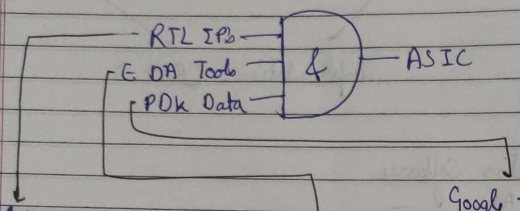
Digital ASIC Design

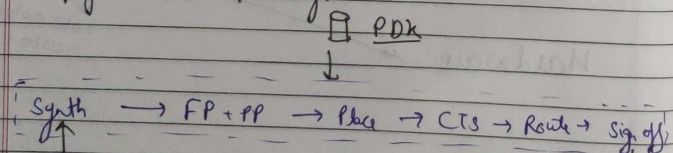


libracores.org opencores.org github.com	QFlow openROAD Open LANE	Google + Skywater ↓ FOSS 130nm Production PDK github.com/google/skywater-pdk

ASIC design flow

ASIC flow objective : RTL to GDS II
- Also, called Automated PnR and/or Physical Implementation

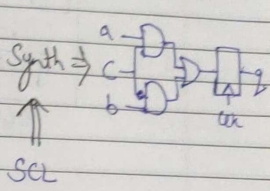
Simplified RTL to GDS II flow



- Synthesis
- Floor / Power planning
- Placement
- Clock Tree Synthesis
- Routing
- Sign off

→ Synthesis :- Converts RTL to a circuit out of components from the standard cell library (SCL)

```
always @ (posedge clk)
  if (c)  q <= a;
  else    q <= b;
```



- "Standard Cells" have regular layout
- Each has different views / models
  - Electrical (HDL, SPICE,
  - Layout (Abstract and Detailed)

→ Floor & Power planning

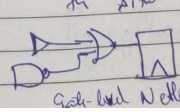
- Chip Floor-planning : Partition the chip die between different system building blocks and the place the I/O Pads

→ Macro Floor Planning :- Dimension, pin location, keep out definition

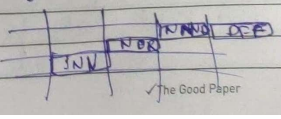
- Power planning

Power pads (VDD, VSS) — Power straps — Power Rings

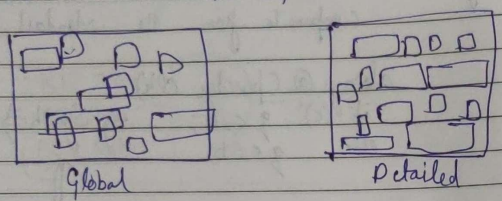
→ Placement :- Place the cells on the floor plan rows aligned with the site



Gate-level Netlist

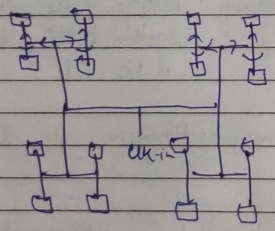


- Usually done in 2 steps : Global and Detailed



→ Clock Tree Synthesis

- Create a clock distribution network
  - To deliver the clock to all sequential elements (eg. FF)
  - With minimum skew (zero is hard to achieve)
  - And in a good shape
  - Usually a Tree (H, X, ...)



→ Routing

- Implementing the interconnect using the available metal layers
- Metal tracks from a routing grid

- Routing grid is huge
- Divide and Conquer
  - Global Routing : Generates routing guides
  - Detailed Routing : Uses the routing guides to implement the actual wiring

→ Sign off

- Physical Verification
  - Design Rules Checking (DRC)
  - Layout vs Schematic (LVS)
- Timing Verification
  - Static Timing Analysis (STA)

* OpenLANE

→ Started as an open-source flow for a True Open Source Tape-out Experiment

→ SkiVe is a family of open everything SoCs
- open PDK, Open EDA, Open RTL

SkiVe SoC family

SoC	Features
SkiVe	Sky 130 SCL + Synthesized 1 Kbyte SRAM
SkiVe 2	Sky 130 SCL + 1 Kbytes Open RAM block
SkiVe 2q	SkiVe 2 with a single chip module
SkiVe 3	OSU SCL + Synthesized 1 Kbyte SRAM
SkiVe 5	Sky 130SCL + 8x1 Kbytes Open RAM banks
SkiVe 6	SkiVe 2 + AXI4 DFT

- Main Goal:-
  Produce a GDSII with no human intervention (no-human-in-the-loop)

→ Clean means:-
- No LVS violation
- No DRC violation
- Timing violation? WIP! (Work in Progress)

→ Tuned for Sky Water 130nm Open PDK
- Also, supports GFAB 180 and GF130G

→ Containerized
- Functional out of the box
- Instructions to build and run natively will follow

→ Can be used to harden Macros or Chips

→ 2 mode of operation:-
- Autonomous or Interactive

→ Design Space Exploration:
- Find the best set of flow configuration

→ Large number of design examples
- 43 designs with their best configuration
- More will be added soon.

→ OpenLANE Regression Testing

→ The design exploration utility is also used for regression testing (CI)

→ We run OpenLane on ~70 designs and compare the results to the best known ones

→ DFT (Design for Test)
- Scan Insertion
- Automatic Test Pattern Generation (ATPG)
- Test Pattern Compaction
- Fault Coverage
- Fault Simulation

→ Physical Implementation
- Also called automated PnR (Place and Route)
- Floor / Power Planning
- End Decoupling Capacitors and Tap cells insertion
- Placement: Global and Detailed
- Post placement optimization
- Clock Tree Synthesis (CTS)
- Routing: Global and Detailed

→ Logic Equivalence Check (LEC)
- Every time the netlist is modified, verification must be performed
- CTS modifies the netlist
- Post Placement optimizations modifies the netlist

Date: Page No.: 7

- LEC is used to formally confirm that the function did not change after modifying the netlist

* Design Preperation Setup

→ cd Desktop/coark/tools/openlane_working_dir/openlane

→ cd designs

→ cd picorv32a

→ ls -ltr

→ cd ../

→ cd ../

→ docker

→ ./flow.tcl -interactive

→ prep -design picorv32a

→ run_synthesis

flop ratio = No. of dfxtp / No. of cells

→ Package require openlane 0.9

New Terminal in same directory.

→ cd runs

→ ls -ltr

→ cd 08-07_13-57

→ ls -ltr

• after run_synthesis

→ cd results

→ cd synthesis

Date: Page No.: 8

→ less picorv32a.synthesis.v

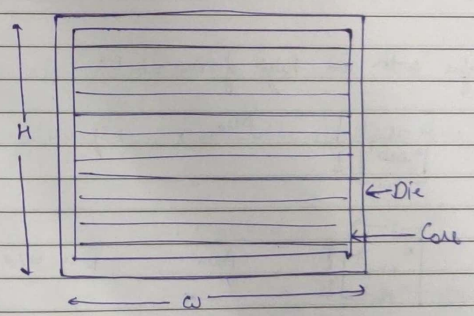
→ cd ../../

→ cd reports

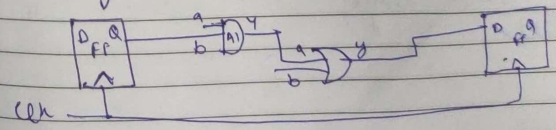
→ cd synthesis

→ less 1-yosys_4.stat.rpt

1→ Define width and Height of Core and Die



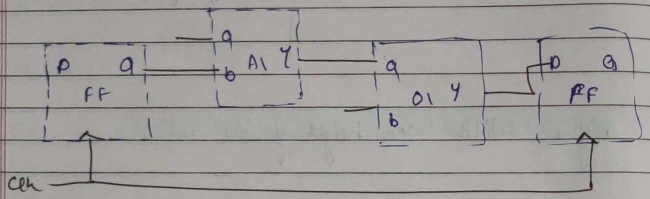
Lets begin with a netlist



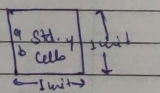
FF = Flip flops / Latches / Registers
A_1, O_1 = Standard Cells (AND, OR, INVERTER)

Note: A "netlist" describes the connectivity of an electronic design

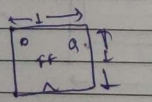
→ Now, lets convert the highlighted symbols into physical dimension



1. Define width and Height of Core and Die

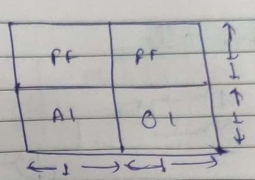


Area → 1 sq. unit

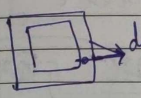


Area → 1 sq. unit

Let's calculate are occupied by the above netlist on a silicon wafer



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

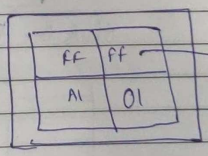


die A 'die', which consist of core, is small semiconductor material specimen on which the fundamental circuit is fabricated

→ Place all logical cells inside the 'Core'

→ The logical cells occupies the complete area of the core

100% utilization



Utilization factor = $\frac{\text{Area occupied by Netlist}}{\text{Total area of Core}}$

Aspect ratio = $\frac{\text{Height}}{\text{Width}}$

$= \frac{2}{2}$

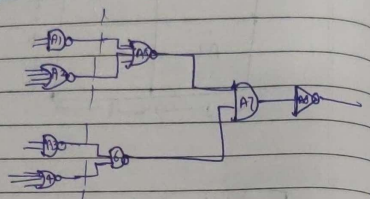
$= 1$

$= \frac{4 \times 1 \text{ sq. unit}}{2 \times 2 \text{ sq. unit}}$

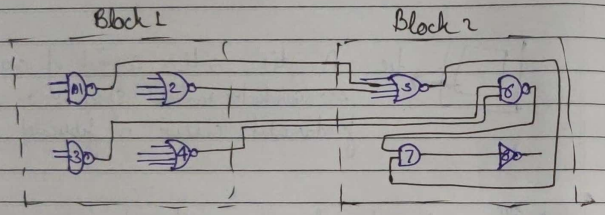
$= 1$

2. Define locations of prefixed cells

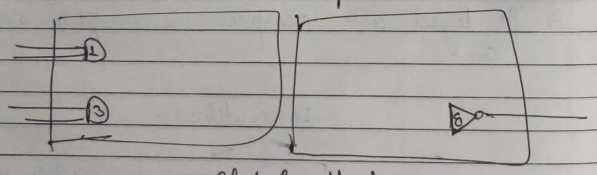
Combinational logic ⇒



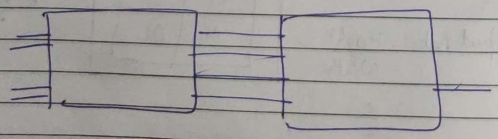
Cut 1 | Cut 2



Extend IO pins

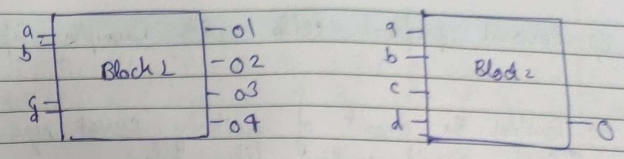


Black Box the boxes

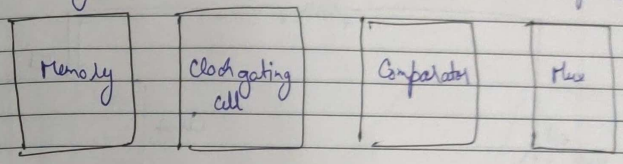


Separate the Black Boxes as two different IPs or modules

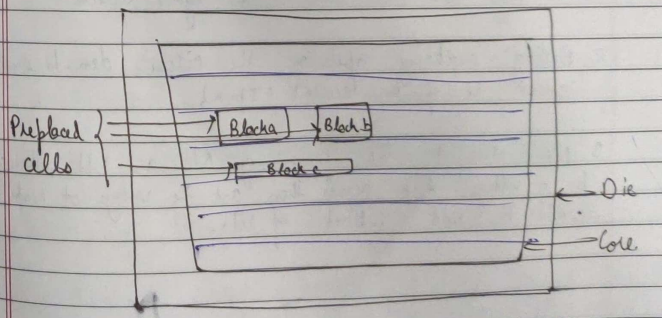
✓The Good Paper



- Similarly, there are other IP's also available, for eg:



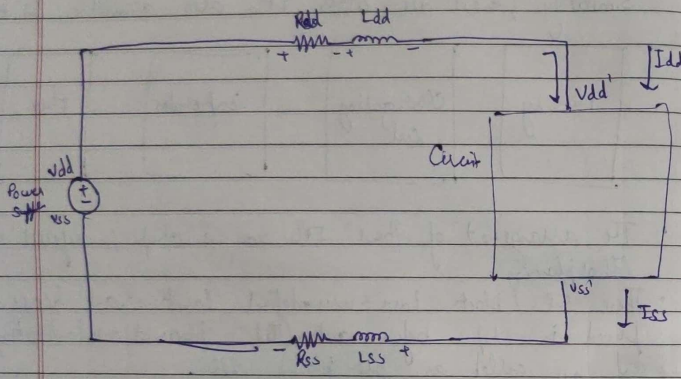
- The arrangement of these IP's in a chip is referred as Floorplanning
- These IP's / block have user-defined locations, and hence are placed in chip before automated placement and routing and are called as pre-placed cells.
- Automated placement and routing tools places the remaining logical cells in the design onto chip.



✓The Good Paper

3) Surround preplaced cells with Decoupling Capacitors.

→ Consider the amount of switching current required for a complex circuit something like below.



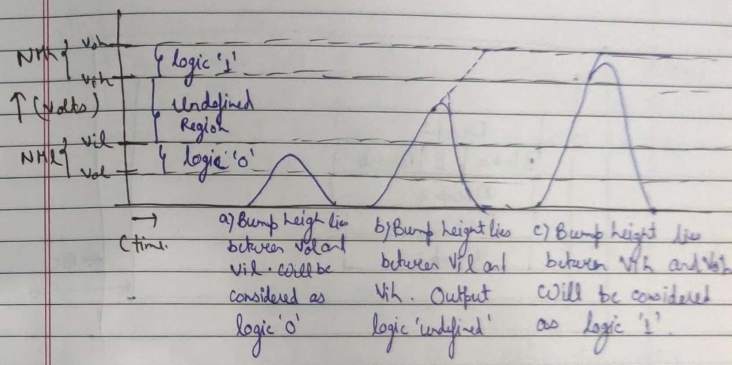
1. Consider capacitance to be zero for the discussion Rdd, Rss, Ldd and Lss are all defined values

2. During switching operation, the circuit demands switching current i.e. peak current (Ipeak)

3. Now, due to the presence of Rdd and Ldd, there will be a voltage drop across them and the voltage at Node 'A' would be Vdd' instead of Vdd.

4. If Vdd' goes below the noise margin due to Rdd and Ldd, the logic '1' at the output of circuit 1 may be detected as logic '1' at the input of the circuit following the circuit

→ Noise Margin summary



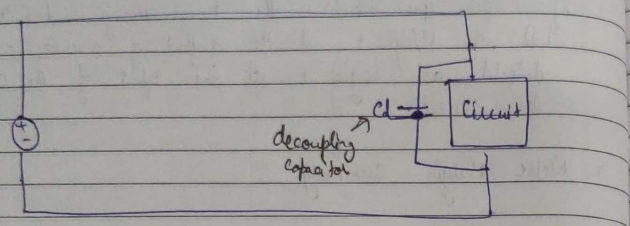
Noise induced bump characteristics at different noise margin levels

→ For any signal to be considered as logic '0' and logic '1', it should be in the NML and NMH ranges, respectively

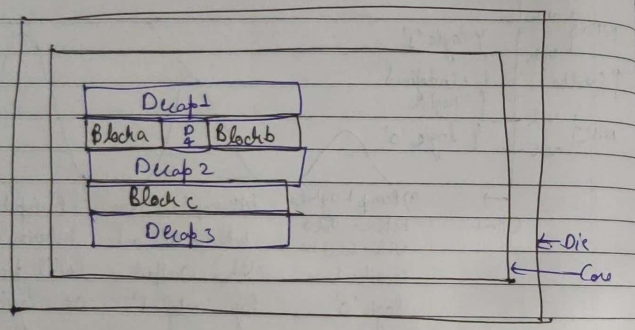
Solution : Add Decoupling Capacitors

1. Addition of Decoupling Capacitor in parallel with the circuit

2. Every time the circuit switches, it draws current from C whereas, the RL network is used to replenish the charge in C

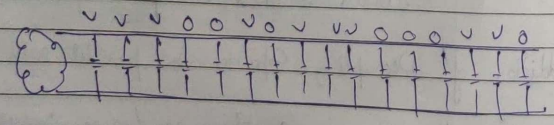


→

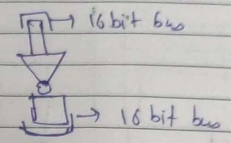


→ Now suppose we are sending a signal ⎍ data from driver to load then this signal should be same all the time but we cannot add capacitor everywhere.

→ Now, Lets the output of 16-bit bus, is connected to an inverter.



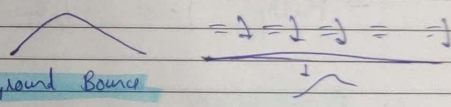
1111 0000 1111 0000



0001 1010 0011 1001

→ What does this mean?.

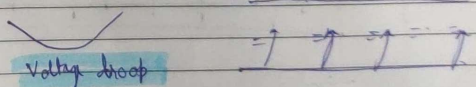
→ This means all the capacitors which were charged to 'V' volts will have to discharge to '0' volts through single 'Ground' top point. This will cause a bump in 'Ground' top point.



Ground Bounce

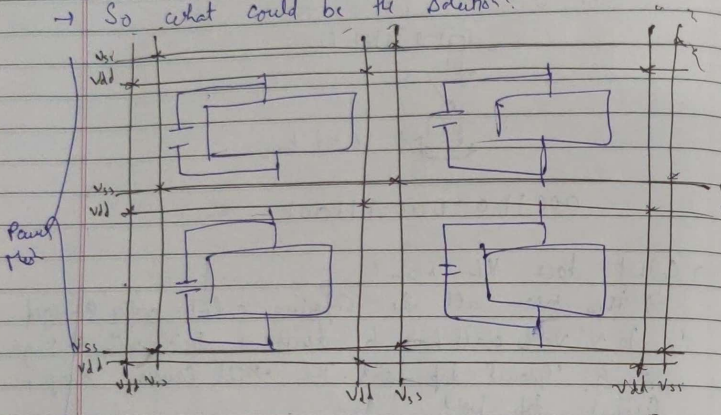
→ If this Ground Bounce is greater than noise margin level it will be in undefined region.

→ Also all capacitors which were '0' volts will have to charge to 'V' volts through single 'Vdd' top point. This will cause lowering the voltage at 'Vdd' top point.



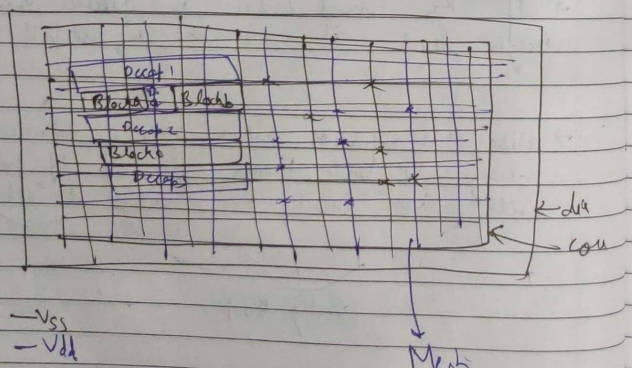
Voltage droop

→ So what could be the solution?



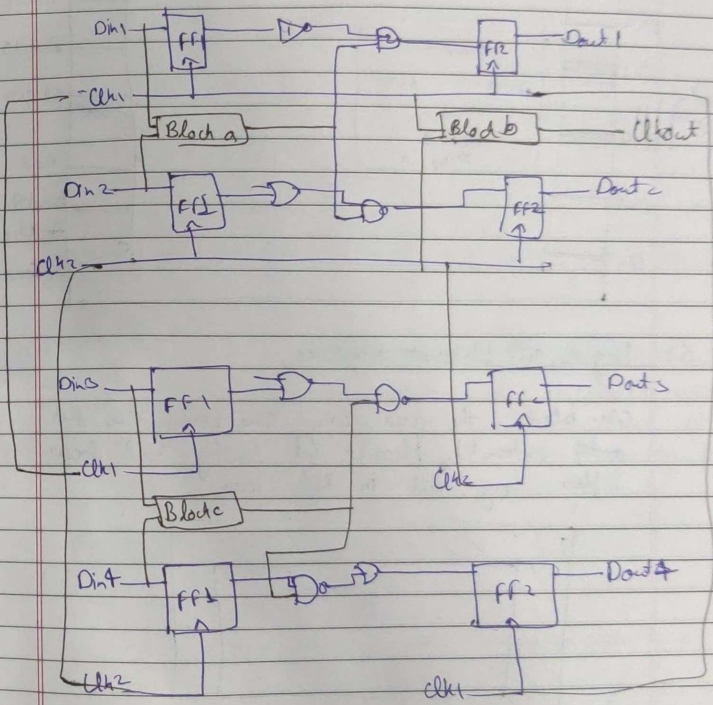
→ Multiple power supply.

4) Power planning



5) Pin Placement

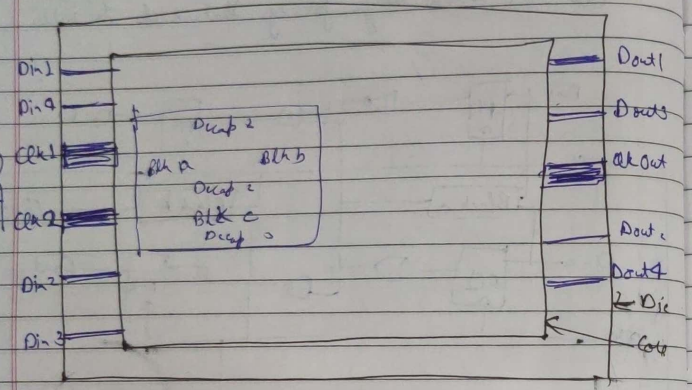
Lets take below design for eg. that needs to implement



The connectivity information between the gates is called using VHDL/Verilog language and is called as the "netlist"

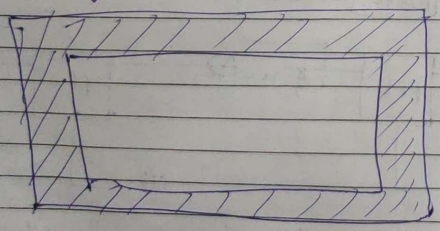
Date:
Page No. 19

Pin Placement



6) Logical cell placement Blockage

We block the area using some blockage so that ~~auto~~ automated placement and routing tool doesn't place any cell in this area.



✓The Good Paper

Date:
Page No. 20

→ After synthesis is run it's not again re-made

→ run_floorplan	→ cd Desktop/coeth/tools/openlane_working_dir/openlane/configuration
	→ less README.md
	→ less floorplan.tcl
	→ cd openlane/designs/picorv32a/runs/08-07_13-57/results/floorplan
	→ less picorv32a.floorplan.def ↑ Here we can find total die area
	→ magic -T /Desktop/coeth/tools/openlane_coeth_dir/pdks/sky130A/libs.tech/magic/sky130A.tech lef read ../../tmp/merged.lef def read picorv32a.floorplan.def

In magic

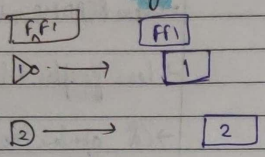
S → select
B → Center
Zoom → ~~Right~~ click → Right click → e

✓The Good Paper

Netlist binding and initial place design

Placement & Routing

1) Bind netlist with physical cells



FF1	1	2	FF2
FF1	1	2	FF2
FF2	1	2	FF2
FF1	1	2	FF2

Physical views of logic gates

Block a | Block b

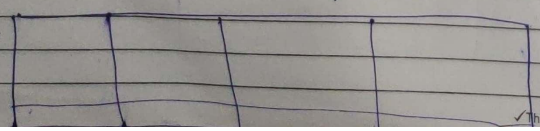
Block c

Library → consist of shape & size info
→ consist of delays

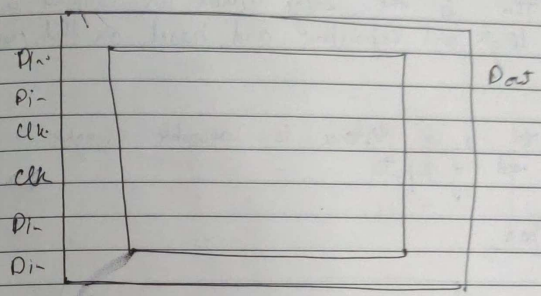
different size

if with load models

FF1	1	2	FF2
FF1	1	2	FF2
FF1	1	2	FF2
FF1	1	2	FF2

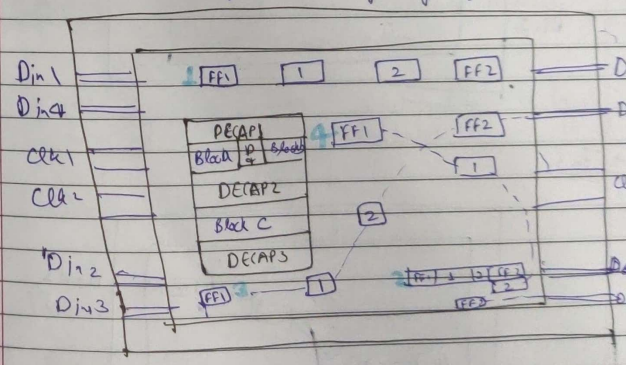


2) Placement



Floorplan.

→ Till now we have Floorplan (Prefixed cell/Pin placement), Netlist and Physical views of logic gates

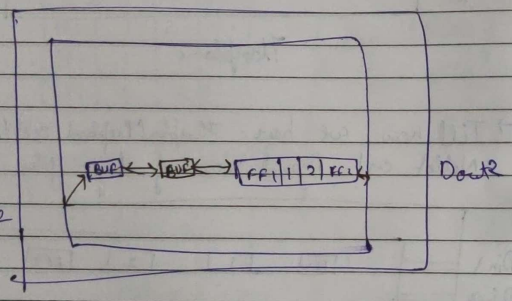


3) Optimize Placement

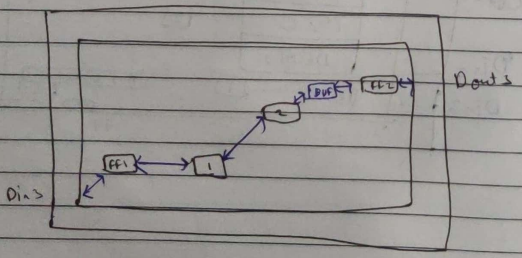
→ This is the stage where we estimate wire length and capacitance and, based on that, insert/delete repeaters

→ for 1 distance is reasonable enough so no need of repeater

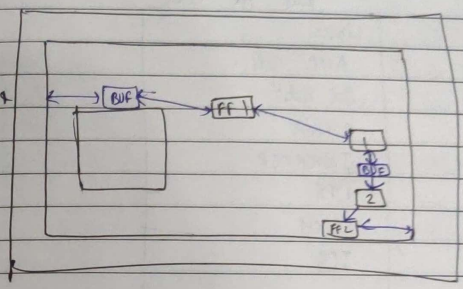
→ for 2



→ for 3



for 4



→ **Need for characterization**

Library characterization and modelling

Part 1 → Concepts and Theory - NLDM, CCS timing, power and noise characterization

Logic synthesis → functionality to legal hardware (code)

↓

Floorplanning

↓

Placement

↓

Clock tree synthesis

↓

Routing

↓

Sign off Timing Analysis

Date:
Page No.: 25

One Common Thing across all stages
"Gates or Cell"

Library
AND gate
OR gate
BUFFER
INVERTER
DFF
LATCH
ICG

→ Congestion aware placement

→ user_placement

→ cd openlane_working_dir/openlane/designs/picorv32a/runs/08-09_13-57/results/placement

→ ls

→ magic (save as previous)

read picorv32a.placement.def

Date:
Page No.: 26

→ Cell Design flow

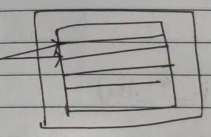
FF1	and	or	FF2
FF1	buf	inv	FF2
FF1	inv	and	FF2
FF1	buf	or	FF2

Foundry

Cell design flow
Inputs
Process design kits (PDKs): DRC & LVS rules, SPICE models, Library & user-defined specs

Design steps
Circuit design, layout design, characterization

Design Outputs
CDL (Circuit description language), GDS II, LEF, extracted SPICE netlist (.cir)



Supply voltage
Metal layers
Pin location
Place gate height
} User defined

Characterization flow

Date: Page No.: 27

Timing characterization

→ Timing threshold definition

slew_low_rise_thr	about 20% →
slew_high_rise_thr	
slew_low_fall_thr	
slew_high_fall_thr	
in_rise_thr	50%
in_fall_thr	50%
out_rise_thr	50%
out_fall_thr	50%

→ Propagation delay

time (out_*_thr) − time (in_*_thr)

→ Transition time.

~~time (slew_high*_rise_thr)~~

time (slew_high*_thr) − time (slew_low_*_thr).

→ output current waveform

✓The Good Paper

MY CHOICE
Date: Page No.: 26

Day 3

IO placer version

cd ... openlane/designs/picorv32a/runs/17-07_09-51/results/floorplan

→ magic -T ... picorv32a.floorplan.def

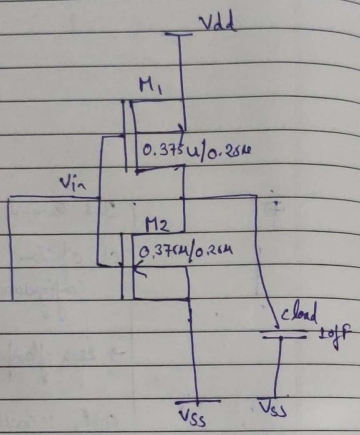
→ 3rd Terminal
cd .openlane/configuration
→ less floorplan.tcl
copy ::env(FP_IO_MODE)
→ set ::env (FP_IO_MODE) 2
→ run_floorplan
→ magic -T ... save above file

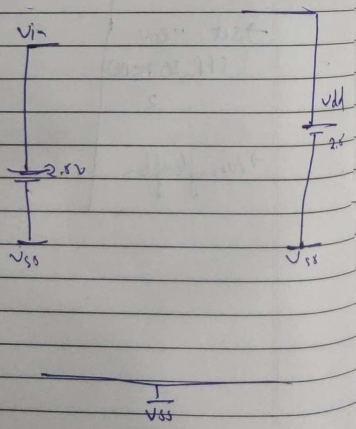
✓The Good Paper

Spice deck creation for CMOS inverter

SPICE deck

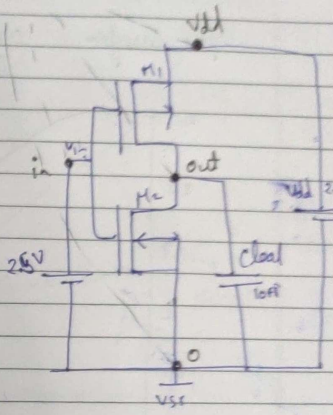
- Componet connectivity
- Compont values





Notes

- Identify 'nodes'



*** MODEL Description ***
*** Netlist description ***

M1 out in vdd vdd pmos w=0.375u L=0.25u
M2 out in 0 0 nmos w=0.375u L=0.25u

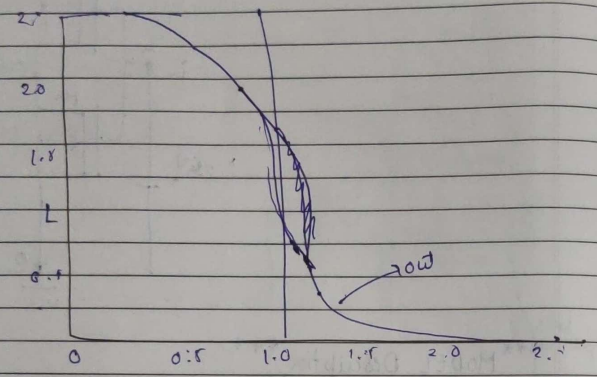
cload out 0 1pf

Vdd vdd 0 2.5
Vin in 0 2.5

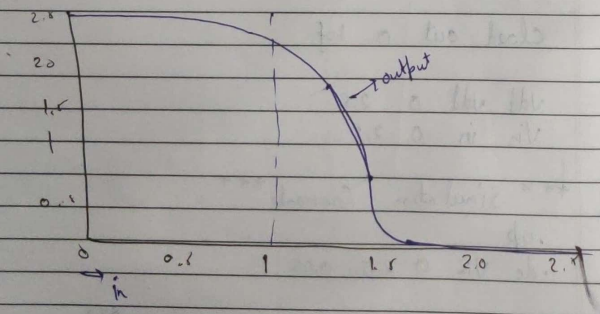
*** Simulation Commands ***
.op
.dc Vin 0 2.5 0.05

*** include tsmc_025um_model.mod ***
.Lib "tsmc_025um_model.mod" CMOS_MODELS
.end

Spice waveform : $W_n = W_p = 0.375u$, $L_{n,p} = 0.25u$ during
$(W_n/L_n = W_p/L_p = 1.5)$

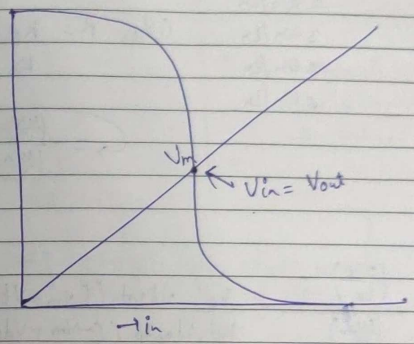


Now $(W_n/L_n = W_p/L_p = 2.5)$
$W_n = 0.375$ $W_p = 0.9375u$ $L_{n,p} = 0.25u$

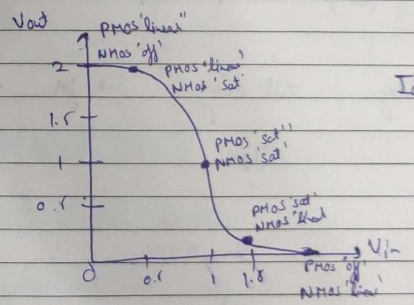


✓The Good Paper

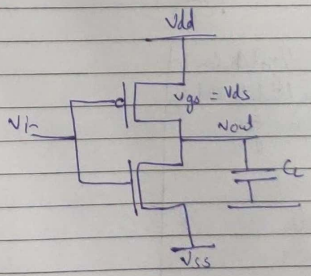
1. Switching Threshold, V_m



V_m is point where $V_{in} = V_{out}$



$I_{dsP} = I_{dsN}$



✓The Good Paper

Static and dynamic simulation of CMOS inverter

Static behaviour Evaluation: CMOS inverter Robustness

Wp/Lp	x. Wn/Ln
"	Wn/Ln
"	2 Wn/Ln
"	3 Wn/Ln
"	4 Wn/Ln
"	5 Wn/Ln

$$V_m = R.V_{dd} / (1+R)$$

Where $R = \dfrac{k_p . V_{dsatp}}{k_n . V_{dsatn}} = \dfrac{\left(\frac{W_p}{L_p}\right) k_p' . V_{dsatp}}{\left(\frac{W_n}{L_n}\right) k_n' . V_{dsatn}}$

$$\frac{\left(\frac{W_p}{L_p}\right)}{\left(\frac{W_n}{L_n}\right)} = \frac{k_n' . V_{dsatn}\left([V_m - V_t]\right) - \frac{V_{dsatn}}{2}}{k_p' . V_{dsatp}\left([-V_m + V_{dd} + V_t]\right) + \frac{V_{dsatp}}{2}}$$

Rise delay	Fall delay	→ for Wp/Lp Wn/Ln
148 ps	71 ps	

$V_m = 0.99V$

→ cd Desktop ... openlane

→ git clone https://github.com/nickson-jose/vsdstdcelldesign.git

→ ls -ltr

→ cd vsdstdcelldesign

→ cd openlane_working_dir/pdks/sky130A/libs.tech/magic

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

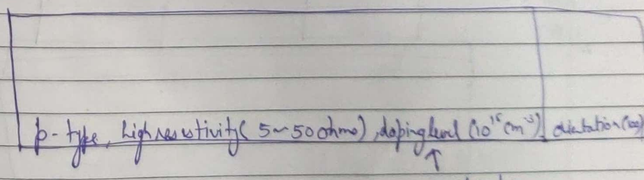
→ ls -ltr

→ magic -T sky130A.tech sky130_inv.mag ... &

Create Active Regions

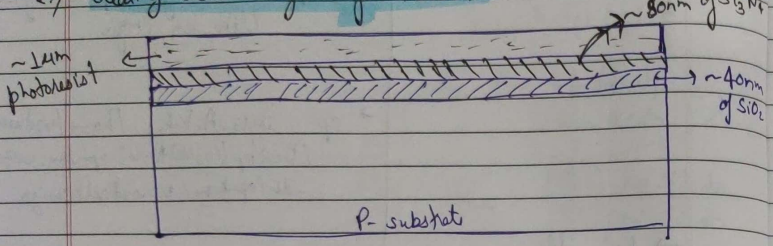
16-Mask CMOS process

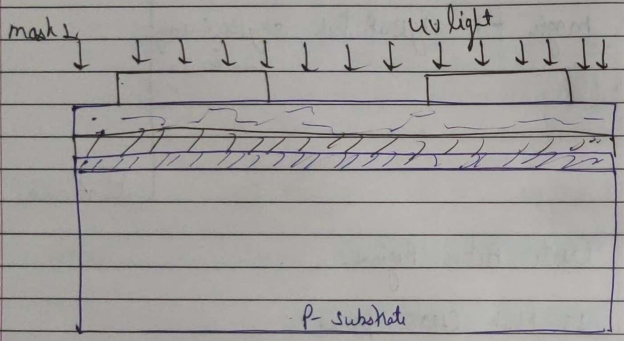
1) Selecting a substrate



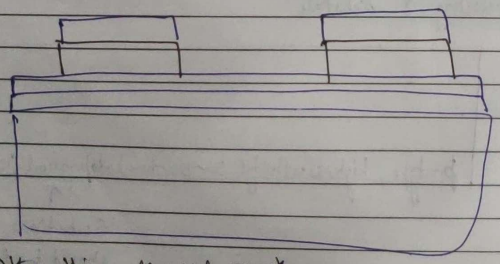
↑ Substrate doping should be less than 'well' doping.

2.) Creating active region for transistors



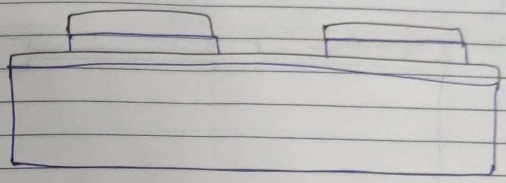


Washed out in developing solution

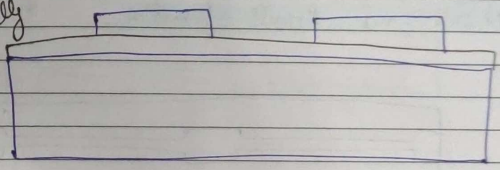


After this remove mask

Si_3N_4 etch

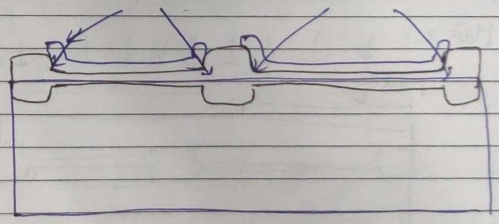


Remove photoresist chemically



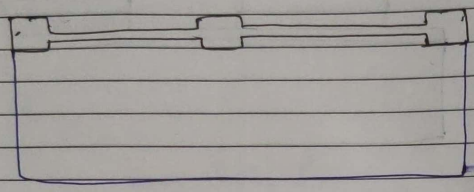
→ Placed in oxidation furnace

Bird's beak

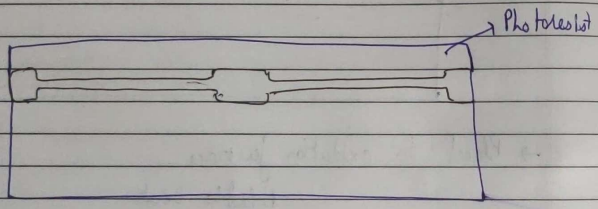


Field oxide is grown This process is ~~is~~ called "LOCOS" "Local Oxidation of Silicon"

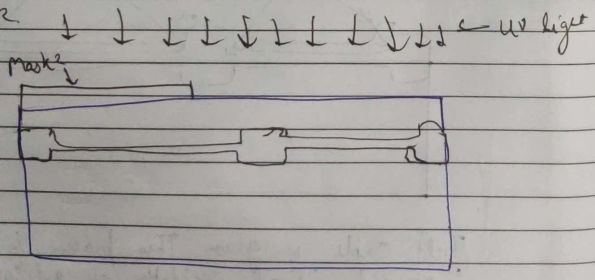
→ Si_3N_4 stripped using hot phosphoric acid

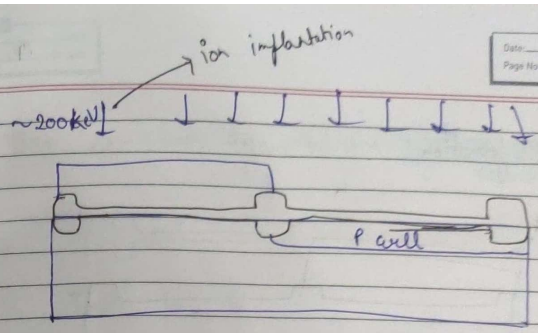


3) N well and P-well formation

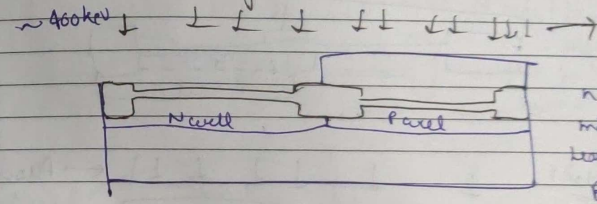


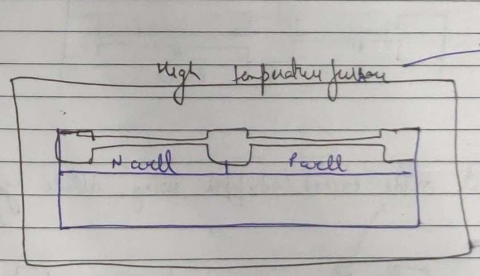
Mask 2

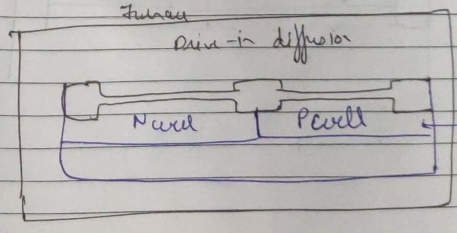




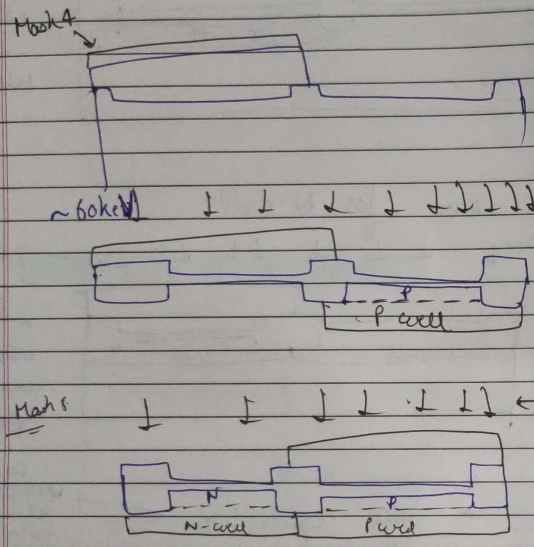
→ Do same for N-well



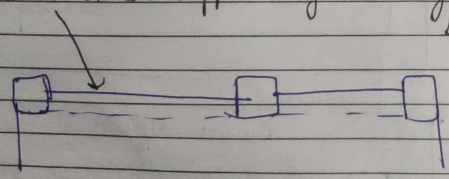




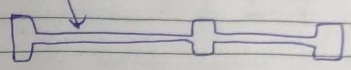
4) Gate formation



→ original oxide etched/stripped using dilute hydrofluoric (HF) solution

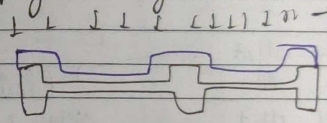


→ Then re-grown again to give high quality oxide (~10nm) thin



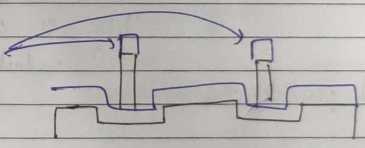
This is

→ ~0.4um polysilicon layer

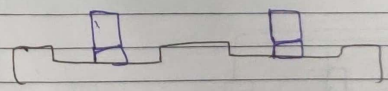


implant for low gate resistance

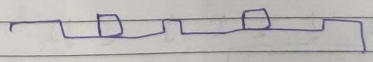
→ Mask 6



→



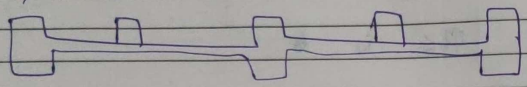
→



5. Lightly doped drain (LDD) formation

P+, P-, N

N+, N-, Y



2) Reason for this

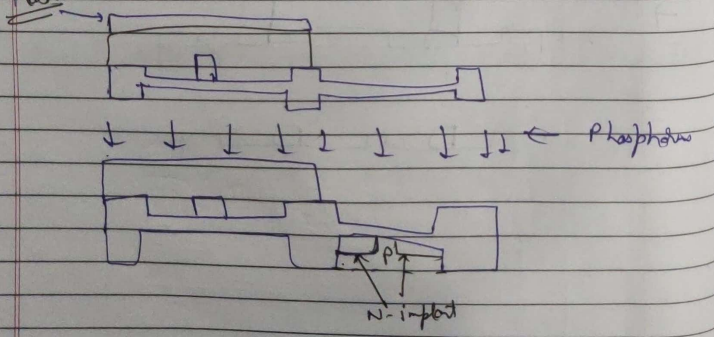
- Hot electron effect → Electric field = E = V/d
- Short channel effect

→ High energy carrier break Si Si bond 3.2eV barrier b/w Si conduction band SiO_2 conduction band

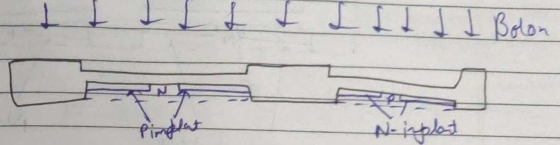
↓

drain field penetrates channel

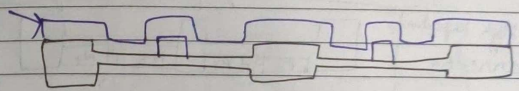
Mask 7



Mask 8

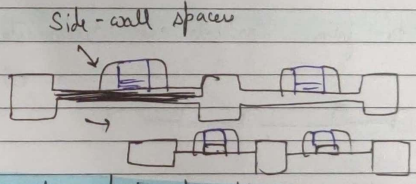


~0.1um Si_3N_4 or SiO_2

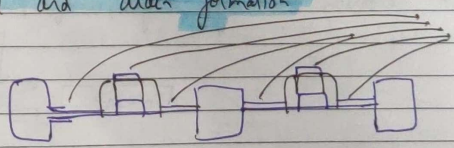


Plasma anisotropic etching

Side-wall spacer

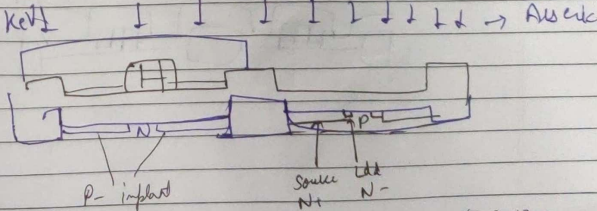


6) Source and drain formation



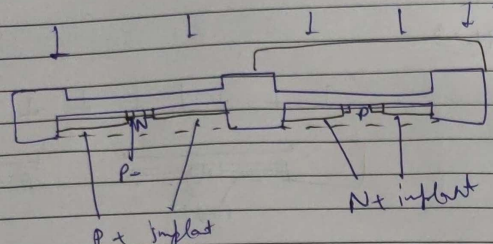
Thin screen oxide to avoid channeling during implant

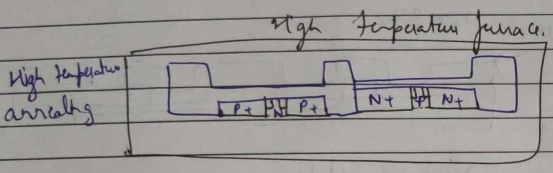
Mask 9 75 KeV



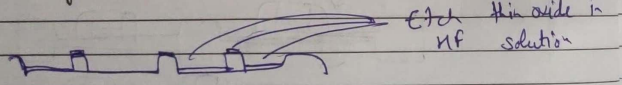
Date: Page No.: 93

Mask 10

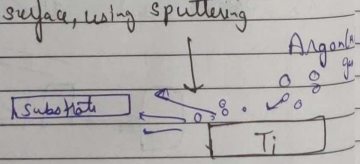


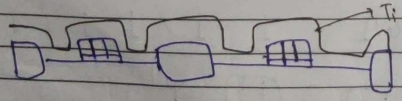


7) Steps to form contacts and interconnects (locals)



→ deposit titanium on wafer surface, using sputtering



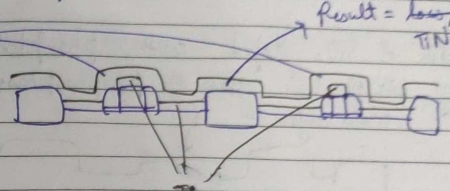


Date: Page No.: 94

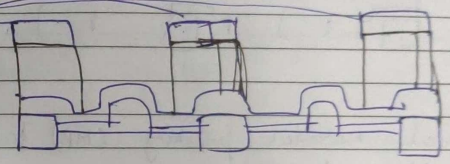
→ Wafer heated at about 650-700°C in N_2 oblect for 60sec

Result = ~~low resistant Si~~ TiN used only for local communication

Result = low resistant $TiSi_2$



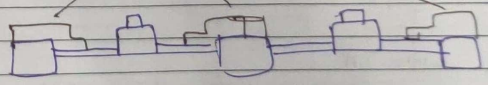
Mask 11



→ TiN is etched using RCA cleaning.

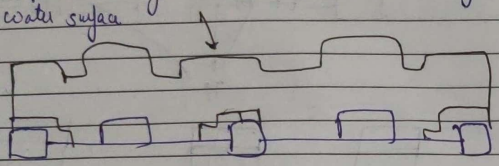
→ A solution with
- De-ionized water (H_2O) 5 part
- Ammonium hydroxide (NH_4OH), 1 part
- Hydrogen peroxide (H_2O_2), 1 part

TiN Local interconnects

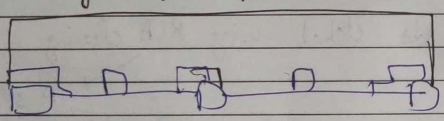


8) Higher Level metal formation

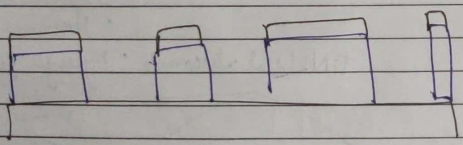
→ Layer of SiO_2 doped with phosphorous or boron (known as phosphosilicate glass or borophosphosilicate glass) deposited on wafer surface



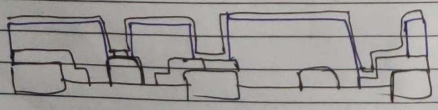
→ chemical mechanical polishing (CMP) technique for planarizing wafer surface



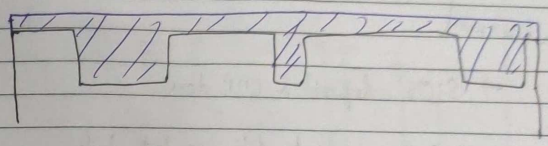
Mask 12



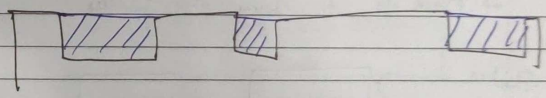
→ ~10nm TiN



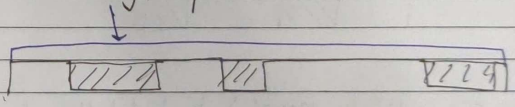
→ Blanket Tungsten (W) layer deposition



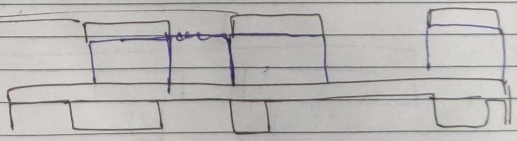
→ Again Chemical mechanical polishing (CMP) technique for planarizing wafer surface

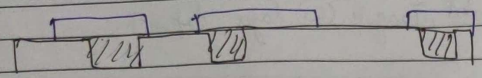


→ Aluminium (Al) layer deposition



Mask 13





for local metal

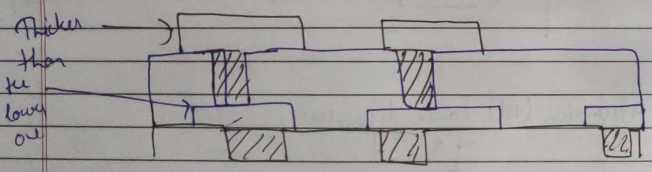
→ SiO_2 deposit & CMP done

→ Mask 14 used to define contact hole

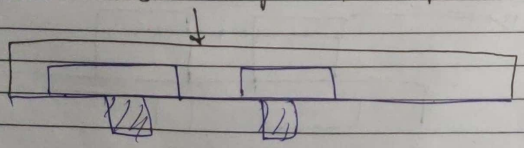
→ TiN deposited

→ Tungsten as contacts

→ Mask 15 to define the pattern

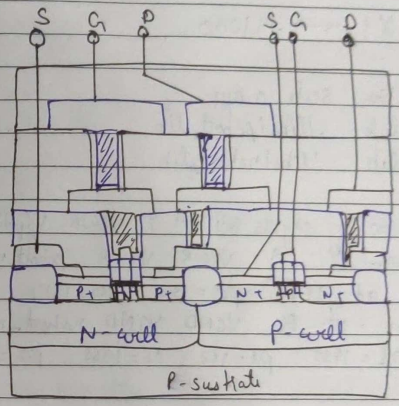


→ Top dielectric (Si_3N_4) to protect the chip



→ Final mask 16 to open contact holes on this layer

Fabricate



→ Labs → open the magic from last lab

S → select

What

3 time S → 2 times - connectivity

in magic

→ pcod

→ extract all

→ ext2spice

→ ls - lm

→ ls - lm

→ vim sky130_inv.spice

To exit
→ Esc
→ :wq
→ exit

Date: Page No.: 49

change spice

X0 → M1001

X1 → M1000

```
.option scale 0.01u
.include ./libs/pshort.lib
.include ./libs/nshort.lib

//.subckt sky130_inv A Y VPWR VGND
M1000 Y A VPWR VPWR pshort_model.0 w=37 l=23
+ ad=1443 pd=152 as=1517 ps=156
M1001 Y A VGND VGND nshort_model.0 w=35 l=23
+ ad=1435 pd=152 as=1365 ps=148

VDD VPWR 0 3.3V
VSS VGND 0 0V

Va A VGND PULSE (0V 3.3V 0 0.1ns 0.1ns 2ns 4ns)
C0 A Y 0.05fF
C1 Y VPWR 0.11fF
C2 A VPWR 0.07fF
C3 Y 0 0.24fF
C4 VPWR 0 0.59fF
//C5 VPWR VGND 0.000017fF
//.ends
.tran 1n 20n

.control
run
.subc
.end
```

✓The Good Paper

Date: Page No.: 50

→ cd opensram/sudotd cells/sky...
→ sudo apt install ngspice (only once)
→ ngspice sky130_inv.spice
→ plot y vs time a
→ exit

→ vim sky130_inv.spice

change in C3

```
C3 Y 0 2fF
```

exit vim

→ ngspice sky130_inv.spice
→ plot y vs time a

- Rise time

x0 = 2.18189e-09, y0 = 0.66
x0 = 2.24444e-09, y0 = 2.64

Rise time = (2.24444 − 2.18189)e-09
= 0.06255

fall time
x0 = 4.04839e-09, y0 = 2.64
x0 = 4.09328e-09, y0 = 0.66

✓The Good Paper

Page No. 51

propagation delay

$x_0 = 2.147580e-09$, $y_0 = 1.65$
$x_0 = 2.20806e-09$, $y_0 = 1.65$

cell fall delay

$x_0 = 4.04906e-09$, $y_0 = 1.65$
$x_0 = 4.07736e-09$, $y_0 = 1.65$

Magic tools & DRC Rules
→

Documentation :- http://opencircuitdesign.com/magic

Magic DRC => http://opencircuitdesign.com/magic/Technology files/The Magic Technology File Manual/Drc Section

Link to Google Skywaters Design Rules :- https://skywater-pdk.readthedocs.io/en/main/rules/periphery.html

For reference :- https://github.com/google/skywater-pdk

To download :→ wget http://opencircuitdesign.com/open_pdks/archive/drc-tests.tgz

To extract the lab files from the downloaded file :-
→ tar xfz drc-tests.tgz

Page No. 52

→ cd drc_tests

→ ls -al {list all the directories}

→ magic -d XR & {Start magic tools with better graphics}

→ vi .magicrc {Content of .magicrc file}

In magic

→ Click file

→ Click open

→ click met3.mag

→ Then click open.

→ drc cutting → Then make any box
Hover to any colour
→ Press P {to choose colour}

→ Press : {to go to tkcon window indirectly}

→ cif see VIA2
→ Load poly.mag

→ Vim Sky130A.tech

Changes → poly.9 → allpolynonres {after making copy}
↓
see SS

Date: Page No.: 83

In magic

~~load polygon~~

tech load sky130A.tech

Make a bolt press 'a'

go to empty area press 'c' to copy and past

→ open ncvell.mag

→ {change in sky130A.tech}

→ tech load sky130A.tech

→ drc. style drc(full).

Day 4

magic -T sky130A.tech sky130_inv.mag &

In magic

cd openlane working dir/
pdks/sky130A/libs.tech/
lopenlane/sky130_fd_sc

→ less tracks.info

4 → grid

Ttcon → help grid

→ grid 0.46um 0.34um 0.23um 0.17um

To create lef file → Text edit → change save sky130_vsdinv.mag

→ open this

→ copy with

cp → copy

To copy this lef to picorv32a

In lef file directory → vsdstd cell design

→ cp sky130_vsdinv.lef /home/vsduser/Desktop/work/Tools/openlane_working_dir/openlane/designs/picorv32a/src

→ cd libs

→ cp sky130_fd_sc_hd__* /home/ _ _ picorv32a/src

→ cd ../../
now in openlane directory

→ docker → ./flow.tcl -interactive

→ package require openlane 0.9

→ prep -design picorv32a -tag 17_07_09_51 -overwrite

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

→ add_lefs -src $lefs

→ run_synthesis

→ cd openlane designs/picorv32a

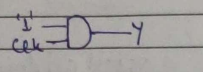
→ Vim config.tcl

→ change it

✓The Good Paper

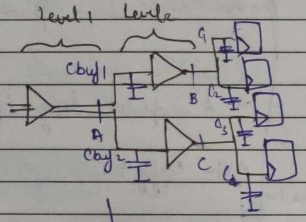
✓The Good Paper

Delay Tables



EN	Clk	Y
0	0	0
0	1	0
1	0	0
1	1	1

EN	Clk	Y
0	0	0
0	1	1
1	0	1
1	1	1



Let us assume $C_1 = C_2 = C_3 = C_4 = 25fF$

How about creating a buffer tree at node 'A'

→ Let us assume $C_{buf1} = C_{buf2} = 30fF$

Therefore, total Cap at node 'A' => 60fF

Therefore, total cap at node 'B' => 50fF

Therefore, total cap at node 'C' => 80fF

↓

Observations.

- 2 levels of buffering
- At every level, each node driven same load
- Identical buffer at same level

ps → picosecond

ff → femtofarad

→ Delay Table for CBUF '1'

Input slew \ Output load	10fF	30fF	50fF	70fF	90fF	110fF
20ps	X1	X2	X3	X4	X5	X6
40ps	X7	X8	X9	X10	X11	X12
60ps	X13	X14	X15	X16	X17	X18
80ps	X19	X20	X21	X22	X23	X24

→ Delay Table for CBUF '2'

Input slew \ Output load	10fF	30fF	50fF	70fF	90fF	110fF
20ps	Y1	Y2	Y3	Y4	Y5	Y6
40	Y7	Y8	Y9	Y10	Y11	Y12
60	Y13	Y14	Y15	Y16	Y17	Y18
80ps	Y19	Y20	Y21	Y22	Y23	Y24

Ex :- Lets input transition = 40ps → C_{buf}'1'

output capacitance = 80fF

	30fF	70fF
40ps	X9	X10

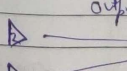
delay = X9'

for C_{buf}'2'

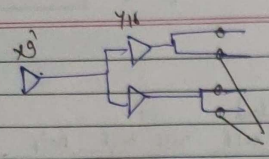
output for C_{buf}'1' = 60fF

input transition for C_{buf}'2' = 60ps

output capacitance for C_{buf}'2' = 80fF



delay = Y15



Continuing from last lab

→ cd openlane/configuration

→ less README.md

In openlane terminal.

→ echo $::env(SYNTH_STRATEGY)

→ set ::env(SYNTH_STRATEGY) 1

→ set ::env(SYNTH_STRATEGY) "DELAY 3"

→ echo $::env(SYNTH_BUFFERING)

→ echo $::env(SYNTH_SIZING)

→ set ::env(SYNTH_SIZING) 1

→ echo $::env(SYNTH_~~BUFFERING~~ DRIVING_CELL)

→ run_synthesis

→ run_floorplan ↳ gives error

To do these changes scan first of all cat will check the value of slack and dip area or not if down then restart the docker and perform all these steps and then check the value of slack and area

slack ↓
area ↑

✓The Good Paper

→ init_floorplan

→ place_io

→ tap_decap_01

→ These command are all together sourced instead of "run_floorplan" command

→ run_floorplan doesn't work directly with custom cells, so we divide it into 3 sub commands

→ run_placement

→ cd /openlane/designs/.../runs/.../results/placement

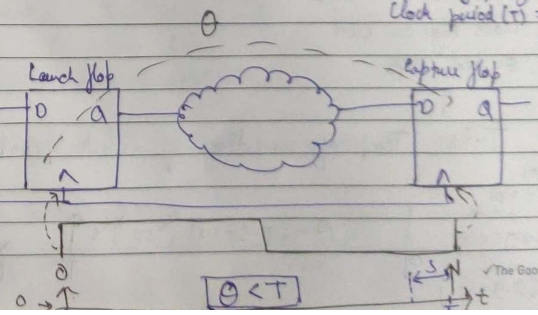
→ magic -T ../../tmp/merged.lef def read ../../results/placement/picorv32a.placement.def

→ In magic
→ expand

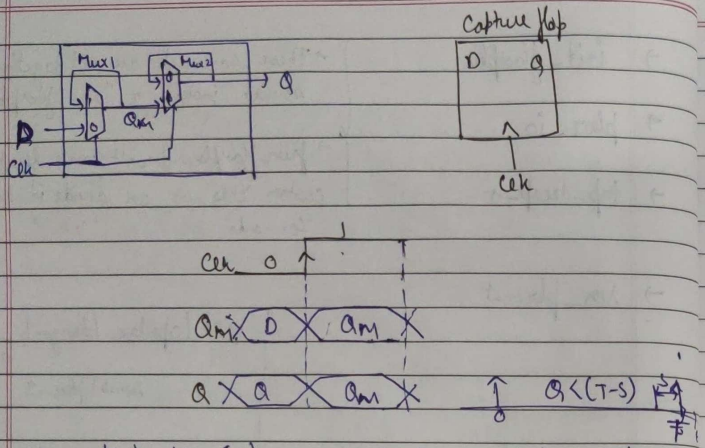
Timing Analysis (with ideal clock)

Setup Analysis - Single Clock

Specifications :-
clock frequency (f) = 1 GHz
clock period (T) = 1/f
= 1/1 GHz
= 1 ns

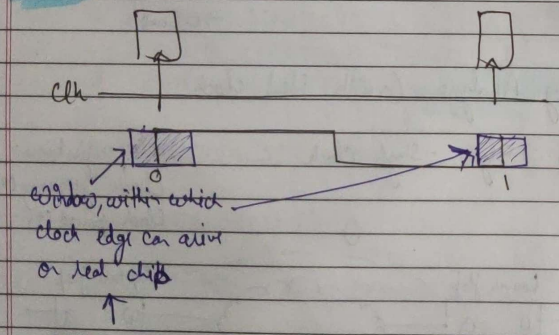


✓The Good Paper

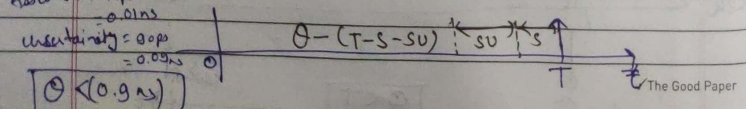


Hence finite time 'S' required (before Clk edge) for 'D' to reach Qm i.e. internal delay of Mux1 = Setup time.

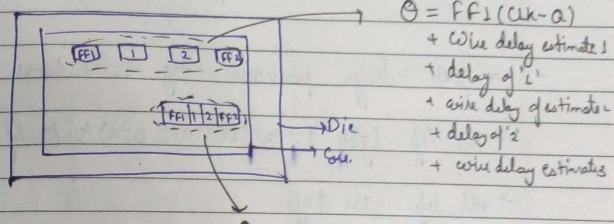
Jitter



Jitter:- Temporary variation of clock period



from the last example



Θ = FF1(Clk-Q)
+ delay of '1'
+ delay of '2'

Θ < 1ns - 0.01ns - 0.09ns
< 0.9ns (expected)

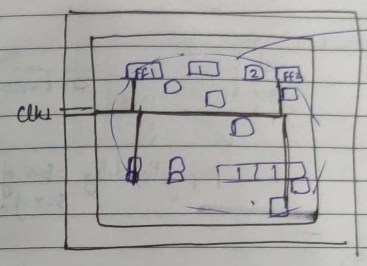
Lab steps to ~~optimize synthesis to reduce setup violation~~ configure OpenSTA for post-synth timing analysis

→ In cd openlane
→ touch pre_sta.conf (Create file)
→ add content (as in ss)
→ cd openlane/designs/picorv32a/src
→ touch my_base.sdc (Create file)
→ add content (as in ss)
→ cd openlane
→ sta pre_sta.conf

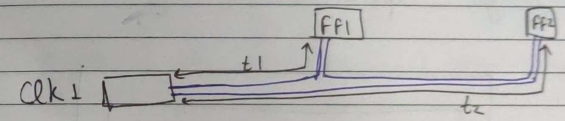
Lab steps to optimize synthesis to reduce setup violation

- → prep -design picorv32a -tag 17-07_09-51 -overwrite
- → set lefs [glob $::env(DESIGN_DIR)/src/*.lef]
- → add_lefs -src $lefs
- → set ::env(SYNTH_SIZING) 1
- → set ::env(SYNTH_MAX_FANOUT) 4
- → echo $::env(SYNTH_DRIVING_CELL)
- → run_synthesis
  - → cd openlane
  - → sta pre_sta.conf
    - → report_net -connections _11672_
    - → help replace_cell
    - → replace_cell _14510_ sky130_fd_sc_hd__o32a_4
    - → report_checks -fields {net cap slew input_pins} -digits 4
    - → report_net -connections _11672_
    - → replace_cell _14510_ sky130_fd_sc_hd__

Clock tree Synthesis



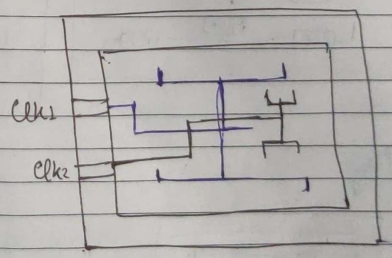
→ Bad tree.

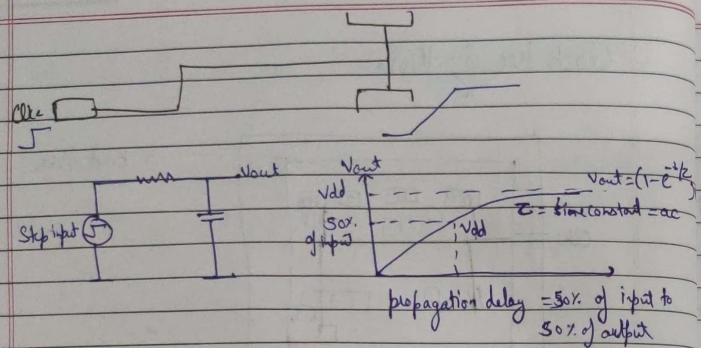


$t_2 - t_1 = skew$

skew should be ≈ 0ps

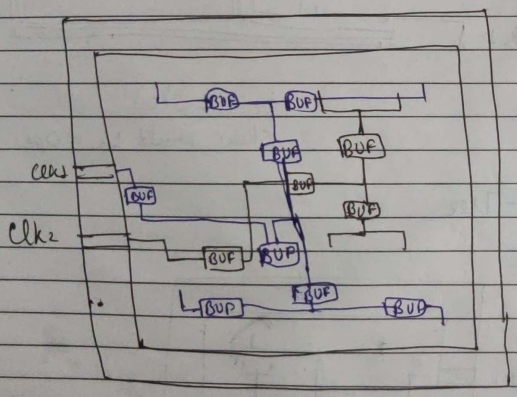
H-Tree



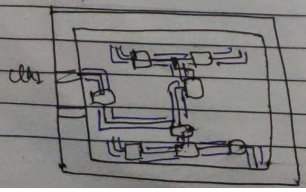


propagation delay = 50% of input to 50% of output

Buffering – repeater

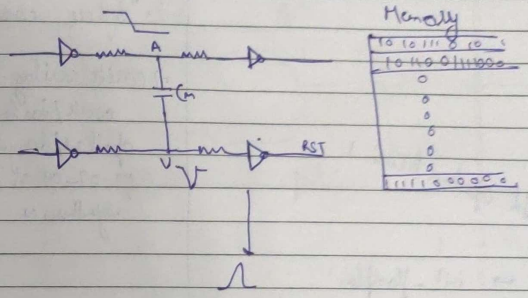


Clock net shielding

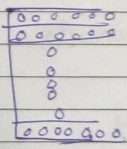


✓The Good Paper

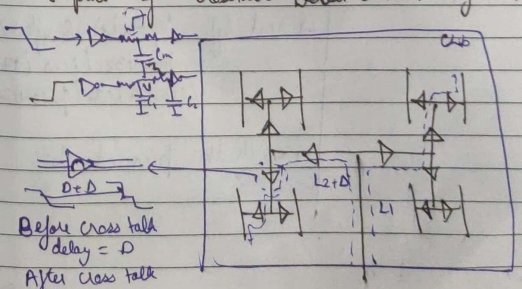
What if can go wrong, if there's a glitch??



→ incorrect data in memory will result in inaccurate functionality.



→ Impact of Crosstalk ~~Delta~~ Delta Delay - skew



Before cross talk delay = D

After cross talk Delay = D + Δ

L1 = L2 + Δ

Skew = L1 − (L2 + Δ) = Δ

✓The Good Paper

→ Lab steps to run CTS using Triton CTS

In terminal
where we run sta pre_sta.sdc

In openlane terminal

→ write_verilog /home/vsduser/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/17-07_09-51/results/synthesis/picorv32a.synthesis.v

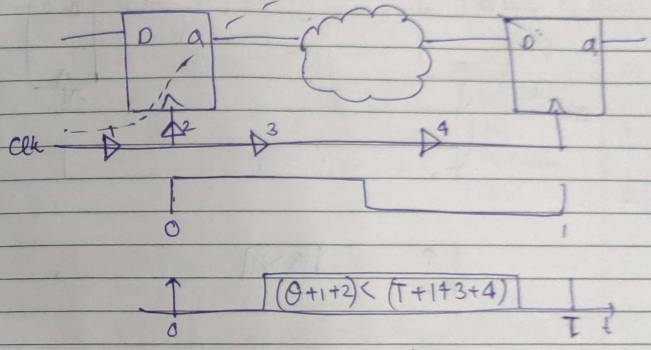
→ init_floorplan
→ place_io
→ tap_decap_or
→ run_placement

→ run_cts

→ cd /openlane_working_dir/openlane/scripts/openroad
→ openroad
→ read_lef /openLANE_flow/designs/picorv32a/runs/17-07_09-51/tmp/merged.lef
→ read_def /openLANE_flow/designs/picorv32a/runs/17-07_09-51/results/cts/picorv32a.cts.def
→ write_db pico_cts.db

Timing analysis (with real clock)

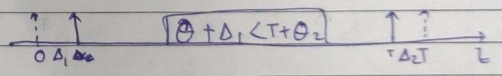
setup analysis - single clock



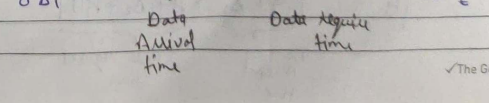
1+2 → $\theta\Delta_1$ → launch flop delay

1+3+4 → Δ_2

$|\Delta_1 - \Delta_2|$ = Δskew

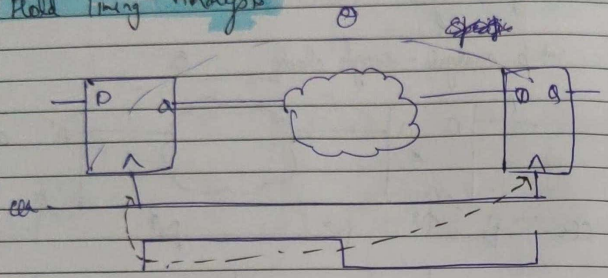


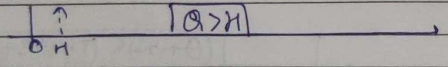
With T = 1ns
Assume S = 10ps
= 0.01ns
Uncertainty = 90ps
= 0.09ns



Data Required Time - Data Arrival time
Slack (should be +ve or 0)

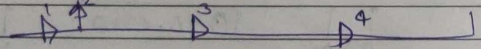
Hold Timing Analysis

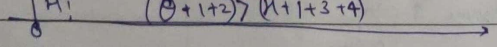




→ from the detailed analysis of capture flop

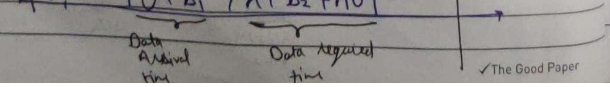
→ finite time 'H' required (after clk edge) for 'Qm' to reach Q_1 is internal delay of MUX2 = hold time

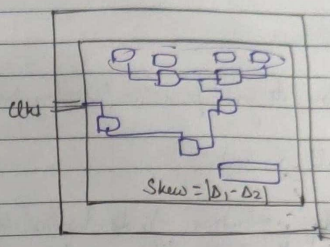




$$(\Theta + \Delta_1) > (H + \Delta_2)$$

With T = 1ns
Assume H = 10ps = 0.01ns
Uncertainty = 50ps = 0.05ns





Δ_1 = Real wire RC delay1
+ BUF delay
+ Real wire RC delay2
+ BUF delay
+ Real _ _ _ 3
+ BUF delay
+ Real _ _ _ 9
+ BUF delay
+ Real _ _ _ 5

Δ_2 = Real wire RC delay1
+ BUF delay
+ Real _ _ _ _ 2
+ BUF delay
+ Real _ _ _ _ 8
+ BUF delay
+ Real _ _ _ _ _ 4'
+ BUF delay
+ Real _ _ _ _ 5'

→ Lab setup to analyze timing with real clock using OpenSTA

After writing db

→ read_db pico_cts.db

→ read_verilog /openLANE_flow/designs/picorv32a/runs/17-07_09-51/results/synthesis/picorv32a.synthesis_cts.v

→ read_liberty $::env(LIB_SYNTH_COMPLETE)

→ link_design picorv32a

→ read_sdc /openLANE_flow/designs/picorv32a/src/my_base.sdc

→ set_propagated_clock [all_clocks]

→ report_checks -path_delay min_max -fields {slew trans net cap input_pins} -format full_clock_expanded -digits 4

→ exit {exit openroad}

→ echo $::env(CTS_CLK_BUFFER_LIST)

→ set ::env(CTS_CLK_BUFFER_LIST) [lreplace $::env(CTS_CLK_BUFFER_LIST) 0 0]

→ echo $::env(CTS_CLK_BUFFER_LIST)

→ echo $::env(CURRENT_DEF)

→ set ::env(CURRENT_DEF) /openLANE_flow/designs/picorv32a/runs/17-07_09-51/results/placement/picorv32a.placement.def

→ run_cts

→ echo $::env(CTS_CLK_BUFFER_LIST)

→ openroad

→ read lef /openLANE_flow/.../17-07_09-51/tmp/merged

→ read_def /openLANE_flow/.../17-07_09-51/results/cts/picorv32a.cts.def

→ write_db pico_cts1.db

→ read_db pico_cts1.db

→ read_verilog /openlane/.../results/synthesis/picorv32a.synthesis.cts.v

→ read_liberty $::env(LIB_SYNTH_COMPLETE)

→ link_design picorv32a

→ report_clock_skew -hold

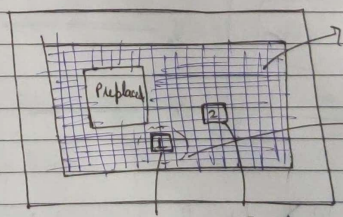
→ report_clock_skew -setup

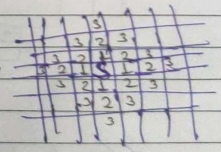
→ exit.

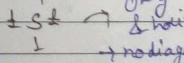
Day 8

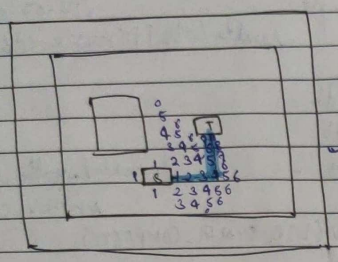
→ Introduction to Maze Routing – Lee's algorithm.

Routing :- Connecting 2 point, best path, best possible solution

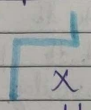




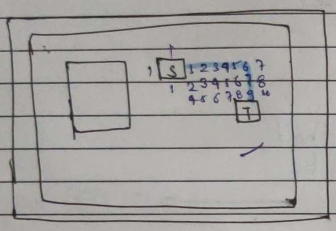




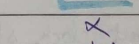
option 2



X
not preferred



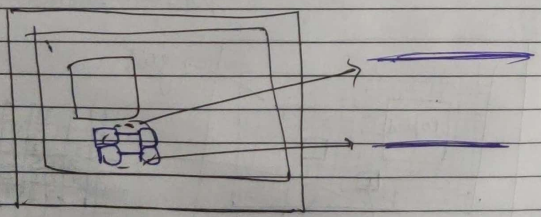
option 2



✗
not preferred
more no. of bends

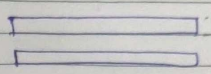
→ complete routing of our example is in ScreenShots

DRC Clear (Design Rule Check)

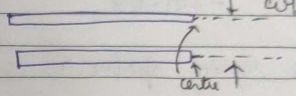


→ 3 Typical design Rules for the above pair of wires

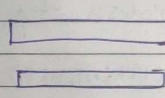
1) Wire width

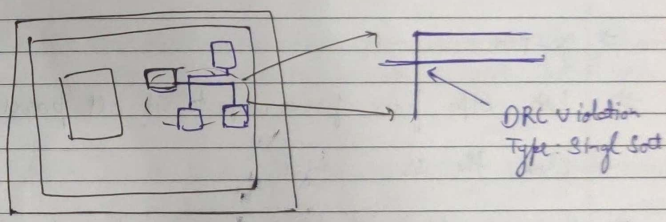


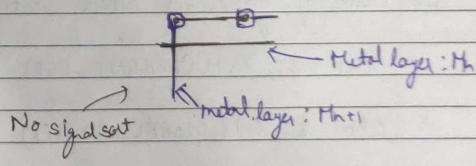
2) Wire pitch



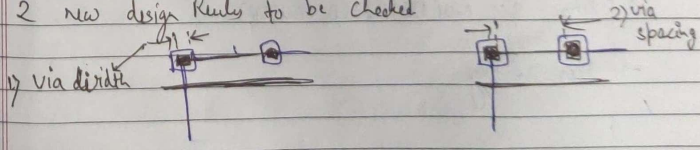
3) Wire spacing







2 new design Rules to be checked



→ After routing & DRC next step is Parasitic extraction
→ Resistance and capacitance present on every wire should be extracted and use for further process (shown in SS)

Lab steps to build power distribution network

→ docker

→ ./flow.tcl -interactive

→ package require openlane 0.9

→ prep -design picorv32a -tag 17_07_09_51

→ echo $::env(CURRENT_DEF)

→ gen_pdn

Lab steps from power straps to std cell power.

→ check the ss

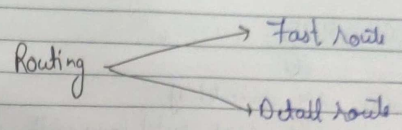
Basic of global and detail routing and configure TritonRoute

→ echo $::env(CURRENT_DEF)

→ echo $::env(GLOBAL_ROUTER)

→ echo $::env(DETAILED_ROUTER)

→ run_routing



Triton features :-

→ Pre-processed route guides

→ perform initial detailed route

→ Honors the preprocessed route guides (obtained after fast route) i.e., attempts as much as possible to route within route guides

→ Assumes route guides for each net satisfy inter-guide connectivity

→ Work on proposed MILP-based panel routing scheme with intra-layer parallel and inter-layer sequential routing framework

Requirements of preprocessed Guides :-

- Should have unit width
- Should be in the preferred direction.

Preprocessing :-

a) initial route guides
b) Splitting
c) merging
d) bridging
e) preprocessed guides

→ The preferred direction for M1 is vertical, and horizontal for M2.

Inter guide Connectivity

→ Two guides are connected if
  → They lie on the same metal layer with touching edges
  → They are on neighboring metal layers with a nonzero vertically overlapped area

→ Each unconnected terminal (i.e., pin of a standard-cell instance) should have its pin shape overlapped by a route guide

Intra-layer parallel and inter-layer sequential pair routing

Triton Route

→ 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

→ Access point (AP):- An on-guide 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 access points derived from same lower-layer segment, upper-layer guide, a pin or an IO port.

Routing Topology Algorithm

Algorithm 1 :- optimization of Routing Topology

```
for all i = 1 to n-1 do
  for all j = i+1 to n do
    cost_i,j ← dist(APC_i, APC_j)
  end for
end for
T ← MST(APCs, COSTs)
Return e_i,j ∈ T
```

→ cd openlane

→ magic -T /Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.tech/magic/sky130A.tech lef read /home/vsdiwsu/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/17-07_09-51/tmp/merged.lef def read /home/vsdiwsu/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/17-07_09-51/results/routing/picorv32a.def