Automated Design of RISC-V Processors using LLM-Guided feedback

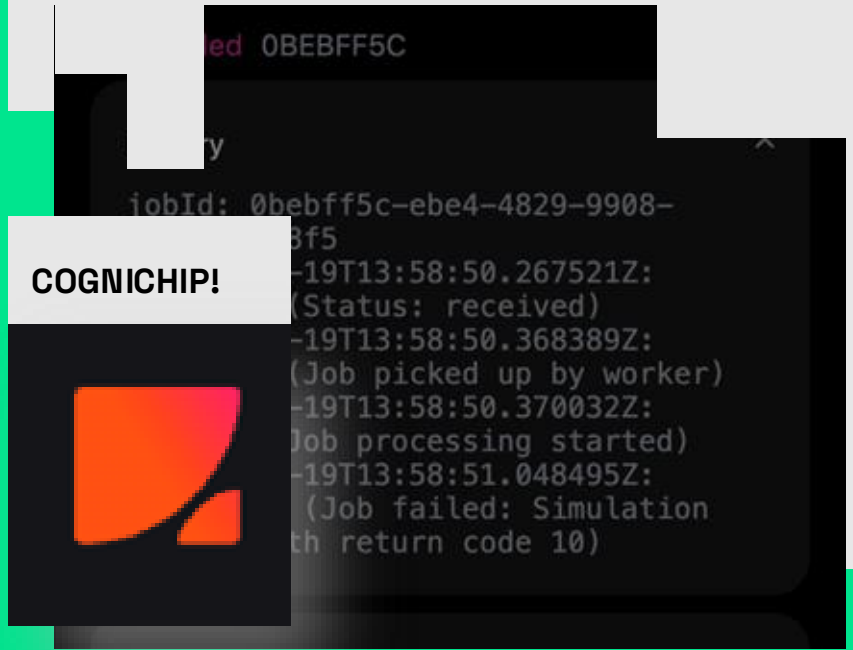
Nathan Macapinlac
Carlos Perez-Guzman

https://github.com/nasan016/risc-v

Building a correct RISC-V CPU is difficult due to subtle control, PC, and timing bugs. Our goal was to use Cognichip's LLM-guided feedback to rapidly converge on a functionally correct RV32I single-cycle CPU verified against a golden reference model.

MOTIVATION

WORKFLOW



1. Specify architecture (RV32I, single-cycle)
2. Generate RTL using Cognichip
3. Simulate using Verilator + golden RV32I reference model
4. Observe mismatches (PC, regwrite, rd, mem)
5. Iteratively refine RTL with Cognichip feedback
6. Repeat until lockstep pass

How Cognichip Was Used

Cognichip acted as a rapid prototyping and debugging assistant, not a one-shot solution.

GENERATED_

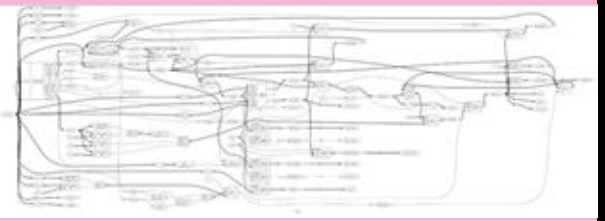
- Control logic skeleton
- Datapath structure
- Initial PC update logic

ASSISTED DEBUGGING_

- Identified incorrect PC update paths
- Fixed regwrite conditions
- Corrected immediate handling
- Highlighted mismatches vs golden model

ARCHITECTURE/RTL DESIGN

This figure below shows the synthesized datapath connectivity of our single-cycle RV32I CPU. The PC feeds instruction memory, instruction fields drive control and immediate generation, operands are selected into the ALU, and results are written back to the register file or memory. Control-flow paths for branch, JAL, and JALR are handled through PC muxing. We also added commit-aligned debug registers to match the golden model.



CORE COMPONENTS

- PC + PC update mux
- Instruction memory interface
- Control Unit
- Register File (32 × 32-bit)
- Immediate Generator
- ALU
- Data memory interface
- Debug commit signals

DESIGN CHOICES

- RV32I base ISA
- Single-cycle datapath
- Harvard-style memory interface
- Commit-aligned debug signals for golden model matching

INSTRUCTION SUPPORT

- **Arithmetic:** add, sub, addi
- **Control flow:** beq, bne, jal, jalr
- **Memory:** lw, sw
- **Upper immediates:** lui, auipc

RTL IMPLEMENTATION HIGHLIGHTS

PC Update Logic

Correctly handles:

- Sequential PC+4
- Branch targets
- JAL
- JALR (LSB cleared)

Register File Semantics

Register File Semantics

- x0 hardwired to zero
- Writes suppressed for rd = x0
- Synchronous write, async read

Commit-Aligned Debug Signals

Commit-Aligned Debug Signals

- dbg_pc reflects *committed* instruction PC
- dbg_regwrite only true when architectural state changes
- dbg_rd, dbg_wdata match golden model

Simulation & Verification Setup

DUT must match golden model on every architectural commit.

VERIFICATION METHOD

DUT (our CPU) runs in lockstep with:

- Golden RV32I reference model

Cycle-by-cycle comparison of:

- PC
- Instruction
- regwrite flag
- rd
- write data
- memory operations

What is our Golden Model?

It is **NOT a CPU implementation.**

It is a **cycle-accurate reference model of RV32I behavior** used for verification.

A **“truth oracle”** that computes what should happen architecturally for each instruction.

Key Observations

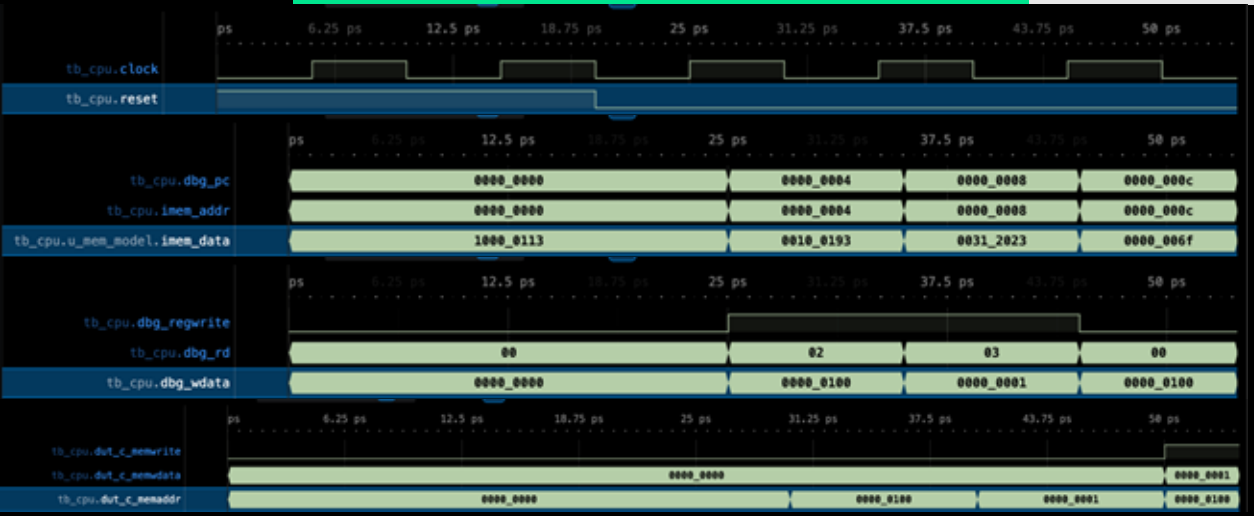
PC increments correctly.

Register writes occur only when expected.

Destination registers match golden model.

Memory accesses occur at correct addresses.

All verification checks passed with zero mismatches.



Simulation Results

Challenges & Lessons Learned

Ask participants to share interesting trivia about themselves before the meeting and add it to these cards. Then, have everyone guess who each card refers to.

Debug Signal Timing Misalignment

Our debug outputs reflected combinational values instead of committed architectural state, leading to off-by-one-cycle mismatches with the

Subtle RV32I Semantics (JAL/JALR, Branches)

Correctly handling PC updates, link register writes, and branch targets required careful interpretation of RV32I semantics beyond

PC Commit Semantics Mismatch

We initially compared the DUT PC before commit against the golden model's committed PC, causing false failures until we aligned the debug PC to architectural commit timing.

Branch Condition and Zero Flag Edge Cases

Branch correctness depended on ALU comparison semantics and sign handling, which exposed subtle bugs

Golden-Model Verification Finds Real Bugs

Cycle-by-cycle comparison against a reference model exposed control-flow and writeback bugs that basic waveform inspection alone would not catch.

Architectural State ≠ Internal Signals

Verification must compare architectural commit state (PC, register writes, memory writes), not internal datapath wires, to avoid misleading "correct-looking" waveforms.

LLMs Accelerate Debugging, Not Design Thinking

Cognichip and LLM feedback helped iterate quickly, but correctness still required manual reasoning about ISA semantics,

Conclusion

- Successfully built a correct RV32I CPU
- Verified using golden model lockstep
- Demonstrated effective AI-assisted RTL iteration
- Identified real-world pitfalls of LLM-based hardware design

Impact

This project shows how Cognichip can meaningfully accelerate early-stage RTL design when paired with rigorous verification discipline.