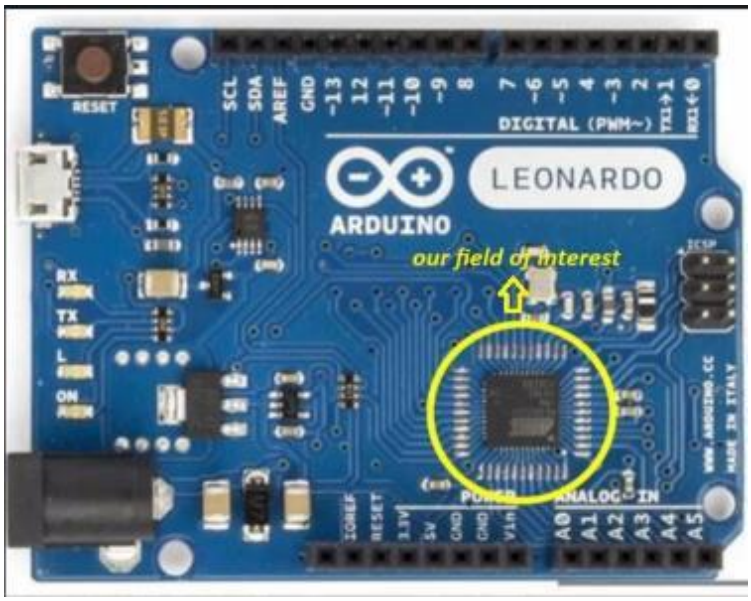
Advanced Physical Design using Openlane/Sky130

Contents:

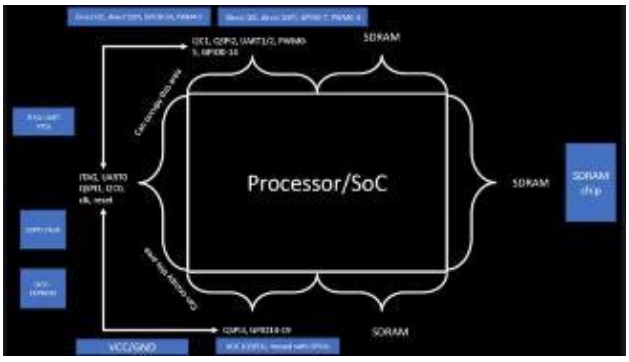
Inception of Open source EDA, OpenLANE and Sky130PDK

Introduction to QFN-48 Package,Chip,pads,core,die and IP's

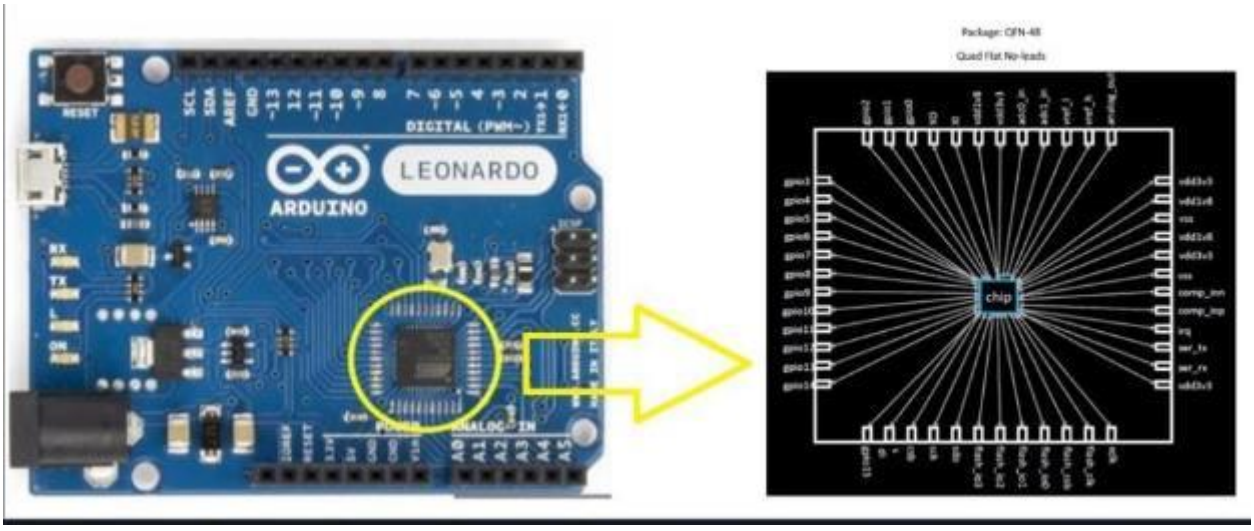
Arduino is a popular open-source electronics platform based on easy-to-use hardware and software. It consists of a microcontroller (originally Atmel AVR family, now expanded to various architectures including ARM) and a development environment for writing software for the microcontroller. Microcontrollers are designed to be compact and contain all essential components on a single chip, including a CPU (Central Processing Unit), memory (both RAM and ROM), timers, and I/O (Input/Output) ports. Microcontrollers are typically programmed using high-level languages such as C or C++, as well as specialized integrated development environments (IDEs) provided by the microcontroller manufacturers. Program code is typically stored in non-volatile memory (ROM or flash memory) and executed by the microcontroller's CPU.



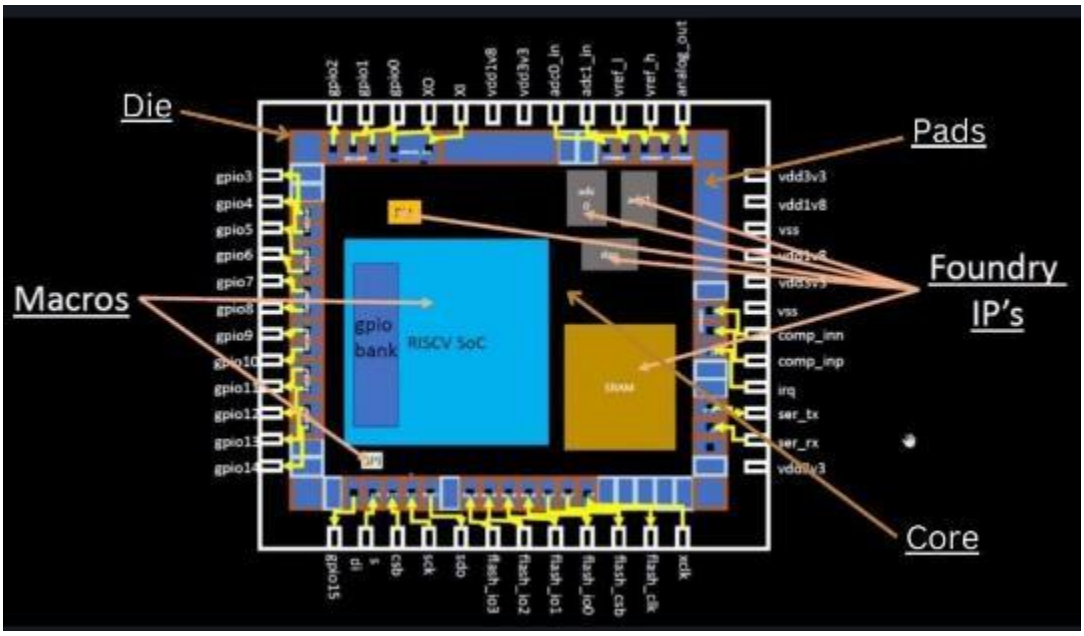
The circled picture in the above image describes about the processor and its connectivity.



The below picture refers to Package. The pin placements are determined by the Arduino board under the development. The connectivity between the chip and the package are illustrated through the wires which determines the transmission of signals in and out of the chip.



Components:



1. **Pads**:
    - Pads refer to the interface between the chip and the outside world. They are the connection points on the integrated circuit (IC) where signals enter and exit the chip. Pads are typically arranged around the periphery of the chip and are used for various purposes such as power supply, ground, input/output (I/O), and test/debugging.
2. **Core**:

- In semiconductor design, the core usually refers to the central processing unit (CPU) or the main computational unit of a chip. It is the part of the chip responsible for executing instructions, performing calculations, and managing data processing tasks. In more general terms, a core can also refer to a fundamental functional block or module within a chip, such as a processing core, memory core, or graphics core.

3. **Die**:
    - Die refers to a single semiconductor chip that has been fabricated on a wafer during the semiconductor manufacturing process. A wafer typically contains multiple copies of the same chip design, known as die. After the fabrication process is complete, the wafer is diced into individual die, each containing a fully functional semiconductor chip.
4. **Macros**:
    - Macros, short for macrocells or macroblocks, are predefined functional blocks or modules that can be integrated into a larger chip design. These macros are often used to implement commonly used functions such as memory controllers, input/output (I/O) interfaces, or digital signal processing (DSP) blocks. Integrating macros into a chip design can help reduce design time, minimize development costs, and improve overall design efficiency.
5. **Foundry IP's (Intellectual Property)**:
    - Foundry intellectual property (IP) refers to reusable design blocks or components provided by semiconductor foundries for use in chip designs. Foundry IP's typically include standard cell libraries, memory compilers, I/O libraries, and specialized IP blocks optimized for the foundry's manufacturing process technology. These IP blocks help chip designers accelerate the design process, reduce development risks, and take advantage of the foundry's manufacturing expertise.

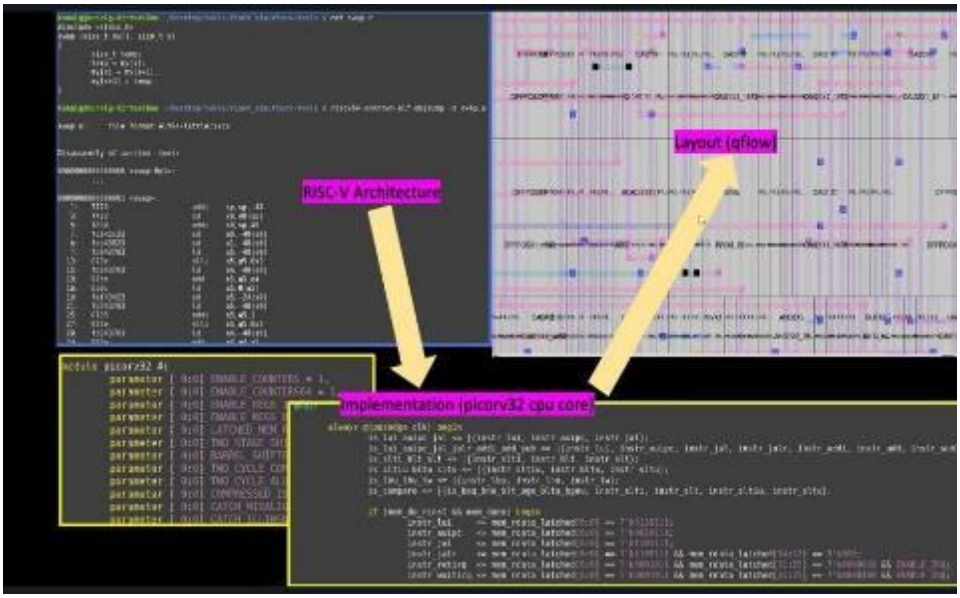
In summary, pads provide the interface between the chip and the outside world, the core is the central computational unit of a chip, die refers to individual semiconductor chips on a wafer, macros are predefined functional blocks used in chip designs, and foundry IP's are reusable design components provided by semiconductor foundries to facilitate chip design and manufacturing.

Introduction to RISC-V:

RISC-V is an open-source instruction set architecture (ISA) based on Reduced Instruction Set Computing (RISC) principles. In order to execute a C-program on a chip with a particular layout,

it has to be compiled into language program like RISC-V. Then this is converted into machine language in a binary format which is easily understood by the computer hardware.

The RISC-V specifications have to be translated to hardware description language(HDL) to pass on layout. So, C Programming is executed and should get automatically executed by the hardware to get the output.



From Software Applications to Hardware

From a software application to hardware, the concept of macros can be understood in various contexts:

1. *Software Application:* In software applications, macros are often used to automate repetitive tasks or to streamline complex operations. For example, in a spreadsheet program like Microsoft Excel, users can create macros to automate calculations, formatting, or data manipulation tasks. These macros can be recorded by the user performing a series of actions and then replayed to execute the same sequence of actions automatically.

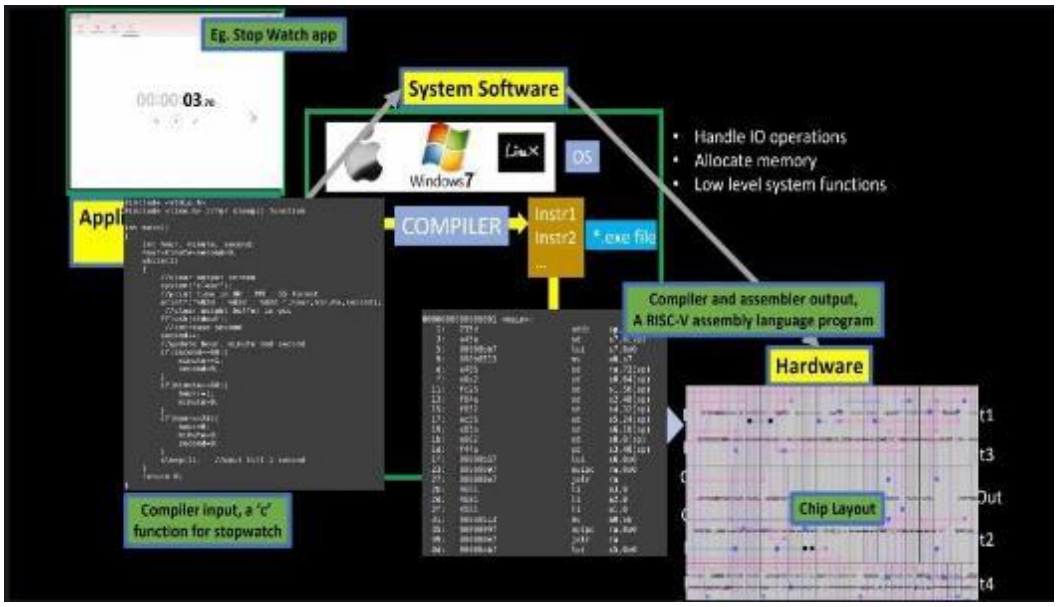
2. *Operating System:* Within an operating system, macros can be used to define system-wide settings or configurations. For instance, in Unix-like systems, shell scripts can include macros to simplify the execution of common tasks or to define environment variables that affect the behavior of various programs.

3. *Programming Languages:* In programming languages, macros are often used to define reusable code snippets or to create domain-specific languages. For example, in the C programming language, macros are defined using the #define directive and can be used to create constants, inline functions, or to implement conditional compilation.

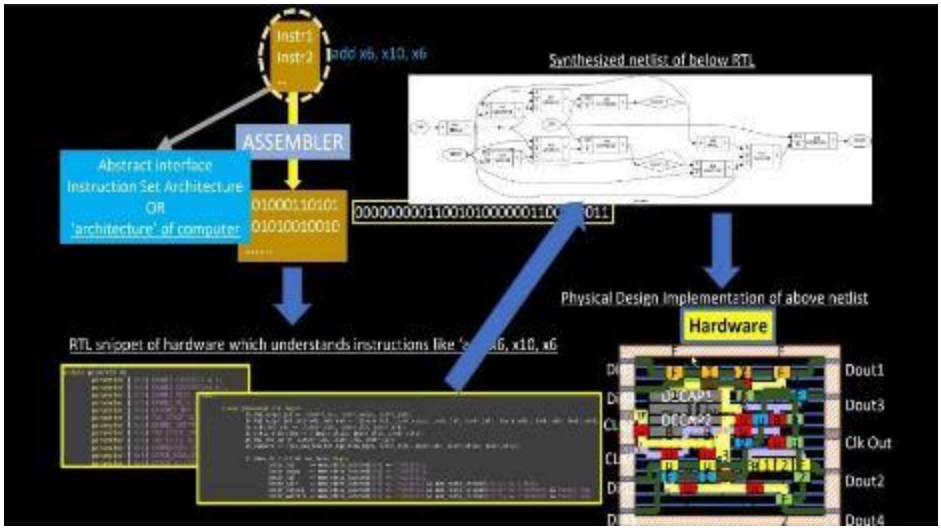
4. *Hardware Description Languages (HDLs):* When it comes to hardware, macros are commonly used in Hardware Description Languages (HDLs) like Verilog or VHDL. In HDLs, macros, often referred to as macros or macros, are used to define reusable hardware components or configurations. These macros can represent complex digital circuits, such as adders, multiplexers, or even entire subsystems. HDL macros are instantiated within larger designs to create hierarchical structures and to facilitate modular design practices.

5. *Electronic Design Automation (EDA) Tools:* In the context of electronic design, macros play a crucial role in EDA tools such as synthesis and place-and-route tools. Macros in this domain represent predefined blocks of hardware that have been optimized for performance, power, or area. These macros can be instantiated multiple times within a design to achieve scalability and efficiency.

Overall, macros bridge the gap between software and hardware by providing a mechanism for abstraction, automation, and reusability in both domains. Whether in software applications, operating systems, programming languages, or hardware design, macros help streamline development processes and improve productivity.



Flow: Stopwatch app to Hardware

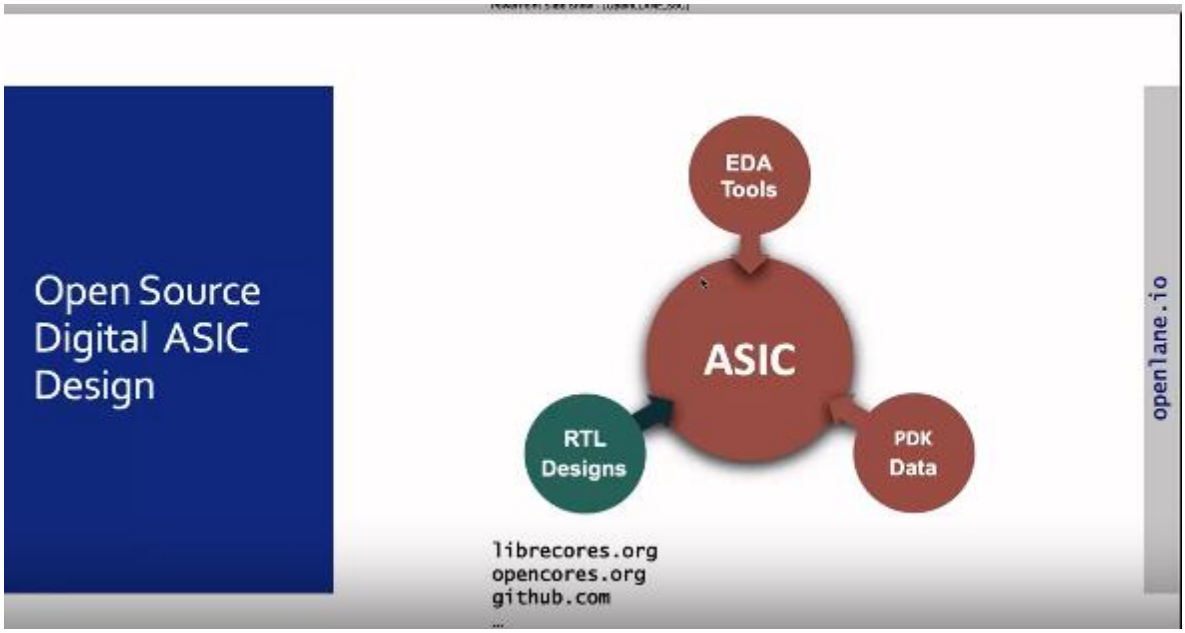


Flow: ISA->HDL->Netlist->Physical design

SoC design and OpenLANE

Introduction to all components of opensource digital asic design:

To design an ASIC, we require RTL designs, EDA tools and PDK data.

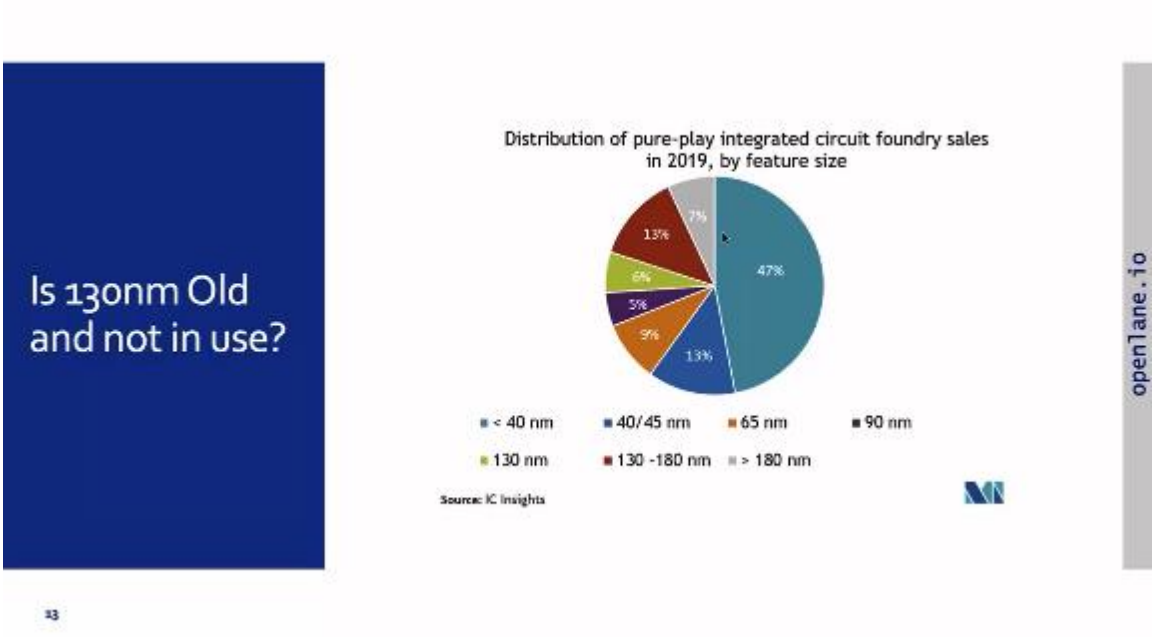


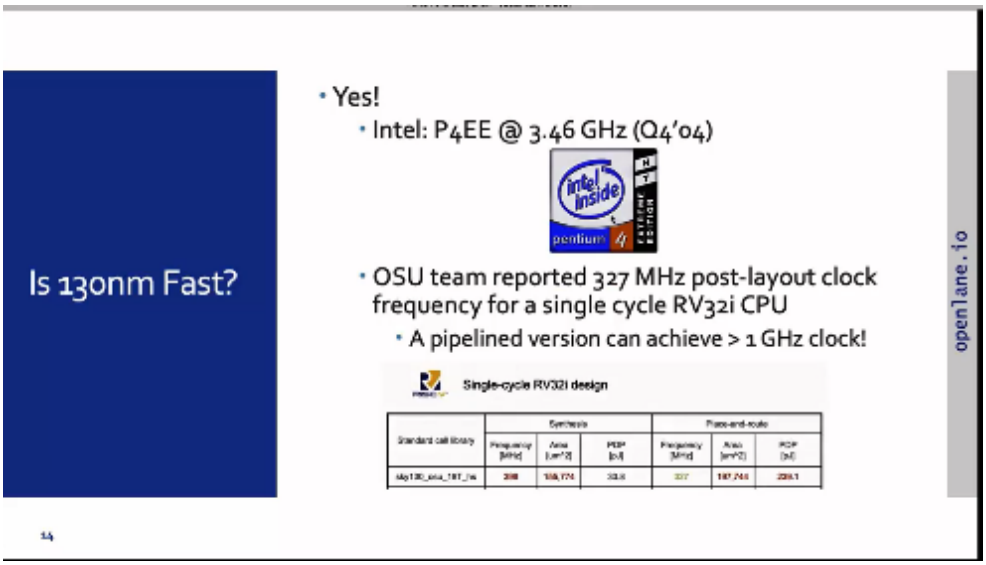
1. **RTL IP's (Register Transfer Level Intellectual Property)**:
    - RTL IP refers to pre-designed and pre-verified functional blocks or modules described at the Register Transfer Level (RTL), which is a level of abstraction in digital design representing the flow of data between registers in a digital circuit. RTL IP's are typically used as building blocks in the design of larger integrated circuits (ICs) or Systems-on-Chip (SoCs). Examples of RTL IP's include processor cores, memory controllers, interface controllers (e.g., USB, PCIe), and specialized accelerators (e.g., DSP, cryptography).
2. **EDA Tools (Electronic Design Automation Tools)**:
    - EDA tools are software applications used by semiconductor and electronic design engineers to design, verify, simulate, and test electronic systems and integrated circuits (ICs). These tools automate various stages of the design process, from logic design and synthesis to physical design, verification, and manufacturing. Common categories of EDA tools include:
        - **RTL Design Tools**: Used for designing and verifying digital logic at the Register Transfer Level (RTL).
        - **Synthesis Tools**: Convert RTL descriptions into gate-level netlists optimized for specific target technologies.
        - **Simulation Tools**: Verify the functional correctness of the design through simulation at various levels of abstraction (RTL, gate-level, transistor-level).
        - **Place and Route Tools**: Determine the physical layout of components on a chip and the routing of interconnects between them.

- **Timing Analysis Tools**: Ensure that the design meets timing constraints and performance requirements.
- **Physical Verification Tools**: Check the design for manufacturing-related issues such as design rule violations, layout errors, and electrical rule violations.

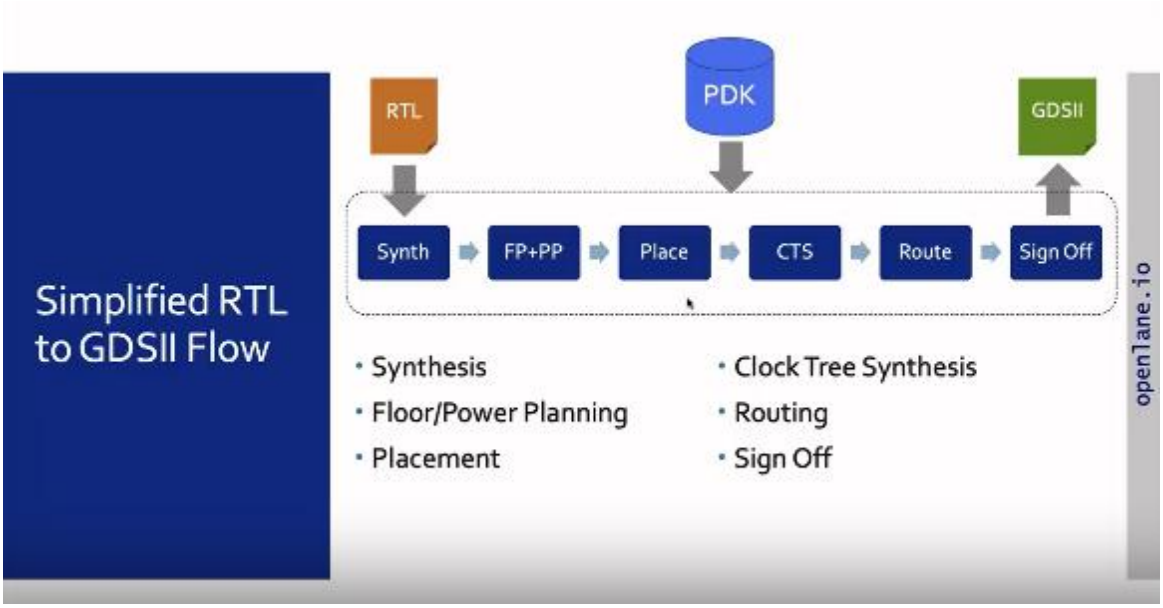
3. **PDK (Process Design Kit)**:
    - A PDK is a collection of files and models provided by a semiconductor foundry that contains information about the manufacturing process technology used to fabricate integrated circuits (ICs). It includes data such as design rules, device models, parameterized cells (PCells), technology files, and simulation models required by electronic design automation (EDA) tools to design and verify ICs. Designers use PDKs to develop chip designs that are compatible with the manufacturing process offered by the foundry. PDKs are essential for ensuring that chip designs meet the fabrication requirements and constraints of the foundry's manufacturing process.

In summary, RTL IP's are pre-designed functional blocks described at the Register Transfer Level, EDA tools are software applications used for electronic design automation, and PDKs are collections of files and models provided by semiconductor foundries that contain information about the manufacturing process technology used to fabricate integrated circuits. These components play crucial roles in the design, verification, and fabrication of electronic systems and ICs.

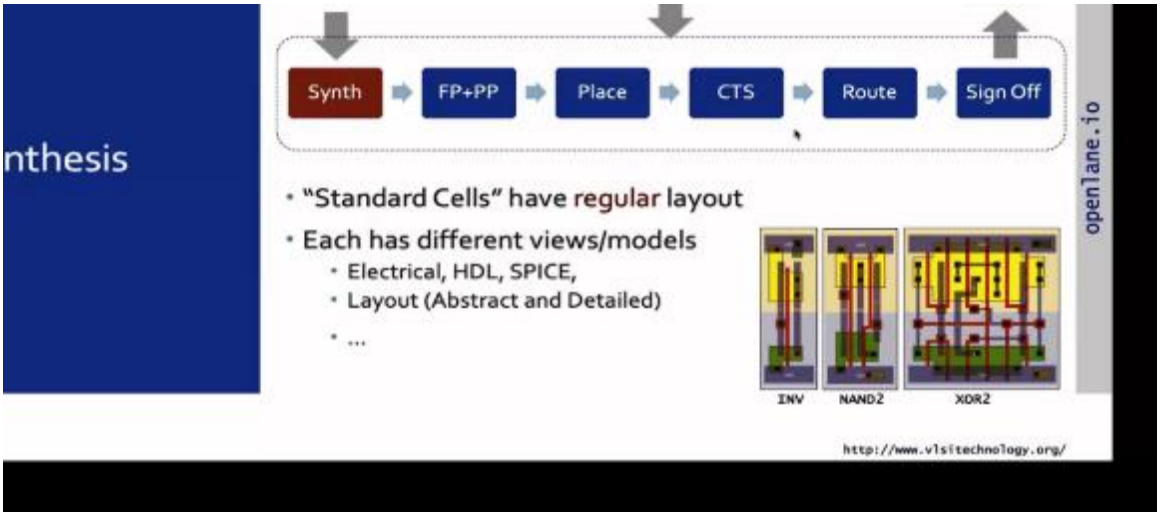




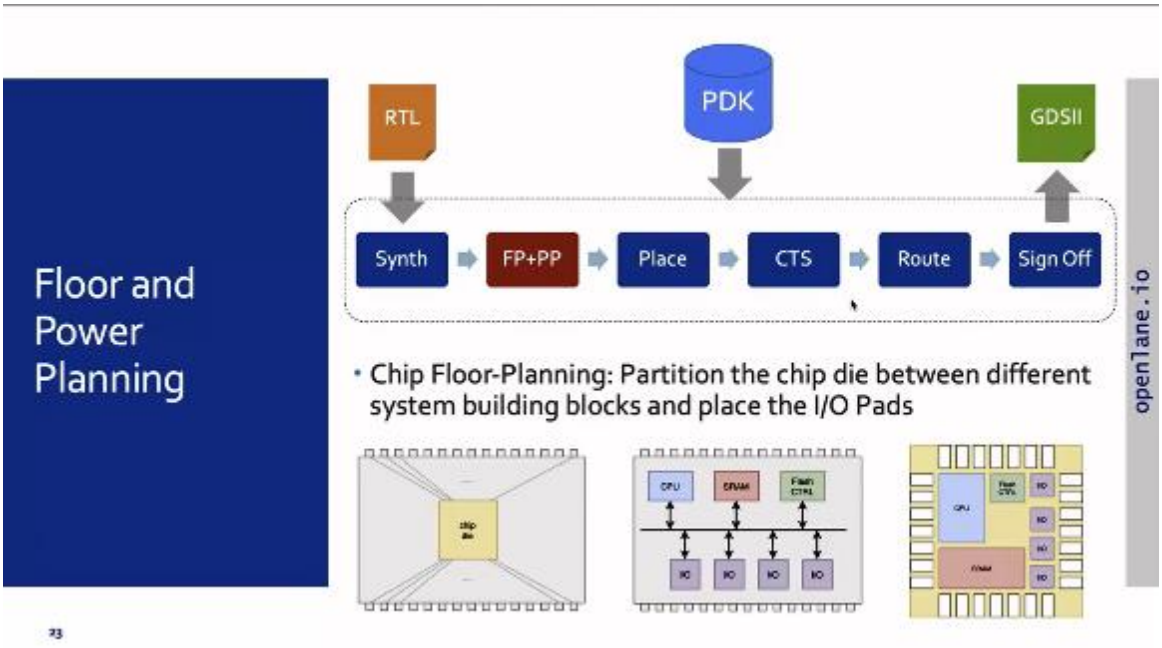
Simplified RTL2GDS flow



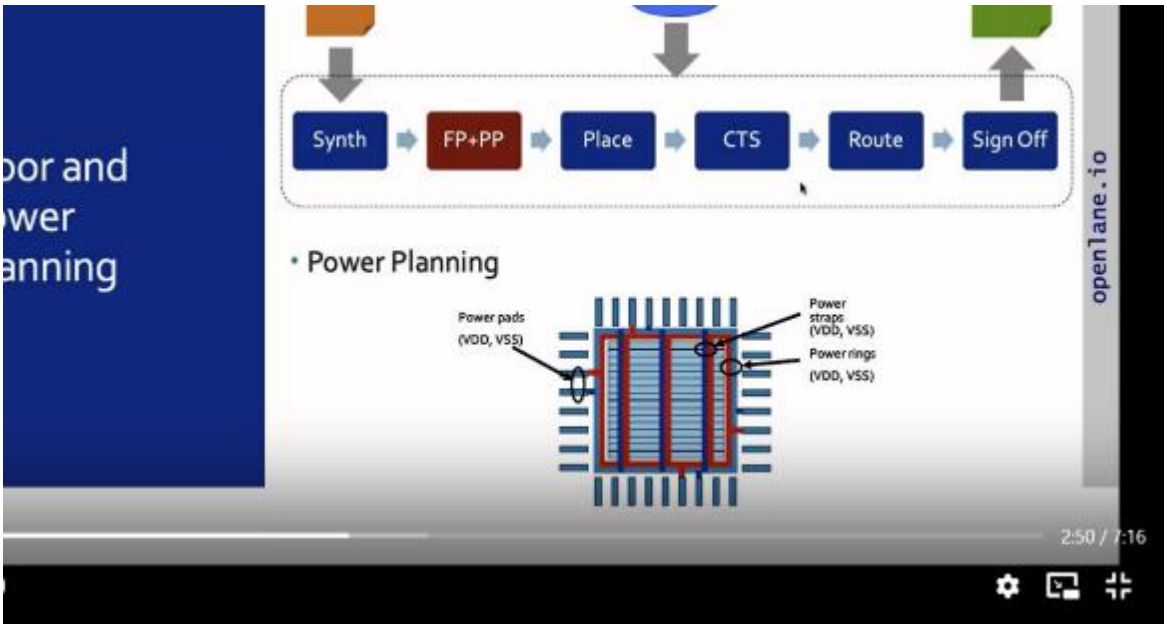
1.Synthesis: In the synthesis stage, the RTL description is translated into a gate-level netlist using synthesis tools. These tools map the RTL code to standard cell libraries provided by the semiconductor foundry. The resulting gate-level netlist represents the circuit in terms of logic gates and their interconnections.



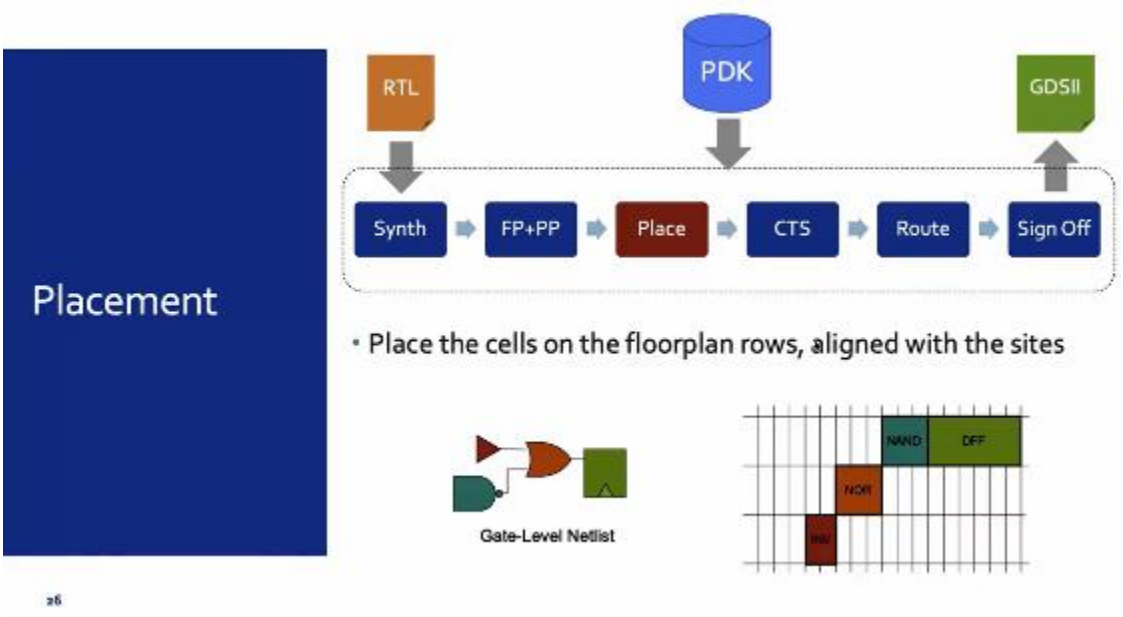
2. **Floorplanning**: Floorplanning involves partitioning the chip area into functional blocks and allocating resources such as memory, logic cells, and I/O pads. The goal is to optimize the physical layout of the chip to minimize signal delays, reduce power consumption, and meet performance targets.

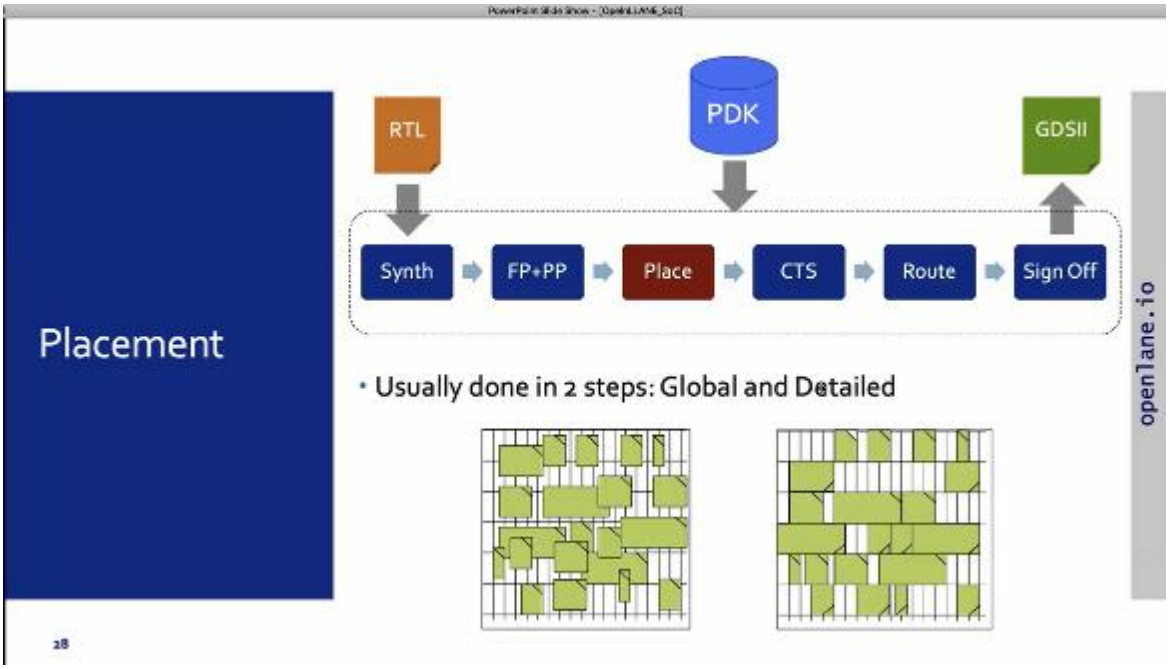


Power planning is a crucial step in the RTL-to-GDS (Register Transfer Level to Graphic Data System) flow, particularly in the physical design stage of integrated circuit (IC) development. Power planning involves the distribution of power and ground signals throughout the chip to ensure reliable operation and efficient power delivery. In the power grid design stage, the chip's floorplan is overlaid with a grid of power and ground rails to distribute power and ground signals uniformly across the chip. Power planning tools determine the optimal placement of power and ground lines to minimize resistance, reduce voltage drop, and mitigate noise.

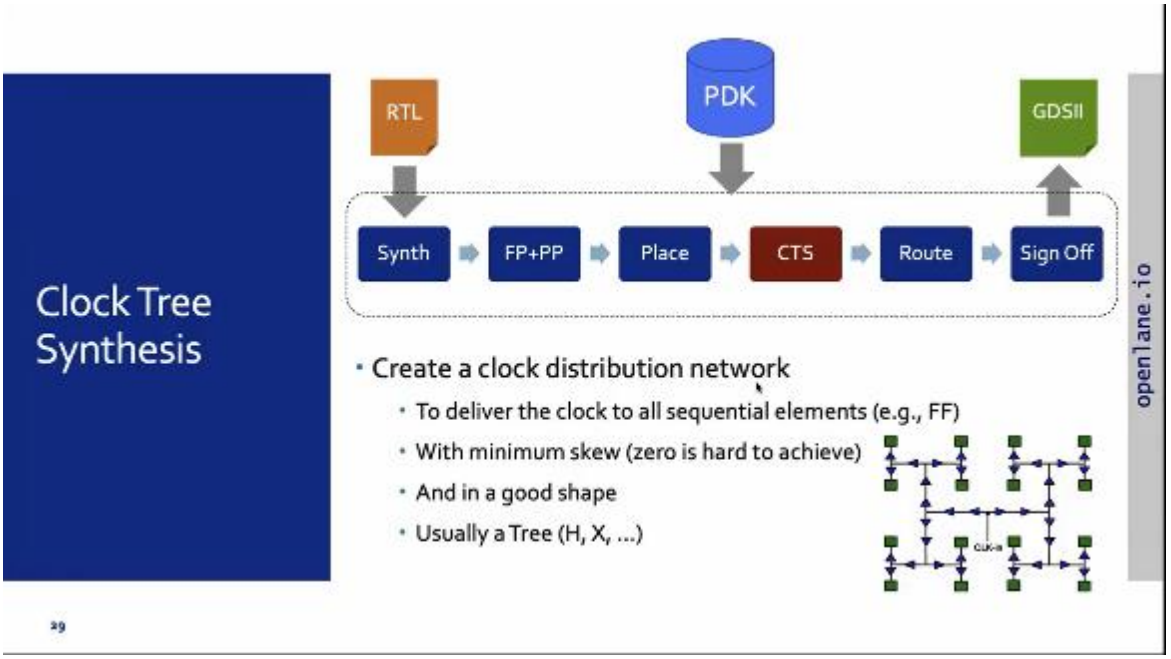


3.placement: In the placement stage, the synthesized and optimized logic cells are placed within the chip's floorplan. Placement tools determine the physical locations of logic cells to minimize wire lengths, reduce congestion, and satisfy timing constraints. Advanced placement algorithms consider factors such as signal timing, power distribution, and thermal effects.



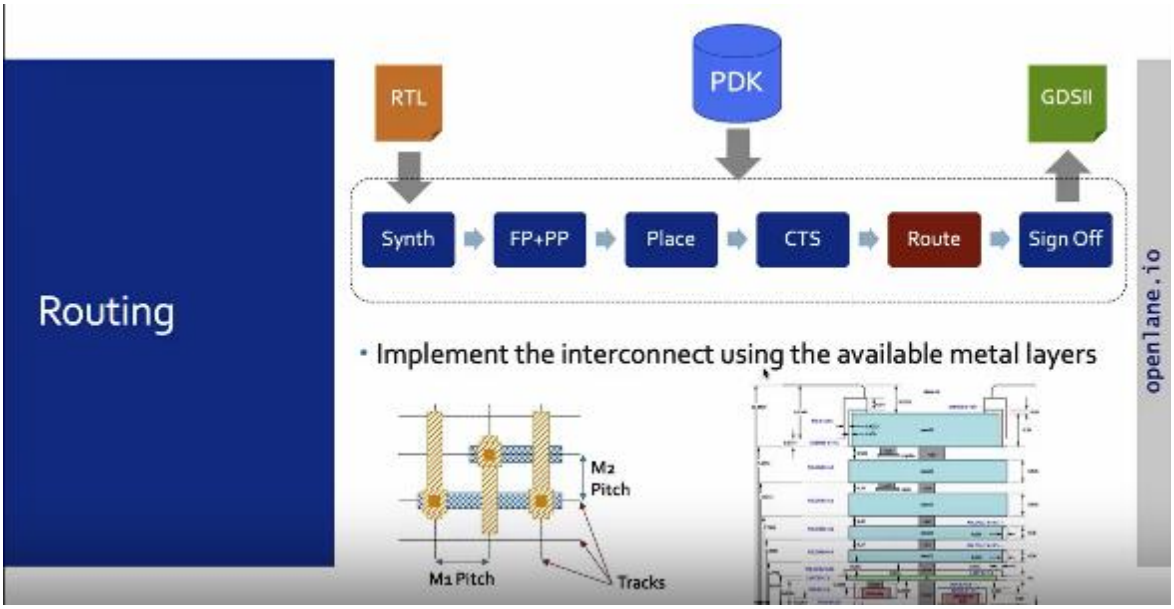


4. **Clock Tree Synthesis (CTS)**: Clock tree synthesis involves the generation of a hierarchical clock distribution network to distribute clock signals uniformly across the chip. CTS tools optimize clock routing to minimize skew, ensure clock signal integrity, and meet timing requirements.



5. **Routing**: The routing stage involves the generation of physical metal interconnects (wires) to connect the placed logic cells according to the synthesized netlist. Routing tools handle the complex task of routing signals while considering design rules, signal

integrity, and timing constraints. Global routing establishes the overall routing topology, while detailed routing handles the routing of individual metal tracks.



1. 6.Sign off: **Functional Verification**:
    - Functional verification ensures that the design behaves correctly according to its specifications. Simulation-based techniques, such as RTL simulation and gate-level simulation, are used to verify the functionality of the design under various operating conditions and test scenarios. Functional coverage analysis is performed to ensure that all functional aspects of the design have been adequately exercised.
2. **Timing Closure**:
    - Timing closure involves ensuring that all timing constraints are met, including setup time, hold time, clock skew, and maximum operating frequency. Static timing analysis (STA) tools are used to analyze the timing paths in the design and identify timing violations. Design optimization techniques, such as logic restructuring, clock tree synthesis, and placement and routing optimizations, may be applied to achieve timing closure.
3. **Power Integrity Analysis**:
    - Power integrity analysis ensures that the power distribution network (PDN) meets the design's power delivery requirements and maintains stable power supply voltages. Analysis techniques, such as power grid integrity checks, voltage drop analysis, and electromigration analysis, are performed to identify and mitigate power-related issues. Decoupling capacitors and power gating techniques may be used to improve power integrity.
4. **Physical Verification**:

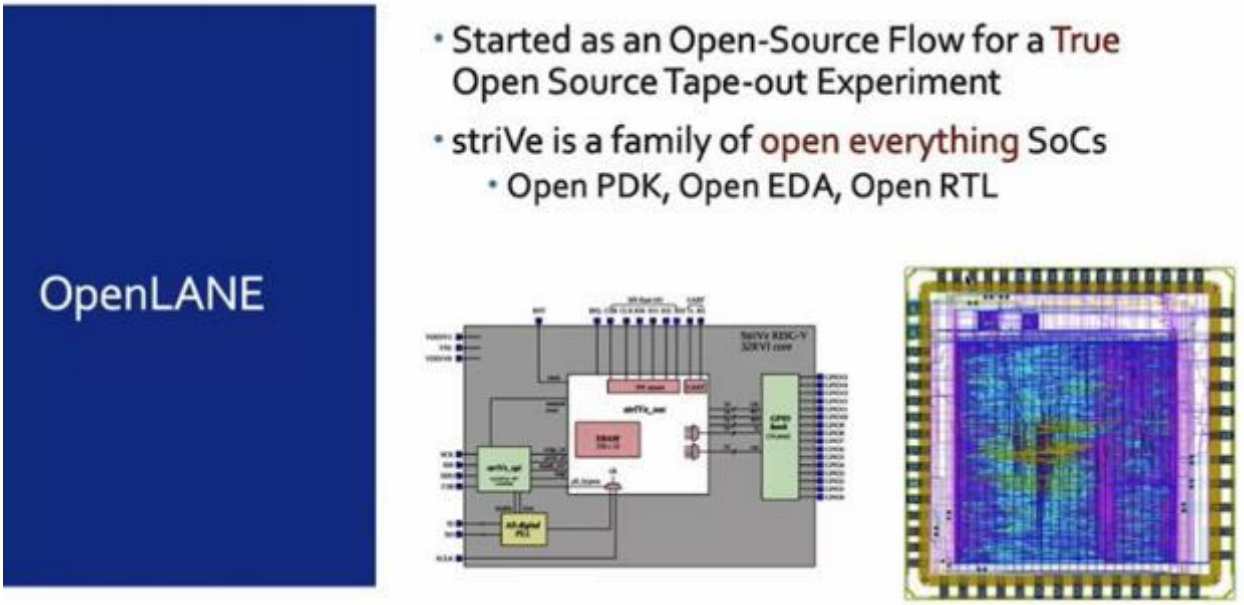
- Physical verification checks ensure that the design adheres to design rules, manufacturing constraints, and reliability requirements. Design rule checking (DRC) and layout versus schematic (LVS) checks are performed to identify and correct layout errors, such as spacing violations, overlapping geometries, and missing connections. Electrical rule checking (ERC) verifies the electrical integrity of the design, including signal integrity and power distribution.

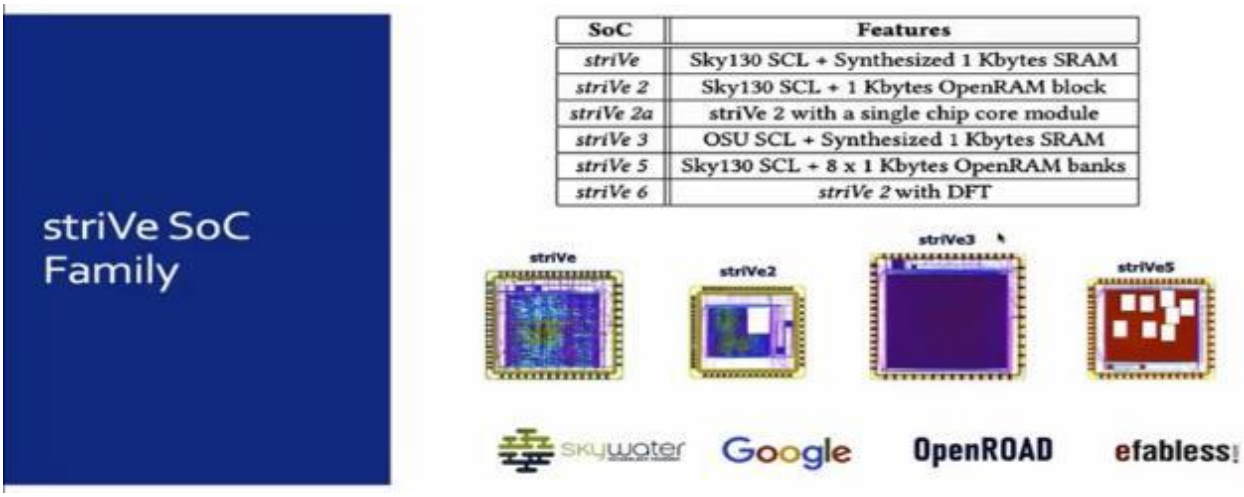
5. **Design for Manufacturability (DFM)**:
    - Design for manufacturability (DFM) checks ensure that the design is manufacturable with high yield and reliability. DFM techniques, such as lithography simulation, metal fill insertion, and manufacturing variability analysis, are employed to identify and address potential manufacturing issues, such as lithography hotspots, electromigration, and process variations.
6. **Reliability Analysis**:
    - Reliability analysis evaluates the design's long-term reliability under various operating conditions and stresses, such as temperature variations, voltage fluctuations, and aging effects. Techniques, such as electromigration analysis, stress testing, and accelerated life testing, are used to assess the design's reliability and identify potential failure mechanisms.
7. **Documentation and Release**:
    - Once all sign-off criteria have been met, the design is documented, and the final design database, including the GDSII layout file, is released for manufacturing. Design documentation typically includes a sign-off report summarizing the results of all verification and analysis activities, as well as any design constraints, assumptions, and recommendations for manufacturing.
8. **Handoff to Manufacturing**:
    - The finalized design database is handed off to the semiconductor foundry for manufacturing. The foundry performs additional checks and optimizations to prepare the design for fabrication, including mask data preparation, reticle layout inspection, and tape-out procedures.

Introduction to OpenLANE and strive chipsets

OpenLane is an automated RTL-to-GDSII (Register Transfer Level to Graphic Data System II) flow for designing integrated circuits. It encompasses a range of tools and scripts to automate various steps in the ASIC design process, from synthesis and placement to routing and timing analysis. OpenLane is built upon the OpenROAD framework, which provides an end-to-end design flow for digital ASICs.

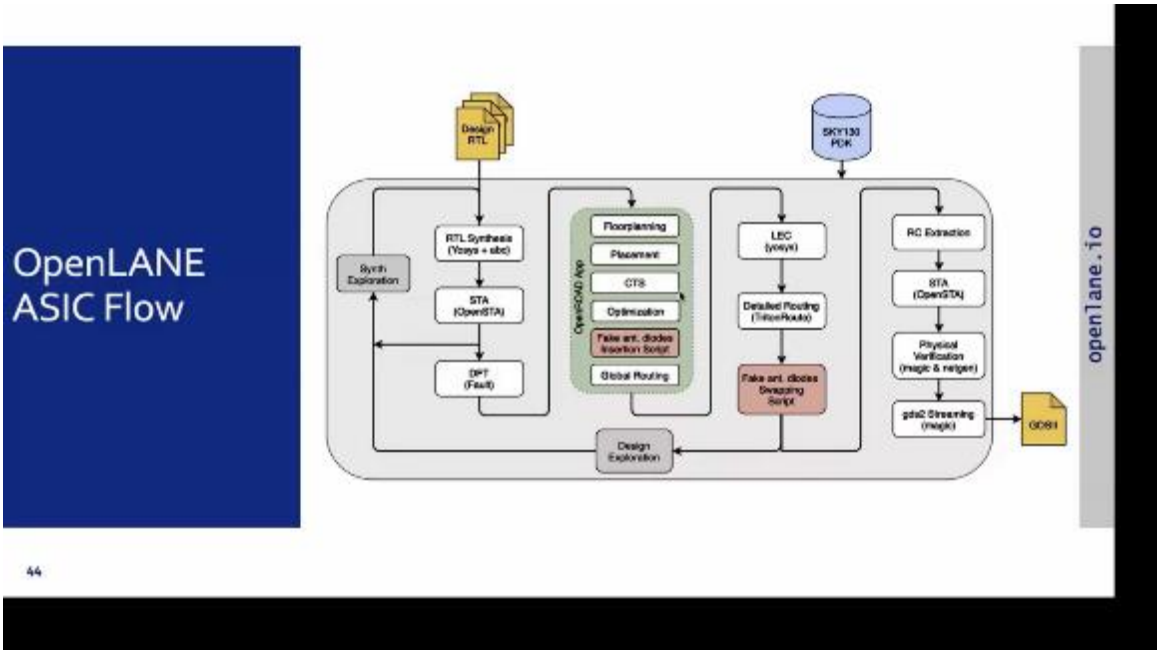
The openlane flow based on several components including OpenROAD, Yosys, Magic, Netgen, CVC, SPEF-Extractor, KLayout and a number of custom scripts for design exploration and optimization.



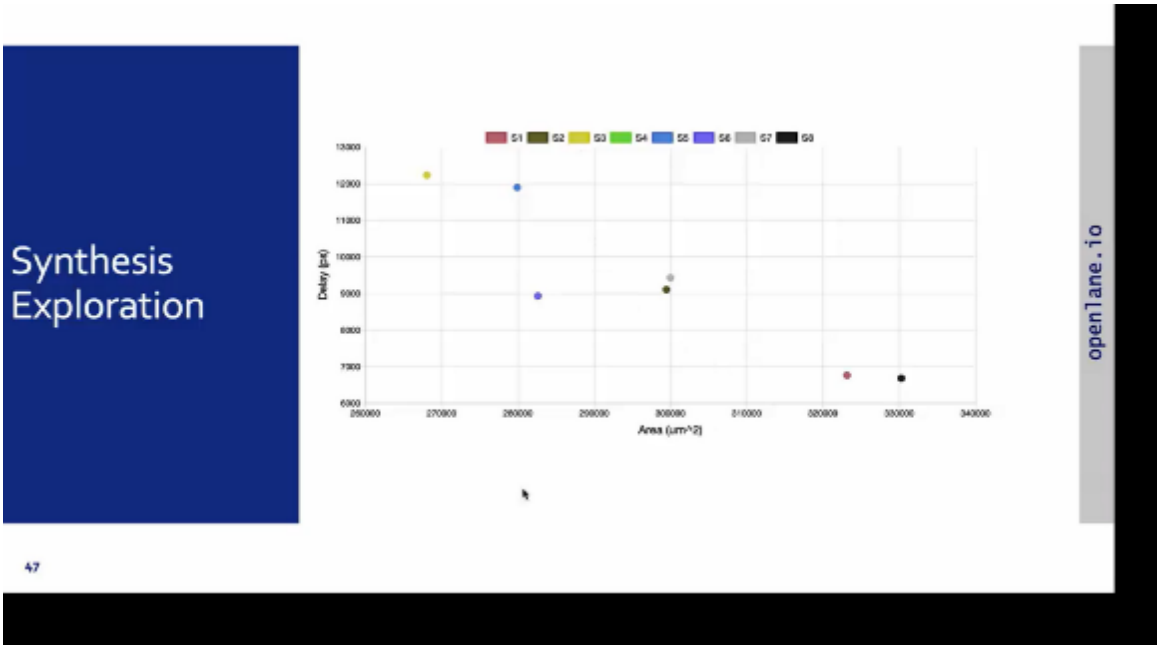


The objective of OpenLANE is to produce a clean GDSII (No LVS violations, No DRC violations, and no Timing violations) with no human intervention.

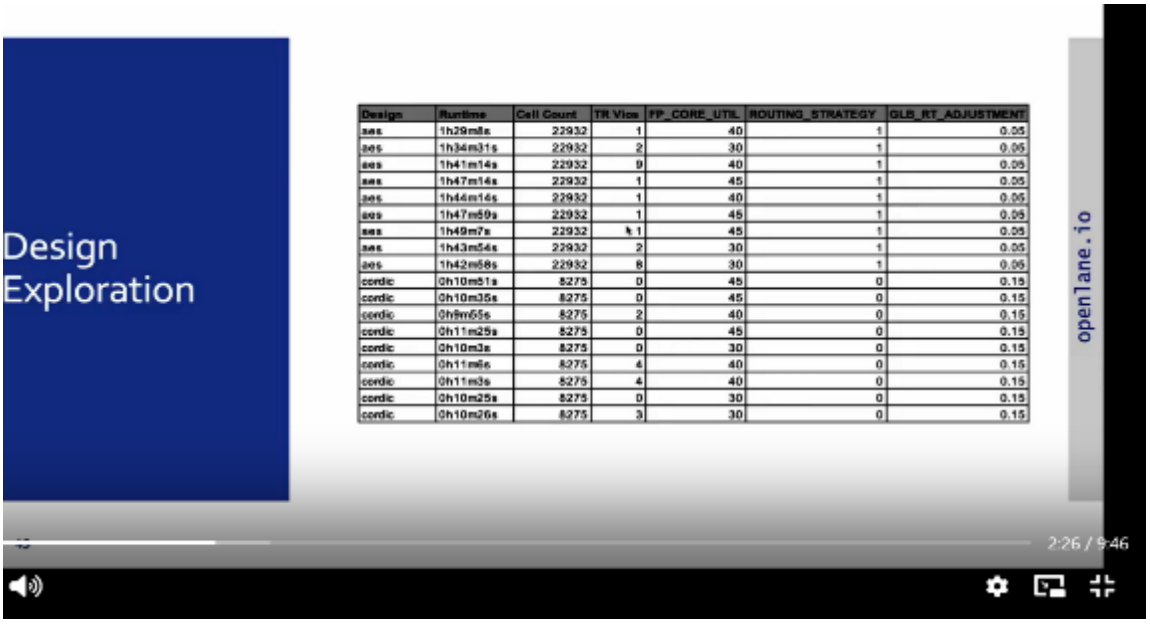
Introduction to OpenLANE detailed ASIC design flow



1. Synthesis Exploration step involves generation of reports showing delay vs area

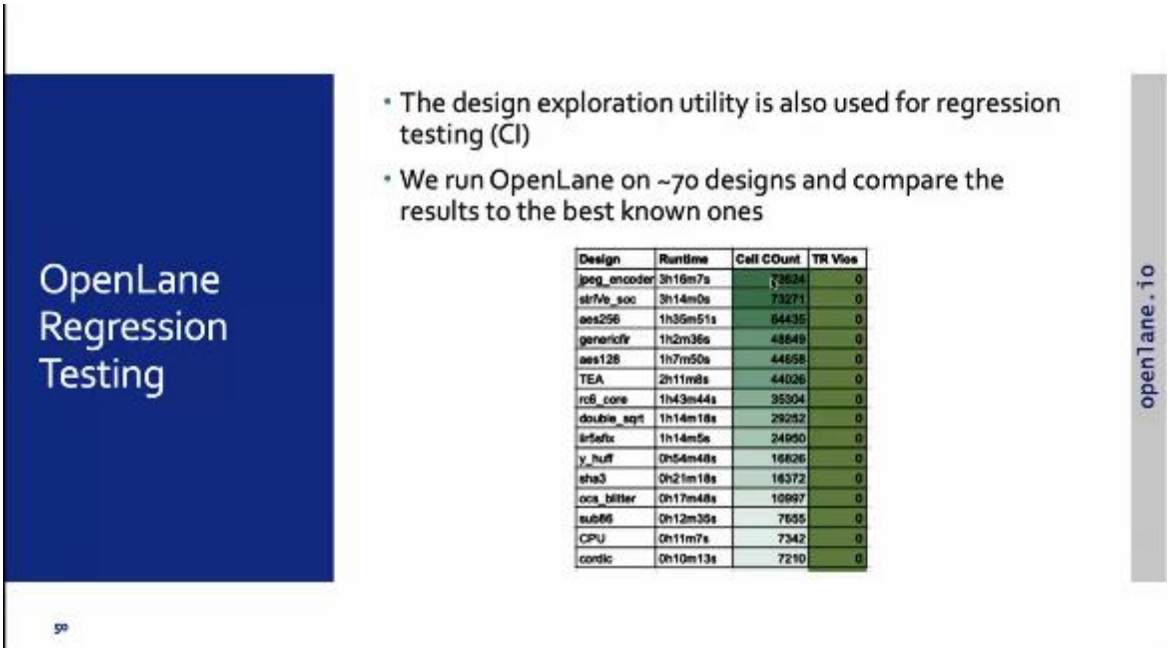


2. Design Exploration step is used to sweep the design configuration and it's useful to find best configuration for any given design.



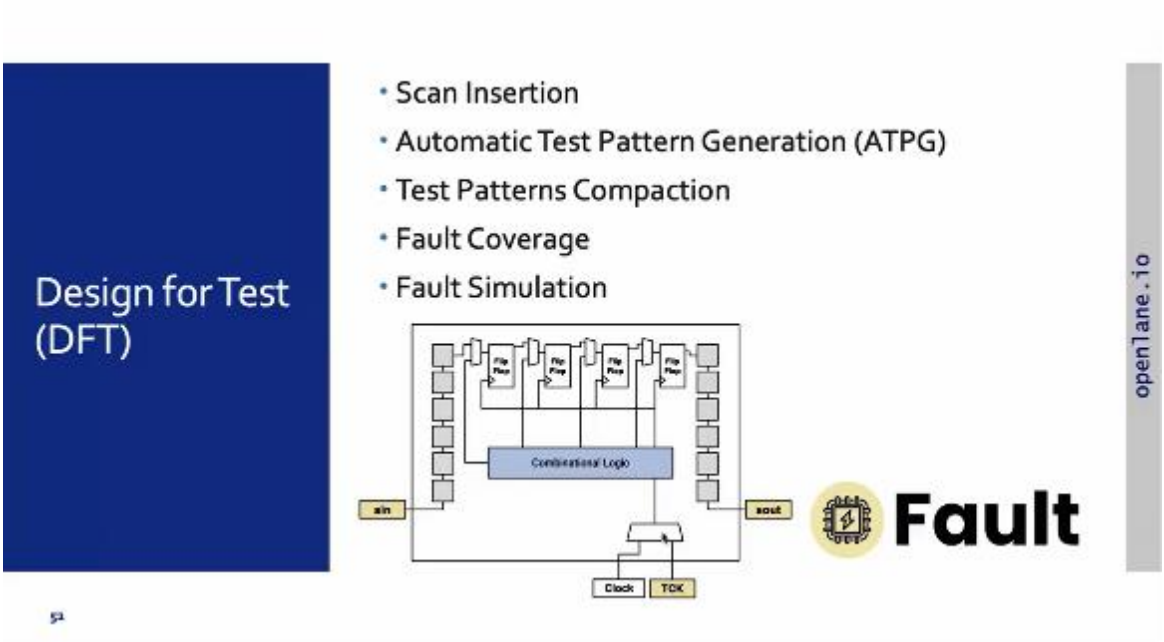
Design	Runtime	Cell Count	TR Vios	FP_CORE_UTIL	ROUTING_STRATEGY	GLB_RT_ADJUSTMENT
aes	1h29m8s	22932	1	40	1	0.05
aes	1h34m31s	22932	2	30	1	0.05
aes	1h41m14s	22932	9	40	1	0.05
aes	1h47m14s	22932	1	45	1	0.05
aes	1h44m14s	22932	1	40	1	0.05
aes	1h47m59s	22932	1	45	1	0.05
aes	1h49m7s	22932	1	45	1	0.05
aes	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

3. OpenLANE Regression Testing



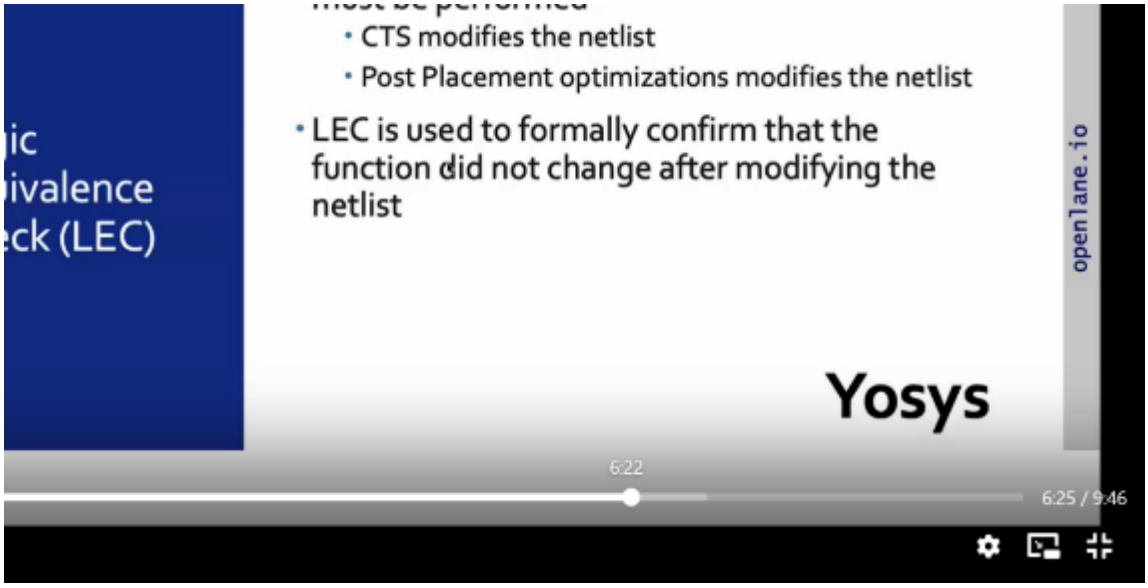
Design	Runtime	Cell COunt	TR Vios
jpeg_encoder	3h16m7s	73624	0
str7ve_soc	3h14m0s	73271	0
aes256	1h35m51s	64435	0
genericfir	1h2m36s	48849	0
aes128	1h7m50s	44658	0
TEA	2h11m8s	44026	0
rc6_core	1h43m44s	35304	0
double_sqrt	1h14m18s	29252	0
iir5efix	1h14m5s	24950	0
y_huff	0h54m48s	16826	0
sha3	0h21m18s	16372	0
oca_blitter	0h17m48s	10997	0
sub66	0h12m36s	7555	0
CPU	0h11m7s	7342	0
cordic	0h10m13s	7210	0

4. Design for Test (DFT)

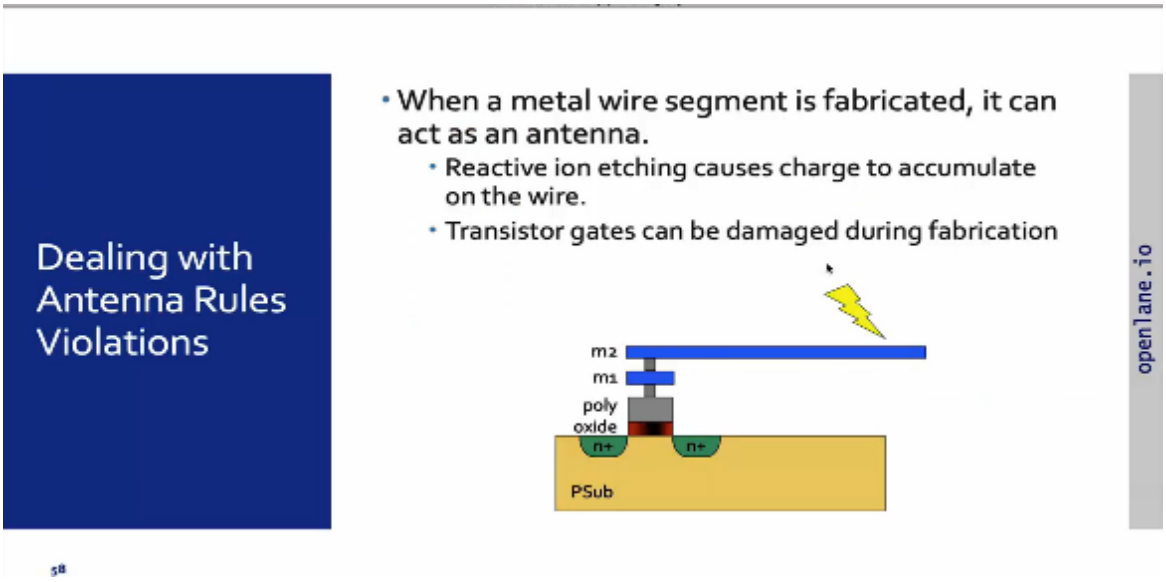


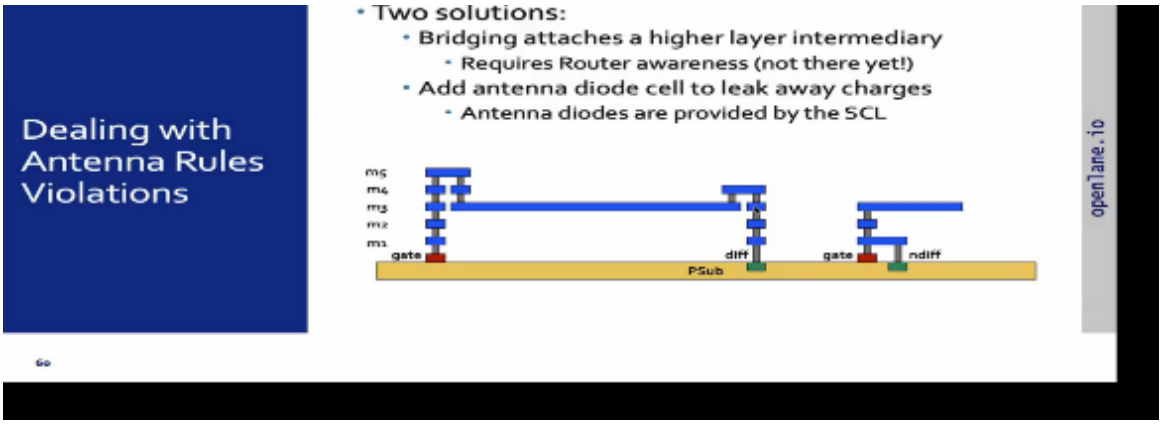
5. Physical verification (DRC & LVS)

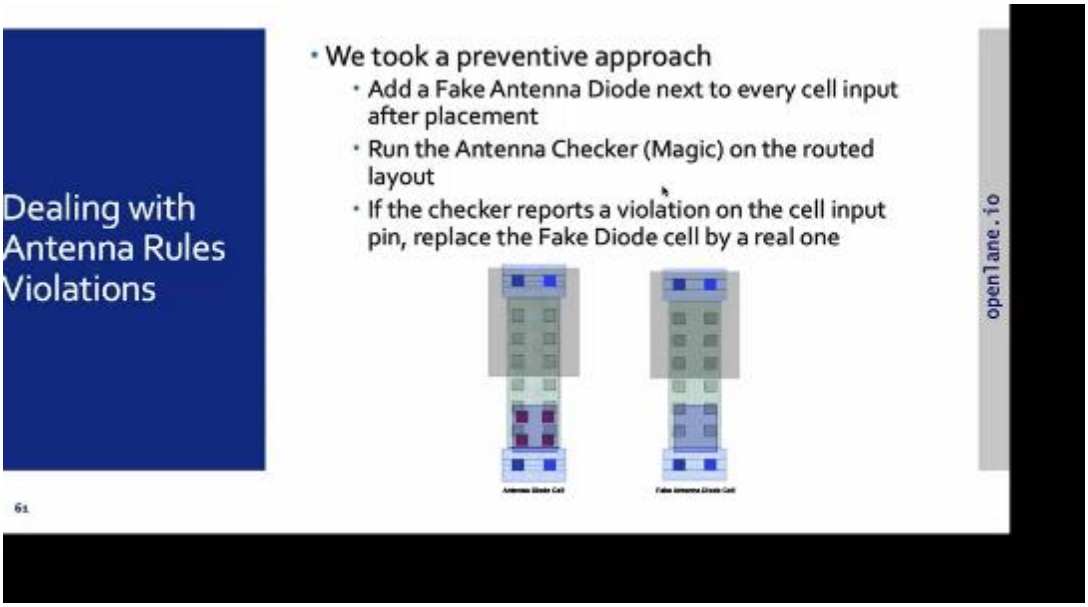
6. Logic Equivalence Check (LEC) checks the logic synchronisation between physical implementation and the netlist.



7. Dealing with Antenna Rules violations





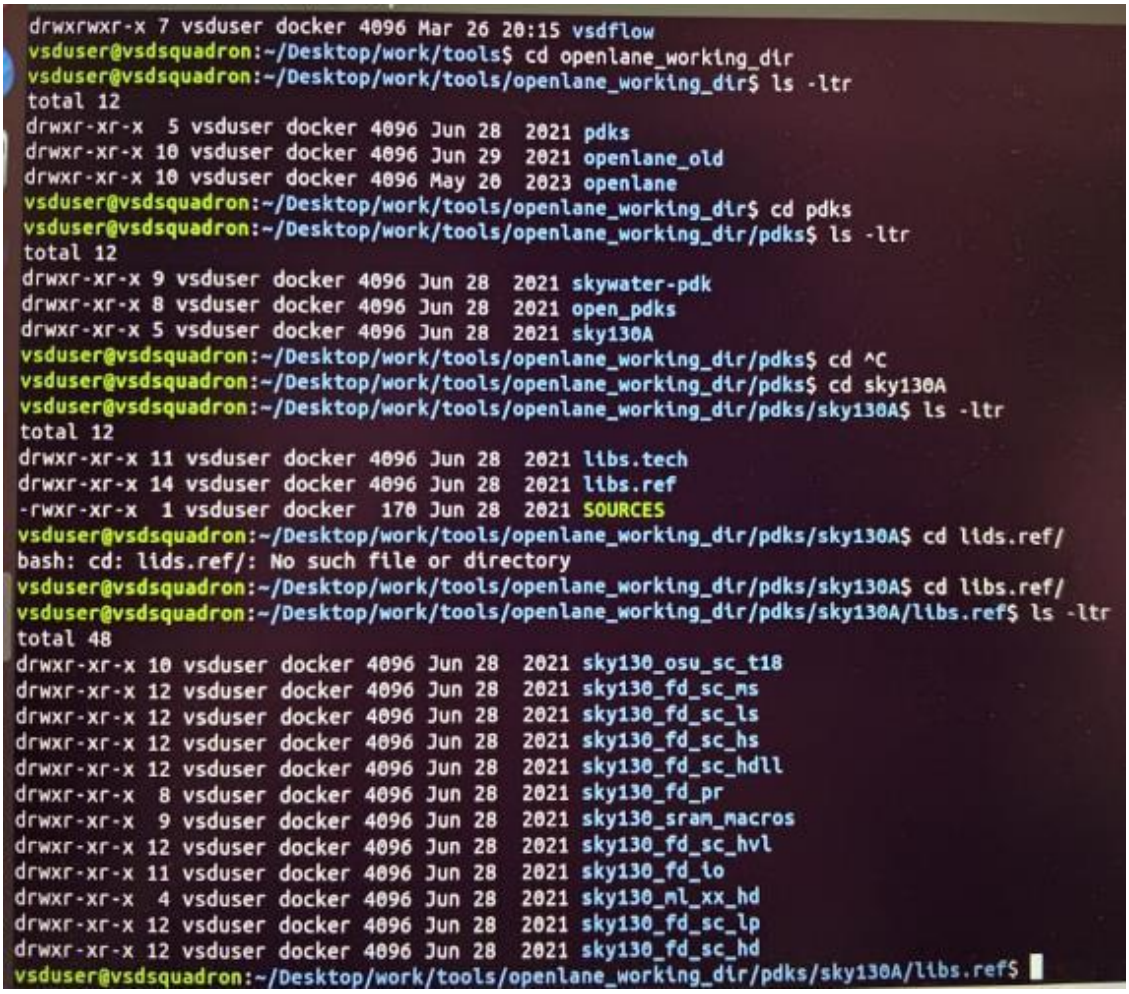


8. Static Timing Analysis (STA) ensures that the design meets timing requirements. STA evaluates the timing behavior of a digital circuit without considering dynamic factors such as signal transitions and clock skew. It determines whether the design meets setup and hold time constraints, maximum clock frequency, and other timing requirements. The input to STA includes the synthesized netlist of the design, and timing constraints.

Getting familiar to open-source EDA tools:

Openlane Directory structure in detail

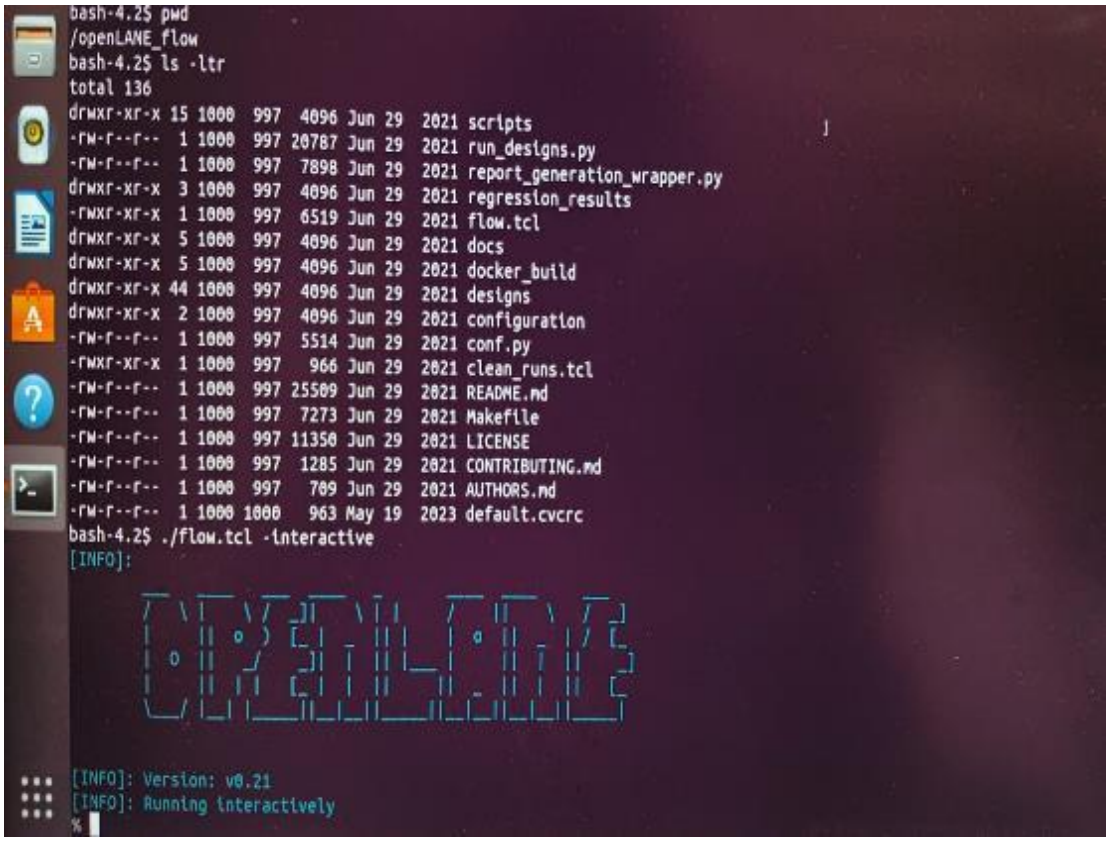
The openlane working has the folders and the subfolders.



- open-pdks contains scripts that makes the commerical PDK to also be compatible with the open-source EDA tool
- sky130A pdk variant is made especially compatible for open-source tools. It contains libs.ref and libs.tech libs.ref contains all the process or technology specific files, example sky130_fd_sc_hd : Sky130nm Foundry Standard Cell High Density libs.tech has files specific for the tool (klayout,netgen,magic...)
- skywater-pdk contains all Skywater 130nm PDKs

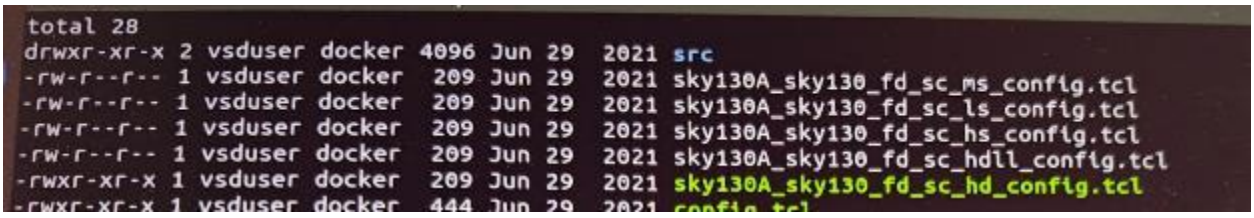
Design Preparation Step

OpenLane can be invoked using `docker` command using the interactive session. `./flow.tcl -interactive` runs the openlane in interactive mode. `% package require openlane 0.9` retrives all dependencies for running v0.9 of OpenLANE



Openlane has multiple designs and we need to work on picorv32a inside a specific design folder there is a config.tcl which has the default settings on OpenLANE.

The priority order for the Openlane settings are: sky130_xxxxx_config.tcl in OpenLane/designs/[design]/ config.tcl in OpenLane/designs/[design]/ default values in OpenLane/configuration/

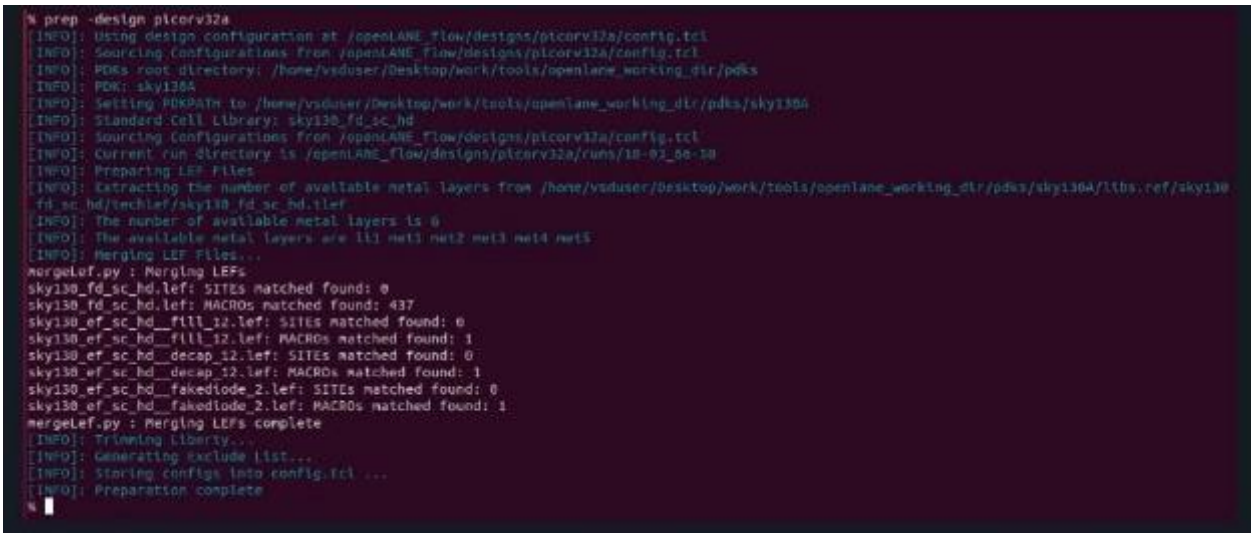


Setup the design stage for the flow and to begin with synthesis of a design

```
prep -design picorv32a
```

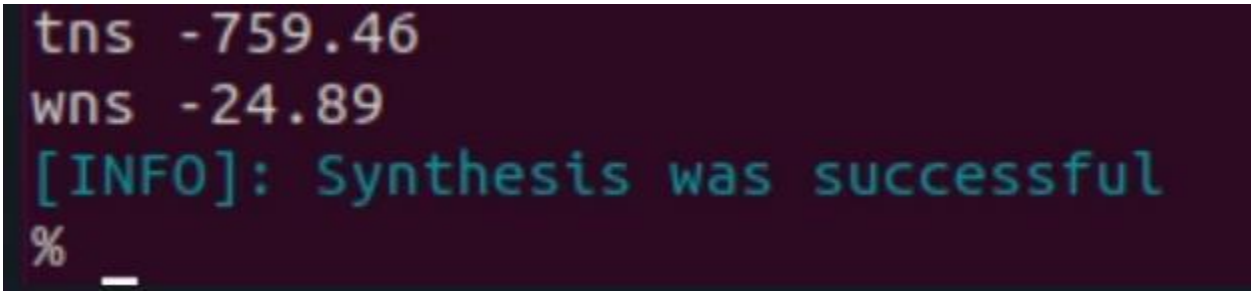
The above command when executed, sets up the filesystem where the OpenLANE tools can dump the outputs. This creates a run/ folder inside the picorv32a directory which contains the command log files, results, and the reports dumped by each tool. The folder will be empty for now except for lef files generated by this design setup stage.

The cell LEF files .lef and technology LEF files .tlef merge to generate merged.lef inside run/tmp/

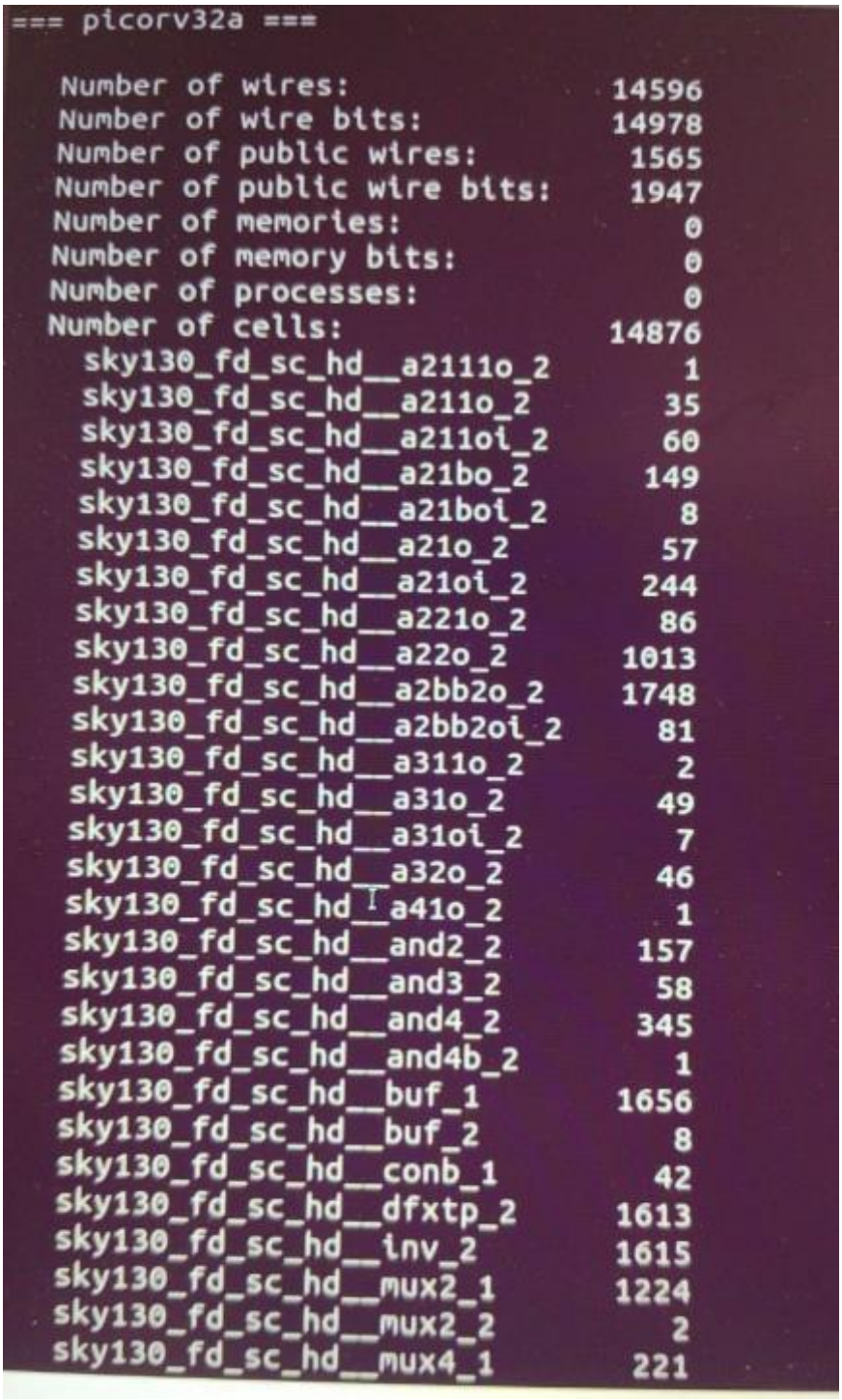


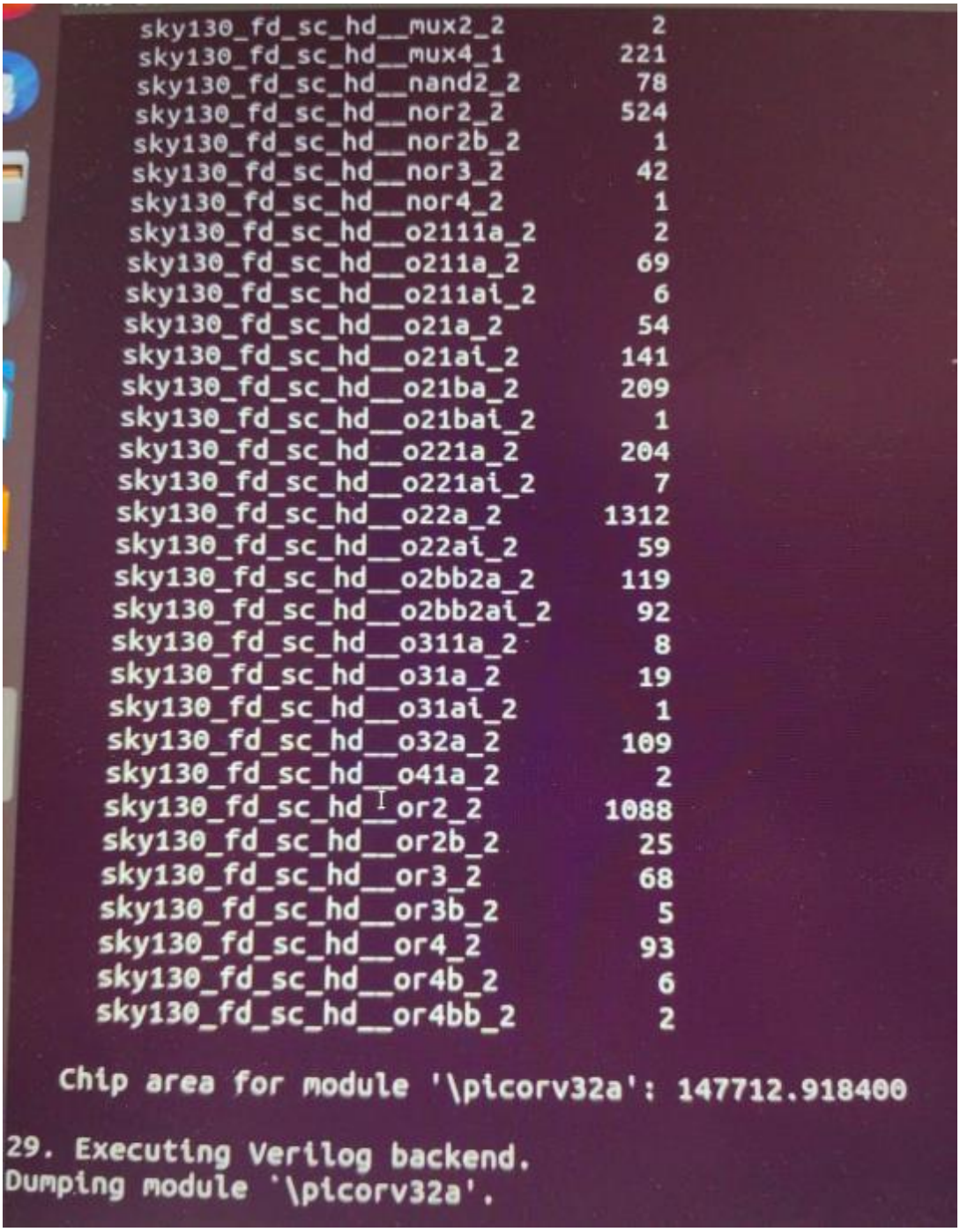
Review files after design prep, run synthesis, and characterize synthesis results

Run synthesis with the command: run_synthesis runs by yosys, RTL synthesis, ABC scripts (for technology mapping) and openSTA.



Results after synthesis is as follows.


=== picorv32a ===
Number of wires: 14596
Number of wire bits: 14978
Number of public wires: 1565
Number of public wire bits: 1947
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

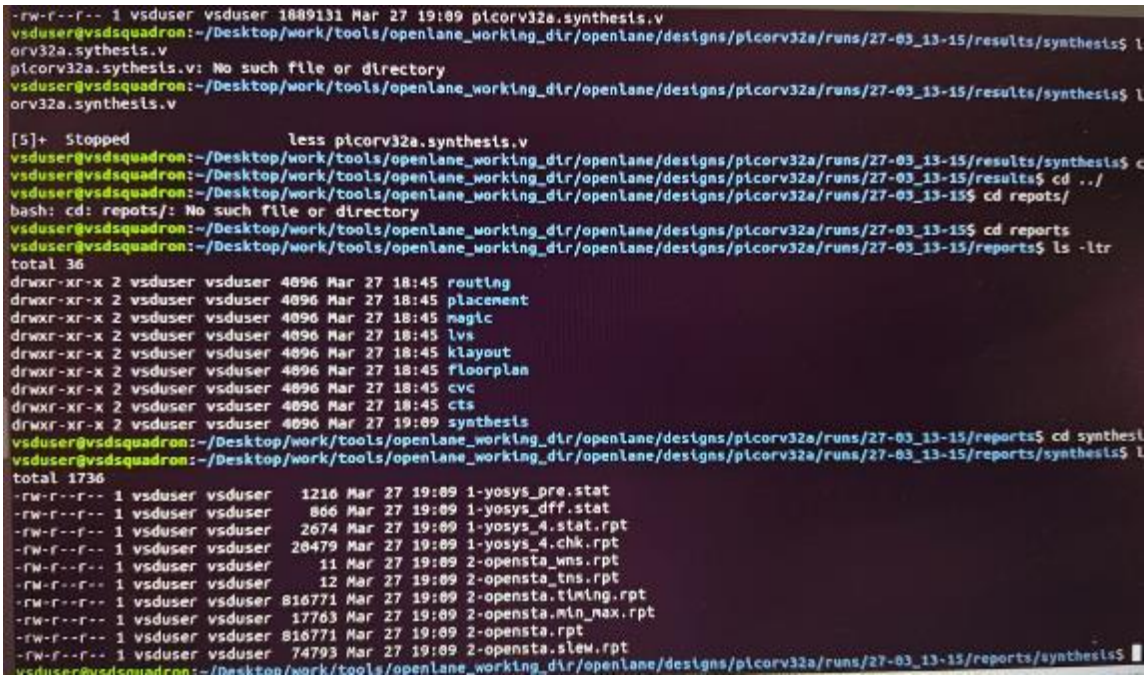


Here the DFF count: sky130_fd_sc_hd__dfxtp_2 = 1613

And total number of cells = 14876

D flip-flop ratio = count of DFFs / total number of cells = 1613 / 14876 = 0.108429685 (OR) 10.8429 %

After running synthesis, inside the runs/[date]/results/synthesis is picorv32a_synthesis.v which is the mapping of the netlist to standard cell library using ABC. The runs/[date]/reports/synthesis will contain synthesis statistic reports and STA reports. The log files are:



Good floorplan vs bad floorplan and introduction to library cells

Chip Floor planning considerations

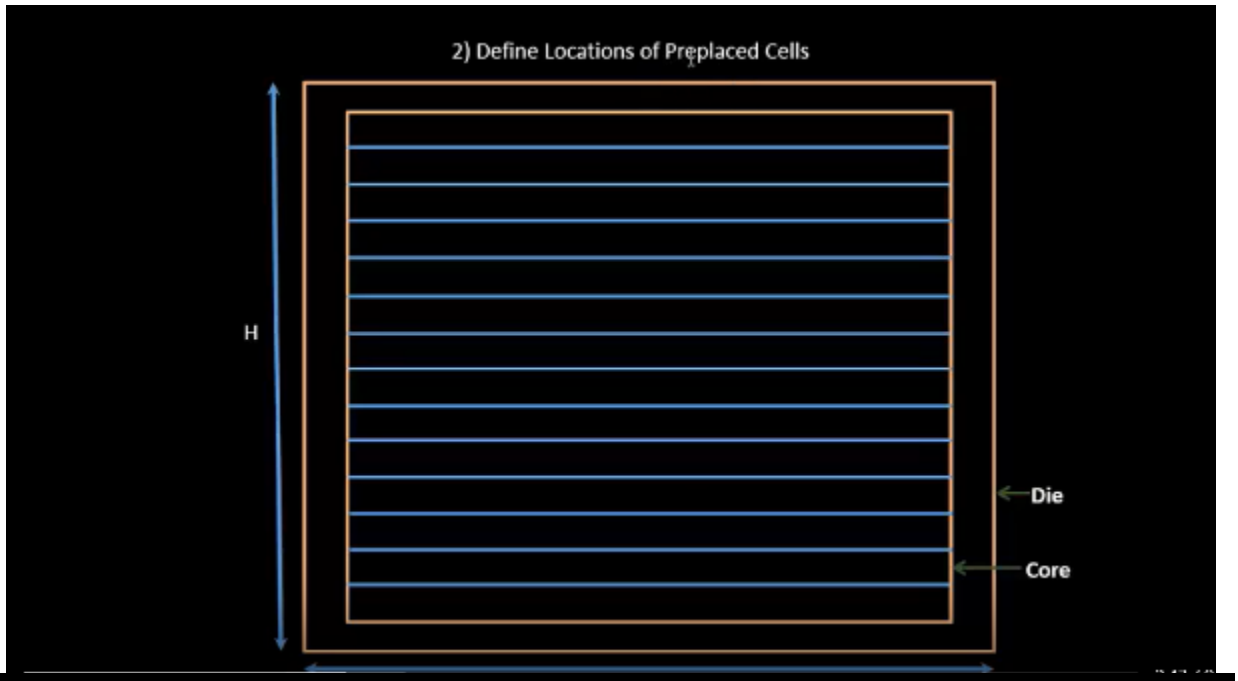
Utilization factor and aspect ratio

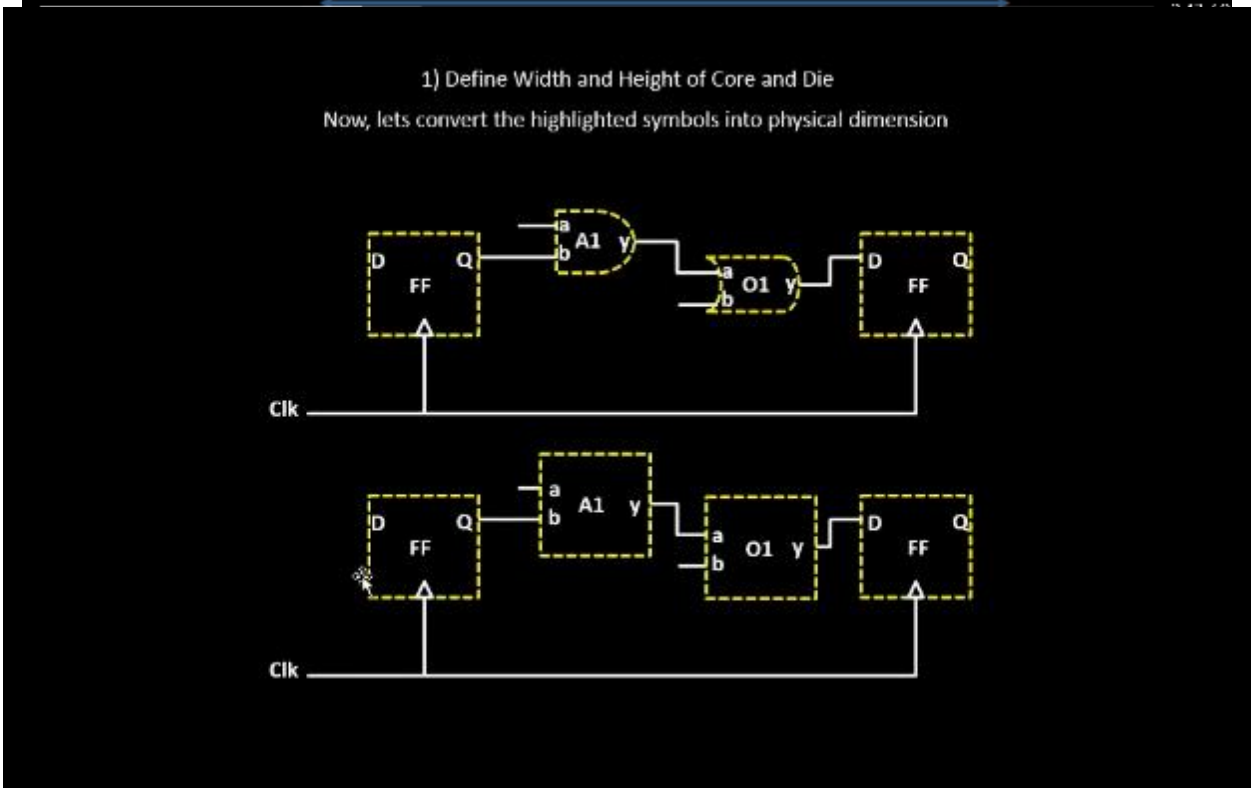
1. Define width and height of core and die

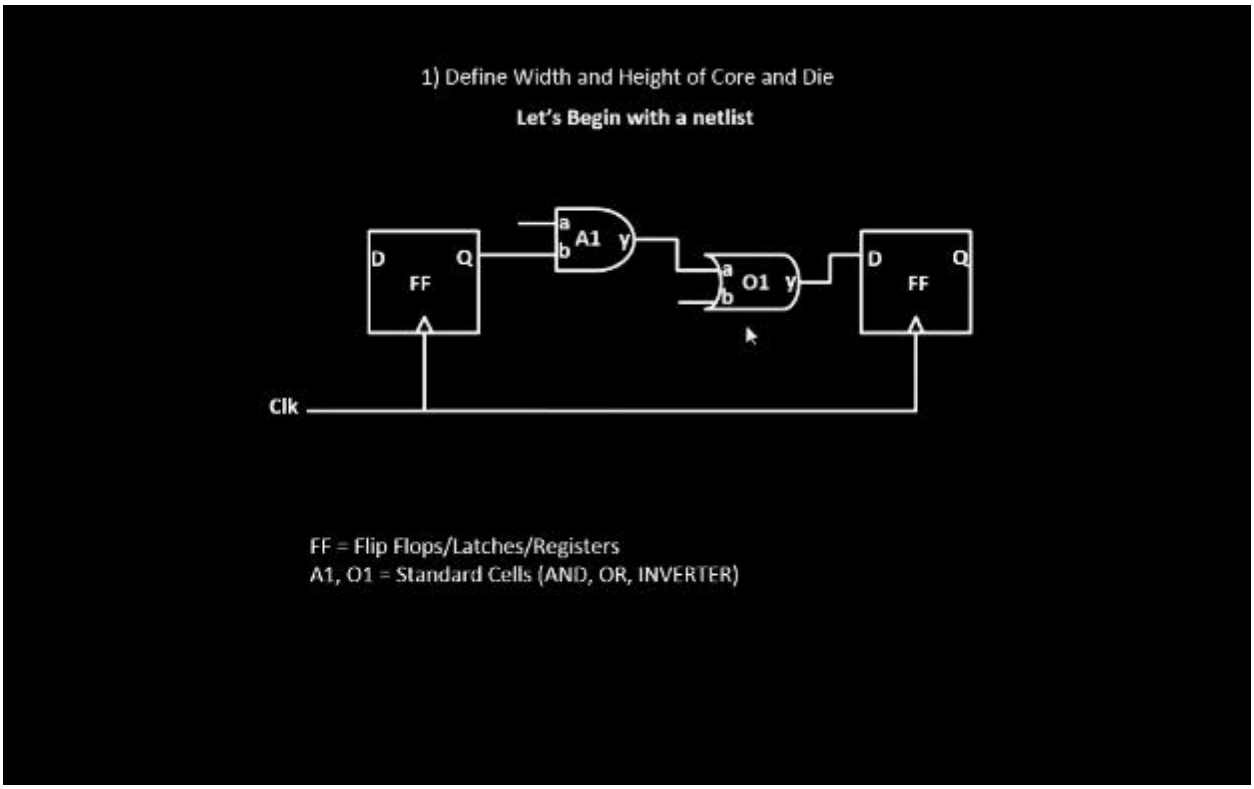
> **Core**: The core refers to the central functional area of the integrated circuit where the actual circuitry and logic reside. The width and height of the core typically refer to the dimensions of this functional area, which contains the active components of the chip, such as logic gates, memory cells, and other functional blocks. The width and height of the core are essential parameters that influence the overall size and aspect ratio of the chip.

2. **Die**: The die represents the individual unit of an integrated circuit that is fabricated on a semiconductor wafer during the manufacturing process. The die contains the entire functional circuitry of the chip, including the core, as well as additional features such as bonding pads, I/O interfaces, and metal layers for interconnections. The width and height of the die correspond to the dimensions of the rectangular or square-shaped semiconductor substrate on which the circuitry is fabricated.

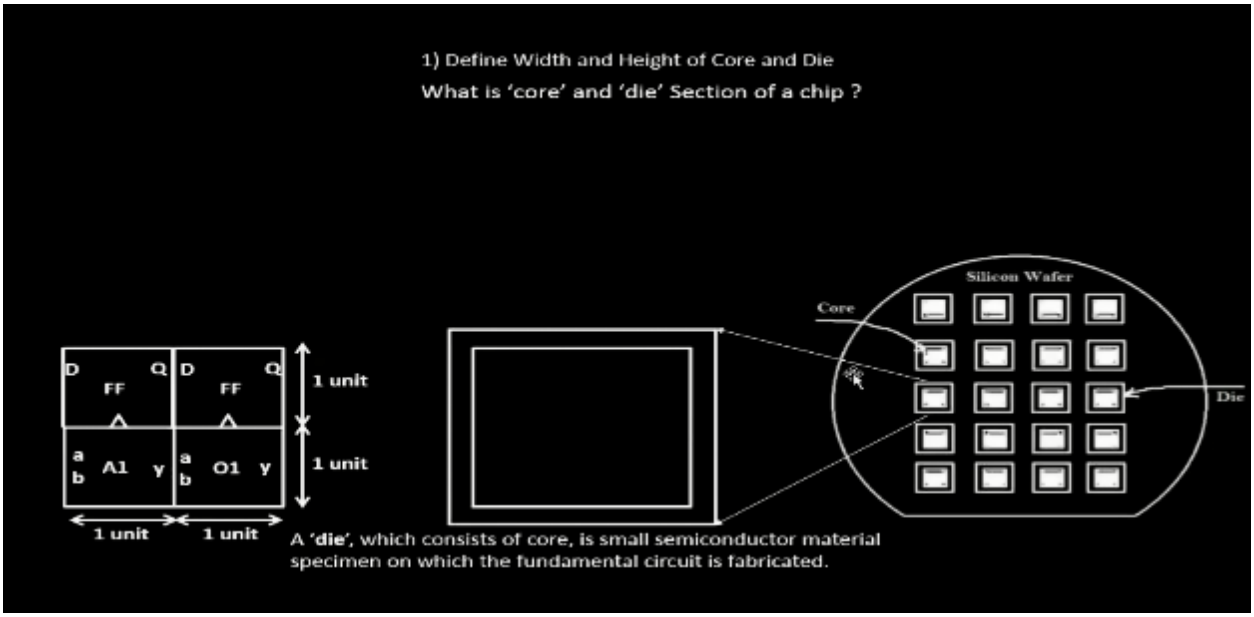
In summary, the width and height of the core and die of an integrated circuit refer to the dimensions of the functional circuit area and the entire semiconductor substrate, respectively. These dimensions are critical parameters in IC design and manufacturing, ifluencing factors such as chip size, layout optimization, and packaging considerations.


2) Define Locations of Preplaced Cells
H
Die
Core


1) Define Width and Height of Core and Die
Now, lets convert the highlighted symbols into physical dimension
D
Q
FF
a
b
A1
y
a
b
O1
y
D
Q
FF
Clk
D
Q
FF
a
b
A1
y
a
b
O1
y
D
Q
FF
Clk

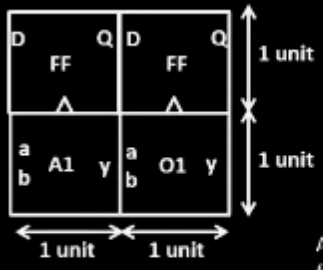


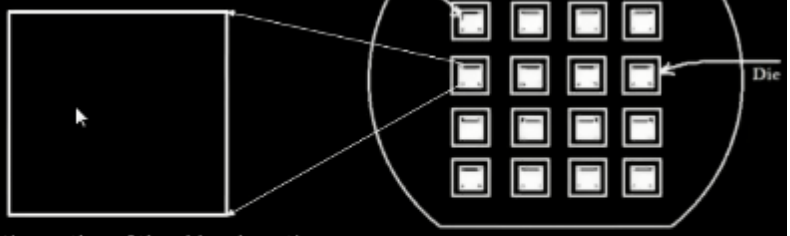
Let's calculate the area occupied by the above netlist on a silicon wafer. Before that the wires are ignored and the flops and combinational logic are combined together to get the total area. So, here, the area will be 4 sq.units.



1) Define Width and Height of Core and Die

What is 'core' and 'die' Section of a chip ?





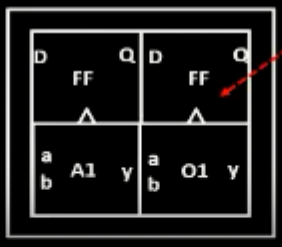
A '**core**' is the section of the chip where the fundamental logic of the design is placed.

Place all logical cells inside the 'core'

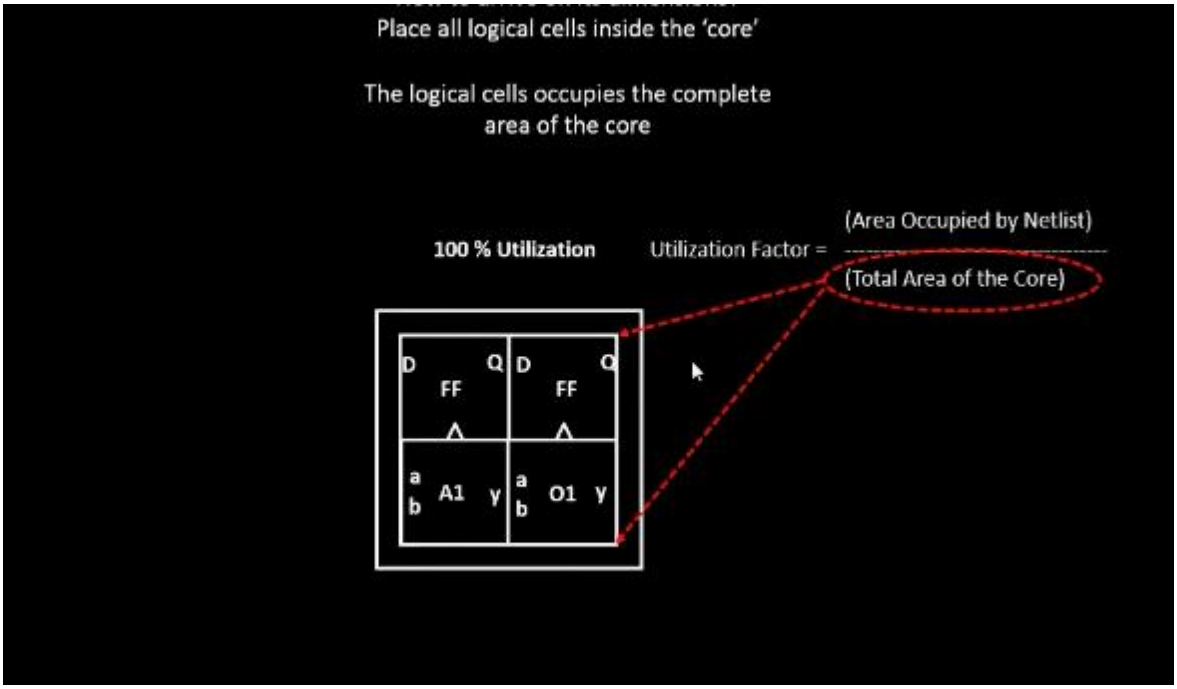
The logical cells occupies the complete area of the core

100 % Utilization

Utilization Factor = (Area Occupied by Netlist) / (Total Area of the Core)



5:12 / 9:09



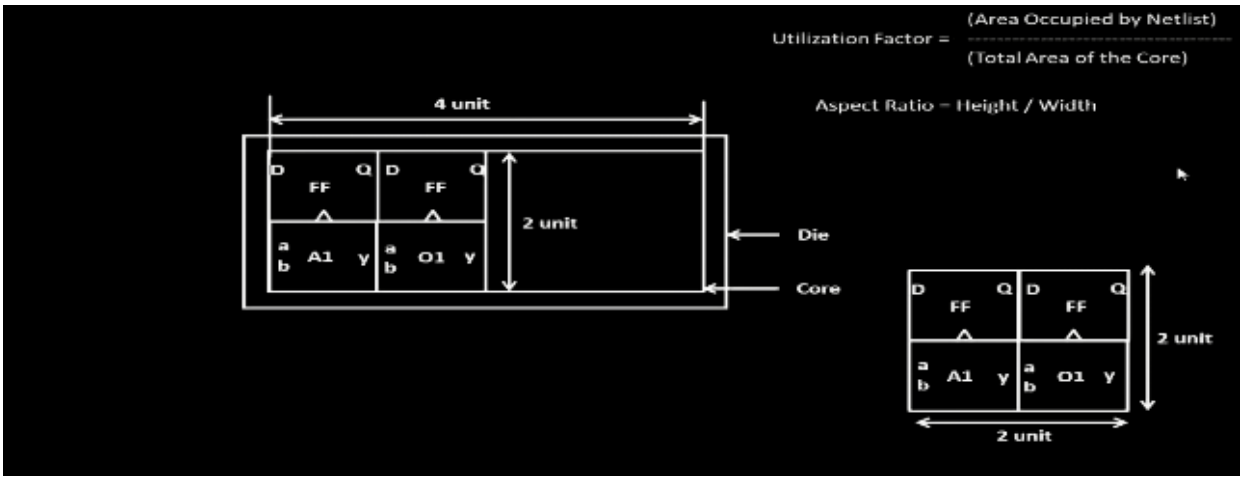
Utilization factor = (4 x 1sq.unit) / (2 unit x 2 unit) = 1

which means the core is completely occupied. In practical scenario, utilization is about 50%

Aspect ratio = Height / Width

In this case, height and width are same, so the aspect ratio is 1 If the aspect ratio is 1 it shows that the chip is square, otherwise it is rectangle.

Example:



1. Utilization factor = (4 sq.units) / (8 sq.units) = 0.5

   Aspect ratio = 2 / 4 = 0.5

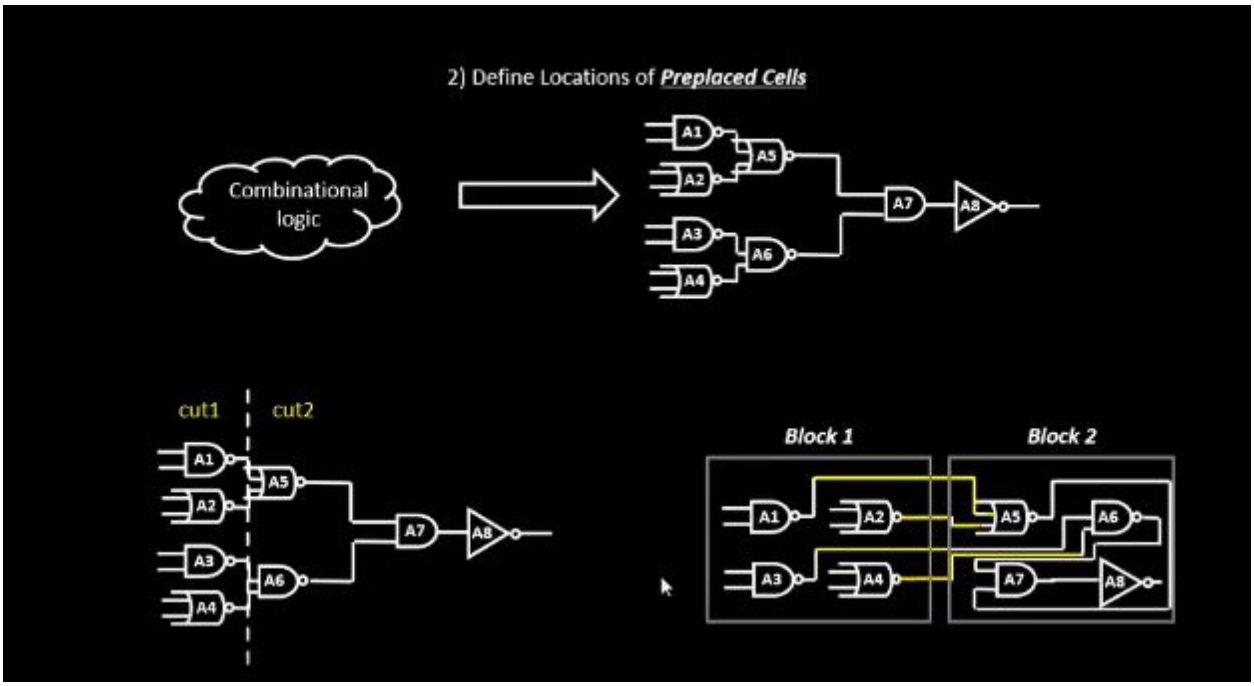
Concept of pre-placed cells

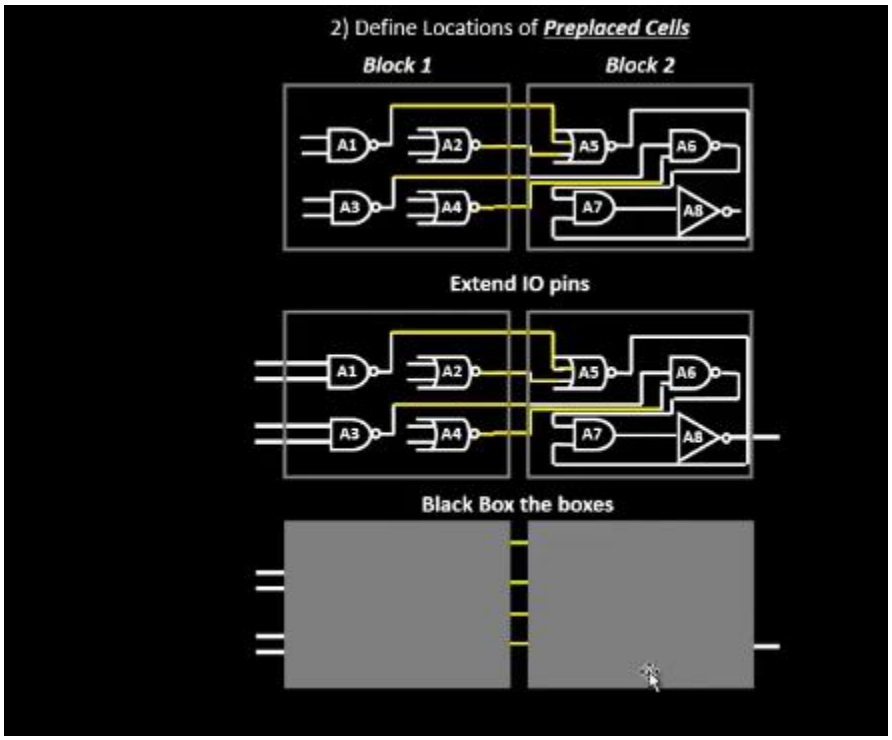
2. Define Locations of Preplaced Cells

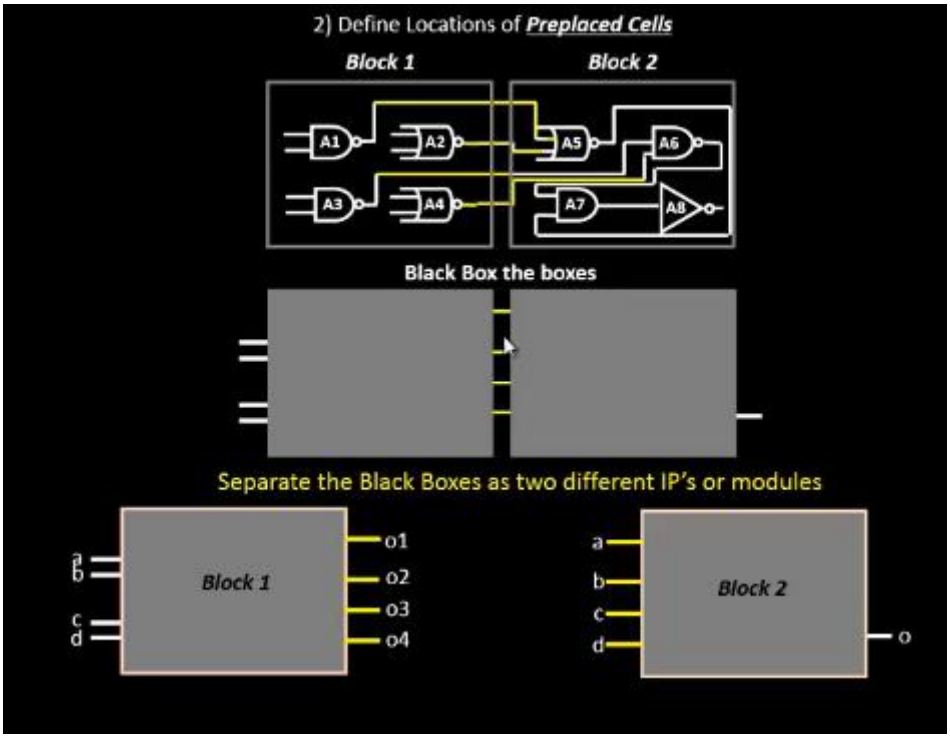
Preplaced cells (also known as pre-placed instances or pre-placed blocks) are specific circuit elements within an integrated circuit design layout that are manually positioned in advance by the designer. The locations of preplaced cells are predetermined and fixed before the automated placement and routing stages of the design flow.

the locations of preplaced cells are manually determined by the designer to optimize layout, performance, and connectivity within an integrated circuit design. By strategically positioning these cells and aligning them with key design elements, designers can improve overall chip performance and manufacturability.

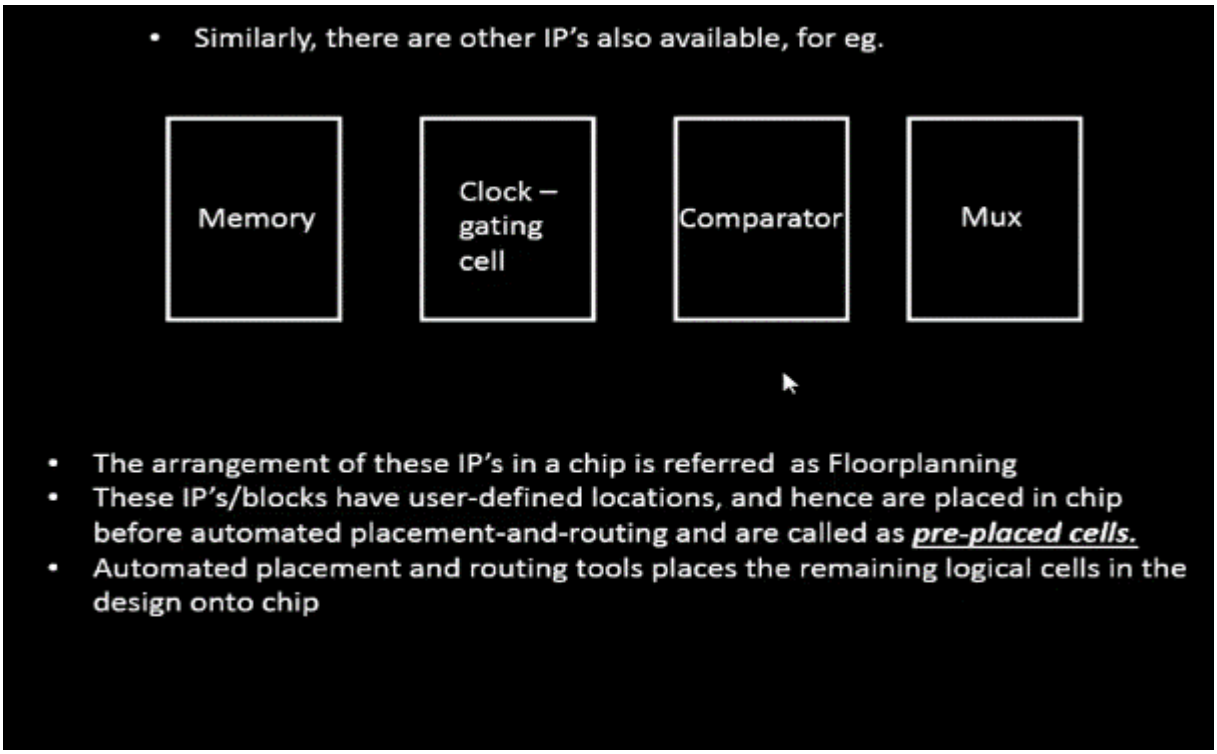
Consider the following netlist being divided into two sets of blocks with connectivity preserved.





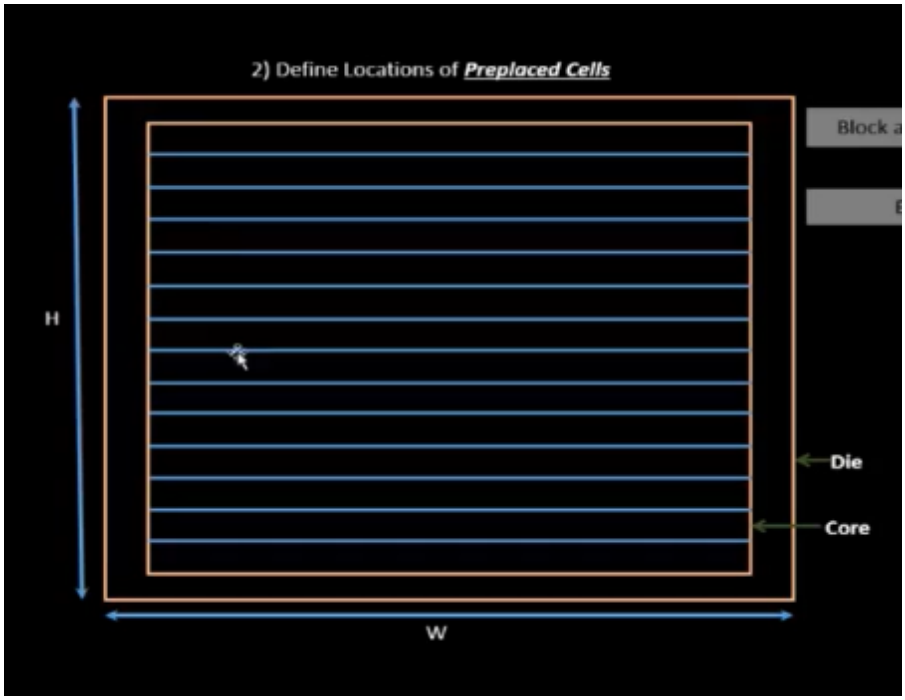


So, if Block1 has to be used by a designer, it can be directly handed over since it is blackboxed. This block can be used across the designs. It means the block can be reused.



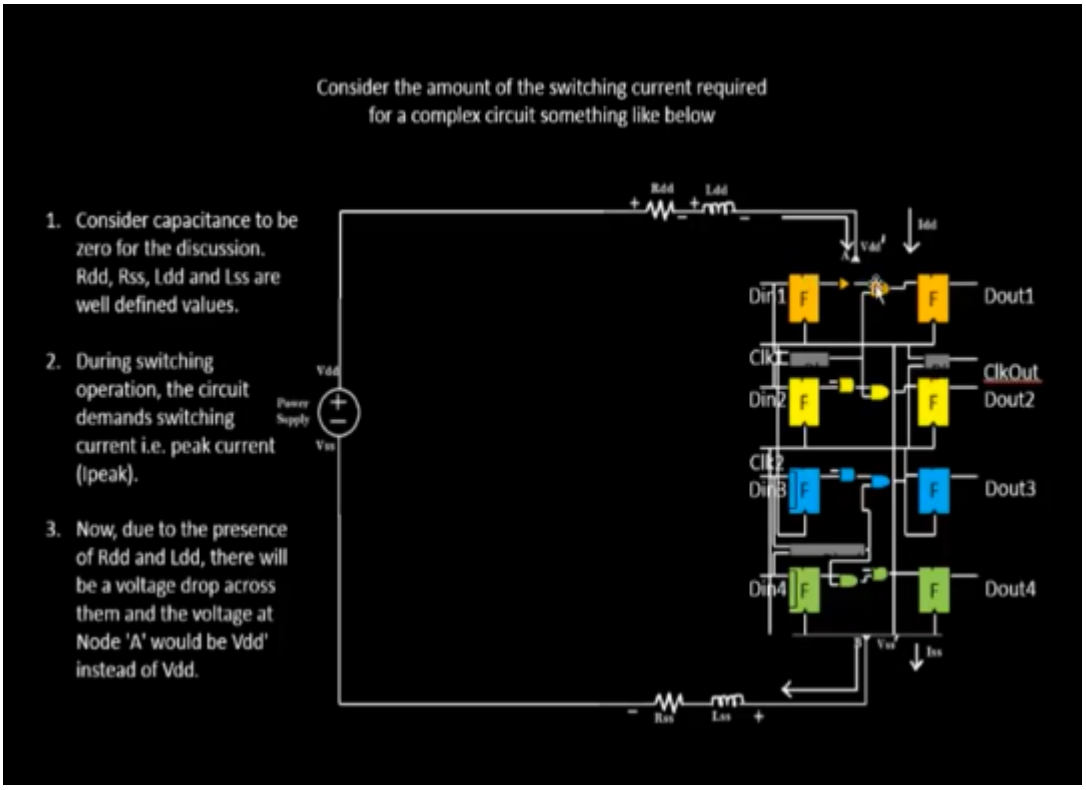
De-coupling capacitors

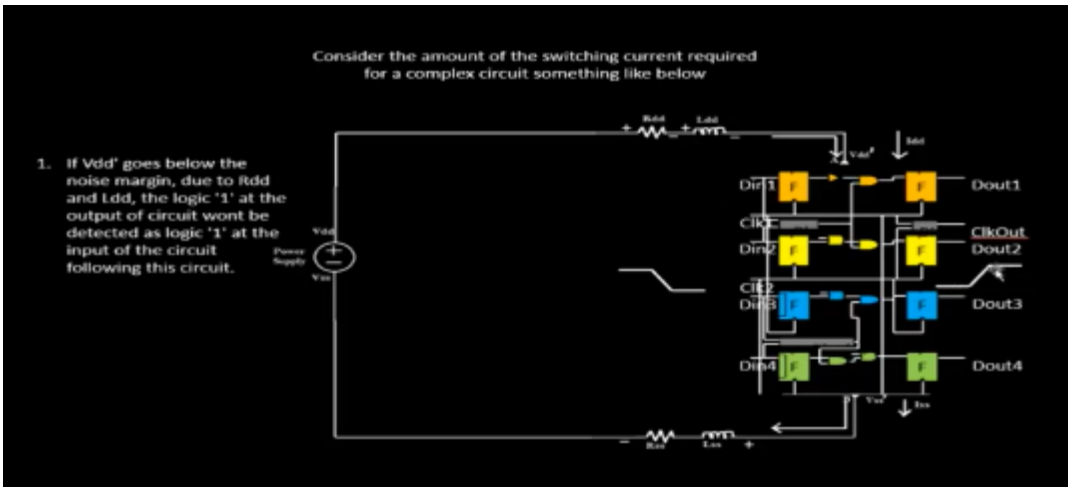
3. Surround pre-placed cells with decoupling capacitors



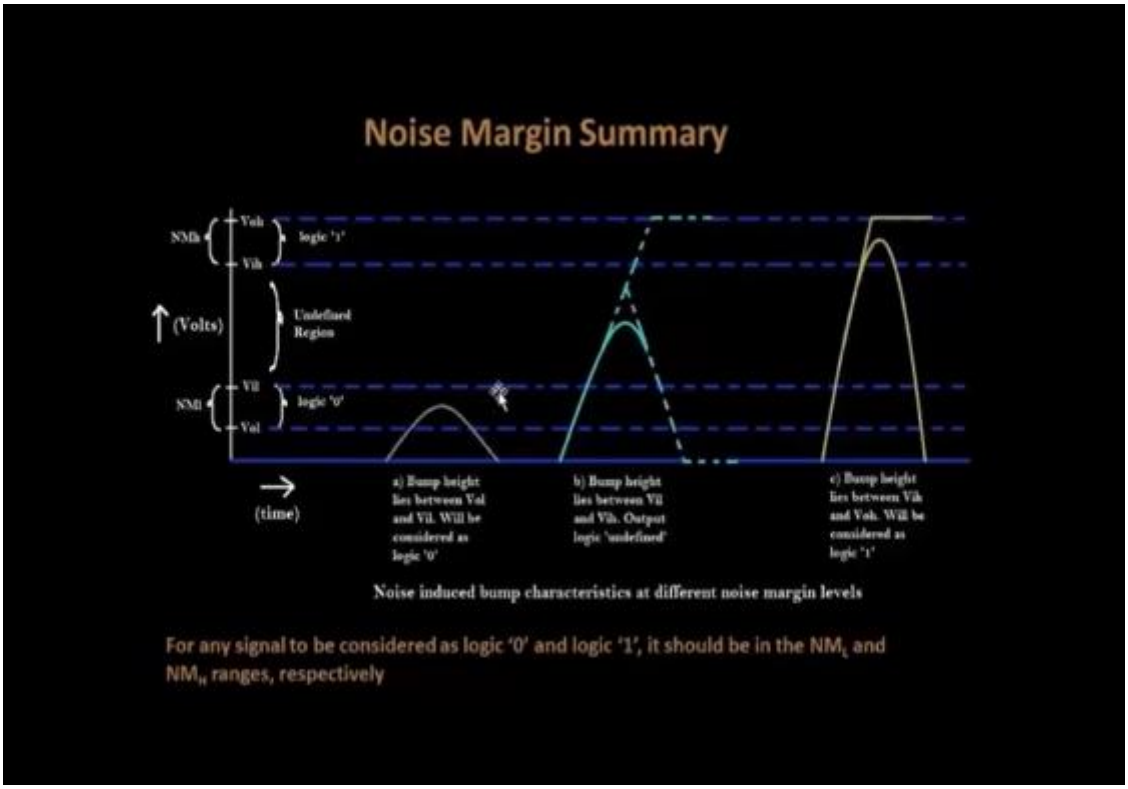
Consider the circuit below as a part of a block. Whenever the circuit switches, there is an amount of current demand. For example, if the AND gate switches from logic0 to logic1, the capacitance has to completely charge. The amount of charge will be sent from the supply voltage. And when the logic switches from logic1 to logic0, the capacitance discharges and it's the responsibility of the Vss to take that discharged current.

In reality, when the Vdd supplies voltage to the circuit, there is a drop due to resistance, inductance and capacitance of the wire and supplied volatge is Vdd'

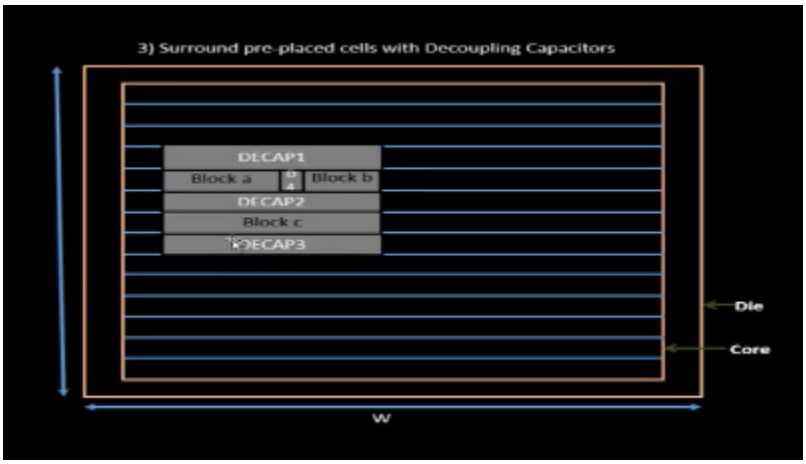


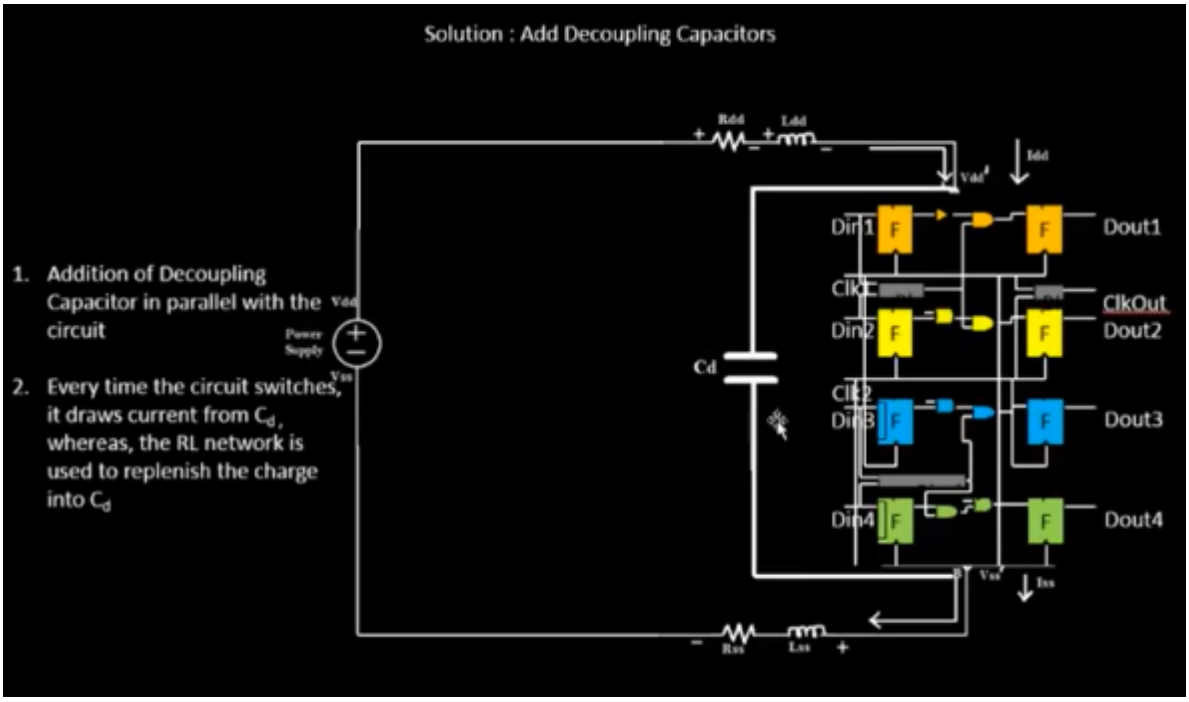


The Vdd' should be within the noise margin range which is from Vih to Voh. If it is present somewhere in the undefined region, then the logic 1 is unstable. This is because of the large physical distance from the main power supply to the circuit.



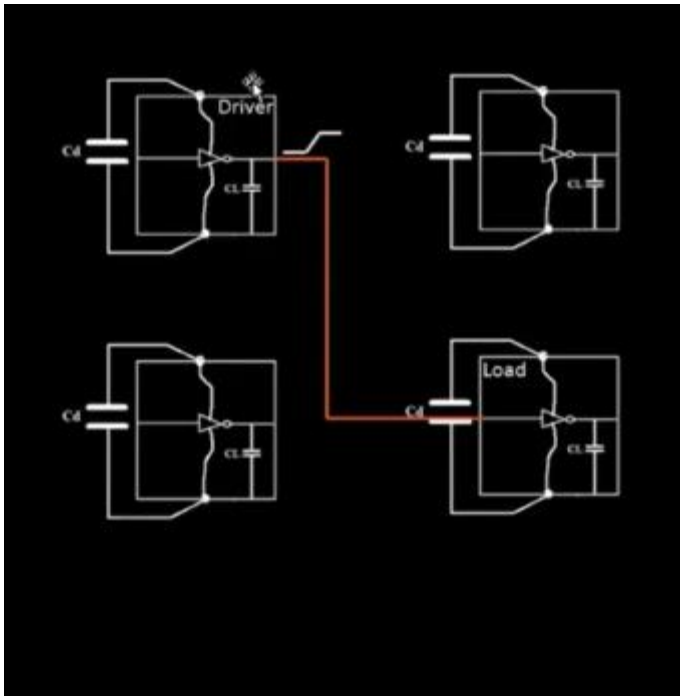
Solution to such problem is the addition of decoupling capacitors. We can consider decoupling capacitor as a huge capacitor completely filled with charge. the equivalent voltage across the capacitor is same as seen across the main supply voltage. The capacitor decouples the circuit from the main supply.



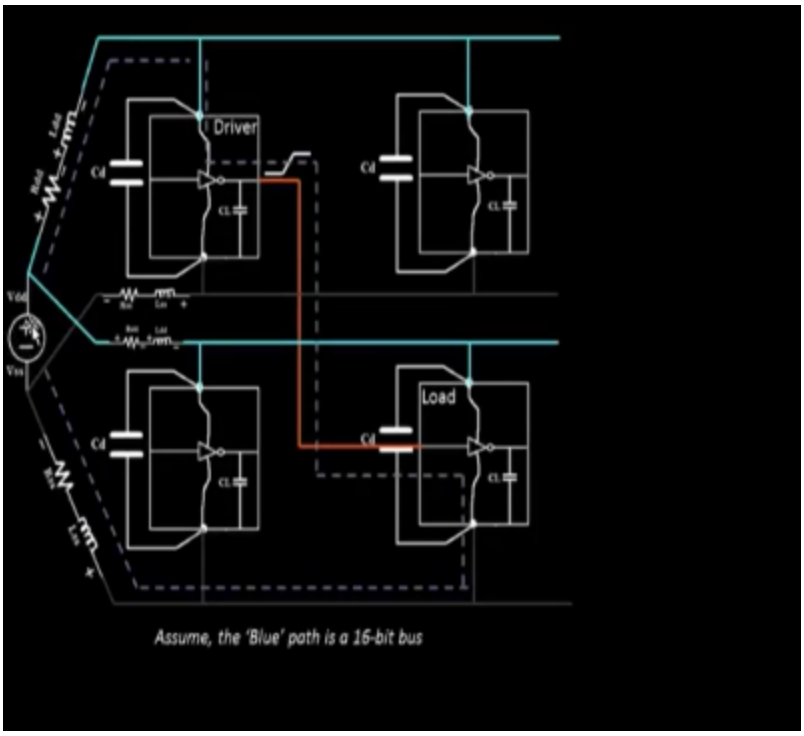


Power Planning

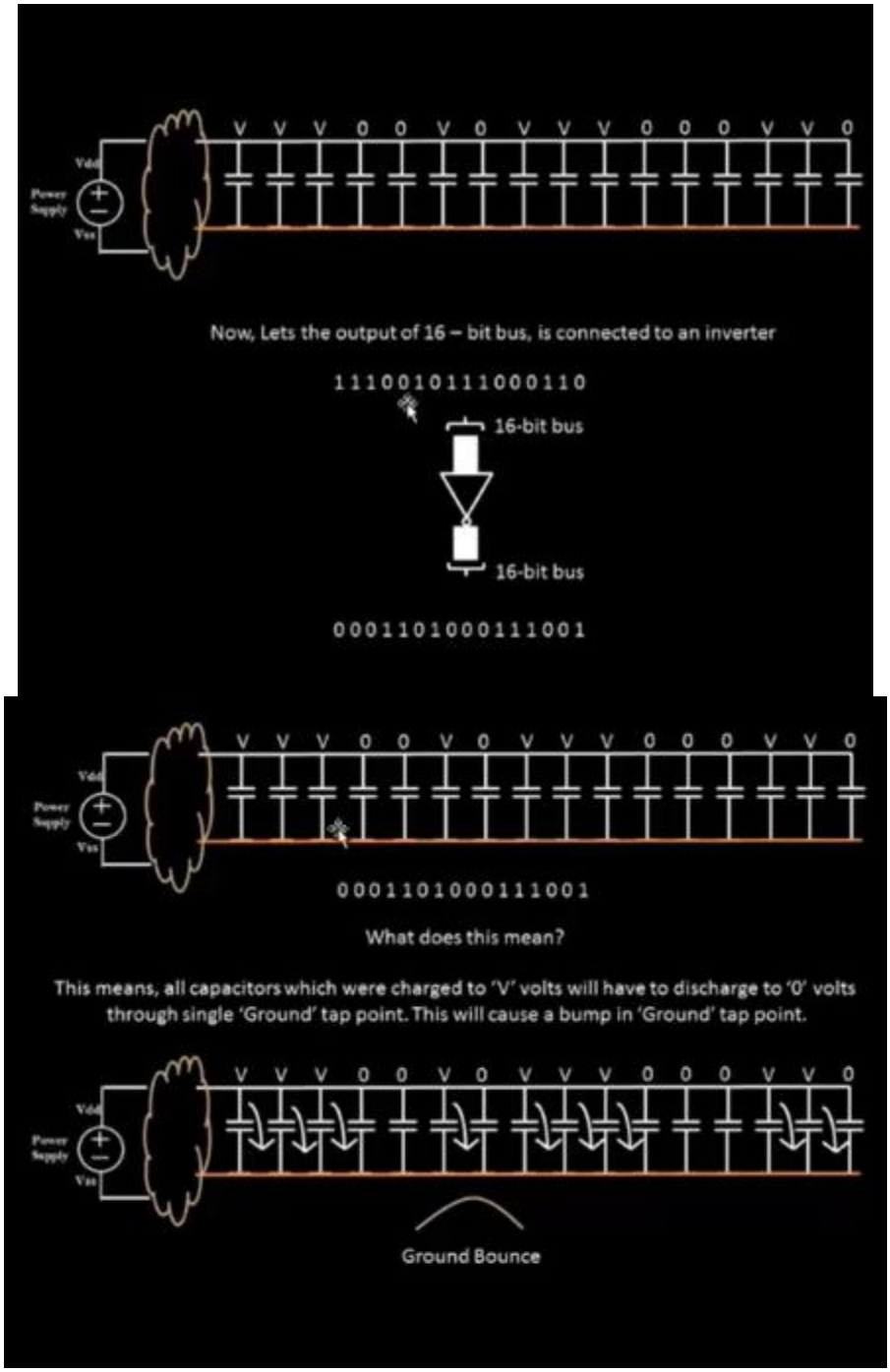
Consider the circuit as a macro and it demands more current. Say, it's used as a driver as well as load, and the load should receive the same signal quality as the driver.



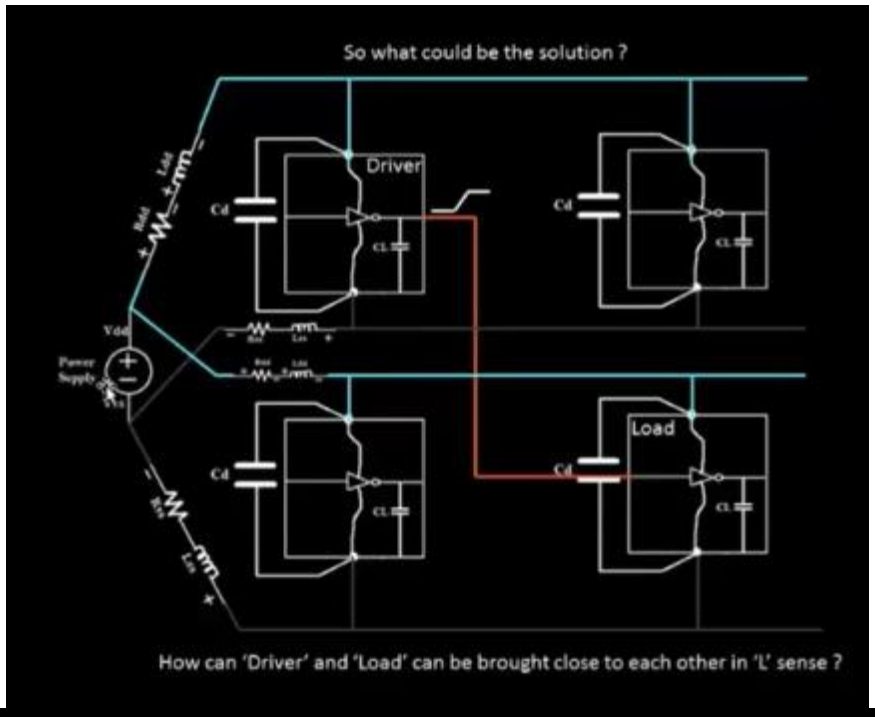
Decoupling capactor is not feasible to be added all over the chip but only on the critical elements.

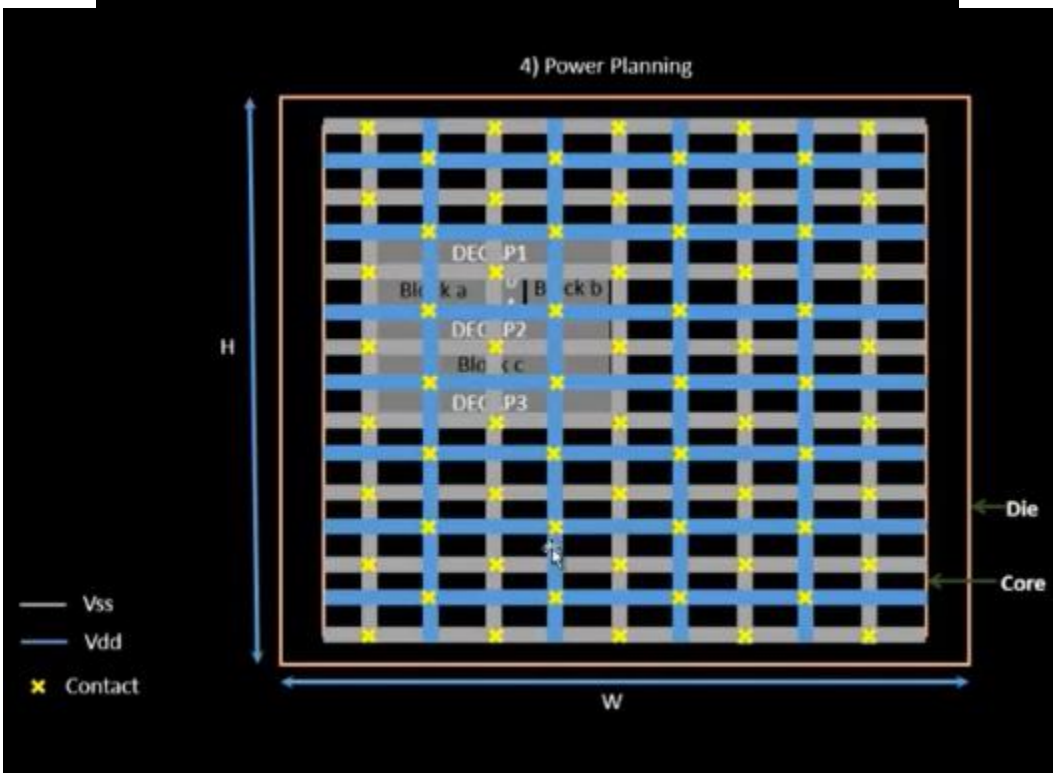


In the below picture, 1 means the capacitor is charged to V and 0 means the capacitor is discharged. If the wire bus is connected to an inverter, then it means that all the capacitors charged to V will discharge at the same time. Large number of elements switching to logic0 might cause Ground Bounce due to huge amount of current that needs to be sinked at the same time, and switcing to logic 1 might cause Voltage Droop due to insufficient current from the power source to all elements. Ground bounce and Voltage Droop might cause the voltage to not be within the noise margin range. The solution is to have multiple powersource taps (power mesh) where elements can source current from the nearest Vdd and sink current to the nearest Vss tap.



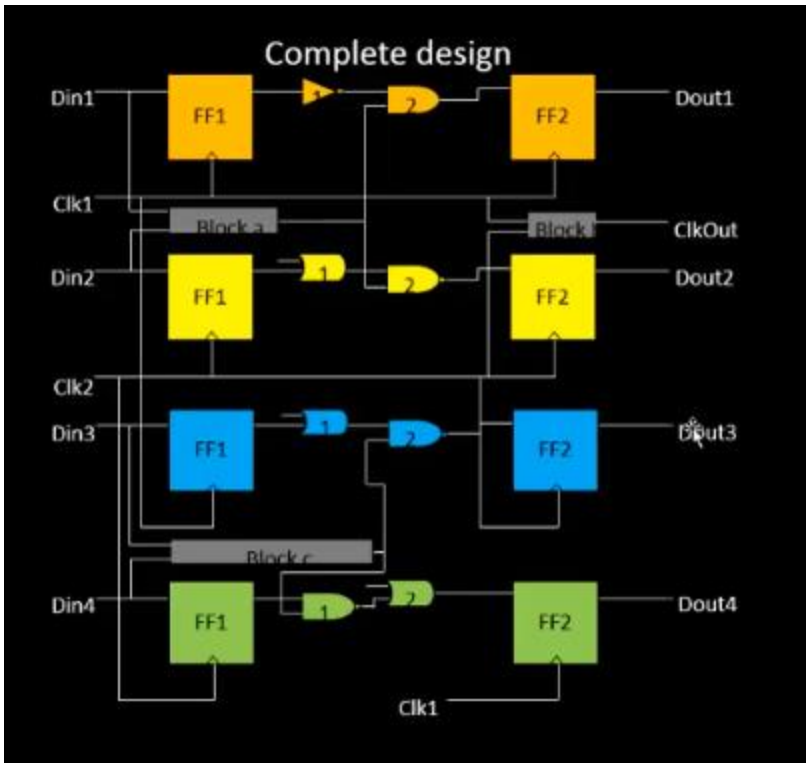
Instead of single power supply as before, there are multiple Vdd and Vss lines. If a logic demands current, it can tap current from the nearest power supply.


So what could be the solution ?
Driver
Load
Cd
CL
Vdd
Power Supply
How can 'Driver' and 'Load' can be brought close to each other in 'L' sense ?

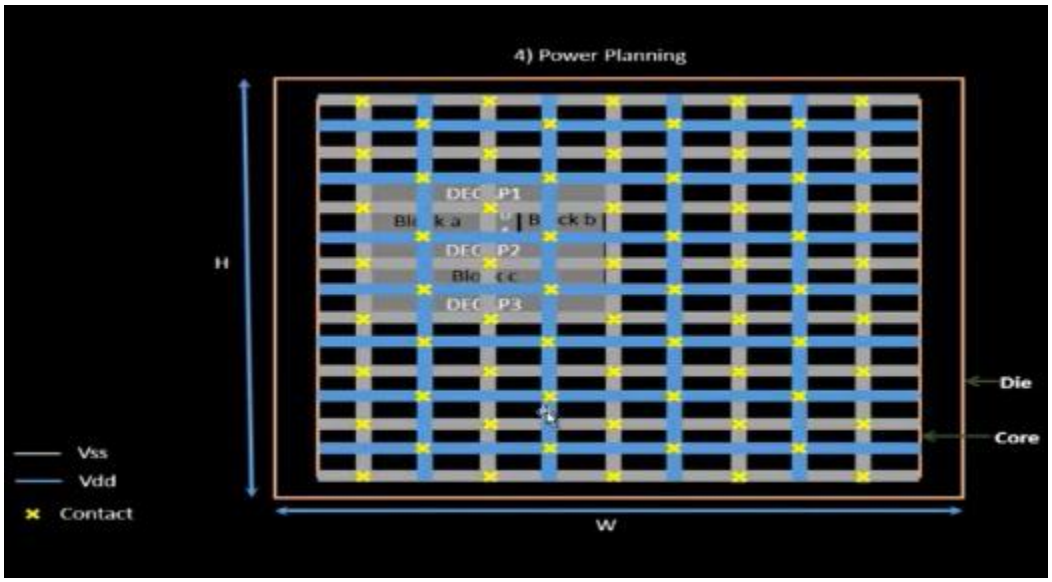

4) Power Planning
H
W
Die
Core
Vss
Vdd
Contact

Pin placement and logical cell placement blockage

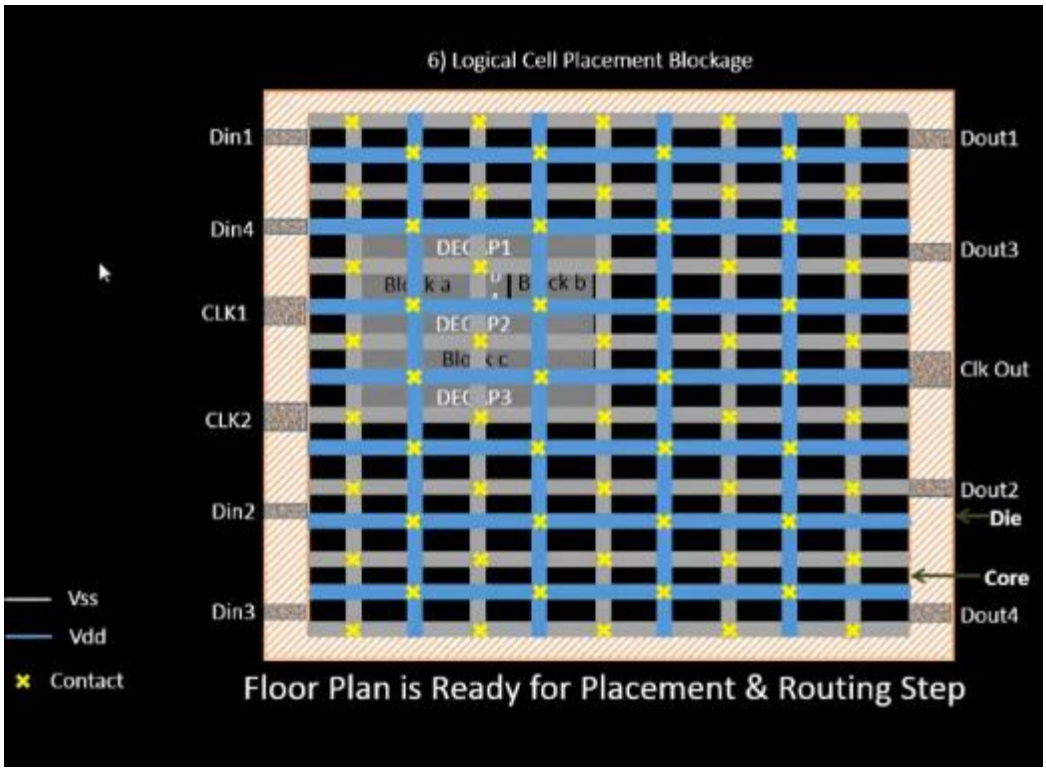
Let's take below design as an example to be implemented. The connectivity information between the gates is coded usign VHDL or Verilog language and is called as netlist.



The input and output ports are placed on the left and right spaces between the core and the die. The placements of the ports depends on where the cells are placed. The clock ports are bigger in size than data ports since the clocks are driving the cells continuously. So, we need the least resistance paths for the clocks. Bigger the size, lesser the resistance.

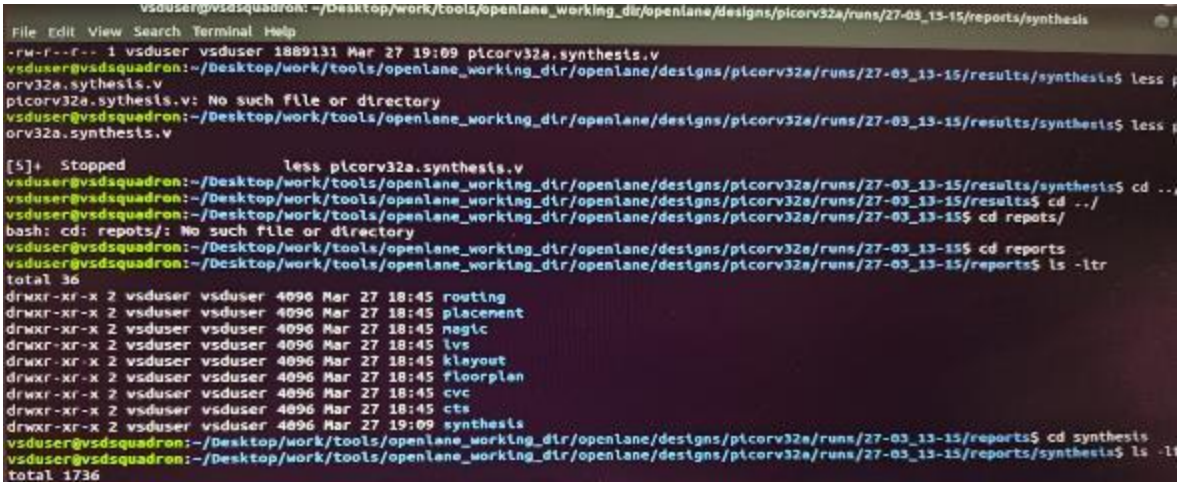


Once pin/port placement is done, Logical Cell Placement Blockage is created to make sure that the APR tool does not place any cell on the pin locations.



Steps to run floorplan using OpenLANE and view floorplan layout in Magic

1. Setting configuration variables: Before running floorplan, the configuration variables or switches must be set. These are present in openlane/configuration directory:



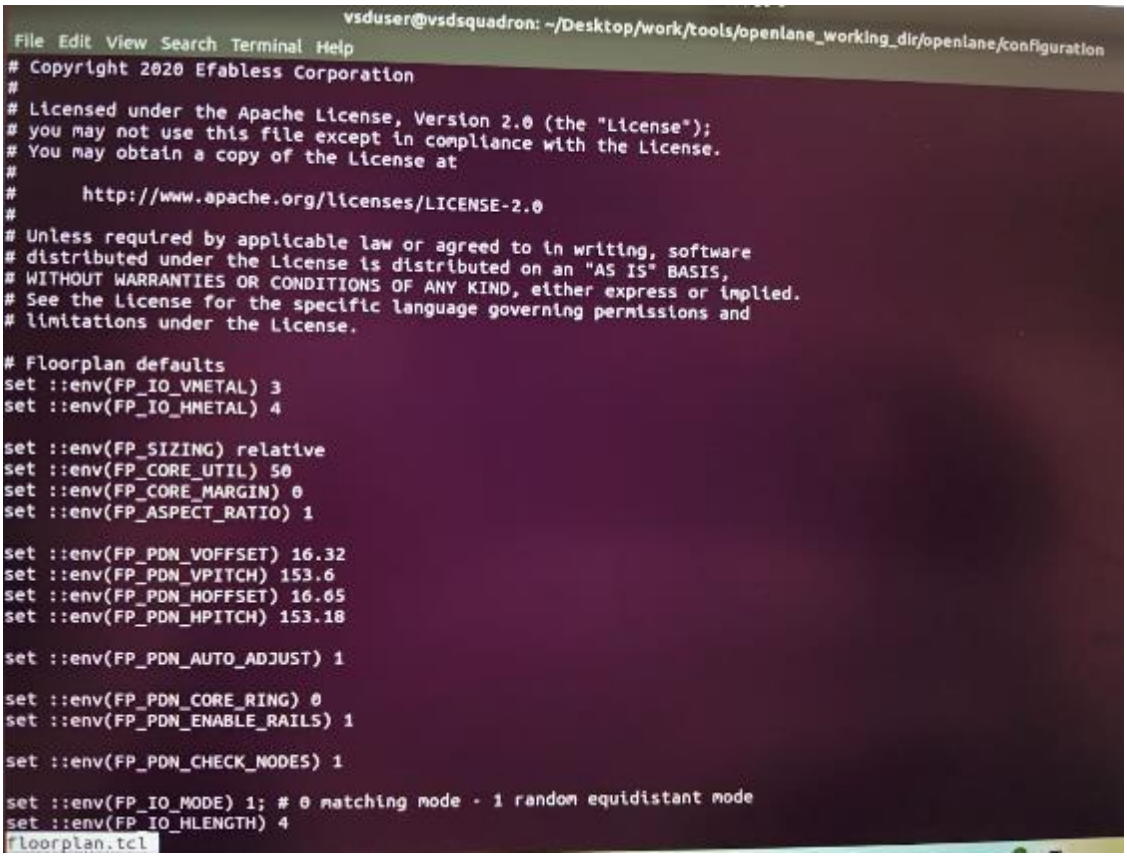
The README.md consists of all configuration variables for every stage and the tcl files contain the default OpenLANE settings.

2. Default parameters are set for floorplan stage in floorplan.tcl in OpenLANE

3. All configurations/switches accepted by the current run are from openlane/designs/[design]/config.tcl

The priority order from highest to lowest is as follows:

- openlane/designs/[design]/sky130A_sky130_fd_sc_hd_config.tcl
- openlane/designs/[design]/config.tcl
- openlane/configuration/floorplan.tcl



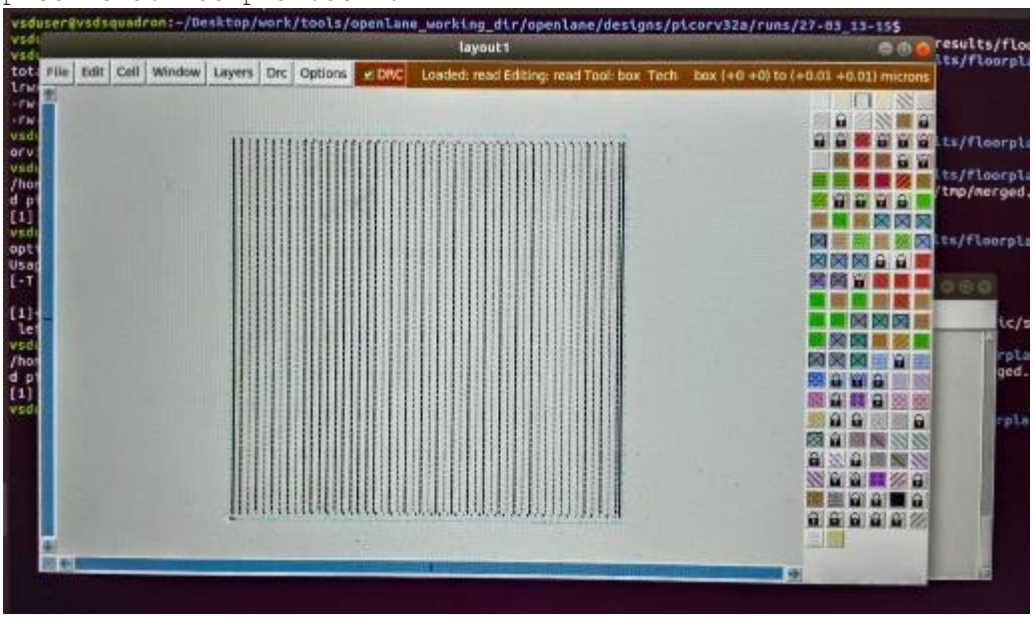
In OpenLANE flow, the vertical and horizontal metals are one more than what we specify. If vertical metal is specified as 4, then it'll be 5, same case for horizontal.

4. Run floorplan on OpenLane: `% run_floorplan`

5. Floorplan output files are generated in this folder openlane/designs/picorv32a/runs/date/results/floorplan/picorv32a.floorplan.def which is a design exchange format, containing the die area and positions. The die area in this file is in database units and 1 micron is equivalent to 1000 database units. Area of die = (554570/1000) microns * (565290/1000) microns = 311829.1653 um^2

6. To view the layout after floorplan, we use magic tool

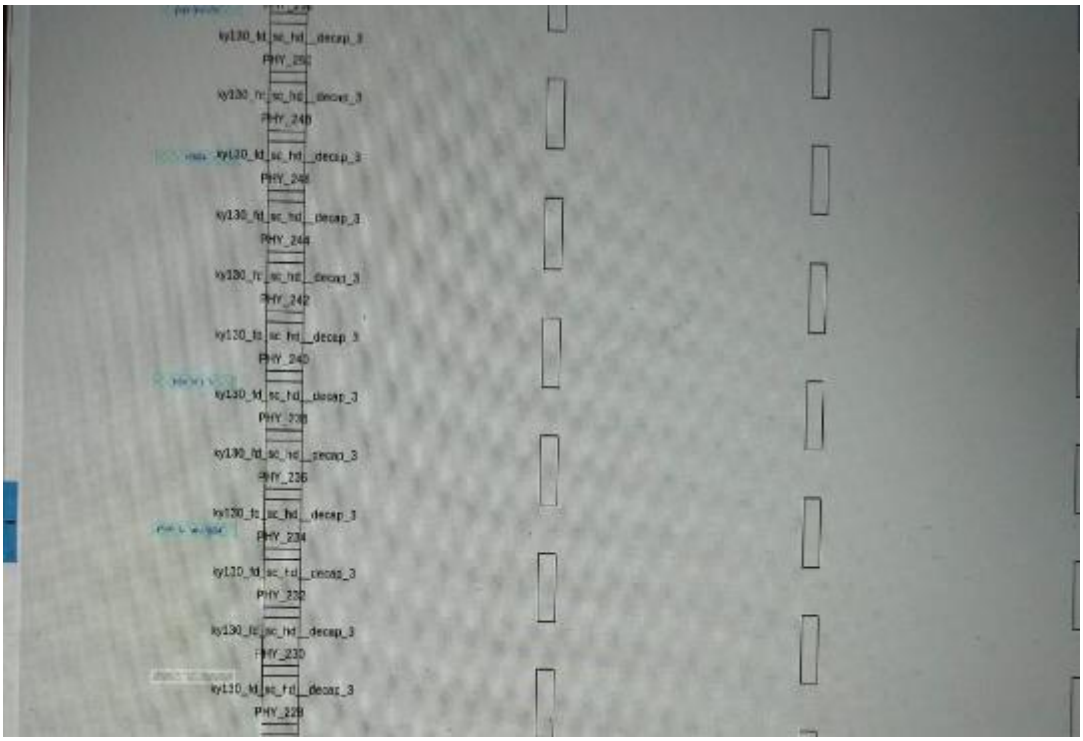
   Command: `magic -T /home/vsduser/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.tech/magic/sky130A.tech lef read ../../tmp/merged.lef def read picorv32a.floorplan.def &`



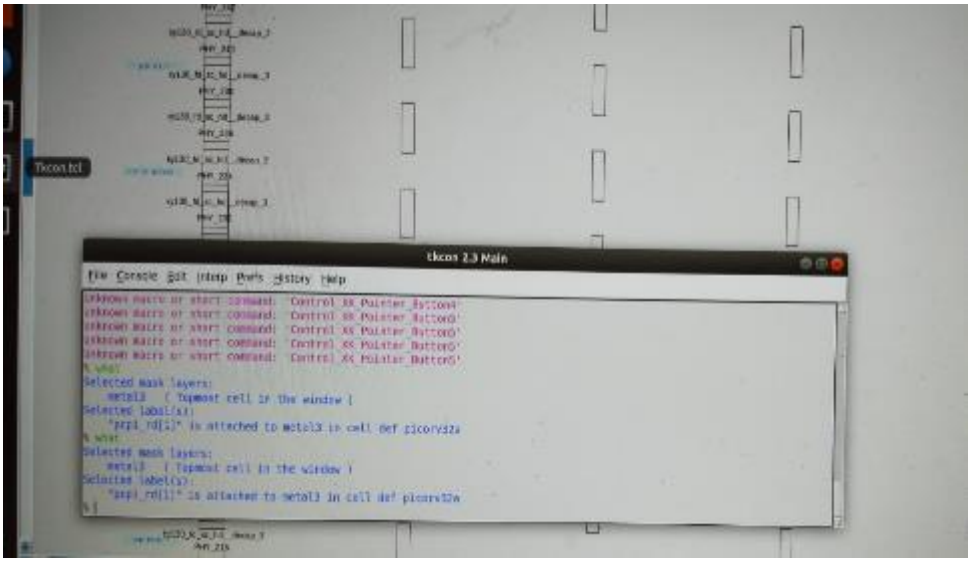
   Press "s" to select whole die then press "v" to center the view

   Point the cursor to a cell then press "s" to select it, zoom into it by pressing "z"

   The IO pins are placed in a random equidistant mode as seen below based on the configuration (FP_IO_MODE = 1) set in openlane/configuration/floorplan.tcl



The components in the layout can be identified by using the "what" command in tkcon window after selecting it.



Standard cells are not placed but can be viewed at the bottom left corner of the layout

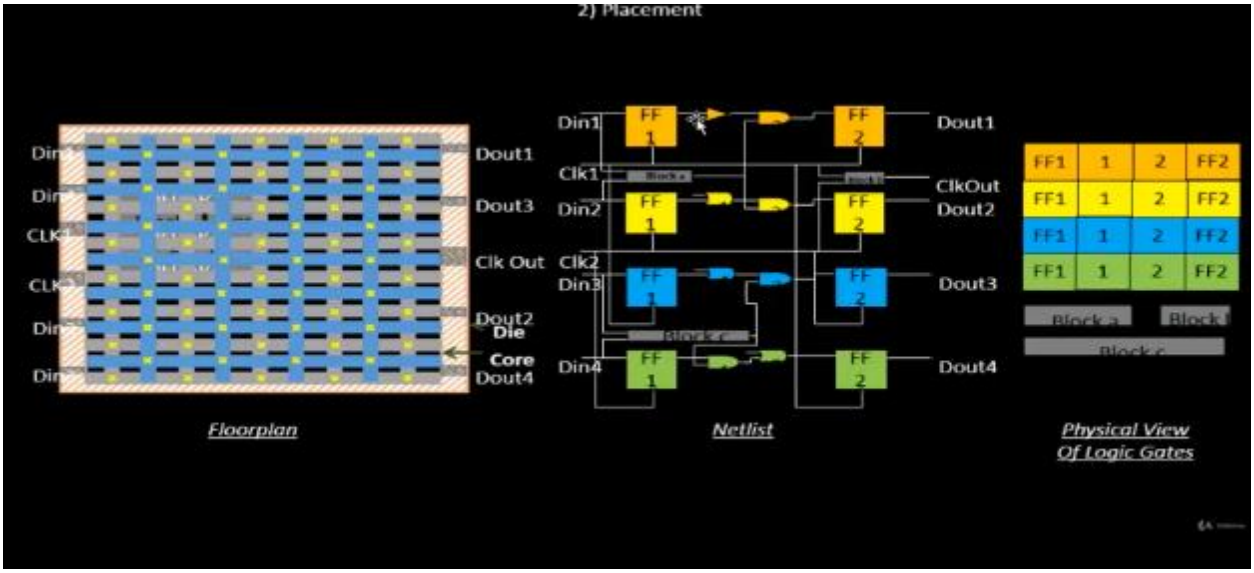
Library Binding and Placement

Netlist binding and initial place design

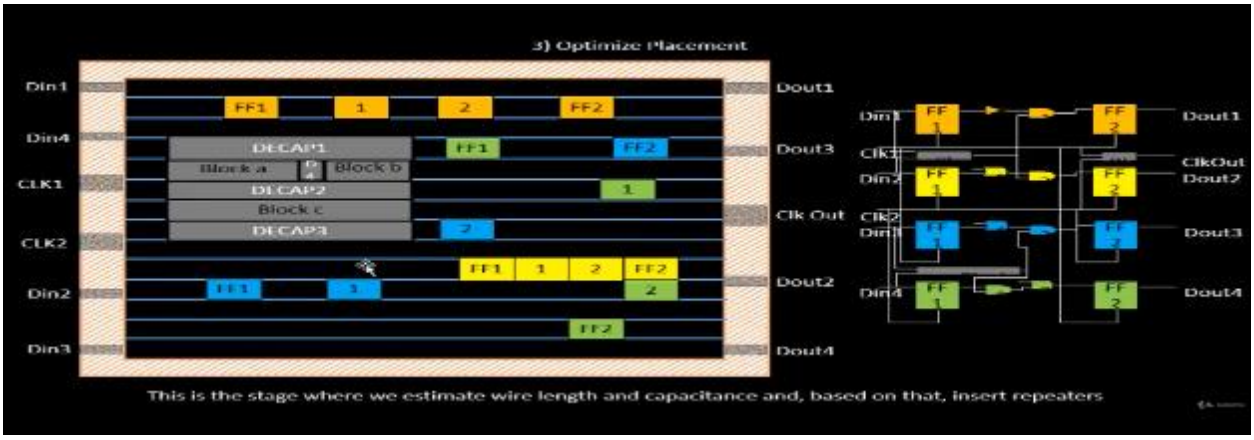
1. Bind netlist with physical cells

   Bind the netlist to physical cells with real dimensions. The physical cells come from a library that contains cells which can have different dimensions, various shapes of the cells, and delay information. Bigger cells have lesser resistance while the functionality is the same. This means that the library has many flavors of cells.

2. Placement:

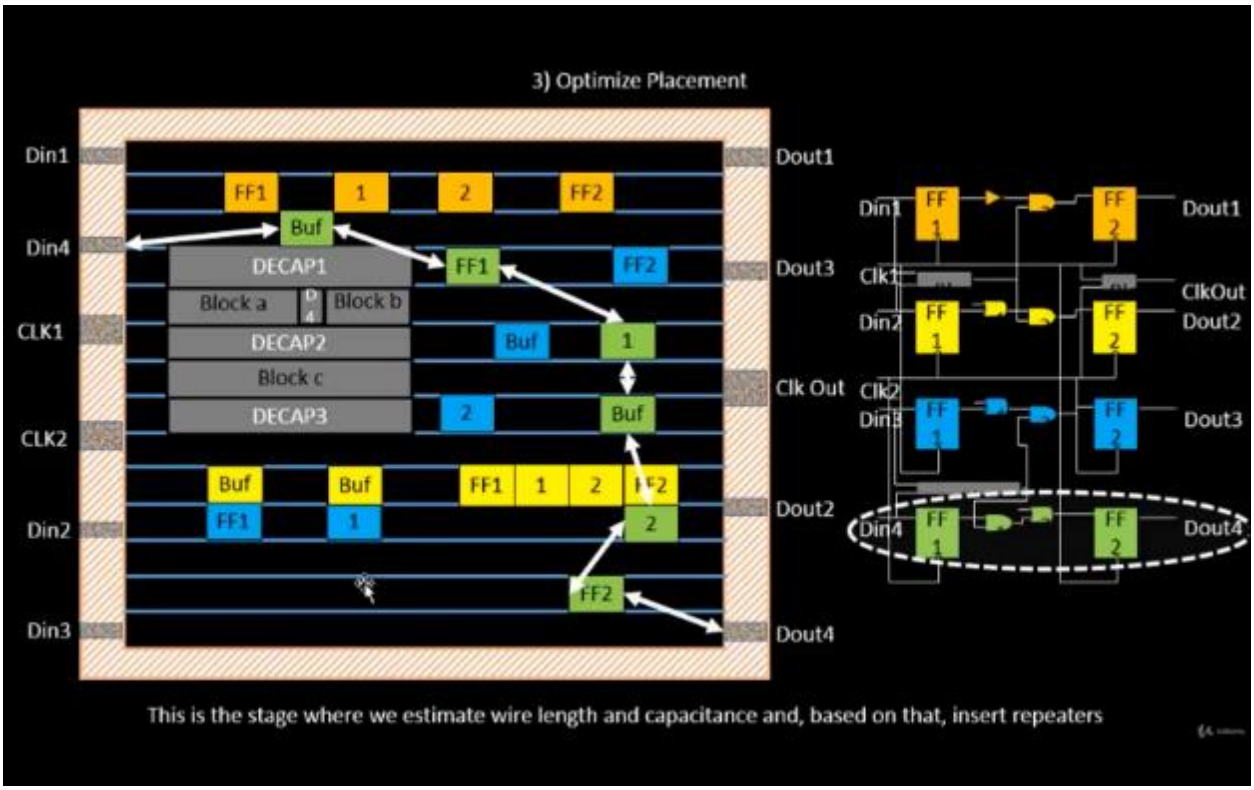


   Placement is done based on connectivity. For example, FF1 is close to Din1 pin and FF2 close to Dout1 pin and combinational cells placed nearer to FF1 and FF2. This is to reduce delay.



Optimize placement using estimated wire-length and capacitance

This is the stage where we estimate wirelength and capacitance (C=EA/d) and insert repeaters based on that. If the wirelength is more, then to maintain signal integrity, we add repeaters to reduce resistance. Repeaters basically reconditions the original signal and transfers.



Congestion aware placement using RePlAce

Run placement on OpenLane: `% run_placement`
This commmand is a wrapper which does global placement (by RePlace tool), Optimization (by Resier tool), and detailed placement (by OpenDP tool).

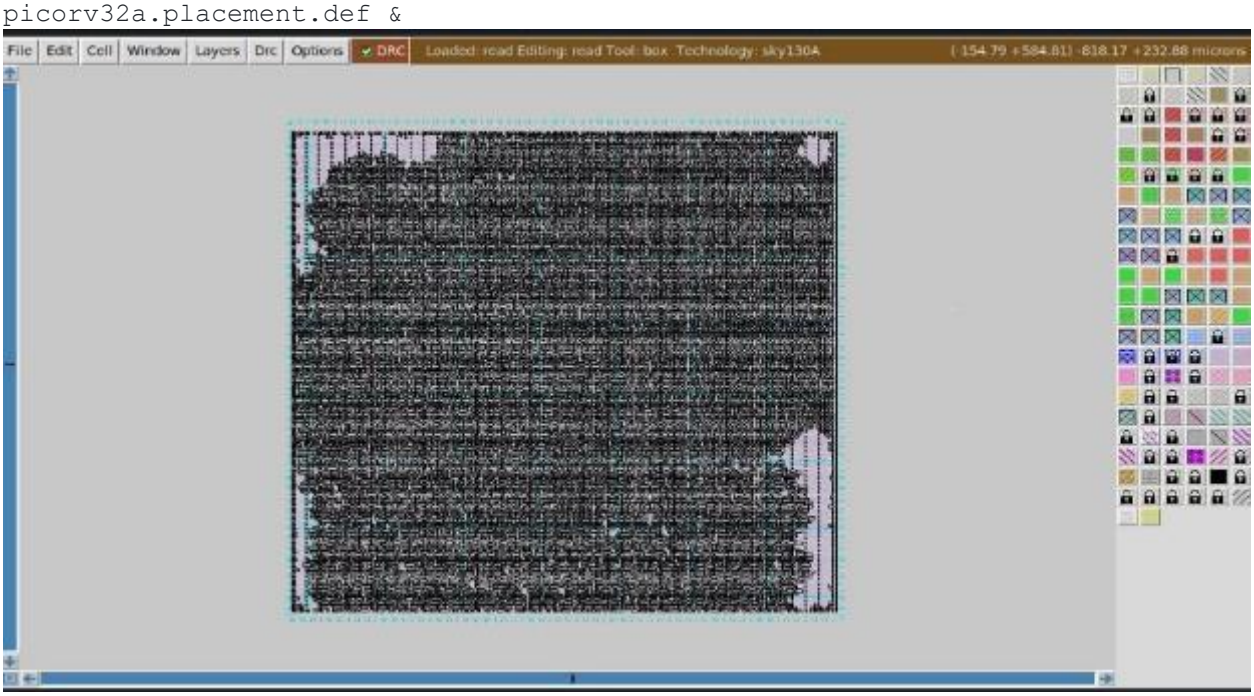
Placement is done in two stages:

- Global Placement : no legalization takes place and uses Half Perimeter Wirelength (HPWL) reduction model.
- Detailed Placement : legalization happens where the standard cells are placed in stadard rows, and there will be no overlaps of the cells.

The objective of placement is to converge the overflow value. If overflow value progressively reduces during the placement,then it implies that the design will converge and placement will be successful.

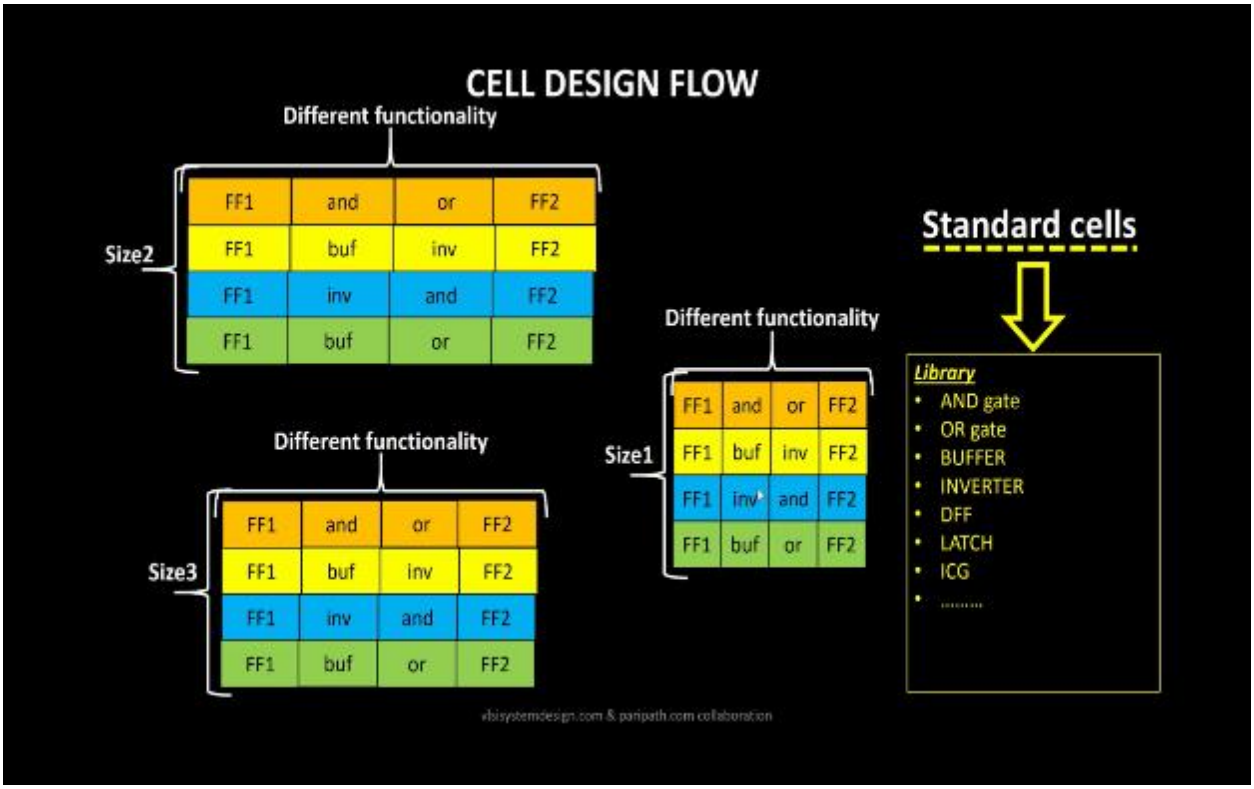
After running the placement, output is generated in this folder
/openlane/designs/picorv32a/runs/date/results/placement/picorv32a.placement.def

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 &`



Cell design and characterization flows

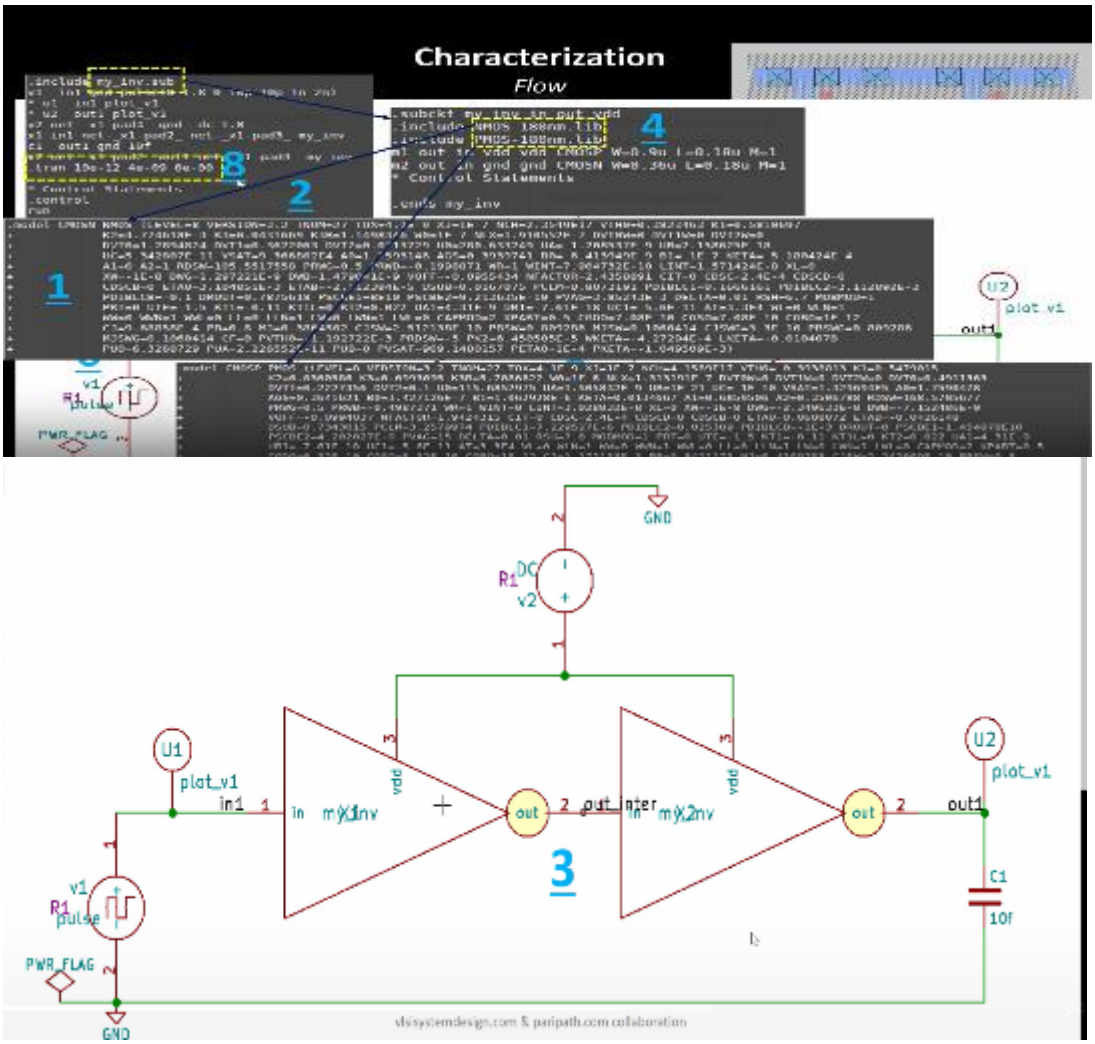
All standards cells (AND, OR, BUFFER, INVERTER, flip-flops etc.) are present in standard cell library. The cells inside the library are of different flavors (different drive strengths, functionality, threshold voltage). If the cell size is more, then the drive strength is high to drive longer wires. If the threshold voltage is high, then it will take more time to switch than the one with lesser threshold voltage.



- DRC & LVS Rules: tech files and poly substrate parameters
- SPICE Models: Threshold, linear regions, saturation region equations with added foundry parameters, including NMOS and PMOS parameteres
- User defined Spec: Cell height, cell width (drive strength), supply voltage, pin locations, metal layer requirement
- The standard cell library developer must adhere to the rules given so that when the cell can be used on a real design without any errors
- Circuit design is done modeling the pmos and nmos to meet input library requirement
- Layout design is done using Euler's path and stick diagram on Magic layout too

Steps of characterization flow:

1. Read the spice model files
2. Read the extracted spice netlist
3. Define/Recognise the buffer behavior
4. Read the subcircuits
5. Attach the necessary power sources
6. Apply stimulus
7. provide necessary output capacitance
8. provide simulation command

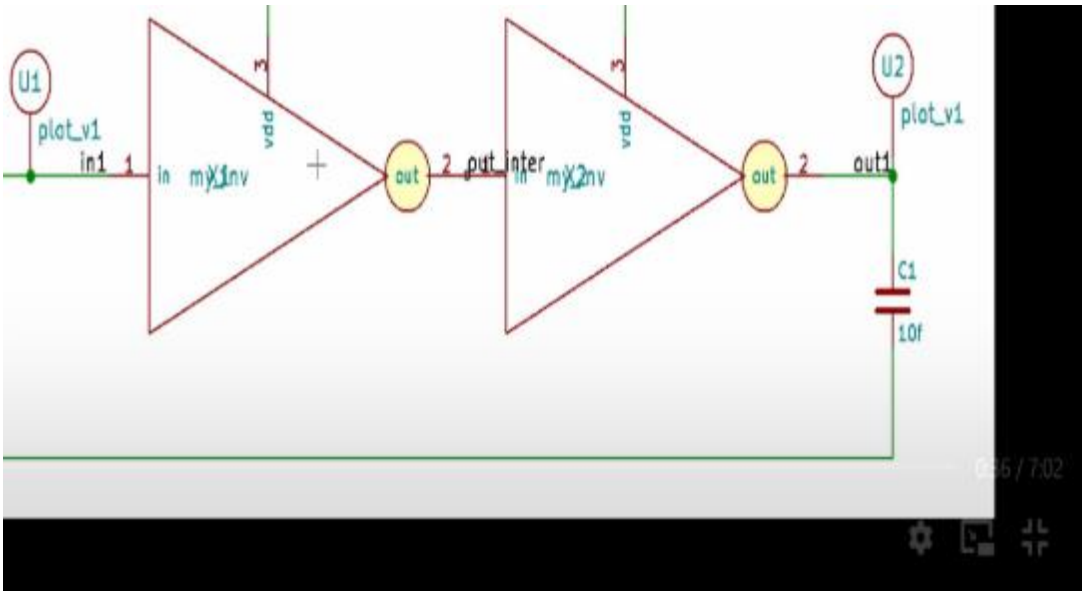


Steps 1 to 8 are fed in the form of configuration file to the characterization software called "GUNA".

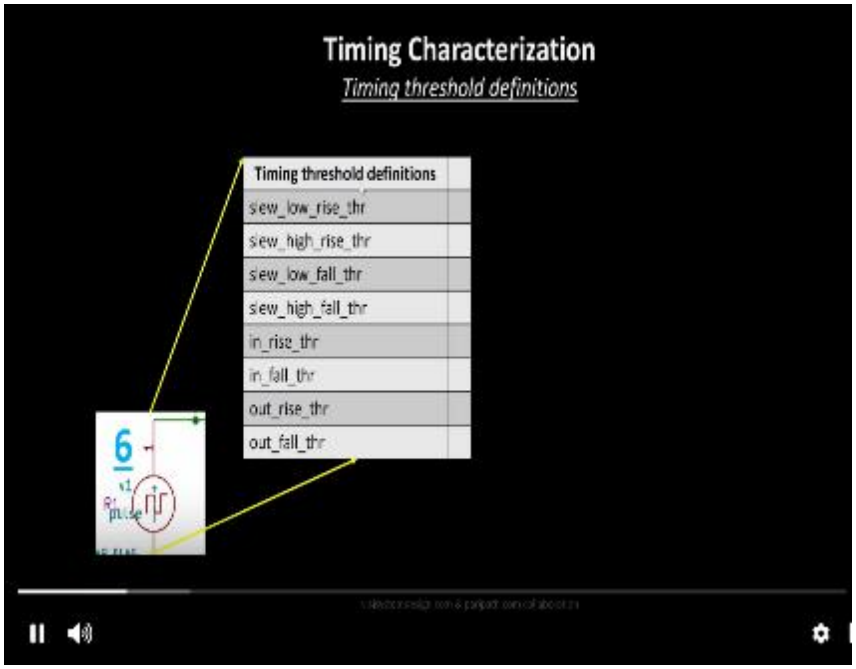
Timing characterization

Syntax and semantics of power.lib, timing.lib and noise.lib. These are necessary to understand the GUNA software workflow.

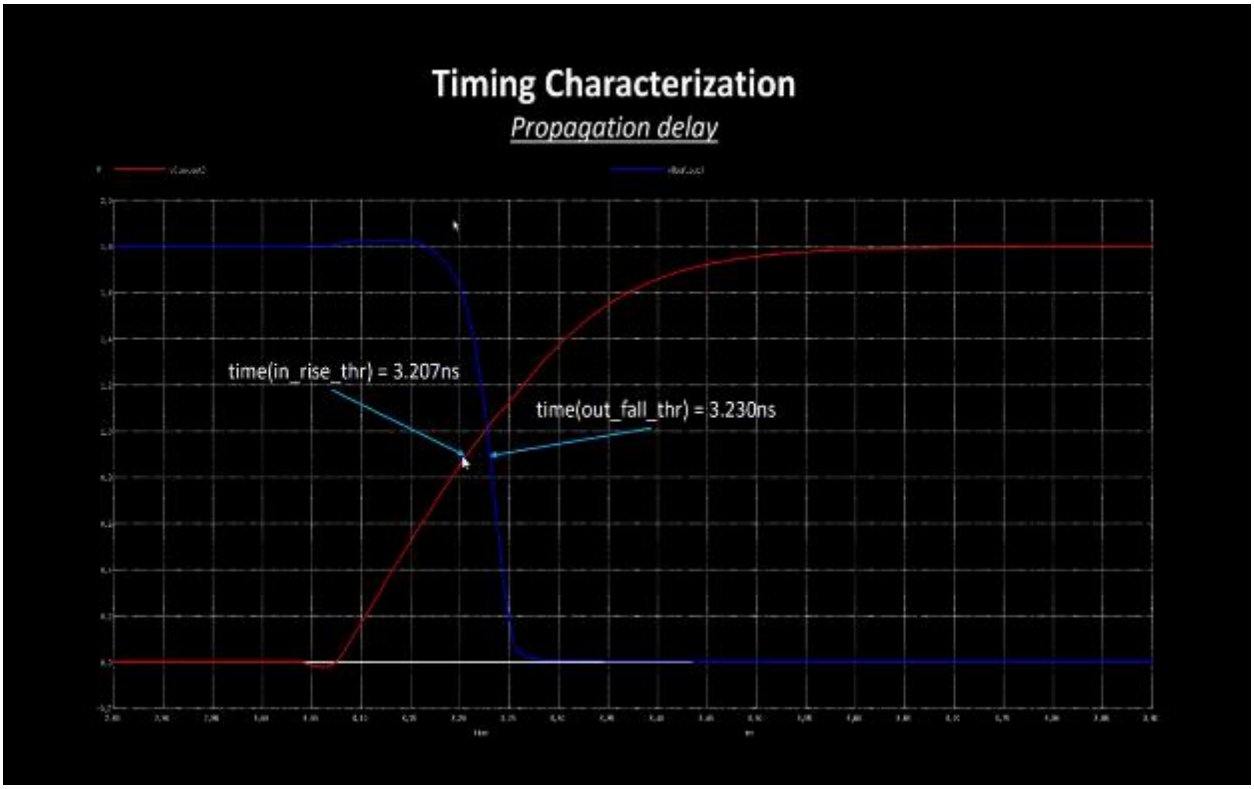
We are taking inverters connected back to back as an example.



Timing thershold definitions:



Two inverters in series, red is output of first inverter and blue is output of second inverter:

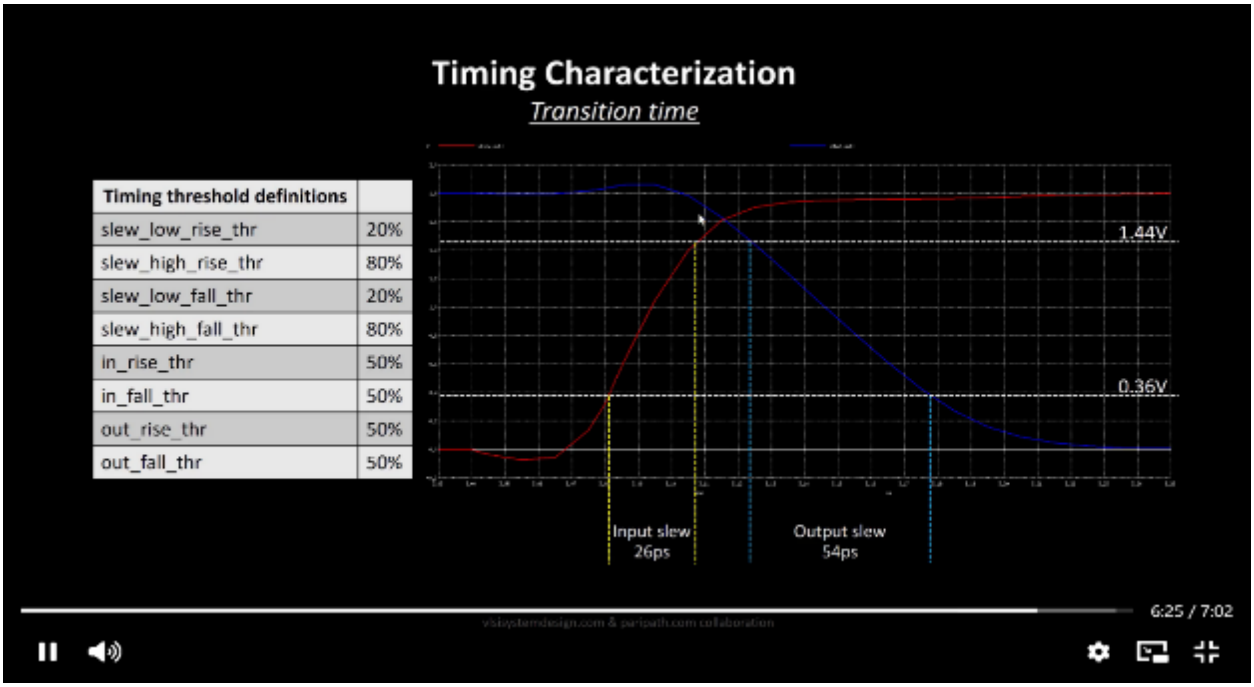


The red is input waveform and blue is output waveform of the buffer. The left side is rise delay and right side is fall delay.

PROPOGATION DELAY= time(out_*_thr)-time(in_*_)

TRANSITION DELAY=time(slew_high_*_thr)-time(slew_low_*_thr)

Negative propagation delay is not expected. This means that the output comes before the input so it's important to choose correct threshold point to produce positive delay. Delay threshold is usually 50% and slew rate threshold is usually 20%-80%.



Design library cell using Magic Layout and ngspice characterization

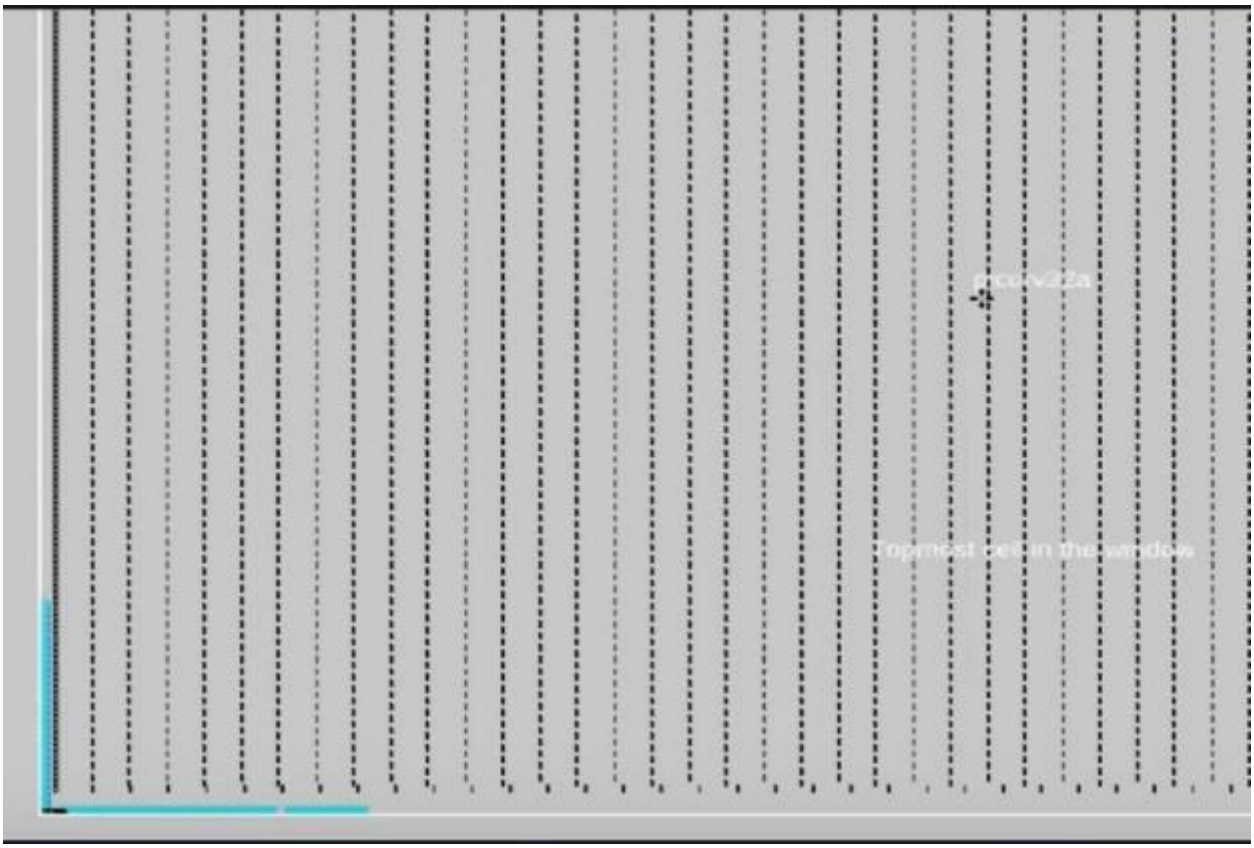
Labs for CMOS inverter ngspice simulations

IO placer revision

Configuration settings in OpenLANE can be changed in the shell itself, on the fly. For example, to make IO_mode not to be "random equidistant",

```
% set ::env(FP_IO_MODE) 2
```

The IO pins will not be equidistant in mode 2 (default of 1). On re-running floorplan, we can see that the pins are placed based on of the Hungarian algorithms. The pins are stacked one over the other. However, changing the configuration on the fly will not change the runs/config.tcl, the configuration will only be available on the current session.



To echo current value of variable,

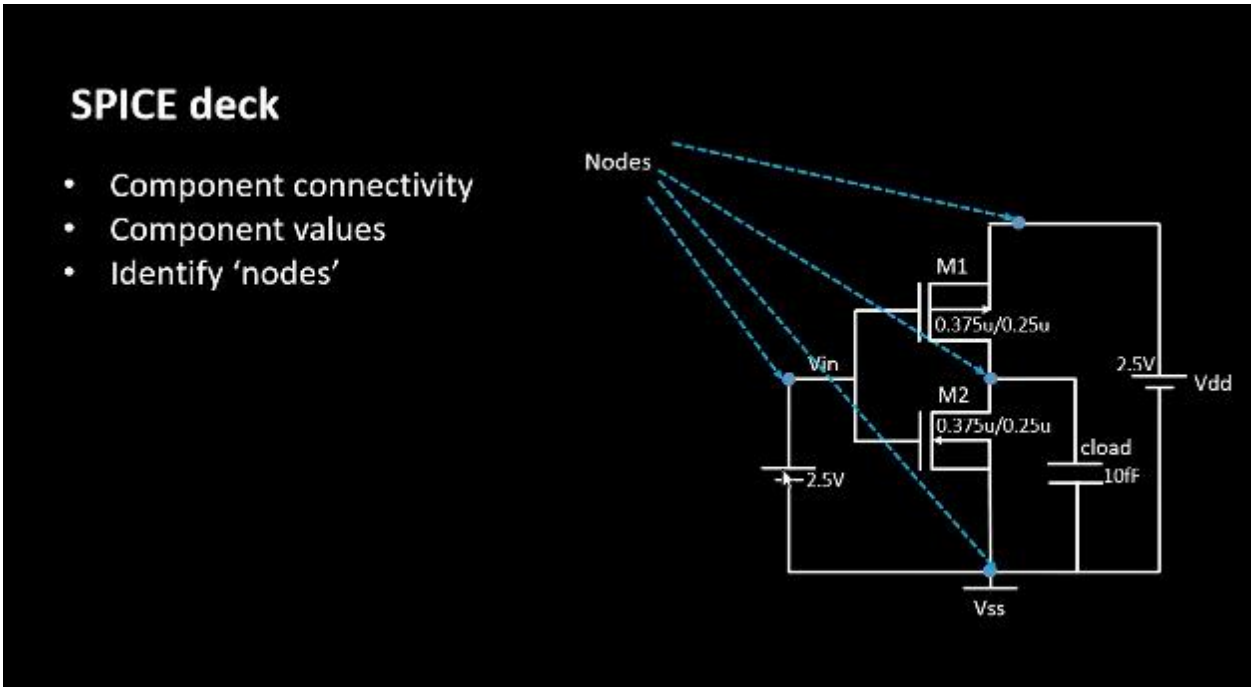
```
echo $::env(FP_IO_MODE)
```

SPICE deck creation and simulation for CMOS inverter

SPICE deck comprises of connectivity information about the netlist, inputs to be provided to the simulation, information on tap points at which output will be taken and so on. Component values in SPICE DECK: For PMOS, W/L (0.375u/0.25u means width is 375nm and lengthis 250nm). PMOS should be wider (atleast 2x or 3x) than NMOS. PMOS hole carrier is slower than NMOS electron carrier mobility, so to match the rise and fall time, PMOS must be wider (less resistance thus higher mobility) than NMOS. But in this case, we are taking same sizes for both PMOS and NMOS. The gate voltage is normally a multiple of length (250nm) (in the example, gate voltage can be 2.5V)

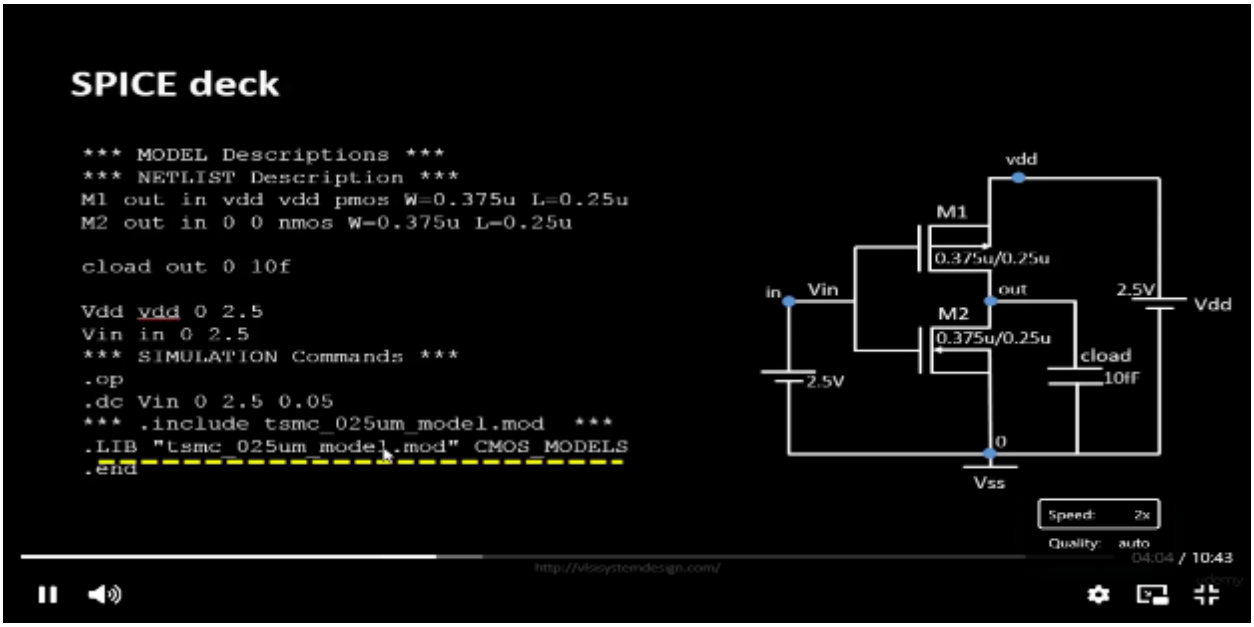
SPICE deck:

- component connectivity
- component values
- identify nodes
- name nodes



SPICE deck netlist description:

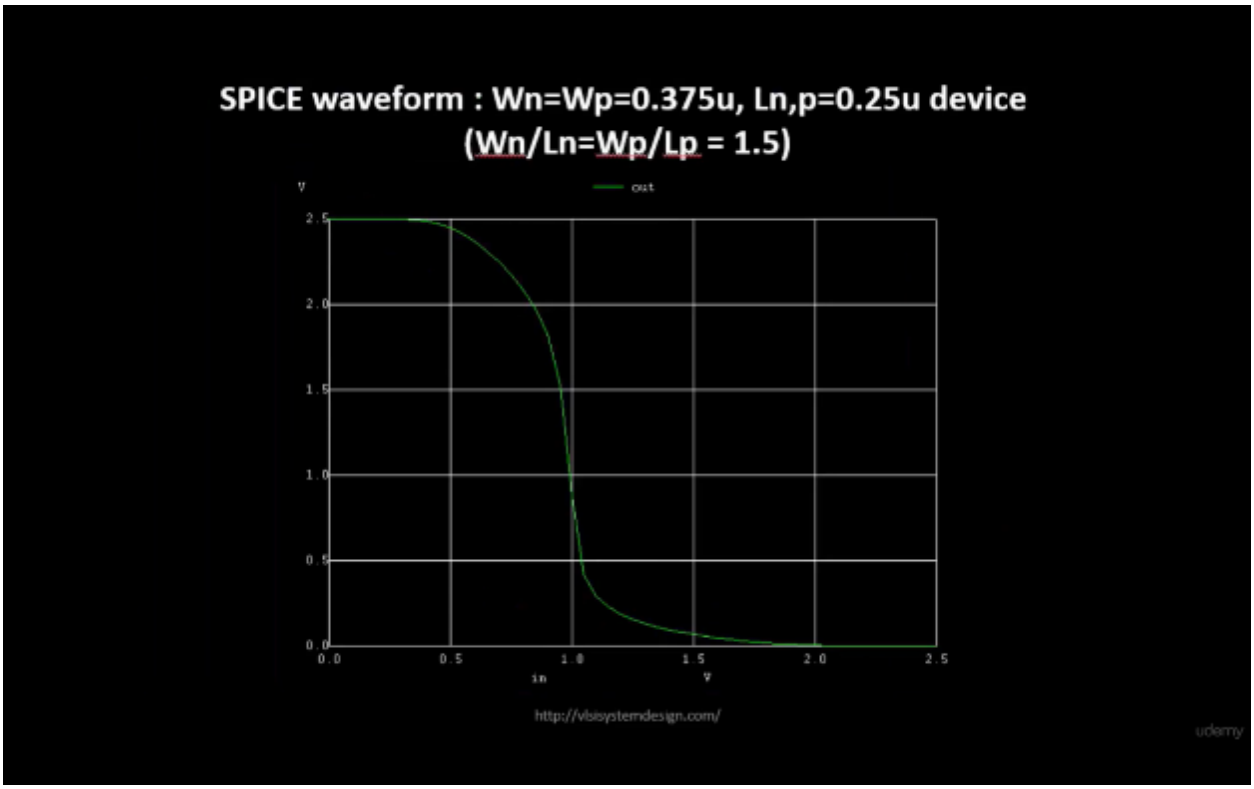
- Syntax for the PMOS and NMOS: [component name] [drain] [gate] [source] [substrate] [transistor type] W=[width] L=[length]
- All components are described based on nodes and its values
- .op is the start of SPICE simulation operation where Vin will be sweep from 0 to 2.5 with 0.05V steps
- tsmc_025um_model.mod is the model file containing the technological parameters for the 0.25um NMOS and PMOS



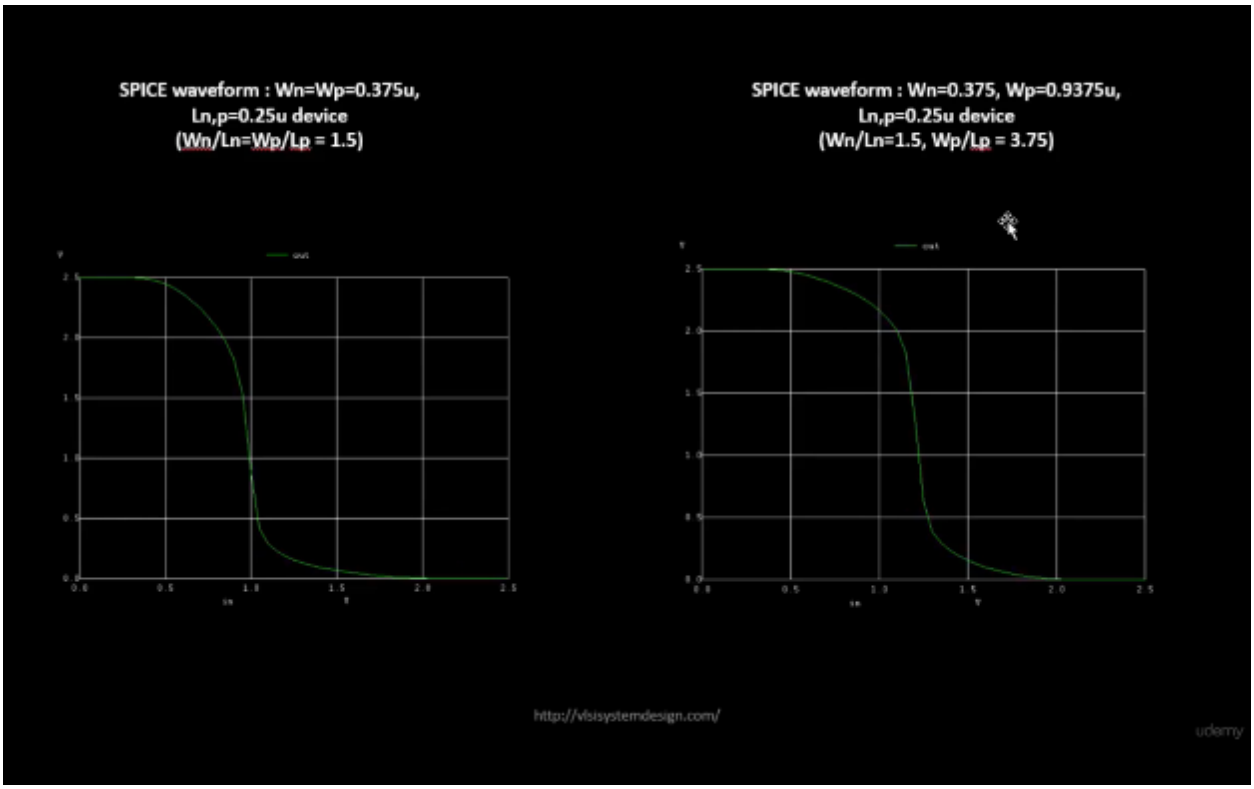
SPICE simulation:

The steps to follow for SPICE simulation,

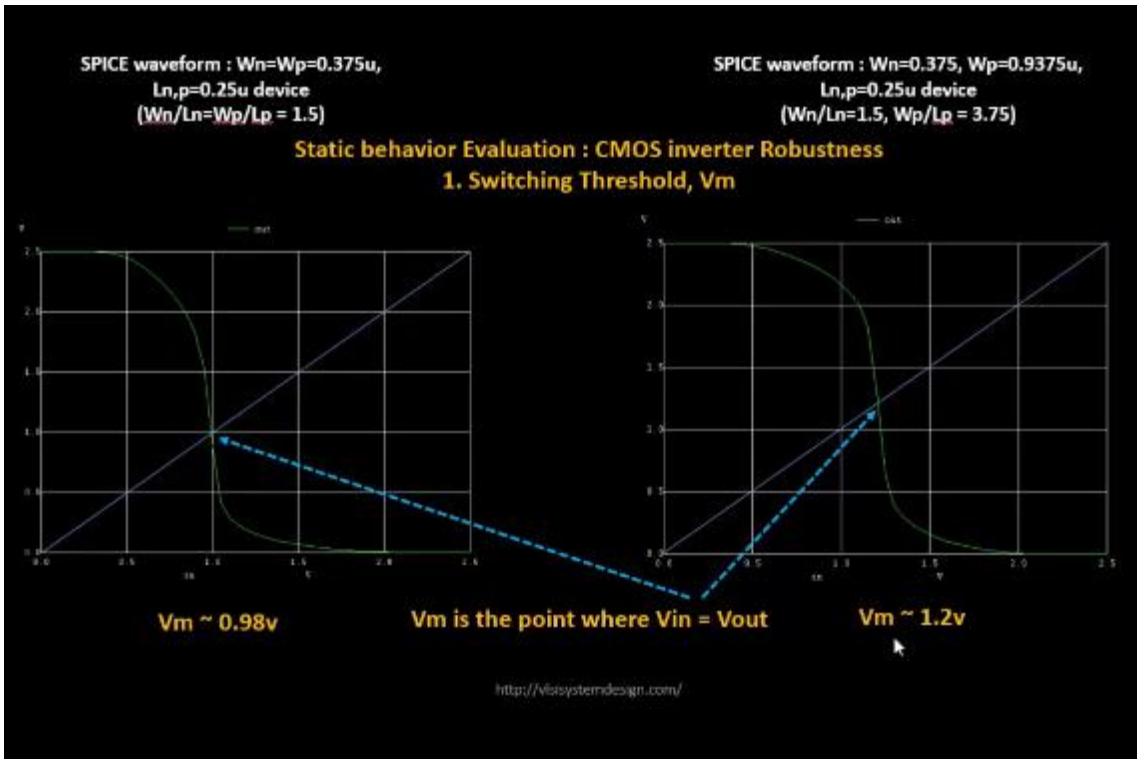
1. Go to ngspice simulator
2. Source the .cir spice deck file
3. Execute the spice file by command: run
4. Execute command: setplot --> allows you to view any plots possible from the simulations specified in the spice deck
5. Select the simulation desired by entering the simulation name in the terminal
6. Execute command: display --> to see which nodes available for plotting
7. Execute command: plot out vs in We can see the plot for above inputs. In this the width of both PMOS &NMOS is same.



Switching Threshold Vm



1. The shapes are almost the same which means that CMOS is a robust device.
2. Parameters that defines the robustness of CMOS
    - switching threshold, Vm. It is the point where the Vin = Vout and both PMOS & NMOS are in saturation region. These will be turned on and there is high chances for leakage. There is a high possibility that the current flows directly from VDD to GND. Due to this, short circuit kind of device is seen.

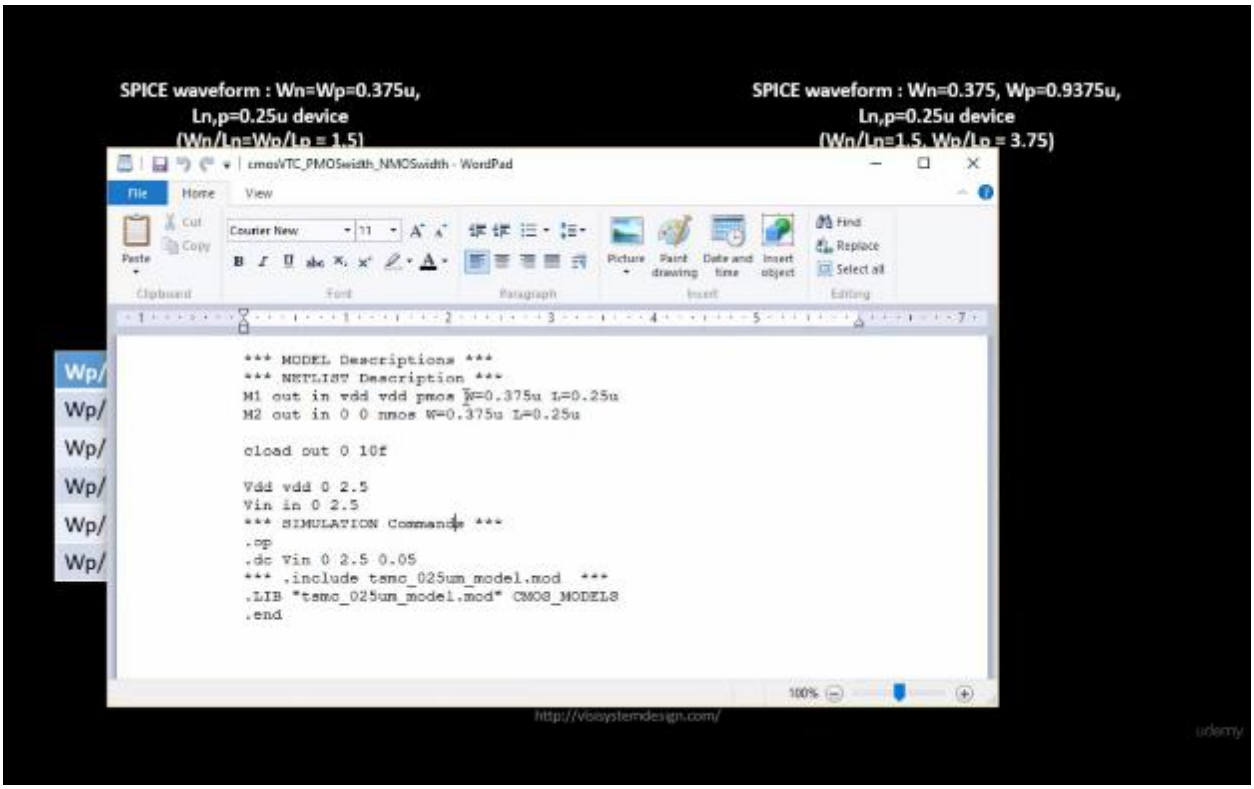
- 

- Propagation delay: rise or fall delay

Static and dynamic simulation of CMOS inverter

DC transfer analysis is used for finding switching threshold. Simulation is DC sweep from 0V to 2.5V with 0.05V steps:

When a pulse is applied to the CMOS, transient analysis is used to find propagation delay.

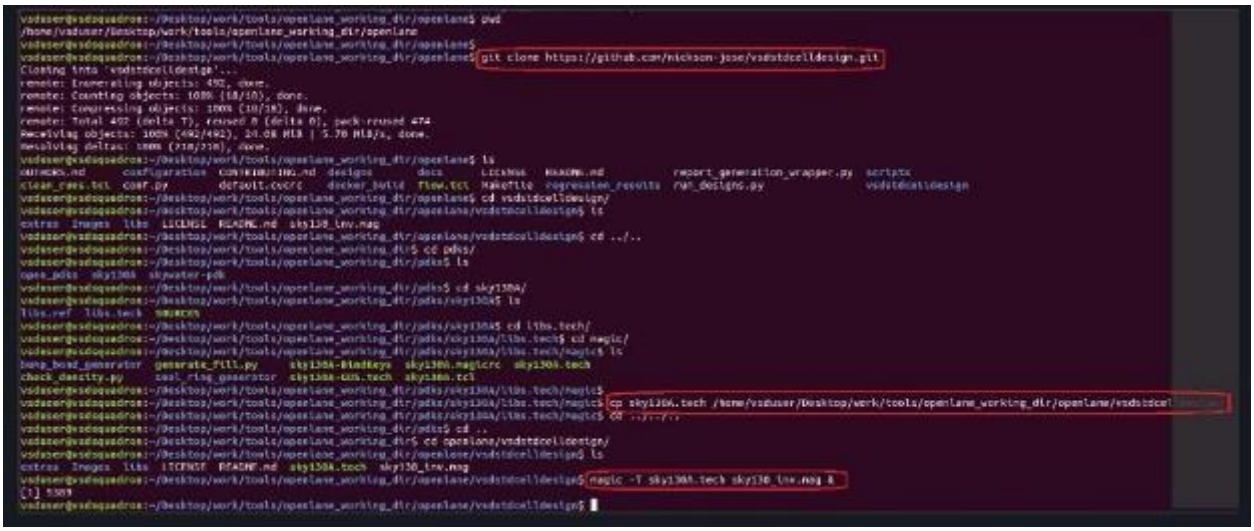


Lab steps to gitclone vsdstdcelldesign

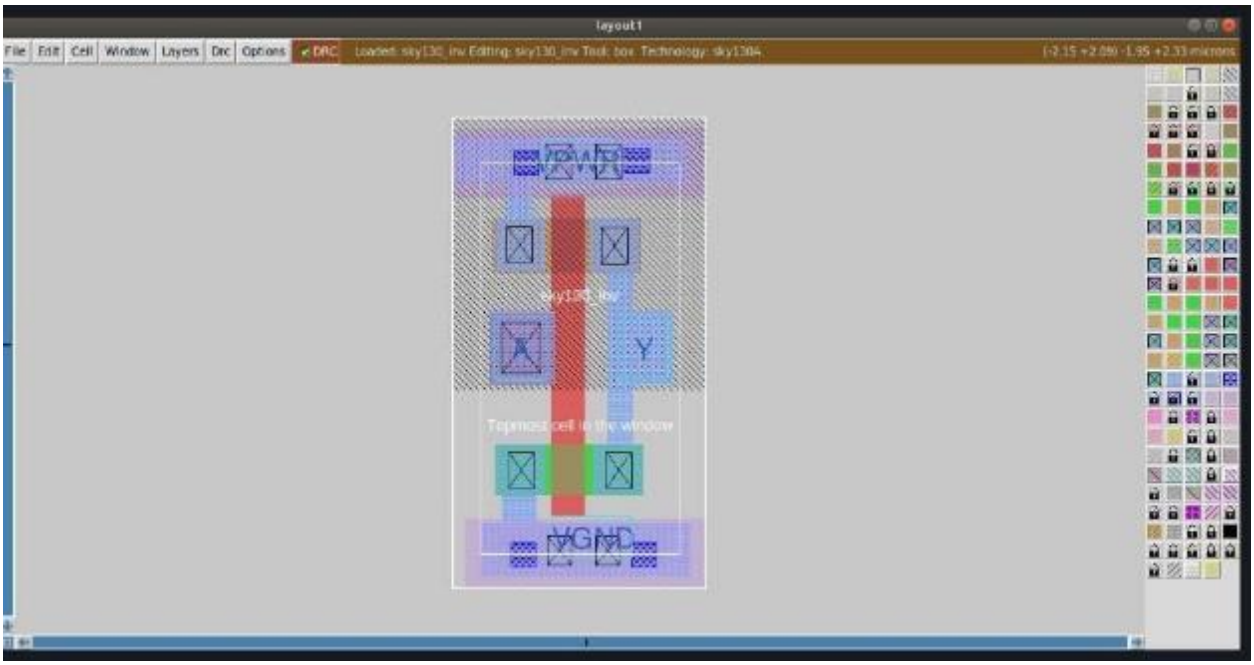
Instead of designing an inverter from scratch, a github repository where this is already done is cloned to observe the layout.

1. Clone custom inverter standard cell design from [github repository]
2. Change directory to openlane: cd ~/Desktop/work/tools/openlane_working_dir/openlane
3. Clone the repository with custom inverter design: git clone [https://github.com/nickson-jose/vsdstdcelldesign]
4. Copy tech file to the vsdstdcelldesign directory: cp /home/vsduser/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.tech/magic/sky130A.tech /home/vsduser/Desktop/work/tools/openlane_working_dir/openlane/vsdstdcelldesign/
5. Open custom inverter layout in magic: magic -T sky130A.tech sky130_inv.mag &

Snippet of commands executed:



Snippet of sky130_inv:



Inception of Layout CMOS fabrication process

CMOS Fabrication Process (16-Mask CMOS Process):

1. Select a substrate

2. Create active region for transistors

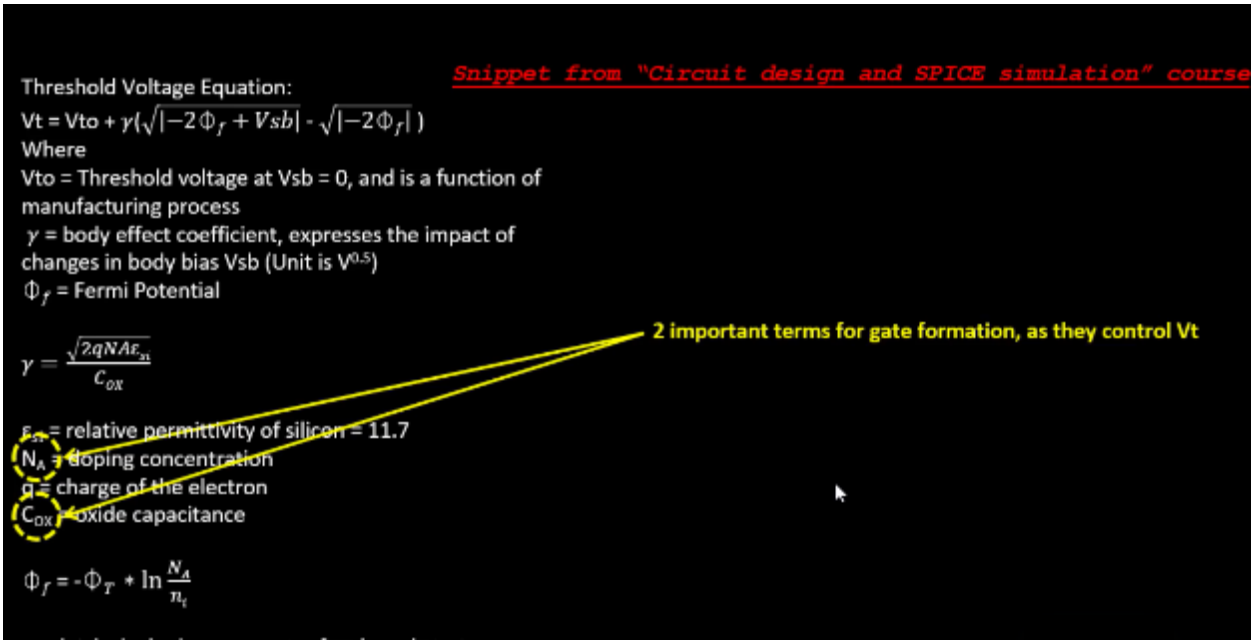
    - Deposit Silicon Dioxide (SiO_2) on the substrate
    - Deposit Silicon Nitride (Si_3N_4). It is a protection layer to prevent SiO_2 layer to grow during oxidation
    - Deposit a layer of photoresist
    - Deposit mask-1 layer on top of photoresist. It covers the photoresist layer that must not be etched away (protects the two transistor active regions)
    - UV light is applied to remove unmasked portions
    - Remove mask-1 and photoresist layers
    - Place in furnace to grow the oxide in the other areas
    - Remove the Si_3N_4 layer using hot phosphoric acid to have only p-substrate and Sio_2

3. N-Well and P-Well formation

    - Deposit photo resist layer to define the areas to protect
    - Deposit mask-2. Mask 2 protects the N-Well (PMOS side) while P-Well (NMOS side) is being fabricated then Mask 3 protects P-Well while N-Well is being formed
    - UV light is applied, and the exposed area of photoresist will be removed
    - Boron is used to form P-Well
    - Phosporus is used to form N-well
    - Place in furnace to diffuse boron and phosphorous to form wells. This process is called Twintub process.

4. Formation of gate terminal

    Gate terminal is where the threshold voltage is controlled.



- Deposit photo resist layer to define the areas to protect
- Deposit mask-4
- UV light is applied, and the exposed area of photoresist will be removed
- Implant low energy boron at the surface of p-well using mask-4 to control the threshold
- Implant phosphorous/arsenic for n-well using mask-5
- Fix the oxide which is damaged by implantation steps by removing extra SiO2 using the hydroflouric acid and re-grow high quality SiO2 on p-substrate to contol the oxide thickness
- Add polysilicon film
- Add mask-6 and etch using photolithography
- Etch off to form the gate terminal

5. Lightly Doped Drain (LDD) formation

   Two reasons for LDD: hot electron effect & short channel effect

   - Mask 7 for NMOS (lightly doped N-type)
   - Mask 8 for PMOS (lightly doped P-type)

- Heavily doped impurity (N+ for NMOS and P+ for PMOS) is for the actual source and drain but the lightly doped impurity will help maintain spacing between the source and drain and prevent hot electron effect and short channel effect.
- To protect the lightly doped regions, add SiO2 and create spacers using plasma anisotropic etching

6. Source and Drain formation

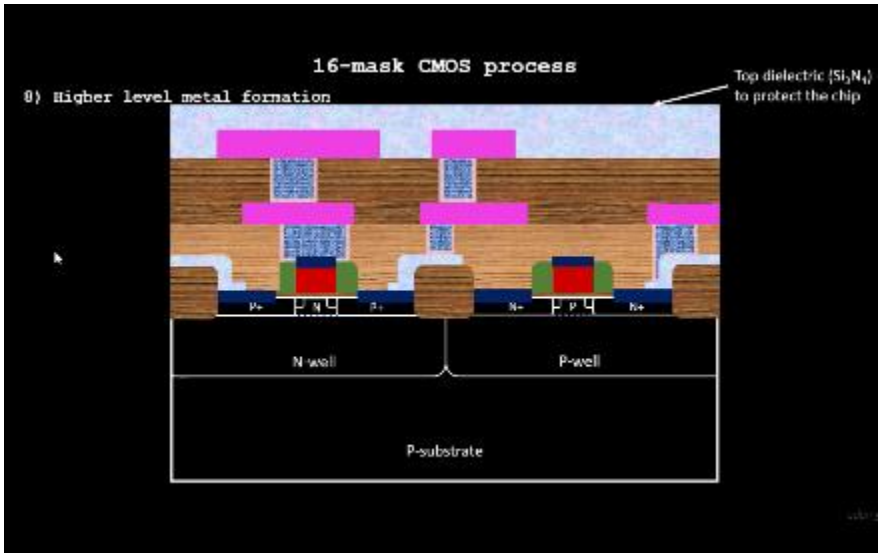
    - Thin screen oxide is formed to avoid channeling. Channeling is when implantations dig too deep into substrate.
    - Mask-9 is for N+ implantation and Mask-10 for P+ implantation
    - The side wall spacers maintain the N-/P- while implanting the N+/P+
    - High temperature annealing is done

7. Local interconnect formation

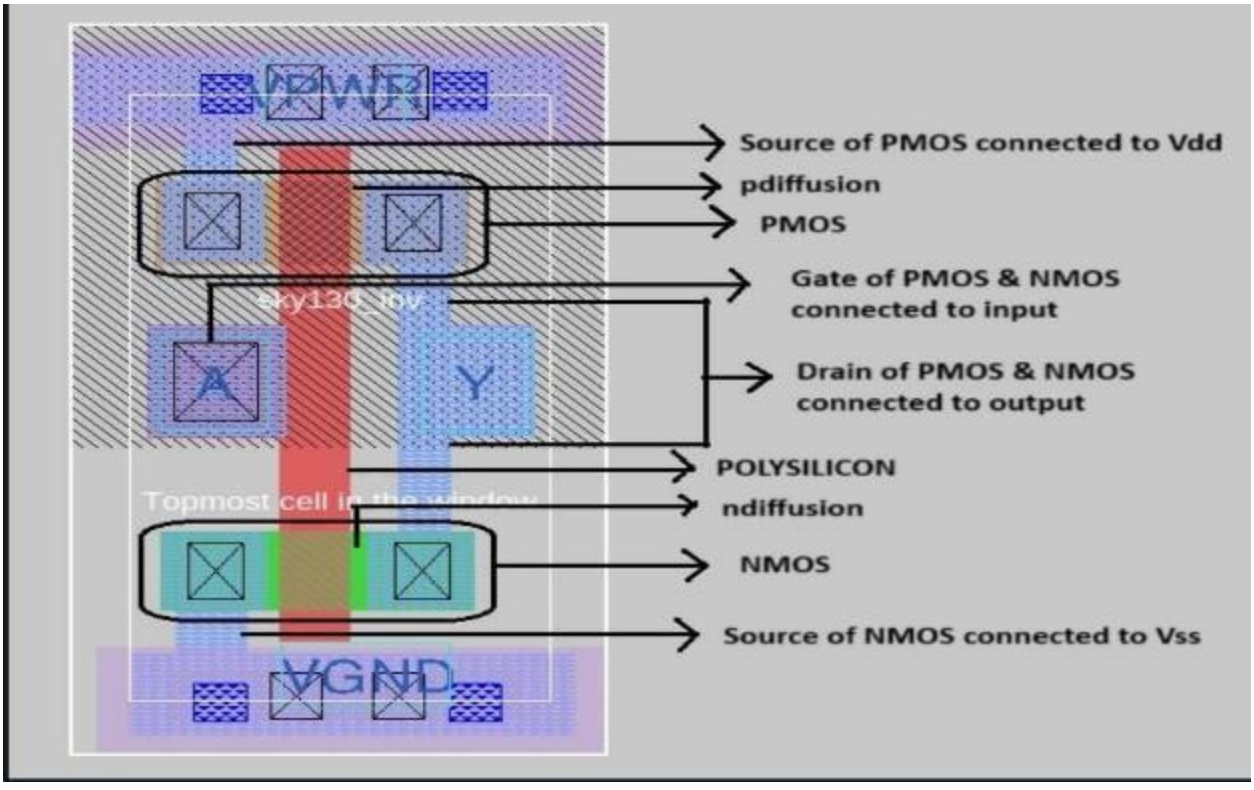
    - Contacts and interconnects are important to control the electrical characteristics by the designer.
    - Remove thin screen oxide to open up the source, drain and gate for building the contacts.
    - Titanium has less resistance and hence used
    - TiSi2 is used for local interconnects
    - Mask 11 is formed and TiN is etched off using RCA cleaning to create first level contact

8. Higher level metal formation

    - CMP (Chemical Mechanical Polishing) technique to planarize the surface
    - Create contact holes using photolithograhy process
    - Mask 12 is for first contact hole
    - Mask 13 is for first Aluminum contact layer
    - Mask 14 is for second contact hole
    - Mask 15 is for second Aluminum contact layer
    - Mask 16 is for making contact to topmost layer



Lab introduction to Sky130 basic layers layout and LEF using inverter



In sky130A, the first layer is local-interconnect layer or local-i and then the m1, m2 and so on. Power and Ground lines are in m1. When polysilicon crosses ndiffusion the it is NMOS and if polysilicon crosses pdiffusion then it is PMOS is created. The output of the layout is the LEF file. It is used by the router in APR to get the location of standard cell pins to route them properly. So it is basically the abstract form of layout of a standard cell.

Commands in tkcon window for spice extraction of the custom inverter layout:

1. `extract all`
2. `ext2spice cthresh 0 rthresh 0` --> this extracts the parasitic information
3. `ext2spice`

Sky130 Tech File Labs

1. Lab steps to create final SPICE deck using Sky130 tech

   The default SPICE deck file using Sky130 is as seen in the previous section. Now we modify the file to plot a transient response. The final SPICE deck file is as below.

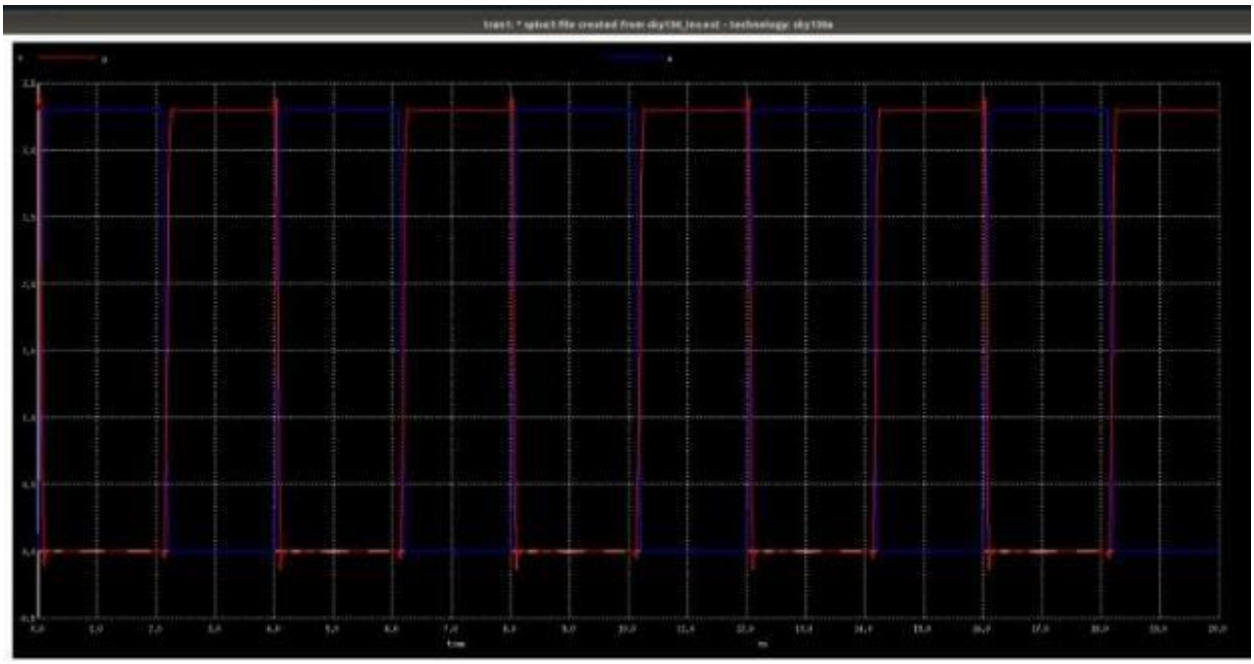
   Command to load spice file for simulation in ngspice:

   ```
   ngspice sky130A_inv.spice
   ```

   Generate a graph using:

   ```
   plot y vs time a
   ```

2. Lab steps to characterize inverter using sky130 model files



Using the above transient plot, we will now characterize the slew rate and propagation delay:

Maximum voltage = 3.3V 80% of maximum voltage = 2.64V 20% of maximum voltage = 0.64V 50% of maximum voltage = 1.65V

- Rise Transition (output transition time from 20% to 80%):
    - Tr_r = 2.20278ns - 2.15946ns = 0.04332ns

- Fall Transition (output transition time from 80% to 20%):
    - Tr_f = 4.06818ns - 4.04073ns = 0.02745ns

- Rise Delay (delay between 50% of input and 50% of output) that is time taken for output to rise to 50% and time taken for input to fall to 50%:
    - D_r = 2.18381ns - 2.15003ns = 0.03378ns

- Fall Delay (delay between 50% of input and 50% of output) that is time taken for output to fall to 50% and time taken for input to rise to 50%:
    - D_f = 4.05402ns - 4.0501ns = 0.00392ns

[Magic Tutorial]

3. Lab introduction to Sky130 pdk's and steps to download labs

   Commands to download and view the corrupted skywater process magic tech file and other files to perform drc corrections:

   - Command to download the lab files: `wget http://opencircuitdesign.com/open_pdks/archive/drc_tests.tgz`
   - Extract it: `tar xfz drc_tests.tgz`
   - Change directory into the lab folder: `cd drc/drc_tests`
   - List all files: `ls -al`
   - Command to open magic tool: `magic -d XR`

4. Lab introduction to Magic and steps to load Sky130 tech-rules

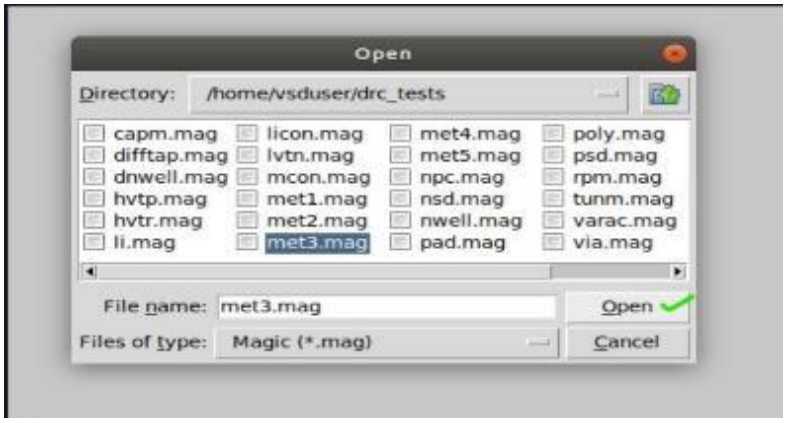
   Useful websites:

   Magic Technology File Format Manual - This site explains about tech files. All technology specific information comes from a technology file. This file includes information as layer types, electrical connectivity, design rules, rules for mask generation, rules for extracting netlists etc.

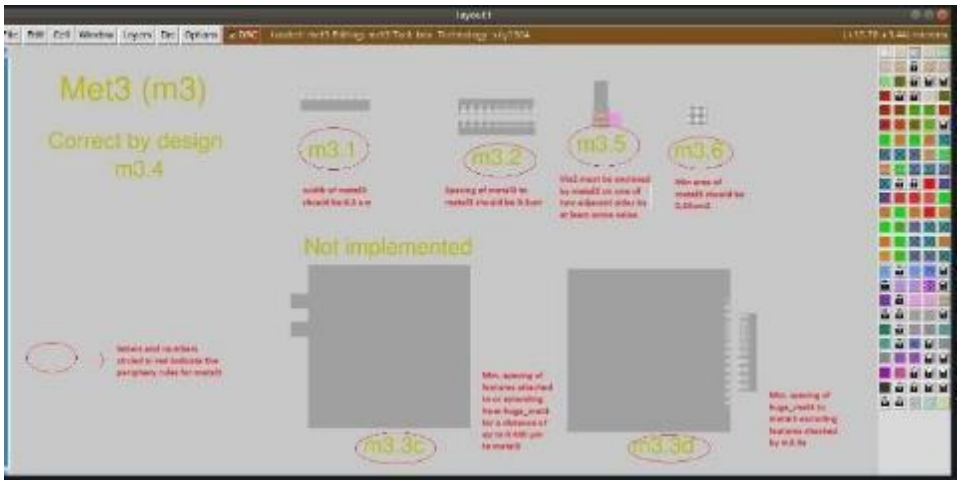
   Rules for SkyWater SKY130 PDK

   Steps:

   - Open magic with met3.mag as input

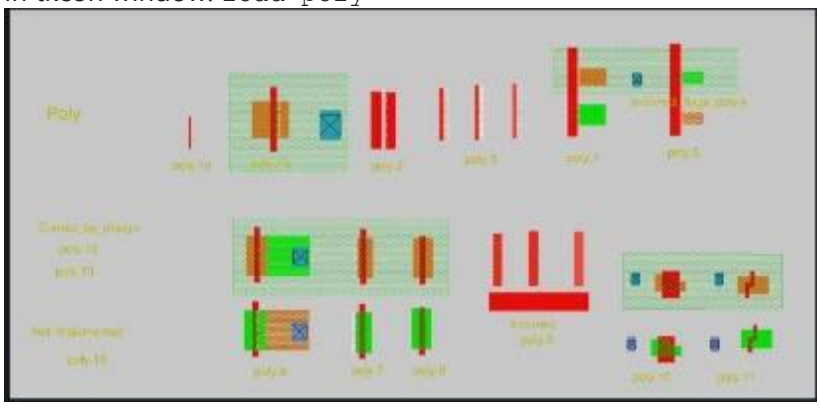


- In this view, we see a number of independent layouts containing some DRC errors

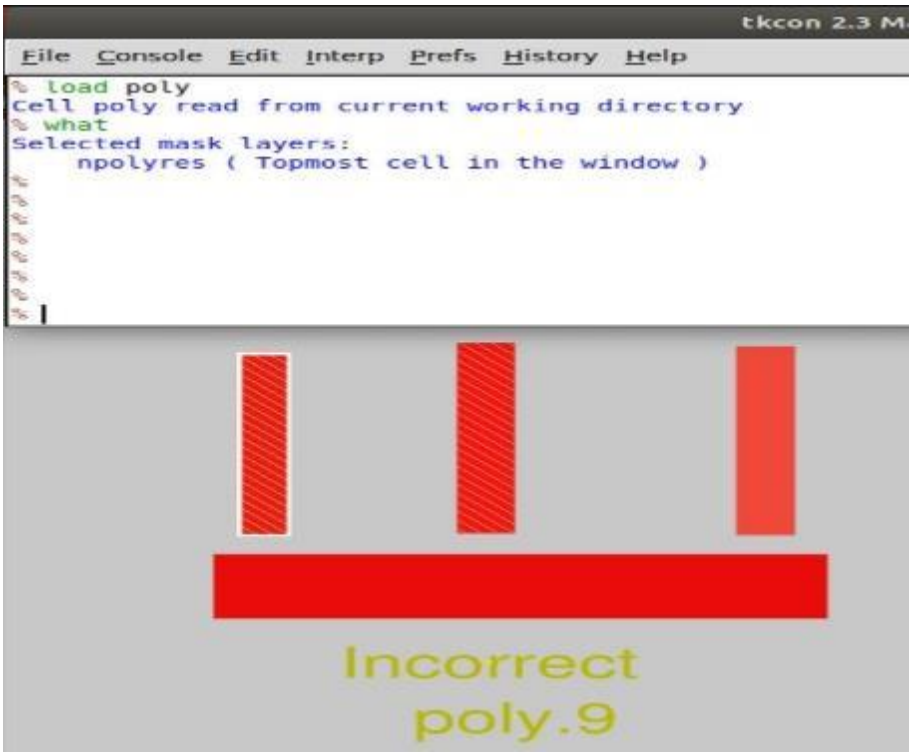


5. Lab exercise to fix poly.9 error in Sky130 tech-file

   - In tkcon window: `load poly`



   - Let's look at rule poly.9 As described in [Rules for SkyWater SKY130 PDK], Poly resistor spacing to poly or spacing (no overlap) to diff/tap should be atleast 0.48um.



That's not the case here, so we have to fix the tech file to include this DRC.

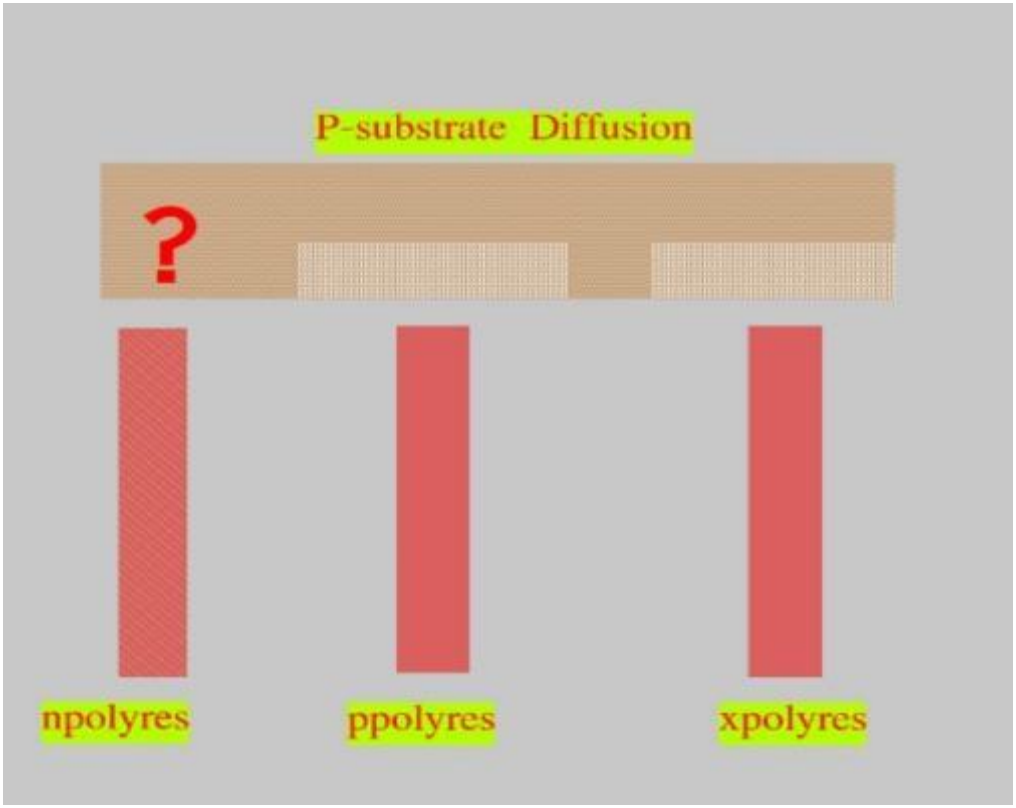
- Open sky130A.tech file in drc_tests directory. The included rules for poly.9 are only for the spacing between the n-poly resistor with n-diffusion and the spacing between the p-poly resistor with diffusion. We will now add new rules for the spacing between the poly resistor with poly non-resistor. Highlighted in green below are the two newly added rules. First one is the rule for the spacing between the p-poly resistor with poly non-resistor and the next one is the rule for spacing between n-poly resistor with poly non-resistor. The allpolynonres is a macro under alias section of techfile.

```
#------------------------------------------------
# uhrpoly (P+ poly resistor, 2kOhm/sq)
#------------------------------------------------

  width uhrpoly 350 "uhrpoly resistor width < %d"
  spacing xhrpoly,uhrpoly,xpc alldiff 480 touching_illegal \
        "xhrpoly/uhrpoly resistor spacing to diffusion < %d (poly.9)"

  spacing xhrpoly,uhrpoly,xpc allpolynonres 480 touching_illegal \
        "xhrpoly/uhrpoly resistor spacing to diffusion < %d (poly.9)"
```

```
#-----------------------------
# POLY
#-----------------------------

width allpoly 150 "poly.width < %d (poly.1a)"
spacing allpoly allpoly 210 touching_ok "poly.spacing < %d (poly.2)"
spacing allpolynonfet alldifflvnonfet 75 corner_ok allfets \
       "poly.spacing to Diffusion < %d (poly.4a)"
spacing npres *nsd 480 touching_illegal \
       "poly.resistor spacing to N-tap < %d (poly.9)"
spacing npres allpolynonres 480 touching_illegal \
       "poly.resistor spacing to N-tap < %d (poly.9)"
overhang *ndiff,rndiff nfet,scnfet,npd,npass 250 "N-Diffusion overhang of nmos < %d (poly.7)"
overhang *mvndiff,mvrndiff mvnfet,mvnnfet 250 \
       "N-Diffusion overhang of nmos < %d (poly.7)"
overhang *pdiff,rpdiff pfet,scpfet,ppu 250 "P-Diffusion overhang of pmos < %d (poly.7)"
overhang *mvpdiff,mvrpdiff mvpfet 250 "P-Diffusion overhang of pmos < %d (poly.7)"
overhang *poly allfets 130 "poly.overhang of transistor < %d (poly.8)"
rect_only allfets "No bends in transistors (poly.11)"
rect_only xhrpoly,uhrpoly "No bends in poly resistors (poly.11)"
extend  xpc/a xhrpoly,uhrpoly 2160 \
       "poly.contact extends poly resistor by < %d (licon.1c + li.5)"
spacing xhrpoly,uhrpoly xhrpoly,uhrpoly 1240 touching_illegal \
       "Distance between precision resistors < %d (rpm.2 + 2 * rpm.3)"
```

- In tkcon window: tech load sky130A.tech to check drc in tkcon window: drc check

  The new DRC rules will now take effect.



6. Lab exercise to implement poly resistor spacing to diff and tap

   To fix what is hown in below pic, modify the tech file to include not only the spacing between npolyres with N-substrate diffusion in poly.9 but also between npolyres and all types of diffusion.

Pre-layout timing analysis and importance of good clock tree

Timing modelling using delay tables

Lab steps to convert grid info to track info

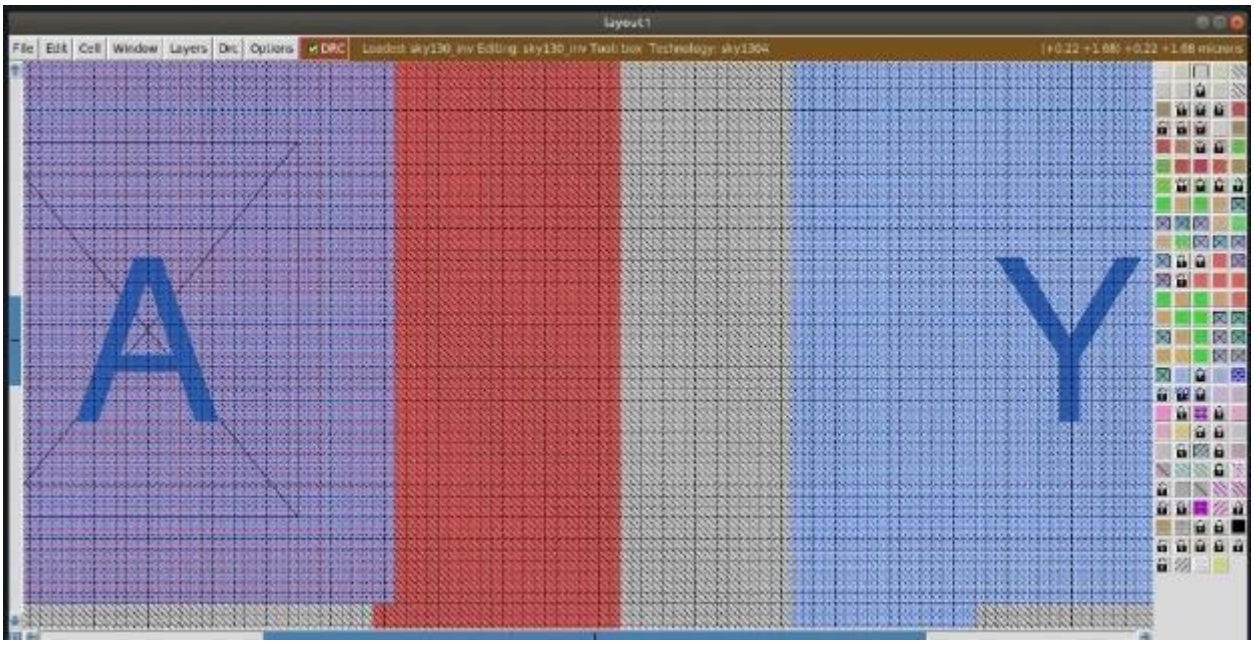
sky130_inv.mag contains all information like PG information, port information, logic etc. OpenLANE is a PnR tool and a PnR tool does not require all the information present in .mag file. The only information that we'll be needing is the boundary, power and ground rails, and the inputs & outputs. This is the reason of using .lef files. So the objective is to extract the LEF file from Magic file and plug into picorv32a design.

From PnR point of view, there are few guidelines to be followed while making standard cell,

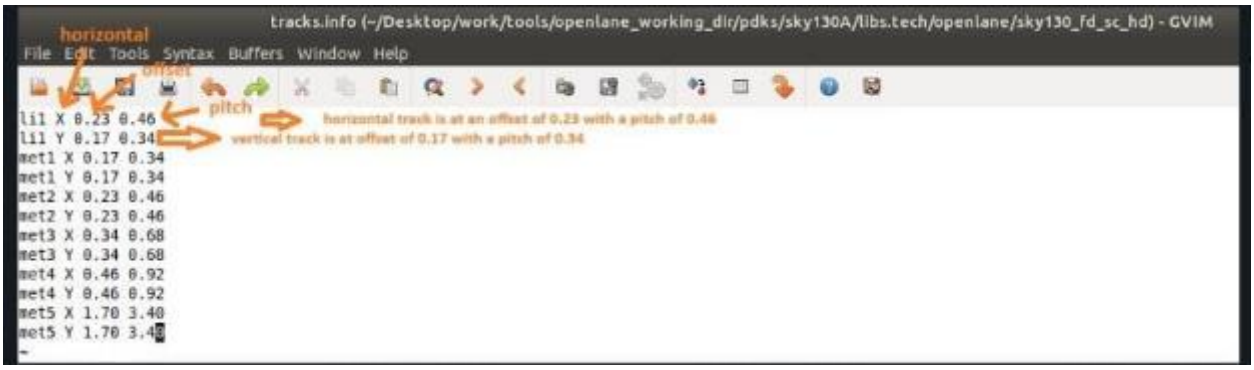
- The input and output ports lies at the intersection of the horizontal and vertical tracks (ensure the routes can reach that ports).
- The width of the standard cell must be odd multiple of the tracks horizontal pitch and height must be odd multiples of tracks vertical pitch

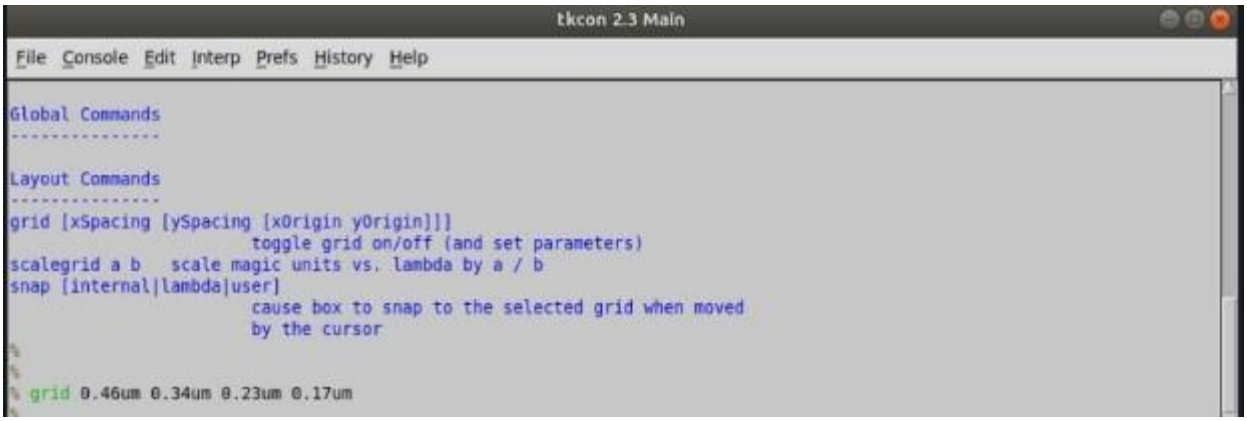
Tracks refer to the horizontal and vertical metal layers on which routing occurs. The grid formed by the intersection of horizontal and vertical tracks creates a routing grid, also known as a routing matrix. The ~/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.tech/openlane/sky130_fd_sc_hd/tracks.info contains track information.

Before changing grid:

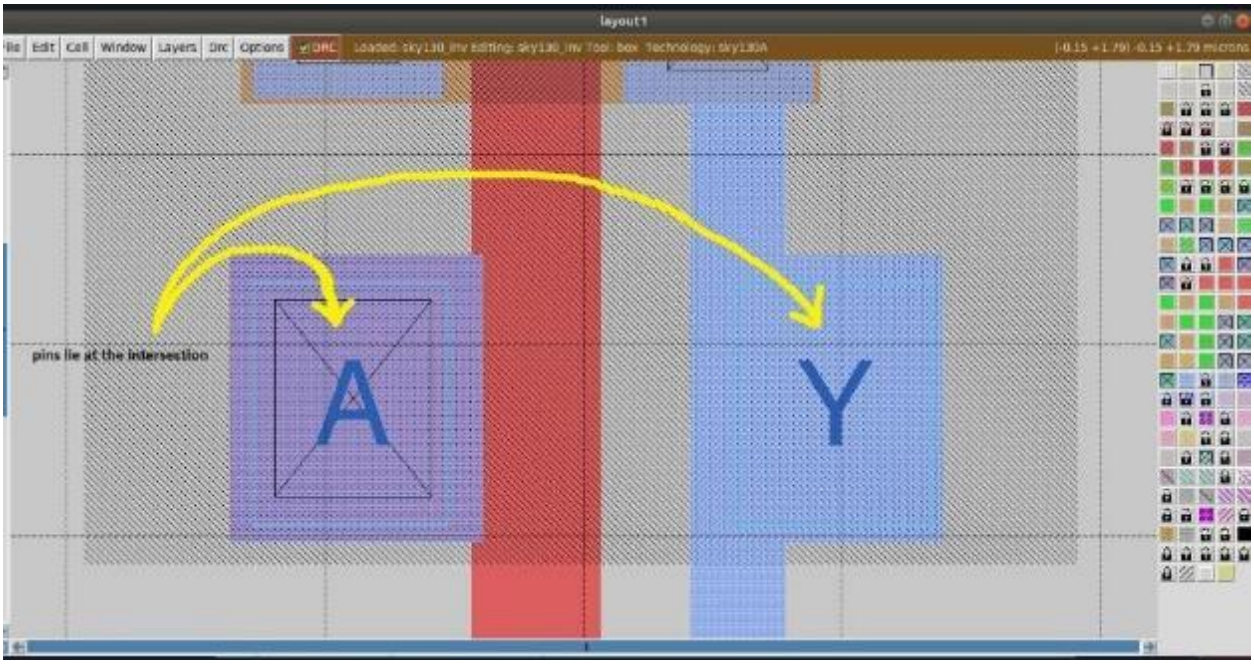


After changing grid values:

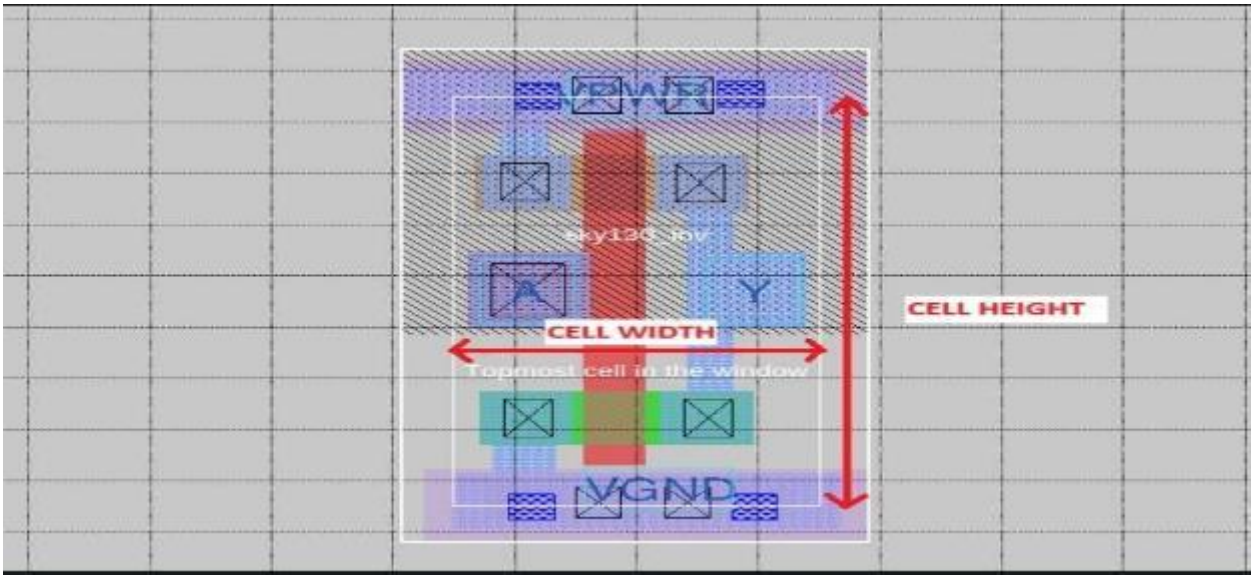




This satisfies the first guideline mentioned above.



Second requirement is satisfied in below picture.



Lab steps to convert magic layout to std cell LEF

LEF (Library Exchange Format) file is a standard file format used to describe the physical layout and characteristics of standard cell libraries or macro libraries. LEF files contain detailed information about the geometric shapes, sizes, layers, and other physical properties of individual cells or macros within the library. The instructions to set the port definitions are in this site

Next, save the .mag file with a new filename. In the tcon terminal: `lef write`
It will generate a LEF file with the new filename.

Introduction to timing libs and steps to include new cell in synthesis

Inside pdks/sky130A/libs.ref/sky130_fd_sc_hd/lib/ are the liberty timing files for SKY130 PDK which contains the timing and power parameters for each cell needed in STA. It can either be slow, typical, fast with different supply voltages (1v80, 1v65, 1v95). These are called PVT corners. The library name sky130_fd_sc_hd__ss_025C_1v80 describes the PVT corner as slow-slow (delay is maximum), 25° Celsius temperature, at 1.8V power supply. Timing and power parameter of a cell is obtained by simulating the cell in a variety of operating conditions (different corners) and these data are represented in the liberty file. The liberty file characterizes all cells and is used during ABC mapping during synthesis stage which maps the generic cells to the actual standard cells available in the liberty file.

1. Copy the extracted lef file sky130_vsdinv.lef and the liberty files sky130*.lib from

   /openlane/vsdstdcelldesign/libs to the src directory of picorv32a.

2. Add the folowing to config.tcl inside the picorv32a:

```
set ::env(LIB_SYNTH) "$::env(OPENLANE_ROOT)/designs/picorv32a/src/sky130_fd_sc_hd__typical.lib"
set ::env(LIB_MIN) "$::env(OPENLANE_ROOT)/designs/picorv32a/src/sky130_fd_sc_hd__fast.lib"
set ::env(LIB_MAX) "$::env(OPENLANE_ROOT)/designs/picorv32a/src/sky130_fd_sc_hd__slow.lib"
set ::env(LIB_TYPICAL) "$::env(OPENLANE_ROOT)/designs/picorv32a/src/sky130_fd_sc_hd__typical.lib"

set ::env(EXTRA_LEFS) [glob $::env(OPENLANE_ROOT)/designs/$::env(DESIGN_NAME)/src/*.lef]
```

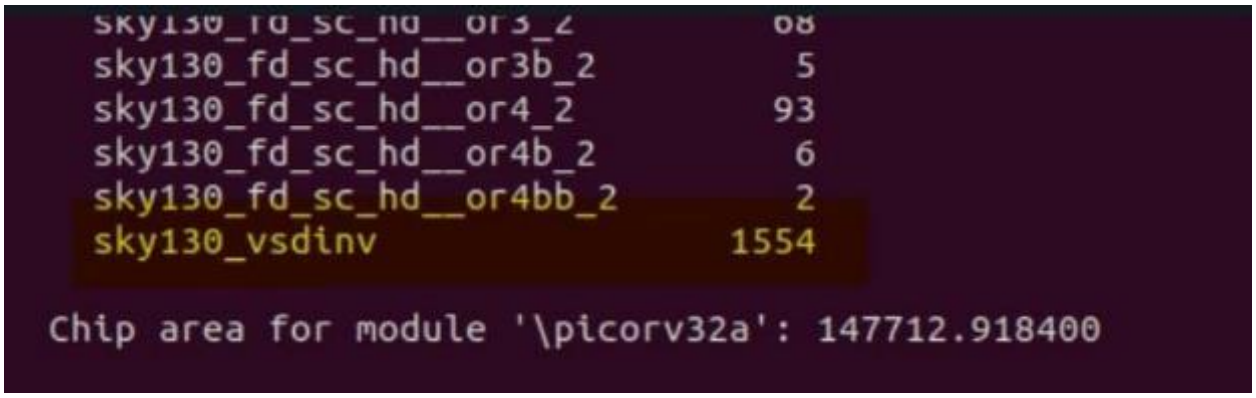
   This sets the liberty file that will be used for ABC mapping of synthesis (LIB_SYNTH) and for STA (_FASTEST,_SLOWEST,_TYPICAL) and also the extra LEF files (EXTRA_LEFS) for the customized inverter cell.

3. Run docker and prepare the design picorv32a. Plug the new lef file to the OpenLANE flow.

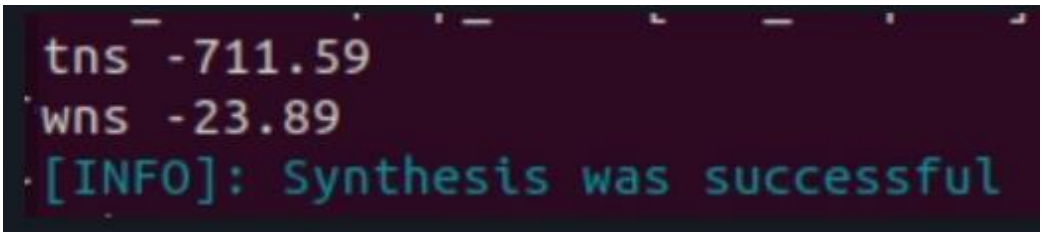
```
docker
./flow.tcl -interactive
package require openlane 0.9
prep -design picorv32a
```

```
set lefs [glob $::env(DESIGN_DIR)/src/*.lef]
add_lefs -src $lefs
```

4. Next run_synthesis. sky130_vsdinv cell is successfully included in the design



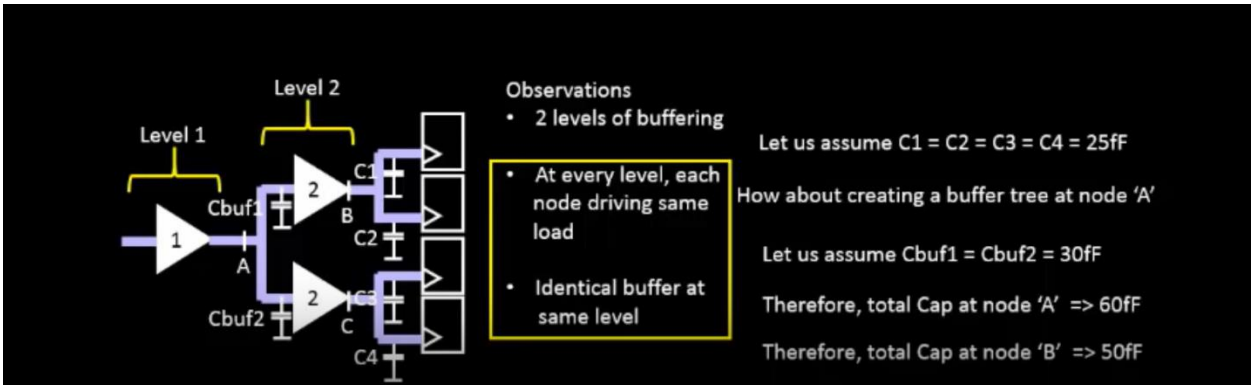
5. However timing is not met, so it has to be fixed.



Delay tables

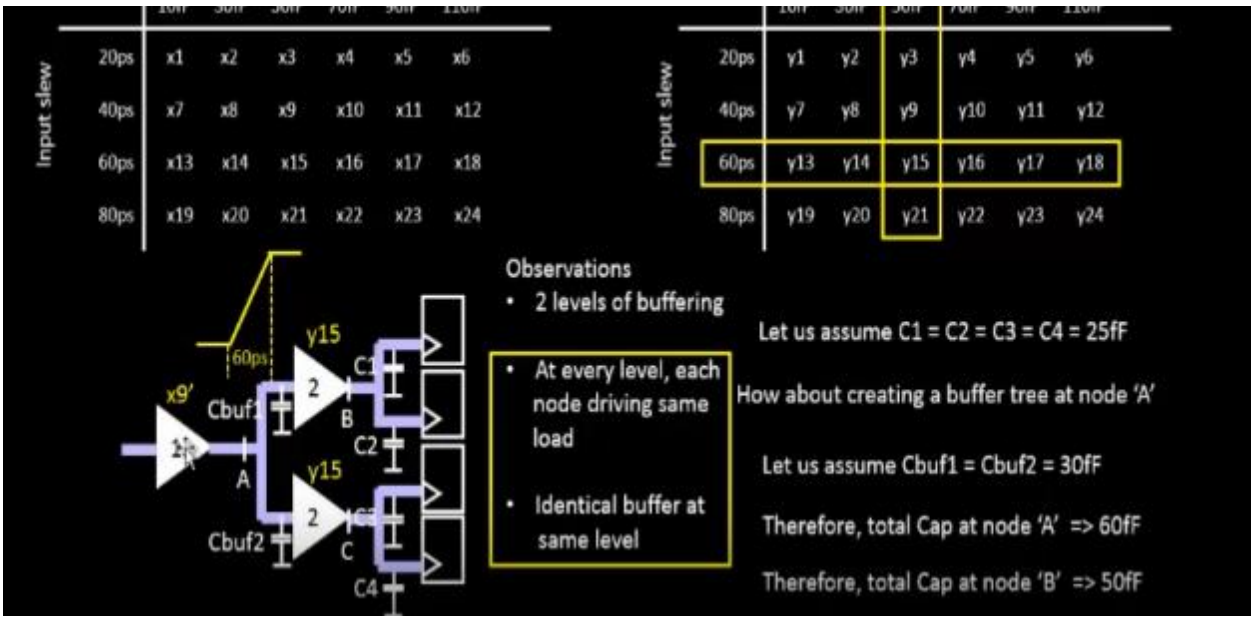
Whenever the enable pin is 1, only then the CLK will propogate to Y in case of AND gate and whenever the enable pin is 0, only then the CLK will propogate to Y in case of OR gate, as shown below. When the enable is 1, the CLK will not propagate and there won't be any short circuit power consumption and switching power consumption when such elements are used in clock tree. This method is referred to as the clock gating technique.

Consider the below clock tree structure.



Buffers on different levels have different capacitive loads and buffer sizes but as long as they have the same loads and sizes in the same level, the total delay for each clock tree path will be the same thus skew will remain zero. Practically, different levels can have varying input transition and output capacitive load and hence varying delay.

Delay tables are used to capture the timing model of each cell and is included inside the liberty file. The main factor in delay is the output slew. The output slew depends on capacitive load and input slew. The input slew is a function of previous stage buffer's output capacitive load and input slew and has its own transition delay table.



At level 2, both the buffers have identical delays with same transition times, load capacitances, and buffer sizes. Consequently, the skew is maintained at 0.If this is not the case, then the skew will be negative leading to timing violations. While these considerations may seem insignificant when

analyzing the delay of just two buffers, their significance is high in designs featuring millions of cells. Failing to adhere to these guidelines during clock tree creation can lead to numerous timing-related complications.

Terminologies:

- CTS is the process of designing a clock distribution network to minimize skew and ensure synchronous operation of the circuit
- Skew refers to the variation in clock signal arrival times
- Latency is the delay experienced by the clock signal
- Slew rate is the rate of change of the signal's voltage over time

Lab steps to configure synthesis settings to fix slack

Currently, tns = -711.59 wns = -23.89 Chip area for module picorv32a = 147712.9184

Next step is to see if synthesis can be more timing-driven.

1. Check synthesis strategy and other timing related variables and modify accordingly

   SYNTH_STRATEGY of delay 0 means the tool will focus more on optimizing the delay, index can be 0, 1, 2, or 3 where 3 is the most optimized for timing at the cost of area. SYNTH_BUFFERING of 1 ensures buffer will be used on high fanout cells to reduce wire delay. SYNTH_SIZING of 1 will enable cell sizing where cell will be upsized or downsized as needed to meet timing. SYNTH_DRIVING_CELL is the cell used to drive the input ports and is vital for cells with a lot of fan-outs since it needs higher drive strength.

2. Run synthesis again and it is seen that area is increased but there is no negative slack

   tns = 0 wns = 0 Chip area for module picorv32a = 209181.872

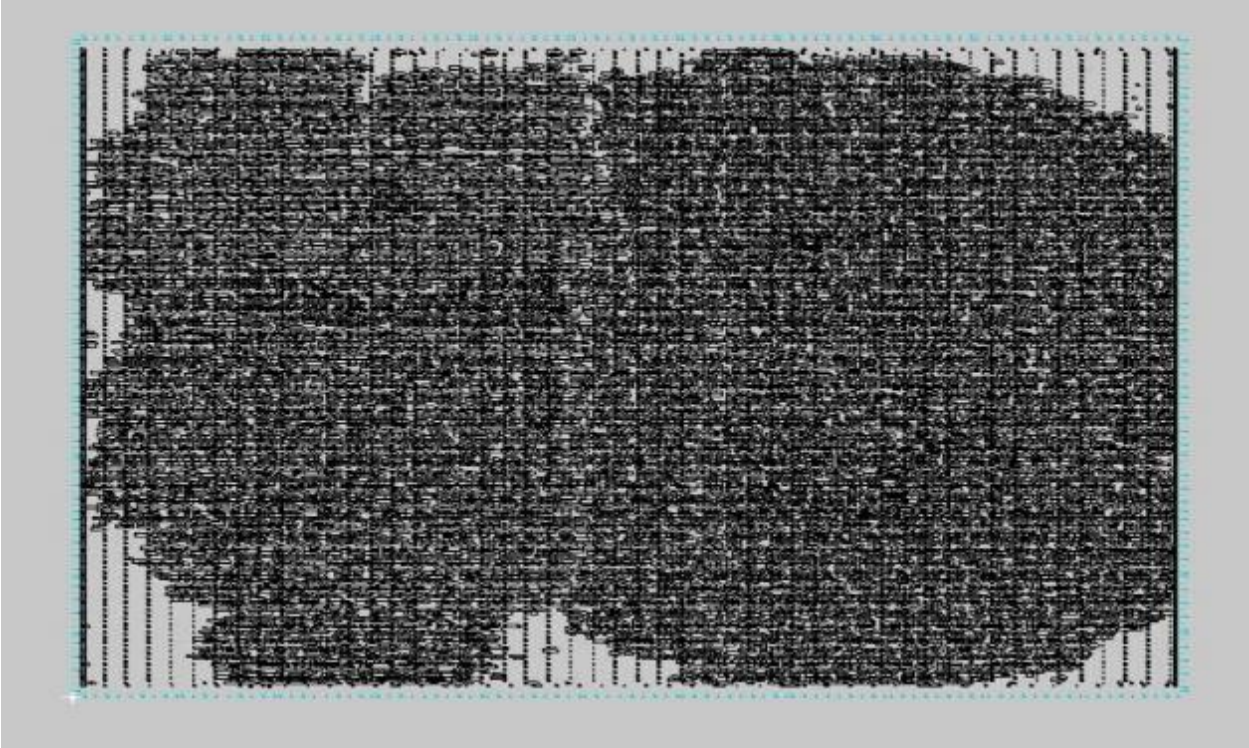
3. Run floorplan and placement

   If any error comes related to macro placement, temporary solution is to comment basic_macro_placement inside the OpenLane/scripts/tcl_commands/floorplan.tcl (this is okay since we are not adding any macro to the design).

   (or)

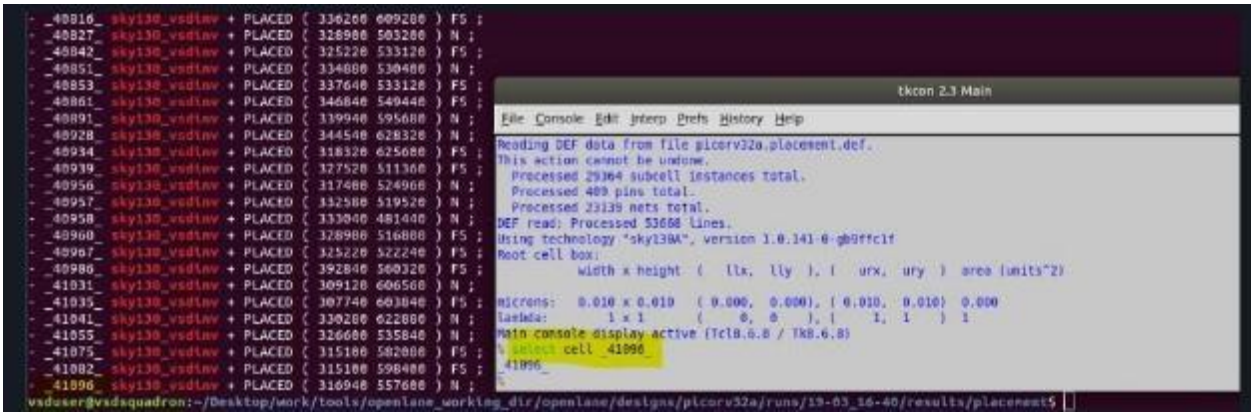
```
init_floorplan
place_io
global_placement_or
detailed_placement
tap_decap_or
detailed_placement
```

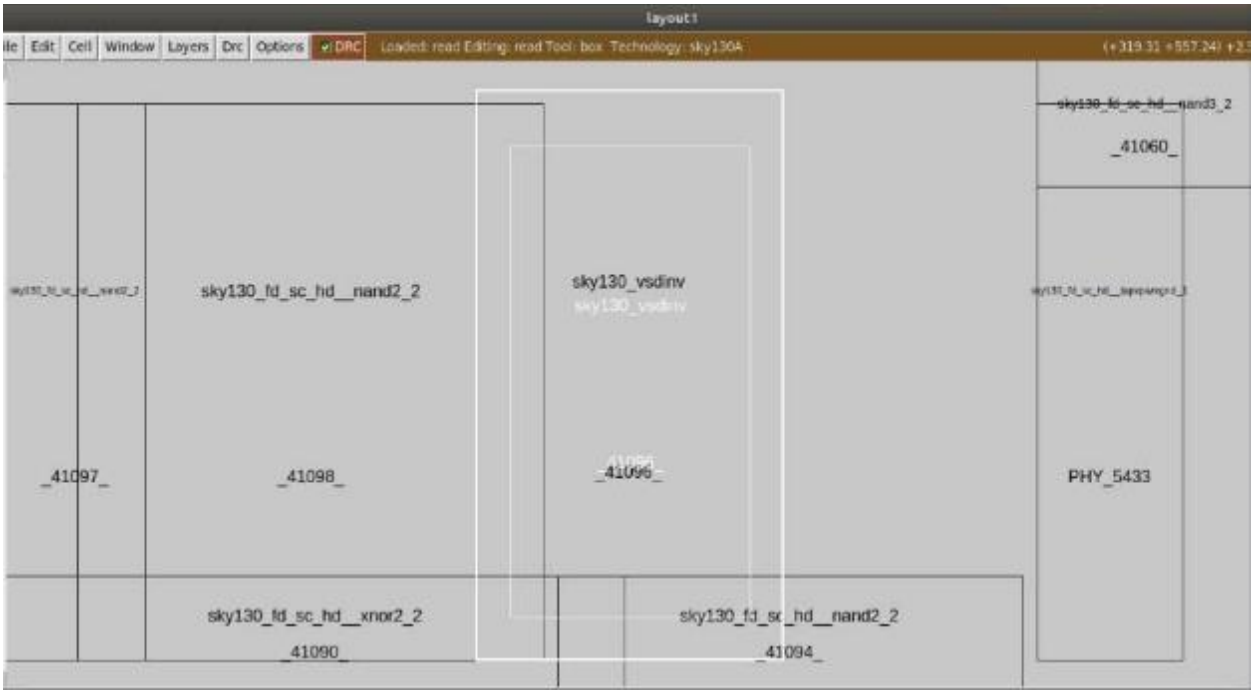
After successful run, runs/[date]/results/placement/picorv32a.placement.def will be created.

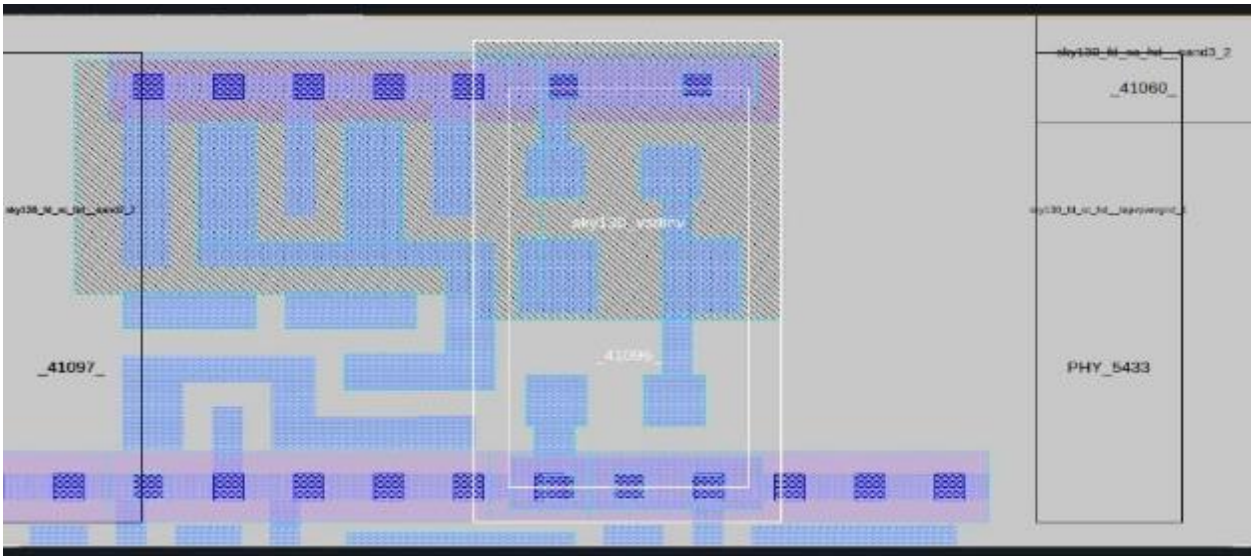


Search for instance of cell sky130_vsdinv inside the DEF file after placement stage: cat picorv32a.placement.def | grep sky130_vsdinv

Select a single sky130_vsdinv cell instance from the list dumped by grep (e.g. *41096*). On tkcon, command % select cell *41096* then ctrl+z to zoom into that cell. As shown below, our customized inverter cell sky130_myinverter is sucessfully placed. Use expand on tkon to show the footprint of the cell and notice how the power and ground of sky130_vsdinv overlaps the power and ground pins of its adjacent cells.


tkcon 2.3 Main
File Console Edit Interp Prefs History Help
Reading DEF data from file picorv32a.placement.def.
This action cannot be undone.
Processed 29364 subcell instances total.
Processed 489 pins total.
Processed 23139 nets total.
DEF read: Processed 53668 lines.
Using technology "sky130A", version 1.0.141-0-gb9ffc1f
Root cell box:
width x height (llx, lly), (urx, ury) area (units^2)
Microns: 0.010 x 0.010 (0.000, 0.000), (0.010, 0.010) 0.000
Lambda: 1 x 1 (0, 0), (1, 1) 1
Main console display active (Tcl8.6.8 / Tk8.6.8)
% select cell _41096_
_41096_
%
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/19-03_16-40/results/placement$

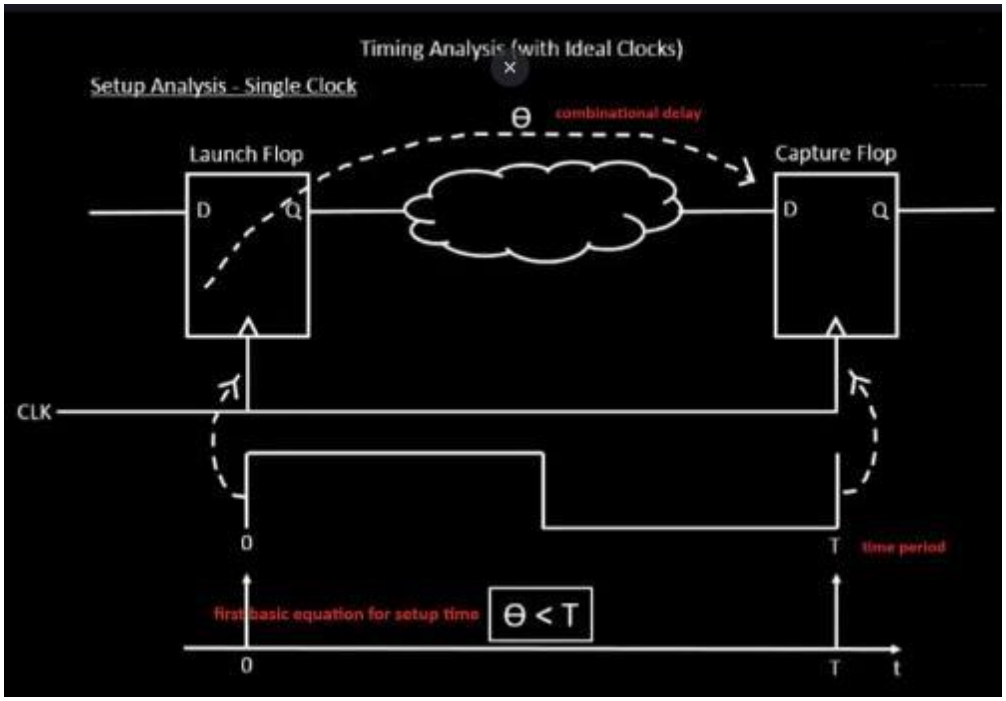

layout1
File Edit Cell Window Layers Drc Options DRC
Loaded: read Editing: read Tool: box Technology: sky130A
sky130_fd_sc_hd__nand3_2
_41060_
sky130_fd_sc_hd__nand2_2
sky130_vsdinv
_41097_
_41098_
_41096_
PHY_5433
sky130_fd_sc_hd__xnor2_2
_41090_
sky130_fd_sc_hd__nand2_2
_41094_

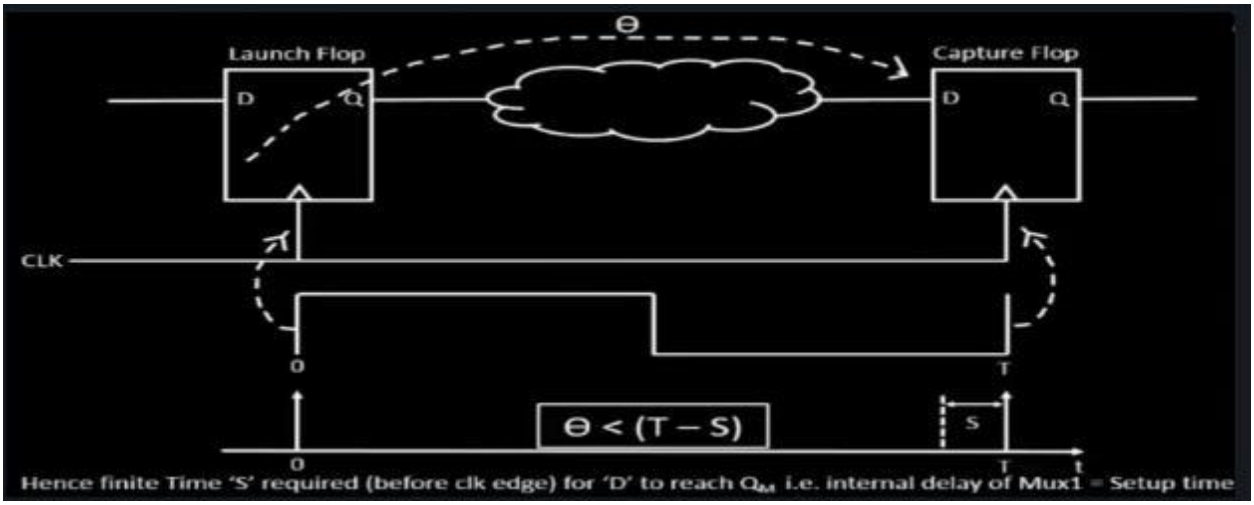

sky130_fd_sc_hd__nand3_2
_41060_
sky130_vsdinv
_41097_
PHY_5433

Timing analysis with ideal clocks using openSTA

Setup timing analysis and introduction to flip-flop setup time

Consider an ideal clock where clock tree is not built and perform timing analysis to understand the parameters. Later the same can be done using real clocks. Specifications are as mentioned in the picture. Clock frequncy (F) is 1GHz and clock period (T) is 1ns.





Setup timing analysis equation is:

$\Theta < T - S$

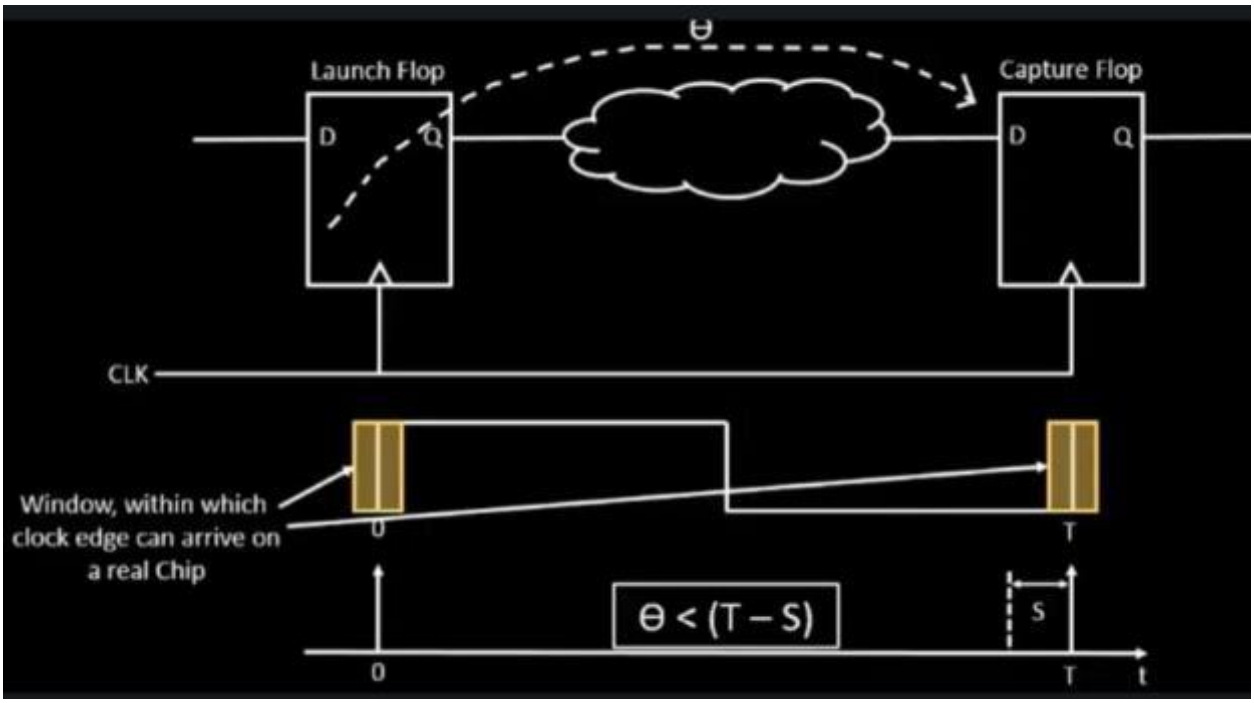
Θ = Combinational delay which includes clk to Q delay of launch flop and internal propagation delay of all gates between launch and capture flop

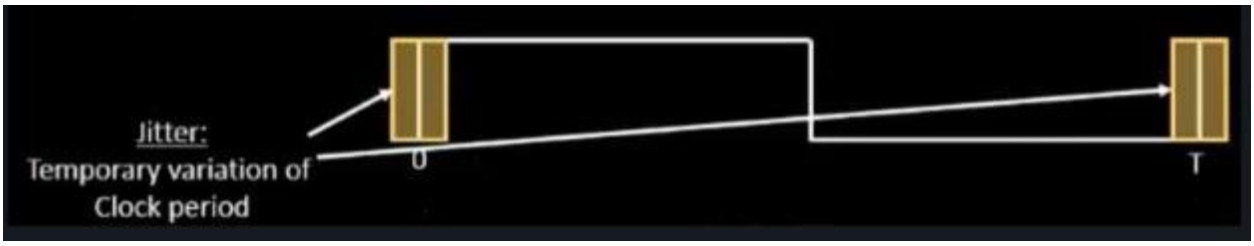
T = Time period, also called the required time

S = Setup time. As demonstrated below, signal must settle on the middle (input of Mux 2) before clock tansists to 1 so the delay due to Mux 1 must be considered, this delay is the setup time.

Introduction to clock jitter and uncertainty

Clock is beign created by PLL (Phase Locked Loop). So, this clock source is expected to send clock signal at 0, T, 2T etc. Even these clock sources might or might not be able to provide a clock exactly at Tns because of its own in-built variation. That is called as the jitter. Jitter can be manifested as short-term fluctuations in the timing of signal transitions, resulting in deviations from the expected clock or data timing.

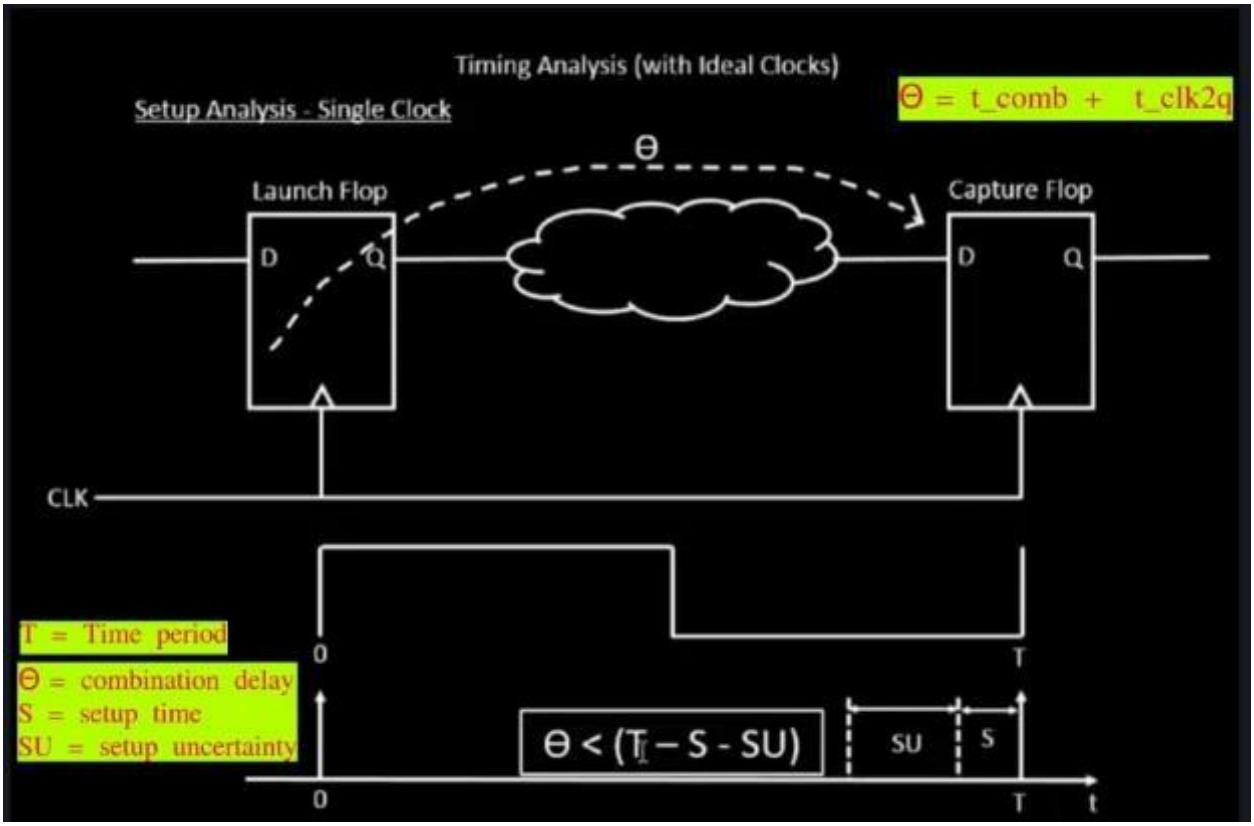




So, a more realistic equation for setup time is,

$\Theta < T - S - SU$

SU = Setup uncertainty due to jitter which is temporary variation of clock period. This is due to non-idealities of PLL/clock source.



Lab steps to configure OpenSTA for post-synth timing analysis

STA can either be single corner which only uses the LIB_TYPICAL library which is the one used in pre-layout(pos-synthesis) STA or multicorner which uses LIB_SLOWEST(setup analysis, high temp low voltage),LIB_FASTEST(hold analysis, low temp high voltage), and LIB_TYPICAL libraries.

`create_clock` command creates clock for the port with specified time period.
`set_input_delay` and `set_output_delay`defines the arrival/exit time of an input/output signal relative to the input clock. This is the delay of the signal coming from an external block and internal delay of the signal to be propagated to external ports. This adds a delay of Xns relative to clk to all signals going to input ports, and delay of Yns relative to clk to all signals going to output ports.
`set_max_fanout` specifies maximum fanout count for all output ports in the design.
`set_driving_cell` models an external driver at the input port of the current design.
`set_load` sets a capacitive load to all output ports.

3. Execute `sta pre_sta.conf` and check timing.

Lab steps to optimize synthesis to reduce setup violations

To reduce negative slack, focus on large delays. Net with big fanout might cause delay increase. Use `report_net -connections <net_name>` to display connections. First thing we can do is to go back to OpenLane and reduce fanouts by setting ::env(SYNTH_MAX_FANOUT) 4 then run_synthesis again.
To further reduce the negative slack, we can also try upsizing the cell with high fanout so that bigger driver will be used. High fanout results in high load cap which then results in high delay. Since we cannot change the load capacitance, we can change the cell size to drive large cap load for less delay.

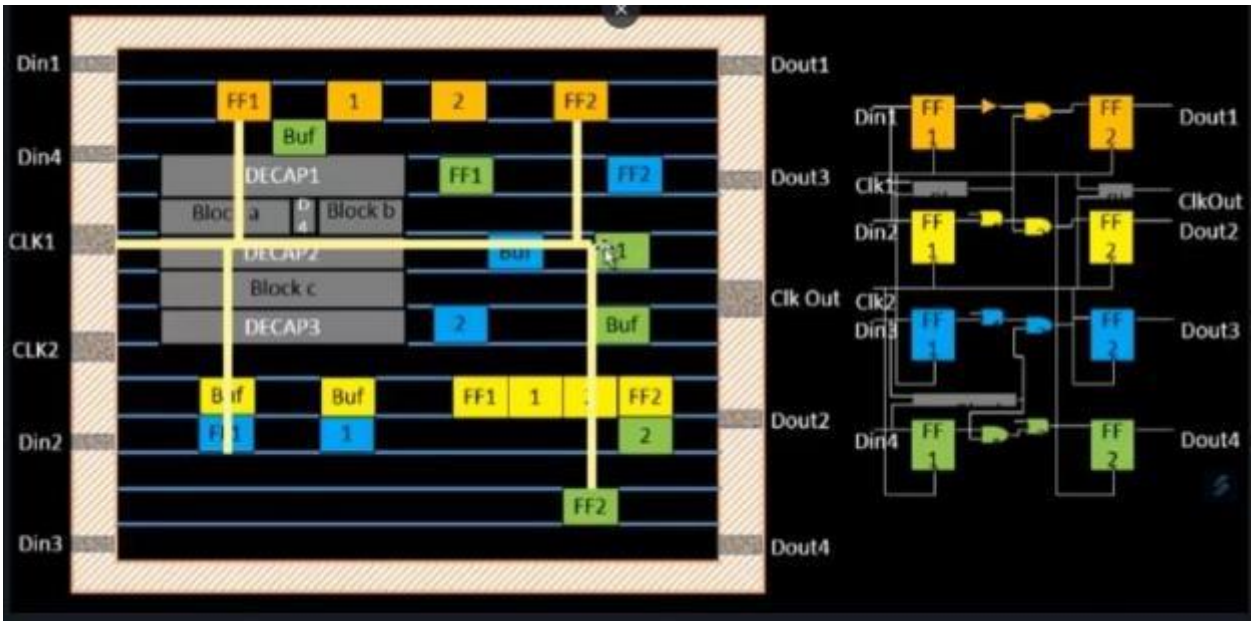
This can be done iteratively until the desired slack is reached, and this is called Timing ECO (Engineering Change Order).

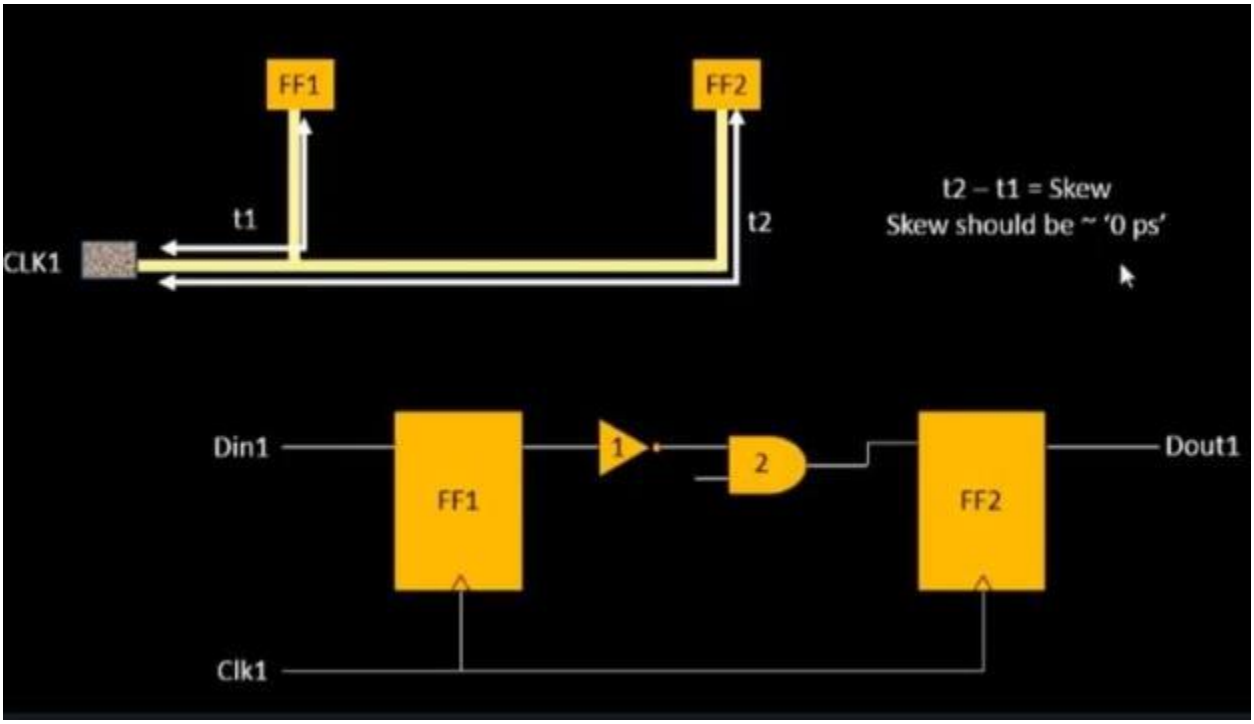
Clock Tree Synthesis TritonCTS and signal integrity

Clock tree routing and buffering using H-Tree algorithm

Consider the clock port that goes to the flip-flops highlighted in the picture. The purpose is to connect the port to the clock pins of the flip-flops based on the connectivity information. If we blindly connect as shown in the picture below, then `t2>t1` and the difference `t2-t1` is nothing but the skew. Clock skew refers to the variation in arrival times of the clock signal at different points within a synchronous digital system. In simpler terms, it is the difference in propagation delay experienced by the clock signal as it travels along different paths within the system. Clock skew can occur due to various factors such as differences in wire lengths, variations in signal routing paths,

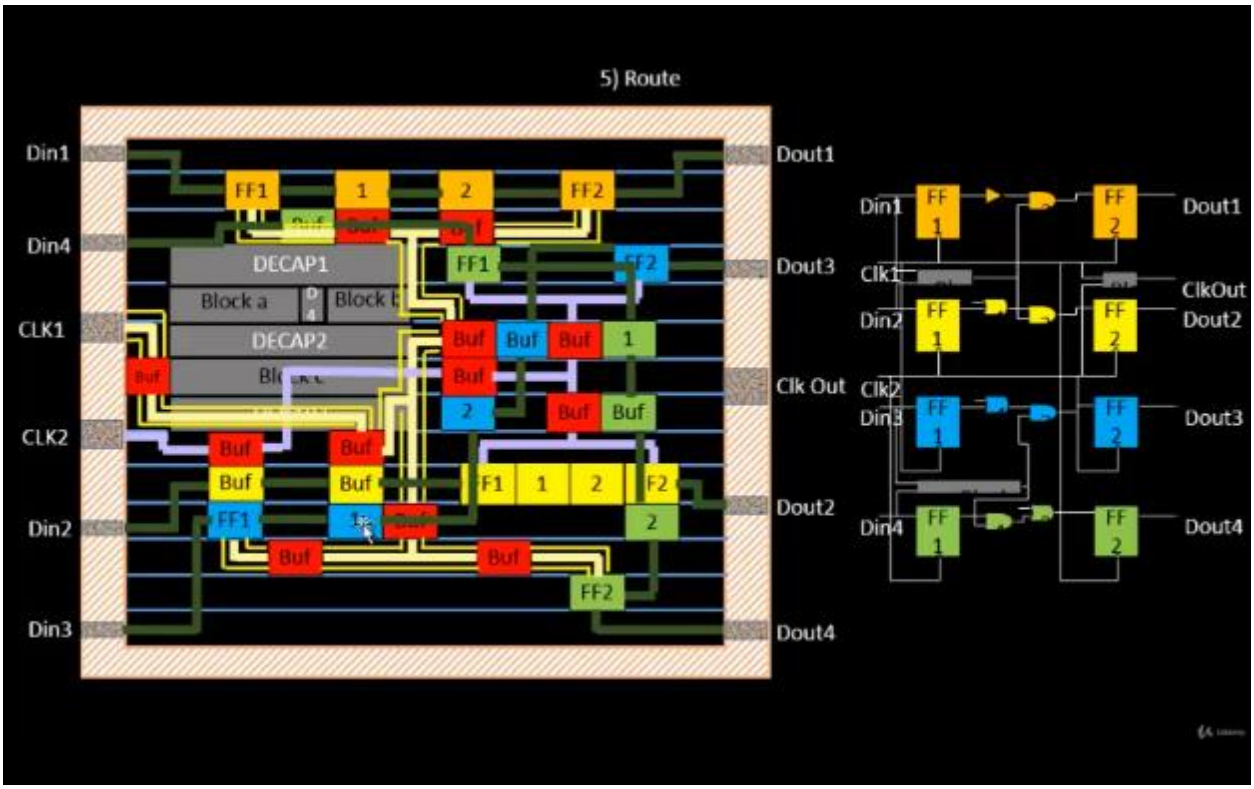
variations in buffer delays, and other physical and environmental factors. These variations can lead to some parts of the system receiving the clock signal earlier or later than others. Minimizing clock skew is essential to ensure proper synchronization of signals and reliable operation of the digital circuit. Ideally, the skew should be zero.





In the above scenario, the skew is not less/zero, so it is a bad tree.

H-Tree is the solution. It analyses the clock route by calculating the distance from the source to all the endpoints and deciding on a midpoint to start building tree from that point. In this case, the clock reaches at all the endpoints at almost the same time.



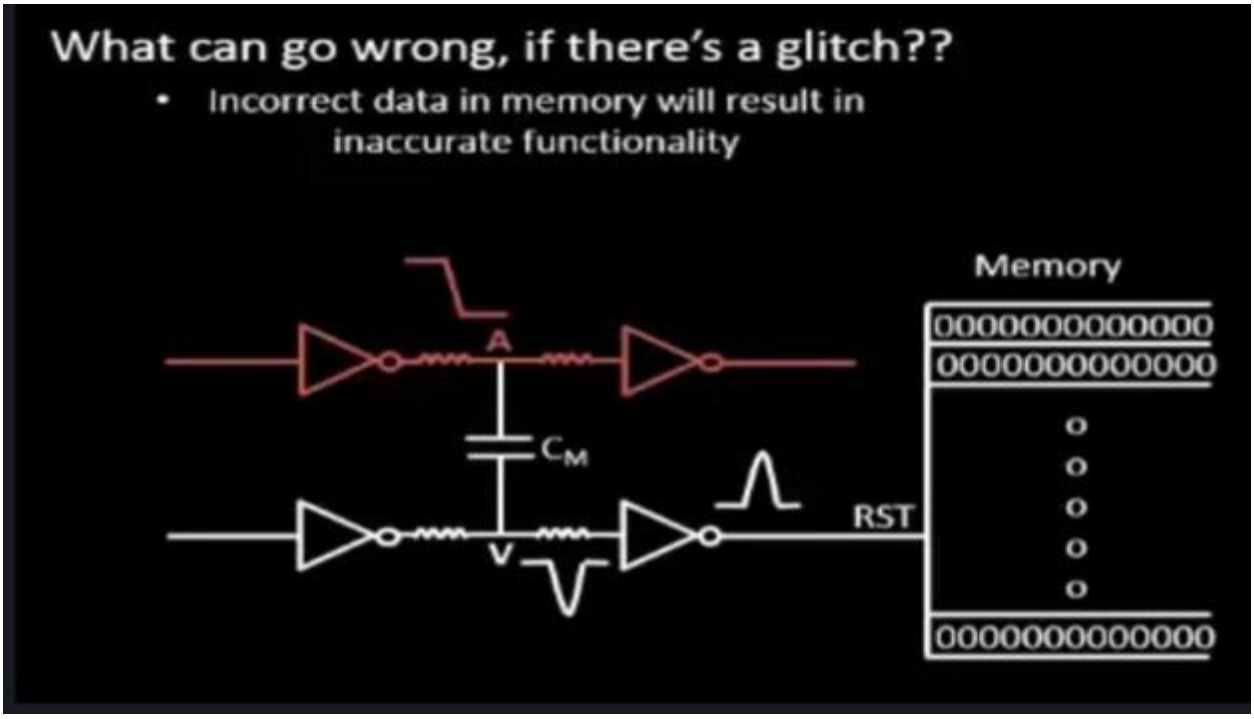
We expect that whatever input is provided, that should be reproduced at the output. However, due to the inherent resistance and capacitance in physical wires, the signal may experience attenuation or distortion, hindering its proper transmission to the output. To address this, repeaters or buffers are inserted along the path to ensure signal integrity and reliable transmission.

The key difference between repeaters used in clock paths and those used in data paths lies in their rise and fall times. Clock buffers have same rise and fall times, ensuring uniform signal propagation throughout the clock distribution network. In contrast, data buffers exhibit varying rise and fall times, which may differ based on the characteristics of the data being transmitted and the components involved in processing it.

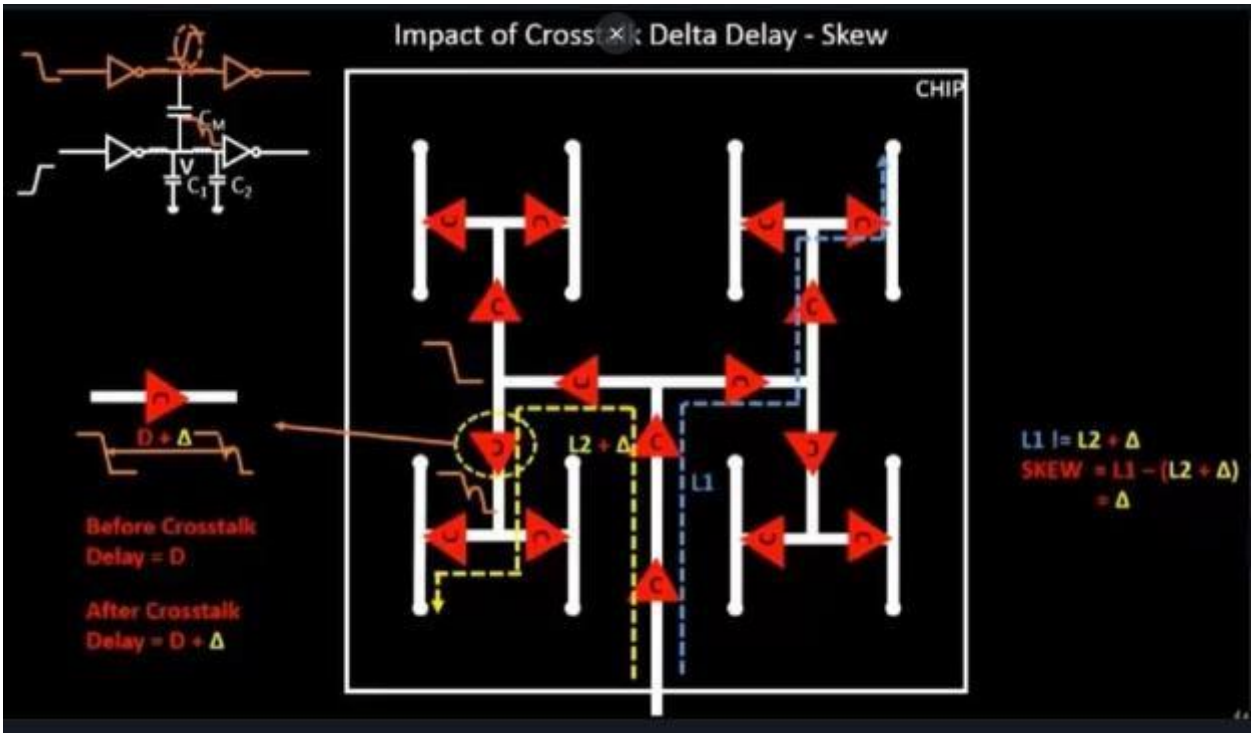
Crosstalk and clock net shielding

Clock nets are critical nets in the design because clock tree is built is such a fashion that the skew is zero. There is a phenomenon called crosstalk where a signal transmitted on one channel unintentionally interacts with or interferes with signals on adjacent channels leading to distortion, noise, timing errors etc. If this happens on clock routes, then the clock tree structure will be deteriorated. So all the clock nets are shielded. By shielding, the clock nets are protected. If there is a wire adjacent to such shields, then there exists a huge coupling capacitance cauign two issues. One is glitch and the other is delta delay.

Whenever there is a switching activity happening on the aggressor, the coupling capacitance is so strong that it directly affects the net sitting close to it called the victim net. the victim net is without any shielding. As a result, there is a dip in the voltage, resulting in glitch.



Shielding basically protects the victim nets by breaking the coupling capacitance between the aggressor and the victim. These shielding nets are either Vdd or Vss. The shields do not switch, so the victim will not switch.



Lab steps to run and verify CTS using TritonCTS

After ECO of cell sizing, currently the timing is as follows.

The slack might increase or decrease as we move forward in the PnR flow. For OpenLANE to use the current netlist,

`write_verilog filename` overwrites the current verilog file in the specified location.
Then,

```
run_floorplan
run_placement
```

Then run cts using the command `run_cts`. Before that we need to check the default setting s that CTS uses.

In CTS stage, clock buffers get added.

OpenLANE takes the procs
from `~/Desktop/work/tools/openlane_working_dir/openlane/scripts/tcl_commands`.
These procedures will then call OpenROAD to run the actual tool.

For example, run_cts can be found in the file `/OpenLane/scripts/tcl_commands/cts.tcl`, this tcl procedure will call OpenROAD and will call `/OpenLane/scripts/openroad/cts.tcl` which contains the OpenROAD commands to run TritonCTS.
Inside the `/OpenLane/scripts/openroad/cts.tcl` contains the configuration variables for CTS such as:
CTS_CLK_BUFFER_LIST = list of clock buffers used in clock tree branches (sky130_fd_sc_hd__clkbuf_1 sky130_fd_sc_hd__clkbuf_2 sky130_fd_sc_hd__clkbuf_4 sky130_fd_sc_hd__clkbuf_8)

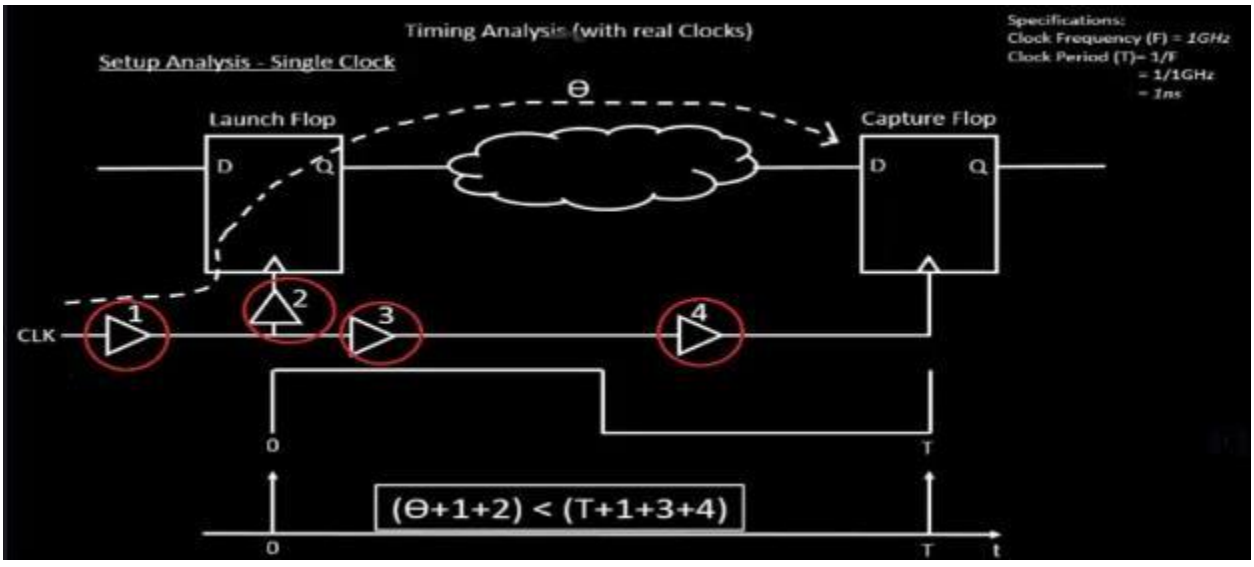
CTS_ROOT_BUFFER = clock buffer used for the root of the clock tree and is the biggest clock buffer to drive the clock tree of the whole chip (sky130_fd_sc_hd__clkbuf_16)

CTS_MAX_CAP = maximum capacitance of the output port of the root clock buffer

Timing analysis with real clocks using openSTA

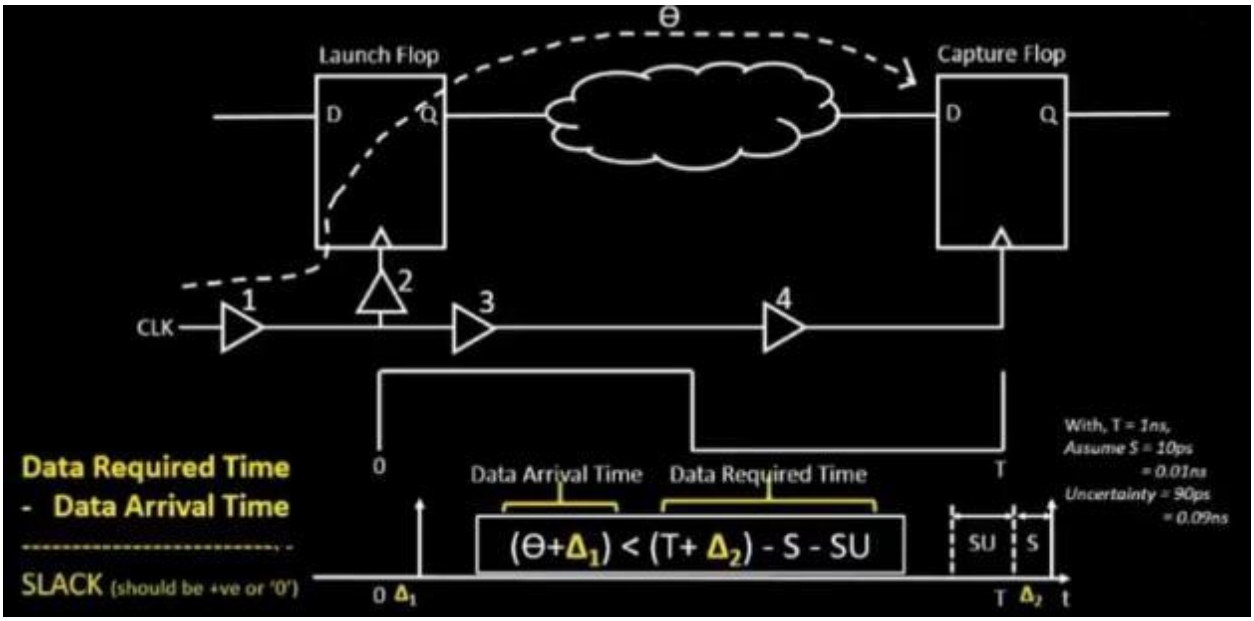
Setup timing analysis using real clocks

Now the clock tree is built and timing analysis is done on real clocks.



delta1 = launch flop clock network delay delta2 = capture flop clock delay

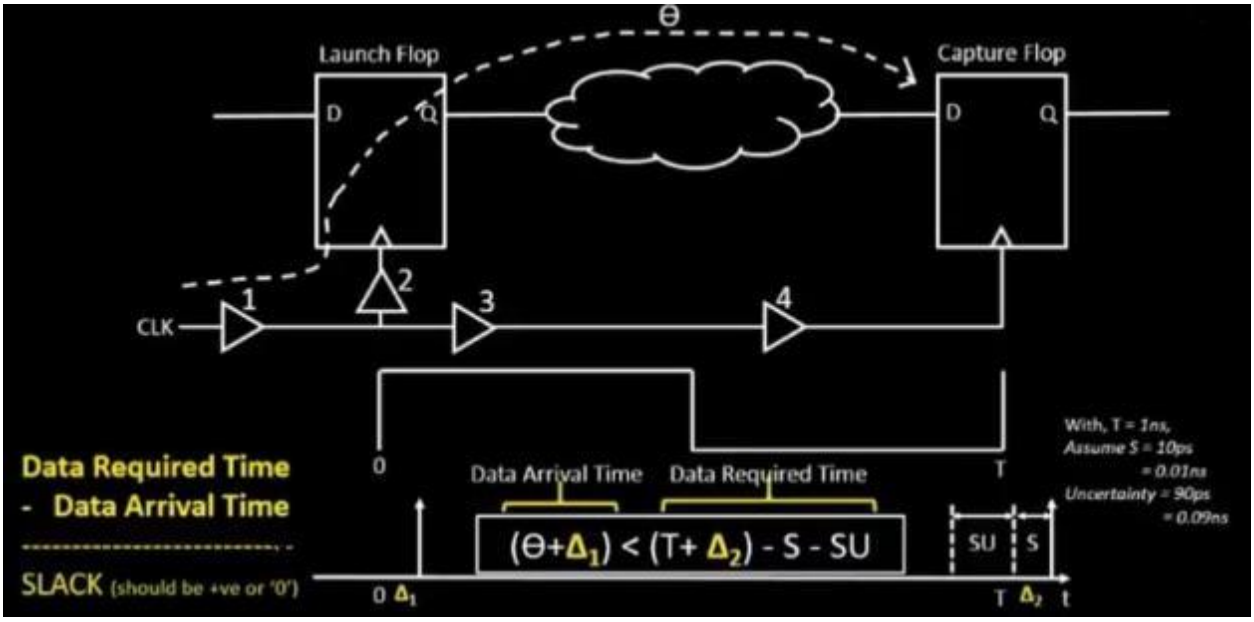
delta2=capture flop clock delay

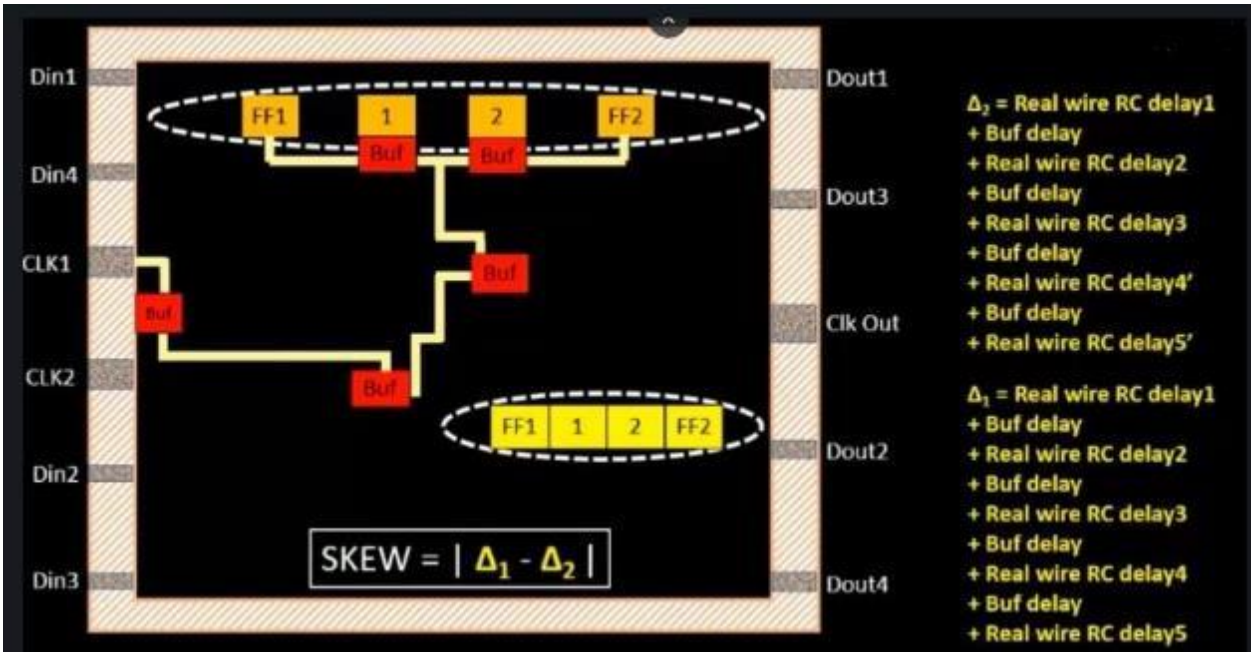


Any design satisfying `Slack = Data required time - Data arrival time` is ready to work in the given frequency. If this equation is violated , then slack will become negative. We expect slack to be 0 or positive.

Hold timing analysis using real clocks

Hold analysis refers to the delay/time required by the MUX2 model within the flip-flop to transfer data outside. It denotes the duration during which the launch flop must retain data before it reaches the capture flop. Unlike setup analysis, which spans two rising clock edges, hold analysis occurs on the same rising clock edge for both the launch and capture flops. A hold violation occurs when the path is too fast, impacted by factors including combinational delay, clock buffer delays, and hold time. Notably, parameters such as time period and setup uncertainty hold no significance, as both launch and capture flops receive identical rising clock edges during hold analysis.





Skew = Launch Clock Network Delay - Capture Clock Network Delay

Lab steps to analyze timing with real clocks using OpenSTA

The objective is to analyse the clock tree. Entering into `openroad` instead of invoking a separate OpenSTA tool. In openroad, timing analysis is done in a different way, where a db is created from lef & def and used.

1. To create the db, read lef

```
% read_lef designs/picorv32a/runs/19-03_16-40/tmp/merged.lef
Notice 0: Reading LEF file:  designs/picorv32a/runs/19-03_16-40/tmp/merged.lef
Notice 0:     Created 13 technology layers
Notice 0:     Created 25 technology vias
Notice 0:     Created 442 library cells
Notice 0: Finished LEF file:  designs/picorv32a/runs/19-03_16-40/tmp/merged.lef
%
```

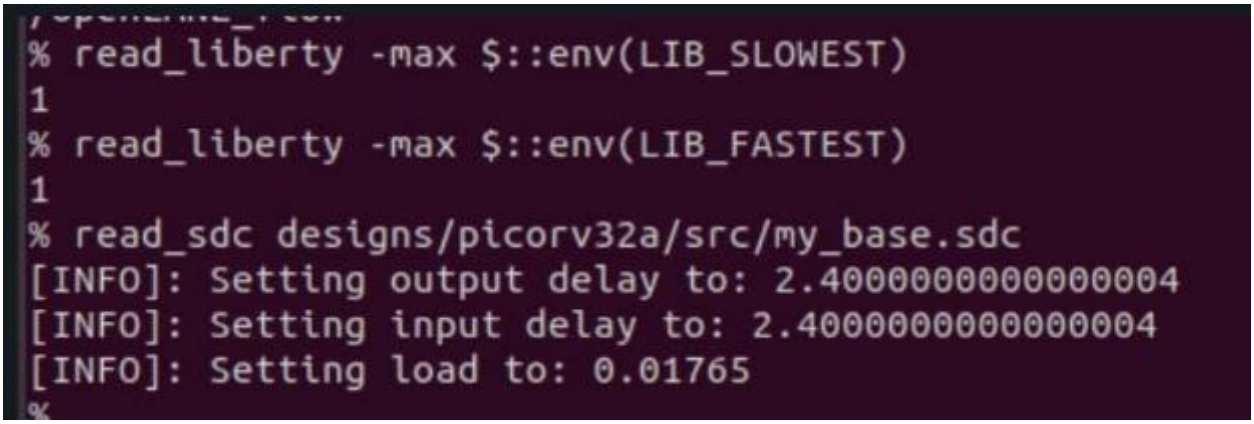
2. Read def from cts stage

```
% read_def designs/picorv32a/runs/19-03_16-40/results/cts/picorv32a.cts.def
Notice 0:
Reading DEF file: designs/picorv32a/runs/19-03_16-40/results/cts/picorv32a.cts.d
ef
Notice 0: Design: picorv32a
Notice 0:     Created 409 pins.
Notice 0:     Created 29675 components and 183125 component-terminals.
Notice 0:     Created 23450 nets and 80421 connections.
Notice 0: Finished DEF file: designs/picorv32a/runs/19-03_16-40/results/cts/pico
rv32a.cts.def
%
```

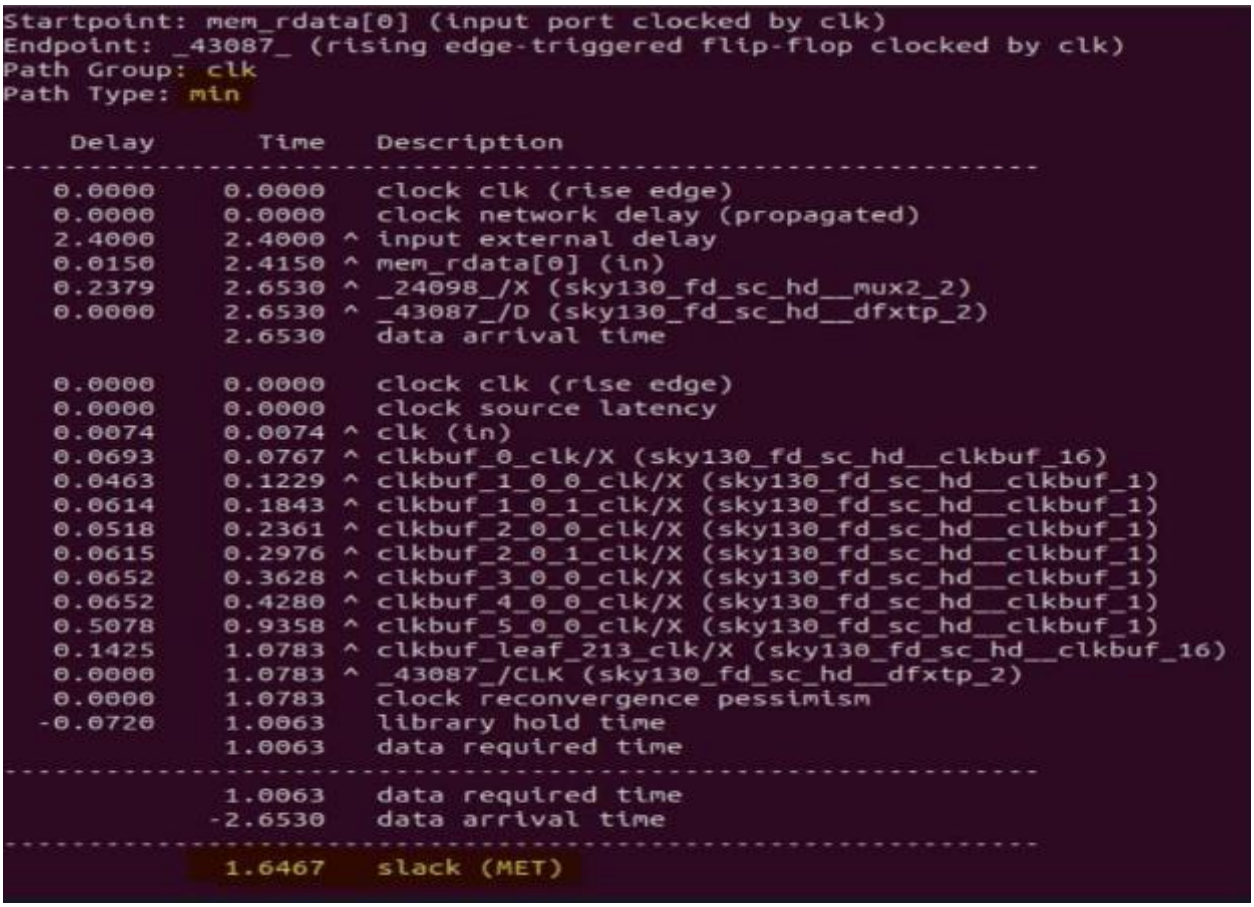
3. Create db

```
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane$ tree -L 1
.
├── AUTHORS.md
├── clean_runs.tcl
├── configuration
├── conf.py
├── CONTRIBUTING.md
├── default.cvcrc
├── designs
├── docker_build
├── docs
├── flow.tcl
├── LICENSE
├── Makefile
├── pico_cts.db
├── pre_sta.conf
├── README.md
├── regression_results
├── report_generation_wrapper.py
├── run_designs.py
├── scripts
└── vsdstdcelldesign
```

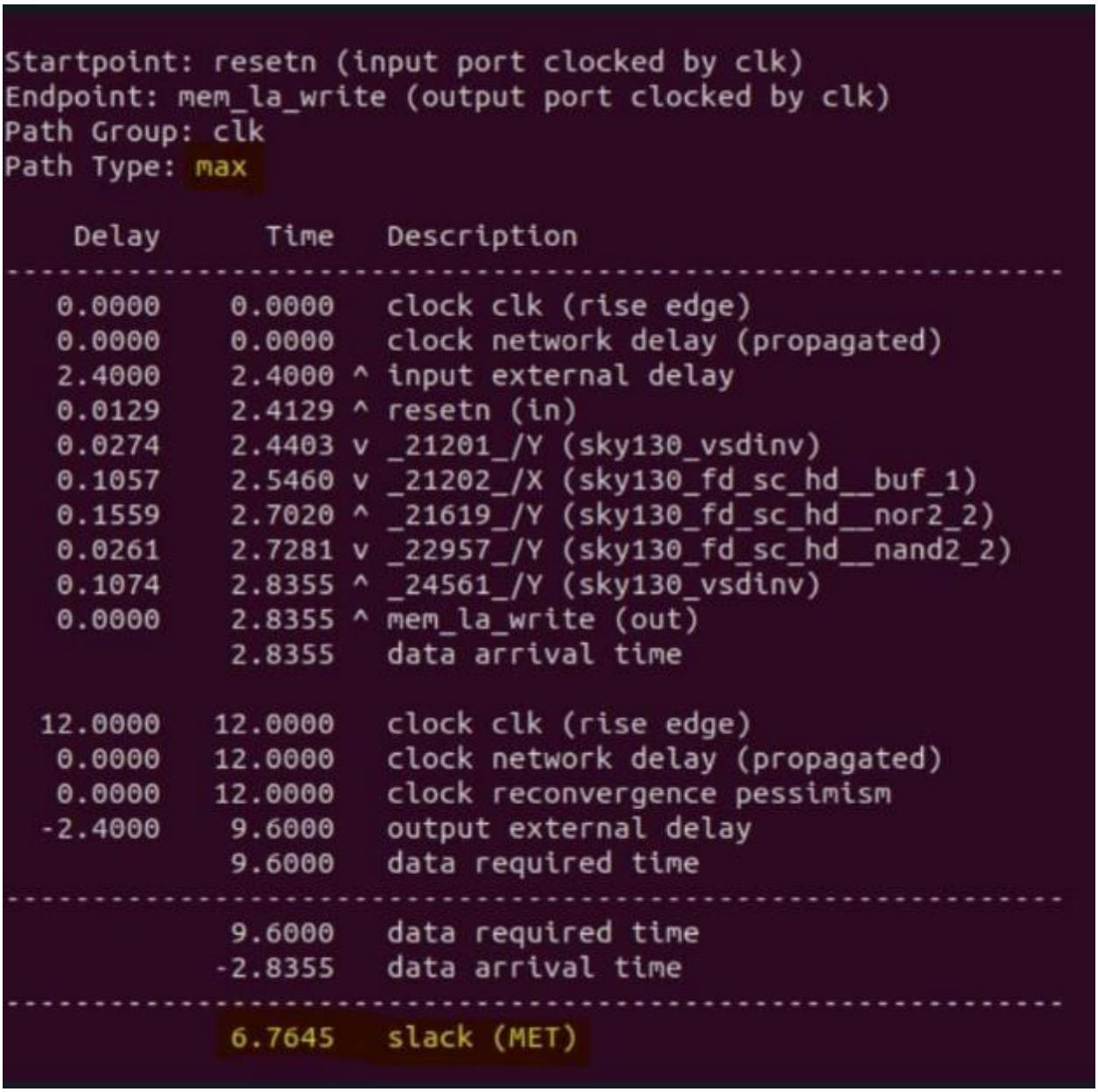
4. Read the db, verilog file, libraries, sdc



5. Set propagated clock, it will calculate the actual delay in the clock path.
6. Check timing



```
Startpoint: mem_rdata[0] (input port clocked by clk)
Endpoint: _43087_ (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: min

    Delay      Time   Description
---------------------------------------------------------
   0.0000    0.0000   clock clk (rise edge)
   0.0000    0.0000   clock network delay (propagated)
   2.4000    2.4000 ^ input external delay
   0.0150    2.4150 ^ mem_rdata[0] (in)
   0.2379    2.6530 ^ _24098_/X (sky130_fd_sc_hd__mux2_2)
   0.0000    2.6530 ^ _43087_/D (sky130_fd_sc_hd__dfxtp_2)
             2.6530   data arrival time

   0.0000    0.0000   clock clk (rise edge)
   0.0000    0.0000   clock source latency
   0.0074    0.0074 ^ clk (in)
   0.0693    0.0767 ^ clkbuf_0_clk/X (sky130_fd_sc_hd__clkbuf_16)
   0.0463    0.1229 ^ clkbuf_1_0_0_clk/X (sky130_fd_sc_hd__clkbuf_1)
   0.0614    0.1843 ^ clkbuf_1_0_1_clk/X (sky130_fd_sc_hd__clkbuf_1)
   0.0518    0.2361 ^ clkbuf_2_0_0_clk/X (sky130_fd_sc_hd__clkbuf_1)
   0.0615    0.2976 ^ clkbuf_2_0_1_clk/X (sky130_fd_sc_hd__clkbuf_1)
   0.0652    0.3628 ^ clkbuf_3_0_0_clk/X (sky130_fd_sc_hd__clkbuf_1)
   0.0652    0.4280 ^ clkbuf_4_0_0_clk/X (sky130_fd_sc_hd__clkbuf_1)
   0.5078    0.9358 ^ clkbuf_5_0_0_clk/X (sky130_fd_sc_hd__clkbuf_1)
   0.1425    1.0783 ^ clkbuf_leaf_213_clk/X (sky130_fd_sc_hd__clkbuf_16)
   0.0000    1.0783 ^ _43087_/CLK (sky130_fd_sc_hd__dfxtp_2)
   0.0000    1.0783   clock reconvergence pessimism
  -0.0720    1.0063   library hold time
             1.0063   data required time
---------------------------------------------------------
             1.0063   data required time
            -2.6530   data arrival time
---------------------------------------------------------
             1.6467   slack (MET)
```



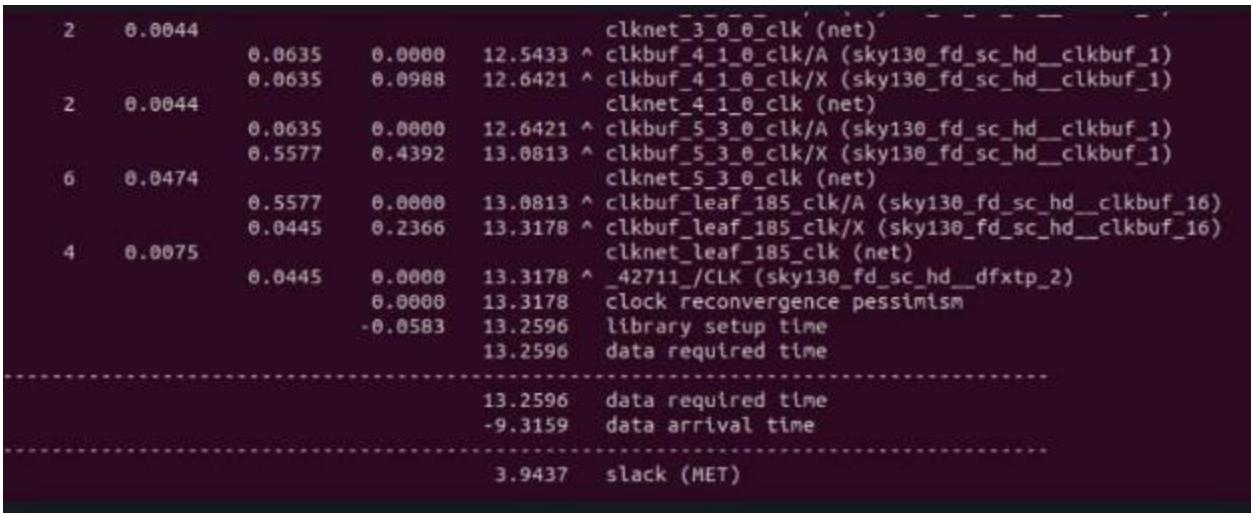
Lab steps to execute OpenSTA with right timing libraries

TritonCTS is built to optimise based on one corner but the libraries that are included in the previous section for timing analysis are min and max corners. This kind of analysis is not accurate. So, exit and re-enter openroad and check timing only for typical corner.

```
% exit
%
% openroad
OpenROAD 0.9.0 1415572a73
This program is licensed under the BSD-3 license. See the LICENSE file for details.
Components of this program may be licensed under more restrictive licenses which must be honored.
%
% read_db pico_cts.db
%
% read_verilog designs/picorv32a/runs/19-03_16-40/results/synthesis/picorv32a.synthesis_cts.v
%
% read_liberty $::env(LIB_SYNTH_COMPLETE)
1
%
% link_design picorv32a
[WARNING ORD-1000] LEF master sky130_fd_sc_hd__tapvpwrvgnd_1 has no liberty cell.
%
% read_sdc designs/picorv32a/src/my_base.sdc
[INFO]: Setting output delay to: 2.4000000000000004
[INFO]: Setting input delay to: 2.4000000000000004
[INFO]: Setting load to: 0.01765
%
% set_propagated_clock [all_clocks]
% _
```

In this typical scenario, slack is met in both setup and hold analysis.

```
                  0.0635    0.0000    0.6421 ^ clkbuf_5_21_0_clk/A (sky130_fd_sc_hd__clkbuf_1)
                  1.0103    0.7511    1.3932 ^ clkbuf_5_21_0_clk/X (sky130_fd_sc_hd__clkbuf_1)
    11    0.0868                              clknet_5_21_0_clk (net)
                  1.0103    0.0000    1.3932 ^ clkbuf_opt_14_clk/A (sky130_fd_sc_hd__clkbuf_16)
                  0.0563    0.2800    1.6732 ^ clkbuf_opt_14_clk/X (sky130_fd_sc_hd__clkbuf_16)
     1    0.0079                              clknet_opt_14_clk (net)
                  0.0563    0.0000    1.6732 ^ clkbuf_leaf_164_clk/A (sky130_fd_sc_hd__clkbuf_16)
                  0.0403    0.1266    1.7998 ^ clkbuf_leaf_164_clk/X (sky130_fd_sc_hd__clkbuf_16)
     9    0.0169                              clknet_leaf_164_clk (net)
                  0.0403    0.0000    1.7998 ^ _43431_/CLK (sky130_fd_sc_hd__dfxtp_2)
                            0.0000    1.7998   clock reconvergence pessimism
                           -0.0292    1.7707   library hold time
                                      1.7707   data required time
-----------------------------------------------------------------------------
                                      1.7707   data required time
                                     -1.8506   data arrival time
-----------------------------------------------------------------------------
                                      0.0799   slack (MET)
```



```
   2    0.0044                                clknet_3_0_0_clk (net)
             0.0635    0.0000   12.5433 ^ clkbuf_4_1_0_clk/A (sky130_fd_sc_hd__clkbuf_1)
             0.0635    0.0988   12.6421 ^ clkbuf_4_1_0_clk/X (sky130_fd_sc_hd__clkbuf_1)
   2    0.0044                                clknet_4_1_0_clk (net)
             0.0635    0.0000   12.6421 ^ clkbuf_5_3_0_clk/A (sky130_fd_sc_hd__clkbuf_1)
             0.5577    0.4392   13.0813 ^ clkbuf_5_3_0_clk/X (sky130_fd_sc_hd__clkbuf_1)
   6    0.0474                                clknet_5_3_0_clk (net)
             0.5577    0.0000   13.0813 ^ clkbuf_leaf_185_clk/A (sky130_fd_sc_hd__clkbuf_16)
             0.0445    0.2366   13.3178 ^ clkbuf_leaf_185_clk/X (sky130_fd_sc_hd__clkbuf_16)
   4    0.0075                                clknet_leaf_185_clk (net)
             0.0445    0.0000   13.3178 ^ _42711_/CLK (sky130_fd_sc_hd__dfxtp_2)
                       0.0000   13.3178   clock reconvergence pessimism
                      -0.0583   13.2596   library setup time
                                13.2596   data required time
-----------------------------------------------------------------------------
                                13.2596   data required time
                                -9.3159   data arrival time
-----------------------------------------------------------------------------
                                 3.9437   slack (MET)
```

When CTS is built, skew values is tried to be met by inserting buffers from the CTS_CLK_BUFFER_LIST. We can also modify this list based on requirements.

When TritonCTS is building the clock tree, it tries to use each buffer listed in `$::env(CTS_CLK_BUFFER_LIST) (sky130_fd_sc_hd__clkbuf_1 sky130_fd_sc_hd__clkbuf_2 sky130_fd_sc_hd__clkbuf_4 sky130_fd_sc_hd__clkbuf_8)` from smallest to largest until the target skew is met. Target skew is stored in `$::env(CTS_TARGET_SKEW)`. The STA result shows that sky130_fd_sc_hd__clkbuf_1 is the mostly used buffer, we can also change the `$::env(CTS_CLK_BUFFER_LIST)` to use other buffers and observe the effect on STA and area.
Use tcl `lreplace` command to modify `$::env(CTS_CLK_BUFFER_LIST)`

The `$::env(CURRENT_DEF)` used by CTS is the DEF file of the previously run CTS, but the DEF file we want for CTS is the placement's DEF file. So change the `$::env(CURRENT_DEF)` to point to placement DEF file then `run_cts`.
Observe the resulting post-CTS STA compared to previous run since we modified the clock buffer. Only buf_2 clock buffer is used now compared to buf_1 used in previous run. The WNS is better now since we used bigger clock buffers.

Final steps for RTL2GDS using tritonRoute and openSTA

Routing and design rule check (DRC)

Introduction to Maze Routing

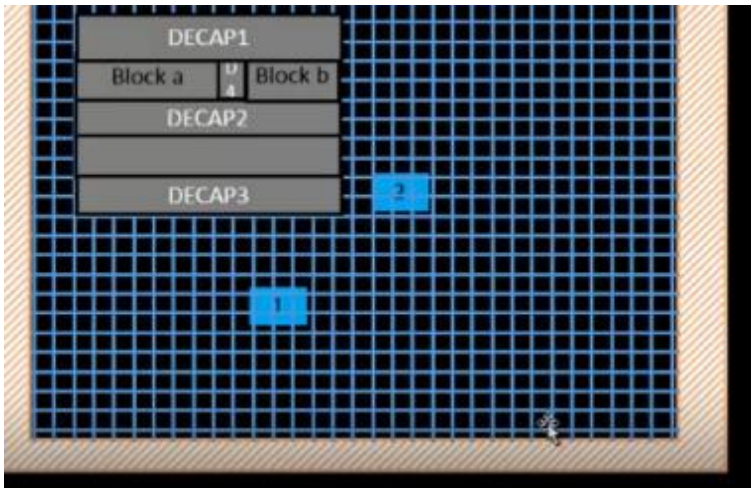
Routing is to find the best possible connection/route between two points. There are many routing algorithms like Steiner Tree algorithm, Line Search algorithm etc. and one such is Maze Routing - Lee's Algorithm (Lee 1961)

Consider and example of connecting two points 1 & 2. Point 1 will act as a source and 2 will act as a target. The requirement is to find the best possible path or the shortest possible path to connect 1 & 2 will less or no zig-zag routes. Mostly the routes are L-shaped. From algorithmic point of view, the software has to search and connect the two points. From physical designer point of view, it is a physical path/wire establishment for signals to travel between components.

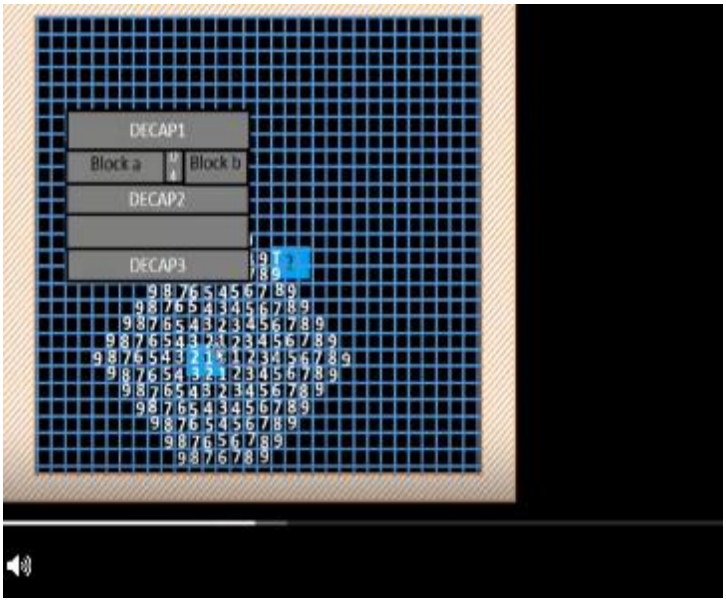
Lee's maze routing algorithm, is a popular pathfinding algorithm used in maze routing, which is a type of routing problem where the goal is to find a path from a source to a destination in a maze-like grid. The Lee algorithm is particularly well-suited for routing on grids or mesh-based structures in integrated circuit design.

Algorithm steps:

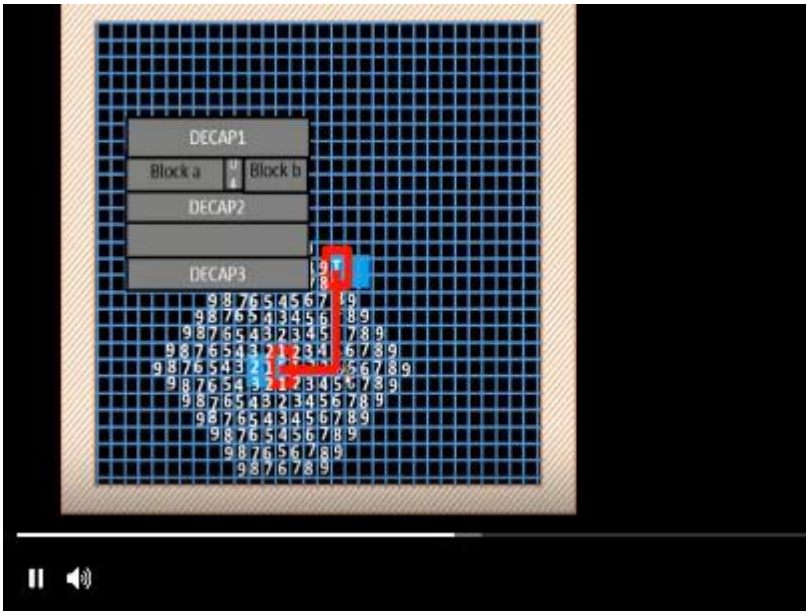
1. Initialization: The algorithm starts by initializing a routing grid or matrix representing the maze. Each cell in the grid can be one of several states: obstacle, empty, source, destination, or visited. The source cell is marked with a value of 0, indicating that it is the starting point.



2. Wave Expansion: The algorithm performs a wave expansion from the source cell, spreading outwards in all directions. At each step, the algorithm examines neighboring cells (up, down, left, and right) and assigns them a value one greater than the minimum value of their neighboring cells (excluding obstacles). This process continues until the destination cell is reached or until no more cells can be visited.



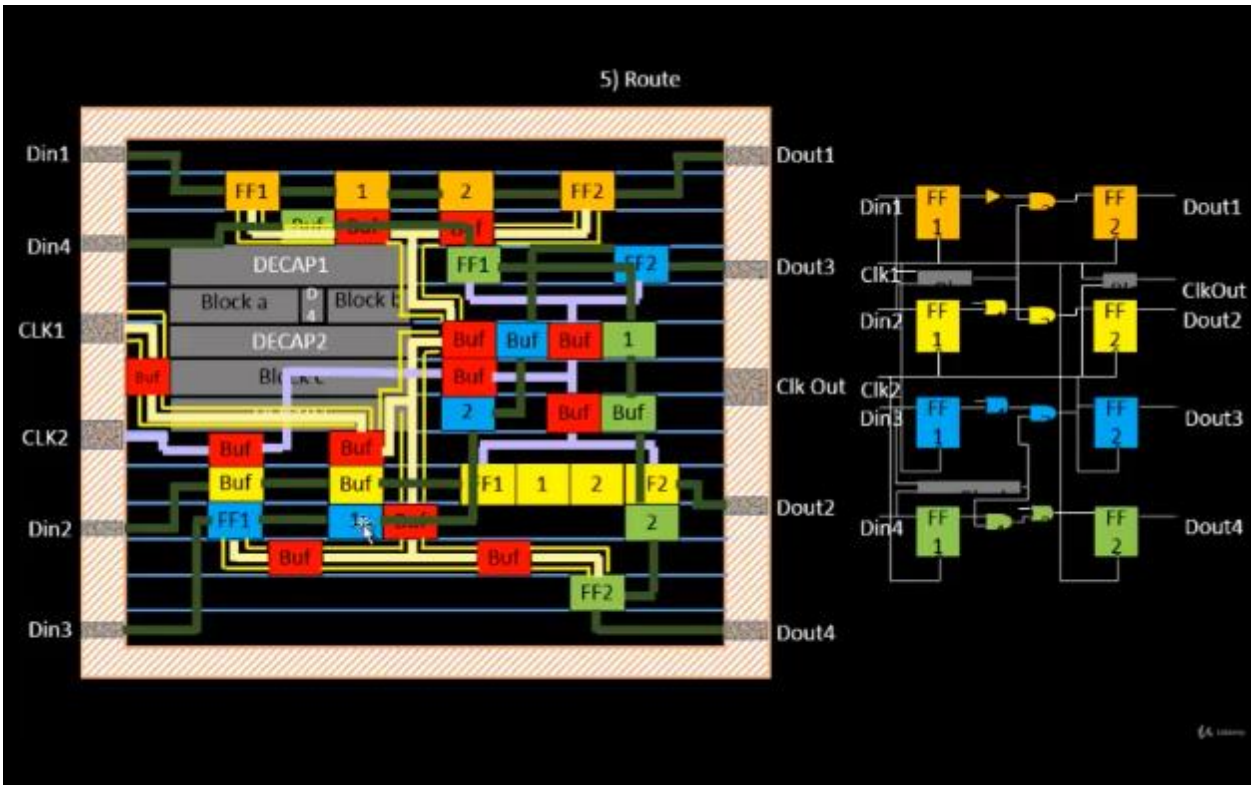
3. Backtracking and Path Reconstruction: Once the destination cell is reached, the algorithm traces back the path from the destination to the source by following the values in each cell. This results in the shortest path from the source to the destination. There might be multiple paths but the best path that the tool will choose is one with less bends. The route should not be diagonal and must not overlap any blockage/obstruction such as macros or HIPs.

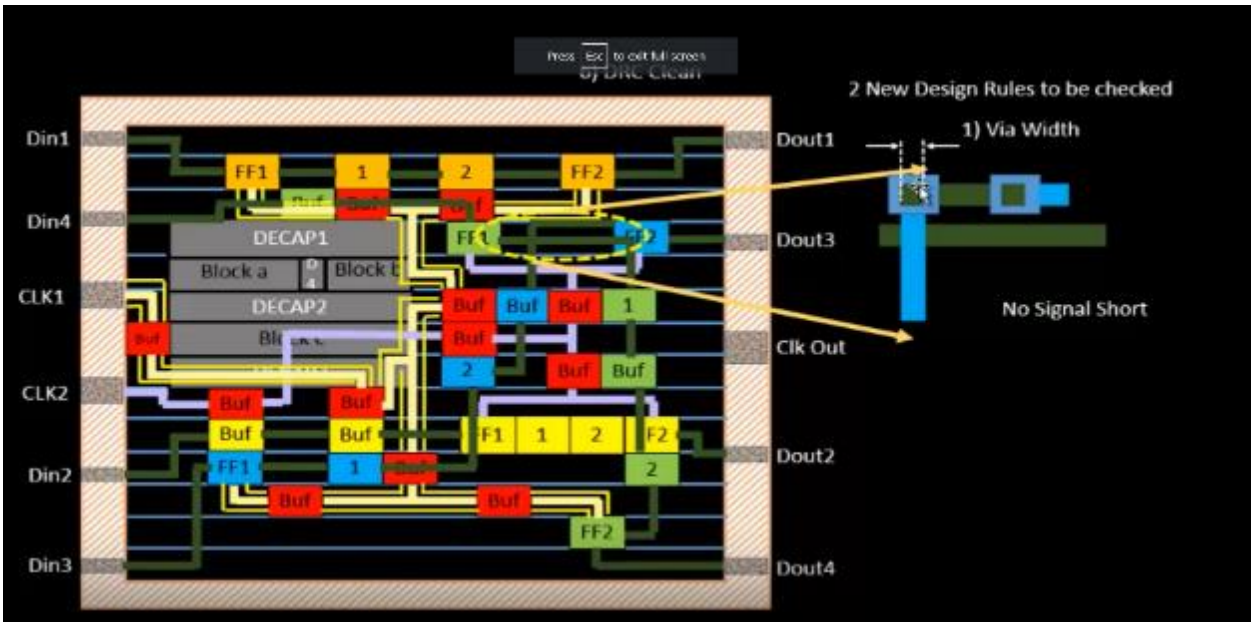


Design Rule Check

When routing, it's not merely about connecting two points; we must also adhere to specific rules. These rules, for example, mention that when constructing two wires, there must be a minimum spacing or distance between them, minimum wire width, minimum wire pitch etc. Hence, DRC cleaning is done to ensure the=at the routes can be fabricated and printed in silicon faithfully.

Signal short is also one of the critical issues as it causes functionality failure. It can be eliminated by moving the route to next layer with vias. This can lead to more DRCs (via width, via spacing, higher metal layer must be wider than lower metal layer etc.).


5) Route
Din1
Din4
CLK1
CLK2
Din2
Din3
Dout1
Dout3
Clk Out
Dout2
Dout4
DECAP1
DECAP2
Block a
FF1
FF2
Buf
1
2
Clk1
Clk2
ClkOut
Dout2
Dout3
Dout4


6) DRC Clean
2 New Design Rules to be checked
1) Via Width
No Signal Short
Din1
Din4
CLK1
CLK2
Din2
Din3
Dout1
Dout3
Clk Out
Dout2
Dout4
DECAP1
DECAP2
Block a
FF1
FF2
Buf
1
2

Power Distribution Network and routing

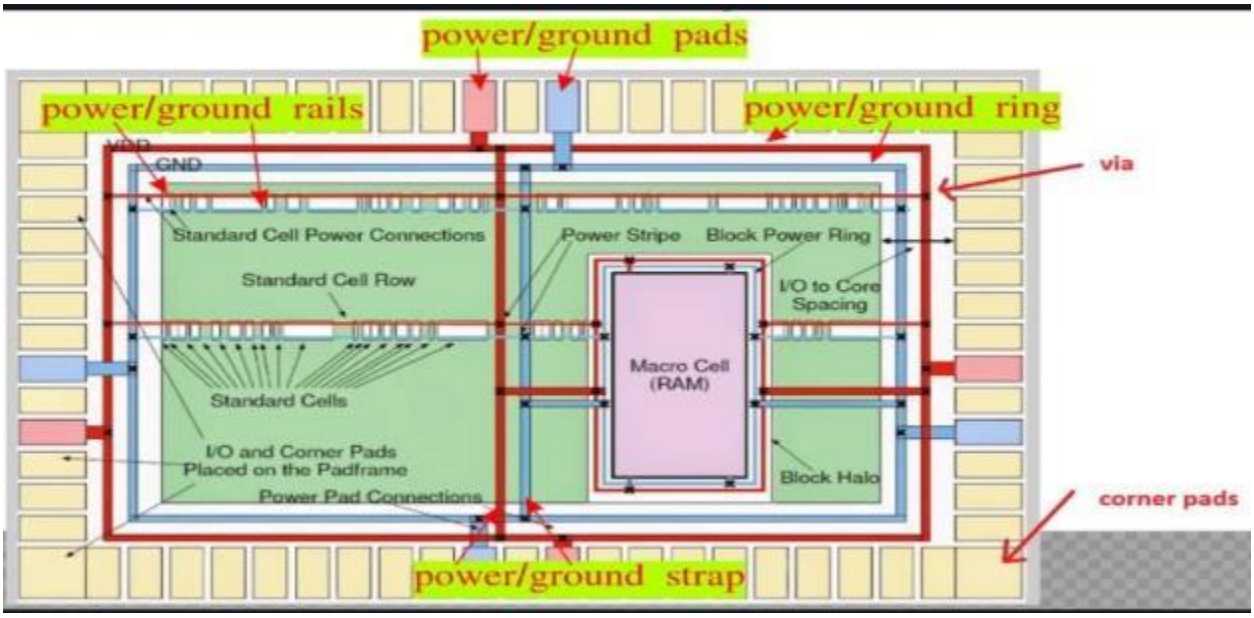
Lab steps to build power distribution network

1. Go to openlane directory

2. `docker`
3. `./flow.tcl -interactive`
4. `package require openlane 0.9`
5. `prep -design picorv32a -tag 19-03_16-40` (this is the folder till cts has been done)
6. `echo $::env(CURRENT_DEF)` /openLANE_flow/designs/picorv32a/runs/19-03_16-40/results/cts/picorv32a.cts.def
7. To generate PDN: `gen_pdn`

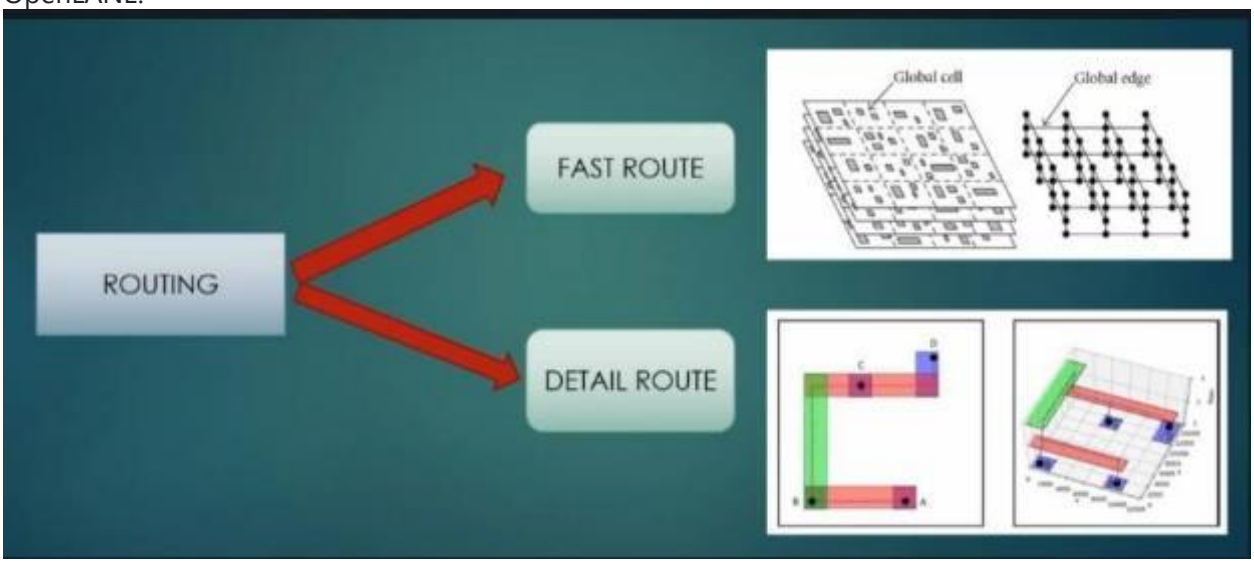
Lab steps from power straps to std cell power

The power and ground rails have a pitch of 2.72um and hence the reason reason why the custom inverter cell has a height of 2.72um, else the power and ground rails will not be able to power the cell. Looking at the LEF file `runs/[date]/tmp/merged.lef`, it is noticed that all cells are of height 2.72um and only width differs.
As shown below, power/ground pads -> power/ground ring-> power/ground straps -> power/ground rails to power up the standard cells.



Basics of global and detail routing and configure TritonRoute

TritonRoute is the engine that is used for routing. `run_routing` command does routing in OpenLANE.



In the VLSI flow, the routing stage is highly critical and can be executed using either open-source or commercial tools. This stage is divided into two phases:

1. Global Route / Fast Route:
    - This is accomplished using fast routing techniques where the area to be routed is partitioned into tiles or rectangles. Global routing establishes the initial framework for routing paths.

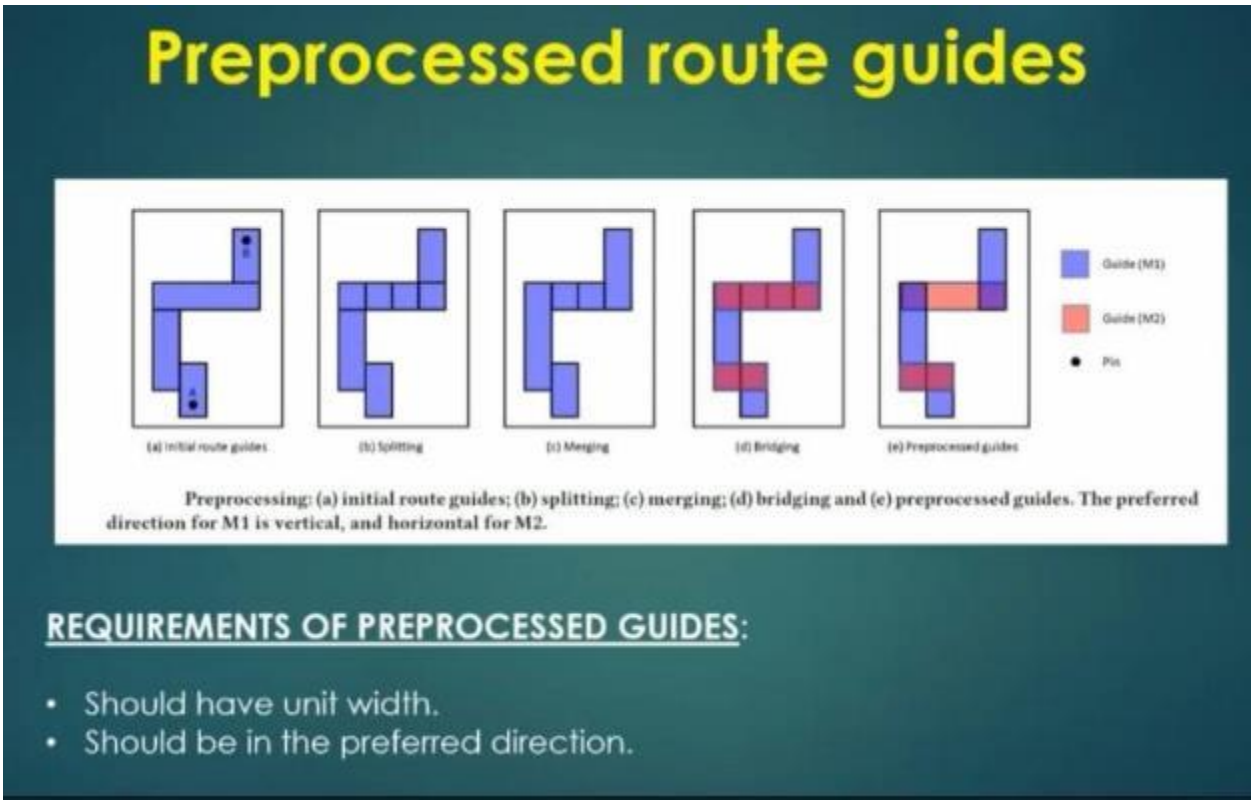
2. Detail Route:

    - This phase involves meticulous tracking routing techniques to complete the routing process. Detailed routing fine-tunes and finalizes the paths to ensure proper connectivity and compliance with design constraints.

TritonRoute features

TritonRoute feature 1 - Honors pre-processed route guides:

M1 preferred direction is vertical and M2 preferred direction is horizontal. Whenever the tool encounters a non preferred direction route, then it divides the route into unit width. This is called splitting. The divided unit width sections that fall in the same line of preferred direction routes are merged. The edges which are parallel to the preferred routing direction are bridged with the upper layer, process caled as bridging. Non preferred routing guides are now converted into preferred routing guides of M2.

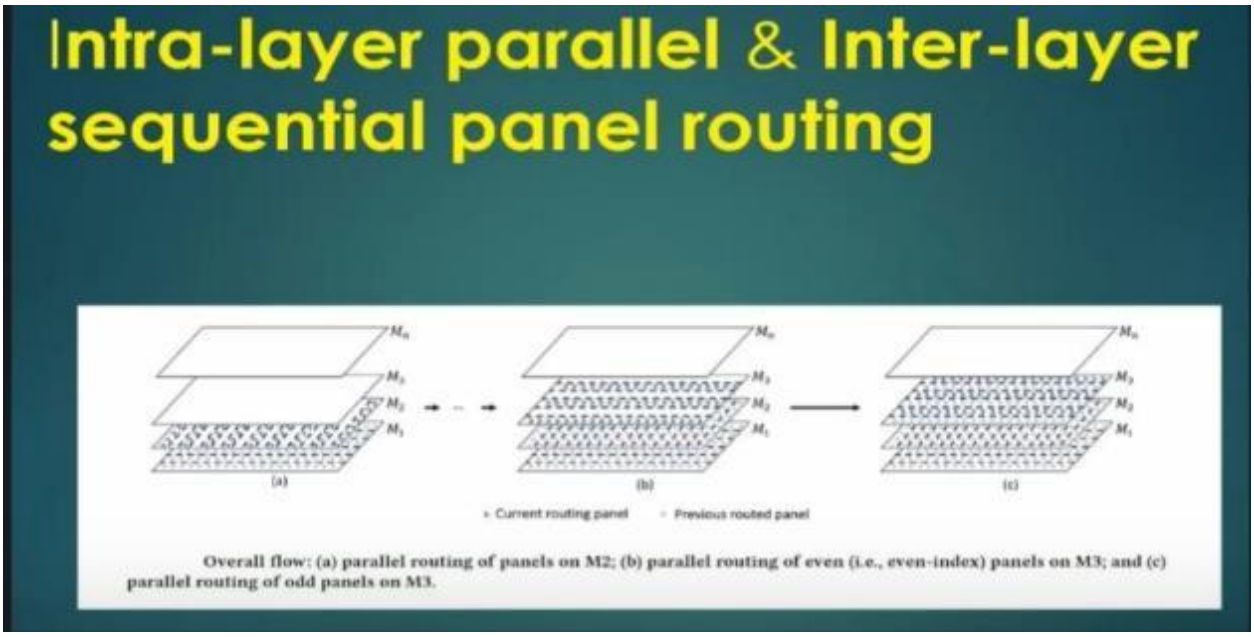


TritonRoute feature 2 - Inter-guide connectivity:

M1 and M2 are connected at the purple colour areas. The tool will understand that there is overlap area, then it will add via to connect M1 & M2.

TritonRoute feature 3 - intra- & inter-layer routing:

The preferred direction of the M1 layer is vertical, resulting in lines oriented vertically. The dashed lines are referred to as panels, with each routing guide assigned to a specific panel. When routing occurs within even-index panels, it's termed as intra-layer parallel panel routing. Initially, routing takes place simultaneously in all even-index panels, followed by routing in odd-index panels. This routing remains confined within a particular layer. Routing progresses from lower to upper layers, ensuring the orderly flow of routing operations.



TritonRoute method to handle connectivity

Problem Statement:

- **INPUTS**: LEF, DEF, Preprocessed route guides
- **OUTPUT**: Detailed routing solution with optimized wire-length and via count
- **CONSTRAINTS:** Route guide honoring, connectivity constraints and design rules

- Handling Connectivity

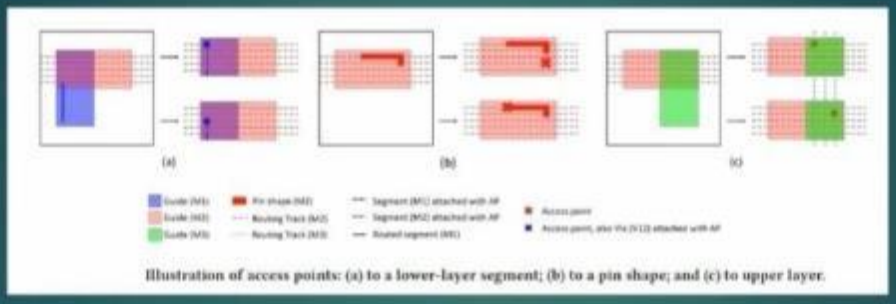
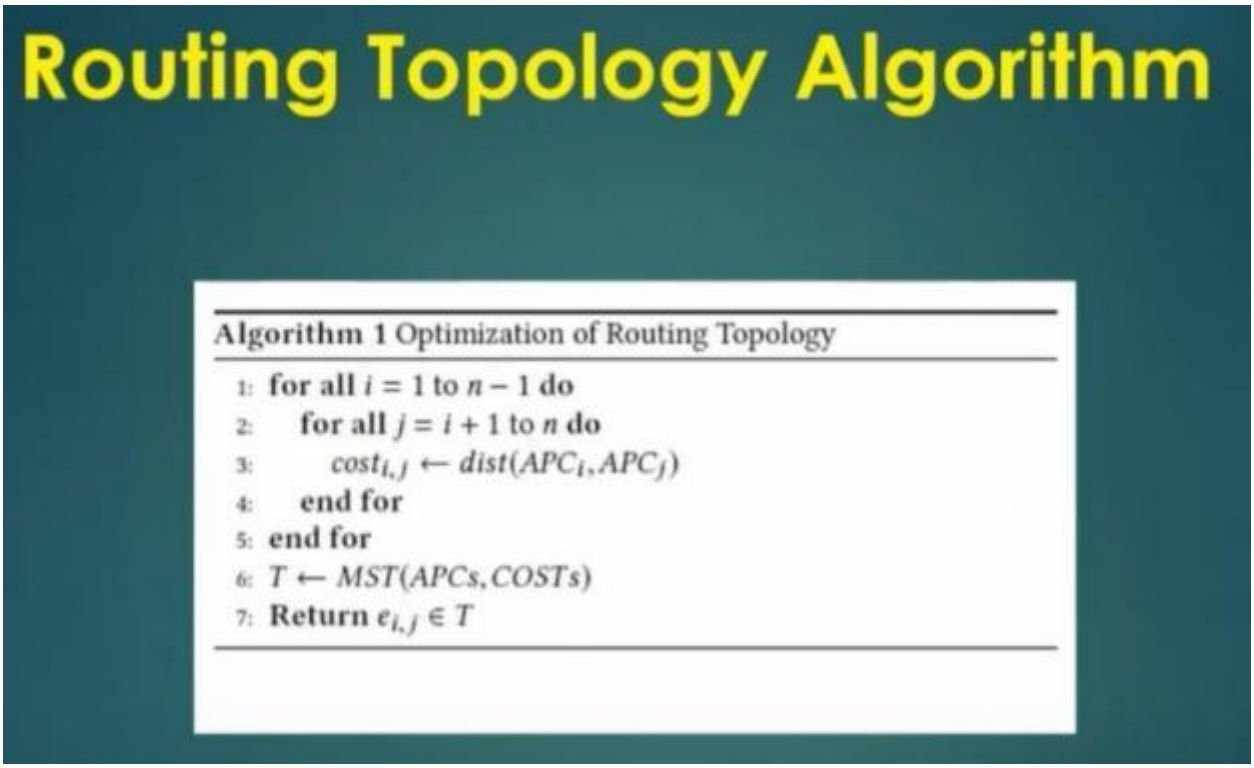


Illustration of access points: (a) to a lower-layer segment; (b) to a pin shape; and (c) to upper layer.

- **Access Point (AP):** An on-grid point on the metal layer of the route guide, and is used to connect to lower-layer segments, upper-layer segments, pins or IO ports.
- **Access Point Cluster (APC):** A union of all Aps derived from same lower-layer segment, upper-layer guide, a pin or an IO port.

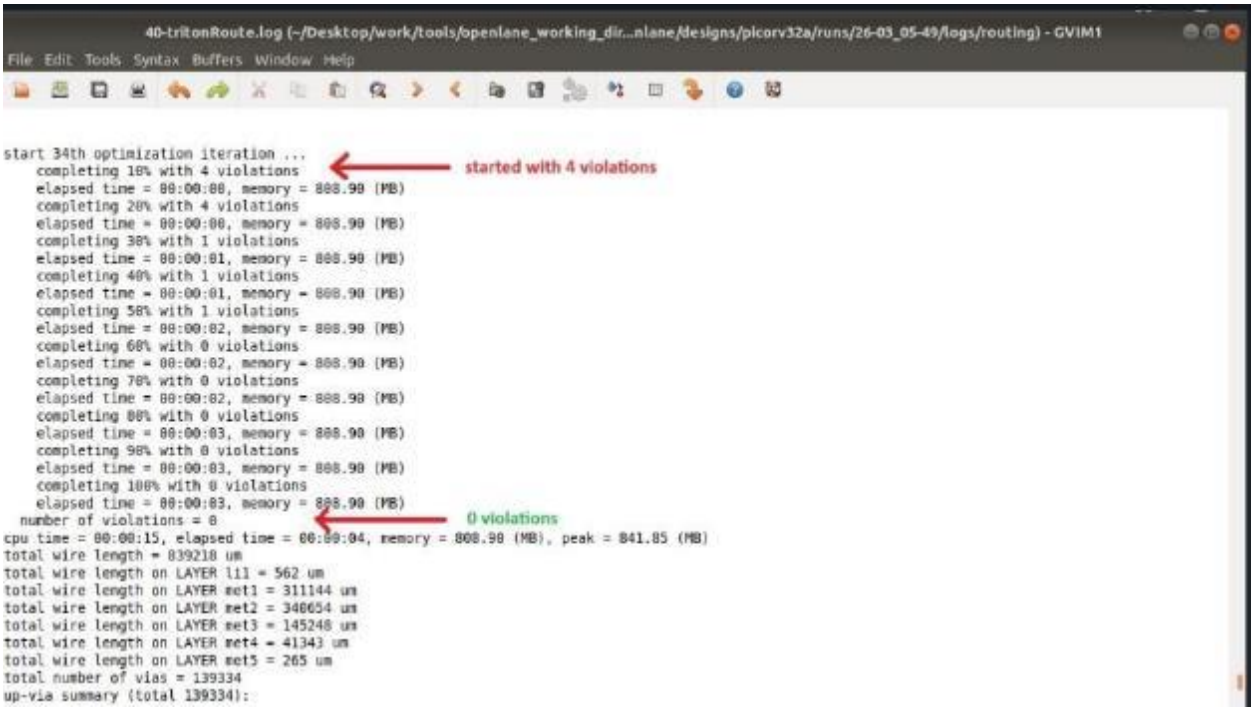
The Goal of MILP (Mixed Integer Linear Programming) algorithm is to find the optimal solution to connect two Access point cluster.

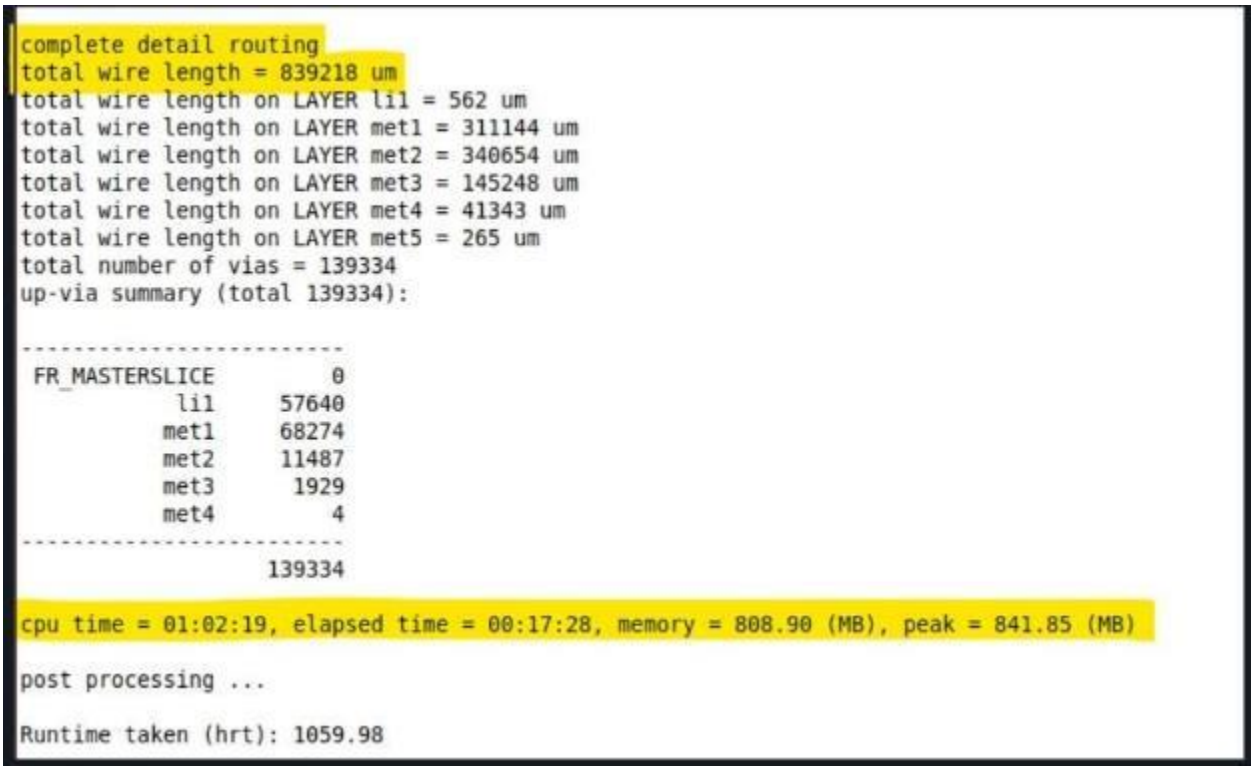


In the above algorithm, for each access point, cost is found. Then, a minimum spanning tree between access points and cost. So, the algorithm says that minimal and most optimal point is nedded between two APCs.

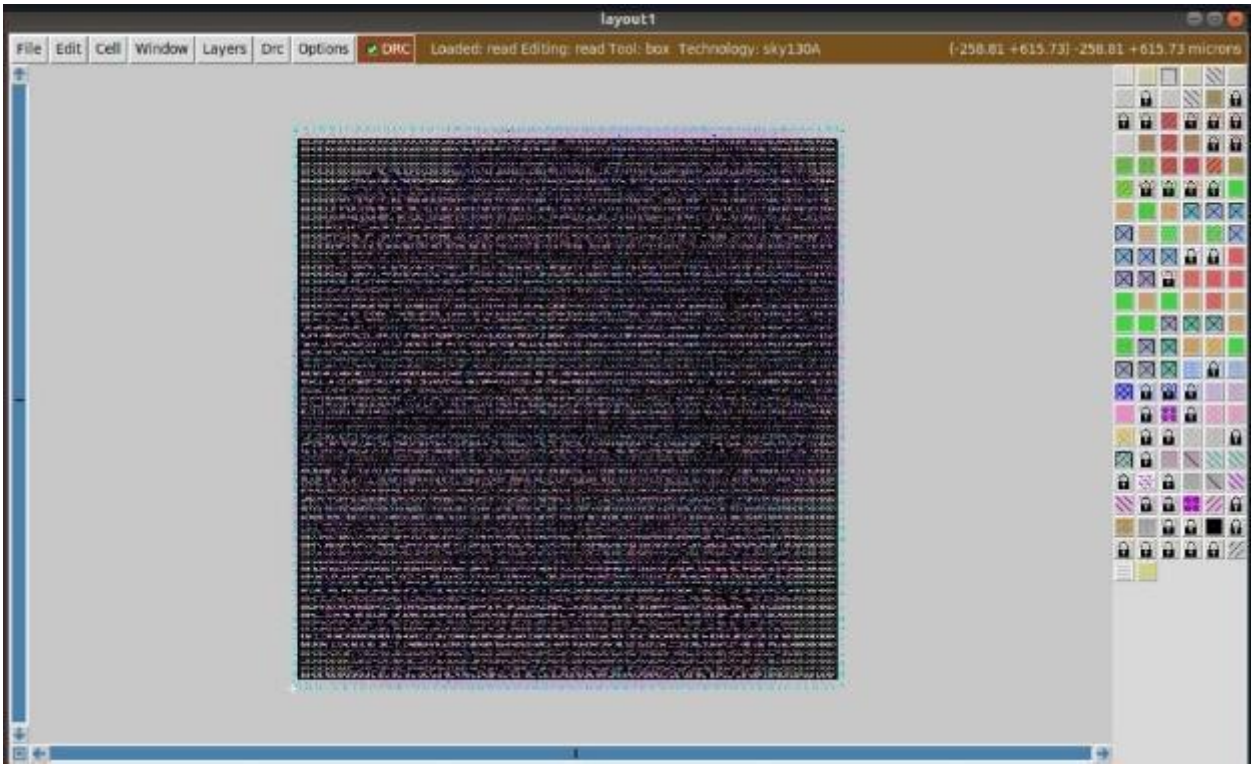
Final files list post-route

With `run_routing` command, routing got completed. Both global routing (fast routing) and detail routing are done. It takes multiple iterations to bring down the DRC violations to 0. routing strategy was set to 0. In the intial iteration, the violation count was close to 25000 and at the 34th iteration, the violations got resolved and became 0. The entire routing operation took nearly 25 minutes.





A DEF file will be formed in `runs/[date]/results/routing/picorv32.def`. Open the DEF file of routing stage in Magic.



Parasitic extraction:

OpenLane does not have any spef extraction tool, so we use a separate tool present in work/tools/ directory.

1. Go to /home/vsduser/Desktop/work/tools
2. Inside that there is a folder named SPEF_EXTRACTOR
3. SPEF_EXTRACTOR contains a list of files, out of which there is a python file called main.py. It helps to generate the SPEF provided there are lef & def files
4. To create SPEF file `python3 main.py /home/vsduser/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/26-03_05-49/tmp/merged.lef /home/vsduser/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/26-03_05-49/tmp/routing/picorv32a.def`
5. spef will be saved in the same location as def file. `/home/vsduser/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/26-03_05-49/tmp/routing`

The last stage will be to extract the GDSII file ready for fabrication `run_magic`
This uses Magic to stream the GDSII file `runs/26-03_05-49/results/magic/picorv32a.gds`.
This GDSII file can then be read by Magic:

The PnR flow is done