onespin

A Formal Methodology for Verifying RISC-V Cores

Chen Wei Wei, 13.11.2019

RISC-V

The First China RISC-V Forum

assuring IC integrity

Leading-Edge Technology

Targeting critical hardware verification challenges

Functional Correctness

Rigorous coverage-driven functional verification from block to chip, leveraging formal technology

- Design Exploration
- Protocol Violations
- Integrate Formal/Sim Coverage
- End-to-End User Assertions
- HLS/SystemC Verification
- Synthesis/P&R Errors

Safety

Safety analysis and higher diagnostic coverage to meet strict certification requirements

- FMEDA of Complex SoCs
- Failure Mode Distribution
- Avoid Excessive Fault Simulations
- Measure Diagnostic Coverage
- ISO 26262 Compliance
- Tool Qualification

Trust and Security

Automated detection of RTL Trojans and hardware vulnerabilities to adversary attacks

- Denial of Service
- Data Leakage
- Privileges Escalation
- Data Integrity/Confidentiality
- Hardware Backdoors
- Hardware Trojans

OneSpin 360® Formal Platform

Heterogeneous Computing

Thorough verification of complex SoC platforms used for 5G wireless, IoT, and AI applications

Automotive and Industrial

Systematic bug elimination and metrics on proper handling of random errors in the field

RISC-V

Efficient and complete verification, including custom extensions. Compliance to ISA.

OneSpin Solutions and Services

 |

RISC-V Verification Challenges

Does the RTL precisely implement the RISC-V ISA spec?

RISC-V ➤ RISC-V Core RTL Implementation ➤ **Verification?**

RISC-V cores become very popular

- Use in critical application domains including mil/aero, automotive, IoT, industry
- **Trust in IP implementation is critical for business and mission success**
- Commercial IP vendors, including internal IP groups must perform extensive verification, demonstrate the results to their clients
- Open source IP users must perform own verification, especially when adding custom extensions

RISC-V processor cores are hard to verify

- Complex µ-architectures to achieve PPA targets
- Many configurations of implementation

Inadequate Methods

- **Months of verification setup, weeks of simulation for each instance**
- **Bounded formal proofs, hard to setup&reuse**
- **Bugs and additional logic remain undetected**

Alternative: Formal methodology

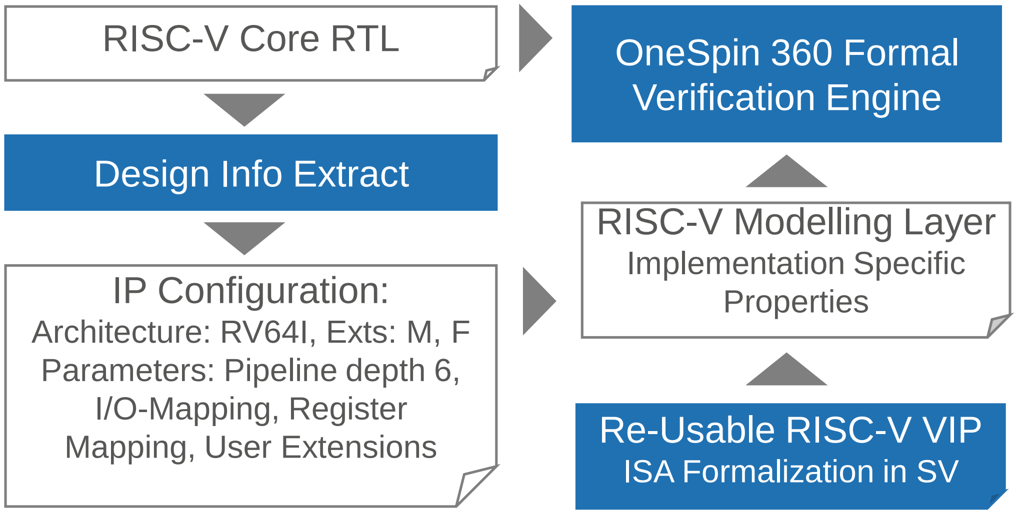
 |

RISC-V Formal Methodology

Automates and Accelerates Verification

OneSpin RISC-V Formal Verification App

- **Completely verifies RISC-V Core RTL Implementations** with full proofs, no bugs escape
- Guarantees full compliance with ISA and privileged ISA
- **Takes less than a week to setup, runs only 2 hours on complete core**
- Identifies unspecified instructions/CSRs
- **Proven on 32-Bit and 64-Bit commercial and open source implementations** with single issue pipeline, out-of-order completion, and branch prediction



Verification Flow with OneSpin RISC-V Formal App

 |

RISC-V ISA

Excerpt of configuration choices

Base	Released extension	Privileged spec	Draft extensions	Full custom
	A(tomic Instructions)	**U**(ser mode)		
RV**32I**	**M**(ultiplication and division)	**sv32** virtual memory	**V**(ector operations)	Custom registers
RV**64I**	**F**(loating point single)	**sv39** virtual memory	**B**(it manipulation)	Custom instructions
RV**128I**	**D**(ouble floating point)	**sv48** virtual memory	**Counters**	Custom exceptions
	C(ompressed)	**PMP** memory protection		

Huge amount of standardized configuration choices

RISC-V Parameterization

onespin

Sample configuration:

- RV64IMA
- Page-based 39-bit virtual memory system
- User mode
- No counters
- 8 PMPs
- Debug mode according to debug spec
- Initial PC 0x10040

SVA instantiation parameter map:

```
#(.MISA('{MXL:xl_64,
        I_BASE_ISA: 1’h1,
        S_MODE: 1’h1, U_MODE: 1’h1,
        A_EXT: 1’h1, M_EXT: 1’h1,
        default: 1'b0}),
  .SATP_MODE(sv39),
  .DBG_SUPPORT(xdbgs_std),
  .PMP_SUPPORT(1),
  .IMPLEMENTED_PMPS(8),
  .IMPLEMENTED_COUNTERS(0),
  .RESET_PC(32’h10040))
```

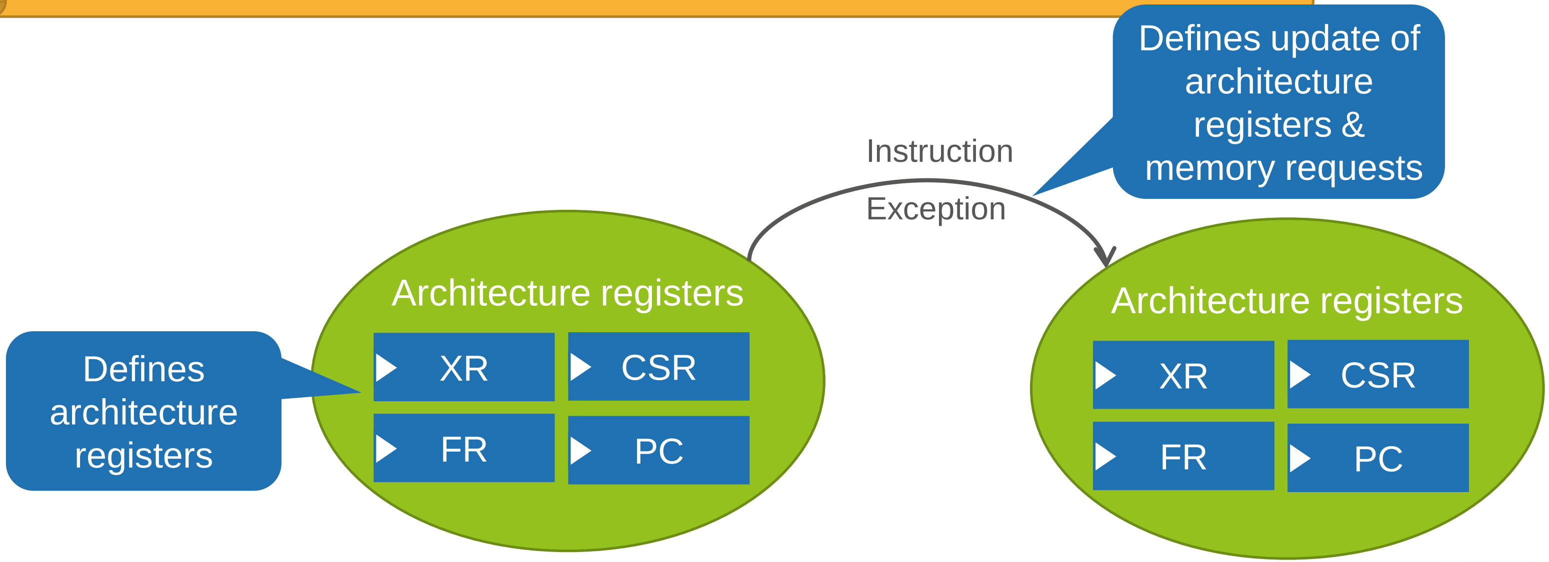
green: provided by App
blue : provided by user

RISC-V ISA Description

LW

31 20	19 15	14 12	11 7	6 0
imm	RS1	010	RD	0000011

```
op_a   = XR[RS1]
addr   = op_a + imm
result = M[addr]
XR[RD] = result
```



 |

Formalized User-Level ISA

onespin

- Captures effect of instructions on architecture state and output to data memory
- Formalized in SystemVerilog Assertions(SVA)

ISA formalization excerpt for LW

```
32'bXXXXXXXXXXXXXXXXX010XXXXX0000011:
  decode.instr      = LW;
  decode.RS1.valid = 1'b1;
  decode.RD.valid  = 1'b1;
  decode.imm        = $signed(iw[31:20]);
  decode.mem        = 1'b1;
…
```


Verification of RISC-V Implementation

- Instructions executed as specified in ISA
  - Example: **Operational SVA** for LW instruction fully verifying forwarding to decode/execute and full register update

```
t##0 Ready2Execute and
t##0 set_freeze(dec,decode(ibuf_io_inst_0_bits_raw,RF)) and
t##0 ibuf_io_inst_0_valid && dec.instr == LW &&
     !dec.xcpt.valid && !ctrl_stalld

implies

t##1 Ready2Execute and
pipe_result(dec,RF,result) and
pipe_dmem_out(dec);
```

Use ISA formalization

Overlapping instructions

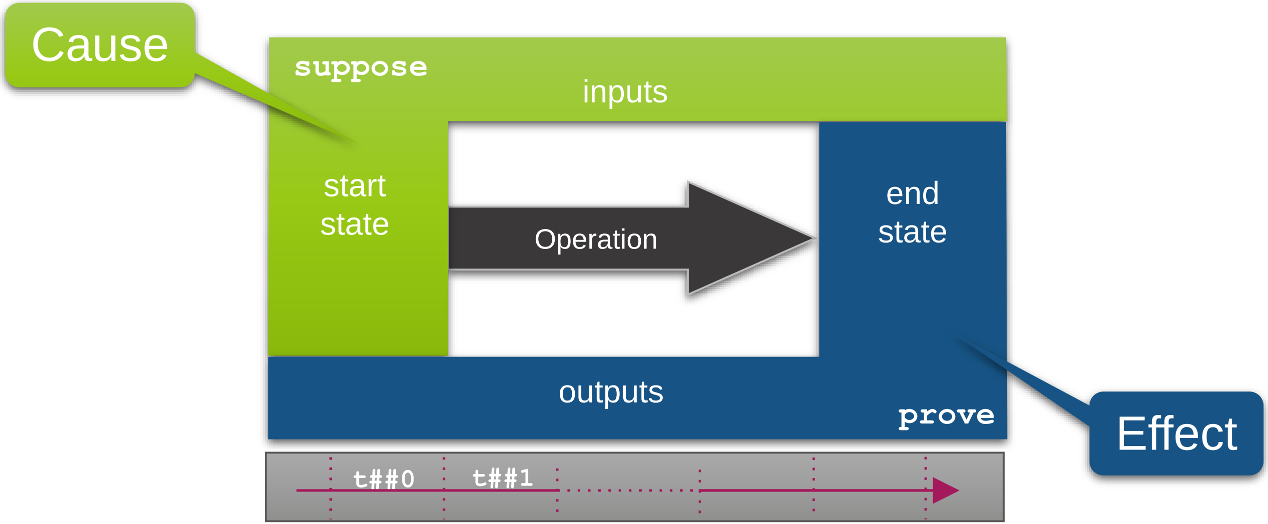
Non-exceptional execution of LW

Check expected register file and DCache request from ISA

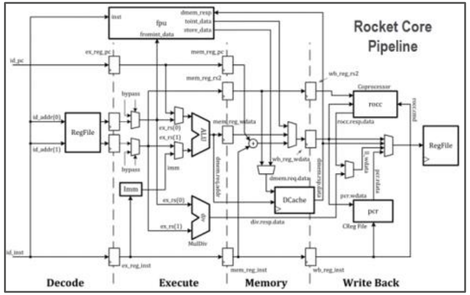
  - Several opcodes can be handled in same property
  - Exceptions, bubbles, and replay handled in separate properties

Operational SVAssertion

- Formally captures single DUV operation
  - Suppose part describes cause – when does assertion apply
  - Prove part specifies effect - intended behavior in that case



Selection of Issues Found in Rocket Core



- DIV (divide) result not written to register file (#1752)
  - Issue confirmed by Rocket Core developers and fixed in RTL
- Illegal opcodes replayed (#1861)
  - Illegal opcodes or fetch side exceptions can cause spurious memory access
- Core contains undocumented non-standard instruction (#1868)
  - Issue confirmed by Rocket Core developers and fixed in RTL (misa.X bit set)
- Core contains undocumented non-standard CSR (#1949)
  - CSR 0x7c1 reads back as 0
- Return from debug mode is executable outside of debug mode (#2022)
  - Issue confirmed by Rocket Core developers and fixed in RTL

Selection of Issues Found in RI5CY Core

- Fetch side exception influences execution of earlier instruction (#132)
  - Earlier instruction executes as if exception of later instruction had already been taken
- Missing illegal exceptions (#136, #137, #170)
  - Several scenarios specified as raising illegal exceptions do not
- Wrong physical memory protection (PMP) computation (#159)
  - PMPs beyond first match are looked up to check for legal access
- Wrong rounding mode for F extension (#169, #174)
  - Updates of rounding mode register not visible for next instruction
- Update of interrupt enable by exception violates spec (#182)
  - Applies both to entering and return from exception

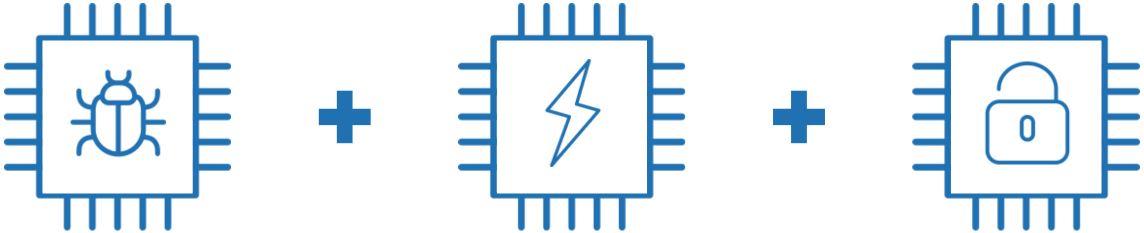
Summary

- RISC-V formal verification methodology proves DUT against RISC-V spec
  - Compliance without hidden additional instructions or registers
  - Finds ISA violations, other functional correctness bugs, and security and trust issues
- App setup in 3 steps
  - Configure app to implemented RISC-V ISA extensions
  - Automatic extraction of architecture registers from DUT
  - Adaption of templates to concrete pipeline implementation and cache interfaces
- Developed on RocketCore standard configuration
  - Exhaustive proofs for core achieved for all instructions on 64-bit pipeline with out-of-order completion in 2-hour sequential runtimes
- Also run on RI5CY 32-bit core
  - 13 issues (so far) reported to core development team

OneSpin for IC Integrity

Visit https://www.onespin.com

Functional Correctness + **Safety** + **Trust and Security**



Thank You!

OneSpin provides certified **IC Integrity Verification Solutions** to develop reliable, safe, secure, and trusted integrated circuits.