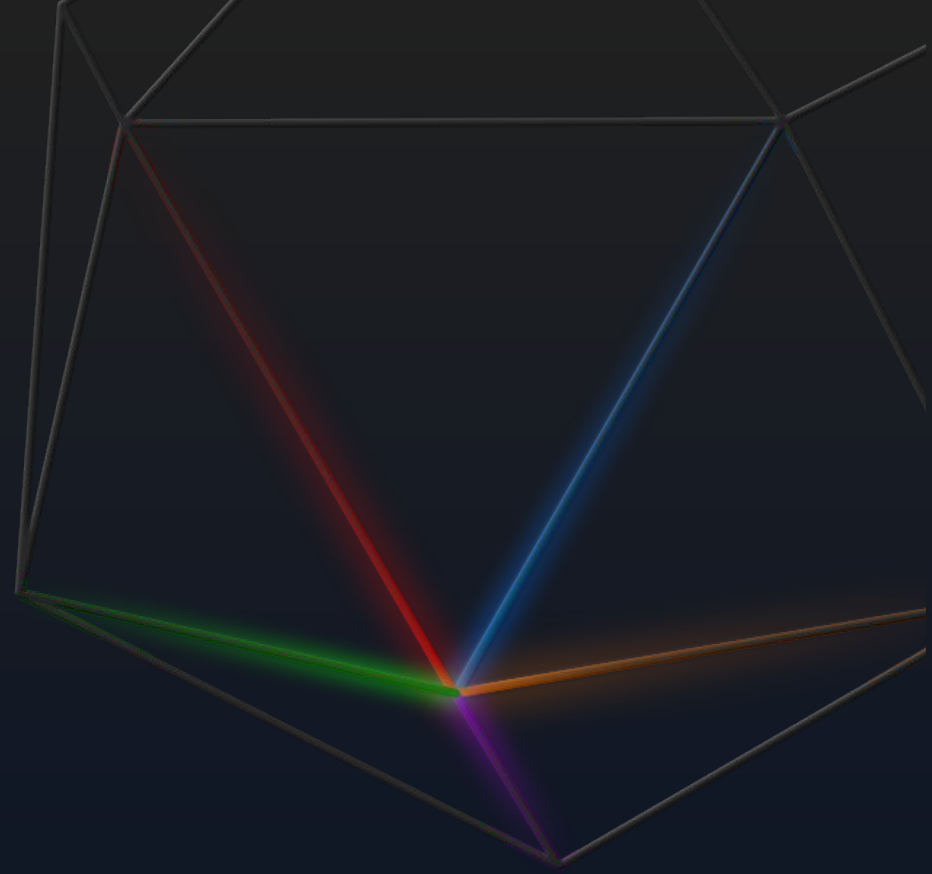
Making RISC-V Market Ready

The Economic Case for Formal Verification

Dr. Ashish Darbari
Founder & CEO
Axiomise



axiomise®
predictable formal verification

Verification trends

Wilson research reports 2022-2024

75%

IC/ASIC projects run behind schedule

60-80%

Overall verification costs

86%

ASICs require two or more respins

83%

FPGA designs with non-trivial bug escapes

62%

Logical/Functional flaws causing re-spins in designs (>1B gates)

10^{30} simulation cycles not finding bugs

axiomise
predictable formal verification

Verification trends

Wilson research reports 2024

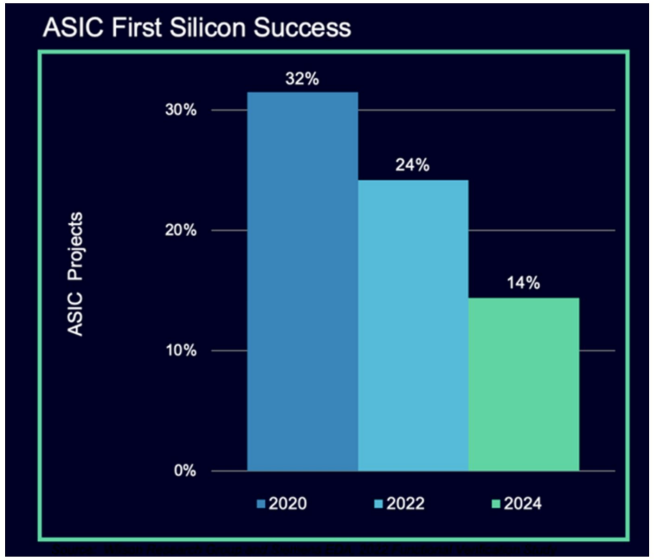


Fig. 1: Number of designs that are functionally correct and manufacturable is declining. Source: Siemens EDA/Wilson Research Group 2024 Functional Verification Study/DVCon

SYSTEMS & DESIGN

First-Time Silicon Success Plummets

564 Shares | 10 | 109 | 161

Number of designs that are late increases. Rapidly rising complexity is the leading cause, but tools, training, and workflows need to improve.

MARCH 27TH, 2025 - BY: **ED SPERLING**

First-time silicon success is falling sharply due to rising complexity, the need for more iterations as chipmakers shift from monolithic chips to multi-die assemblies, and an increasing amount of customization that makes design and verification more time-consuming.

TECHNICAL PAPERS

Scalable And Energy Efficient Solution For Hardware-Based ANNs (KAUST, NUS)

MARCH 30, 2025 **BY TECHNICAL PAPER LINK**

GPU Analysis Identifying Performance Bottlenecks That Cause Throughput Plateaus In Large-Batch Inference

MARCH 30, 2025 **BY TECHNICAL PAPER LINK**

Strategies For Reducing The Effective GaN/Diamond TBR

SPONSORS

SIEMENS · cadence · SYNOPSYS · KEYSIGHT · MOVELLUS · ARTERIS IP · ALPHAWAVE SEMI · axiomise · eliyan · BLUE CHEETAH ANALOG DESIGN

NEWSLETTER SIGNUP

axiomise
predictable formal verification

Formal verification services

Scaling formal for big designs – enabling end-to-end sign-off

The Axiomise team has experience in verifying over 150 designs

- DMA controller
- Multi-threaded processor
- Bus bridges (AXI/CHI/OCP/TileLink)
- Cache sub-systems
- GPU shaders
- I2C/USB/HDMI/I2S
- Network-on-chip
- AI/ML accelerator
- Ethernet Switch
- Mixed-signal
- Low-power
- Power controller

150+

axiomise
predictable formal verification

Why is chip verification hard?

Why bugs escape to silicon?

A holistic approach is missing

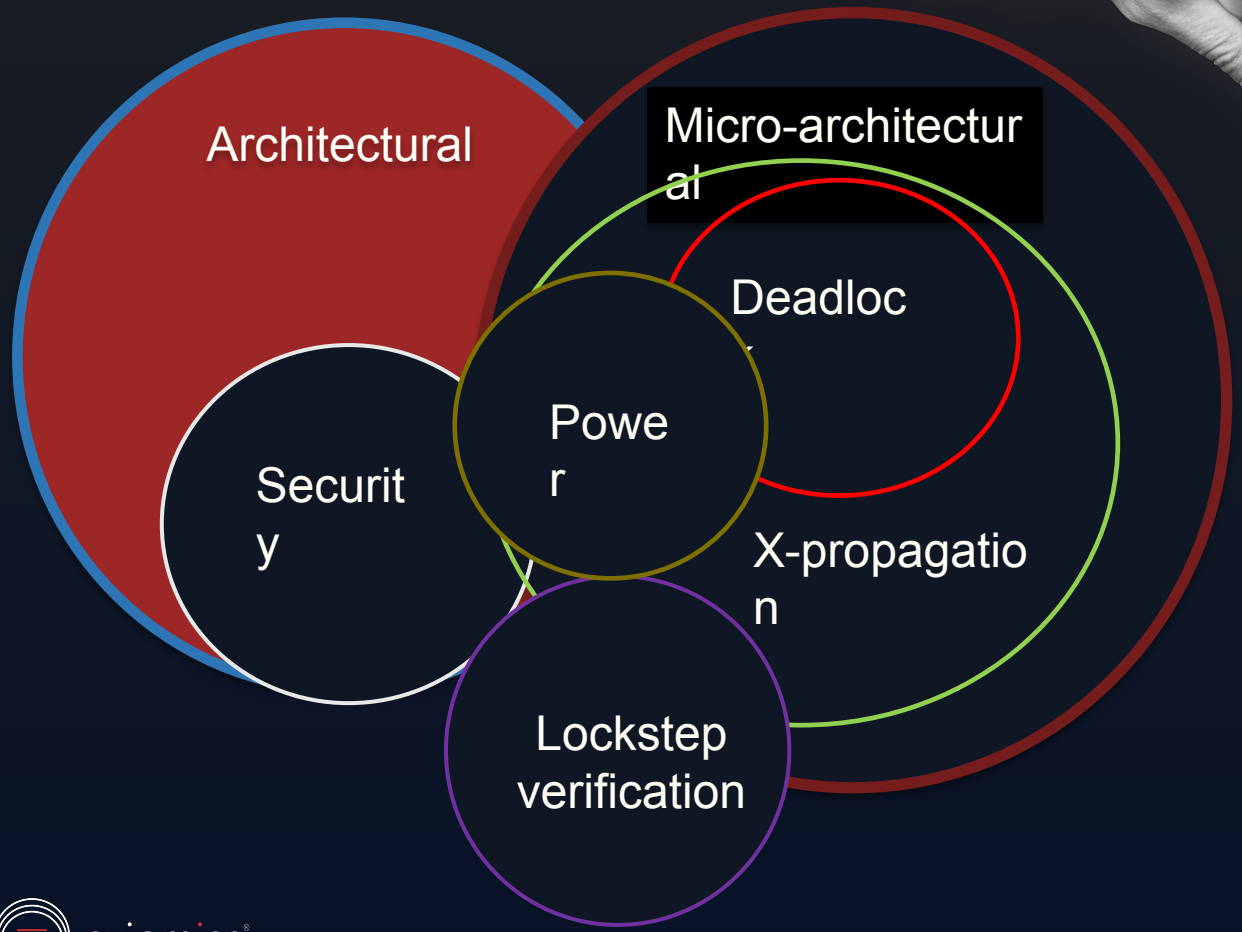
A unifying perspective is missing

ARCHITECTURE

DESIGN/MICROARCHITECTURE

NETLIST

SILICON



axiomise
predictable formal verification

Modern-day processors

Massively optimised

Pipelining **Interlocking** **Forwarding**

Branches **Jumps** **Exceptions**

Stalls **Interrupts** **Debug**

Extensions **Clock gating** **Arithmetic**

Power **Safety** **Security**

axiomise®
predictable formal verification

Complex control and data dependencies

Cores have in-order or out-of-order behaviour?

Branches:

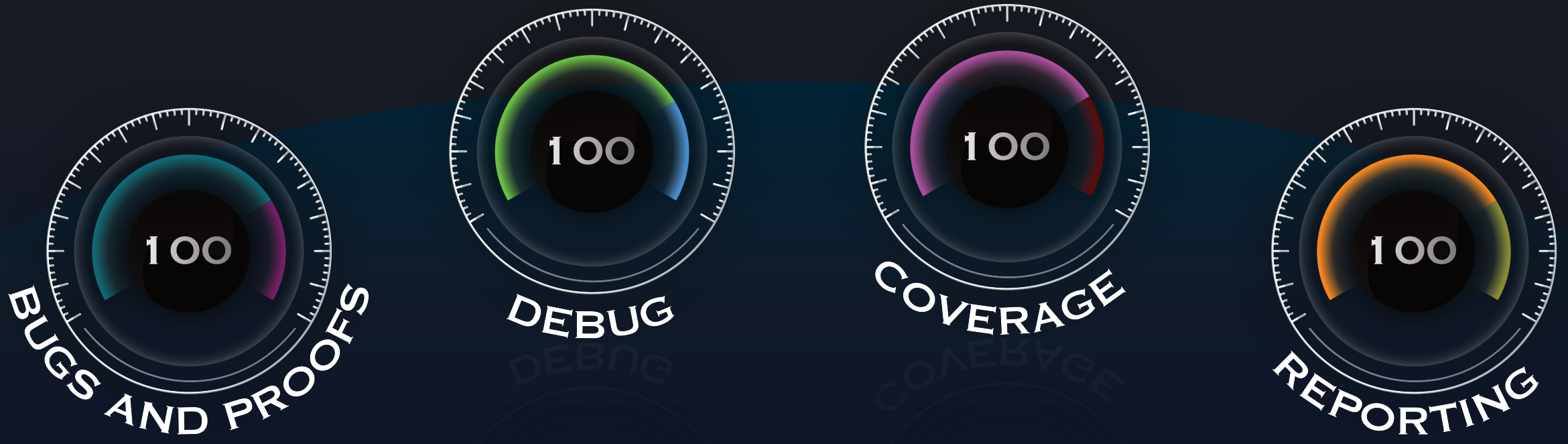
- Speculative branches
- Forward jumps, Backward jumps, Page size jumps, Page boundary jumps, Jumps across pages (same or different pages)

Back-to-back memory operations:

- Cache hits & cache misses
- Write-through stores
- Cache bypasses, atomics and cache coherency

formalISA®

Making formal normal for RISC-V



Accelerating debug and sign-off for custom designs using exhaustive formal

axiomise®
predictable formal verification

Our formal RISC-V solution

Enables adoption of formal methods more widely

1. No test case to write
2. No manual checker to write
3. No verification code to be written
4. Exhaustively prove that all ISA instructions work as expected under all conditions

What goes in our APP?

1. Your RISC-V core
2. Set up file
3. Coverage specification

What comes out?

Exhaustive proofs that “mathematically” prove under all conditions:

- ✓ Each instruction in the ISA works always as expected
- ✓ Scenarios specified in the coverage specification can “always” happen
- ✓ Visualize that scenarios in the coverage specification “can happen”

axiomise®
predictable formal verification

Agile formal verification for end-to-end sign-off

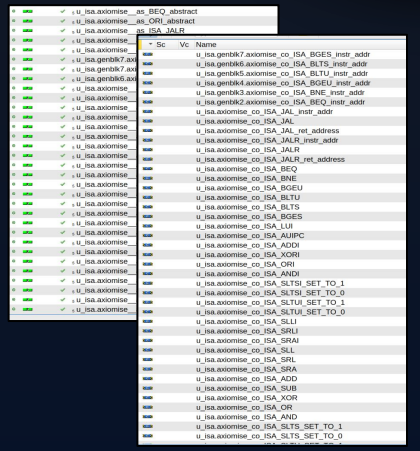
Saving simulation time, obtaining exhaustive proofs, finding corner-case bugs

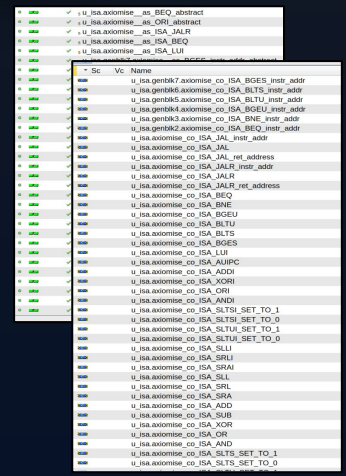
cv32e40p 9 July

cv32e40p 3 Dec

Setup 9 July

Setup 3 Dec

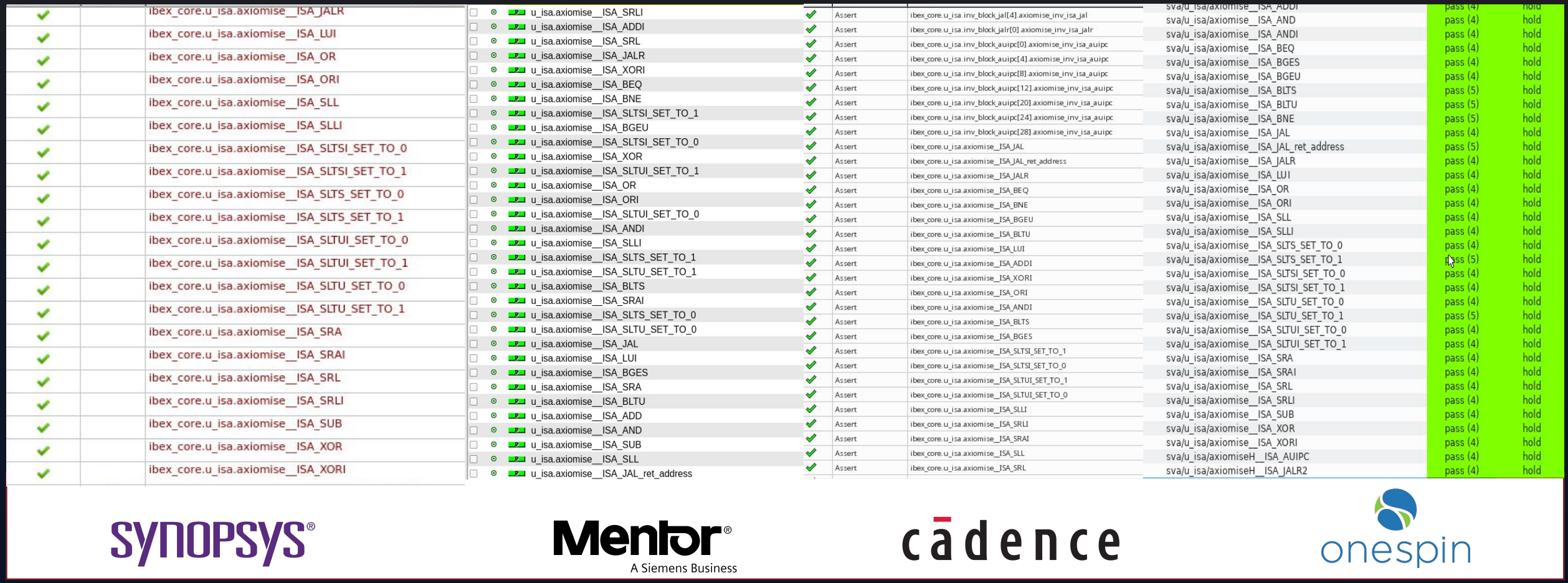




BUGS, PROOFS of BUG ABSENCE & INTER-OPERABLE COVERAGE MODEL

ibex

Complete democracy – use any tool you like



SYNOPSYS®

Mentor® A Siemens Business

cadence

onespin

axiomise® predictable formal verification

Formal verification

Agile bug hunting and proofs of bug absence

https://github.com/openhwgroup/cv32e40p/issues/551

Specification bugs in RISC-V ISA

Inconsistencies in the RISC-V ISA v2.2

Page 30

31 26	25	24 20	19 15	14 12	11 7	6 0
imm[11:6]	imm[5]	imm[4:0]	rs1	funct3	rd	opcode
6	1	5	5	3	5	7
000000	shamt[5]	shamt[4:0]	src	SLLI	dest	OP-IMM
000000	shamt[5]	shamt[4:0]	src	SRLI	dest	OP-IMM
010000	shamt[5]	shamt[4:0]	src	SRAI	dest	OP-IMM
000000	0	shamt[4:0]	src	SLLIW	dest	OP-IMM-32
000000	0	shamt[4:0]	src	SRLIW	dest	OP-IMM-32
010000	0	shamt[4:0]	src	SRAIW	dest	OP-IMM-32

Shifts by a constant are encoded as a specialization of the I-type format using the same instruction opcode as RV32I. The operand to be shifted is in *rs1*, and the shift amount is encoded in the lower 6 bits of the I-immediate field for RV64I. The right shift type is encoded in bit 30. SLLI is a logical left shift (zeros are shifted into the lower bits); SRLI is a logical right shift (zeros are shifted into the upper bits); and SRAI is an arithmetic right shift (the original sign bit is copied into the vacated upper bits). For RV32I, SLLI, SRLI, and SRAI generate an illegal instruction exception if $imm[5] \neq 0$.

Page 104

0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI

axiomise
predictable formal verification

formalISA®

Making formal normal for RISC-V

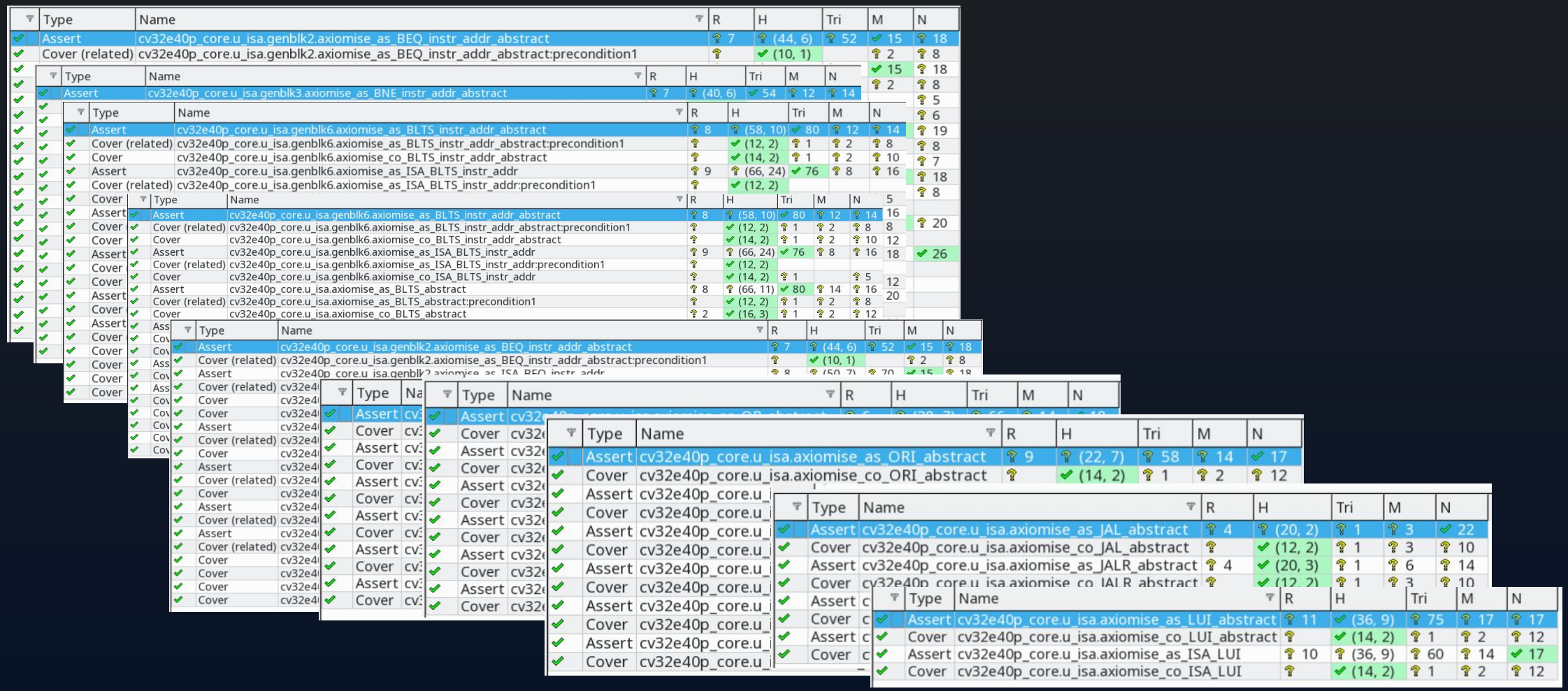
	ibex	zeroriscy	cv32e40p	WARP-V			cheriot-ibex
Pipeline stages	2-stage	2-stage	4-stage	6-stage	4-stage	2-stage	2-stage
No. of issues	65	77	5	30	30	30	6
Previously verified	Yes	Yes	No	Yes	Yes	Yes	Yes
How was it previously verified?	Simulation	Simulation	Simulation & Formal	Formal	Formal	Formal	Simulation & Formal
Time taken to find issues	< 30 seconds	< 30 seconds	< 30 seconds	< 30 seconds	< 30 seconds	< 30 seconds	<1 min
Nature of analysis and issues	Microarchitectural Deadlocks and Architectural	Microarchitectural Deadlocks and Architectural	Architectural	Architectural	Architectural	Architectural	Corner-case bugs
When was the issue found?	2019	2019	2020	2021	2021	2021	2024

axiomise®

predictable formal verification

cv32e40p

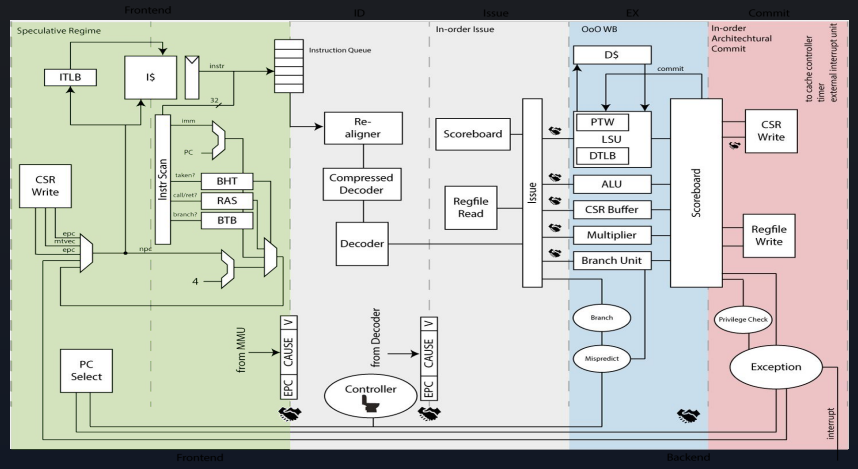
32-bit, 4-stage in-order pipeline



axiomise
predictable formal verification

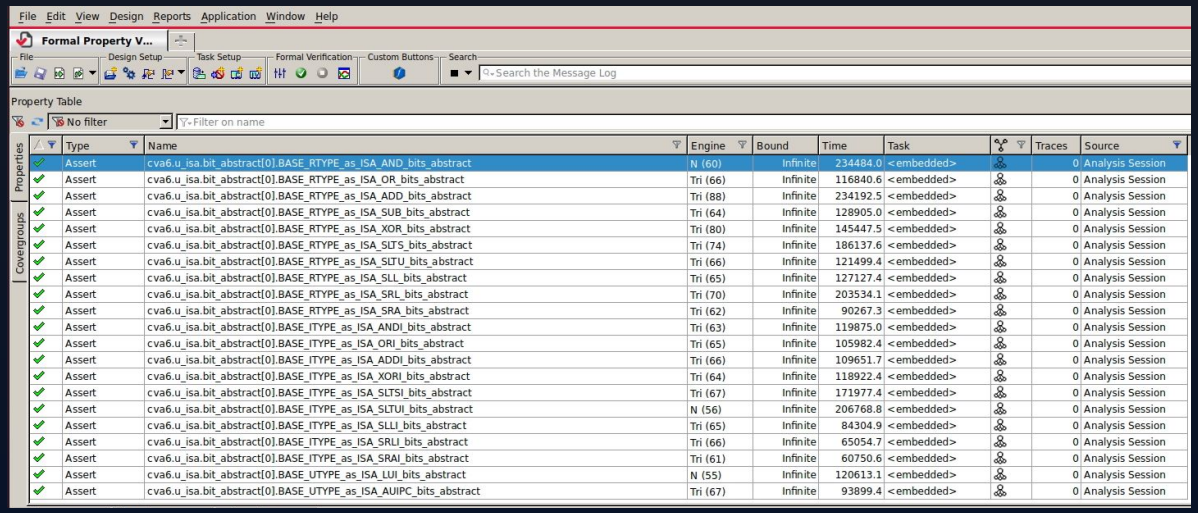
CVA6

64-bit six-stage, in-order issue, out-of-order execution, in-order commit



From the OPENHW Group Page

CVA6 is a 6-stage, single issue, in-order CPU which implements the 64-bit RISC-V instruction set. It fully implements I, M, A and C extensions as specified in Volume I: User-Level ISA V 2.3 as well as the draft privilege extension 1.10. It implements three privilege levels M, S, U to fully support a Unix-like operating system. Furthermore, it is compliant to the draft external debug spec 0.13. It has configurable size, separate TLBs, a hardware PTW and branch-prediction (branch target buffer and branch history table). The primary design goal was on reducing critical path length.

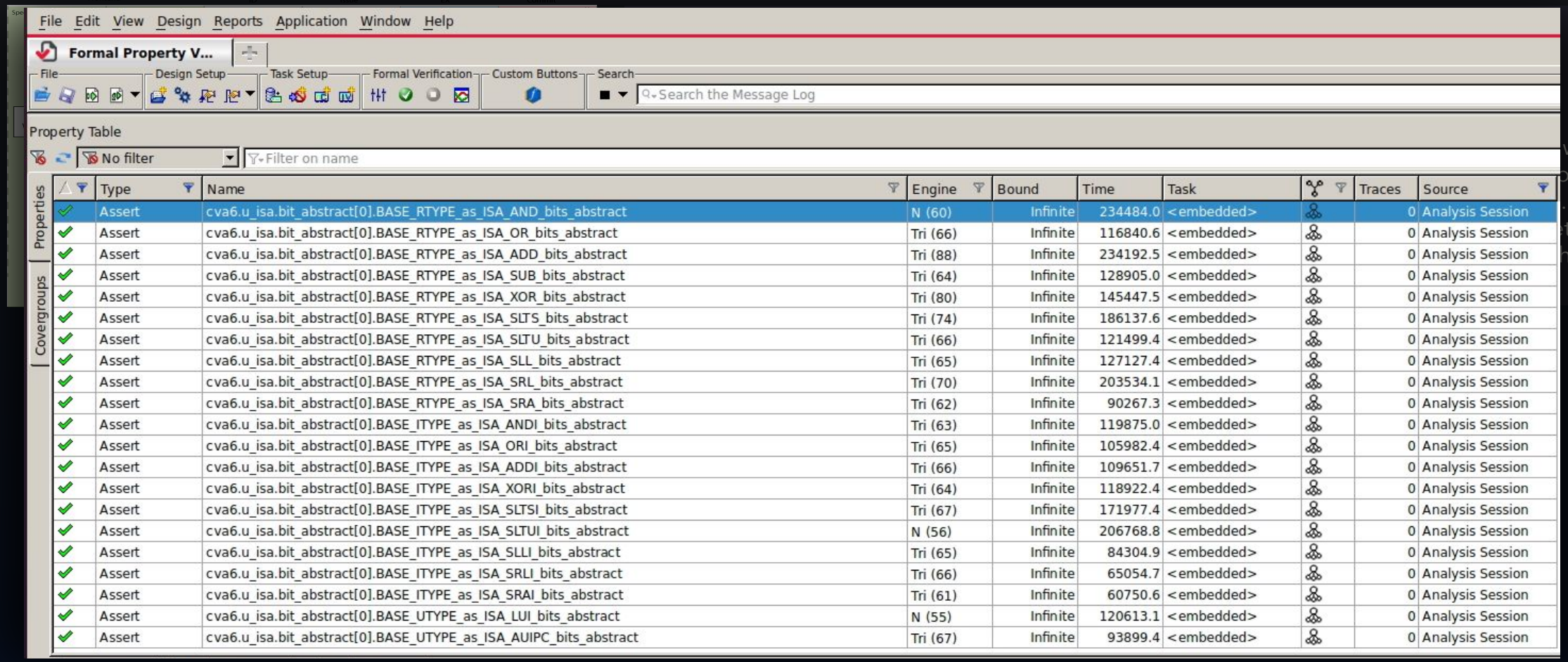


Type	Name	Engine	Bound	Time	Task	Traces	Source
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_AND_bits_abstract	N (60)	Infinite	234484.0	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_OR_bits_abstract	Tri (66)	Infinite	116840.6	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_ADD_bits_abstract	Tri (88)	Infinite	234192.5	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SUB_bits_abstract	Tri (64)	Infinite	128905.0	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_XOR_bits_abstract	Tri (80)	Infinite	145447.5	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SLTS_bits_abstract	Tri (74)	Infinite	186137.6	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SLTU_bits_abstract	Tri (66)	Infinite	121499.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SLL_bits_abstract	Tri (65)	Infinite	127127.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SRL_bits_abstract	Tri (70)	Infinite	203534.1	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SRA_bits_abstract	Tri (62)	Infinite	90267.3	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_ANDI_bits_abstract	Tri (63)	Infinite	119875.0	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_ORI_bits_abstract	Tri (65)	Infinite	105982.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_ADDI_bits_abstract	Tri (66)	Infinite	109651.7	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_XORI_bits_abstract	Tri (64)	Infinite	118922.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SLTSI_bits_abstract	Tri (67)	Infinite	171977.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SLTUI_bits_abstract	N (56)	Infinite	206768.8	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SLLI_bits_abstract	Tri (65)	Infinite	84304.9	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SRLI_bits_abstract	Tri (66)	Infinite	65054.7	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SRAI_bits_abstract	Tri (61)	Infinite	60750.6	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_UTYPE_as_ISA_LUI_bits_abstract	N (55)	Infinite	120613.1	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_UTYPE_as_ISA_AUIPC_bits_abstract	Tri (67)	Infinite	93899.4	<embedded>	0	Analysis Session

axiomise
predictable formal verification

CVA6

64-bit six-stage, in-order issue, out-of-order execution, in-order commit



Type	Name	Engine	Bound	Time	Task	Traces	Source
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_AND_bits_abstract	N (60)	Infinite	234484.0	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_OR_bits_abstract	Tri (66)	Infinite	116840.6	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_ADD_bits_abstract	Tri (88)	Infinite	234192.5	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SUB_bits_abstract	Tri (64)	Infinite	128905.0	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_XOR_bits_abstract	Tri (80)	Infinite	145447.5	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SLTS_bits_abstract	Tri (74)	Infinite	186137.6	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SLTU_bits_abstract	Tri (66)	Infinite	121499.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SLL_bits_abstract	Tri (65)	Infinite	127127.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SRL_bits_abstract	Tri (70)	Infinite	203534.1	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SRA_bits_abstract	Tri (62)	Infinite	90267.3	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_ANDI_bits_abstract	Tri (63)	Infinite	119875.0	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_ORI_bits_abstract	Tri (65)	Infinite	105982.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_ADDI_bits_abstract	Tri (66)	Infinite	109651.7	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_XORI_bits_abstract	Tri (64)	Infinite	118922.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SLTSI_bits_abstract	Tri (67)	Infinite	171977.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SLTUI_bits_abstract	N (56)	Infinite	206768.8	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SLLI_bits_abstract	Tri (65)	Infinite	84304.9	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SRLI_bits_abstract	Tri (66)	Infinite	65054.7	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SRAI_bits_abstract	Tri (61)	Infinite	60750.6	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_UTYPE_as_ISA_LUI_bits_abstract	N (55)	Infinite	120613.1	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_UTYPE_as_ISA_AUIPC_bits_abstract	Tri (67)	Infinite	93899.4	<embedded>	0	Analysis Session

axiomise
predictable formal verification

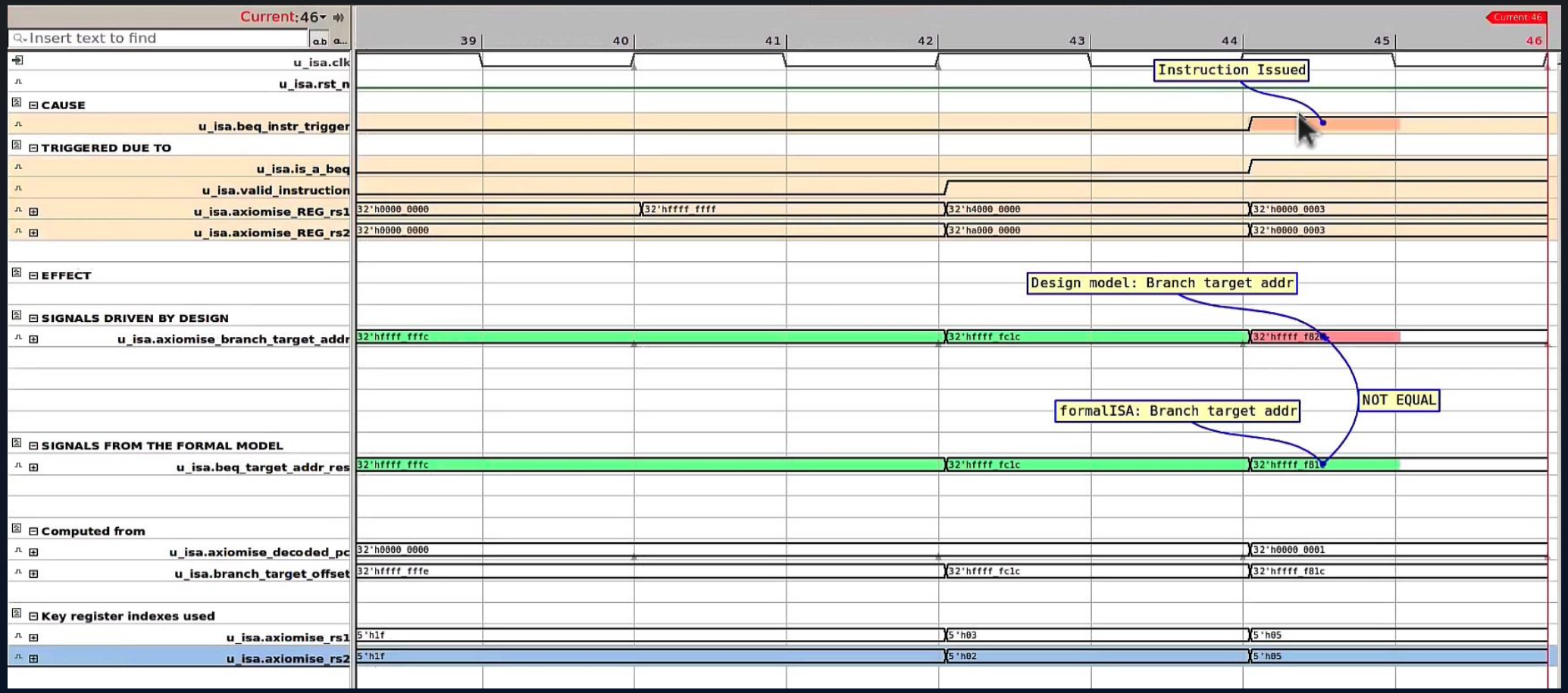
i-RADAR Debug

Intelligent Rapid Analysis Debug and Reporting

axiomise
predictable formal verification

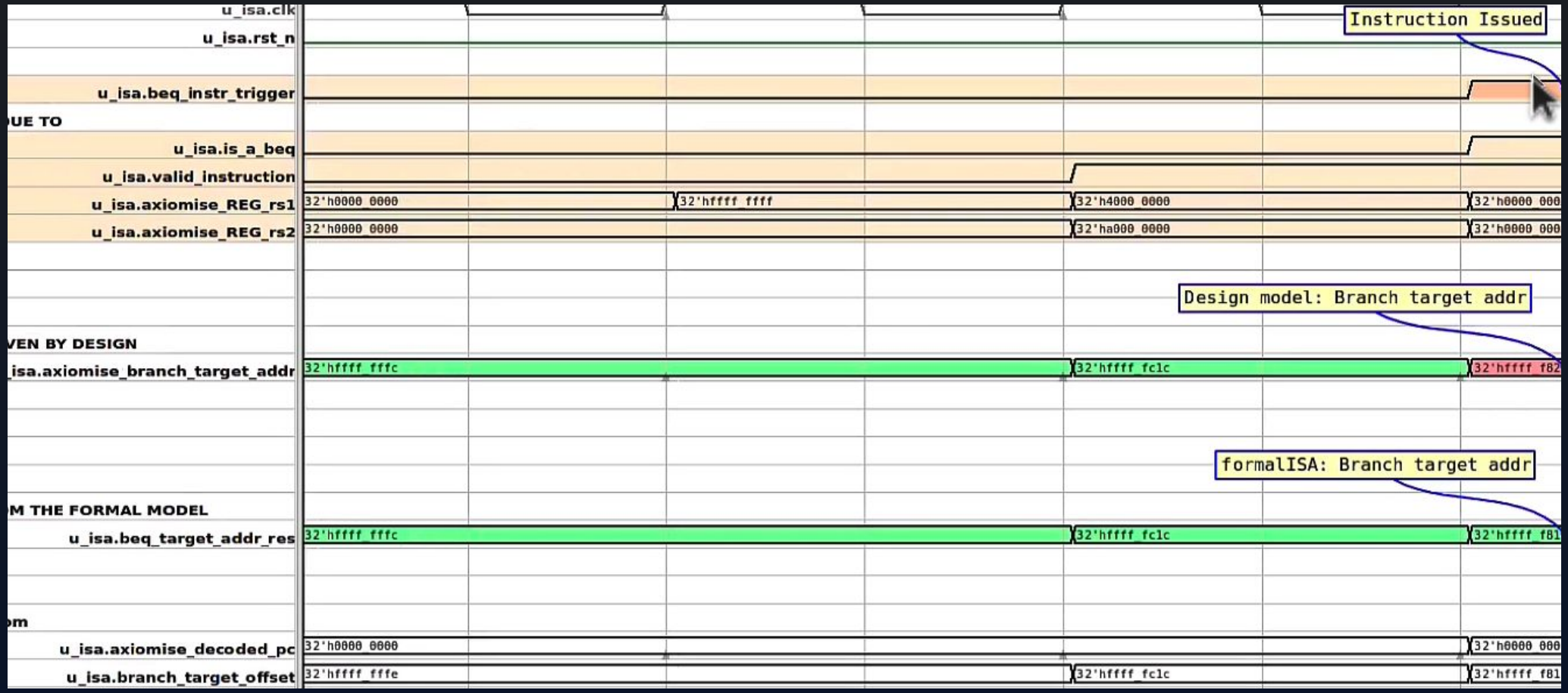
Intelligent debug

Waveforms, reports



Intelligent debug

Waveforms, reports



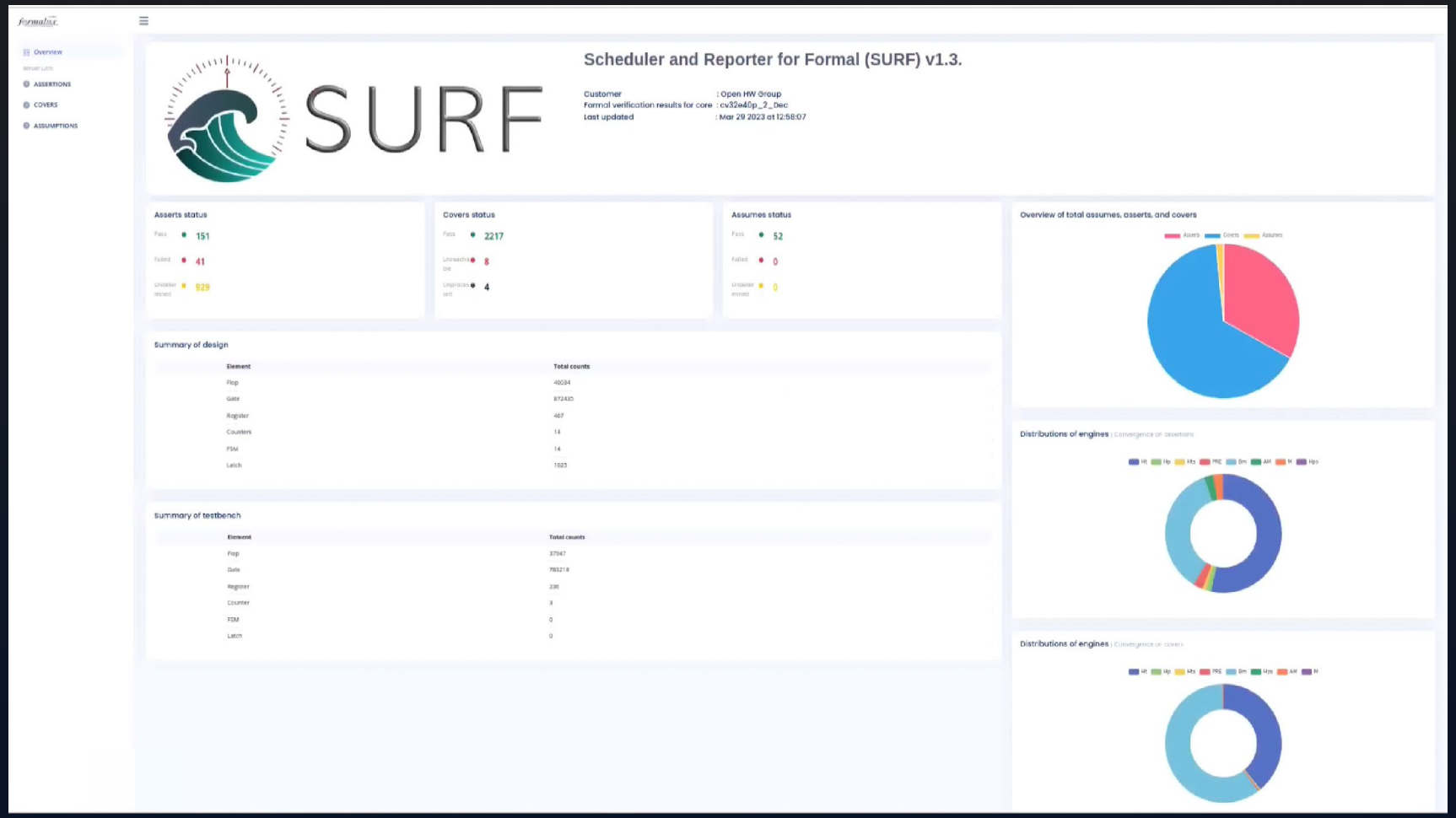
axiomise
predictable formal verification

SURF Reporting

Scheduler and Reporter for Formal

SURF dashboard

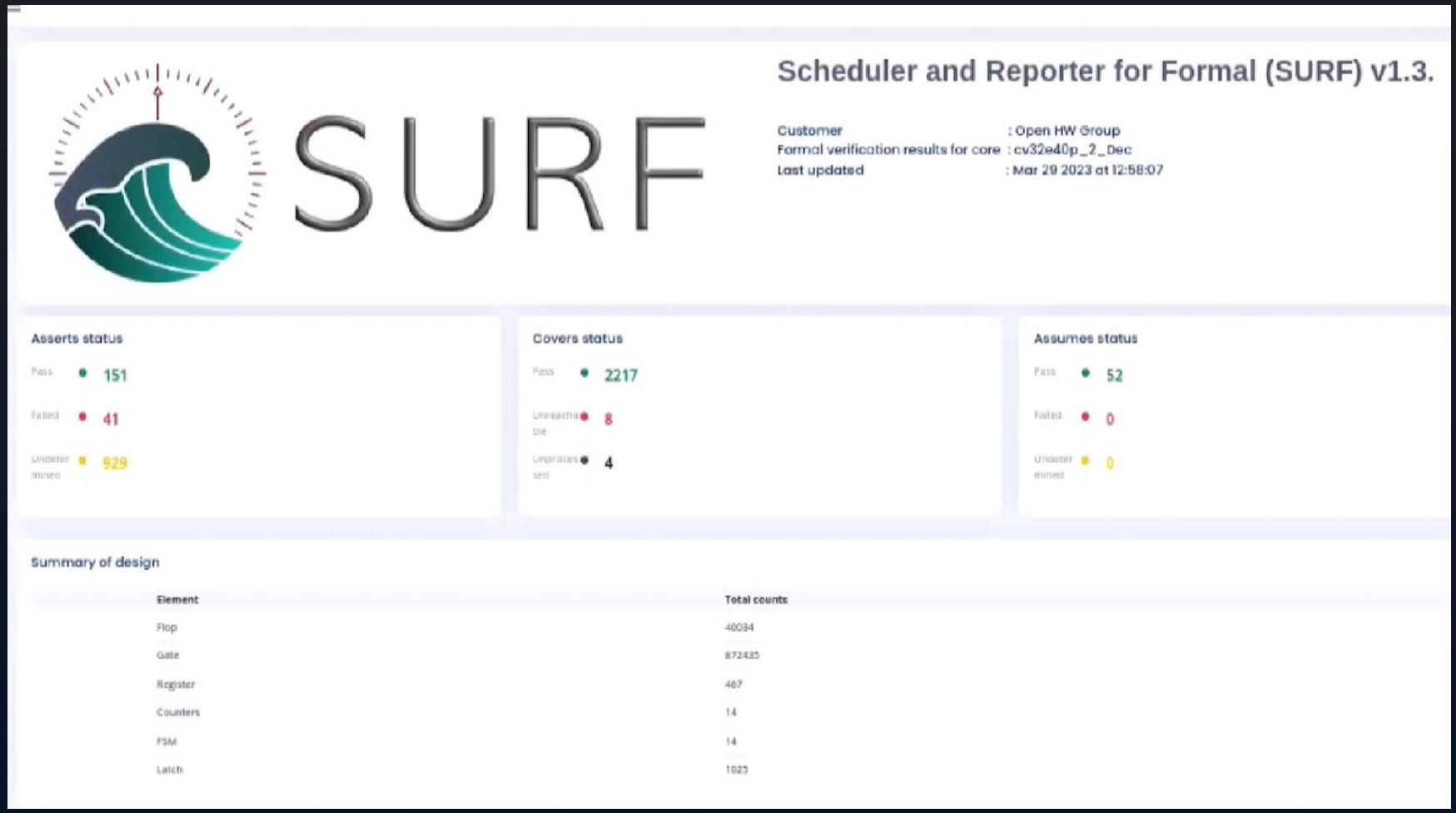
RISC-V



axiomise
predictable formal verification

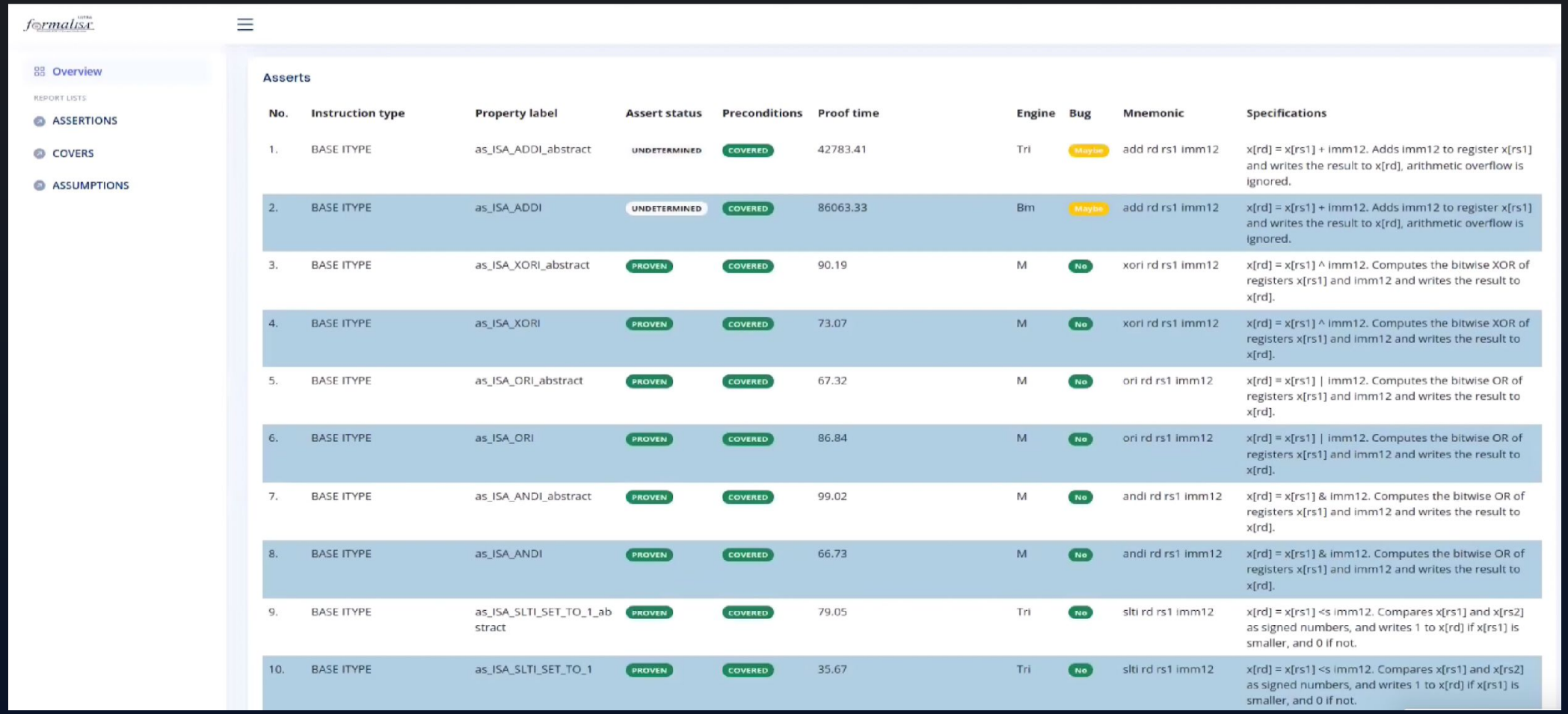
SURF dashboard

RISC-V



SURF dashboard

Example reporting bugs



Asserts

No.	Instruction type	Property label	Assert status	Preconditions	Proof time	Engine	Bug	Mnemonic	Specifications
1.	BASE ITYPE	as_ISA_ADDI_abstract	UNDETERMINED	COVERED	42783.41	Tri	Maybe	add rd rs1 imm12	x[rd] = x[rs1] + imm12. Adds imm12 to register x[rs1] and writes the result to x[rd], arithmetic overflow is ignored.
2.	BASE ITYPE	as_ISA_ADDI	UNDETERMINED	COVERED	86063.33	Bm	Maybe	add rd rs1 imm12	x[rd] = x[rs1] + imm12. Adds imm12 to register x[rs1] and writes the result to x[rd], arithmetic overflow is ignored.
3.	BASE ITYPE	as_ISA_XORI_abstract	PROVEN	COVERED	90.19	M	No	xori rd rs1 imm12	x[rd] = x[rs1] ^ imm12. Computes the bitwise XOR of registers x[rs1] and imm12 and writes the result to x[rd].
4.	BASE ITYPE	as_ISA_XORI	PROVEN	COVERED	73.07	M	No	xori rd rs1 imm12	x[rd] = x[rs1] ^ imm12. Computes the bitwise XOR of registers x[rs1] and imm12 and writes the result to x[rd].
5.	BASE ITYPE	as_ISA_ORI_abstract	PROVEN	COVERED	67.32	M	No	ori rd rs1 imm12	x[rd] = x[rs1] | imm12. Computes the bitwise OR of registers x[rs1] and imm12 and writes the result to x[rd].
6.	BASE ITYPE	as_ISA_ORI	PROVEN	COVERED	86.84	M	No	ori rd rs1 imm12	x[rd] = x[rs1] | imm12. Computes the bitwise OR of registers x[rs1] and imm12 and writes the result to x[rd].
7.	BASE ITYPE	as_ISA_ANDI_abstract	PROVEN	COVERED	99.02	M	No	andi rd rs1 imm12	x[rd] = x[rs1] & imm12. Computes the bitwise OR of registers x[rs1] and imm12 and writes the result to x[rd].
8.	BASE ITYPE	as_ISA_ANDI	PROVEN	COVERED	66.73	M	No	andi rd rs1 imm12	x[rd] = x[rs1] & imm12. Computes the bitwise OR of registers x[rs1] and imm12 and writes the result to x[rd].
9.	BASE ITYPE	as_ISA_SLTI_SET_TO_1_abstract	PROVEN	COVERED	79.05	Tri	No	slti rd rs1 imm12	x[rd] = x[rs1] <s imm12. Compares x[rs1] and x[rs2] as signed numbers, and writes 1 to x[rd] if x[rs1] is smaller, and 0 if not.
10.	BASE ITYPE	as_ISA_SLTI_SET_TO_1	PROVEN	COVERED	35.67	Tri	No	slti rd rs1 imm12	x[rd] = x[rs1] <s imm12. Compares x[rs1] and x[rs2] as signed numbers, and writes 1 to x[rd] if x[rs1] is smaller, and 0 if not.

axiomise
predictable formal verification

SURF dashboard

Example reporting bugs

Asserts

No.	Instruction type	Property label	Assert status	Preconditions	Proof time	Engine	Bug	Mnemonic	Specifications
1.	BASE ITYPE	as_ISA_ADDI_abstract	UNDETERMINED	COVERED	42783.41	Tri	Maybe	add rd rs1 imm12	x[rd] = x[rs1] + imm12. Adds imm12 to register x[rs1] and writes the result to x[rd], arithmetic overflow is ignored.
2.	BASE ITYPE	as_ISA_ADDI	UNDETERMINED	COVERED	86063.33	Bm	Maybe	add rd rs1 imm12	x[rd] = x[rs1] + imm12. Adds imm12 to register x[rs1] and writes the result to x[rd], arithmetic overflow is ignored.
3.	BASE ITYPE	as_ISA_XORI_abstract	PROVEN	COVERED	90.19	M	No	xori rd rs1 imm12	x[rd] = x[rs1] ^ imm12. Computes the bitwise XOR of registers x[rs1] and imm12 and writes the result to x[rd].
4.	BASE ITYPE	as_ISA_XORI	PROVEN	COVERED	73.07	M	No	xori rd rs1 imm12	x[rd] = x[rs1] ^ imm12. Computes the bitwise XOR of registers x[rs1] and imm12 and writes the result to x[rd].
5.	BASE ITYPE	as_ISA_ORI_abstract	PROVEN	COVERED	67.32	M	No	ori rd rs1 imm12	x[rd] = x[rs1] | imm12. Computes the bitwise OR of registers x[rs1] and imm12 and writes the result to x[rd].
6.	BASE ITYPE	as_ISA_ORI	PROVEN	COVERED	86.84	M	No	ori rd rs1 imm12	x[rd] = x[rs1] | imm12. Computes the bitwise OR of registers x[rs1] and imm12 and writes the result to x[rd].
7.	BASE ITYPE	as_ISA_ANDI_abstract	PROVEN	COVERED	99.02	M	No	andi rd rs1 imm12	x[rd] = x[rs1] & imm12. Computes the bitwise OR of registers x[rs1] and imm12 and writes the result to x[rd].
8.	BASE ITYPE	as_ISA_ANDI	PROVEN	COVERED	66.73	M	No	andi rd rs1 imm12	x[rd] = x[rs1] & imm12. Computes the bitwise OR of registers x[rs1] and imm12 and writes the result to x[rd].
9.	BASE ITYPE	as_ISA_SLTI_SET_TO_1_abstract	PROVEN	COVERED	79.05	Tri	No	slti rd rs1 imm12	x[rd] = x[rs1] <s imm12. Compares x[rs1] and x[rs2] as signed numbers, and writes 1 to x[rd] if x[rs1] is smaller, and 0 if not.

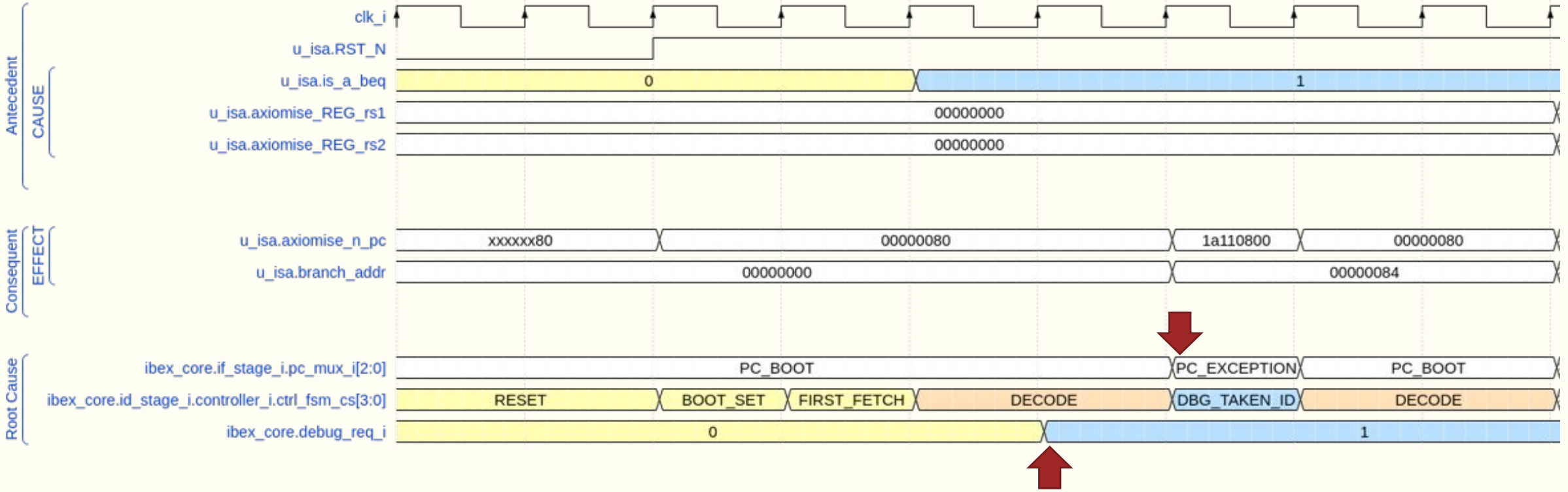
axiomise
predictable formal verification

Anatomy of bugs

Processor bugs caught by *formalISA*

BEQ failure

Functional verification - ibex



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 failure

Functional verification - ibex

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 dynamic simulation.

Formal catches it in seconds in 7 cycles!

Illegal instruction handling

cheriot-ibex: Verified in September 2024

The illegal instruction affected the execution of the valid instruction that followed it.

Issues

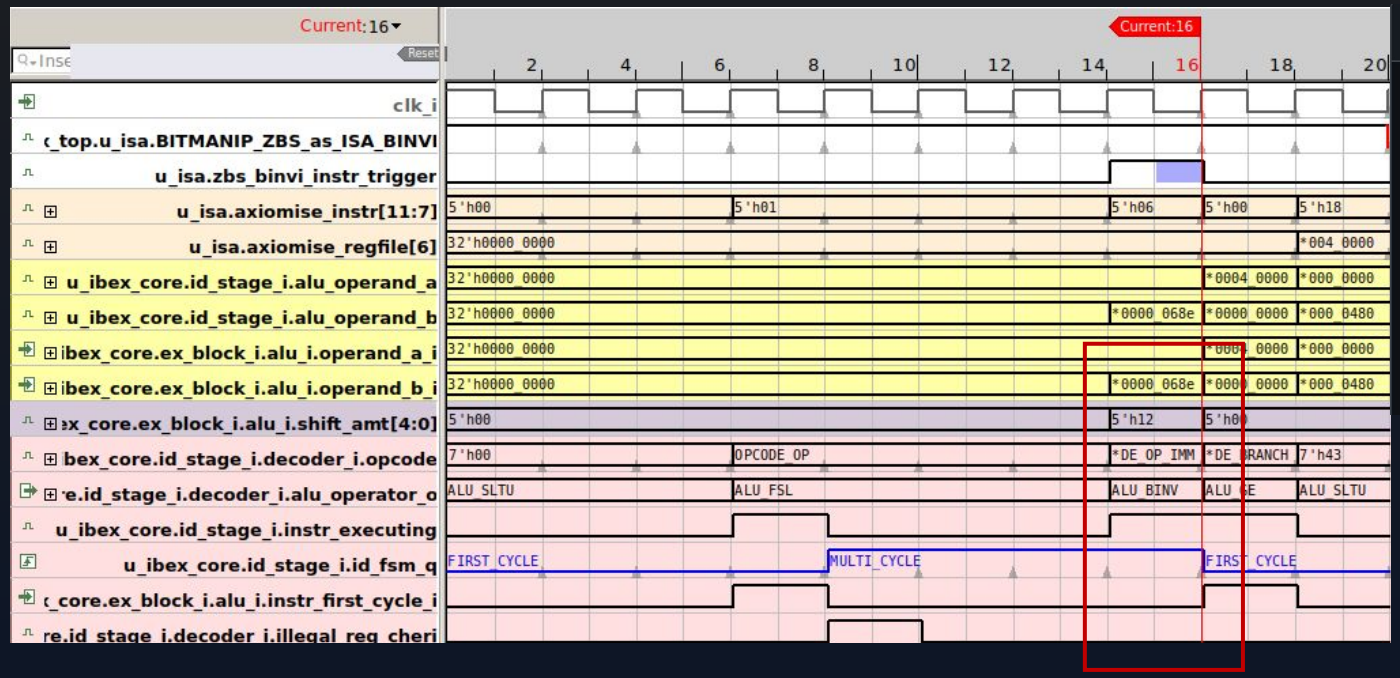
- Sending the illegal instruction request to the memory.
- Wasted execution power.
- Invalid data in the register file and subsequently in memory.

The illegal load instruction affected the execution of the valid AND (or any R-TYPE) instruction that followed it.

axiomise
predictable formal verification

Illegal instruction handling – bit manipulation

After the first bug fix, bit manipulations instructions were broken



kliuMsft on Oct 15, 2024 — Contributor

Looking further into the issue, the culprit seems to be that the id_fsm_d logic can't handle exception being issued in the 2nd half of a multi-cycle instruction. Specifically, the illegal_reg_cheri results in an EX stage exception but instr_kill is only raised in the 2nd half of a bit manipulation instruction (when rs3 is accessed). In this case multicycle_done is never issued and thus id_fsm_q will not updated properly.

@GregAC do you plan to keep supporting the bit instructions with rs3? if so I can try fix the behavior in cheriot-ibex. You may want to take a look at the upstream ibex implementation as well.

https://github.com/microsoft/cheriot-ibex/issues/51

https://github.com/darbaria/axiomise-warpv-formal-6-stage/issue

WARP-V

Six stage pipelined processor with a range of bugs

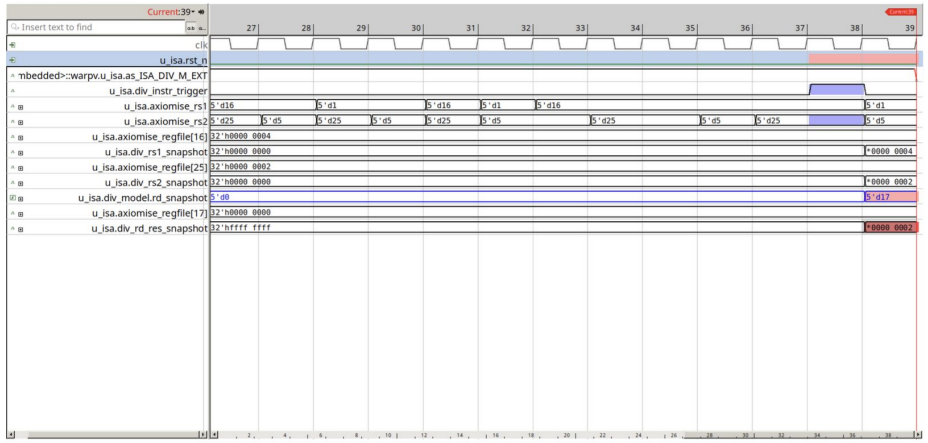
DIV Instruction not working correctly #29

Open **shivanishah269** opened this issue on 6 Jun 2021 · 0 comments

shivanishah269 commented on 6 Jun 2021 — Collaborator

Page 44 of RISC-V ISA mandates "DIV: Divides x[rs1] by x[rs2] rounding towards 0, treating the values as signed numbers and writes the quotient to x[rd]"

Our checker fails showing that the updates did not happen in cycle 38 to the register 17 in response to a prior **div** instruction detected in cycle 37. x[16] is divided by x[25] and rd is 17. We expect x[17] to be 2 as x[16] is 4 and x[25] is 2, but it isn't.



Assignees: stevehoover, shivanishah269

Labels: bug

Projects: None yet

Milestone: No milestone

Development: No branches or pull requests

2 participants

30 bugs filed

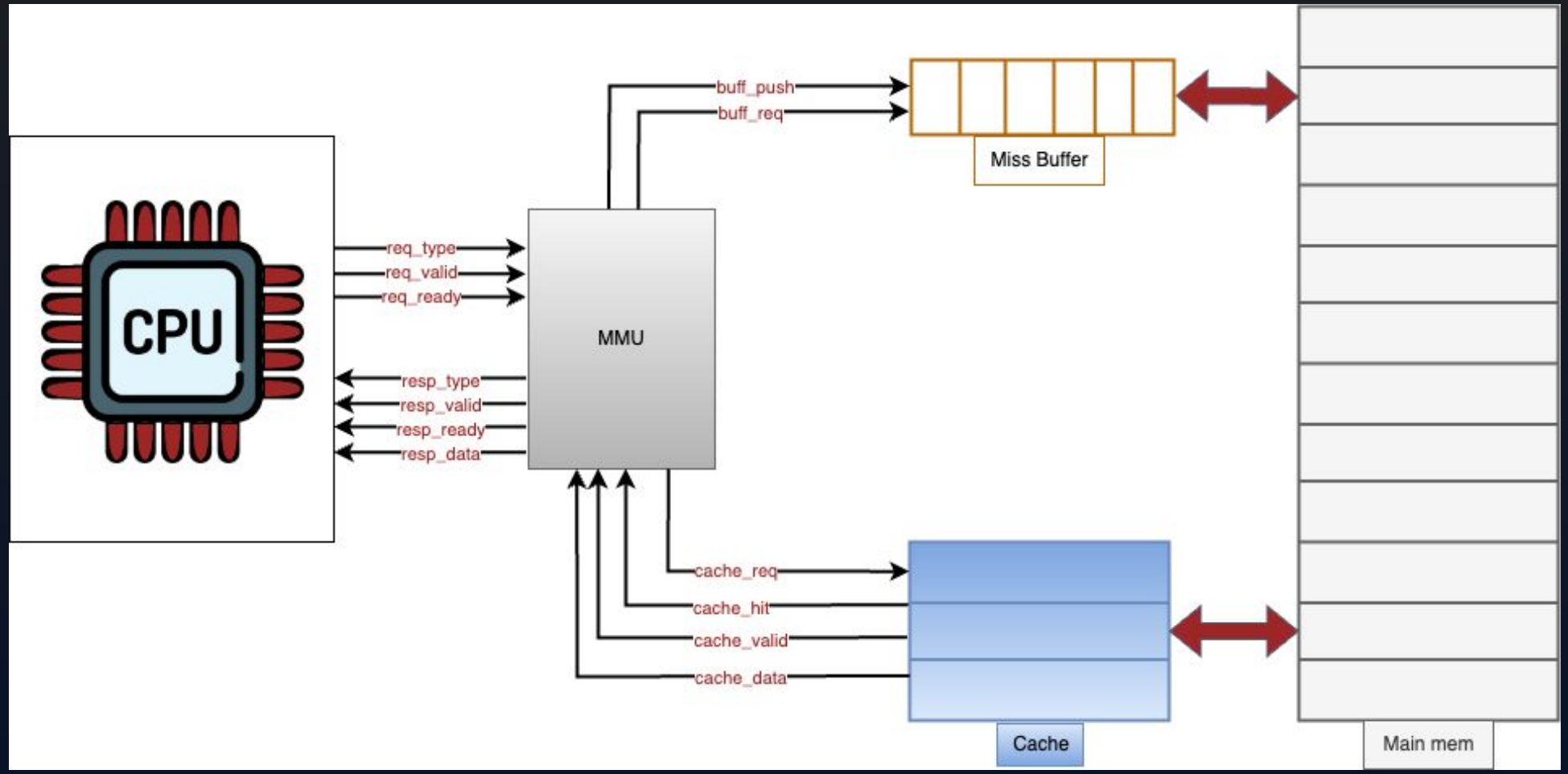
axiomise — predictable formal verification

Memory subsystem

Caught by our *formalISA*

Cache issues

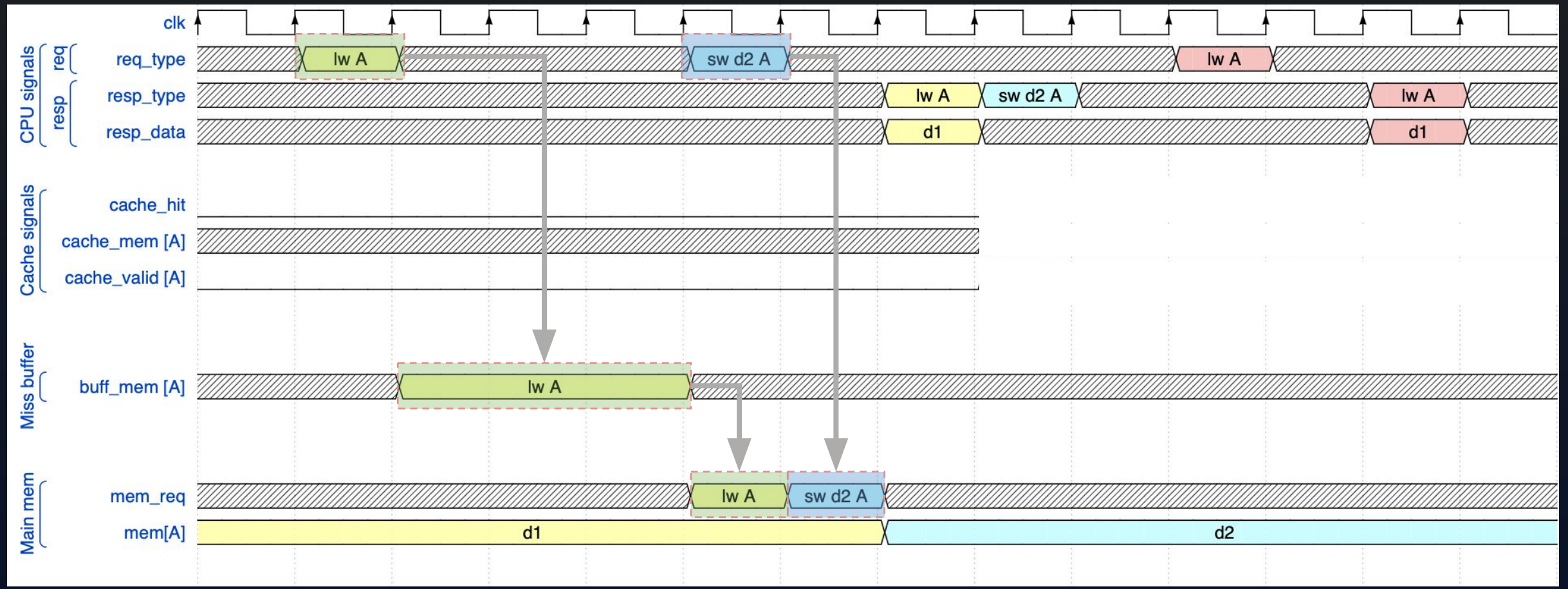
Bugs hard to catch with simulation



axiomise
predictable formal verification

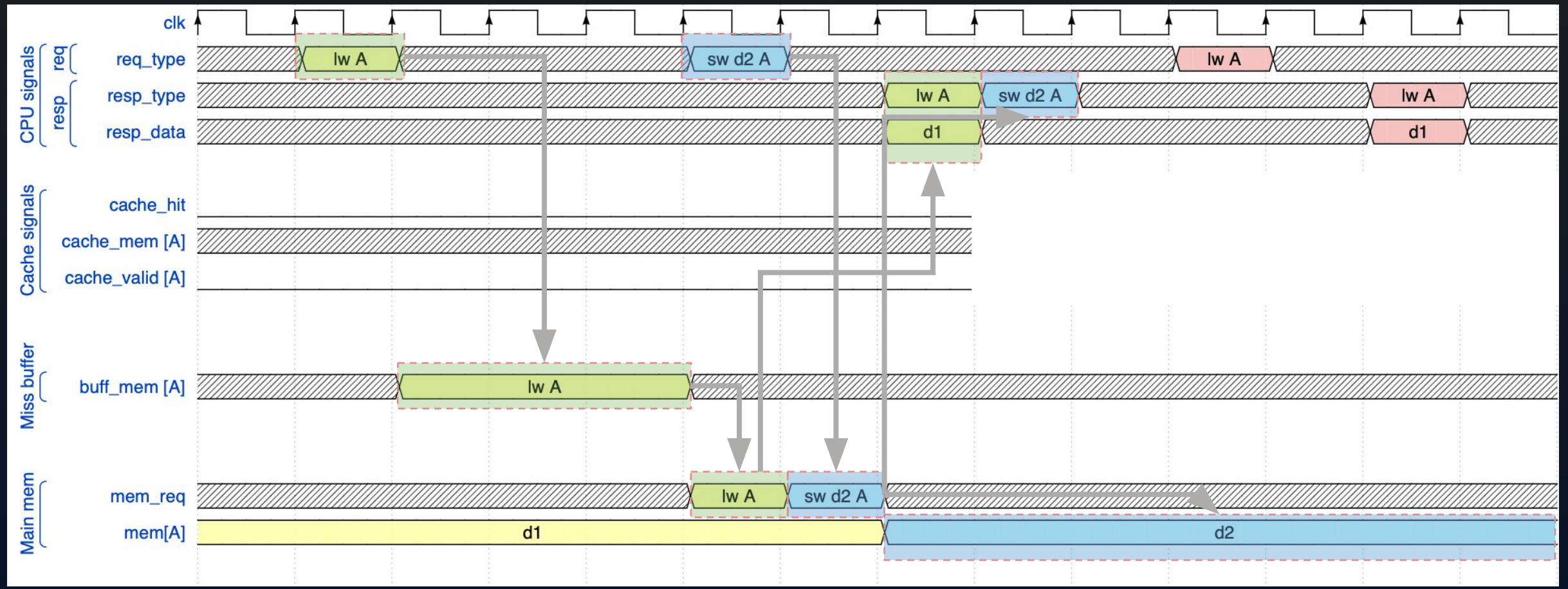
Cache issues

Bugs hard to catch with simulation



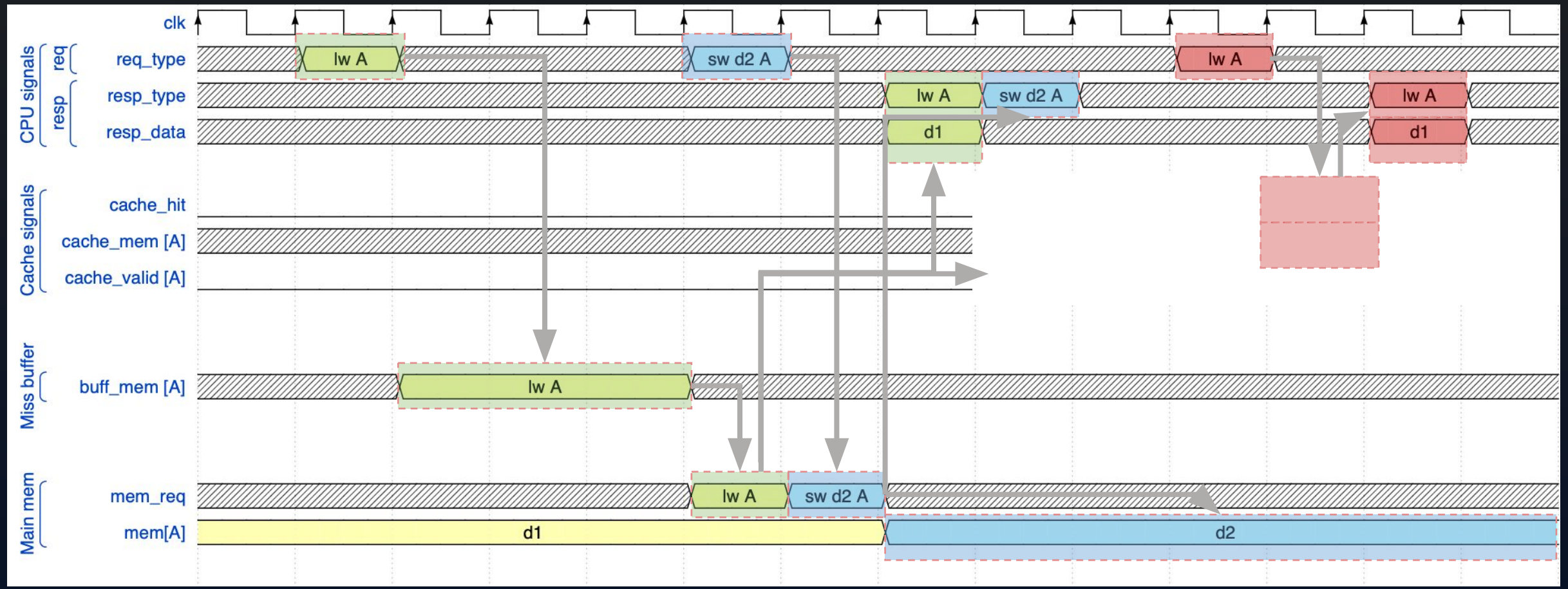
Cache issues

Bugs hard to catch with simulation



Cache issues

Incorrect validation of cache line due to the bypass store



Footprint®

Area analyser for silicon

axiomise®
predictable formal verification

footprint Results

Open-source designs

Designs	Gate count	Flop count	Redundant component		Estimated redundant gates
Cheriot-ibex	303,737	14,723	Counter	3	768
			Register	313	16,440
			Array	23	7,872
Nocgen – NoC (Network-on-Chip)	590,144	35,200	Fifo	32	96,352
			Mux	16	1,872
			Fsm	48	864
			Counter	160	3,456
			Register	624	43,296
			Array	32	33,600
Chipyard – TinyRocket_ChipTop	29,684,024	322,776	Arbiter	1	2
			Counter	260	23,220
			Register	1,118	852,552
			Array	4	6144
Chipyard – Boomv3Large_BoomCore	850,191	79,989	Register	3,858	138,438
			Counter	17	5,598
			Array	3	18,432
Sdram_controller	14,507	1,356	Fsm	7	144
			Counter	17	570
			Register	135	6,498
			Array	1	792
Verilog-ethernet-udp_complete	1,851,130	46,807	Register	104	9,096
			Array	9	4,032

Why Axiomise formal verification matters?

Covering the entire spectrum of verification requirements

High Proof Convergence

So that you have higher confidence

Bugs & Exhaustive Proofs

Making sure your design is bug free

Innovation in Abstractions

Allowing you to have the highest quality designs, without re-spin

Scalable Proof Engineering

Our solutions scale as your designs do

High Quality Sign-off

Functional Safety Security PPA

We find bugs that no-one else can; nobody gets proof convergence like us

axiomise
predictable formal verification

Cost of Failure is Expensive

https://www.perforce.com/blog/mdx/semiconductor-startups#cost-of-failure-for-semiconductor-startups

Cost of Failure For Semiconductor Startups

As mentioned above, semiconductor design needs to be done correctly because the cost of failure is simply too high.

How high? It takes an average **$250+ million investment just to get started**. If it doesn't work correctly, a respin (design correction and re-fabrication) is long and costly – to the tune of around **$25 million per re-spin**.

What causes re-spins? The problems often stem from specifications: either incomplete specifications or changing specifications that don't get communicated to the design teams.

The root of these problems is a lack of trace of traceability, managing the process of moving from design specifications, through design, to verification and validation -- and all changes along the way.

Summary

Formal methods is a necessity to reduce costs

Bugs caught late in the design cycle result in costly fixes and catastrophic failures

Formal enables efficient bug hunting, a natural for shift-left paradigm

Exhaustiveness establishes ”proofs of bug absence” avoiding respins

10^{30} simulation cycles are not going to find bugs that formal finds in 7 cycles

Mantra for success:

- Architects use formal for validation
- Designers use formal for verification
- Verification Engineers use formal for IP and sub-system level, simulation for interfaces



axiomise
predictable formal verification

www.axiomise.com

CONSULTING & SERVICES

TRAINING

CUSTOM SOLUTIONS

info@axiomise.com