Project Report On

Characterization of a Standard Cell Library

Submitted by

Ayush Saran (181EE114)

Subham Mohapatra (181EE147)

Maji Soma Varun (181EE226)

VI SEM B.Tech (EEE)

Under the guidance of

Professor Ramesh Kini

Dept of ECE, NITK Surathkal

in partial fulfillment for the award of the degree

of

Bachelor of Technology

In

Electrical and Electronics Engineering

at

Department of Electrical and Electronics Engineering

National Institute of Technology Karnataka, Surathkal. (April '21)

Abstract -

VLSI has adapted to the open source revolution well with one of it's strongest features, ASIC flow being driven by popular open source technologies. OpenLANE is now an industry-hold name, that is primarily tasked with walking the user through the process of bringing their hardware solutions to life, all the way from it's design phase in RTL all the way to the GDS stage.

Hardware design at its root level is run by gates. Gates can be considered the building blocks of any hardware design. This project aims to focus on the custom design of 7 of these such cells, with varying drive strengths and compiling a Standard Cell Library. The technology assisting us in this project is the Skywater sky130 open source technology. This Library will be compiled in formats that can be used by proprietary and open source tools so as to enable the open source community to make use of our customized standard cells, whose layouts are designed with the help of the Magic toolset, in any of their desired hardware designs. The library consists of the NAND, NOR, AND, OR, NOT, XOR and Buffer cells each with drive strengths of 1x, 2x and 4x totalling our library's strength to 21 cells in total. The design of these cells is followed by a swift simulation of each on ngspice to validate the designs.

These cells have been further characterized (timing and internal power) by means of ngspice, the process being further automated using python3. This characterization process gives us an idea about the behaviour of each of these cells. If characterized correctly, this information can display the upper and lower bounds of the functionalities and limitations subject to the cell, which in turn affects the overall working of the hardware design in question.

Furthermore, to validate our results, we have implemented a total of 5 designs, one of which is explained in detail in this report, using only our standard cells from the synthesis stage all the way up to generating the final GDSII file for the same. We have worked on the interactive interface option offered by OpenLANE and gone stage by stage, in steps to demonstrate how to include our cells in the mix and still effectively form a complete design.

Theory/Methodology

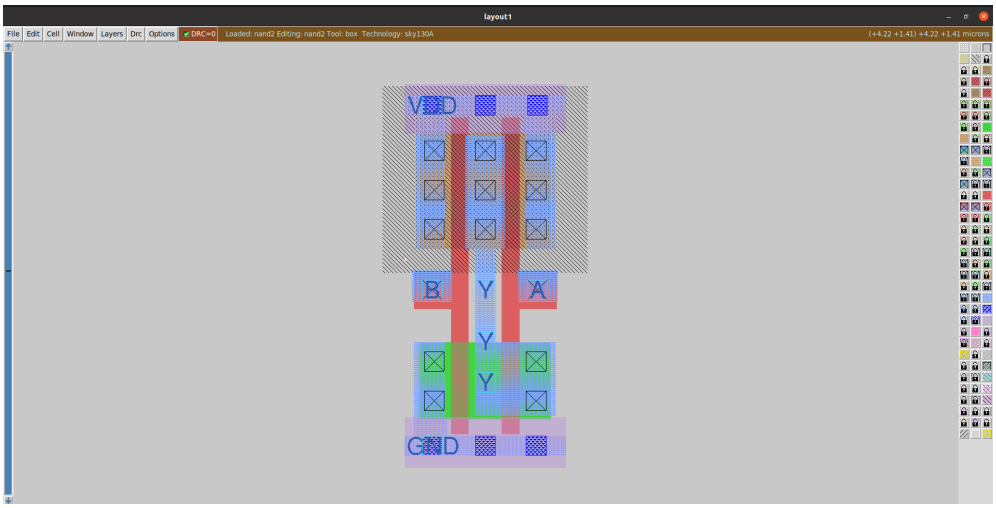
1.Magic layout design

The Layout of all the standard cells are done in the Magic tool. To invoke magic we use the command :

$ magic -T <tech_file> <design_file>

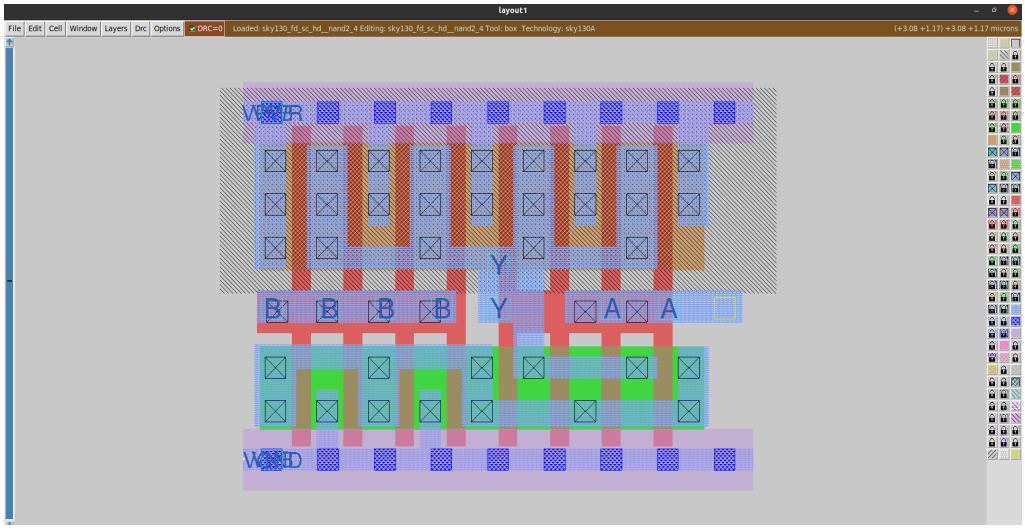
Example $ magic -T sky130.tech nand2.mag

The layout of a 2 input NAND gate of drive strength 1 is as follows:



Design Rule Checking (DRC) verifies as to whether a specific design meets the constraints imposed by the process technology to be used for its manufacturing. DRC checking is an essential part of the physical design flow and ensures the design meets manufacturing requirements and will not result in a chip failure.We should make sure that the Design Rule Check(DRC) error is 0. This ensures that our design doesn't violate any design constraints. For better layout always follow Euler's path while drawing the layout. The DRC error parameter can be checked in the top left corner of the Magic interface.

For higher drive strength the layout needs to be modified accordingly. The layout for 2 input NAND gate with drive strength 4 :



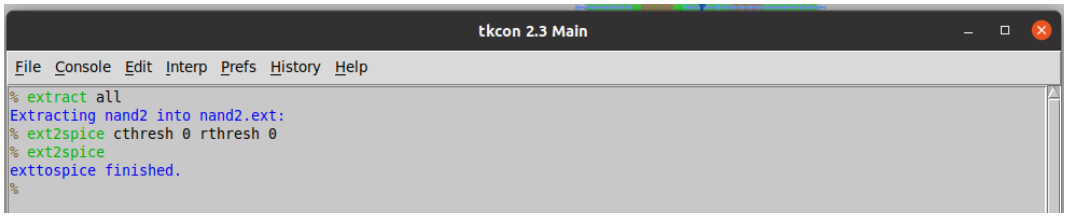
The layouts of the cells are done for the sky130_fd_sc_hd library. This library is concerned with "high density" (hd) which focuses on optimizing it's cells to maximise the density of cells placed on the IC core.

The unit inverter was taken as a reference to determine the necessary parameters. We've referred to *tracks.info* file in *pdks/open_pdks/sky130A/libs.tech/openlane/sky130_fd_sc_hd* directory.
Referred to metal and interconnect layer parameters, as shown below:-

```
li1 X 0.23 0.46
li1 Y 0.17 0.34
met1 X 0.17 0.34
met1 Y 0.17 0.34
met2 X 0.23 0.46
met2 Y 0.23 0.46
met3 X 0.34 0.68
met3 Y 0.34 0.68
met4 X 0.46 0.92
met4 Y 0.46 0.92
met5 X 1.70 3.40
met5 Y 1.70 3.40
```

After the layout is done we need to extract the spice netlist and simulate it in ngspice.

1. In the terminal, the following commands, in that order, should be entered to extract the SPICE information:-
   % extract all
   % ext2spice cthresh 0 rthresh 0
   % ext2spice



The LEF file has the information about input ports, output ports , ground and power ports. These are the only information needed for place and route. LEF files in a way protect our IP. Our aim is to extract lef file from the mag file.

Input and output port must lie on vertical and horizontal tracks. Width of standard cell should be odd multiples of track horizontal pitch and likewise height should also be an odd multiple of track vertical pitch.

We need to extract the lef file from the layout. Command to create .lef file is -> "lef write"

```
tkcon 2.3 Main
File  Console  Edit  Interp  Prefs  History  Help
% lef write
Generating LEF output nand2.lef for cell nand2:
Diagnostic:  Write LEF header for cell nand2
Diagnostic:  Writing LEF output for cell nand2
Diagnostic:  Scale value is 0.010000
%
```

The lef file extracted is shown below:-

```
Open                                                    nand2.lef
1 VERSION 5.7 ;
2  NOWIREEXTENSIONATPIN ON ;
3  DIVIDERCHAR "/" ;
4  BUSBITCHARS "[]" ;
5 MACRO nand2
6  CLASS BLOCK ;
7  FOREIGN nand2 ;
8  ORIGIN 0.190 1.790 ;
9  SIZE 1.770 BY 3.310 ;
10 PIN B
11   ANTENNAGATEAREA 0.249000 ;
12   PORT
13     LAYER li1 ;
14       RECT 0.060 -0.360 0.400 -0.090 ;
15   END
16 END B
17 PIN A
18   ANTENNAGATEAREA 0.249000 ;
19   PORT
20     LAYER li1 ;
21       RECT 0.980 -0.360 1.320 -0.090 ;
22   END
23 END A
24 PIN GND
25   ANTENNADIFFAREA 0.178200 ;
26   PORT
27     LAYER li1 ;
28       RECT 0.080 -1.370 0.350 -0.710 ;
29       RECT 0.080 -1.520 0.360 -1.370 ;
30       RECT 0.000 -1.690 1.390 -1.520 ;
31     LAYER mcon ;
32       RECT 0.150 -1.690 0.320 -1.520 ;
33       RECT 0.610 -1.690 0.780 -1.520 ;
34       RECT 1.050 -1.690 1.220 -1.520 ;
35     LAYER met1 ;
36       RECT 0.000 -1.790 1.390 -1.360 ;
37   END
38 END GND
39 PIN VDD
40   ANTENNADIFFAREA 0.540000 ;
41   PORT
42     LAYER li1 ;
43       RECT 0.000 1.250 1.390 1.420 ;
44       RECT 0.090 0.100 0.350 1.250 ;
45       RECT 1.030 1.100 1.300 1.250 ;
46       RECT 1.040 0.100 1.300 1.100 ;
47     LAYER mcon ;
48       RECT 0.150 1.250 0.320 1.420 ;
49       RECT 0.610 1.250 0.780 1.420 ;
50       RECT 1.060 1.250 1.230 1.420 ;
```

We can include this lef file in OpenLANE so that our standard cells are integrated into it.

We repeat this process for all the other standard cells involved in the building of this library.

2. Characterization of Standard Cells

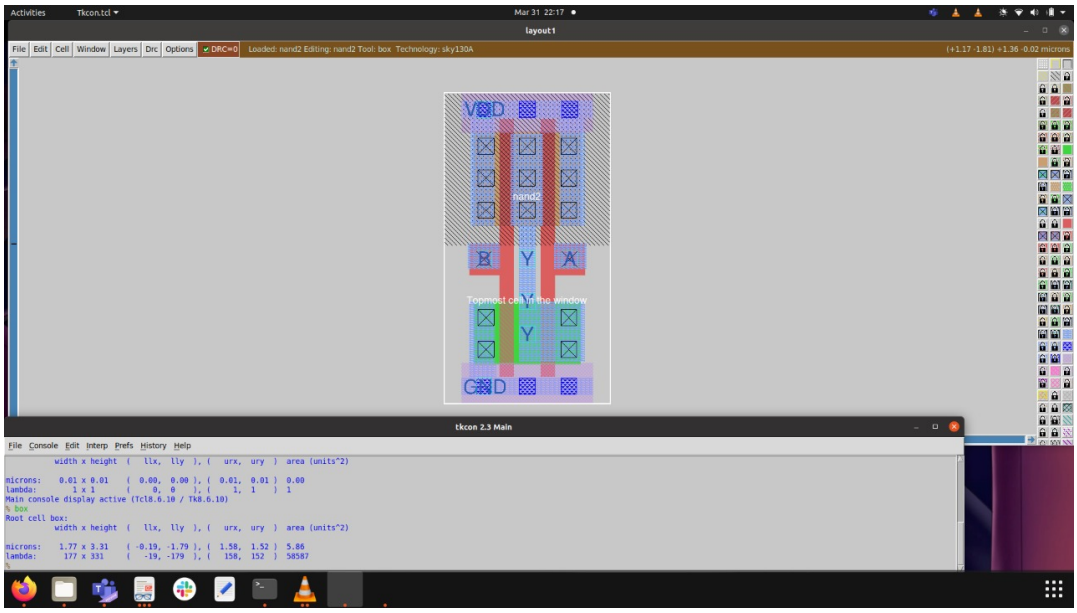
The beauty and inclusiveness of open-source has allowed us to experiment with multiple approaches to finally characterizing the cells we've designed.
Characterization of a Standard Cell, is the process of gaining a behavioral understanding of the cells in question. Once the cells have been designed, following all the necessary steps and rules that govern the designing process, it becomes elemental to understand the nature of your cells, and to determine the executable parameters, specifically related to power and time which define the behaviour of the cell. This behaviour is subject to many changes, could be temperature, could be voltage/current spikes, could be a plethora of situations and this characteristic report on the cells helps us achieve an educated understanding of the behaviour of our cells, subject to such changes.
Now the question is why are these indicators important for each and every cell. Circuits have become more dense by the year and even though a generational chip contains billions of transistors and millions of sub-circuits, at the root level the cells form the building blocks of these circuits and each and every cell plays an important role in the whole flow. Characteristics such as speed, power and timing of each cell are directly related to the parameters exhibited by the entire circuit.
The Liberty File Format is used to store information about a set of cells. In our project we aim to design and test a collection of 21 standard cells, namely the NAND, NOR, AND, OR, XOR, NOT and BUFFER, each with varying drive strengths, (1x, 2x, 4x). These cells have been designed by us constrained by the design rules described for the sky130A process design kit. Each of these cells have been characterized accordingly and without bias for any one. We've focussed on the timing characteristics of the cells. This governs their delays, constraints and edge rates.

We make use of an open-source repository which automates the process of characterization using ngspice for a single cell. Future work will see us extend this program to incorporate multiple cells in a single program. This automation is carried out by means of a python script which invokes ngspice to do the requiem for the project.

The ngspice script to characterize the timing characteristics of each and every cell has been dissected as below.

Stored as .cir files, the beginning steps of this process begin with the tool "Magic".
Once the design of the layout is completed in Magic, we then use the command terminal provided by this application to extract the SPICE information of the required cell. This is achieved as follows :-

1. In the Magic terminal, select the entire cell designed.



2. In the terminal, the following commands, in that order, should be entered to extract the SPICE information:-
   % extract all
   % ext2spice cthresh 0 rthresh 0
   % ext2spice

This will generate a .spice file by the name entered while designing the file. Hence the .mag and .spice files will share the same name.

Once done, we can extract the netlist generated by Magic and begin our scripting in ngspice.

3. We run a basic ngspice simulation passing the necessary parameters to check the behaviour of the cell designed. We make use of a NAND cell to demonstrate our results. Hence, for the simulation to testify the nature of working, we set a PULSE and a fixed voltage signal to determine the behaviour.

```
V1 Vin 0 PULSE (0 1.8 0 0.01n 0.01n 50ns 100ns)
V2 VDD 0 dc 1.8V
```

4. It's imperative to understand that certain spice models are required to be imported to make sure the characterization script works.
   The libraries are extracted from the Primitive folder in sky130, and are namely the tt_corner spice models from pfet and nfet files, coupled with the mismatch.corner.spice files from both again.
   The other 3 required go by all.spice, rf.spice, nonfet.spice.

```
* SKY130 Spice File.

.include "../cells/nfet_01v8/sky130_fd_pr__nfet_01v8__tt.corner.spice"

.include "../cells/pfet_01v8/sky130_fd_pr__pfet_01v8__tt.corner.spice"

.include "../cells/nfet_01v8/sky130_fd_pr__nfet_01v8__mismatch.corner.spice"
.include "../cells/pfet_01v8/sky130_fd_pr__pfet_01v8__mismatch.corner.spice"

.include "corners/tt/rf.spice"
.include "all.spice"
.include "corners/tt/nonfet.spice"
```

5. The netlist extracted from Magic is inserted into the scripting file and should resemble as follows :-

```
XM1 Vout Vin VDD VDD sky130_fd_pr__pfet_01v8 w=2 l=0.5
XM2 Vout B VDD VDD sky130_fd_pr__pfet_01v8 w=2 l=0.5
XM3 Vout Vin Vx 0 sky130_fd_pr__nfet_01v8 w=1 l=0.5
XM4 Vx B 0 0 sky130_fd_pr__nfet_01v8 w=1 l=0.5
CLOAD Vout 0 10pf
```

   Do note the presence of XMi (i=1,2,3,4). The netlist will generate it as Mi and they should be corrected to the XMi form.

6. The first step is to figure out where and how in your system you are going to store the characterization results. We've opted to go for a traditional .txt file and thus write basic file handling code to generate the necessary text files for each characterization result (cell_rise, cell_fall, rise_transition, fall_transition, rise_power, fall_power).

```
** Initiating Text Files in folder data
echo "input_delay:$in_delay" >> /home/majji_open/vlsi/practice/Ngspice_sim/cmos_cir1/input_delay.txt
echo "input_delay:$in_delay" >> /home/majji_open/vlsi/practice/Ngspice_sim/cmos_cir1/cell_fall.txt
echo "input_delay:$in_delay" >> /home/majji_open/vlsi/practice/Ngspice_sim/cmos_cir1/cell_rise.txt
echo "input_delay:$in_delay" >> /home/majji_open/vlsi/practice/Ngspice_sim/cmos_cir1/fall_transition.txt
echo "input_delay:$in_delay" >> /home/majji_open/vlsi/practice/Ngspice_sim/cmos_cir1/rise_transition.txt



echo "input_delay:$in_delay" >> /home/majji_open/vlsi/practice/Ngspice_sim/cmos_cir1/rise_power.txt

echo "input_delay:$in_delay" >> /home/majji_open/vlsi/practice/Ngspice_sim/cmos_cir1/fall_power.txt
```

7. It's important to note that we characterize these cells for fixed values of input delays and output capacitances. These inputs are looped over to find the results for each combination of inputs. The initiation of these inputs is coded as below :-

```
let run  = 0
foreach in_delay  0.01n 0.023n 0.0531329n 0.122474n 0.282311n 0.650743n 1.5n

foreach out_cap 0.0005p 0.0012105800p 0.002931p 0.00709641p 0.0171815p 0.0415991p 0.100718p
```

8. The next step is the characterization bit. After consulting many documentations we arrived at the "meas" command, which plays a crucial role in characterizing the various parameters of the cell.
   Once computed, the results are then translated into the .txt files created in the beginning.
   This text file holds the final output that we require.
   The code for the above is represented as follows :-

```
**Cell rise delay
meas tran crise TRIG v(Vin) VAL=0.9 FALL=1 TARG v(Vout) VAL=0.9 RISE=1
print crise
echo "out_cap:$out_cap:cell_rise:$&crise" >> /home/majji_open/vlsi/practice/Ngspice_sim/cmos_cir1/cell_rise.txt
```

9. We've documented the text files to be represented in an organized fashion, by mentioning the input delay first, and displaying the computed values of the associated parameters beside the output_capacitance value. Here is an example taken from cell_rise.txt :-

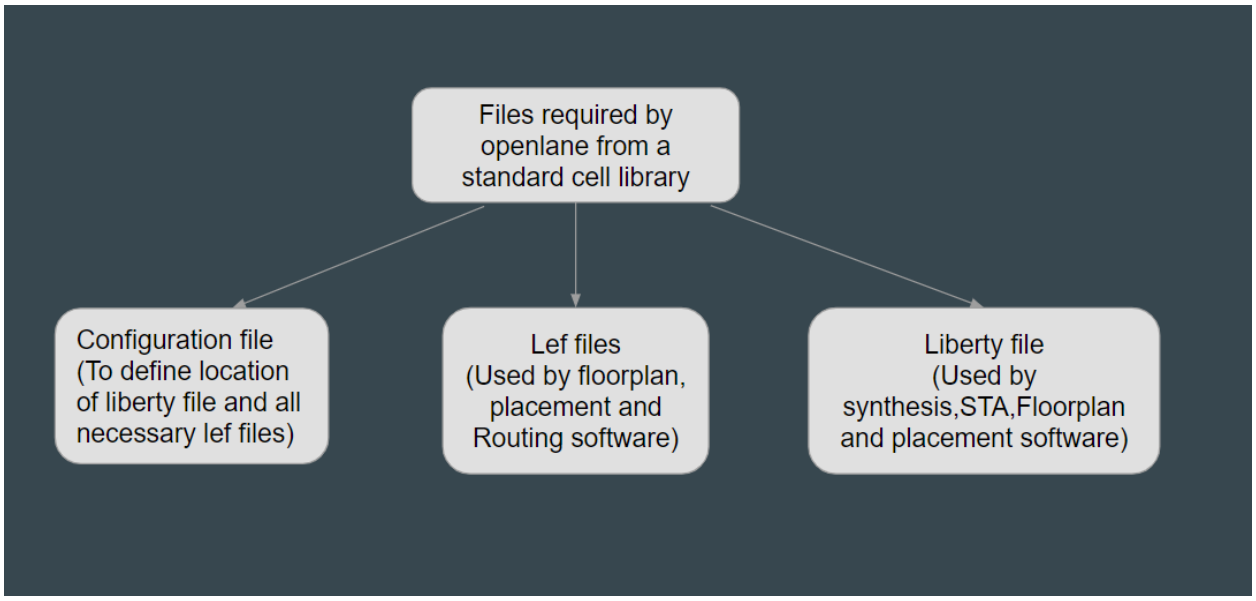
```
input_delay:0.023n
out_cap:0.0005p:cell_rise:6.38313E-11
out_cap:0.0012105800p:cell_rise:8.64024E-11
out_cap:0.002931p:cell_rise:1.3341E-10
out_cap:0.00709641p:cell_rise:2.22329E-10
out_cap:0.0171815p:cell_rise:4.18571E-10
out_cap:0.0415991p:cell_rise:8.82292E-10
out_cap:0.100718p:cell_rise:2.00006E-09
```

This shows that for an input delay of 0.023 nanoseconds, the cell_rise values are as follows for the given values of out_cap's (in picoFarads).
Hence a combination of input_delay = 0.023 n, and out_cap = 0.0005p, gives a cell_rise of 6.383131e-11 seconds.

This concludes the characterization of the cell.

The python script has been used to automate all the necessary steps without manual intervention, although manually going through the entire process was a good learning curve to understand the backend working of ngspice, Magic and other open sourced associated tools.

3. Integrating our Standard Cell Library into OpenLANE



Steps

1.First have to make sure LEF file contains all necessary cells other than standard cells.Include these cells names in Configuration file of respective Design.

Well Tap cells and decap cells

Well tap cells (or Tap cells) are used to prevent the latch-up issue in the CMOS design. Well tap cells connect the nwell to VDD and p-substrate to VSS in order to prevent the latch-up issue.

Decap cells are temporary capacitors added in the design between power and ground rails to counter functional failures due to dynamic IR drop

Below figure contains the names of cells that we are using for tap and decap cells from the standard library sky130_fd_sc_hd.

```
# welltap and endcap cells
set ::env(FP_WELLTAP_CELL) "sky130_fd_sc_hd__tapvpwrvgnd_1"
set ::env(FP_ENDCAP_CELL) "sky130_fd_sc_hd__decap_3"
```

Driving cell

Driving cell is the external driver that drives an input port has impedance and parasitic load characteristics that can affect the signal timing.
Below figure contains the names of cells that we are using for driving cell from the custom library.

```
# defaults (can be overridden by designs):
set ::env(SYNTH_DRIVING_CELL) "sky130_fd_sc_hd__inv_4"
#capacitance : 0.017653;
set ::env(SYNTH_DRIVING_CELL_PIN) "Y"
```

Antenna diodes
These diodes are added inorder to prevent antenna effect.Initially fake diodes are added everywhere during detailed placement stage if antenna effect is observed after routing stage then those diodes are replaced with real diodes in routing stage.
Below figure contains the names of cells that we are using for Diode cells from the standard library sky130_fd_sc_hd.

```
# Diode insertaion
set ::env(DIODE_CELL) "sky130_fd_sc_hd__diode_2"
set ::env(FAKEDIODE_CELL) "sky130_ef_sc_hd__fakediode_2"
set ::env(DIODE_CELL_PIN) "DIODE"
```

Clock Tree Buffers
Clock tree synthesis is a procedure to construct a clock path between various flops so that the slew is made minimum in a circuit.Here there are various clock tree structure and in openlane Triton CTS uses H-tree structure.Here in Clock Tree buffer list variable we mention various clock tree buffer sizes available of which we can chose during clock tree synthesis.
Below figure contains the names of cells that we are using for clock buffers from the standard library sky130_fd_sc_hd.

```
# Clk Buffers info CTS data
set ::env(ROOT_CLK_BUFFER) sky130_fd_sc_hd__clkbuf_16
set ::env(CLK_BUFFER) sky130_fd_sc_hd__clkbuf_4
set ::env(CLK_BUFFER_INPUT) A
set ::env(CLK_BUFFER_OUTPUT) X
set ::env(CTS_CLK_BUFFER_LIST) "sky130_fd_sc_hd__clkbuf_1 sky130_fd_sc_hd__clkbuf_2 sky130_fd_sc_hd__clkbu
f_8"
```

2.Include Location of Liberty files,Lef files and Tech lef file location in Configuration file of the Design.

Here liberty file is used for synthesis and STA.Cell_lef files are used during floorplan,placement and routing.Tech lef file is a txt file which contains information of the layers and metals present in SKY130 technology which is used by Floorplan and routing stage .

```
set ::env(VERILOG_FILES) [glob $::env(DESIGN_DIR)/src/*.v]

# turn off clock
set ::env(CLOCK_TREE_SYNTH) 0
set ::env(CLOCK_PORT) ""
set ::env(SYNTH_STRATEGY) 1
set ::env(LIB_SYNTH) "./designs/spm/src/sky130_fd_sc_hd__tt_025C_1v80.lib"
set ::env(LIB_FASTEST) "./designs/spm/src/sky130_fd_sc_hd__tt_025C_1v80.lib"

set ::env(LIB_SLOWEST) "./designs/spm/src/sky130_fd_sc_hd__tt_025C_1v80.lib"
set ::env(TECH_LEF) "./designs/spm/src/sky130_tech.lef"
set ::env(CELLS_LEF) "./designs/spm/src/sky130.lef"
```

*3.After defining the config file ,place it in "**./openlane/designs/added/src**" location*

4.Then enter into openlane directory through terminal and enter DOC command to enter the bash cell of openlane.Then type "./flow.tcl -interactive" command to invoke openlane environment in interactive mode.

```
majji_open@majji-open:~/openlane_build_script/work/tools/openlane_working_dir/openlane$ export P
ild_script/work/tools/openlane_working_dir/pdks
majji_open@majji-open:~/openlane_build_script/work/tools/openlane_working_dir/openlane$ docker r
PDK_ROOT:$PDK_ROOT -e PDK_ROOT=$PDK_ROOT -u $(id -u $USER):$(id -g $USER) openlane:rc6
bash-4.2$ ./flow.tcl -interactive
```

*5.Then type "**package require openlane 0.9**" .This command opens TCL window to run openlane TCL scripts that control all vlsi design softwares embedded in openlane.Then Type **"prep -design added -tage run70".**This command includes the config file and verilog file design added into openlane and save the final run results in directory name **run70.***

```
% package require openlane 0.9
0.9
% prep -design added -tag run70
[INFO]: Using design configuration at /openLANE_flow/designs/added/config.tcl
[INFO]: Sourcing Configurations from /openLANE_flow/designs/added/config.tcl
[INFO]: PDKs root directory: /home/majji_open/openlane_build_script/work/tools/openlane_working_dir/pdks
[INFO]: PDK: sky130A
[INFO]: Setting PDKPATH to /home/majji_open/openlane_build_script/work/tools/openlane_working_dir/pdks/sk
[INFO]: Standard Cell Library: sky130_fd_sc_hd
[INFO]: Sourcing Configurations from /openLANE_flow/designs/added/config.tcl
[INFO]: Current run directory is /openLANE_flow/designs/added/runs/run70
[INFO]: Preparing LEF Files
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
[INFO]: Trimming Liberty...
[INFO]: Preparation complete
```

*5.Start Synthesis Procedure by typing "**run_synthesis**". Yosis is invoked which synthesize the netlist file targeting SKY130 technology.*

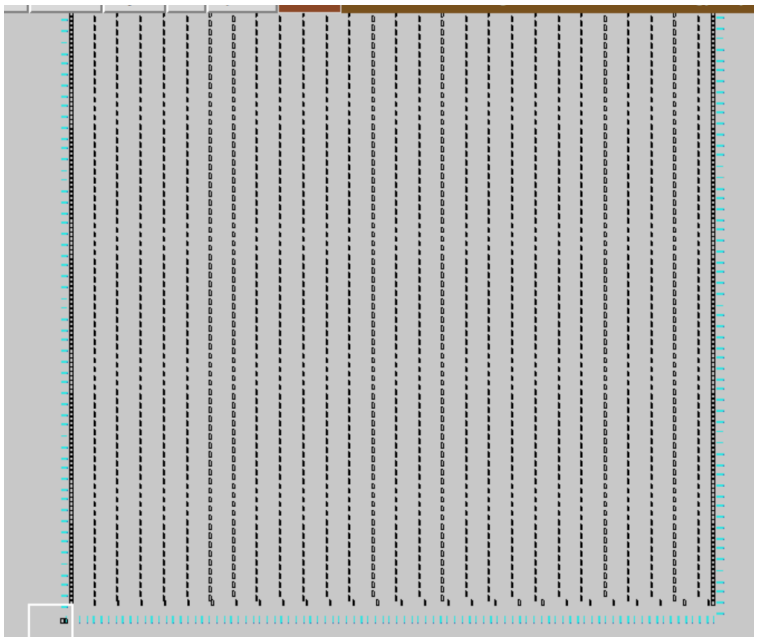
*Below figure shows the yosis report after final netlist generated which can be found in **"./openlane/designs/added/runs/run70/reports/synthesis"***
*Also synthesis verilog netlist file can be found in **"./openlane/designs/added/runs/run70/results/synthesis"***
Here we can see the custom cells of size 1x are used for our example design in final synthesis

```
21. Printing statistics.

=== added ===

   Number of wires:                854
   Number of wire bits:           1154
   Number of public wires:           3
   Number of public wire bits:     303
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                952
     and2_1                         53
     buf_1                           4
     inv_1                         123
     nand2_1                       425
     nor2_1                        143
     or2_2                          25
     xor2_1                        179

   Chip area for module '\added': 4664.473600
```

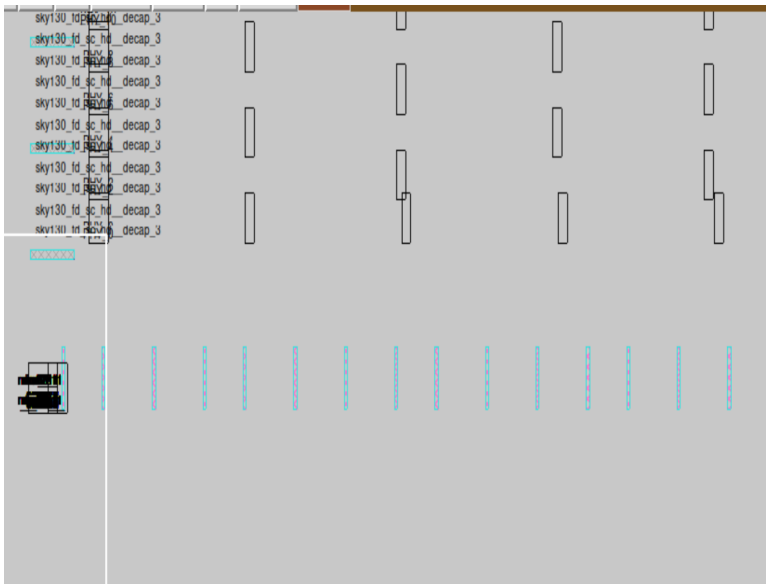
6.*Start Floorplan Procedure by typing "****run_floorplan****".In this stage the preplaced cells,input and output ports,decap cells and tap cells are placed in core area of die .In openlane the Power distribution network layout also happens in floorplan stage.Here we are using metal 4 and metal 5 of sky130 technology for Power distribution network .*

The Final floorplan def file can be found in ***"./openlane/designs/added/runs/results/run70/floorplan"*** *which can be opened in magic*

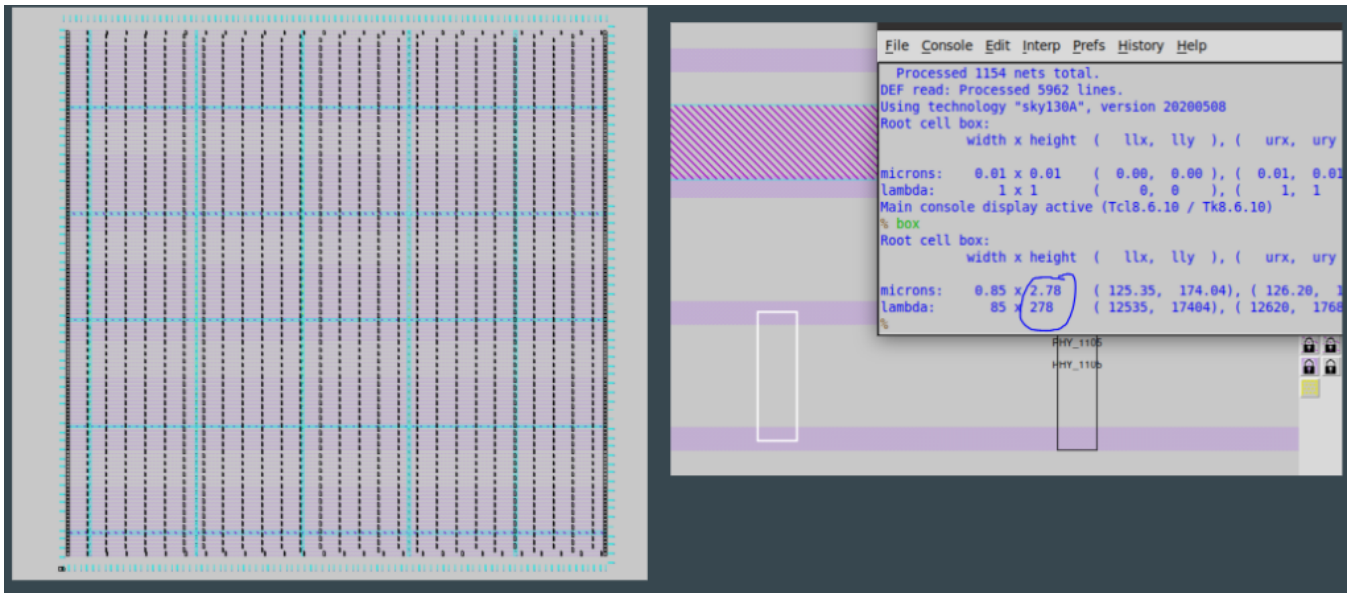
Below is the figure of the Floorplan stage.



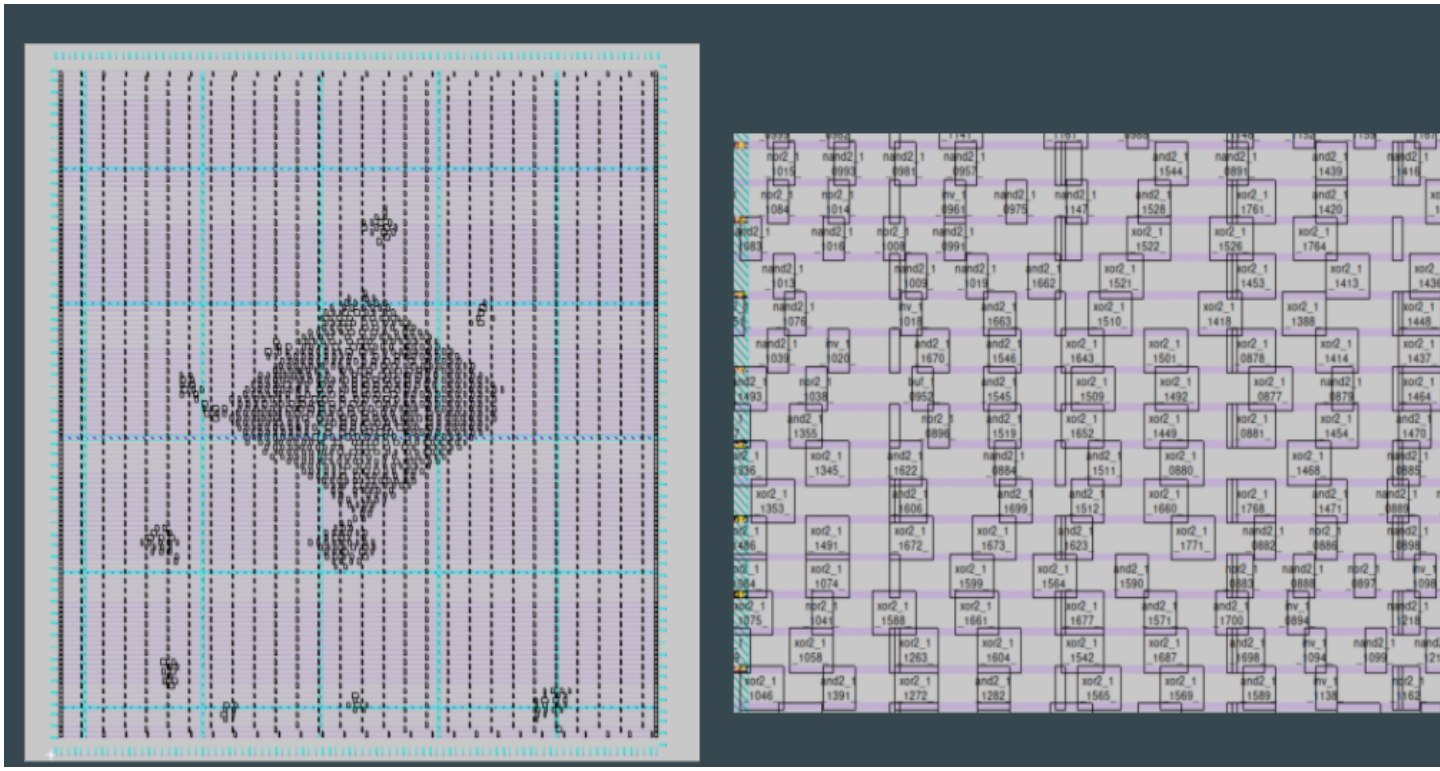
Below figure contains a magnified image where ports in blue color with decap cells,tap cells arranged in core and standard cells in the corner surrounded by white box are visible.



Below figure shows the power distribution layout by metal5 and metal4 of sky130 technology. Here we can see in TCL/TK window pitch of metal 5 is 2.78 microns which was mentioned in track info file.The PDN def file can be found in ***“./openlane/designs/added/runs/run70/temp/floorplan/”***

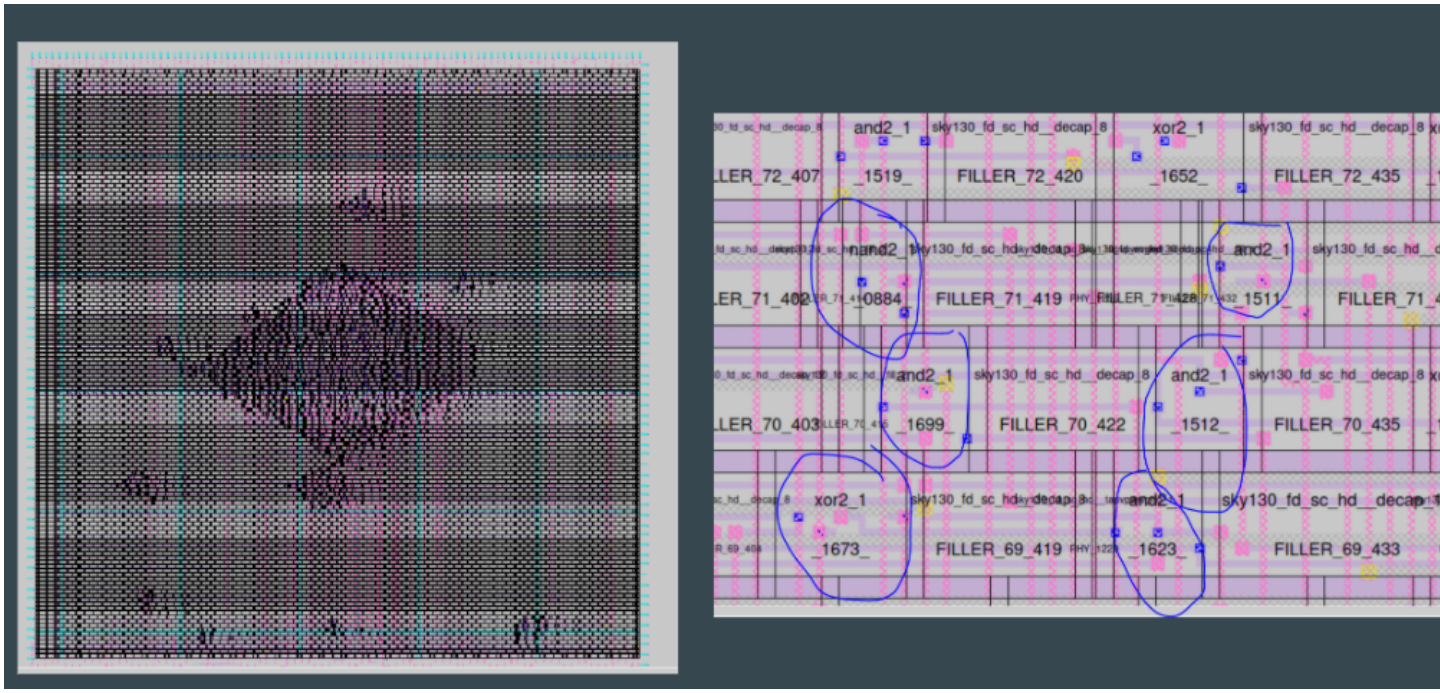


*7.Start Floorplan Procedure by typing "**run_placement**".This starts the global and detailed placement .Here we used random placement procedure for placement of adder as random placement is ideal placement procedure for small designs.The final def file is present in **"./openlane/designs/added/runs/run70/results/placement/"***



*8. Use command **"run_routing"** to complete routing procedure.Global and detailed routing is done.The def file after routing stage is found in **"./openlane/designs/added/runs/run70/results/routing/"***

Below figure shows the def file after routing.Here custom cells are highlighted in blue colour.



9.Finally use command "run_magic" to get final GDS file of design.
The gds file after routing stage is found in
"./openlane/designs/added/runs/run70/results/magic/"

Conclusions -

We've successfully compiled a standard cell library comprising 21 cells and integrated that into OpenLANE to be used with custom hardware designs that use these cells.
The characterization process is successful and accurately gives us a picture of the behaviour, functionality and limitations of each of the standard cells. This will help the user determine how effectively he can optimise his designs to incorporate our standard cells.
We have also detailed in steps how to not only integrate our cells, but also run through all the necessary stages in ASIC flow using OpenLANE to generate a gds file of your design comprising our cells. The Synthesis, Floorplan, Placement and Routing stages have been exemplified and detailed by means of a simple adder circuit but do note that the complexity of the design isn't a roadblock to this project.

Literature Survey -

1. http://opencircuitdesign.com/magic/index.html - Used this reference to determine DRC rules and regulations for designing layouts of various combinational cells.
2. https://github.com/YosysHQ/yosys - Used this to accurately synthesize the netlists for hardware circuits using our standard cell library.
3. https://github.com/efabless/openlane - Used for post synthesis stages.
4. http://www.ece.iit.edu/~eoruklu/courses/ece429/tutorial/MAGIC1x.html - For Creating Layouts with Magic.
5. http://www.iosrjournals.org/iosr-jvlsi/papers/vol4-issue1/Version-1/F04112933.pdf - Characterization of Standard Cells.
6. https://shareok.org/bitstream/handle/11244/10182/Anne_okstate_0664M_11266.pdf?sequence=1 - Design and Characterization of Standard Cells.
7. https://skywater-pdk.readthedocs.io/en/latest/contents/libraries/foundry-provided.html - Used for understanding the sky130 standard library.

---------- Thank You!----------