RTL and gate-level tests

This page answers: *your cocotb test runs twice on every push, once in `test` and once in `gl_test`. What's different about the second run, and why can a design pass one and fail the other?*

Your Verilog describes what the design should do. The `gds` job turns it into a layout, and on the way it produces a **gate-level netlist**: a Verilog file that describes the same design as instances of SkyWater's standard cells (flip-flops, NAND gates, buffers and the like), wired together. The `test` job simulates your Verilog as you wrote it, which is called **RTL simulation**. The `gl_test` job simulates the netlist, which is **gate-level simulation**. The test is the same file both times. You'll see both runs on your own design in Lab 03.

One test, two designs

The template's `test/Makefile` decides what to compile. Without any setting it compiles the files in `PROJECT_SOURCES` from `src/`, plus `test/tb.v`. With `GATES=yes`, which is how `gl_test` calls it, it compiles three other files instead of your sources: the netlist, which `gl_test` copies to `test/gate_level_netlist.v`, and two files of models for the sky130 cells from the PDK. It also defines a few macros for the compile:

Macro	What it does
`GL_TEST`	Read by the template's `tb.v`, which then connects two power wires, `VPWR` (1) and `VGND` (0), to the design
`USE_POWER_PINS`	Gives every cell model its power pins, so the netlist's power connections have somewhere to go
`FUNCTIONAL`	Uses the cells' logic-only models, without the detailed timing of the real cells
`UNIT_DELAY=#1`	Gives every cell a delay of 1 ns, so a change ripples through the logic in steps rather than all at once

The testbench in `tb.v` stays the same, and so does `test.py`, so one test has to work on both designs, and that puts two limits on how you write it.

Use only the pins. In RTL simulation, cocotb can reach inside your design and read, say, `dut.user_project.count`. In the netlist, synthesis has flattened the design into cells and renamed or removed most internal signals, so a test that reads one may find nothing there at gate level. A test that drives and reads only `ui_in`, `uo_out`, `uio_in`, `uio_out`, `uio_oe`, `clk` and `rst_n` reads the same pins in both.

Sample away from the clock edge. With unit delays, a flip-flop's output changes about 1 ns after the rising edge, and the logic after it settles a few nanoseconds later. A test that changes inputs and reads outputs on the falling edge, half a clock period away, gives the same answers in both simulations. Lab 03's starter test does that, and so does the review sandbox's in Lab 04.

Where they disagree

Tiny Tapeout's documentation warns that on a real chip "flops will have a random initial value at power on", and therefore that "Initial statements are not allowed, so you must use an explicit reset." An `initial` statement in your Verilog sets a register when the simulation starts, so in RTL simulation the register starts at the value you gave it. Synthesis can't build that into a flip-flop, though, so in the netlist the register starts with no defined value, which a simulator shows as `X`, and anything computed from an `X` stays `X`.

We tried this on Lab 03's counter. With its reset replaced by `initial count = 8'd0;`, all three of its tests passed in RTL simulation with the test job's tools. On a netlist that Yosys made from the same file with the sky130 cells (not LibreLane's netlist, but built from the same cell library), Icarus 13.0 printed:

```
AssertionError: after reset uo_out is XXXXXXXX, expected 0
AssertionError: after 10 cycles with EN high uo_out is XXXXXXXX, expected 10
```

The third test, of `LOAD`, still passed, because loading a value sets every bit at once. A failure that shows up in some tests and not others, only at gate level, is often this.

The two simulations also treat an unknown value differently, which is harder to spot. In RTL simulation, an `if` whose condition is `X` takes the `else` branch, so a design can behave as if an unknown signal were 0 without any sign of it in the results. Gates don't do that; an `X` on a gate's input usually gives `X` on its output. RTL simulation is the optimistic one, so a design that only works because of that optimism passes `test` and fails `gl_test`.

`gl_test` doesn't check timing, though. Every cell has the same made-up delay, and nothing compares the delays with the clock period. LibreLane's timing analysis in the `gds` job does that, against the clock period in `src/config.json`.

The tinyCGRA history

`ttsky-verilog-tinyCGRA` shows how this goes wrong in a team. In May 2026, two of its `gds` runs built the layout, passed precheck and then failed at `gl_test`. The logs are gone, but the commit messages from those days survive: "commented config done (test)", "comment assert input in load_input()", "comment all user_project asserts", "commented asserts, no functional testing". The gate-level failures went away because the test stopped checking anything. The pull request that was finally merged includes a commit with a reason, "removed config_done assert (unreliable in GLS)", which at least says what was removed and why. When `gl_test` fails and `test` passes, ask what's different about the netlist. The answer is usually in the design: a missing reset, a signal read before it's defined, or a test that reaches inside.

Running each one yourself

`make test` in the course's `labs/common/tt-tools` runs the RTL test with the test job's tools, as you'll do in Lab 03. Running the gate-level test needs the netlist, which only the `gds` job makes: the tools package can run `gl_test`'s steps on a netlist you download from a run's `tt_submission` artifact (`make gl-test`, in the package's README), with Icarus 13.0 and the sky130 cells at the PDK commit the runner used. It's optional, and most of the time reading `gl_test`'s log is quicker.

Check yourself

1. A test reads `dut.user_project.state` to check a state machine. Which job do you expect it to pass in, and which to fail in?

Answer: A1, in Answers at the end.

2. Why does `tb.v` connect `VPWR` and `VGND` only when `GL_TEST` is defined?

Answer: A2, in Answers at the end.

3. A design passes both `test` and `gl_test`. Does that show it will run at the clock rate in `info.yaml`?

Answer: A3, in Answers at the end.

Next: Your tools and the runner's tools.

Answers

A1. Check yourself

It can pass in `test`, where the RTL has a signal called `state`, and fail in `gl_test`, where synthesis has turned the state register into cells whose names it chose. Check the state through the pins it drives instead.

A2. Check yourself

Your RTL module has no power pins, because power isn't part of the logic you describe. The netlist does, because every cell needs power, and LibreLane adds `VPWR` and `VGND` ports to the top module. The testbench has to connect them in one case and mustn't in the other.

A3. Check yourself

No. Neither simulation models the real delays. The `gds` job's timing analysis, against `CLOCK_PERIOD` in `src/config.json`, is the check for that.