Sim Toolchain Quick-Reference

1,013 words · ~5 min read

---

*VCA-CSA-101 cross-chapter quick-reference handout. Anchors: §5.10.5 (sim-then-silicon discipline) + Lab 5.6 step-minus-one (sim toolchain pre-flight) + (engineering-side doctrine doc) + (xpm primitive selection) + (RAM inference). *

Purpose: the canonical commands + helpers + pre-emption checklist for the Virtus build-out's HDL simulation discipline. Print and pin during Lab 3.3 (memory-array sim verification), Labs 5.1-5.6 (CPU + IP integration sim), Lab 11.1 (compiler library-call codegen sim coverage), Labs 12.1-12.5 (OS implementation cocotb regression). **The discipline this card encodes (sim until convergence, then silicon) is the most-central pedagogical lesson of CSA-101.**

At a glance

Property	Value
Container	`cocotb-sim:latest` (Docker)
Simulator	iverilog 12.0 (event-driven; SDF-capable for post-PAR timing sim)
Test framework	cocotb 2.0.1
Test language	Python 3.12
Waveform format	VCD / FST → GTKWave
Default backend flag	`-g2012` (cocotb forces; SV-2012 reserved-keyword discipline applies)
Cost ratio vs Vivado silicon iter	**~24-36× sim-wins** (sim ~30 sec; silicon ~12-15 min per iter)
When to use	Any HDL lab where iter cost > 30 sec on silicon target. Effectively all CSA-101 + CSA-201 hardware labs

---

Lab 5.6 step-minus-one. Verify sim toolchain BEFORE you need it

The handout's central pre-emption: **before any HDL lab, confirm `cocotb-sim:latest` is mounted-and-runnable.** This is the discipline rule from `feedback_unblock_via_docker_or_alt_host.md`: when the toolchain blocks on a privilege escalation (`apt install iverilog` requiring sudo), the right move is **pivot to the container, not wait**.

```
# Step-minus-one verification - should take < 30 seconds
docker run --rm cocotb-sim:latest bash -c "
  iverilog -V 2>&1 | head -2
  echo '---'
  python3 -c 'import cocotb; print(cocotb.__version__)'
"
# Expected:
#   Icarus Verilog version 12.0 (stable)
#   ---
#   2.0.1
```

If the container isn't present locally:

```
docker pull cocotb-sim:latest    # or: docker build -t cocotb-sim:latest .
```

Pre-flight is mandatory; it is not the silicon round. Lab 5.6 D2 reflection prompt asks specifically: *"did your sim toolchain provisioning ever block a round? what would you change about pre-fabbing the toolchain next time?"*, the answer is: pre-fab it now, not when you need it.

Canonical Docker run command

For any cocotb test directory containing a `Makefile` + `test_*.py`:

```
cd path/to/your/lab/tests/sim
docker run --rm \
  -v "$(pwd)/../../..":/work \
  -w /work/path/to/your/lab/tests/sim \
  cocotb-sim:latest \
  bash -c "make 2>&1"
```

Mount the **repo root** (3 levels up typically) so cocotb sees both your test files AND the HDL sources they reference. Working directory = the test directory. Keep all simulator outputs (VCD, results.xml, sim_build/) inside the test directory; gitignore them in `.gitignore`.

Minimum viable cocotb Makefile

```
SIM                 ?= icarus
TOPLEVEL_LANG       ?= verilog
TOPLEVEL            := your_top_module
COCOTB_TEST_MODULES := your_test_module
COCOTB_RESULTS_FILE := results.xml

VERILOG_SOURCES := \
    ../../hdl/your_top.v \
    ../../src/upstream_files.v
    # ... add all .v files needed

include $(shell cocotb-config --makefiles)/Makefile.sim
```

Critical: `COCOTB_TEST_MODULES` (not legacy `MODULE`); `COCOTB_RESULTS_FILE` lets CI parse PASS/FAIL counts. **Do not** add `SIM_ARGS += -fst`. Iverilog 12.0 vvp doesn't accept `-fst` flag (use `WAVES=1` env var via cocotb's standard mechanism if waveforms are needed).

Canonical AXI4-Lite handshake helpers

Reuse-as-is across HDMI / fpga_pio / future audio / PS-2 / DS2 / GPIO peripheral wrappers. Cite verbatim from `virtus-peripheral-ip-pack/fpga_pio/PORT_DELTAS.md` §2 + `kb/build-out/sim-as-debugger-pattern.md` §6:

```python
import cocotb
from cocotb.clock import Clock
from cocotb.triggers import RisingEdge


async def axi_write(dut, addr, data):
    """Single AXI4-Lite write transaction with combinational handshake."""
    dut.s_axi_awaddr.value  = addr
    dut.s_axi_awvalid.value = 1
    dut.s_axi_wdata.value   = data
    dut.s_axi_wstrb.value   = 0xF
    dut.s_axi_wvalid.value  = 1
    dut.s_axi_bready.value  = 1
    while True:
        await RisingEdge(dut.clk)
        # Sample post-edge directly; cocotb 2.x forbids ReadOnly-phase set
        if int(dut.s_axi_awready.value) == 1 and int(dut.s_axi_wready.value) == 1:
            break
    dut.s_axi_awvalid.value = 0
    dut.s_axi_wvalid.value  = 0
    while True:
        await RisingEdge(dut.clk)
        if int(dut.s_axi_bvalid.value) == 1:
            break
    dut.s_axi_bready.value = 0
    await RisingEdge(dut.clk)


async def axi_read(dut, addr):
    """Single AXI4-Lite read transaction. Returns 32-bit data."""
    dut.s_axi_araddr.value  = addr
    dut.s_axi_arvalid.value = 1
    dut.s_axi_rready.value  = 1
    while True:
        await RisingEdge(dut.clk)
        if int(dut.s_axi_arready.value) == 1:
            break
    dut.s_axi_arvalid.value = 0
    while True:
        await RisingEdge(dut.clk)
        if int(dut.s_axi_rvalid.value) == 1:
            data = int(dut.s_axi_rdata.value)
            break
    dut.s_axi_rready.value = 0
    await RisingEdge(dut.clk)
    return data
```

Why these exist: cocotb-bus's stock AXI4-Lite master had compatibility quirks with iverilog 12.0 + cocotb 2.0.1 surfaced during fpga_pio Phase 2. Hand-rolling the helpers is ~30 lines and works deterministically across all Virtus IP Pack peripherals. **Update once, propagate everywhere.**

---

4 bug-class preemption checklist

Before authoring HDL or cocotb tests, scan this list. Every item caught early saves a sim iteration (~30 sec), but more importantly, **each catches a bug-class that has cost 4-8 hours to isolate when found late.**

1. SV-2012 reserved-word collisions (cocotb forces `-g2012`)

```
Reserved in SV-2012 - DO NOT use as identifier:
   null  ref  void  let  bind  unique  priority  global  packed
   tagged  union  interface  endinterface  modport  endmodport
   class  endclass  package  endpackage  property  endproperty
```

If your upstream Verilog uses any of these as a wire/reg name, **rename it before cocotb compiles**. R1E-PIO.1 hit `null` in upstream lawrie/fpga_pio's `machine.v`; renamed to `null_src` in 9 occurrences (1-line patch documented in PORT_DELTAS.md §1).

2. AXI4-Lite handshake: combinational AWREADY/WREADY, not registered

```
// ❌ WRONG - registered handshake creates a 1-cycle gap; master+slave
// can never see each other's high state simultaneously
output reg s_axi_awready;
always @(posedge clk) begin
    if (state == W_IDLE) s_axi_awready <= 1'b1;
    else                 s_axi_awready <= 1'b0;
end

// ✅ RIGHT - combinational; both visible same edge
output wire s_axi_awready;
assign s_axi_awready = (state == W_IDLE);
```

This bug class repeats across every AXI4-Lite slave. Catching it at sim-time is significantly cheaper than finding it on silicon; the canonical helpers above preempt it in every peripheral wrapper.

3. Cocotb 2.x ReadOnly phase strictness

```
# ❌ WRONG - cocotb 2.x raises RuntimeError on signal-set after ReadOnly
while True:
    await RisingEdge(dut.clk)
    await ReadOnly()
    if int(dut.s_axi_awready.value) == 1: break
dut.s_axi_awvalid.value = 0   # ← FAILS in ReadOnly phase

# ✅ RIGHT - sample post-RisingEdge directly without ReadOnly
while True:
    await RisingEdge(dut.clk)
    if int(dut.s_axi_awready.value) == 1: break
dut.s_axi_awvalid.value = 0   # ← OK in normal time phase
```

4. AXI memory-region alignment

```
INSTR_MEM at 0x040..0x0BF spans bit[7]=0..1 → range NOT power-of-2 aligned →
single-bitmask range decode silently drops upper half of writes.

✅ RIGHT - relocate to 0x080..0x0FF (128-byte aligned; bit[11:7]=5'b00001).
```

Force memory regions to **power-of-2 size aligned to power-of-2 offset**. The decode `b_waddr[11:N] == constant` only works when the range is a clean prefix.

When-to-use decision rule

```
Iteration cost on silicon target > 30 seconds?
└── Yes (every CSA-101+ HDL lab; Vivado bitstream gen ≥ 12 minutes;
         Tang Primer 25K bitstream gen + load ≥ 5 minutes)
       └── Iterate in sim until cocotb harness PASSes all gates
              └── Only THEN generate bitstream for silicon validation
```

Never iterate on silicon for logical-functional bugs. The sim catches: AXI handshake races, address-decode misalignments, FSM stuck-states, primitive-elaboration failures, register-clobber violations, syscall-ABI bugs. Silicon catches: timing closure, IO-pad behavior, reset-glitches, DRAM PHY tuning, EMI/thermal effects (per `kb/build-out/sim-as-debugger-pattern.md` §3-§4 cost-ratio analysis).

Cross-chapter integration points

Lab 5.6: Reading Your Own Hardware

Step zero: consult KB before iterating (per ch5 §5.10.5 + project KB rule). **Step minus-one**: verify sim toolchain provisioned (this handout). The two prefixes compose: KB lookup is instant (~0 sec); sim iter is ~30 sec; silicon iter is ~12-15 min.

Lab 11.1: Library-call Codegen Sim Coverage

The forensic_syscall_* cocotb suite (vca-csa-101 commit `521d628`) catches stack-frame ordering bugs, register-clobber violations, and return-value-on-stack discipline at sim-time. **The compiler trusts the OS-resident services to exist; the OS contracts to provide them; drift between the two is a curriculum bug.** Sim coverage of the syscall ABI surface keeps drift from reaching silicon.

Lab 11.3 + Lab 12.4: Console + Sound IP Integration Sim

When you wire the AXI4-Lite Console + Sound IP cores into the M0-2 IP Pack manifold, the canonical AXI handshake helpers above are exactly the ones the testbench imports. Same pattern. Different register map.

Lab 12.5: Capstone OS-impl Walkthrough

OS-impl bugs (especially the Memory.alloc free-list discipline + the Sys.error trap path) are caught at sim-time long before silicon iteration would surface them. R6B Memory.init's heapBase wrap was a 4-round silicon flake that sim caught deterministically once cpu_addr range coverage extended.

CSA-201 Ch 4 forward-pointer

"Programmable I/O Substrates" hosts the lawrie/fpga_pio + Virtus AXI wrapper substrate-revelation moment. Students who internalised this handout's discipline encounter the substrate ready to **iterate in sim then promote to silicon**, the doctrine COMPOSES with the substrate-multiplier finding (per VHWP3 §B-uplift).

What this handout doesn't cover

- **Vivado xsim / Verilator backends**: cocotb supports both; this handout standardises on iverilog because it's what `cocotb-sim:latest` ships with. If a future round adopts Verilator (per `simulator-stack-research-2026-04-30.md` recommendation), update the handout's `SIM` Makefile variable.
- **Post-PAR timing sim**: requires Vivado-emitted SDF (Xilinx) or apicula-emitted SDF (Gowin; pending). When that path lands, a follow-on handout `cross-chapter-post-par-timing-sim-quick-ref.md` will cover it.
- **GTKWave waveform analysis**: separate skill; covered in Lab 3.3 (memory-array verification) and CSA-201 driver-track.

Further reading

- **Xilinx UG901** (Vivado Design Suite User Guide: Synthesis). Vivado synthesis attribute reference.
- **Xilinx PG058** (XPM Memory Library). `xpm_memory_*` family reference.
- **PYNQ Overlay API docs** (readthedocs.io). `Overlay()`, `MMIO`, `ip_dict` reference.

Pin during every HDL lab from Ch 3 forward. The discipline compounds: every iteration in sim is one fewer iteration on silicon, and every silicon iteration saved is ~12-15 minutes you don't burn. Sim is the cheap tool; silicon is the expensive oracle. Use them in that order.
