

Unbounded Formal Verification of RISC-V CSRs with Interval Property Checking

Nicolae Tusinschi, Sven Beyer - OneSpin Solutions
nicolae.tusinschi@onespin.com

Motivation: RISC-V Potential

- RISC-V cores becoming very popular**
 - Extensive ecosystem promises huge advantages
 - Used in critical application domains including mil/aero, automotive, industrial, IoT
 - Challenges for functional safety, security, trust
- Confidence in IP implementation critical for business and mission success**
 - Commercial IP vendors and internal IP groups must perform extensive verification and demonstrate the results to their clients
 - Open-source IP users must perform own verification, especially when adding custom extensions (allowed by RISC-V specs)

1

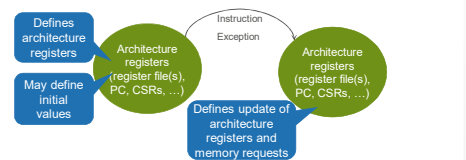
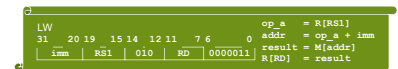
Motivation: RISC-V Verification Challenges



- RISC-V cores hard to verify**
 - Complex microarchitectures to hit PPA targets
 - Branch prediction, forwarding, out-of-order execution
 - Many configurations of implementation
- Inadequate methods**
 - Months of verification setup and simulation
 - Formal verification focused only on finding bugs
 - riscv-formal* - related work
 - Incomplete formal proofs, hard to set up and reuse
 - Bugs and additional logic remain undetected

2

RISC-V Instruction Set Architecture (ISA) Spec



3

Formalized User-Level ISA

- Captures effect of instructions on architecture state and output to data memory
- Formalized in SystemVerilog Assertions (SVA)
- Different extensions such as C, A can be enabled

```
32'bXXXXXXXXXXXXXXXXXXXX01XXXXXXXXXXXX0000111;
decode.instr = LW;
decode.RS1.valid = 1'b1;
decode.RD.valid = 1'b1;
decode.imm = $signed(iw[31:20]);
decode.mem = 1'b1;
```

ISA formalization
excerpt for LW

4

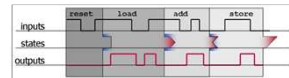
Pipelined Microarchitecture Verification

- Various implementation choices for microarchitecture
 - Specific pipeline length
 - Forwarding paths to decode state (or additionally also to later stages)
 - Separate ICACHE/DCACHE units with specific protocols
 - Branch prediction for instruction fetch unit
 - Stalling of later pipeline stages or replay mechanism
 - Out-of-order termination for long-latency instructions (like DIV, DCACHE miss)
- Verification links pipeline to sequential execution of instruction
 - Capture full effect of one instruction/exception in pipeline one property
 - Regardless of preceding or succeeding instructions
 - Next sequential instruction "starts" when leaving decode
 - Need to capture "sequential" register file where effect of instruction is visible in 1 step

5

Key Idea: Interval Property Checking (IPC)

- Each assertion focuses on a RISC-V instruction
- Interval goes from instruction decode to retire (typically writeback pipeline stage)
- The exact sequence instructions previous or subsequent to the one under verification is irrelevant
- Cases such as bubbles, replays, and cancel must be verified separately



A finite number of IPC assertions, each verifying a limited number of cycles, can cover any arbitrarily long trace

6

Anatomy of an IPC Assertion

- Reusable SystemVerilog Assertions (SVA) achieving unbounded proofs
 - Decouple ISA spec requirements from microarchitectural details
- Starting state
 - NOT the reset state, otherwise hard to achieve unbounded proofs and decouple assertions to verify individual instructions
 - Does not start from reset but from a generic valid state
 - Limited number of cycles (interval) to reach generic valid state
- Start state explicitly constrained in left-hand side of assertions implication to ensure that processor is ready to start processing a new instruction
- Instruction will have effects for a number of cycles (pipeline stages), however assertion must prove that on next cycle the core is again ready to fill the pipeline with the next instruction

7

Example IPC Assertion

```
property instruction_A;
  // conceptual state
  t ##0 ready_to_issue() and
  // trigger
  t ##0 opcode==instr_A_opc
  implies
  // conceptual state
  t ##1 ready_to_issue() and
  // memory interfaces outputs
  // read next instruction
  t ##0 imm_access(instr_A_opc) and
  // data load/store
  t ##1 data_access(instr_A_opc) and
  // architectural registers
  t ##1 R[RD].update(instr_A_opc) and
  t ##1 PC.update(instr_A_opc)
endproperty
```



8

Evidence of IPC Effectiveness

- Rocket core**
 - 64-bit processor core with a 39-bit virtual memory system
 - Extensions such as compressed and atomic instructions
 - Five-stage, single-issue, in-order pipeline
 - Out-of-order completion for long-latency instructions (division, cache misses)
 - Branch prediction
- IPC Verification**
 - Cover pipeline and CSRs (FPU, some special instructions, MUL/DIV units excluded)
 - CSRWW, CSRWS, CSRRC, CSRRI, CSRRI, CSRRCI, CSRRCI
 - ECALL, EBREAK, C_EBREAK, URET, SRET, MRET
 - Properties for exception handling in ID, EX, MEM, WB pipe stages and interrupt events
 - ~40 assertions developed

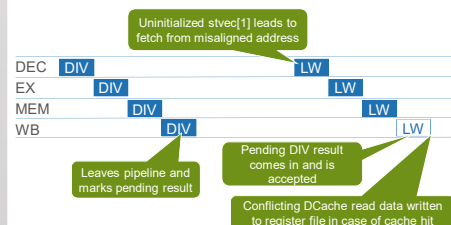
9

Results and Issues Found in Rocket Core

- Issues found**
 - DIV result not written in register file (#1752)
 - Corner-case scenario impossible to foresee
 - Jump instructions store different return PC (#1757)
 - Unexpected replay of illegal opcodes (#1861)
 - Undocumented non-standard instruction (#1868)
 - CEASE instruction added to core but not ISA spec
 - Undocumented CSR that reads back 0 (#1949)
- Runtime results**
 - Each property returns a result in less than 10 minutes with helper assertions
 - Each property returns a result in max. 5 hours with helper assertions
- Proof results**
 - Each property has an unbounded proof result
 - Each property is reachable

10

Rocket Core Issue: DIV Result Not Written



11

Results

- RISC-V formal verification**
 - Possible to achieve exhaustive, unbounded proofs
 - IPC is a suitable method
 - Requires formal and processor verification expertise
 - Provides very high confidence that RISC-V implementation fulfills ISA
 - Match or exceed quality of established architecture implementations
- Additional work**
 - Develop additional assertions to cover all missing instructions/functionalities
 - Package SVA code in formal app for easy deployment on other RISC-V cores

12