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

Timing Modelling Using Delay Tables

Lab Steps To Convert Grid Information Into Track Information

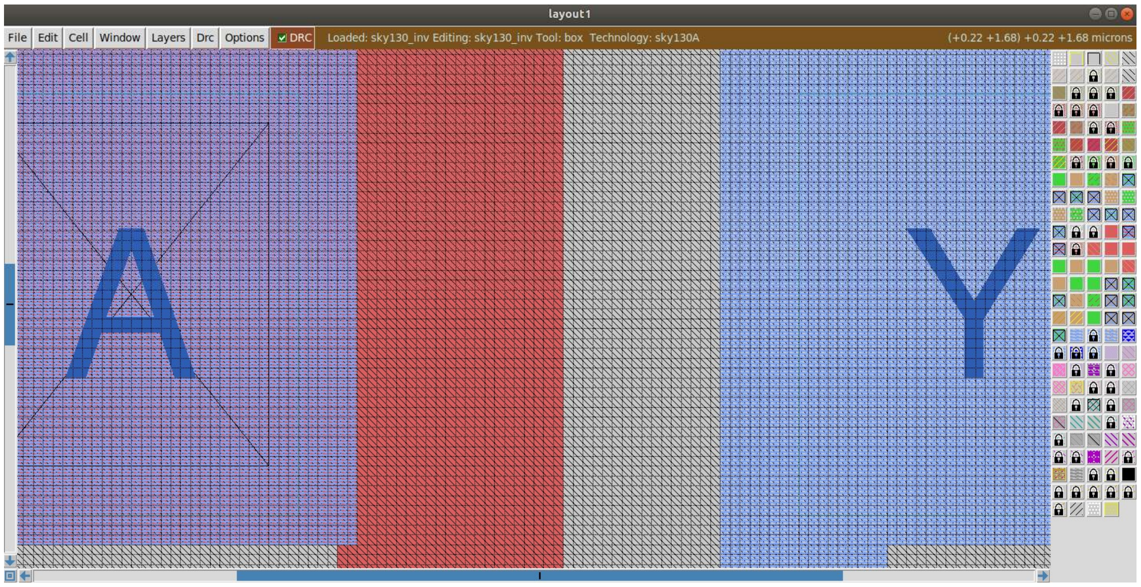
The sky130_inv.mag file in the vsdstdcelldesign directory contains information like PG, ports, and logic. Since OpenLANE is a PnR tool, it only needs boundary, power/ground rails, and input/output info, not the full .mag file data. Therefore, we extract the LEF file from the MAGIC file and integrate it into the picorv32a design.

The guidelines to be followed while making a standard cell are -:

1. Input and output ports should be at the intersection of horizontal and vertical tracks to ensure routes can connect to them.
2. The standard cell's width and height must be odd multiples of the track's horizontal and vertical pitch.

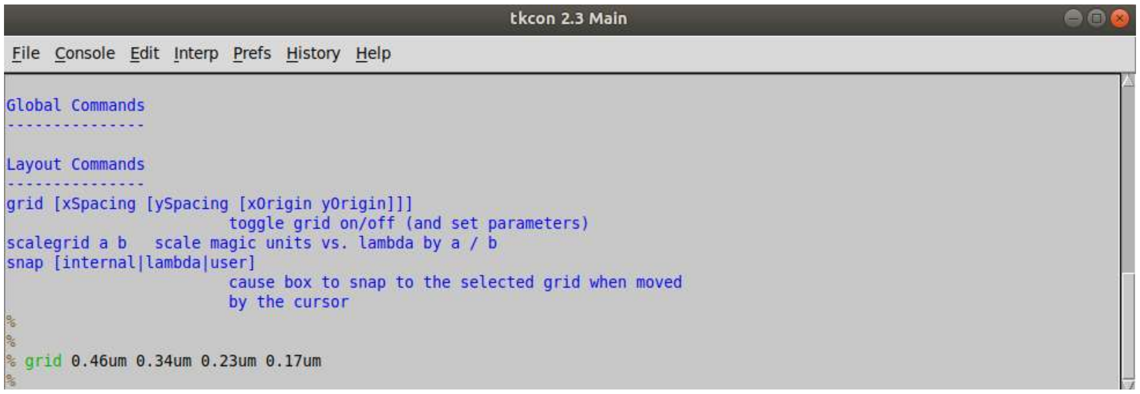
Tracks are the horizontal and vertical metal layers used for routing. Their intersection forms a routing grid, also known as a routing matrix.

The ~/Desktop/work/tools/openlane_working_dir/pdks/sky130A/libs.tech/openlane/sky130_fd_sc_hd/tracks.info contains track information. Before and After changing the grid values -:



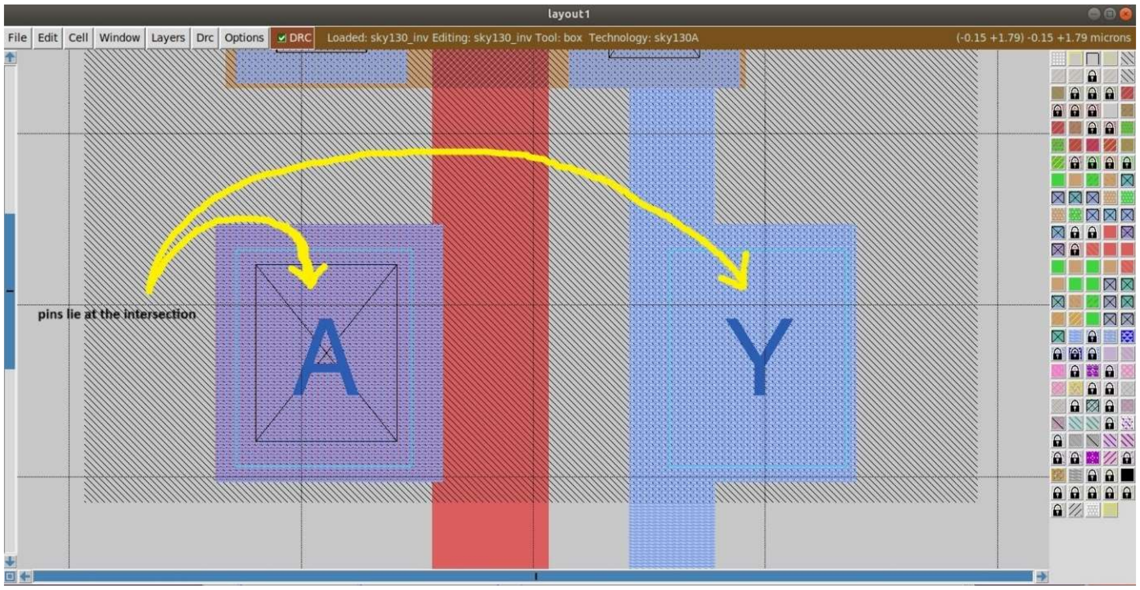
Loaded: sky130_inv Editing: sky130_inv Tool: box Technology: sky130A

(+0.22 +1.68) +0.22 +1.68 microns



```
Global Commands
---------------

Layout Commands
---------------
grid [xSpacing [ySpacing [xOrigin yOrigin]]]
                        toggle grid on/off (and set parameters)
scalegrid a b   scale magic units vs. lambda by a / b
snap [internal|lambda|user]
                        cause box to snap to the selected grid when moved
                        by the cursor
%
%
% grid 0.46um 0.34um 0.23um 0.17um
%
```

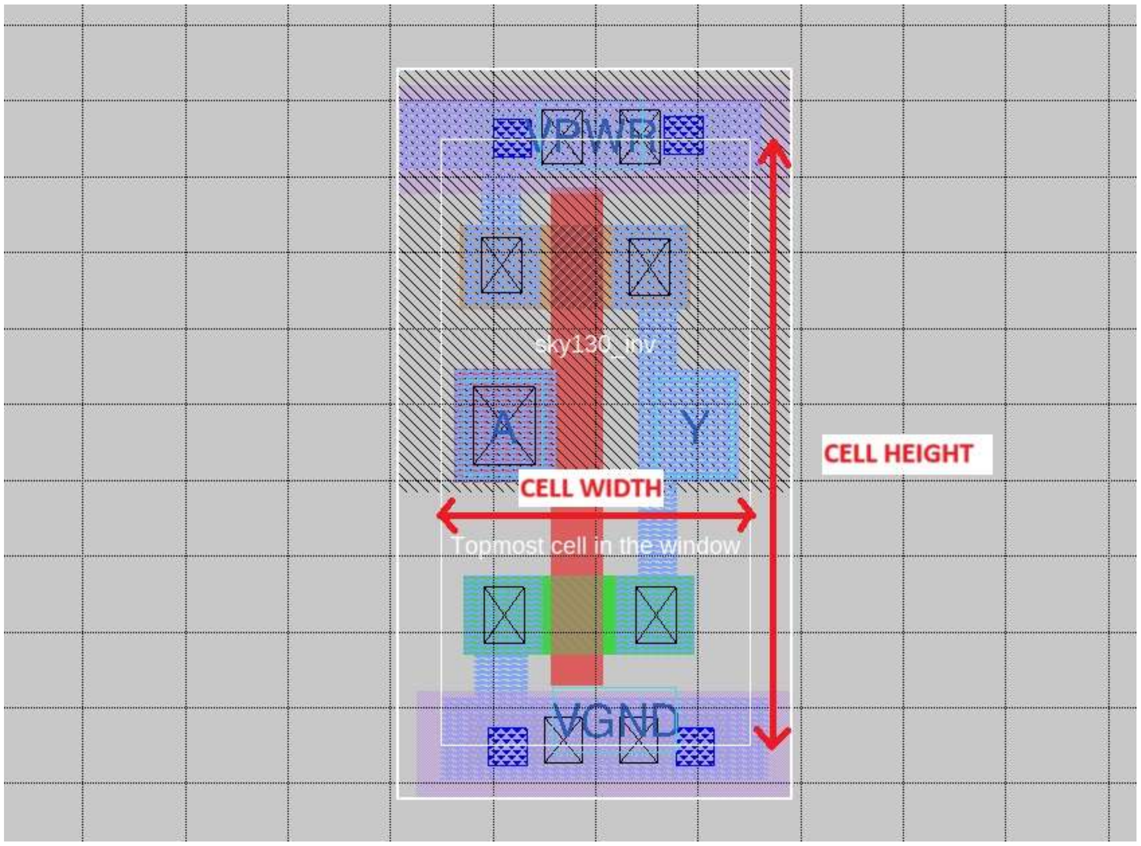
Hence, we are able to satisfy the first guideline.



```
Loaded: sky130_inv Editing: sky130_inv Tool: box  Technology: sky130A
(-0.15 +1.79) -0.15 +1.79 microns
```

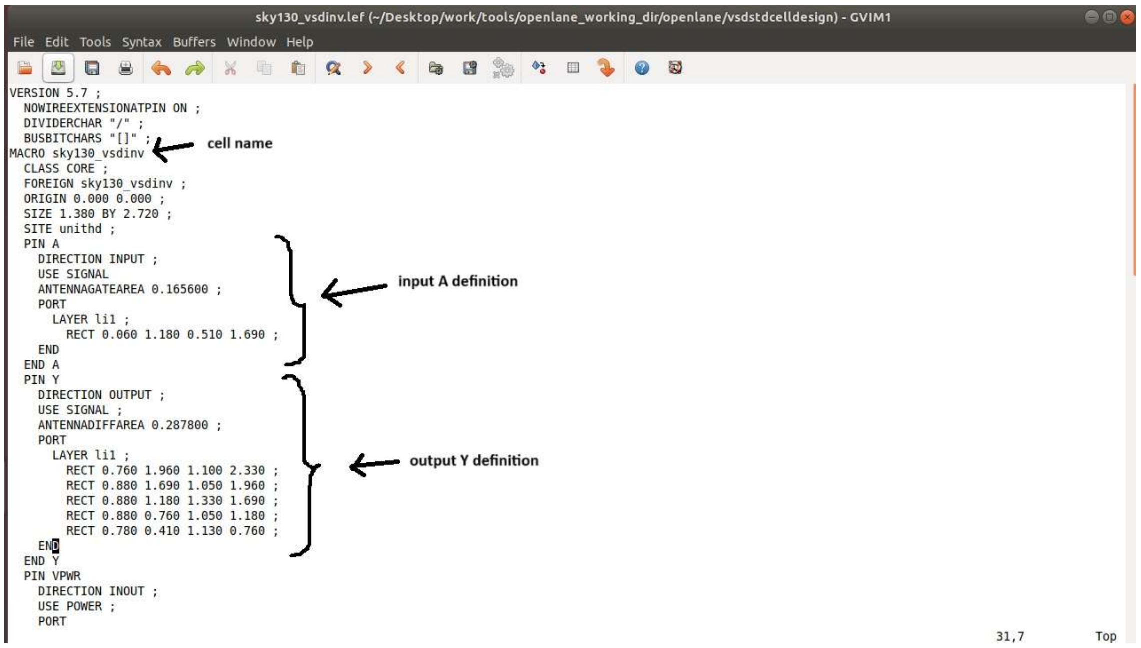
layout1
File Edit Cell Window Layers Drc Options DRC
pins lie at the intersection
A
Y

Then, we are able to satisfy the second guideline as follows -:



Lab Steps to Convert Magic Layout to STD Cell LEF: LEF (Library Exchange Format) contains details of the standard cell library, such as geometric shapes, sizes, layers, and physical properties of cells or macros. Instructions for setting port definitions are available here.

Next, save the *.mag* file with a new filename by typing ***lef write*** in the tkcon terminal, which will generate a new **lef** file with the new filename -:

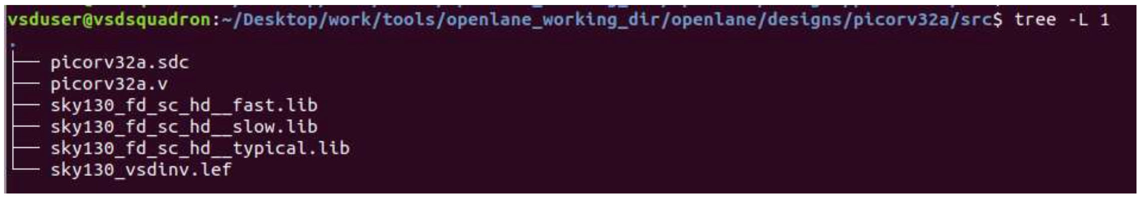

sky130_vsdinv.lef (~/Desktop/work/tools/openlane_working_dir/openlane/vsdstdcelldesign) - GVIM1
File Edit Tools Syntax Buffers Window Help
cell name
input A definition
output Y definition
31,7 Top

```
VERSION 5.7 ;
  NOWIREEXTENSIONATPIN ON ;
  DIVIDERCHAR "/" ;
  BUSBITCHARS "[]" ;
MACRO sky130_vsdinv
  CLASS CORE ;
  FOREIGN sky130_vsdinv ;
  ORIGIN 0.000 0.000 ;
  SIZE 1.380 BY 2.720 ;
  SITE unithd ;
  PIN A
    DIRECTION INPUT ;
    USE SIGNAL ;
    ANTENNAGATEAREA 0.165600 ;
    PORT
      LAYER li1 ;
        RECT 0.060 1.180 0.510 1.690 ;
    END
  END A
  PIN Y
    DIRECTION OUTPUT ;
    USE SIGNAL ;
    ANTENNADIFFAREA 0.287800 ;
    PORT
      LAYER li1 ;
        RECT 0.760 1.960 1.100 2.330 ;
        RECT 0.880 1.690 1.050 1.960 ;
        RECT 0.880 1.180 1.330 1.690 ;
        RECT 0.880 0.760 1.050 1.180 ;
        RECT 0.780 0.410 1.130 0.760 ;
    END
  END Y
  PIN VPWR
    DIRECTION INOUT ;
    USE POWER ;
    PORT
```

Introduction of Timing libs and Steps To Include New Cell in Synthesis

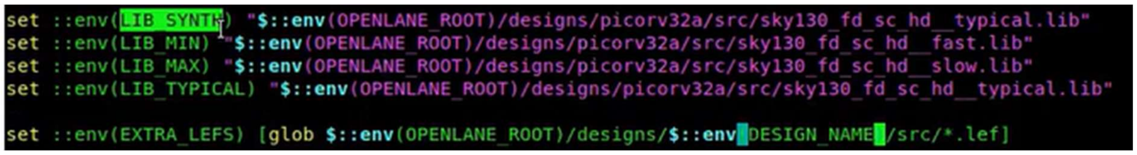
In the directory [pdks/sky130A/libs.ref/sky130_fd_sc_hd/lib/] are the liberty timing files for SKY130 PDK, containing timing and power parameters for each cell needed in STA. These files represent different PVT corners (slow, typical, fast) with varying supply voltages (1.8V, 1.65V, 1.95V). For example, the library sky130_fd_sc_hd__ss_025C_1v80 describes a slow-slow corner at 25°C and 1.8V. Timing and power data are obtained by simulating cells under different conditions, and this information is stored in the liberty file, which is used during ABC mapping in synthesis to map generic cells to actual standard cells.

- Firstly, copy the extracted lef file - named ***sky130_vsdinv.lef*** and the liberty files named *sky130.lib** from this repository - ***/openlane/vsdstdcelldesign/libs*** to the **src** directory of **picorv32a**.



```
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/src$ tree -L 1
.
├── picorv32a.sdc
├── picorv32a.v
├── sky130_fd_sc_hd__fast.lib
├── sky130_fd_sc_hd__slow.lib
├── sky130_fd_sc_hd__typical.lib
└── sky130_vsdinv.lef
```

-
- Add the following to the config.tcl file inside the picorv32a directory to set the liberty file for ABC mapping (LIB_SYNTH) and STA (_FASTEST, _SLOWEST, _TYPICAL), as well as the extra LEF files (EXTRA_LEFS) for the customized inverter cell.



```
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]
```

- After this, invoke the ***docker*** command and prepare the ***picorv32a*** design. Plug the new *lef* file to the OpenLANE flow through the following commands

docker

./flow.tcl -interactive

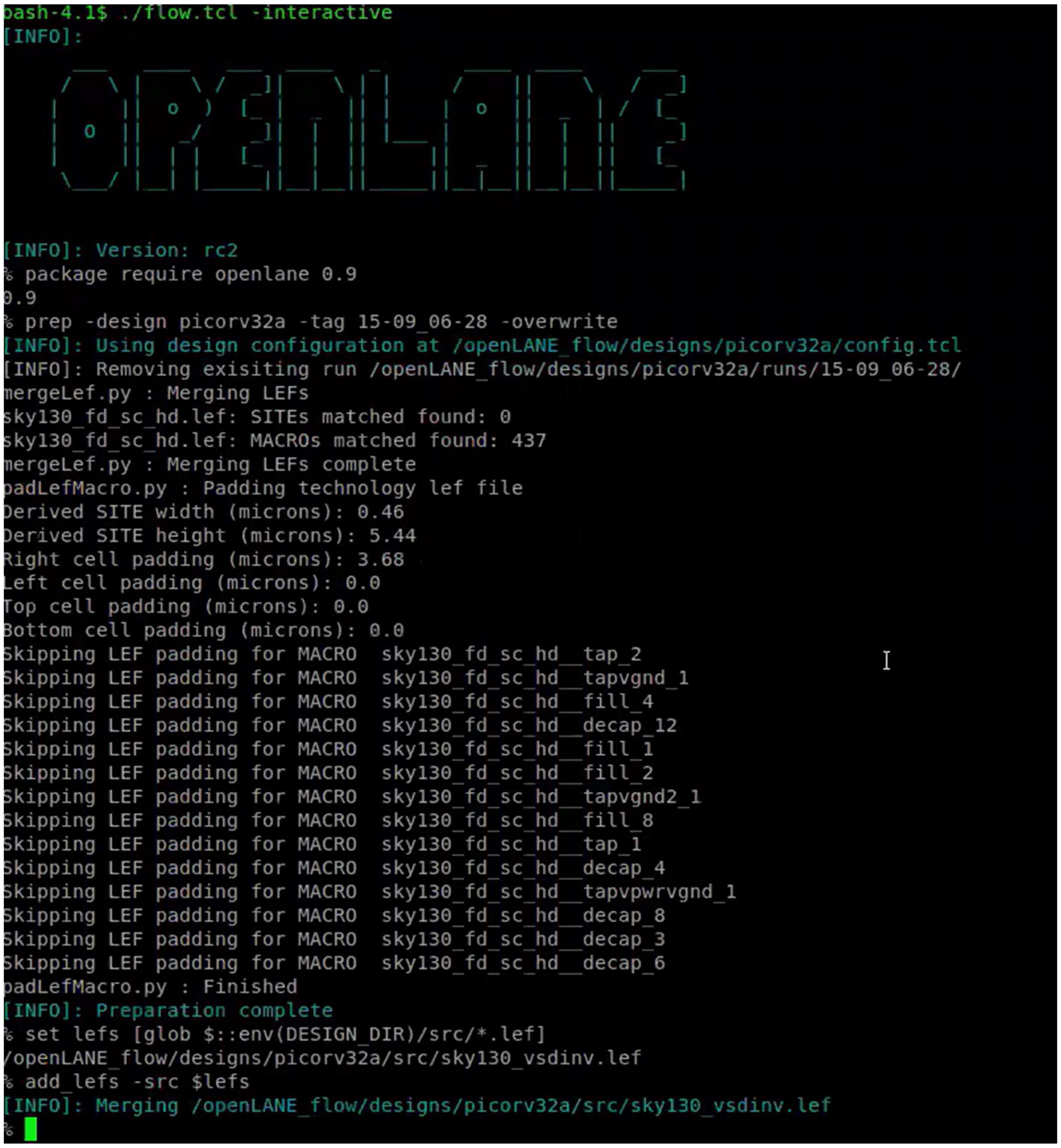
package require openlane 0.9

prep -design picorv32a

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

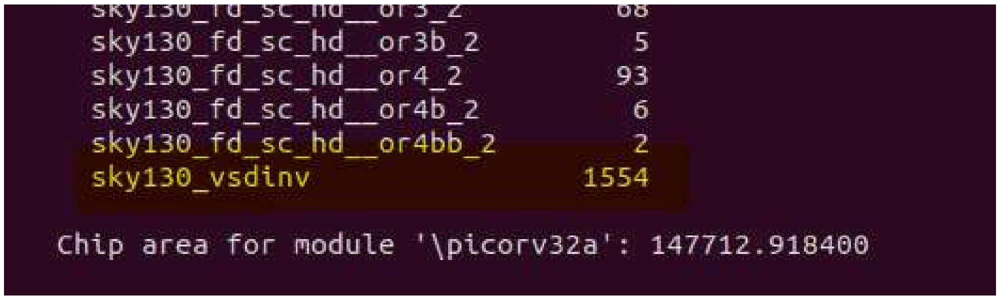
add_lefs -src $lefs

This will generate the following picture -:



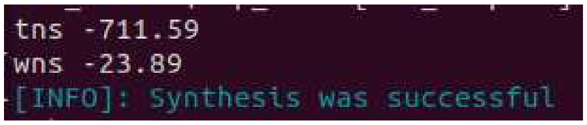
```
bash-4.1$ ./flow.tcl -interactive
[INFO]:
    ___   ____   ___  ____   _       ____  ____    ___
   /   \ |    \ /  _]|    \ | |     /    ||    \  /  _]
  |     ||  o  )  [_ |  _  || |    |  o  ||  _  |/  [_
  |  O  ||   _/    _]|  |  || |___ |     ||  |  |    _]
  |     ||  | |   [_ |  |  ||     ||  _  ||  |  |   [_
   \___/ |__| |_____||__|__||_____||__|__||__|__||_____|


[INFO]: Version: rc2
% package require openlane 0.9
0.9
% prep -design picorv32a -tag 15-09_06-28 -overwrite
[INFO]: Using design configuration at /openLANE_flow/designs/picorv32a/config.tcl
[INFO]: Removing exisiting run /openLANE_flow/designs/picorv32a/runs/15-09_06-28/
mergeLef.py : Merging LEFs
sky130_fd_sc_hd.lef: SITEs matched found: 0
sky130_fd_sc_hd.lef: MACROs matched found: 437
mergeLef.py : Merging LEFs complete
padLefMacro.py : Padding technology lef file
Derived SITE width (microns): 0.46
Derived SITE height (microns): 5.44
Right cell padding (microns): 3.68
Left cell padding (microns): 0.0
Top cell padding (microns): 0.0
Bottom cell padding (microns): 0.0
Skipping LEF padding for MACRO  sky130_fd_sc_hd__tap_2
Skipping LEF padding for MACRO  sky130_fd_sc_hd__tapvgnd_1
Skipping LEF padding for MACRO  sky130_fd_sc_hd__fill_4
Skipping LEF padding for MACRO  sky130_fd_sc_hd__decap_12
Skipping LEF padding for MACRO  sky130_fd_sc_hd__fill_1
Skipping LEF padding for MACRO  sky130_fd_sc_hd__fill_2
Skipping LEF padding for MACRO  sky130_fd_sc_hd__tapvgnd2_1
Skipping LEF padding for MACRO  sky130_fd_sc_hd__fill_8
Skipping LEF padding for MACRO  sky130_fd_sc_hd__tap_1
Skipping LEF padding for MACRO  sky130_fd_sc_hd__decap_4
Skipping LEF padding for MACRO  sky130_fd_sc_hd__tapvpwrvgnd_1
Skipping LEF padding for MACRO  sky130_fd_sc_hd__decap_8
Skipping LEF padding for MACRO  sky130_fd_sc_hd__decap_3
Skipping LEF padding for MACRO  sky130_fd_sc_hd__decap_6
padLefMacro.py : Finished
[INFO]: Preparation complete
% set lefs [glob $::env(DESIGN_DIR)/src/*.lef]
/openLANE_flow/designs/picorv32a/src/sky130_vsdinv.lef
% add_lefs -src $lefs
[INFO]: Merging /openLANE_flow/designs/picorv32a/src/sky130_vsdinv.lef
%
```

- Then, run synthesis using the *run_synthesis* command and check that sky130_vsdinv cell is successfully included in the design



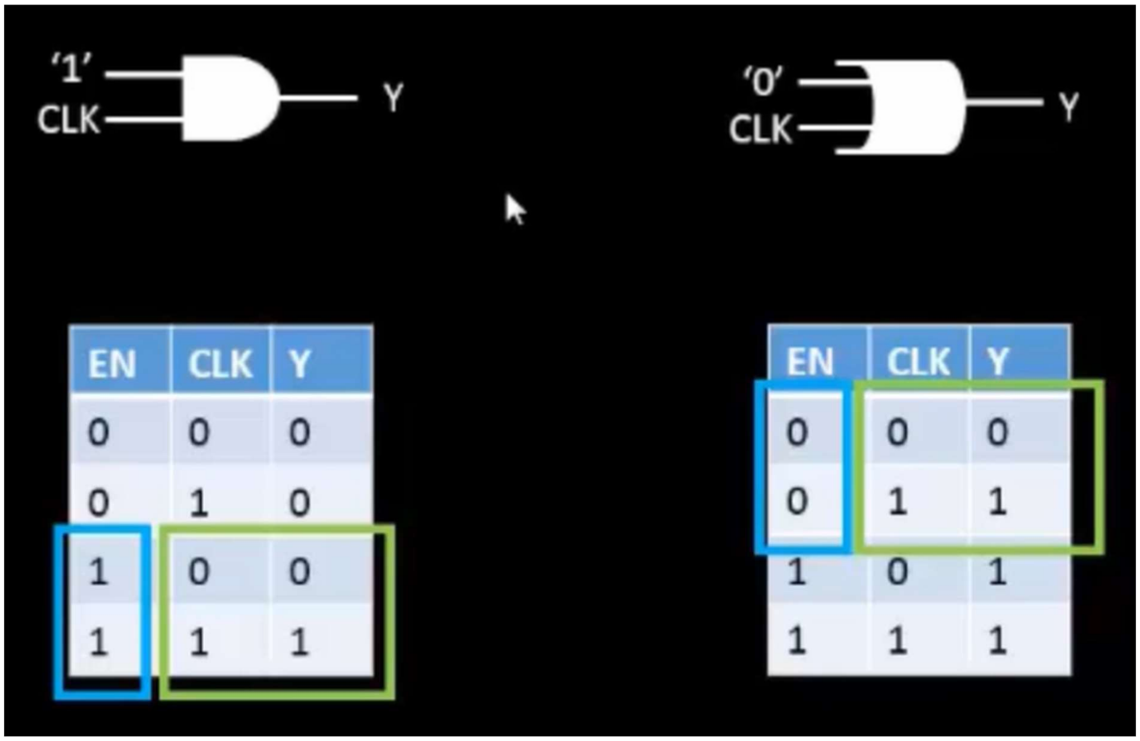
```
     sky130_fd_sc_hd__or3_2            68
     sky130_fd_sc_hd__or3b_2            5
     sky130_fd_sc_hd__or4_2            93
     sky130_fd_sc_hd__or4b_2            6
     sky130_fd_sc_hd__or4bb_2           2
     sky130_vsdinv                   1554

   Chip area for module '\picorv32a': 147712.918400
```

NOTE : here, timing is not fixed, which then needs to be fixed.



```
tns -711.59
wns -23.89
[INFO]: Synthesis was successful
```

Delay Tables

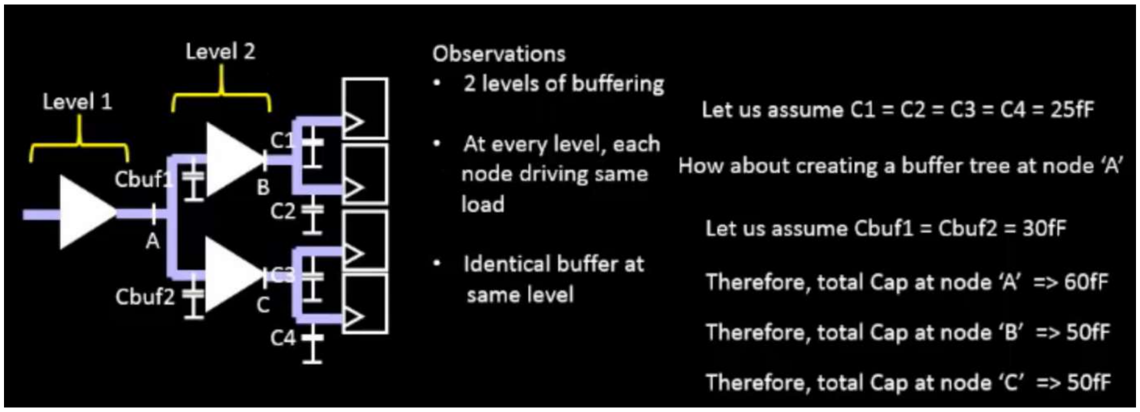


EN	CLK	Y
0	0	0
0	1	0
1	0	0
1	1	1

EN	CLK	Y
0	0	0
0	1	1
1	0	1
1	1	1

The image shows how the enable pin affects CLK propagation. For an AND gate, CLK propagates to Y only when the enable pin is 1. For an OR gate, CLK propagates to Y only when the enable pin is 0. When the enable pin is 1, CLK doesn't propagate, preventing short circuit power consumption. This technique, used in the clock tree, is called Clock Gating.

However, when we think of this, the question that arises is ***HOW DOES ONE USE THIS?*** For this purpose, consider the below clock tree structure -:



Observations

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

Let us assume C1 = C2 = C3 = C4 = 25fF

How about creating a buffer tree at node 'A'

Let us assume Cbuf1 = Cbuf2 = 30fF

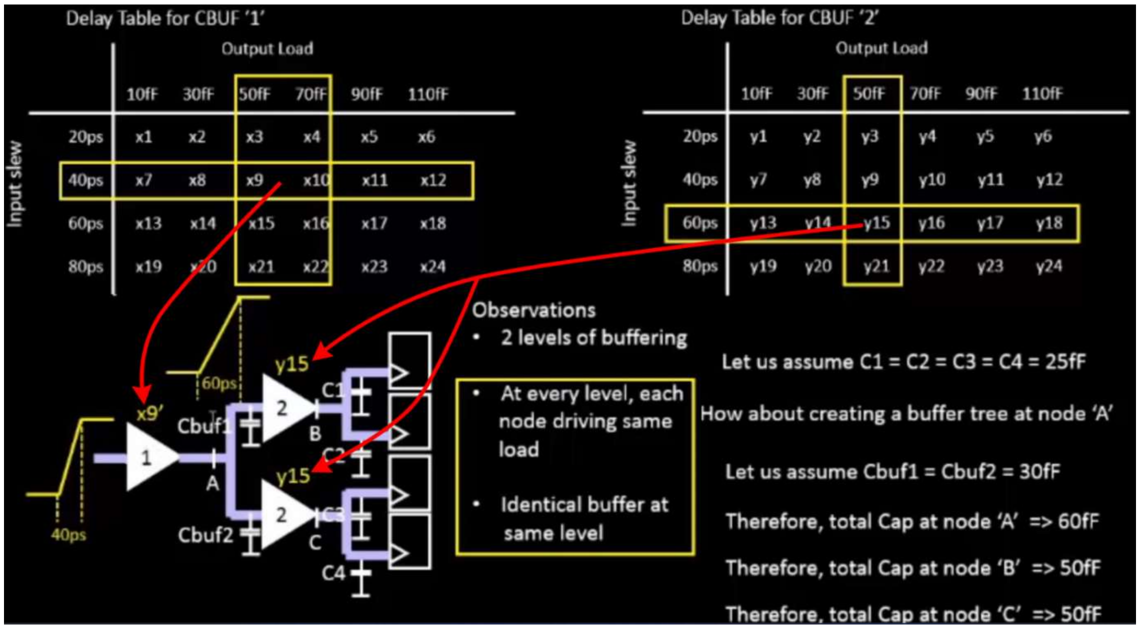
Therefore, total Cap at node 'A' => 60fF

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

Therefore, total Cap at node 'C' => 50fF

The picture shows that buffers at different levels have different capacitive loads and sizes. As long as buffers within the same level have the same loads and sizes, the total delay for each clock tree path will be the same, keeping skew at zero. However, in practice, different levels may have varying input transitions, output capacitive loads, and delays.

Delay tables in .lef files show each cell's timing models. The main factor impacting delay is the output slew, which depends on capacitive load and input slew. Input slew is influenced by the previous stage buffer's output capacitive load and input slew, each with its own transition delay table.



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
40ps	y7	y8	y9	y10	y11	y12
60ps	y13	y14	y15	y16	y17	y18
80ps	y19	y20	y21	y22	y23	y24

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

Let us assume C1 = C2 = C3 = C4 = 25fF

How about creating a buffer tree at node 'A'

Let us assume Cbuf1 = Cbuf2 = 30fF

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

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

Therefore, total Cap at node 'C' => 50fF

At level 2, both buffers have identical delays due to equal transition times, load capacitances, and buffer sizes, resulting in zero skew. If this balance is not maintained, negative skew can lead to timing violations. While small-scale designs may overlook this, large designs with millions of cells can face significant issues if clock tree guidelines aren't followed. CTS (Clock Tree Synthesis) is the process of designing a clock distribution network to minimize skew and ensure synchronous operation. Skew refers to variations in clock signal arrival times, slew rate is the change in signal voltage over time, and latency is the delay experienced by the clock signal.

Lab Steps To Configure Synthesis Settings to Fix Slack and Include VSDINV

We can see through previous pictures that currently,

- tns = -711.59
- wns = -23.89
- chip area for the module picorv32a = 147712.9184

Hence, our next step is to try to make the synthesis more **timing driven**. This can be done as follows-:

1. First, check the synthesis strategy and timing-related variables, then modify them accordingly:

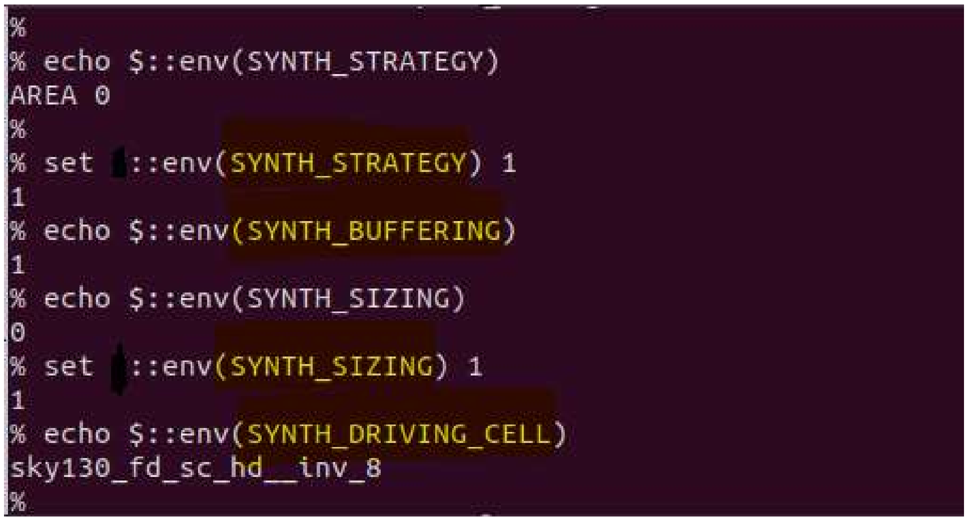
SYNTH_STRATEGY (delay 0): Focuses more on optimizing delay.

Index (0, 1, 2, 3): 3 is the most optimized for timing, at the cost of area.

SYNTH_BUFFERING (1): Uses buffers on high fanout cells to reduce wire delay.

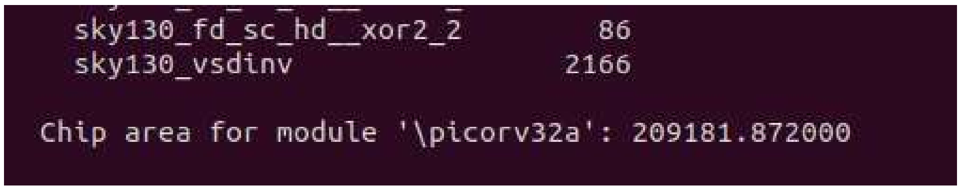
SYNTH_SIZING (1): Enables cell sizing to upsize or downsize cells as needed to meet timing.

SYNTH_DRIVING_CELL: The cell used to drive input ports, essential for cells with high fanout requiring higher drive strength.



```
%
% echo $::env(SYNTH_STRATEGY)
AREA 0
%
% set [redacted]::env(SYNTH_STRATEGY) 1
1
% echo $::env(SYNTH_BUFFERING)
1
% echo $::env(SYNTH_SIZING)
0
% set [redacted]::env(SYNTH_SIZING) 1
1
% echo $::env(SYNTH_DRIVING_CELL)
sky130_fd_sc_hd__inv_8
%
```

2. After this, run synthesis again. It is will be seen that area is increased but there is no negative slack
   - tns = 0
   - wns = 0
   - Chip area for module picorv32a = 209181.872



```
  sky130_fd_sc_hd__xor2_2          86
  sky130_vsdinv                  2166

Chip area for module '\picorv32a': 209181.872000
```

3. Subsequently, we may run floorplan and placement

NOTE: If there are errors related to macro placement, a temporary solution is to comment out basic_macro_placement in the file OpenLane/scripts/tcl_commands/floorplan.tcl. This is fine since no macros are being added to the design.

We can also execute the following commands:-

init_floorplan

place_io

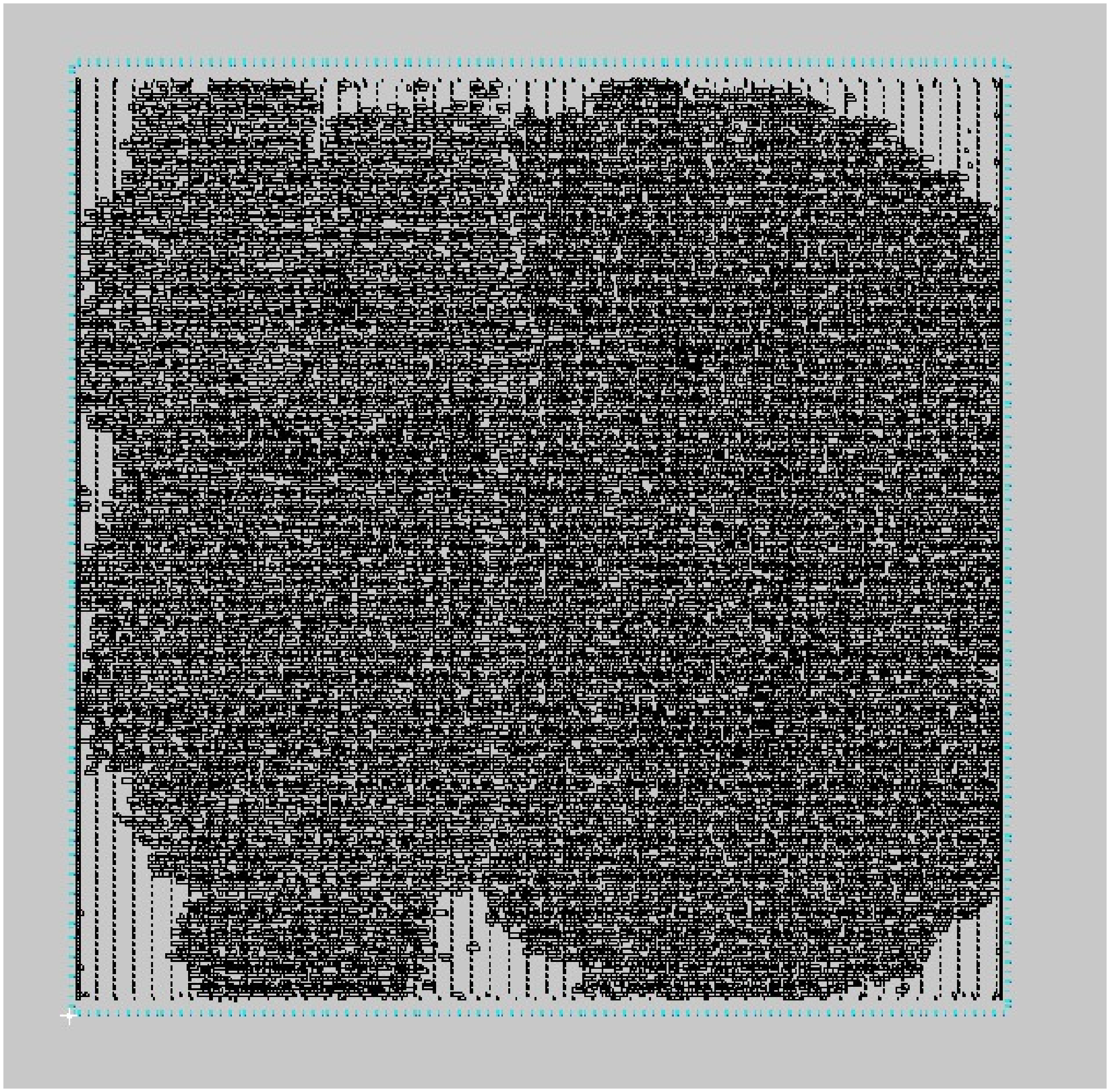
global_placement_or

detailed_placement

tap_decap_or

detailed_placement

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



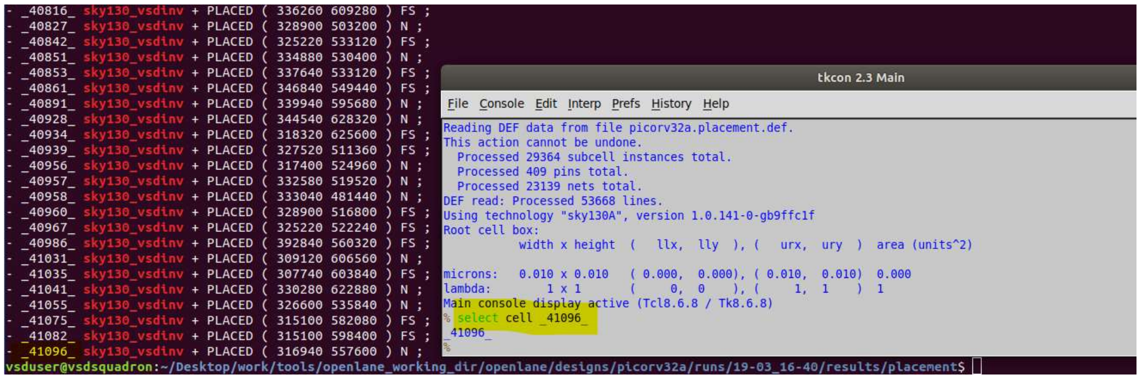
4. After placement, search for the instance of the cell sky130_vsdinv in the DEF file using the command:

   cat picorv32a.placement.def | grep sky130_vsdinv

5. Next, select a single sky130_vsdinv cell instance from the list (e.g., 41096). Run the following command in tkcon:

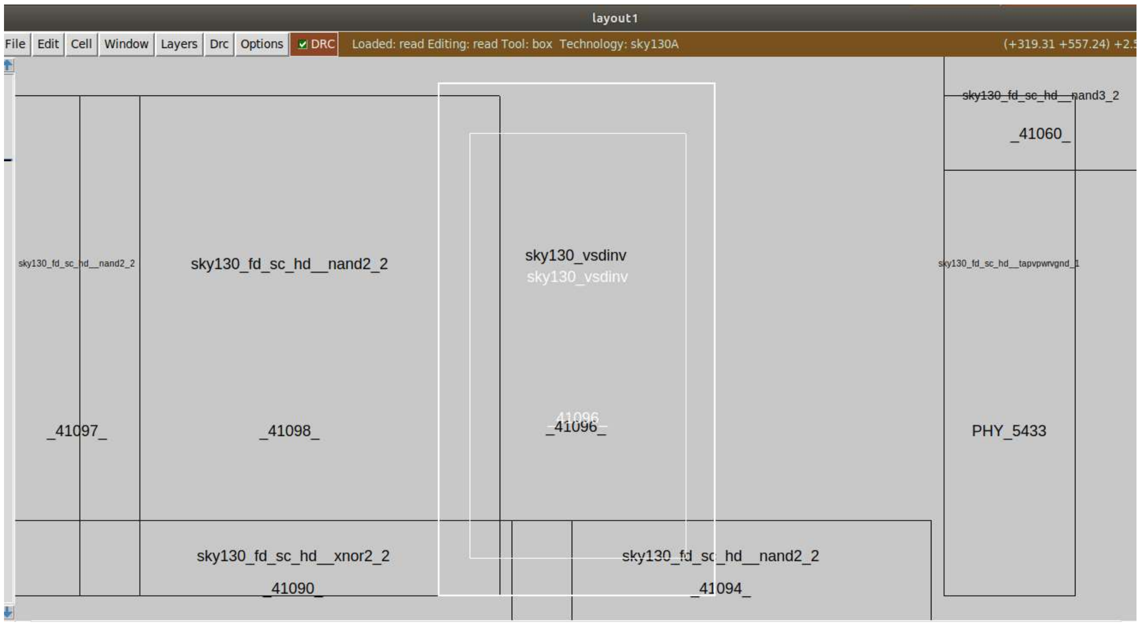
   select cell 41096

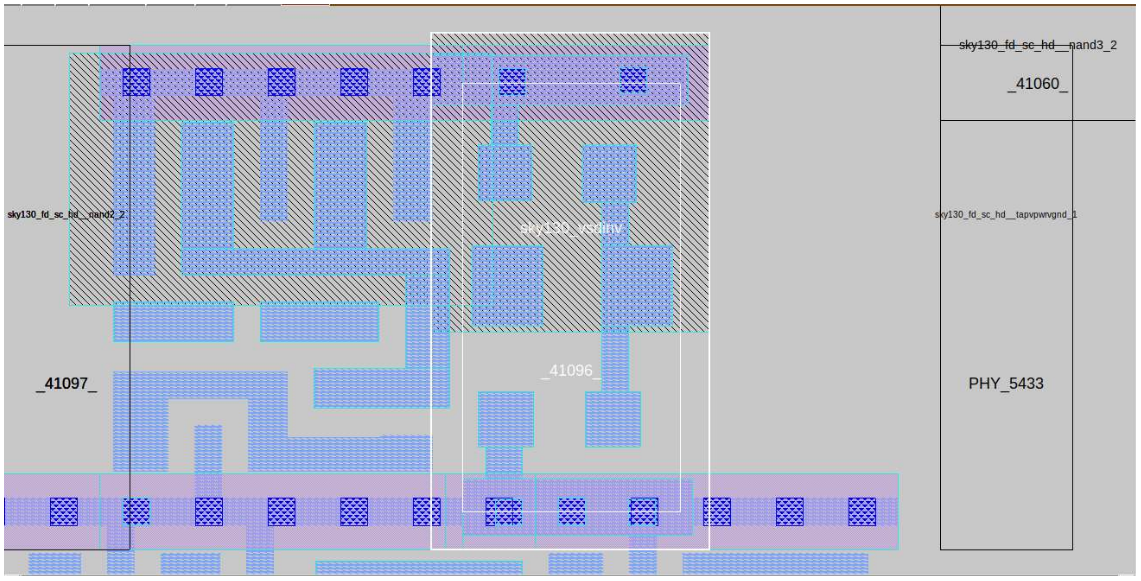
6. After selecting the cell, press **Ctrl+Z** to zoom into it. This confirms that the customized inverter cell **sky130_myinverter** is successfully placed. Then, use the **expand** command in tkcon to show the footprint of the cell. Notice how the power and ground pins of **sky130_vsdinv** overlap with the power and ground pins of its adjacent cells.



```
- _40816_ sky130_vsdinv + PLACED ( 336260 609280 ) FS ;
- _40827_ sky130_vsdinv + PLACED ( 328900 503200 ) N ;
- _40842_ sky130_vsdinv + PLACED ( 325220 533120 ) FS ;
- _40851_ sky130_vsdinv + PLACED ( 334880 530400 ) N ;
- _40853_ sky130_vsdinv + PLACED ( 337640 533120 ) FS ;
- _40861_ sky130_vsdinv + PLACED ( 346840 549440 ) FS ;
- _40891_ sky130_vsdinv + PLACED ( 339940 595680 ) N ;
- _40928_ sky130_vsdinv + PLACED ( 344540 628320 ) N ;
- _40934_ sky130_vsdinv + PLACED ( 318320 625600 ) FS ;
- _40939_ sky130_vsdinv + PLACED ( 327520 511360 ) FS ;
- _40956_ sky130_vsdinv + PLACED ( 317400 524960 ) N ;
- _40957_ sky130_vsdinv + PLACED ( 332580 519520 ) N ;
- _40958_ sky130_vsdinv + PLACED ( 333040 481440 ) N ;
- _40960_ sky130_vsdinv + PLACED ( 328900 516800 ) FS ;
- _40967_ sky130_vsdinv + PLACED ( 325220 522240 ) FS ;
- _40986_ sky130_vsdinv + PLACED ( 392840 560320 ) FS ;
- _41031_ sky130_vsdinv + PLACED ( 309120 606560 ) N ;
- _41035_ sky130_vsdinv + PLACED ( 307740 603840 ) FS ;
- _41041_ sky130_vsdinv + PLACED ( 330280 622880 ) N ;
- _41055_ sky130_vsdinv + PLACED ( 326600 535840 ) N ;
- _41075_ sky130_vsdinv + PLACED ( 315100 582080 ) FS ;
- _41082_ sky130_vsdinv + PLACED ( 315100 598400 ) FS ;
- _41096_ sky130_vsdinv + PLACED ( 316940 557600 ) N ;
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/19-03_16-40/results/placement$
```

```
Reading DEF data from file picorv32a.placement.def.
This action cannot be undone.
  Processed 29364 subcell instances total.
  Processed 409 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_
%
```





Timing Analysis With Ideal Clocks Using Open STA

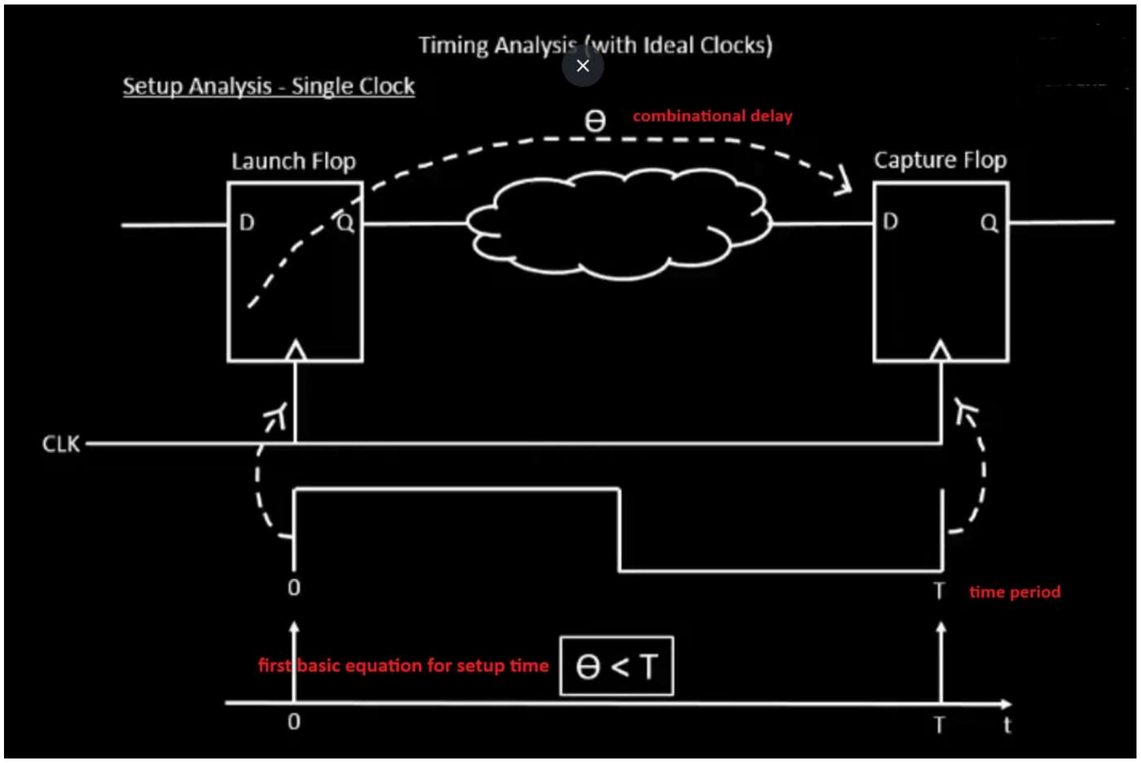
Setup Timing Analysis And Introduction to Flip-Flop Setup Time

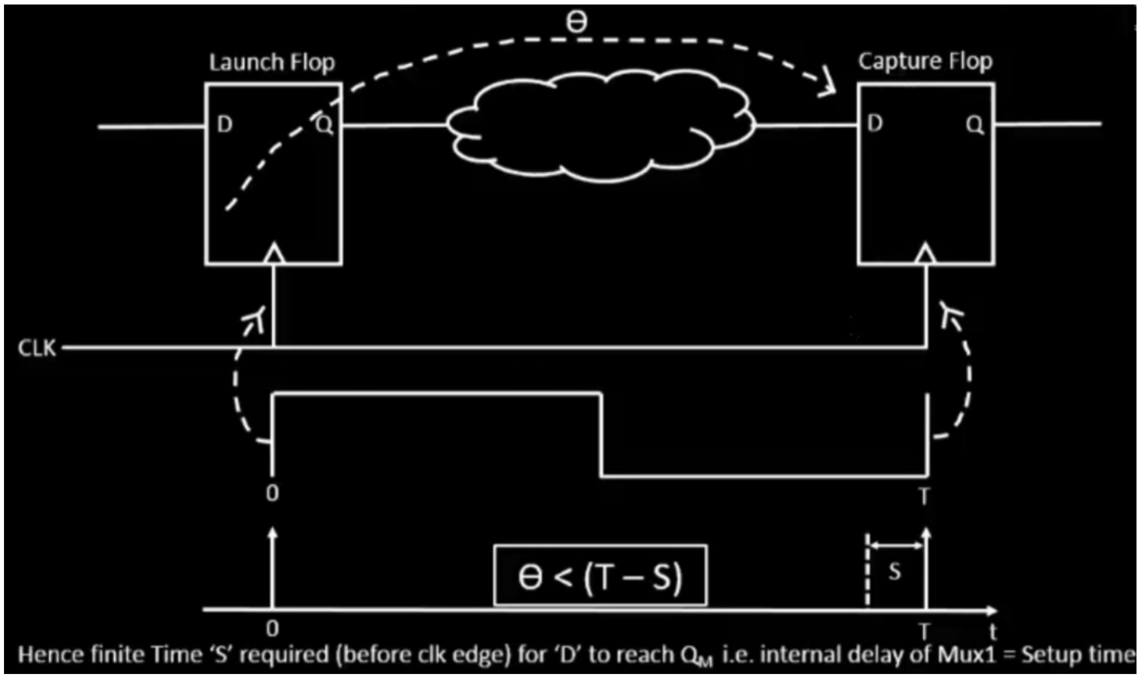
Consider an ideal clock (before the clock tree is built). Perform timing analysis to understand the parameters, which can later be applied using real clocks. Given specifications:

Clock Frequency (F): 1 GHz

Clock Period (T): 1 ns

Using these, you can calculate other timing parameters such as setup time, hold time, and propagation delay, and understand how they impact the overall timing analysis:


Timing Analysis (with Ideal Clocks)
Setup Analysis - Single Clock
ϴ
combinational delay
Launch Flop
Capture Flop
D
Q
D
Q
CLK
0
T
time period
first basic equation for setup time
ϴ < T
0
T
t


ϴ
Launch Flop
Capture Flop
D
Q
D
Q
CLK
0
T
ϴ < (T – S)
S
0
T
t
Hence finite Time 'S' required (before clk edge) for 'D' to reach Q_M i.e. internal delay of Mux1 = Setup time

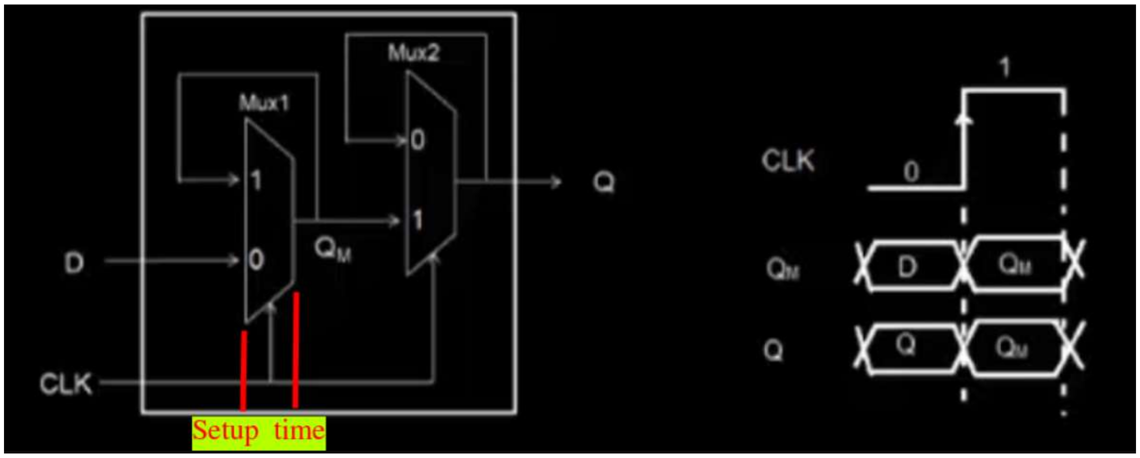
The setup timing analysis equation is = ϴ < T - S

Where:

ϴ: Combinational delay, which includes the clk-to-Q delay of the launch flip-flop and the internal propagation delay of all gates between the launch and capture flip-flops.

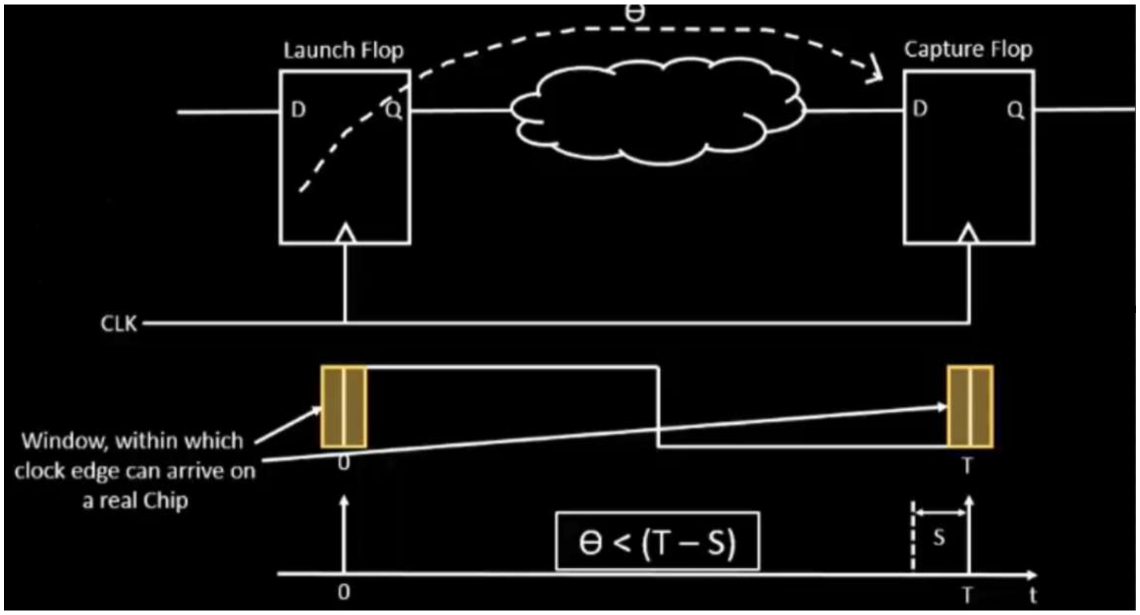
T: Time period, also known as the required time.

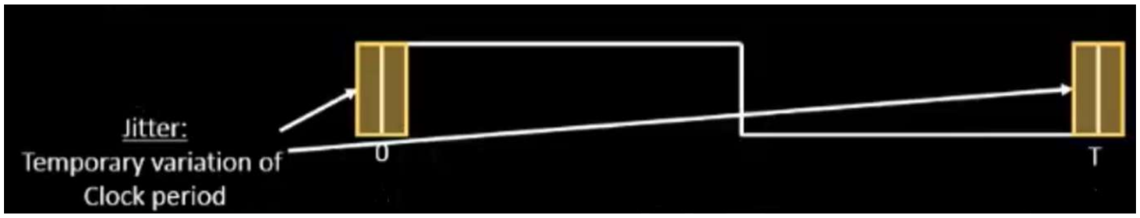
S: Setup time, which ensures that the signal settles at the middle (input of Mux 2) before the clock transitions to 1. The delay due to Mux 1 must be considered, as it contributes to the setup time.



Introduction to Clock Jitter and Uncertainty

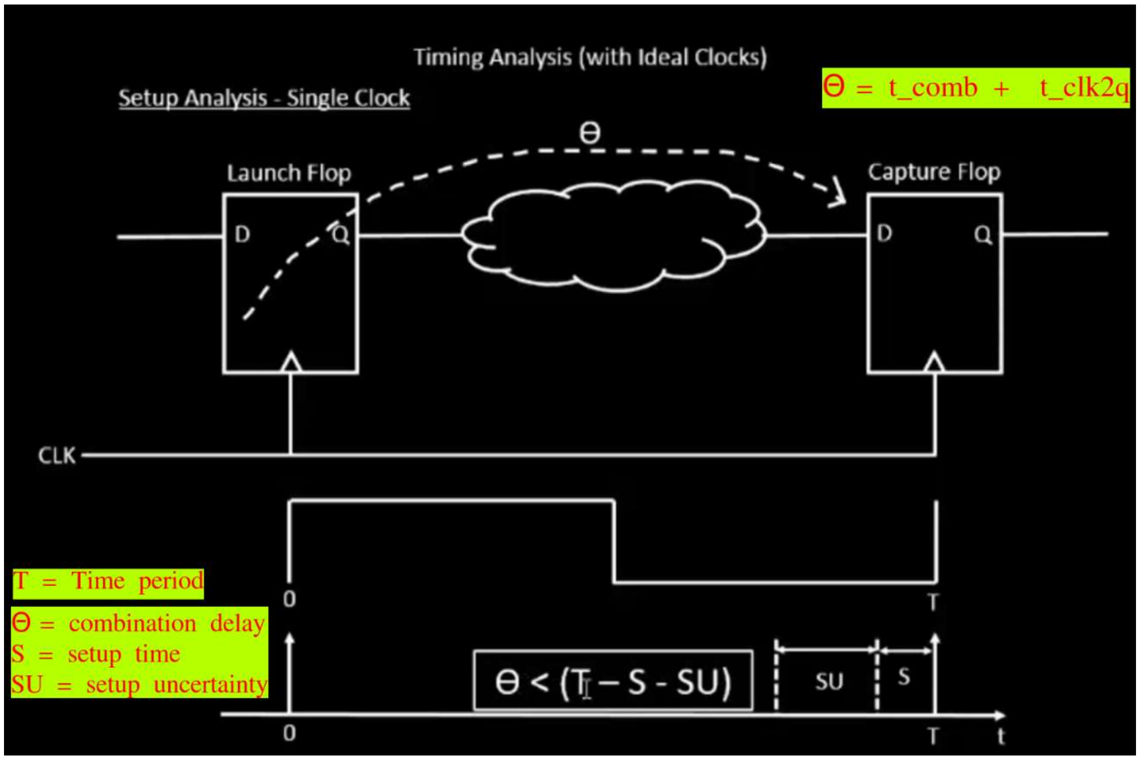
CLK signals are provided to the device by the PLL (Phase Locked Loop), which is expected to send the clock signal at 0, T, 2T, etc. However, these clock sources may not always deliver the clock exactly at T ns due to an inherent variation known as **jitter**. Jitter refers to short-term fluctuations in the timing of signal transitions, leading to deviations from the expected clock or data timing.





Hence, we see that a more realistic equation for setup time is = Θ < T - S - SU

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

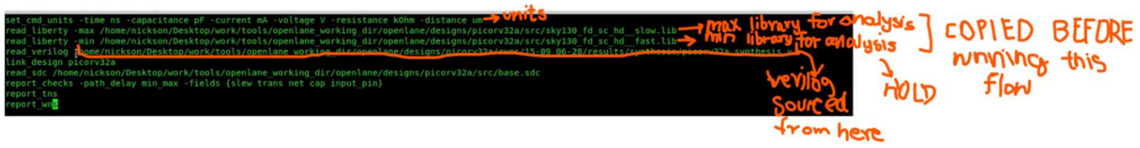


Lab Steps to Configure OpenSTA for Post-Synth Timings Analysis

STA can be either:

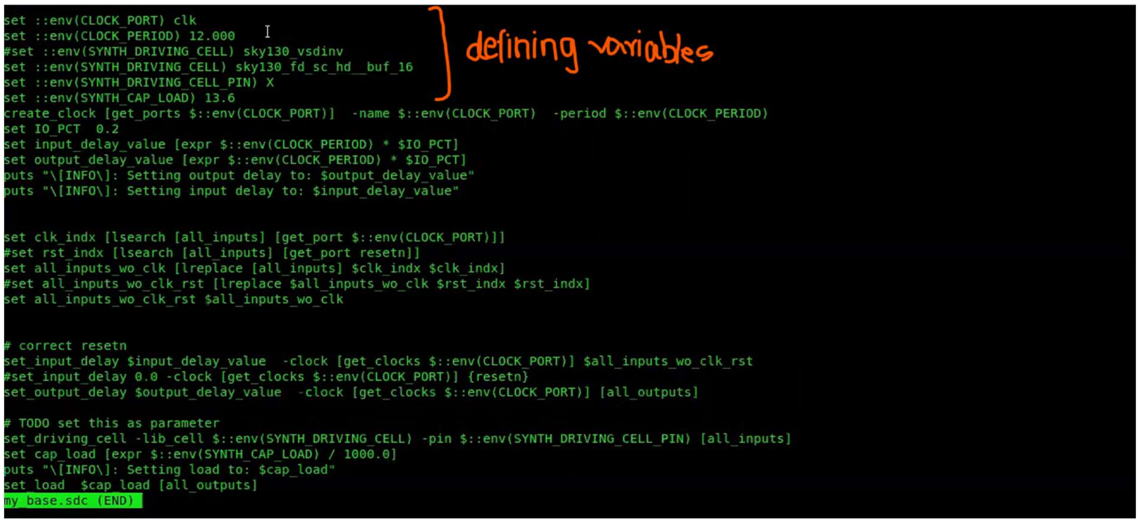
- **Single corner**: Uses only the **LIB_TYPICAL** library, typically used in pre-layout (post-synthesis) STA.
- **Multi-corner**: Uses **LIB_SLOWEST** (for setup analysis at high temperature and low voltage), **LIB_FASTEST** (for hold analysis at low temperature and high voltage), and **LIB_TYPICAL** libraries.

1. In the location *Desktop/Work/tools/openlane_working_dir/openlane_, there is a file named **pre_sta.conf** that will be used for pre-layout analysis. The file contains the following contents:



```
set_cmd_units -time ns -capacitance pF -current mA -voltage V -resistance kOhm -distance um
read_liberty -max /home/nickson/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/src/sky130_fd_sc_hd__slow.lib
read_liberty -min /home/nickson/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/src/sky130_fd_sc_hd__fast.lib
read_verilog /home/nickson/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/runs/15-09_06-28/results/synthesis/picorv32a_synthesis.v
link_design picorv32a
read_sdc /home/nickson/Desktop/work/tools/openlane_working_dir/openlane/designs/picorv32a/src/base.sdc
report_checks -path_delay min_max -fields {slew trans net cap input_pin}
report_tns
report_wns
```

2. This is the SDC file on which analysis will be done is located at *Desktop/Work/tools/openlane_working_dir/openlane/designs/picorv2a/src* and is named *my_base.src_. It contains the following contents -:



```
set ::env(CLOCK_PORT) clk
set ::env(CLOCK_PERIOD) 12.000
#set ::env(SYNTH_DRIVING_CELL) sky130_vsdinv
set ::env(SYNTH_DRIVING_CELL) sky130_fd_sc_hd__buf_16
set ::env(SYNTH_DRIVING_CELL_PIN) X
set ::env(SYNTH_CAP_LOAD) 13.6
create_clock [get_ports $::env(CLOCK_PORT)]  -name $::env(CLOCK_PORT)  -period $::env(CLOCK_PERIOD)
set IO_PCT  0.2
set input_delay_value [expr $::env(CLOCK_PERIOD) * $IO_PCT]
set output_delay_value [expr $::env(CLOCK_PERIOD) * $IO_PCT]
puts "\[INFO\]: Setting output delay to: $output_delay_value"
puts "\[INFO\]: Setting input delay to: $input_delay_value"


set clk_indx [lsearch [all_inputs] [get_port $::env(CLOCK_PORT)]]
#set rst_indx [lsearch [all_inputs] [get_port resetn]]
set all_inputs_wo_clk [lreplace [all_inputs] $clk_indx $clk_indx]
#set all_inputs_wo_clk_rst [lreplace $all_inputs_wo_clk $rst_indx $rst_indx]
set all_inputs_wo_clk_rst $all_inputs_wo_clk


# correct resetn
set_input_delay $input_delay_value  -clock [get_clocks $::env(CLOCK_PORT)] $all_inputs_wo_clk_rst
#set_input_delay 0.0 -clock [get_clocks $::env(CLOCK_PORT)] {resetn}
set_output_delay $output_delay_value  -clock [get_clocks $::env(CLOCK_PORT)] [all_outputs]

# TODO set this as parameter
set_driving_cell -lib_cell $::env(SYNTH_DRIVING_CELL) -pin $::env(SYNTH_DRIVING_CELL_PIN) [all_inputs]
set cap_load [expr $::env(SYNTH_CAP_LOAD) / 1000.0]
puts "\[INFO\]: Setting load to: $cap_load"
set_load  $cap_load [all_outputs]
my_base.sdc (END)
```

3. The various commands and their functions are:

   create_clock: Creates a clock for the port with a 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 specifies the delay of signals coming from an external block and the internal delay of signals being propagated to external ports. It adds a delay of **X ns** relative to the clock for input ports, and a delay of **Y ns** relative to the clock for output ports.

   set_max_fanout: Specifies the 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.

4. 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 caused by nets with high fanout. High fanout leads to increased load capacitance, which in turn causes higher delays. You can use the command:

report_net -connections <net_name>

to display the connections of the nets. If the output reveals large fanouts, you can reduce them by going back to OpenLane and setting:

::env(SYNTH_MAX_FANOUT) 4

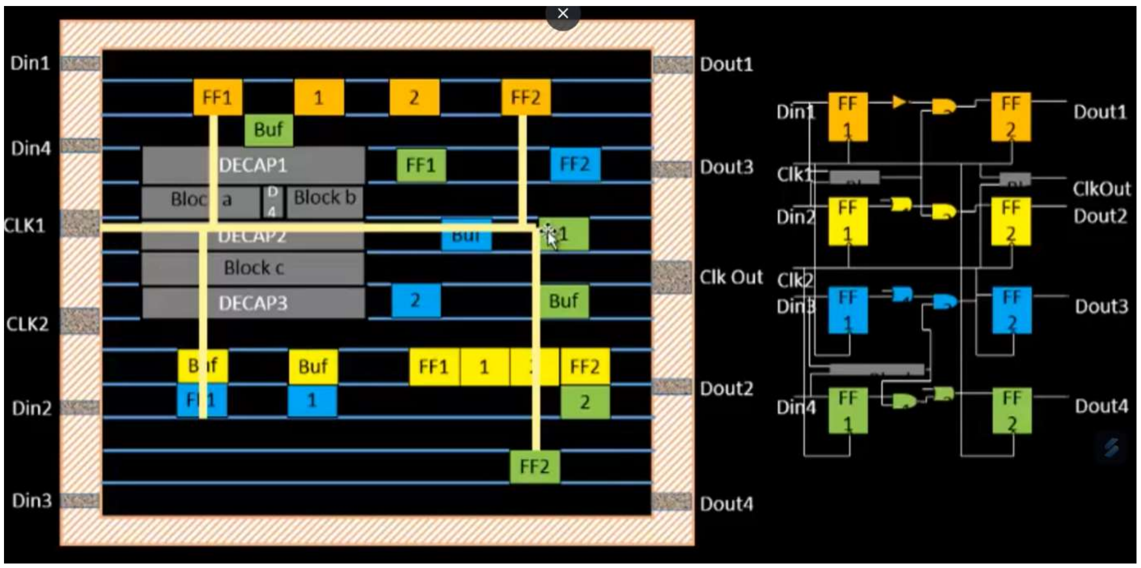
Then, run **run_synthesis** again to obtain the expected slack value with the reduced fanouts.

Lab Steps to do Basic Timing ECO

To further reduce negative slack, you can upsize cells with high fanouts, using larger drivers to handle the increased load capacitance. Since you can't change the load capacitance directly, increasing the size of the cell allows it to drive larger capacitive loads, resulting in lower delays. This process is typically done iteratively until the desired slack is achieved. This method is known as Timing ECO (Engineering Change Order), where adjustments are made to the design to meet timing requirements.

Clock Tree Synthesis TritonCTS and Signal Integrity

Clock Tree Routing and Buffering Using H-Tree Algorithm



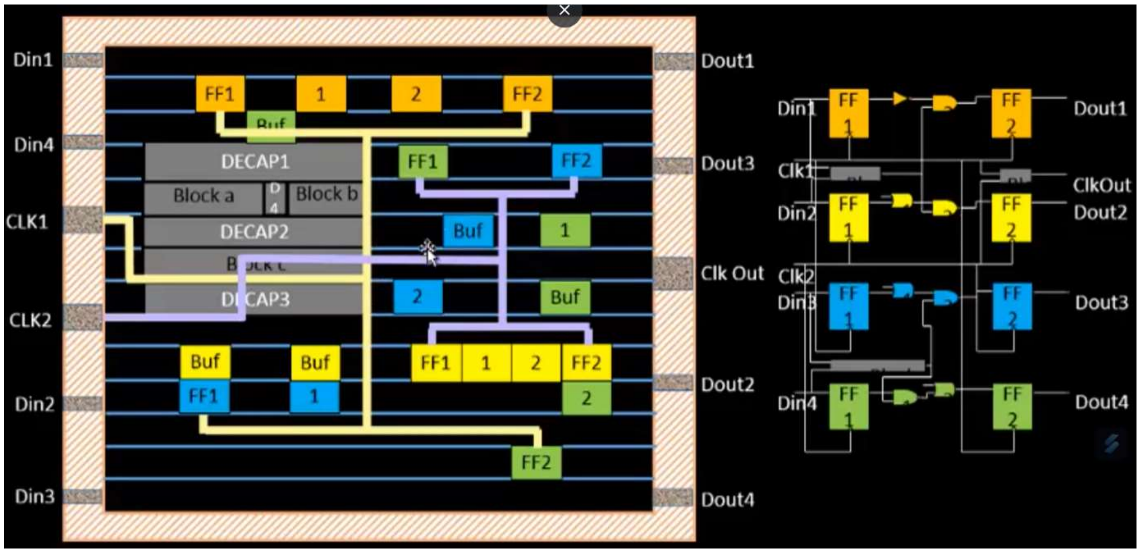
- In the above picture, the **CLK port** connects to the CLK pins of various flip-flops based on the given connectivity information. However, if connections are made blindly, **t2** (arrival time at one flip-flop) might be greater than **t1** (arrival time at another flip-flop), leading to **clock skew**. Clock skew is the difference between **t2** and **t1** and refers to the variation in clock signal arrival times at different points in the system. It occurs due to:
- Differences in wire lengths
- Variations in signal routing paths
- Variations in buffer delays
- Other physical and environmental factors
- As a result, some parts of the system may receive the clock signal earlier or later than others. Minimizing clock skew is crucial for:
- Ensuring proper synchronization of signals

- Reliable operation of the digital circuit

NOTE : Ideally, the skew should be zero.

Since the skew is not minimized in the above picture, we can refer to this clock tree as a "bad tree." To optimize this scenario, we can use **H-Tree routing**. This method improves the clock distribution by analyzing the clock route and calculating the distance from the clock source to all the endpoints. It then selects a midpoint and begins building the tree from there.

The process is repeated by finding subsequent midpoints and creating splits in the clock wire at each stage. This approach helps distribute the clock signal more evenly, ensuring that the clock reaches all flip-flops at almost the same time, minimizing clock skew.



Although it's expected that the input **CLK** signal is exactly reproduced at the output, this does not happen in **H-Tree routing** due to inherent resistance and capacitance in the physical wires, which can distort the signal. To address this issue and maintain signal integrity, **buffers/repeaters** can be inserted along the clock path.

There's a key distinction between the **repeaters used in clock paths** and those used in **data paths**. Clock buffers are designed with **identical rise and fall times** to ensure uniform signal propagation across the clock distribution network, which helps maintain synchronization. On the other hand, **data buffers** have varying rise and fall times, depending on the characteristics of the data being transmitted and the components processing it.

Crosswalk and Clock Net Shielding

Clock nets are critical in a design because the clock tree is built to minimize skew to zero. However, a phenomenon called **crosstalk** can occur, where signals transmitted on one channel interfere with adjacent signals, leading to distortion, noise, timing errors, and potential deterioration of the clock tree structure.

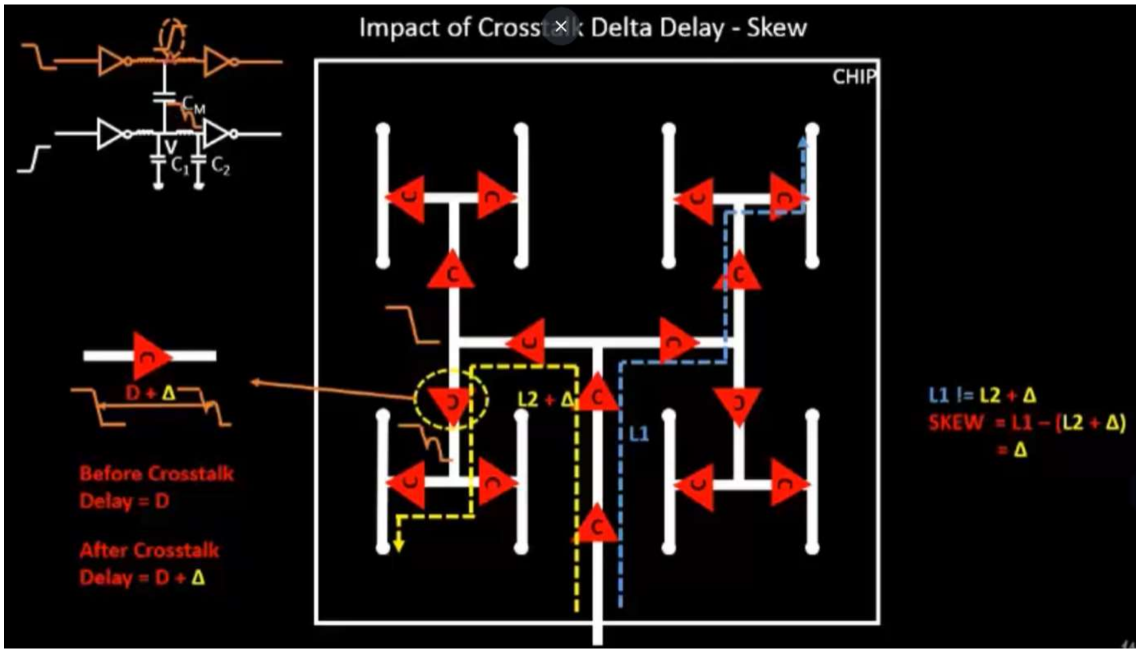
To address this, **clock nets are shielded** (protected). If a wire is adjacent to these shields, **coupling capacitance** can cause two major issues:

1. **Glitch**: When there is switching activity in the aggressor (the signal causing interference), the coupling capacitance can affect the victim net (the signal affected by

the interference). This results in a voltage dip, causing a glitch. Glitches can lead to incorrect data in memory, resulting in functional inaccuracies.

2. **Delta delay**: The interference can cause delay variations in the victim net due to the coupling capacitance.

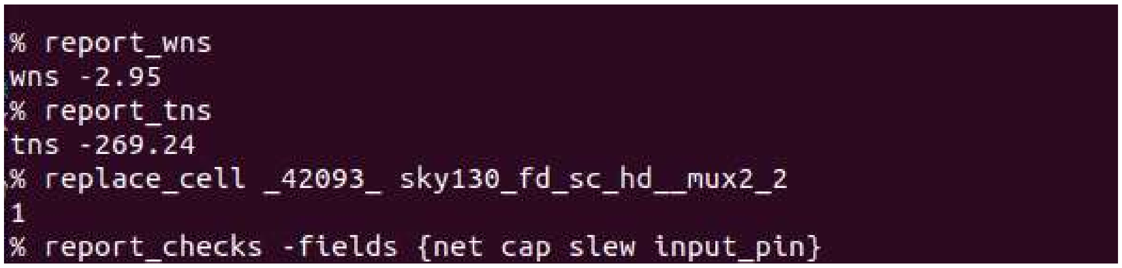
To prevent these issues, **shielding nets** are used, typically connected to **Vdd** or **Vss**. These shields don't switch, so they prevent the victim net from switching as well, thereby breaking the coupling capacitance between the aggressor and the victim, ensuring signal integrity and minimizing glitches and delays.



Lab Steps to run CTS using TritonCTS

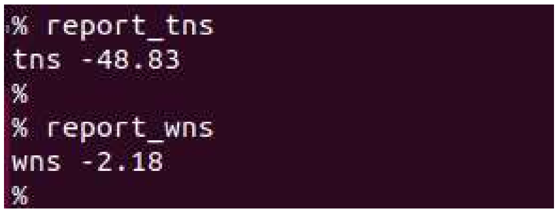
After completing **Timing ECO**, if the timing is still above the desired value (e.g., above 1), you can review the results to identify the cells causing significant slack. These cells can be switched out for **buffers**, which are better suited to handle high fanout and reduce delays.

By replacing these cells with buffers, the **negative slack** can be reduced, ultimately bringing it below 1 and meeting the timing requirements. This process may involve iterating through the design, analyzing the slack, and adjusting cells as needed to ensure the timing constraints are satisfied.

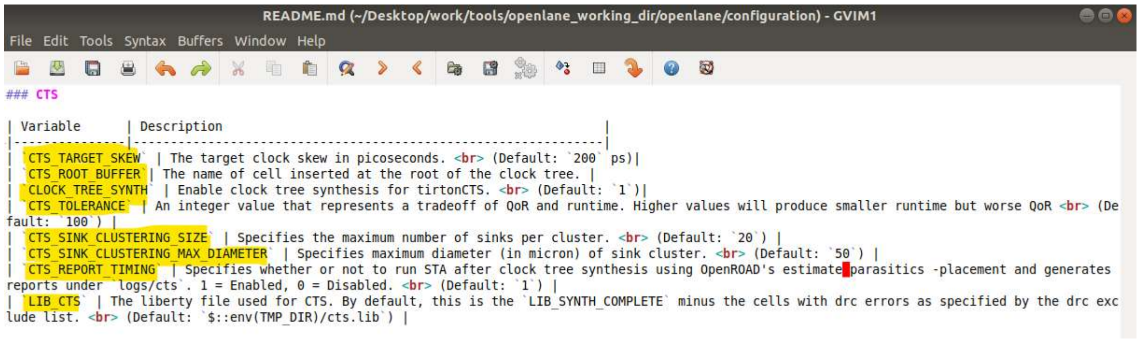


```
% report_wns
wns -2.95
% report_tns
tns -269.24
% replace_cell _42093_ sky130_fd_sc_hd__mux2_2
1
% report_checks -fields {net cap slew input_pin}
```

{IMAGE CREDITS: VSDIAT; SCREENSHOT TAKEN FROM LECTURE}

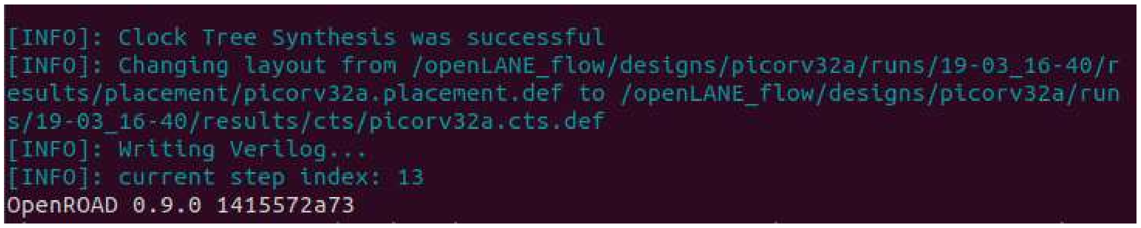

```
% report_tns
tns -48.83
%
% report_wns
wns -2.18
%
```

After this, for OpenLANE to use the modified netlist, we can type **write_verilog [filename]** and then run floorplan and placement.


README.md (~/Desktop/work/tools/openlane_working_dir/openlane/configuration) - GVIM1
File Edit Tools Syntax Buffers Window Help

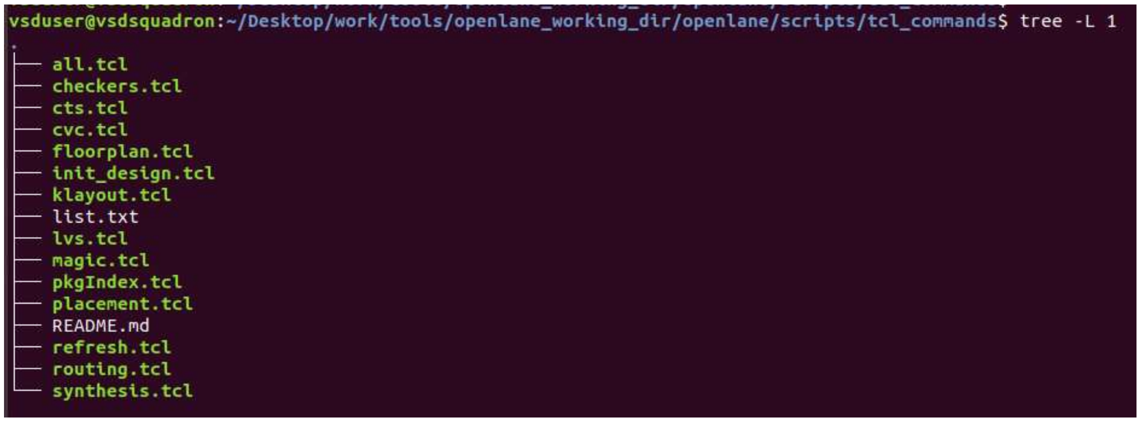
```
### CTS

| Variable      | Description                                                    |
|---------------|----------------------------------------------------------------|
| `CTS_TARGET_SKEW` | The target clock skew in picoseconds. <br> (Default: `200` ps)|
| `CTS_ROOT_BUFFER`| The name of cell inserted at the root of the clock tree. |
| `CLOCK_TREE_SYNTH` | Enable clock tree synthesis for tirtonCTS. <br> (Default: `1`)|
| `CTS_TOLERANCE` | An integer value that represents a tradeoff of QoR and runtime. Higher values will produce smaller runtime but worse QoR <br> (Default: `100`) |
| `CTS_SINK_CLUSTERING_SIZE` | Specifies the maximum number of sinks per cluster. <br> (Default: `20`) |
| `CTS_SINK_CLUSTERING_MAX_DIAMETER` | Specifies maximum diameter (in micron) of sink cluster. <br> (Default: `50`) |
| `CTS_REPORT_TIMING` | Specifies whether or not to run STA after clock tree synthesis using OpenROAD's estimate parasitics -placement and generates reports under `logs/cts`. 1 = Enabled, 0 = Disabled. <br> (Default: `1`) |
| `LIB_CTS` | The liberty file used for CTS. By default, this is the `LIB_SYNTH_COMPLETE` minus the cells with drc errors as specified by the drc exclude list. <br> (Default: `$::env(TMP_DIR)/cts.lib`) |
```

After running CTS, we can see that the clock buffers get added -:


```
[INFO]: Clock Tree Synthesis was successful
[INFO]: Changing layout from /openLANE_flow/designs/picorv32a/runs/19-03_16-40/results/placement/picorv32a.placement.def to /openLANE_flow/designs/picorv32a/runs/19-03_16-40/results/cts/picorv32a.cts.def
[INFO]: Writing Verilog...
[INFO]: current step index: 13
OpenROAD 0.9.0 1415572a73
```

Lab Steps to Verify CTS Runs

After the previous step, OpenLANE will take the procedures from **/Desktop/work/tools/openlane_working_dir/openlane/scripts/tcl_commands** and eventually invoke OpenROAD to run the tool.


```
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane/scripts/tcl_commands$ tree -L 1
.
├── all.tcl
├── checkers.tcl
├── cts.tcl
├── cvc.tcl
├── floorplan.tcl
├── init_design.tcl
├── klayout.tcl
├── list.txt
├── lvs.tcl
├── magic.tcl
├── pkgIndex.tcl
├── placement.tcl
├── README.md
├── refresh.tcl
├── routing.tcl
└── synthesis.tcl
```

For example, the command **run_cts** can be found in the location /OpenLane/scripts/tcl_commands/cts.tcl. This TCL process invokes **OpenROAD**, which then calls **cts.tcl** to execute the OpenROAD commands for **TritonCTS** (Clock Tree Synthesis).

The **cts.tcl** file contains several configuration variables for the clock tree synthesis (CTS), such as:

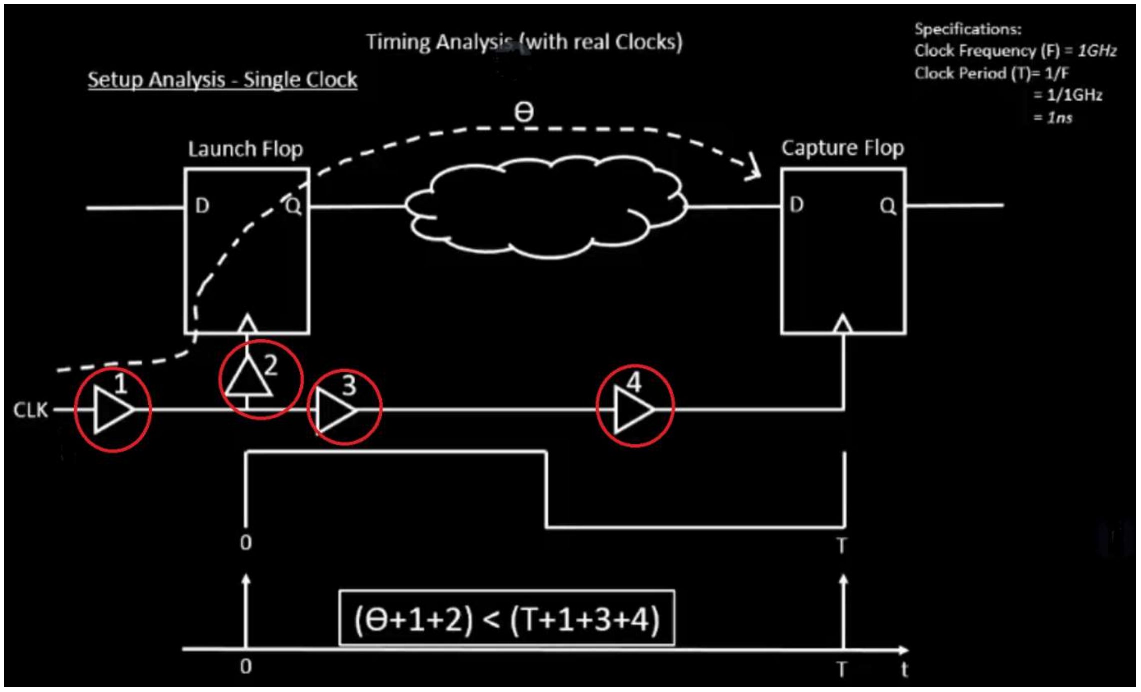
- **CTS_CLK_BUFFER_LIST**: This variable lists the clock buffers used in the branches of the clock tree, like:
  - sky130_fd_sc_hd__clkbuf_1
  - sky130_fd_sc_hd__clkbuf_2
  - sky130_fd_sc_hd__clkbuf_4
  - sky130_fd_sc_hd__clkbuf_8
- **CTS_ROOT_BUFFER**: This specifies the clock buffer used for the root of the clock tree, which is the largest buffer designed to drive the clock signal across the whole chip. For example:
  - sky130_fd_sc_hd__clkbuf_16
- **CTS_MAX_CAP**: This is a numerical value defining the maximum capacitance for the output port of the root clock buffer, which is important for determining the size of the buffer and ensuring signal integrity.

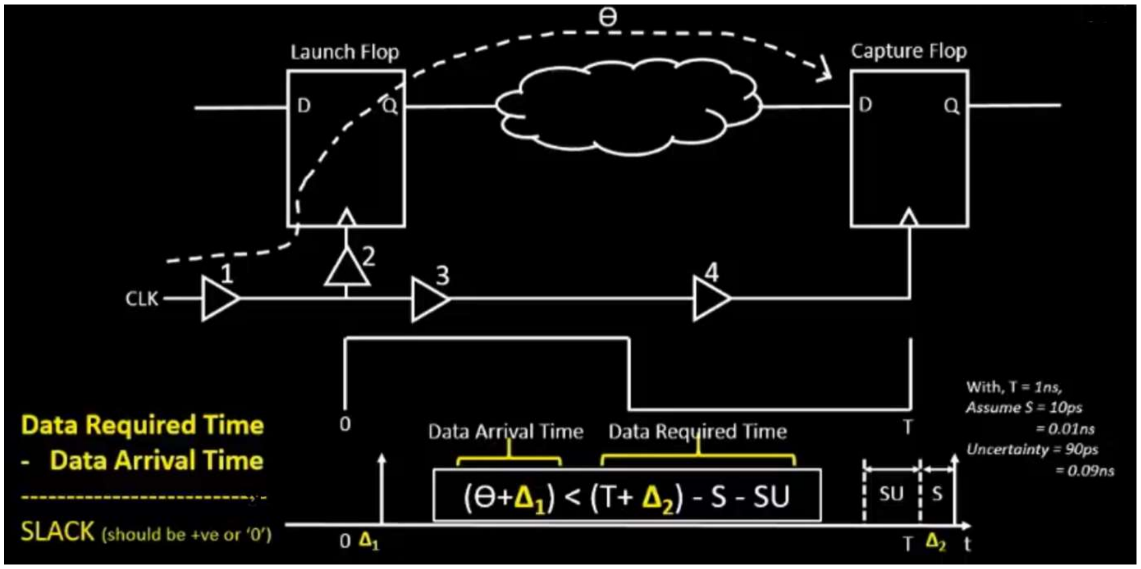
Timing Analysis With Real Clocks Using Open STA

Setup Timing Analysis Using Real Clocks

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

{IMAGE CREDITS: VSDIAT; SCREENSHOT TAKEN FROM LECTURE}





delta1 = launch flop clock network delay

delta2 = capture flop clock delay

Any design that satisfies the **Slack** condition (i.e., **Data required time - Data arrival time**) is ready to operate at the given frequency. In this case, **Slack** represents the time difference between when the data is required to be stable (data required time) and when the data actually arrives at the destination (data arrival time).

If this equation is violated (i.e., the **data arrival time** exceeds the **data required time**), the **slack becomes negative**, which is undesirable and indicates a timing violation.

Ideally, **slack** should be **zero** (perfectly timed) or **positive** (indicating extra time available for the signal to propagate). **Negative slack** signifies a timing failure, and steps must be taken (like optimizing delays or adjusting the design) to resolve the issue.

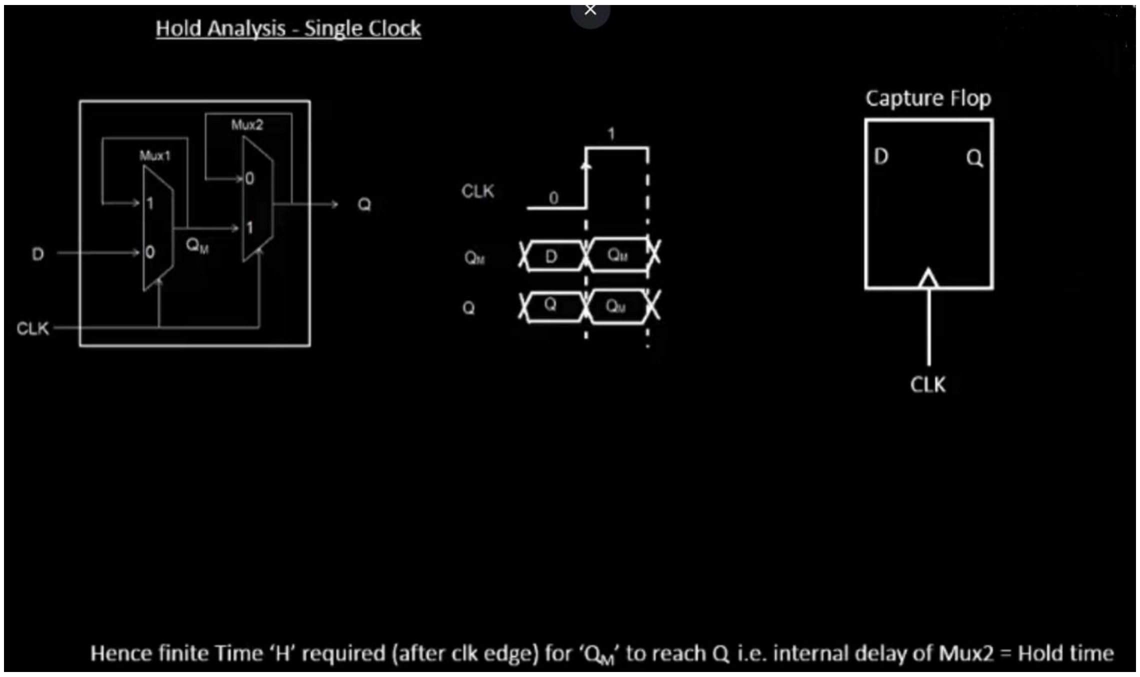
Hold Timing Analysis Using Real Clocks

Hold Time is delay [time] needed by the MUX2 model within a flip-flop to transfer certain data outside. [i.e. how long it **holds** the data]. It is the time period during which the launch flop must retain dat before it reaches the capture flop. Unlike setup analysis, which has two rising clock edges, hold analysis occurs on the same rising clock edge for both the launch and capture flops.

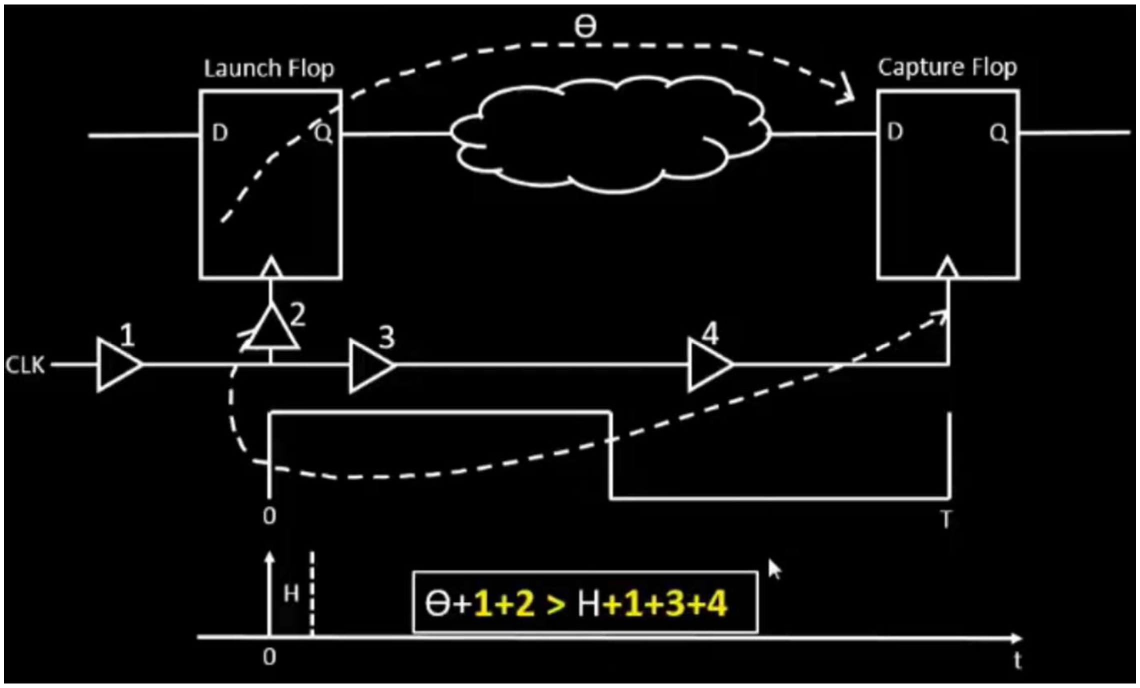
A hold violation occurs when the path is too fast, impacted by factors such as -:

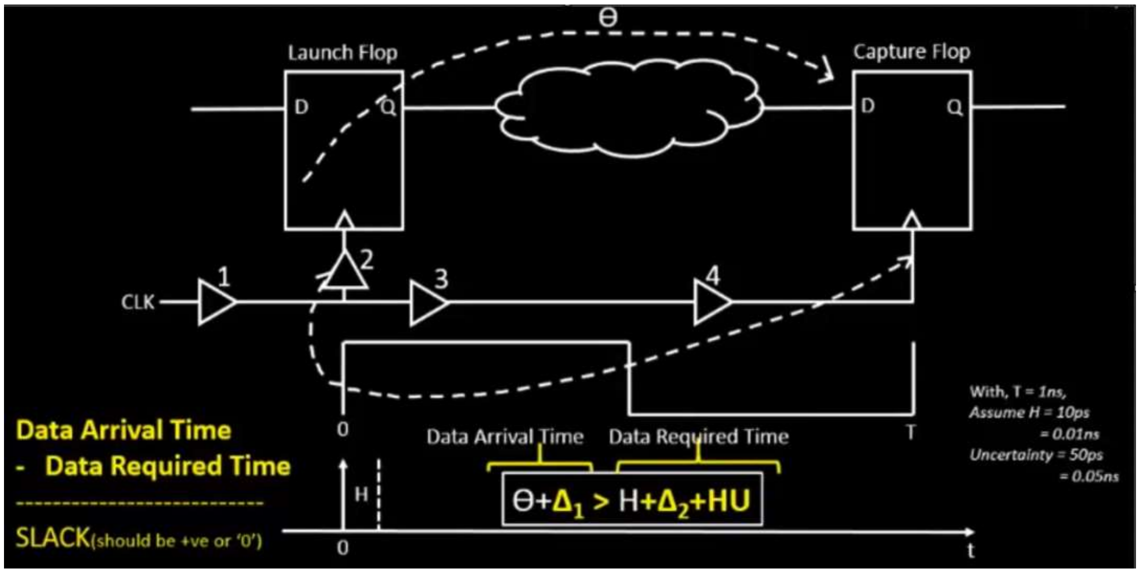
- combinational delay
- clock buffer delays
- hold time.

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.



Hence finite Time 'H' required (after clk edge) for 'Q_M' to reach Q i.e. internal delay of Mux2 = Hold time



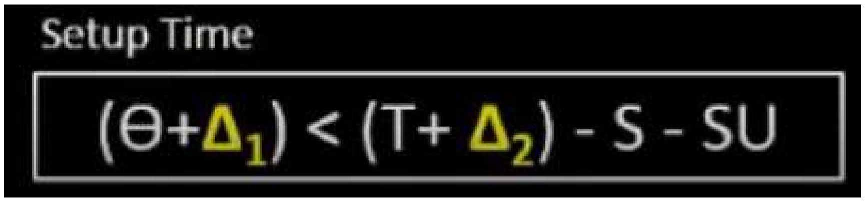


Data Arrival Time
- Data Required Time

SLACK(should be +ve or '0')

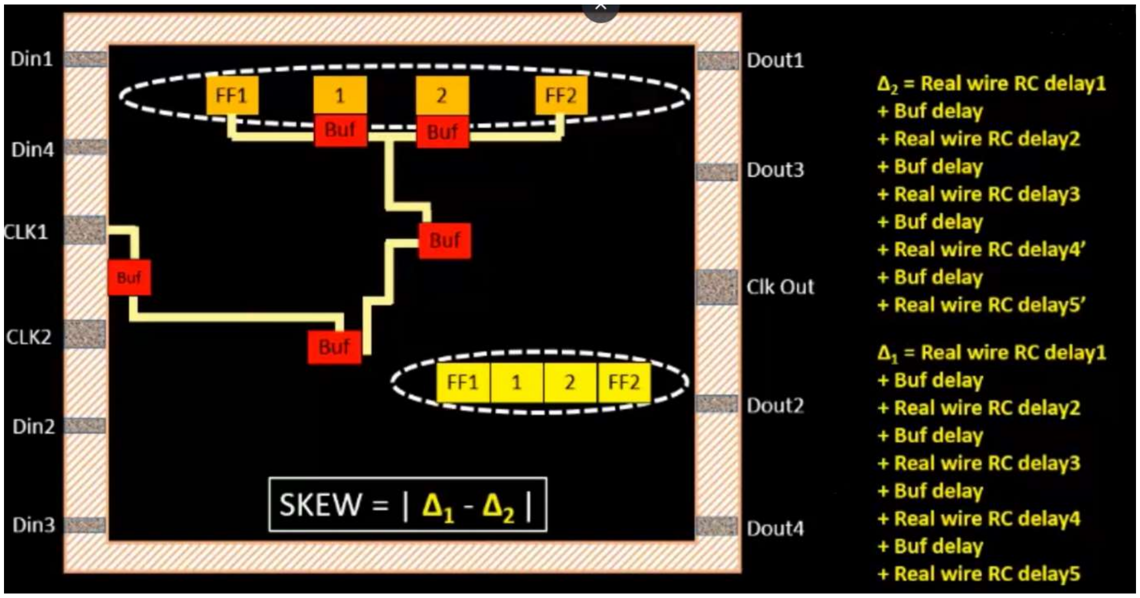
$$\underbrace{\Theta+\Delta_1}_{\text{Data Arrival Time}} > \underbrace{H+\Delta_2+HU}_{\text{Data Required Time}}$$

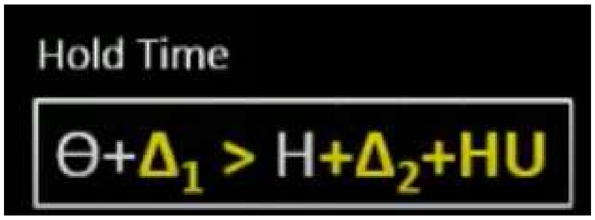
With, T = 1ns,
Assume H = 10ps
= 0.01ns
Uncertainty = 50ps
= 0.05ns



Setup Time

$$\boxed{(\Theta+\Delta_1) < (T+\Delta_2) - S - SU}$$




Hold Time

$$\boxed{\Theta+\Delta_1 > H+\Delta_2+HU}$$

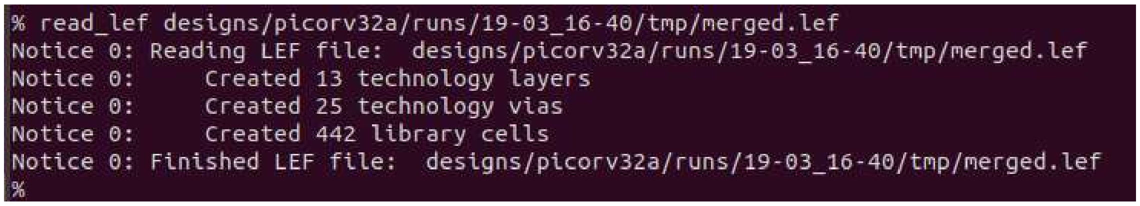
Lab Steps to Analyse Timing With Real Clocks Using OpenSTA

The aim is to analyse the clock tree explained previously. The steps for analysis of timings with real clocks are -:

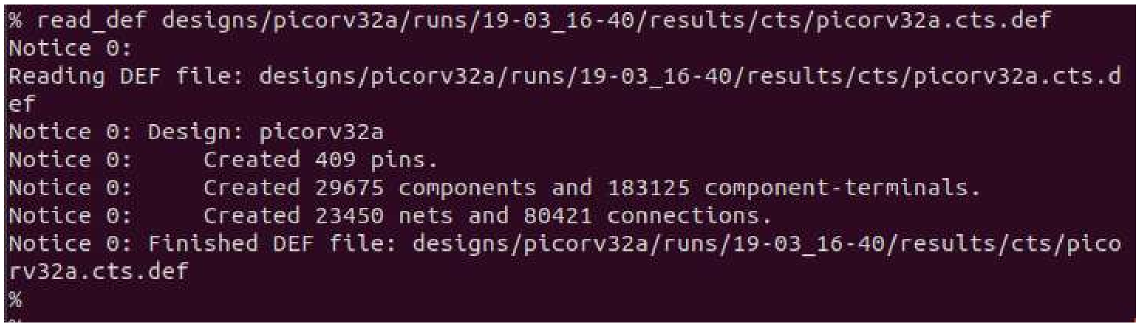
1. Enter into OpenROAD using the **openroad** command.

NOTE: In OpenROAD, timing is done slightly differently, where in a db is created from lef and def files and subsequently used

2. Subsequently create the db file through this command -: *read_lef [location of merged.lef]

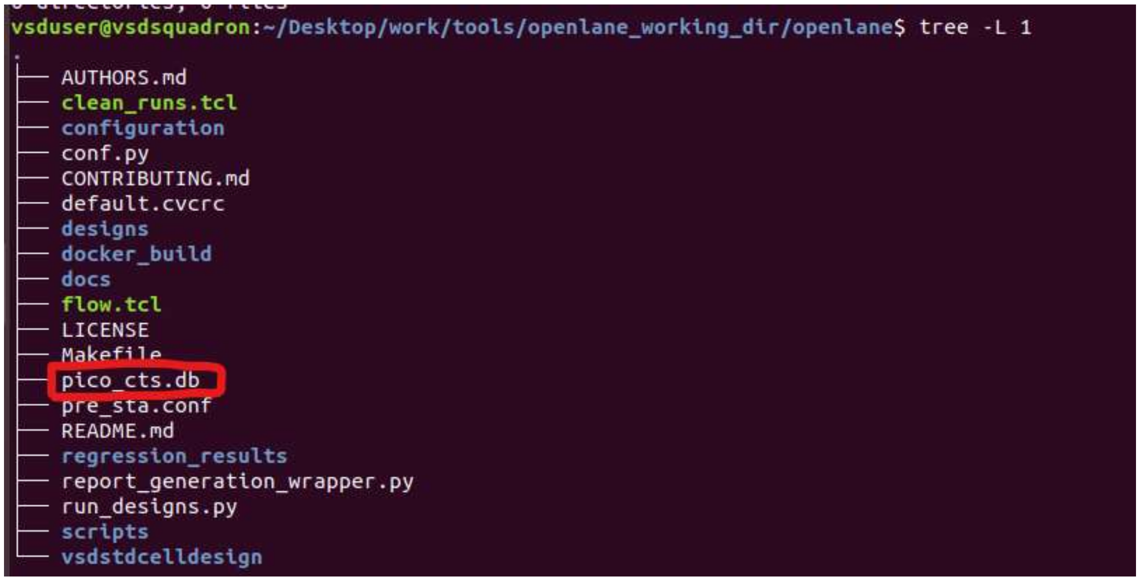

```
% read_lef designs/picorv32a/runs/19-03_16-40/tmp/merged.lef
Notice 0: Reading LEF file:  designs/picorv32a/runs/19-03_16-40/tmp/merged.lef
Notice 0:     Created 13 technology layers
Notice 0:     Created 25 technology vias
Notice 0:     Created 442 library cells
Notice 0: Finished LEF file:  designs/picorv32a/runs/19-03_16-40/tmp/merged.lef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

3. Then, we must read the def file from the CTS stage



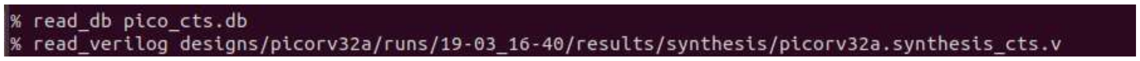
```
% read_def designs/picorv32a/runs/19-03_16-40/results/cts/picorv32a.cts.def
Notice 0:
Reading DEF file: designs/picorv32a/runs/19-03_16-40/results/cts/picorv32a.cts.d
ef
Notice 0: Design: picorv32a
Notice 0:     Created 409 pins.
Notice 0:     Created 29675 components and 183125 component-terminals.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

4. Subsequently create the db through this command -: *write_db pico_cts.db*

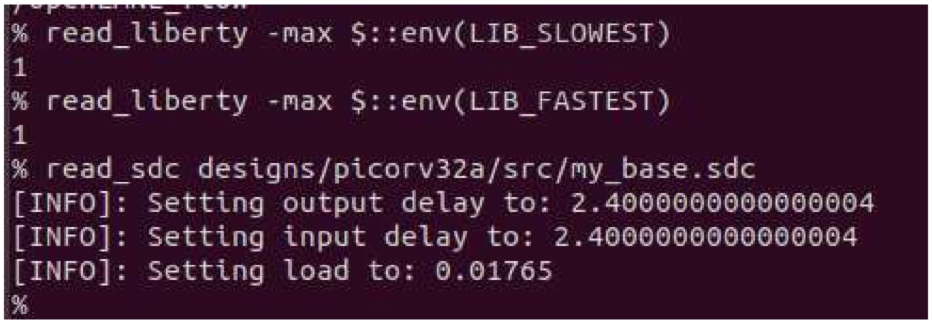


```
vsduser@vsdsquadron:~/Desktop/work/tools/openlane_working_dir/openlane$ tree -L 1
.
├── AUTHORS.md
├── clean_runs.tcl
├── configuration
├── conf.py
├── CONTRIBUTING.md
├── default.cvcrc
├── designs
├── docker_build
├── docs
├── flow.tcl
├── LICENSE
├── Makefile
├── pico_cts.db
├── pre_sta.conf
├──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

5. Then, read the db, verilog files, library and sdc



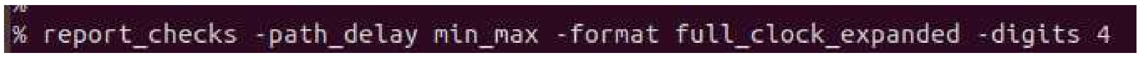
```
% read_db pico_cts.db
% read_verilog designs/picorv32a/runs/19-03_16-40/results/synthesis/picorv32a.synthesis_cts.v
```



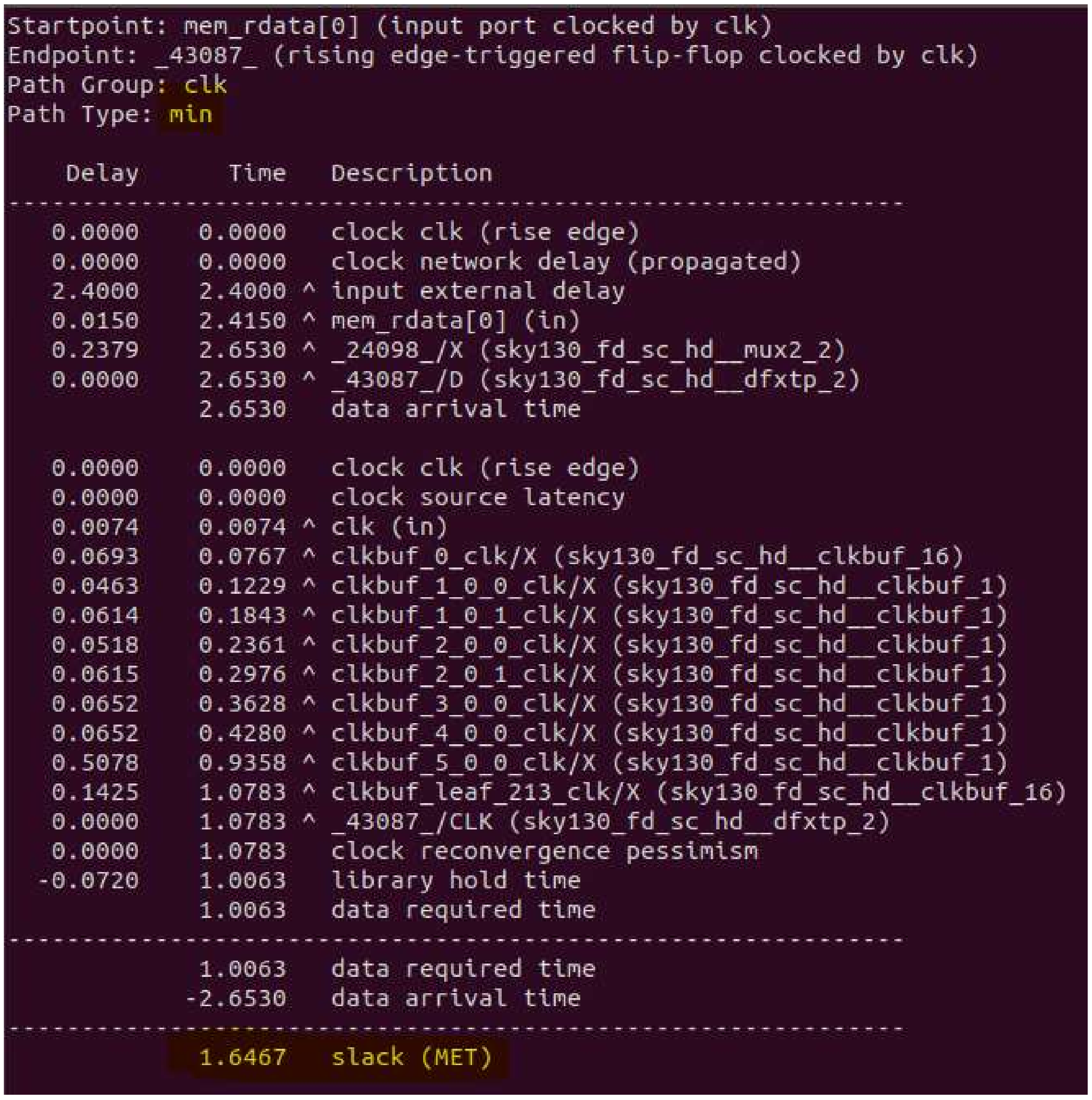
```
% read_liberty -max $::env(LIB_SLOWEST)
1
% read_liberty -max $::env(LIB_FASTEST)
1
% read_sdc designs/picorv32a/src/my_base.sdc
[INFO]: Setting output delay to: 2.4000000000000004
[INFO]: Setting input delay to: 2.4000000000000004
[INFO]: Setting load to: 0.01765
%
```

6. After this, set the propogated clock to calculate the actual delay in the clock path through the command -: *set_propogated_clock [all_clocks]*

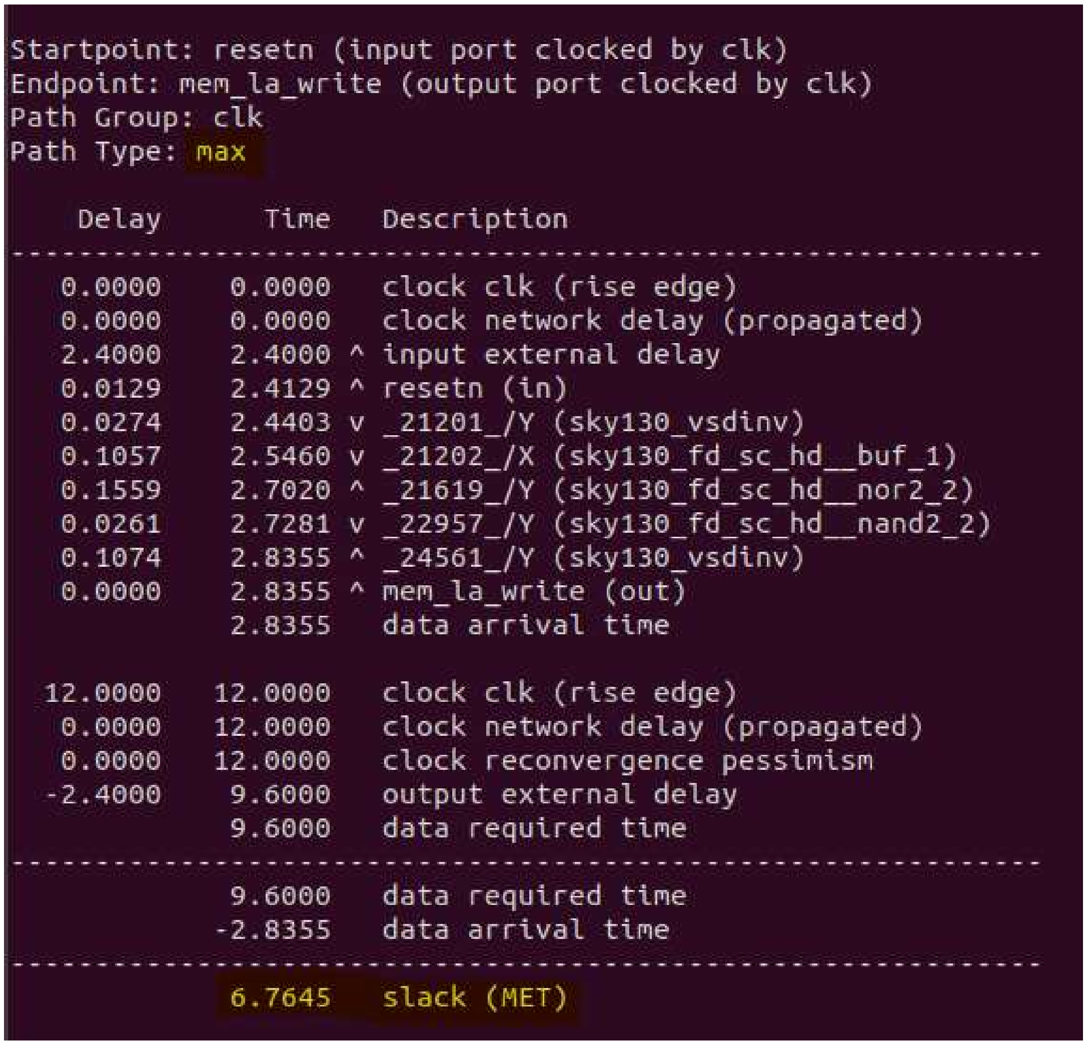
7. Subsequently check the timings -:



```
% report_checks -path_delay min_max -format full_clock_expanded -digits 4
```



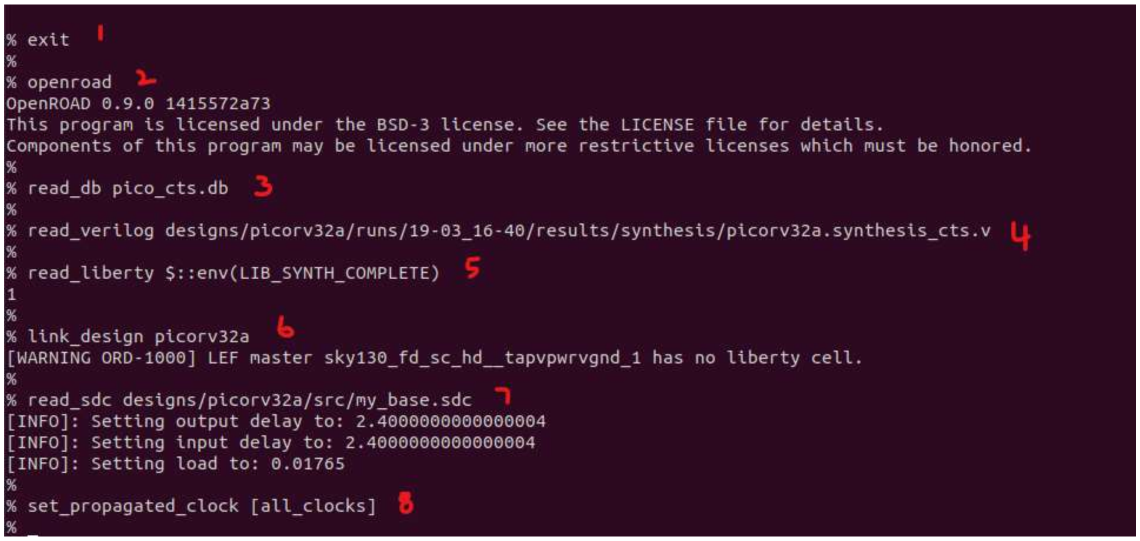
```
Startpoint: mem_rdata[0] (input port clocked by clk)
Endpoint: _43087_ (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: min

    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)
```



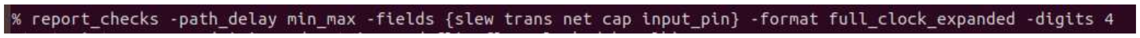
```
Startpoint: resetn (input port clocked by clk)
Endpoint: mem_la_write (output port clocked by clk)
Path Group: clk
Path Type: max

    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
   0.0129    2.4129 ^ resetn (in)
   0.0274    2.4403 v _21201_/Y (sky130_vsdinv)
   0.1057    2.5460 v _21202_/X (sky130_fd_sc_hd__buf_1)
   0.1559    2.7020 ^ _21619_/Y (sky130_fd_sc_hd__nor2_2)
   0.0261    2.7281 v _22957_/Y (sky130_fd_sc_hd__nand2_2)
   0.1074    2.8355 ^ _24561_/Y (sky130_vsdinv)
   0.0000    2.8355 ^ mem_la_write (out)
             2.8355   data arrival time

  12.0000   12.0000   clock clk (rise edge)
   0.0000   12.0000   clock network delay (propagated)
   0.0000   12.0000   clock reconvergence pessimism
  -2.4000    9.6000   output external delay
             9.6000   data required time
-------------------------------------------------------------
             9.6000   data required time
            -2.8355   data arrival time
-------------------------------------------------------------
             6.7645   slack (MET)
```

Lab Steps to Excecute OpenSTA With Right Timing Libraries and CTS Assignment

TritonCTS is build to optimise only based on one corner. However, the libraries included previously that also take part in timing analysis optimise based on both min and max corners, which is not accurate. Hence, we need to exit and re-enter OpenROAD and then check timing only for typical corner.

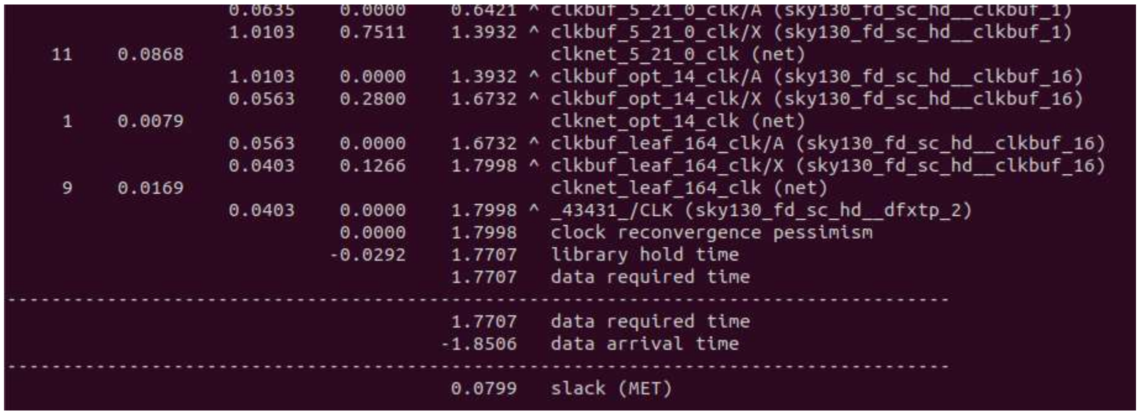


```
% exit  1
%
% openroad  2
OpenROAD 0.9.0 1415572a73
This program is licensed under the BSD-3 license. See the LICENSE file for details.
Components of this program may be licensed under more restrictive licenses which must be honored.
%
% read_db pico_cts.db  3
%
% read_verilog designs/picorv32a/runs/19-03_16-40/results/synthesis/picorv32a.synthesis_cts.v  4
%
% read_liberty $::env(LIB_SYNTH_COMPLETE)  5
1
%
% link_design picorv32a  6
[WARNING ORD-1000] LEF master sky130_fd_sc_hd__tapvpwrvgnd_1 has no liberty cell.
%
% read_sdc designs/picorv32a/src/my_base.sdc  7
[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]  8
% _
```

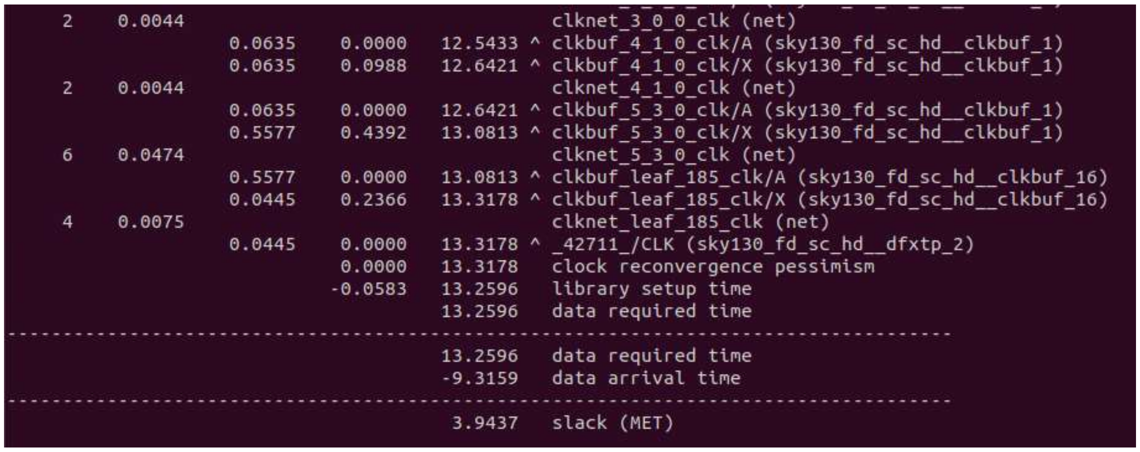


```
% report_checks -path_delay min_max -fields {slew trans net cap input_pin} -format full_clock_expanded -digits 4
```

Here, we can see that slack is met for both setup and hold analysis in the typical scenario-:



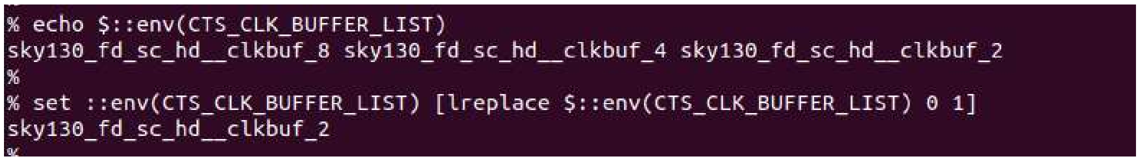
```
                        0.0635    0.0000    0.6421 ^ clkbuf_5_21_0_clk/A (sky130_fd_sc_hd__clkbuf_1)
                        1.0103    0.7511    1.3932 ^ clkbuf_5_21_0_clk/X (sky130_fd_sc_hd__clkbuf_1)
    11    0.0868                                     clknet_5_21_0_clk (net)
                        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 building the clock tree, **TritonCTS** uses buffers from $::env(CTS_CLK_BUFFER_LIST) (e.g., sky130_fd_sc_hd__clkbuf_1 to sky130_fd_sc_hd__clkbuf_8) to meet the target skew. The target skew is stored in $::env(CTS_TARGET_SKEW). **STA results** show that sky130_fd_sc_hd__clkbuf_1 is most used, but you can modify $::env(CTS_CLK_BUFFER_LIST) to use other buffers and assess the impact on **STA** and **area**. The **tcl lreplace** command can be used to modify the list.

Example:



```
% echo $::env(CTS_CLK_BUFFER_LIST)
sky130_fd_sc_hd__clkbuf_8 sky130_fd_sc_hd__clkbuf_4 sky130_fd_sc_hd__clkbuf_2
%
% set ::env(CTS_CLK_BUFFER_LIST) [lreplace $::env(CTS_CLK_BUFFER_LIST) 0 1]
sky130_fd_sc_hd__clkbuf_2
```

The *$::env(CURRENT_DEF)* used by CTS is the DEF file of the previously run CTS, but the DEF file we want for CTS is the placement's DEF file. So to implement this, change the **$::env(CURRENT_DEF)** to point to placement DEF file then *run_cts*.

After modifying the clock buffer, observe the post-CTS STA. Now, only buf_2 is used instead of buf_1 from the previous run. The WNS (Worst Negative Slack) is improved because larger clock buffers (buf_2) are now being used, resulting in better timing performance.