Analog Circuit Design

Q1 - 2024

This document describes the use of open-source PDK (SKY130) and tools for designing, simulating, and fabricating analog integrated circuits.

The installation process for the tools is not covered in this document.

The complete process has the following steps:

1. Schematic Design
2. Layout Design
3. Layout versus Schematic (LVS)
4. Design Rule Check (DRC)
5. Parasitic Extraction and Post Layout Simulation
6. Integration of Final Layout with Analog Caravel (Caravan) Wrapper
7. Precheck and Tapeout

- The selected test circuit to understand the entire process is a basic two-stage open-loop op-amp. It can operate as a comparator in saturation mode. However, this setup will result a comparatively slow comparator with a relatively high detectable differential input voltage (input offset). Still, the opamp is simulated in saturation mode to understand and plot some characteristics of a comparator. Furthermore, no design optimizations have been implemented on the circuit.

Schematic Design

Necessary Tools

- **xschem**
- **ngspice**

To launch xschem with sky130A PDK, use the following command:

```
xschem --rcfile /home/zerotoasic/asic_tools/pdk/sky130A/libs.tech/xschem/xschemrc
```

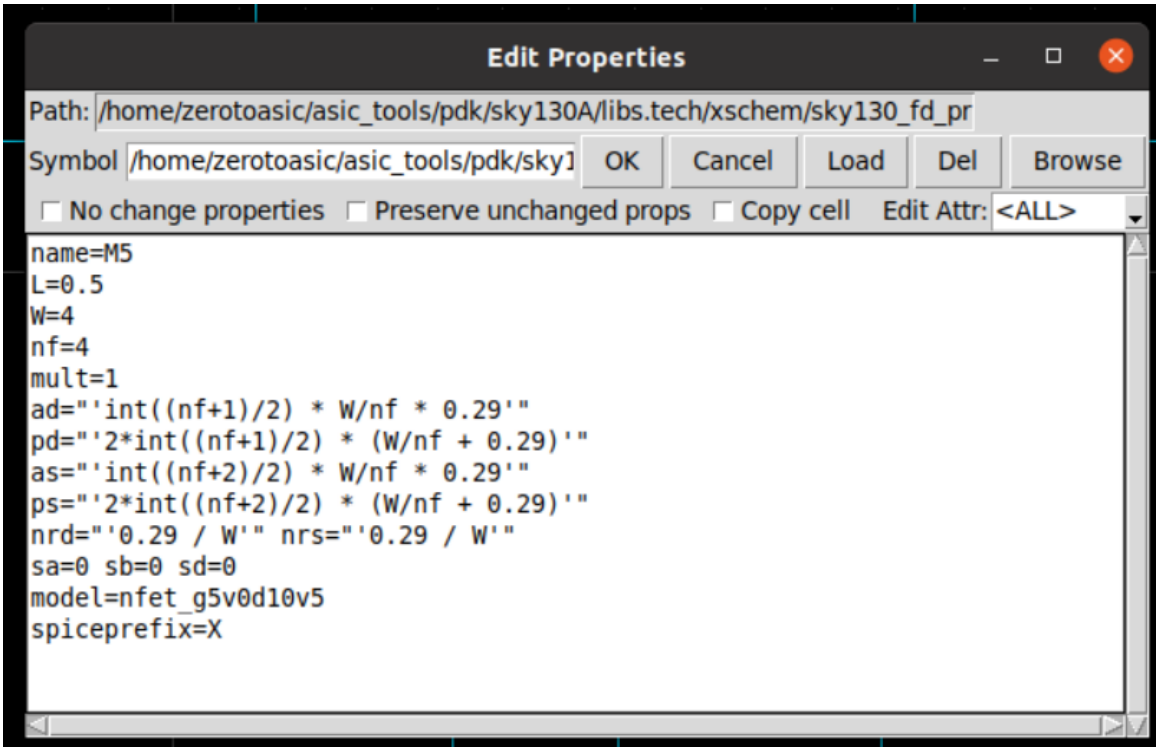
`Note:` To enable xschem console control using commands, the `tclreadline` plugin is installed.

Design Process

Creating Schematic

- **Circuit Components:**
  - Utilize parametric standard cells provided by the foundry.
  - The current design (opamp) operates with a supply voltage of VDD = 5V. Therefore, the **g5v0d10v5** nfet and pfet transistors are used in the design. These transistors have an operating gate voltage range of 0 V - 5.5 V.
  - Access detailed specifications of standard cells in the foundry documentation.
- **Inserting Transistors:**
  - Use the `insert` key to select the symbol of the desired cell.
  - The correct path to library needs to be given for selection.

    In the current case, the path to transistors is : `/home/zerotoasic/asic_tools/pdk/sky130A/libs.tech/xschem/sky130_fd_pr/`
  - Once placed, change transistor properties by pressing `q`.
  - Example transistor properties: `L` (length), `W` (total width), `nf` (number of fingers).



The above snapshot shows a dialog box displaying the properties of a transistor. Here, `L` represents the total length of the transistor. For g5vod10v5, the minimum required channel length is 0.5mm. `W` refers to the total width of the transistor. `nf` indicates the total number of fingers the transistor has.

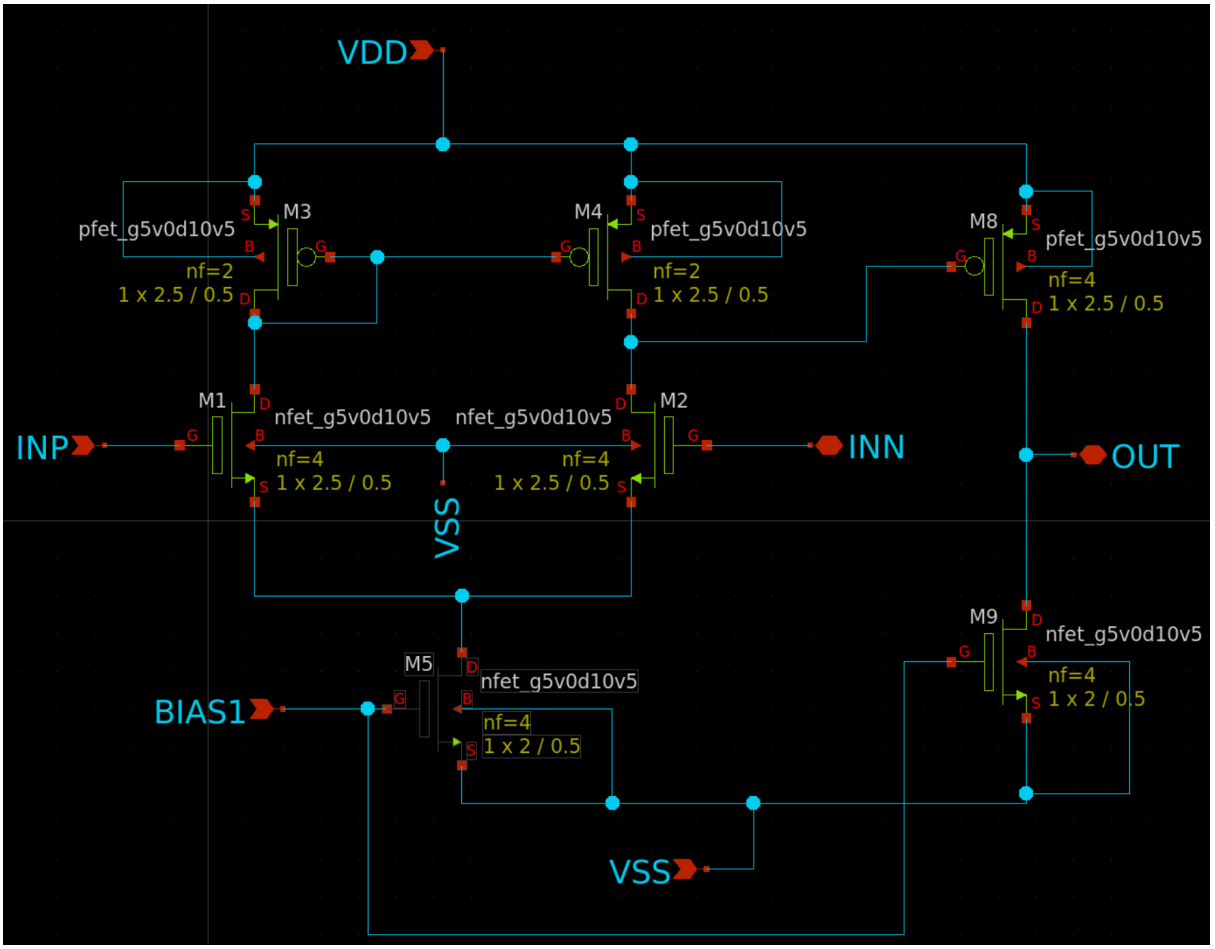
In the snapshot above, a 4 mm wide transistor is divided into 4 fingers, resulting in each finger being 1 mm wide. Multi-finger transistors are commonly utilized to reduce the gate resistance (caused by the wide polysilicon layer) of a single finger transistor.

- **Connecting Components:**
  - Connect transistors with wires using the `w` key.
  - Connect pins to nodes using input, output, or bidirectional pins from the xschem local library (`/usr/local/share/xschem/xschem_library/devices`).

    (simply press `home` in the "choose symbol" box).

> The `lab_pin.sym` from the same library is used to label a wire.

The image of the complete two-stage opamp schematic is shown below:



Creating symbol

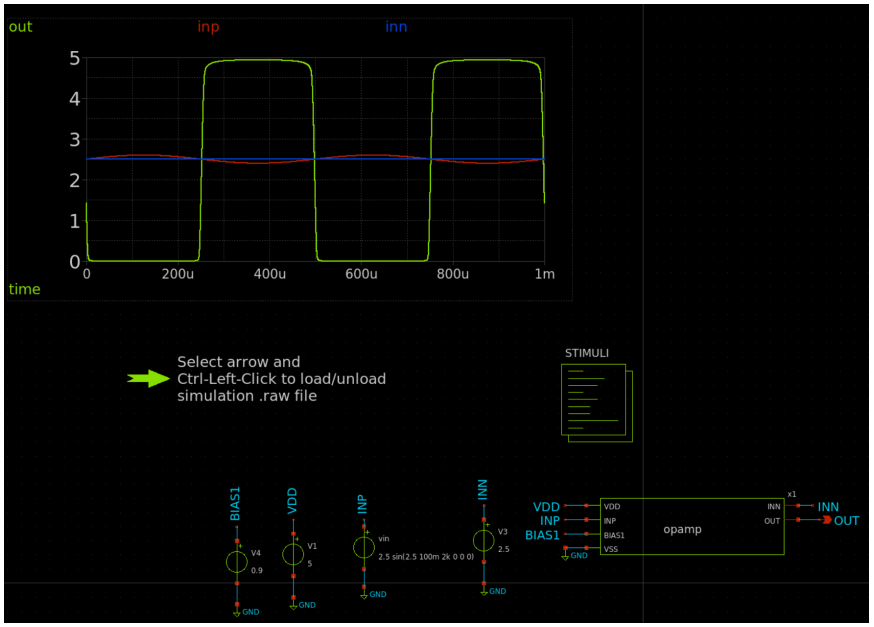
- The next step is to design a symbol for the schematic. To do this, simply press "**a**" from the schematic window, and the symbol will be generated. The symbol can then be opened and edited to achieve the desired shape or structure.
- In the symbol shown below (for opamp), VDD represents the supply voltage, BIAS1 is the biasing voltage for transistors M5 and M9, INP and INN are the positive and negative inputs of the op-amp, OUT represents the output, and VSS is the ground terminal. The symbol is used to create a test bench and run different simulations on the design.

Creating Test Bench

A test bench is used to verify that the designed schematic is generating the desired output under the required input conditions. To create a test bench, follow these steps:

1. Open a new schematic window.
2. Insert the schematic symbol created before (opamp.sym in the current case).
3. Connect the power, DC, and input ports in the symbol to the required sources. These can be found in the xschem home library.
4. Use a DC voltage source `vsource.sym` for supply and biasing voltages. The DC voltage is defined as the "value" inside the properties of the voltage source.
5. Use a ground symbol `gnd.sym` to connect the VSS terminal to zero potential.
6. Apply a sine voltage at the INP (positive input terminal) of the opamp. The function for generating the sine wave is given as `value="2.5 sin(2.5 100m 2k 0 0 0)"`. The negative input of the opamp (INN) is kept at the reference DC voltage of 2.5 V. This is later used as the reference voltage terminal of the comparator.
7. For more details on different signal generation for input, refer to the ngspice manual here.

The test bench created for the opamp schematic is shown below:



`Test bench of designed schematic opamp_tb.`

Defining Simulation Setup

To set up a simulation, you can utilize the `code.sym` cell, where all the commands are hidden, or the `code_shown.sym` cell, where all the commands are visible. These cells are provided by the xschem library.

Below are instructions for configuring the simulation using the `code` cell

Name ⇒ Provide a name (**STIMULI** in current case) for the entire setup in this field.

Value ⇒ Define all the simulation parameters, data extraction, and data manipulation requirements in this section.

- First specify the correct path to the SkyWater PDK library spice file and define the desired process for the models using the `.lib` command.

  For example, in the current case, the command is:

  ```
  .lib /home/zerotoasic/asic_tools/pdk/sky130A/libs.tech/ngspice/sky130.lib.spice tt
  ```

  The `.lib` command sets the library path, and `sky130.lib.spice` contains the spice definitions for all the models. In this case, the simulation is using the `tt` (typical typical) process corner.

- Next, define the simulation under a `control block`. It is structured as follows:

```
.control
---

**define simulation type here**

---
.endc
```

Simulation:

There are many simulations which are performed for different circuit analysis in different cases. In current case, three simulations are used for circuit analysis.

- **OP (operating point) simulation**

  Conduct an OP simulation to observe the operating point of each transistor, DC voltage, and current at different nodes. The syntax is as follows:

  ```
  .control
  op
  remzerovec
  write opamp_tb.raw
  .endc
  ```

  - Here `remzerovec` is used to remove any vector with zero value before writing the raw file. The use of raw file will be explained in plotting section.
  - To view the voltages, navigate to `Simulation` ⇒ `Annotate operating point into schematic`. This will highlight the operating points, such as transconductance and threshold voltage etc. of each transistor.
  - The command `show` also can be used in ngspice to list different DC parameters of transistors.

> To view the voltage at a specific node, place the `ngspice_probe.sym` cell in the desired location. This cell can be found in the xschem home library.

> Another feature provided by xschem is the cell `ngspice_get_value.sym`. It allows you to highlight any desired operating point parameter. The syntax is as follows:

```
ngspice_get_value.sym

name=r1
node=define the node here
descr="what is it"
```

For example, if you want to see the drain-to-source voltage (vds) of transistor M1, the node can be described as:

```
name=r1
node=v(@m.xm1.msky130_fd_pr__nfet_g5v0d10v5[vds])
decscr="vds"
```

- **DC simulation**

To simulate the DC voltages and current with respect to each other (at any node), DC simulation is used. The syntax is as follows:

```
.control
dc vin 2.3 2.7 0.001
write opamp_tb.raw
.endc
```

Here, the DC voltage named `vin` is varied from 2.3 V to 2.7 V with a step size of 1 mV.

- **Transient simulation**

For plotting voltages and current signals with respect to time, use transient simulation. The command `tran` is used to initiate the simulation. Syntax is as follows:

```
.control
tran 10n 1m
write opamp_tb.raw
.endc
```

Here a transient simulation is performed for 1 mS with a step size of 10 nS.

Run the Simulation

Once the schematic and test bench is designed properly, we can run the simulation.

To run the simulation, follow these steps:

1. Click on the netlist tab to generate the netlist. This will create the netlist of the schematic inside the `~/.xschem/simulation` folder. If there are any issues with the schematic or the test bench, an error box will appear.
2. Next, click on the "Simulate" tab to initiate the design simulation.
3. Once clicked, an ngspice simulation will be executed.

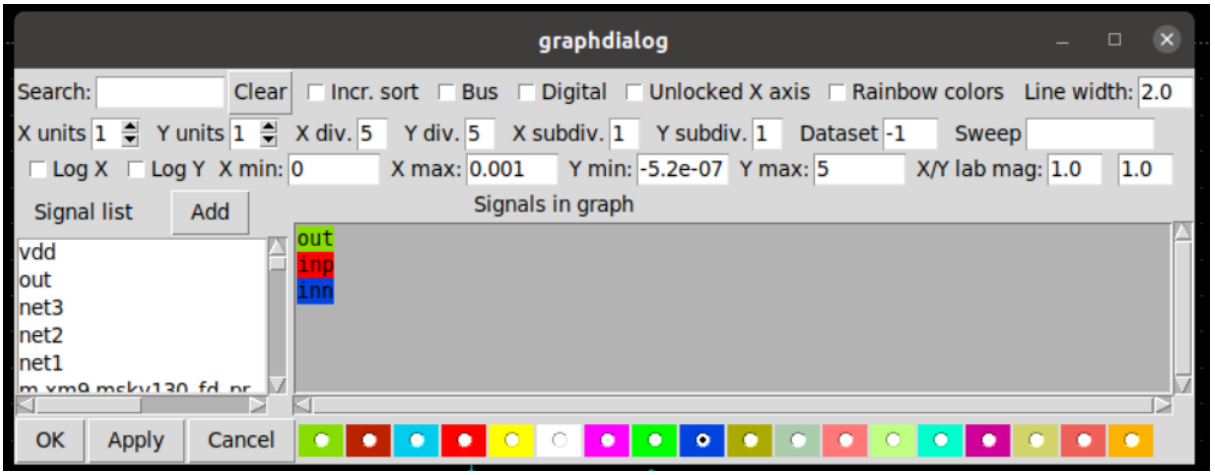
Plotting the Simulation Results

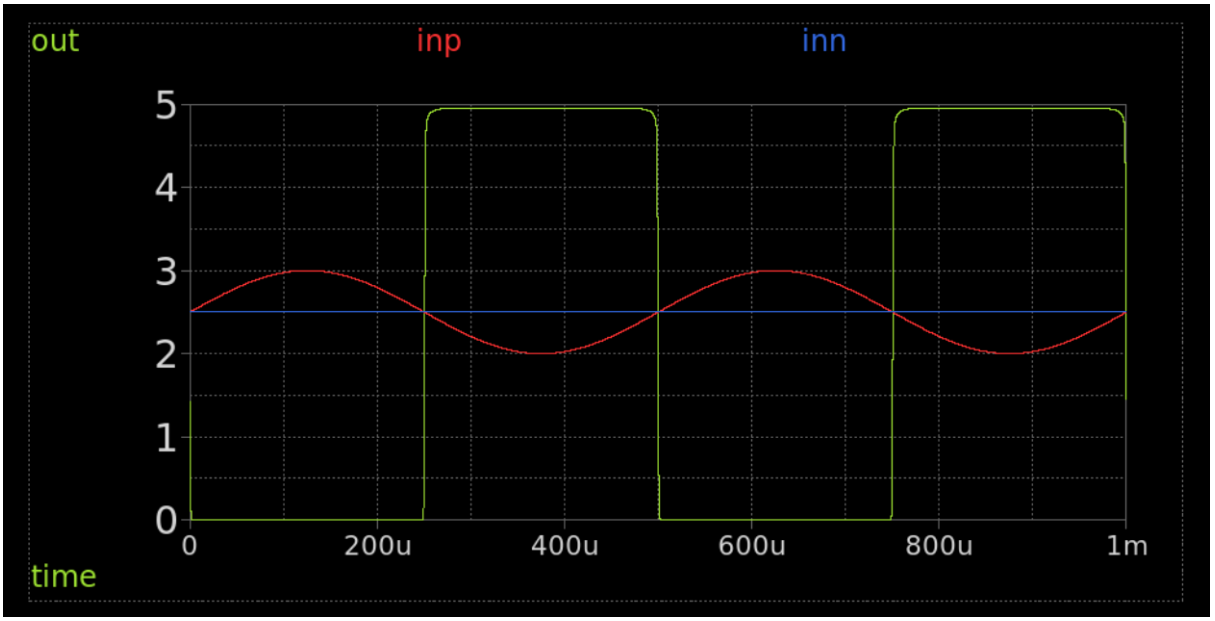
There are two ways to plot the simulation results.

- **Using xschem**

The xschem allows to plot simulation results by reading the .raw file which is written while setting up the simulation inside the `code.sym` cell. The generated raw file can be found in `~/.xschem/simulation` folder.

1. To plot the desired results, first add a waveform graph from `simulation` ⇒ `Add waveform graph`.
2. Next, the `*.raw` file need to be loaded. It can be done by clicking `simulation` ⇒ `Load/Unload spice.raw`.
3. Once the raw file is loaded, double click on the graph and it will open a graph dialog. From here the desired signals can be selected for plots. There are options for changing the colors of waveform, change scale axis etc.
4. To fit the waveform in the graph properly press `f` in the center of graph and the waveforms will fit according to the axis scale.
5. Cursor `a` and `b` can be used to create annotation probes in the waveform graph.





- When making any changes to the circuit and subsequently running a new simulation, it becomes necessary to unload the existing .raw file and reload it. To simplify this process, a `launcher.sym` cell can be used from xschem home library.

  Use the following commands in the launcher symbol properties.

```
launcher.sym

name=h1
descr="Select arrow and Ctrl-Left-Click to load/unload simu
lation .raw file"
tclcommand="xschem raw_read $netlist_dir/[file tail [file r
ootname [xschem get current_name]]].raw"
```

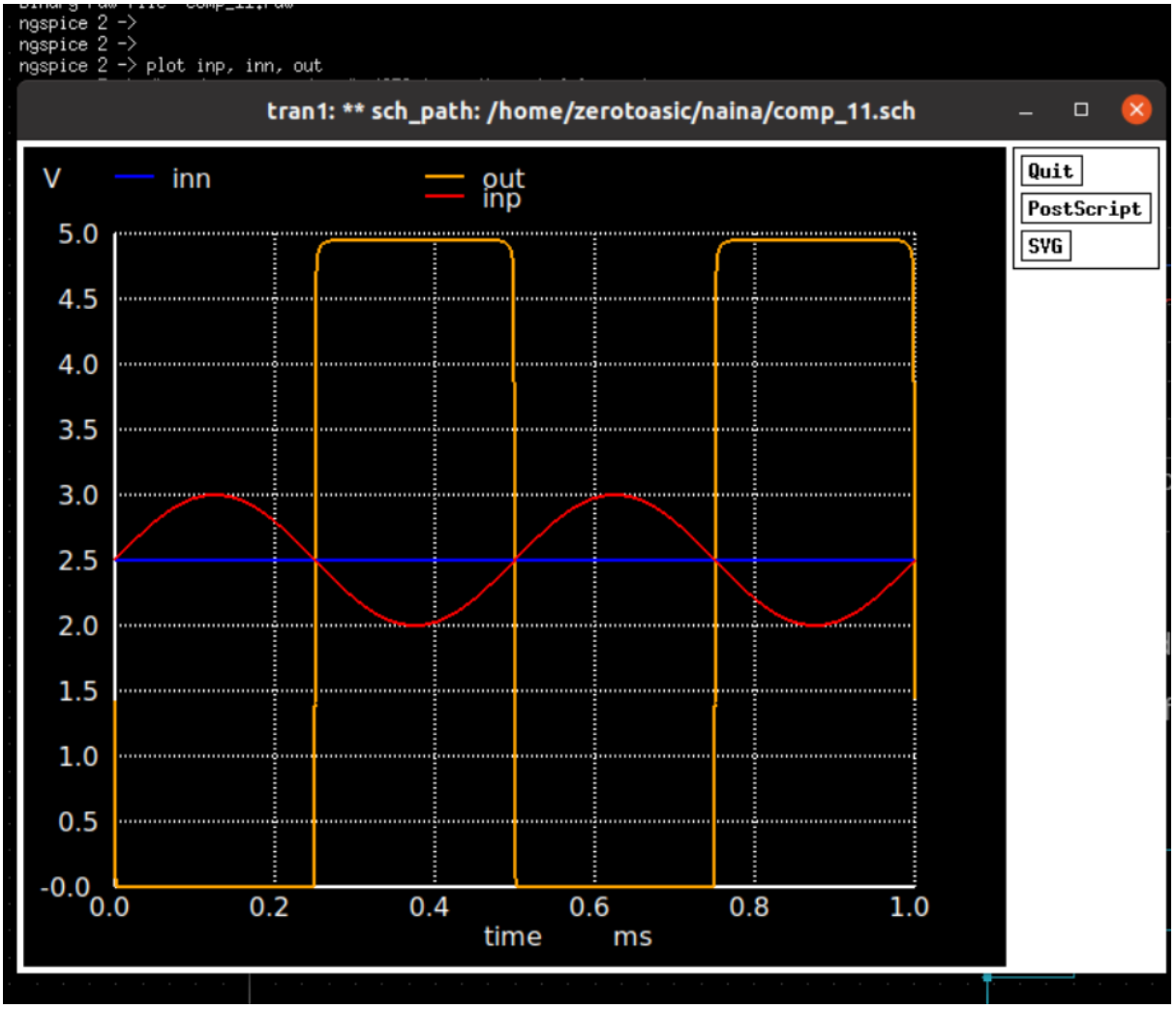
The launcher will look like the image below.

Here the tcl command `xschem raw_read` allows automatic loading and unloading of the raw file by pressing `“ctrl + left mouse click”`

- **Using ngspice**
  - Another way to plot the results after a simulation is through ngspice.
  - When the simulation is initiated, it opens a ngspice window, starting the simulation process. Once the simulation is complete, you can use spice commands in the same ngspice window to examine the results.
  - For a comprehensive guide on these commands, please refer to the ngspice user manual available here.

  In this document, we will refer to the designed op-amp, while utilizing it as a comparator. To plot input and output voltages, use the following command:

```
plot inp, inn, out
```



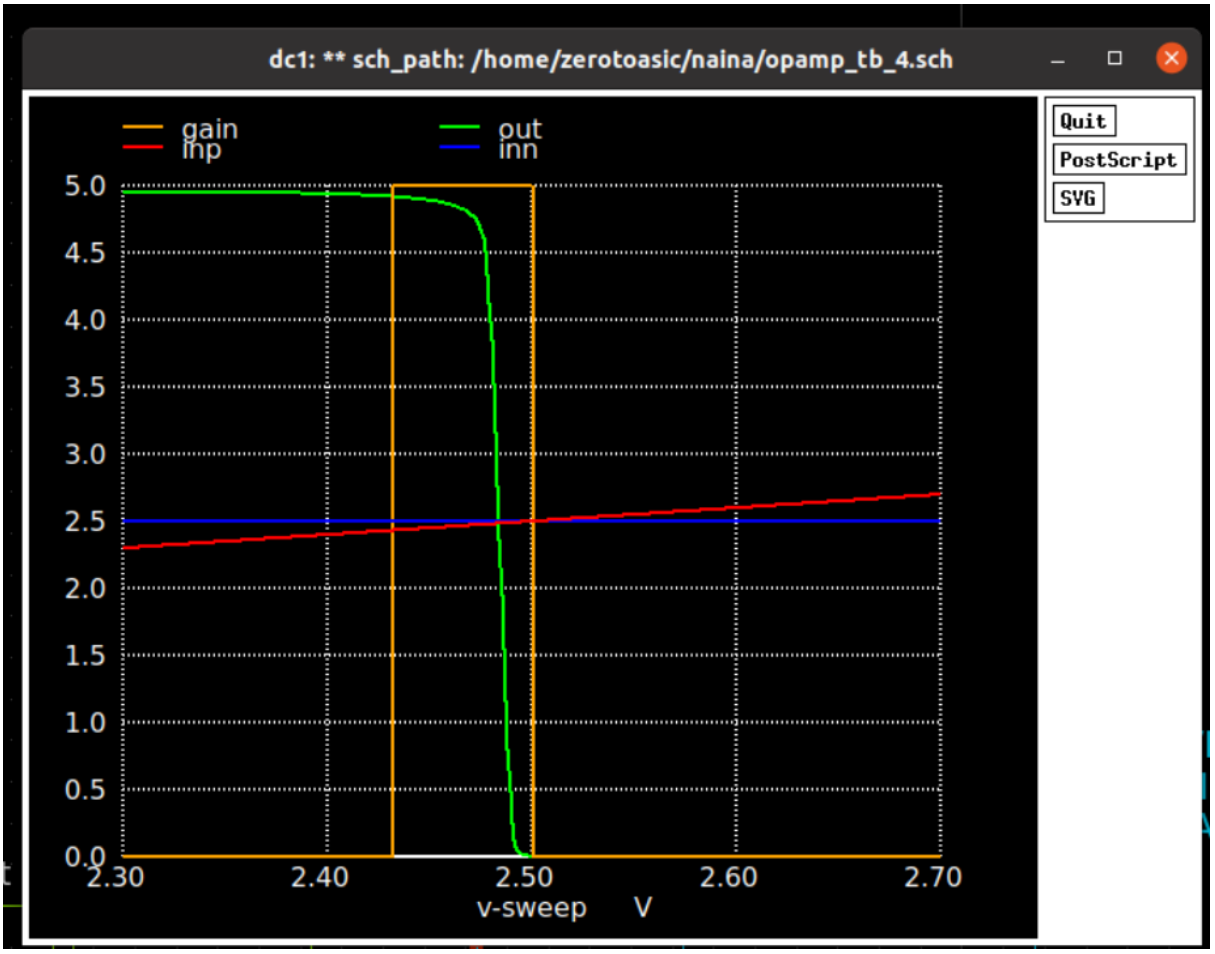
Furthermore, Ngspice enables scripting through spice commands, allowing users to extract specific information from simulation results.

Example for calculating input offset voltage of the comparator

For instance, to determine the input offset voltage (the minimum value of input voltage 'inp' for which the output of comparator ‘out’ changes its state), a DC simulation can be performed by varying the ‘inp' value in a desired range.

```
code.sym

name=STIMULI
only_toplevel=false
value="
.lib /home/zerotoasic/asic_tools/pdk/sky130A/libs.tech/ngsp
ice/sky130.lib.spice tt_mm
.option savecurrents
.save all
.control
dc vin 2.3 2.7 0.001 //Run DC simlation and vary inp(vin) f
rom 2.3 V to 2.7 V//
write opamp_tb.raw //Write raw file for plotting data using
xschem (if required)//
let gain = (abs(deriv(out)) >= 1)*5 //Take absolute derivat
ive of the output voltage (set as variable gain) and scale
it around 5 (just for better visualisation)//
meas vio dc find inp when gain=1 cross=1 //measure the valu
e of the input voltage when gain crosses 1 for first time
(input offset voltage for output transition from low to hig
h)//
plot inp inn gain
.endc
"
```



DC simulation when inn = 2.5V and inp = 2.3 V - 2.7 V

- In the above plot, when the input voltage `inp` is below the reference voltage `inn`, the output remains high (approximately VDD). Conversely, when the input voltage exceeds the reference voltage, the output transitions to the ground.
- To enhance the visibility of this transition and calculate the input offset voltage, the derivative of the output voltage is taken and set as a parameter named `gain`.
- The input offset voltage for output transitioning from `"high to low"` is defined as the value of the input voltage at which the `gain` first peaks up (crosses) for first time. This signifies the minimum input voltage at which the output switches from a high to low state. `meas vio dc find inp when gain=1 cross=1` command measures the value of inp `(vio)` when `gain` is equal to 1 (peaks up) for **first** time.
- In this particular case, the determined value is approximately 2.443 V. As a result, the detectable **differential input voltage** can be calculated as 2.5 - 2.443 = 57 mV.

Additional results for both the high to low and low to high transitions of the comparator output can be obtained using transient simulation.

Similarly, various spice commands can be employed to calculate the desired information from the waveforms plotted in Ngspice.

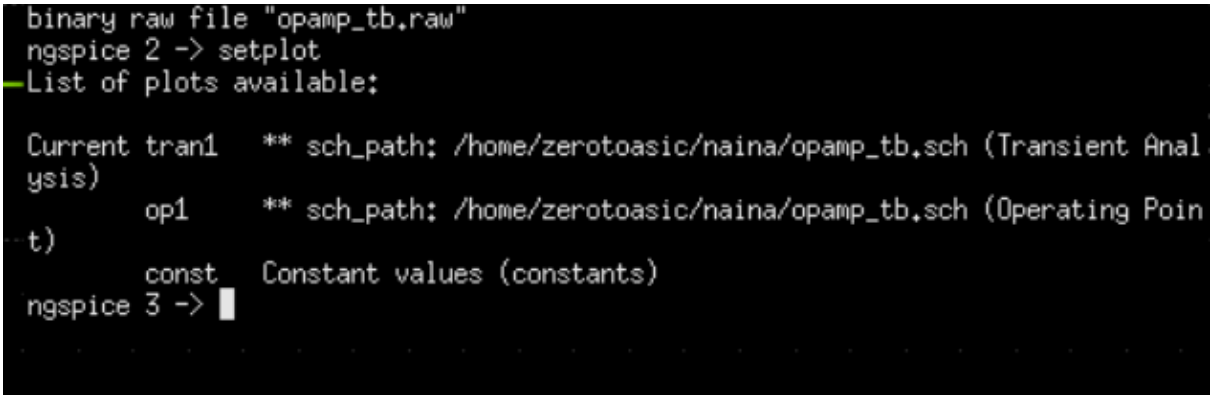
Defining multiple simulations in one control block

- Multiple simulations can be defined under the same block easily. The command `set appendwrite` is used to merge the results from different simulation in one raw file.

For Example:

```
.control
op
rezerovec
write opamp_tb.raw
set appendwrite
tran 10n 1m
write opamp_tb.raw
.endc
```

- In ngspice `setplot` command can be used and it will list all the plots available from different simulations.



`Selecting plots after running multiple simulations from one code block.`

In above snapshot it can be seen that the current plot is set to `tran1` (transient analysis).

It can be changed to operating point `(op1)` simulation by command:

```
setplot op1
```

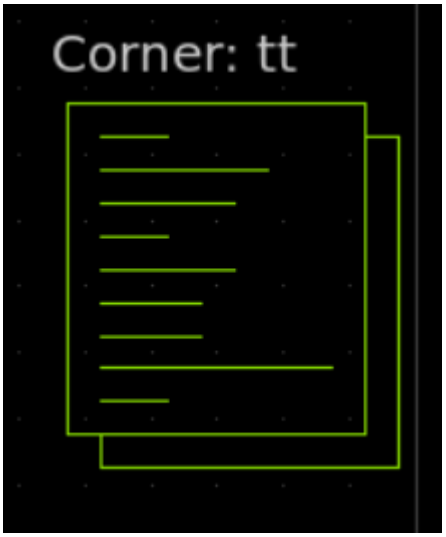
Useful Links

1. **SourceForge Forum for XSCHEM:**
    - Community platform for discussing and resolving issues related to XSCHEM.
2. **Engineering Wikis by Ming Sun:**
    - Collection of articles about XSCHEM, a good reference.
3. **XSCHEM Tutorial:**
    - Official tutorial for XSCHEM.
4. **NGSPICE Manual:**
    - Comprehensive manual for NGSPICE.
5. **Video Tutorials by bminch:**
    - Engaging tutorials covering XSCHEM, NGSPICE, and other open-source IC design tools.

.

Additional points

1. Instead of using the `.lib` command, the `corner.sym` located at `/home/zerotoasic/asic_tools/pdk/sky130A/libs.tech/xschem/sky130_fd_pr/corner.sym` can be used to define the process corner. This will result in faster simulation.



2. After running multiple simulations (DC, AC, TRAN, etc.) together, use different `launcher.sym` files to plot the results of each simulation.

For example, to read the result of the DC simulation, use the following command:

```
telecommand="raw_read $netlist_dir/comparator_tb.raw dc"
```

To plot the DC simulation results, press `CTRL + left click` on the launcher and the results will be displayed after reading the `.raw` file.

3. The device performance can be simulated using the dc simulation with the following commands

```
dc temp -40 125 1
```

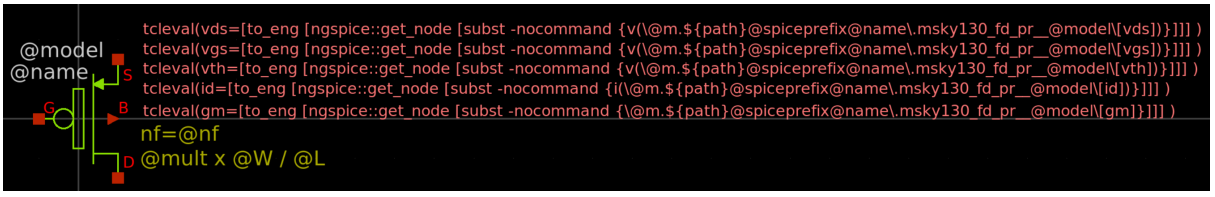
The above command will vary the temperature from -40°C to 125°C with a step size of 1°C.

4. NGPICE encountered issues when setting W as the total width of the transistor while using multi-finger transistors. In the layout generation process using the .spice file, W is considered as the width of a single finger. To resolve this issue, the `wnflag` switch is available. Enable it using the following command:

```
.option wnflag=1
```

5. If you want to annotate any desired operating parameter of a device, it can be directly added in the device symbol.

For example I have added `Vgs`, `Vth` and `Vds` in the transistor symbol. It makes it easier to calculate the operating region of the transistor and find optimum device size.



Monte Carlo Simulation

- A Monte Carlo simulation is used to verify the performance of a design considering different process variations and device mismatches.
- In the sky130 PDK, the Monte Carlo process corner is labeled as "mc" and the mismatch is labeled as "mm".

To enable the Monte Carlo simulation with mismatch, the mismatch switch is turned on using the following command:

```
.param mc_mm_switch = 1
```

However, there are certain limitations on modeling the devices for the Monte Carlo simulation in the current version of the sky130 PDK. The devices are not modeled for simulation beyond a specific aspect (W/L) ratio.

- Currently, the maximum transistor length is 20um for MC simulation.

Moreover, the Monte Carlo simulation does not work on a test bench with symbol. Run it directly on schematic.

Layout Design

Tools Used

- Magic
- Klayout

In my experience, KLayout has user-friendly interface, making it comparatively easy to use. In contrast, Magic is closely associated with the sky130 Process Design Kit (PDK), offering advanced capabilities but requiring some time for users to become familiar with its features. Notably, the latest version of KLayout (which I have not used yet) appears to have better connectivity with open-source PDKs.

Layout Design Using Magic

Magic is launched using the command as follows:

```
magic -rcfile /home/zerotoasic/asic_tools/pdk/sky130A/libs.
tech/magic/sky130A.magicrc
```

> **It will automatically launch the sky130A PDK into the magic using the rcfile.**

- This command opens a layout window and a command window.
- In magic all the the layout files are stored in .mag format. It also allows reading and writing GDS for the layout file.

Firstly, to work with any layout or to design a new one, define the grid and snap size so that later there are not any off-grid DRC errors. The grid size required for sky130 is `0.005 um`. It can be set using command:

```
grid 0.005um 0.005um //set the grid size equal to 0.005um//
snap user //set snap equal to the defined grid size//
drc style drc(full) //Full DRC check is turned on//
```

> All the commands are entered in console window. A good way to start working with magic is to go through the tutorials provided here by Tim Edwards.

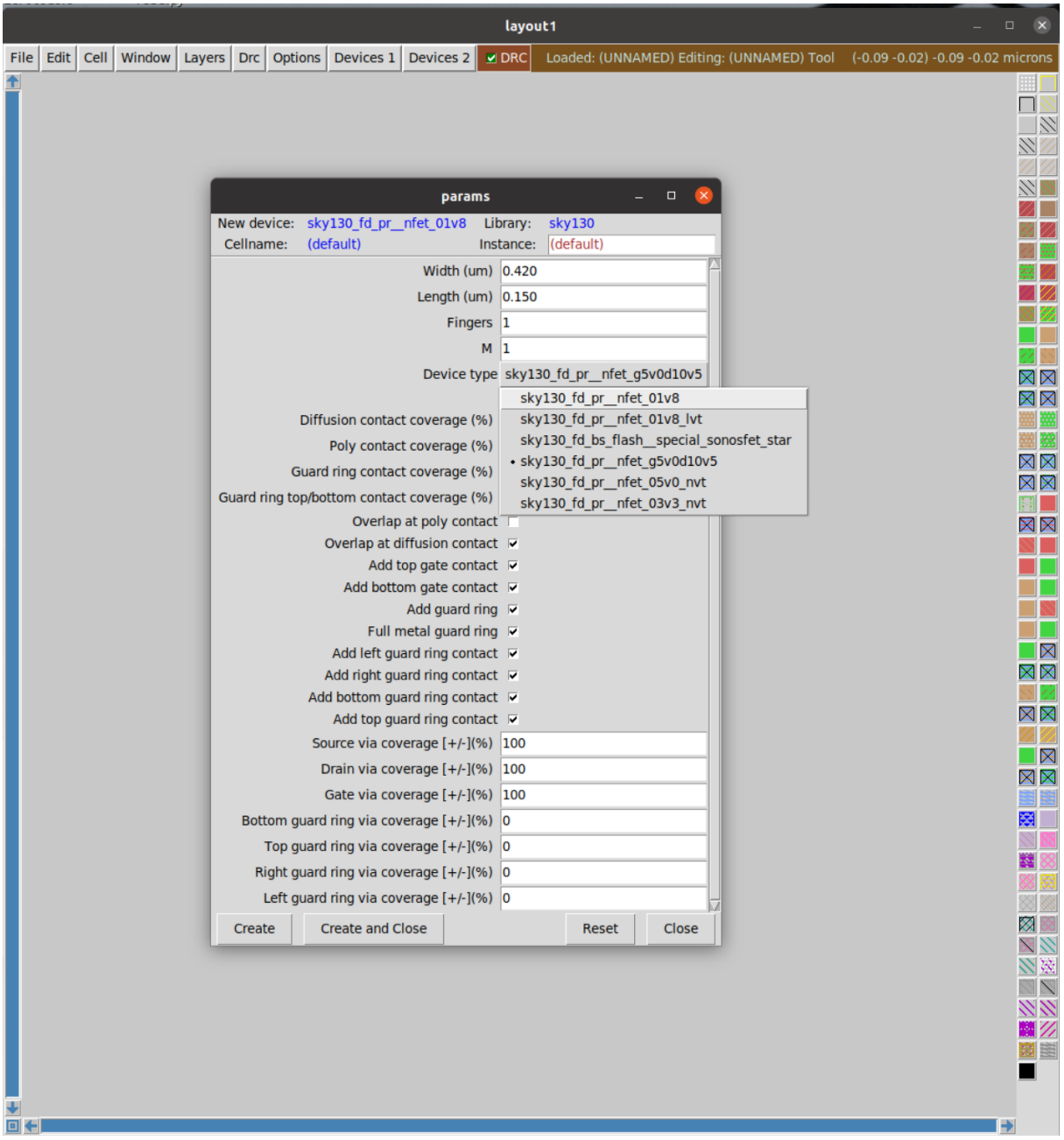
To begin the layout design for the reference opamp, the process involves placing layouts for each transistor from `devices1` ⇒ `nmos/pmos` in Magic Layout Editor. Here are the steps to initiate the layout design:

Manual Placement of Transistors

- **Accessing Transistors:**

  Navigate to `devices1` ⇒ `nmos/pmos` in the Magic tool.

- **Setting Transistor Parameters:**
  - A property box will open where parameters such as width, length, and other transistor properties can be set.
  - Choose the proper transistor type from `Device type`. In this case, `g5v0d10v5` NMOS and PMOS transistors are used.
  - In this way, all the required transistors (or other cells if required) can be placed one by one.



Importing Devices from Spice Netlist:

Alternatively, the layout of all devices can be generated using the spice netlist of the schematic.

- Navigate to `File` ⇒ `Import spice` and locate the correct spice file, typically stored in `~/.xschem/simulations`.
- This action generates the layout of all cells and pins defined in the spice file.

1. **Transistor Placement:**
   - Place all transistors in the desired location to get a rough layout. Select a single cell by placing the white box on top and pressing `'s'`. Use `'m'` to move it to the desired location.
2. **View Control:**
   - Toggle the view of cells using `shift+x`. Use `'s'` and `'m'` after turning off the view to select and move cells.
3. **Cell Properties:**
   - Check the properties of the selected cell using `ctrl + p`.
4. **Show/Hide Cells:**
   - Use `'x'` to show and `'shift+x'` to hide the selected cell.
5. **Saving the Layout:**
   - Once the proper layout is achieved, use the `save` command to save the design.

Layout connection and wiring

When working on layout design, it is good to understand the metal stack of the PDK first. Refer to the Skywater PDK Metal Stack for detailed information on the metal layers used in the layout.

Displaying Metal Layers

- **Metal Layers:**

  Locate the metal layers on the right side of the layout window in Magic.

- **Layer Identification:**

  Place the mouse pointer on the desired layer. The taskbar will display the name of the selected layer.

Show/Hide Layers

- **Displaying a Layer:**

  To show a specific layer in the design, use a left-click on the layer name.

- **Hiding a Layer:**

  Similarly, right-clicking on a visible layer can hide it from the design.

Creating Connections/Wires

Magic Layout Editor functions similarly to a paint tool for IC layout design. The white box in the window is utilized to define the area for various operations like painting, deleting, selecting for copy/paste, and moving. Here's a breakdown of how Magic operates:

- The white box shown in the layout window is used to define the region for different actions.
- The 'left' mouse button controls the lower-left corner of the box.
- The 'right' mouse button controls the upper-right corner of the box.
- The center button is used to add or delete the desired layer within the selected box area.

Connecting Metal Layers using Vias

- **Accessing Vias:**

  To connect one metal layer to another, vias are used. Find the vias in the tool hierarchy: `Devices1` ⇒ `mcon/via1/2/3/4.`

- **Placing Vias:**

  Place vias on the layout in the same way as wires by using the required box size.

Magic provides different tools for the layer design. The default one is the `“box tool”` These tools can be selected by pressing `spacebar key`. Please refer to the tutorials here for the details.

💡 One of the useful tool is wiring tool. It is particularly useful in routing longer wire/path.

To utilize the wiring tool, follow these steps:

1. Select the net you want to route on the layout.
2. Press the spacebar once, and the command window will switch to the WIRING tool.
3. You can now route the wire in both the x and y directions.
4. To adjust the width of the wire, use the center mouse button by scrolling up or down to increase or decrease the width, respectively.
5. After completing the routing, use the right mouse button to exit the routing mode.
6. Press the space key to switch back to the box tool.

Creating and connecting ports (pins)

Pins play important role in connecting the top-level cell (in this case, the opamp) to various input sources, outputs, and other circuits. They serve as the interface between the internal components of the cell and the external environment.

Connecting Imported Pins

- **Accessing Pins:**

  After making all the required connections in the design, locate the pins generated while importing SPICE netlist.

- **Connecting to Nets:**

  Connect pins to the required net by ensuring proper metal connections.

Creating Pins using Labels Manually

If the cells were placed without importing spice netlist, all the pins need to created manually. To do so follow these steps:

- **Create a Label:**

  Select the desired location and size for the pin using the “white box”.

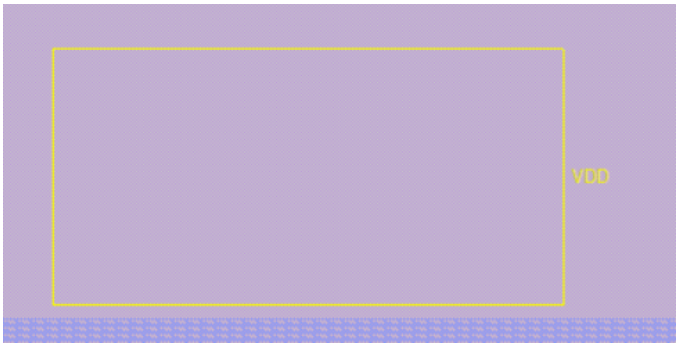
  Type the command:

```
label <name> <direction> <layer>
```

For example:

```
label VDD e metal1
```

This creates a label named VDD in the metal1 layer, pointing to the east direction.



- **Convert the Label to Port (Pin):**

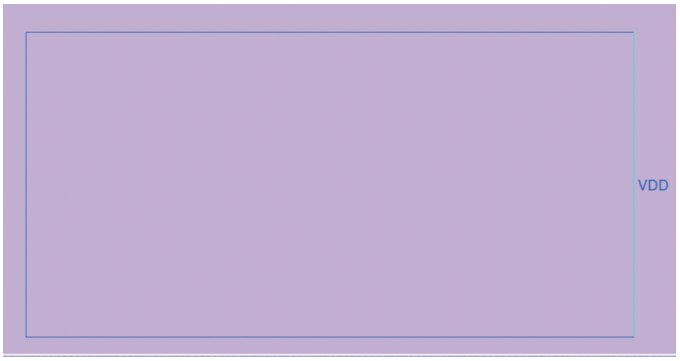
  To convert the label into a pin, select the label and use the command:

  ```
  port make <port number>
  ```

  For example:

  ```
  port make 10
  ```

  This converts the selected label into pin number 10.



- **Repeat for Other Pins:**

  Create labels and convert them into ports for all the required pins.

Ensure proper connections between pins and nets.

`Note`

- Pin matching is crucial when comparing the layout against the schematic.
- For more details regarding command `label` or other Magic commands and their details please refer to the Magic User Guide.

To save the final layout use the commands as follows:

```
writeall
save <filename>
```

- `writeall` command generate the mag file for each cell in the layout.
- `save` command will save the final layout.

The layout will be saved with *.mag extension.

To save it in GDS format, click on `File` ⇒ `Write GDS` and select a location to save the gds.

Layout Design Using KLayout

`Note` → In order to integrate SKY130 in Klayout, macro cells are designed using python. To open them `"pandas"` needs to be installed first. Install it using the following command :

```
pip3 install pandas
```

Launching KLayout

Open KLayout using the following command:

```
klayout -e
```

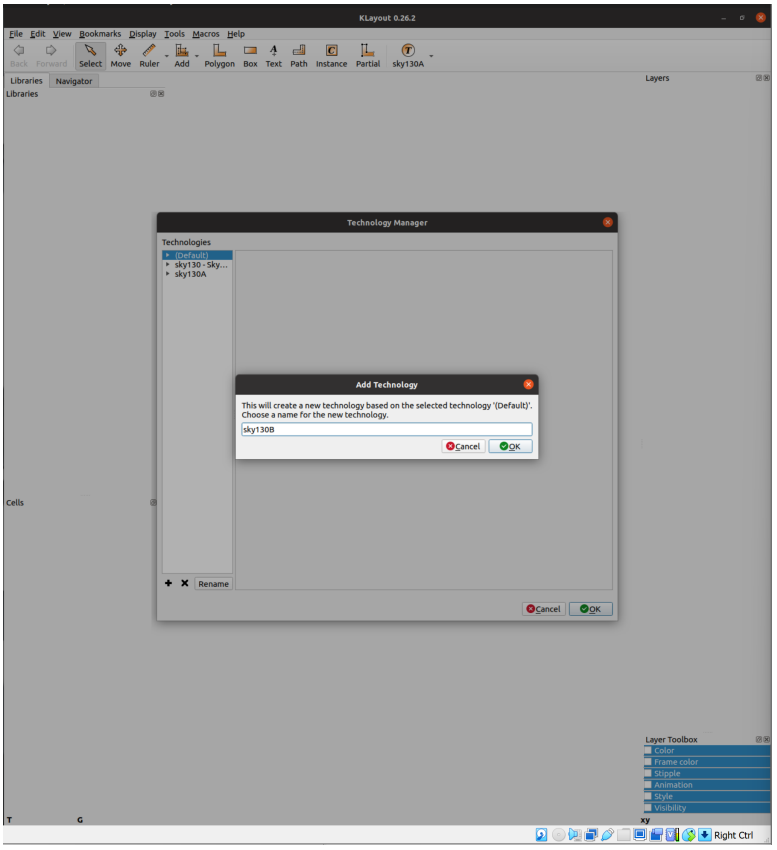
Make sure there are no errors present in the terminal after launching KLayout.

Installing PDK (Process Design Kit)

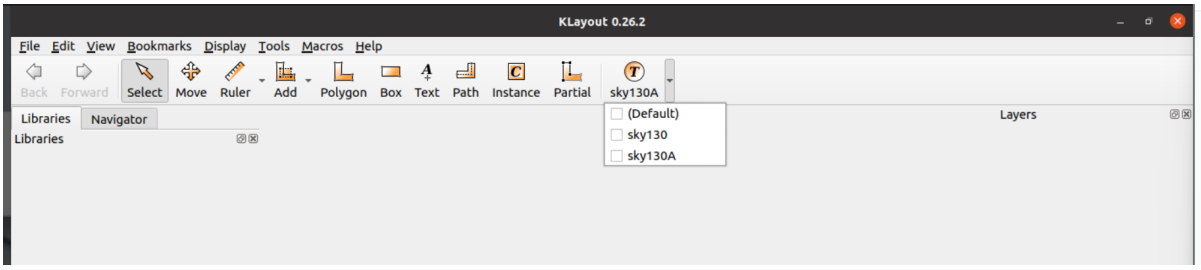
To use KLayout for layout design, the SKY130 PDK needs to be installed. Follow these steps:

- In KLayout, go to `Tools` ⇒ `Manage Technologies`.

- This will open the "Technology Manager" box. Add a new technology by clicking on the `+` button.
- Provide a name for the technology, e.g., "sky130B."



- This will create a folder for the technology inside `~/.Klayout/tech`. Copy all the folders from `/home/zerotoasic/asic_tools/pdk/sky130A/libs.tech/klayout/` to `~/.Klayout/tech/sky130A`.
- After copying the folders, relaunch KLayout.
- Select the technology from the `T` button in the toolbar.



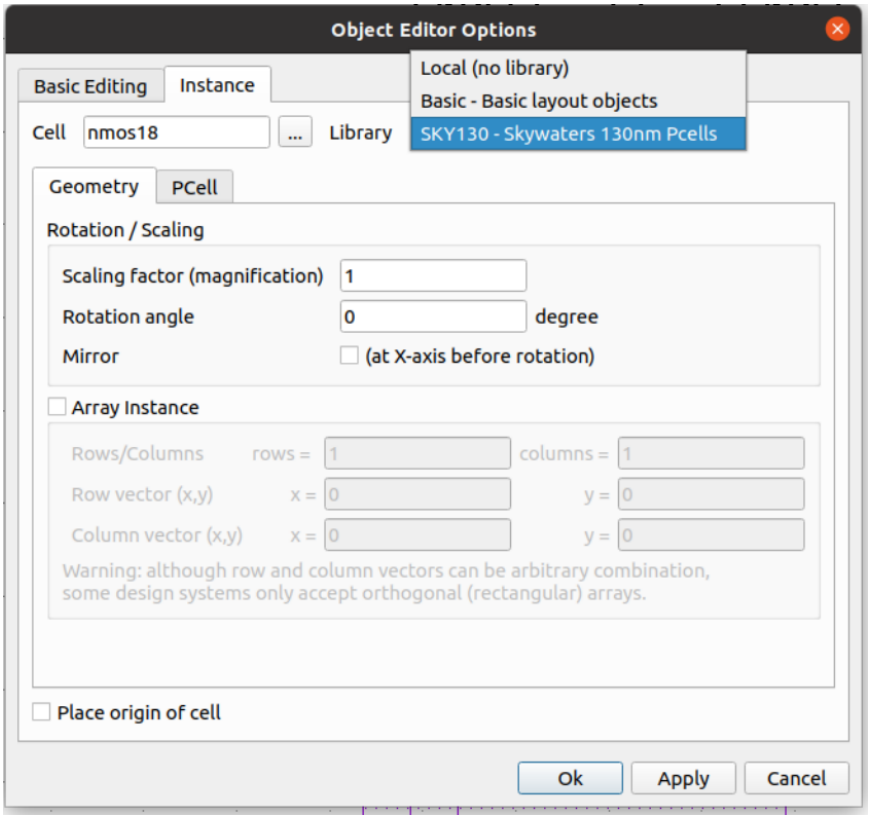
If the PDK is installed properly, there will be a library of `"Skywaters 130nm Pcells"` inside libraries.

Creating a New Layout

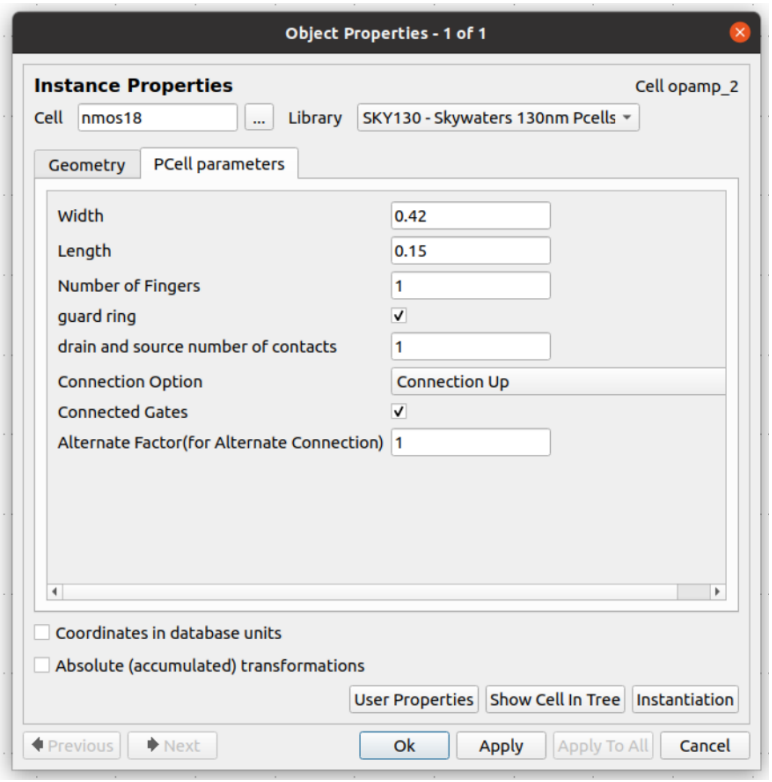
- Open a new layout by going to `File` ⇒ `New Layout`.
- In the dialog box, select the technology, name the Top cell as "opamp," and set the `Database unit` to `0.005um` (sky130 PDK grid). If this unit does not match the grid of PDK, there are issues while exporting the GDS to magic.
- A new layout widow will open after entering all the details.

Placing Parametric Cells

- Place parametric cells by clicking `Instance`.
- Select the desired pcells from the `SKY130 - Skywaters 130nm Pcells` library.



- Change the properties of the cell by pressing `'q'` It will open the `Object Properties box`.



- Modify transistor properties such as width, length, and gate connection type.

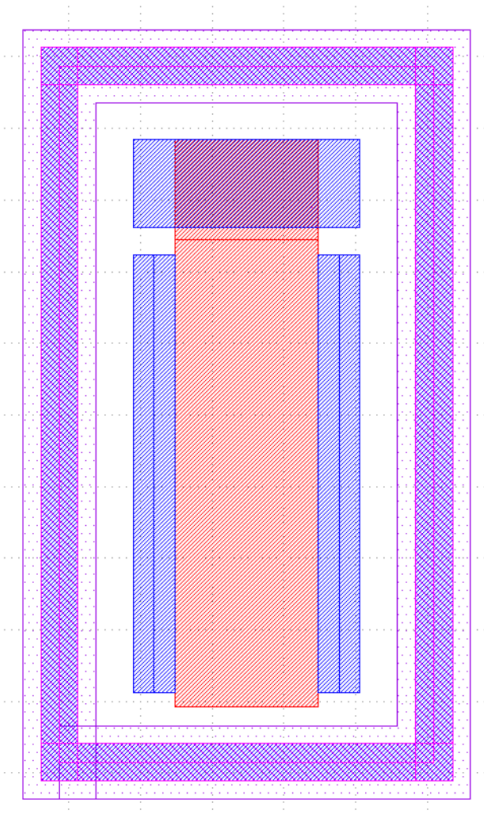
Creating Connections

- Select a layer and click on `Box` to create a box (wire) of the selected layer.
- The `Path` tool can also be used to create wires of the desired width for connections.
- Use the move tool to move the selected object vertically or horizontally.
- All the vias are also present as instance inside the pcell library.

Once the layout is completed, it can be saved in GDS format which can be exported to other layout tools.

Issues faced with Klayout

While using KLayout for layout design, certain issues were encountered, particularly regarding the connection of guard rings around transistors pcells up to Metal2. It restricts the routing of Metal1 and Metal2 in the top cell.



As a workaround, it was suggested to import the layout of each cell (transistor) in GDS format individually from Magic layout editor and then use these exported cells to create the complete layout. However, this solution has not been implemented or tested in this work.

Useful Links

Starting with Magic → Magic Tutorials

All the commands for Magic → Magic User Guide.

Sky 130nm PDK stack → Skywater PDK Metal Stack

KLayout forum for discussion → KLayout Forum

Good video tutorial for Magic →

- Magic Tutorial Part 1
- Magic Tutorial Part 2

Layout versus Schematic (LVS)

Tool Used

- netgen

After completing the layout, a Layout versus Schematic (LVS) check is conducted to ensure matching between the schematic and layout. LVS also identifies any floating nets and checks for open or short circuits in the layout.

Generate SPICE Netlist from Schematic

To create the netlist from xschem for LVS:

- Click on `Simulation` ⇒ `LVS netlist: Top level is a .subckt`.
- Now click on `Netlist` to generate netlist.
- The spice netlist is created in `~/.xschem/simulations/` with a .spice extension.

Generate SPICE Netlist From Layout

After saving the layout in Magic, use the following commands to extract a netlist:

```
extract all
ext2spice lvs
ext2spice
```

- The `extract all` command extracts the top (root) cell and all its children in separate files with .ext extension.
- `ext2spice lvs` runs various built-in commands, ensuring proper formatting and settings for LVS. The commands are as follows :

```
ext2spice hierarchy on
ext2spice scale off
ext2spice format ngspice //Netlist follows NGSPICE format//
ext2spice cthresh infinite //No pariasitic capacitance extr
action during LVS//
ext2spice rthresh infinite //No pariasitic resistance extra
ction during LVS//
ext2spice global off
```

```
ext2spice blackbox on
ext2spice subcircuit top auto
ext2spice renumber off
```

- The final `ext2spice` command generates the ultimate SPICE netlist, which is then saved in the folder from where Magic was launched. The current working directory can be obtained using the `pwd` command in the Magic command window.

Install Netgen

If Netgen is not installed, follow these steps:

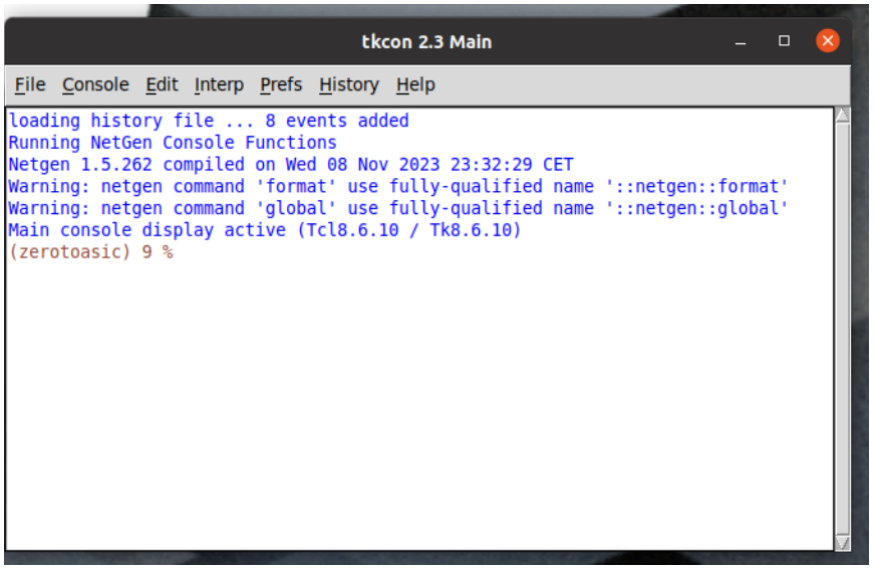
- Download the latest version from here.
- Unzip the folder and use the commands:

```
./configure
make
make install
```

Running LVS with Netgen

Launch Netgen with the command:

```
netgen
```



To run LVS:

```
lvs <spice file 1> <spice file 2> <path to netgen tech file
>
```

Run Netgen in batch mode:

```
netgen -batch lvs <netlist 1> <netlist 2> <path to netgen t
ech file>
```

The output log is created in a file named `comp.out`.

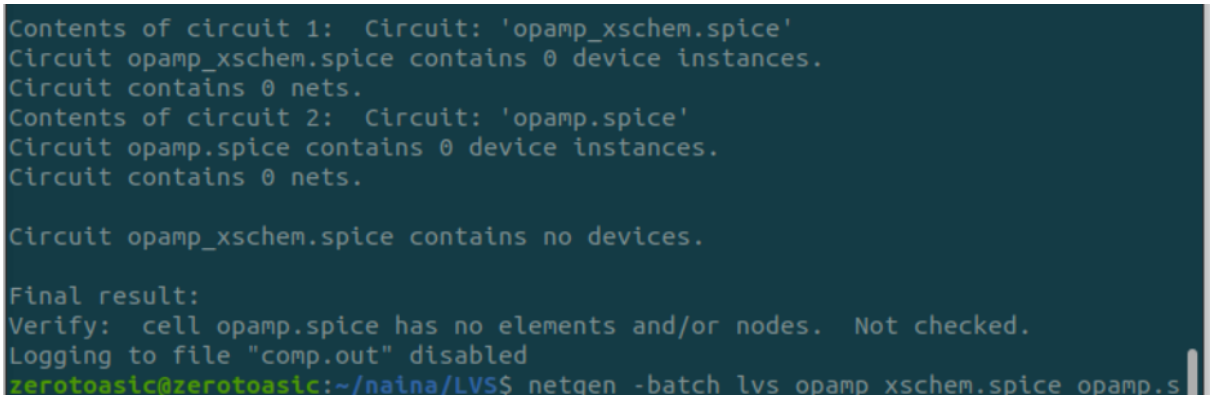
Example

Consider the current circuit (opamp), where two netlists are generated. The spice netlist from the schematic is located at `~/.xschem/simulations/opamp.spice`, while the netlist from the layout is in the working folder of Magic. The schematic netlist is renamed as `opamp_xschem.spice`.

Once both netlists are stored in a folder (or path to the files is known), LVS is executed as follows:

```
netgen -batch lvs opamp_xschem.spice opamp.spice /home/zero
toasic/asic_tools/pdk/sky130A/libs.tech/netgen/sky130A_setu
p.tcl
```

This will result in following error :



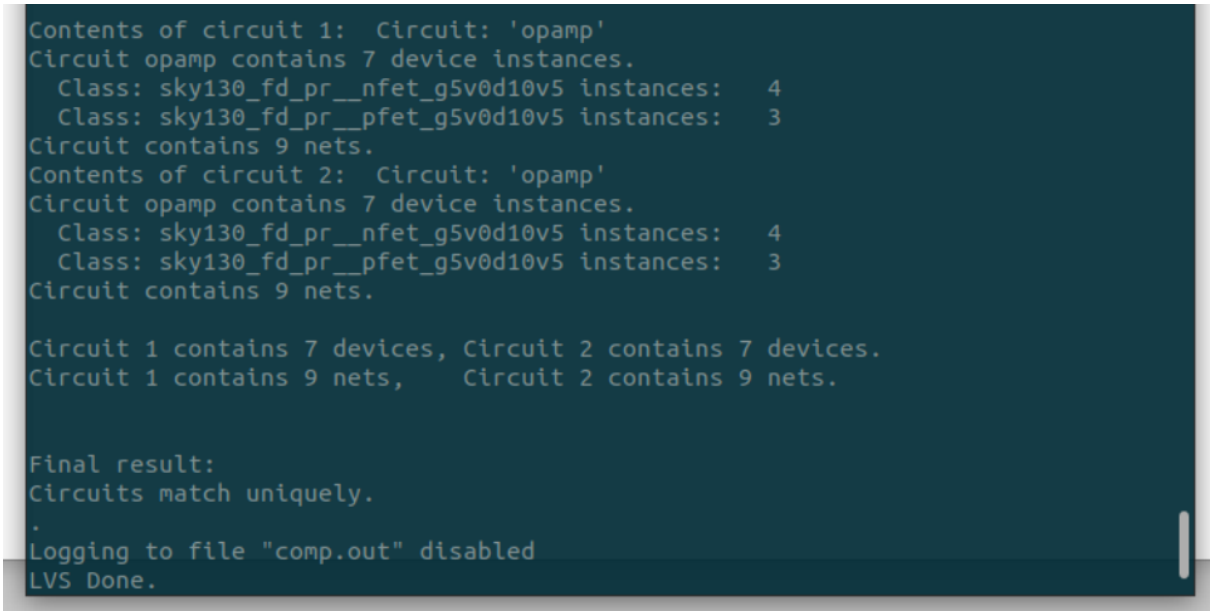
This error indicates that the circuits are empty despite content in both netlists. It is because the top level cell (opamp) is not instantiated in the netlists.

- To resolve this, define the top level cell (the cell that needs to be compared) with the netlist as below:

```
netgen -batch lvs "opamp_xschem.spice opamp" "opamp.spice o
pamp" /home/zerotoasic/asic_tools/pdk/sky130A/libs.tech/net
gen/sky130A_setup.tcl
```

In above code, the top-level cell is defined along with the netlists to be compared. This enables Netgen to recognize the subcircuits needing comparison, searching for them inside the specified netlist.

The final result of the LVS process is displayed, detailing the hierarchical comparison of devices, nets, and pins between the two circuits.



Netgen manual

For a comprehensive tutorial on Netgen, refer to this page.

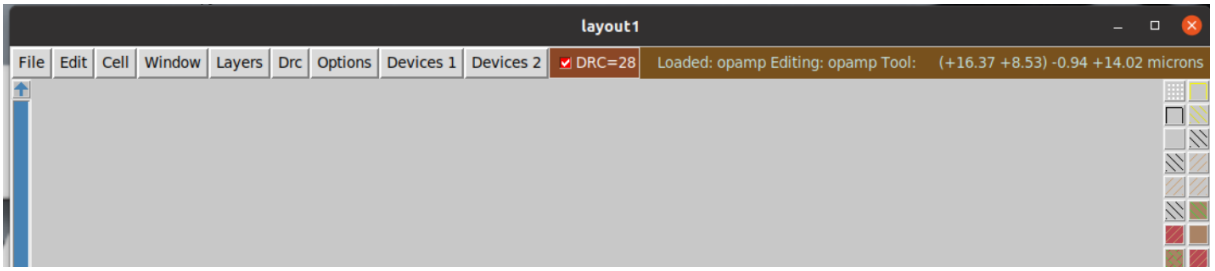
Design Rule Check (DRC)

Tool Used

- Magic

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.

- For understanding of the DRC rules specific to the SKY 130 Process Design Kit (PDK), please refer to the official documentation available here.
- In the Magic layout editor, all DRC errors are visible during the layout editing process:



- The first layer in the layer window serves as the error layer, highlighting all instances where DRC violations occur.

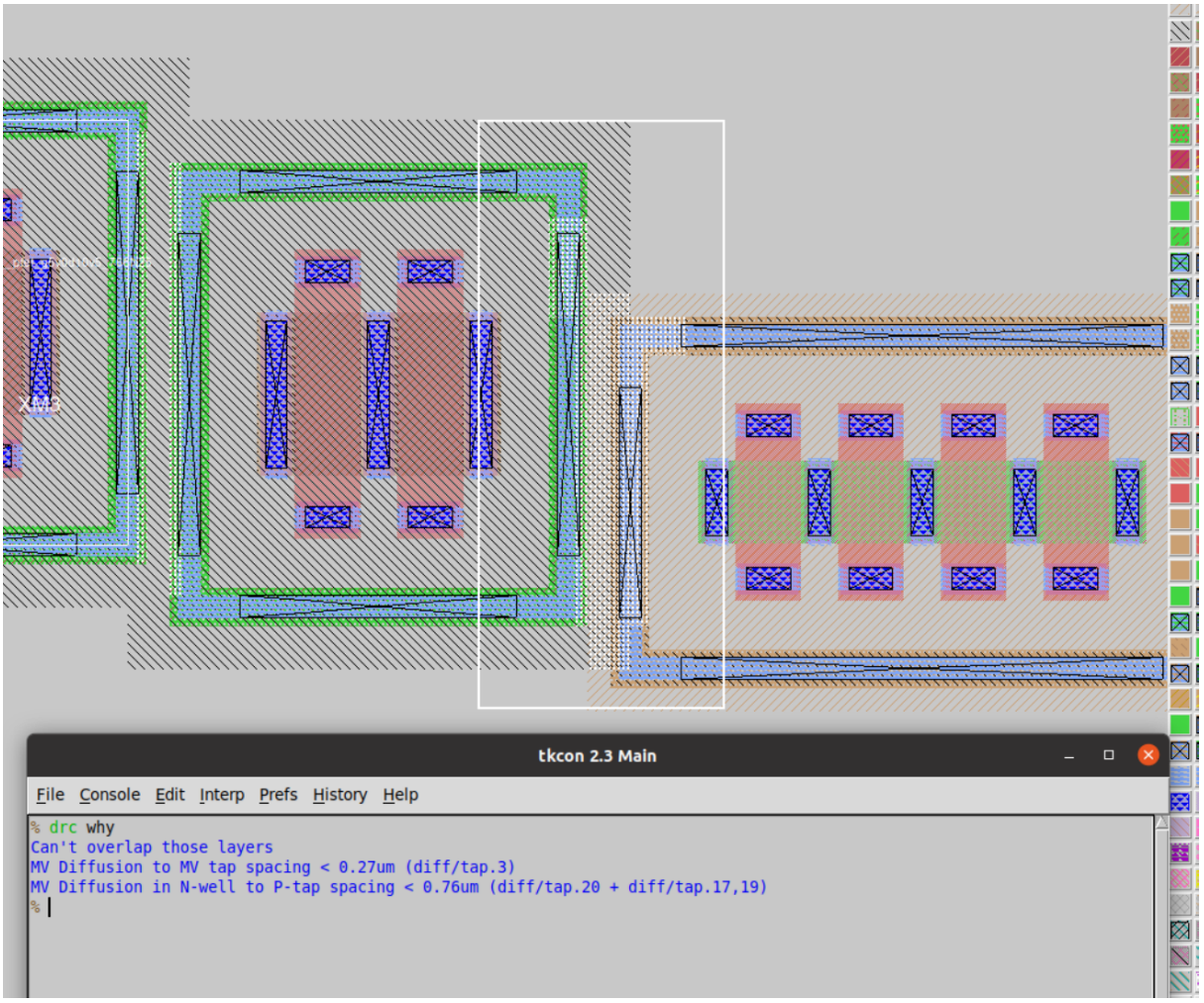
To view the full DRC errors, use the following command in the Magic command window:

```
drc style drc(full)
```

To see the explanation for a particular error select it and use command:

```
drc why
```

Example



In the above snapshot, the selected area has two DRC errors. Understanding these errors is crucial for rectifying the design issues. The identified errors are:

- **Proximity of Diffusion Area and Local Interconnect Via:**

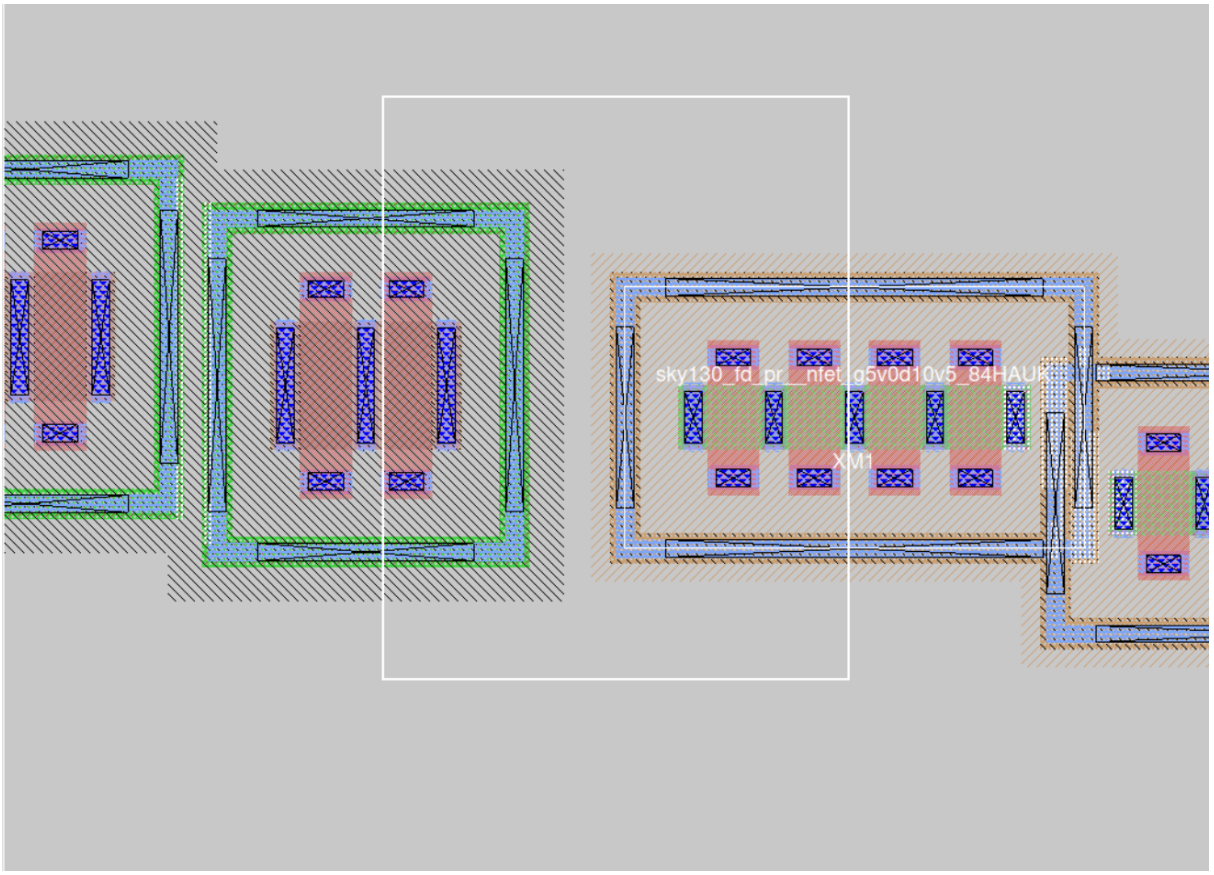
  The first error arises due to the close proximity of the diffusion area and the local interconnect via.

- **Overlap of nwell and pwell Layers:**

  The second error is associated with the overlap of the nwell and pwell layers. DRC flags this condition as an error since the nwell and pwell, which typically represent different doping regions, should not overlap.

- **Resolution:**

  To address these errors, consider adjusting the placement of the transistors in the highlighted area. Moving one transistor appropriately can effectively resolve both DRC issues.



- Go to next DRC error using the following command:

```
drc find
```

All DRC errors can be fixed following the same process.

Parasitic Extraction and Post-Layout Simulation

Parasitic extraction is a crucial step in integrated circuit (IC) design that involves modeling and analyzing the parasitic elements in the layout. These parasitic elements can significantly impact the performance of the IC and need to be considered for accurate simulation and design optimization.

Tools Used

- Magic
- xschem
- ngspice

Firstly, Magic is used to extract the parasitic resistance and capacitance from the layout.

Parasitic capacitance extraction

- **Load the layout in Magic:**

```
load opamp.mag
```

- **Generate Spice Netlist with Parasitic Capacitance Extraction:**

```
extract all
ext2spice hierarchy on //Extraction is hierarchal//
ext2spice scale off
ext2spice cthresh 0 //Define the thrushold value for parasi
tic capacitance//
ext2spice -d -o opamp_postlayout.spice -f ngspice //save th
e extracted SPICE netlist as opamp_postlayout.spice//
```

- **Review Parasitic Capacitance:**

Open `opamp_postlayout.spice` to inspect all parasitic capacitances present in the layout.

> 💡 *Note: If hierarchy is turned off using `ext2spice hierarchy off`, the extraction of parasitic capacitance is more conservative since the cell is flattened.*

Parasitic resistance extraction

The parasitic resistance extraction is bit tricky. It uses a chain of commands including `“extract”`, `“ext2sim”`, `“extresist”` and `“ext2spice”`.

- **Load Layout in Magic:**

```
load opamp.mag
```

- **Execute Parasitic Resistance Extraction Commands:**

1. **Convert Hierarchical Netlist:**

   ```
   extract all
   ext2sim labels on
   ext2sim
   ```

   The `ext2sim` command transforms the hierarchical netlist generated by the `extract` command into a flattened representation stored in **.sim** format.

2. **Generate Detailed Resistance Model:**

   ```
   extresist tolerance 10
   extresist
   ```

   The `extresist` command enhances the resistance model by replacing long network routes and branching routes with resistor devices and networks. Resistances above the specified tolerance (threshold) value are generated.

3. **Parasitic Capacitance Extraction and SPICE Netlist Generation:**
   Follow the subsequent commands for parasitic capacitance extraction and SPICE netlist generation, similar to the process outlined earlier.

   ```
   ext2spice lvs
   ext2spice cthresh 0
   ```

```
ext2spice extresist on
ext2spice -d -o opamp_postlayout_withR.spice -f ngspice
```

These commands ensure the extraction of parasitic capacitance, incorporate the enhanced resistance model, and generate the final SPICE netlist.

- `All together the commands for PEX are as follows:`

```
load opamp.mag
select top cell
extract all
ext2sim labels on
ext2sim
extresist tolerance 10
extresist
ext2spice lvs
ext2spice cthresh 0
ext2spice extresist on
ext2spice -d -o opamp_postlayout_withR.spice -f ngspice
```

`Issue →`

One challenge encountered during the parasitic resistance extraction process is the presence of dummy transistors with zero length and width in the generated SPICE netlist. This issue is illustrated in the snapshot of the netlist below:

```
69 XXM8 VDD VDD OUT m1_4266_424# m1_4266_424# m1_4266_424# VDD m1_4266_424# VDD OUT
   sky130_fd_pr__pfet_g5v0d10v5_2S4URJ
70 XXM9 VSS BIAS1 VSS BIAS1 BIAS1 BIAS1 VSS OUT VSS OUT sky130_fd_pr__nfet_g5v0d10v5_DTEAGN
71 X0 OUT BIAS1.t4 VSS VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan w=0 l=0
72 **devattr s=2900,158 d=2900,158
73 X1 VSS BIAS1.t1 m1_1174_n578# VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan w=0 l=0
74 **devattr s=5800,316 d=2900,158
75 X2 m1_2346_n562.t3 m1_2346_n562.t2 VDD VDD sky130_fd_pr__pfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan
   w=0 l=0
76 **devattr s=14500,616 d=7250,308
77 X3 m1_4266_424# INN.t0 m1_1174_n578# VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan
   w=0 l=0
78 **devattr s=3654,184 d=3654,184
79 X4 VDD m1_2346_n562.t0 m1_2346_n562.t1 VDD sky130_fd_pr__pfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan
   w=0 l=0
80 **devattr s=7250,308 d=14500,616
81 X5 VSS BIAS1.t7 OUT VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan w=0 l=0
82 **devattr s=2900,158 d=5800,316
83 X6 m1_1174_n578# BIAS1.t2 VSS VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan w=0 l=0
84 **devattr s=2900,158 d=2900,158
85 X7 m1_1174_n578# INN.t3 m1_4266_424# VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan
   w=0 l=0
86 **devattr s=3654,184 d=7308,368
87 X8 m1_4266_424# INN.t1 m1_1174_n578# VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan
   w=0 l=0
88 **devattr s=7308,368 d=3654,184
89 X9 VSS BIAS1.t0 m1_1174_n578# VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan w=0 l=0
90 **devattr s=2900,158 d=2900,158
91 X10 OUT BIAS1.t5 VSS VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan w=0 l=0
92 **devattr s=5800,316 d=2900,158
93 X11 m1_1174_n578# INN.t2 m1_4266_424# VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan
   w=0 l=0
94 **devattr s=3654,184 d=3654,184
95 X12 m1_4266_424# m1_2346_n562.t4 VDD VDD sky130_fd_pr__pfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan
   w=0 l=0
96 **devattr s=14500,616 d=7250,308
97 X13 m1_1174_n578# BIAS1.t3 VSS VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan w=0 l=0
98 **devattr s=2900,158 d=5800,316
99 X14 VDD m1_2346_n562.t5 m1_4266_424# VDD sky130_fd_pr__pfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan
   w=0 l=0
100 **devattr s=7250,308 d=14500,616
101 X15 VSS BIAS1.t6 OUT VSS sky130_fd_pr__nfet_g5v0d10v5 ad=-nan pd=-nan as=-nan ps=-nan w=0 l=0
102 **devattr s=2900,158 d=2900,158
103 R0 BIAS1.n1 BIAS1.t7 212.305
104 R1 BIAS1.n2 BIAS1.t5 212.305
105 R2 BIAS1.n5 BIAS1.t1 212.291
106 R3 BIAS1.n6 BIAS1.t3 212.291
107 R4 BIAS1.t1 BIAS1.n4 212.291
```

The extracted transistors, labeled from X1 to X15, have dimensions specified as width (w) and length (l) equal to zero.

Fix:

To address this issue, it is recommended to flatten the top cell using the following commands:

```
flatten <cell name>
load <cell name>
```

For Example :

```
load opamp
flatten opamp_flat
```

```
load opamp_flat
select top cell
```

After flattening the top cell, proceed with the previous commands for parasitic R & C extraction.

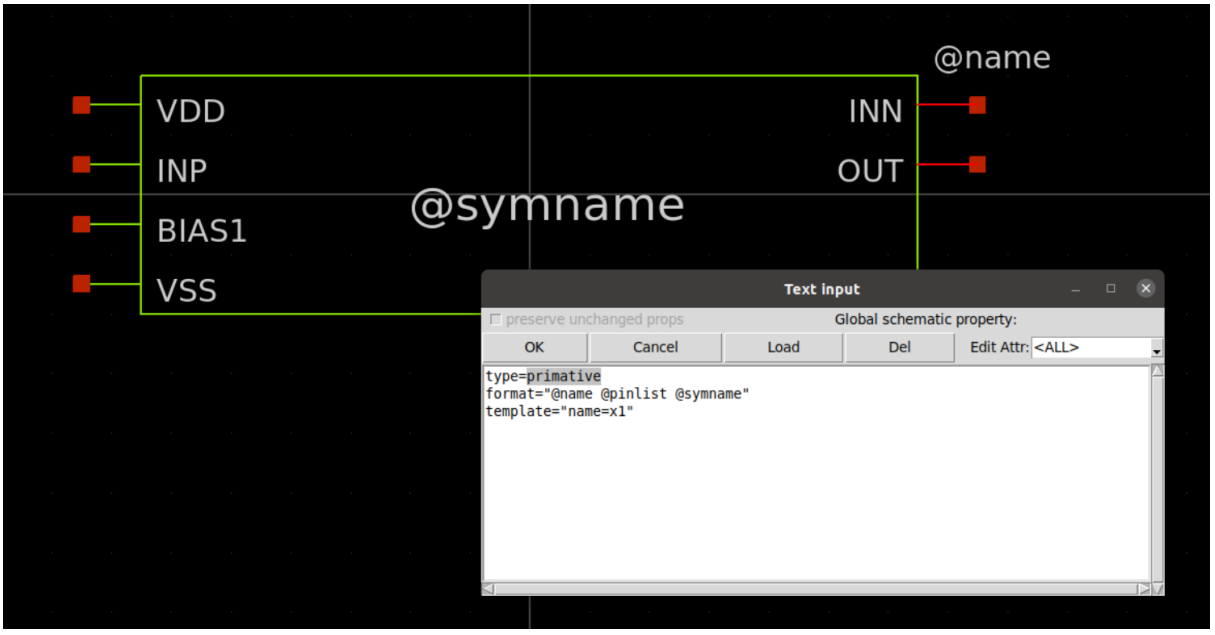
Running Post-Layout Simulation

To check the impact of parasitic resistance (R) and capacitance (C) on the functionality of the circuit, follow these steps to incorporate the extracted netlist into the simulation:

- **Modify Top Cell Symbol Properties:**

  Open the top cell symbol (e.g., opamp.sym).

  In its properties, change the ‘type’ from `"subcircuit"` to `"primitive"`.



- **Include Extracted Netlist in Test Bench Simulation Setup:**

  Open the test bench (e.g., opamp_tb.sch).

  Inside the simulation setup `(code.sym)`, include the extracted netlist using the `.include` command.

```
.include <SPICE file>
```

For example, in the current case:

```
.include /home/zerotoasic/naina/Magic_scrh/opamp_postlayout
_withR.spice
```

- **Verify Pin Order Alignment:**

  Ensure that the pin order of the cell in the test bench `(opamp.sym)` matches the pin order of the netlist with parasitic R & C `( opamp_postlayout_withR.spice)`.

  If needed, edit the pin order of the second netlist (or change the port order in layout).

- **Edit Top Cell Name:**

  Edit the top cell name in the final netlist `(opamp_postlayout_withR.spice)` from the flattened cell `(opamp_flat)` to the original one `(opamp)` to ensure correct post-layout simulation.

- **Create Netlist and Run Simulation:**

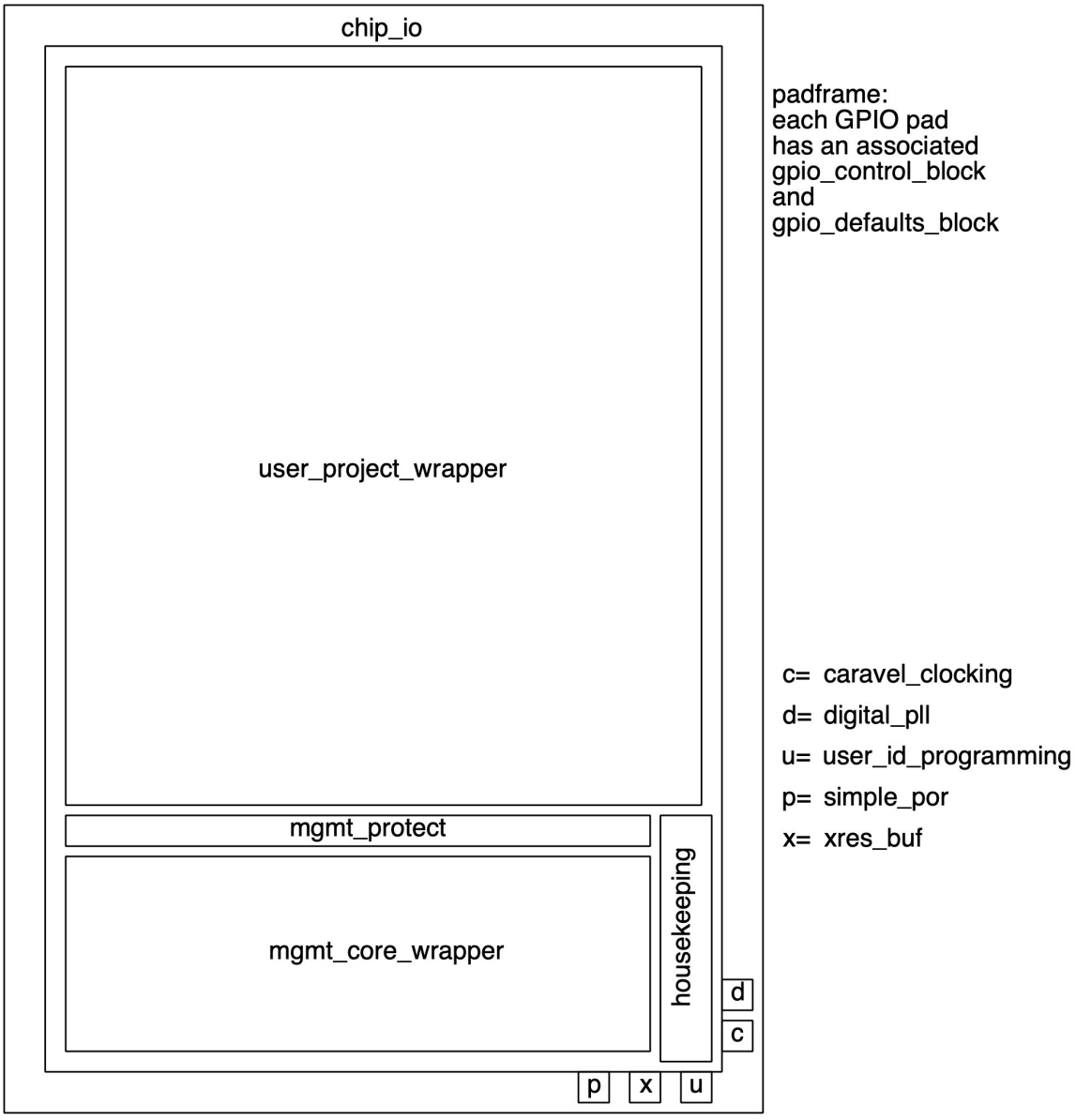
  Generate the netlist for the test bench and proceed to run the simulation, following standard procedures.

Useful Links

- SKY130 PDK Parasitic Calculation: Detailed information on parasitic calculation in SKY130 PDK.
- Magic Parasitic Extraction Fundamentals: A tutorial providing fundamental insights into parasitic extraction using Magic.
- Open Source Silicon Forum: A valuable online open-source forum for finding solutions and engaging in discussions related to various aspects of silicon design and development using open source tools.

Integration of Final Layout with Analog Caravel (Caravan) Wrapper

Caravel/Caravan



The Efabless Caravel chip is a ready-to-use test harness for creating designs with the Google/Skywater 130nm Open PDK. The Caravel harness comprises of base functions

supporting IO, power and configuration as well as drop-in modules for a management SoC core, and a `2.92 mm x 3.52 mm` open project area for the placement of user IP blocks.

The Caravel wrapper divides the complete area into two sections, as illustrated above:

- **Management Area**

The management area is a drop-in module implemented as a separate repo. It typically includes a RISC-V based SoC that includes a number of peripherals like timers, uart, and gpio. The management area runs firmware that can be used to:

  - Configure User Project I/O pads
  - Observe and control User Project signals (through on-chip logic analyzer probes)
  - Control the User Project power supply

- **User Project Area**

This is the user space. It has a limited silicon area `2.92mm x 3.52mm` as well as a fixed number of I/O pads `38` and power pads `4`.

The user space has access to the following utilities provided by the management SoC:

  - `38` IO Ports
  - `128` Logic analyzer probes
  - Wishbone port connection to the management SoC wishbone bus.

There are two design process given by efabless.

- **Digital User Project**
  - If you are building a digital project for the user space, check a sample project at caravel_user_project.
  - If you will use OpenLANE to harden your design, go through the instructions in this README.

Digital user projects should adhere the following requirements:

  - Top module is named `user_project_wrapper`
  - The `user_project_wrapper` adheres to the pin order defined for digital wrapper.
  - The user project repository adheres to the Required Directory Structure.

- **Analog User Project**

If you are building an analog project for the user space, check a sample project at caravel_user_project_analog.

Analog user projects should adhere the following requirements:

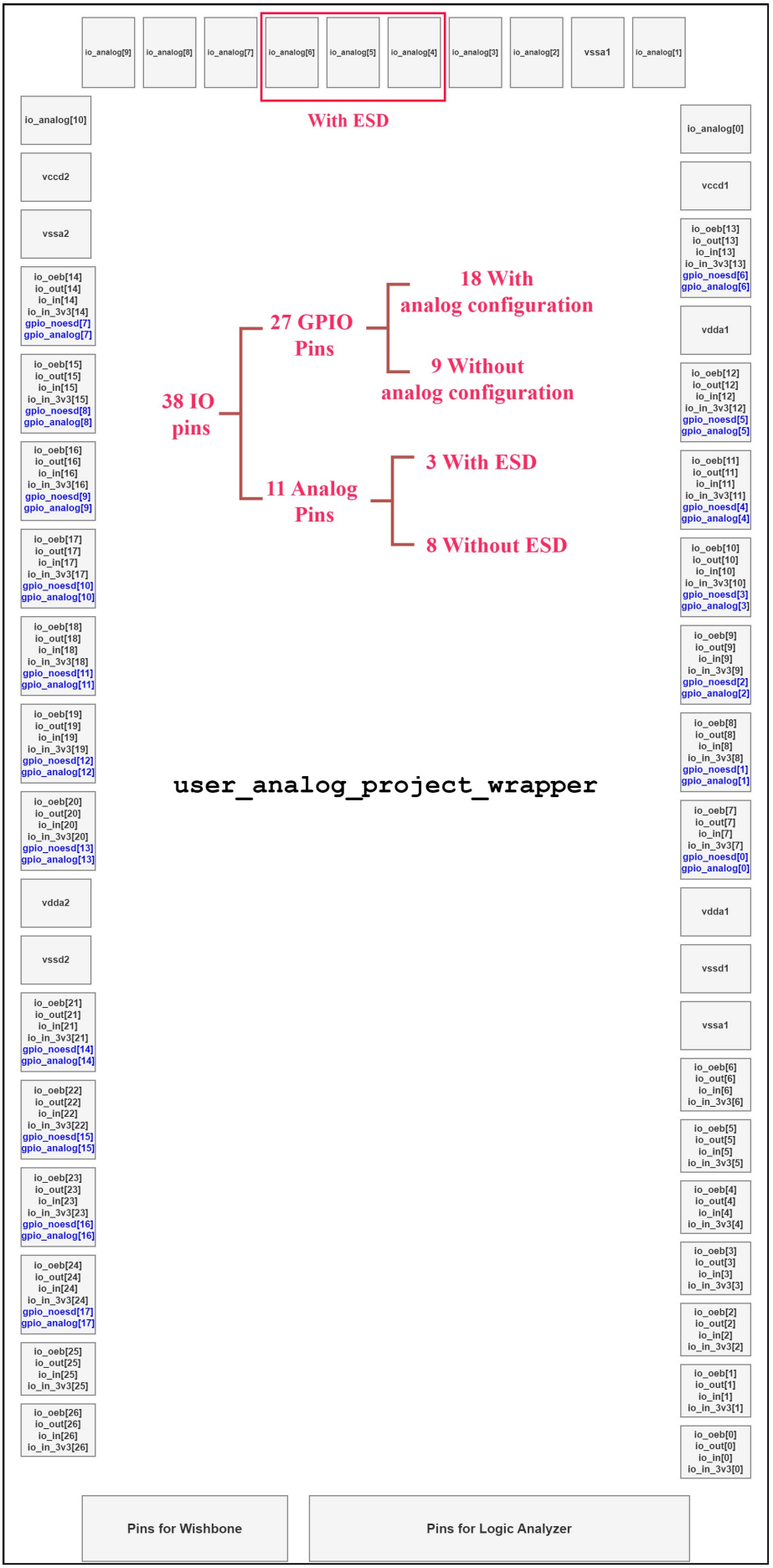
- Top module is named `user_analog_project_wrapper`
- **The requirements regrading pin order and project repository are explained in details below.**

Pin Description

There are 38 IO pins defined as `mprj_io[37:0]`. These pins can be used as general input or output pins (GPIO). In case of an analog design, some of these pins can be configured as analog pins.

> 💡 **In this document, the pin order and pin description is provided for `user_analog_project_wrapper` since the documentation focuses on analog circuit design**

The figure below shows the pin configuration of caravel analog wrapper:


io_analog[9]
io_analog[8]
io_analog[7]
io_analog[6]
io_analog[5]
io_analog[4]
io_analog[3]
io_analog[2]
vssa1
io_analog[1]
With ESD
io_analog[10]
vccd2
vssa2
io_oeb[14] io_out[14] io_in[14] io_in_3v3[14] gpio_noesd[7] gpio_analog[7]
io_oeb[15] io_out[15] io_in[15] io_in_3v3[15] gpio_noesd[8] gpio_analog[8]
io_oeb[16] io_out[16] io_in[16] io_in_3v3[16] gpio_noesd[9] gpio_analog[9]
io_oeb[17] io_out[17] io_in[17] io_in_3v3[17] gpio_noesd[10] gpio_analog[10]
io_oeb[18] io_out[18] io_in[18] io_in_3v3[18] gpio_noesd[11] gpio_analog[11]
io_oeb[19] io_out[19] io_in[19] io_in_3v3[19] gpio_noesd[12] gpio_analog[12]
io_oeb[20] io_out[20] io_in[20] io_in_3v3[20] gpio_noesd[13] gpio_analog[13]
vdda2
vssd2
io_oeb[21] io_out[21] io_in[21] io_in_3v3[21] gpio_noesd[14] gpio_analog[14]
io_oeb[22] io_out[22] io_in[22] io_in_3v3[22] gpio_noesd[15] gpio_analog[15]
io_oeb[23] io_out[23] io_in[23] io_in_3v3[23] gpio_noesd[16] gpio_analog[16]
io_oeb[24] io_out[24] io_in[24] io_in_3v3[24] gpio_noesd[17] gpio_analog[17]
io_oeb[25] io_out[25] io_in[25] io_in_3v3[25]
io_oeb[26] io_out[26] io_in[26] io_in_3v3[26]
38 IO pins
27 GPIO Pins
18 With analog configuration
9 Without analog configuration
11 Analog Pins
3 With ESD
8 Without ESD
user_analog_project_wrapper
io_analog[0]
vccd1
io_oeb[13] io_out[13] io_in[13] io_in_3v3[13] gpio_noesd[6] gpio_analog[6]
vdda1
io_oeb[12] io_out[12] io_in[12] io_in_3v3[12] gpio_noesd[5] gpio_analog[5]
io_oeb[11] io_out[11] io_in[11] io_in_3v3[11] gpio_noesd[4] gpio_analog[4]
io_oeb[10] io_out[10] io_in[10] io_in_3v3[10] gpio_noesd[3] gpio_analog[3]
io_oeb[9] io_out[9] io_in[9] io_in_3v3[9] gpio_noesd[2] gpio_analog[2]
io_oeb[8] io_out[8] io_in[8] io_in_3v3[8] gpio_noesd[1] gpio_analog[1]
io_oeb[7] io_out[7] io_in[7] io_in_3v3[7] gpio_noesd[0] gpio_analog[0]
vdda1
vssd1
vssa1
io_oeb[6] io_out[6] io_in[6] io_in_3v3[6]
io_oeb[5] io_out[5] io_in[5] io_in_3v3[5]
io_oeb[4] io_out[4] io_in[4] io_in_3v3[4]
io_oeb[3] io_out[3] io_in[3] io_in_3v3[3]
io_oeb[2] io_out[2] io_in[2] io_in_3v3[2]
io_oeb[1] io_out[1] io_in[1] io_in_3v3[1]
io_oeb[0] io_out[0] io_in[0] io_in_3v3[0]
Pins for Wishbone
Pins for Logic Analyzer

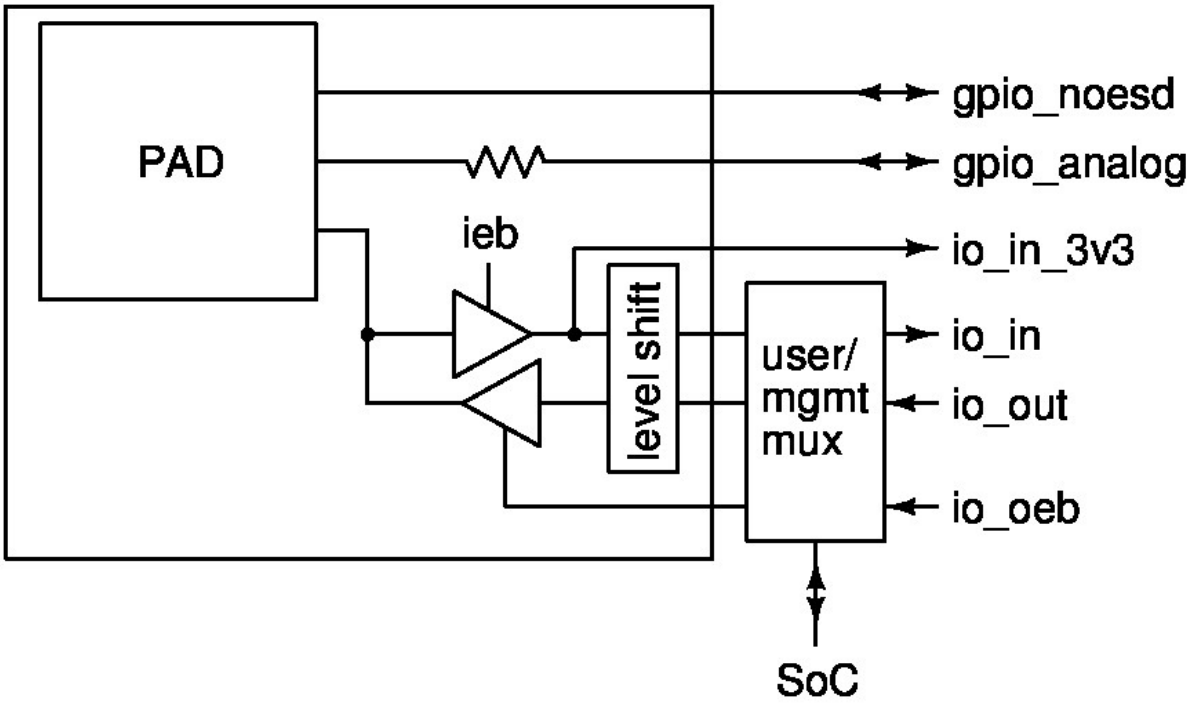
Power & Ground Pins

There are a total of four power and ground pins present in wrapper.

- vccd1/vccd2 : Power pins for 1.8 V.
- vdda1/vdda2 : Power pins for 3.3 V - 5.5 V.
- vssd1/vssd2 : Ground pins for 1.8 V supply.
- vssa1/vssa2 : Ground pins for 3.3 V - 5.5 V supply.

GPIO Pins

There are 27 general purpose IO pins. They can be configured in several ways:



Above diagram illustrates different configurations of GPIO pins and their final connection to the pad.

- **io_in → 1.8 V input pin.**
- **io_out → 1.8 V output pin.**
- **io_oeb → output enable bar (active low) pin.**

This configuration works as a tristate buffer. If io_oeb is high, the pin works as an input pin and input buffers are on. If io_oeb is low, the pin works as an output pin and output buffer is on.

- **io_in_3v3 → 3.3V input pin.**
- **gpio_analog → General purpose analog pin with ESD protection.** `The ESD adds a 150 ohm of series resistance.`
- **gpio_noesd → General purpose analog pin without ESD protection.**

Analog Pins

There are 11 pins present in thew wrapper which can be specifically used for analog connection (input or output).

- **analog_io[10:0] → Analog pins.**

`analog_io[6:4] has ESD protection for up to 5.5 V.`

Logic analyzer and Wishbone Pins

- The logic analyzer is a 128-bit port going to/from the user area and management SoC. It allows the SoC to "probe" any desired signals from the user area after fabrication (but they cannot be changed once fabricated, so choose wisely). This is useful for debugging signals during the bring up process, possibly to discover bugs in the hardware, or to verify that they are working correctly.
- The wishbone bus is a 32-bit bus connecting the user project wrapper with the management SoC.
  It includes the following user project wrapper ports:
- **Port Description for Wishbone bus**
  Clock
  ⇒ wb_clk_i

  Reset ⇒ wb_rst_i

  Strobe ⇒ wbs_stb_i

  Cycle ⇒ wbs_cyc_i

  Write enable ⇒ wbs_we_i

  Select ⇒ wbs_sel_i

  Data in (to user area) ⇒ wbs_dat_i

Data out (to SoC) ⇒ wbs_dat_o

Address ⇒ wb_adr_i

Acknowledgement ⇒ wbs_ack_o

`Logic analyzer and wishbones bus are not utilised in current design.`

Integrating with Caravan

Layout Design

To integrate your layout with the Caravel framework for analog designs (caravan), follow these steps:

1. Clone the latest Caravan repository for analog designs from here.
2. Load the layout located at `*~/caravel_user_project_analog/mag/user_analog_project_wrapper_empty.mag**` into Magic.
3. Save the loaded layout as `**user_analog_project_wrapper**`.
4. To incorporate the layout of the opamp cell into the wrapper, follow these sub-steps:

   a. Navigate to `**Cell**` ⇒ `**Place Instance**` in the Magic tool.

   b. Select `opamp.mag` from the correct location. Once selected, the opamp cell will be placed inside the wrapper.
5. Connect the input, output, and other relevant ports of the opamp cell to the corresponding ports in the wrapper.
6. Save the changes made to the layout.
7. To generate the GDS of the final layout, navigate to `*File**` ⇒ `*GDS Write**`.
8. With this, the integration process is complete, and you have successfully integrated your analog layout with the Caravel framework.

Schematic Design

It is advisable to create a comprehensive schematic that includes the analog wrapper and the top cell (opamp). This aids in understanding the connections, and the final netlist is crucial for the LVS precheck.

A schematic is given in efabless repo cloned previously. It is located in `~/caravel_user_project_analog/xschem/user_analog_project_wrapper.sch`. Please use this example as a reference; it encompasses all the pins present in the wrapper.

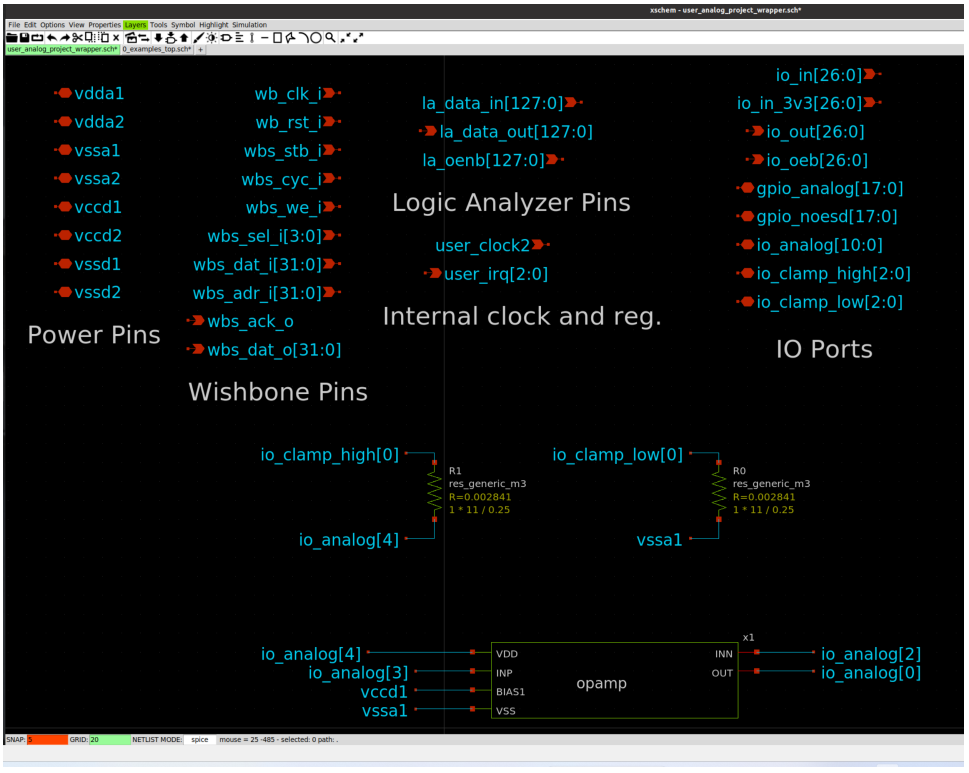
Opamp Connections Inside the Wrapper

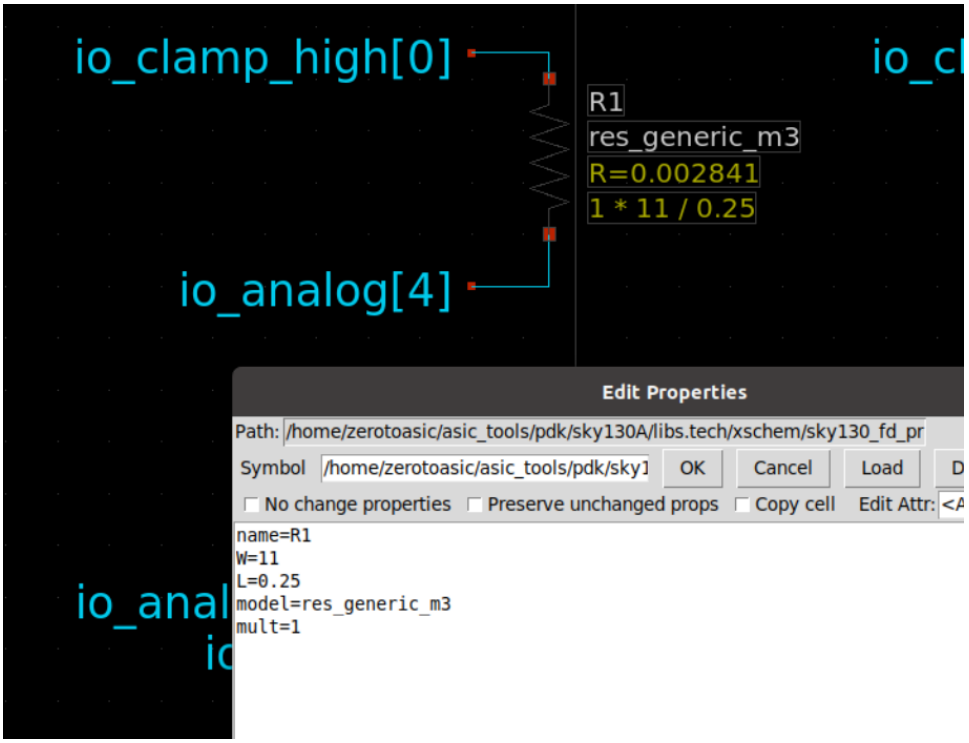
- The input and output pins INP, INN and OUT are connected to io_analog pins io_analog[3], io_analog[2] and io_analog[0] (analog pins without ESD) respectively.
- The supply `VDD` and biasing `BIAS1` are connected to `io_analog[4]` (analog pin with ESD) and `vccd1` respectively.
- Finally the VSS pin is connected to vssa1.

The connection for `io_analog[4]` pin requires special attention, as it comes with ESD protection. The ESD has two clamp pins: `io_clamp_high[0]` and `io_clamp_low[0]`.

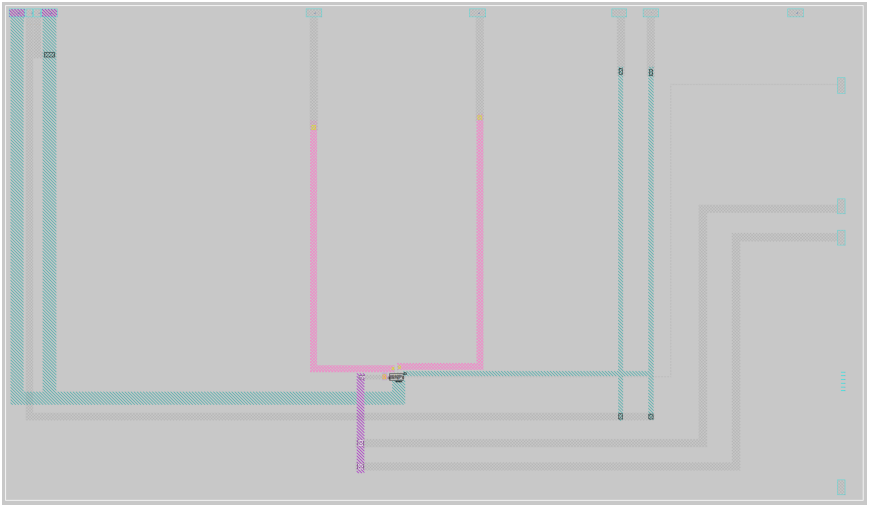
1. Connect the clamp high pins to VDD (io_analog[4]) and the clamp low pin to VSS (vssa).
2. To avoid shorts between clamp and power pins, small generic resistors of approximately 2 m ohms are used.

- Refer to the final schematic below:

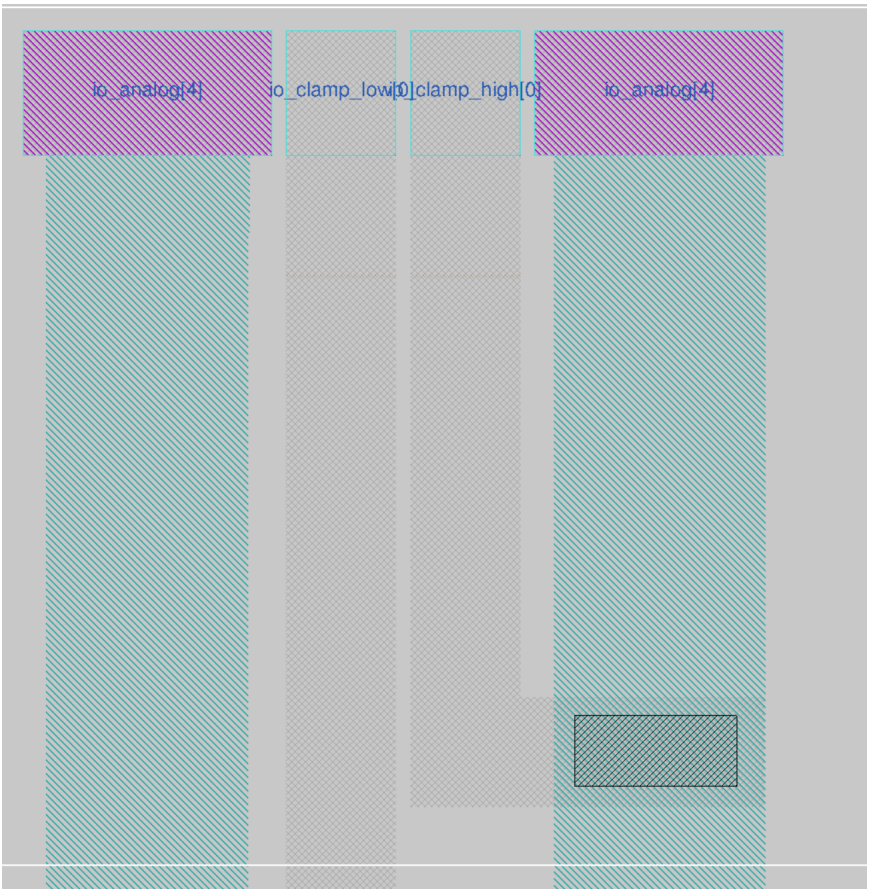




- These resistors contribute to preventing shorts without compromising the circuit's functionality. Ensure that any unused pins remain unconnected.
- The final layout of the opamp inside the wrapper is shown below:



- The connection for io_analog[4] is shown below:



- The `clamp_high` pin is connected to supply and `clamp_low` pin is connected to ground.

- Note the usage of two small resistors in the metal3 layer for the `clamp_high` and `clamp_low` connections to prevent shorts.

Once the final schematic and layout are prepared, follow the previous steps to generate SPICE netlists for LVS checks and final submission.

Antenna Check

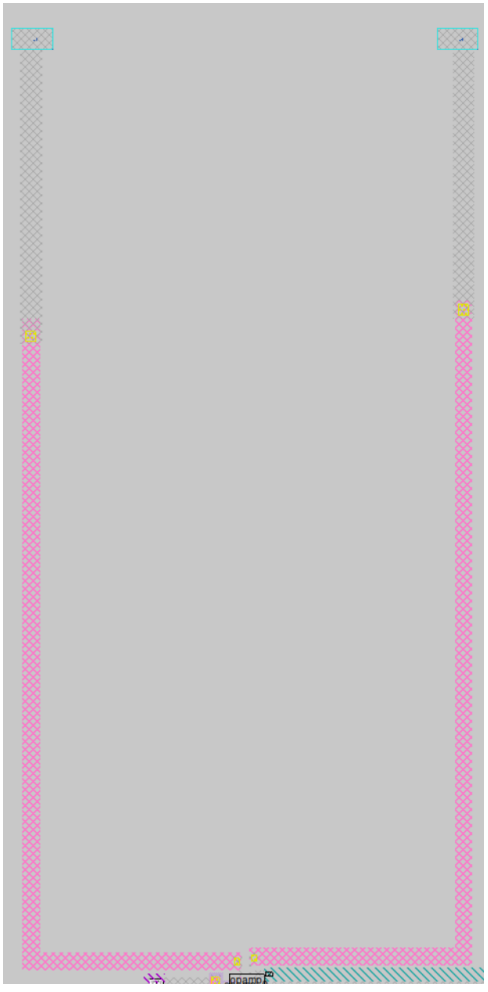
- Please refer to the details regarding the Antenna Issue in ICs here.
- Once the layout is integrated inside the wrapper, an antenna check needs to be run in Magic to identify any long wires connected to the gates of transistors causing antenna errors.

Please execute the following commands in Magic:

```
antennacheck debug  # Explains the error
antennacheck
```

The `antennacheck` command will highlight the nets with antenna errors.

- In the current layout, there were two antenna errors in signals `INP` and `INN` (input signals for the opamp). These signals are connected to the io_analog[2] and io_analog[3] pins. Initially, the connection was established using `metal 3` wires. However, the length of these wires exceeded the limit set by the design rule. To address this issue, jumpers to `metal 2` are utilized as shown below.



- **Currently, the precheck run on the Efabless platform does not include the checking of antenna errors for analog designs. Therefore, it's crucial to ensure that the design is free of antenna errors manually, as passing the precheck does not guarantee the absence of antenna errors.**

Updating Folders for Design Tapeout

GDS →

- Paste the final layout (cell with the wrapper) here. Please make sure that the name of the top cell is `user_analog_project_wrapper`.

LVS →

- Edit the `lvs_config.json` file. For analog designs, it is enough to use the spice netlist generated from complete schematic of the wrapper, and the final gds for the lvs.
- Paste the spice netlist generated from the schematic in the lvs folder itself.
- *.json files are very strict on syntax so make sure everything is correct otherwise the LVS run will have issuers and it will be very hard to debug the issue.

MAG →

- This folder should have the final layout `user_analog_project_wrapper.mag` generated by Magic.

NETGEN →

- This folder should have the SPICE netlist `user_analog_project_wrapper.spice` generated from the layout.
- `Make sure that the netlist is consistent with the final layout otherwise precheck will fail at consistency check without any explanation.`

OPENLANE →

- It can be kept empty since openlane is not used for analog design.

VERILOG →

In the RTL folder inside VERILOG, there is a file `user_defines.v.` This file is used to define the mode of all 38 GPIO pins.

> The configuration for each GPIO is a set of 13 bits. It exists in two places: Inside the SoC (housekeeping), and next to each GPIO. The values in housekeeping are accessible through the memory map and so they can be defined in C code and then loaded into the locations next to

> each GPIO. That's the normal way to configure the GPIOs, because it's software-definable.

> However, there are cases in which a user may want the GPIO pins to be operational for the user project immediately on power-up, or they may want the user project to run without the SoC running at all, in which case the state of the GPIOs on power-up is critical. The `"user_defines.v"` specifies the default state of the configuration locally at the GPIO itself. It is implemented as a small via-programmed ROM close to the GPIO. The via programming is done by python script from the Makefile.

Pin Configuration :

Each pin can be configured in one of the 13 mode provided.

```
define GPIO_MODE_MGMT_STD_INPUT_NOPULL    13'h0403
define GPIO_MODE_MGMT_STD_INPUT_PULLDOWN  13'h0c01
define GPIO_MODE_MGMT_STD_INPUT_PULLUP    13'h0801
define GPIO_MODE_MGMT_STD_OUTPUT          13'h1809
define GPIO_MODE_MGMT_STD_BIDIRECTIONAL   13'h1801
define GPIO_MODE_MGMT_STD_ANALOG          13'h000b
define GPIO_MODE_USER_STD_INPUT_NOPULL    13'h0402
define GPIO_MODE_USER_STD_INPUT_PULLDOWN  13'h0c00
define GPIO_MODE_USER_STD_INPUT_PULLUP    13'h0800
define GPIO_MODE_USER_STD_OUTPUT          13'h1808
define GPIO_MODE_USER_STD_BIDIRECTIONAL   13'h1800
define GPIO_MODE_USER_STD_OUT_MONITORED   13'h1802
define GPIO_MODE_USER_STD_ANALOG          13'h000a
```

- For analog designs with analog input and output, two useful configurations are `GPIO_MODE_MGMT_STD_ANALOG` and `GPIO_MODE_USER_STD_ANALOG`.

  In these modes both the input and output buffers are kept off and the pad/pin can be used for analog signals.

- In `GPIO_MODE_USER_STD_ANALOG`, tie all `io_oeb` pins to 1.8 V power and all `io_out` pins to either power or ground to remove the leakage from the buffers. It is not necessary in case of `GPIO_MODE_MGMT_STD_ANALOG` but is suggested to do anyway for precaution.

- The GPIO are good for analog voltages up to the value of VDDIO, which is nominally 3.3V. (I am not sure about 5 V case yet.)

1					DIGITAL_MODE_MASK		TRIPPOINT_SEL	SLOW_SLEW_MODE	ANALOG_POLARITY	ANALOG_SELECT	ANALOG_ENABLE	MODE_SELECT	INPUT_DISABLE	HOLD_OVERRIDE	OUTPUT_DISABLE	MGMT_ENABLE
2	`define	GPIO_MODE_MGMT_STD_INPUT_NOPULL	13'h0403	0010000000011	0	0	1	0	0	0	0	0	0	0	1	1
3	`define	GPIO_MODE_MGMT_STD_INPUT_PULLDOWN	13'h0c01	0110000000001	0	1	1	0	0	0	0	0	0	0	0	1
4	`define	GPIO_MODE_MGMT_STD_INPUT_PULLUP	13'h0801	0100000000001	0	1	0	0	0	0	0	0	0	0	0	1
5	`define	GPIO_MODE_MGMT_STD_OUTPUT	13'h1809	1100000001001	1	1	0	0	0	0	0	0	1	0	0	1
6	`define	GPIO_MODE_MGMT_STD_BIDIRECTIONAL	13'h1801	1100000000001	1	1	0	0	0	0	0	0	0	0	0	1
7	`define	GPIO_MODE_MGMT_STD_ANALOG	13'h000b	0000000001011	0	0	0	0	0	0	0	0	1	0	1	1
8																
9	`define	GPIO_MODE_USER_STD_INPUT_NOPULL	13'h0402	0010000000010	0	0	1	0	0	0	0	0	0	0	1	0
10	`define	GPIO_MODE_USER_STD_INPUT_PULLDOWN	13'h0c00	0110000000000	0	1	1	0	0	0	0	0	0	0	0	0
11	`define	GPIO_MODE_USER_STD_INPUT_PULLUP	13'h0800	0100000000000	0	1	0	0	0	0	0	0	0	0	0	0
12	`define	GPIO_MODE_USER_STD_OUTPUT	13'h1808	1100000001000	1	1	0	0	0	0	0	0	1	0	0	0
13	`define	GPIO_MODE_USER_STD_BIDIRECTIONAL	13'h1800	1100000000000	1	1	0	0	0	0	0	0	0	0	0	0
14	`define	GPIO_MODE_USER_STD_OUT_MONITORED	13'h1802	1100000000010	1	1	0	0	0	0	0	0	0	0	1	0
15	`define	GPIO_MODE_USER_STD_ANALOG	13'h000a	0000000001010	0	0	0	0	0	0	0	0	1	0	1	0

The figure above shows the 13 bit configuration of available 13 modes.

- Edit the `user_defines.v` according to your layout and configuration of each pin.

In the current work, no GPIO pins are used, so all the pins are kept at `GPIO_MODE_MGMT_STD_ANALOG`.

XSCHEM →

- This folder should have schematic and symbol of `top cell` (opamp) and final `user_analog_project_wrapper`.

Update `README` file as well.

Useful Links

1. **Open Source Silicon Forum**:
   - Description: The go-to platform for addressing issues related to Caravel/Caravan.
2. **Caravel Documentation**:
   - Description: Comprehensive documentation for Caravel.

Precheck & Tapeout

Precheck

The final thing that you will have to do is to pass all of the prechecks set out by efabless. The design needs to pass following checks before submission :

License check:

- It will check your directory and make sure that there are no prohibited license files within it.

Make file:

- It will check your make file in your directory and ensure that it is valid.

Default:

- It will check to see if you have edited the 'README.md' file to ensure that you do not maintain the default 'README.md' file that comes with the caravan repository.

Documentation:

- It will check to see if there is appropriate documentation in your directory.

Consistency:

- Runs a series of checks on the user netlist (user_analog_project_wrapper), and the top netlist (caravan) to make sure that both conform to the constraints put by the golden wrapper.
    - Both Netlists share the following checks:
        - Modeling check: check netlist is structural and doesn't contain behavioral constructs
        - Complexity check: check netlist isn't empty and contains at least eight instances
    - Remaining Top Netlist checks:
        - Sub-module hooks: check the user wrapper submodule port connections match the golden wrapper ports (All the pjns match to the empty wrapper).
        - Power check: check all submodules in the netlist are connected to power.
    - Remaining User Netlist checks:
        - Ports check: check netlist port names match the golden wrapper ports.
        - Layout check: check netlist matches the provided user wrapper layout in terms of the number of instances, and the instance names.

GPIO Defines:

- A verilog directives check that validates the project's `'verilog/rtl/user_defines.v'` netlist.

XOR:

- No modification in the `user_analog_project_wrapper` (versus default `user_project_wrapper.gds` in caravel_user_project_analog for analog projects outside the user defined area lower left corner (0,0) and upper right corner (2920, 3520).

Magic DRC:

- Checks for any violations in the design rule check

Klayout FEOL:

- Checks for DRC violations

Klayout BEOL:

- Checks for DRC violations

Klayout Offgrid:

- The user_project_wrapper.gds does not contain any shapes that have offgrid violations(rules x.1b, x.3a, x.2, x.2c)

Klayout Metal Minimum Clear Area Density:

- The user_project_wrapper.gds has metal density (for each of the 5 metal layers) that is the lower than the maximum metal density specified by the li1.pd.ld, m1.pd.ld, m2.pd.ld, m3.pd.ld, m4.pd.ld, m5.pd.ld rules

LVS:

- Runs hierarchy check, soft check, lvs check, ERC check on the user project. For more information click here.
- **LVS is disabled on the platform by default, it is only enabled when running locally**

OEB:

- Runs oeb check, to make sure that user connected all needed oeb signals. For more information click here.

K**layout Pin Label Purposes Overlapping Drawing:**

- Checks for DRC violations

Klayout ZeroArea:

- Checks for DRC violations

Before design submission, it is a good idea to run the precheck locally first to fix all the issues.

To run precheck locally:

Go inside your user_analog_project_wrapper folder and use commands:

```
export CARAVEL_ROOT=/home/zerotoasic/asic_tools/caravel_user_project/caravel/
make run-precheck
```

After running the prechecks all the logs and reports are stored inside the `~/user_analog_project_wrapper/precheck_results` folder.

Issues and Solution

Precheck was not running LVS and OEB checks.

Reason: The `mpw_precheck` was an old version.

- To update it, delete the `mpw_precheck` folder from `/home/zerotoasic/` and run the following command from the `user_analog_project_wrapper` folder:

```
make run-precheck
```

- This will create an updated `mpw_precheck` folder containing all the necessary checks using the Make file in `user_analog_project_wrapper`.
- The updated mpw_precheck at `/home/zerotoasic/` has the LVS and OEB checks.

After updating the folders, the LVS and OEB checks were failing.

Reason: For the LVS check, a netlist is extracted from the final GDS and is located at `precheck_results/<tag>/tmp/ext/user_analog_project_wrapper.gds.spice`. This netlist did not have the unconnected pins of the wrapper, causing a mismatch with the netlist generated from the schematic (located at `~/user_analog_project_wrapper/lvs/user_analog_project_wrapper/user_analog_project_wrapper.spice`), leading to LVS failure.

Solution: PDK needed to be updated to the latest version. To update:

Update volare:

- `Volare` is a version manager for builds of Google open-source PDKs. Update it using the command:

```
python3 -m pip install --upgrade --no-cache-dir volare
```

For information regarding `volare` please go here.

Update PDK:

- Check the latest versions available for PDK:

```
volare ls-remote
```

- Download the latest version of PDK:

```
volare enable --pdk sky130 <version tag>
```

- For example:

```
volare enable --pdk sky130 e0f692f46654d6c7c99fc70a0c94a080dab53571
```

- **Update magic**

Once the PDK was updated to the latest version and enabled, the Magic was not compatible with PDK and needed to be updated.

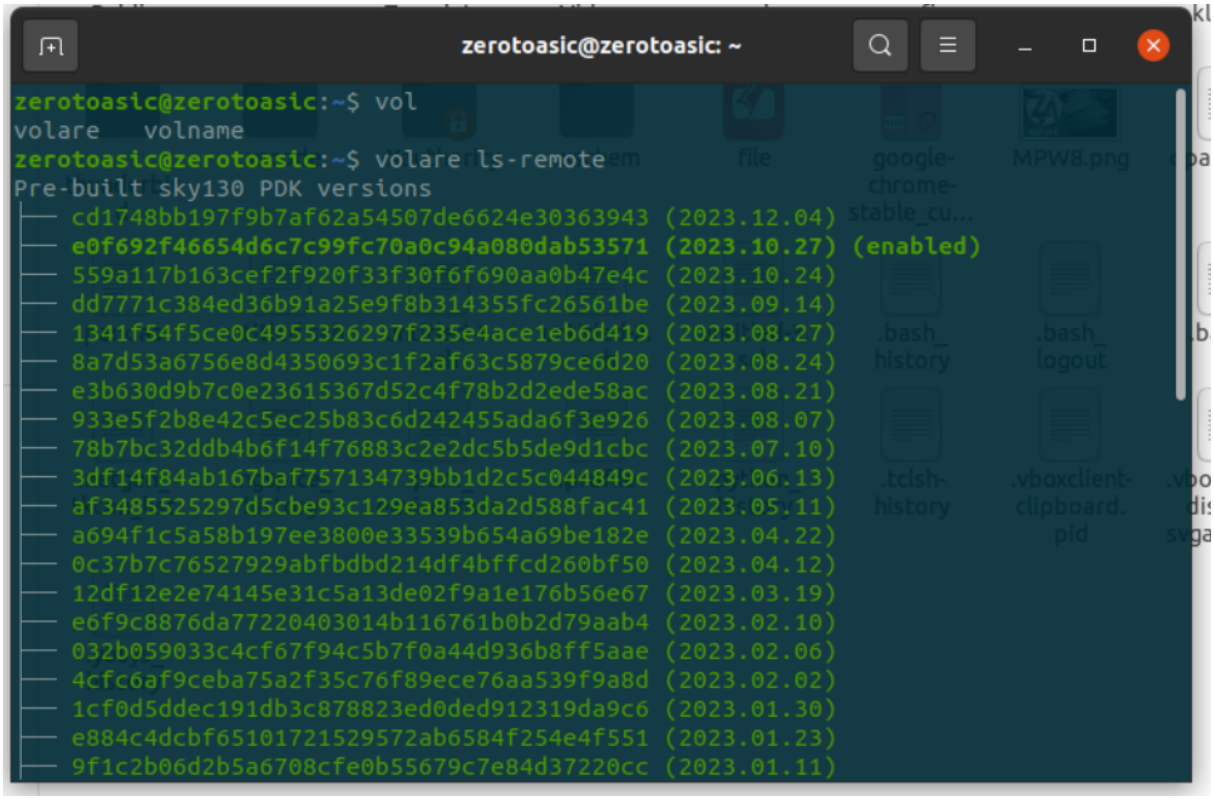
- Pull the latest version from GitHub and install it.

`Finally, the precheck run passed locally.`

Note → Please always make sure that you are using the latest version of the PDK. Use the command :

```
volare ls-remote
```

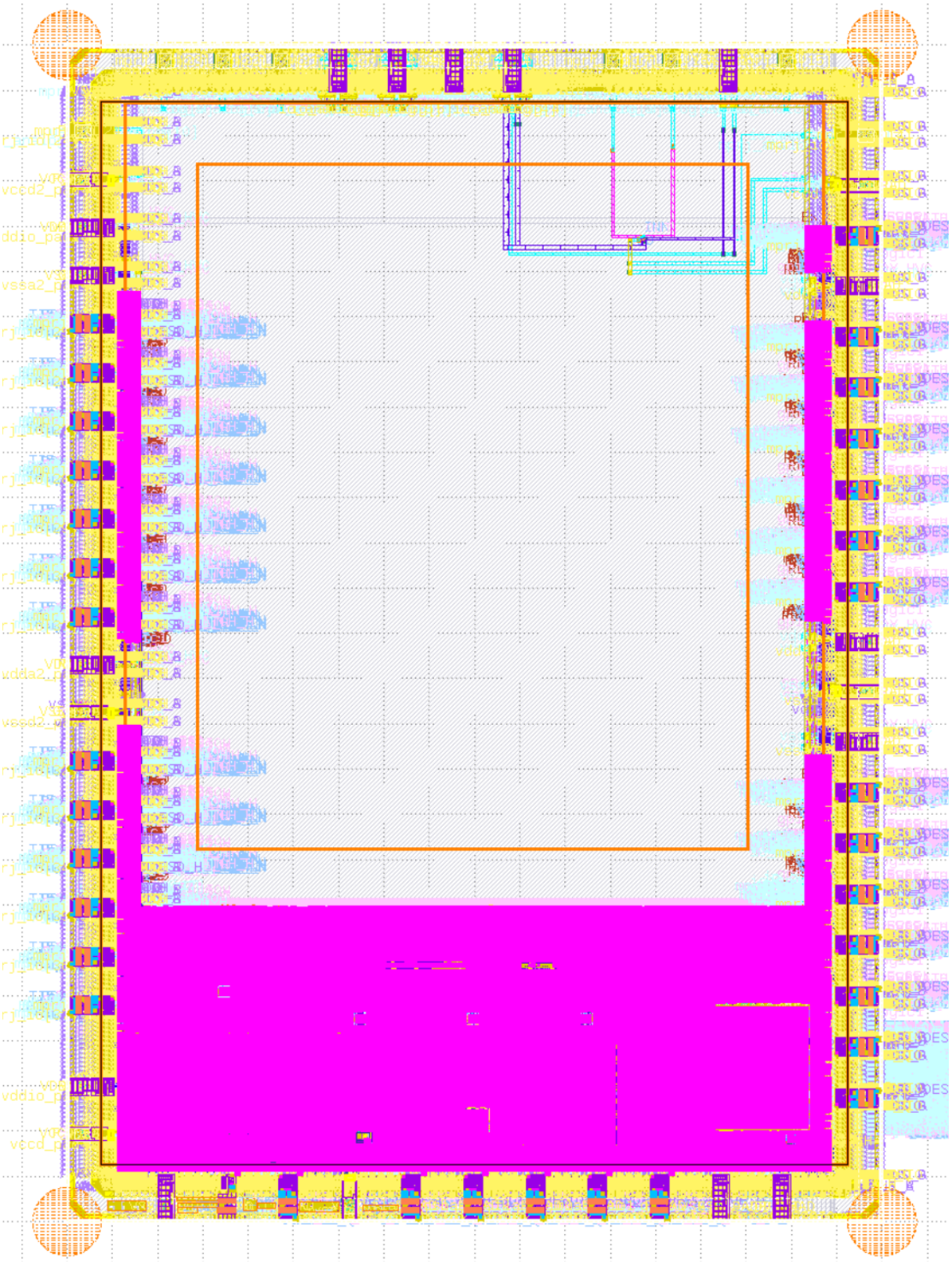
This will show available versions, allowing you to update if needed.



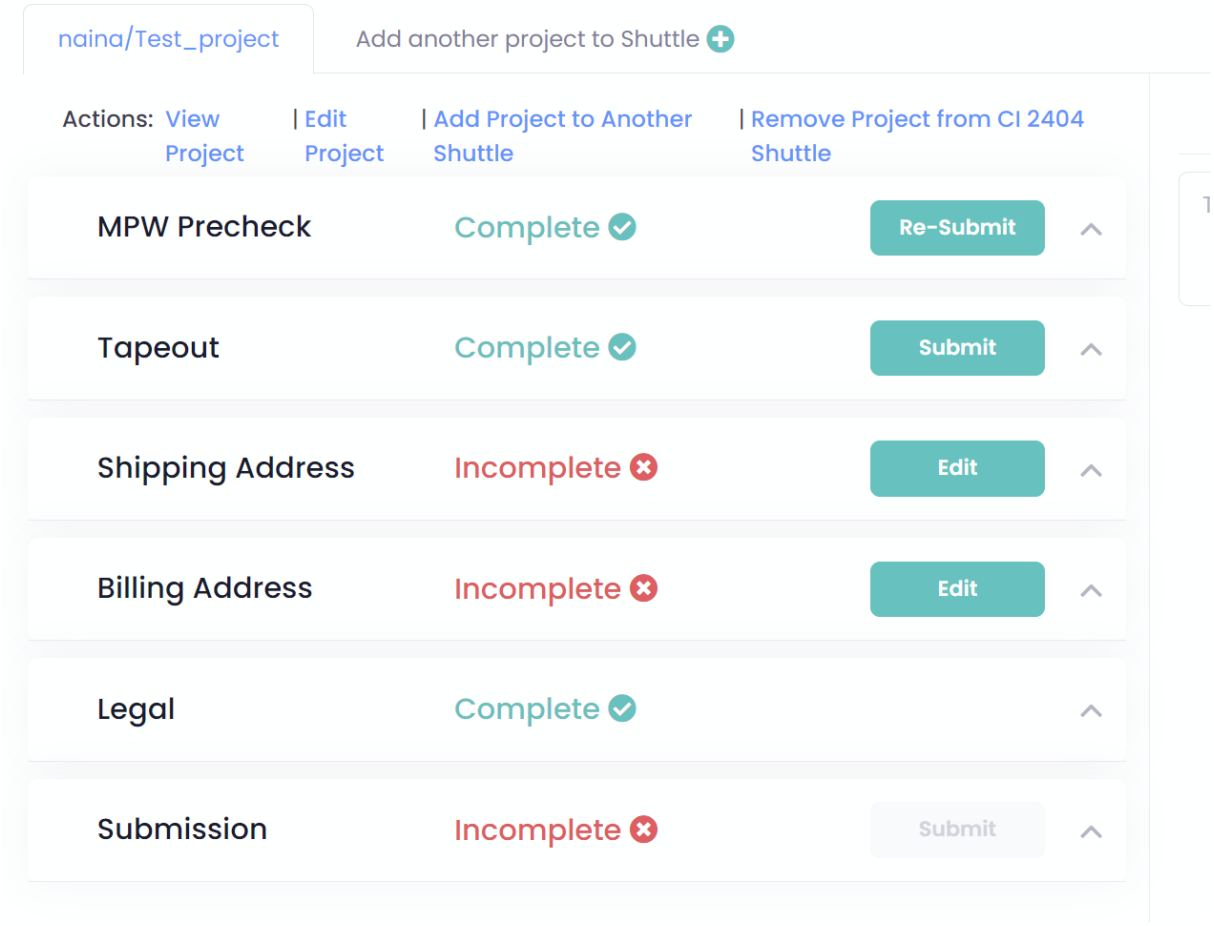
For example, the figure above shows a more recent version of PDK available than the one currently in use.

Submitting Design

- Create an account on the efabless platform, follow the provided tutorial here, push your design, and run precheck locally first to fix any issues.
- For tapeout submission, it takes up to 2 hours. Request logs and the folder after the job is completed. In the `gds` folder, you will find the GDS of the full layout with the caravan block.



Complete Caravan Block



`Checks on Efabless Platform`

`The latest files are available here:`

git@github.com:SinghalNaina/Two_stage_opamp.git

Credits

Written By : Naina Singhal

Reviewed By: Robin Alden

(Members of New Zealand Semiconductor Association)