Advanced physical design using Sky-130 nm process

This is a compilation of notes of the 3rd stage of the chip design program offered by IIIT-Bangalore and Samsung . The below is a table of contents of this document.

2.Optimise Placement Using Estimated Wire-Length and Capacitance
3.Final Placement Optimization
4.Need for Libraries and Characterisation
5.Congestion Aware Placement Using RePLACE

3.Cell Design and Characterisation Flows -

1.Inputs for Cell Design Flow
2.Circuit Design Step
3.Layout Design Step
4.Typical Characterisation Flow

4.General Timing Characterisation Parameters -

1.Timing Threshhold Definitions
2.Propogation Delay and Transition Time

SKY130 DAY 3: Design Library Cell Using Magic Layout and NGSPICE characterisation

1.Labs for CMOS Inverter NGSPICE Simulations -

1.IO Placer Revision
2.SPICE Deck Creation For CMOS Inverter
3.SPICE Simulation Lab for CMOS Inverter
4.Switching Threshhold Vm
5.Static and Dynamic Simulation of CMOS Inverter
6.Lab Steps to GitClone VSDSTD Cell Design

Inception of Layout Â CMOS Fabrication Process -
1.Create Active Regions
2.Formation of N and P well
3.Formation of Gate Terminal
4.Lightly Doped Drain [LDD] Formation
5.Source Â Drain Formation
6.Local Interconnect Formation
7.Higher Level Metal Formation
8.Lab Introduction to SKY130 Basic Layers Layout and LEF using Inverter
9.Lab Steps to Create STD Cell Layout and Extract SPICE Netlist

3.SKY130 Tech File Labs -

1.Lab Steps To Create Final SPICE deck using SKY130 tech
2.Lab Steps to Characterise Inverter using SKY130 Model Files
3.Lab Introduction to SKY130 PDKs And Steps to Download Labs
4.Lab Introduction to Magic Tool Options and DRC Rules
5.Lab Introduction to Magic and Steps to Load SKY130 Tech Rules
6.Lab Excercise to Fix poly.9 error in SKY130 Tech-File
7.Lab Excercise to Implement Poly-Resistor Spacing to Diff and Tap

8.Lab challenge to describe DRC error as geometrical construct
9.Lab challenge to find missing or incorrect rules and fix them

SKY130 DAY 4 : Pre-Layout Timing Analysis and Importance of Good Clock Tree -

1.Timing Modelling Using Delay Tables

1.Lab Steps To Convert Grid Information Into Track Information
2.Lab Steps to Convert Magic Layout to STD Cell LEF
3.Introduction of Timing libs and Steps To Include New Cell in Synthesis
4. Introduction to Delay tables
5.Delay Tables usage Part 1
6.Delay Tables Usage Part 2
7.Lab Steps To Configure Synthesis Settings to Fix Slack and Include VSDINV

2.Timing Analysis With Ideal Clocks Using Open STA -

1.Setup Timing Analysis And Introduction to Flip-Flop Setup Time
2.Introduction to Clock Jitter and Uncertainty
3.Lab Steps to Configure OpenSTA for Post-Synth Timings Analysis
4.Lab Steps to Optimize Synthesis to Reduce Setup Violations
5.Lab Steps to do Basic Timing ECO

3.Clock Tree Synthesis TritonCTS and Signal Integrity -

1.Clock Tree Routing and Buffering Using H-Tree Algorithm
2.Crosswalk and Clock Net Shielding
3.Lab Steps to run CTS using TritonCTS
4.Lab Steps to Verify CTS Runs

4.Timing Analysis With Real Clocks Using Open STA -

1.Setup Timing Analysis Using Real Clocks
2.Hold Timing analysis using Real clocks
3.Lab Steps to Analyse Timing With Real Clocks Using OpenSTA
4.Lab Steps to Excecute OpenSTA With Right Timing Libraries and CTS Assignment
5.Lab Steps to Observe Impact of Bigger CTS Buffers On Setup And Hold Timing

SKY130 DAY 5: Final Steps For RTL2GDS Using TritonRoute and OpenSTA

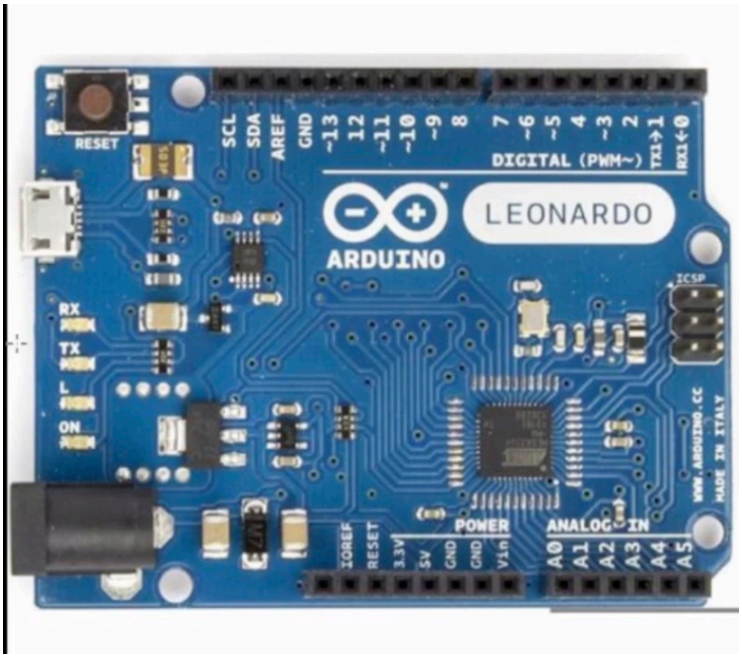
1.Routing and Design Check [DRC] -

1.Introduction to Maze Routing and Lee's Algorithm
2.Lee’s Algorithm Conclusion
3.Design Rule Check

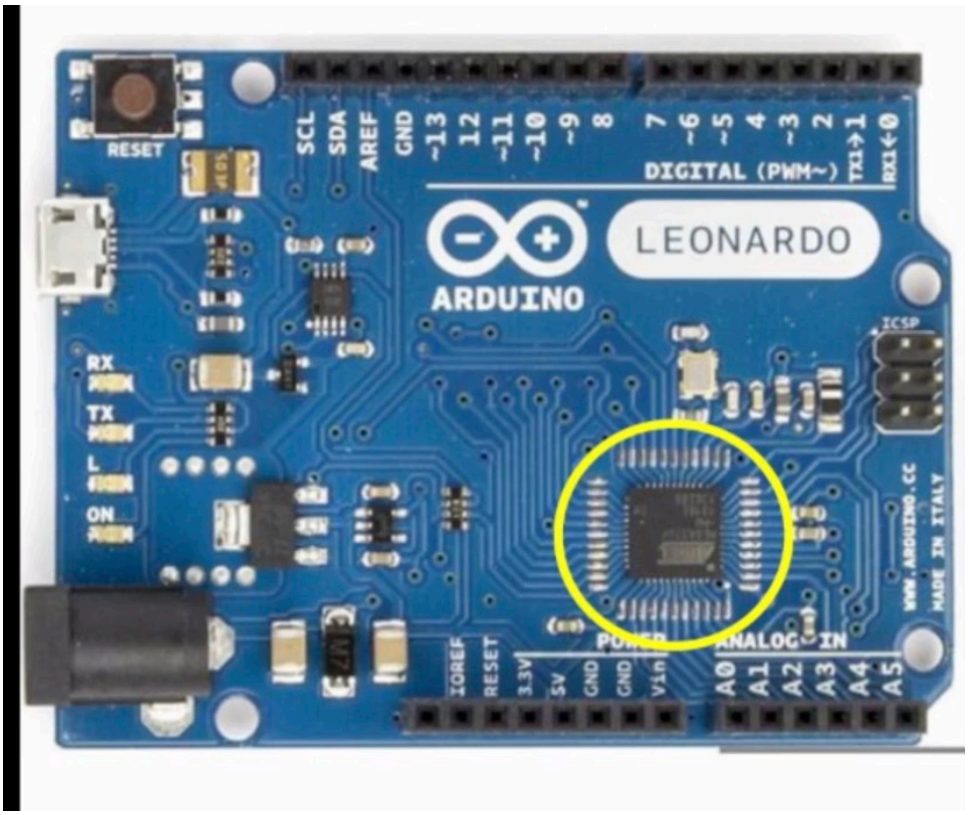
2.Power Distribution Network and Routing -

Sky130 Day1

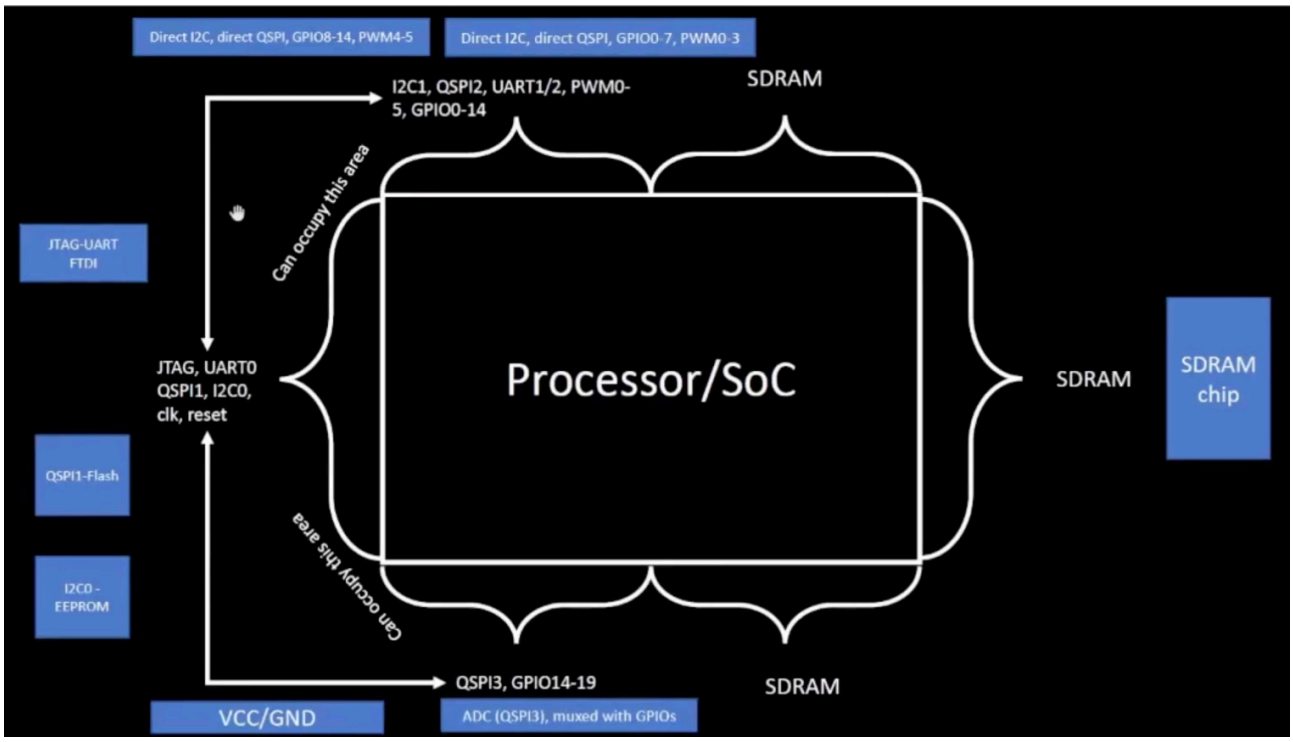
Let us start by looking at a common board that many of us can recognise, that is an Arduino Board . It usually used in small scale projects



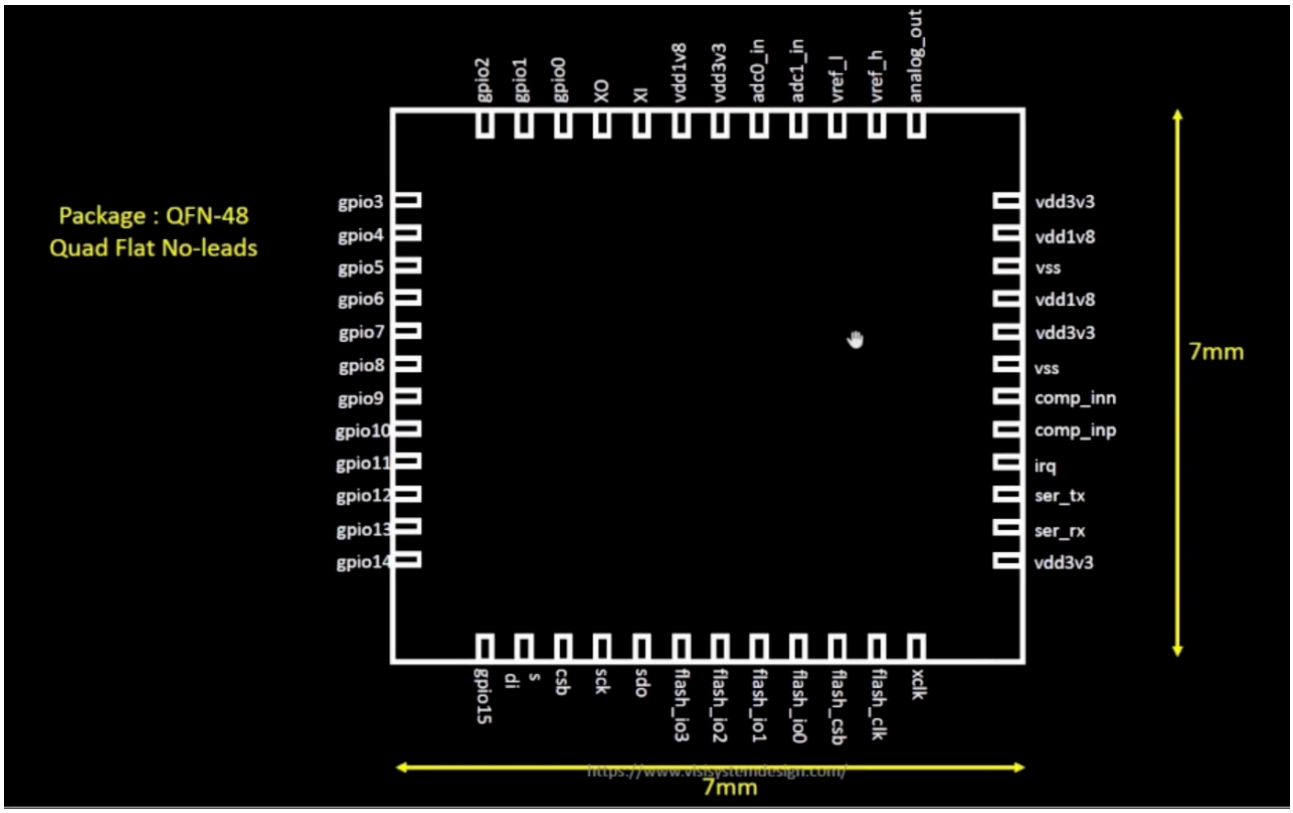
Here, we can see the pins, buttons, and all the other bells and shingles that this board has. But we are not invested in that . What we really interested in, is the chip, which is known as an SoC(System on CHIP). The SoC is show in the below figure.



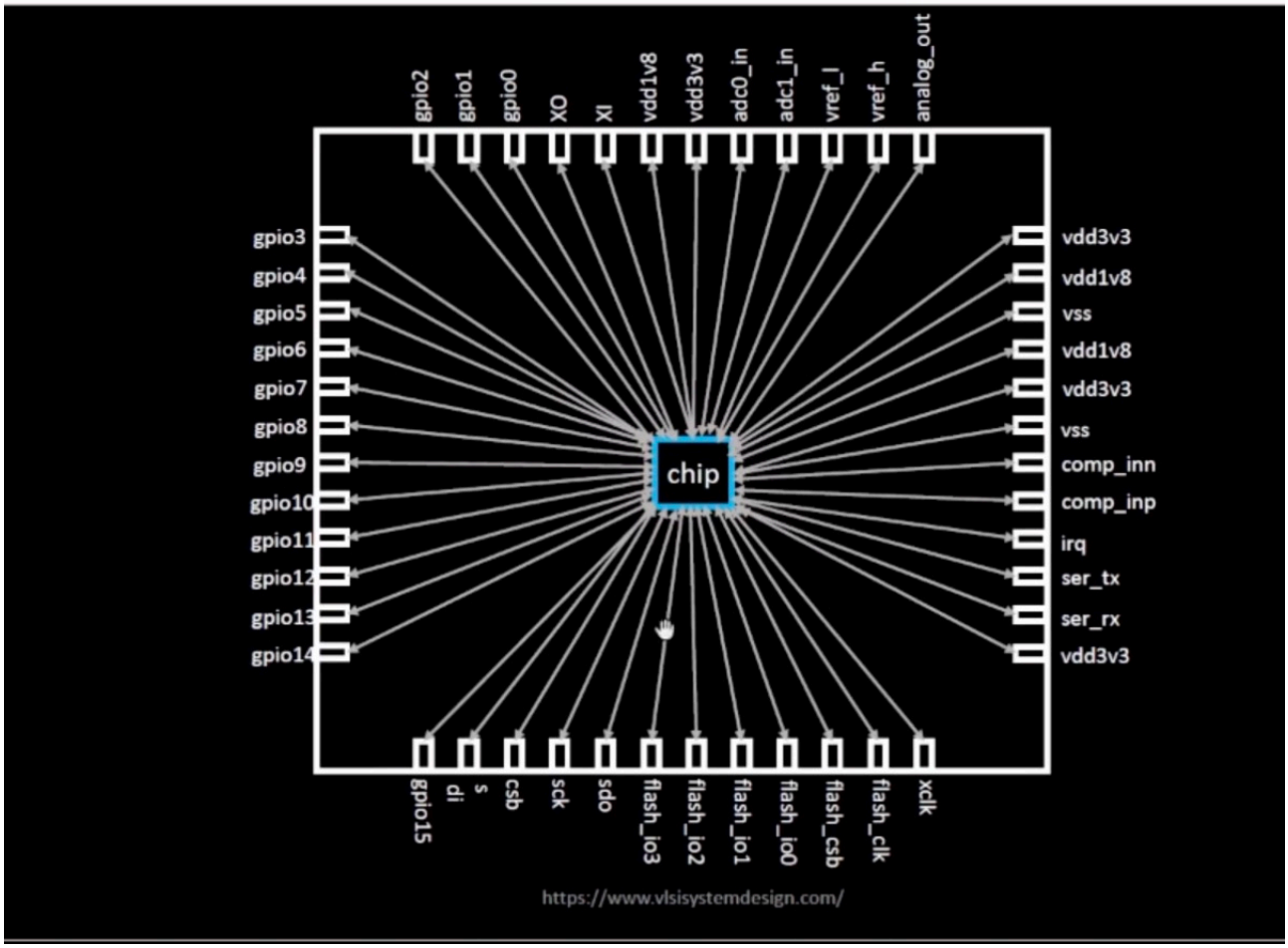
The picture below is an diagram of block diagram of a Arduino board.



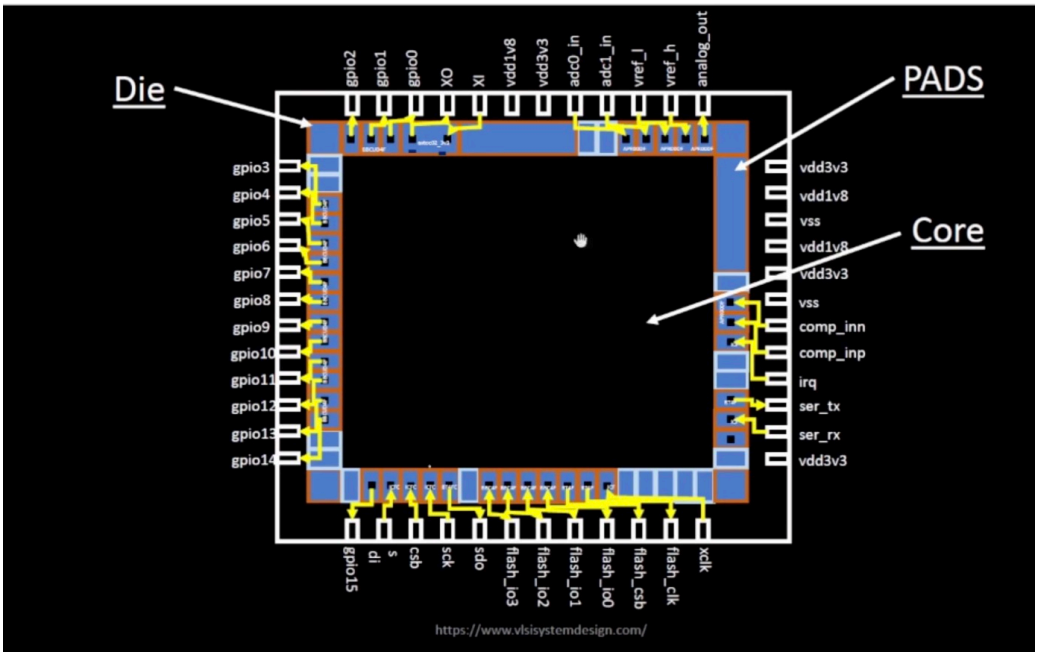
Now lets look deeper into the the workings of the Processor or the SoC, as we can see the above picture.



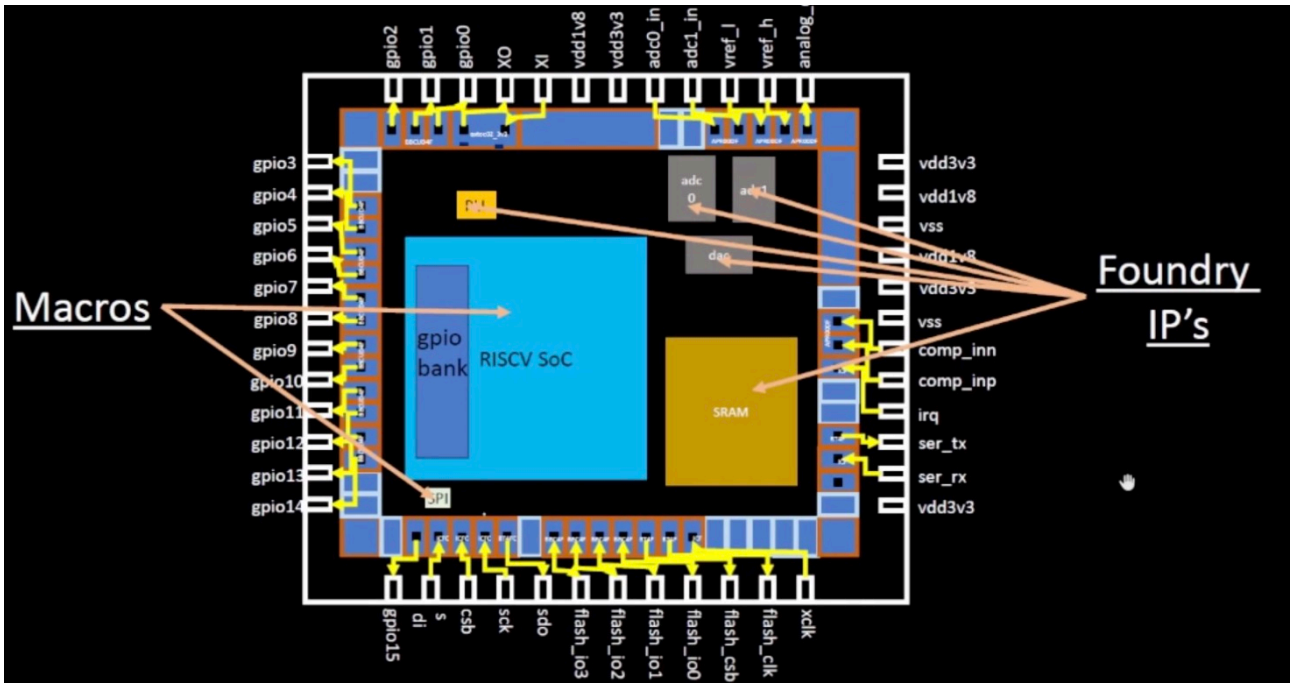
Now, in the above picture, the box you is not called a chip, but instead called a package. We can see that there are many types of packages such as QFT-48, which is the one shown here. Now, lets go deeper inside the package.



Now, the blue box is the chip, and the chip is connected to the input/output pins through connections. Now, this is a really oversimplified diagram. The real diagram (still way oversimplified) is given below.



The pads are the connections that connect the chip to the external world, here, input/output Pins. The Die is an independent part of the package that can be used to fit other devices that can do other functions. So now, lets get into the impoertnant part, i.e, the core. This houses the logic, I.e your AND gates, OR gates, MUXes, etc. now lets go deeper into the core



Now, the core consists of two main parts, that are the Macros and the Foundry IPs . A foundry is basically the factory where chips are made and an IP is a short form for an intellectual property . So, the Foundry IPs are parts manufactured by the company producing the SoC. They can include Sram and many other components

Macros are almost the same as Foundry IPs but, they consist of pure digital logic such as gates, MUXes etc. they can include the SPI, the GPIO bank and more.

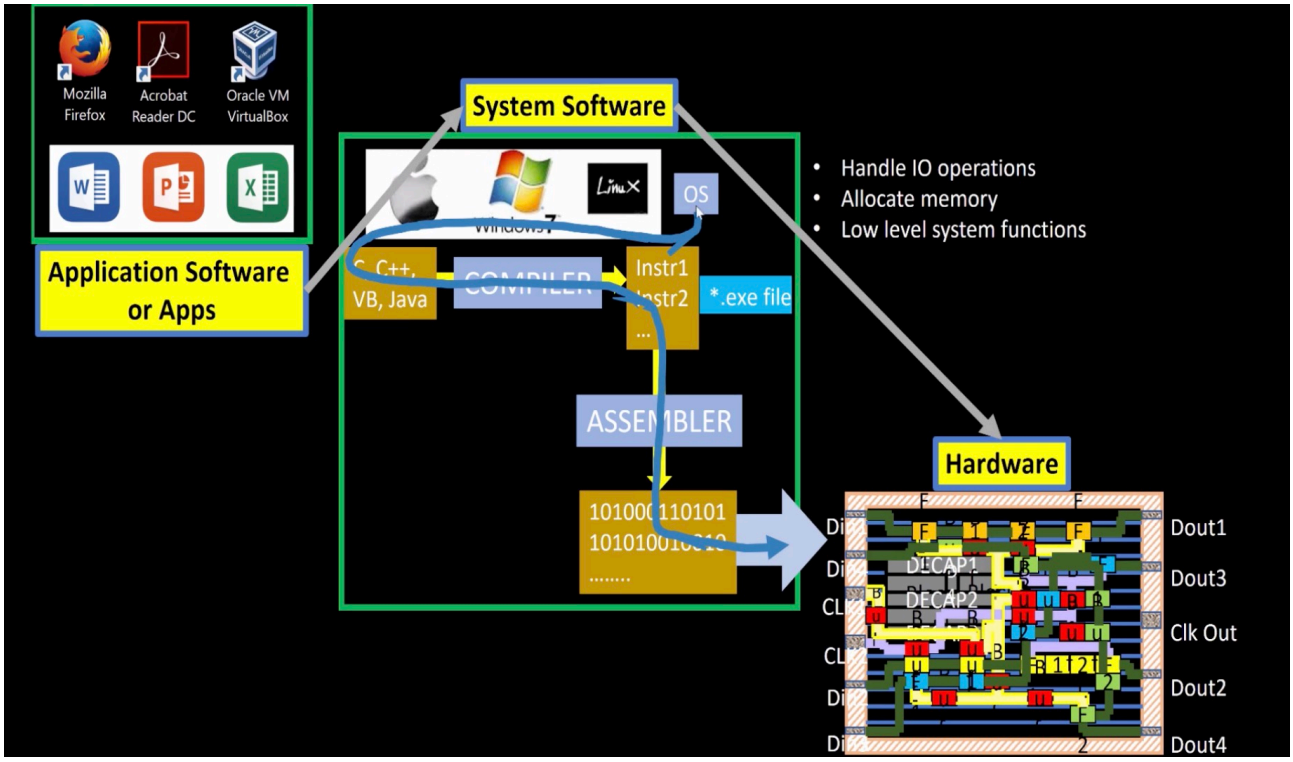
Now, there are many types of chips such as x86, ARM chips and so many more. But here there is a type of chip that is known as a RISC-V chip , which runs on RISC-V ISA , the ISA stands for instruction set architecture. What do these complicated terms mean?

In simple language, suppose, you have program in C, C++ or any programming language, and I want the chip to execute it. But the chip does not know, what the program means, so for the RISC-V chip to understand the program, the program is converted into the language that the chip understands using the RISC-V ISA.

We also use an HDL , which stands for Hardware Description Language, to make the most optimal circuit possible . In the last week we have seen an example of an hardware description language, that being Verilog HDL. Here , instead of that we are a HDL known as picorv2

So, now, how do software apps (applications) run on these chips? . so, between the app and the chip in the laptop that executes the function, the system OS converts the program in the apps written at a high level language such as python, and converts it into a set of instructions that is in machine language, (which is written in 0s and 1s) which is understandable by a chip. Now how does it do that?

It does this by passing the program through a compiler ,which converts the program Into the instruction set used by the chip, here RISC-V ISA and the assembler converts the instructions written in the RISC-V ISA format and converts it into a string of 0s and 1s.



Now, lets get started on designing a chip completely by using open sourced tools. That means no complex tools that can cost a pretty penny and those that are extremely finicky and hard to use. Now to design a chip, we mainly need three things. The first one are RTL designs. They define the information about the pure logic within the chip, such as their amount, inputs, outputs, etc These are usually easy to find and some of the places where we can find them are GitHub.com etc.

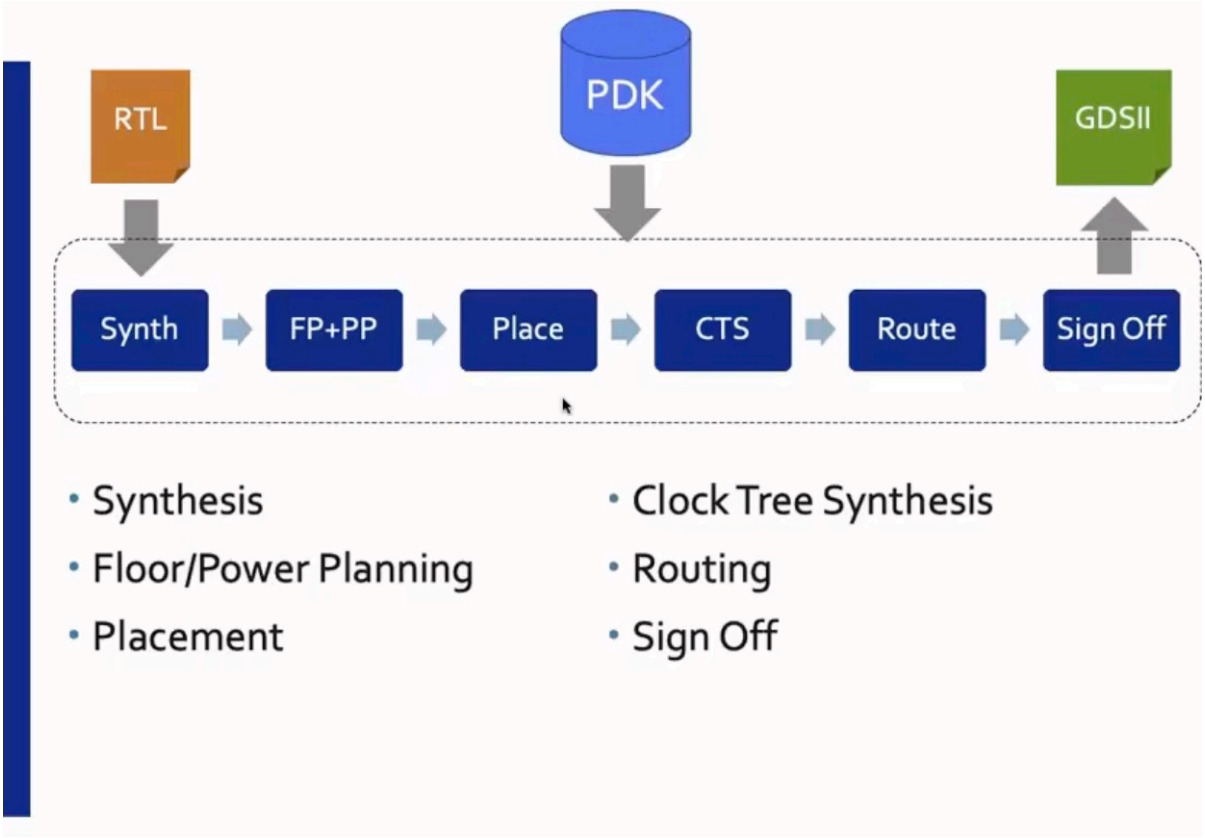
The second one of them is the EDA tools. EDA, which expands to Electronic Design Automation tools are used for design, simulation and verification and the analyzing of circuit designs. Common tools are OpenSTA, OpenRoad etc.

The third one is the PDK data. PDK which expands to , Process Design Kit data is a collection of files used to model the fabrication process for EDA tools. It is usually provided by a foundry and include Process Design Rules, Device Models, Digital Standard Cell Libraries and IO libraries.

But. Until June 2020, there were no open source PDKs as the company that made the chips made their own PDKs and would not share them. This all changed as in June 30 2020, Google collaborated with Skywater and made a PDK for 130nm processes

Now, is 130 nm fast? As nowadays computers have much smaller transistors , the latest being just 1.8 nm. But still, 130 nm is pretty fast as it has been proved by intel using their pentium 4 chips, with pipelined versions reaching upto speeds of 1GHz !

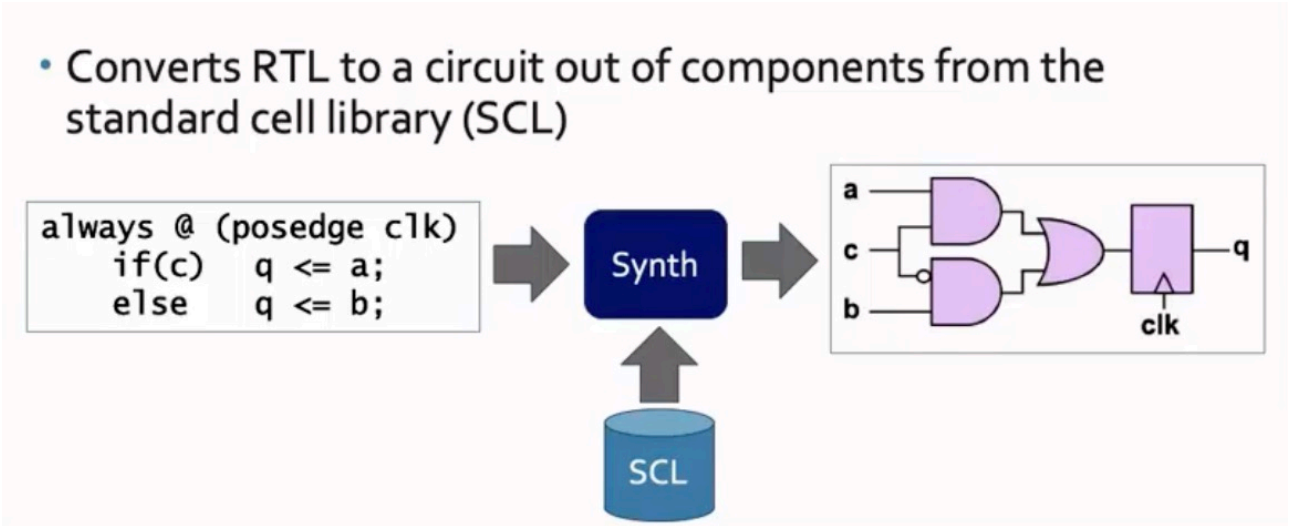
So now that we have all the open source versions of RTL designs, EDA and PDK data? Yes. But there is a flow that we have to follow a flow that is RTL2GDS flow this is a flow that converts the RTL designs into a GDSII format, which is used to define the final layout.



Now, we are going to go over all the steps in the RTL2GDSII flow.

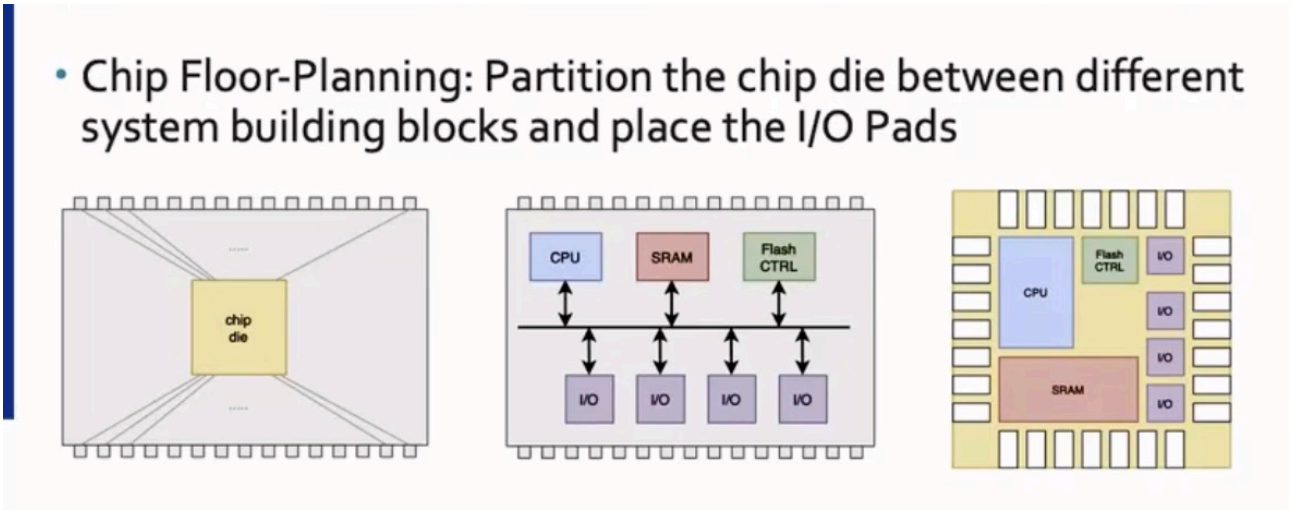
Step 1 :- Synthesis

In this step, the RTL is converted into a circuit from the Standard library cells

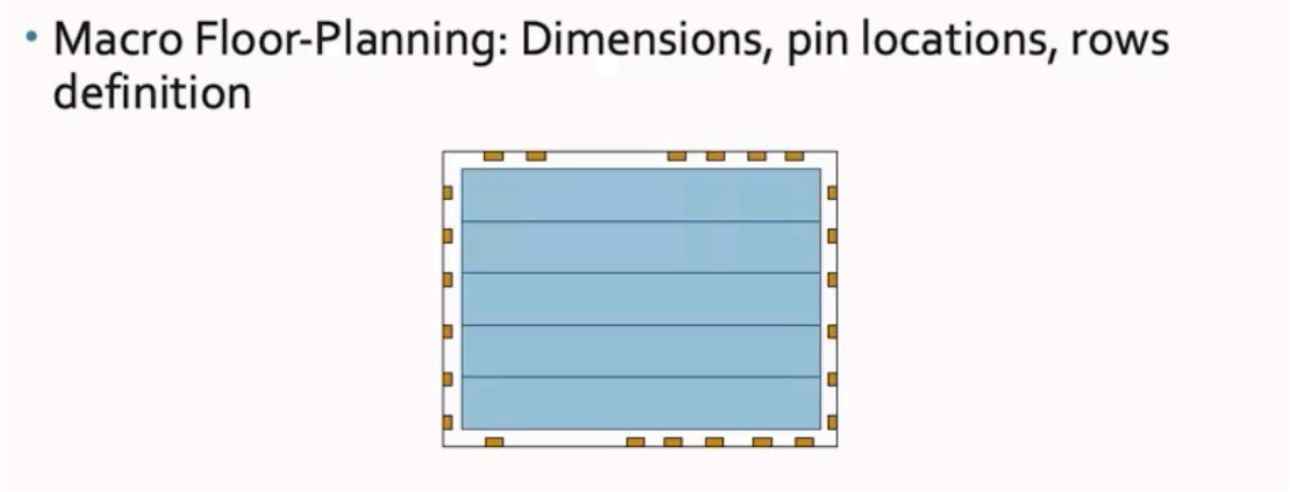


Step-2 Floor/Power Planning:-

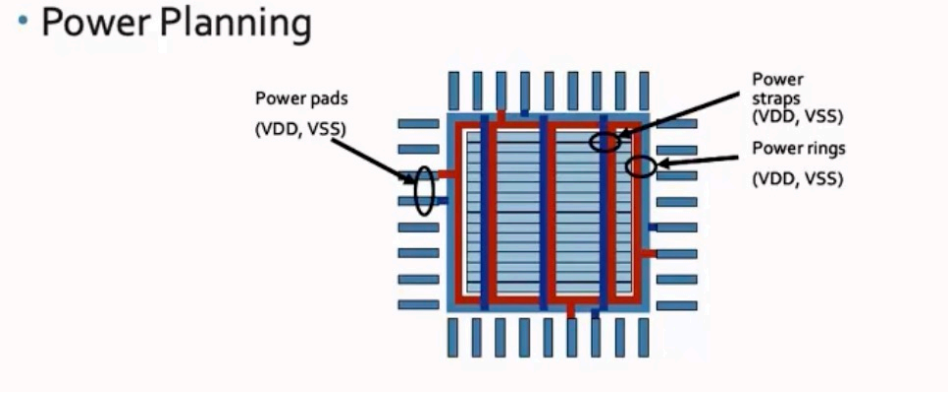
Floor planning is of two types. You are either planning for the whole chip



Or, you are planning for a single Macro

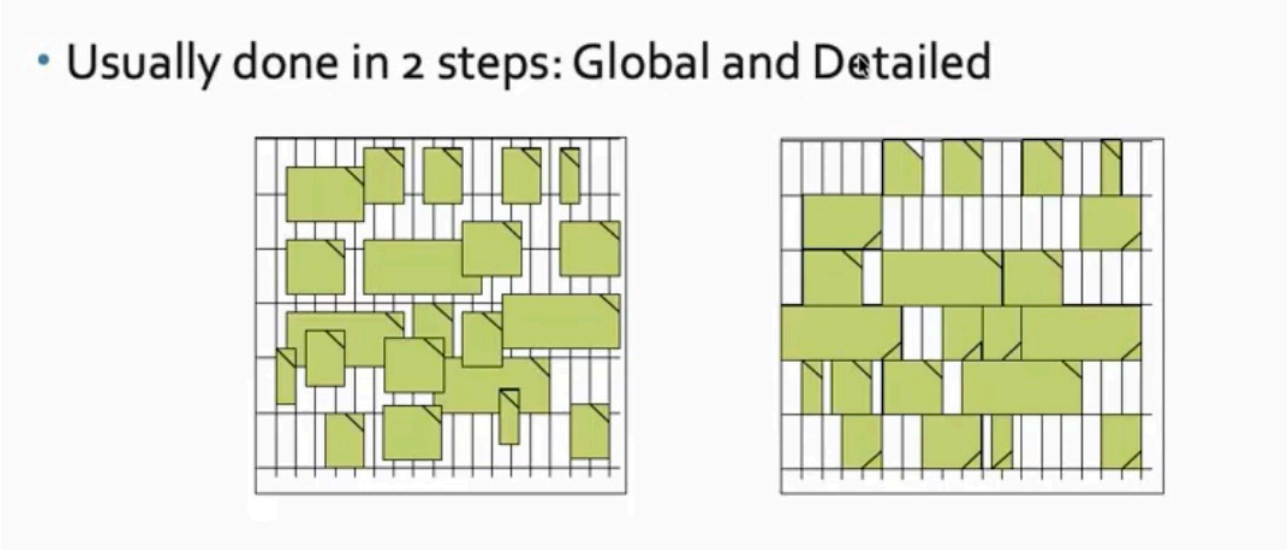


In power planning, the power supply lines that supply power to all components of a chip. There are multiple VSS and GND pins in a chip. These all run in parallel to each other so that they reduce resistance.



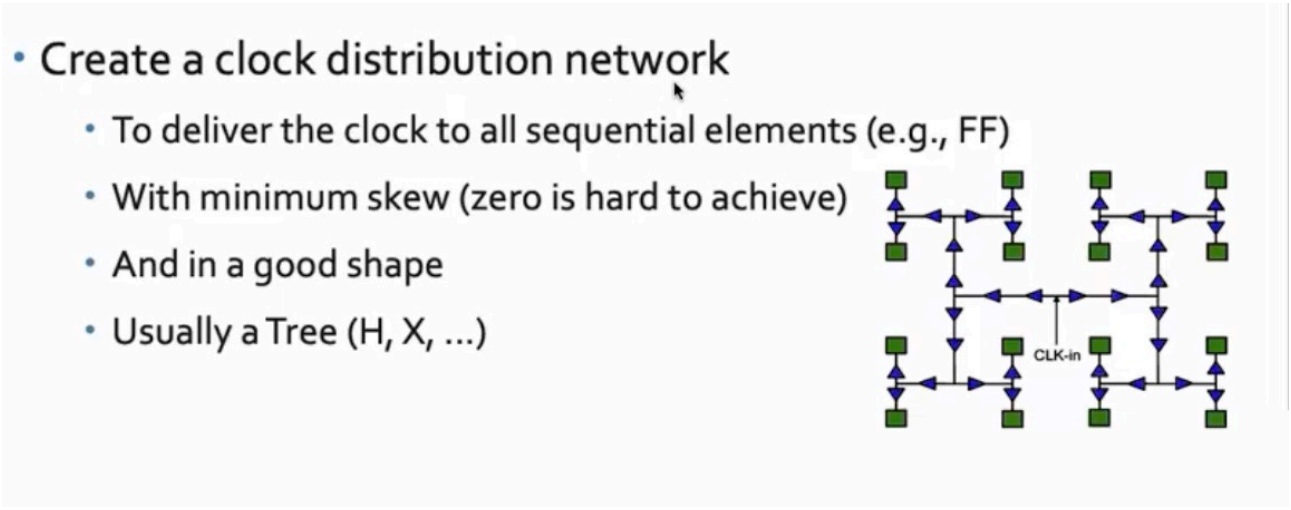
Step 3 :- Placement

Placement defines where each component, logic gates, etc should be placed in a chip. There are two main types of placement. Global and detailed. Global placement is a rough approximation on the placement and Detailed placement does minor changes so that the placement is in its best form.



Step 4 :- CTS (clock tree synthesis)

We need to do CTS to route the clock to all the components of the chip , otherwise the clocks will be off and the chip cannot receive and send signals between its components



Step 5 :- Routing

Routing is the process of detailing the connections between components as detailed by the PDK. In this case, Sky 130 has 6 layers of routing. It becomes especially important when the components are on different layers.

Step 6:- Sign Off

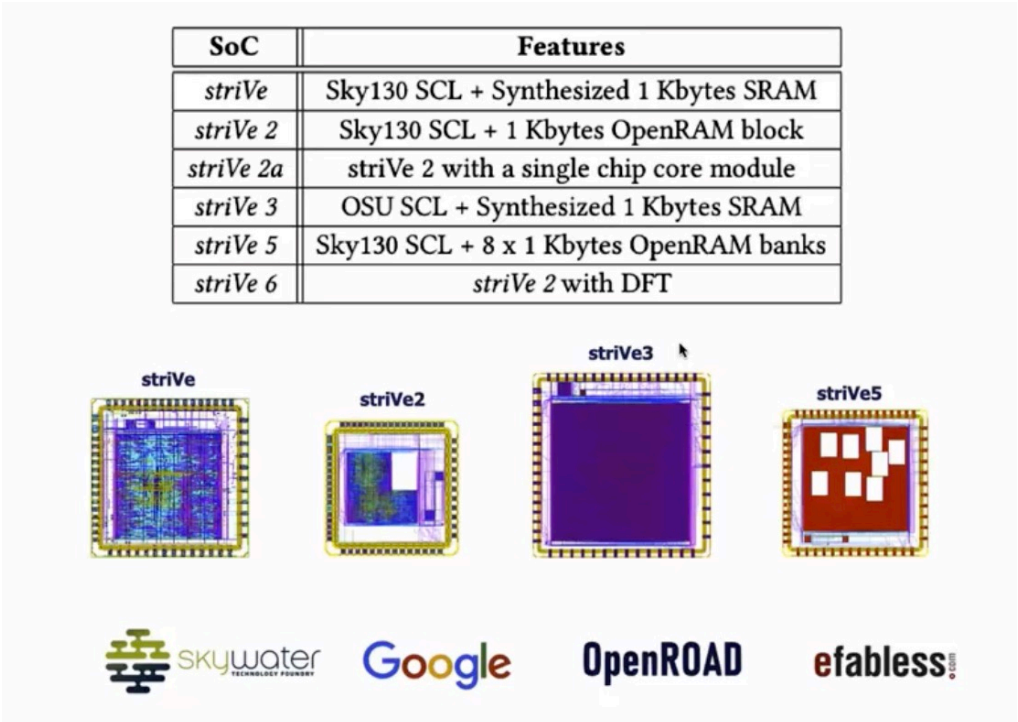
The Sign off step consists of three main parts

> Design Rule Check (DRC) - this makes sure the design complies with manufacturing guidelines and is compatible for fabrication. It aims to detect and correct layout errors so that fabrication defects do not occur.
>
> Layout vs. Schematic (LVS) - here the layout is contrasted against the schematic to ensure consistency. LVS tools extract netlists and compare them for differences, after which the design proceeds to physical design flow
>
> Static Timing Analysis (STA) - it evaluates timing behaviour of a digital circuit to ensure design meets setup and hold time constraints, maximum clock frequency, and other timing requirements

Now, lets learn about OpenLane, the program that we are going to be using on this course. Opensource started as a Open-source Flow for ASIC design.

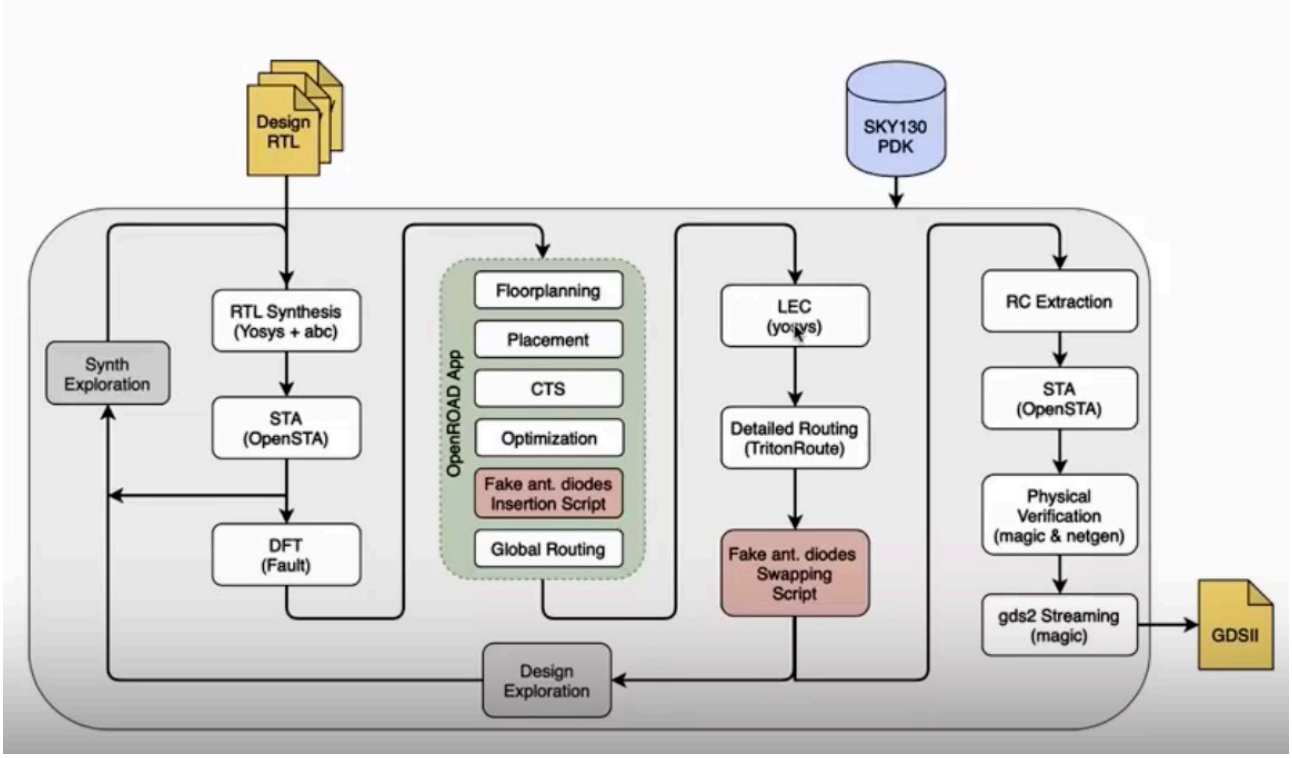
stiVe is a open-everything family of Chips that means , they have Open-source RTL, Open-source EDA , and Open-source PDKs

SoC	Features
striVe	Sky130 SCL + Synthesized 1 Kbytes SRAM
striVe 2	Sky130 SCL + 1 Kbytes OpenRAM block
striVe 2a	striVe 2 with a single chip core module
striVe 3	OSU SCL + Synthesized 1 Kbytes SRAM
striVe 5	Sky130 SCL + 8 x 1 Kbytes OpenRAM banks
striVe 6	*striVe 2* with DFT



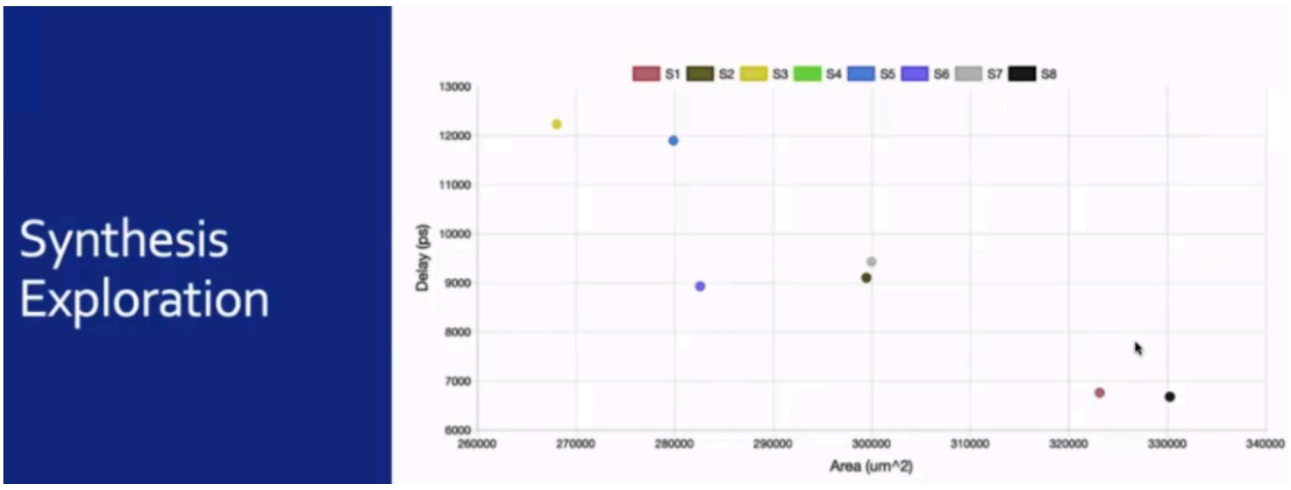
The main goal of OpenLane is to have clean GDSII with no human intervention. Clean means that there can be no LVS violations, DRC violations and Timing violations

OpenLane is tuned to the Skywater 130 nm PDK. OpenLane is generally preferred as it can be used to harden macros and Chips, it also has two modes of operation , which are interactive and autonomous. It has a Very neat little feature known as Design Space Exploration which finds the best connections between components. OpenLane has a large number of prebuilt designs , 43 to be exact.

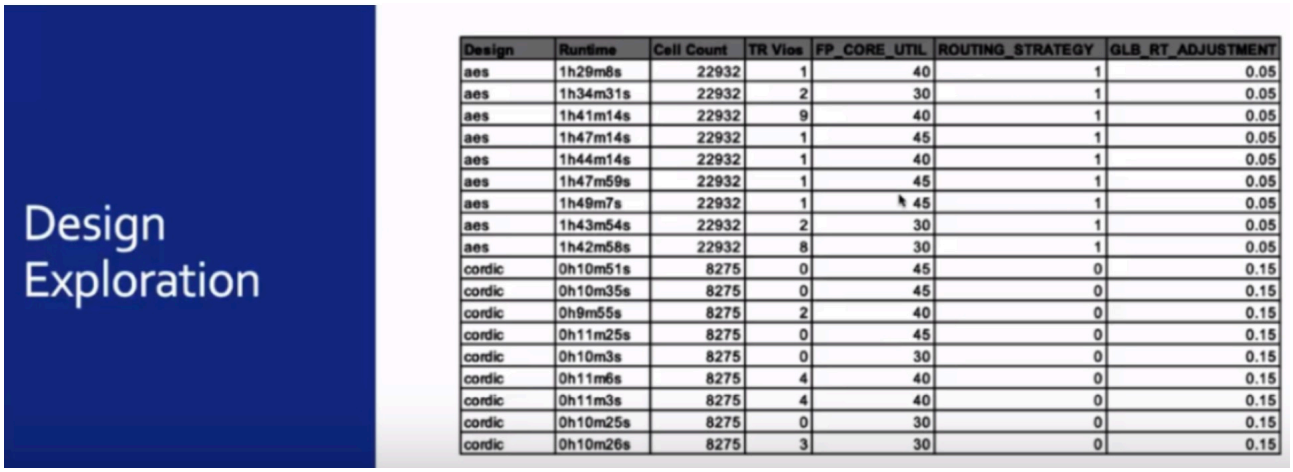


The above diagram is the diagram of ASIC flow In OpenLane. You might notice that it has a lot more components then the flow that I Showed earlier. But, let us start to understand the flow. We know that we have an RTL which we want to convert into a GDSII format, with the help of a PDK. For the first step, that is RTL synthesis, OpenLane Uses an Open-source programs known as Yosys and abc to do the RTL synthesis.

We also use synthesis exploration to generate various synthesis designs and it puts them into a graph . Judging by their Delay time and the Area they take up , we can pick the best design and then use .

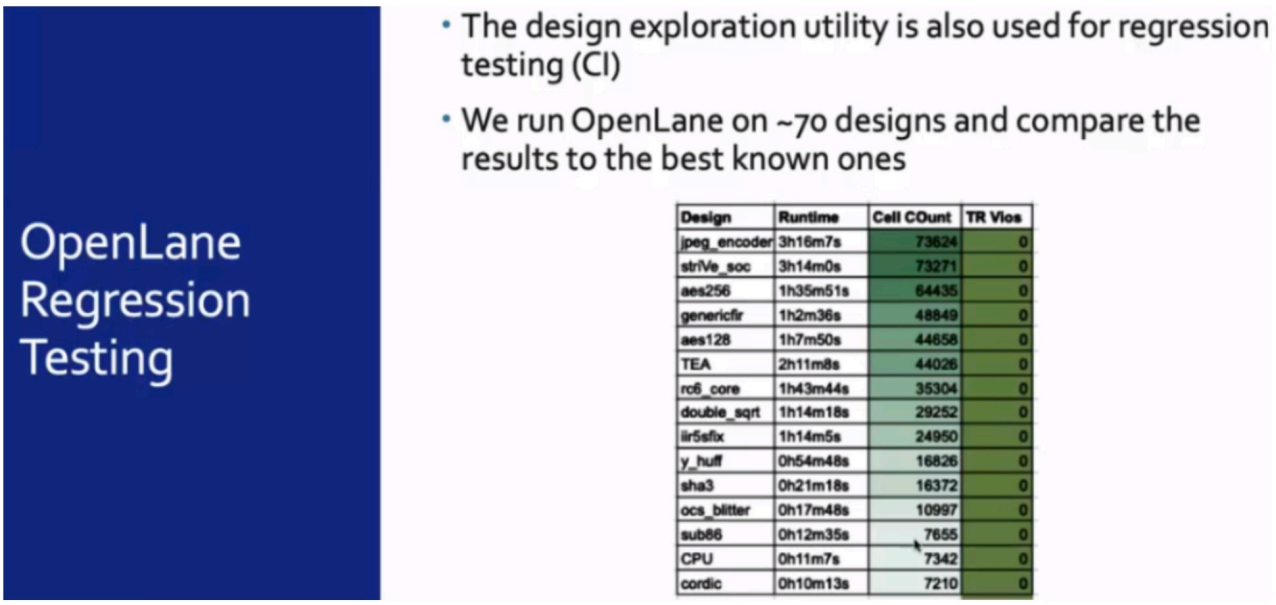


Design Exploration, sweeps through all the variations of a design and gives a report with information and metrics of each design like this



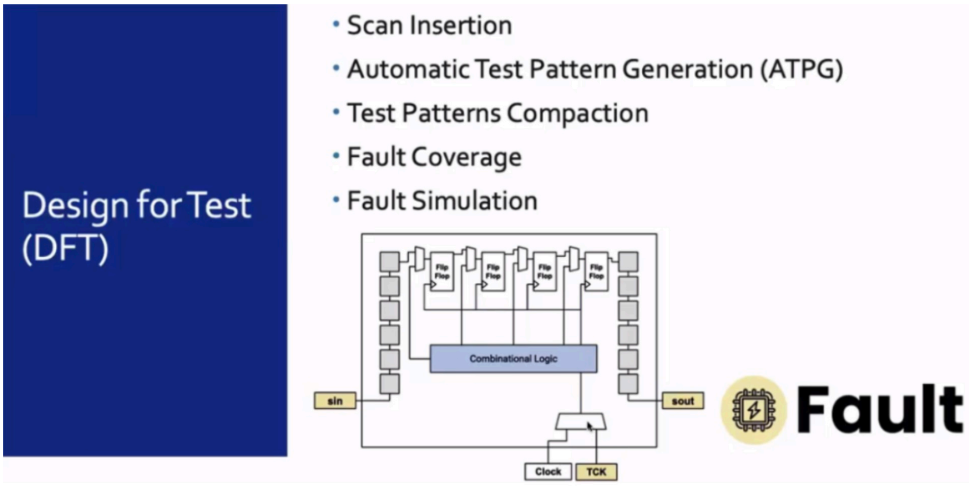
Design	Runtime	Cell Count	TR Vios	FP_CORE_UTIL	ROUTING_STRATEGY	GLB_RT_ADJUSTMENT
aes	1h29m8s	22932	1	40	1	0.05
aes	1h34m31s	22932	2	30	1	0.05
aes	1h41m14s	22932	9	40	1	0.05
aes	1h47m14s	22932	1	45	1	0.05
aes	1h44m14s	22932	1	40	1	0.05
aes	1h47m59s	22932	1	45	1	0.05
aes	1h49m7s	22932	1	45	1	0.05
aes	1h43m54s	22932	2	30	1	0.05
aes	1h42m58s	22932	8	30	1	0.05
cordic	0h10m51s	8275	0	45	0	0.15
cordic	0h10m35s	8275	0	45	0	0.15
cordic	0h9m55s	8275	2	40	0	0.15
cordic	0h11m25s	8275	0	45	0	0.15
cordic	0h10m3s	8275	0	30	0	0.15
cordic	0h11m6s	8275	4	40	0	0.15
cordic	0h11m3s	8275	4	40	0	0.15
cordic	0h10m25s	8275	0	30	0	0.15
cordic	0h10m26s	8275	3	30	0	0.15

The next step is regression testing. Design Exploration can be used for Regression testing.



Design	Runtime	Cell COunt	TR Vios
jpeg_encoder	3h16m7s	73624	0
striVe_soc	3h14m0s	73271	0
aes256	1h35m51s	64435	0
genericfir	1h2m36s	48849	0
aes128	1h7m50s	44658	0
TEA	2h11m8s	44026	0
rc6_core	1h43m44s	35304	0
double_sqrt	1h14m18s	29252	0
iir5sfix	1h14m5s	24950	0
y_huff	0h54m48s	16826	0
sha3	0h21m18s	16372	0
ocs_blitter	0h17m48s	10997	0
sub86	0h12m35s	7655	0
CPU	0h11m7s	7342	0
cordic	0h10m13s	7210	0

After this, we have all the main steps such floor/power planning, placement, CTS, routing . This is all done by the OpenRoad app. After this we have DFT that stands for Design For Test. It is performed by an Open-Souce program known as Fault



The Next steps are:-

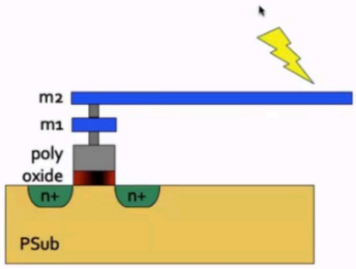
Physical verification (DRC & LVS) - Magic is used for DRC and Magic and Netgen for LVS.

Logic Equivalence Check (LEC) - checks that the physical implementation and the netlist have the same logic. It is performed each time netlist is modified and checks that changing netlist did not change function.

Dealing with Antenna Rules violations

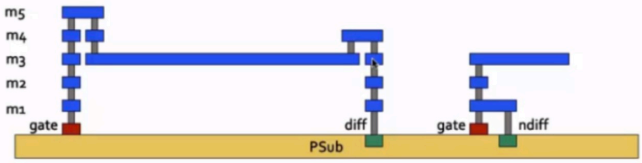
Dealing with Antenna Rules Violations

- When a metal wire segment is fabricated, it can act as an antenna.
  - Reactive ion etching causes charge to accumulate on the wire.
  - Transistor gates can be damaged during fabrication



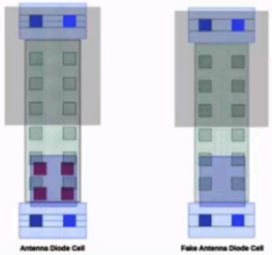
Dealing with Antenna Rules Violations

- Two solutions:
  - Bridging attaches a higher layer intermediary
    - Requires Router awareness (not there yet!)
  - Add antenna diode cell to leak away charges
    - Antenna diodes are provided by the SCL

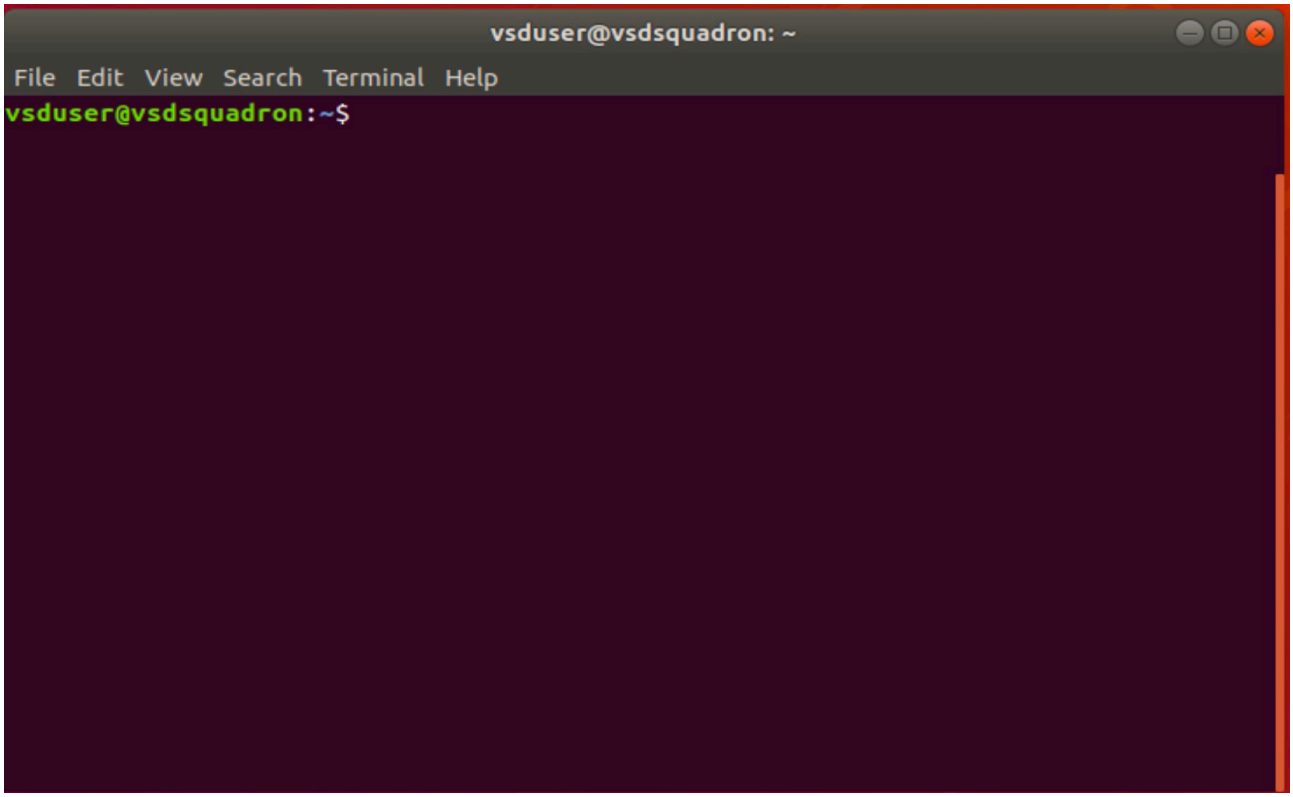


Dealing with Antenna Rules Violations

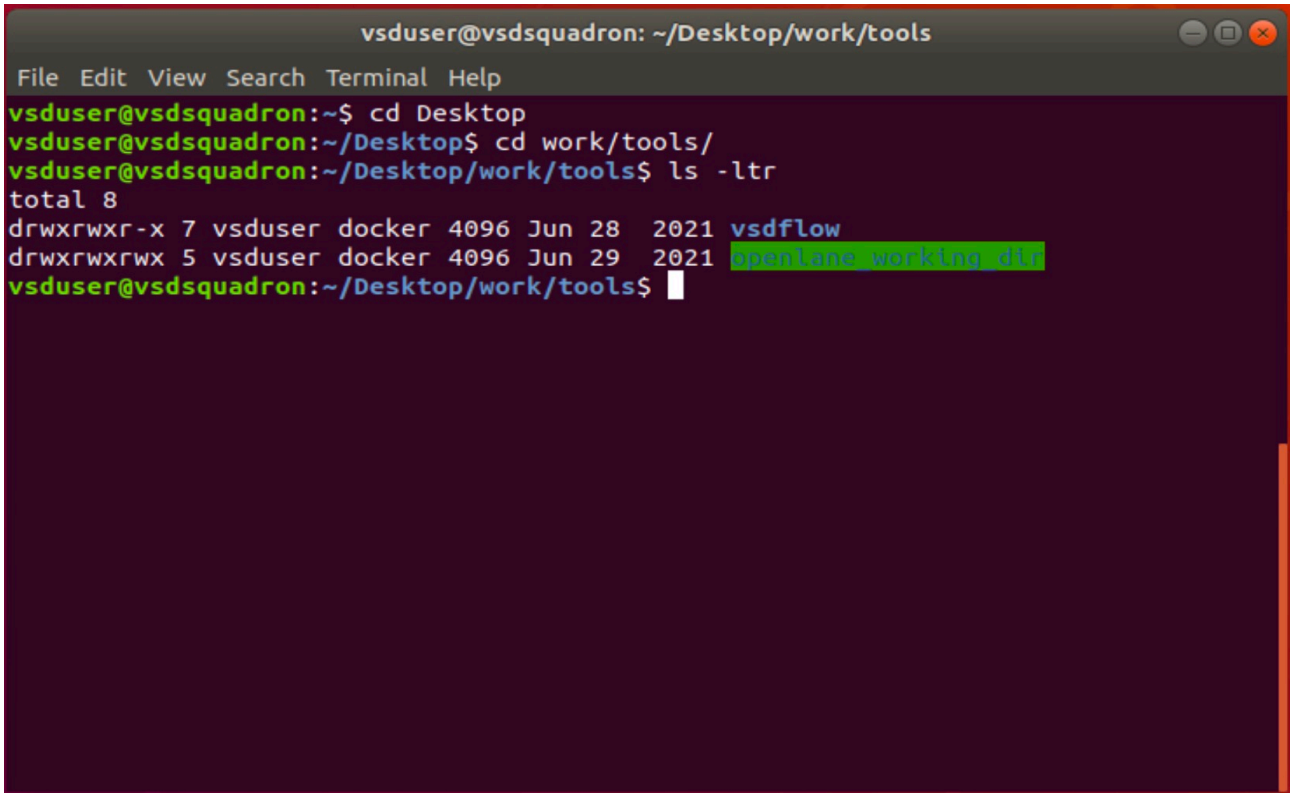
- We took a preventive approach
  - Add a Fake Antenna Diode next to every cell input after placement
  - Run the Antenna Checker (Magic) on the routed layout
  - If the checker reports a violation on the cell input pin, replace the Fake Diode cell by a real one



Now, lets get started with using OpenLane. Now, as we have already seen, OpenLane is not really one tool. It is a flow, in which multiple Open-Source programs are used. Now, lets hop into the ubuntu terminal to locate the files and directories of OpenLane



Starting with a blank screen we enter the command ‘cd work/tools/‘ to enter into the tools part of the work directory, and then we use the “ls-ltr” command to list out all the things in the file/ directory.



Now, lets go into the openlane_working_dir directory

```
drwxrwxr-x 7 vsduser docker 4096 Jun 28  2021 vsdflow
drwxrwxrwx 5 vsduser docker 4096 Jun 29  2021 openlane_working_dir
vsduser@vsdsquadron:~/Desktop/work/tools$ cd openlane_working_dir
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir$ ls -ltr
total 12
drwxr-xr-x  5 vsduser docker 4096 Jun 28  2021 pdks
drwxr-xr-x 10 vsduser docker 4096 Jun 29  2021 openlane_old
drwxr-xr-x 10 vsduser docker 4096 May 20  2023 openlane
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir$ cd pdks
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks$ ls -ltr
total 12
drwxr-xr-x 9 vsduser docker 4096 Jun 28  2021 skywater-pdk
drwxr-xr-x 8 vsduser docker 4096 Jun 28  2021 open_pdks
drwxr-xr-x 5 vsduser docker 4096 Jun 28  2021 sky130A
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks$
```

Now, lets explore the sky130A directory in the PDKs directory as we are using the Sky130 nm PDK for this project. We can see that it has two more directories inside - libs.ref and libs.tech

```
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir$ cd pdks
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks$ ls -ltr
total 12
drwxr-xr-x 9 vsduser docker 4096 Jun 28  2021 skywater-pdk
drwxr-xr-x 8 vsduser docker 4096 Jun 28  2021 open_pdks
drwxr-xr-x 5 vsduser docker 4096 Jun 28  2021 sky130A
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks$ cd sky130A
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A$ ls -
ltr
total 12
drwxr-xr-x 11 vsduser docker 4096 Jun 28  2021 libs.tech
drwxr-xr-x 14 vsduser docker 4096 Jun 28  2021 libs.ref
-rwxr-xr-x  1 vsduser docker  170 Jun 28  2021 SOURCES
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A$
```

lets explore the libs.tech directory. Upon running the ls -ltr command, we can see that, it contains all the tech parts of the flow that we need

```
vsduser@vsdsquadron: ~/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.tech
File  Edit  View  Search  Terminal  Help
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A$ ls -
ltr
total 12
drwxr-xr-x 11 vsduser docker 4096 Jun 28  2021 libs.tech
drwxr-xr-x 14 vsduser docker 4096 Jun 28  2021 libs.ref
-rwxr-xr-x  1 vsduser docker  170 Jun 28  2021 SOURCES
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A$ cd l
ibs.tech
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.
tech$ ls -ltr
total 36
drwxr-xr-x  9 vsduser docker 4096 Jun 28  2021 xschem
drwxr-xr-x  2 vsduser docker 4096 Jun 28  2021 xcircuit
drwxr-xr-x  2 vsduser docker 4096 Jun 28  2021 qflow
drwxr-xr-x 10 vsduser docker 4096 Jun 28  2021 openlane
drwxr-xr-x  2 vsduser docker 4096 Jun 28  2021 ngspice
drwxr-xr-x  2 vsduser docker 4096 Jun 28  2021 netgen
drwxr-xr-x  4 vsduser docker 4096 Jun 28  2021 magic
drwxr-xr-x  2 vsduser docker 4096 Jun 28  2021 klayout
drwxr-xr-x  2 vsduser docker 4096 Jun 28  2021 irsim
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.
tech$
```

Now, lets go back and explore the libs.ref directory

```
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.
tech$ cd ../
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A$ cd l
ibs.ref
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.
ref$ ls -ltr
total 48
drwxr-xr-x 10 vsduser docker 4096 Jun 28  2021 sky130_osu_sc_t18
drwxr-xr-x 12 vsduser docker 4096 Jun 28  2021 sky130_fd_sc_ms
drwxr-xr-x 12 vsduser docker 4096 Jun 28  2021 sky130_fd_sc_ls
drwxr-xr-x 12 vsduser docker 4096 Jun 28  2021 sky130_fd_sc_hs
drwxr-xr-x 12 vsduser docker 4096 Jun 28  2021 sky130_fd_sc_hdll
drwxr-xr-x  8 vsduser docker 4096 Jun 28  2021 sky130_fd_pr
drwxr-xr-x  9 vsduser docker 4096 Jun 28  2021 sky130_sram_macros
drwxr-xr-x 12 vsduser docker 4096 Jun 28  2021 sky130_fd_sc_hvl
drwxr-xr-x 11 vsduser docker 4096 Jun 28  2021 sky130_fd_io
drwxr-xr-x  4 vsduser docker 4096 Jun 28  2021 sky130_ml_xx_hd
drwxr-xr-x 12 vsduser docker 4096 Jun 28  2021 sky130_fd_sc_lp
drwxr-xr-x 12 vsduser docker 4096 Jun 28  2021 sky130_fd_sc_hd
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.
ref$
```

Now, we can see that there are many versions of the Sky130A pdk that we are using. We going to use sky130_sc_hd for this project. Upon going into this file, we can see all the files that is in the sky130_sc_hd file

```
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.ref$ cd sky130_fd_sc_hd
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.ref/sky130_fd_sc_hd$ ls -ltr
total 88
drwxr-xr-x 2 vsduser docker  4096 Jun 28  2021 verilog
drwxr-xr-x 2 vsduser docker  4096 Jun 28  2021 techlef
drwxr-xr-x 2 vsduser docker  4096 Jun 28  2021 spice
drwxr-xr-x 2 vsduser docker 28672 Jun 28  2021 maglef
drwxr-xr-x 2 vsduser docker 28672 Jun 28  2021 mag
drwxr-xr-x 2 vsduser docker  4096 Jun 28  2021 lib
drwxr-xr-x 2 vsduser docker  4096 Jun 28  2021 lef
drwxr-xr-x 2 vsduser docker  4096 Jun 28  2021 gds
drwxr-xr-x 2 vsduser docker  4096 Jun 28  2021 doc
drwxr-xr-x 2 vsduser docker  4096 Jun 28  2021 cdl
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.ref/sky130_fd_sc_hd$
```

So now that we have seen all about the pdks directory, lets go into the openlane directory and invoke the tool. We can use the docker command to invoke the tool.

```
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir$ cd openlane
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane$ docker
bash-4.2$ pwd
/openLANE_flow
bash-4.2$ ls -ltr
total 136
drwxr-xr-x 15 1000  997  4096 Jun 29  2021 scripts
-rw-r--r--  1 1000  997 20787 Jun 29  2021 run_designs.py
-rw-r--r--  1 1000  997  7898 Jun 29  2021 report_generation_wrapper.py
drwxr-xr-x  3 1000  997  4096 Jun 29  2021 regression_results
-rwxr-xr-x  1 1000  997  6519 Jun 29  2021 flow.tcl
drwxr-xr-x  5 1000  997  4096 Jun 29  2021 docs
drwxr-xr-x  5 1000  997  4096 Jun 29  2021 docker_build
drwxr-xr-x 44 1000  997  4096 Jun 29  2021 designs
drwxr-xr-x  2 1000  997  4096 Jun 29  2021 configuration
-rw-r--r--  1 1000  997  5514 Jun 29  2021 conf.py
-rwxr-xr-x  1 1000  997   966 Jun 29  2021 clean_runs.tcl
-rw-r--r--  1 1000  997 25509 Jun 29  2021 README.md
-rw-r--r--  1 1000  997  7273 Jun 29  2021 Makefile
-rw-r--r--  1 1000  997 11350 Jun 29  2021 LICENSE
-rw-r--r--  1 1000  997  1285 Jun 29  2021 CONTRIBUTING.md
-rw-r--r--  1 1000  997   709 Jun 29  2021 AUTHORS.md
-rw-r--r--  1 1000 1000   963 May 19  2023 default.cvcrc
bash-4.2$
```

Now, lets open an interactive session in openlane

```
drwxr-xr-x 15 1000  997  4096 Jun 29  2021 scripts
-rw-r--r--  1 1000  997 20787 Jun 29  2021 run_designs.py
-rw-r--r--  1 1000  997  7898 Jun 29  2021 report_generation_wrapper.py
drwxr-xr-x  3 1000  997  4096 Jun 29  2021 regression_results
-rwxr-xr-x  1 1000  997  6519 Jun 29  2021 flow.tcl
drwxr-xr-x  5 1000  997  4096 Jun 29  2021 docs
drwxr-xr-x  5 1000  997  4096 Jun 29  2021 docker_build
drwxr-xr-x 44 1000  997  4096 Jun 29  2021 designs
drwxr-xr-x  2 1000  997  4096 Jun 29  2021 configuration
-rw-r--r--  1 1000  997  5514 Jun 29  2021 conf.py
-rwxr-xr-x  1 1000  997   966 Jun 29  2021 clean_runs.tcl
-rw-r--r--  1 1000  997 25509 Jun 29  2021 README.md
-rw-r--r--  1 1000  997  7273 Jun 29  2021 Makefile
-rw-r--r--  1 1000  997 11350 Jun 29  2021 LICENSE
-rw-r--r--  1 1000  997  1285 Jun 29  2021 CONTRIBUTING.md
-rw-r--r--  1 1000  997   709 Jun 29  2021 AUTHORS.md
-rw-r--r--  1 1000 1000   963 May 19  2023 default.cvcrc
bash-4.2$ ./flow.tcl -interactive
[INFO]:
```

```
[INFO]: Version: v0.21
[INFO]: Running interactively
```

Now, lets import our designs. Where are getting our designs from? We are getting it from the design directory in openlane. There are a variety of designs in openlane but now, we are going to use picorv32a for this flow. Going into into this file, we can see that we have src ,our pdks and config.tcl

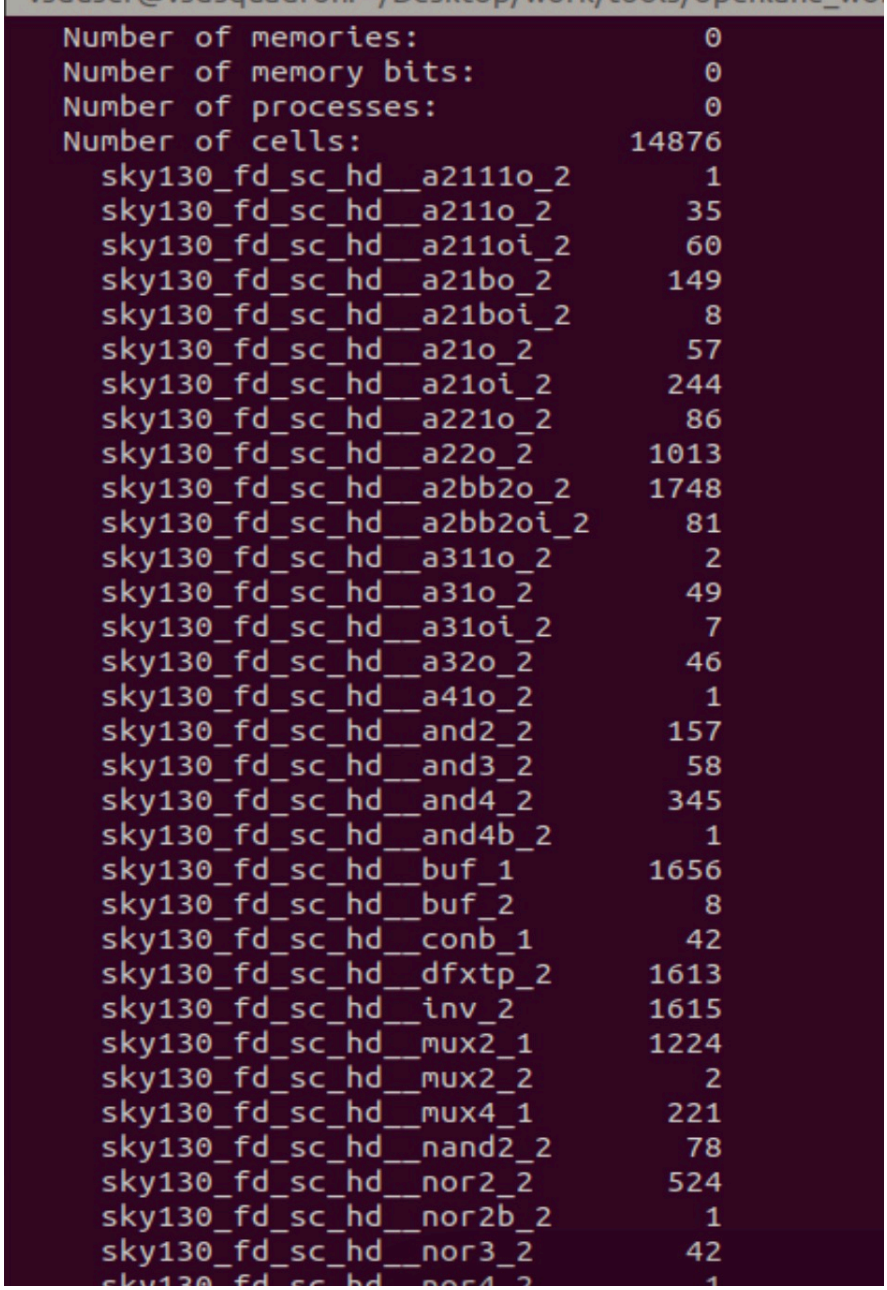
```
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 usb
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 synth_ram
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 sound
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 sha512
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 sha3
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 salsa20
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 s44
-rw-r--r-- 1 vsduser docker 10029 Jun 29  2021 README.md
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 PPU
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 point_scalar_mult
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 point_add
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 ocs_blitter
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 md5
drwxr-xr-x 4 vsduser docker  4096 Jun 29  2021 manual_macro_placement_test
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 ldpcenc
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 ldpc_decoder_802_3an
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 jpeg_encoder
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 inverter
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 genericfir
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 digital_pll_sky130_fd_sc_hd
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 des3
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 des
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 cic_decimator
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 chacha
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 BM64
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 blabla
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 APU
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 aes_core
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 aes_cipher
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 aes256
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 aes192
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 aes128
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 aes
drwxr-xr-x 3 vsduser docker  4096 Jun 29  2021 151
drwxr-xr-x 4 vsduser docker  4096 Jun 29  2021 spm
drwxr-xr-x 4 vsduser docker  4096 Mar 29  2024 picorv32a
```

```
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane/designs$ cd picorv32a
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a$ ls -ltr
total 32
drwxr-xr-x 2 vsduser docker  4096 Jun 29  2021 src
-rw-r--r-- 1 vsduser docker   209 Jun 29  2021 sky130A_sky130_fd_sc_ms_config.tcl
-rw-r--r-- 1 vsduser docker   209 Jun 29  2021 sky130A_sky130_fd_sc_ls_config.tcl
-rw-r--r-- 1 vsduser docker   209 Jun 29  2021 sky130A_sky130_fd_sc_hs_config.tcl
-rw-r--r-- 1 vsduser docker   209 Jun 29  2021 sky130A_sky130_fd_sc_hdll_config.tcl
-rwxr-xr-x 1 vsduser docker   209 Jun 29  2021 sky130A_sky130_fd_sc_hd_config.tcl
-rwxr-xr-x 1 vsduser docker   444 Jun 29  2021 config.tcl
drwxr-xr-x 3 vsduser vsduser 4096 Mar 29  2024 runs
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a$
```

So now that we have looked at our design files, we can start preparing the design file for the flow

```
[INFO]: Version: v0.21
[INFO]: Running interactively
% package require openlane 0.9
0.9
% prep -design picorv32a
[INFO]: Using design configuration at /openLANE_flow/designs/picorv32a/config.tcl
[INFO]: Sourcing Configurations from /openLANE_flow/designs/picorv32a/config.tcl
[INFO]: PDKs root directory: /home/vsduser/Desktop/work/tools/openlane_working_dir/pdks
[INFO]: PDK: sky130A
[INFO]: Setting PDKPATH to /home/vsduser/Desktop/work/tools/openlane_working_dir/pdks/sky130A
[INFO]: Standard Cell Library: sky130_fd_sc_hd
[INFO]: Sourcing Configurations from /openLANE_flow/designs/picorv32a/config.tcl
[INFO]: Current run directory is /openLANE_flow/designs/picorv32a/runs/27-01_09-57
[INFO]: Preparing LEF Files
[INFO]: Extracting the number of available metal layers from /home/vsduser/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.
ref/sky130_fd_sc_hd/techlef/sky130_fd_sc_hd.tlef
[INFO]: The number of available metal layers is 6
[INFO]: The available metal layers are li1 met1 met2 met3 met4 met5
[INFO]: Merging LEF Files...
mergeLef.py : Merging LEFs
sky130_fd_sc_hd.lef: SITEs matched found: 0
sky130_fd_sc_hd.lef: MACROs matched found: 437
sky130_ef_sc_hd__fill_12.lef: SITEs matched found: 0
sky130_ef_sc_hd__fill_12.lef: MACROs matched found: 1
sky130_ef_sc_hd__decap_12.lef: SITEs matched found: 0
sky130_ef_sc_hd__decap_12.lef: MACROs matched found: 1
sky130_ef_sc_hd__fakediode_2.lef: SITEs matched found: 0
sky130_ef_sc_hd__fakediode_2.lef: MACROs matched found: 1
mergeLef.py : Merging LEFs complete
[INFO]: Trimming Liberty...
[INFO]: Generating Exclude List...
[INFO]: Storing configs into config.tcl ...
[INFO]: Preparation complete
```

so now, we can run the commands run_synthesis to synthesise the RTL design. After the command is executed in the end, we get this.(picture in next page)

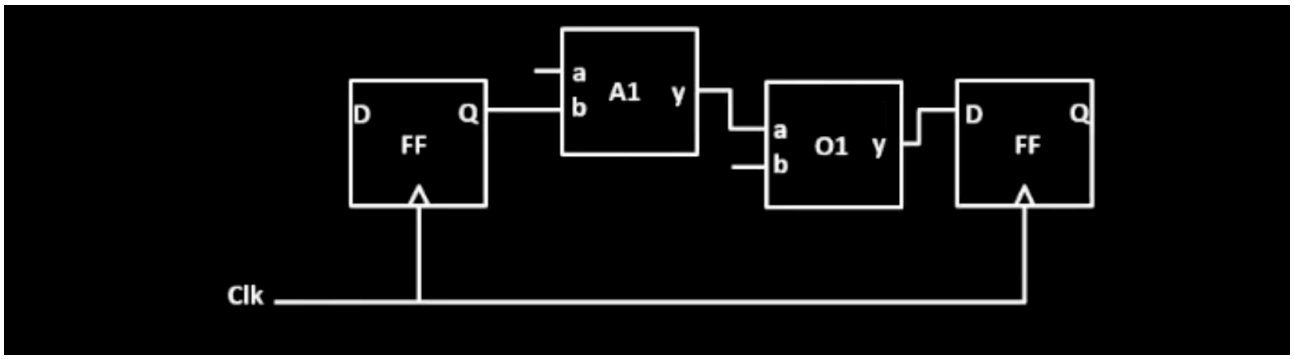


```
Number of memories:               0
Number of memory bits:            0
Number of processes:              0
Number of cells:              14876
  sky130_fd_sc_hd__a2111o_2       1
  sky130_fd_sc_hd__a211o_2       35
  sky130_fd_sc_hd__a211oi_2      60
  sky130_fd_sc_hd__a21bo_2      149
  sky130_fd_sc_hd__a21boi_2       8
  sky130_fd_sc_hd__a21o_2        57
  sky130_fd_sc_hd__a21oi_2      244
  sky130_fd_sc_hd__a221o_2       86
  sky130_fd_sc_hd__a22o_2      1013
  sky130_fd_sc_hd__a2bb2o_2    1748
  sky130_fd_sc_hd__a2bb2oi_2     81
  sky130_fd_sc_hd__a311o_2        2
  sky130_fd_sc_hd__a31o_2        49
  sky130_fd_sc_hd__a31oi_2        7
  sky130_fd_sc_hd__a32o_2        46
  sky130_fd_sc_hd__a41o_2         1
  sky130_fd_sc_hd__and2_2       157
  sky130_fd_sc_hd__and3_2        58
  sky130_fd_sc_hd__and4_2       345
  sky130_fd_sc_hd__and4b_2        1
  sky130_fd_sc_hd__buf_1       1656
  sky130_fd_sc_hd__buf_2          8
  sky130_fd_sc_hd__conb_1        42
  sky130_fd_sc_hd__dfxtp_2     1613
  sky130_fd_sc_hd__inv_2       1615
  sky130_fd_sc_hd__mux2_1      1224
  sky130_fd_sc_hd__mux2_2         2
  sky130_fd_sc_hd__mux4_1       221
  sky130_fd_sc_hd__nand2_2       78
  sky130_fd_sc_hd__nor2_2       524
  sky130_fd_sc_hd__nor2b_2        1
  sky130_fd_sc_hd__nor3_2        42
```

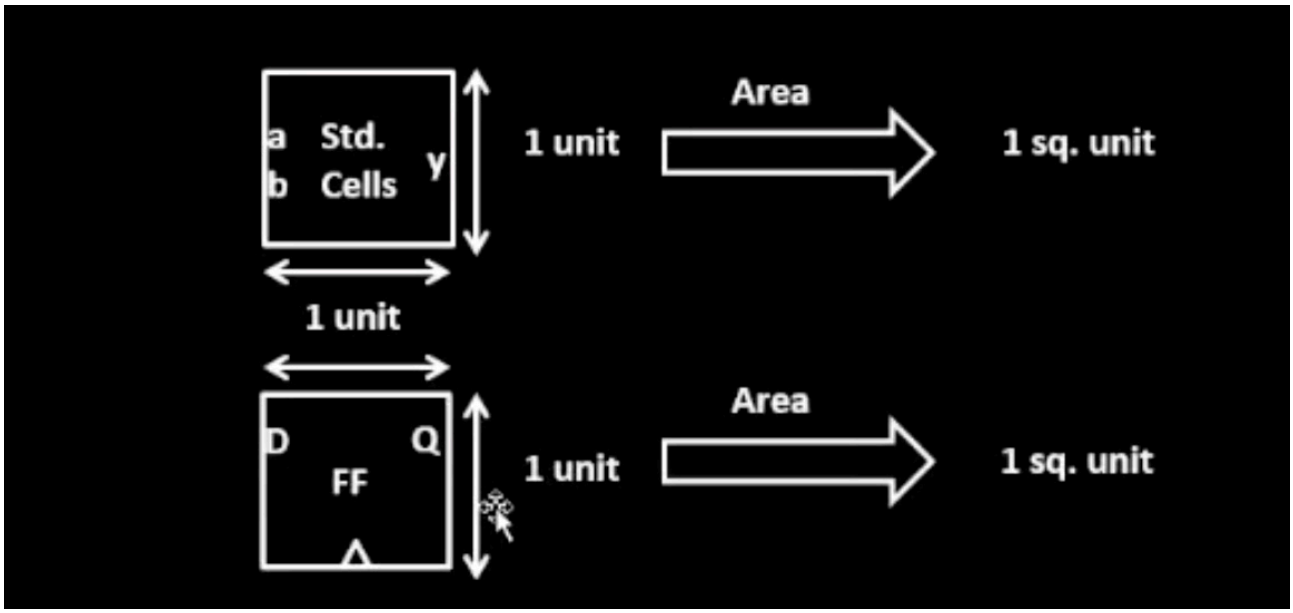
So now, our first task is to calculate the clock ratio, which the number of filpflops / the no.of cells. That is , 1613/14876 = 0.10842. an percentage, it is 10.842%. We can also see the results of the synthesis in the picorv32a folder

Sky130 - Day2

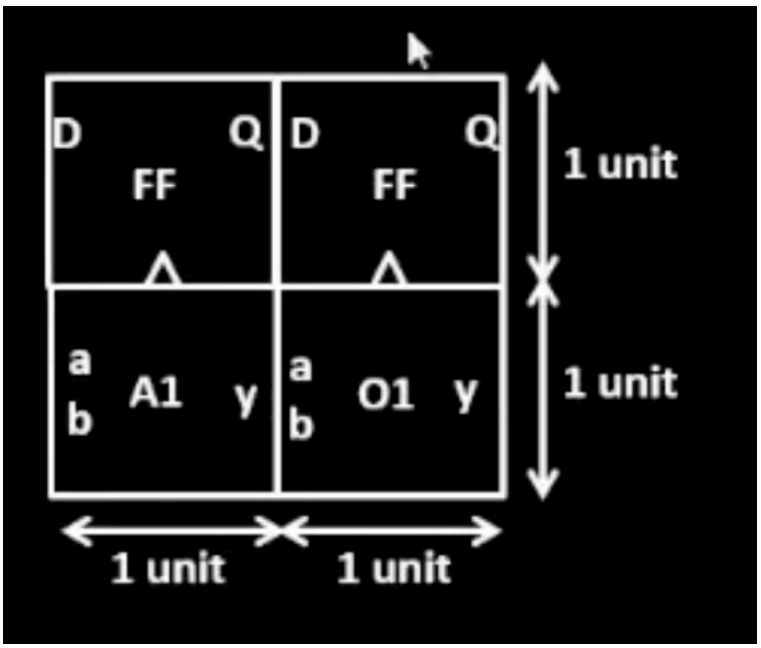
The first step in physical design is to define the width and height of the core and die : Beginning with a very simple netlist, that can extrapolated later we will first draw a basic diagram in the form of symbols that we will later convert into physical designs. We will take each cell (gates, specific cell like flip flop) and give it a standard (although rough for now) dimensions. As an example here, each unit will be 1 unit x 1 unit - i.e. 1 sq. unit in size, and since there are 4 gates/flip-flops here, the total size of the silicon wafer will 4 sq. units.



This is our netlist. Suppose each standard cell has an area of 1x1 unit , which is 1 sq.unit

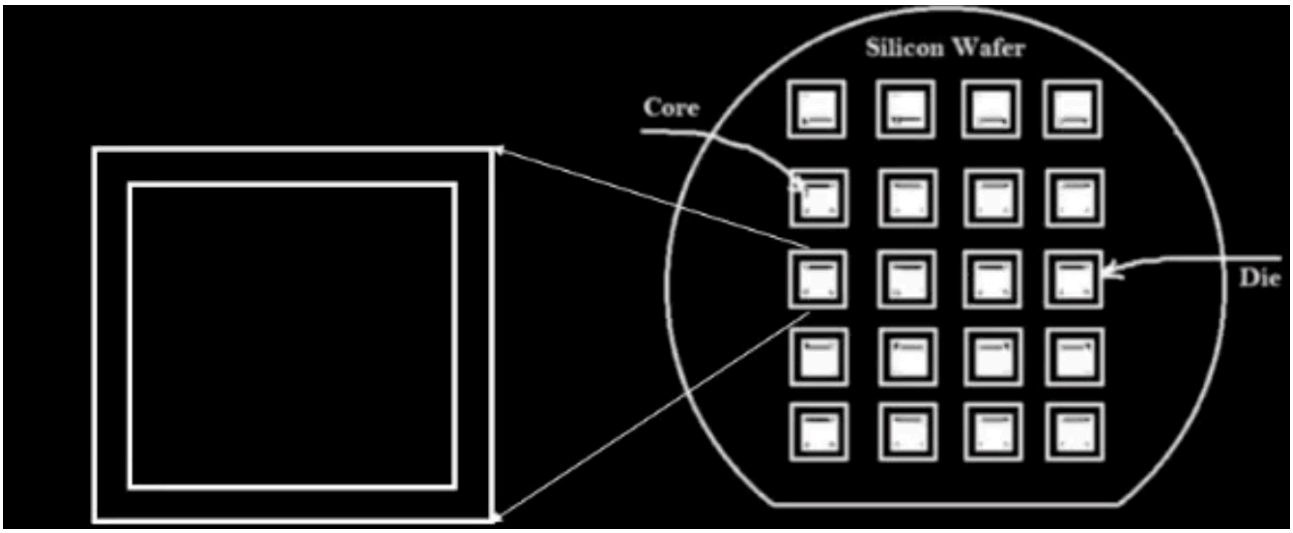


note here that we are not showing the wires. So lets calculate the area by pouting all of these into a plate (here a silicon wafer) and we get this

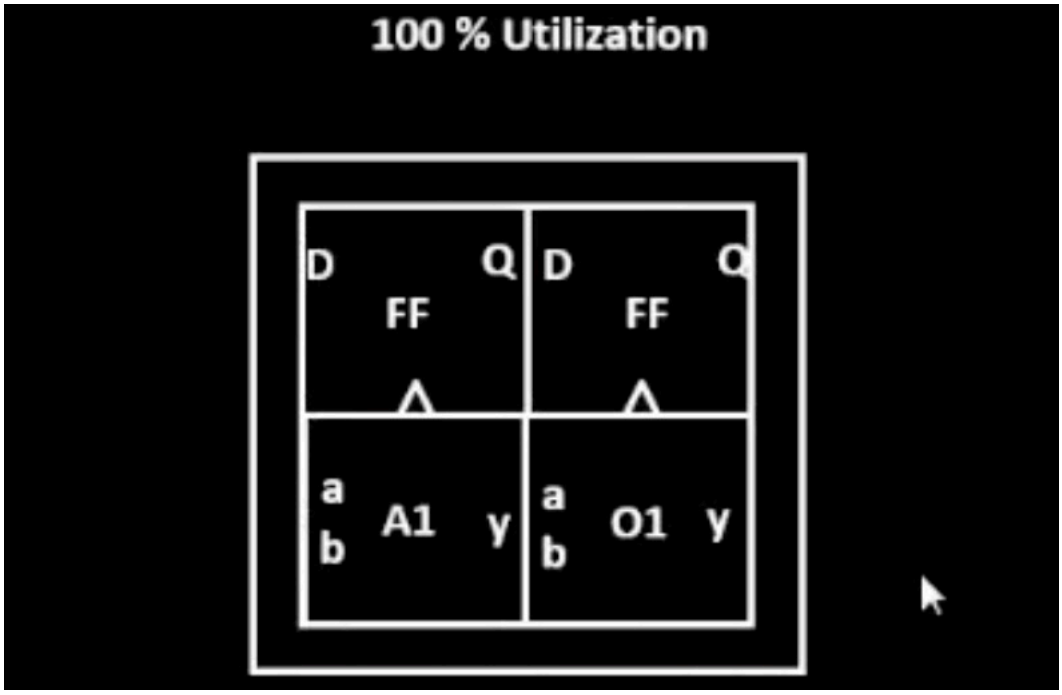


So here, we have kept all the gates and flops in one place. The result has an area of 4 sq.units as the length is 2 units and the breadth is 2 units.so, what is a core and a die?

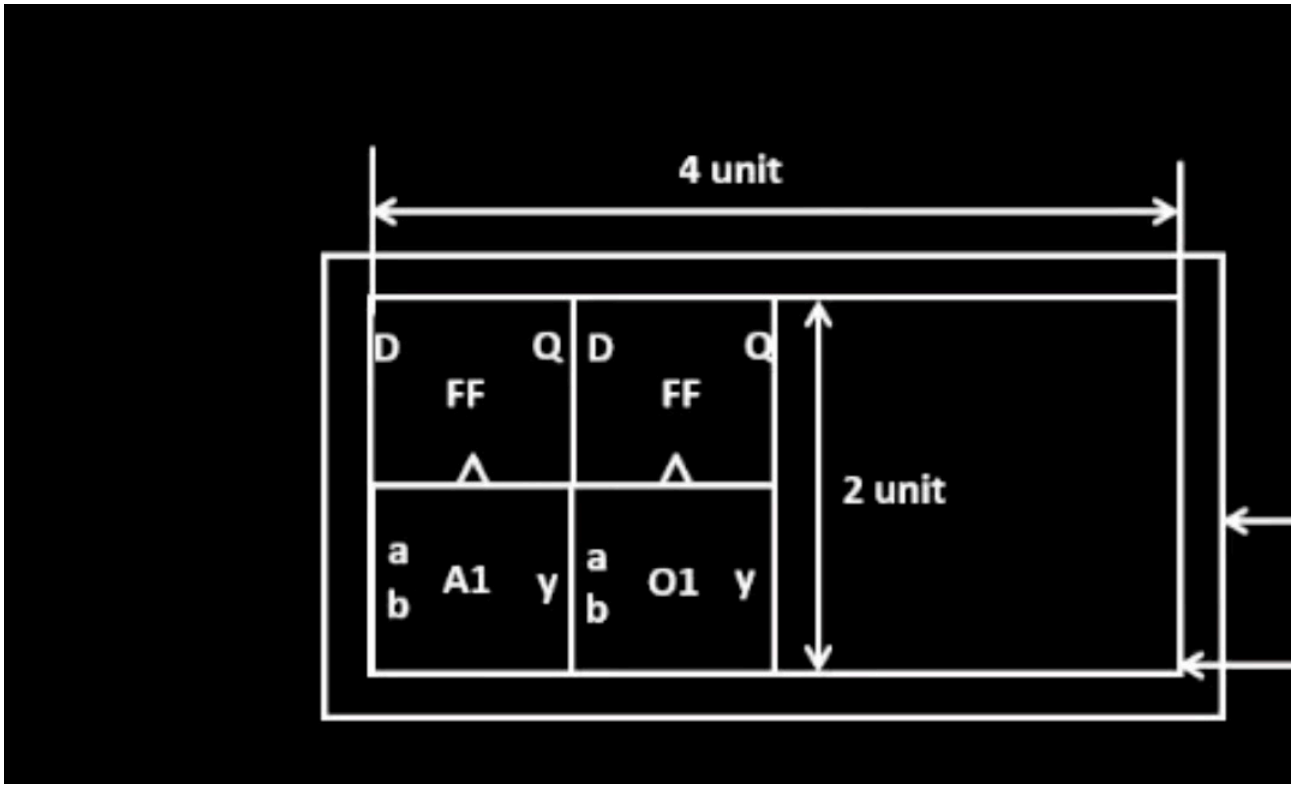
In the diagram below, we can see that a silicon wafer consists of many compartments. Each having a core and a die. A die is a piece of semiconductor water on which the core rests so that the chip has more throughput. The core is where all the logic of the chip is stored.



Now lets place all the logic cells into the core as follows, so that we can define the proper dimensions of core and die



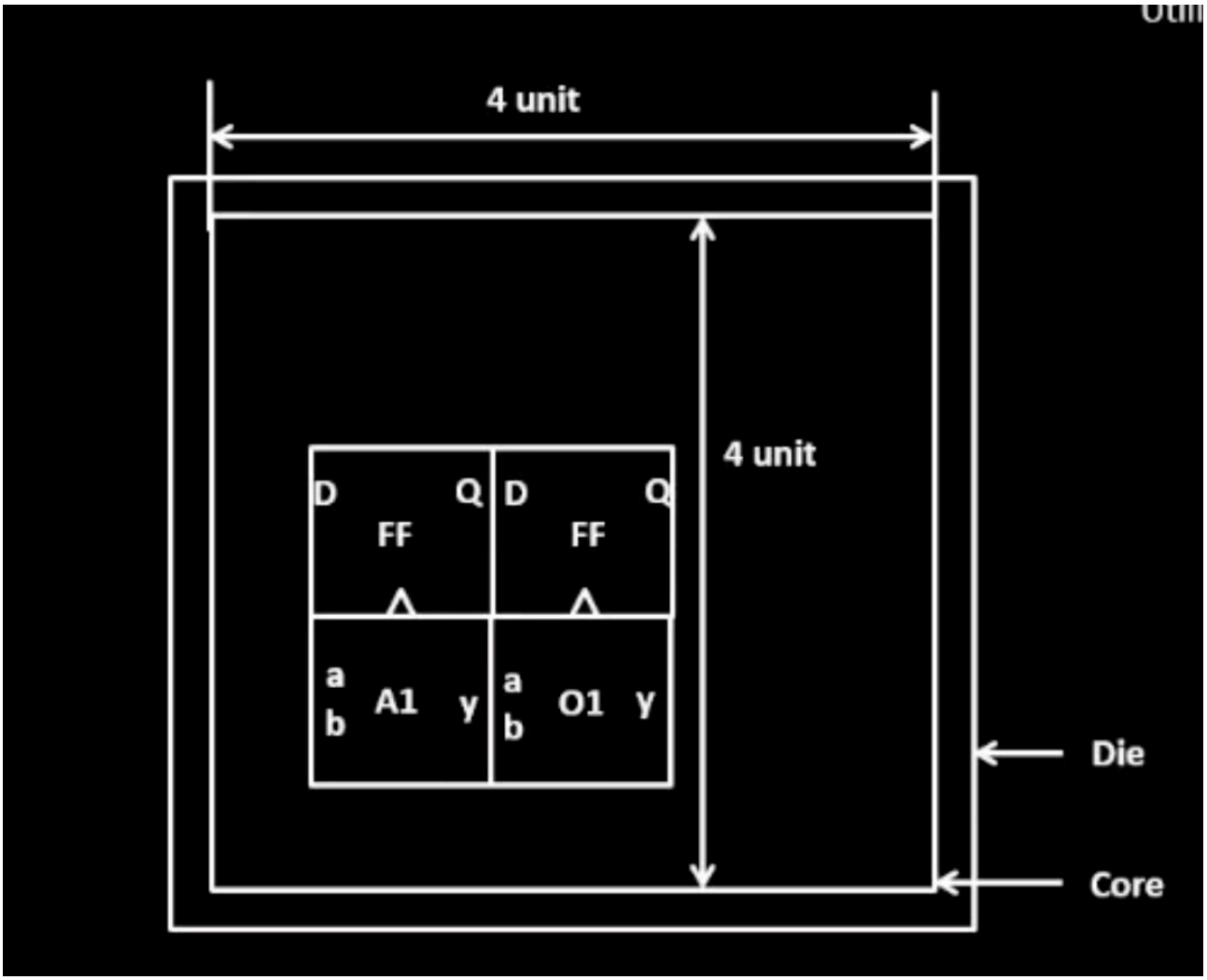
So, now it looks like we are utilising all the space of the core. The utilisation factor determines how much of the space of the core we have used. It is area of core/area of netlist. So, 4/4 =1. We never go for a utilisation factor of 1, we try to go for a utilisation factor of 0.5 or 0.6. we have heard of aspect ratio, which is height/width. For this module, the aspect ratio is 2unit/2unit =1



Now, we have the same logic, but we have put it in a bigger core. Now lets calculate the utilisation factor and aspect ratio for this module.

1. Utilisation ratio = (2x2)sq units/(4x2) sq.units = 4/8 =0.5
2. Aspect ratio = 2/4=0.5

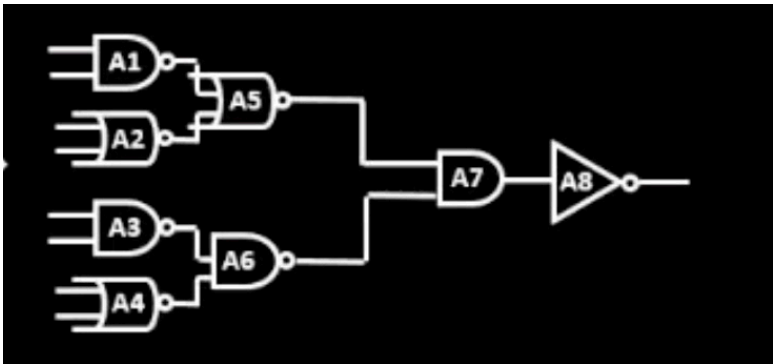
Let us look at another module



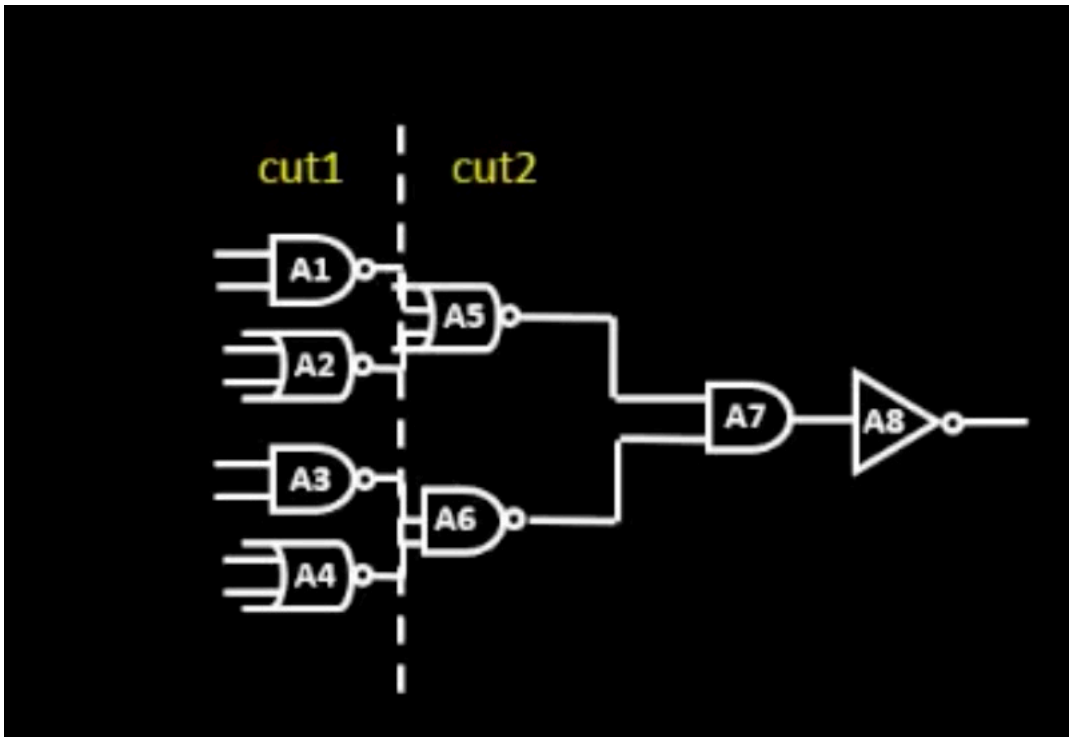
Now, lets calculate the utilisation factor and aspect ratio for the following.

1. Utilisation factor = (2x2) sq.units/(4x4)sq.units =4/16 = 0.25
2. Aspect ratio = 4 units/ 4 units = 1

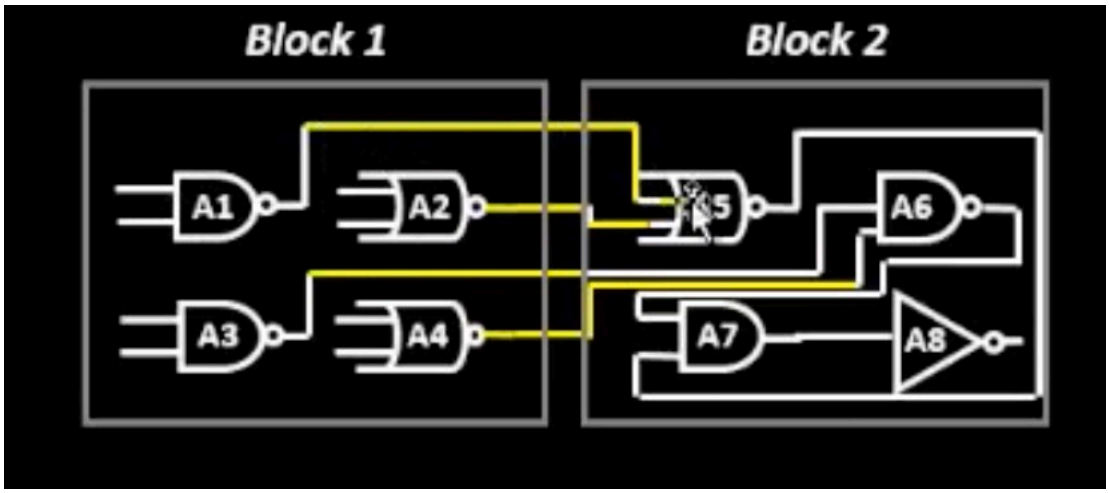
The next step in the physical design is to define the location of preplanned cells. So for understanding this, let us take an example of a combinational logic. This combinational logic does some function within the chip and is a part of the netlist.



So ,now we are going to split the circuit into two just like so

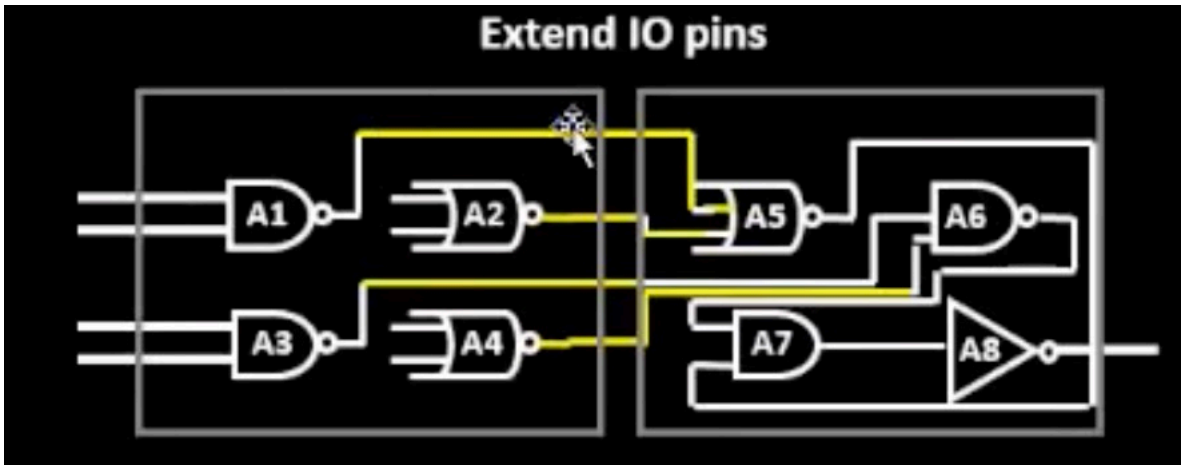


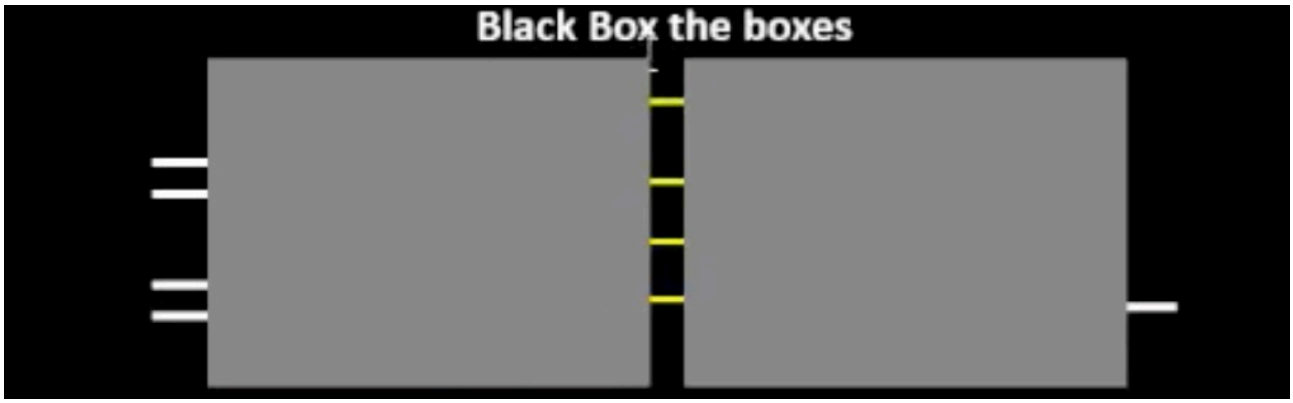
We can consider cut1 and cut2 as two separate modules, that when connected together in the same fashion, it performs a function.



here we can see the connections as A1 is connected to A5, A2 is connected to A5, and so on. Now as these are two separate modules, we have to implement them separately , but how to we implement them?

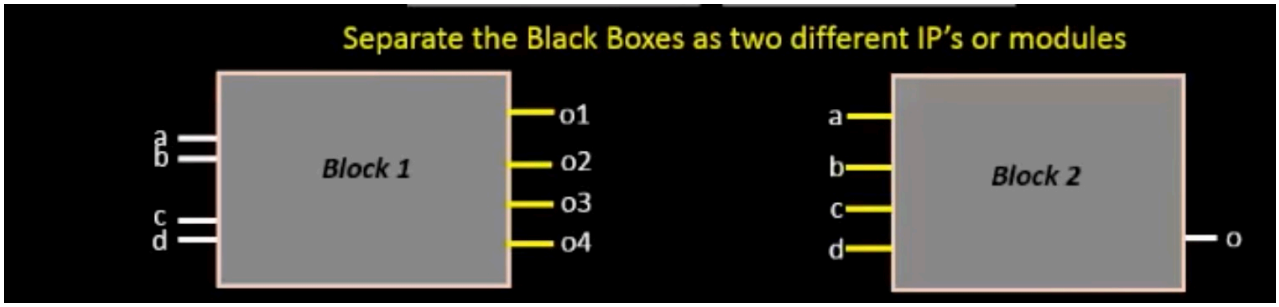
The first step is to extend the input and output pins, right now, there are many input pins .when we extend them, it looks like this.



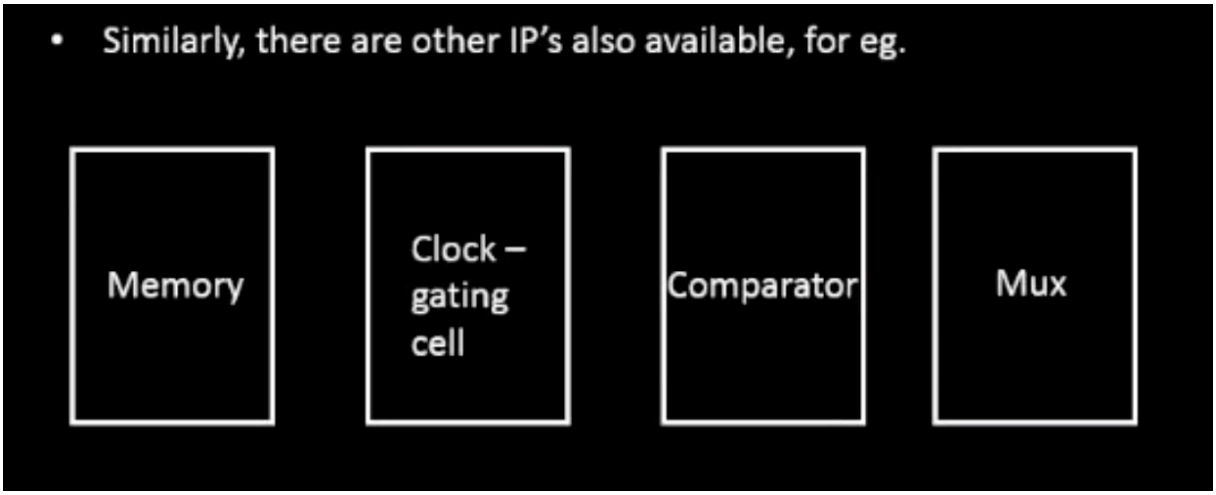


Now, lets Black box the two modules. Black box means that the components inside the netlist will not be visible for a person looking from above. It looks like this in the second diagram above. Now, lets separate the 2 black boxes like so

(continued in next page)

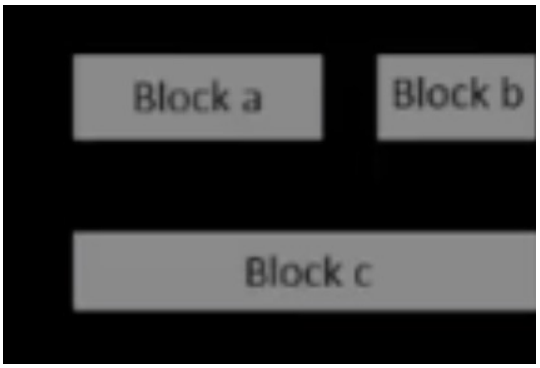


So now we can assign 1 black box to a user and we can the other black box to another user and they will both implement it once and it will only be implemented once and can be replicated troughout the chip

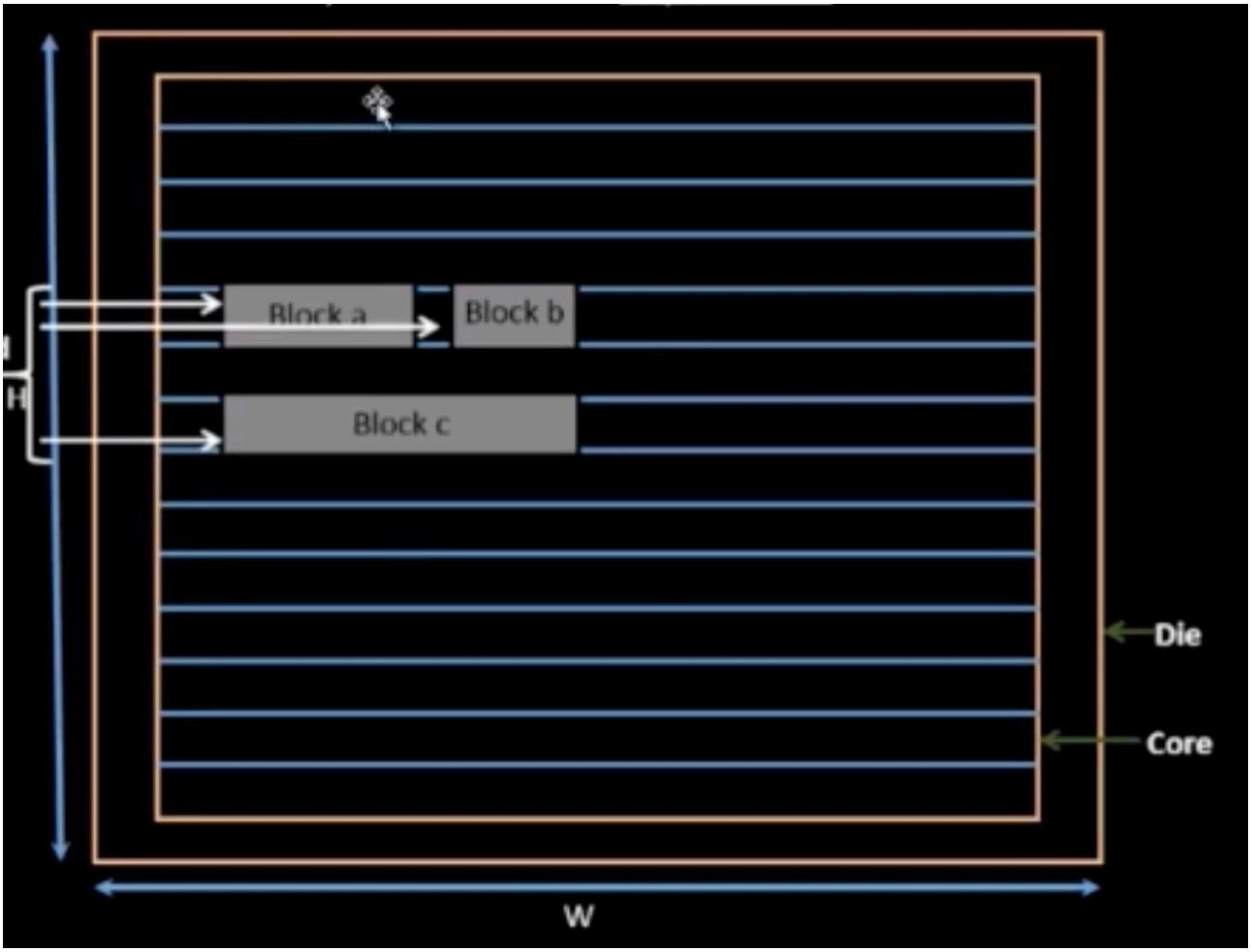


The arrangement of these IPs in a chip is referred to as floor planning. The IPs which have used defined placements before the automated routing and placement are known as pre-placed cells. The automated software usually ignores the replaced cells. Now we have to define pre-placed cells

Now lets us assume I have a bunch of cells, i.e Macros and IPs known as Block A, Block B and Block C

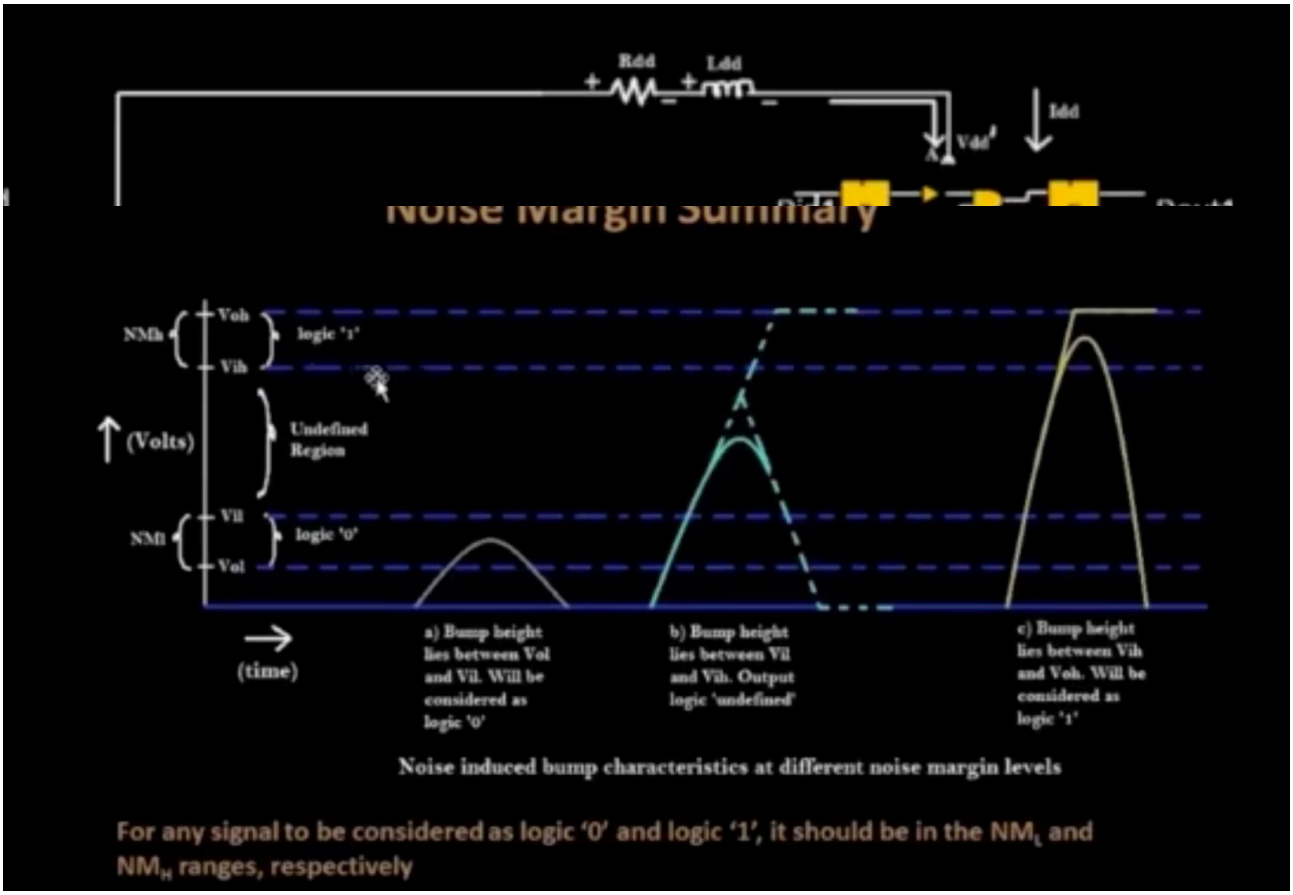


Now, these blocks, typically memory blocks, are situated near the input pins, so we can place them as such

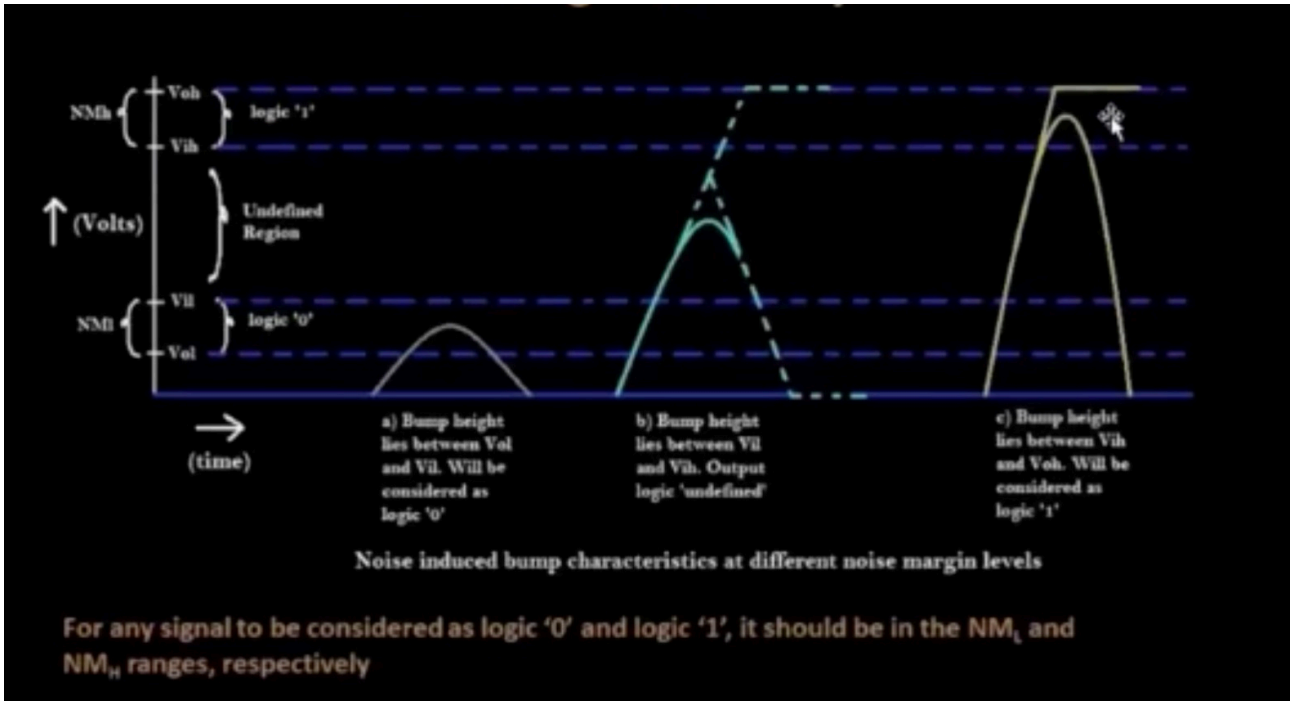


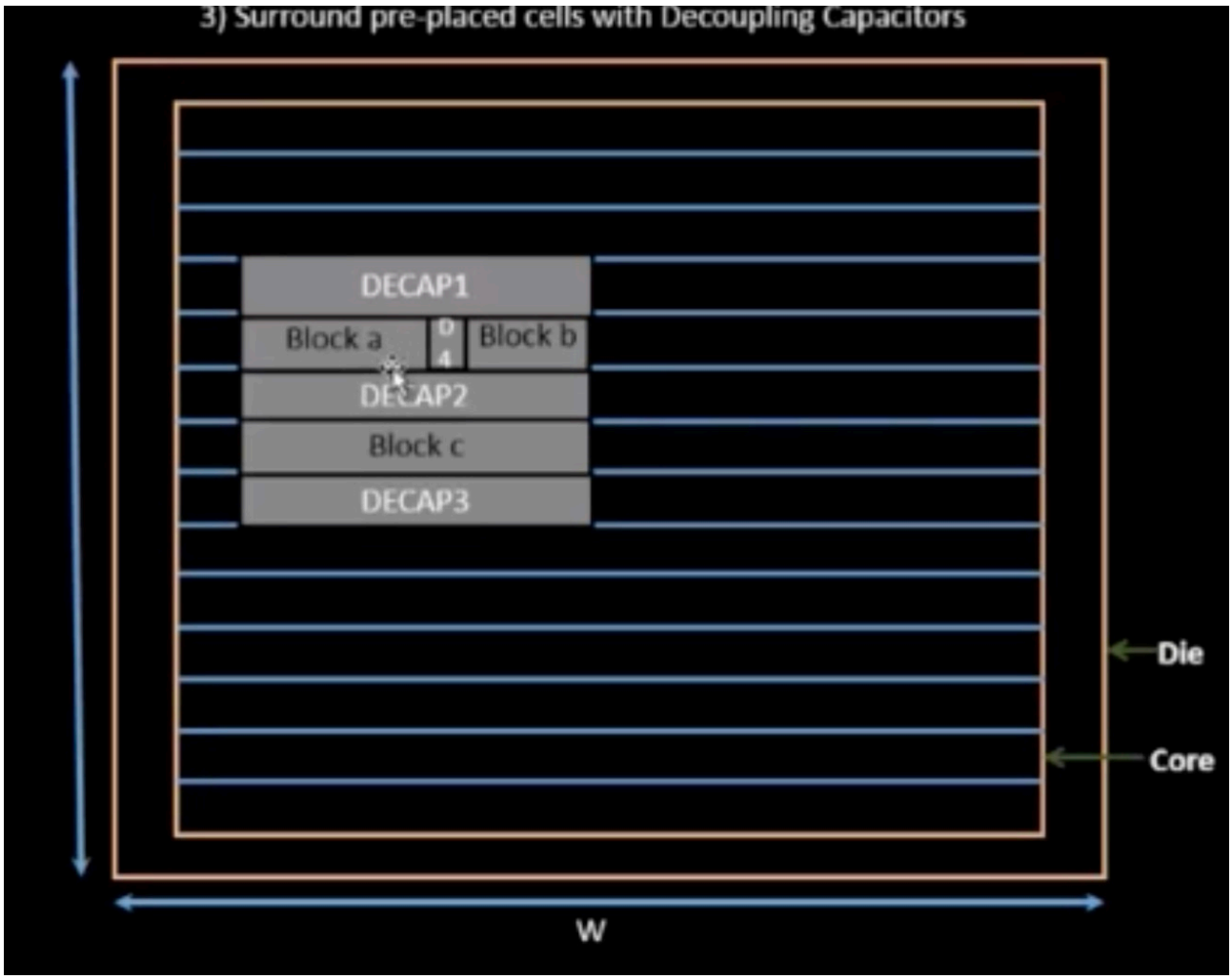
We surround pre-placed cells with de-coupling capacitors. If we think of a circuit to be part of a block, whenever it is switched on there is a demand for current, which is supplied by the Vdd. Upon switching the circuit off, there is a discharge, which the ground accepts. However, practically when voltage is supplied is passes through a wire which causes it to reduce slightly due the resistance, inductance and capacitance in the wire, and the reduced voltage is called Vdd'. The Vdd' always needs to stay in the noise margin - which ranges from Vih to Voh. If this is not true, the circuit is unstable. This is due to the large physical distance between the actual voltage supply and the circuit.

Decoupling capacitors is a solution to this problem. Decoupling capacitors can be thought of as as a huge capacitor completely filled with charge. The equivalent voltage across the capacitor is same as across the main supply voltage. The capacitor decouples the circuit from the main supply. Hence, all the pre-placed cells get their power supply from the capacitors and hence are completely stable.



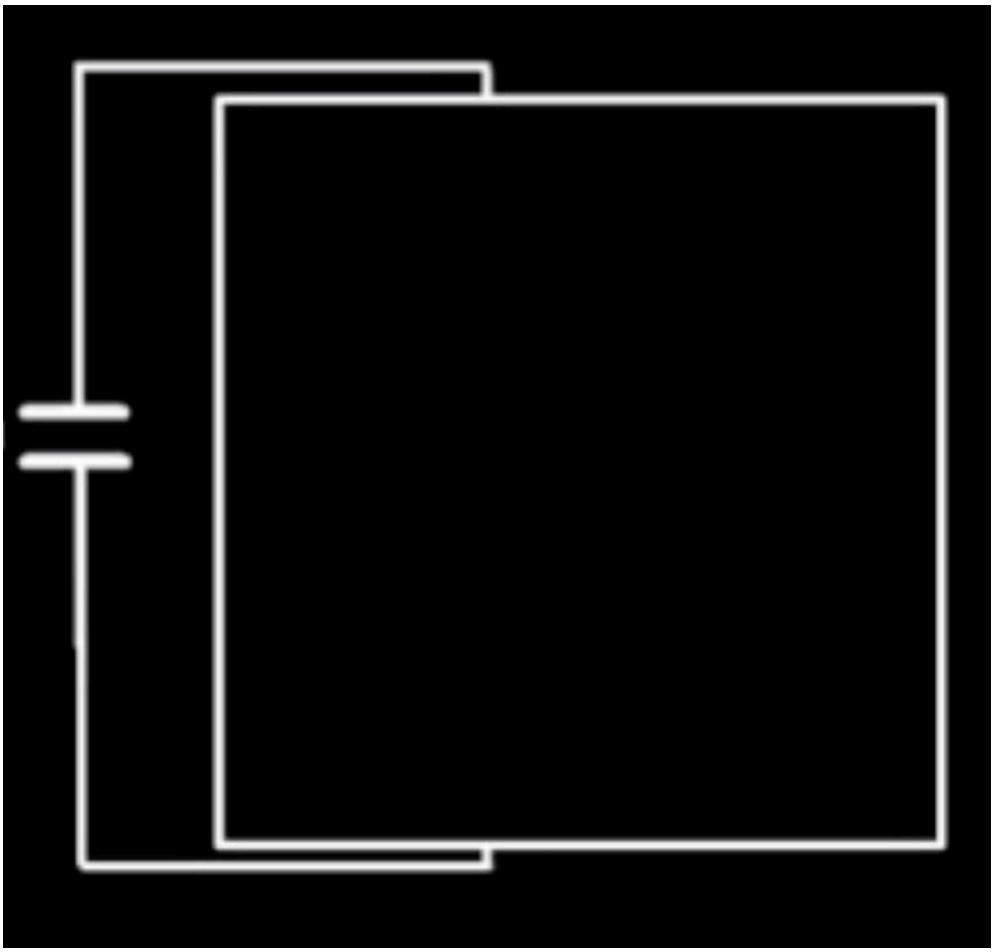
The noise margin summary looks something like this





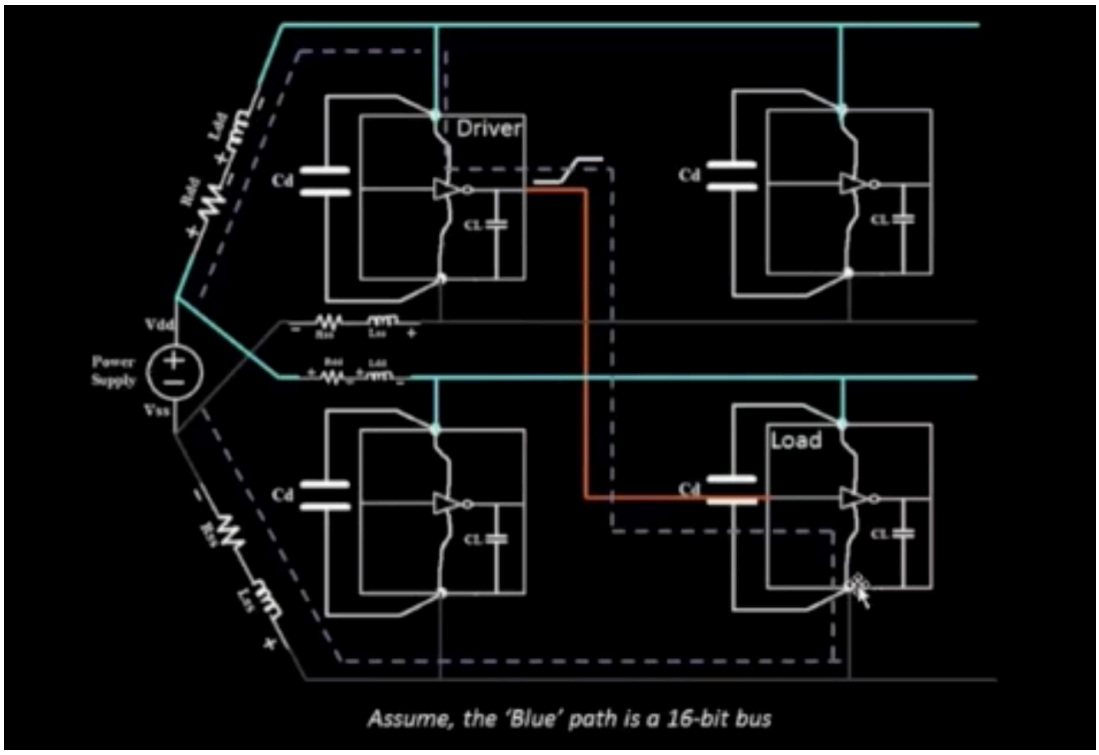
This is how the diagram looks like after the pre-placed cells have been surrounded by decoupling capacitors

Now, lets get started with power planning, but first let me introduce you to a problem I have

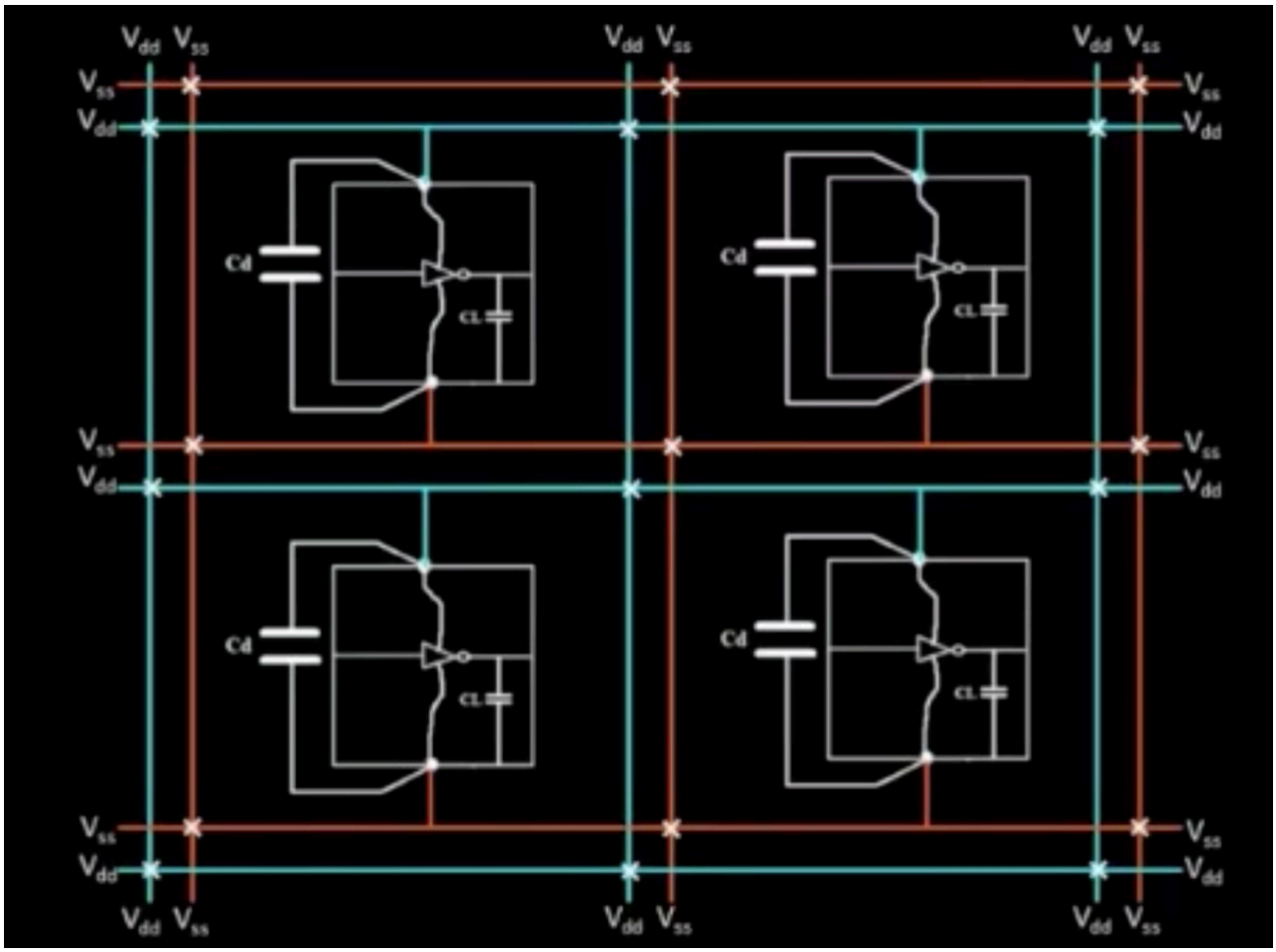


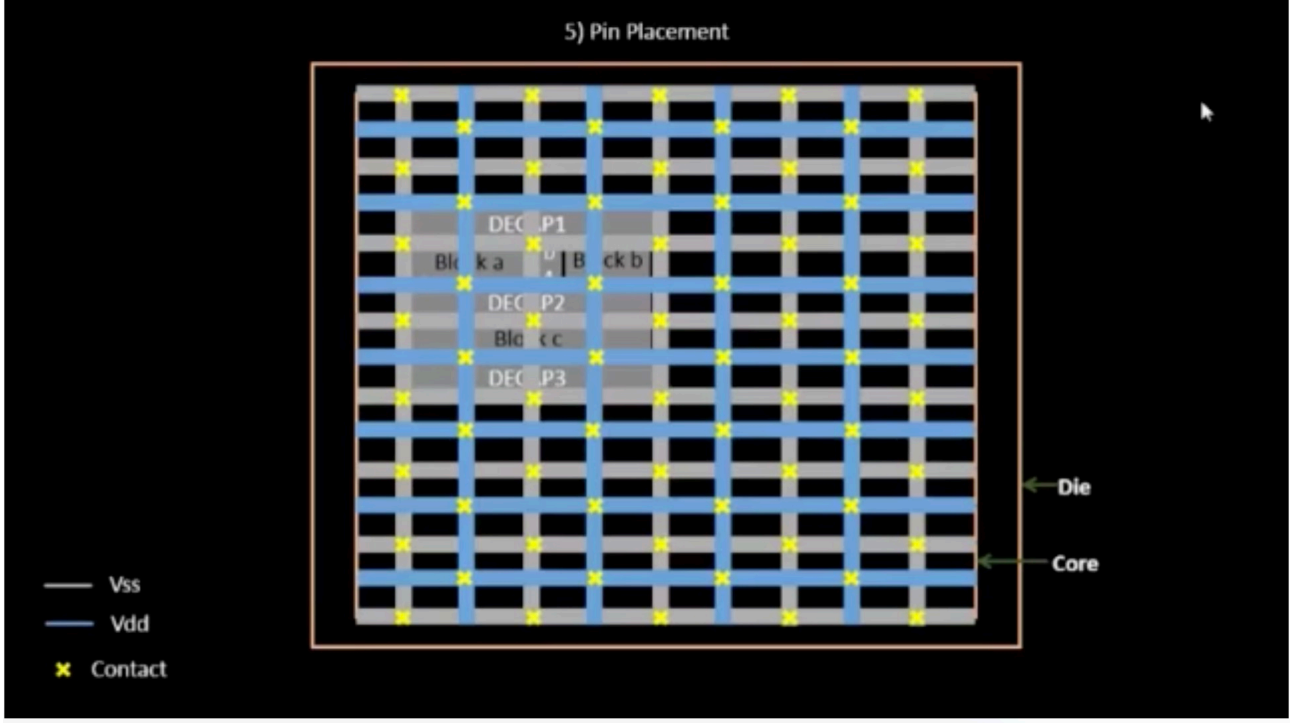
You suppose this particular piece of logic to be a macro, that is repeated many times on a single chip. It requires a lot of voltage, so voltage must be supplied through a decoupling capacitor. However, it is not feasible to add the de-coupling capacitors on the entire circuit - only critical elements can be decoupled.

If the 16 bit bus is connected to an inverter, then it means that all the capacitors will discharge the voltage at once. But, lot of capacitors discharging at once cam cause Ground Bounce due to great amount of voltage needed to drained at the same time, and turning the capacitor on might cause Voltage Drop due to insufficient current. Ground bounce and voltage drop might cause the voltage to not be within the noise margin range. To solve this problem, we can have multiple powersource taps and sources which is known as a power mesh, where capacitors can source current from the nearest Vdd and sink current to the nearest Ground

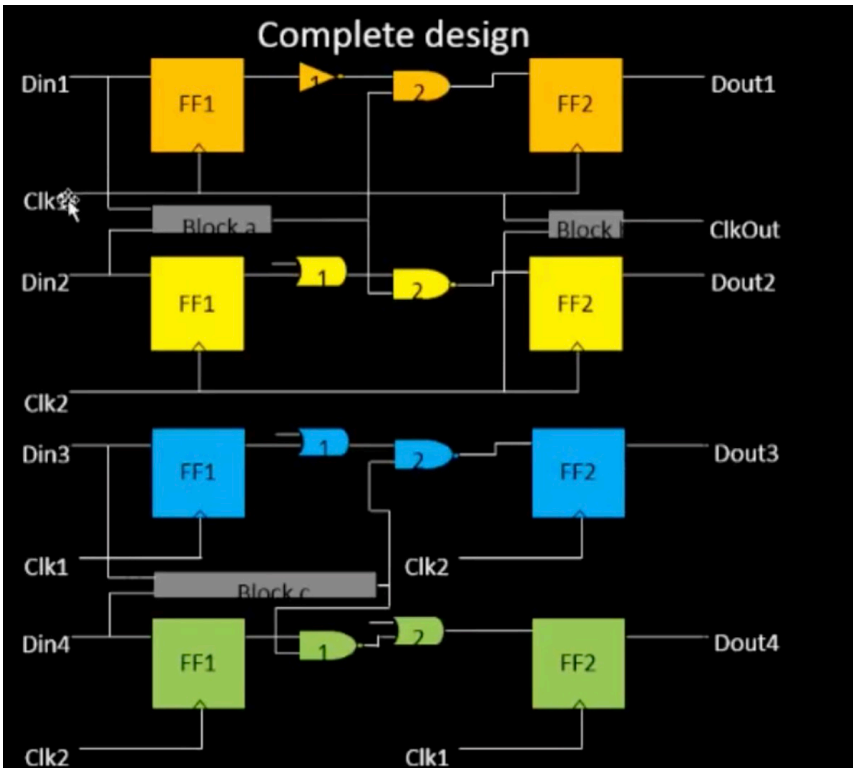


The problem to our dilemma looks a little something like this

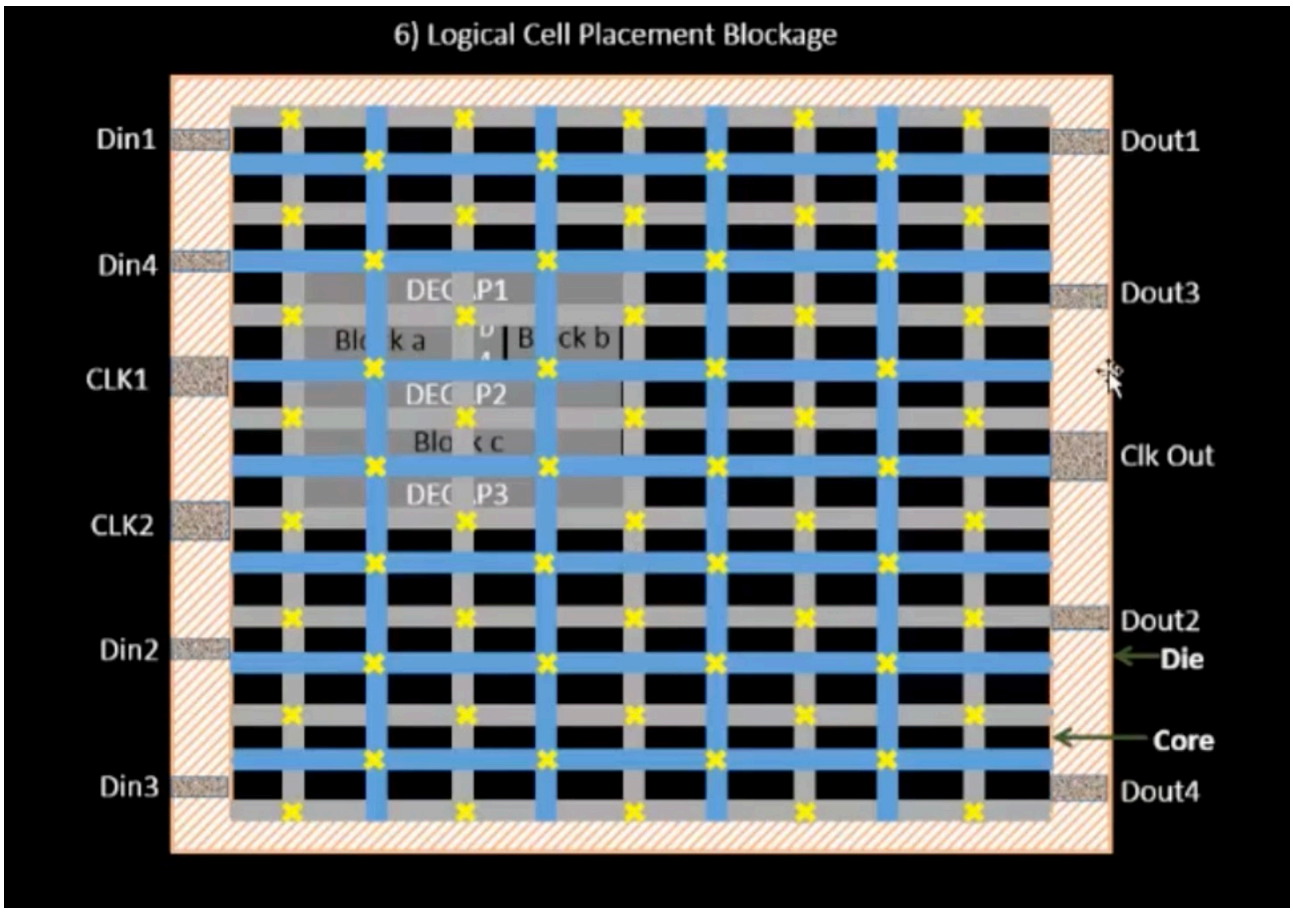




This is our diagram after Power Planning. The next step is known as pin Placement. Let us take an Example before implementing it into our design



the connections between all the components are defined in Verilog.



Taking the above netlist as an example needing to be implemented,.The input and output ports are placed left and right between the core and the die respectively. The placements of the ports is cell-specific. The clocks are continuously drive the cells and hence clock ports are bigger than data ports. Due to this, we also need the least resistance paths for the clocks. The size is inversely proportional to the resistance.

After the pin placement, we create Logical Cell Placement Blockage to ensure that the APR tool does not place any cell on the pin locations.

Now, lets get started with floorplanning . But before we can simply run the run_floorplan command, there is a couple of things that we need to check

The first step is setting the configuration variables, which can be found in the README file in openlane/configurations directory.

The README.md files contain all the configuration variables for floor planning. It is important that we look over and set the configuration variables to our desired values

All the variables that we need will be in the floorplan.tcl in the same directory that the README.md file is located

Now that we have looked over the config variables, we can run the run_floorplan command

After we have run the run_floorplan command, we can go into the runs directory of the picorv32a directory to check the results of the run_floorplan command.

By running the command “cd results/floorplan” (if you are in the picorv32a directory), we can see a def file. a def file is a Design Exchange Format file. It is the end product of the run_floorplan command.

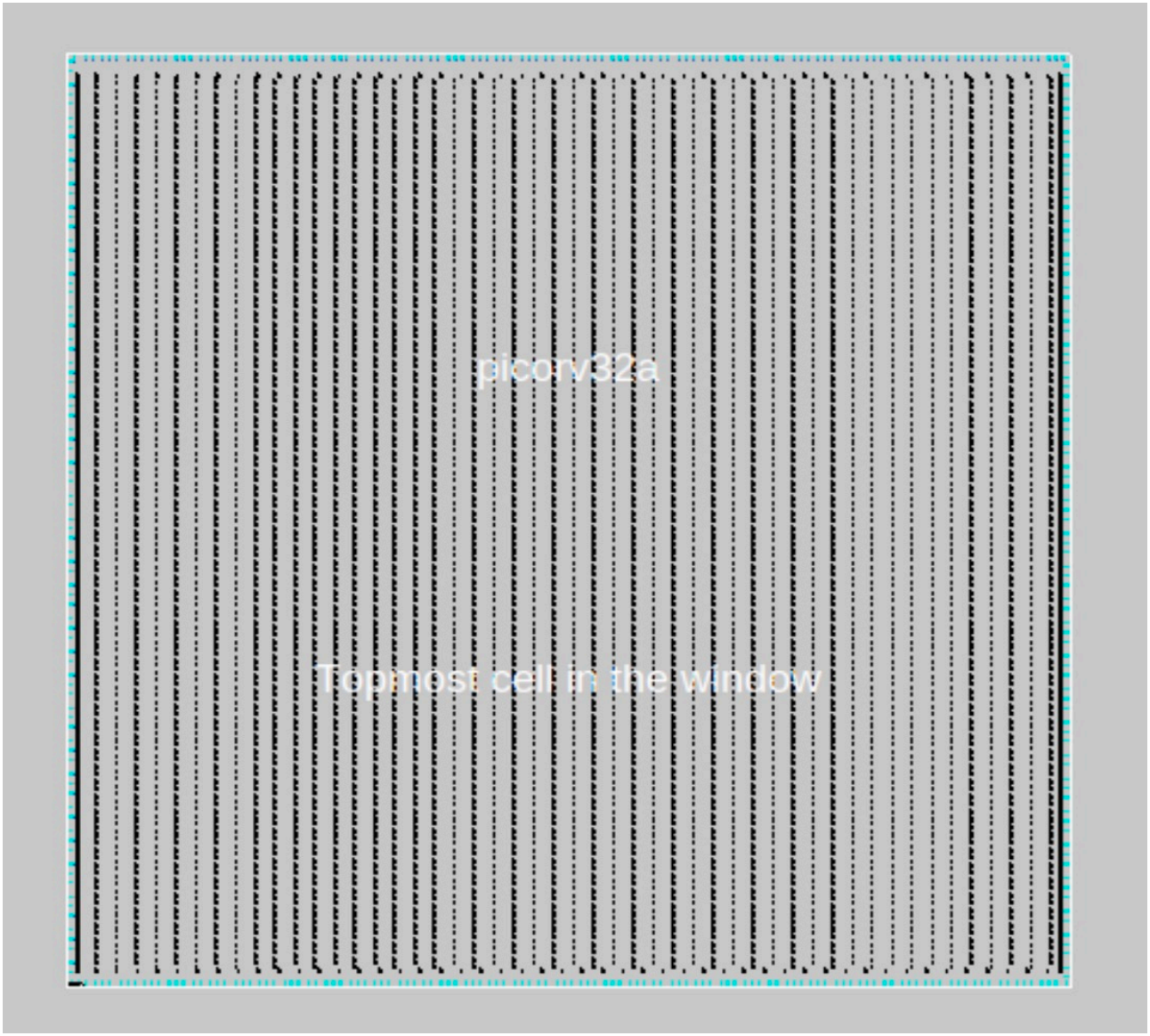
When we open the def file, we can see a the placements of each logic gates and more, but first we need to look at the DIE AREA. Here the die area is given in Microns and Database units. 1 micron = 1000 Database units.

So, the area of the die is 660.685 x 671.405 sq.microns = 443587.212425 sq.microns

Next, to run the MAGIC tool for looking at the floorplan layout

magic -T /home/vsduser/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.tech/magic/sky130A.tech lef read ../../tmp/merged.lef def read picorv32a.floorplan.def

After running this command , we get a screen like this



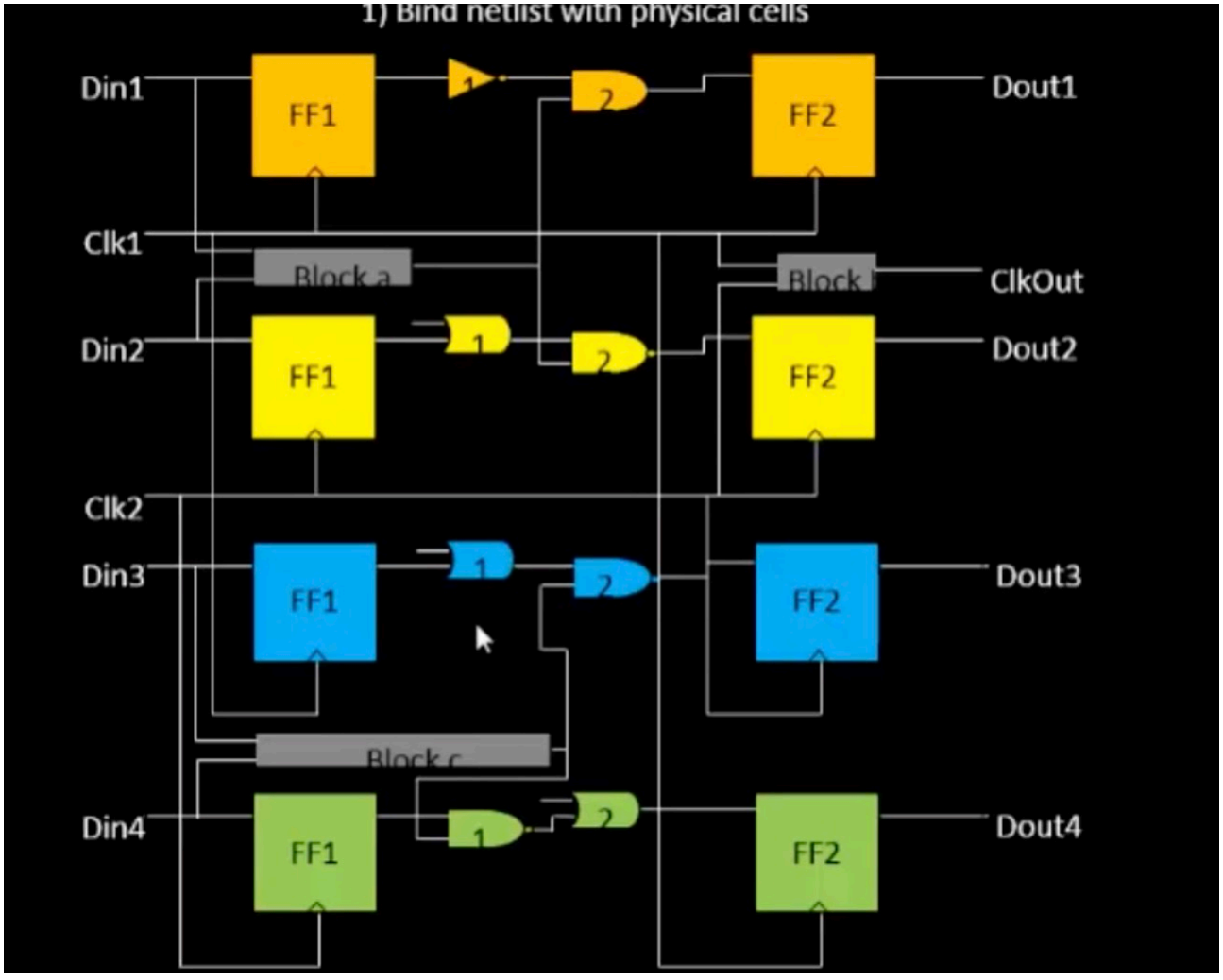
We can use S to select the entire die and we can use V to to centre it onto the screen. We can also use left click, right click to look a specific area and we can then press Z to zoom in

When select any part of the design, such as an I/O pin, we can type what into the tkcon window to see which layer of metal it is on.

In he bottom left corner of the window we can see the standard cells there as normally floorplan programs ignore standard cells

Now, lets get started with library binding and placement.

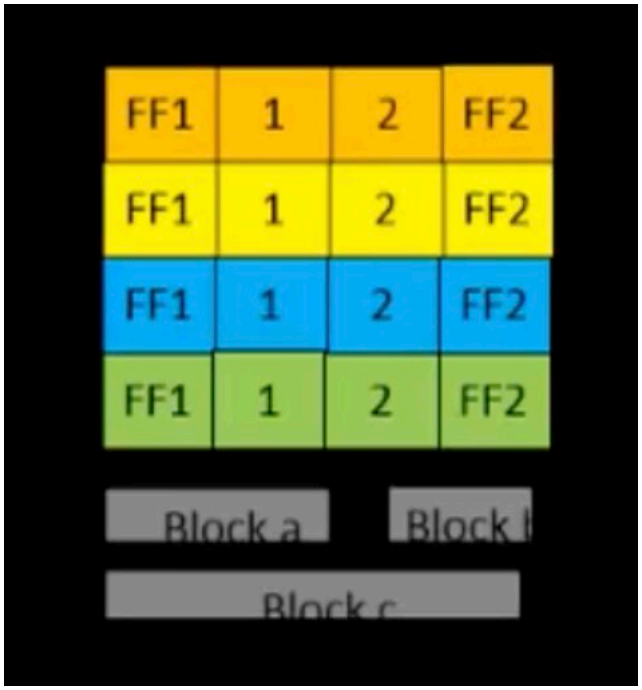
The first step, is to bind the netlist with physical cells I.e cells which have some physical dimensions. Imagine we have a circuit like this



Now, we can see that every shape determine a functionality, but in reality everything will look like a box in physical view. Suppose take the flip-flop, it looks like a rectangle no?, but in a physical view , it looks like a square. Lets take the NOT gate , here it is made up of a triangle and a circle. But, in a physical view, it looks like a box like this, here, the 1 is the not gate

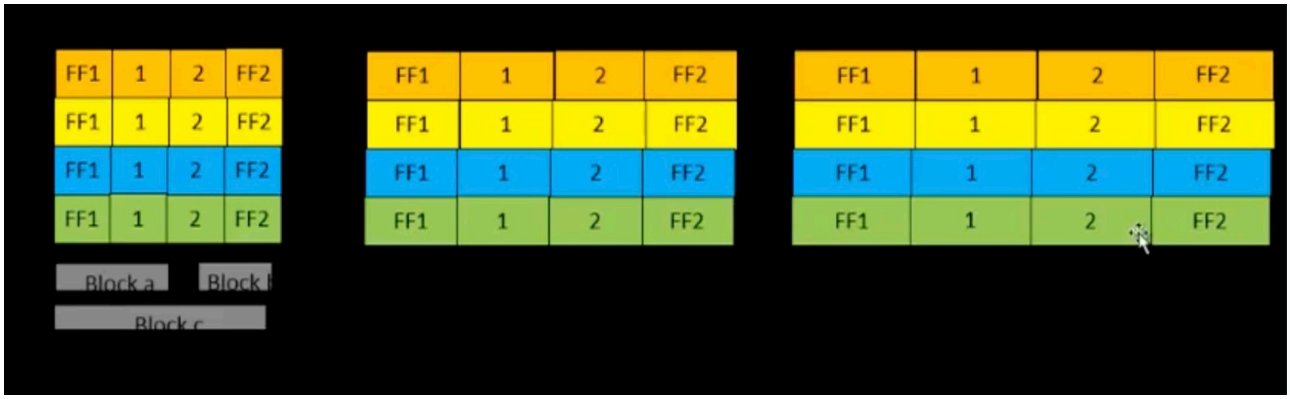
Now, there is another AND gate, it looks curved, but in a physical view, you guessed it, it also looks like a square like this,

Note that the 2 represents the AND gate, now we have given them a physical dimension, I.e length and width. Now doing this for the entire netlist gives us,

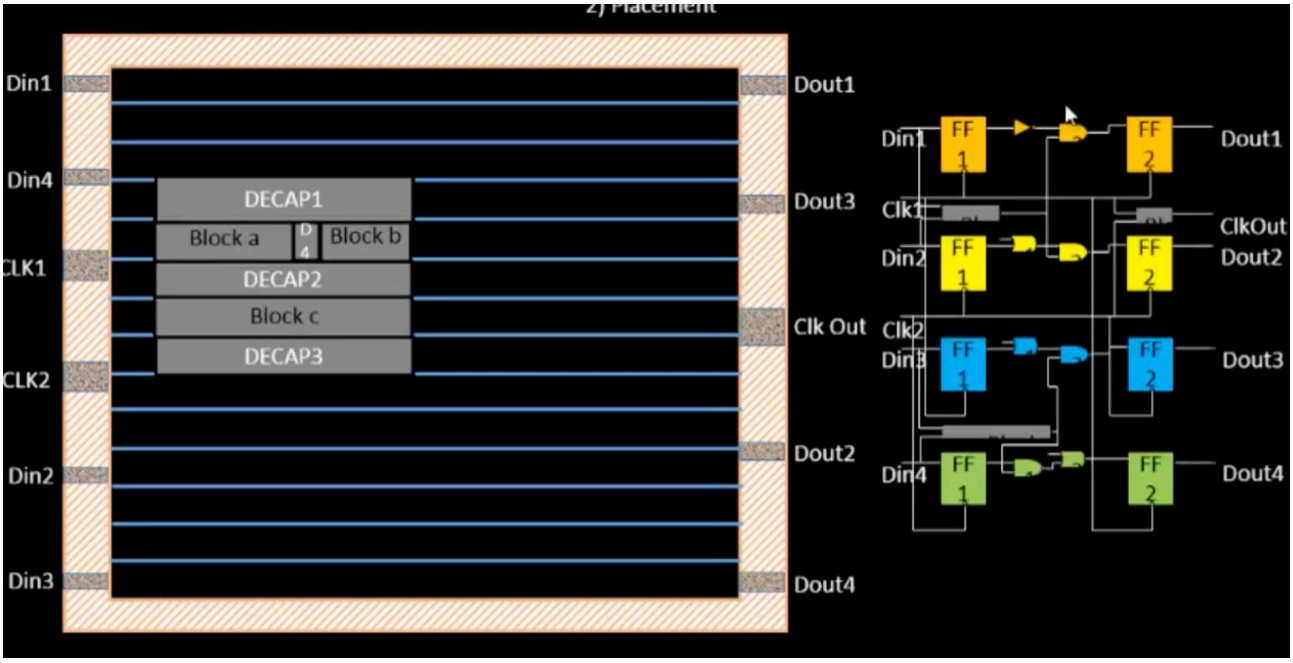


Now, this looks like a shelf no? But, we call this a library . In libraries, we can see all sorts of boxes like these. They store the timing information of the gate, the delay for the gates and so on. Some libraries only store timing information, some libraries store only delay information and so on, but lets us call it a library for now. The library store the size of the cell, the width of the cell, the length of the cell, the delay of each cell and the specific conditions for each of those cells

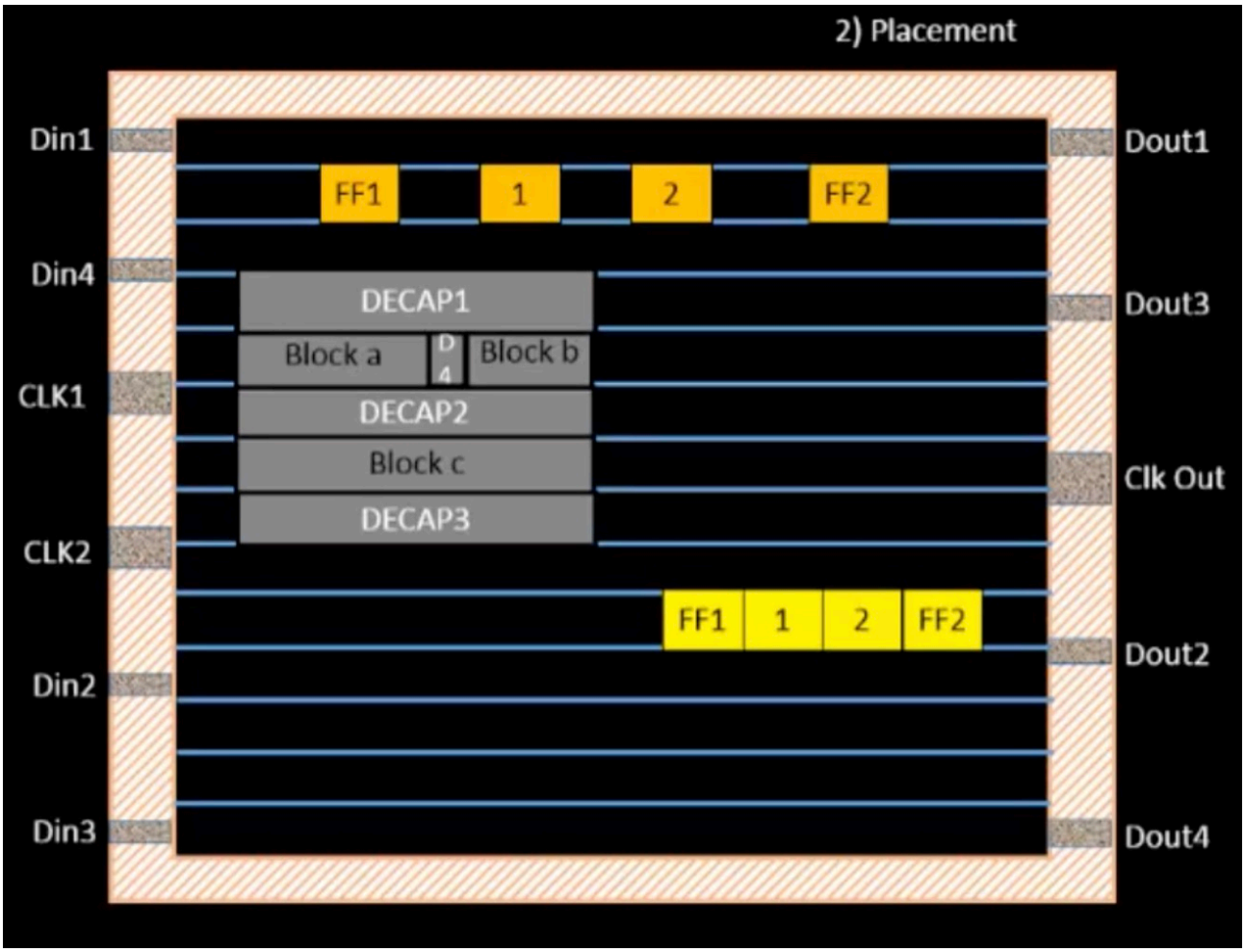
Now, each cell in library has different flavours of sort, like an ice cream, but these cells can vary in length and width, and hence are faster, because the offer the path of least resistance. So , in each cell , there is usually 3 or more “flavours” of cells like this



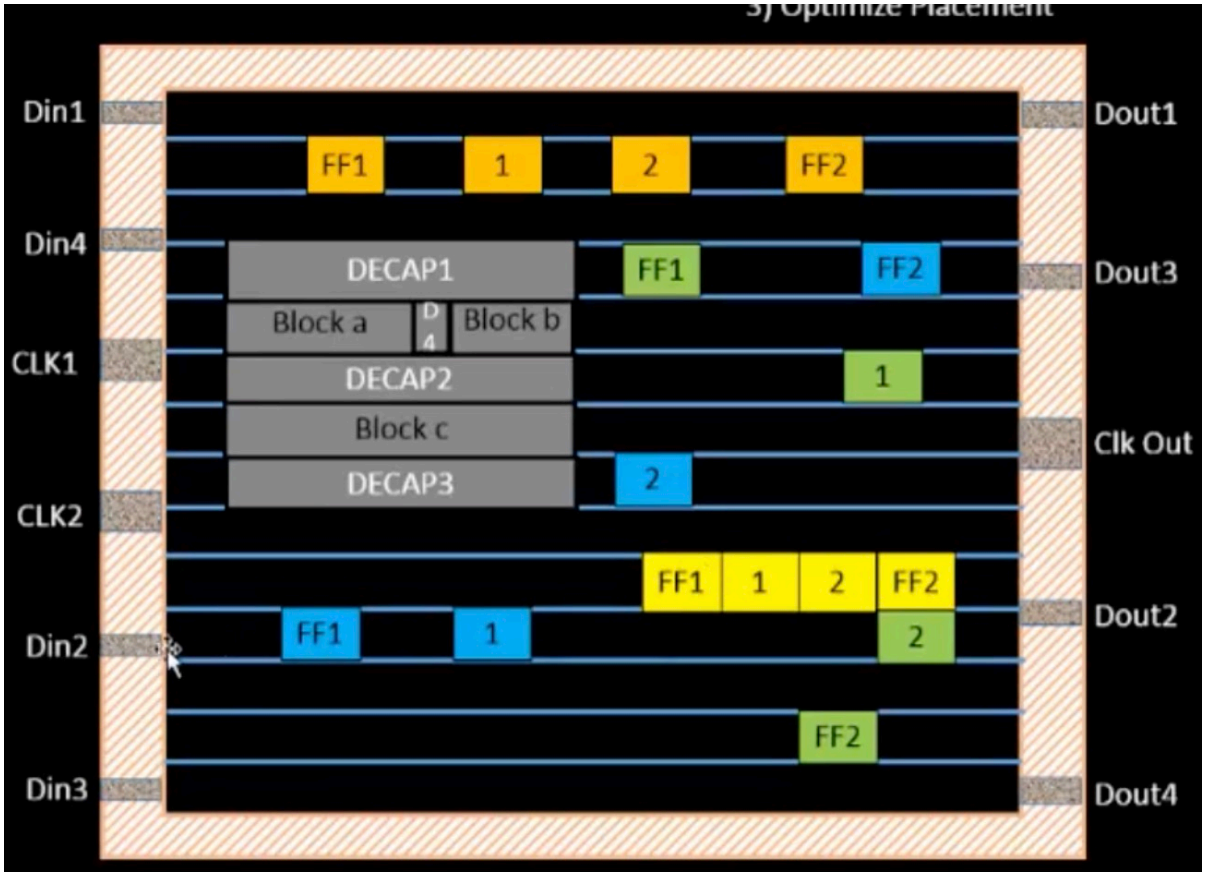
Now, the next step is placement, that is to place the cells into the design (floorplan) that we already have as physical cells



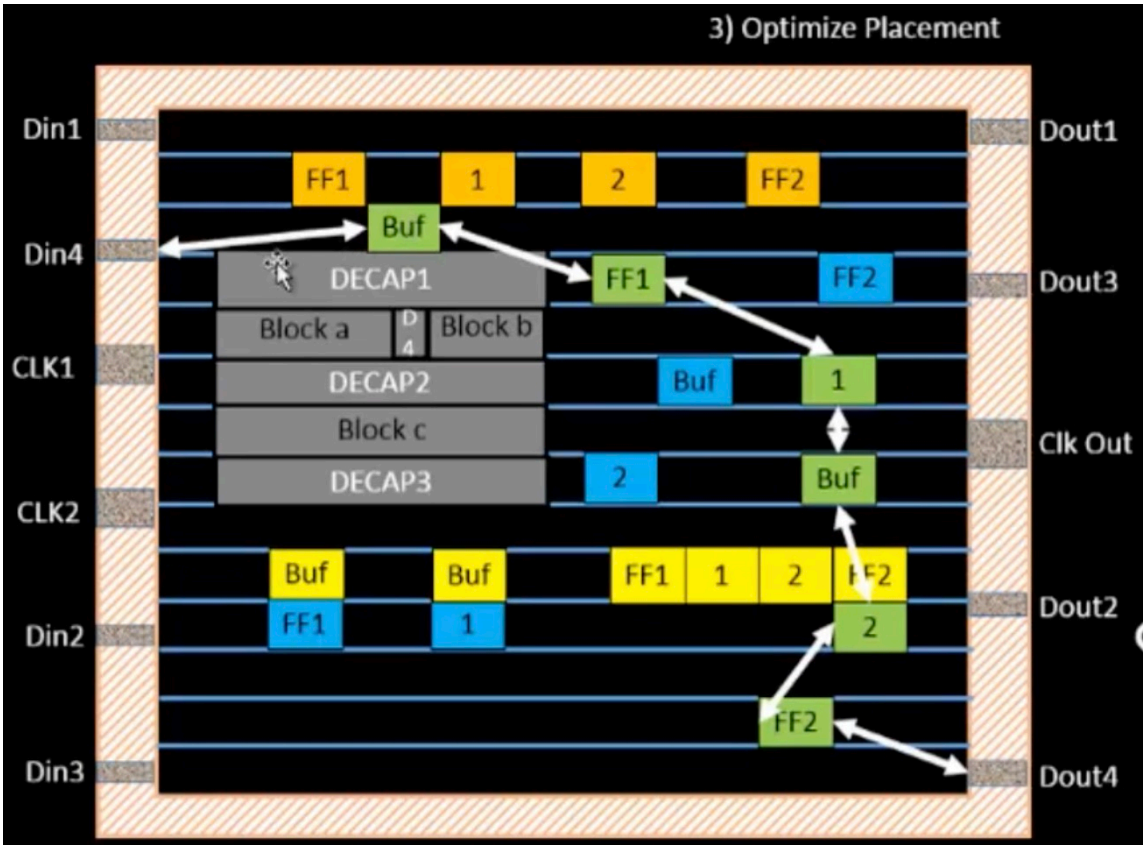
Now, lets start with the first section as it is close to Din1 and Dout 1. Now that we have done that let us plan section 2 near Din2 and Dout 2 as it is closer to them.



Now, lets place the 3rd section, we got two flip-flops and some gates. But now, notice When Din3 is and where Dout3 is, they are diagonal to each other unlike with the first section, where they were in a straight line. When we also place section four into the design we, get a design like this



Now, lets try to optimise this placement as it does not look clean. In a nutshell, we estimate the length of the wire and then we measure its capacitance, now, based on that we put repeater in certain places so that the flip-flops and gets receive a proper input.



Now that we have properly optimised the circuit, we will run a timing analysis to see if everything is working properly, because, if any problems crop up in the next few stage, it will get ugly

Now, lets run the placement command in openlane, which is run_floorplan, but it is a wrapper for

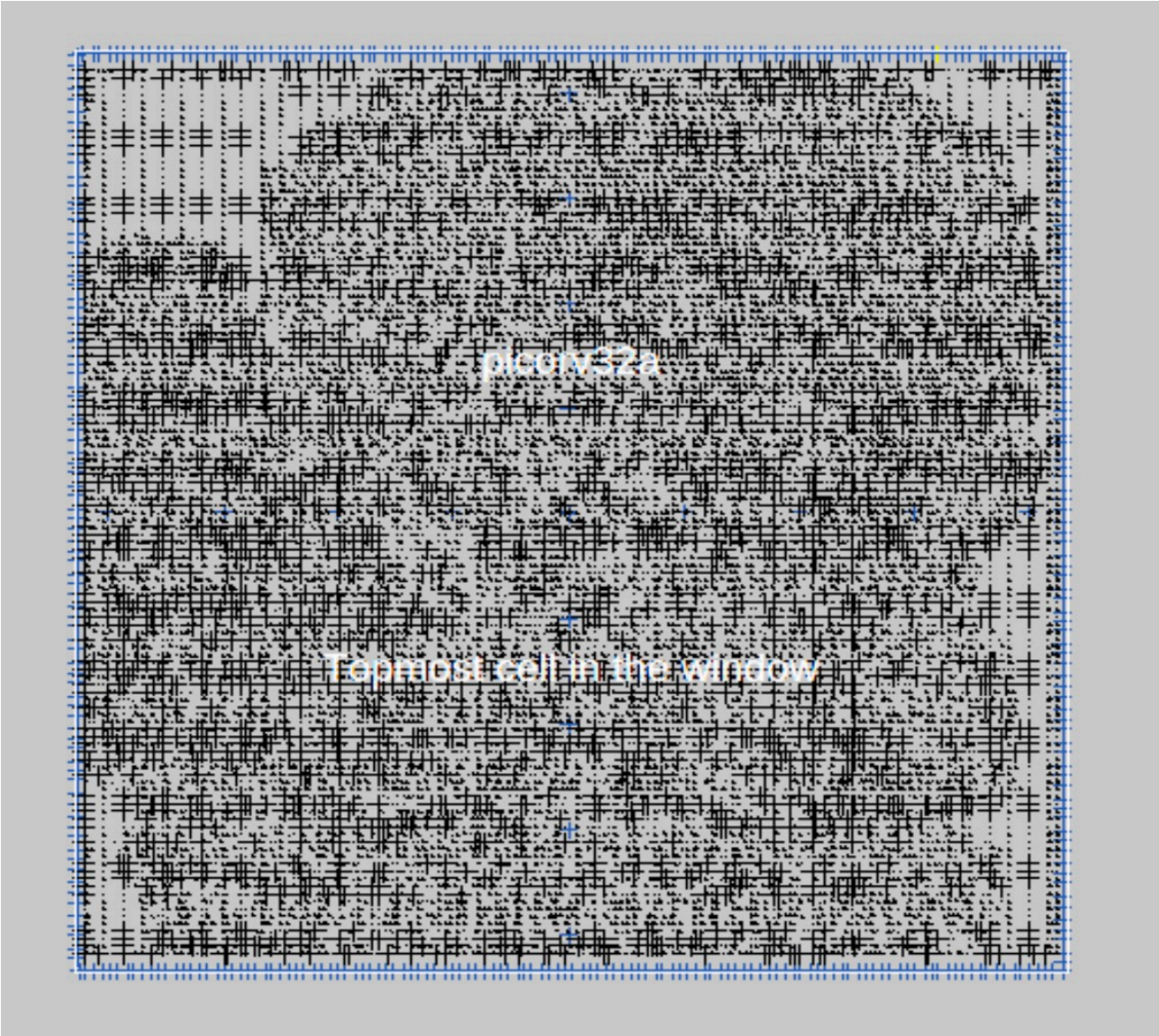
Global Placement (by using the RePlace tool) - there is no legalisation and HPWL reduction model is used

Optimization (by Resier tool)

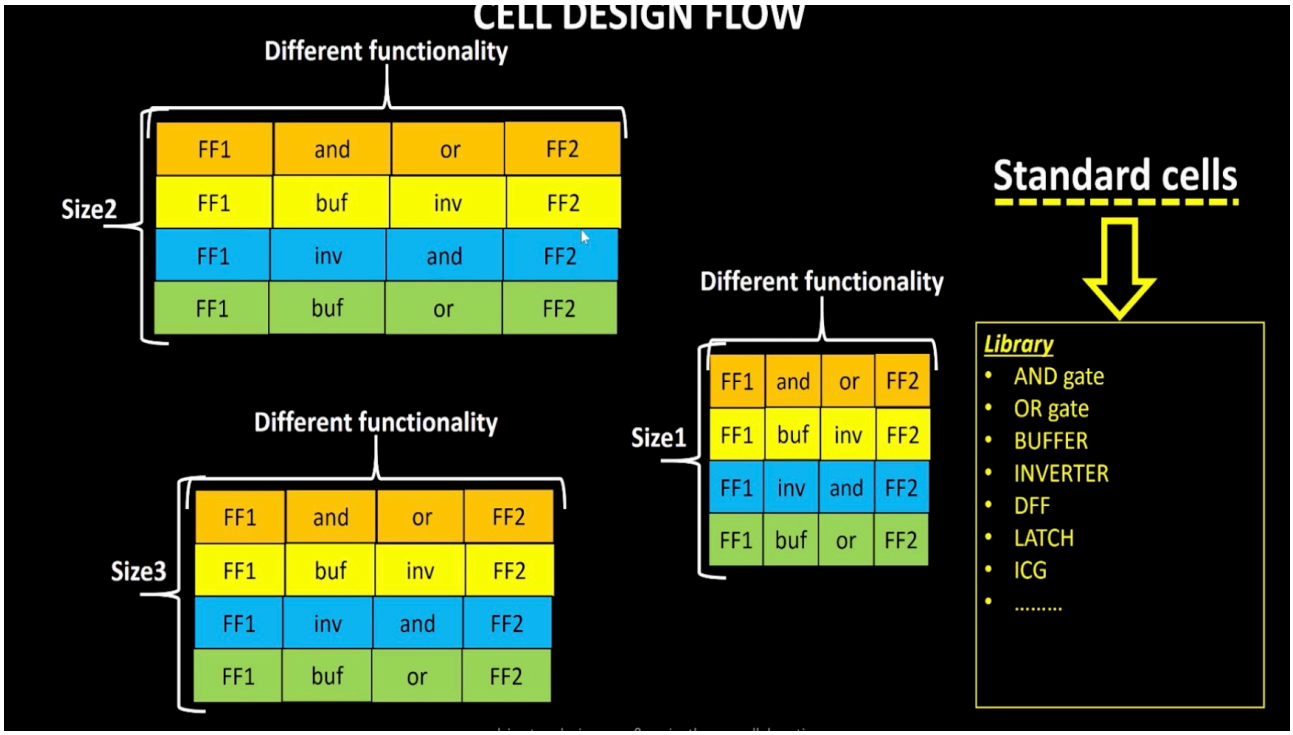
Detailed Placement (by OpenDP tool) - legalisation occurs - where standard cells are placed in rows and there will be no overlap of the cells.

Placement aims to converge the overflow value. When the placement is successful and the design converges, the overflow value will progressively get lesser and lesser.

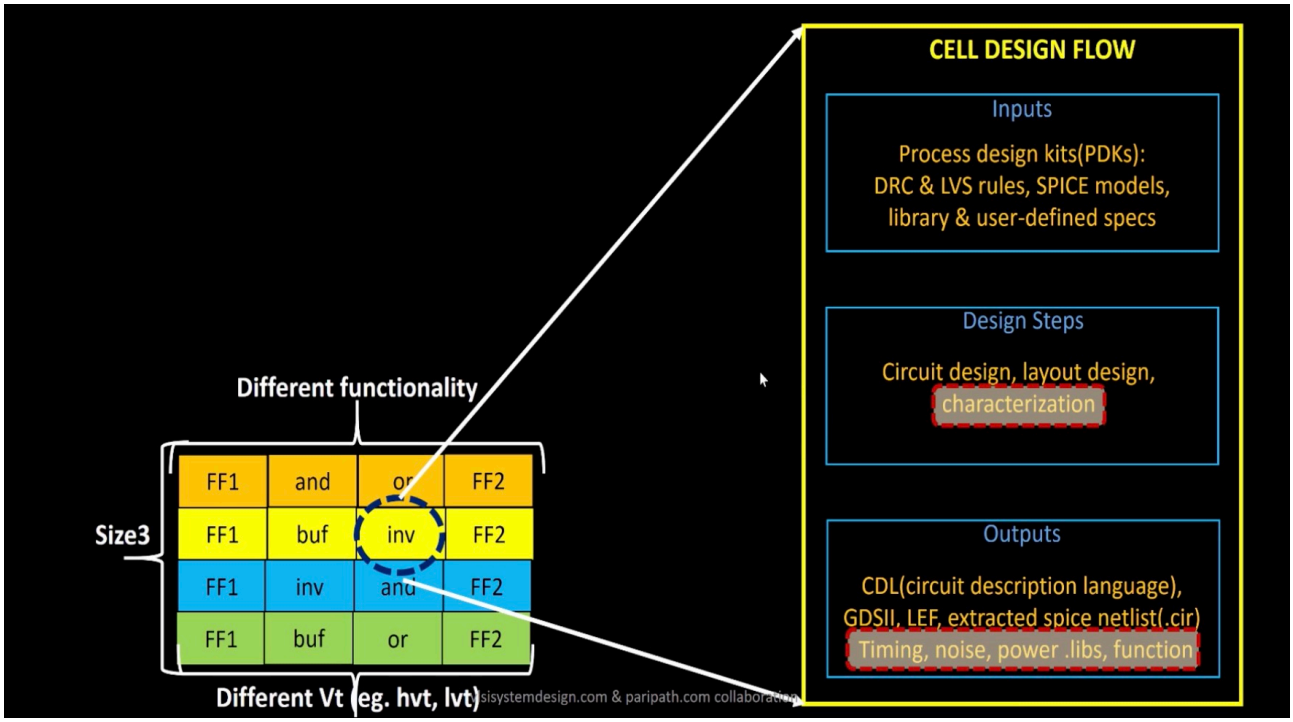
We can type the command magic -T /home/vsduser/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.tech/magic/sky130A.tech lef read ../../tmp/merged.lef def read picorv32a.placement.def & to view it in magic. It looks like this



Now, lets talk about standard cell design flow, now, we already know about standard cells and libraries as we have talked about them above.



Now, lets consider the cell design flow for an inverter



Now, DRC & LVS Rules contain tech files and poly substrate parameters . SPICE Models contain threshold, linear regions, saturation region equations with added foundry parameters, including NMOS and PMOS parameters.User defined specifications include cell height and cell width, supply voltage, pin locations, and metal layer requirement

IMPORTANT: The standard cell library developer must adhere to the rules given by the foundry so that when the cell can be used on a real design without any errors

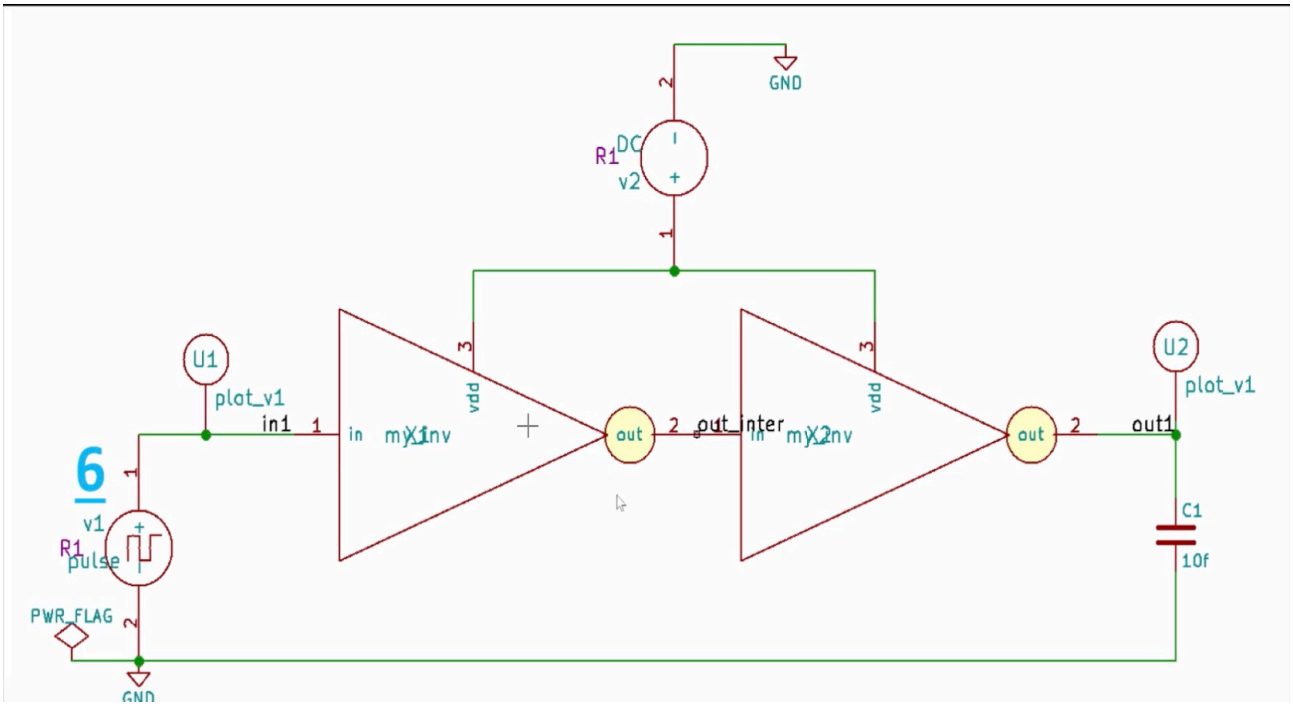
Circuit design is done by modeling the PMOS and NMOS to meet input library requirement
Layout design is done using Euler's path and stick diagram on Magic layout tool

The steps for a Standard Cell Characterisation Flow are:-

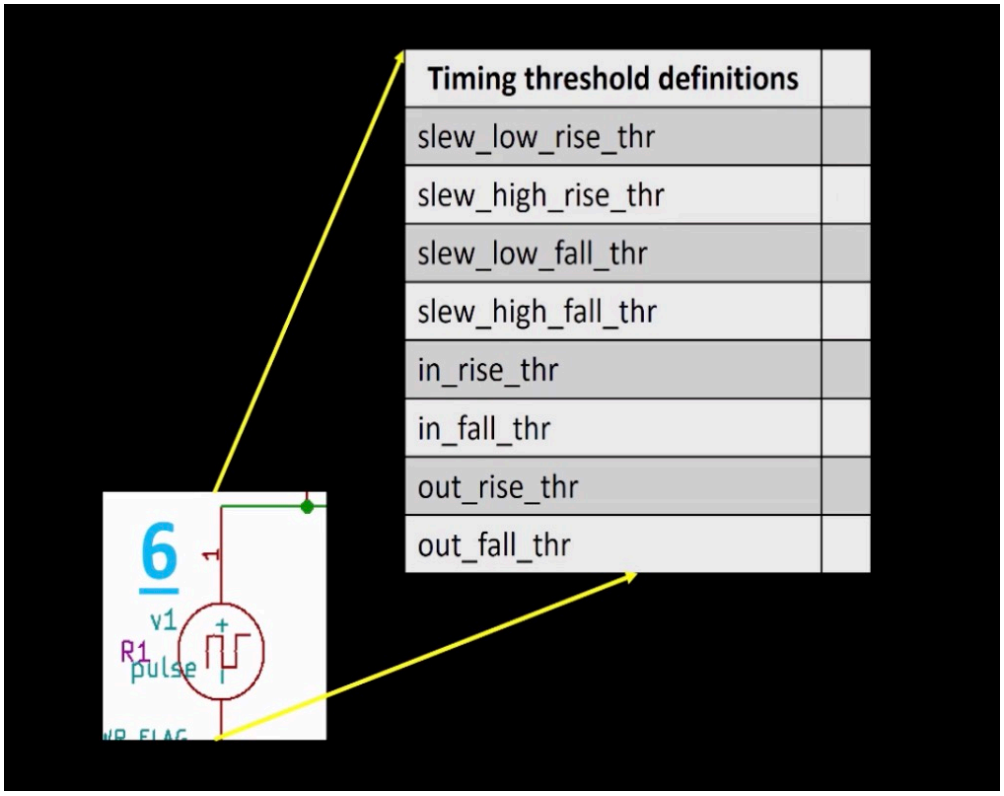
I. Reading of SPICE module files
II. Reading of netlist extracted by SPICE
III. Recognising buffer behaviour
IV. Reading subcircuits
V. Attaching neccessary power sources
VI. Applying stimulus
VII. Provision of of neccessary output capacitance
VIII. Provision of simulation command

These steps are given to the charecterization software known as GUNA in the form of a configuration file, which will generate timing, noise and power models in the form of .libs files.

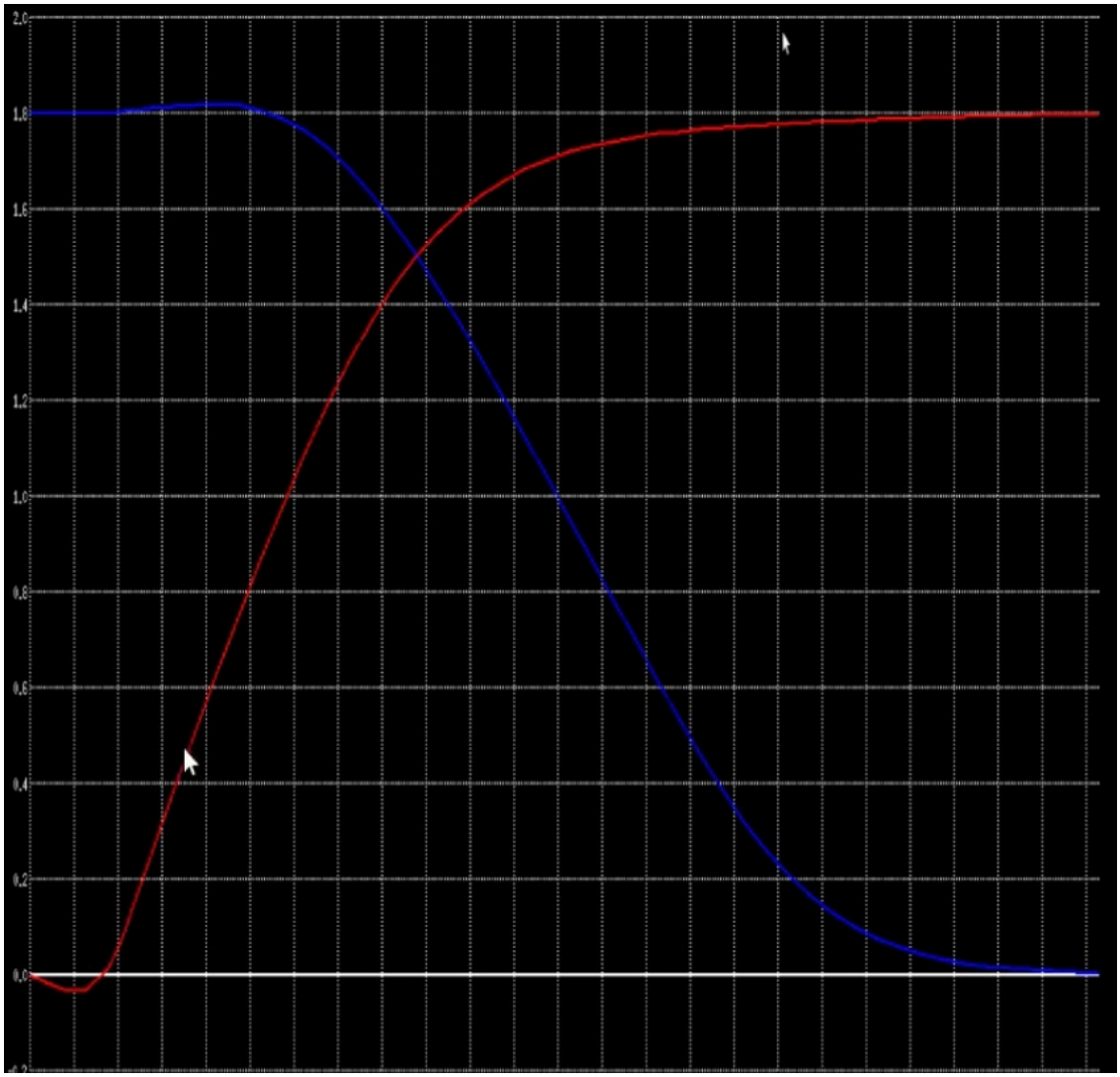
Now, lets talk about Timing characterisation. Here we will learn about the syntax and semantics of the timing.lib, noise.lib and power.lib. we must understand these to be able to run GUNA, which will create these files. Lets take an example of a buffer circuit



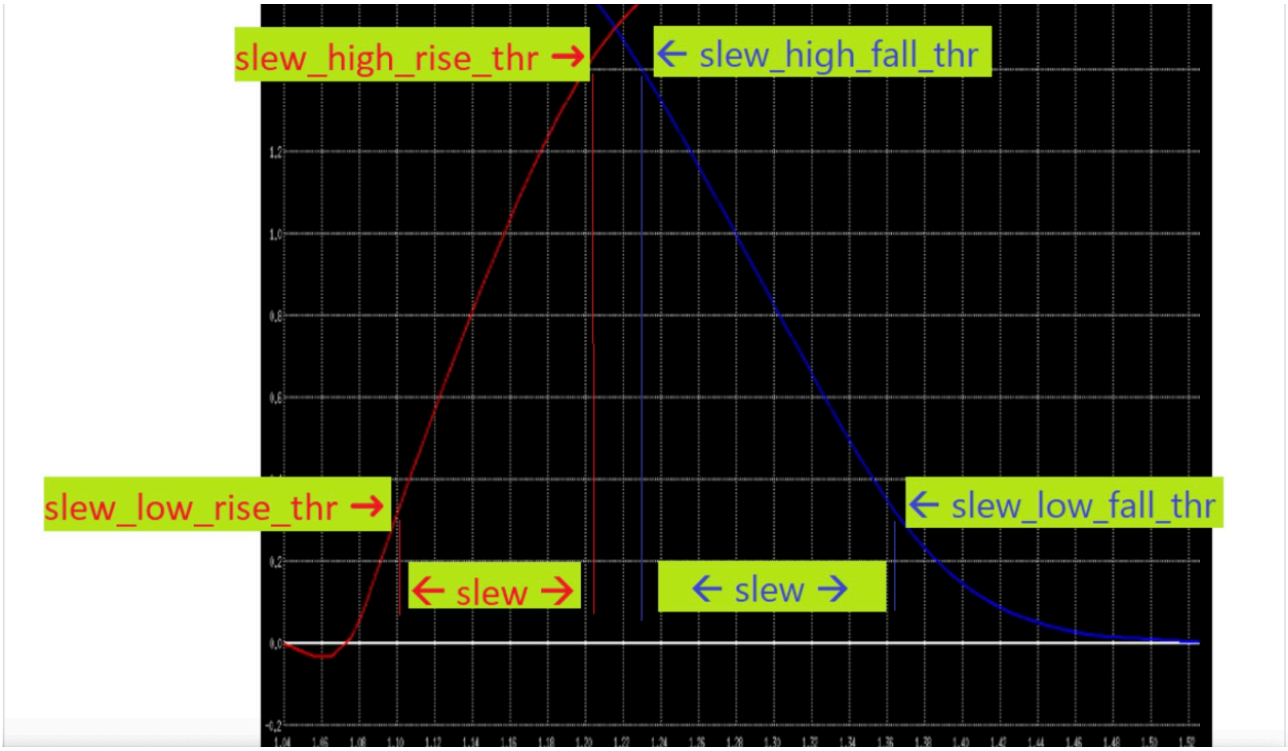
These are the variables to any waveform that we see, it is important here because we are using as AC electrical circuit as our stimulus.

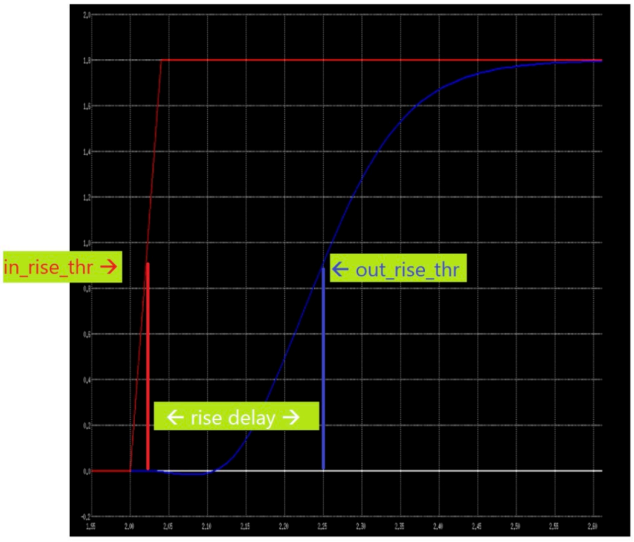


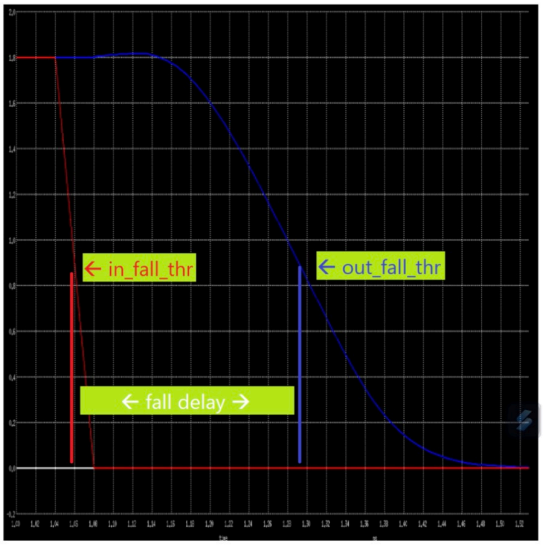
These are the variables for our waveform.



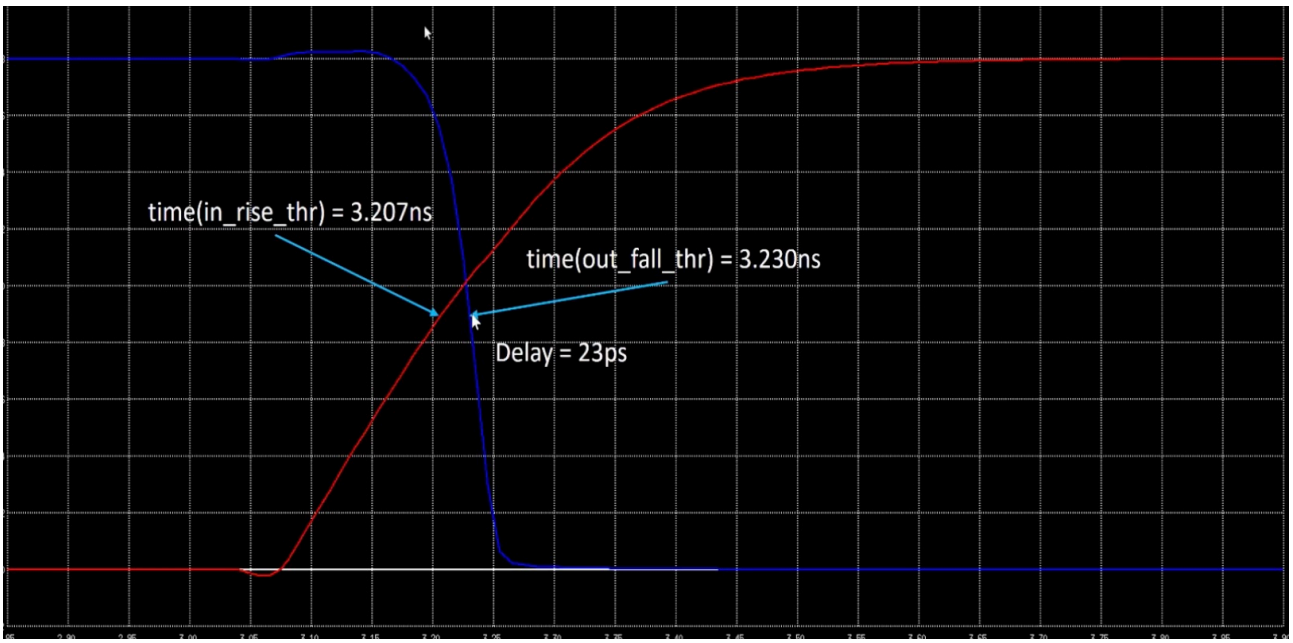
Here, we will using the graph to understand the different variables. The red line the input for the first inverter shon in the picture and the blue line is the output of the second inverter





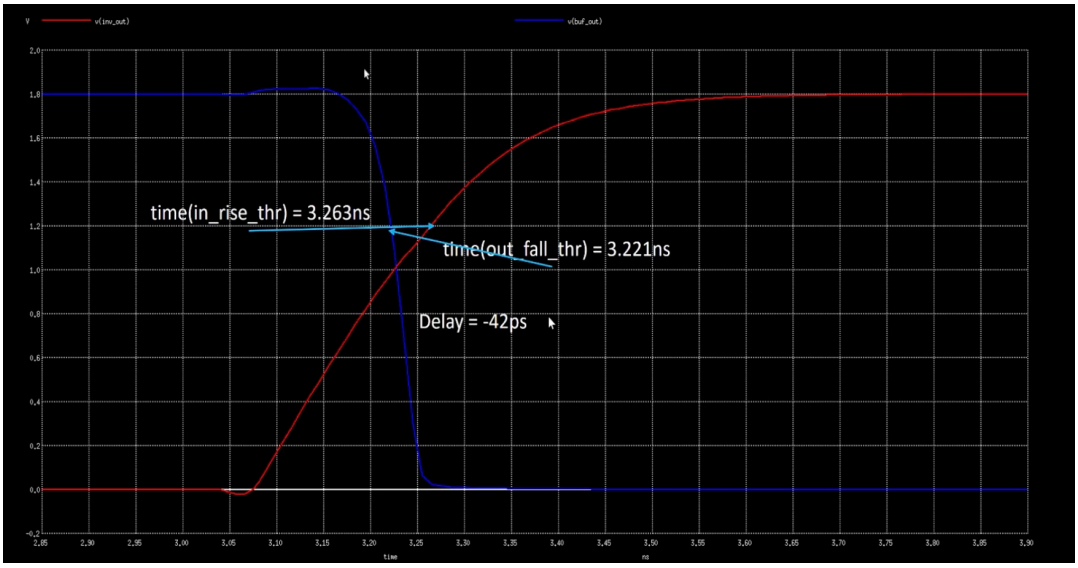


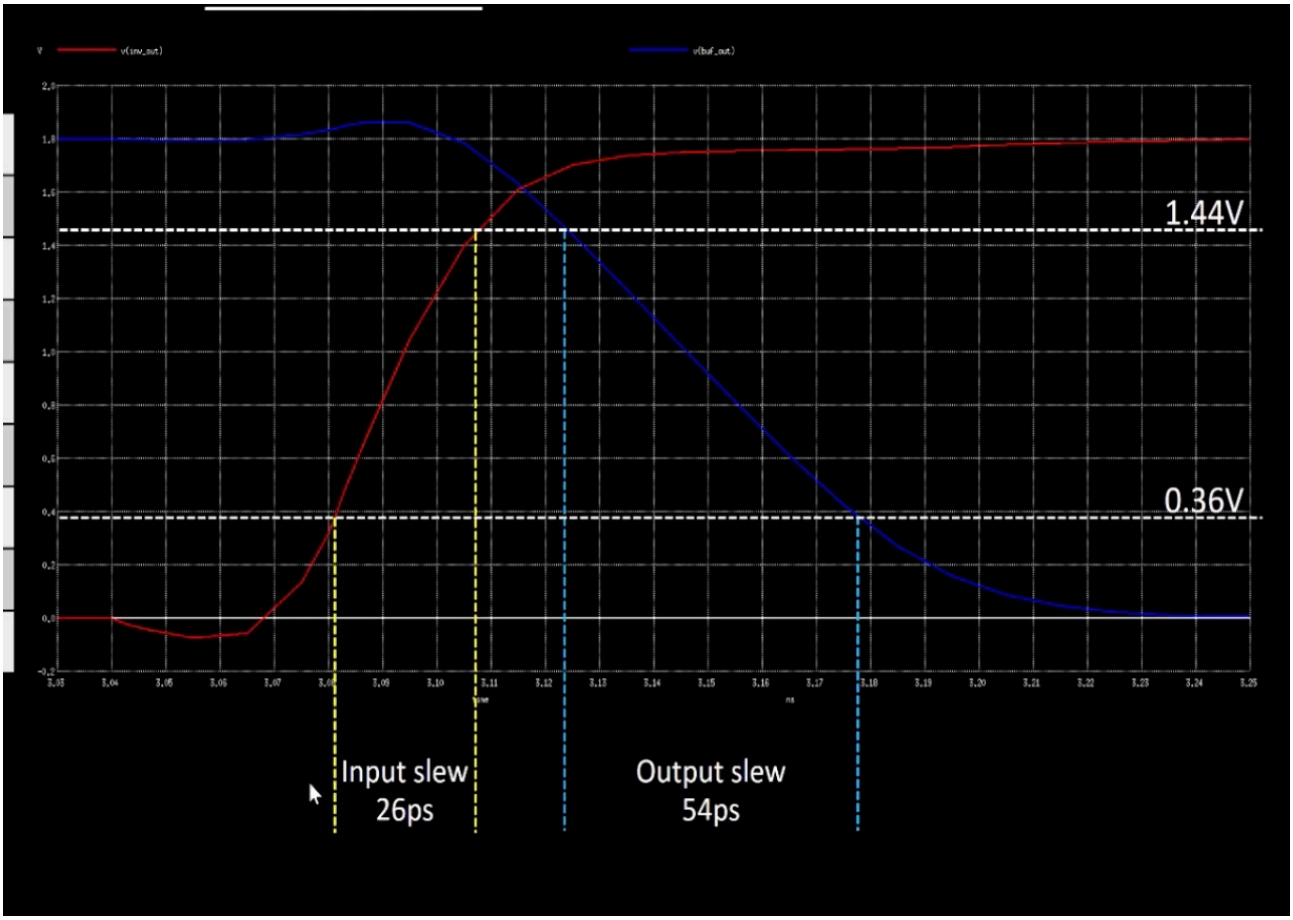
Now, lets start with propagation delay, now this may sound simple, but if this is not taken care of, the circuit will have a lot of problems later on. The way to calculate propagation delay is = time(out * thr) - time(in * thr) . Here, we will look at an example



Here, as we applied our equation here, we get an delay of 23 ps (picosecond). Now this is an gooddelay. But when the time(in_rise_thr) goes a bit higher than the time(out_fall_thr), we get a negative delay like below, it does not make sense as the output will come before the input. It is like me trying to turn a lamp and it already turns on before I touch the switch, like, this is an error.

So, we always try to keep the propagation delay positive as much as possible



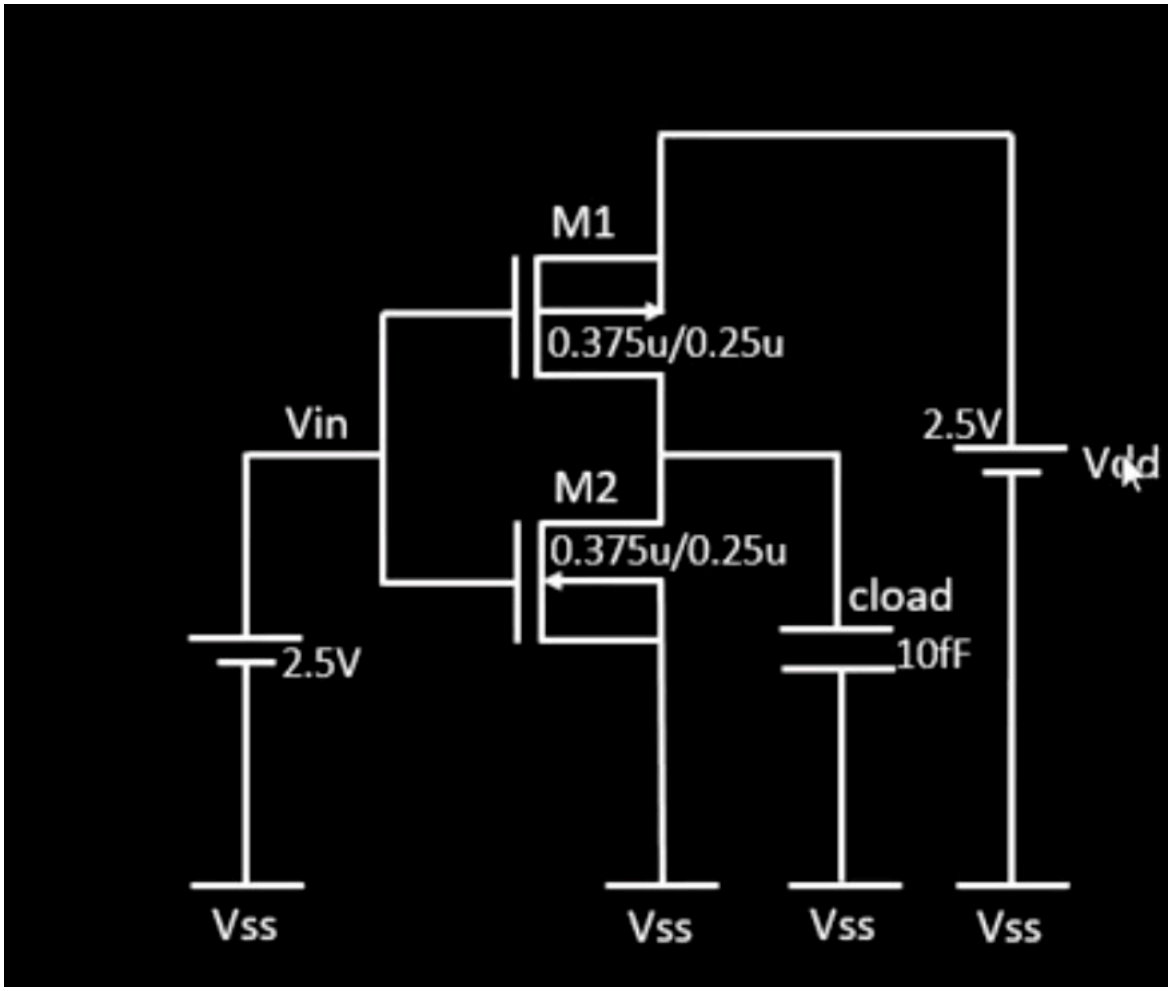


Another example given here for slew delay, where the delay is -8 ps

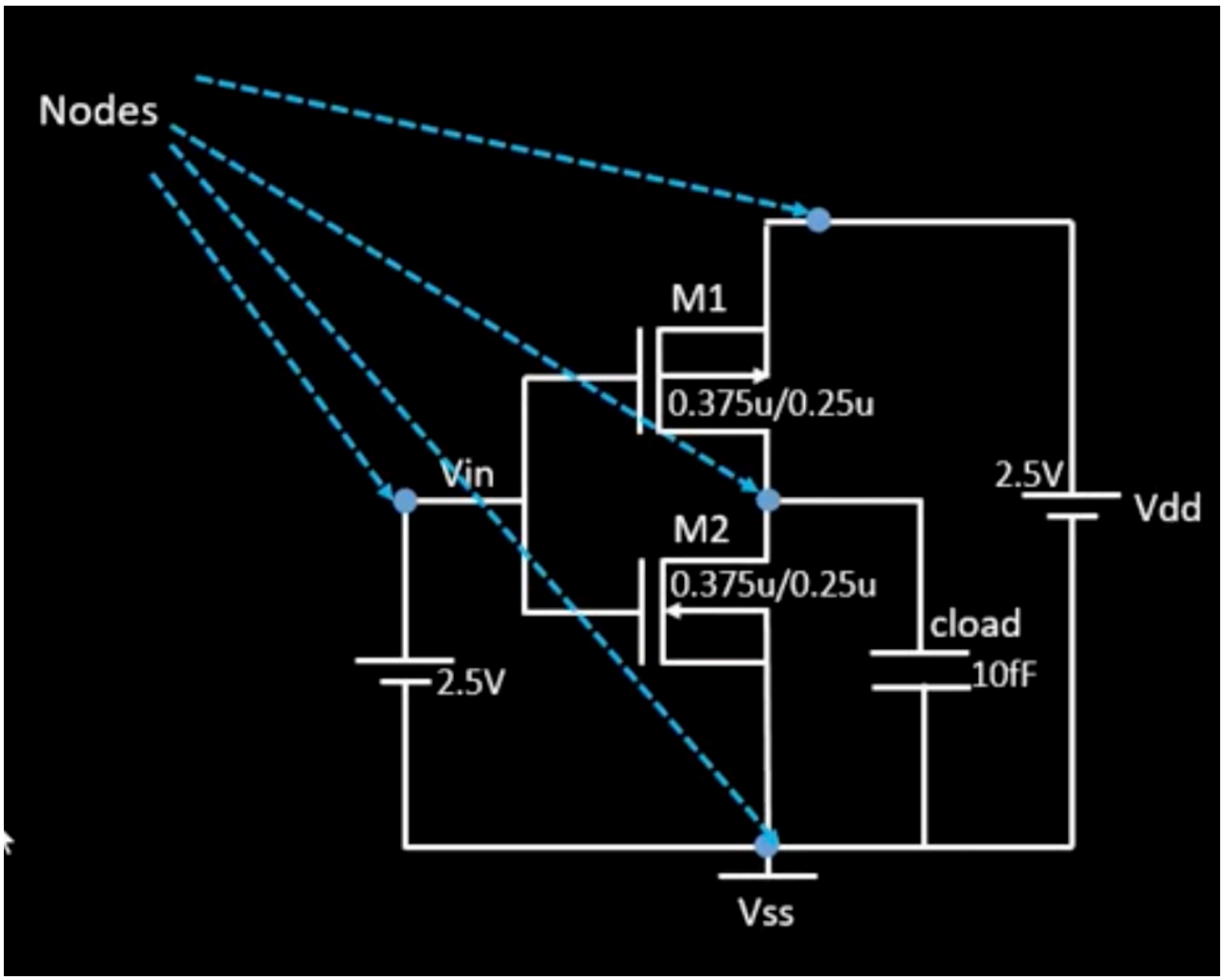
Sky130 Day 3

Now, lets learn about revision for the placement of I/O pins. This is one of the many features of openlane which allows us to make changes on the fly. We can type the command set ::env(FP_IO_MODE) 2, for this. Normally, if the variable is one, it is randomly equidistant. But here two means they are all stacked on top of each other. This how we can use openlane to make changes to anything on the fly.

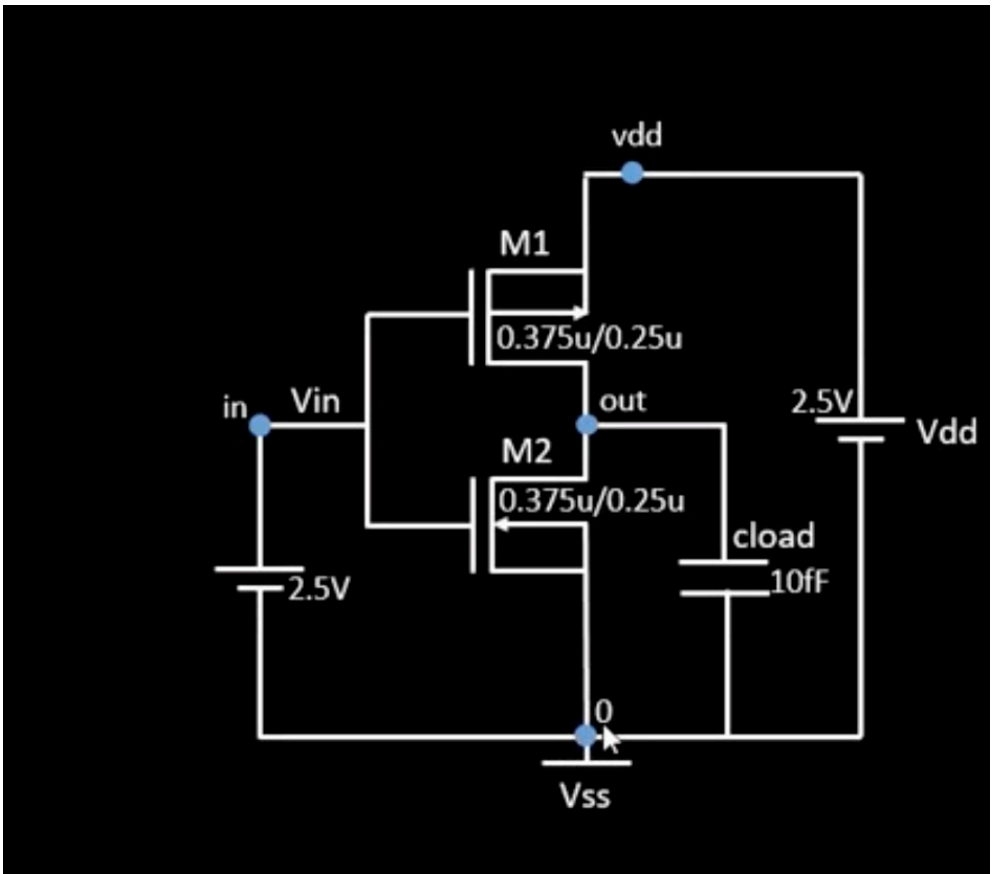
Now, lets learn how to create a Spice deck , as we are going to do, most of our labs on the inverter example we have seen in Day 2. The Spice deck for a component, describes it's connectivity, it component values,



Now, we can see all the components. So now, the next step is to identify nodes. A node is a point where in between them a component is located.



Now, that we have located the nodes, lets name them (how to place these nodes will be shown later in the document)



So, now that we have our nodes named, lets start writing our spice deck so that we can run labs later. This is what our spice deck looks like

```
*** MODEL Descriptions ***
*** NETLIST Description ***
Ml out in vdd vdd pmos W-0.375u L=0.25u
M2 out in 0 0 nmos W=0.375u L=0.25u
cload out 0 10f
Vdd vdd 0 2.5
Vin in 0 2.5
*** SIMULATION Commands ***
.op
.dc Vin 0 2.5 0.05
*** .include tsmc 025um model.mod
***
.LIB "tsmc 025um model.mod" CMOS MODELS .end
```

Before we run the SPICE simulation, there are various steps-:

I. Open the NGSPICE simulator

II. Source the Circuit File through source command

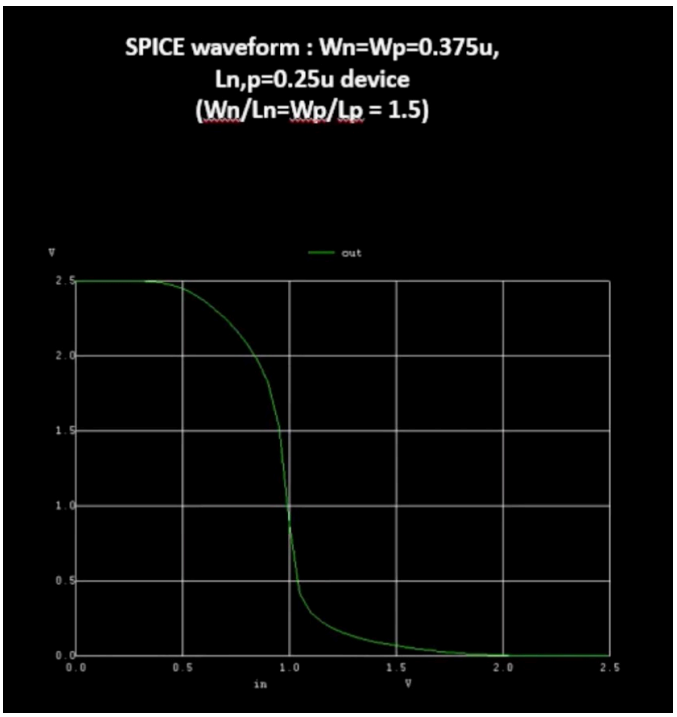
III. Execute it by the command run and then use setplot which allows one to view any plots possible from the simulations specified in the spice deck and will give you a choice for which simulation to be run

IV. Then, type display which will give you a choice of nodes to be plotted which when plot out vs in is typed will be plotted on a graph.

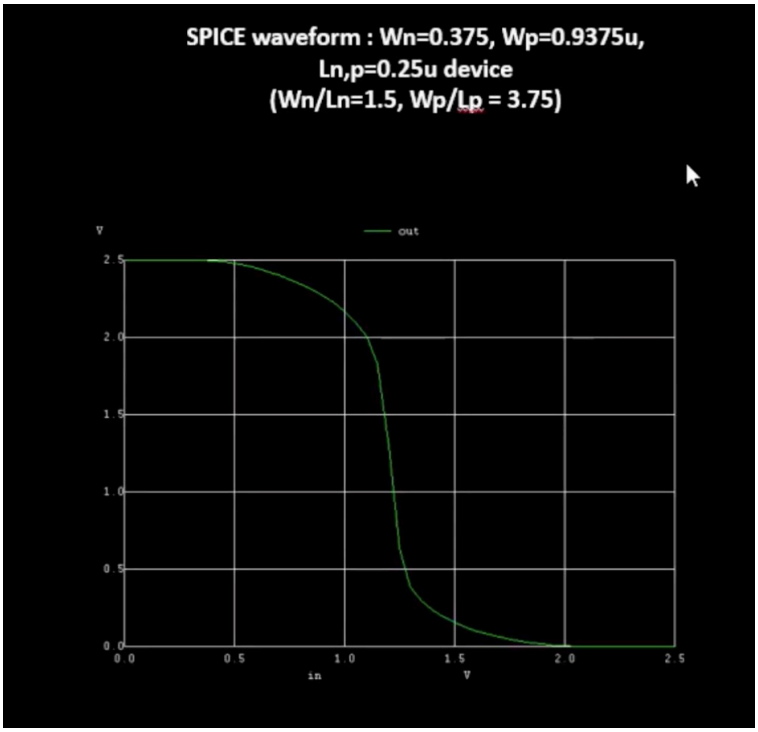
It is to be noted that here , the NMOS and the PMOS is the same size. But, most of the time, the PMOS is 2 or 3 time bigger than the NMOS. Now, there are certain use cases where the PMOS is the same size as NMOS, and the other use cases, the PMOS is bigger than the NMOS.

Now, lets run a SPICE simulation on this. Now, the input is swept from 0 V to 2.5 V with a step of 0.05 V. This how the output graph looks like.

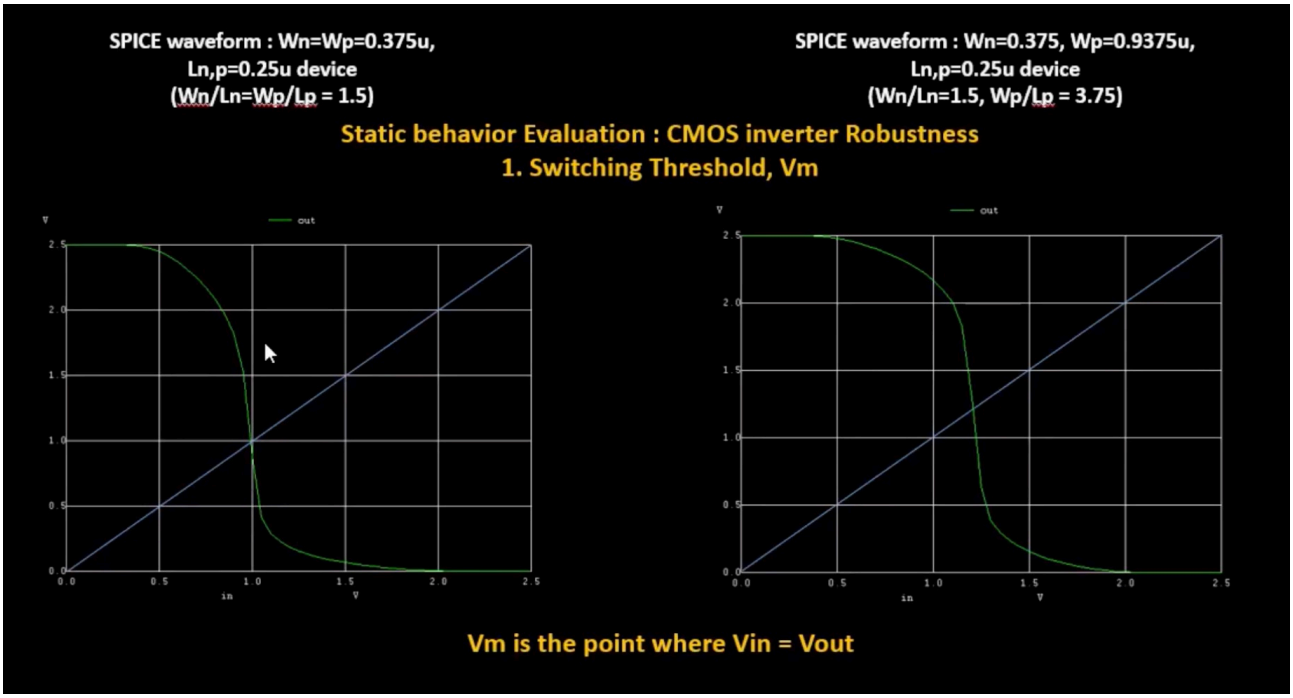
(In the next page)



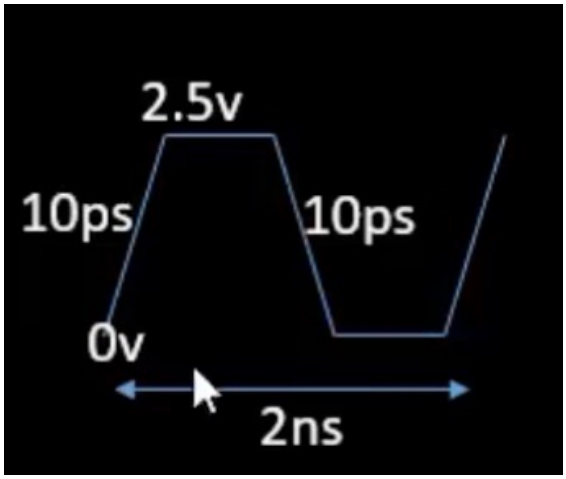
Now, lets see a situation where the PMOS is bigger than the NMOS, as talked about before.

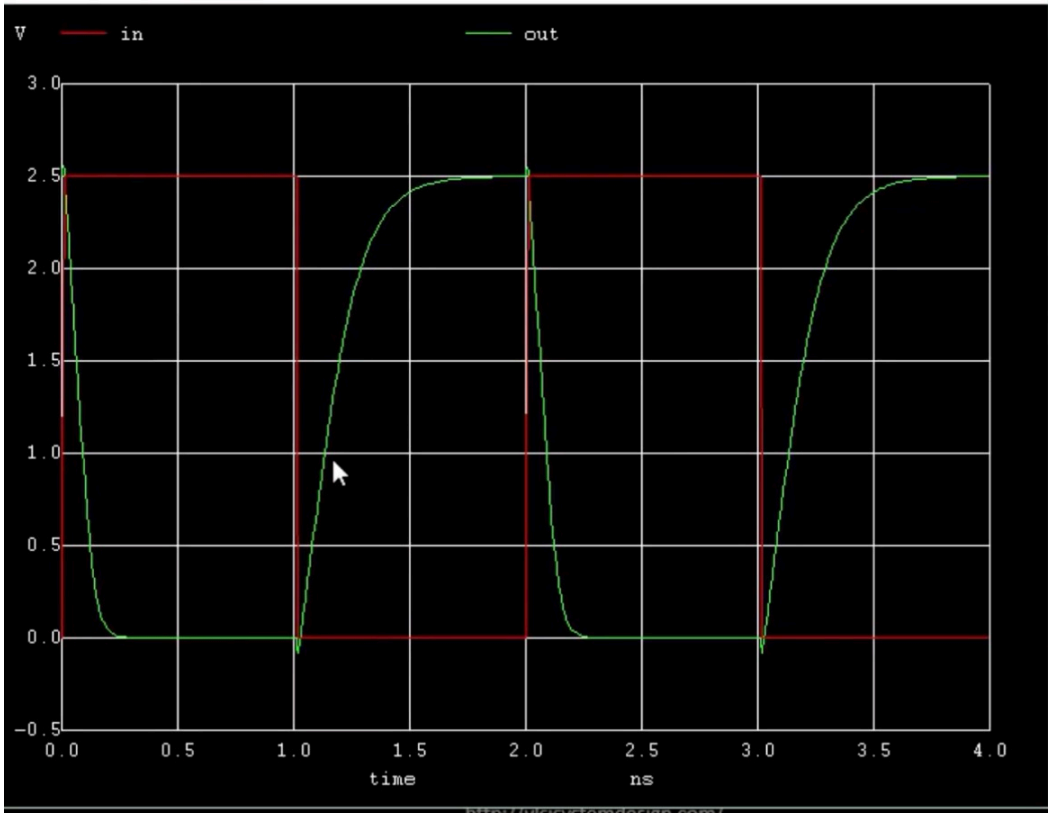


We can see that the graphs are almost the same! Because CMOS architecture is incredibly robust and that is why the graphs look almost the same. Now, the first step to test the CMOS inverter robustness, we cd run Switching threshold vm, where vm is the point where vin = vout



Now, how do we find the vm? We use a process known as DC transient analysis. Now, we will supply the following waveform as the input signal for our CMOS inverter. Now, lets look at the output graph for our CMOS inverter. The below is the input waveform





Now, this is the graph of the input vs output. Here we can zoom in and find out the delay. After our calculations, the rise delay = 148 ps and the fall delay = 71 ps

Next lets import the design we have created into openlane, cloning the cell design into openlane. Now here is how to do it. The cell design is from https://github.com/nickson-jose/vsdstdcelldesign .

The first step is to clone the design into openlane using the command

Git clone https://github.com/nickson-jose/vsdstdcelldesign

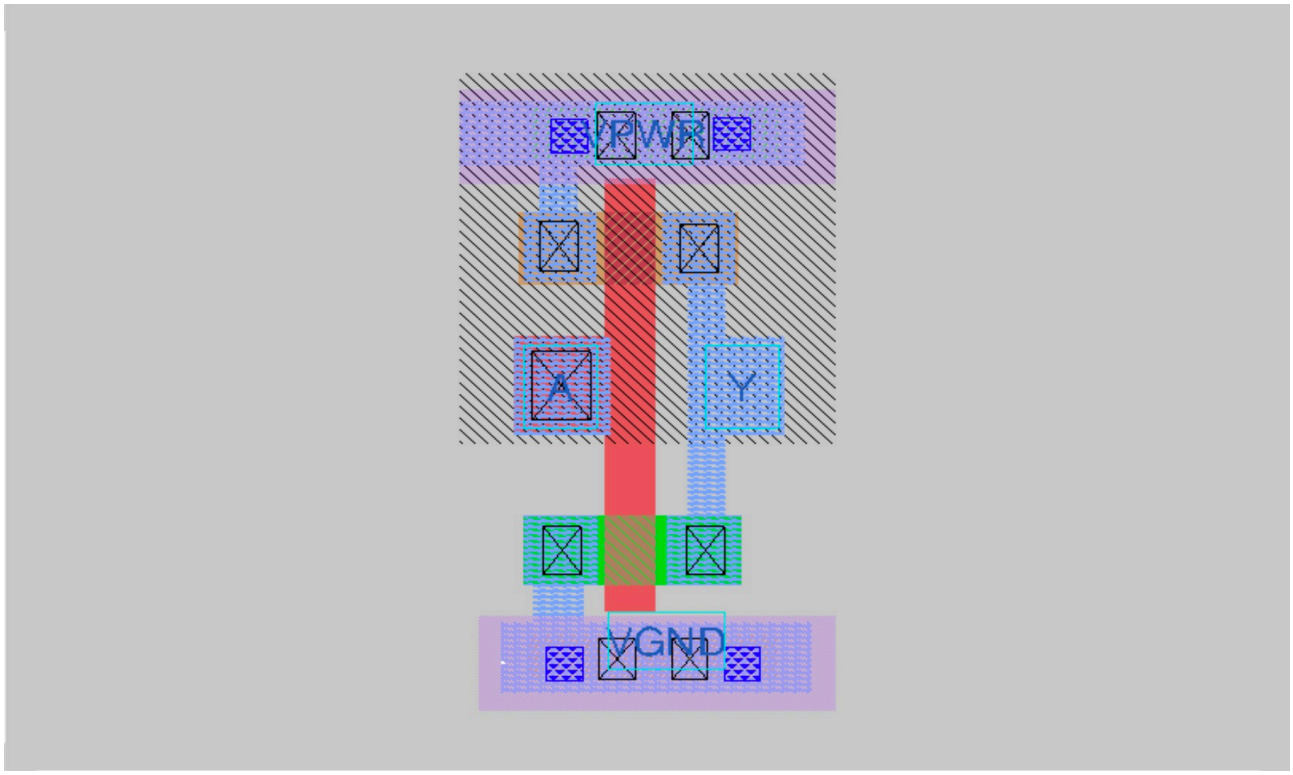
The next step is to move the magic file from the pdfs folder to the vsdstdcelldesign folder. The code is

cp sky130A.tech /home/vsduser/Desktop/work/tools/openlane_working_dir/openlane/vsdstdcelldesign/

Now, we can see the layout in magic with the command

magic -T sky130A.tech sky130.inv_mag

The layout looks like this



Now, lets look at the 16 mask CMOS process

1. ***Selecting a substrate***. The substrate should be lesser doped than the wells in it

2. ***Creating a Active region:-*** for the transistors by depositing 40 nm of silicon dioxide , 80 nm of Si3N4 and creating 1 micrometer of photoresist.

2a. We apply a mask (mask 1) so that it will be able to protect the areas that are needed from the chemical reaction, etching etc

2b. We etch off the extra Si3N4

2c. We remove the resist chemically

2d. We place the entire thing into an oxidation furnace so that wells are formed

2e. We etch off the excess silicon nitrate using hot phosphoric acid

2f. Now we are left with wells which prevent the transistors from communicating with each other

3. ***Creating N-well and P-well (continued in next page)***

3a. Apply a layer of photoresist and put a mask (Mask2) for the preparation of the P-well

3b.now, be shine UV light at it to remove the photoresist that we don't need.

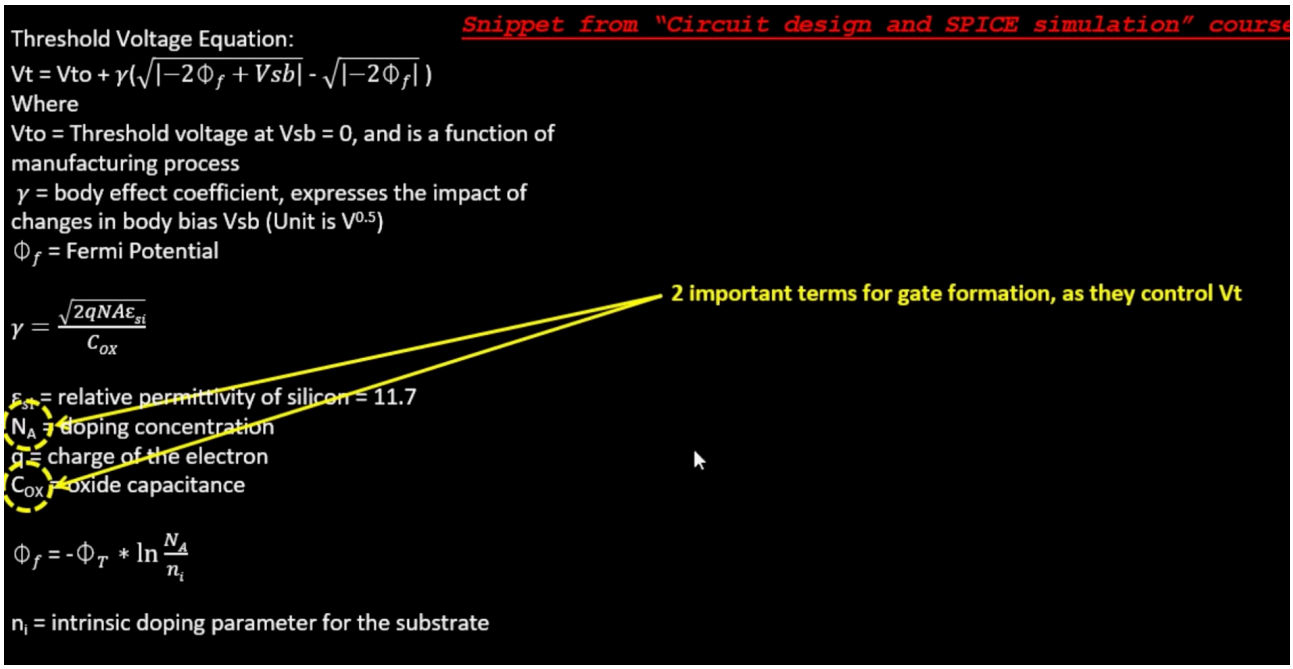
3c. We then shoot boron which is used for doping a substrate into P-type.

3d. We have successfully created the P-well in which we will be putting our transistors in

3e. For N-well , we repeat all the steps above, using mask 3 and phosphorus as phosphorus is used create n-type substrate

3f. After creating the n and p wells we use a driving furnace to deepen these wells

4. ***Creation of gate terminal***- gate terminal controls the threshold voltage which determines the input voltage.



4a. First we deposit a photoresist layer to define the areas that need to be protected. And then we deposit Mask -4 . Then we apply UV light then the exposed photoresist is removed

4b. We then shoot low energy boron too the surface of the P-well to create a layer to control the threshold.

4c. We then we use Mask -5 and Arsenic to create the same thing for the N-well

4d. We create a poly silicon layer of about 400 nm to create a low resistance region. Before this we etch off the original oxide layer with dilute hydrochloric acid.

4e. Next we shoot phosphorus or arsenic to create a N-type low resistance layer for the gate terminals

4f. We deposit Mask 6 and etch off the remaining poly silicon layer.

5. ***Lightly doped drain (LDD) formation*** - We form this because of two reasons. That being Hot electron effect and Short channel effect. Hot electron effect is where the high energy carriers break the Si-Si bonds passes the 3.2eV barrier between the conduction band and the SiO2 conduction band. Short electron effect can cause the gates to malfunction, which is obviously problematic.

5a. We use mask 7 and mask 8 for NMOS and PMOS (which are lightly doped) respectively

5b. We deposit heavily doped N-type layer on the lightly doped P-type and then we do the same thing for the lightly doped N-type using heavily doped p type.

5c. To protect the lightly doped P-type regions, we create a layer of silicon dioxide and we create spacers in them using plasma anisotropic etching

6. ***Source and Drain formation***

6a. We first add a thin screen oxide layer to avoid channeling which is when the drain digs too deep into the substrate and then it becomes problematic again.

6b. We deposit Mask 9 for N+ -type implementation and Mask 10 for P+ - Type implementation.

6c. The side wall spacers maintain the N-/P- while implementing the above

6d. High temperature annealing is done

7 ***Local interconnect formation*** :- these interconnects are very important as the help in controlling the electrical characteristics. They are only accessible by the user.

7a. The thin screen oxide is removed for opening up the source, drain and gate for contact building. Titanium is used because it has low resistance.

7b.We use Titanium Diselenide i.e Ti2Se2 for local interconnects.

7c. Mask 11 is formed and the titanium Nitride TiN is etched off by RCA to make the first level contact

8.***Higher Level Metal Formation***

8a. The previous steps in the MASK process have created an uneven surface layer. So we use a layer of Silicon Dioxide [SiO2] doped with phosphorous or boron known as phosphosilicate glass and borophosphosilicate glass is deposited on the wafer surface

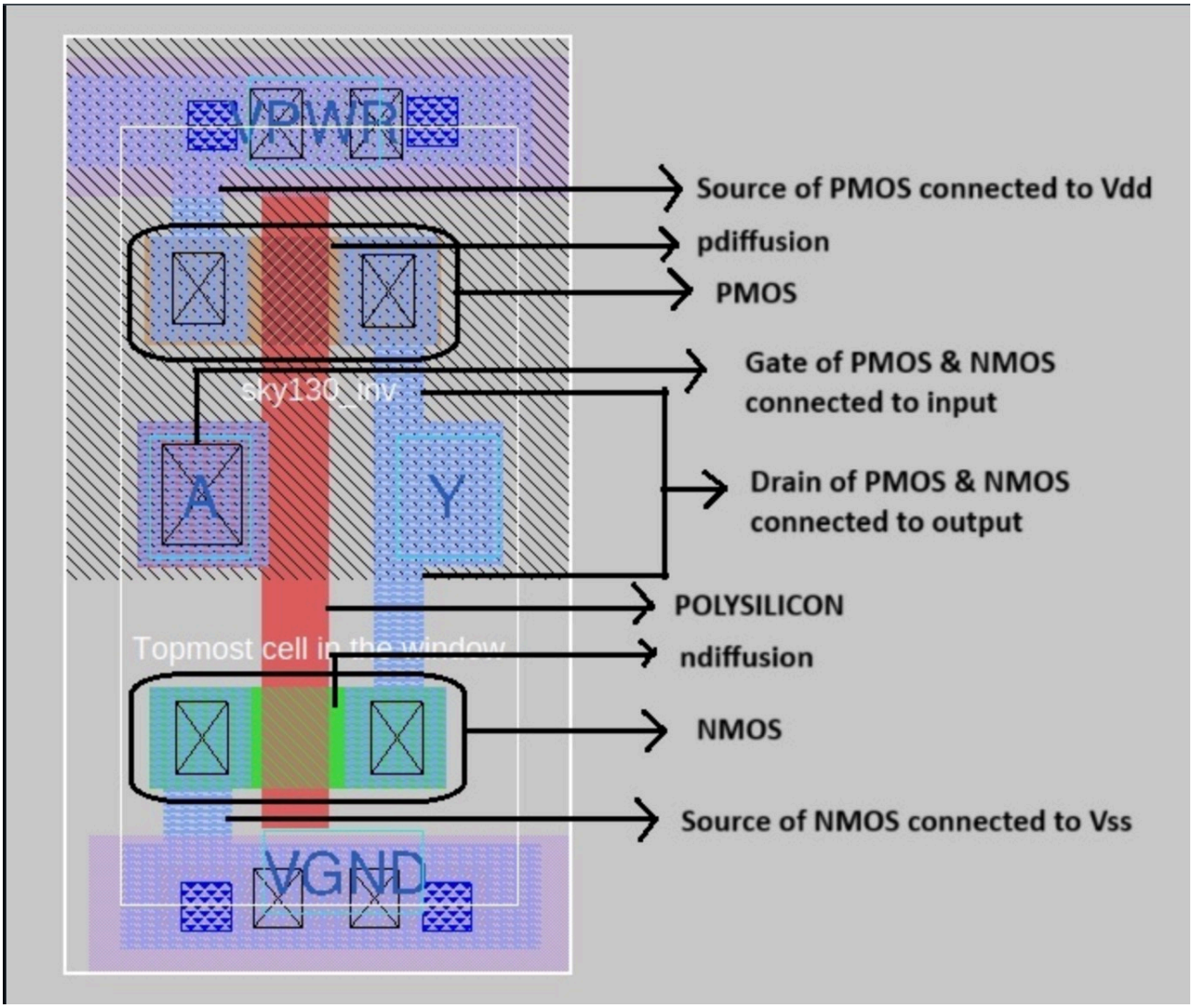
8b. Then the surface is levelled using CMP short form for Chemical mechanical Polishing.

8c. We drill the contact holes using photolithography.

8d. We use various masks such as:-

- 1.Mask 12 is created for the first contact holes
- 2.Mask 13 is used for the first Aluminum contact layer, which the contact holes are connected to.
- 3. Mask 14 creates the second contact holes
- 4. Mask 15 is similarly, for the second Aluminum contact layer
- 5. Finally, we use Mask 16 for making contact to topmost layer

Next, lets get back to our inverter in magic



In sky130A, the first layer is the LDD or local-i and then the m1, m2 layers and so on. The power and ground lines are located in m1. When polysilicon crosses ndiffusion then NMOS and if polysilicon crosses pdiffusion then PMOS is created. The output of the layout is the LEF file, which is used by the router in APR to get the location of standard cell pins for proper routing. So it is essentially an abstract form of the layout of a standard cell.

Next, to extract the SPICE layout for our custom inverter. We use the command in token window

extract all
ext2spice ctheresh 0 rthresh 0

ext2spice

In openlane

vim sky130_inv.ext

To view what is inside the file that as been extracted. Now, we are changing some parameters in the file for the analysis we are going to execute