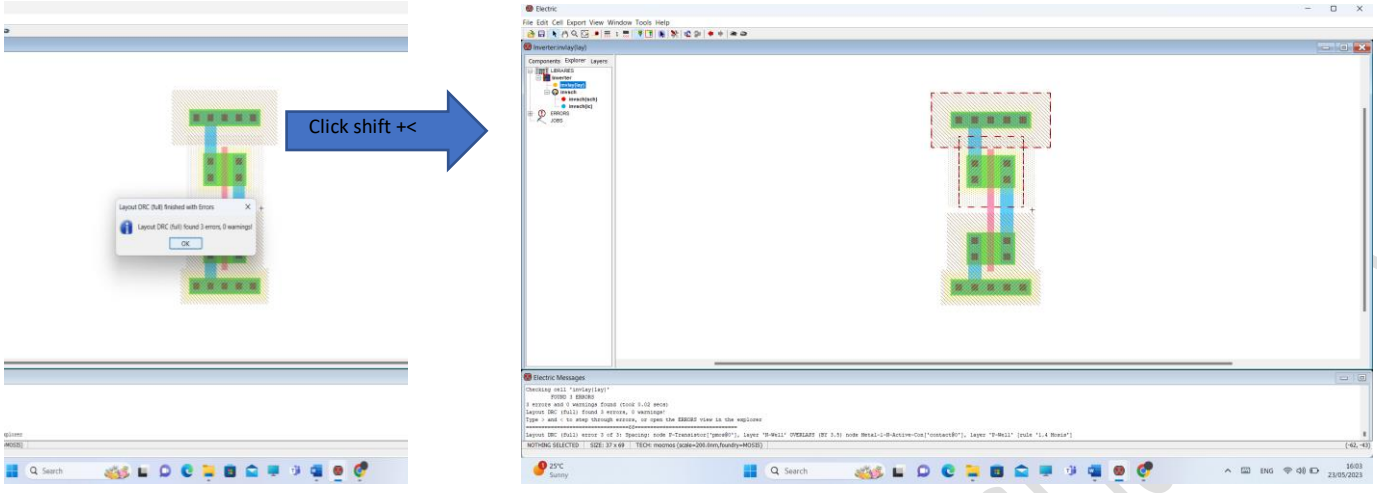

Figure 1.42: Finding the errors.

9. Put the contact for input as shown below. (see Figure 1.43).

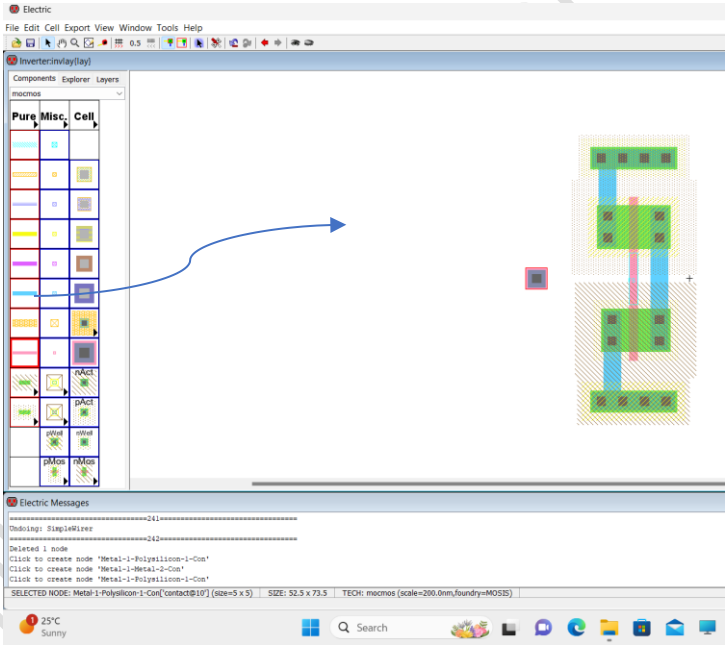

Figure 1.43: Adding a contact for input.

10. Create export for inputs and outputs: click on the (Input or output) then click on Export>>Create Export >> After that named the input or output >>then click OK. You can see the name shown in the circuit. repeat these steps for all inputs and outputs as shown in Figure 1.46.

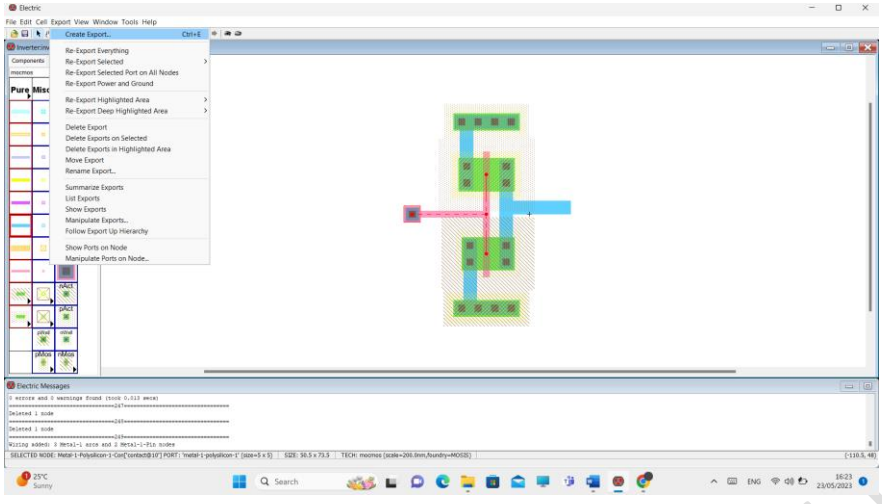

Figure 1.44: Create export for input.

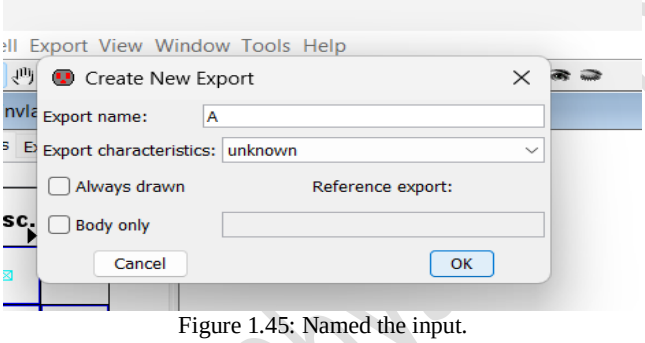

Figure 1.45: Named the input.

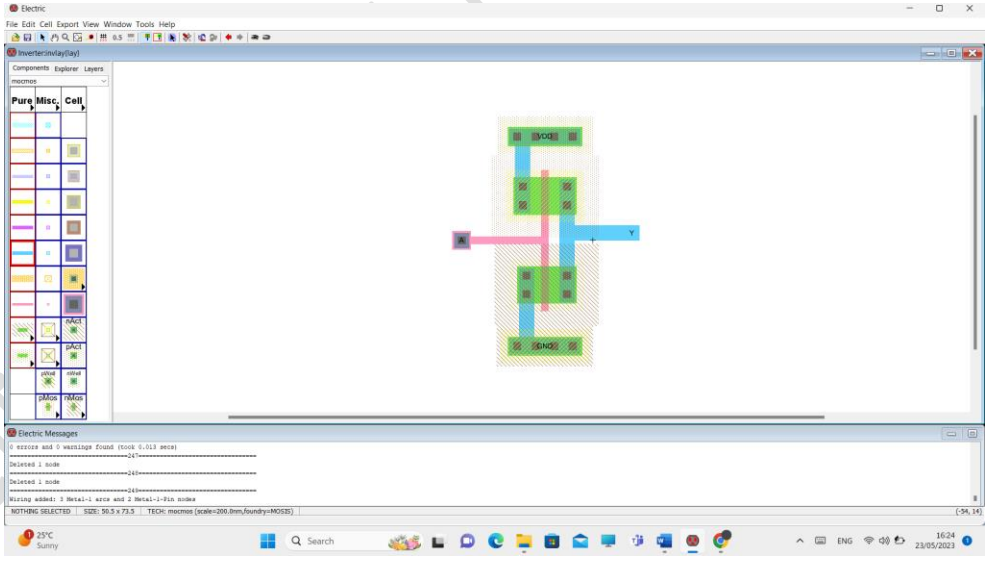

Figure 1.46: Named the input and output.

11. Now we must set the NMOS AND PMOS transistors model in the circuit. Click on the component (transistor)then click on tool >> Simulation (Spice)>> Set Spice Model. Then will the label shown in the transistor change the label to NMOS or PMOS by clicking on the label. repeat these steps for all transistors as shown in Figure 1.48.

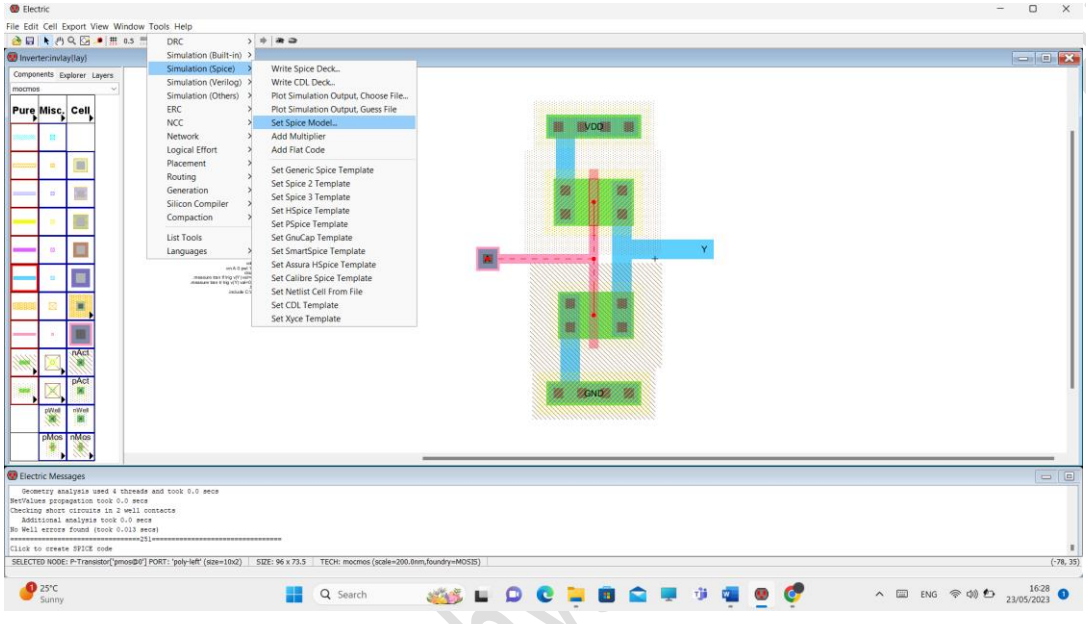

Figure 1.47: Define the model.

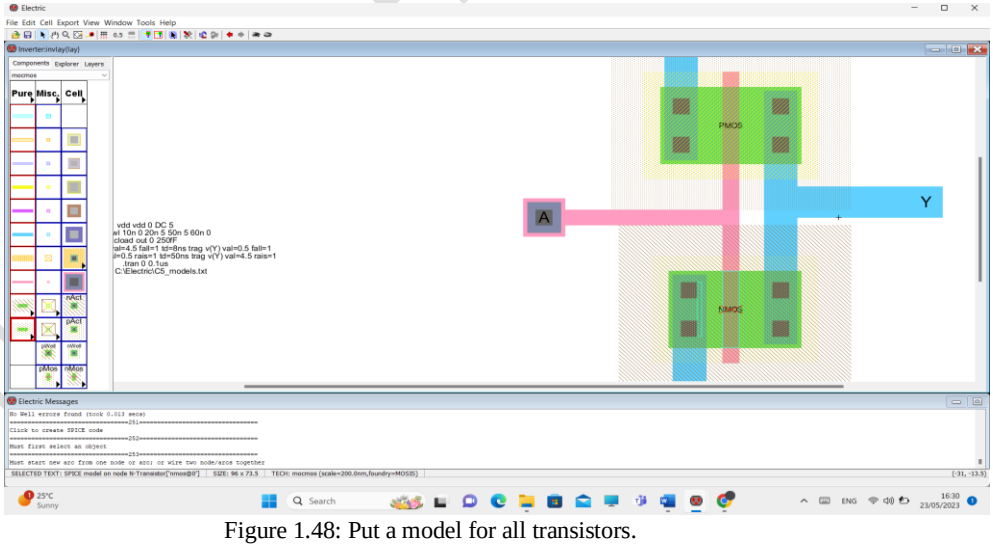

Figure 1.48: Put a model for all transistors.

12. Write spice code to describe how the inputs change in waveforms. Click on the component on the left side >>.MISC >> Spice code .then click on the main windows. (see Figure 1.48).

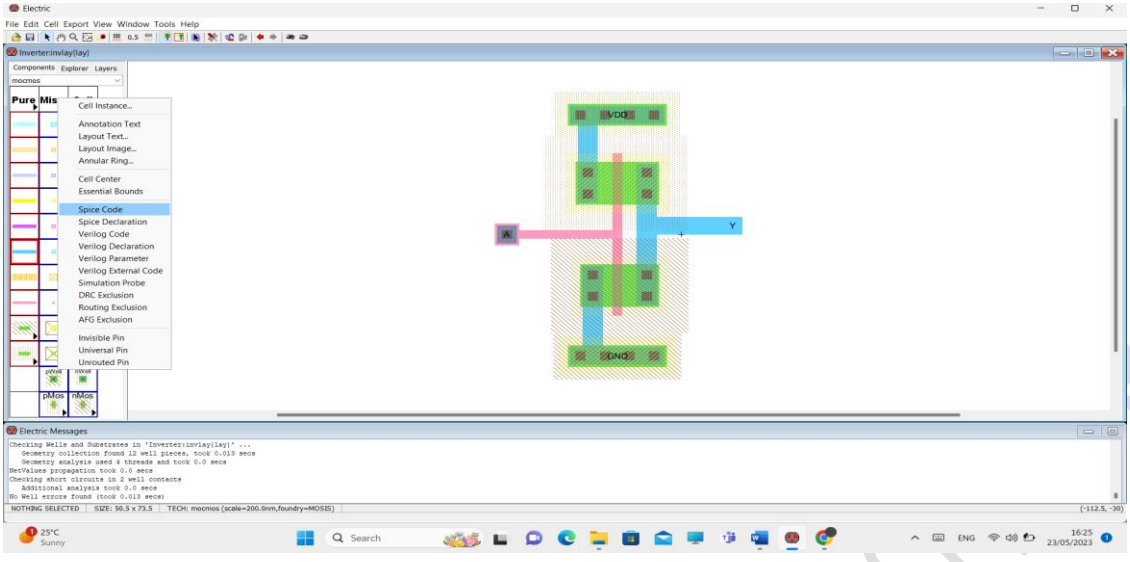

Figure 1.48: Select spice code.

13. Then write the below code by clicking on the text and then click on Edit>> Properties>>Object Properties then write the code then click OK.

```
vdd vdd 0 DC 5
vin A 0 pwl 10n 0 20n 5 50n 5 60n 0
cload out 0 250fF
.measure tran tf trig v(Y) val=4.5 fall=1 td=8ns trag v(Y) val=0.5 fall=1
.measure tran tr trig v(Y) val=0.5 rais=1 td=50ns trag v(Y) val=4.5 rais=1
.tran 0 0.1us
.include C:\Electric\C5_models.txt
```

14. Now, to simulate your code, click on Tools >> Simulation (Spice) >> Write Spice Deck. then repeat step 16,17,18,19,20 and show the result.

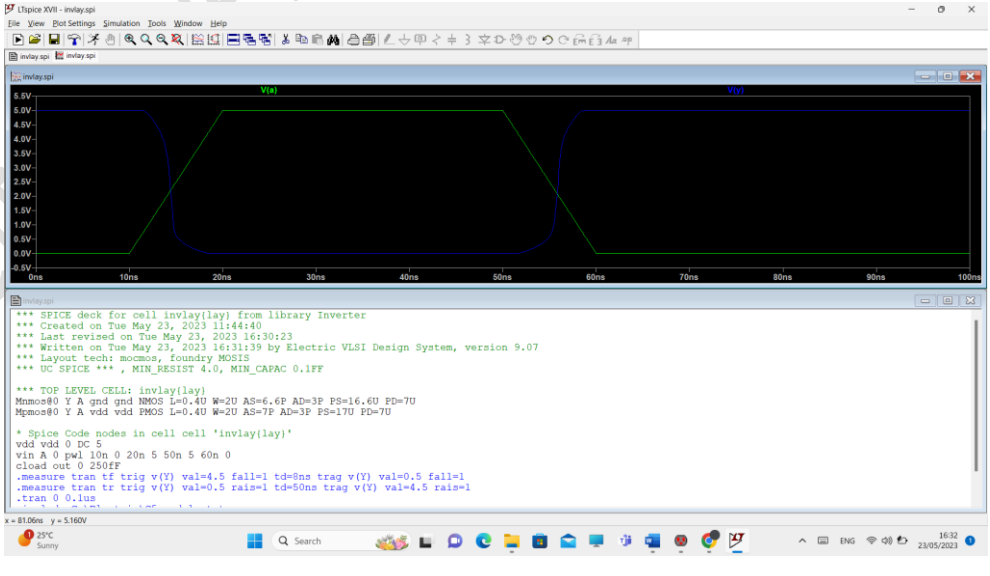

Figure 1.49: Waveform of Not gate.

1.6 Lab work:

1) Design, layout, and simulation of CMOS NAND gate:

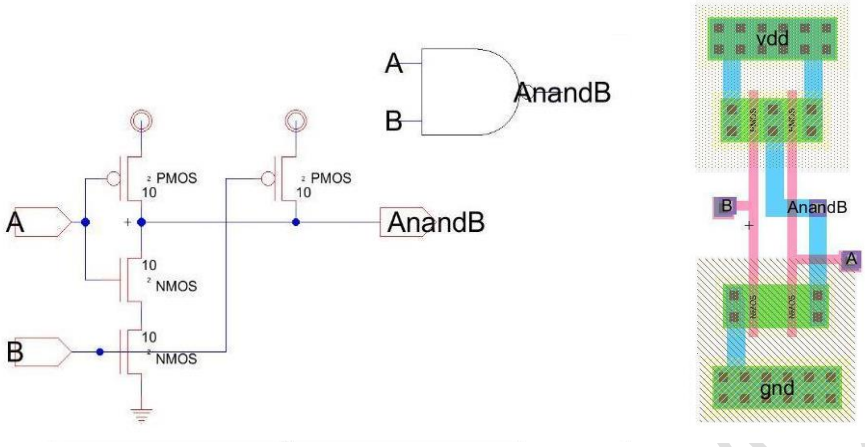

Figure 1.50: The schematic and layout for NAND gate.

2) Design, layout, and simulation of CMOS XOR gate:

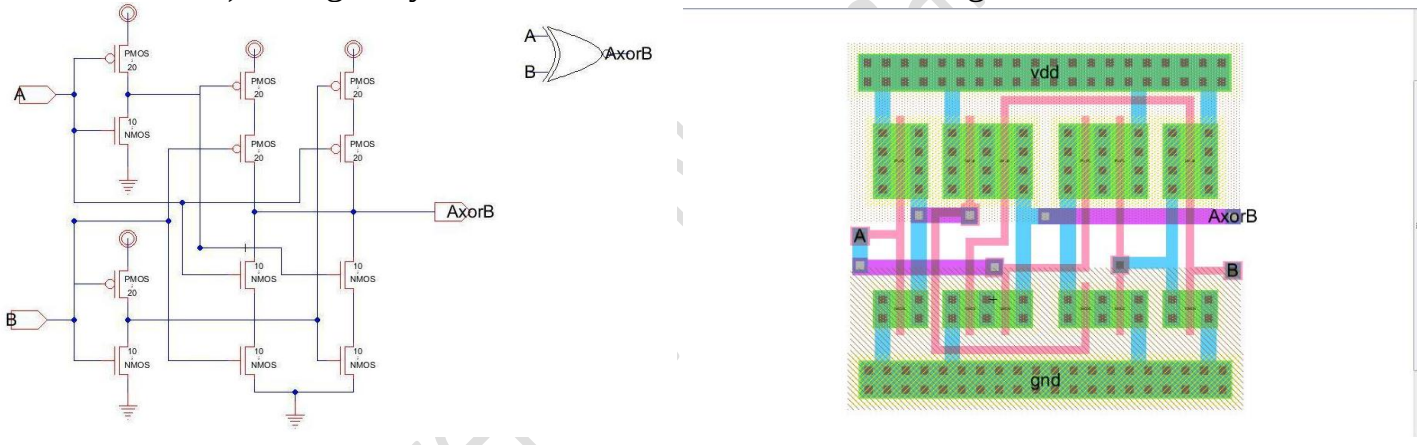

Figure 1.51: The schematic and layout for NOR gate.

3) Design, layout, and simulation of CMOS Full adder gate using the previous components:

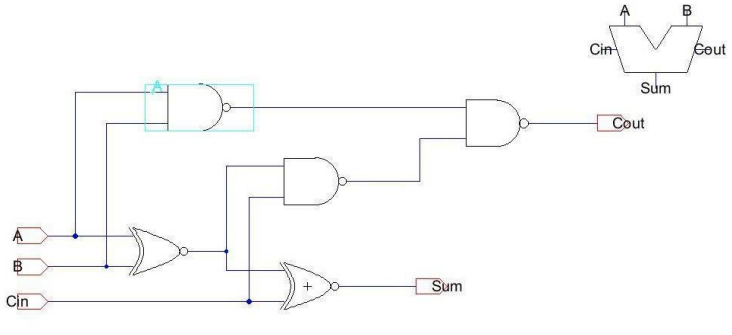

Figure 1.52: the schematic for Full adder.

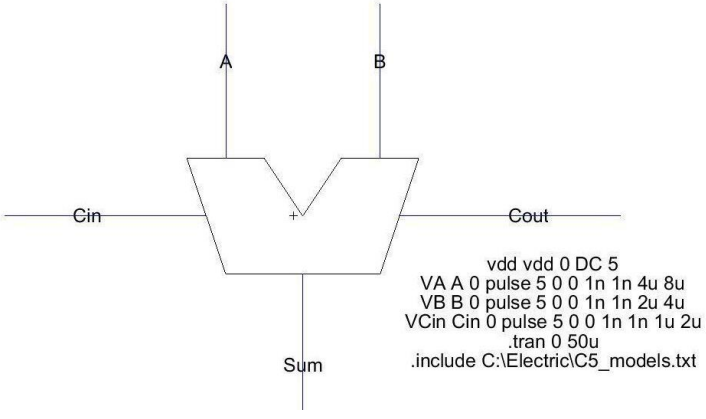

Figure 1.53: The icon for Full adder.

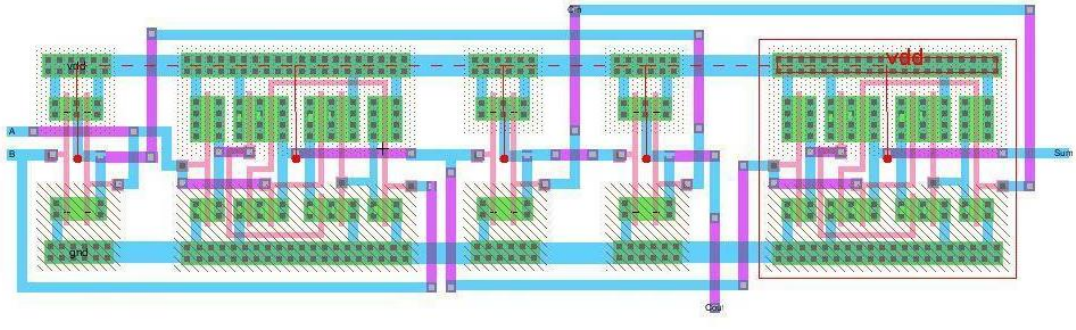

Figure 1.54: The layout for Full adder.

1.7 References:

[1] https://staticfreesoft.com/productsFree.html [Accessed 1/8/2023]

Experiment No. 2 - Design of a Low Power and Delay 4-Bit CAM using 9T SRAM

2.1 Objectives

- Design a 4-bit content addressable memory (CAM) using 9-Transistors SRAM.
- How to balance between low power and low delay will be taken into consideration.
- How to make the schematic, layout, and waveform for SRAM.

2.2 Equipment Required

- Electric VLSI Design System Version 9.07.

2.3 Pre-Lab

1. Read the experiment to learn about SRAM.

2.4 Introduction

2.4.1 Random Access Memory

RAM, or (Random Access Memory) is an important hardware component of a computer's main memory, allowing for direct access by the Central Processing Unit (CPU). It is used to read or write data in a targeted address on the memory. In addition, RAM serves as a temporary store for the running programs that are being used by the processor. RAM is a volatile type of memory, meaning the information will be lost when the power is turned off. When searching in RAM, it takes the address of the data as an input and returns the content word of that address; this operation is done by many cycles until getting the required data stored in RAM.

Content Addressable Memory (CAM), also referred Associative Memory, that performs similar operations of RAM that include the read and write on a given address, but it has additional searching functionalities; it takes the data as input, searches in its memory about that data by using a parallel comparison between the input word data and the whole memory, and returns the list of addresses where the data word was located. As shown in the following Figure 2.1, the data word (search word) is the input and the search line for the parallel comparison; if the data stored in the memory cell matches the incoming data, the match line will be activated. The encoder outputs an encoded version of the match location using log2 w bits.

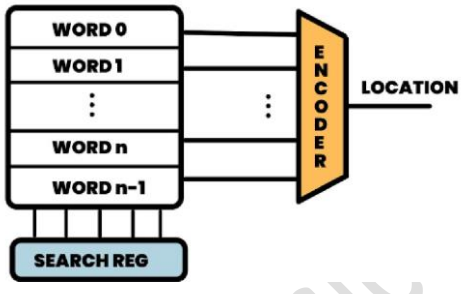

Figure 2.1: Conceptual view of a CAM [1].

As a functionality comparison between both RAM, and CAM; the CAM is an inverse operation of RAM in reading operation as shown in Figure 2.2.

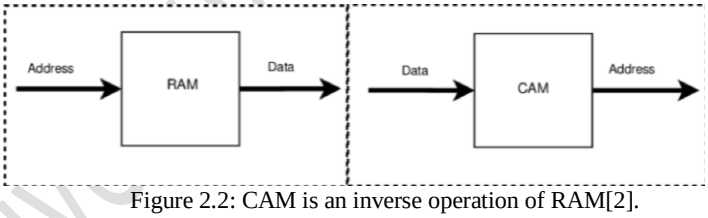

Figure 2.2: CAM is an inverse operation of RAM[2].

In terms of speed, when reading data words using the traditional RAM, multiple clock cycles are needed to find the address of the input data. While CAM can perform the search operation in parallel as mentioned in one single cycle, which makes CAMs have faster search times compared to RAMs, which makes the CAM widely used in database management systems and network switching that require fast searching functionalities which CAM offers. However, the speed of a CAM comes at the cost of increased silicon area and power consumption compared with the normal RAM due to the additional circuit required to make comparisons and generate the search and match lines.

Implementing CAM can be done using the SRAM schematic since it performs the same operations as SRAM. In addition, the SRAM can be implemented using different

numbers of transistors, such as 6T, 8T, 9T, and 10T all depending on the requirements and the specifications of the application or project. In our project, the CAM will be designed and implemented using 9T SRAM. The following Figure 2.3 shows the 9T SRAM schematic.

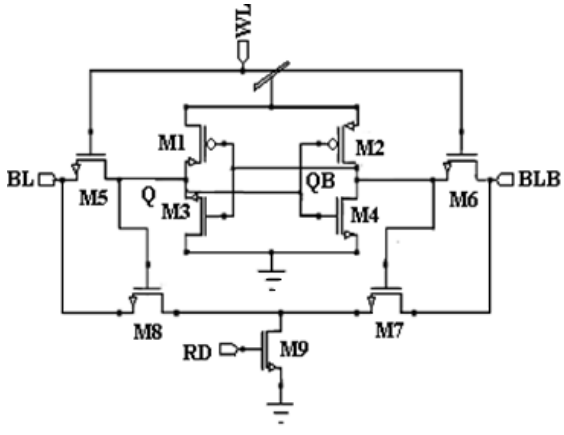

Figure 2.3: Schematic of 9T SRAM Circuit.

2.5 Procedure

To implement 4-bit CAM using 9T SRAM, several components are required to be designed and built independently and then used later in the 4-bit CAM schematic circuit as one block. These components are as the following:

2.5.1 4-bit CAM:

a) 9T SRAM schematic and layout:

As shown in the following figure, the 9T SRAM circuit has 4 input lines, which are the word line (WL), (RL), and other 2 bits lines (BL and BLB). While they have two outputs (QB), and (Q).

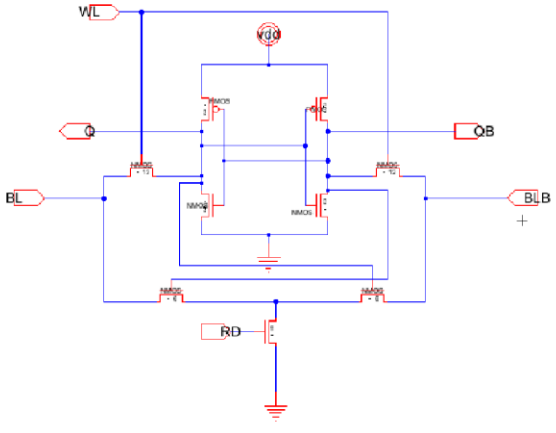

Figure 2.4: Schematic of 9T SRAM Circuit.

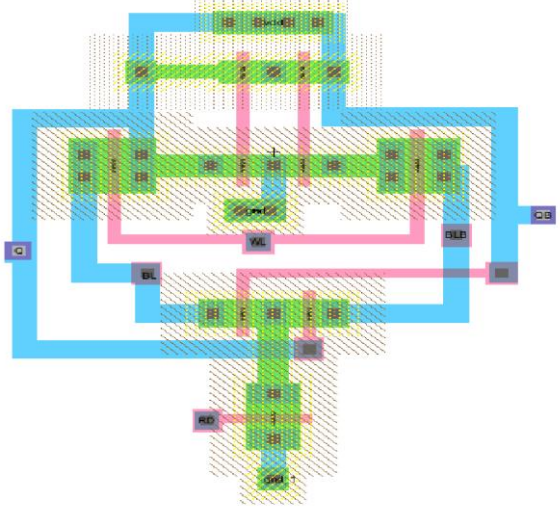

Figure 2.5: The layout of 9T SRAM Circuit.

b) 9T SRAM Simulation:

In SRAM memory, the write operation is performed when the write line, also known as the write, enable or WE line, is in a high state. This line acts as a control signal for the memory, determining whether the memory cells are in a read-only or write mode. When the write line is high, the memory cells can receive and store new data. Conversely, when the write line is low, the memory cells are in a "read-only" state, and any attempts to write to the memory will be ignored. This mechanism ensures the stability and integrity of the stored data while allowing for dynamic updates as needed.

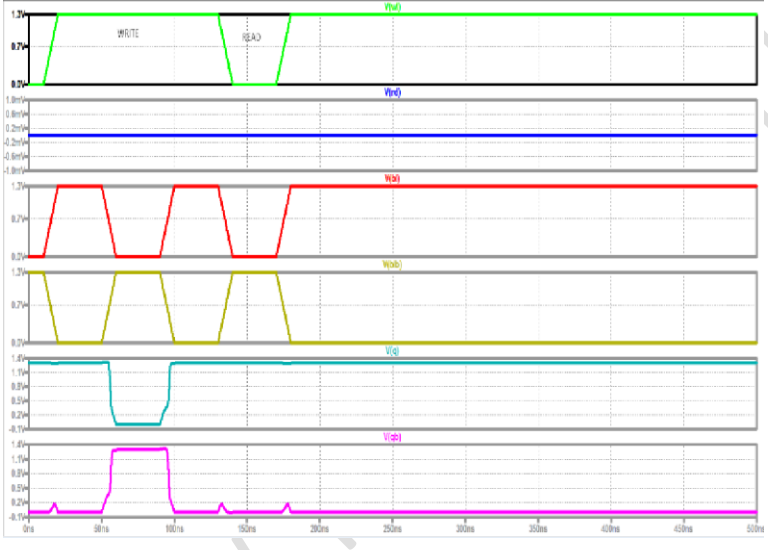

Figure 2.6: The simulation of 9T SRAM Circuit.

c) 1-Bit CAM Schematic and Layout:

Then, the circuit of the 1-bit CAM was implemented as shown in the following circuit: two additional inventors were added, and two pass gates for the comparison operation. And finally, the output of the circuit that represents the match signal.

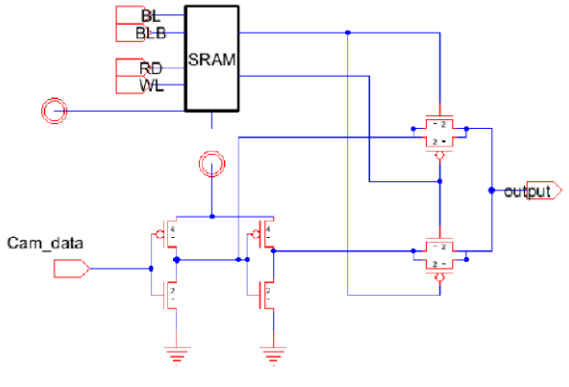

Figure 2.7: Schematic of 1-Bit CAM Circuit.

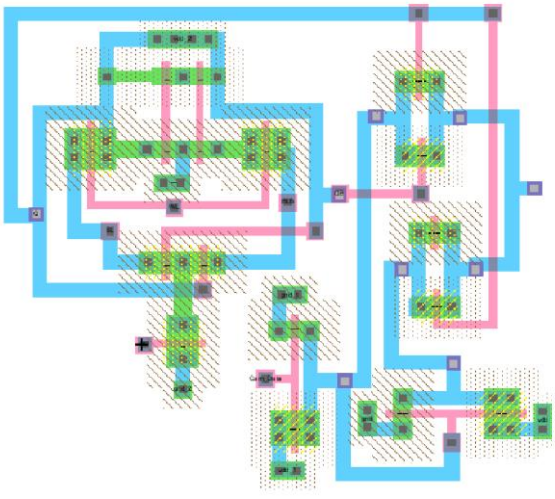

Figure 2.8: The layout of 1-Bit CAM Circuit.

d) 2x4 Decoder Schematic and Layout:

In the 4-bit CAM implementation, the 2x4 decoder is needed, because it helps in selecting the appropriate 1-Bit CAM cell from the 4-bits for the search operation. The following figures represent the schematic and the layout of the decoder.

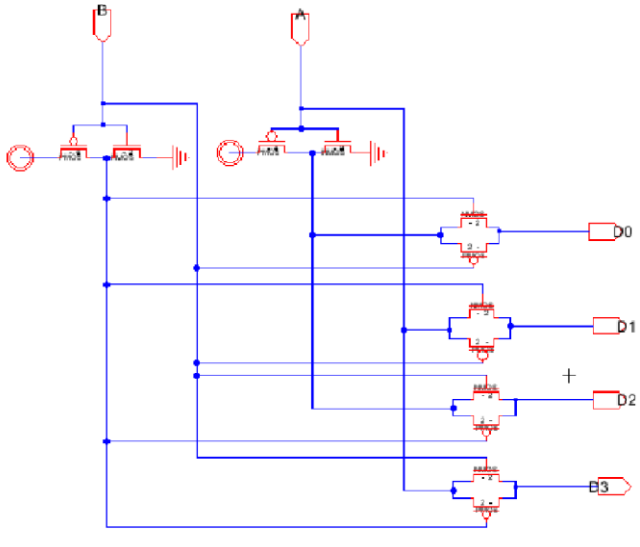

Figure 2.9: The schematic of the 2x4 Decoder.

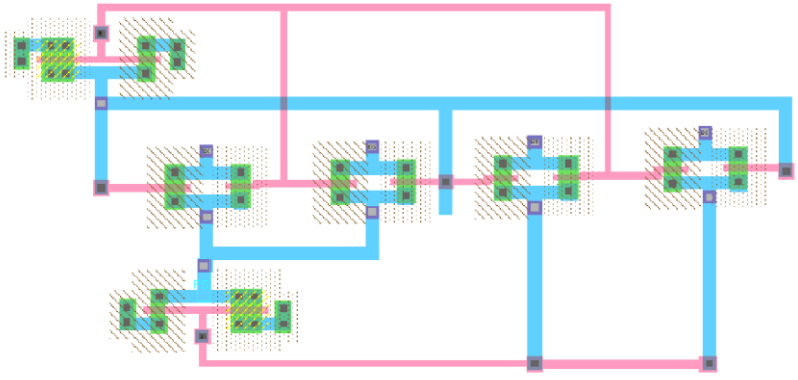

Figure 2.10: The layout of the 2x4 Decoder.

e) 4-inputs NAND gate Schematic and Layout:

The 4-input NAND gate implementation will be used when implementing the 4-Bit CAM when connecting the 4 blocks of the 1-bit CAM to the final output.

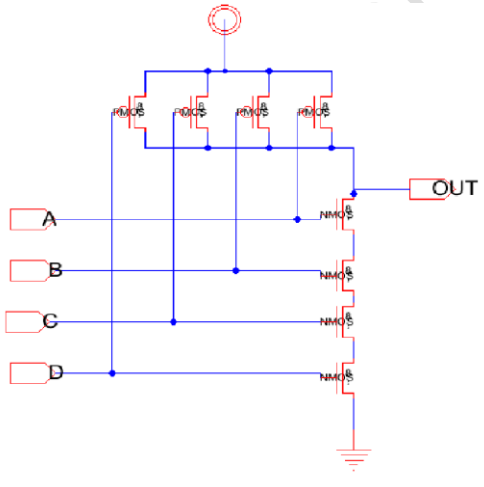

Figure 2.11: 4-inputs NAND gate schematic.

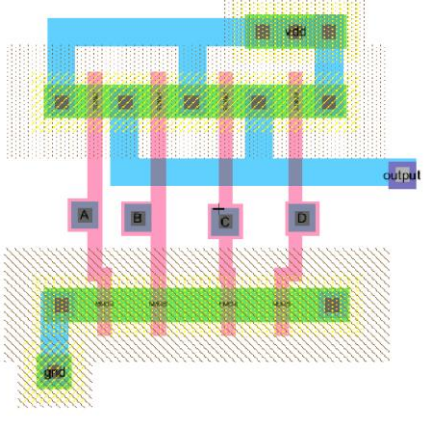

Figure 2.12: 4-inputs NAND gate layout.

f) Inverter gate Schematic and Layout:

Lastly, the last component needed for the 4-Bit CAM circuit is the inverter circuit, the following figures show the used schematic and layout of the component.

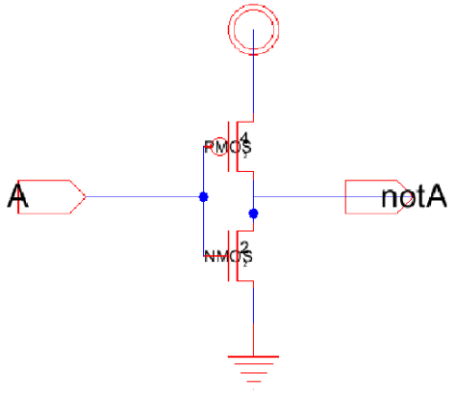

Figure 2.13: Inverter gate schematic.

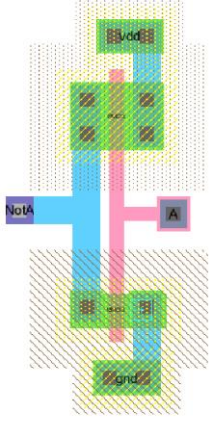

Figure 2.14: Inverter gate layout.

g) 4-Bit CAM Schematic and Layout:

Finally, the circuit of the 4-Bit CAM using 9T SRAM was implemented as shown in the following figure, the decoder, and 1-bit CAM, NAND gates, and the invertor were used to design and implement to find the output of the 4-Bit CAM.

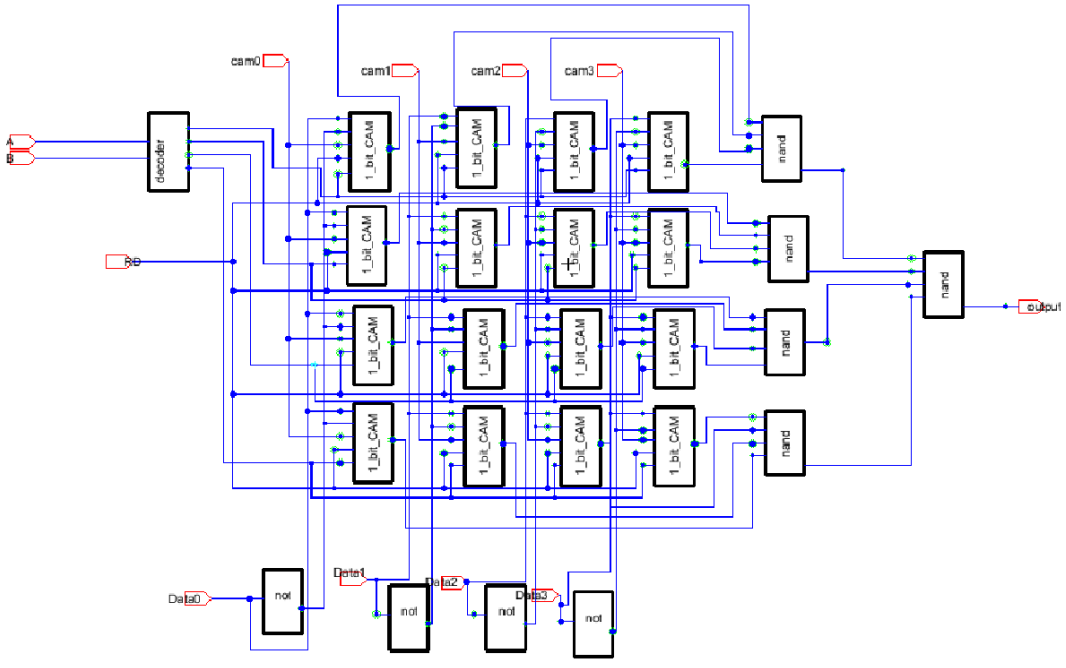

Figure 2.15: The schematic of the 4-Bit CAM using 9T SRAM.

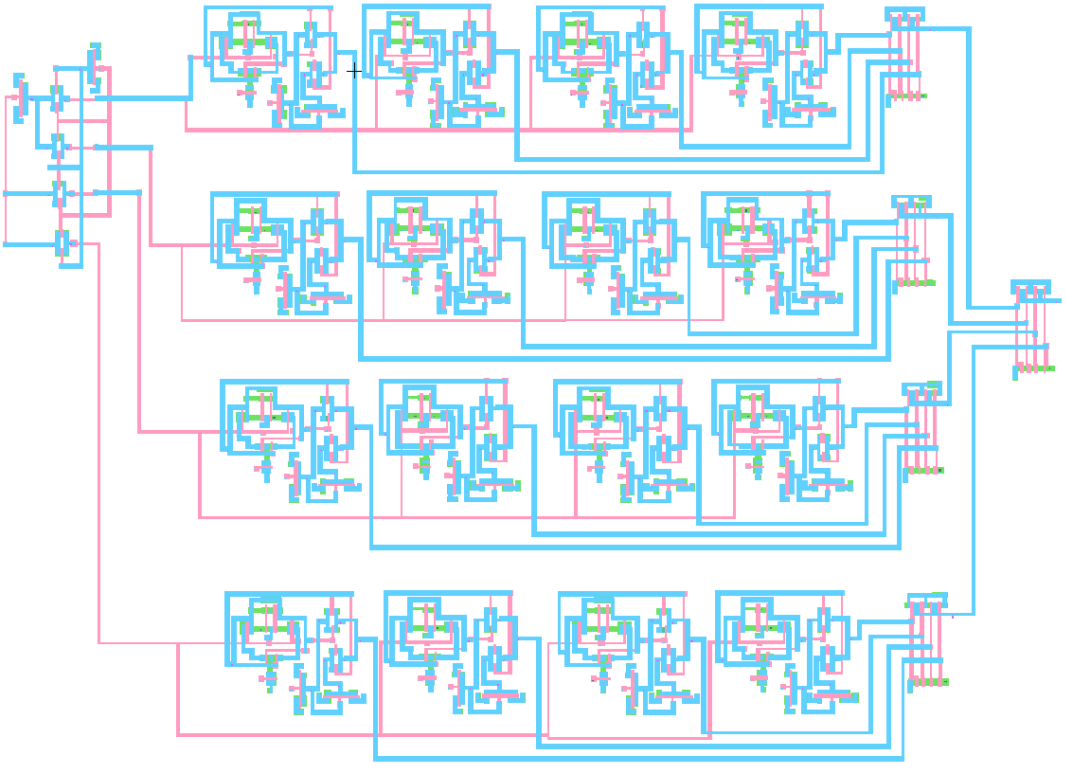

Figure 2.16: The layout of the 4-Bit CAM using 9T SRAM.

2.6 Reference

[1] https://rb.gy/e114v [Accessed 10/8/2023]

[2] Difference between CAM and RAM operations. | Download Scientific Diagram (researchgate.net) [Accessed 10/8/2023]

[3] Schematic of 9T SRAM Cell | Download Scientific Diagram (researchgate.net) [Accessed 10/8/2023]

[4] https://en.wikipedia.org/wiki/Electric_(software) [Accessed 9/23/2023]

Experiment No. 3 - Synthesis/Constraint and design environment

3.1 Objectives

- How to login to Putty environment at BZU VLSI Lab
- To understand the connection required to run synopsis synthesis tool
- To study setup of Design Compiler, RTL synthesis for technology specific libraries.
- How to interactively use the tool and set up its environment.

3.2 Equipment Required

- Synopses tool.
- PuTTY
- Xming

3.3 Pre-Lab

- Read the experiment to learn how to use the tool.
- Download the tool we need like PuTTy and Xming .

3.4 Introduction

3.4.1 Synthesis

synthesis refers to the process of converting a high-level description of a digital circuit into a lower-level representation that can be used for physical implementation. VLSI synthesis is a critical step in the design of integrated circuits (ICs), where complex digital systems are realized on a silicon chip.

VLSI synthesis involves several key aspects:

1. High-Level Design: The process begins with a high-level design description, often written in a hardware description language (HDL) like Verilog or VHDL. This high-level description specifies the behavior and functionality of the digital circuit.

2. Logical Synthesis: During this phase, the high-level description is transformed into a logical gate-level representation. This involves mapping the functional blocks and operations described in the HDL to actual logic gates (AND, OR, XOR, etc.), flip-flops, and other fundamental building blocks.

3. Technology Mapping: Technology mapping involves mapping the logical gates from the previous step to the specific gates available in the target technology library. Different technology libraries provide different gate types and sizes optimized for specific manufacturing processes.

4. Optimization: The synthesized circuit is optimized to meet specific design criteria, such as minimizing power consumption, maximizing speed, or reducing area. Various optimization techniques are applied to achieve these goals without compromising the circuit's functionality.

5. Timing Analysis and Closure: Timing analysis is performed to ensure that the circuit operates within specified timing constraints. If timing violations are detected (such as setup and hold violations), the circuit may need to be further optimized or modified to meet the timing requirements.

6. Physical Design Considerations: While not strictly part of synthesis, physical design considerations start to come into play at this stage. These include placement of the synthesized logic cells, routing of interconnects, and dealing with physical design challenges such as signal integrity, power distribution, and heat dissipation.

7. Outputs: The final output of the synthesis process is typically a netlist, which is a list of interconnected gates and flip-flops that represent the synthesized circuit. This netlist is used in subsequent steps of the VLSI design flow, such as placement, routing, and manufacturing.

VLSI synthesis plays a crucial role in bridging the gap between high-level design and physical implementation. It transforms abstract functional descriptions into concrete circuit structures that can be fabricated on silicon wafers to create integrated circuits.

3.4.2 Synthesis Tool

A synthesis tool, in the context of digital design and electronic circuits, is a software tool that automates the process of transforming a high-level description of a circuit's behavior into a lower-level implementation that can be realized using logic gates and other digital components. The primary goal of a synthesis tool is to bridge the gap between a designer's high-level description and the physical implementation of the circuit on a chip.

The synthesis process involves mapping the functional description of the circuit to a set of library cells (logic gates) from a technology library, while also considering design constraints and optimization goals. To perform this synthesis, the tool requires several input files:

1. Design Files (HDL): These are the high-level descriptions of the digital circuit you want to design. Common hardware description languages include VHDL and Verilog. These files contain the functional behavior of the circuit, including registers, combinational logic, and other components.

2. Library Files (Libraries): Synthesis tools rely on libraries that contain standard cell definitions. A standard cell library includes a collection of pre-designed and characterized logic gates, flip-flops, multiplexers, etc. Each cell comes with its functional behavior, timing characteristics, power consumption, and other important attributes. The synthesis tool uses these cells to map your design's functionality to a gate-level representation.

3. Design Constraints (Constraints): Constraints guide the synthesis tool in its optimization process. They include timing constraints, area constraints, power constraints, and more. For example, you can specify the maximum delay allowed between different parts of the design, or you can specify the desired clock frequency. These constraints help the synthesis tool make design decisions that align with your goals.

4. Design for Testability (DFT) Files: If you want to include features to facilitate testing, you might provide DFT-related files. These could include test insertion specifications, boundary scan descriptions (using standards like IEEE 1149.1, also known as JTAG), and other mechanisms to improve the testability of the design.

5. Technology Files (Techfile): The technology file, also known as the technology library or process file, provides information about the manufacturing process, cell libraries, and other process-related parameters. It describes the characteristics of the transistors, interconnect layers, metal stack, and other process-specific details that are crucial for mapping the logical design to a physical layout.

6. Power Intent Files (Optional):In modern designs, power is a critical consideration. Power intent files like the Common Power Format (CPF) or Unified Power Format (UPF) specify the power domains, power modes, and power-related constraints for the design. These files guide the synthesis tool in optimizing for low power consumption.

Each of these input files is essential for the synthesis tool to accurately and effectively transform your high-level design into a gate-level netlist that can be further processed in subsequent stages of the design flow. The specifics of the files and their formats can vary depending on the synthesis tool and the design flow being used.

3.5 Procedure

Design Compiler provides the following ways to read the design files: analyze, elaborate, read. During design process Design Compiler uses technology specific libraries target and link libraries. Target library is used to build a circuit. During mapping, Design Compiler selects functionally correct gates from the target library. It also calculates the timing of the circuit using the vendor supplied timing data for these gates. Design Compiler uses the link library to resolve references. For the design to be complete, it must connect to all library components and designs it references.

3.5.1 Putty Login:

a) Session Information:

- Open putty and run Xming then the following screen will be shown.

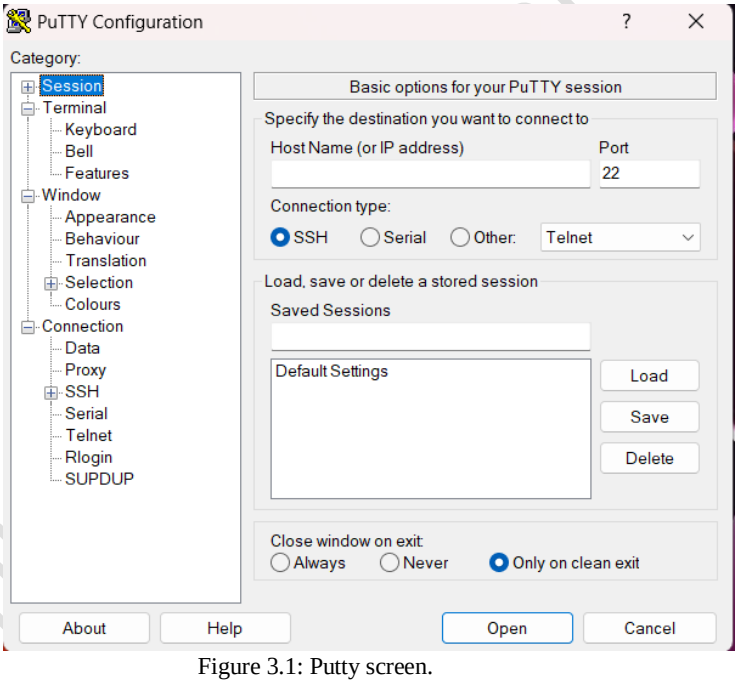

Figure 3.1: Putty screen.

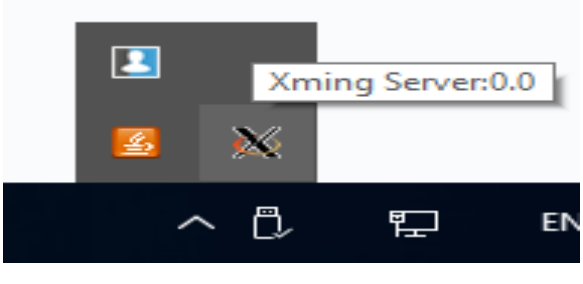

Figure 3.2: Xming running.

- Enter the following information to remote server management and network communication as shown in figure below:

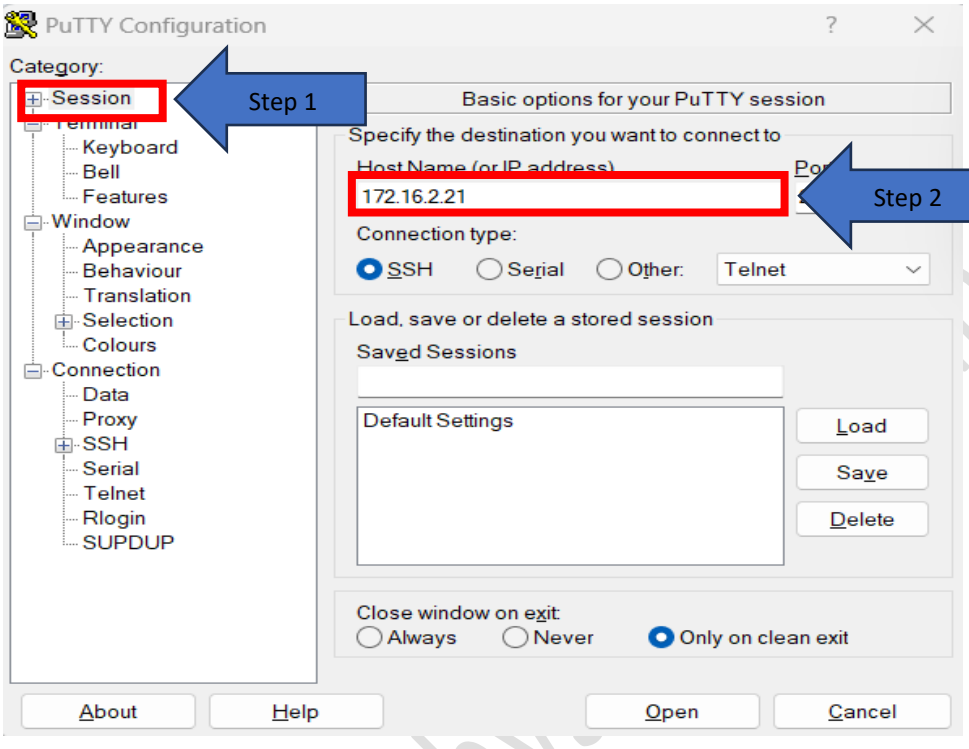

Figure 3.3: Enter IP address.

- X11 Forwarding: For Unix-based systems, PuTTY supports X11 forwarding, allowing graphical applications to be run remotely and displayed locally.

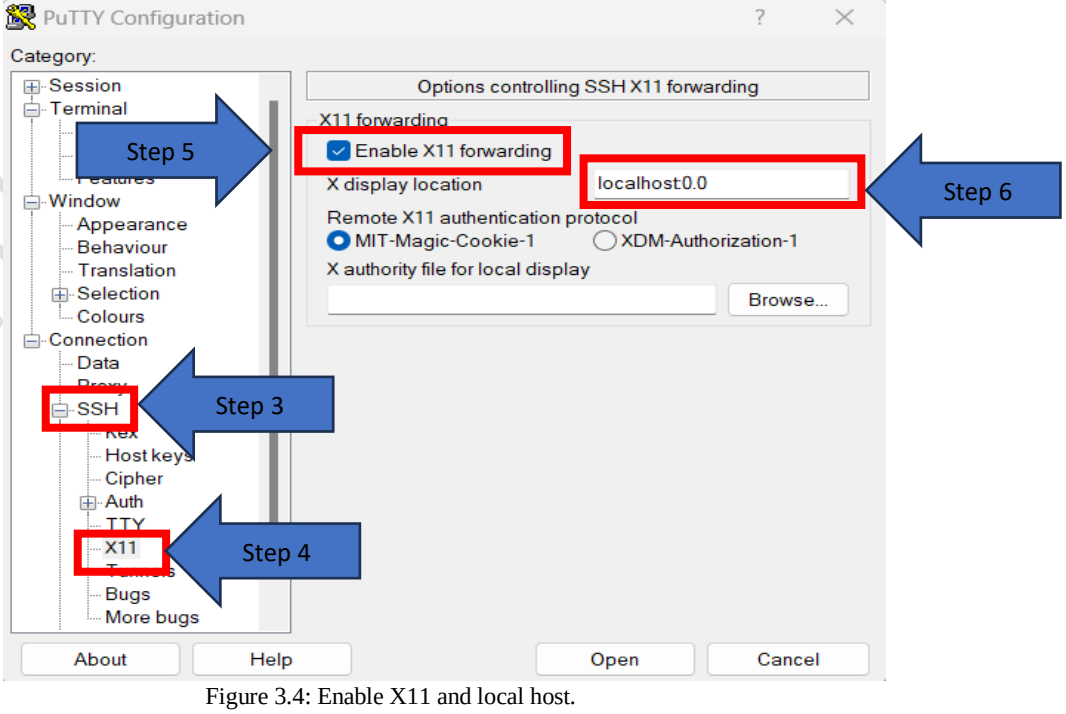

Figure 3.4: Enable X11 and local host.

- To save this information to use it later . bout name for sessions then click save .

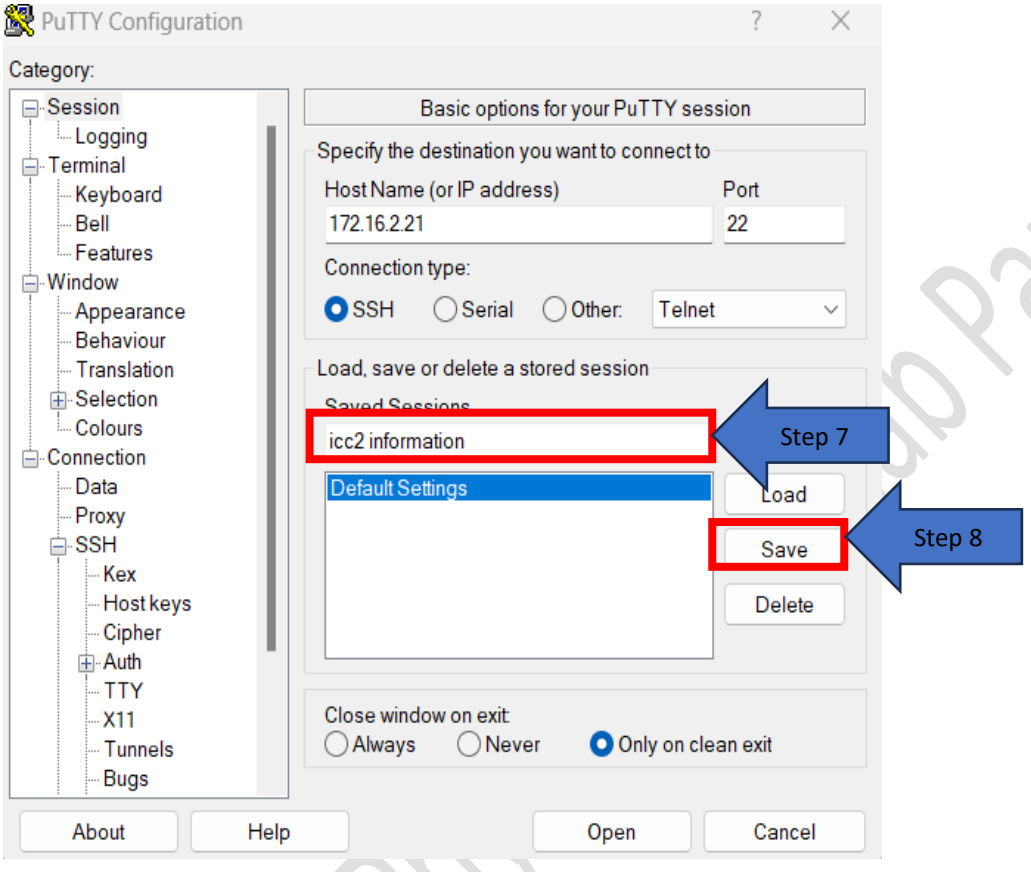

Figure 3.5: Save information.

- To use this information later click on sessions name then click on load .then the information will put in the boxes.

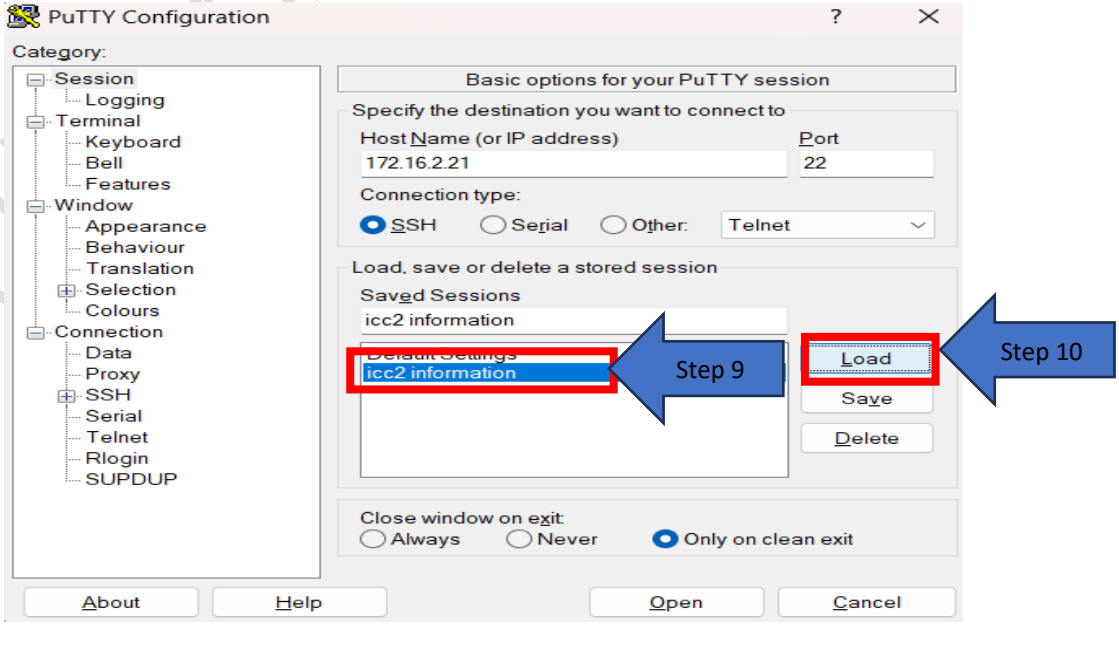

Figure 3.6: load the session information.

- After select the information click on open.

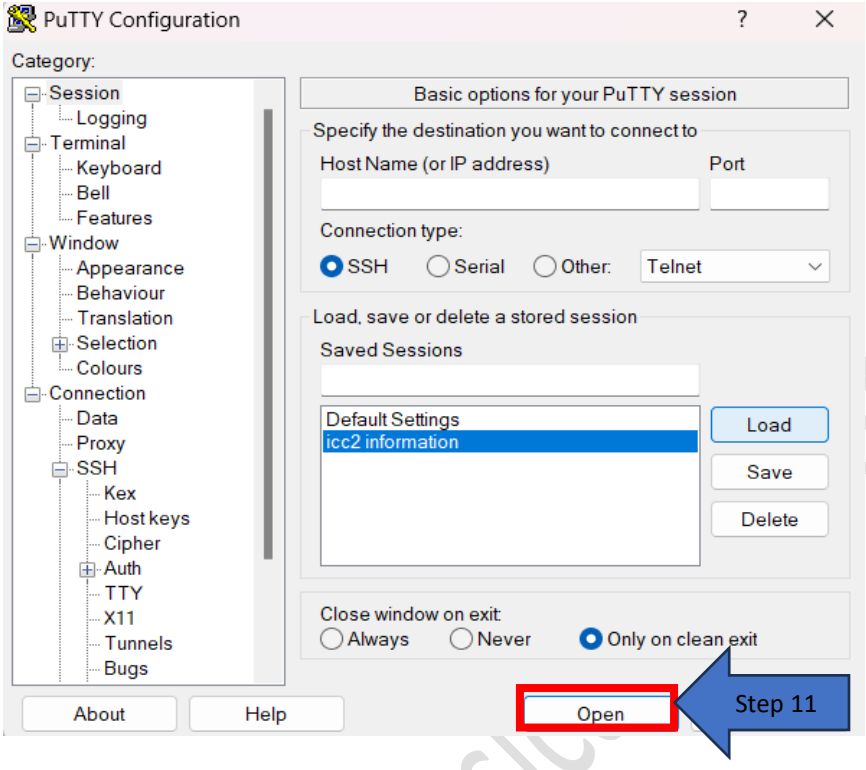

Figure 3.7: Load the session information.

- The bellow screen will show. Enter your user's name then your password.

Figure 3.8: Enter your user's name then your password [1].

3.5.2 Environment Setup:

- **Go to your EXP directory by going to your user directory (cd ~), then do: (mkdir -p ./ST_VLSI/EXP3)**

```
[hanan@synopsys ~]$ cd ~
[hanan@synopsys ~]$ mkdir -p ./ST_VLSI/EXP3
[hanan@synopsys ~]$
```

Figure 3.9: Make directory [1].

- **go to EXP3 directory (cd ./ST_VLSI/EXP3)**

```
[hanan@synopsys ~]$ cd ./ST_VLSI/EXP3
[hanan@synopsys EXP3]$
```

Figure 3.10: Open the directory [1].

- **Copy the tool environment to your user directory (change the word in red color):**

  cp -R /home/iccTA/all_labs/dc_EXP3_env/* .

```
[hanan@synopsys EXP3]$ cp -R /home/iccTA/all_labs/dc_EXP3_env/*    .
```

Figure 3.11: Copy the tool environment [1].

- **To view the copied files: ls**

```
[hanan@synopsys EXP3]$ ls
inputs  MY_DESIGN.v  rm_dc_scripts  rm_setup  run_dc.csh
```

Figure 3.12: show files [1].

- **Check the setup and run files for these variables:**
  i. **gedit rm_setup/common_setup.tcl enter to this file and search about the following things: "DESIGN_NAME","TARGET_LIBRARY_FILES", "MIN_ROUTING_LAYER"/"MAX_ROUTING_LAYER".**

```
[hanan@synopsys EXP3]$ gedit rm_setup/common_setup.tcl
```

Figure 3.13: Open common_setup.tcl [1].

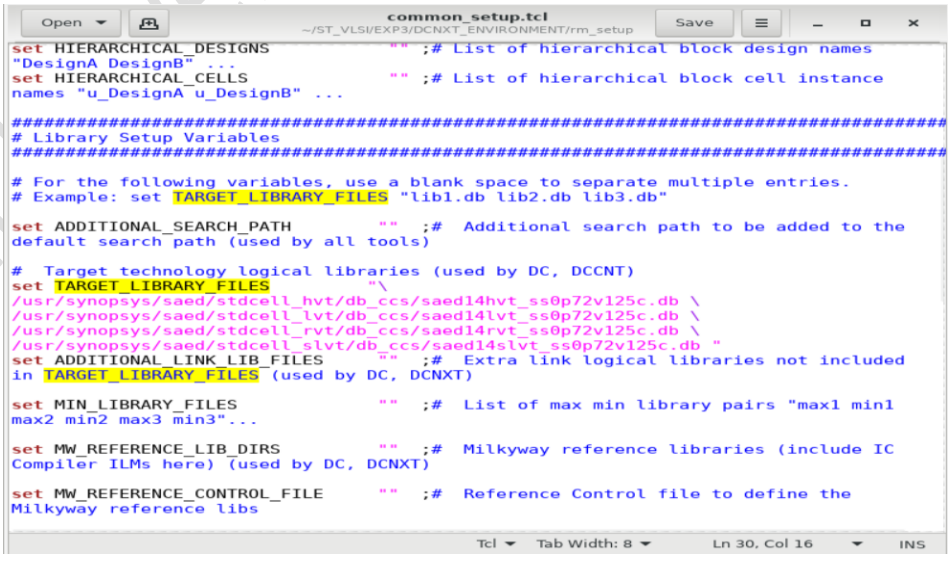

Figure 3.14: Open common_setup.tcl file [1].

ii. **gedit rm_setup/dc_setup_filenames.tcl: "DCRM_CONSTRAINTS_INPUT_FILE"**

```
[hanan@synopsys EXP3]$ gedit rm_setup/dc_setup_filenames.tcl
```

Figure 3.15: Open dc_setup_filenames.tcl [1].

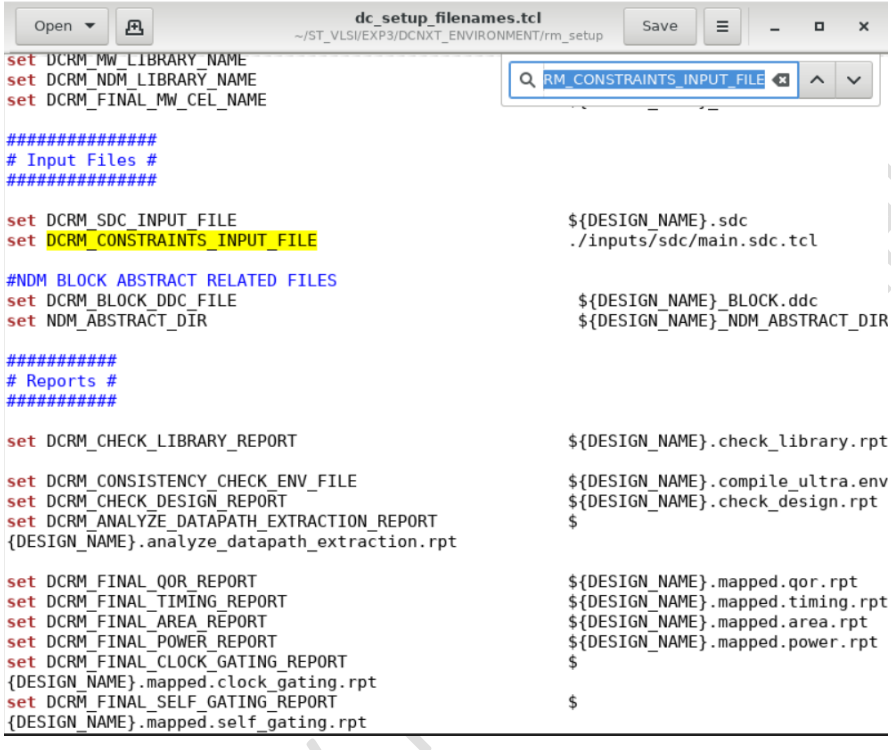

Figure 3.16: Open dc_setup_filenames.tcl [1].

iii. **gedit rm_dc_scripts/dc.tcl check these commands: analyze, elaborate, source @#125, set_ignored_layers, compile_ultra -gate_clock, compile_ultra -incremental, write -format -verilog**

```
[hanan@synopsys EXP3]$ gedit rm_dc_scripts/dc.tcl
```

Figure 3.17: Open dc.tcl [1].

- **Open the Verilog file to read a code and understand how it work: gedit MY_DESIGN.v**

```
ynopsys EXP3]$ ls
MY_DESIGN.v  rm_dc_scripts  rm_setup  run_dc.csh
ynopsys EXP3]$ gedit MY_DESIGN.v
```

Figure 3.18: open the Verilog file [1].

3.5.3 Design after elaborate (check in gui). to open gui and load the design (ddc file):

i. dcnxt_shell

```
[hanan@synopsys DCNXT_ENVIRONMENT]$ dcnxt_shell

                           Design Compiler (R) NXT

             Version R-2020.09-SP5-5 for linux64 - Jan 19, 2022

                    Copyright (c) 1988 - 2022 Synopsys, Inc.
   This software and the associated documentation are proprietary to Synopsys,
 Inc. This software may only be used in accordance with the terms and conditions
 of a written license agreement with Synopsys, Inc. All other use, reproduction,
   or distribution of this software is strictly prohibited.  Licensed Products
     communicate with Synopsys servers for the purpose of providing software
    updates, detecting software piracy and verifying that customers are using
    Licensed Products in conformity with the applicable License Key for such
  Licensed Products. Synopsys will use information gathered in connection with
    this process to deliver software updates and pursue software pirates and
                                 infringers.

 Inclusivity & Diversity - Visit SolvNetPlus to read the "Synopsys Statement on
           Inclusivity and Diversity" (Refer to article 000036315 at
                       https://solvnetplus.synopsys.com)
Initializing...
dcnxt_shell>
```

Figure 3.19: Dcnxt shell [1].

ii. dcnxt_shell> source rm_dc_scripts/dc.tcl

```
dcnxt_shell> source rm_dc_scripts/dc.tcl
source -echo -verbose ./rm_setup/common_setup.tcl
puts "RM-Info: Running script [info script]\n"
RM-Info: Running script /home/hanan/ST_VLSI/EXP3/DCNXT_ENVIRONMENT/rm_setup/comm
on_setup.tcl

##########################################################################
##########
# Variables common to all reference methodology scripts
# Script: common_setup.tcl
# Version: R-2020.09-SP4
# Copyright (C) 2007-2021 Synopsys, Inc. All rights reserved.
##########################################################################
##########
#set DESIGN_NAME                 ""  ;#  The name of the top-level design
set DESIGN_NAME [lindex [split [glob *.v] .] 0]
SimpleSystem
set DESIGN_REF_DATA_PATH          ""  ;#  Absolute path prefix variable for libr
ary/design data.
#  Use this variable to prefix the common absolute path
#  to the common variables defined below.
#  Absolute paths are mandatory for hierarchical
#  reference methodology flow.
##########################################################################
##########
# Hierarchical Flow Design Variables
##########################################################################
##########
set HIERARCHICAL_DESIGNS          "" ;# List of hierarchical block design names
 "DesignA DesignB" ...
set HIERARCHICAL_CELLS            "" ;# List of hierarchical block cell instanc
e names "u_DesignA u_DesignB" ...
##########################################################################
##########
# Library Setup Variables
##########################################################################
##########
```

Figure 3.20: Source dc.tcl [1].

iii. dcnxt_shell> read_ddc results/<design_name>.elab.ddc

```
dcnxt_shell> read_ddc results/MY_DESIGN.elab.ddc
Loading db file '/usr/synopsys/syn/R-2020.09-SP5-5/libraries/syn/gtech.db'
Loading db file '/usr/synopsys/syn/R-2020.09-SP5-5/libraries/syn/standard.sldb'
  Loading link library 'gtech'
Reading ddc file '/home/hanan/ST_VLSI/EXP3/DCNXT_ENVIRONMENT/results/MY_DESIGN.elab.ddc'.
Loaded 3 designs.
Current design is 'MY_DESIGN'.
MY_DESIGN ARITH COMBO
dcnxt_shell>
```

Figure 3.21: Make design for Verilog [1].

iv. dcnxt_shell> start_gui

```
dcnxt_shell> start_gui
dcnxt_shell> Current design is 'MY_DESIGN'.
dcnxt_shell> 4.1
Current design is 'MY_DESIGN'.
```

Figure 3.22: Start gui [1].

v. The following windows will show:

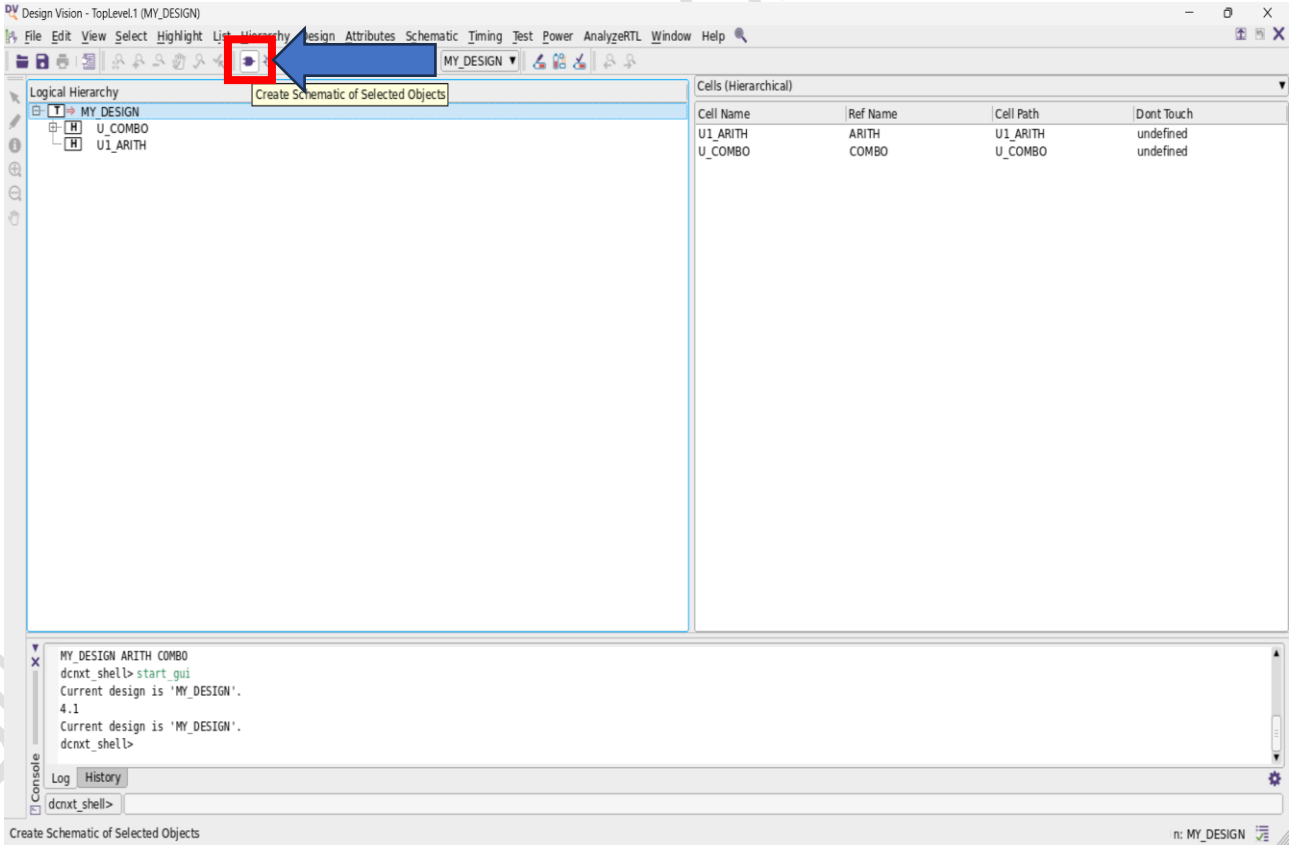

Figure 3.23: Design vision [1].

vi. To show the circuit:

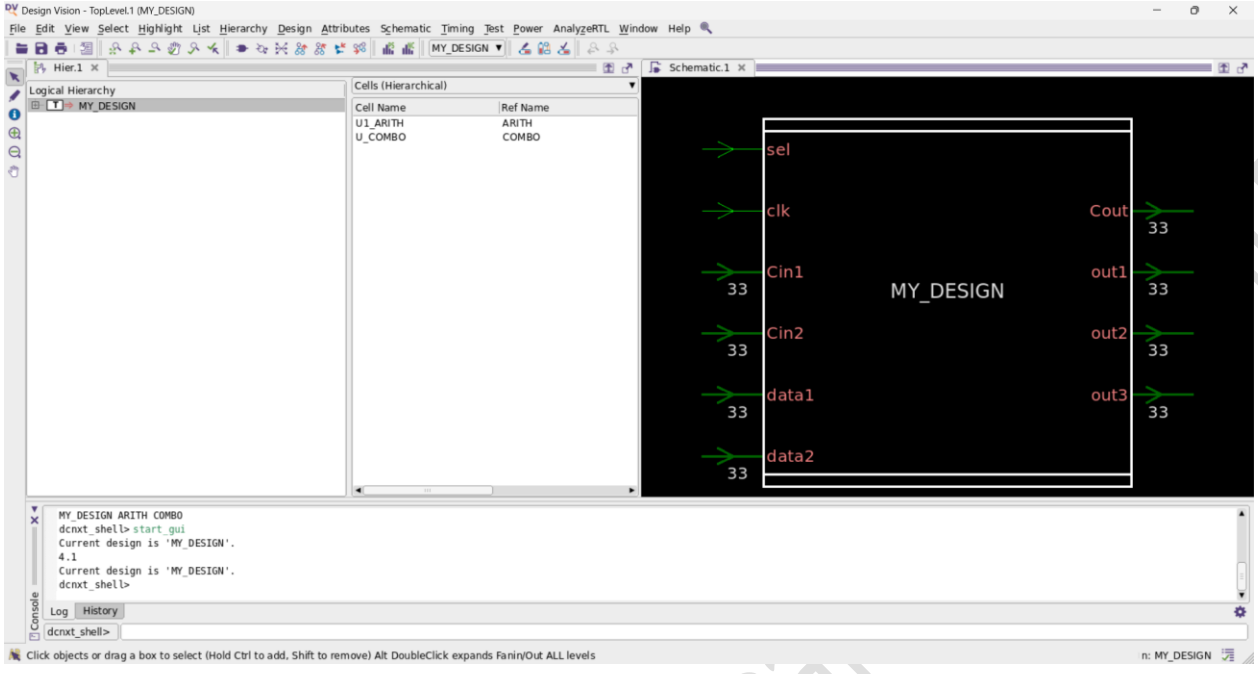

Figure 3.24: Shown the circuit [1].

vii. To zoom in the circuit:

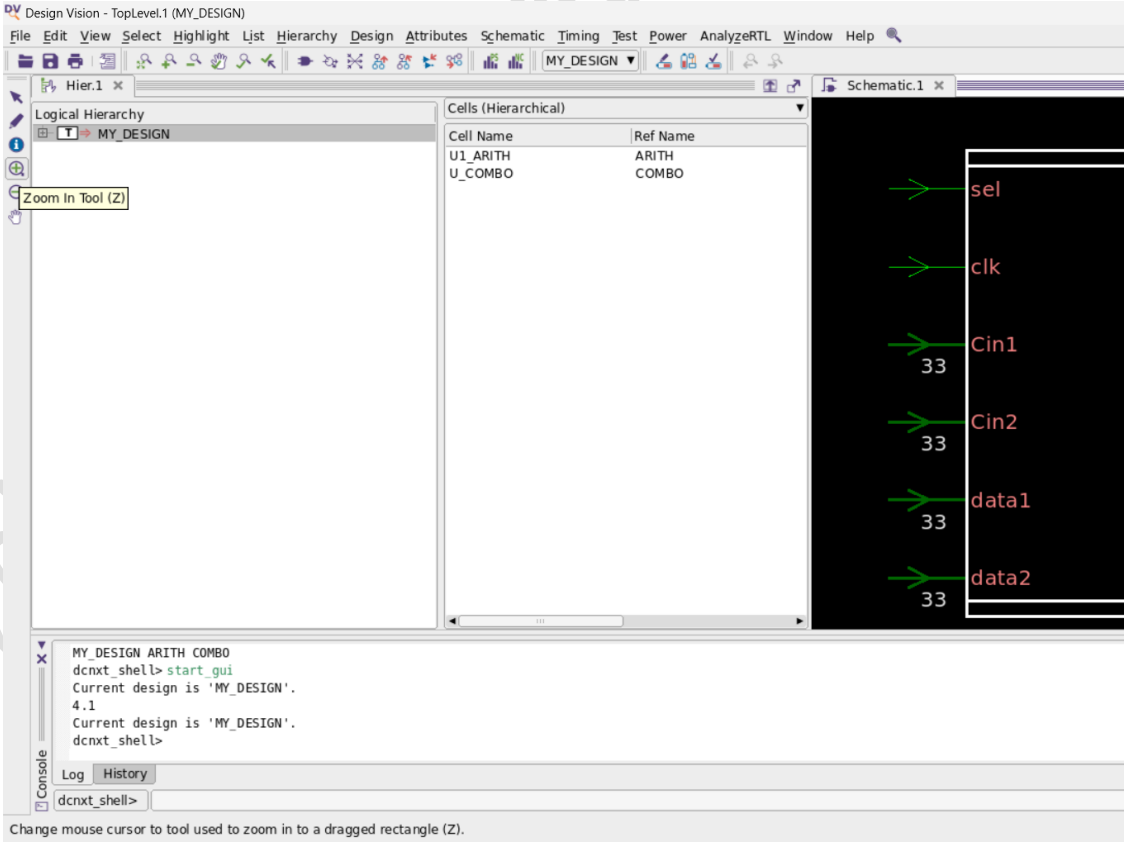

Figure 3.25: Zoom in [1].

viii. **The following picture show the gates for this circuit:**

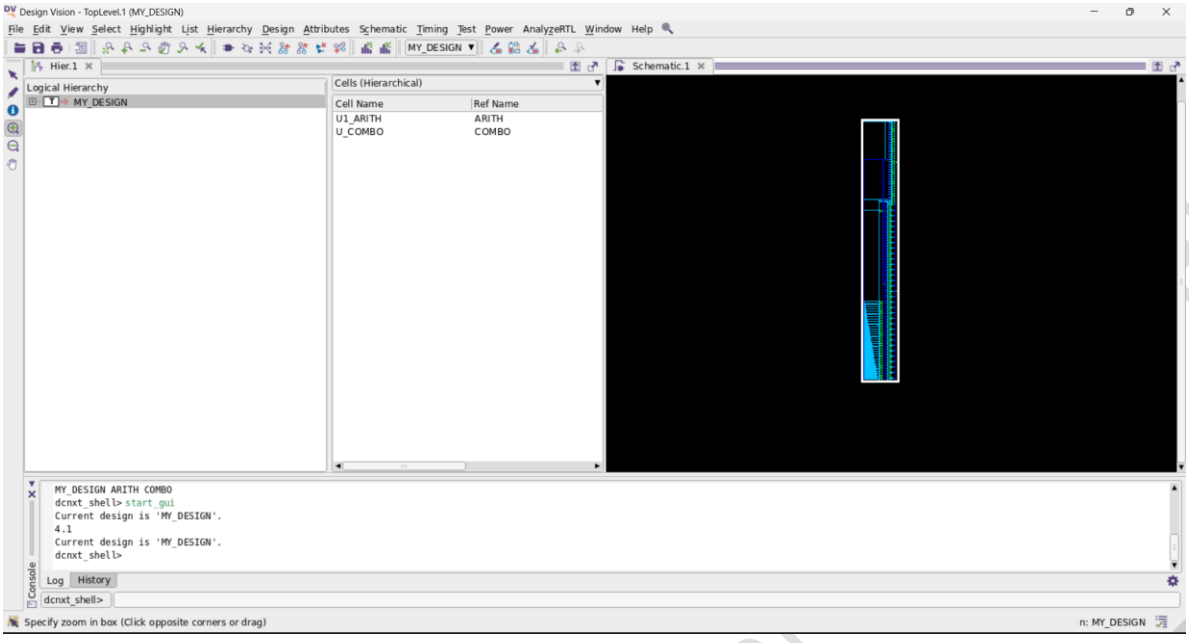

Figure 3.26: Circuit gates [1].

3.6 Linux Reference:

a. **list files: ls , add "-ltr" to print a time sorted list of files, ex: ls -ltr work_dir**

b. **make directory: mkdir -p <new directory>, ex: mkdir -p work_dir**

c. **change directory: cd <dir>, ex: cd work_dir/**

d. **remove file: rm <file> , to remove directory add -r, to force remove add -f, ex: rm -rf work_dir**

e. **to edit text files: gedit <file> , dont forget to save before closing**

f. **copy file: cp <file> <destination>, ex: cp file1 work_dir/file1**

g. **move file: mv <file> <destination>, ex: mv file1 file2**

h. **display file content in terminal: cat <file>, ex: cat file1**

i. **redirect stdout: ">", ex: echo "hey" > file.txt**

3.7 References:

[1] Synopsys: Synopsys | EDA Tools, Semiconductor IP and Application Security Solutions

3.8 Appendix:

```
module MY_DESIGN ( Cin1, Cin2, Cout, data1, data2, sel, clk, out1, out2, out3);
  input [32:0] Cin1, Cin2, data1, data2;
  input sel, clk;
  output [32:0] Cout, out1, out2, out3;
  reg   [32:0] R1, R2, R3, R4, out1, out2, out3;
  wire   [32:0] arth_o;

  ARITH U1_ARITH ( .a(data1), .b(data2), .sel(sel), .out1(arth_o) );
  COMBO U_COMBO ( .Cin1(Cin1), .Cin2(Cin2), .sel(sel), .Cout(Cout) );

always @ (posedge clk)
  begin
    R1 <= arth_o;
    R2 <= data1 & data2;
    R3 <= data1 + data2;
    R4 <= R2 + R3;
  end

always @ (out2, R1, R3, R4)
  begin
    out1 <= R1 + R3;
    out2 <= R3 & R4;
    out3 <= out2 - R3;
  end

endmodule


module ARITH ( a, b, sel, out1 );
  input [32:0] a, b;
  input sel;
  output [32:0] out1;
  reg [32:0] out1;

always @(sel, a, b)
```

```
  begin
    case({sel})
        1'b0: out1 <= a + b;
        1'b1: out1 <= a - b;
    endcase
  end

endmodule

module COMBO ( Cin1, Cin2, sel, Cout );
  input [32:0] Cin1, Cin2;
  input sel;
  output [32:0] Cout;
  reg [32:0] Cout;

  wire   [32:0] arth_o;

  ARITH U2_ARITH ( .a(Cin1), .b(Cin2), .sel(sel), .out1(arth_o) );

always @ (Cin1, arth_o)
  begin
    Cout <= arth_o + Cin1;
  end

endmodule
```

Experiment No. 4 - Floorplan and Power planning

4.1 Objectives

- Physically implement a netlist generated by DC tool through ICC2 flow.
- Understand the initial ICC2 implementation flow (Initialize Design & Power planning).
- How to interactively create floorplan & view initialized design.

4.2 Equipment Required

- Synopses tool - IC Compiler II
- PuTTY
- Xming

4.3 Pre-Lab

1. Read the experiment to learn how to use the tool.

4.4 Introduction

4.4.1 Floorplanning

Chip floor planning is a critical step in the integrated circuit (IC) design process. It involves determining the physical placement of various components and blocks on a semiconductor chip or printed circuit board (PCB) to optimize performance, power consumption, and overall chip area. The goal of floor planning is to create an efficient and manufacturable layout that meets the design specifications and constraints. floor planning is a crucial step in semiconductor design, as the physical layout significantly impacts the chip's performance, power efficiency, and manufacturability. Advanced tools and algorithms are often employed to help designers create optimal floorplans while meeting the ever-increasing demands of modern semiconductor technology.

inputs for floorplan:

- Netlist (.v)
- Technology file (techlef)
- Timing Library files (.lib)

- Physical library (.lef)
- Synopsys design constraints (.sdc)

Floorplan Flowchart:

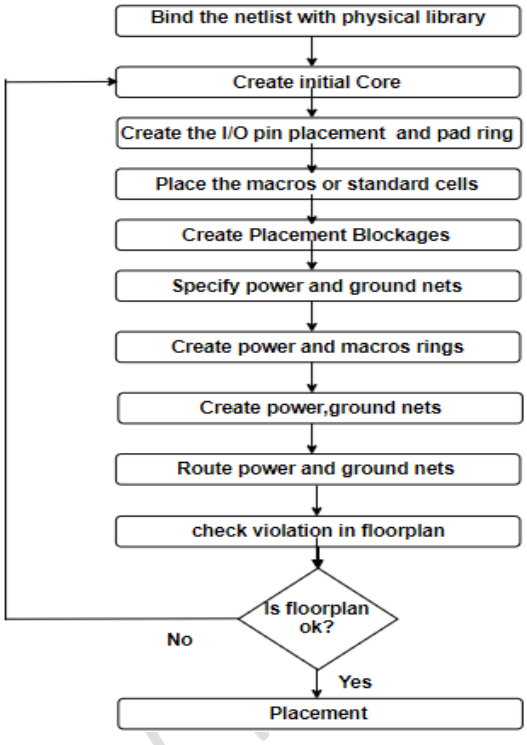

Figure 4.1: Floorplan Flowchart [1].

Types of Floorplan Techniques:

- Abutted floorplan : Channel less placement of blocks.
- Non-Abutted Floorplan : Channel based placement of blocks.
- Mix of both: partially abutted with some channels.

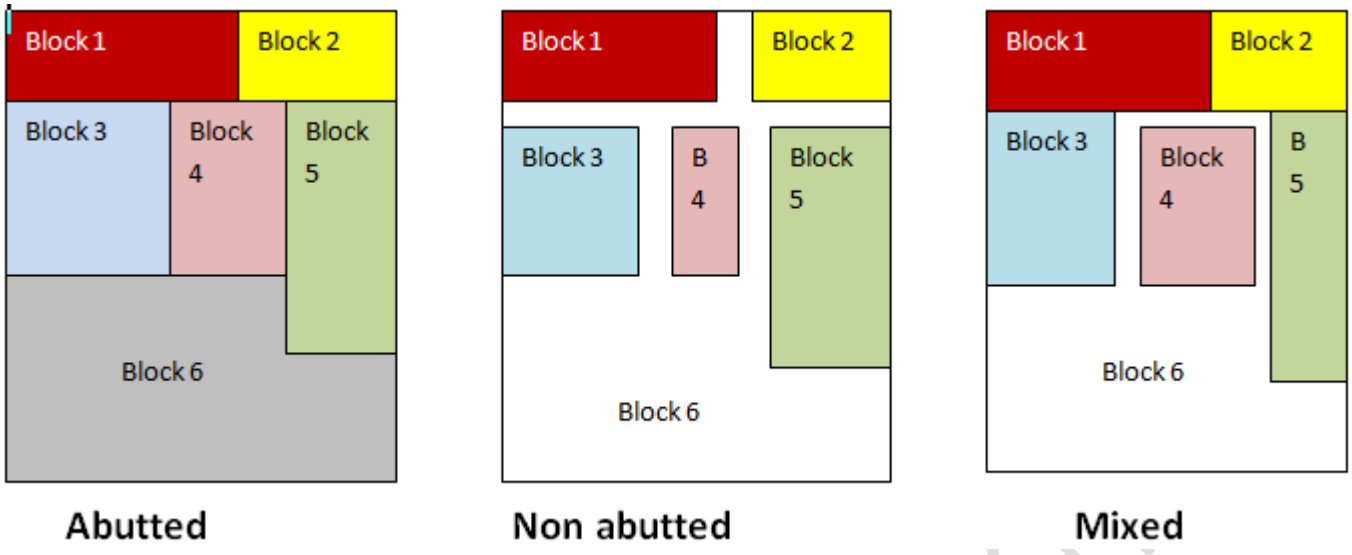

Figure 4.2: Types of Floorplan Techniques [2].

Terminologies and Definitions

- ❖ Utilization
  - Area of the core that is used by placed Standard Cells and Macros expressed inpercentage
- ❖ Manufacturing Grid
  - The smallest geometry that semiconductor foundry can process or smallest resolution of your technology process (e.g. 0.005)
  - All drawn geometries during Physical Design must snap to this grid
  - While Masking fab. use this as reference lines
- ❖ Standard Cell Site/ Standard Cell Placement Tile/ unit Tile
  - The minimum Width and Height a Cell that can occupy in the design
  - The Standard Cell Site will have the same height as Standard Cells, but the width will be as small
  - as your smallest Filler Cell
  - It's one Vertical Routing Track and the Standard Cell Height
  - All Standard Cells must be multiple of Unit Tile
- ❖ Standard Cell Rows
  - Rows are actually the Standard Cell Sites abut side by side and then Standard Cells are placed on these Rows
  - Cells with the equal no. of Track definition will have same height
- ❖ Placement Grid
  - Placement Grid is made up of Standard Cell Site
  - It's always a multiple of Manufacturing Grid
  - Placement Grid is made up of the Rows which are composed of Sites

- ❖ Routing Grid and Routing Track
  - Horizontal and Vertical line drawn on the layout area which will guide for making interconnections
  - The Routing Grid is made up of the Routing Tracks
  - Routing Tracks can be Grid-based, Gridless based or Subgrid-based
- ❖ Flight-line/ Fly-line
  - Virtual connection between Macros and Macro or Macros and IOS
- ❖ Macro
  - Any instances other than Standard Cell and is as loaded as black box to the design is Macro
  - Intellectual Property (IP) e.g. RAM, ROM, PLL, Analog Designs etc.
  - Hard Macro: IP with Layout implemented
  - Soft Macro: IP without Layout implemented (HDL)

Steps in Floorplan

- Initialize with Chip & Core Aspect Ratio (AR)
- Initialize with Core Utilization
- Initialize Row Configuration & Cell Orientation
- Provide the Core to Pad/ IO spacing (Core to IO clearance)
- Pins/ Pads Placement
- Macro Placement by Fly-line Analysis
- Macro Placement requirements are also need to consider
- Blockage Management (Placement/ Routing)

4.4.2 Power planning:

Power planning means to provide power to every macro, standard cells, and all other cells are present in the design. Power and Ground nets are usually laid out on the metal layers. In this create power and ground structure for both IO pads and core logic. The IO pads power and ground buses are built into the pad itself and will be connected by abutment.

For core logic there is a core ring enclosing the core with one or more sets of power and ground rings. The next consideration is to construct cell power and ground that is internal to core logic these are called power and ground stripes that repeat at regular intervals across the logic or specified region, within the design. Each of these stripes run both vertically and horizontally at regular interval then this is called power mesh.

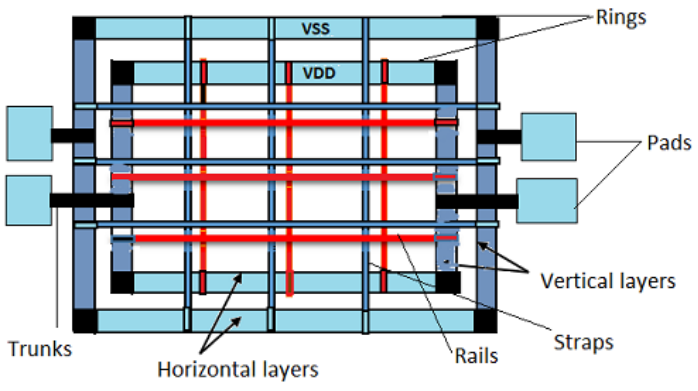

Figure 4.3: Overview of Power Planning [3].

List of Input files for Power Planning tools:

- Netlist (.v)
- Synopsys Design Constraints(SDC)
- Database of floor plan
- Power Strap and Power ring width
- Tie cells information(VDD or VSS)

List of Output files from Power Planning tools:

- Power Routed database
- Power estimation Report
- Pre Route DRC checking report
- Power planning involves calculating the number of power pins required, Number of rings, strips and I R Drop.

Power Information**:**

- The power information can obtain from the front end design.
- the synthesis tool reports static power information
- dynamic power can be calculated using value change dump (VCD) or switching activity interchange format (SAIF) file in conjunction with RTL description and test bench.
- Exhaustive test coverage is required for efficient calculation of peak power. this methodology is depicted in figure as shown below.

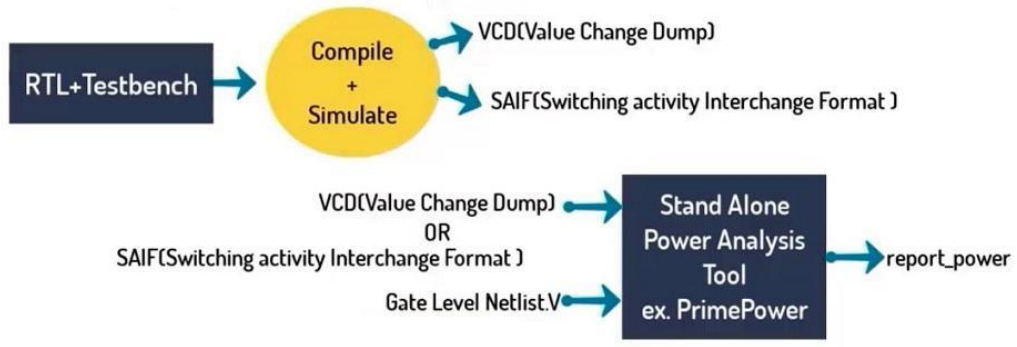

Figure 4.4: Power Information [3].

There are two types of power planning and management

a) Core Cell Power Management
   - Power rings are formed around the core.
   - Straps and trunks are created for macros according to power requirement.
   - The horizontal and vertical layers are connected each other using proper via cut.

b) B) 1/0 Cell Power Management
   - Power rings are formed for 1/0 cells.
   - Trunks are created between core power ring and power pads.

4.5 Procedure

4.5.1 Environment Setup:

- **Go to your EXP directory by going to your user directory (cd ~), then do: (mkdir -p ./ST_VLSI/EXP4)**

```
[hanan@synopsys ~]$ cd
[hanan@synopsys ~]$ mkdir -p ./ST_VLSI/EXP4
```

Figure 4.5: Make directory [4].

- **Go to EXP4 directory (cd ./ST_VLSI/EXP4)**

```
[hanan@synopsys ~]$ cd ./ST_VLSI/EXP4
[hanan@synopsys EXP4]$
```

Figure 4.6: Open the directory [4].

- **Copy the tool environment to your user directory(change the word in red color): cp -R /home/iccTA/all_labs/icc2_EXP4_env/* .**

```
[hanan@synopsys EXP4]$ cp -R  /home/iccTA/all_labs/icc2_EXP4_env/*  .
```

Figure 4.7: Copy the tool environment [4].

- **To view the copied files: ls**

```
[hanan@synopsys EXP4]$ ls
inputs  Makefile  rm_icc2_pnr_scripts  rm_setup
```

Figure 4.8: show files [4].

- **Link your synthesized netlist to the input's directory: ln -s /home/<username>/ST_VLSI/EXP3/results/{DESIGN}.mapped.v inputs/.**

```
[hanan@synopsys EXP4]$ ln -s /home/hanan/ST_VLSI/EXP3/results/MY_DESIGN.mapped.v
 inputs/.
[hanan@synopsys EXP4]$ ls
inputs  Makefile  rm_icc2_pnr_scripts  rm_setup
```

Figure 4.9: Link synthesized netlist to the inputs directory [4].

- **Check the setup and run files for these variables:**
  - **i. gedit rm_setup/icc2_common_setup.tcl enter to this file and search about the following things: "ROUTING_LAYER_DIRECTION_OFFSET_LIST".**

```
[hanan@synopsys EXP4]$ gedit rm_setup/icc2_common_setup.tcl
```

Figure 4.10: Open common_setup.tcl [4].

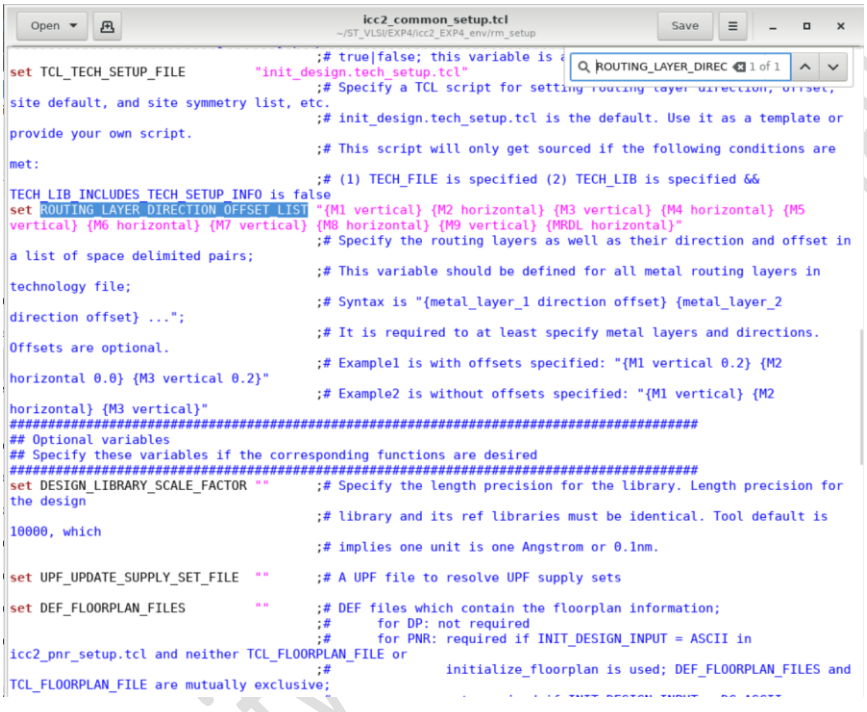

Figure 4.11: Open common_setup.tcl file [4].

4.5.2 Run the design script: shell> make init_design: (it will take little time)

```
[hanan@synopsys EXP4]$ make init_design
```

Figure 4.12: Run the design script [4].

- **i. Check the log for error messages, use " \less logs_icc2/init_design.log" and search using "/^Error" (^: Shift+6) ,then press Enter - If you see any error messages, check with TA. To exit from file press "q" character/**

Figure 4.13: Check the log for error messages [4].

4.5.3 Check the design interactively after it's finished:

1. **shell> icc2_shell [enter] (it will take a time)**

2. **icc2_shell> open_lib <DESIGN_NAME>.nlib [enter]**

```
icc2_shell> open_lib MY_DESIGN.nlib
Information: Loading library file '/home/hanan/ST_VLSI/EXP4/icc2_EXP4_env/MY_DESIGN.nlib' (FI
LE-007)
Information: Loading library file '/usr/synopsys/saed/stdcell_lvt/ndm/saed14lvt_frame_timing_
ccs.ndm' (FILE-007)
Information: Loading library file '/usr/synopsys/saed/stdcell_hvt/ndm/saed14hvt_frame_timing_
ccs.ndm' (FILE-007)
Information: Loading library file '/usr/synopsys/saed/stdcell_slvt/ndm/saed14slvt_frame_timin
g_ccs.ndm' (FILE-007)
Information: Loading library file '/usr/synopsys/saed/stdcell_rvt/ndm/saed14rvt_frame_timing_
ccs.ndm' (FILE-007)
{MY_DESIGN.nlib}
icc2_shell>
```

Figure 4.14: Open design library [4].

3. **icc2_shell> list_blocks [enter]**

```
icc2_shell> list_blocks
Lib MY_DESIGN.nlib /home/hanan/ST_VLSI/EXP4/icc2_EXP4_env/MY_DESIGN.nlib tech current
  ->  0 MY_DESIGN.design Sep-10-17:07
  -   0 MY_DESIGN/init_design.design Sep-10-17:07
2
```

Figure 4.15: Block list [4].

4. **icc2_shell> open_block <DESIGN_NAME>/init_design.design [enter]**

```
icc2_shell> open_block MY_DESIGN/init_design.design
Information: User units loaded from library 'saed14lvt_frame_timing_ccs' (LNK-040)
Opening block 'MY_DESIGN.nlib:MY_DESIGN/init_design.design' in edit mode
{MY_DESIGN.nlib:MY_DESIGN/init_design.design}
icc2_shell>
```

Figure 4.16: open block [4].

5. **icc2_shell> start_gui**

```
icc2_shell> start_gui
```

Figure 4.17: Gui start [4].

6. Press F to reset the gui view

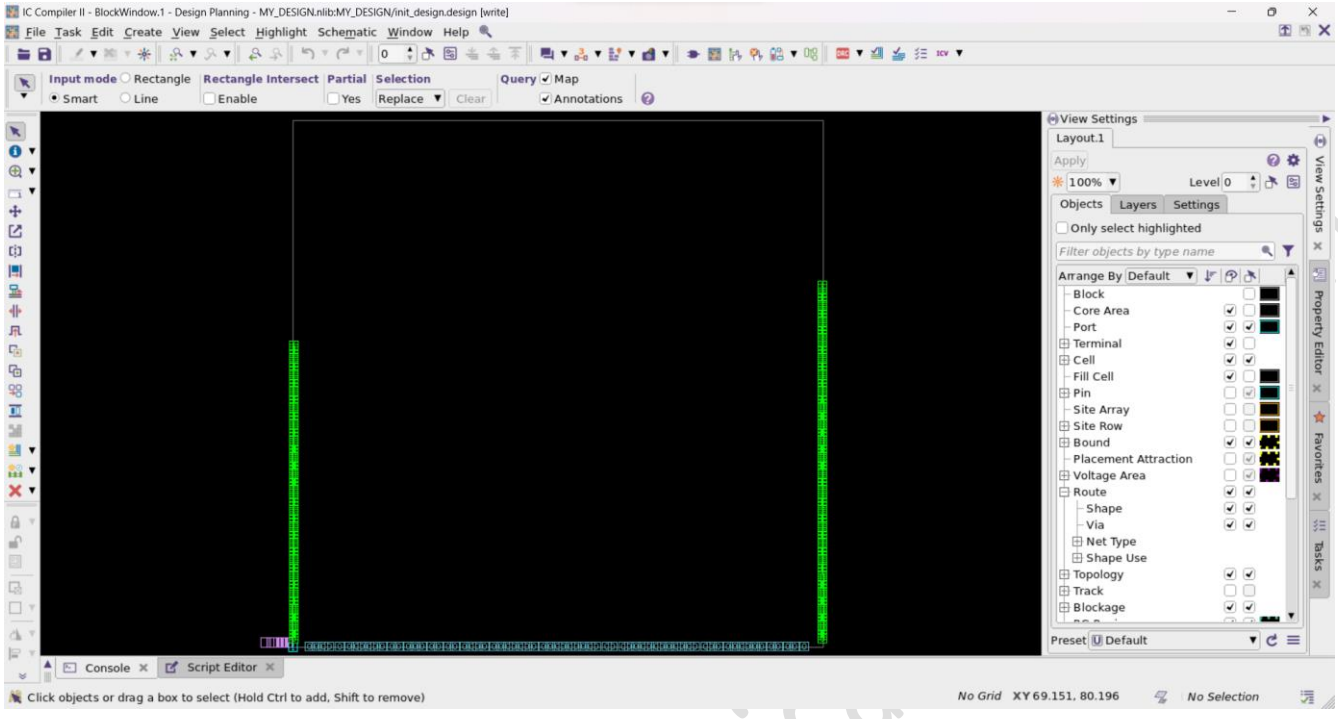

Figure 4.18: Our design [4].

7. Zoom in left side you can see blocks

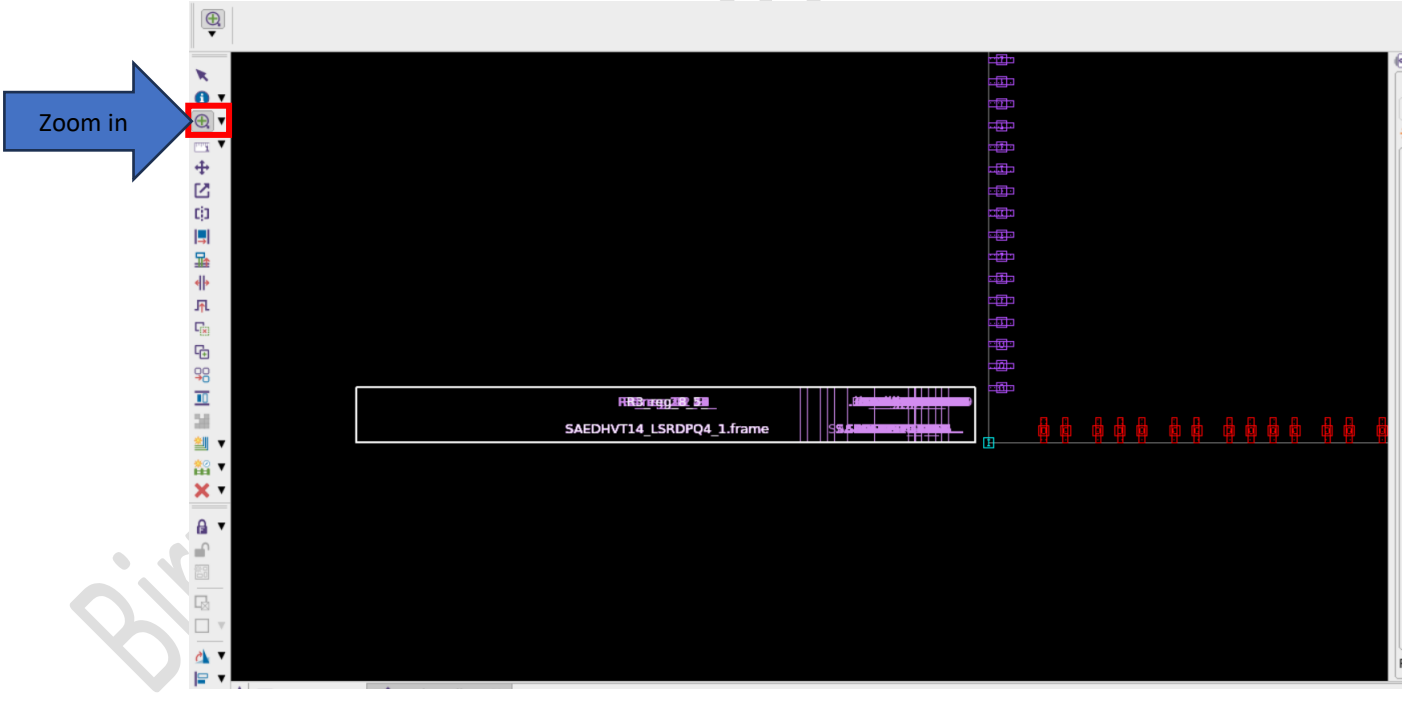

Figure 4.19: Zoom in [4].

8. Enable the “Site Row” & “Cell Site” in the View Settings/Objects Tab

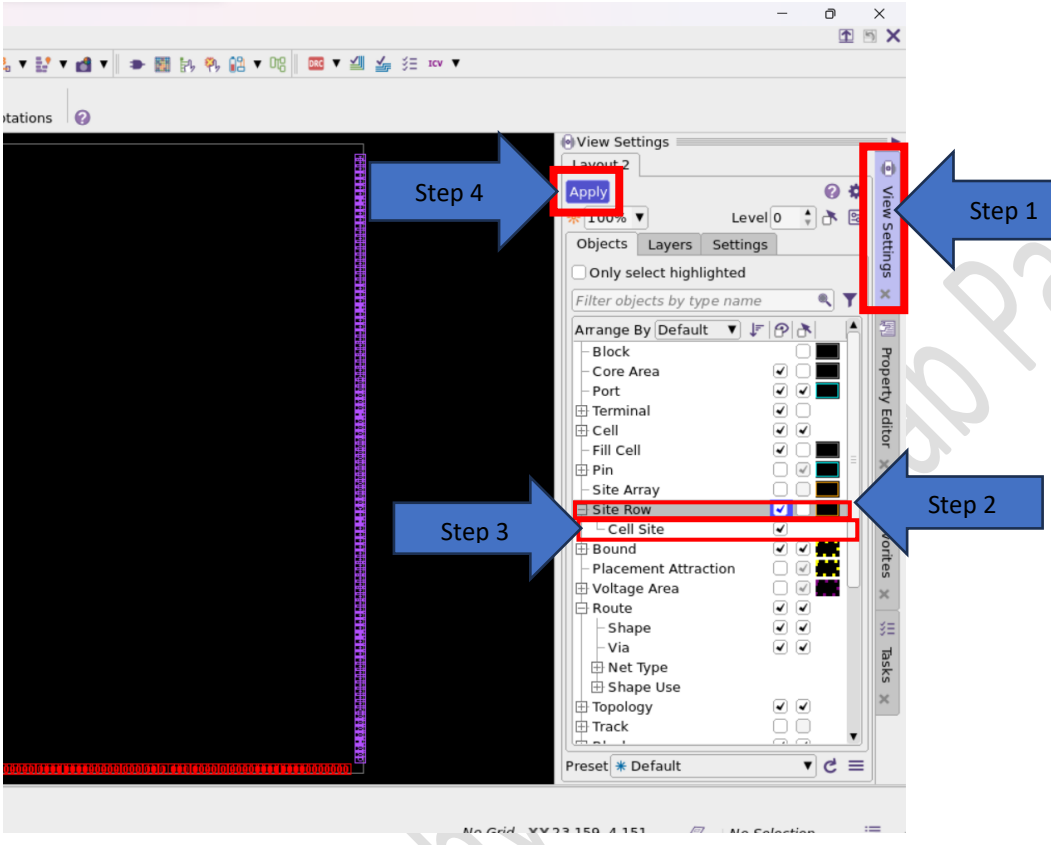

Figure 4.20: Enable the Site Row & Cell Site [4].

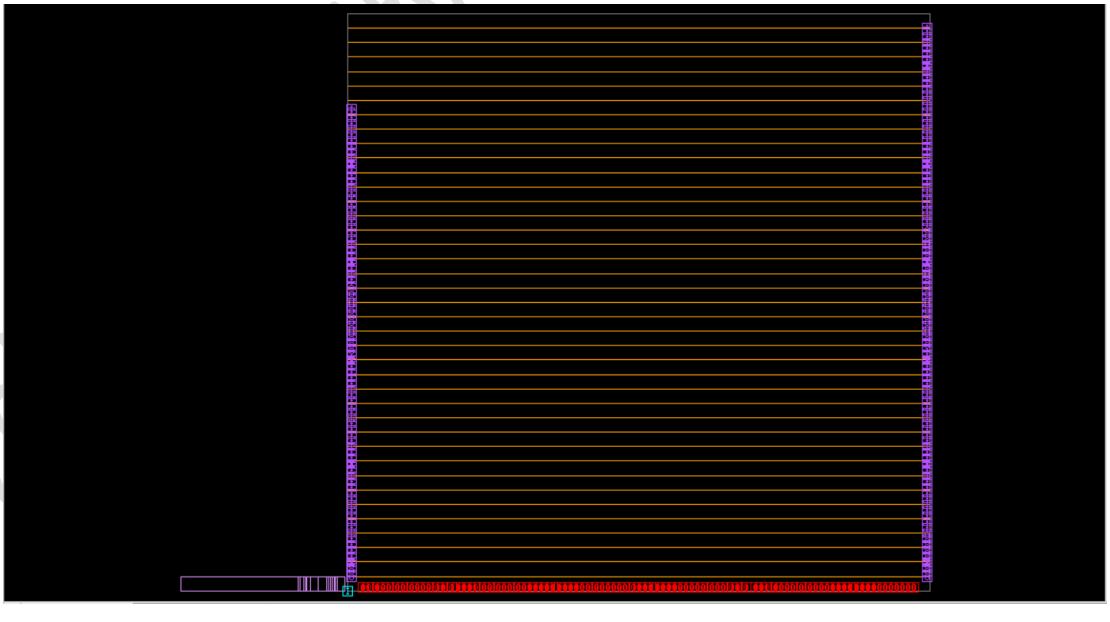

Figure 4.21: The Site Row & Cell Site on the design [4].

9. **Vertical orange lines are "Cell Sites" where each standard cell width must be a multiple of the cell site width (Measure it in the GUI). Horizontal orange lines are "Site Rows", and they have a height difference equal to the height of the shortest standard cell in the design(Measure in the gui). To verify this, move one of the unplaced cells in the bottom left corner to the design area.**

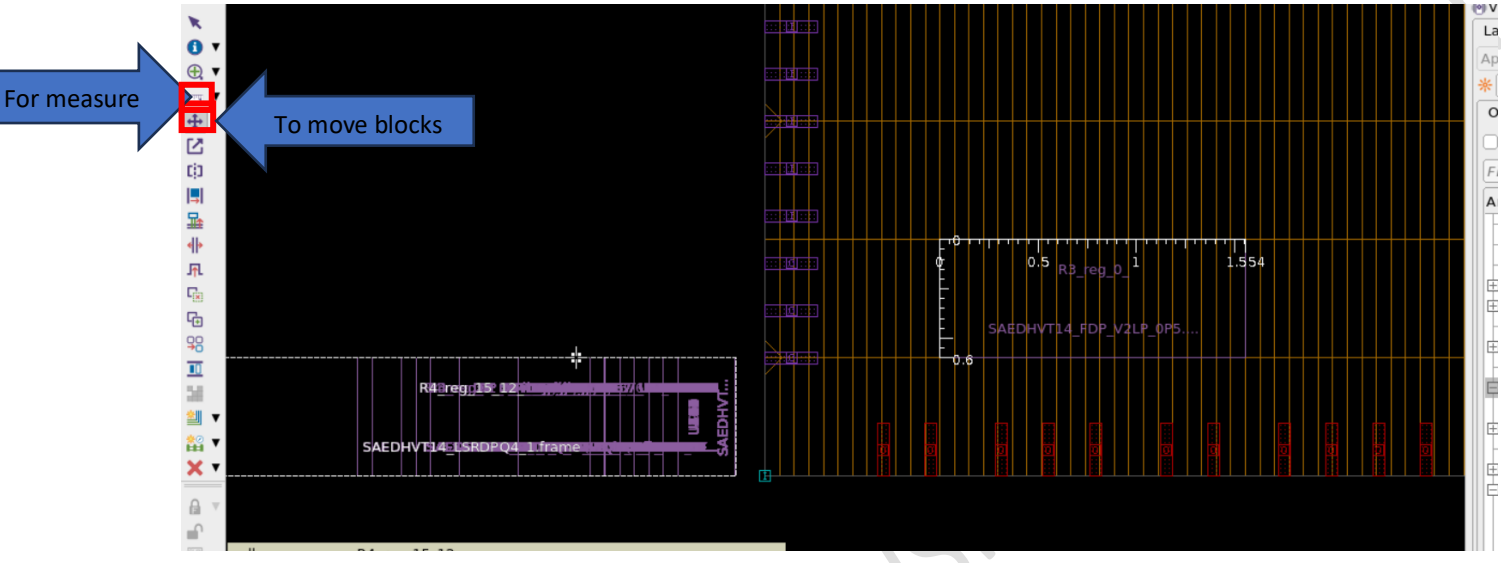

Figure 4.22: Measure the width and length of block [4].

10. **Select the blue squares at the bottom left of the block corner, then type in the shell "get_selection" , you should be getting 2 results. Note down those net names for later.**

Figure 4.23: Use get selection command [4].

11. Execute the following commands to create the power grid for the chip and observe the effects of the highlighted commands in the gui:

```
connect_pg_net -automatic
remove_vias [get_vias -of [get_nets {VDD VSS}]]
remove_shapes [get_shapes -of [get_nets {VDD VSS}]]
remove_terminals *
create_pg_mesh_pattern mesh_pattern -layers { {{vertical_layer: M9}
{width: 2.4} {pitch: 48} {spacing: interleaving}}  {{horizontal_layer:
M8} {width: 0.84} {pitch: 16.8} {spacing:
interleaving}}  {{vertical_layer: M7} {width: 0.84} {pitch: 16.8}
{spacing: interleaving}} }
set_pg_strategy mesh_strategy -core -pattern {{pattern:
mesh_pattern}{nets: {VDD VSS}}} -blockage {macros: all}
create_pg_std_cell_conn_pattern std_cell_pattern
set_pg_strategy std_cell_strategy -core -pattern {{pattern:
std_cell_pattern}{nets: {VDD VSS}}}
compile_pg -ignore_via_drc
create_terminal -of_objects [get_shapes -of_objects [get_nets {VDD
VSS}] -filter layer.name==M9]
create_placement -effort very_low
place_pins -self
```

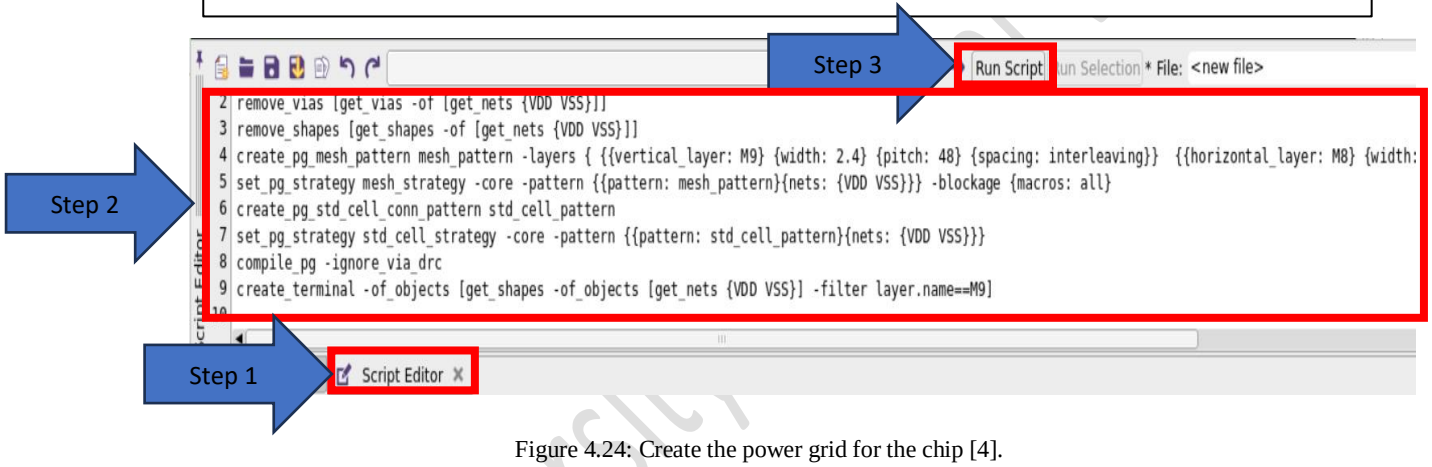

Figure 4.24: Create the power grid for the chip [4].

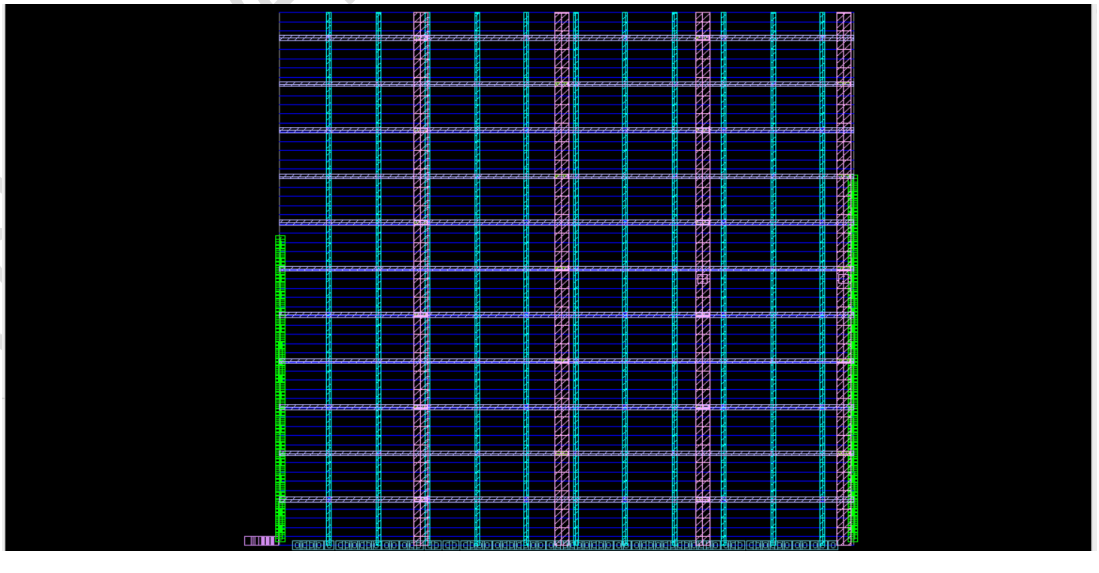

Figure 4.25: Design layers [4].

12. **Check that all the ports are now placed by in the design by checking the bottom left corner, there should be no "blue" squares which are unassigned ports. Execute the command "change_selection [get_ports]" and confirm none are not assigned onto a metal shape.**

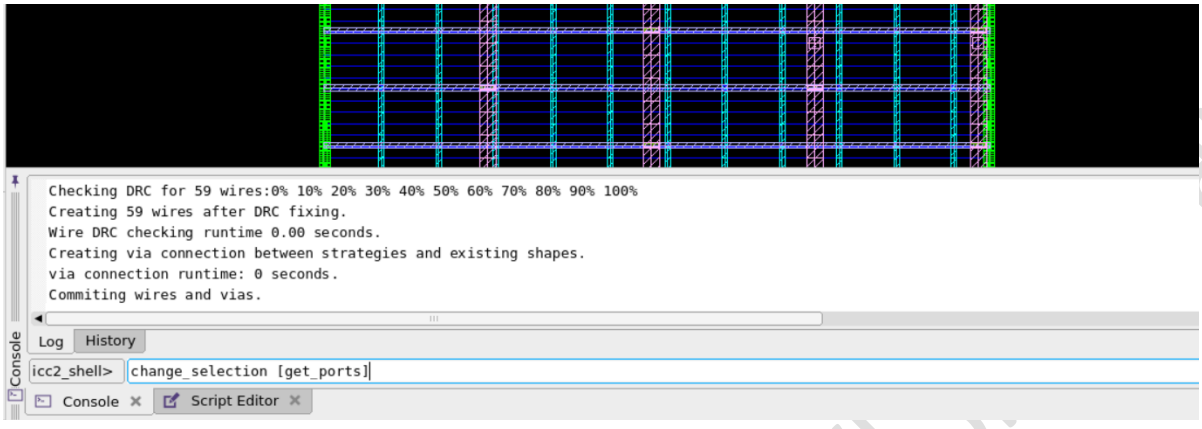

Figure 4.26: Ports placed by in the design [4].

To see all the ports make brightness 25%

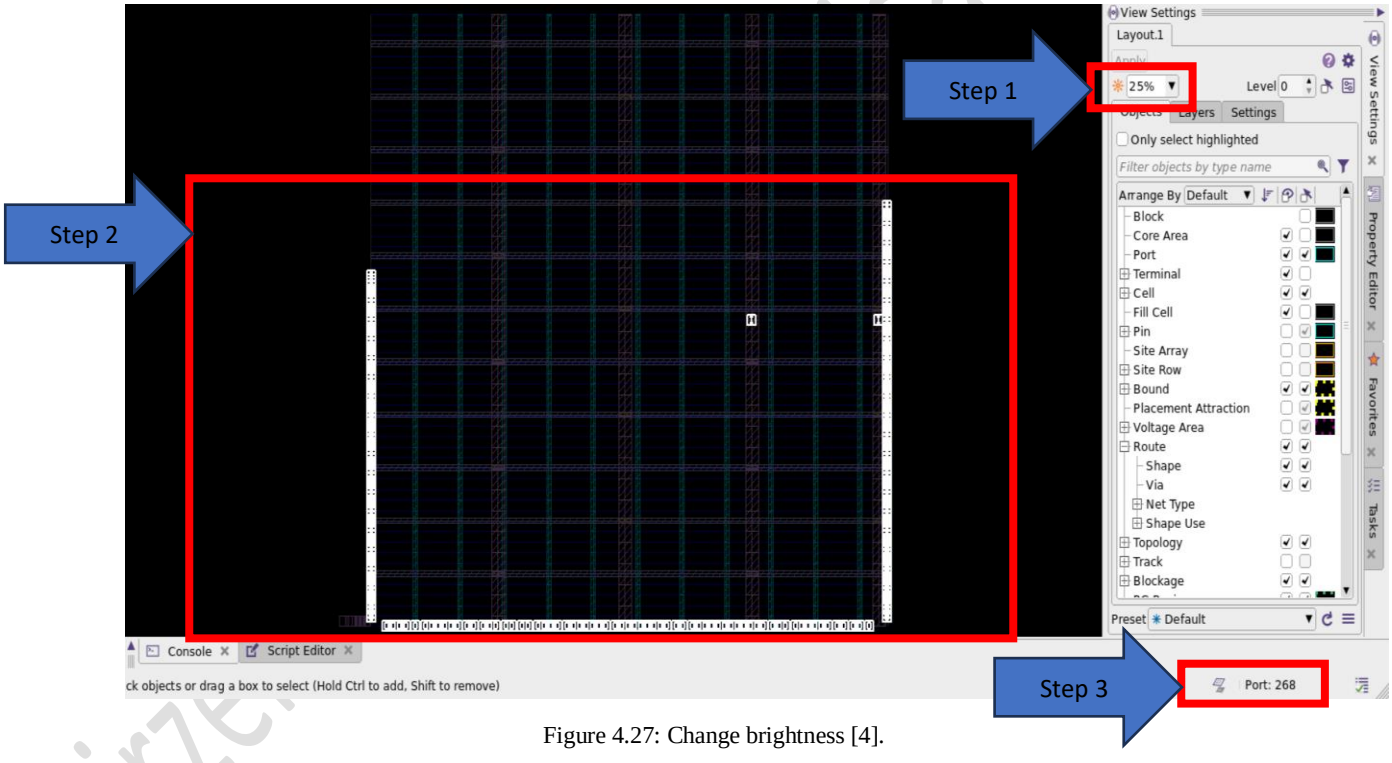

Figure 4.27: Change brightness [4].

13. How assigned ports should look like zoomed in:

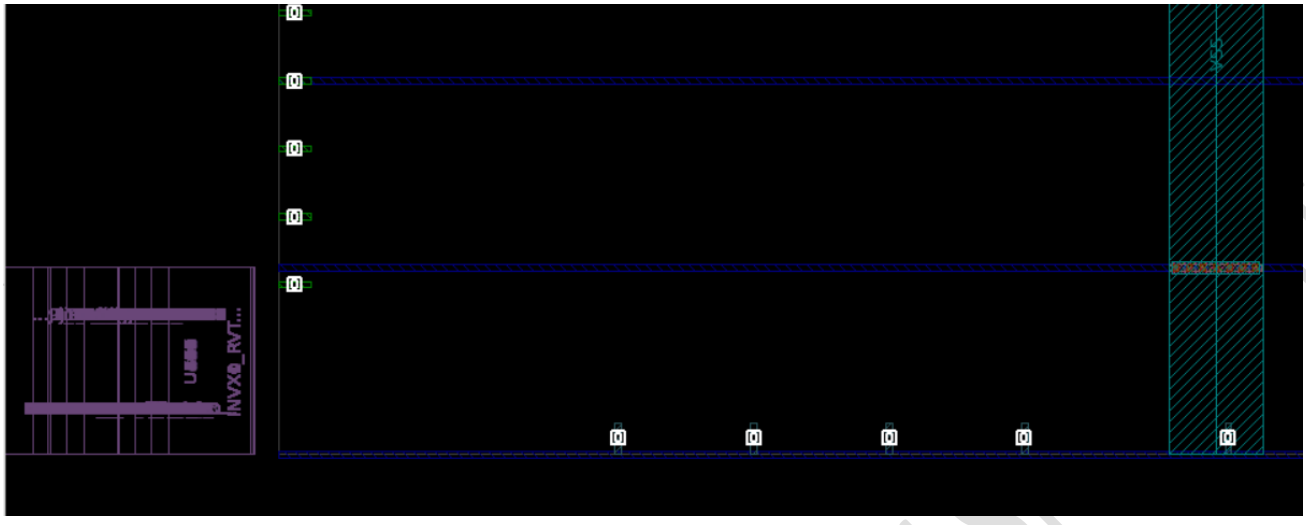

Figure 4.28: Ports zoomed in [4].

14. Save the block with the power connections using: save_block

```
icc2_shell> save_block
Information: Saving block 'MY_DESIGN.nlib:MY_DESIGN/init_design.design'
1
icc2_shell>
```

Figure 4.29: Save blocks [4].

4.6 References:

[1] https://vlsidesignbasic.com/floorplan-basic/

[2] https://myvlsibasics.blogspot.com/2019/10/floorplan-vlsi.html

[3] https://siliconvlsi.com/power-planning/

[4] Synopsys: Synopsys | EDA Tools, Semiconductor IP and Application Security Solutions

Experiment No. 5 - Placement

5.1 Objectives

- Load design with floorplan & powerplan information.
- Perform placement of standard cells in the core area.
- Check placement status, congestion and timing.

5.2 Equipment Required

- Synopses tool - IC Compiler II
- PuTTY
- Xming

5.3 Pre-Lab

1. Read the experiment to learn how to use the tool.

5.4 Introduction

5.4.1 Placement

Placement is the process of placing standard cell in the design. the tool determines the location of each standard cell on the die. the tool places these based on the algorithms which it uses internally.

Placement does not just place the standard cells available in the synthesized netlist. it also optimizes the design. placement also determines the rout ability of design. Placement will be driven by different criteria like timing driven, congestion driven and power optimization. Automated Standard Cell Placement for placing the Standard Cells in Placement Tracks.

The goal of placement optimization timing, power and area, minimum congestion, Minimal cell density, pin density and congestion hot-spots and Minimal timing DRVs

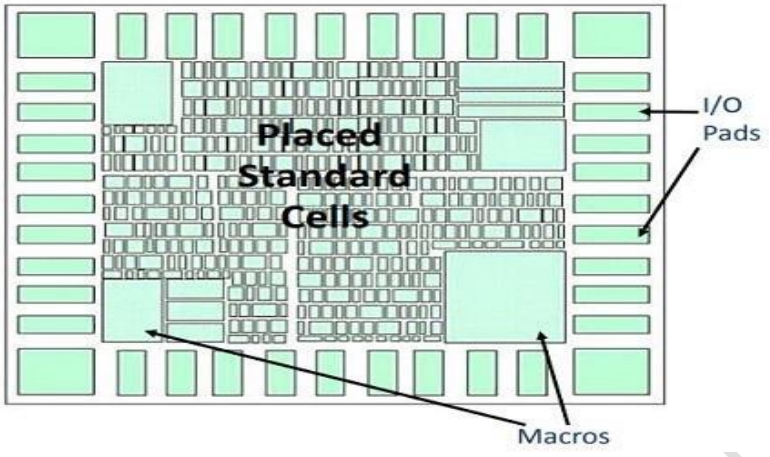

Figure 5.1: Placed standard cells [1].

Inputs of Placement:

- Technology file (.tf)
- Netlist
- SDC
- Library files (.lib & .lef) & TLIJ+ file
- Floorplan & Powerplan DEF file

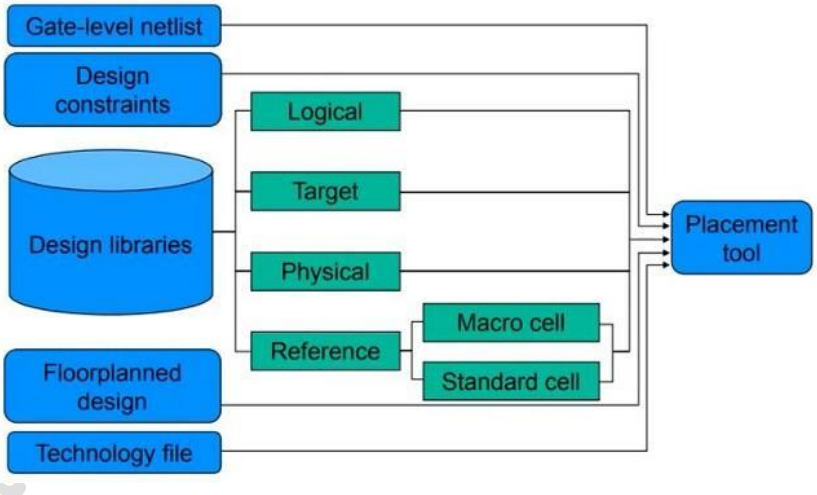

Figure 5.2: dataflow for files to make placement [1].

Things to be checked before placement

- Check for any missing/extra placement & routing blockages.
- Don't use cell list and whether it is properly applied in the tool.
- Don't touch on cells and nets (make sure that, these are applied).
- Better to have limit the local density (otherwise local congestion can create issue in routing/eco

stage).

- ❖ Understand all optimization options and placement switches set in the tool.
- ❖ There should not be any high WNS timing violations.
- ❖ Make sure that clock is ideal network.
- ❖ Take care of integration guidelines of any special IPs (these won't be reported in any of the checks).
- ❖ Have custom scripts to check these guidelines.
- ❖ Fix all the hard macros and pre-placed cells.
- ❖ Check the pin access

Placement Methods

- ❖ Timing Driven Placement
  - To Refine placement based on congestion, timing and power
  - To optimize large sets of path delays
  - Net Based
- ❖ Congestion Driven Placement
  - To distance standard cell instances from each other such that more routing tracks are created between them

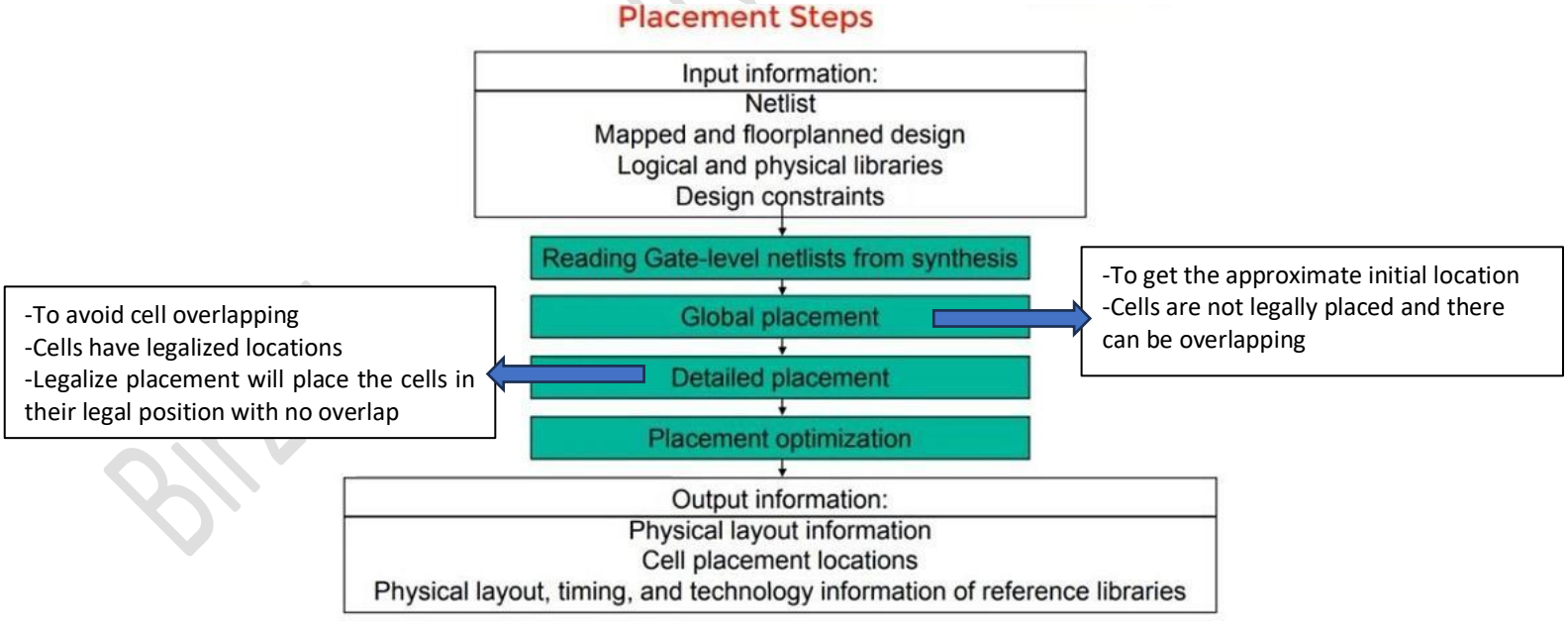

Figure 5.3: Placement Steps [1].

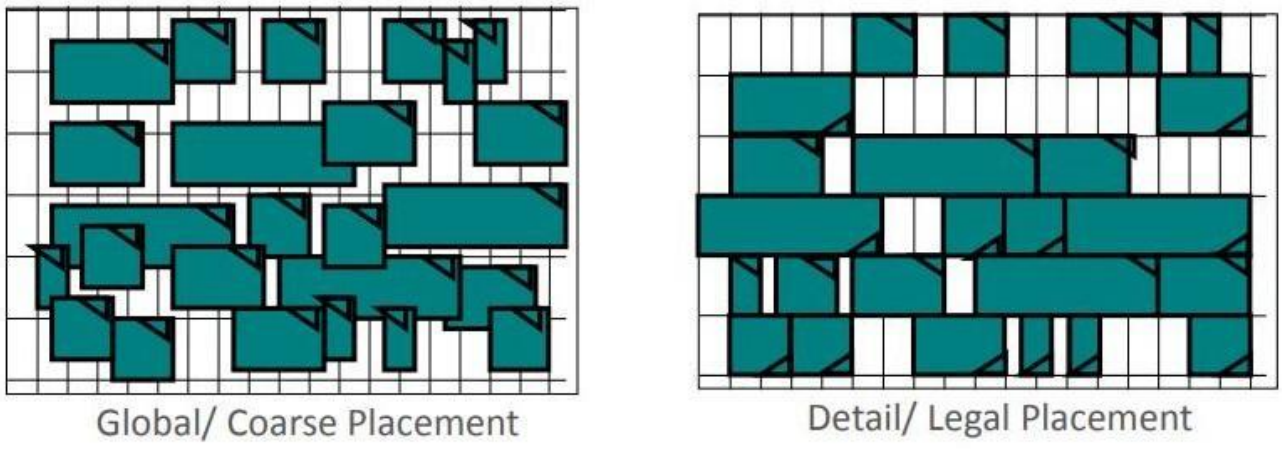

Figure 5.4: type of placement [1].

5.5 Procedure

5.5.1 Environment Setup:

- **Go to your EXP directory by going to your user directory (cd ~), then do: (mkdir -p ./ST_VLSI/EXP5)**

```
[hanan@synopsys ~]$ cd
[hanan@synopsys ~]$ mkdir -p ./ST_VLSI/EXP5
```

Figure 5.5: Make directory [2].

- **go to EXP5 directory (cd ./ST_VLSI/EXP5)**

```
[hanan@synopsys ~]$ cd ./ST_VLSI/EXP5
[hanan@synopsys EXP5]$
```

Figure 5.6: Open the directory [2].

- **Copy the tool environment to your user directory(change the word in red color):**

  cp -R /home/iccTA/all_labs/icc2_EXP5_env/* .

```
[hanan@synopsys EXP5]$ cp -R  /home/iccTA/all_labs/icc2_EXP5_env/*
```

Figure 5.7: Copy the tool environment [2].

- **To view the copied files: ls**

```
[hanan@synopsys EXP5]$ ls
inputs  Makefile  rm_icc2_pnr_scripts  rm_setup
```

Figure 5.8: Show files [2].

- **Link your synthesized netlist to the inputs directory:**

  ln -s /home/<username>/ST_VLSI/EXP3/results/{DESIGN}.mapped.v inputs/.

```
[hanan@synopsys EXP5]$ ln -s /home/hanan/ST_VLSI/EXP3/results/MY_DESIGN.mapped
 inputs/.
```

Figure 5.9: Link synthesized netlist to the input's directory from exp 3 [2].

- **Link your synthesized netlist to the inputs directory**:

  cp -RPi /home/< username >/ST_VLSI/EXP4/<DESIGN_NAME>.nlib .

```
[hanan@synopsys EXP5]$ cp -RPi /home/hanan/ST_VLSI/EXP4/MY_DESIGN.nlib .
[hanan@synopsys EXP5]$ ls
inputs  Makefile  MY_DESIGN.nlib  rm_icc2_pnr_scripts  rm_setup
```

Figure 5.10: Link synthesized netlist to the input's directory from exp 4 [2].

- **Check the SDC files that'll define the timing constraints for your design:**

  i. **gedit inputs/sdc/init_design.mcmm.tcl**

```
[hanan@synopsys EXP5]$ gedit inputs/sdc/init_design.mcmm.tcl
```

Figure 5.11: Check the SDC files [2].

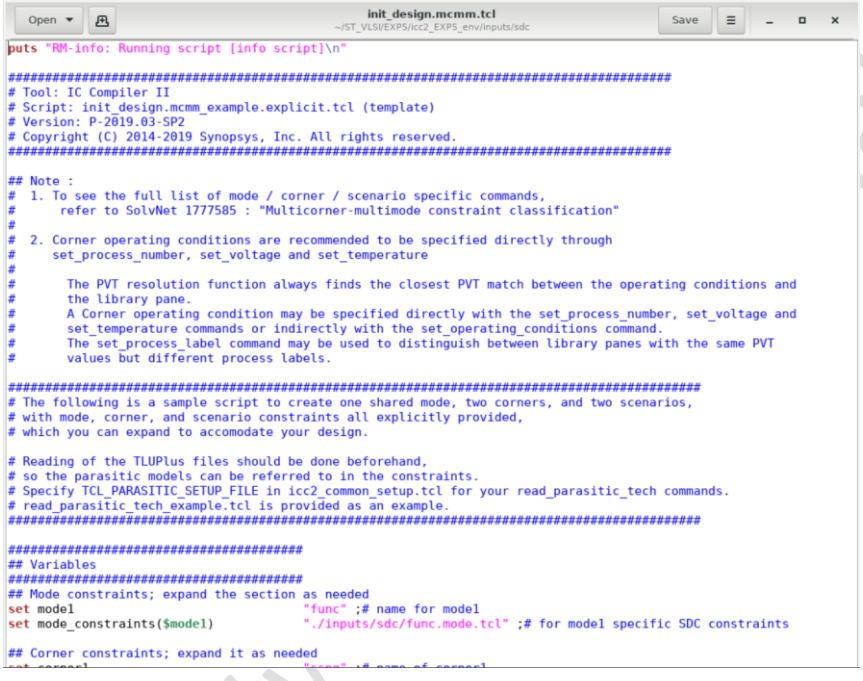

Figure 5.12: Open init_design.mcmm.tcl [2].

  ii. **gedit inputs/sdc/func.mode.tcl**

```
[hanan@synopsys EXP5]$ gedit inputs/sdc/func.mode.tcl
```

Figure 5.13: Open func.mode.tcl [2].

Figure 5.14: Open func.mode.tcl [2].

iii. gedit inputs/sdc/sspg.corner.tcl

```
[hanan@synopsys EXP5]$ gedit inputs/sdc/sspg.corner.tcl
```

Figure 5.15: Open sspg.corner.tcl [2].

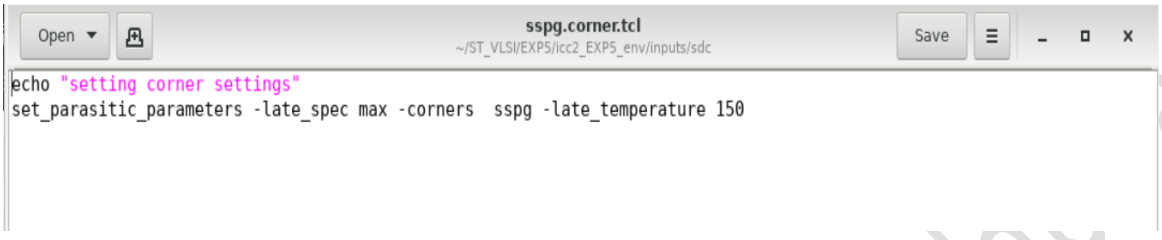

Figure 5.16: Open sspg.corner.tcl [2].

5.5.2 Run the design script: shell> make place_opt: (it will take little time)

```
[hanan@synopsys EXP5]$ make place opt
```

Figure 5.17: Run the design script [2].

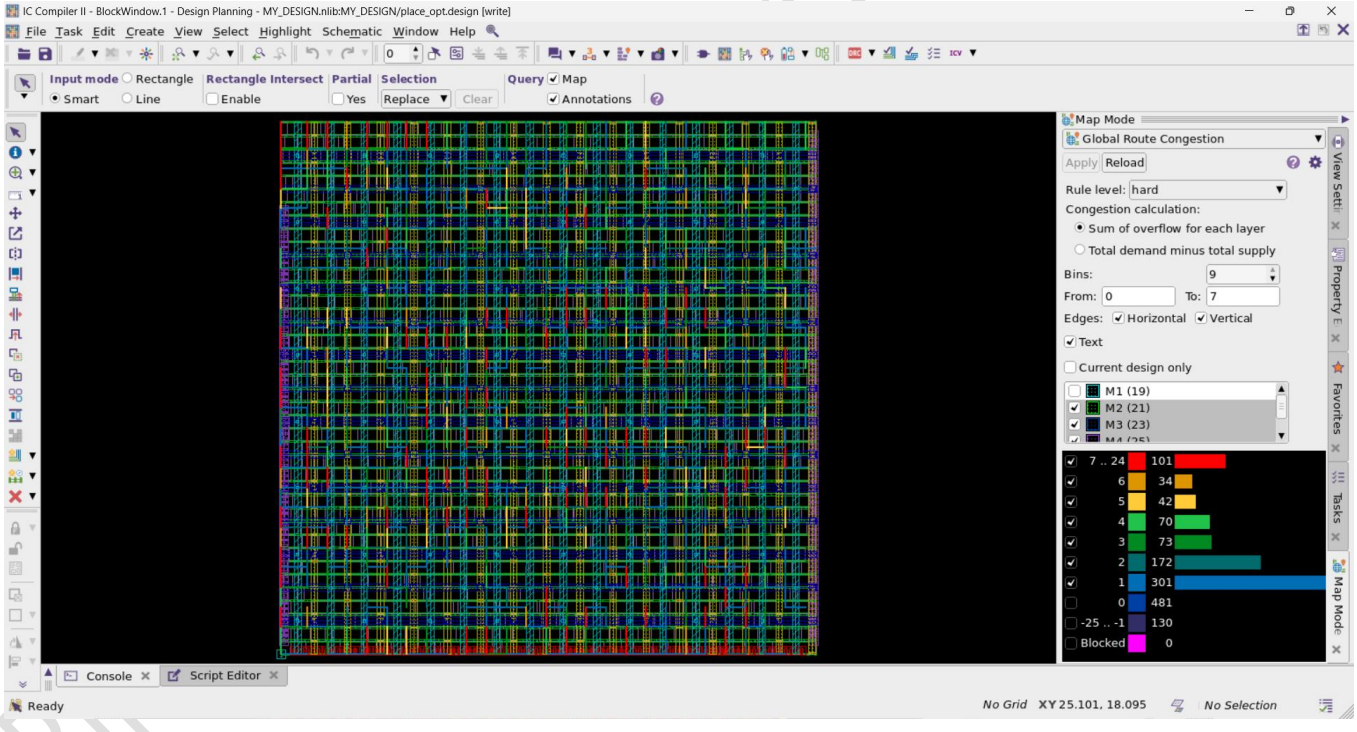

Figure 5.18: Design metal [2].

i. Check the log for error messages, use " \less logs_icc2/ place_opt.log" and search using "/^Error" (^: Shift+6), then press Enter - If you see any error messages, check with TA (ignore PLACE-007).To exit from file press "q" character

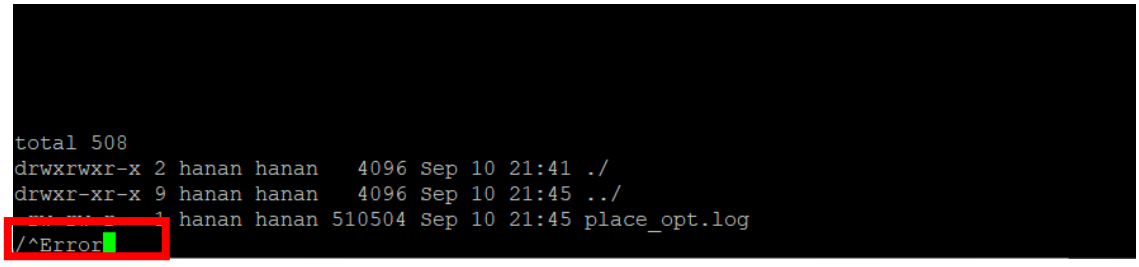

Figure 5.19: Check the log for error messages [2].

5.5.3 Check the design interactively after it's finished:

1. **shell> icc2_shell [enter] (it will take a time)**

2. **icc2_shell> open_lib <DESIGN_NAME>.nlib [enter]**

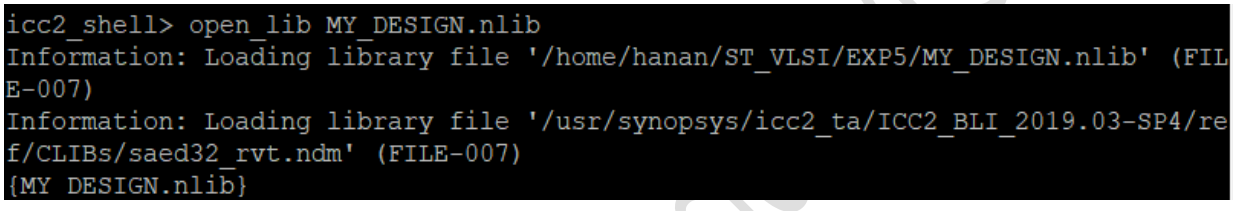

Figure 5.20: Open design library [2].

3. **icc2_shell> list_blocks [enter]**

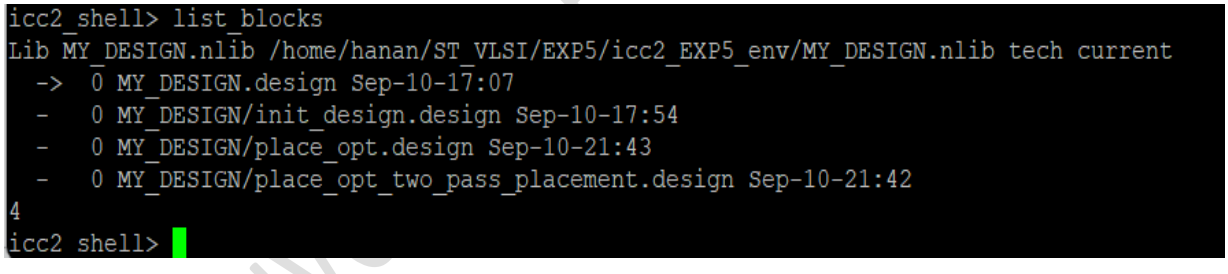

Figure 5.21: block list [2].

4. **icc2_shell> open_block <DESIGN_NAME>/place_opt.design [enter]**

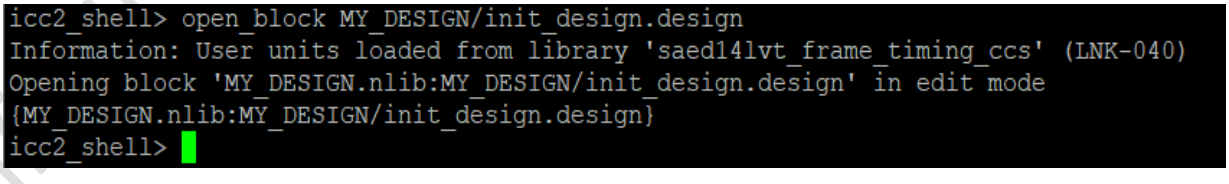

Figure 5.22: Open block [2].

5. **icc2_shell> start_gui**

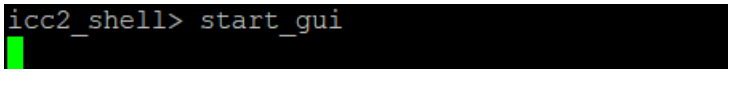

Figure 5.23: gui start [2].

6. Press F to reset the gui view

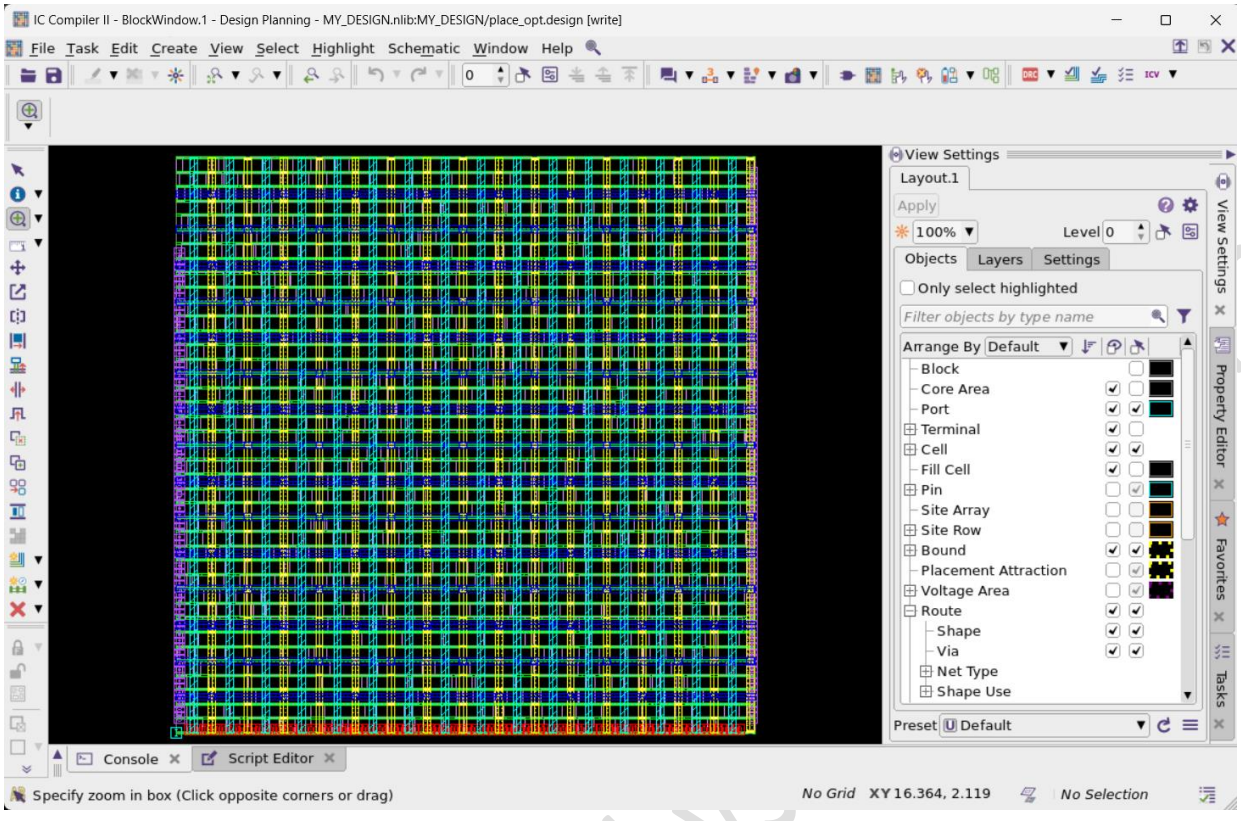

Figure 5.24: Our design [2].

7. Enable the "Site Row" & Disable "Route"" in the View Settings/Objects Tab

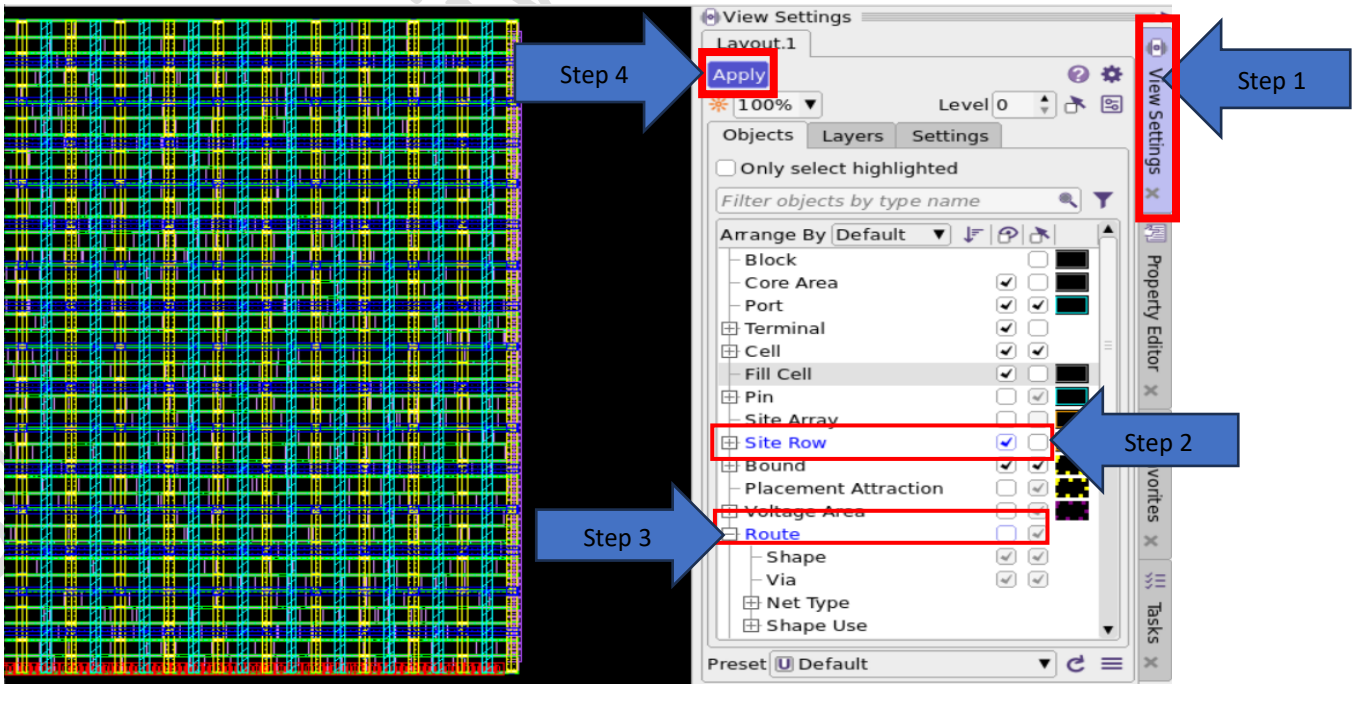

Figure 2.25: Enable the Site Row & Cell Site [2].

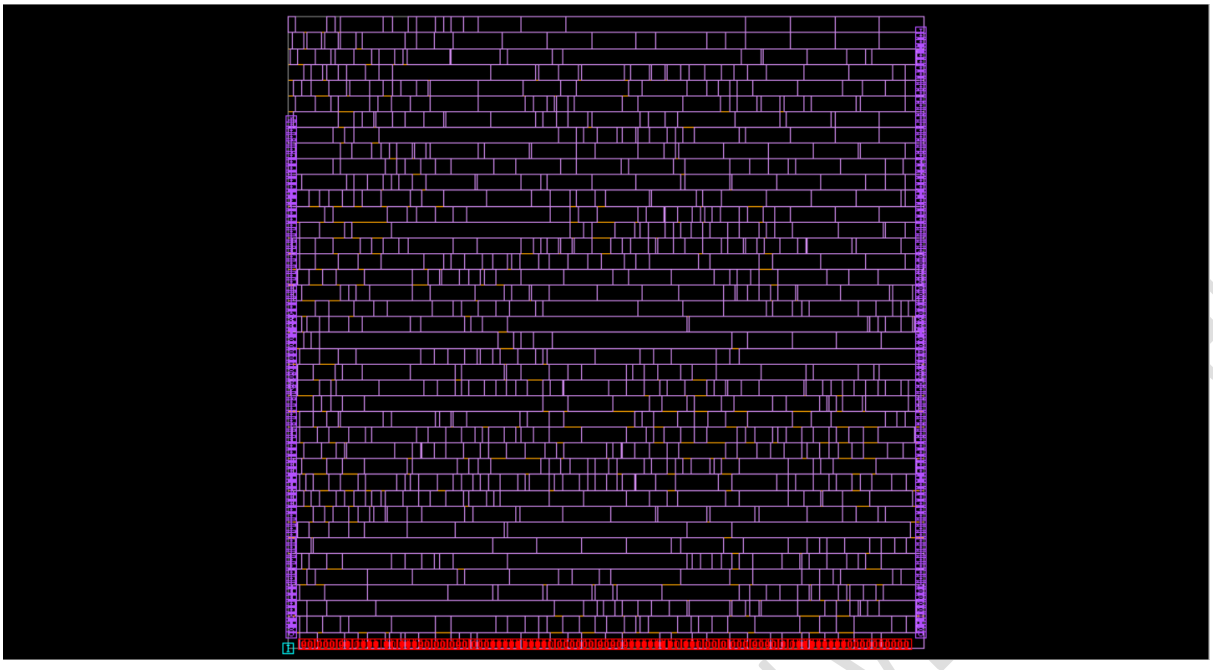

Figure 5.26: The Site Row & Cell Site on the design [2].

8. Locate the cell sites and confirm every cell is sitting within a row

Figure 5.27: Zoom in [2].

9. **Enable “Pin” & select any of the top/bottom metal shapes around any cell. Check the name is “VSS” or “VDD” by doing right click->clicking on “Properties”.**

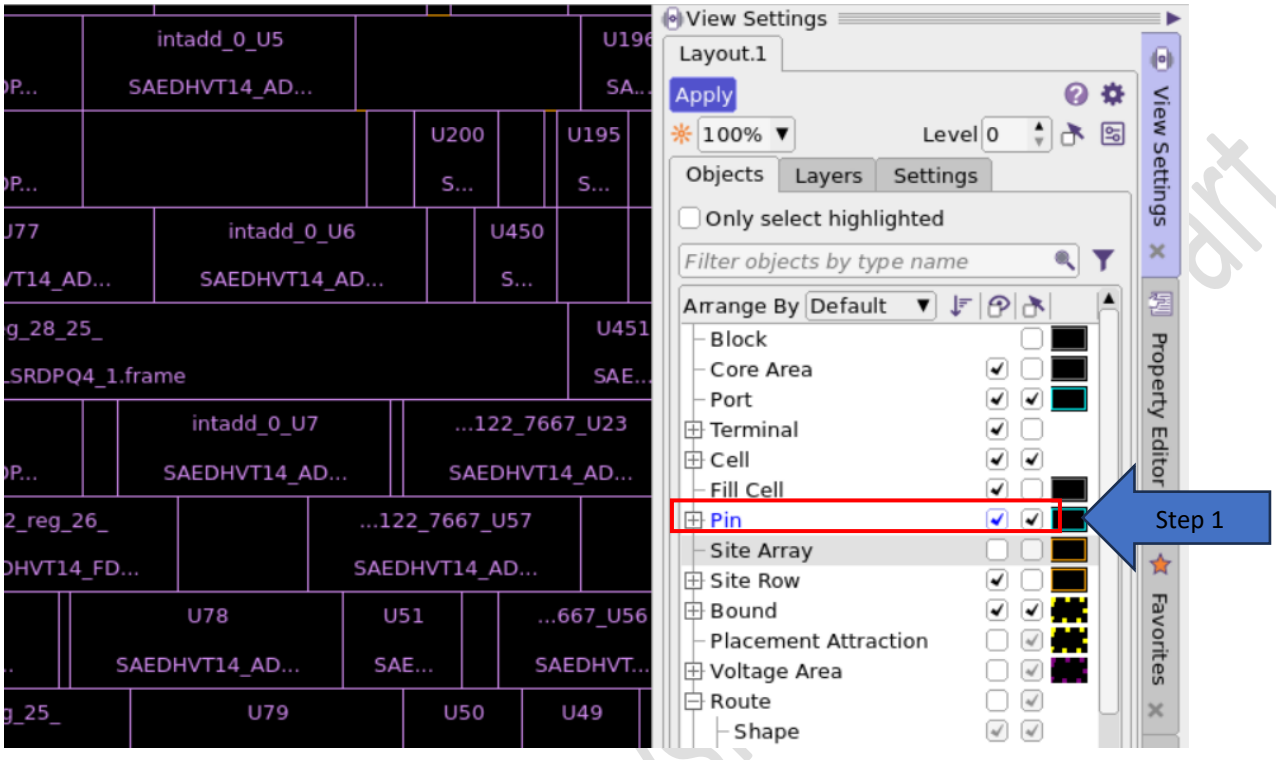

Figure 5.28: Enable “Pin” [2].

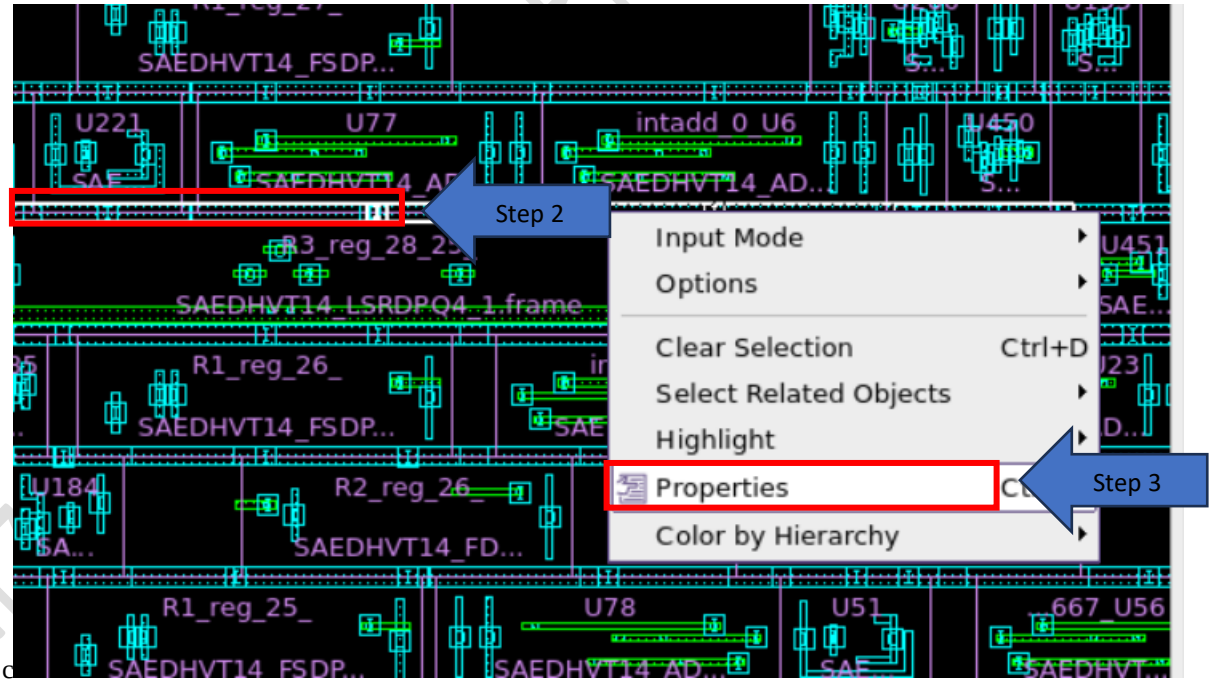

Figure 5.29: Select metal properties [2].

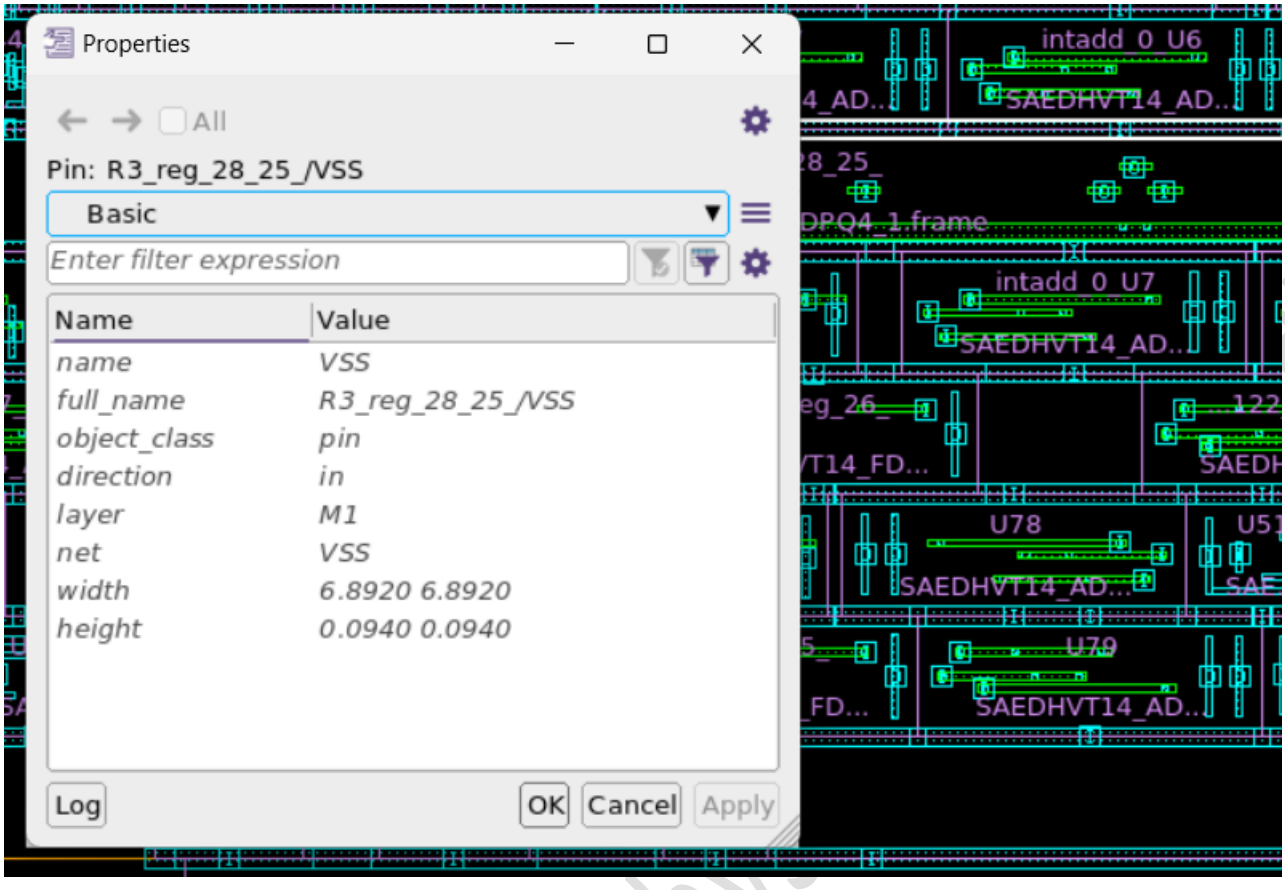

Figure 5.30: Check the name is “VSS” or “VDD” for metal [2].

10. After **confirming** the shape name, enable “Route” and select the long shape sitting on the row edge, and click properties and confirm it’s the same name as the one for the cell pin

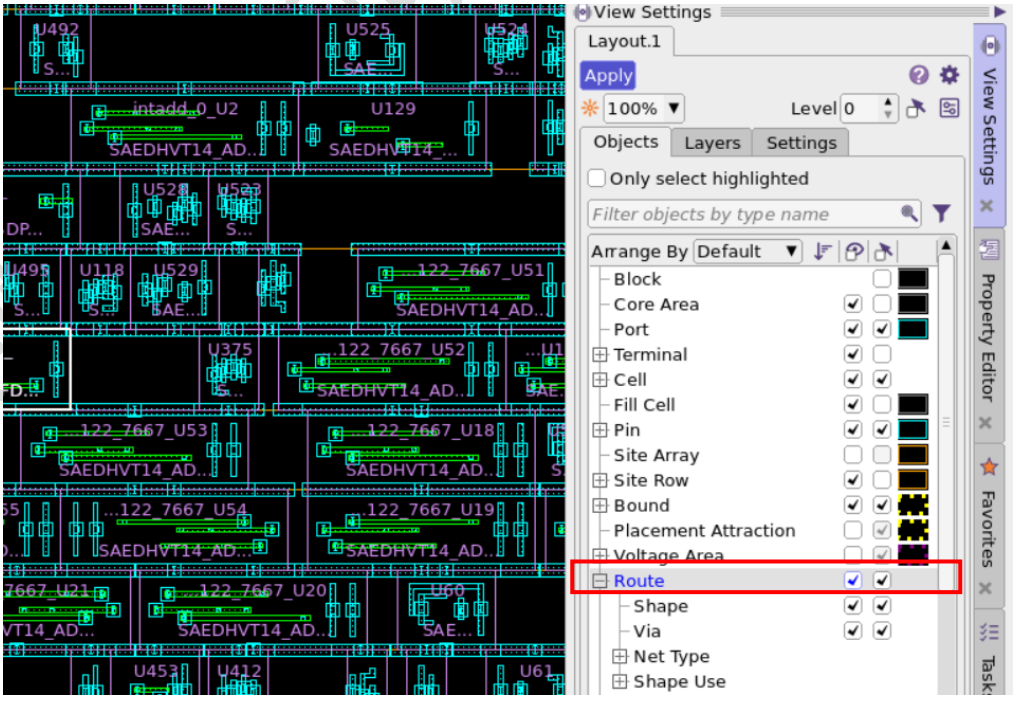

Figure 5.31: Enable “Route” [2].

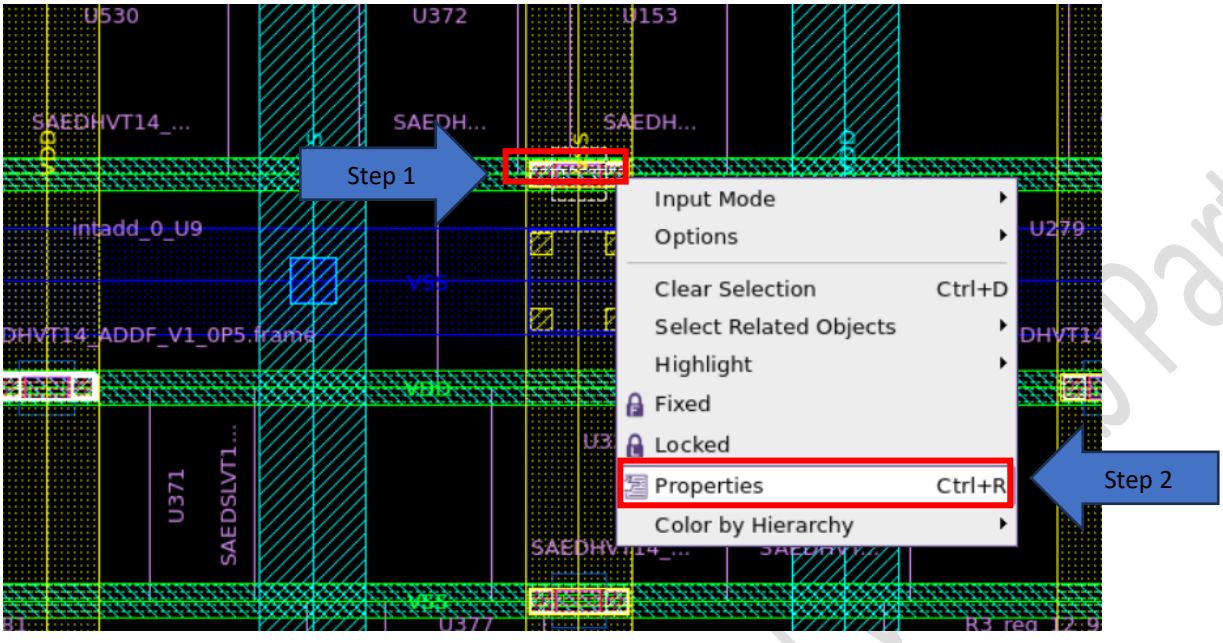

Figure 5.32: Select cell properties [2].

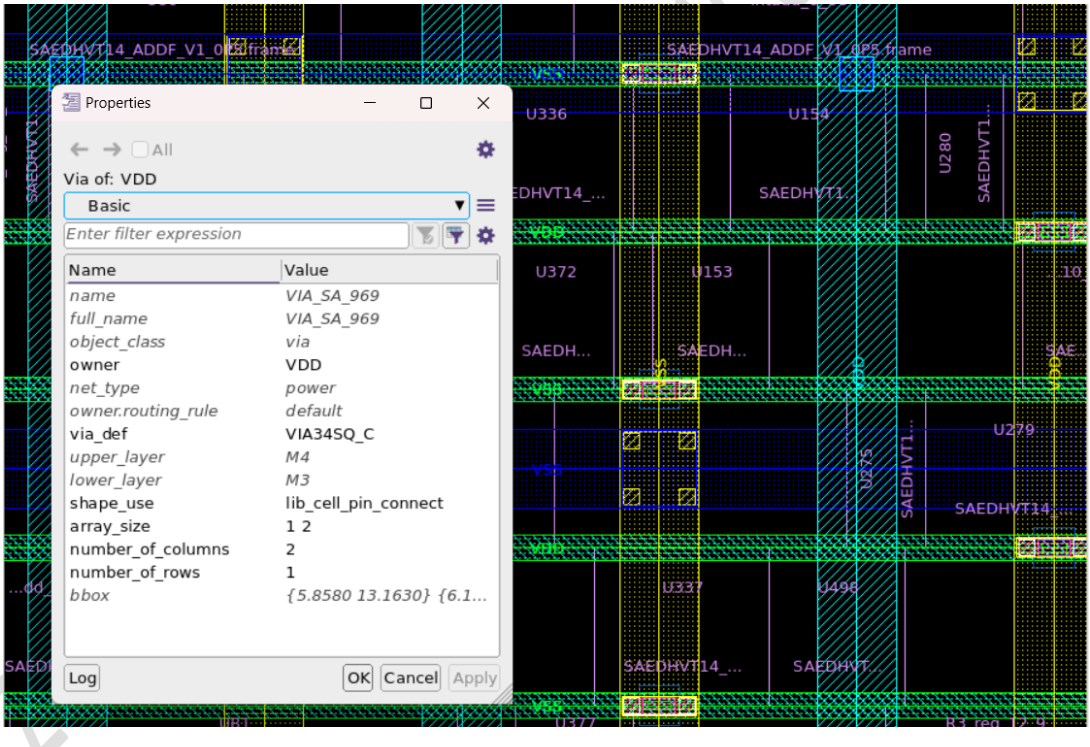

Figure 5.33: Check the name is “VSS” or “VDD” for cell [2].

11. Perform the following reports* and make sure you have 0 reported violations:

- **icc2_shell>** check_legality **[enter]**
- **icc2_shell>** report_placement **[enter]**

12. Check density in the design->Reload

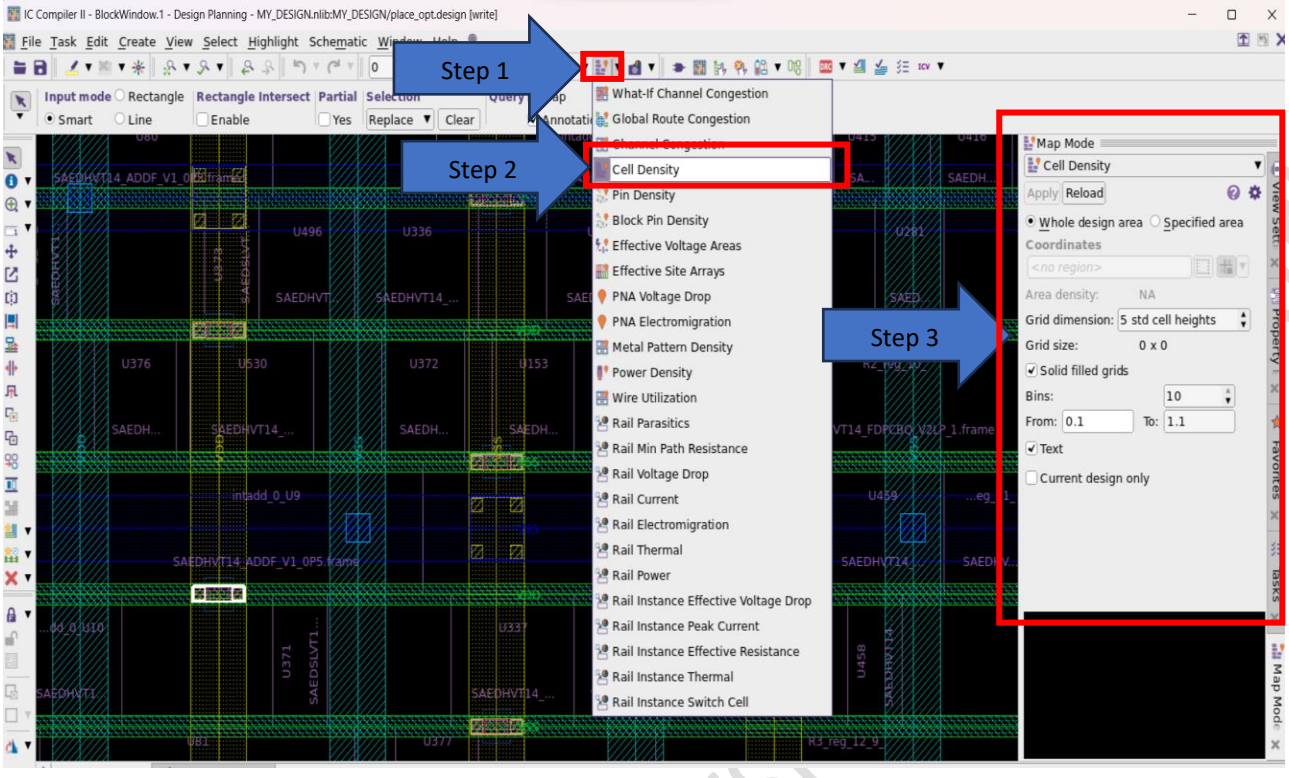

Figure 5.34: Check density in the design [2].

13. *do “report_scenarios, report_modes, report_corners, report_clocks” and confirm all the settings from inputs/sdc/* are read in correctly.

14. Re-create the main_clk clock with double the default frequency.

15. Confirm the new clock is created by performing “report_clocks” again.

```
icc2_shell> report_clocks
****************************************
Report : clock
Design : MY_DESIGN
Mode   : func
Version: P-2019.03-SP2
Date   : Mon Sep 11 16:28:49 2023
****************************************


Attributes:
    p - Propagated clock
    G - Generated  clock
    U - Unexpanded generated clock

Clock          Period   Waveform            Attrs     Sources
------------------------------------------------------------------------------
main_clk         1.00   {0 0.5}                       {clk}
```

Figure 5.35: Report clocks in the design [2].

16. Execute the command “save_block” and then “exit”

```
icc2_shell> save_block
Information: Saving block 'MY_DESIGN.nlib:MY_DESIGN/place_opt.design'
1
```

Figure 5.36: Save block [2].

*To understand what each report is showing - do “man <command name>” and read the command DESCRIPTION section

5.6 References :

[1] Placement | Physical Design | VLSI Back-End Adventure (vlsi-backend-adventure.com)

[2] Synopsys: Synopsys | EDA Tools, Semiconductor IP and Application Security Solutions

Experiment No. 6 - Clock Synthesis and Timing Analysis

6.1 Objectives

- Load design with placed, legalized standard cells & updated timign constraints.
- Perform clock tree synthesis and routing for the design.
- Check timing, clock skew, congestion and timing.

6.2 Equipment Required

- Synopses tool - IC Compiler II
- PuTTY
- Xming

6.3 Pre-Lab

1. Read the experiment to learn how to use the tool.

6.4 Introduction

In the world of Very Large-Scale Integration (VLSI), where millions or even billions of transistors are packed onto a single chip, ensuring precise synchronization and optimal performance is a formidable challenge. This is where clock synthesis and timing analysis come into play. These essential components of the VLSI design process are instrumental in guaranteeing that digital circuits operate correctly and meet their performance specifications.

6.4.1 Clock Synthesis: Crafting the Heartbeat of Digital Circuits

Clock Generation: Clock synthesis begins with the generation of clock signals that orchestrate the dance of electrons within a digital circuit. These clock signals serve as the metronome, dictating when and how various components should perform their tasks.

Clock Sources: Various sources, such as crystal oscillators, phase-locked loops (PLLs), and delay-locked loops (DLLs), are employed to create these clock signals. PLLs and DLLs, in particular, allow designers to generate multiple clock domains with precise phase relationships, a vital aspect of modern chip design.

Frequency and Phase Control: Control over clock frequency and phase is paramount. PLLs and DLLs enable designers to fine-tune these parameters, ensuring that the circuit meets its timing requirements and performance goals.

Skew Control: Clock skew, the variation in arrival times of clock signals at different points on the chip, is a nemesis to precision. Minimizing skew is vital to prevent setup and hold time violations and maintain the chip's reliability and performance.

Clock Distribution: Once generated, clock signals must be distributed evenly and with minimal skew throughout the chip. Clock tree synthesis (CTS) techniques are employed to construct balanced and efficient clock distribution networks.

6.4.2 Timing Analysis: The Art of Ensuring Temporal Harmony

Setup and Hold Time Analysis: Timing analysis ensures that signals arrive at flip-flops and latches with sufficient setup and hold times for accurate data capture. Violations of these constraints can lead to incorrect operation and chip failure.

Propagation Delay Analysis: Propagation delay analysis quantifies the time it takes for signals to traverse the combinational logic between flip-flops. This analysis ensures that signals meet their required arrival times at sequential elements.

Critical Path Analysis: In any design, there's a critical path—the longest route from input to output. Analyzing this path identifies the maximum achievable clock frequency and highlights areas where optimization is needed.

Clock Domain Crossing (CDC) Analysis: In multi-clock domain designs, CDC analysis ensures that data crossing from one clock domain to another is handled correctly, guarding against data corruption and metastability issues.

Static Timing Analysis (STA): STA tools provide a comprehensive assessment of timing under various operating conditions and process variations. They calculate setup and hold time violations, clock-to-q delays, and other critical metrics.

Clock Domain Analysis: This analysis ensures that different clock domains within a chip are properly synchronized, and any asynchronous communication between domains is managed correctly.

Margin Analysis: Timing analysis includes margin calculations to account for process, voltage, and temperature variations, ensuring chip reliability even under the harshest conditions.

Constraints: Timing constraints, such as clock-to-q constraints and input delays, guide synthesis and optimization tools to achieve desired timing objectives.

Clock synthesis and timing analysis are the unsung heroes of VLSI design. They breathe life into complex digital circuits, orchestrating the precision and performance that modern technology demands. As VLSI designs continue to push the boundaries of complexity, clock synthesis and timing analysis remain indispensable tools in the hands of skilled designers, ensuring that our digital world keeps ticking seamlessly and efficiently.

6.5 Procedure

6.5.1 Environment Setup:

- **Go to your EXP directory by going to your user directory (cd ~), then do: (mkdir -p ./ST_VLSI/EXP6)**

```
[hanan@synopsys ~]$ cd
[hanan@synopsys ~]$ mkdir -p ./ST_VLSI/EXP6
```

Figure 6.1: Make directory [1].

- **Go to EXP6 directory (cd ./ST_VLSI/EXP6)**

```
[hanan@synopsys ~]$ cd ./ST_VLSI/EXP6
[hanan@synopsys EXP6]$
```

Figure 6.2: Open the directory [1].

- **Copy the tool environment to your user directory (change the word in red color):**

  cp -R /home/iccTA/all_labs/icc2_EXP6_env/* .

```
[hanan@synopsys EXP6]$ cp -R /home/iccTA/all_labs/icc2_EXP6_env/*  .
```

Figure 6.3: Copy the tool environment [1].

- **To view the copied files: ls**

```
[hanan@synopsys EXP6]$ ls
inputs  Makefile  rm_icc2_pnr_scripts  rm_setup
```

Figure 6.4: Show files [1].

- **Link your synthesized netlist to the input's directory:**

  ln -s /home/<username>/ST_VLSI/EXP3/results/{DESIGN}.mapped.v inputs/.

```
[hanan@synopsys EXP6]$ ln -s /home/hanan/ST_VLSI/EXP3/results/MY_DESIGN.mapped.v
 inputs/.
[hanan@synopsys EXP6]$ ls
inputs  Makefile  rm_icc2_pnr_scripts  rm_setup
```

Figure 6.5: Link synthesized netlist to the input's directory from exp 3 [1].

- **Copy the previous experiments database**:

  cp -RPi /home/< username >/ST_VLSI/EXP5/<DESIGN_NAME>.nlib .

```
[hanan@synopsys EXP6]$ cp -RPi /home/hanan/ST_VLSI/EXP5/MY_DESIGN.nlib .
cp: overwrite './MY_DESIGN.nlib/MY_DESIGN/design_label.init_design/cstrs/design.
cintrf.gz'? ^C
[hanan@synopsys EXP6]$ ls
inputs  Makefile  MY_DESIGN.nlib  rm_icc2_pnr_scripts  rm_setup
```

Figure 6.6: Copy the previous experiments database from exp 5 [1].

6.5.2 Run the design script: shell> make clock_opt_opto: (it will take time)

```
[hanan@synopsys EXP6]$ make clock_opt_opto
```

Figure 6.7: Run the design script [1].

i. **Check the log for error messages, use " \less logs_icc2/ clock*.log" and search using "/^Error" (^: Shift+6) ,then press Enter - If you see any error messages, check with TA (ignore PLACE-007).To exit from file press "q" character.**

```
total 1016
drwxrwxr-x 2 hanan hanan   4096 Sep 11 18:55 ./
drwxr-xr-x 9 hanan hanan   4096 Sep 11 18:58 ../
-rw-rw-r-- 1 hanan hanan 339129 Sep 11 18:55 clock_opt_cts.log
-rw-rw-r-- 1 hanan hanan 691188 Sep 11 18:58 clock_opt_opto.log
/^Error
```

Figure 6.8: Check the log for error messages [1].

6.5.3 Check the design interactively after it's finished:

1. **shell> icc2_shell [enter] (it will take a time)**
2. **icc2_shell> open_lib <DESIGN_NAME>.nlib [enter]**

```
icc2_shell> open_lib MY_DESIGN.nlib
Information: Loading library file '/home/hanan/ST_VLSI/EXP6/icc2_EXP6_env/MY_DES
IGN.nlib' (FILE-007)
Information: Loading library file '/usr/synopsys/saed/stdcell_lvt/ndm/saed14lvt_
frame_timing_ccs.ndm' (FILE-007)
Information: Loading library file '/usr/synopsys/saed/stdcell_hvt/ndm/saed14hvt_
frame_timing_ccs.ndm' (FILE-007)
Information: Loading library file '/usr/synopsys/saed/stdcell_slvt/ndm/saed14slv
t_frame_timing_ccs.ndm' (FILE-007)
Information: Loading library file '/usr/synopsys/saed/stdcell_rvt/ndm/saed14rvt_
frame_timing_ccs.ndm' (FILE-007)
{MY_DESIGN.nlib}
icc2_shell>
```

Figure 6.9: Open design library [1].

3. icc2_shell> list_blocks [enter]

```
icc2_shell> list_blocks
Lib MY_DESIGN.nlib /home/hanan/ST_VLSI/EXP6/icc2_EXP6_env/MY_DESIGN.nlib tech cu
rrent
  ->   0 MY_DESIGN.design Sep-10-17:07
  -    0 MY_DESIGN/clock_opt_cts.design Sep-11-18:55
  -    0 MY_DESIGN/clock_opt_opto.design Sep-11-18:57
  -    0 MY_DESIGN/init_design.design Sep-10-17:54
  -    0 MY_DESIGN/place_opt.design Sep-11-16:29
  -    0 MY_DESIGN/place_opt_two_pass_placement.design Sep-10-21:42
6
icc2_shell>
```

Figure 6.10: Block list [1].

4. icc2_shell> open_block <DESIGN_NAME>/clock_opt_opto.design [enter]

```
icc2_shell> open_block MY_DESIGN/clock_opt_opto.design
Information: User units loaded from library 'saed14lvt_frame_timing_ccs' (LNK-04
0)
Opening block 'MY_DESIGN.nlib:MY_DESIGN/clock_opt_opto.design' in edit mode
{MY_DESIGN.nlib:MY_DESIGN/clock_opt_opto.design}
```

Figure 6.11: Open block [1].

5. icc2_shell> start_gui

```
icc2_shell> start_gui
```

Figure 6.12: Gui start [1].

6. Press F to reset the gui view

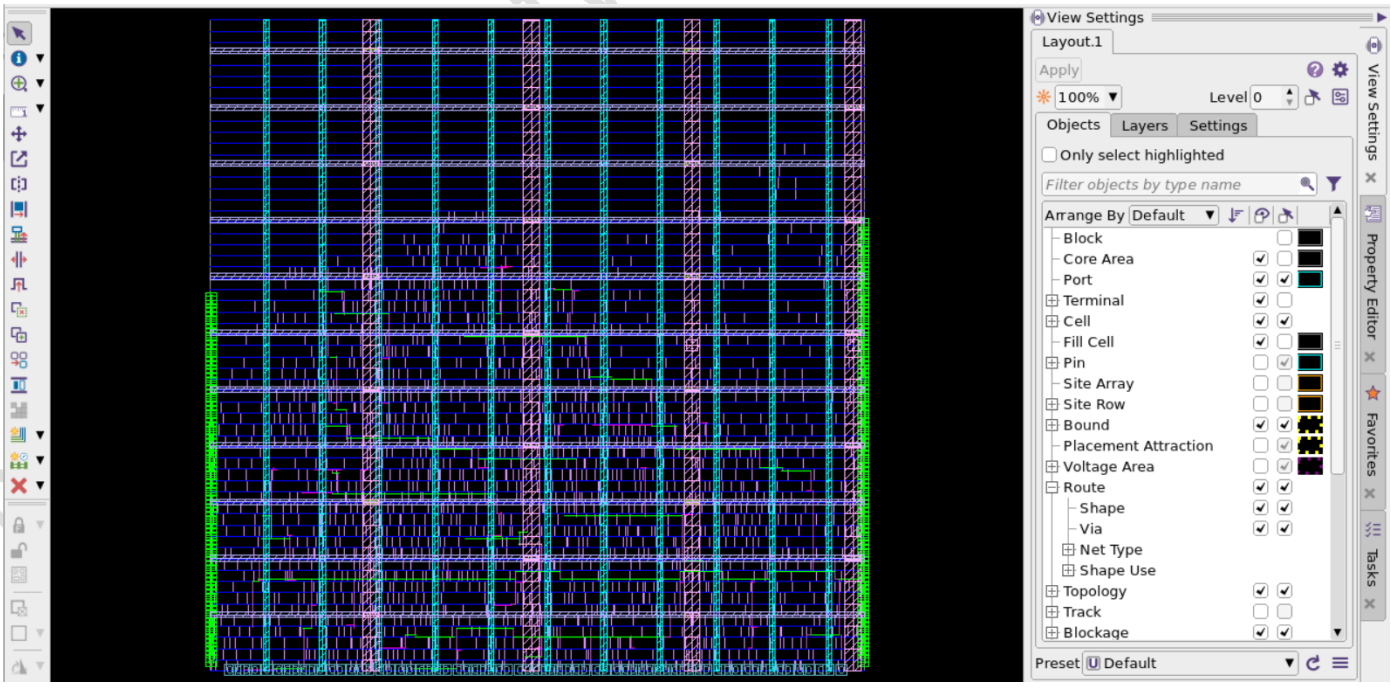

Figure 6.13: Our design [1].

7. **Disable "Route->Net Type->Power/Ground" in the View Settings/Objects Tab**

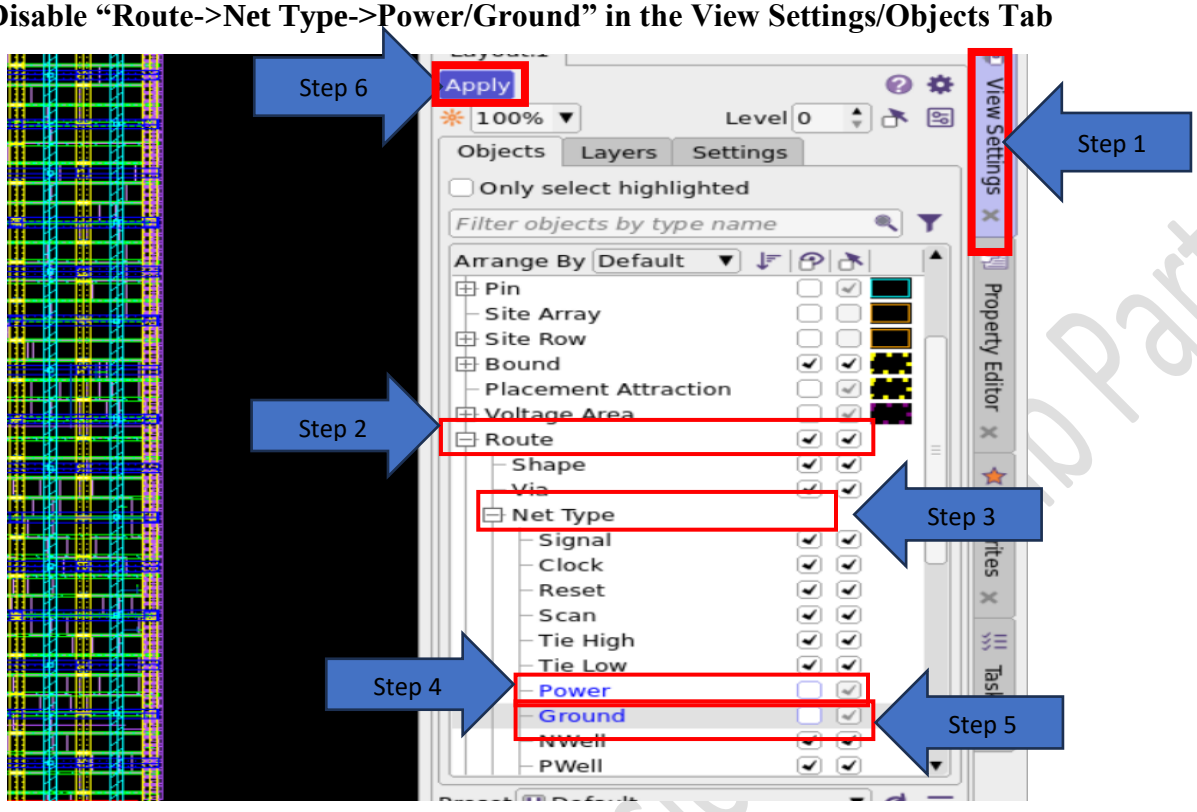

Figure 6.14: Disable the power & ground [1].

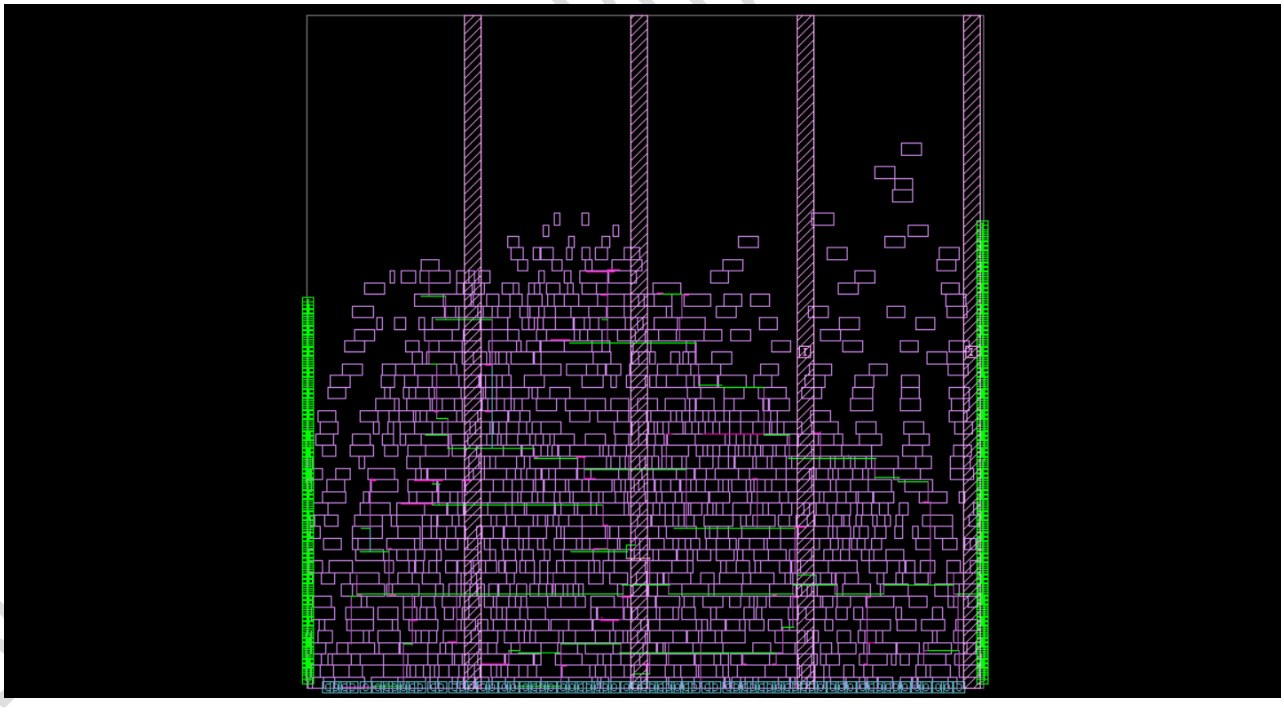

Figure 6.15: Design [1].

8. **Enable pins from view settings, and then select any of the metal shapes between cells and verify they are indeed clock nets**

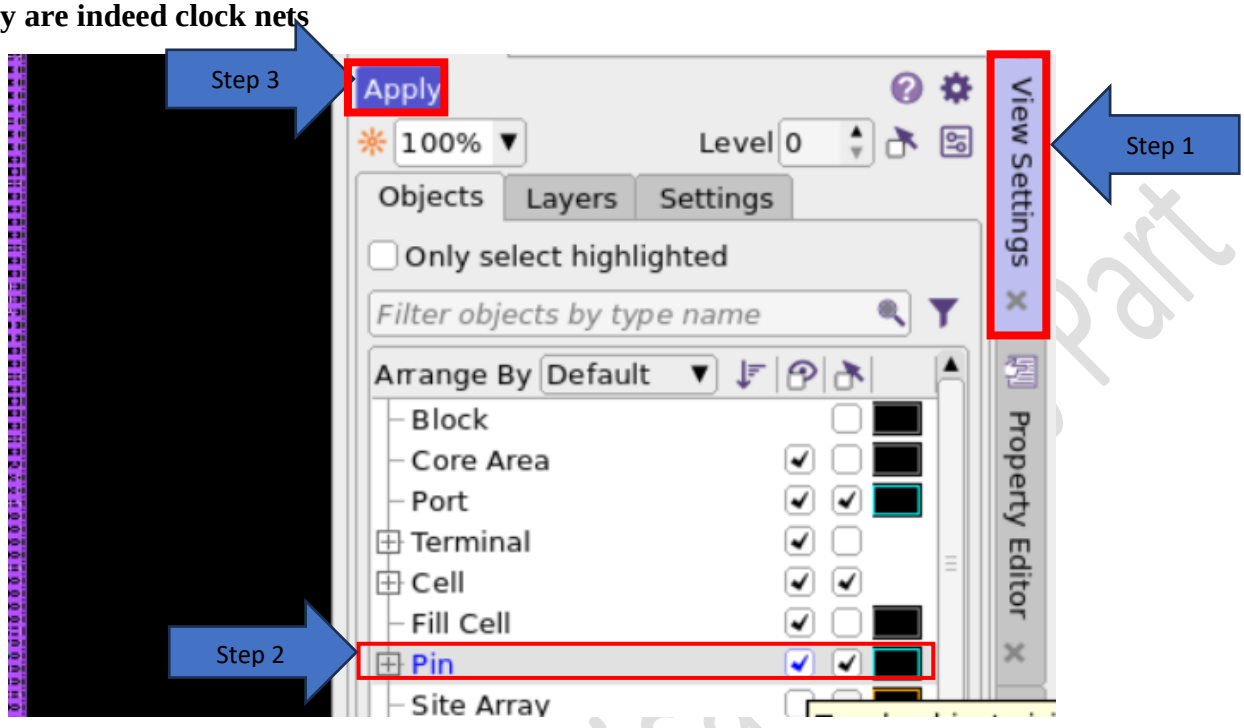

Figure 6.16: Enable pin [1].

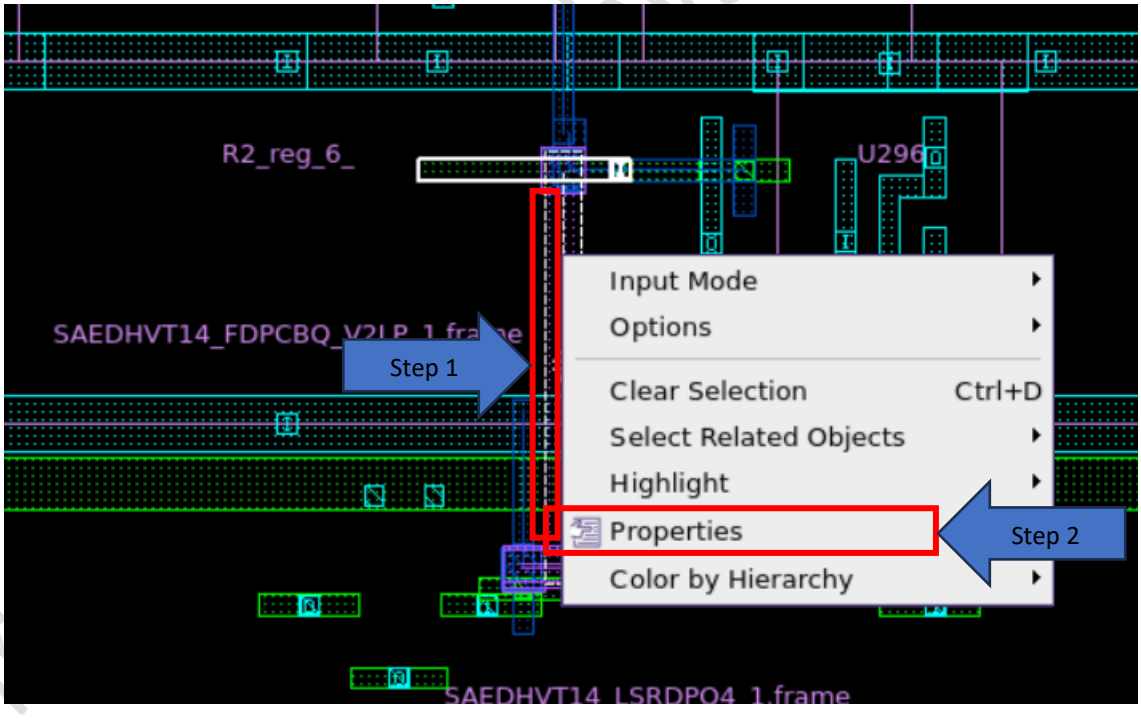

Figure 6.17: Select metal properties [1].

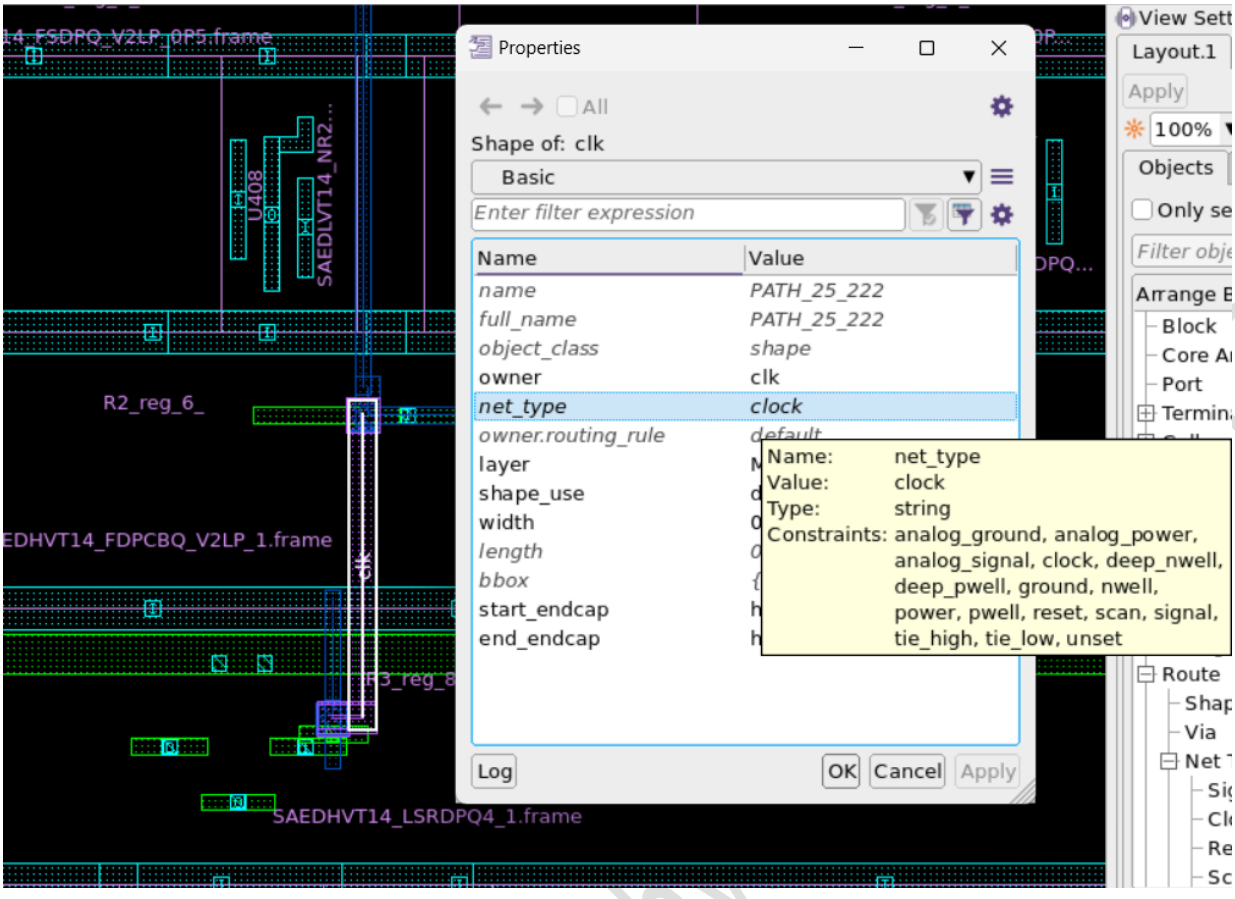

Figure 6.18: Check the name is “VSS” or “VDD” and net_type for metal [1].

9. **After confirming the shape name, execute “report_timing” in the shell, to know more about this command, do “man report_timing”.**

```
icc2_shell> report_timing
```

Figure 6.19: Run report timing [1].

```
Report : timing
        -path_type full
        -delay_type max
        -max_paths 1
        -report_by design
Design : MY_DESIGN
Version: P-2019.03-SP2
Date   : Thu Sep 14 14:34:14 2023
****************************************
Information: Timer using 'SI, Timing Window Analysis, CRPR'. (TIM-050)

  Startpoint: R3_reg_20_ (rising edge-triggered flip-flop clocked by main_clk)
  Endpoint: R4_reg_32_ (rising edge-triggered flip-flop clocked by main_clk)
  Mode: func
  Corner: sspg
  Scenario: func::sspg
  Path Group: main_clk
  Path Type: max

  Point                                              Incr       Path
  -----------------------------------------------------------------------
  clock main_clk (rise edge)                         0.00       0.00
  clock network delay (propagated)                   0.01       0.01

  R3_reg_20_/CLK (DFFX1_RVT)                         0.00       0.01 r
  R3_reg_20_/Q (DFFX1_RVT)                           0.09       0.10 f
  U241/Y (OR2X1_RVT)                                 0.04       0.14 f
  U540/Y (AND2X1_RVT)                                0.04       0.17 f
  U260/Y (NAND2X0_RVT)                               0.04       0.21 r
  U154/Y (OA21X1_RVT)                                0.04       0.25 r
  U153/Y (INVX0_RVT)                                 0.05       0.30 f
  U172/Y (AOI21X1_RVT)                               0.06       0.35 r
  U598/Y (OA21X1_RVT)                                0.04       0.39 r
  U602/Y (XNOR2X1_RVT)                               0.05       0.44 r
  R4_reg_32_/D (DFFX1_RVT)                           0.00       0.44 r
  data arrival time                                             0.44

  clock main_clk (rise edge)                         1.00       1.00
  clock network delay (propagated)                   0.00       1.00
  clock reconvergence pessimism                      0.00       1.00
  R4_reg_32_/CLK (DFFX1_RVT)                         0.00       1.00 r
  library setup time                                -0.02       0.98
  data required time                                            0.98
  -----------------------------------------------------------------------
  data required time                                            0.98
  data arrival time                                            -0.44
  -----------------------------------------------------------------------
  slack (MET)                                                   0.54
```

Figure 6.20: Report timing [1].

Analyze the report(Startpoint/Endpoint/Launch Network Delay/Data Path/Capture Network delay/library setup time/slack calculation.

```
icc2_shell> man report_timing
2.  Synopsys Commands                                    Command Reference
                              report_timing

NAME
       report_timing
              Displays timing information about a design.

SYNTAX
       status report_timing
              [-to to_list]
                  | -rise_to rise_to_list
                  | -fall_to fall_to_list]
              [-from from_list
                  | -rise_from rise_from_list
                  | -fall_from fall_from_list]
```

Figure 6.21: Using man command [1].

10. Do ‘report_qor , report_global_timing , report_clock_qor’, check the info reported and note down

    1. Number of sinks for the main_clk
    2. Critical Path Slack
    3. Nets with Violations, setup/hold timing violations summary

```
report_qor
****************************************
Report : qor
Design : MY_DESIGN
Version: P-2019.03-SP2
Date   : Thu Sep 14 14:35:32 2023
****************************************
Information: Timer using 'SI, Timing Window Analysis, CRPR'. (TIM-050)


Scenario           'func::sspg'
Timing Path Group  'main_clk'
---------------------------------------
Levels of Logic:                    8
Critical Path Length:            0.43
Critical Path Slack:             0.54
Critical Path Clk Period:        1.00
Total Negative Slack:            0.00
No. of Violating Paths:             0
---------------------------------------
```

Figure 6.22: Using report_qor command [1].

```
icc2_shell> report_global_timing
Information: Timer using 'SI, Timing Window Analysis, CRPR'. (TIM-050)
****************************************
Report : global timing
        -format { narrow }
Design : MY_DESIGN
Version: P-2019.03-SP2
Date   : Thu Sep 14 14:36:31 2023
****************************************

No setup violations found.


No hold violations found.
```

Figure 6.23: Using report_global_timing command [1].

```
icc2_shell> report_clock_qor
 Info: Initializing timer in CLOCK_SYN_REPORT_MODE
****************************************
Report : clock qor
        -type summary
Design : MY_DESIGN
Version: P-2019.03-SP2
Date   : Thu Sep 14 14:37:03 2023
****************************************

Attributes
==========
M Master Clock
G Generated Clock
& Internal Generated Clock
U User Defined Skew Group
D Default Skew Group
* Generated Clock Balanced Separately

============================================
==== Summary Reporting for Corner sspg ====
============================================

================================================= Summary Table for Corner sspg
 =================================================
Clock /                              Attrs     Sinks Levels    Clock     Clock
    Clock       Max    Global  Trans DRC  Cap DRC
Skew Group                                                   Repeater  Repeater
  Stdcell   Latency      Skew     Count     Count
                                                               Count      Area
     Area
-------------------------------------------------------------------------------
-------------------------------------------------
### Mode: func, Scenario: func::sspg
main_clk                               M,D       131      1        0      0.00
     0.00      0.01      0.01         0         0
-------------------------------------------------------------------------------
-------------------------------------------------
All Clocks                                       131      1        0      0.00
     0.00      0.01      0.01         0         0
```

Figure 6.24: using report_clock_qor command [1].

6.6 References :

[1] Synopsys: Synopsys | EDA Tools, Semiconductor IP and Application Security Solutions

Experiment No. 7 - Routing, and Sign off (Timing, Formal verification)

7.1 Objectives

- Load design with placed, legalized standard cells & synthesized/routed clock tree.
- Perform detailed signal routing for the design and final modifications to the functionality of the design.
- Check real timing, actual net delays, congestion maps and trace layers/vias connections for some nets.

7.2 Equipment Required

- Synopses tool - IC Compiler II
- PuTTY
- Xming

7.3 Pre-Lab

1. Read the experiment to learn how to use the tool.

7.4 Introduction

7.4.1 Routing

Routing is a fundamental process that involves connecting electronic components like transistors, gates, and flip-flops on semiconductor chips to create integrated circuits. This crucial step in the physical design of integrated circuits follows logic synthesis and placement and plays a pivotal role in determining the circuit's functionality. The primary objective of routing is to establish electrical connections while adhering to strict design constraints, such as signal timing, power consumption, and chip area.

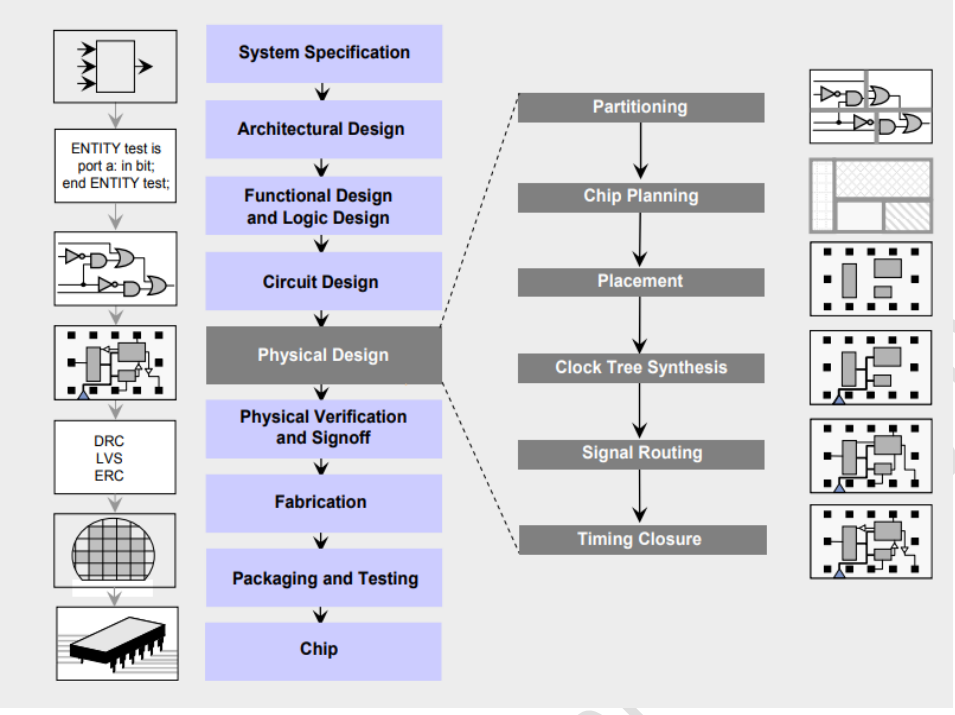

Figure 7.1: Vlsi data flow [1].

There are two main types of routing in VLSI: global routing and detailed routing. Global routing focuses on connecting larger blocks or macros at a high level, determining their approximate locations within the chip. On the other hand, detailed routing involves connecting individual transistors, gates, and other components to create the precise paths required for circuit functionality.

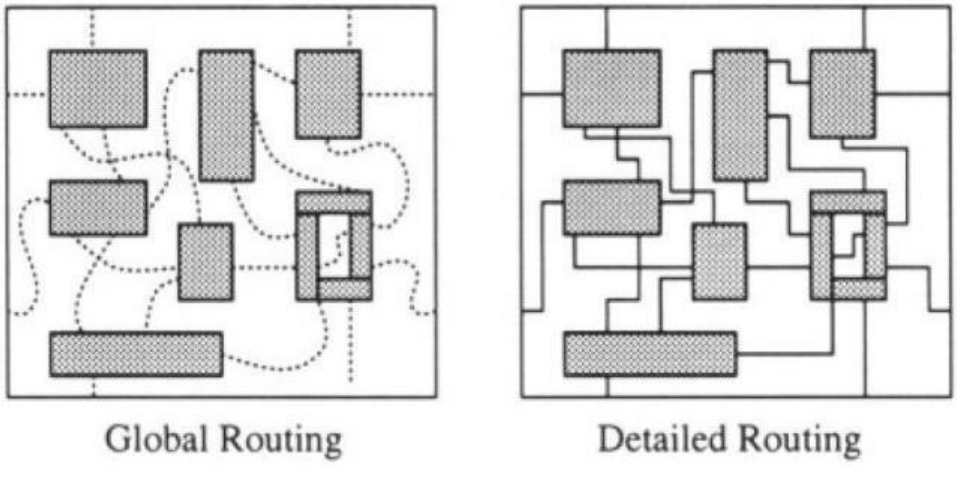

Figure 7.2: Global and Detailed routing [2].

The main difference between global and detailed routing that the global routing is a higher-level step that focuses on determining the routing paths, while detailed routing is a lower-level step that focuses on physically laying out the interconnections on the chip.

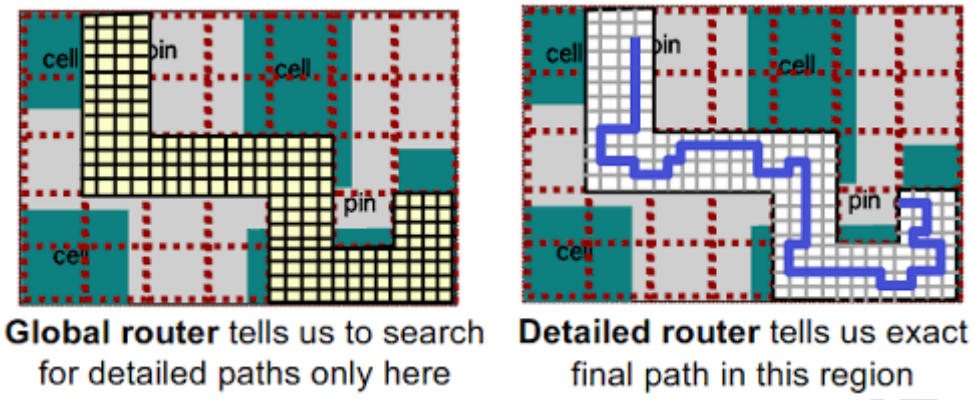

Figure 7.3: Global and Detailed routing [3].

Routing in VLSI relies on various routing algorithms. These algorithms determine how connections are established between components. Some commonly used routing algorithms include maze routing, Steiner tree routing, Lee's algorithm, and A* algorithm. Each of these algorithms has specific advantages and is chosen based on the design's requirements and constraints.

However, routing in VLSI also presents several challenges. Congestion is a common issue when multiple wires need to pass through limited routing resources like metal layers, vias, or routing tracks. Wirelength minimization is crucial to reduce signal delay and power consumption. Noise and crosstalk must be managed to ensure signal integrity, and achieving timing closure is essential to meet strict timing constraints.

7.5 Procedure

7.5.1 Environment Setup:

- **Go to your EXP directory by going to your user directory (cd ~), then do: (mkdir -p ./ST_VLSI/EXP7)**

```
[hanan@synopsys ~]$ cd
[hanan@synopsys ~]$ mkdir -p ./ST_VLSI/EXP7
```

Figure 7.4: Make directory [4].

- **Go to EXP7 directory (cd ./ST_VLSI/EXP7)**

```
[hanan@synopsys ~]$ cd ./ST_VLSI/EXP7
```

Figure 7.5: Open the directory [4].

- **Copy the tool environment to your user directory(change the word in red color):**

cp -R /home/iccTA/all_labs/icc2_EXP7_env/* .

```
[hanan@synopsys EXP7]$ cp -R /home/iccTA/all_labs/icc2_EXP7_env/*     .
```

Figure 7.6: Copy the tool environment [4].

- **To view the copied files: ls**

```
[hanan@synopsys EXP7]$ ls
inputs  Makefile  rm_icc2_pnr_scripts  rm_setup
```

Figure 7.7: Show files [4].

- **Link your synthesized netlist to the inputs directory:**

ln -s /home/<username>/ST_VLSI/EXP3/results/{DESIGN}.mapped.v inputs/.

```
[hanan@synopsys EXP7]$ ln -s /home/hanan/ST_VLSI/EXP3/results/MY_DESIGN.mapped.v
 inputs/.
[hanan@synopsys EXP7]$ ls
inputs  Makefile  rm_icc2_pnr_scripts  rm_setup
```

Figure 7.8: Link synthesized netlist to the input's directory from exp 3 [4].

- **Copy the previous experiments database**:

cp -RPi /home/< username >/ST_VLSI/EXP6/<DESIGN_NAME>.nlib .

```
[hanan@synopsys EXP7]$ cp -RPi /home/hanan/ST_VLSI/EXP6/MY_DESIGN.nlib .
[hanan@synopsys EXP7]$ ls
inputs  Makefile  MY_DESIGN.nlib  rm_icc2_pnr_scripts  rm_setup
```

Figure 7.9: Copy the previous experiments database from exp 6 [4].

7.5.2 Run the design script: shell> make route_opt: (it will take time)

```
[hanan@synopsys EXP7]$ make route_opt
```

Figure 7.10: Run the design script [4].

i. **Check the log for error messages, use " \less logs_icc2/ clock*.log" and search using "/^Error" (^: Shift+6) ,then press Enter - If you see any error messages, check with TA (ignore PLACE-007).To exit from file press "q" character.**

```
total 1224
drwxrwxr-x 2 hanan hanan   4096 Sep 12 00:29 ./
drwxr-xr-x 9 hanan hanan   4096 Sep 12 01:47 ../
-rw-rw-r-- 1 hanan hanan 285052 Sep 12 00:29 route_auto.log
-rw-rw-r-- 1 hanan hanan 957967 Sep 12 01:47 route_opt.log
/^Error
```

Figure 7.11: Check the log for error messages [4].

7.5.3 Check the design interactively after it's finished:

1. **shell> icc2_shell [enter] (it will take a long time)**
2. **icc2_shell> open_lib <DESIGN_NAME>.nlib [enter]**

```
icc2_shell> open_lib MY_DESIGN.nlib
Information: Loading library file '/home/hanan/ST_VLSI/EXP7/MY_DESIGN.nlib' (FIL
E-007)
Information: Loading library file '/usr/synopsys/icc2_ta/ICC2_BLI_2019.03-SP4/re
f/CLIBs/saed32_rvt.ndm' (FILE-007)
{MY_DESIGN.nlib}
icc2_shell>
```

Figure 7.12: Open design library [4].

3. **icc2_shell> list_blocks [enter]**

```
icc2_shell> list_blocks
Lib MY_DESIGN.nlib /home/hanan/ST_VLSI/EXP7/icc2_EXP7_env/MY_DESIGN.nlib tech current
  ->   0 MY_DESIGN.design Sep-10-17:07
  -    0 MY_DESIGN/clock_opt_cts.design Sep-11-18:55
  -    0 MY_DESIGN/clock_opt_opto.design Sep-11-18:57
  -    0 MY_DESIGN/init_design.design Sep-10-17:54
  -    0 MY_DESIGN/place_opt.design Sep-11-16:29
  -    0 MY_DESIGN/place_opt_two_pass_placement.design Sep-10-21:42
  -    0 MY_DESIGN/route_auto.design Sep-12-00:28
  -    0 MY_DESIGN/route_opt.design Sep-12-01:47
8
icc2_shell>
```

Figure 7.13: Block list [4].

4. **icc2_shell> open_block <DESIGN_NAME>/route_opt.design [enter]**

```
icc2_shell> open_block MY_DESIGN/route_opt.design
Information: User units loaded from library 'saed32_rvt|saed32_rvt_pg' (LNK-040)
Opening block 'MY_DESIGN.nlib:MY_DESIGN/route_opt.design' in edit mode
{MY_DESIGN.nlib:MY_DESIGN/route_opt.design}
```

f

Figure 7.14: Open block [4].

5. **icc2_shell> start_gui**

```
icc2_shell> start_gui
```

Figure 7.15: Gui start [4].

6. **Press F to reset the gui view**

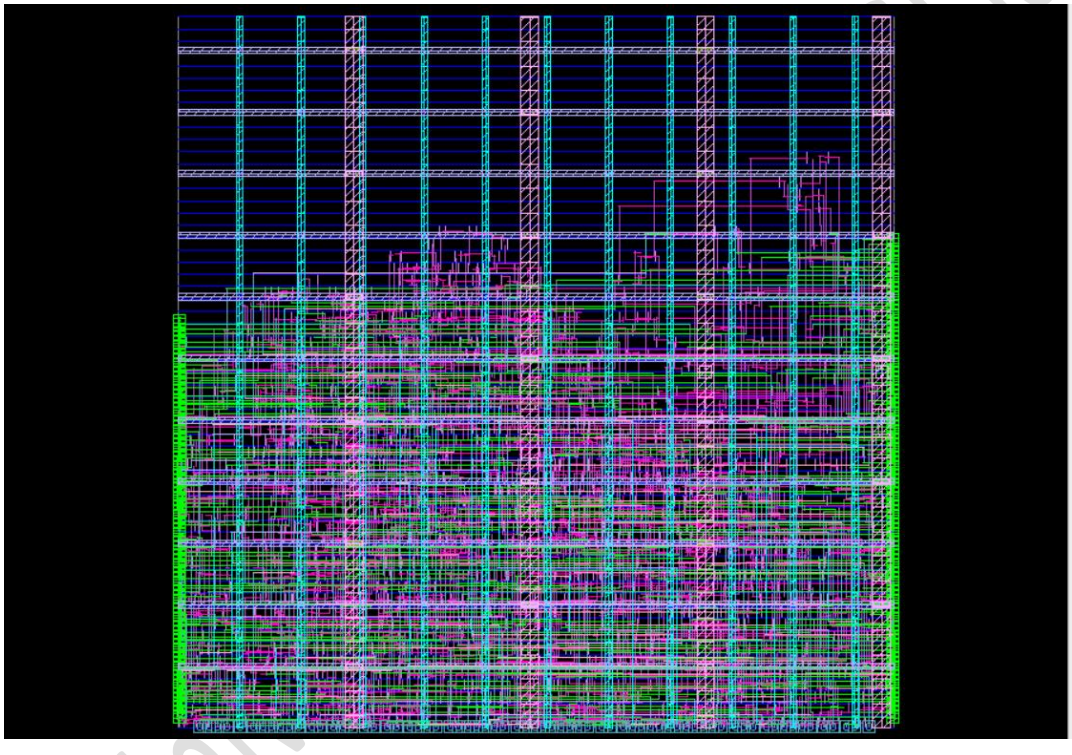

Figure 7.16: Our design [4].

7. **Disable "Route->Net Type->Power/Ground /Clock" in the View Settings/Objects Tab**

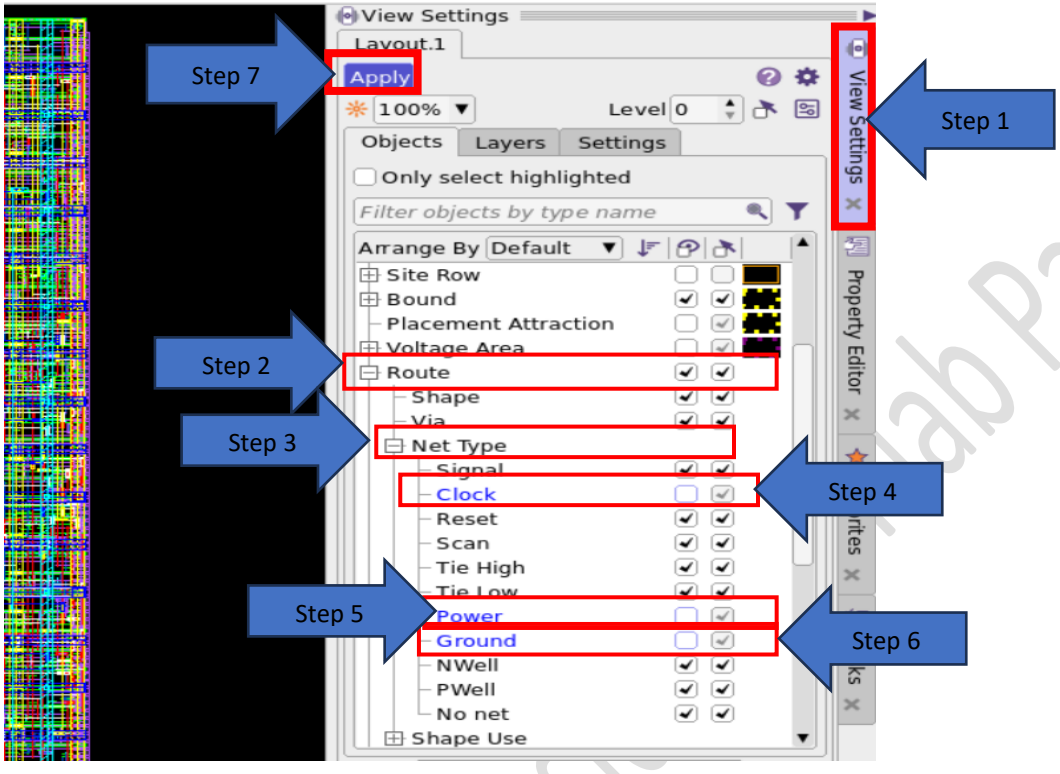

Figure 7.17: Disable the power, clock & ground [4].

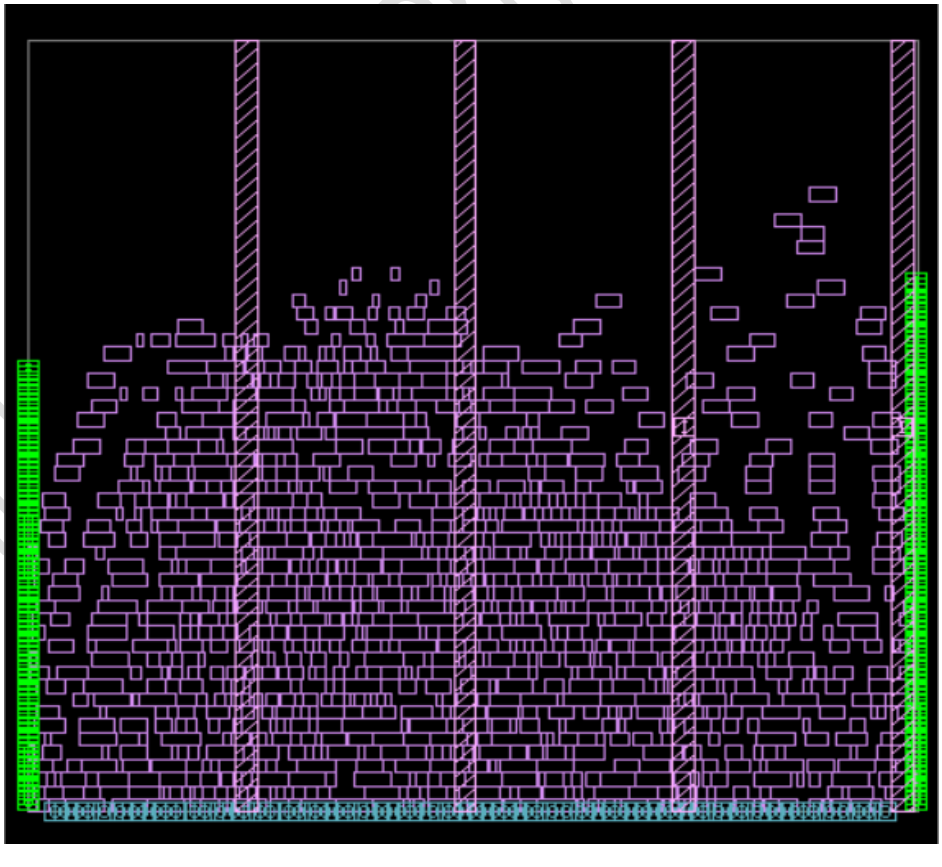

Figure 7.18: Design layout [4].

8. **Enable pins from view settings, and disable Pin->Pin Type->Power/Ground**

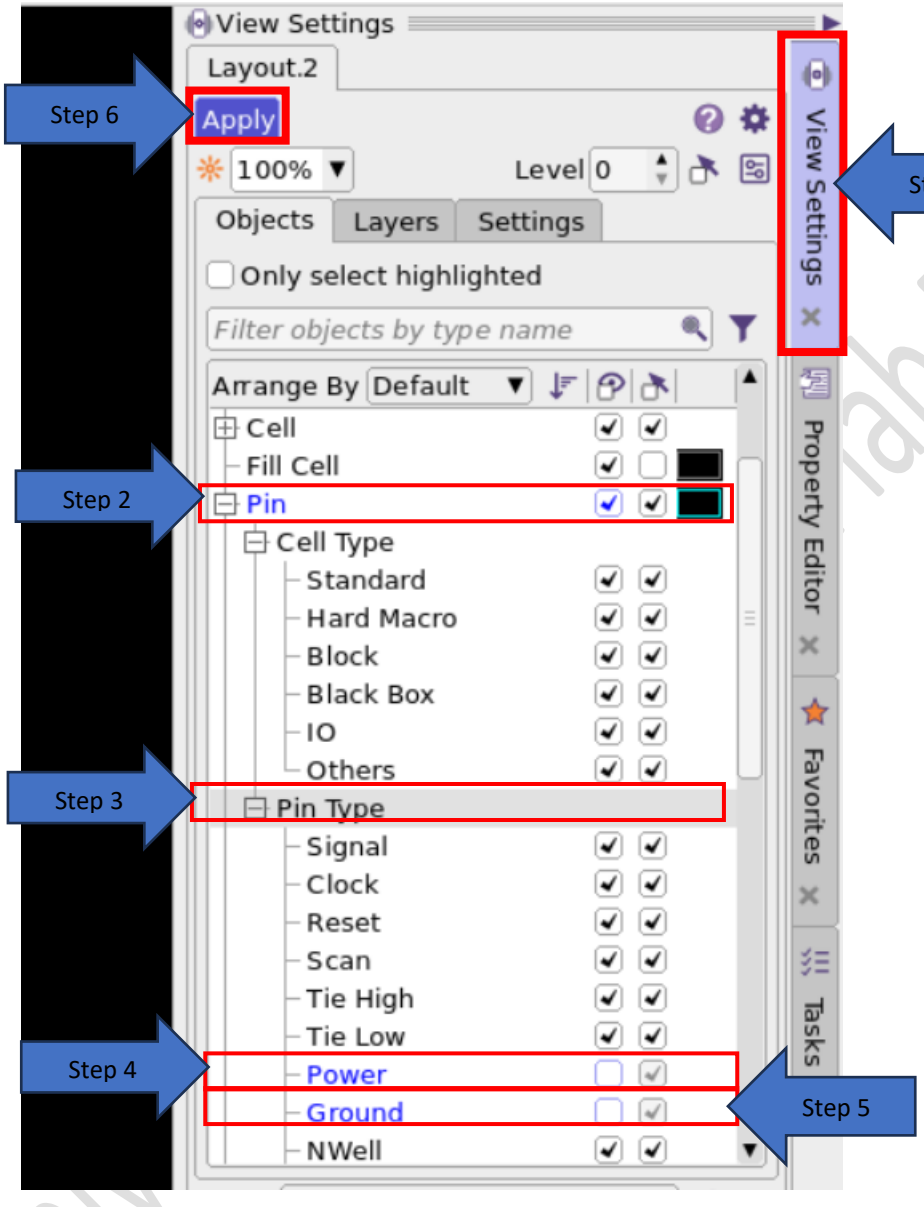

Figure 7.19: disanable pwer pin & ground pin [4].

9. **Check the fanout of any output pin (denoted by “o” in the layout) by first enabling net connection view and then selecting any output pin:**

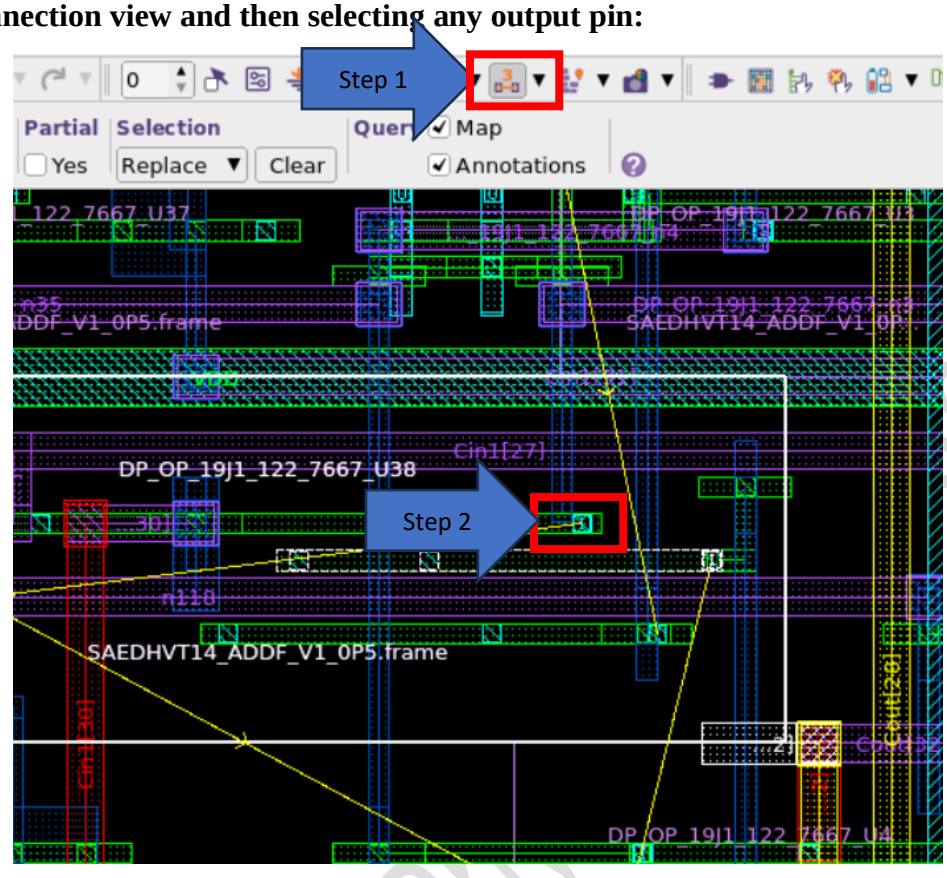

Figure 7.20: Check the fanout of the output pin [4].

10. Trace the net routing after selecting any output pin, right click-> Select Related Objects -> Nets and then note which layer/via it’s moving through .

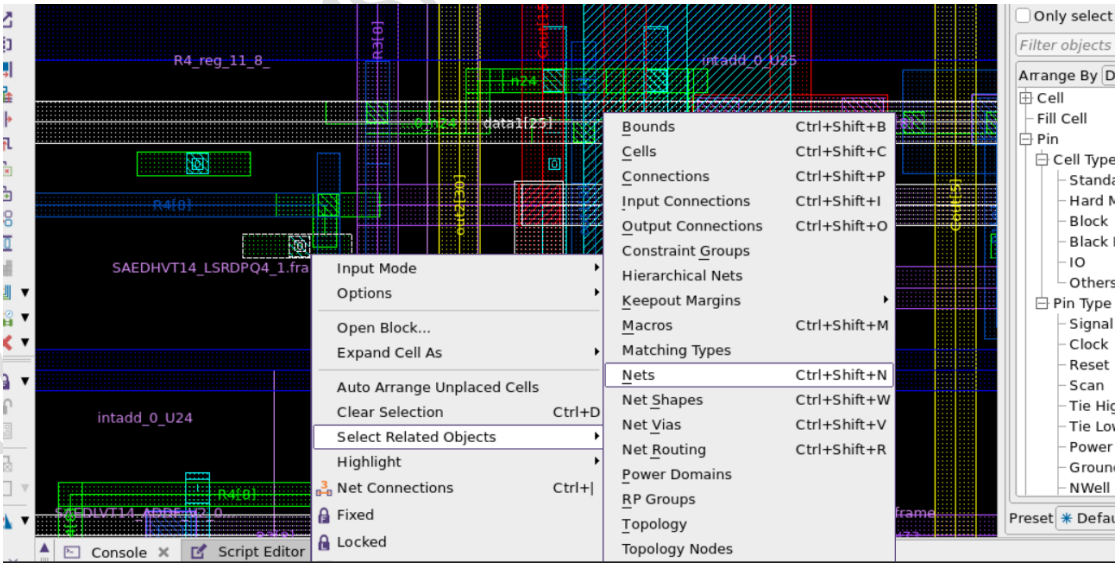

Figure 7.21: Trace the net routing [4].

11. You can know the layer/via type by looking at the bottom-left corner while hovering over the shape (Look at Layer row, for example Layer M3, and Via M1-M2 in the following pictures)

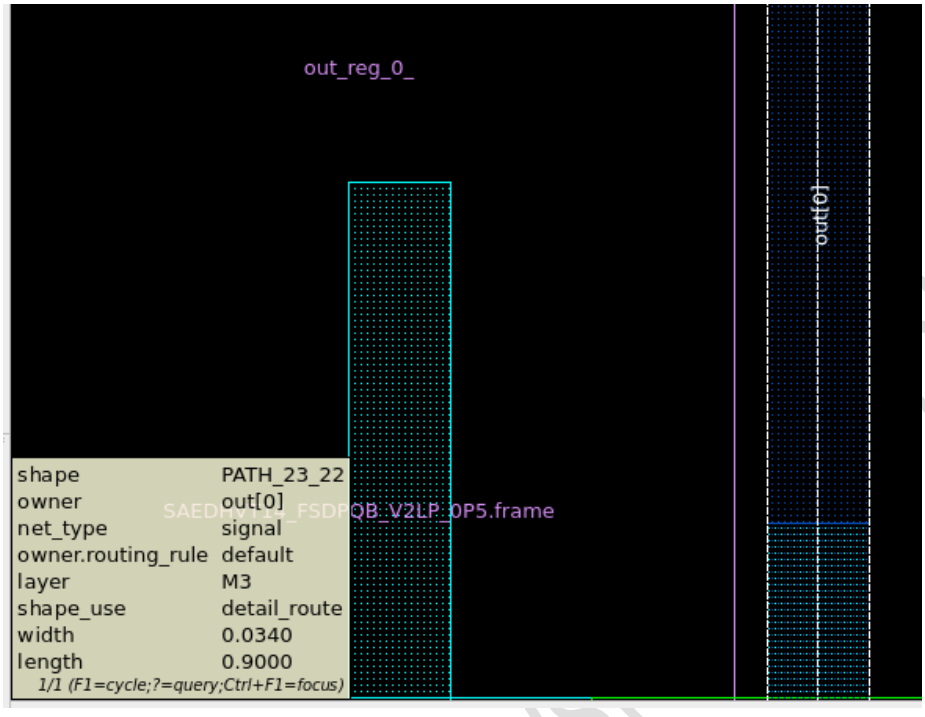

Figure 7.22: Layer/via type [4].

12. Check the congestion in the design by opening congestion map and click “reload” in the gui:

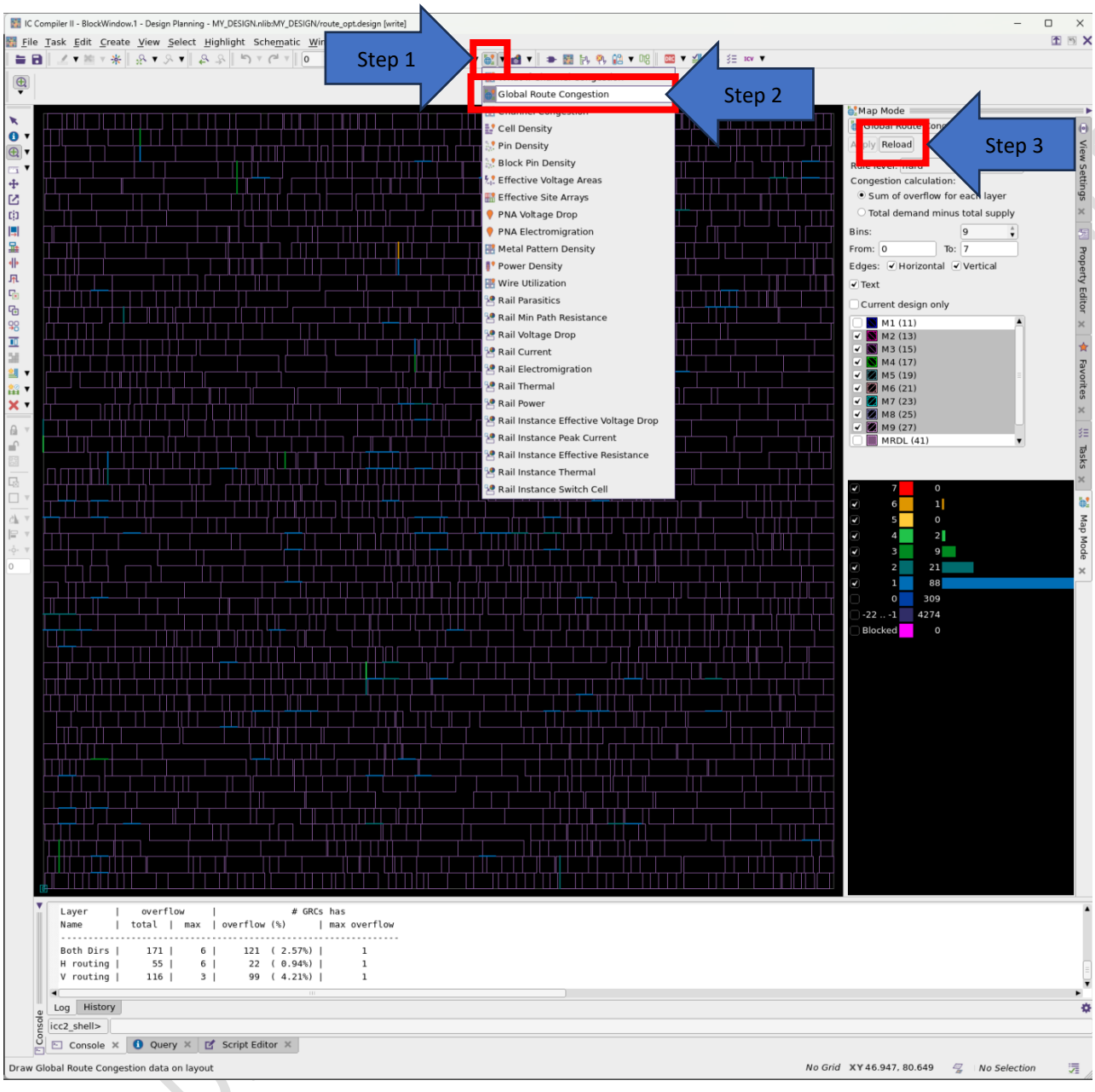

Figure 7.23: Global route [4].

13. The edges indicate the demand vs supply of routing resources & availability to routing. Redder color indicates more demand than is available, to see this information, hover over any of the colored lines in the layout to see the information bubble in the bottom left corner.

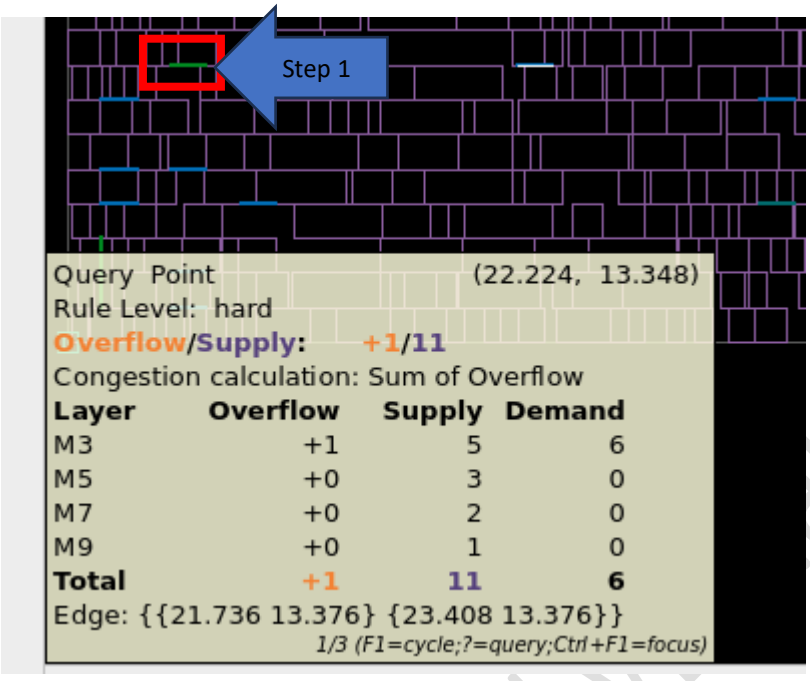

Figure 7.24: Global route [4].

14. To confirm you have no DRC errors, open the Error Browser -> double click on zroute.err

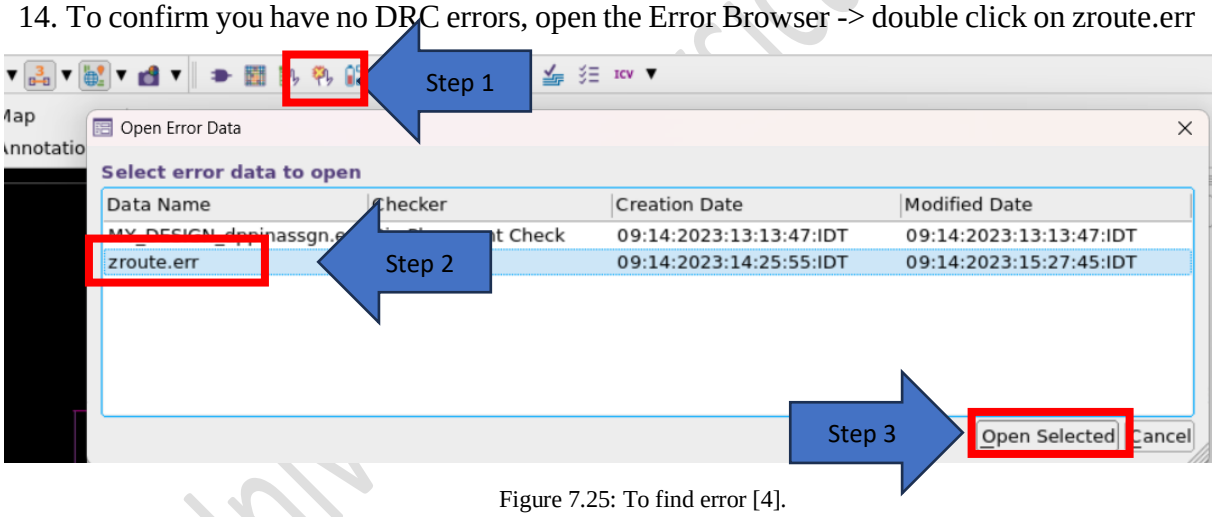

Figure 7.25: To find error [4].

15. If any errors are present, analyze them briefly and ways they can be fixed. If none are reported, your design has routed with no issues!

16. Execute “report_timing” in the shell, to know more about this command, do “man report_timing”.

```
icc2_shell> icc2_shell> report_timing
```

Figure 7.26: Run report timing command [4].

```
Information: Timer using 'PrimeTime Delay Calculation, SI, Timing Window Analysi
s, AWP, CRPR'. (TIM-050)

  Startpoint: R3_reg_21_ (rising edge-triggered flip-flop clocked by main_clk)
  Endpoint: R4_reg_32_ (rising edge-triggered flip-flop clocked by main_clk)
  Mode: func
  Corner: sspg
  Scenario: func::sspg
  Path Group: main_clk
  Path Type: max

  Point                                                  Incr       Path
  -----------------------------------------------------------------------
  clock main_clk (rise edge)                             0.00       0.00
  clock network delay (propagated)                       0.01       0.01

  R3_reg_21_/CLK (DFFX1_RVT)                             0.00       0.01 r
  R3_reg_21_/Q (DFFX1_RVT)                               0.09       0.10 f
  U542/Y (NAND2X0_RVT)                                   0.04       0.14 r
  SGI22_65/Y (OA221X1_RVT)                               0.04       0.18 r
  U256/Y (OA21X1_RVT)                                    0.03       0.22 r
  U154/Y (OA21X1_RVT)                                    0.03       0.25 r
  U153/Y (INVX0_RVT)                                     0.04       0.29 f
  U172/Y (AOI21X1_RVT)                                   0.05       0.35 r
  U598/Y (OA21X1_RVT)                                    0.04       0.38 r
  U602/Y (XNOR2X1_RVT)                                   0.05       0.43 r
  R4_reg_32_/D (DFFX1_RVT)                               0.00       0.43 r
  data arrival time                                                 0.43

  clock main_clk (rise edge)                             1.00       1.00
  clock network delay (propagated)                       0.00       1.00
  clock reconvergence pessimism                          0.00       1.00
  R4_reg_32_/CLK (DFFX1_RVT)                             0.00       1.00 r
  library setup time                                    -0.02       0.98
  data required time                                                0.98
  -----------------------------------------------------------------------
  data required time                                                0.98
  data arrival time                                                -0.43
  -----------------------------------------------------------------------
  slack (MET)                                                       0.55


1
```

Figure 7.27: Report timing [4].

17. Analyze the report(Startpoint/Endpoint/Launch Network Delay/Data Path/Capture Network delay/library setup time/slack calculation. - Note that net delays are now not estimated, since the clock AND signal nets are routed, thus the calculations are based on real RC Extraction values of both launch and capture paths."report_qor , report_global_timing , report_clock_qor"

```
report_qor
****************************************
Report : qor
Design : MY_DESIGN
Version: P-2019.03-SP2
Date   : Thu Sep 14 14:35:32 2023
****************************************
Information: Timer using 'SI, Timing Window Analysis, CRPR'. (TIM-050)


Scenario           'func::sspg'
Timing Path Group  'main_clk'
-----------------------------------------
Levels of Logic:                    8
Critical Path Length:            0.43
Critical Path Slack:             0.54
Critical Path Clk Period:        1.00
Total Negative Slack:            0.00
No. of Violating Paths:             0
-----------------------------------------
```

Figure 7.28: Using report_qor command [4].

```
icc2_shell> report_global_timing
Information: Timer using 'SI, Timing Window Analysis, CRPR'. (TIM-050)
****************************************
Report : global timing
         -format { narrow }
Design : MY_DESIGN
Version: P-2019.03-SP2
Date   : Thu Sep 14 14:36:31 2023
****************************************

No setup violations found.


No hold violations found.
```

Figure 7.29: Using report_global_timing command [4].

```
icc2_shell> report_clock_qor
 Info: Initializing timer in CLOCK_SYN_REPORT_MODE
****************************************
Report : clock qor
         -type summary
Design : MY_DESIGN
Version: P-2019.03-SP2
Date   : Thu Sep 14 14:37:03 2023
****************************************

Attributes
============
M Master Clock
G Generated Clock
& Internal Generated Clock
U User Defined Skew Group
D Default Skew Group
* Generated Clock Balanced Separately

================================================
==== Summary Reporting for Corner sspg ====
================================================

================================================== Summary Table for Corner sspg
 ==================================================
Clock /                                  Attrs      Sinks Levels     Clock      Clock
    Clock        Max     Global   Trans DRC   Cap DRC
Skew Group                                                        Repeater   Repeater
  Stdcell    Latency       Skew      Count      Count                  Count       Area
     Area
-------------------------------------------------------------------------------------
-------------------------------------------------
### Mode: func, Scenario: func::sspg
main_clk                                   M,D        131       1         0       0.00
     0.00       0.01       0.01           0          0
-------------------------------------------------------------------------------------
-------------------------------------------------
All Clocks                                             131       1         0       0.00
     0.00       0.01       0.01           0          0
```

Figure 7.30: Using report_clock_qor command [4].

7.6 References :

[1] Introduction to Digital VLSI Design | by Yu | LazyYu's Blog | Medium

[2] PPT - Parallel Algorithms for VLSI Routing PowerPoint Presentation, free download - ID:1691807 (slideserve.com)

[3] https://zhuanlan.zhihu.com/p/164919161

[4] Synopsys: Synopsys | EDA Tools, Semiconductor IP and Application Security Solutions

Experiment No. 8- Physical Verifications

8.1 Objectives

- Load design with placed, synthesized/routed clock tree and signal routed design.
- Perform final steps of closing the chip.
- Check the lvs, DRC, placement and timing status to signoff the chip.

8.2 Equipment Required

- Synopses tool - IC Compiler II
- PuTTY
- Xming

8.3 Pre-Lab

1. Read the experiment to learn how to use the tool.

8.4 Introduction

8.4.1 Routing

Physical verification in the context of semiconductor design and VLSI (Very Large Scale Integration) refers to the process of checking and ensuring that the physical layout of an integrated circuit (IC) or chip is compliant with manufacturing rules and specifications. The goal of physical verification is to identify and correct any potential issues in the layout that could lead to manufacturing defects or performance problems.

Here are some key aspects and steps involved in physical verification:

1. **Design Rules Checking (DRC):** DRC is the initial step in physical verification. It involves checking the design layout against a set of predefined design rules provided by the semiconductor foundry or the design team. These rules define the minimum and maximum dimensions, spacing between components, metal layer overlaps, and other geometric constraints that must be met for successful manufacturing. DRC tools flag violations of these rules.

2. **Layout vs. Schematic (LVS) Checks:** LVS is used to verify that the physical layout matches the intended schematic or logical design. It ensures that all connections are correctly made, and there are no missing or incorrect connections. LVS tools compare the layout netlist with the schematic netlist.

3. **Electrical Rule Checking (ERC):** ERC focuses on electrical connectivity and checks for issues such as shorts, open circuits, and other electrical violations that could impact circuit functionality. ERC tools analyze the physical layout to identify these problems.

4. **Antenna Effect Checks:** Antenna effect refers to the buildup of charge on certain layers of a transistor during manufacturing processes like ion implantation. Antenna effect checks ensure that no excessive charge buildup occurs, as it can cause damage to the device. These checks are critical for preventing long-term reliability issues.

5. **Dummy Fill Insertion:** In advanced semiconductor processes, it's common to use dummy fill to ensure that all areas of the chip have a consistent density of material. This helps prevent manufacturing issues like chemical-mechanical polishing (CMP) non-uniformity. Dummy fill insertion tools automatically add extra shapes to the layout as needed.

6. **Metal Density Checks:** Metal density checks ensure that there is a balanced distribution of metal in the layout. Excessive metal density in certain areas can lead to manufacturing problems, while low metal density can affect circuit performance. Metal density checks aim to maintain a uniform distribution.

7. **Electromigration and IR Drop Analysis:** These analyses assess the chip's ability to handle current and voltage requirements without causing issues like electromigration (movement of metal atoms due to high current density) or excessive voltage drops. These checks ensure long-term reliability.

8. **Manufacturability Analysis:** Some physical verification processes assess the manufacturability of the chip, considering factors like yield, lithography constraints, and process variations. Design changes may be recommended to improve manufacturing yield.

9. Wafer Tape-Out: Once all physical verification checks are passed and the layout is deemed manufacturable, the design is ready for tape-out. This involves generating the final data files for mask production, which are used to fabricate the actual semiconductor wafers.

Physical verification is a crucial step in the semiconductor design and manufacturing process, as it helps ensure that chips are free of defects and meet their performance specifications. It is typically performed using specialized Electronic Design Automation (EDA) tools and is an integral part of the overall VLSI design flow

8.5 Procedure

8.5.1 Environment Setup:

- **Go to your EXP directory by going to your user directory (cd ~), then do: (mkdir -p ./ST_VLSI/EXP8)**

```
[hanan@synopsys ~]$ cd
[hanan@synopsys ~]$ mkdir -p ./ST_VLSI/EXP8
```

Figure 8.1: Make directory [1].

- **Go to EXP8 directory (cd ./ST_VLSI/EXP8)**

```
[hanan@synopsys ~]$ cd ./ST_VLSI/EXP8
```

Figure 8.2: Open the directory [1].

- **Copy the tool environment to your user directory(change the word in red color):**

cp -R /home/iccTA/all_labs/icc2_EXP8_env/* .

```
[hanan@synopsys EXP8]$ cp -R /home/iccTA/all_labs/icc2_EXP8_env/*
```

Figure 8.3: Copy the tool environment [1].

- **To view the copied files: ls**

```
[hanan@synopsys EXP7]$ ls
inputs  Makefile  rm_icc2_pnr_scripts  rm_setup
```

Figure 8.4: Show files [1].

- **Link your synthesized netlist to the input's directory:**

ln -s /home/<username>/ST_VLSI/EXP3/results/{DESIGN}.mapped.v inputs/.

```
[hanan@synopsys EXP8]$ ls
inputs  Makefile  rm_icc2_pnr_scripts  rm_setup
```

Figure 8.5: Link synthesized netlist to the input's directory from exp 3 [1].

- **Copy the previous experiments database**:

cp -RPi /home/< username >/ST_VLSI/EXP7/<DESIGN_NAME>.nlib .

```
[hanan@synopsys EXP8]$ cp -RPi /home/hanan/ST_VLSI/EXP7/MY_DESIGN.nlib .
[hanan@synopsys EXP8]$ ls
inputs  Makefile  MY_DESIGN.nlib  rm_icc2_pnr_scripts  rm_setup
```

Figure 8.6: Copy the previous experiments database from exp 6 [1].

8.5.2 Run the design script: shell> make chip_finish: (it will take time)

```
[hanan@synopsys EXP8]$ make chip_finish
```

Figure 8.7: Run the design script [1].

i. **Check the log for error messages, use " \less logs_icc2/ clock*.log" and search using "/^Error" (^: Shift+6) ,then press Enter - If you see any error messages, check with TA (ignore PLACE-007).To exit from file press "q" character.**

```
total 124
drwxrwxr-x 2 hanan hanan   4096 Sep 14 17:24 ./
drwxrwxr-x 9 hanan hanan   4096 Sep 14 17:24 ../
-rw-rw-r-- 1 hanan hanan 115950 Sep 14 17:24 chip_finish.log
/^Error
```

Figure 8.8: Check the log for error messages [1].

8.5.3 Check the design interactively after it's finished:

1. **shell> icc2_shell [enter] (it will take a long time)**
2. **icc2_shell> open_lib <DESIGN_NAME>.nlib [enter]**

```
icc2_shell> open_lib MY_DESIGN.nlib
Information: Loading library file '/home/hanan/ST_VLSI/EXP8/MY_DESIGN.nlib' (FIL
E-007)
Information: Loading library file '/usr/synopsys/icc2_ta/ICC2_BLI_2019.03-SP4/re
f/CLIBs/saed32_rvt.ndm' (FILE-007)
{MY_DESIGN.nlib}
```

Figure 8.9: Open design library [1].

3. **icc2_shell> list_blocks [enter]**

```
icc2_shell> list_blocks
Lib MY_DESIGN.nlib /home/hanan/ST_VLSI/EXP8/MY_DESIGN.nlib tech current
  ->  0 MY_DESIGN.design Sep-14-13:13
  -   0 MY_DESIGN/chip_finish.design Sep-14-17:24
  -   0 MY_DESIGN/clock_opt_cts.design Sep-14-14:25
  -   0 MY_DESIGN/clock_opt_opto.design Sep-14-14:26
  -   0 MY_DESIGN/init_design.design Sep-14-13:17
  -   0 MY_DESIGN/place_opt.design Sep-14-14:06
  -   0 MY_DESIGN/place_opt_two_pass_placement.design Sep-14-13:57
  -   0 MY_DESIGN/route_auto.design Sep-14-15:27
  -   0 MY_DESIGN/route_opt.design Sep-14-15:27
9
```

Figure 8.10: Block list [1].

4. **icc2_shell> open_block <DESIGN_NAME>/chip_finish.design [enter]**

```
icc2_shell> open_block MY_DESIGN/chip_finish.design
Information: User units loaded from library 'saed32_rvt|saed32_rvt_pg' (LNK-040)
Opening block 'MY_DESIGN.nlib:MY_DESIGN/chip_finish.design' in edit mode
{MY_DESIGN.nlib:MY_DESIGN/chip_finish.design}
```

Figure 8.11: Open block [1].

5. **icc2_shell> start_gui**

```
icc2_shell> start_gui
```

Figure 8.12: gui start [1].

6. **Press F to reset the gui view**

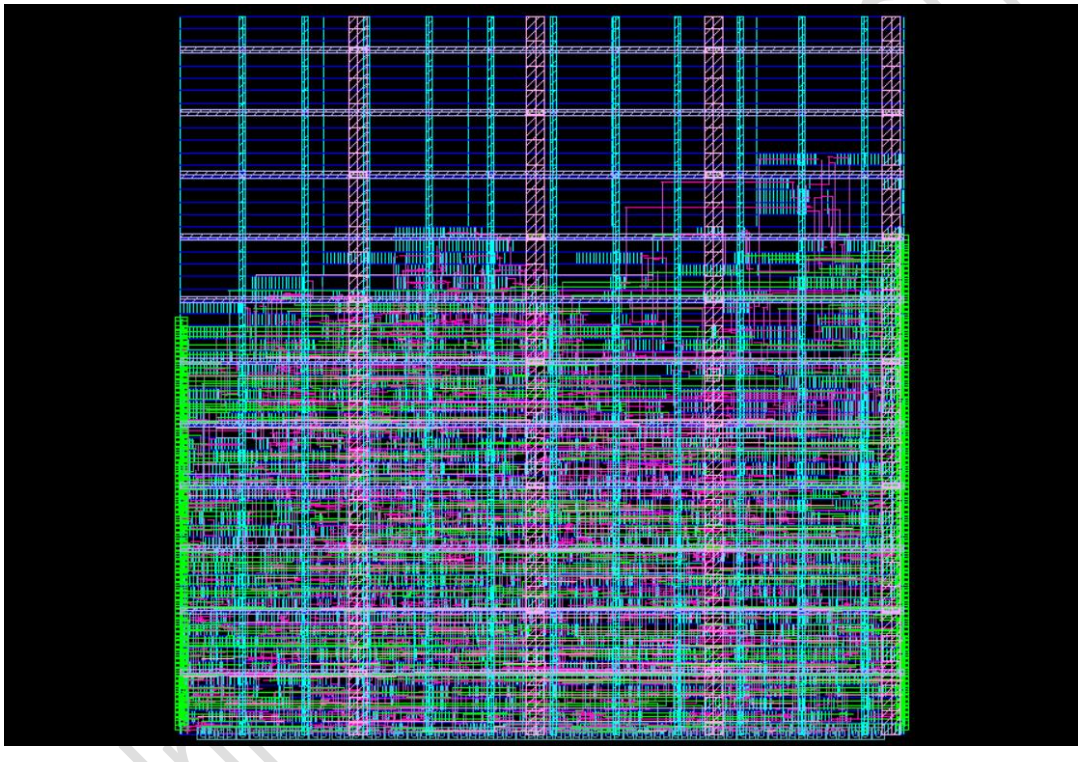

Figure 8.13: Our design [1].

7. **Disable “Route” in the View Settings/Objects Tab**

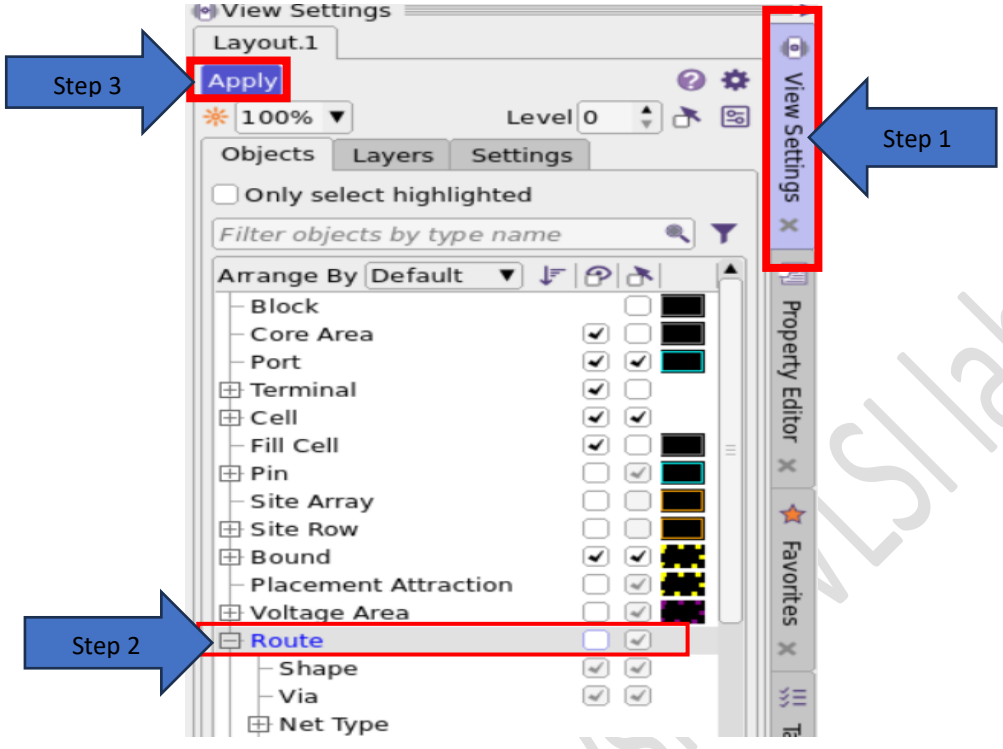

Figure 8.14: Disable the Route [1].

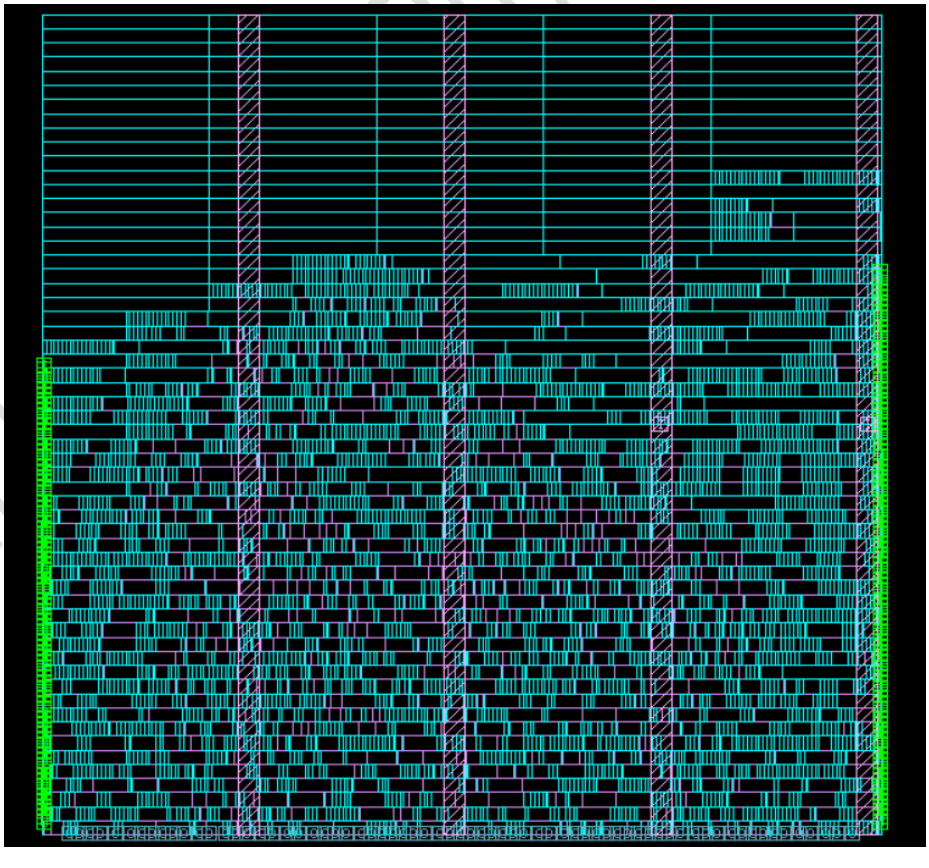

Figure 8.15: Design layout after Disable the Route [1].

8. **Zoom into one of the blue cells, select it, right click->properties, check the ref_phys_view to see the cell type.**

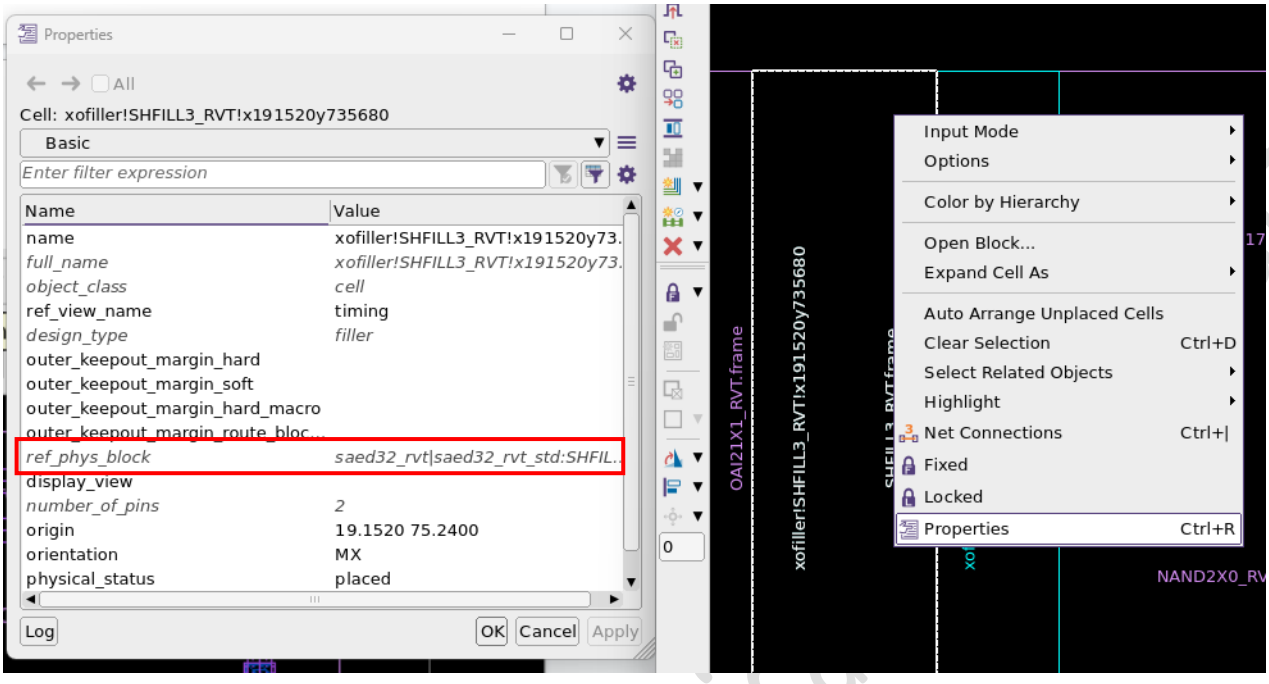

Figure 8.16: Check the ref_phys_view [1].

9. **Enable pins from view settings, and Zoom into one of the blue cells, select it, right click->properties, check the ref_phys_view to see the cell type.**

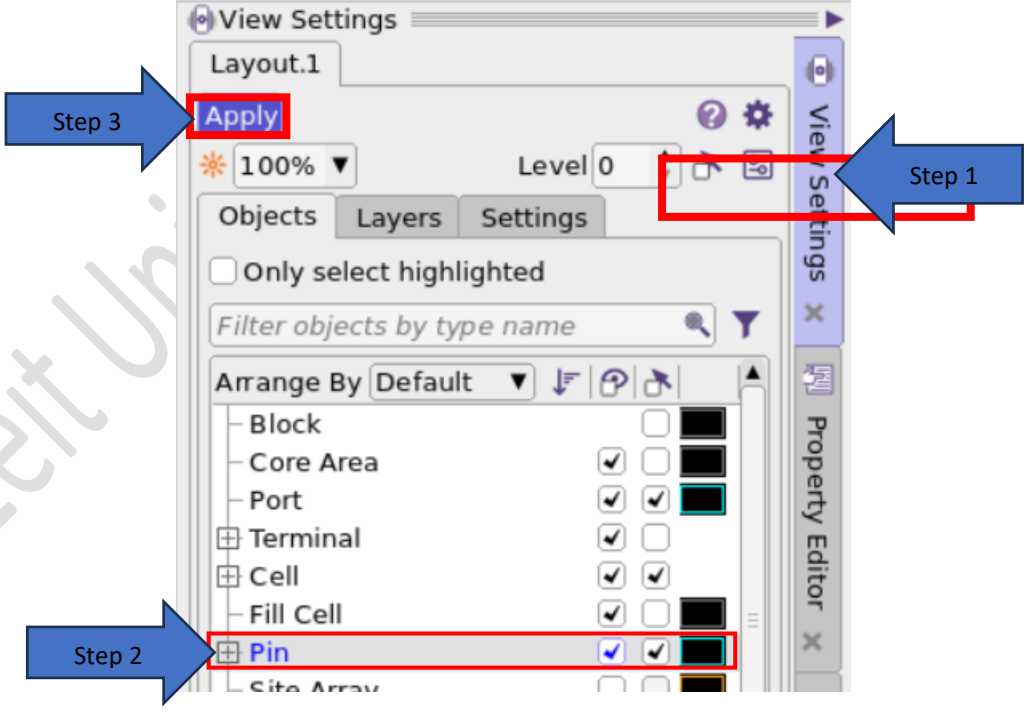

Figure 8.17: Enable pins [1].

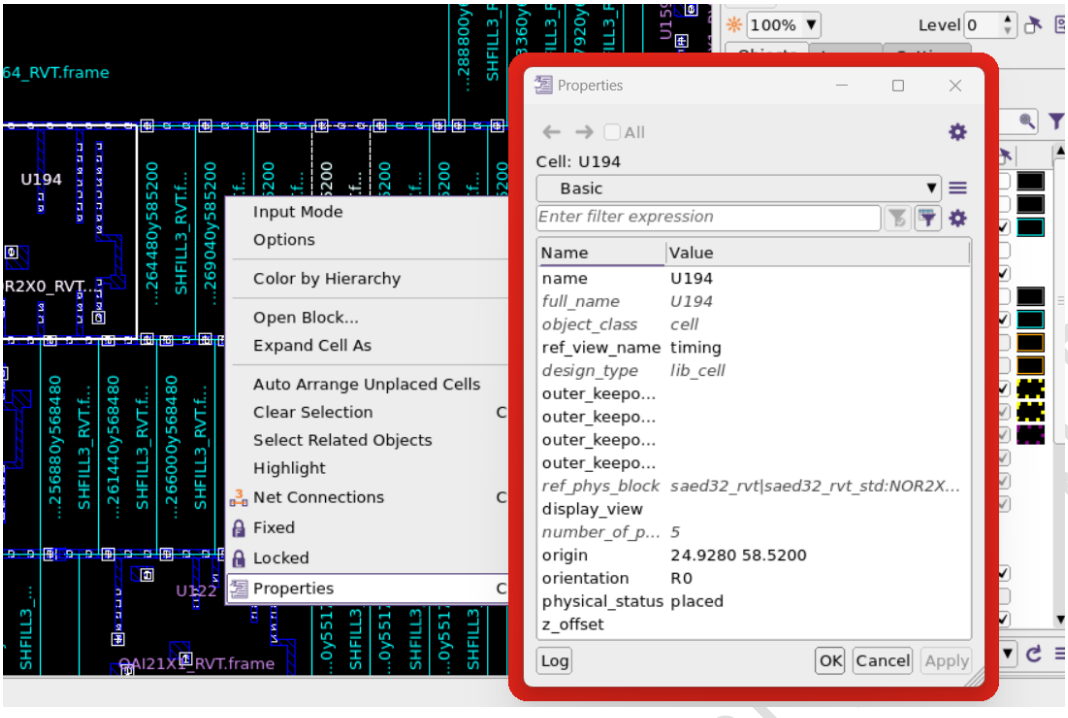

Figure 8.18: Check the ref_phys_view [1].

Confirm there's no empty sites in the design - This is the purpose of chip finish, to fill in the design with decap and filler cells that serve no logic functionality.

10. **After checking the filler cell insertion - verify all cells are legalized by executing "check_legality" command.**

```
icc2_shell> check_legality
```

Figure 8.19: Check_legality [1].

11. **If it returns a success message, move on to the next step. If not, open the error browser to view check_legality.err and locate misplaced cells. Then if a cell is truly not in any site/row Check with TA.**

```
check_legality for block design MY_DESIGN succeeded!

check_legality succeeded.

**************************
```

Figure 8.20: Success message [1].

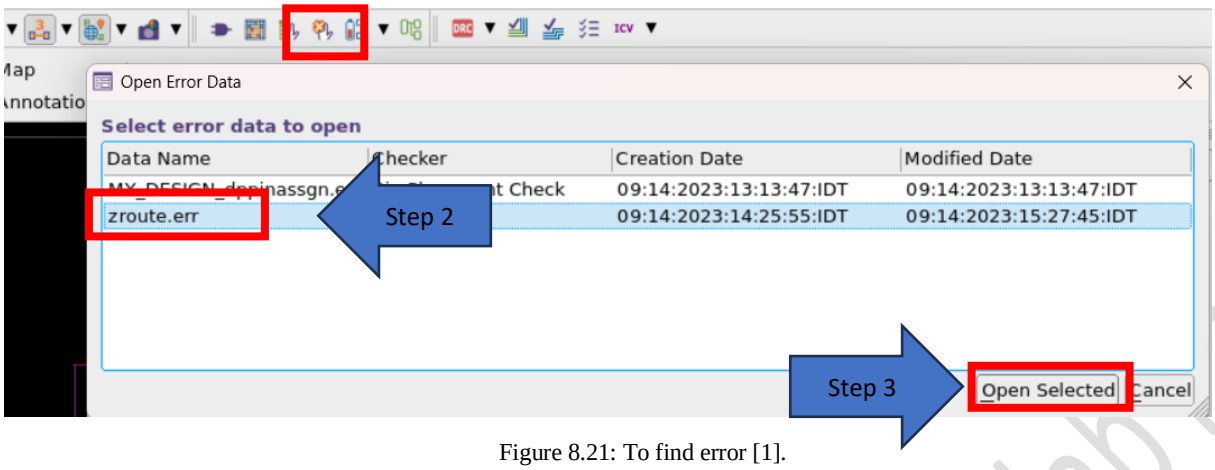

Figure 8.21: To find error [1].

12. Exit the tool by typing "exit" [enter]

13. execute "make icv_in_design"

```
[hanan@synopsys EXP8]$ make icv_in_design
```

Figure 8.22: Make icv_in_design [1].

14. Initiate the tool icc2_shell, open the library as done previously(repeat step 1,2,3 from 8.5.3) and open the block <DESIGN_NAME>/icv_in_design.design and start the gui with "start_gui".

1. **Run icc2_shell**
2. **icc2_shell> open_lib MY_DESIGN.nlib**
3. **icc2_shell> list_blocks**
4. **icc2_shell> open_block MY_DESIGN/icv_in_design.design**
5. **icc2_shell> start_gui**

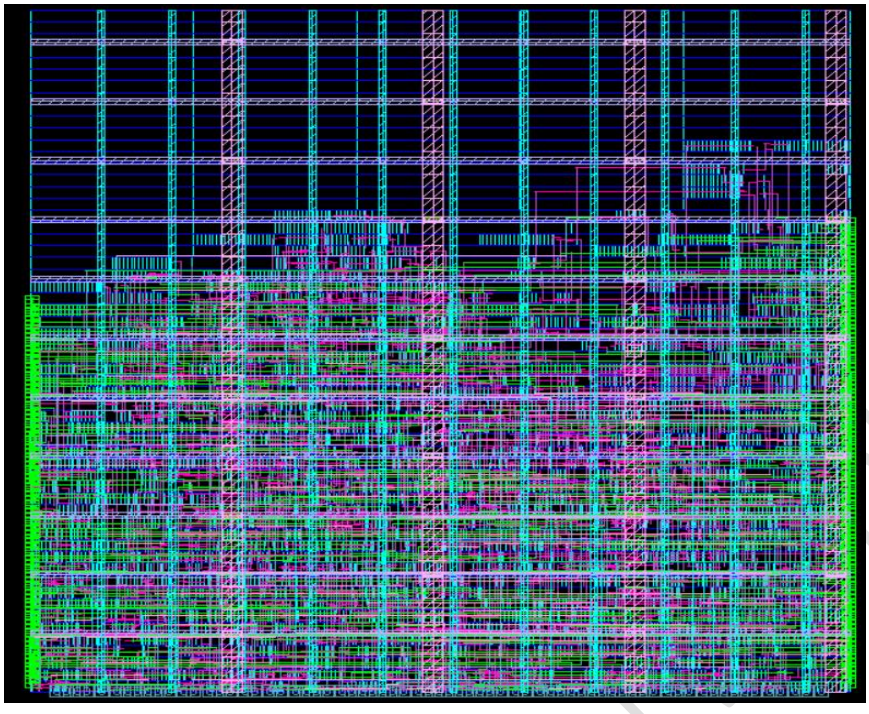

Figure 8.23: The layout [1].

15. **No major layout changes are expected in this step - It is considered the final step in the chip design process where the chip is checked against all DRC rules. To verify this, with the block open, continue to the next steps.**
16. **perform the following commands "check_routes", "report_timing", "report_qor", "check_legality" as do in the previous experiment . Check the manual description of each command to summarize your understanding of each one and what it tells of the design. Note inside report_qor the cell area #.**
17. **After confirming all the above reports giving clean results, move on to the final check.**
18. **Execute "check_lvs -open_reporting detailed", which confirms the layout is conforming to the schematic of the design provided, thus creating a routed, clock-synthesized, and placed schematic-accurate design. To confirm the report is clean, after the command finishes, open the error browser -> double click on <DESIGN_NAME>.check_lvs.err**

```
============================================================
    Maximum number of violations is set to 20
    Abort checking when more than 20 violations are found
    All violations might not be found.
============================================================
Total number of input nets is 1296.
Total number of short violations is 10.
Total number of open locations is 0 (0 open nets).
Total number of floating route violations is 0.

Elapsed =    0:00:01, CPU =    0:00:01
1
icc2_shell>
```

Figure 8.24: conforming to the schematic of the design [1].

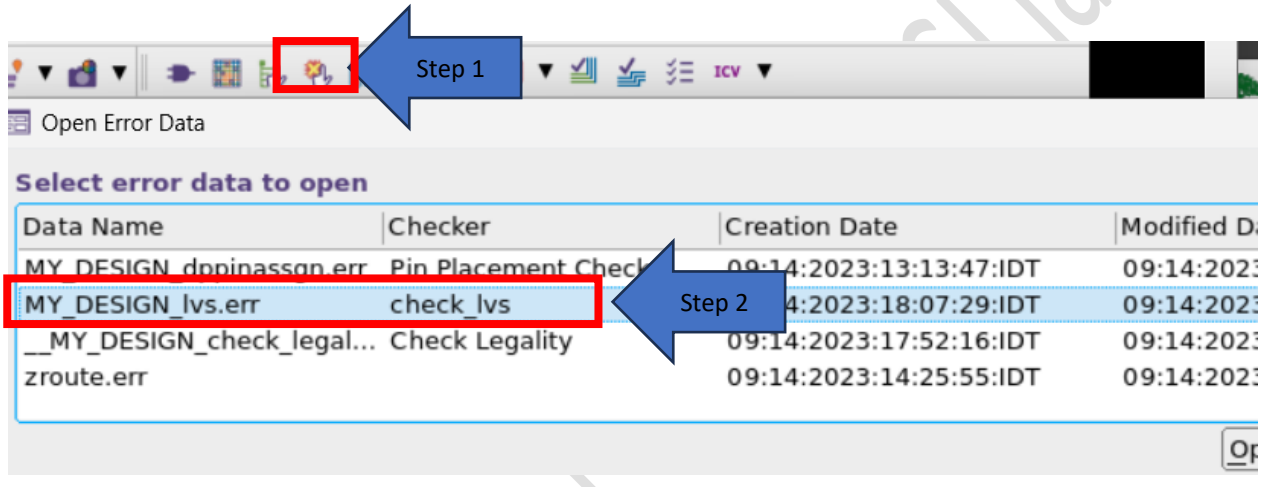

Figure 8.25: To find error [1].

8.6 References :

[1] Synopsys: Synopsys | EDA Tools, Semiconductor IP and Application Security Solutions