

Faculty of Engineering and Technology
Department of Electrical and Computer Engineering



Dr. Khader Mohammad

Nour Daghlas

4/10/2019

ICC Design Flow

Lab3

Goal: Introduction to VLSI Layout Synthesis tool ICC and ASIC design flow

Procedure:

During this lab, the students are expected to be able to perform the following:

- 1-Initiate a session through putty/xming
- 2-Be briefly introduced to TCL Coding, its uses and advantages
- 3-Launch the ICC© tool by Synopsys
- 4-Use the gui to view standard cells provided by Synopsys
- 5- Use the gui to view a premade simple design of 2 input AND gate using 2-Input NAND gates
- 6- Use the gui to import a premade 2x4decoder
- 7-Use gui to create power nets, create floorplan, create power rings, do automatic placement, pre-route standard cells and eventually route design.
- 8-Introduce students to netlists and let them create a 2x1Mux netlist after giving them the list of standard cells they can use and do the above steps.
- 9-Redo step 7, do LVS, save the design/Verilog file somewhere they know and exit.

Part 1:

- I) Double Click xming application and make sure the tray icon on the bottom left corner appears

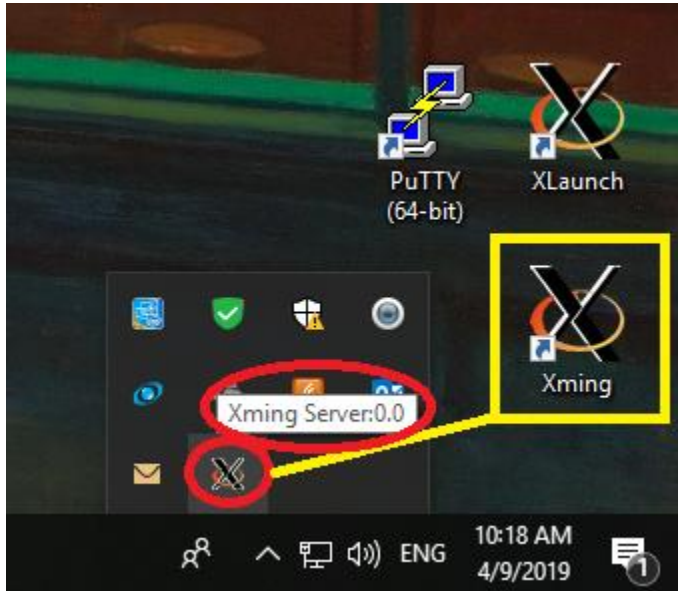


Figure 1

From here, you should keep the ID Number(0.0 in this case) for later steps.

- II) Double click PuTTY application and configure it as shown in figures 2 and 3.

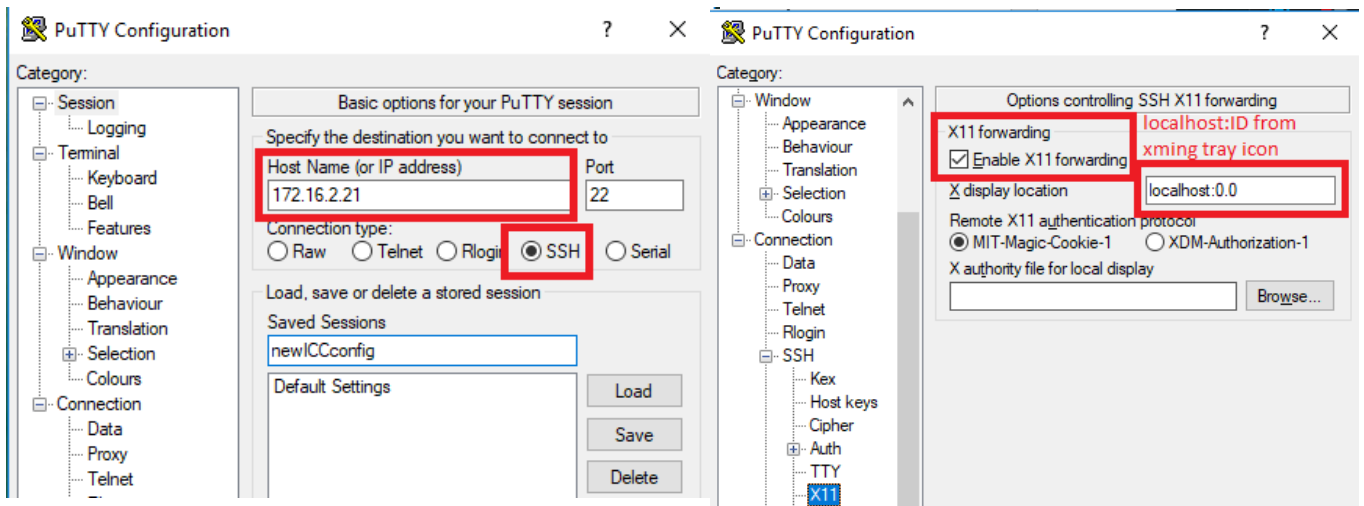


Figure 3

Figure 2(Connection->SSH->X11)

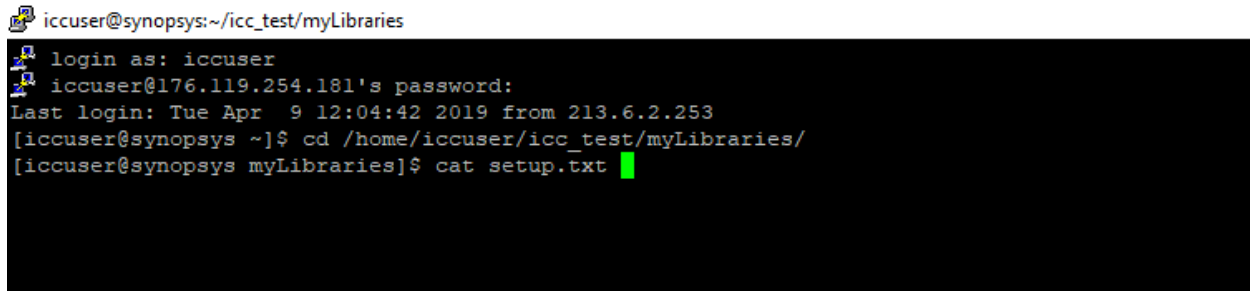
After changing the highlighted settings, go back to “Session”, save your configurations to be able to not have to change the settings every time you log in. Press open and a command line should appear. Input your username and password as directed.

Part 2:

The ICC tool could be fully operated using the command line. When larger teams work on projects and people are testing different things, usually there's a specific set-up for each project, that's why TCL(Tool Command Language, pronounced "TICKLE") is used to automate the process of setting the application for design, testing, verification and viewing. TCL is a radically simple open-source interpreted programming language that provides common facilities such as variables, procedures, and control structures as well as many useful features that are not found in any other major language.

Part 3:

- I) When launching the ICC Tool, the current directory you're in will be where command logs, output logs are stored, so be sure to launch the application from the same directory every time to not have a lot of similar files all over. To launch the application, move into the directory `/home/iccuser/icc_test/myLibraries` as shown in figure 4 and type the command **cat setup.txt** and follow it with **icc_shell -gui** to launch the shell environment and gui of the tool.



```

iccuser@synopsys:~/icc_test/myLibraries
login as: iccuser
iccuser@176.119.254.181's password:
Last login: Tue Apr  9 12:04:42 2019 from 213.6.2.253
[iccuser@synopsys ~]$ cd /home/iccuser/icc_test/myLibraries/
[iccuser@synopsys myLibraries]$ cat setup.txt

```

Figure 4

- II) Next thing you want to do is execute the first two lines that were displayed from the setup.txt file on your command window in the "Every Time" section of the file. Or copy them from here:

```

source /home/iccuser/myTCLCodes/lib_container.tcl

set_tlu_plus_files -max_tluplus /home/iccuser/32-
28nm_EDK_01312018/SAED32_EDK/tech/star_rcxt/saed32nm_1p9m_Cmax.tluplus -min_tluplus
/home/iccuser/asic_design_flow_tutorial/library/syngen_32nm/32-
28nm_EDK_01312018/SAED_EDK32.28nm_TECH_v_01132015/SAED32_EDK/tech/star_rcxt/saed32nm_
1p9m_Cmin.tluplus -tech2itf_map /home/iccuser/asic_design_flow_tutorial/library/syngen_32nm/32-
28nm_EDK_01312018/SAED_EDK32.28nm_TECH_v_01132015/SAED32_EDK/tech/star_rcxt//saed32nm
_tf_itf_tluplus.map

```

- III) Now we have to open a design library from the gui as shown in figure 5.

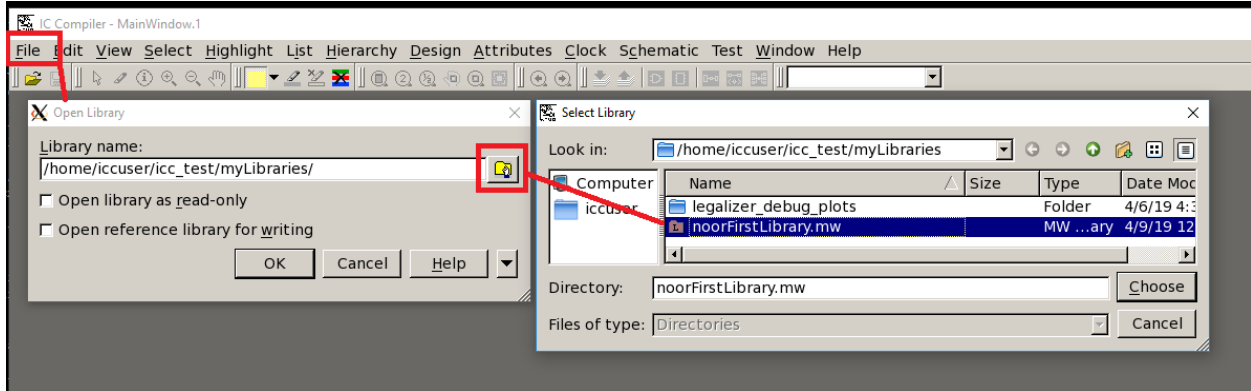


Figure 5

To see the open library dialogue, go to **File->Open Library** and complete the rest of the steps as instructed. Now after this step, we have loaded the library which we previously created designs in and can reopen designs and perform actions on them. For now, we'll be viewing a few standard cells provided with the tool to see in detail how each cell we'll be using in the future is constructed.

Part 4: Use the gui to view standard cells provided by Synopsys

To do this, we need to have done all the steps above and have our library open.

Go to **File -> Open Design -> Choose "saed32nm_lvt_1p9m"** from Libraries drop menu -> **Choose design to open**. The list displayed is the list of cells which we can use to create our netlists and future designs and we'll be using it lightly towards the end of the lab.

For now, lets view a simple inverter "INVX0_LVT", use the filter option and write down the name of the inverter and double click on it. The layout window in figure 6 should appear. To view the names of nets/inputs/outputs, scroll down and tick "Text" and press "Apply" when it turns to blue.

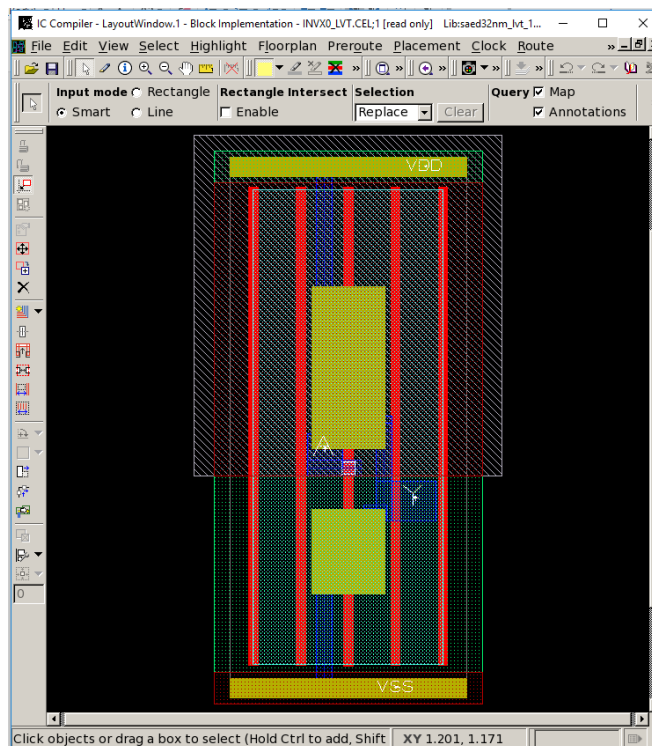


Figure 6

The name denotes how many inputs the device has and how much load it can drive in multiples of CSL (Capacitive Standard Load). For example, NAND4x2 is a 4-input NAND gate and can drive loads up to 2*CSL. These details can be found in the standard cell library DATABOOK provided by Synopsys for the specific process. In figure 7, you can see the datasheets regarding 90nm process inverter. Go back to the main window and load the "INVX8_LVT" design and notice the difference in construction between the first and second designs. What changed? Is the functionality the same? Why can the X8 inverter handle greater output load than the X0 inverter(construction wise)?

Table 9.2. Inverter Electrical Parameters and Areas

Cell Name	Operating Conditions: VDD=1.2 V DC, Temp=25 Deg.C, Operating Frequency: Freq=300 MHz, Capacitive Standard Load: Csl=13 fF				Area (μm^2)
	Cload	Prop Delay (Avg) ps	Power (VDD=1.32 V DC, Temp=25 Dec.C)		
			Leakage nW	Dynamic nW/MHz	
INVX1	1 x Csl	38	88	12	6.4512
INVX8	8 x Csl	39	582	78	14.7456
INVX32	32 x Csl	41	2510	358	47.0016

Figure 7: http://web.engr.oregonstate.edu/~traylor/ece474/reading/SAED_Cell_Lib_Rev1_4_20_1.pdf

Part 5: Use the gui to view a premade simple design of 2 input AND gate using 2-Input NAND gates

- 1) Instead of using saed library when opening a design, we'll choose our own design library now and see how an AND gate is built and connected using NAND gates. Do you see any issues with this design? (Floorplan) Can you trace where the input/output ports lead? Can you draw the logic gate representation of this design?

Part 6: Use the gui to import a premade 2x4decoder

Close all open layouts by choosing from the main window: **File -> Close Design** until all designs are closed while discarding any changes made.

Now to synthesize a premade Verilog netlist, go to **File -> Import -> Verilog** and navigate to the Verilog file we'll be working on, which is titled "decoder2x4_Noor.v".

A layout window looking like figure 8 should open.

This window is displaying all the standard cells that'll be a part of your design after finishing the rest of the design steps(Floorplanning/Placement/Routing).

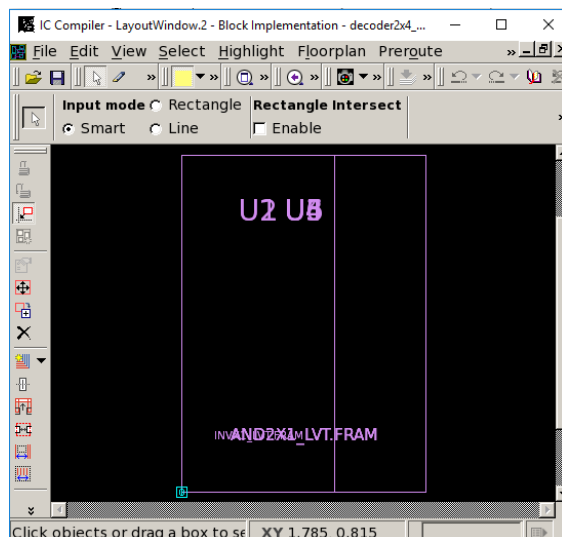


Figure 8

II) Floorplanning

“At the floorplanning stage, we have a netlist which describes the design and the various blocks of the design and the interconnection between the different blocks. The netlist is the logical description of the ASIC and the floorplan is the physical description of the ASIC. Therefore, by doing floorplanning, we are mapping the logical description of the design to the physical description. The main objectives of floorplanning are to minimize Area and Timing(delay).

During floorplanning, the following are done:

- The size of the chip is estimated.
- The various blocks in the design, are arranged on the chip.
- Pin Assignment is done.
- The I/O and Power Planning are done.
- The type of clock distribution is decided” – ASIC Design Flow Tutorial 3228GL

To perform floorplanning, we simply go to **Floorplan -> Create Floorplan** and set the values as shown in figure 9 and press OK.

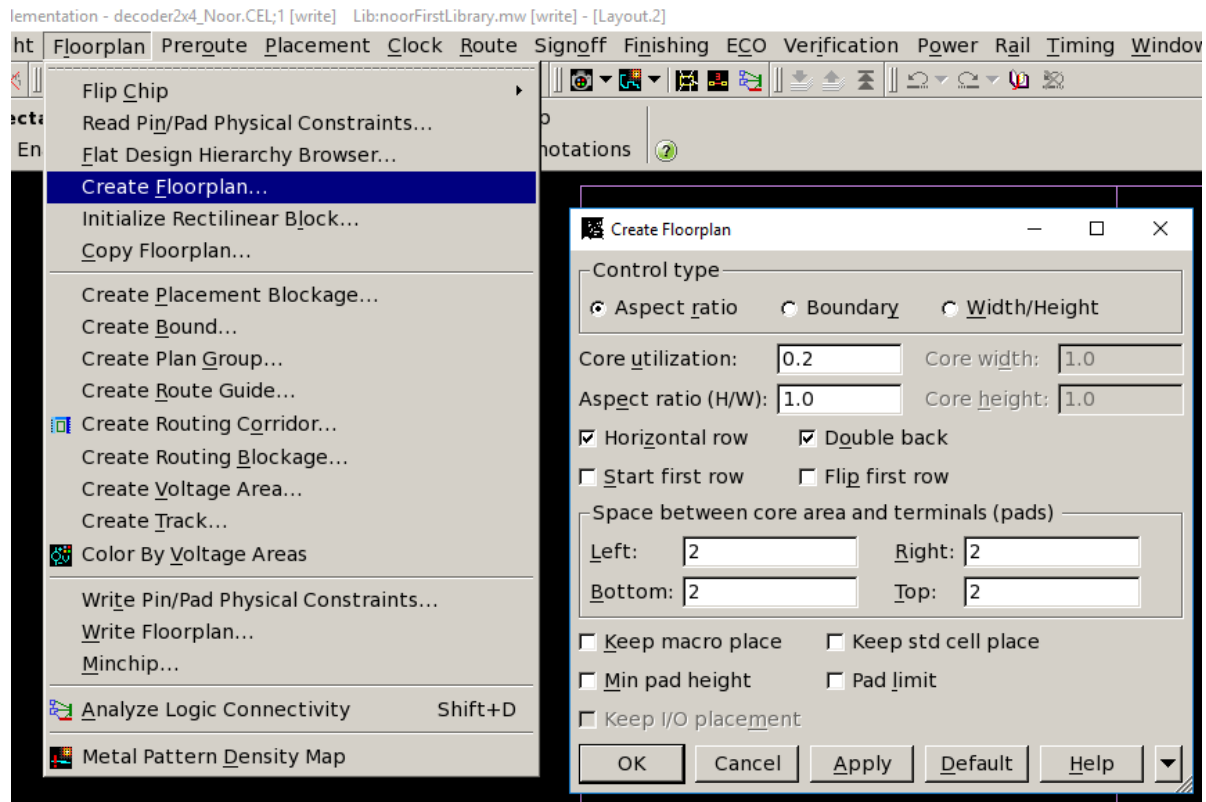


Figure 9

Core utilization is how much area will be utilized. For this small design, we’re specifying 0.2 because optimizing it more than that will interfere with the DRC checking. For relatively larger designs(500 devices+) this option will be more flexible and affected by many factors.

Next, we’ll pre-route the ground/power nets for the standard cells. To do this, go to **Preroute -> Derive PG Connection** and specify the VDD and VSS as shown in figure 10 and press OK.

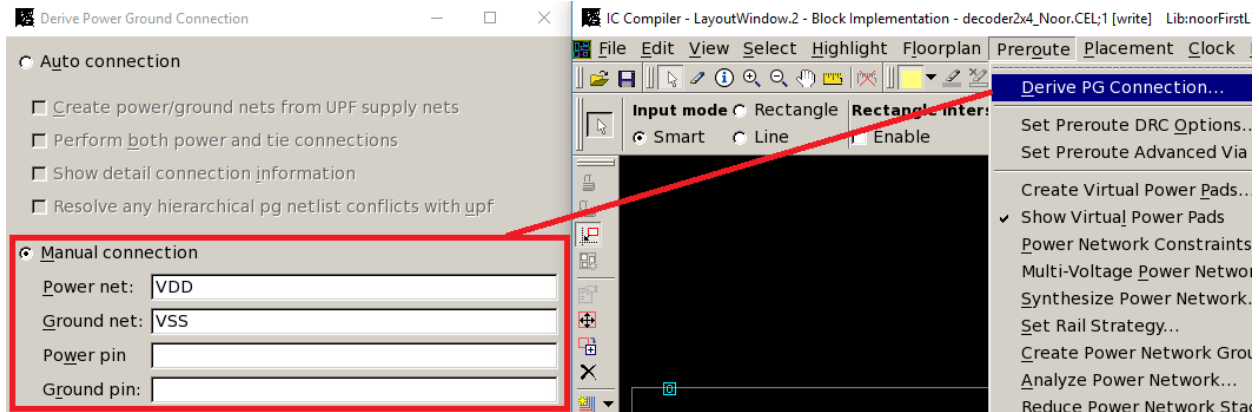


Figure 10

Next step is creating the power rings which will provide the power to the chip. We can achieve this by going to **Preroute -> Create Rings -> Rectilinear** and specify the options as shown in figure 11.

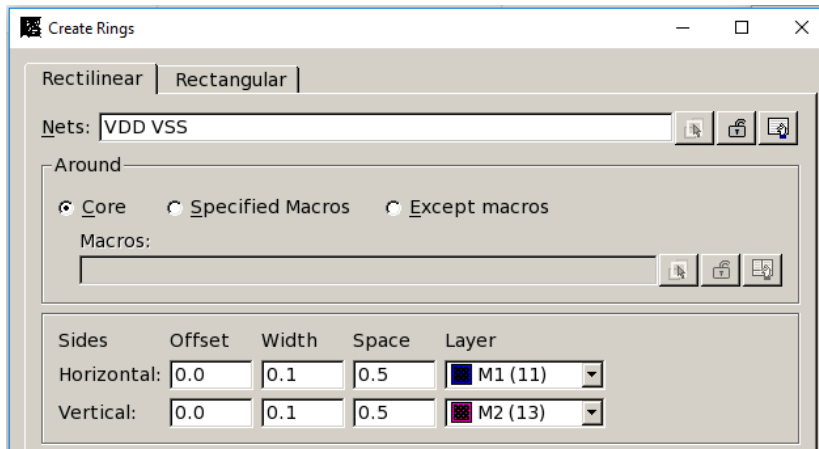


Figure 11

In figure 12, we can see the exact detailed floorplanning flowchart but we’ll skip most of the steps for now.

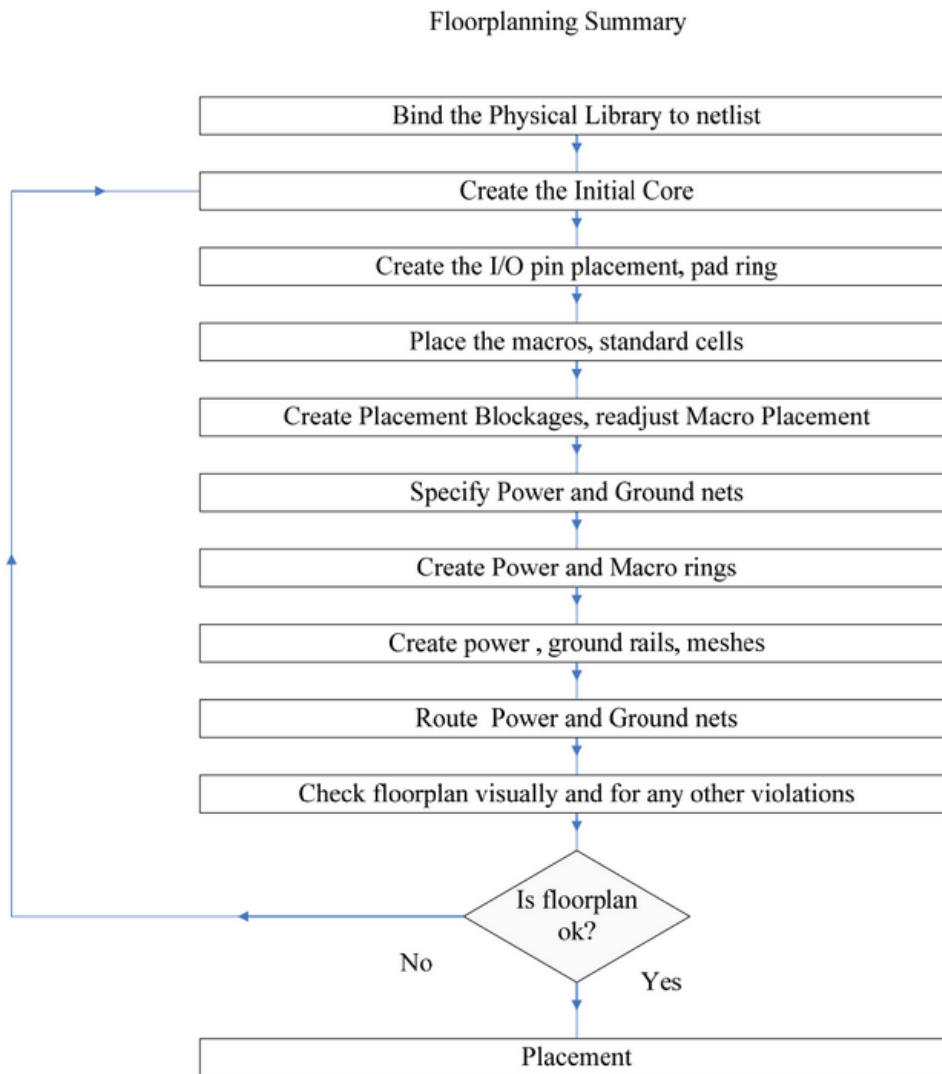


Figure 12

III) Placement

Placement is a step in the Physical Implementation process where the standard cells location is defined to a particular position in a row. Space is set aside for interconnect to each logic/standard cell. After placement, we can see the accurate estimates of the capacitive loads of each standard cell must drive. The tool places these cells based on the algorithms which it uses internally. It is a process of placing the design cells in the floorplan in the most optimal way.

To perform automatic placement. Go to **placemen -> Core Placement and Optimization -> Keep default values and press OK**. The cells now should be placed automatically within the floorplan and you should obtain an arrangement such as in figure 13.

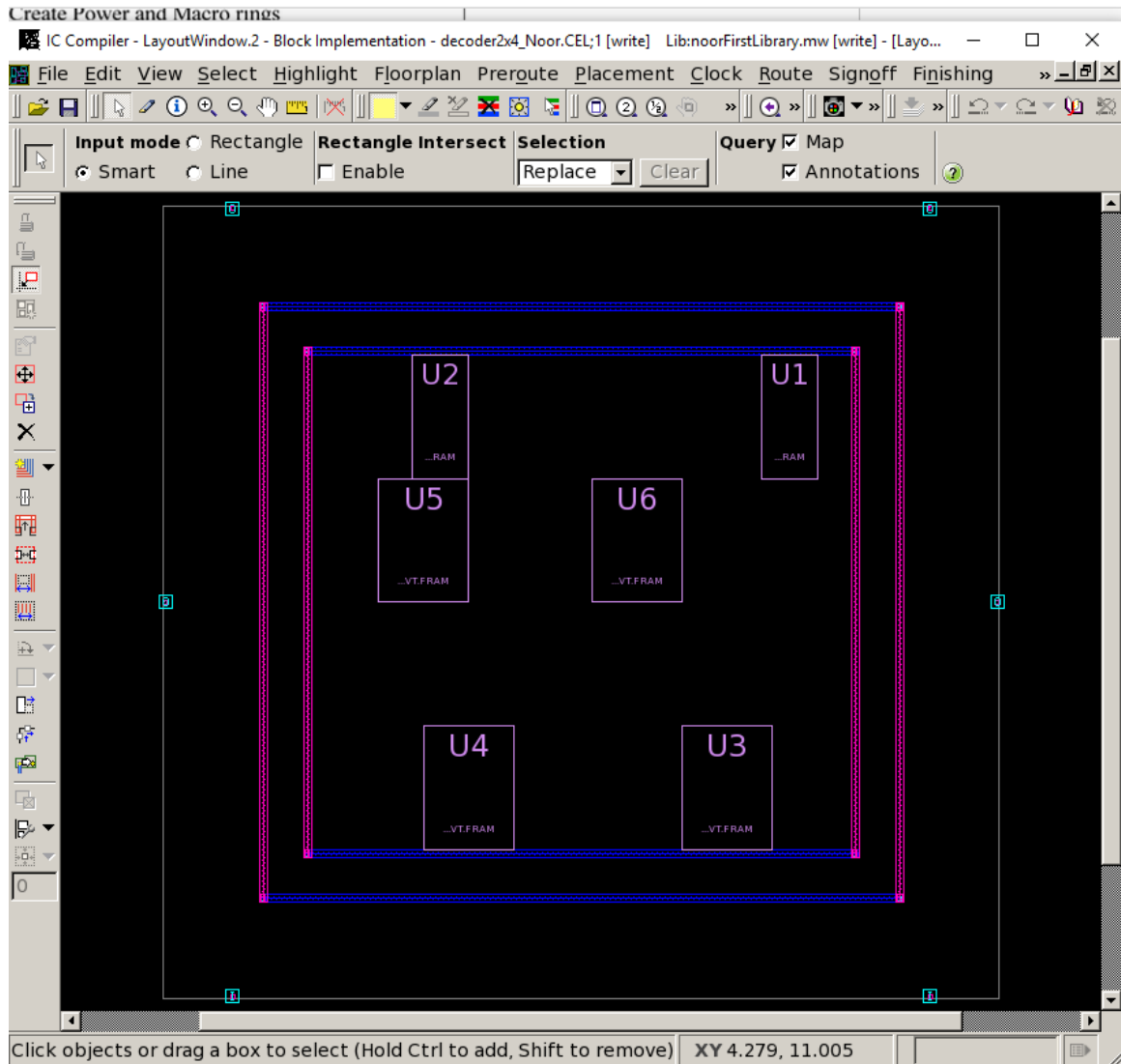


Figure 13

Now we are done with placement. But many options for placement can be used such as are recovery, congestion and power optimization and they can be optimized for separately at advanced levels.

We will run one last operating which is pre-routing the standard cells to the power nets. We can achieve this by pressing **preroute** -> **preroute standard cells** while specifying VDD and VSS as the nets we're prerouting like in figure 14.

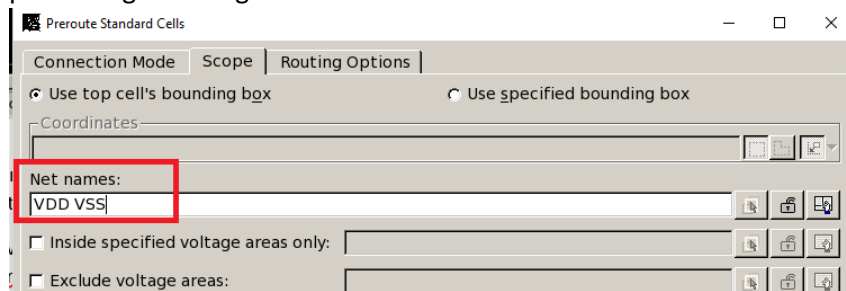


Figure 14

Next step would be synthesizing the clock tree, but since we have no sequential elements in our design, we'll skip this step and head straight to routing.

IV) Routing

This is one of the most crucial steps. After routing, we can start checking for DRC's and ensuring no LVS or timing violations are occurring. For now, we'll do basic routing by clicking on **Route-> Core Routing and Optimization -> Keep Default Values and press OK**. The layout window should update with the routes similar to figure 15.

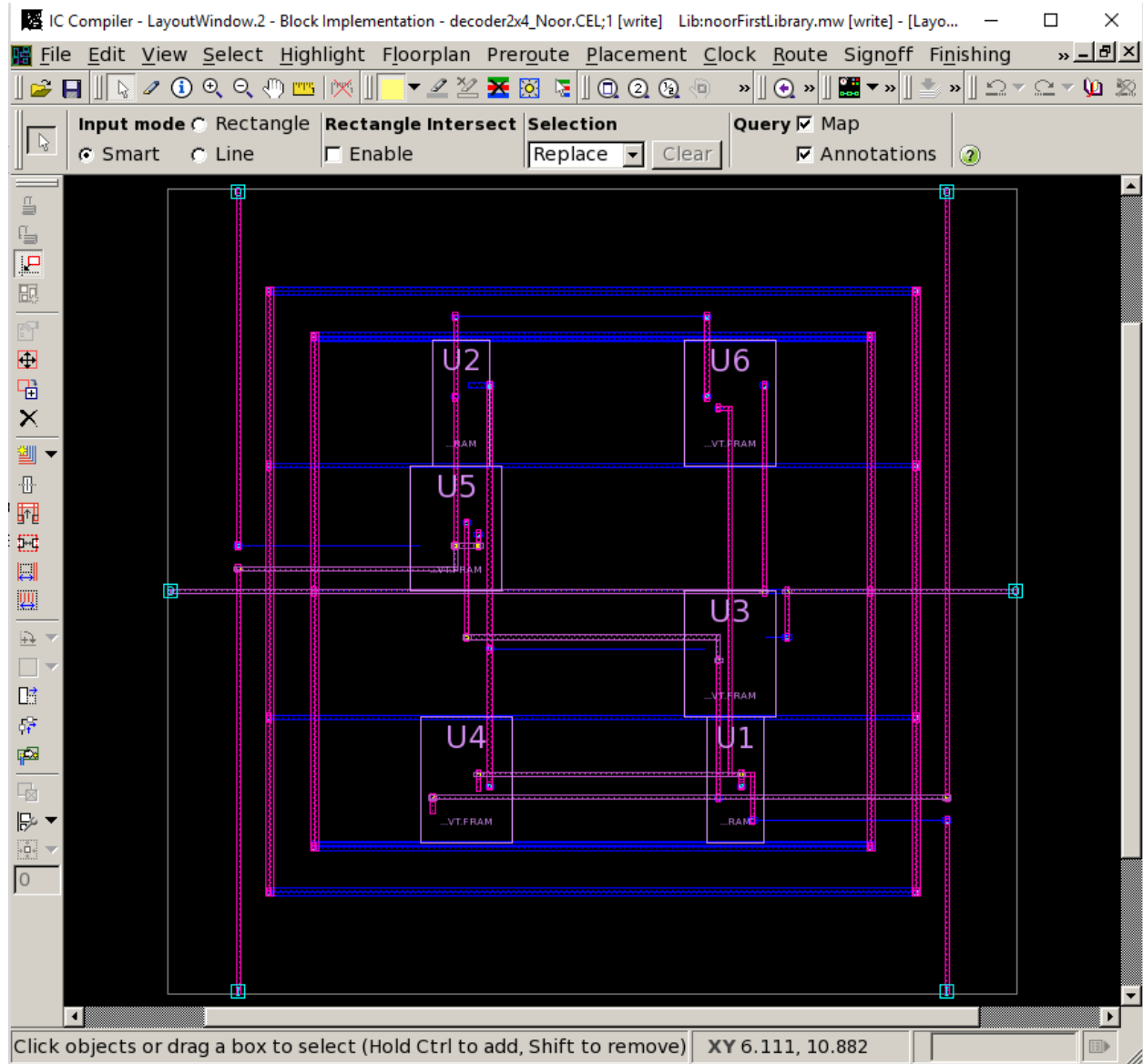


Figure 15

Now we'll just save our design by first going to the main window, pressing **File -> Save Design -> Save All** then **File -> Close Design**. These are two of the most important steps since if not saved correctly, the entire design can be unreadable. It is also standard practice to save the design after each step of Floorplanning, Placement, Clock Distribution and Routing with respective names such as design_floorplan, design_place, design_cts and design_route.

Part 8: Introduce students to netlists and let them create a 2x1Mux netlist after giving them the list of standard cells they can use and do the above steps.

Standard Cells are the cells that are given by Synopsys to implement any future desired logic behavior. These include optimized AND, NAND, NOR, XOR, MUX, DFF, INV and a variety of them like we've seen in part one(X1, X8, 2X1,2X2, 4X2, etc.). We link the specific process standard cell library to our designs when creating our library for the first time.

In this part, we'll use the gate construction of a 2x1MUX shown in figure 16 to type the netlist ourselves, redo steps in part 7 and save our design and verify the design's Layout Vs. Schematic(LVS)/Check for DRC violations.

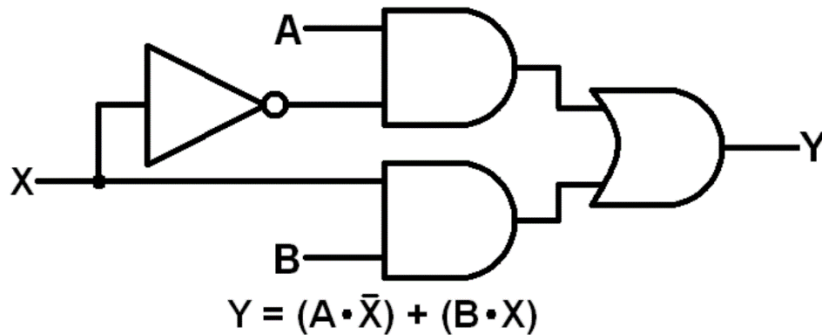


Figure 16

Our Verilog netlist should start with the module name and the inputs/outputs stated in the beginning, for example

```
module AND2X1 (A1, A2, Y);
input A1,A2;
output Y;
.....
endmodule
```

The list of the standard cells we'll need to use in order to build the mux shown in figure 17. We'll be using Verilog to write the netlist as Verilog structural code, saving it and then importing it like we did to the decoder. (Avoid using INVX1(A,Y))

```
AND2X1_LVT (A1, A2, Y)
OR2X1_LVT (A1, A2, Y)
NAND2X1_LVT (A1, A2, Y)
NOR2X1_LVT (A1, A2, Y)
```

Figure 17

The way we'd use these cells is for example:

```
AND2X1_LVT (.A1(input1), .A2(input2), .Y(output));
```

We can define nets using the wire directive. For example:

```
wire and1Output;
```

Part 9: Redo step 7, do LVS, save the design/Verilog file somewhere they know and exit.

Redo part 7 using the mux, save the design. Can you see how the cells are arranged and can you navigate through it and follow the gate design by looking only at the layout? Try to increase the level of viewing by pressing F8, incrementing the viewing level by 1 and pressing apply. You'll be able to see the design in its transistor level. As you can see in figure 18, it's also easier to trace the inputs/outputs and analyze the logic in this view.

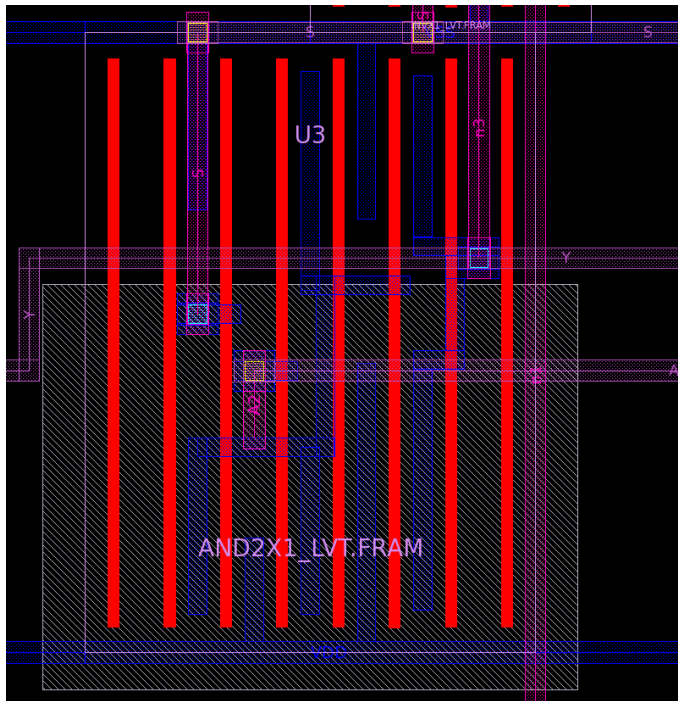


Figure 18

We can do LVS verification by going to the design window and pressing **verification -> LVS -> Keep Default Settings and press OK** and notice if any errors pop up. If the design is clean(DRC 0 violations, LVS 0 violations) the designer can proceed to optimize the design or run other tests until the layout is fab-ready.

Lastly, if you have any questions, please ask your lab assistant.