

RISC-V in OCaml

What can we check before programming the FPGA?

Kiwamu Okabe
kiwamu@metasepi.org
https://metasepi.org/

I would like your advice on this verification flow.

A RISC-V SoC written in HardCaml

- Started with an RV32E MCU on an iCEstick
- Now boots Linux 6.18, including SMP, on a Colorlight i9 (ECP5)
- HardCaml: an RTL library embedded in OCaml
- The designer chooses the registers and cycles; no HLS scheduling

```
                          +-> Cyclesim -> OCaml reference model
OCaml / HardCaml -------+
                          +-> Verilog -> Yosys / nextpnr -> FPGA
```

Most code is written by coding agents. Tests must check their work.

github.com/takibi-lang/takibi-hardware



S0: Can we construct the circuit?

RTL counterpart: compilation, elaboration and structural checks

- OCaml checks host-language types; HardCaml checks circuit structure
- Detect width mismatches, undriven signals and combinational loops
- Also prove selected optimized blocks equal simple references (`Hardcaml_verify.Sec` + z3)

Example: optional S-mode registers were left undriven (B-0025).

Boundary: well-formed hardware can still implement the wrong function.

Scope: circuit construction; equivalence checks for selected blocks.



S0f: Does a property hold across cycles?

RTL counterpart: property checking with formal tools

- Generated Verilog + hand-written SystemVerilog properties
- SymbiYosys: bounded model checking and k-induction
- Check shared-bus exclusion and atomic-operation ordering

Example: one core could decide that an SC succeeded while the other core held the bus for an AMO. A **3-cycle counterexample** exposed it before two-core simulation (B-0048).

Boundary: proofs depend on the properties and environment assumptions.

Scope: selected dual-core arbiter properties, not the complete CPU.



S1: Does the circuit do the right thing?

RTL counterpart: cycle simulation with a reference model / scoreboard

- Cyclesim runs directly from the HardCaml circuit
- Directed tests, random programs, interrupts and memory wait states
- Compare architectural effects at every retired instruction against an OCaml instruction-set model

Example: two redirects let an old fetch response become valid again. Random lockstep with interrupts and wait states caught it (B-0047).

Boundary: two-state, zero-delay; shared mistakes in models remain a risk.

Scope: IP, cores and system benches; independent ISA tests add another oracle.



S2: Does the generated Verilog behave the same?

RTL counterpart: simulation of the emitted RTL

- Compile the generated Verilog with Verilator
- Reuse the OCaml benches through a Cyclesim-compatible interface
- Apply the same stimulus, reference model and checks

Evidence: MCU firmware and UART benches run on both backends.
No ledger entry was first found at S2 proper in this audit.

Boundary: reusing a bench also reuses its blind spots;
this is not a four-state initialization check.

Scope: shared benches for smaller designs; a separate SoC Verilator bench also boots Linux in lockstep.



S2b: Does the board top start correctly in RTL?

RTL counterpart: four-state, board-top RTL simulation

- Icarus runs the generated board Verilog with firmware
- Exercise reset/start-up and UART at the pins
- Look for uninitialized state hidden by two-state simulation

Example: a port named `time` passed Cyclesim but failed Verilog generation for board-sim: it is a reserved word (B-0031).
The regression now emits Verilog at S1.

Boundary: X behavior depends on the models and testbench setup; simulation does not establish when real silicon is ready.

Scope: iCEstick and i9 MCU board tops, not a complete board model.



S2n: Does the synthesized circuit still work?

RTL counterpart: post-synthesis functional simulation

- Read Yosys iCE40 / ECP5 netlists back into HardCaml
- Model LUTs, flip-flops and block RAM; reuse Cyclesim benches
- Check firmware in BRAM initialization and cell power-up values

Example: Yosys chose power-up 1 for a counter bit, bypassing the i9 start-up hold. RTL tests had missed this assumption (B-0023).

Boundary: two-state cell models; no routed delays.
The netlist reader and cell models also need verification.

Scope: MCU benches and board tops for iCE40 / ECP5.



S3: Does it fit, and does it meet timing?

RTL counterpart: synthesis, place-and-route and static timing analysis

- Yosys + nextpnr; check logic budget and expected BRAM count
- Check target frequency and flip-flop naming
- Record tool versions: placement and compiler results depend on them

Examples:

- Unwanted read/write collision logic around BRAM (B-0004)
- First Linux SoC build: **34.27 MHz vs. a 50 MHz target** (B-0043)

Boundary: results depend on correct timing constraints and device models. There is no post-route delay simulation in this flow.

Scope: board builds, including the Linux SoC.



S4: Do our assumptions match the board?

RTL counterpart: FPGA bring-up and hardware validation

- Start small: blink, UART, memory test, then firmware / Linux
- Change one thing at a time, or keep an ablation switch
- Turn discoveries into cheaper regressions where possible

Example: consecutive SDRAM reads returned the next word's data.
The CAS-latency timing derivation used the wrong edge (B-0046).
The simulation model used our chosen latency, so it agreed with us.

Boundary: passing a board test covers only the exercised conditions.

Scope: iCEstick MCU; i9 MCU, I/O bring-up and Linux SoC, including SMP.



What the defect ledger says -- and what I want to ask

First found	S0	S0f	S1	S2	S2b	S2n	S3	S4	Total
Defects	3	1	28	0	2	3	10	4	**51**

32/51 first found at S0/S0f/S1. Includes benches, models and tools; this is not an RTL coverage measurement or a comparison with Verilog.

- Where are the most serious gaps in this verification flow?
- Where would you spend the next unit of verification effort?

BUGS.md: entries, causes and regressions
/ Workflow



Takibi: the software side

Move runtime failures into compile-time checks

- A research systems language and a kernel developed together
- OCaml + LLVM compiler; runs Alpine BusyBox on RPi5 and QEMU/AArch64
- Future goal: hardware/software co-design on FPGA

Today: feedback on the hardware verification flow is very welcome.

github.com/takibi-lang/takibi

