Democratising Formal Verification of RISC-V Processors

Dr Ashish Darbari
Founder & CEO
About me

Founder & CEO, Axiomise

Global citizen – Lived, studied & worked in 3 continents and 5 countries

Author of the Axiomise RISC-V ISA formal proof kit™

Previously worked at OneSpin Solutions, Imagination Technologies, General Motors, Arm, Intel

Doctorate, Computer Science, University of Oxford [Formal Verification]

Visiting Professor at University of Southampton, 2015-2018

Theorem proving, Property checking, Equivalence checking (20+ years)

24 US & UK patents in formal verification
Axiomise vision

Formal 2025

Enable formal to be used by all DV engineers

Make formal predictable

We don’t want to lock you in any specific tool

Democratically choose whatever tool you like

Nearly 75 years since the first computer generated proof
How Axiomise is Driving Change?
About Axiomise

Enabling Formal for all Design Verification Engineers

Training
we teach what we have learnt in the field

Consulting
ensure you can transfer expertise from training to your projects

Services
when you don’t have enough resources, our team will help

Proof Kits
accelerating predicable and scalable formal verification
Parameterised NoC

We verify designs with billions of gates for functional verification not connectivity

I ❤️ FORMAL

338 million flops, 1.1 Billion gates, 100.2 sec to find bugs!

Methodology is the key
Open source RISC-V has changed everything!

CPU design is getting democratic

But what about high-quality verification?
The V & V Challenges

Validation and Verification

Validation & Verification consumes nearly 70% time

Why will RISC-V be any different?

Who is going to verify these processors?

How will these be verified?

How much time will it take to verify these?

Who ensures that bugs do not reach the silicon?
Verification challenges – I

Architectural and micro-architectural

Architectural

All instructions in the ISA work as expected

Only those instructions that are part of the ISA are implemented, no more

Micro-architectural

Design optimizations do not introduce bugs

No functional bugs, no deadlocks, livelocks, deadcode, redundant code, underflow, overflow
Verification challenges – II

*When security and safety add further challenges*

**X-checks**
- No reachable X on the output pins
- X propagation should not change the functional behaviour

**Safety**
- No systematic bugs
- Two CPU cores can run in lockstep mode
- Random fault propagation & detection

**Security**
- No malicious code (instructions) can be executed, intentionally or unintentionally

**Power**
- No unnecessary, dynamic power consumption
Verification concerns

When we think about architecture and micro-architecture

Architectural

Micro-architectural

Deadlock

Power

X-propagation

Security

Lockstep verification
Axiomise and RISC-V

Focus on real designs

We worked on several cores designed by the PULP platform group

We looked at zeroriscy and microriscy initially

ibex – currently under development at lowRISC (Cambridge)
Zeroriscy and ibex

*Almost twins*

Low-power

Optimized for arithmetic-control

Implements RV32IC ISA

Two pipelined stages – IF & IDE

ALU: One 32-bit adder, 32-bit shifter, and logic unit

- The adder computes addition, subtraction, comparison
- Shared with address generation unit, branch engine and divider
- Branch, multiplication, division, LSU instructions are computed iteratively

Ibex – a near clone of zeroriscy, error signals introduced on the LSU side
Specification & Verification

Finding relevant information in a haystack

Verification without specifications is pointless

Okay what information?

I don’t know
What did we do?

RISC-V Verification
ADEPT-FV®: An agile flow for using formal

First blue-print in industry for rolling out formal
ADEPT-FV®: An agile flow for using formal

First blue-print in industry for rolling out formal

Deadlocks
Redundant code found
Autocheck issues
Stability issues on interfaces

How do we go about ERASE & PROVE?
Our solution

Going beyond functional verification – architectural & micro-architectural

Use RISC-V ISA to build architectural properties

- Functional correctness, security, safety, power

Build automated flows & checks to verify

- Deadlocks
- X-propagation
- Lockstep

Micro-architectural checks on

- Interfaces (check protocol)
- FSM transitions, livelocks
- Power

MUST ENSURE

Vendor neutral

Easy-to-use

Predictable

Scalable
Explore architectural correctness

Holy grail for formal!

- RISC-V CPU Design
- Formal Tool
- Axiomise ISA Formal Proof Kit™
- Setup File
- RISC-V ISA
- Bugs
- Proof of Bug Absence


ISA formal properties

**BEQ**

70 main properties (so far), over 900 invariants (aux checks)

Each property checks for one instruction exceptions being LOAD/STORES, JAL/JALR

Cause/Effect Model – “if a then b”

```markdown
assert property ($rise(is_a_beq) \&\& (axiomise_REG_rs1==axiomise_REG_rs2)

|->

## `REG_DELAY axiomise_n_pc==branch_addr);
assign branch_addr = axiomise_decoded_pc +

{19{axiomise_instr[31]}},
axiomise_instr[31],
axiomise_instr[7],
axiomise_instr[30:25],
axiomise_instr[11:8], 1'b0};
assign is_a_beq = is_a_branch \&\& axiomise_instr[14:12]==3'b000;
assign is_a_branch = axiomise_instr[6:0] == AXIOMISE_OPCODE_BRANCH;
```
Memory abstraction

For Load-Store units

FIGURE TAKEN FROM ZERORISCY USER MANUAL
Exercise your democratic right

*Our solution works with the best tools in the market*
Just Axiomise it!

One setup file and one command

[ashish@shell] axiomise_riscv ibex_core ~/ibex mentor
Features of our ISA formal proof kit™

Easy-to-use, powerful performance, works with all major tools

Use a tool you like
Processors verified against their ISA
Catch bugs
Prove bug absence
Demonstrate full ISA compliance
User does not write a single line of verification code

Axiomise formal verification methodology however remains a key
Proof kit in action
50% proven in less than half hour!

<table>
<thead>
<tr>
<th>Name</th>
<th>Groups</th>
<th>Health</th>
<th>Radius</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>u_isa.avionise_ISA_SRLI</td>
<td>ISA</td>
<td>★</td>
<td>2</td>
<td>2m 26s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_ADDI</td>
<td>ISA</td>
<td>★</td>
<td>2</td>
<td>2m 46s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SRL</td>
<td>ISA</td>
<td>★</td>
<td>2</td>
<td>2m 46s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_JALR</td>
<td>ISA</td>
<td>★</td>
<td>2</td>
<td>2m 46s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_XORI</td>
<td>ISA</td>
<td>★</td>
<td>2</td>
<td>2m 46s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_BEQ</td>
<td>ISA</td>
<td>★</td>
<td>2</td>
<td>2m 53s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_BNE</td>
<td>ISA</td>
<td>★</td>
<td>2</td>
<td>2m 59s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SLTSI_SET_TO_1</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 0s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_BGEU</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 6s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SLTSI_SET_TO_0</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 6s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_XOR</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 7s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SLTU_SET_TO_1</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 13s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_OR</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 13s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_ORI</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 19s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SLTU_SET_TO_0</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 19s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_ANDI</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 25s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SLU</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 25s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SLTS_SET_TO_1</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 27s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SLTU_SET_TO_1</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 27s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_BLTS</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 32s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SRAI</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 32s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SLTS_SET_TO_0</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 33s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SLTU_SET_TO_0</td>
<td>ISA</td>
<td>★</td>
<td>3</td>
<td>3m 33s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_JAL</td>
<td>ISA</td>
<td>★</td>
<td>4</td>
<td>4m 26s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_LUI</td>
<td>ISA</td>
<td>★</td>
<td>4</td>
<td>4m 32s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_BGES</td>
<td>ISA</td>
<td>★</td>
<td>4</td>
<td>4m 45s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SRA</td>
<td>ISA</td>
<td>★</td>
<td>4</td>
<td>4m 45s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_BLTU</td>
<td>ISA</td>
<td>★</td>
<td>4</td>
<td>4m 52s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_ADD</td>
<td>ISA</td>
<td>★</td>
<td>4</td>
<td>4m 53s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_AND</td>
<td>ISA</td>
<td>★</td>
<td>4</td>
<td>4m 58s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SUB</td>
<td>ISA</td>
<td>0</td>
<td>5m</td>
<td>5m 6s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_SLL</td>
<td>ISA</td>
<td>★</td>
<td>5m</td>
<td>5m 31s</td>
</tr>
<tr>
<td>u_isa.avionise_ISA_JAL_rt_address</td>
<td>ISA</td>
<td>★</td>
<td>24m</td>
<td>24m 13s</td>
</tr>
</tbody>
</table>

~50% exhaustively proven in less than 30 min!
Proof kit in action

80% proven in a couple of hours!

80% exhaustively proven in approx. 2 hrs!

First set of 23 LSU checks finish off in time it takes to have a luxury lunch!

The last set of 14 checks for LSU take more than 24 hours

<table>
<thead>
<tr>
<th>Name</th>
<th>Groups</th>
<th>Health</th>
<th>Radius</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 0</td>
<td></td>
<td>2s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 7</td>
<td>⭐️ 10</td>
<td></td>
<td>20m 28s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 10</td>
<td></td>
<td>23m 56s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 7</td>
<td>⭐️ 12</td>
<td></td>
<td>30m 51s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 10</td>
<td></td>
<td>38m 23s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 10</td>
<td></td>
<td>40m 19s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 7</td>
<td>⭐️ 10</td>
<td></td>
<td>40m 52s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 10</td>
<td></td>
<td>41m 37s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 7</td>
<td>⭐️ 10</td>
<td></td>
<td>41m 38s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 10</td>
<td></td>
<td>1h 6m 58s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 7</td>
<td>⭐️ 10</td>
<td></td>
<td>1h 9m 49s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 10</td>
<td></td>
<td>1h 10m 36s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 7</td>
<td>⭐️ 10</td>
<td></td>
<td>1h 12m 44s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 10</td>
<td></td>
<td>1h 13m 14s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 7</td>
<td>⭐️ 10</td>
<td></td>
<td>1h 14m 58s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 10</td>
<td></td>
<td>1h 16m 39s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 7</td>
<td>⭐️ 10</td>
<td></td>
<td>1h 19m 31s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 0</td>
<td></td>
<td>1h 20m 5s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 7</td>
<td>⭐️ 10</td>
<td></td>
<td>1h 43m 33s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 10</td>
<td></td>
<td>1h 47m 35s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 7</td>
<td>⭐️ 10</td>
<td></td>
<td>1h 48m 43s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 2</td>
<td>⭐️ 0</td>
<td></td>
<td>1h 55m 34s</td>
</tr>
<tr>
<td>LSU</td>
<td>▶️ 7</td>
<td>⭐️ 10</td>
<td></td>
<td>2h 10m 39s</td>
</tr>
</tbody>
</table>
Results

One proof kit, several cores being verified

Our ISA Formal Proof Kit™
Verified several pre-verified RISC-V processors

Found 90+ bugs in zeroriscy
Found 60+ bugs in microriscy
Found 65 bugs in ibex/lowRISC [so far]

Proved exhaustively that design conforms to RV32IC subset of ISA
Anatomy of bugs

Corner case bugs
Bug caused due to incoming debug request on the debug interface when the controller is in the DECODE state.
Nothing in the design to take care of such requests, causing the PC to be not updated correctly.
BEQ bug

Corner-case bug

Only seen when debug arrives and the controller FSM is in the DECODE state.

Precise timing of arrival of debug makes this bug really hard to catch in simulation.

Formal catches it in seconds in 7 cycles!
A family of corner-case bugs

ibex

All instructions were affected other than 3 (SLLI, SRAI, SRLI)

By preventing debug to arrive at all, all properties prove, no bugs seen

By controlling the precise arrival of debug, we can mask the bugs
Architectural flaw

zeroriscy

LOAD address computation check fails. One of the operands needed for the address sum is derived from an incorrect source due to the MUX being overridden incorrectly due to a REG-REG load implementation not part of the ISA.

LOAD issued

ADDRESS

CAUSE EXPECTED

clk_

u_isa.RST_N

is_a_load_mem

OPCODE

offset

rs1

data_addr_o

alu_op_b_mux_sel

alu_op_b_mux_sel

DECODERInstrRdata[14:12]

0

0x3

0x0

0x0

0x0

OP_B_IMM

OP_B_REGB_OR_FWD

3'b000

3'b111
What about micro-architecture?

Deadlocks, X-checks, Lockstep, Power, Security
Deadlock

*Can cause a lot of pains later on*

28 failed checks on ibex, and 77 on zeroriscy (when debug was enabled)

40 prove exhaustively on ibex, and 77 on zeroriscy (with debug enabled)

When debug was disabled, all deadlock checks proved on zeroriscy
X-checks

*Can cause safety and security hazards*

Not relied on any specific vendor-based X-checking solution

We see X-checking as intrinsic to functional verification

8 serious violations on ibex were found, several non-serious ones on zeroriscy
Lockstep

Requirement for functional safety

Mandatory requirement for automotive domain

We check that if two copies of a core do not diverge provided

Their inputs are tied together

Several reasons for divergence

Explicit use of X in low-power designs
Non-reset registers
Dangling signals (unintended)

Checks found 8 serious issues with ibex, exhaustive proofs for zerorrisy
Power

*Power is a big deal for embedded processors*

We catch issues when flip-flops toggle un-necessarily.

We targeted handshake payloads such as data, address

“ready_o AND not valid_i implies data and address payload remains stable”

Checks found issues with dynamic power with zerorisicy and ibex
Security

*Crossing the architectural and micro-architectural*

**Architectural**
- Only intended instructions are implemented
  - Intention matches the ISA
  - No un-intended instruction is implemented

**Micro-architectural**
- All intended micro-architectural implementation is as expected
- Check
  - No underflow, overflow for counters, FIFOs
  - No un-intended X-propagation
  - No un-used reg/wires possible to create any un-intended bypass
  - Reset registers remain reset, and non-reset registers can only be read once they’ve been written

One serious violation found on zeroriscy, several on ibex
Lockstep failures

Micro-architecture
Lockstep failure

Data_wdata_o failure

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_req_o</td>
<td>output</td>
<td>Request ready, must stay high until data_gnt_i is high for one cycle</td>
</tr>
<tr>
<td>data_addr_o[31:0]</td>
<td>output</td>
<td>Address</td>
</tr>
<tr>
<td>data_we_o</td>
<td>output</td>
<td>Write Enable, high for writes, low for reads. Sent together with data_req_o</td>
</tr>
<tr>
<td>data_be_o[3:0]</td>
<td>output</td>
<td>Byte Enable. Is set for the bytes to write/read, sent together with data_req_o</td>
</tr>
<tr>
<td>data_wdata_o[31:0]</td>
<td>output</td>
<td>Data to be written to memory, sent together with data_req_o</td>
</tr>
<tr>
<td>data_rdata_i[31:0]</td>
<td>input</td>
<td>Data read from memory</td>
</tr>
<tr>
<td>data_rvalid_i</td>
<td>input</td>
<td>data_rdata_i holds valid data when data_rvalid_i is high. This signal will be high for exactly one cycle per request.</td>
</tr>
<tr>
<td>data_gnt_i</td>
<td>input</td>
<td>The other side accepted the request. data_addr_o may change in the next cycle</td>
</tr>
</tbody>
</table>

assert property (data_gnt_i |→ data_wdata_o_1 == data_wdata_o_2)
What's the reason?

```c
default: begin
    if (csr_addr_i == CSR_MCUON_SET_UP_MASK) begin
        csr_rdata_int = mhpmevent[mhpmcounter_idx];
        5'h05
        // check access to non-existent or already covered CSRs
        if ((csr_addr[4:0] == 5'b00000) || CSR_MCOUNTINHIBIT)
            if ((csr_addr[4:0] == 5'b00001) ||
                (csr_addr[4:0] == 5'b00010)) begin
                illegal_csr = csr_access_1;
            end
    end
end
```

---

```
// activate all
for (int i=0; i<32; i++) begin : gen_mhpmevent_active
    mhpmevent[1][i] = '1; 1024'h3c3c3c3c3c3c3c3c3c3c3c3c3c3c3c3c3c3c3c3c
end

// deactivate
mhpmevent[1] = '0; // not existing, reserved
for (int i=34; i<36; i++) begin : gen_mhpmevent_inactive
    mhpmevent[1][i] = '0;
end
```
Lockstep on zerorisicy

What do we find?

```verilog
assert property (axiomise_data_wdata_o_1 == axiomise_data_wdata_o_2);
```

<table>
<thead>
<tr>
<th>Assert</th>
<th>axiomise_lockstep.lockstep_instr_req_o</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_instr_addr_o</td>
</tr>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_data_req_o</td>
</tr>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_data_we_o</td>
</tr>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_data_be_o</td>
</tr>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_data_addr_o</td>
</tr>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_data_wdata_o</td>
</tr>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_irq_ack_o</td>
</tr>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_irq_id_o</td>
</tr>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_debug_gnt_o</td>
</tr>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_debug_rvalid_o</td>
</tr>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_debug_rdata_o</td>
</tr>
<tr>
<td></td>
<td>axiomise_lockstep.lockstep_debug_halted_o</td>
</tr>
</tbody>
</table>

```verilog
always_comb
begin
  illegal_instr_o = 1'b0;
instr_o = '0;
end
```
How do we know you are done?

Can I trust you?
Coverage & Completeness

Ensure all requirements are ticked off

- Check that all instruction checks in ISA have been implemented
- Review that Pass/Fail are as expected
- Review constraints
- Review structural coverage
  - Line coverage
  - Toggle coverage
  - Conditional coverage
- Review functional coverage
  - No over-constraint failure (detected by tools)
  - No witness failure for any assertion
  - Tool driven reporting reviewed, understand & fix any gaps
- Hand-based mutation

Most good tools provide you with hooks, and push-button commands to get this data.
Summary

Accelerate predictable formal verification using vendor-neutral Axiomise RISC-V ISA formal proof kit

Built using widely supported open standard of SVA (non-proprietary)

Validate ISA compliance

Find architectural bugs

Find micro-architectural bugs
  - Deadlocks, X-checks, Power, Security, Lockstep (Safety)

Predictable & scalable results
  - Greater than 99% proof convergence seen on different processors

You design, we verify, using the tool of your choice
You design, you verify, using our proof kit and a tool of your choice