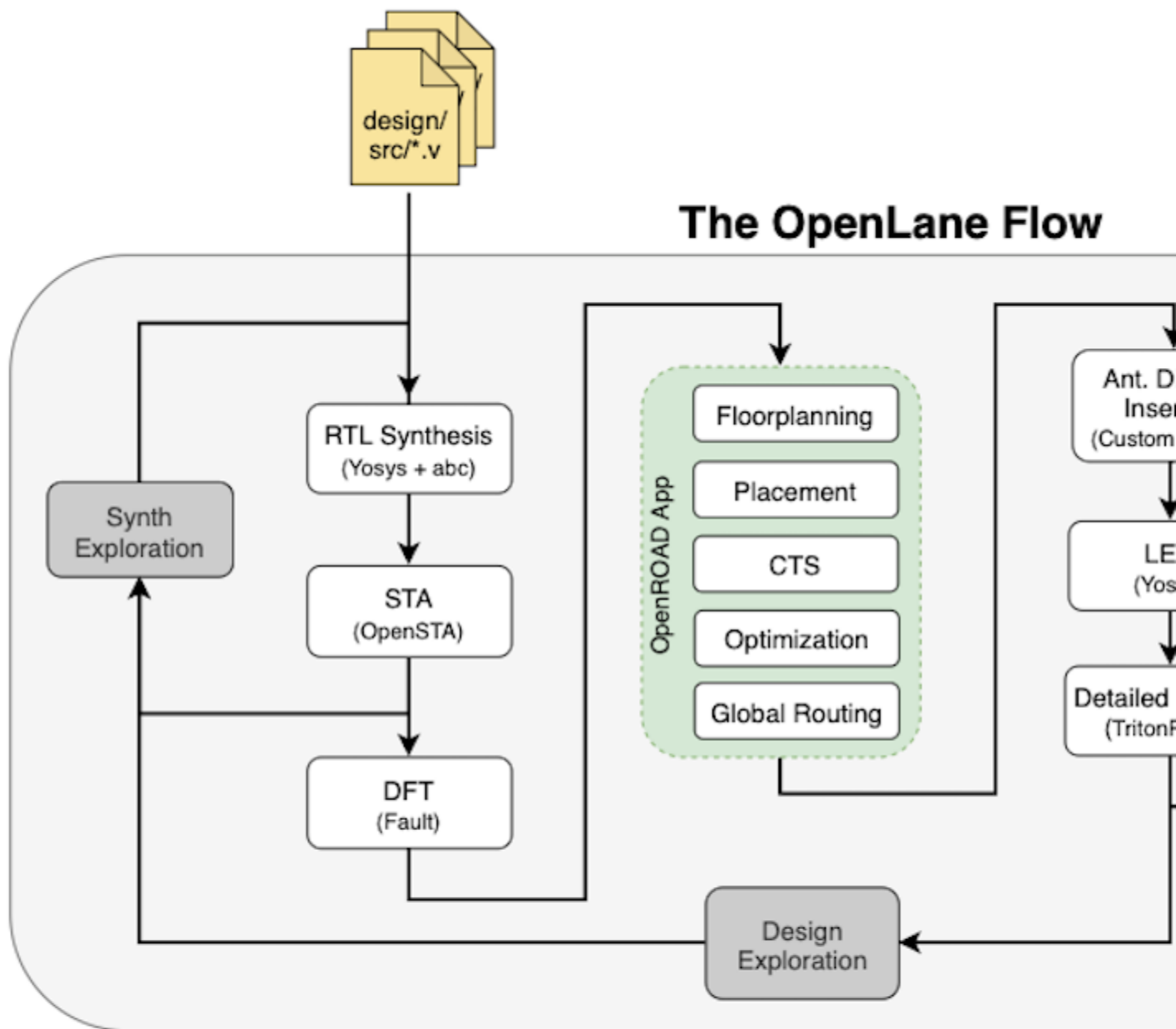


# Introduction to Openlane flow

OpenLANE is a completely automated RTL to GDSII flow which embeds in it different opensource tools, namely, OpenROAD, Yosys, ABC, Magic etc., apart from many custom methodology scripts for design exploration and optimization. Openlane is built around Skywater 130nm process node and is capable of performing full ASIC implementation steps from RTL all the way down to GDSII. The flow-chart below gives a better picture of openlane flow as a whole (**Image Courtesy:** [efabless/openlane](https://efabless.com/openlane/))



## Overview of Physical Design flow

Place and Route (PnR) is the core of any ASIC implementation and Openlane flow integrates into it several key open source tools which perform each of the respective stages of PnR. Below are the stages and the respective tools (in ( )) that are called by openlane for the functionalities as described:

- Synthesis
  - Generating gate-level netlist ([yosys](#)).
  - Performing cell mapping ([abc](#)).
  - Performing pre-layout STA ([OpenSTA](#)).
- Floorplanning
  - Defining the core area for the macro as well as the cell sites and the tracks ([init\\_fp](#)).
  - Placing the macro input and output ports ([ioplacer](#)).
  - Generating the power distribution network ([pdn](#)).
- Placement
  - Performing global placement ([RePLace](#)).
  - Performing detailed placement to legalize the globally placed components ([OpenDP](#)).
- Clock Tree Synthesis (CTS)
  - Synthesizing the clock tree ([TritonCTS](#)).
- Routing
  - Performing global routing to generate a guide file for the detailed router ([FastRoute](#)).
  - Performing detailed routing ([TritonRoute](#))
- GDSII Generation
  - Streaming out the final GDSII layout file from the routed def ([Magic](#)).

## Build and Invoke openlane

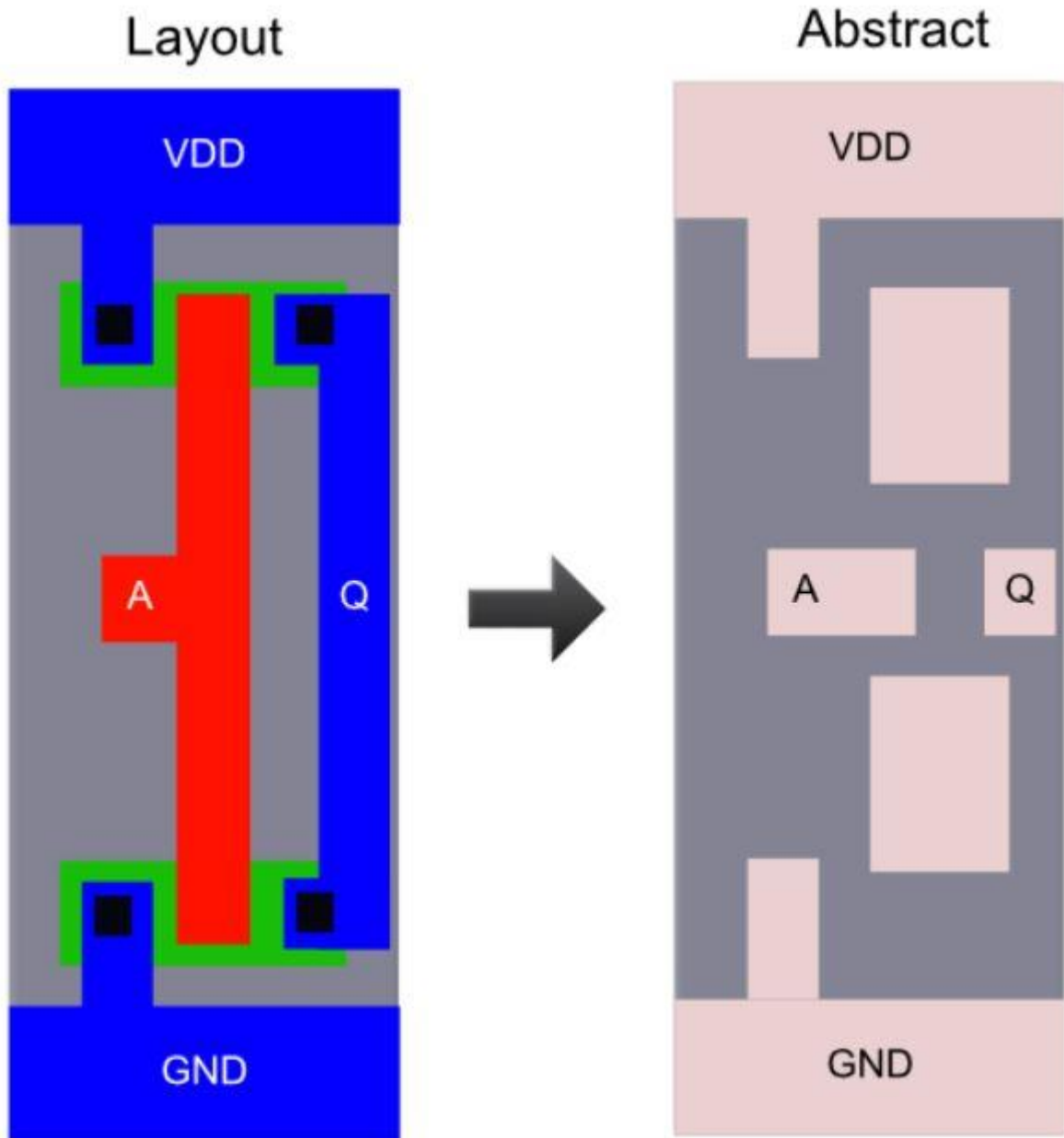
Detailed description on how to build and invoke openlane is given in this [link](#).

## Introduction to LEF

For a PnR tool to correctly place and route a block (a macro or a std. cell), it doesn't need to know entire layout information of the block; just the pin positions, PR boundary is sufficient. These minimal and abstracted information is provided to the tool by the

Library Exchange Format (LEF) file. LEF file also serves the purpose of protecting intellectual property and is basically of two types:

- Cell LEF - It's an abstract view of the cell and only gives information about PR boundary, pin position and metal layer information of the cell.
- Technology LEF - It contains information about available metal layer, via information, DRCs of particular technology used by placer and router etc.. The below diagram highlights the difference between a layout and a LEF (**Image Courtesy:** Google):



# Standard cell design and characterization in openlane

## Objective

The goal of the project is to design a single height standard cell and plug this custom cell into a more complex design and perform it's PnR in the openlane flow. The standard cell chosen is a basic CMOS inverter and the design into which it's plugged into is a pre-built [picrov32a](#) core.

## About PicoRV32

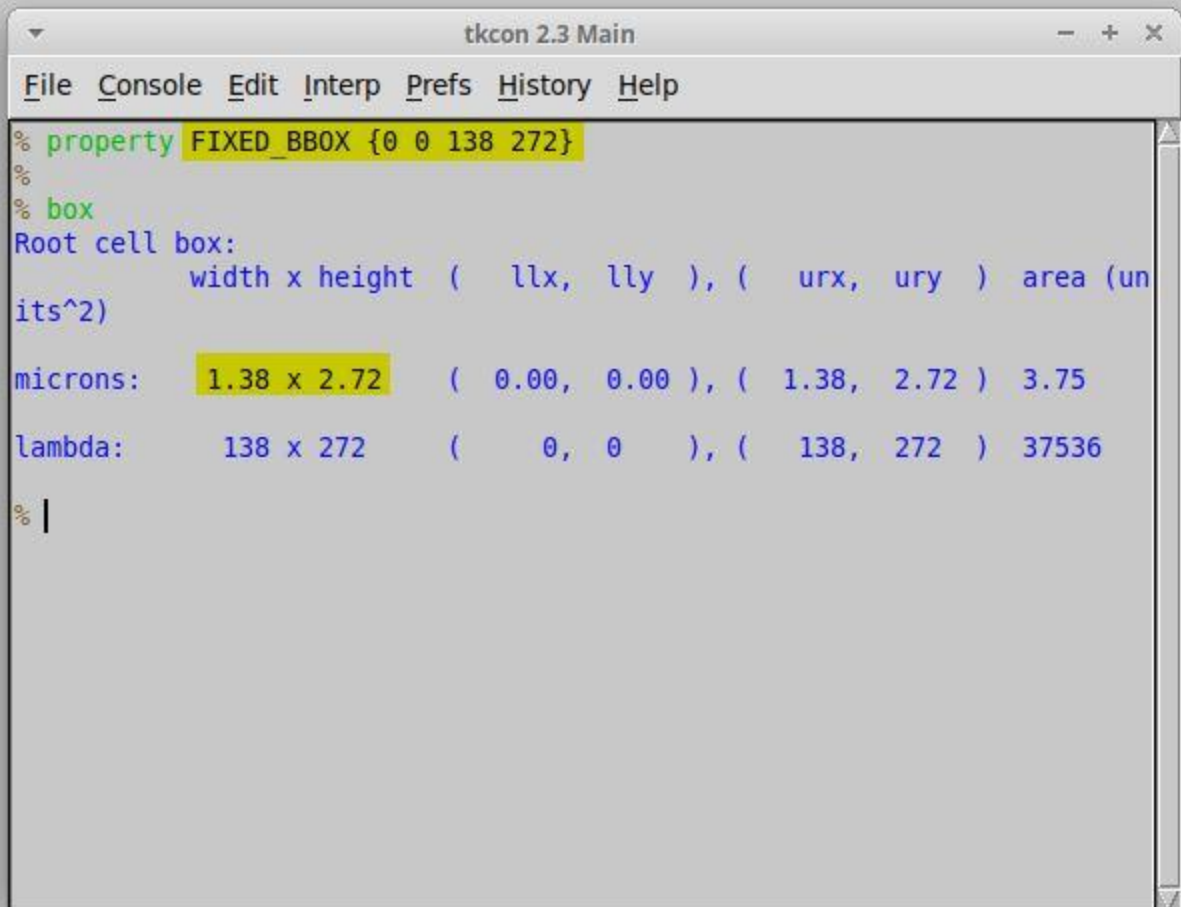
PicoRV32 is a CPU core that implements the RISC-V RV32IMC Instruction Set. It can be configured as RV32I, RV32IC, RV32IM, or RV32IMC core; where the suffixes stand for:

- M - Multiply extension
- I - Base Integer Instructions
- C - Compressed Instructions

PicoRV32 is free and open hardware licensed under the ISC license. All features and data-sheet related to picoRV32 core can be obtained [here](#).

## Standard cell layout design in Magic

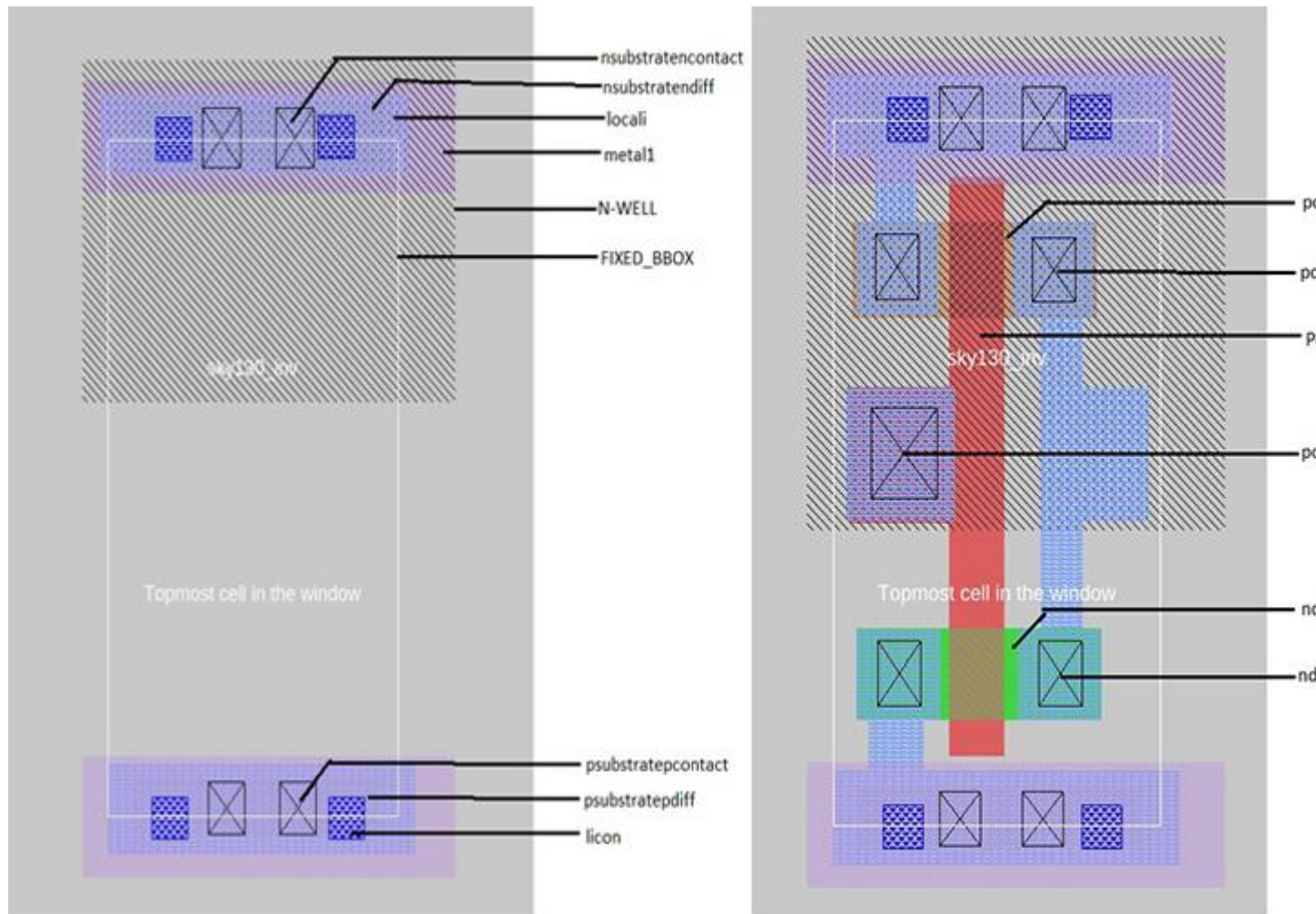
- The proposed inverter for the design is a single height standard cell, so the dimensions needs to be a multiple of the single height place site; which for sky130 node has a nomenclature of `unithd` with dimensions(in microns): 0.46 x 2.72 (width x height) for **sky130\_fd\_sc\_hd** PDK variant. The magic tool is invoked with sky130 tech file as `magic -T sky130A.tech` & (the magic tech file (sky130A.tech) has also been included in this repo under `/libs` as reference).
- Thus, the first step in magic layout tool is to create a bounding box with a width of 1.38 (3 x width(`unithd`)) and height of 2.72. This can be done by using command `property FIXED_BBOX {0 0 138 272}` in magic tkcon window.



The screenshot shows a window titled "tkcon 2.3 Main" with a menu bar containing "File", "Console", "Edit", "Interp", "Prefs", "History", and "Help". The main text area contains the following Verilog code:

```
% property FIXED_BB0X {0 0 138 272}
%
% box
Root cell box:
width x height ( llx, lly ), ( urx, ury ) area (un
its^2)
microns: 1.38 x 2.72 ( 0.00, 0.00 ), ( 1.38, 2.72 ) 3.75
lambda: 138 x 272 ( 0, 0 ), ( 138, 272 ) 37536
% |
```

- This is followed by defining the ground and power segments (in metal 1), the respective contacts and finally the layout of the logic part. Same procedure can be followed for any standard cell layout.



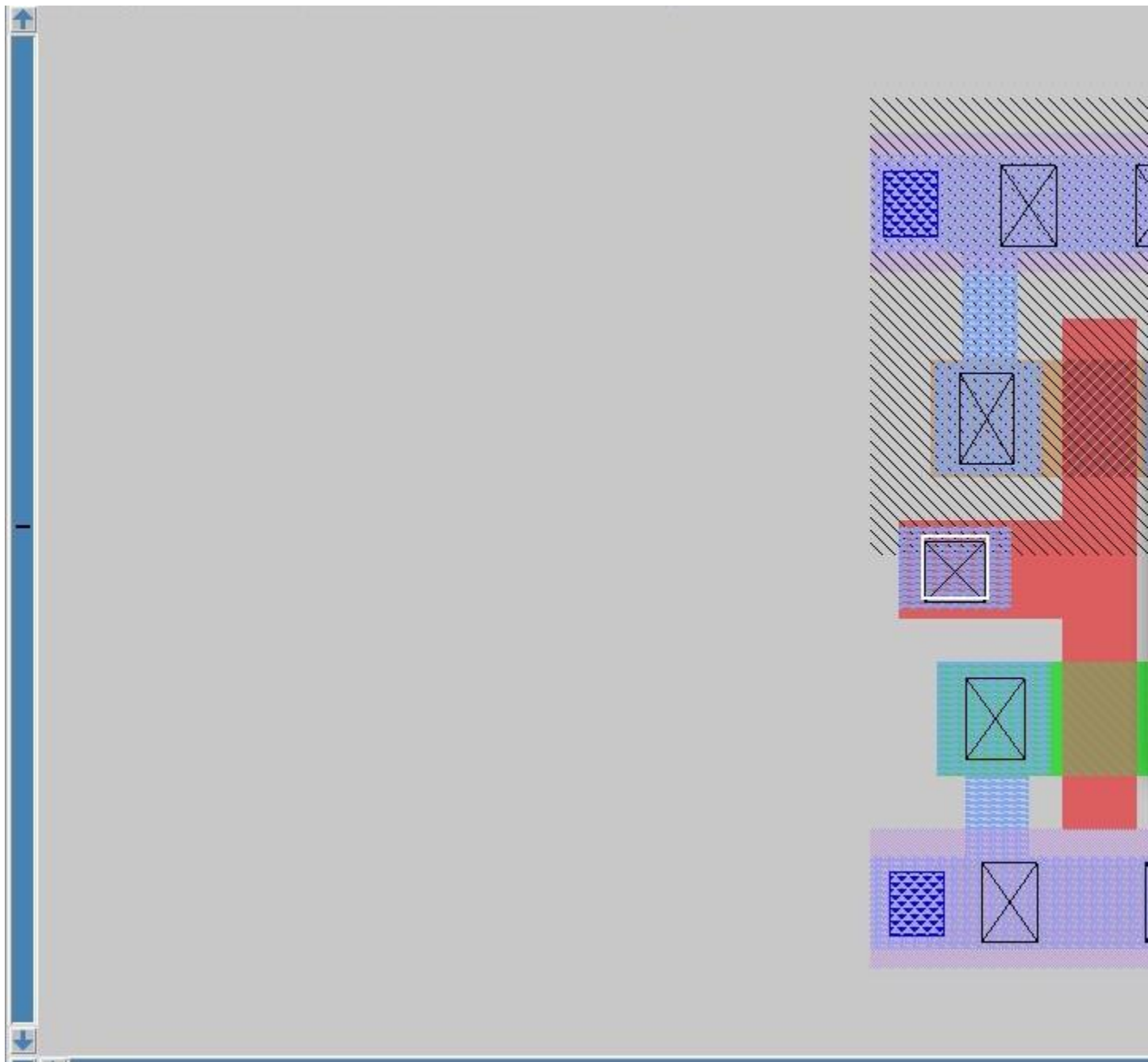
## Note:

The layout (also included as the part of this repo, viz., sky130\_inv.mag) can be viewed in magic layout window as `magic -T sky130A.tech sky130_inv.mag &`

## Create port definition

Once the layout is ready, the next step is extracting LEF file for the cell. However, certain properties and definitions need to be set to the pins of the cell which aid the placer and router tool. For LEF files, a cell that contains ports is written as a macro cell, and the ports are the declared PINs of the macro. Our objective is to extract LEF from a given layout (here of a simple CMOS inverter) in standard format. Defining port and setting correct class and use attributes to each port is the first step. The easiest way to define a port is through Magic Layout window and following are the steps:

- In Magic Layout window, first source the .mag file for the design (here inverter). Then **Edit >> Text** which opens up a dialogue box.



- For each layer (to be turned into port), make a box on that particular layer and input a label name along with a sticky label of the layer name with which the port needs to be associated. Ensure the Port enable checkbox is checked and default checkbox is unchecked as shown in the figure:





In the above two figures, port A (input port) and port Y (output port) are taken from locali (local interconnect) layer. Also, the number in the text area near enable checkbox defines the order in which the ports will be written in LEF file (0 being the first).

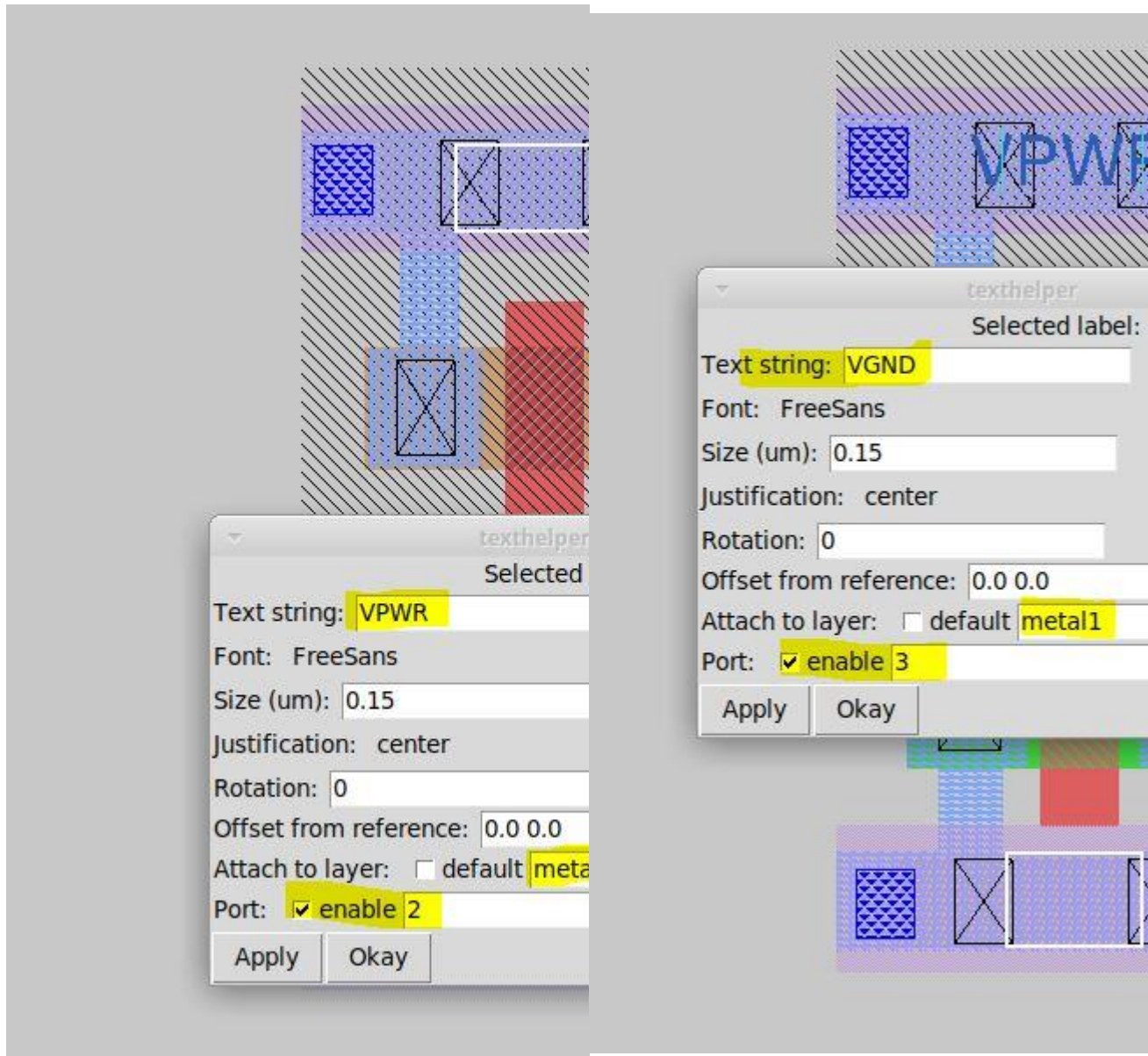
- For power and ground layers, the definition could be same or different than the signal layer. Here, ground and power connectivity are taken from metal1 (Notice the sticky label)

**VPWR**

**VGND**

VPWR

VGND

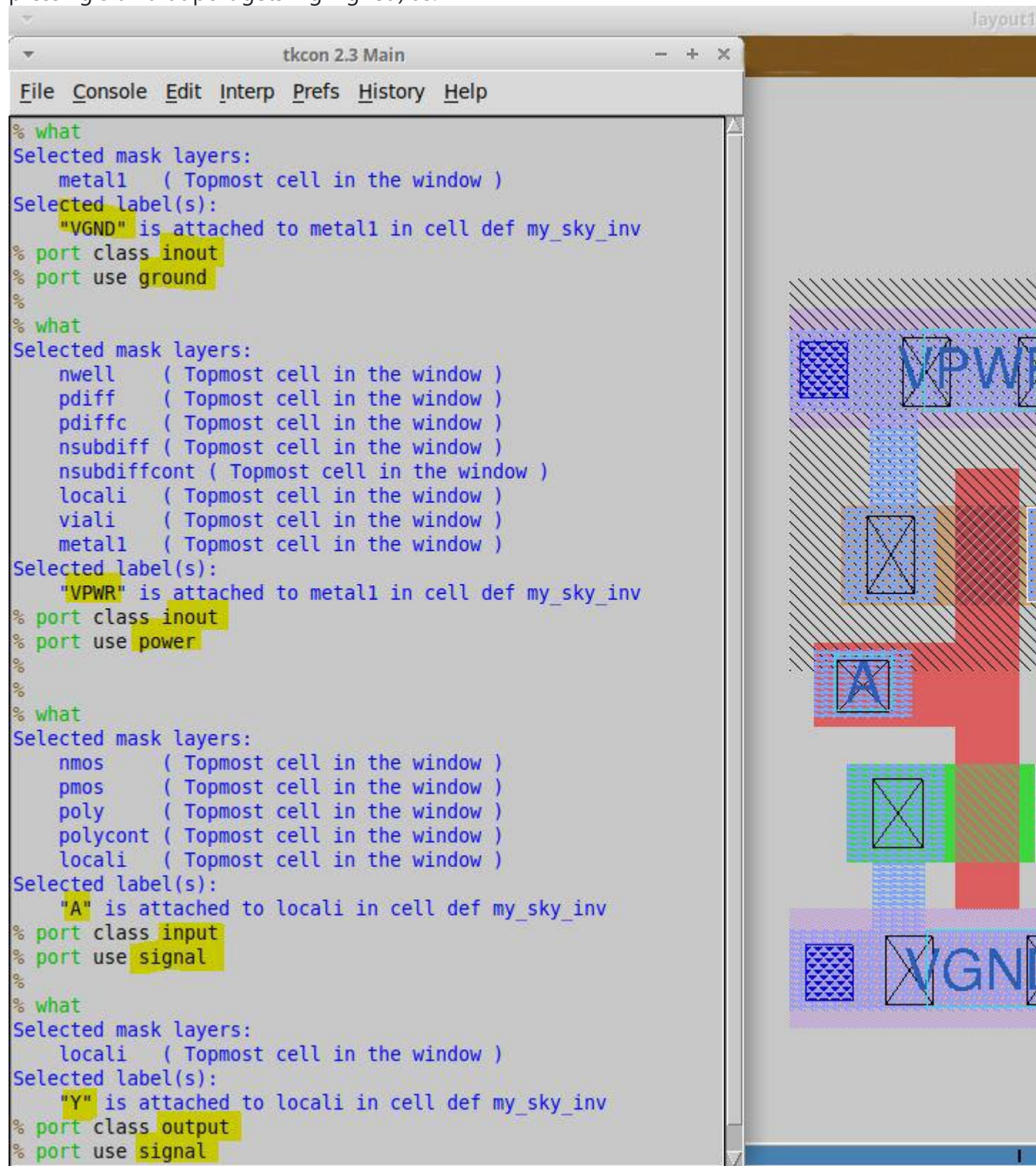


## Set port class and port use attributes for a layout

Post port definition, the next step is setting **port class** and **port use** attributes. The "class" and "use" properties of the port have no internal meaning to magic but are used by the LEF and DEF format read and write routines, and match the LEF/DEF CLASS and USE properties for macro cell pins. Valid classes are: default, input, output, tristate, bidirectional, inout, feedthrough, and feedthru. Valid uses are: default, analog, signal, digital, power, ground, and clock. These attributes are set in tkcon window (after



selecting each port on layout window. A keyboard shortcut would be repeatedly pressing s till that port gets highlighted) as:



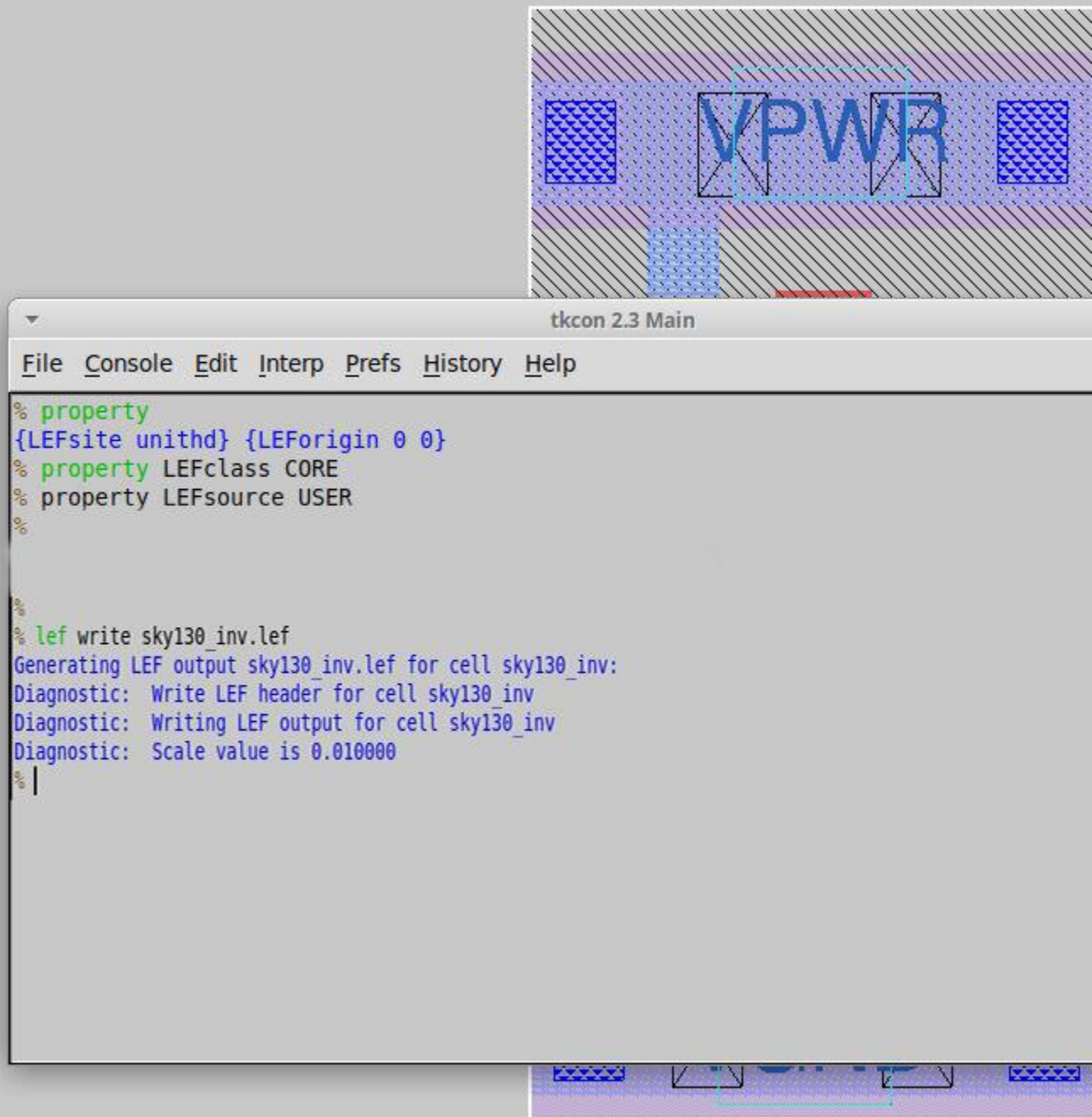
### **Additional:**

You can delete or remove any port by first selecting the port (key s) and then executing below two commands in order (in tkcon window):

- `port remove`
- `label erase`

## **Defining LEF properties and extracting LEF file**

Certain properties need to be set before writing the LEF. As mentioned before, these values are fetched by placer and router to determine, for instance, site where a cell needs to be placed. Macro cell properties common to the LEF/DEF definition but that have no corresponding database interpretation in magic are retained using the cell **property** method in magic. There are specific property names associated with the LEF format. Once the properties are set, `lef write` command writes the LEF file with the same nomenclature as that of the layout (.mag) file.



## Plugging custom LEF to openlane flow

If a new custom cell needs to be plugged into openlane flow, include the lefs (the one extracted in [Step-5](#)) as below:

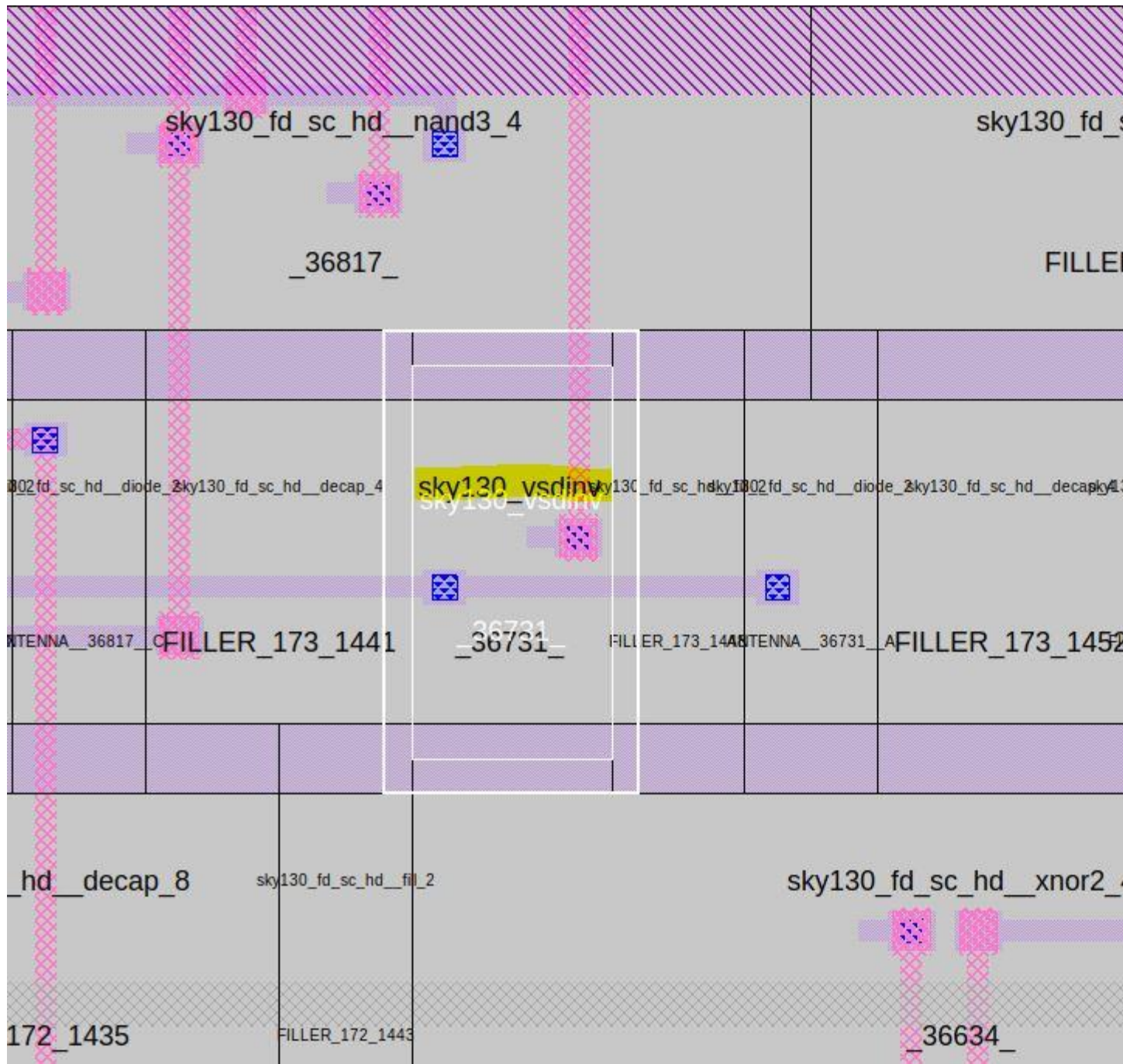
- In the design's config.tcl file add the below line to point to the lef location which is required during spice extraction.
- ```
set ::env(EXTRA_LEFS) [glob
$::env(OPENLANE_ROOT)/designs/$::env(DESIGN_NAME)/src/*.lef]
```
- Include the below command to include the additional lef into the flow:
- ```
set lefs [glob $::env(DESIGN_DIR)/src/*.lef]
```
- ```
add_lefs -src $lefs
```
- Run the interactive flow as described [here](#).

**Note:** A sample inverter magic file (*sky130\_inv.mag*) has been included as a reference resource.

## Observations

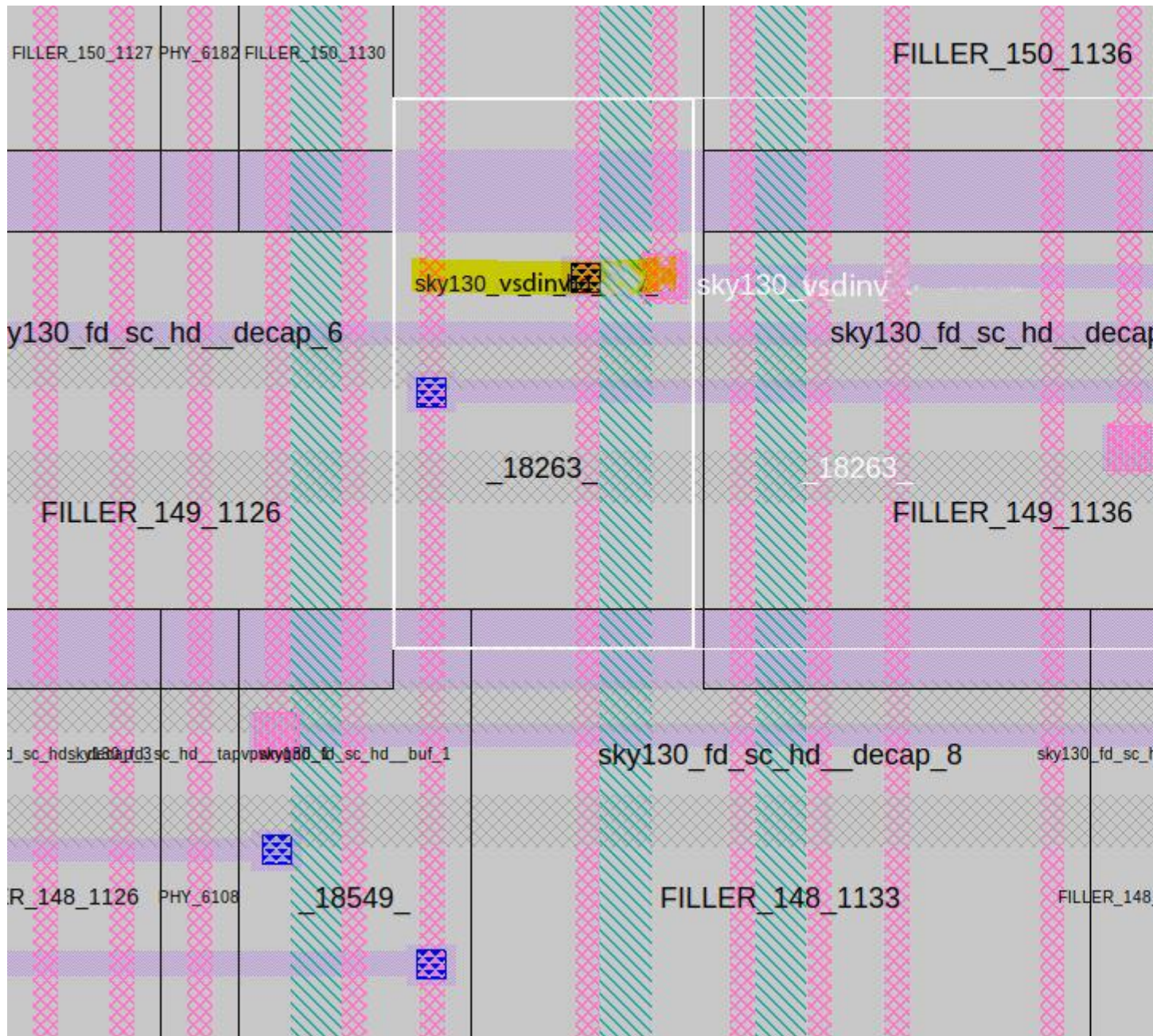
The custom inverter successfully included in the picorv32a design. Below is the final routed picorv32a design with the custom cell zoomed in and highlighted.





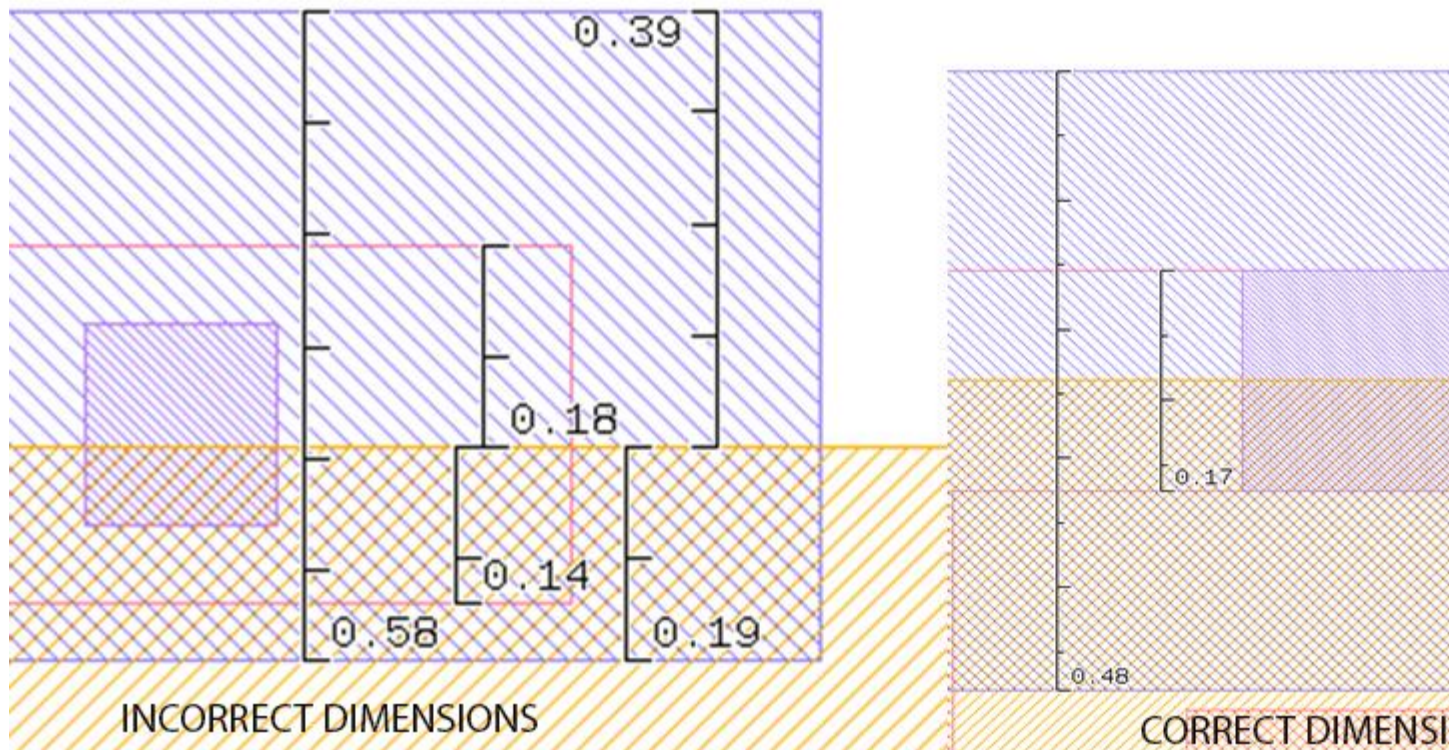
## Challenges

- The biggest challenge was with the legalization of the cell. Initial iteration showed illegal positioning of the cell away from the standard cell rails.



- Upon closer inspection, the issue seemed to be with the dimensions of the drawn power and ground rails and also with the positioning of local1 -> metal1 contacts which was then corrected for further iterations.





## Future work

The upcoming work of this project would be detailed IP characterization once all corner spice models is made availabe in Google-skywater [public forum](#).

## Limitations

At present, Openlane has following limitations:

- Limited Timing Constraints.
- Timing Closure.
  - No post-routing optimizations.