DD104 — Pre-Silicon Verification

In this course you find the bugs hidden in a real 8-bit processor, using cocotb, Icarus Verilog and Verilator, before they cost a mask set.

DD104 is part of the UofT ASIC Team education materials, published at https://uoftasic.com/dd104/. The labs run in the workbench: the IIC-OSIC-TOOLS container you start from the `workspace` clone you set up in IC101. You get this course's files with the workbench's `mod` command (`mod add dd104`); Lab 01, Part 1 walks you through it.

Who this course is for

You have finished the digital track up to DD103: you can write SystemVerilog, simulate it, and take it through a flow to layout. None of those steps checked whether the design computes what you meant it to, and that question is what this course is about.

SimProc is an 8-bit processor written by students on this team and wrapped for a real chip (a TinyTapeout shuttle): an ALU, four registers, 64 bytes of memory and a UART, about 1,200 lines of SystemVerilog. It shipped with one test, and that test still passes on a processor whose `SUB` has its operands swapped and computes `B - A`. DD104 ships thirteen broken versions of SimProc, each made by a one-line edit, and all thirteen pass that test. By the end of the course you will have written a test suite that catches all thirteen, and a bug report for one of them.

Track	Digital
Prerequisites	IC101, then DD101, DD102 and DD103, in that order
Tools	cocotb 2.0.1, Icarus Verilog 14.0, Verilator 5.050, Surfer and GTKWave, all in the workbench image `hpretl/iic-osic-tools:2026.08`
Machine	The workbench you already have. This course needs no PDK, no LibreLane and no synthesis. Most runs take under ten seconds; the longest, Lab 06's `make check-all`, takes about four and a half minutes.
Total time	About 19 hours, self-paced. The Course path rows below add up to 18 h 55 min.

Check your cocotb version before you search the web for help: `cocotb-config --version` should print `2.0.1`. That matters because most cocotb tutorials online were written for 1.x, and some of the things they use are errors in 2.x, among them `cocotb.fork` (use `cocotb.start_soon`), `@cocotb.coroutine` (use a plain `async def`) and `dut.sig <= 1` (use `dut.sig.value = 1`). Most of the rest, such as `units=`, `MODULE=` and `.value.integer`, still runs and only prints a deprecation warning. cocotb 2.x and older tutorials has the full list.

Course path

Work down this table in order. A Read row is a set of guide pages, which are for reading only and have no commands in them. A Do row is one lab writeup, and the lab writeups are where every command in the course lives. Each lab's "Before you start" section repeats which pages it expects you to have read. The counter lab in row 4 has no number, but it isn't optional.

#		What	Time
1	Read	Getting started · What a passing test proves · What a testbench is · Your first cocotb test · The lazy	1 h 15 min

#		What	Time
		testbench · Coverage as a question	
2	Do	Lab 01: The adder you already trust. Load the course, watch a reasonable testbench pass on a broken adder, then fix the testbench so it catches it.	1 h 30 min
3	Read	Clocks, resets and triggers · cocotb 2.x and older tutorials · Driving and checking a design · Waveforms	1 h
4	Do	Lab: cocotb on a counter. Run the guide pages' cocotb examples on an 8-bit counter, including the 1.x forms that fail in 2.x, then break the counter on purpose and see which tests notice.	1 h 15 min
5	Read	Two simulators · SimProc in one page	45 min
6	Do	Lab 02: The only test this chip has. Run the shipped test against all thirteen broken cores, then write a better one.	1 h 45 min
7	Read	Flags as architectural state	20 min
8	Do	Lab 03: Three blocks before the processor. Unit-test three SimProc modules, including all 393,216 ALU vectors.	2 h
9	Read	The golden model · Where bugs hide · Directed and random tests	1 h
10	Do	Lab 04: Your first bug hunt. Get a mystery core, find the bug, and prove the fix with a test.	2 h 30 min

#		What	Time
11	Read	Handshakes and timing	25 min
12	Do	Lab 05: Every answer is right. Write three checks that catch two cores on which every value is correct, then hunt both.	1 h 15 min
13	Read	Writing the bug report	25 min
14	Do	Lab 06: Capstone. Build one suite that catches all thirteen bugs, and write a bug report for one of them.	3 h 30 min

Keep these reference pages open in a tab while you work. They're for looking things up rather than for reading start to finish:

- The SimProc instruction set
- The SimProc harness
- Output that is not an error

Getting help

Ask in the team Discord; nobody expects you to work all of this out alone. Getting started says what to include in a question so it can be answered quickly.

Next courses

- DD201 — High-Level Synthesis
- DD202 — Computer Architecture