Your tools and the runner's tools

This page answers: *the workbench has its own Icarus and cocotb, and GitHub's runner installs others. Which versions does each job use, where do they differ, and what does the course do about it?*

What a cloud runner is explains why the runner's tools needn't match yours: every job installs its own, at the versions its workflow asks for, on a fresh machine. This page lists them for the digital template, and says what the course installs so that a test you run in the workbench means the same as the runner's. You install them in Lab 01, Part 1 and use them from Lab 03 on.

The versions

We read these from Tiny Tapeout's action files at the `ttsky26d` tag (commit `395eedb`) and from runner logs of Nov 15 and 16, 2026. "Course" is what `make install` in `labs/common/tt-tools` puts next to the workbench's own tools; "Workbench" is what the image `hpretl/iic-osic-tools:2026.08` has.

Job	Tool	Runner	Course	Workbench
`test`	Python	3.11.18	3.11.16	3.12.3
`test`	cocotb, pytest	2.1.0, 8.4.2	2.1.0, 8.4.2	2.0.1, 9.1.1
`test`	Icarus Verilog	12.0 (Ubuntu's package)	12.0, the same package	14.0
`gl_test`	Icarus Verilog	13.0 (Tiny Tapeout's package)	13.0, the same package	14.0
`gl_test`	sky130 standard cells	PDK commit `8afc8346`	the same, optional	PDK commit `026824c7`
`gds`	LibreLane	3.0.14, in LibreLane's Docker image	not installed	3.1.0.dev3
`precheck`	Magic, KLayout	8.3.568, 0.30.8, PDK commit `0536d02d`	not installed	8.3.681, 0.30.11

The course's Python is 3.11.16 rather than the runner's 3.11.18, because 3.11.16 is the newest 3.11 that the workbench's installer offers. Both are 3.11, which is all the workflow asks for (`python-version: '3.11'`), and cocotb's package is built for 3.11 either way. Three different commits of the sky130 PDK are in play: the one LibreLane used to build your netlist, which `gl_test` uses too, the one precheck pins, and the workbench's. For a digital design only the first matters to you.

What `make test` reproduces

`make test PROJECT=/foss/designs/projects/ttsky-ic102-d` runs the lines of the `test` job's "Run tests" step in your `test/` folder (`make clean`, `make`, and `python -m cocotb_tools.check_results results.xml`), with the runner's Python, cocotb and Icarus 12.0 ahead of the workbench's on the `PATH`. It prints the same two lines the runner's log starts the simulation with. On the runner, in `hart-1`'s test run of Nov 15:

```
Initialized cocotb v2.1.0 from /opt/hostedtoolcache/Python/3.11.18/x64/lib/python3.11/site-packages/cocotb
Running on Icarus Verilog version 12.0 (stable)
```

and with `make test`, the second line is identical and the first names the course's folder instead. The simulated times in the summary table are the same in both: the template's test reports `100000.00` ns on the runner and with `make test`. The real-time column measures whichever computer ran it.

With cocotb 2.x, `make` fails by itself when a test fails, because cocotb runs `check_results` at the end of every simulation (in the `results.xml` recipe of its `Makefile.icarus`). The workflow's comment that `make` "will return success even if the tests fail" dates from cocotb 1.x. You'll see cocotb's `Error 1` and the course's `Error 2` in Lab 03. In Lab 05 your bisect script has to work around the fact that `make` fails the same way for a failing test and for a design that doesn't compile.

Where Icarus 12.0 differs

On the designs we tried, the three Icarus versions agreed whenever all three accepted the code. The template's test and the test of `sentry` (at commit `544696e`) passed with the same simulated times under 12.0, 13.0 and 14.0, at RTL and on `sentry`'s gate-level netlist; so did Lab 03′s counter.

They differ in what they accept. Icarus 12.0 rejects some SystemVerilog that 13.0 and 14.0 compile, so code you write and try in the workbench can stop the runner's `test` job at the compile step. When we changed the template's adder to compute its sum in a `for` loop that ends with `break`, which is legal SystemVerilog, Icarus 12.0 stopped with:

```
project.v:27: sorry: break statements not supported.
```

and `make` ended with `make: *** [Makefile:42: test] Error 2`. Icarus 13.0 and 14.0 printed `project.v:27: internal error: NetProc::nex_input not implemented`, carried on, and the test passed. Assigning a whole array at once (`q = '{8'd1, 8'd2, 8'd3, 8'd4};`) fails on 12.0 only, with `sorry: Assignment to an entire array or to an array slice is not yet supported.` Icarus says `sorry` when it recognises a construct and doesn't implement it, so a `sorry` from `make test` is one the runner's `test` job will print too.

This is the main reason `make test` uses 12.0 rather than the workbench's 14.0: the runner's `test` job will compile your code with 12.0, and only 12.0 can tell you in advance whether that compile will succeed. The course's `make test-icarus13` and `make test-workbench` run the same test with the other two versions, when you want to compare.

Your `requirements.txt`

Both test jobs install their Python packages from your repository's `test/requirements.txt`, so the cocotb on the runner is whatever that file says, not what the template said. If you change the file, `make test` warns you before it runs. For `sentry`, which pins cocotb 2.0.1, it printed:

```
warning: the runner installs your test/requirements.txt, and it differs from this venv:
  cocotb: your test/requirements.txt pins 2.0.1, this venv has 2.1.0
  Results here may not match the runner's. See README.md, 'Your own requirements.txt'.
```

`sentry`'s `gl_test` log shows the runner following the file: `Initialized cocotb v2.0.1`, on Icarus 13.0. Leave the template's pins alone unless you have a reason, and if you do change them, the tools package's README says how to rebuild the course's copy to match.

What isn't reproduced

- **The `gds` job.** It runs LibreLane 3.0.14 inside LibreLane's own Docker image, and Docker doesn't run inside the workbench. The workbench's own LibreLane, 3.1.0.dev3, can harden a design, with different versions of Yosys, OpenROAD and Magic, so its numbers won't match the runner's exactly. For the digital labs the runner is the place a layout gets built.

- **The `docs` job.** It checks two text files and builds a PDF; its log names the field or section that's wrong, and it takes under a minute on the runner.
- **`precheck`.** It checks the layout the `gds` job built, which only exists on the runner. The analog version of this course does run precheck locally, because there you draw the layout.
- **The runner's machine.** It's a fresh Ubuntu 24.04 virtual machine with 4 CPUs and 16 GB for a public repository; the workbench is whatever your laptop gives Docker.

When the versions change

Tiny Tapeout updates its template for each shuttle, and can move a tag such as `ttsky26d` to a new commit. Your repository keeps the tag it was made with, but if the tag moves, the runner's tools can change under you. The course records every version it installs in one file, `labs/common/tt-tools/pins.env`, with the run or file each was read from and the steps for reading them again, so that when the runner changes, the course's copy can follow it.

Check yourself

1. Your test passes with `make test-workbench` and fails to compile with `make test`, with a line that says `sorry`. What will the runner's `test` job do?

Answer: A1, in Answers at the end.

2. Which file decides the cocotb version the runner uses for your test?

Answer: A2, in Answers at the end.

3. Why doesn't the course install LibreLane 3.0.14?

Answer: A3, in Answers at the end.

Next: Lab 03, where you run your own test with these tools and on the runner.

Answers

A1. Check yourself

Fail to compile in the same way, because it uses the same Icarus 12.0. Rewrite the construct in a form 12.0 accepts; the workbench's 14.0 accepting it doesn't help on the runner.

A2. Check yourself

Your repository's `test/requirements.txt`. Both `test` and `gl_test` install from it.

A3. Check yourself

The `gds` job runs it inside LibreLane's Docker image, and Docker doesn't run inside the workbench, which is itself a container. The layout is built on the runner instead, and Lab 03 reads the numbers from its artifact.