What precheck checks

This page answers: *what does each row of precheck's table check, and what does a green table still not tell you?*

Precheck is a Python program, `precheck/precheck.py` in Tiny Tapeout's support tools, and the `precheck` job runs it on the GDS from the `tt_submission` artifact. Tiny Tapeout's own guide says of it that "All the checks need to be green to submit to the chip". The details on this page come from `precheck.py` at commit `9d13ff3`, which is the commit the runner checked out in November 2026, and from runs of it on this course's layouts and on public projects' layouts. You'll see most of these rows fail in Lab 02, and find two mistakes in Lab 03 that none of them catch.

How it runs

Precheck is given one file, `tt_submission/<top_module>.gds`. It finds `info.yaml` by looking in the folders above that file, reads `top_module`, `tiles`, `analog_pins`, `uses_vapwr` and the pinout from it, and picks the DEF template to compare your layout with: `tt_analog_1x2.def` for a 1x2 analog project, or `tt_analog_1x2_3v3.def` if `uses_vapwr` is true. (With `analog_pins: 0` it would pick a digital template instead, so an analog layout has to use at least one analog pin.) The LEF and the Verilog are found next to the GDS, under the same name.

Then it runs every check for `sky130A`, in a fixed order, and doesn't stop at the first failure. At the end it writes the table you see on the run's summary page, with a ✅ or a ❌ and a message for each row, and the job fails if any row failed. The reports go into a folder that the job uploads as the `precheck_reports` artifact.

The fifteen checks

#	Row	What it checks	Fails like this
1	Magic DRC	Magic's full SKY130 design-rule deck, `drc(full)`, on the top cell. Writes `magic_drc.txt`, each broken rule with its boxes	`Magic DRC failed`; the report names the rule, such as `Metal2 spacing < 0.14um (met2.2)`
2	KLayout FEOL	KLayout's SKY130 rule deck for the layers below metal: wells, diffusion, poly, contacts	`Klayout feol failed with 1740 DRC violations` (a public project, Nov 2026)
3	KLayout BEOL	The same deck for the metal and via layers	`Klayout beol failed with 1 DRC violations`
4	KLayout offgrid	Shapes whose corners aren't on the manufacturing grid	
5	KLayout pin label overlapping drawing	Every pin label sits on drawn metal of its own layer	`Klayout pin_label_purposes_overlappi failed`
6	KLayout zero area	Shapes of zero area	
7	KLayout Checks	Three things: the GDS's top cell is named `top_module`; no forbidden layer (metal 5) is	`Top macro name mismatch: expected …, got …`, or `Forbidden`

#	Row	What it checks	Fails like this
		used; the boundary layer exists	`layer met5.drawing found in …`
8	Pin check	Every pin of the DEF template is in the LEF and the GDS, on the right layer, with the right size and place; the die is the template's size; power ports are on metal 4, at least 1.2 µm wide, and reach within 10 µm of the top and bottom edges	`Some ports are missing or have wrong dimensions, see 53 LEF errors above`, `Wrong cell at GDS top-level`, or `Inconsistent die area between LEF and template DEF`
9	Boundary check	Nothing outside the tile, and a boundary rectangle that covers exactly the tile	`Shapes outside project area`, `Boundary layer doesn't cover project area`
10	Power pin check	`VGND` and `VDPWR` are in the Verilog and the LEF, with `USE GROUND` and `USE POWER`; `VAPWR` is there if and only if `uses_vapwr` is true	`Verilog doesn't contain VAPWR, VGND does not have a corresponding 'USE GROUND ;'`
11	Layer check	Every layer in the GDS is one Tiny Tapeout allows	`Invalid layers in GDS: …`
12	Cell name check	No cell name contains `#` or `/`	
13	urpm/nwell check	A KLayout rule about the spacing of a resistor implant from n-well	`Klayout nwell_urpm failed`
14	Analog pin check	Each `ua` pin touches metal if and only if it's below `analog_pins`, and has a pinout description if and only if it touches metal	four messages, below
15	Verilog syntax check	`<top_module>.v`, your `src/project.v`, parses in Yosys	`Verilog syntax check failed`

Rows 2 to 6 and 13 are KLayout scripts; row 1 is Magic. Precheck builds its own Magic and KLayout for this, at versions fixed in its Nix file, and installs its own copy of the PDK, which is why the course installs the same versions for you (Your tools and the runner's tools).

The analog pin check has four messages, one for each way the metal, `analog_pins` and the pinout can disagree. Each says which pin and suggests two fixes:

- `Analog pin` ua[2] `is connected to some metal but` analog_pins `is set to 2 in` info.yaml`.`
- `Analog pin` ua[0] `is connected to some metal but the description of` ua[0] `in the pinout section of` info.yaml `is empty.`
- `Analog pin` ua[1] `is not connected to any adjacent metal but` analog_pins `is set to 2 in` info.yaml`.`
- `… is not connected to any adjacent metal but the description of` ua[n] `in the pinout section of` info.yaml `is non-empty.`

"Connected" here means metal 4 reaching into a band from 0.1 to 0.5 µm around the pin's rectangle, or a via 3 on the pin itself. Precheck doesn't follow the metal any further, so a stub that goes nowhere counts, which is why Lab 01's placeholder passes.

Reading the table and the log

Because precheck runs every check, one mistake can turn several rows red. In Lab 02's breakage 5, one line in `info.yaml`, `uses_vapwr: true`, makes four rows fail, because it changes the template every later check compares against. The table on the run's summary page is the quickest place to start: read every ❌ row, and look for the one that explains the others.

The log isn't always ordered the way the table is. Some checks print their errors as they go and some only fail at the end, so the first `ERROR:` line in the log can be a consequence of another check's failure. In Lab 02's breakage 3, a renamed `top_module` with an unchanged GDS, the log's first 53 error lines are the pin check's, one per pin:

```
ERROR: Pin clk not found in /foss/designs/projects/ttsky-ic102-a/tt_submission/
tt_um_yourname_ic102_v2.lef
```

while the cause is in row 7 of the table, `Top macro name mismatch: expected tt_um_yourname_ic102_v2, got tt_um_yourname_ic102`.

A ✅ needs reading as well. In the same breakage, Magic DRC passed. It had been asked to check a cell called `tt_um_yourname_ic102_v2`, which the GDS doesn't contain, and its log says what it did instead:

```
Cell tt_um_yourname_ic102_v2 couldn't be read
No such file or directory
Creating new cell
...
No errors found.
```

It made an empty cell, checked that, and found nothing wrong, so the table shows a ✅ for Magic DRC, and only the log tells you that the check ran on an empty cell. You'll write a script in Lab 05 that has to tell a pass like this one from a real pass.

What precheck leaves to you

Precheck checks that a layout is legal and fits the tile. It doesn't check that it's the circuit you meant, and several things a reviewer cares about are outside every one of its rows:

- **Whether the layout is your circuit.** There's no LVS (layout versus schematic) in precheck, so nothing compares your layout with a netlist. In Lab 03 you'll make a short between an inverter's input and output: Magic DRC finds 0 errors, all 15 rows are ✅, and only `make lvs` objects. The course runs LVS with Magic's extractor and netgen, on your machine, against a reference netlist kept in your repository.

- **The tie-offs.** Tiny Tapeout's specs say "Connect any unused uo_out, uio_out and uio_oe pins to GND". Nothing in precheck looks; in Lab 03, with the 24 tie-offs removed, precheck still passes and LVS lists the 24 pins.
- **What the pinout says.** Precheck checks only that a used analog pin has a description, never what it says, and doesn't read the digital pins' descriptions at all. A pinout that calls the input the output passes.
- **Whether the circuit works.** Precheck simulates nothing. For an analog design, simulation of the extracted layout, with its parasitics, is how you find out, and it happens in the workbench.
- **Whether the GDS is the one your sources build.** Precheck checks the file you committed. If you changed `wires.tcl` and forgot to run `make`, it checks the old layout, and passes it.

Between them, the `docs` job and precheck check that your files are complete and legal. Checking that they describe the circuit you meant is left to you and your reviewer, and Pull requests and review is about how you do that.

Check yourself

1. A layout passes all 15 checks. Name two things that could still be wrong with it.

   Answer: A1, in Answers at the end.

2. Your table has ❌ on "KLayout Checks" and "Pin check", and the first error in the log is `ERROR: Pin ua[0] not found in …lef`. Where do you start?

   Answer: A2, in Answers at the end.

Answers

A1. Check yourself

Any two of: a short or an open that LVS would find; missing tie-offs; a pinout that describes the pins wrongly; a circuit that doesn't do what it should; a GDS that doesn't match the committed sources.

A2. Check yourself

With row 7, "KLayout Checks", because it comes first in the table and its message names the cause. If it says `Top macro name mismatch`, the pin errors follow from it: the pin check looks for pins on a macro with the expected name and can't find any.