Getting the tools outside the course image

If you want to run the labs somewhere other than the course image, this page lists the tools they use, with the version of each that the image has, and says where each tool comes from and what works differently once you're outside the image.

The course image is IIC-OSIC-TOOLS 2026.08, and it's what the workbench you set up in IC101 runs. Every number and every output on this course's pages comes from a run in it. The image already has Verilator, cocotb and pyuvm. From Lab 03 on, the labs also need pyvsc and cocotb-coverage, which aren't in the image, so `make venv` installs them into a Python virtual environment that these pages call the course venv. We checked every fact on this page on November 17, 2026, either in the image or against the source it names. The commands for installing the tools on other systems come from each project's own documentation, except Verilator's, which we copied from DD105-A's page, where they're quoted from Verilator's install guide. We haven't run any of these commands, so each block of them is marked *untested by us*.

Tool versions

Tool	In the course image	What the labs use it for
Verilator	5.050 (`Verilator 5.050 2026-07-01 rev v5.050`)	builds the design and any bound assertions into a simulator, through `labs/common/pyuvm.mk` and cocotb's own Verilator makefile
g++	13.3.0	compiles the C++ that Verilator writes
mold	2.30.0	the linker that the image's Verilator was configured to use (`-fuse-ld=mold` in its `verilated.mk`)
ccache	4.9.1	keeps compiled C++ between builds, so that a file that hasn't changed isn't compiled again; the image's `verilated.mk` has `OBJCACHE ?= ccache`
GNU make, bash, GNU coreutils, GNU awk	make 4.3, bash 5.2.21, coreutils 9.4, gawk 5.2.1	run `pyuvm.mk`
Python	3.12.3, as `/usr/bin/python3`, with `/usr/lib/x86_64-linux-gnu/libpython3.12.so.1.0`	the testbenches, which run inside the simulator, and the graders in `labs/common`
cocotb	2.0.1, with find_libpython 0.5.1	starts Python inside the simulator and runs the tests; its Verilator makefile does the build
pyuvm	4.0.1	the UVM classes, in Lab 01 and from Lab 04 on
pytest	9.1.1	if it is installed, cocotb uses it to print more detailed `AssertionError` messages

Tool	In the course image	What the labs use it for
pyvsc	1.1.0, in the course venv	constrained random, from Lab 03 on
cocotb-coverage	2.0, in the course venv	functional coverage, from Lab 03 on
Surfer	0.7.0	the waveform viewer the labs use
`vcd2fst`	from GTKWave, in `/foss/tools/bin`	`make waves` converts the simulator's VCD to FST with it

cocotb 2.0.1 won't build with a Verilator older than 5.036. Line 25 of its Verilator makefile (`cocotb_tools/makefiles/simulators/Makefile.verilator`) sets `VLT_MIN := 5.036`, and with an older version the build stops with an error message that begins `cocotb requires Verilator 5.036 or later`. Every output in the labs comes from 5.050, so any other version, even one that cocotb accepts, may print different numbers.

The graders in `labs/common` use only Python's standard library, but two of their jobs need a Python that has pyuvm or cocotb-coverage installed. The check of the register model imports your model, and pyuvm along with it, and `make cov` merges coverage with cocotb-coverage. The next section says which Python each of them uses.

The course image

The image is Ubuntu 24.04.4 LTS. It is published on Docker Hub as `hpretl/iic-osic-tools:2026.08`, and the course pins it by digest:

```
hpretl/iic-
osic-tools@sha256:3c371645b19c6f6564dc8c7b21e39ad1c1833d274fe5b85639afe1ba9d7987e7
```

On our machine, `docker image inspect` reports the image under that digest as `linux/amd64`, with 61 layers and 11.7 GB unpacked. DD105-A pins the same digest, and its page gives the download as 3.76 GB. The course's CI (`.github/workflows/labs.yml`) runs every lab in this image through `scripts/ci/in-image.sh`, which is a single `docker run` command with the course's limits of 2 CPUs and 3 GB. If you have Docker and the image on your machine, these two commands, run from the top of a clone, build and run Lab 01's pyuvm testbench with the same image and limits that the CI uses:

```
export                                                                IMAGE=hpretl/iic-
osic-tools@sha256:3c371645b19c6f6564dc8c7b21e39ad1c1833d274fe5b85639afe1ba9d7987e7
scripts/ci/in-image.sh make -C labs/lab-01-one-alu-two-frameworks
```

In our run the test passed, and its verdict line read `ok: 0 errors  1.10 s`. `in-image.sh` puts `--skip` in front of your command. That tells the image's entrypoint to run your command instead of starting its desktop, and the entrypoint prints `[INFO] SKIPPING UI STARTUP` when it does so.

How `pyuvm.mk` finds Python

The tests run in a Python interpreter that the simulator embeds. Activating a venv in your shell doesn't change which interpreter that is, because cocotb starts the one named by the environment variable `PYGPI_PYTHON_BIN` (`labs/common/README.md`). So `pyuvm.mk` gives each Python it uses a make variable of its own:

Variable	Default	What uses it
`SYSTEM_PY`	`/usr/bin/python3`	the build: `pyuvm.mk` finds cocotb's makefiles with `$(SYSTEM_PY) -m cocotb_tools.config --makefiles` and passes `PYTHON_BIN=$(SYSTEM_PY)` to them. After the build it writes `LIBPYTHON_LOC`, from `$(SYSTEM_PY) -m cocotb_tools.config --libpython`, into `build/sim.env`
`VENV_DIR`	`/foss/designs/.venvs/dd105-b` if `/foss/designs` is writable, otherwise `.venv` at the top of the clone	where `make venv` makes the course venv
`PYTHON_BIN`	the course venv's `python` if `$(VENV_DIR)/bin/python` exists, otherwise `$(SYSTEM_PY)`	the simulator's Python: every run gets `PYGPI_PYTHON_BIN=$(PYTHON_BIN)`. `make help` prints it
`PYTHON`	`python3`	the graders, and `make regcheck`, which builds the register model with it. The register-model check that `make check` and the other graders run before a verdict uses `PYTHON_BIN` instead, and so does `make cov`

`make venv` runs `$(SYSTEM_PY) -m venv --system-site-packages $(VENV_DIR)` and then installs `labs/common/requirements.txt` into the venv. That file pins `pyvsc==1.1.0` and `cocotb-coverage==2.0`. Installing them needs the network, and if pip fails, `make venv` deletes the venv and asks whether the network is up. In a bare container, with the clone mounted at `/w` and no writable `/foss/designs`, it made `/w/.venv`:

```
========================================================================
 VENV  /w/.venv
   from /w/labs/common/requirements.txt
========================================================================
 imports: vsc, cocotb_coverage, cocotb 2.0.1 pyuvm 4.0.1
 created in 9.9 s, 37M
cocotb                      2.0.1
cocotb-coverage             2.0
pyuvm                       4.0.1
pyvsc                       1.1.0
```

The `--system-site-packages` option lets the venv see a cocotb and a pyuvm that are installed for the system's Python. In the image, though, any venv sees the image's cocotb, pyuvm and pytest, whether it was made with that option or not. The image sets `PYTHONPATH` to a list of folders that includes `/usr/local/lib/python3.12/dist-packages`, where those three packages are, and `pyuvm.mk` passes your `PYTHONPATH` on to the simulator after the lab's own folders. We made a venv without `--system-site-packages` and installed cocotb, pyuvm and the course's two packages into it. `pip` listed the image's pytest 9.1.1 in that venv, although we hadn't installed pytest,

and a run's log said `Initialized cocotb v2.0.1 from /usr/local/lib/python3.12/dist-packages/cocotb`, so the run used the image's copy of cocotb even though the venv had its own.

Everything in one venv

By default `pyuvm.mk` expects cocotb and pyuvm to be installed for the system's Python, `/usr/bin/python3`. If you install them with `pip` into a venv instead, the labs still run, as long as you point `SYSTEM_PY` and `PYTHON` at the venv's `python` and put the venv where `VENV_DIR` expects it, so that `PYTHON_BIN` is the venv's `python` too. We checked this in the course image, from the top of a clone mounted at `/w`, with `PYTHONPATH` unset so that the image's own cocotb and pyuvm were out of reach:

```
unset PYTHONPATH
python3 -m venv .venv
.venv/bin/python -m pip install -q cocotb==2.0.1 pyuvm==4.0.1 -r labs/common/requirements.txt
cd labs/lab-06-the-register-block
make regcheck SYSTEM_PY=/w/.venv/bin/python PYTHON=/w/.venv/bin/python
make run SYSTEM_PY=/w/.venv/bin/python PYTHON=/w/.venv/bin/python
make check SYSTEM_PY=/w/.venv/bin/python PYTHON=/w/.venv/bin/python
```

`make regcheck` printed `the model builds: 10 registers`, and `make run` printed `ok: 0 errors, 0 assertions 5.78 s`. The log of that run had these lines:

```
Initialized cocotb v2.0.1 from /w/.venv/lib/python3.12/site-packages/cocotb
pytest not found, install it to enable better AssertionError messages
[SB] 1123 packets in: 282 to output 0, 286 to output 1, 179 NOROUTE, 376 FULL; the highest
level a header saw was 16
[SB] beats checked: 1283 on output 0, 1254 on output 1; 6 DROP_CNT reads checked
[SB] registers: 6 reads, 0 of STATUS checked; 0 transfers to bad addresses; irq checked in 0
cycles, wrong in 0
```

The three `[SB]` lines are the same as the ones in Lab 06, Part 1, and `make check` gave the verdict that part shows for the starter, `VERDICT: FAIL -- your test passes B01_status_write_clears_all too.` In an earlier run of the same commands, which used `/usr/bin/python3` and `pip`'s `--disable-pip-version-check`, the venv took 14.0 s to make and came to 102 MB. The venv has no pytest, because nothing the course installs requires it, which is why cocotb printed `pytest not found` in the log.

Installing on other systems

pyuvm's README is also its PyPI description, and the image's copy of the package includes it. It gives this command, and says that the command also installs cocotb, which pyuvm requires (*untested by us*):

```
% pip install pyuvm
```

cocotb-coverage's README gives this one (*untested by us*):

```
pip install cocotb-coverage
```

Neither cocotb's README nor pyvsc's gives an install command. Each points to its project's documentation instead, at https://docs.cocotb.org/en/stable/ and https://dvkit.org/fvutils/pyvsc/.

The metadata of the four packages, read from the copies in the image and in the venv that `make venv` made, says which Python each one needs and what else it requires:

Package	Python	Other requirements
cocotb 2.0.1	3.6.2 or later	find_libpython
pyuvm 4.0.1	3.6 or later	cocotb 1.6.0 or later, and earlier than 3.0
pyvsc 1.1.0	3.10 or later	dv-solve, pyucis and toposort; on Linux x86-64 and macOS, also py-boolector
cocotb-coverage 2.0	3.11 or later	cocotb 2.0 or later, python-constraint, pyyaml

pyvsc needs pyucis, and pyucis pulls in an MCP server stack (mcp, starlette, uvicorn, httpx2 and more), none of which the course uses (`labs/common/README.md`). Neither install command above names a version. The labs were measured with cocotb 2.0.1, pyuvm 4.0.1, pyvsc 1.1.0 and cocotb-coverage 2.0, which are the versions that the command under Everything in one venv pins.

For Verilator, DD105-A's page (Getting the tools outside the course image) quotes these steps from Verilator 5.050's install guide (`docs/guide/install.rst` at tag `v5.050`), for building it from Git on Ubuntu. We copied them from DD105-A's page, not from the guide itself (*untested by us*):

```
# Prerequisites:
#sudo apt-get install git help2man perl python3 make autoconf g++ flex bison ccache
#sudo apt-get install libgoogle-perftools-dev libjemalloc-dev numactl perl-doc
#sudo apt-get install libfl2  # Ubuntu only (ignore if gives error)
#sudo apt-get install libfl-dev  # Ubuntu only (ignore if gives error)
#sudo apt-get install zlibc zlib1g zlib1g-dev liblz4 liblz4-dev  # Ubuntu only (ignore if
gives error)

git clone https://github.com/verilator/verilator   # Only first time

# Every time you need to build:
unsetenv VERILATOR_ROOT  # For csh; ignore error if on bash
unset VERILATOR_ROOT  # For bash
cd verilator
git pull         # Make sure git repository is up-to-date
git tag          # See what versions exist
#git checkout master      # Use development branch (e.g. recent bug fixes)
#git checkout stable      # Use most recent stable release
#git checkout v{version}  # Switch to specified release version

autoconf         # Create ./configure script
./configure      # Configure and create Makefile
make -j `nproc`  # Build Verilator itself (if error, try just 'make')
sudo make install
```

For this course, the last of the three checkout lines becomes `git checkout v5.050`, without the `#`. DD105-A's page also quotes the guide's note that ccache and mold are "If present at build, needed for run", which means that if they were installed when you configured Verilator, they have to be installed wherever you use it.

Other differences outside the image

What	In the image	Elsewhere
The machine	every build and run time on these pages was measured with 2 CPUs and 3 GB, and `pyuvm.mk` builds with two g++ jobs at a time (`JOBS=2`)	set `JOBS` to the number of g++ jobs you want to run at once
Shell tools	GNU make 4.3, bash, GNU coreutils	`pyuvm.mk` uses `timeout`, `stdbuf`, `date +%s.%N`, `seq`, `awk`, `du`, `cmp` and bash's `printf %q`; we haven't run it on a system where these work differently, such as macOS
Python	3.12.3, with cocotb, pyuvm and pytest in `/usr/local/lib/python3.12/dist-packages`	3.11 or later, because cocotb-coverage needs it; we have run only 3.12.3
The course venv	in `/foss/designs/.venvs/dd105-b` in the workspace, or `.venv` in the clone	`.venv` in the clone, unless you set `VENV_DIR`
ccache	in use, because the image's `verilated.mk` sets `OBJCACHE ?= ccache`	look for `OBJCACHE` in your Verilator's `verilated.mk`. The build times on these pages assume ccache is in use. Lab 01's pyuvm build, for example, took 1.7 s because six of its seven files came from the cache, and 7.3 s with an empty cache
Waves	the simulator's VCD is converted to FST with `vcd2fst`	`make waves` needs `vcd2fst` on your `PATH`