



# How the Right Mindset Increases Quality in RISC-V Verification

Philippe Luc, Director of Verification, Cudasip

Salaheddin Hetalani, Field Application Engineer, Siemens EDA



# RISC-V – Why is it important to today?

- Often called “open source”... but “open standard” is better
- Standardized ISA
  - With a wide range of extensions (FP, DSP, Vector etc.)
  - Ability to customize ISA for user applications
- Strong ecosystem
  - Tools support - (LLVM, Lauterbach, IAR etc.)
  - Frameworks - (Linux, Android, AI/ML, Hypervisor...)
- Both free (open source) and commercial cores

# RISC-V in the future

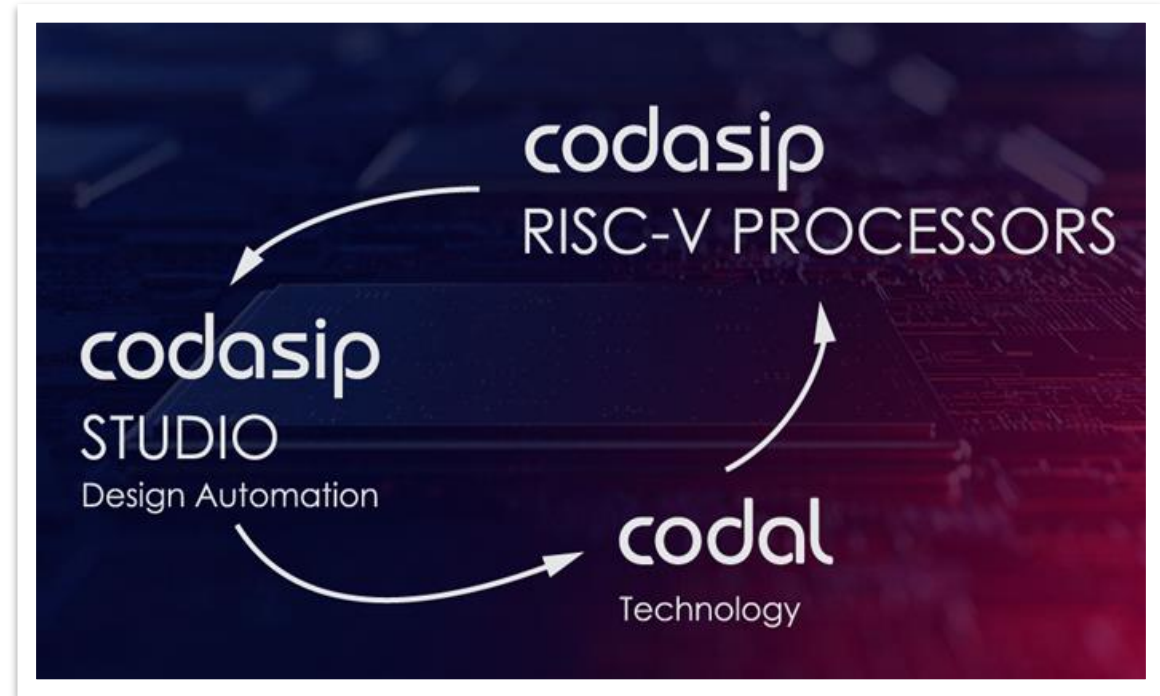
- Market is pushing towards open standards
- Customers need a better way to address their applications
  - Additional RISC-V Extensions
  - Users doing their own customization of the ISA



Codasip supports the open ISA and has automation for customization

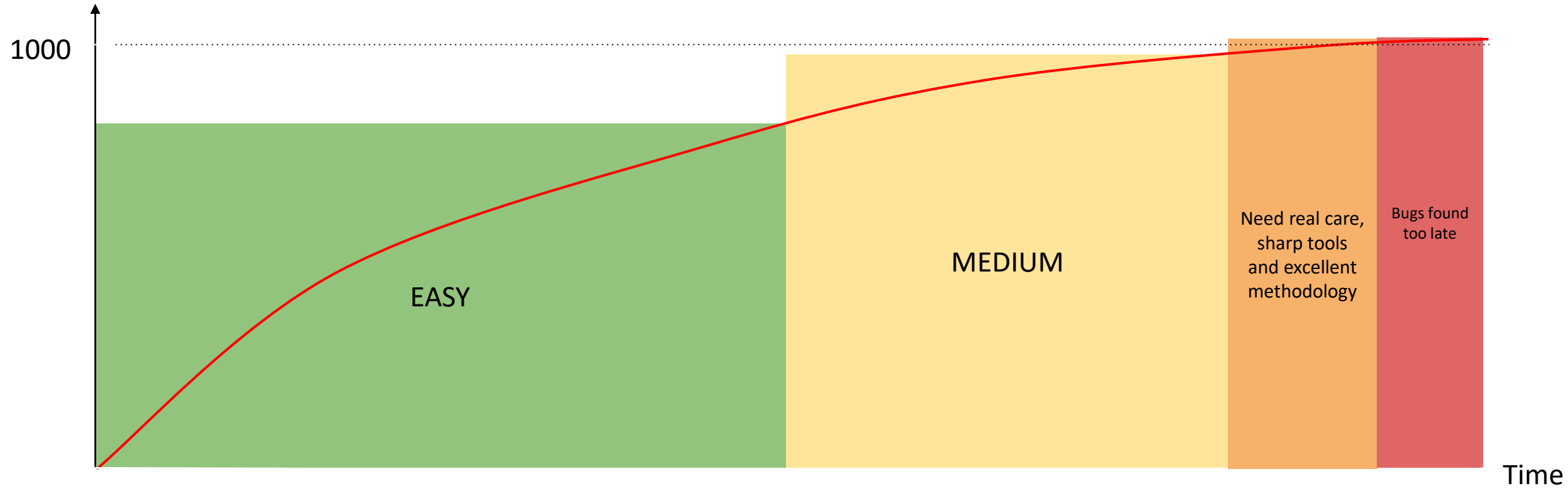
# Codasip

We do RISC-V with a twist

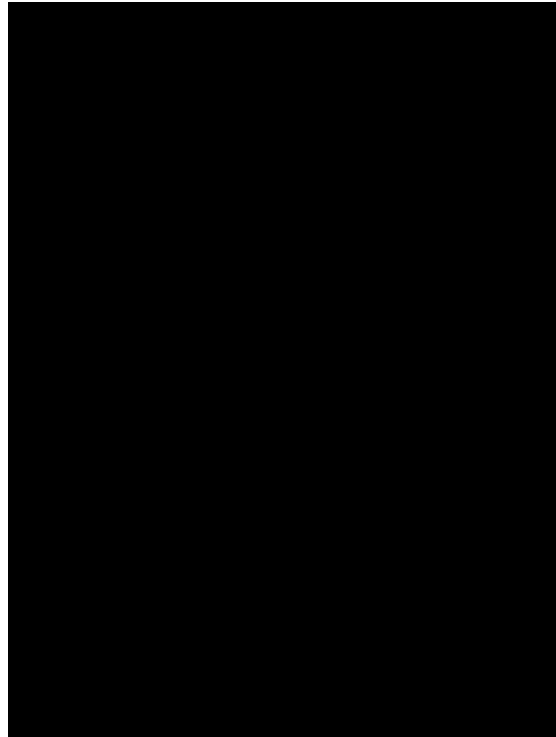


# Bugs' life

Number of bugs found



# Where to search for bugs



Everywhere



Lighted spaces

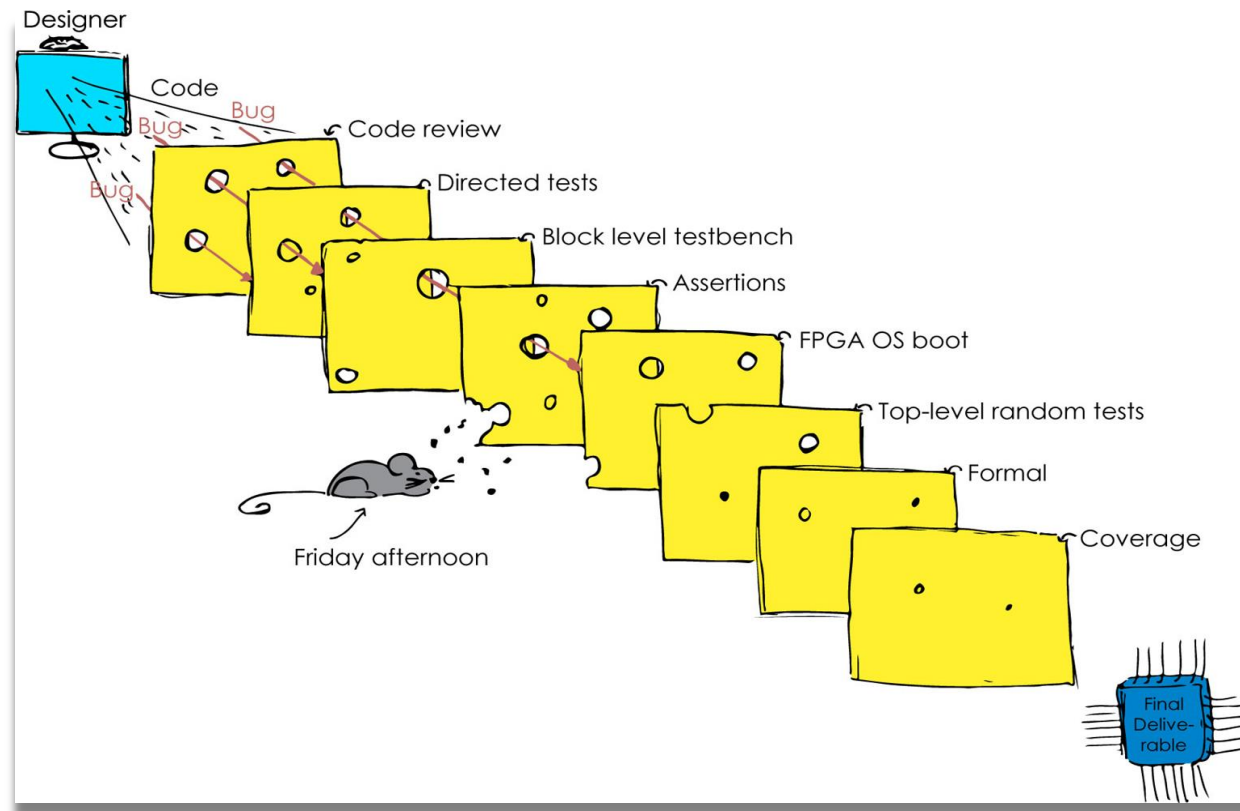
# Customer request: Make an unbreakable chain

- RISC-V is open standard
  - Same baseline ISA
  - Optional standard extensions
  - Custom extensions
  - Different microarchitectures
- Test the chain
  - Pull it
  - Shake it
- Check it is pulled and shaken



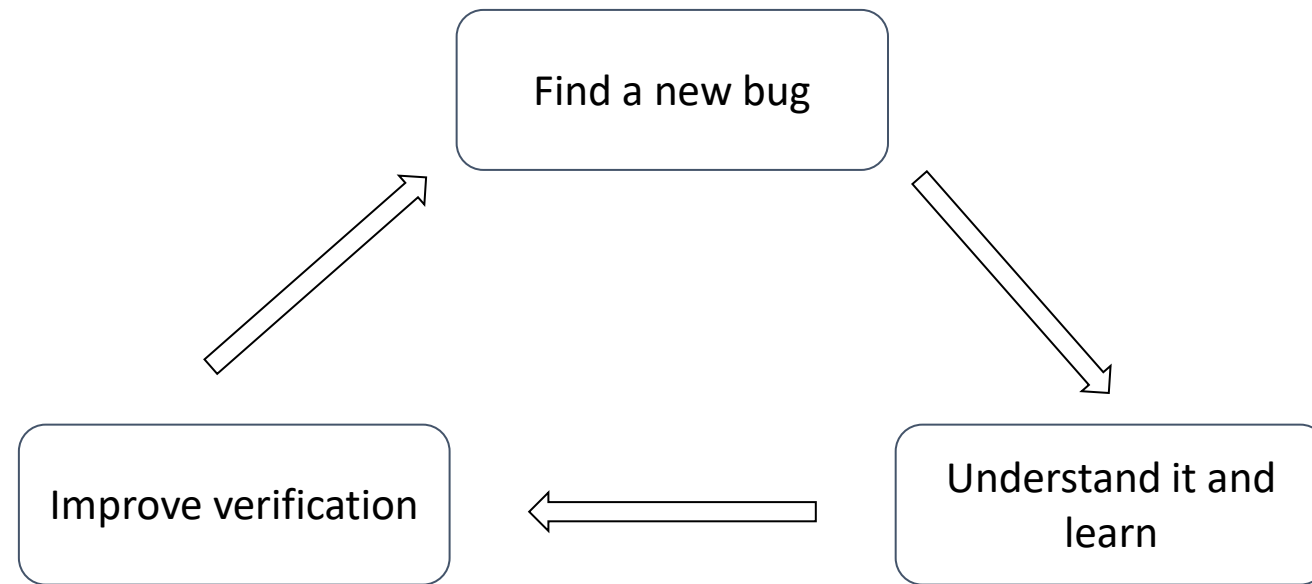


# Verification flow: plugging the holes





# Fix bugs, and improve verification



# Codasip strategy with Siemens

- Codasip started using formal methodology thoroughly
  - Consistency checks
  - Siemens “Processor Verification App”
    - Dedicated OneSpin formal checker for RISC-V architecture

```
272 property RESET_p;  
273 reset_sequence | =>  
274     t##1 !(ifu.pma_fetch_s_fault_o_Q|ifu.pmp_fetch_s_fault_o_Q) and  
275     during(t, nxt(t,4), rst_n)  
276 implies  
277     t##3 Ready2Execute and  
278     t##0 !fetch_req_valid ##1 fetch_req_valid && fetch_req_PC=={l31_32IMFCB_UPTIDpbid_ca_core_t.p_boot_addr_Q[31:2],2'b00} and  
279     t##4 pc_check({l31_32IMFCB_UPTIDpbid_ca_core_t.p_boot_addr_Q[31:2],2'b00}) and  
280     t##4 all_regs_valid(Arch) && Arch.X[0].data=='0 and  
281     t##4 RISC_V_CSR_reset_state(Arch.CSR) and  
282     t##0 !dmem_req_valid [*4] and  
283     t##3 right_hook;  
284 endproperty  
285
```

- For a first use Codasip tried in a core that was in development

# OneSpin's industry proven solutions

*The content of this article was presented at DVCon 2007 and is posted with DVCon's permission.*

## Complete Formal Verification of TriCore2 and Other Processors



## Complete Formal Verification of a Family of Automotive DSPs



**BOSCH**  
Invented for life



## Formal Verification Applied to the Renesas MCU Design Platform

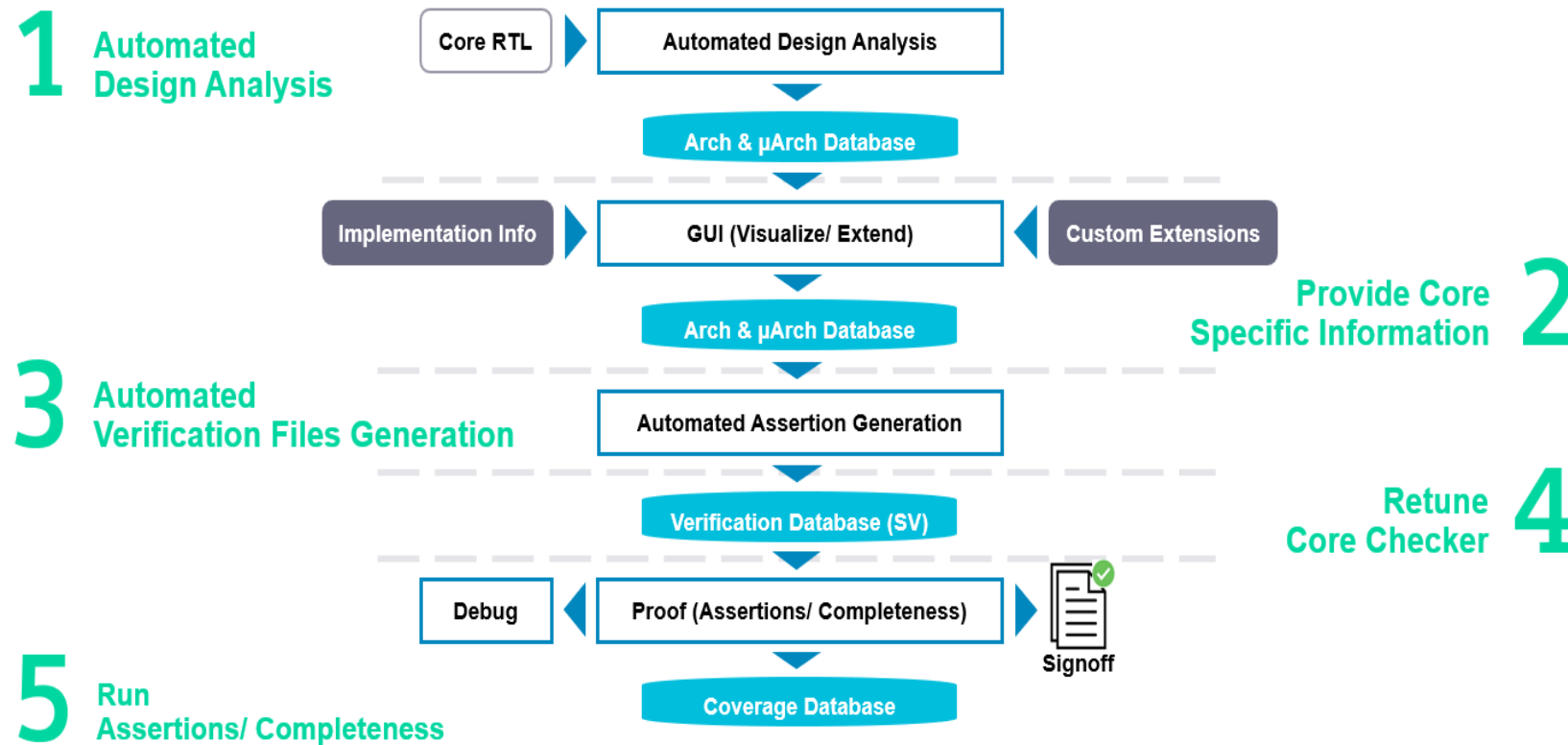


## Complete Formal Verification of RISC-V Processor IPs for Trojan-Free Trusted ICs

GOMACTech  
2019



# OneSpin's processor verification app – Flow



# Processor verification app – GUI

The screenshot displays the onespin Processor Integrity application interface. The top navigation bar is blue with the 'onespin' logo. The main content area is divided into several sections:

- Status:** Shows 'Running' with a blue status icon. Below it, text indicates 'Generating assertions for RV32IMFCBZicsr\_Zifencei\_Zba\_Zbb\_Zbc\_Zbs with user mode' and '41 assertions generated for 147 instructions and 48 CSRs'. A callout box labeled 'Merge core specific information' points to the 'Merge', 'Sync', and 'Clear' buttons in this section.
- Apps:** Contains three buttons: 'Extract from design', 'Generate assertions', and 'Generate GFV'. Callouts point to these buttons: 'Automated design analysis' for 'Extract from design', 'Generate assertions' for 'Generate assertions', and 'Generate completeness plan' for 'Generate GFV'.
- ISA:** Displays configuration parameters: 'XLEN: 32', 'Number of Counters: 16', 'Number of PMPs: 0', 'Extensions: A C D F I M N S U X', 'Reset PC: 0', and 'Z: Zifencei Zicsr Zfinx Zdinx Zba Zbb Zbc Zbs ...'. A callout box labeled 'Design ISA information' points to this section.
- Custom Extensions:** Features a row of buttons: 'Instructions', 'Bitfields', 'Registers', 'Register Files', 'CSR Map', and 'CSR Attributes'. A callout box labeled 'Custom/ User extensions' points to this section.
- μ-Architecture:** Shows detailed configuration for the 'DUT Module: codix\_berkelium\_ca\_core\_t'. It includes fields for 'Fetch Interface', 'Data Memory Interface', and 'Write Data'. Below these are four buttons: 'Pipeline', 'Mappings', 'Parameters', and 'Invariants'. A callout box labeled 'Design micro-architecture information' points to this section.

# Processor verification app – Custom Instructions

ISA Custom Extensions - Instructions

	Mnemonic	Decoding	Restrictions	Disassembly	Execution
<input type="checkbox"/>	CV.LB	imm[11:0] rs1/rd2 000 rd 0001011		cv.lb {rd},{imm}	let addr : xlenbits = X(rs1) + EXTS(imm); X
<input type="checkbox"/>	CV.LH	imm[11:0] rs1/rd2 001 rd 0001011		cv.lh {rd},{imm}	let addr : xlenbits = X(rs1) + EXTS(imm); X
<input type="checkbox"/>	CV.LW	imm[11:0] rs1/rd2 010 rd 0001011		cv.lw {rd},{imm}	let addr : xlenbits = X(rs1) + EXTS(imm); X
<input type="checkbox"/>	CV.LBU	imm[11:0] rs1/rd2 100 rd 0001011		cv.lbu {rd},{imm}	let addr : xlenbits = X(rs1) + EXTS(imm); X
<input type="checkbox"/>	CV.LHU	imm[11:0] rs1/rd2 101 rd 0001011		cv.lhu {rd},{imm}	let addr : xlenbits = X(rs1) + EXTS(imm); X
<input type="checkbox"/>	CV.SB	imm[11:5] rs2 rs1/rd2 000 imm[4:0]		cv.sb {rs2},{imr}	let addr : xlenbits = X(rs1) + EXTS(imm); r
<input type="checkbox"/>	CV.SH	imm[11:5] rs2 rs1/rd2 001 imm[4:0]		cv.sh {rs2},{imr}	let addr : xlenbits = X(rs1) + EXTS(imm); r
<input type="checkbox"/>	CV.SW	imm[11:5] rs2 rs1/rd2 010 imm[4:0]		cv.sw {rs2},{imr}	let addr : xlenbits = X(rs1) + EXTS(imm); r
<input type="checkbox"/>	CV.ADD.B	0000000 rs2 rs1 001 rd 1010111		cv.add.b {rd},{r}	X(rd)=(X(rs1)[31..24]+X(rs2)[31..24]) @ (X(rs1)[7..0])
<input type="checkbox"/>	CV.ADD.S	0000000 rs2 rs1 101 rd 1010111		cv.add.sc.b rd, rs	X(rd)=(X(rs1)[31..24]+X(rs2)[7..0]) @ (X(rs1)[7..0])
<input type="checkbox"/>	CV.ADD.S	0000000 imm[0]5:1 rs1 111 rd 1010		cv.add.sci.b {rd}	X(rd)=(X(rs1)[31..24]+EXTS(imm)[7..0]) @ (X(rs1)[7..0])
<input type="checkbox"/>	CV.DOTUI	1000000 rs2 rs1 001 rd 1010111		cv.dotup.b {rd},	X(rd)=mul8(X(rs1)[7..0],X(rs2)[7..0])+mul8(X(rs1)[31..24],X(rs2)[31..24])
<input type="checkbox"/>	CV.SDOTI	1010000 rs2 rs1 001 rd/rs3 101011		cv.sdotup.b {rd}	X(rd)=mul8(X(rs1)[7..0],X(rs2)[7..0])+mul8(X(rs1)[31..24],X(rs3)[31..24])



# Processor verification app – Generated Assertions

	Name	Proof Status	Witness Status	Case Split Status	Validity	Source
	! <any stat >	! <any stat >	! <any stat >	! <any status >	! <any >	! <any >
	Constraints					
	Properties					
Interrupts	RV_chk.ops.BUBBLE_a	open	open	open	up_to_date	core_checker.sv:735
	RV_chk.ops.INTR_Handle_a	open	open	open	up_to_date	core_checker.sv:740
	RV_chk.ops.RESET_a	open	open	open	up_to_date	core_checker.sv:734
	RV_chk.ops.RV32I.ARITH_a	open	open	open	up_to_date	core_checker.sv:745
	RV_chk.ops.RV32I.BRANCH_a	open	open	open	up_to_date	core_checker.sv:744
	RV_chk.ops.RV32I.Debug.EBREAK_ForcedEntry_a	open	open	open	up_to_date	core_checker.sv:757
	RV_chk.ops.RV32I.Debug.EBREAK_HaltReq_a	open	open	open	up_to_date	core_checker.sv:756
	RV_chk.ops.RV32I.EBREAK_BreakPoint_a	open	open	open	up_to_date	core_checker.sv:753
I - Base Integer	RV_chk.ops.RV32I.ECALL_a	open	open	open	up_to_date	core_checker.sv:752
	RV_chk.ops.RV32I.FENCE_a	open	open	open	up_to_date	core_checker.sv:749
	RV_chk.ops.RV32I.JUMP_a	open	open	open	up_to_date	core_checker.sv:743
	RV_chk.ops.RV32I.MEM_a	open	open	open	up_to_date	core_checker.sv:746
	RV_chk.ops.RV32I.MEM_MultiAccess_a	open	open	open	up_to_date	core_checker.sv:747
	RV_chk.ops.RV32I.WFI_a	open	open	open	up_to_date	core_checker.sv:750
	RV_chk.ops.RV32I.xRET_a	open	open	open	up_to_date	core_checker.sv:751
M	RV_chk.ops.RV32M.DIV_a	open	open	open	up_to_date	core_checker.sv:792
	RV_chk.ops.RV32M.MUL_a	open	open	open	up_to_date	core_checker.sv:791
	RV_chk.ops.RVC.ARITH_a	open	open	open	up_to_date	core_checker.sv:905
C	RV_chk.ops.RVC.BRANCH_a	open	open	open	up_to_date	core_checker.sv:904
	RV_chk.ops.RVC.JUMP_a	open	open	open	up_to_date	core_checker.sv:903
	RV_chk.ops.RVC.MEM_a	open	open	open	up_to_date	core_checker.sv:906
	RV_chk.ops.RVC.MEM_MultiAccess_a	open	open	open	up_to_date	core_checker.sv:907
X	RV_chk.ops.RVX.instr_CV_LB_2_a	open	open	open	up_to_date	custom_extensions.sv:1802
	RV_chk.ops.RVX.instr_CV_LB_a	open	open	open	up_to_date	custom_extensions.sv:1801
Zicsr & Zifencei	RV_chk.ops.RVZicsr.CSRx_a	open	open	open	up_to_date	core_checker.sv:777
	RV_chk.ops.RVZifencei.FENCE_I_a	open	open	open	up_to_date	core_checker.sv:773
	RV_chk.ops.XCPT_IF_ID_a	open	open	open	up_to_date	core_checker.sv:737
Exceptions	RV_chk.ops.XCPT_MEM_a	open	open	open	up_to_date	core_checker.sv:738
	RV_chk.ops.XCPT_WB_a	open	open	open	up_to_date	core_checker.sv:739
	SVA Named Properties					
	SVA Sequences					

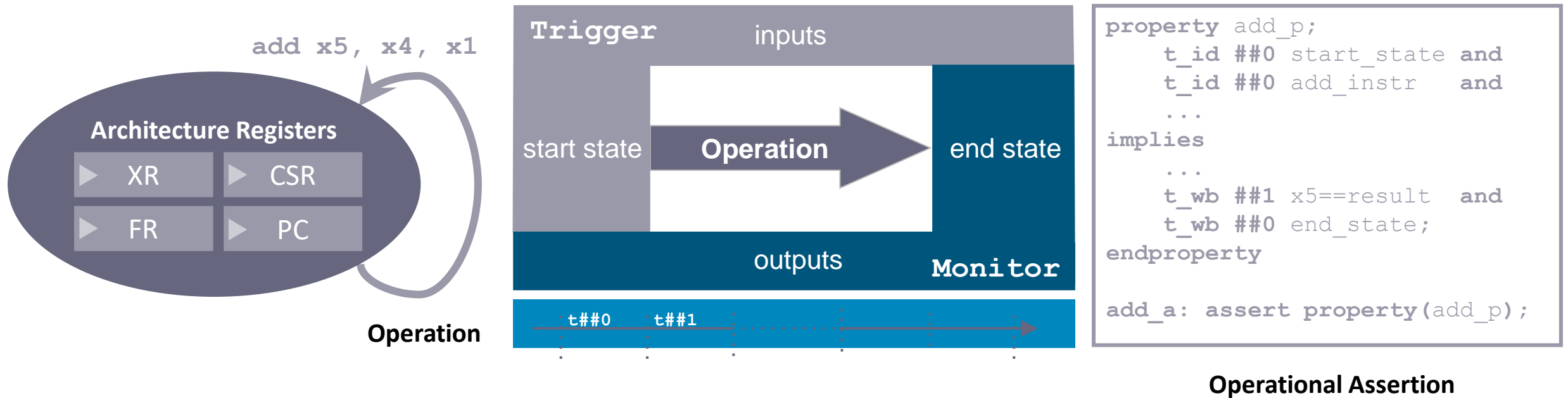
RV32IMC  
\_Zicsr  
\_Zifencei

27

Assertions



# OneSpin 360's unique modelling principle



# Siemens findings at Codasip (1/2)

Consistency Checks/Dead Code - Found a signal that was hardwired

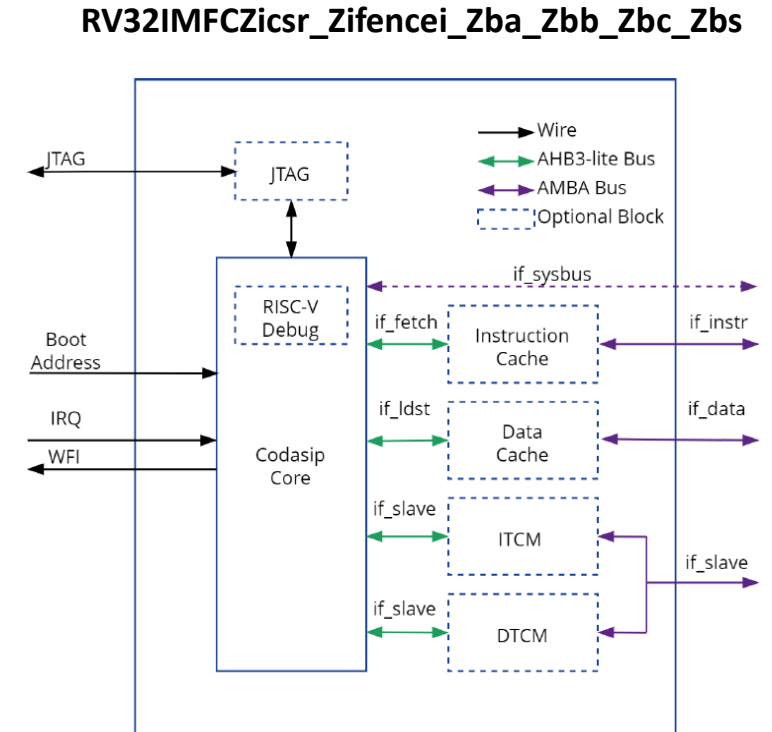
```
.....// "Virtual" EX1 stage handling
.....if (s_ex1_clear_i)
.....    pipe.EX1.clear();
.....else if (s_ex1_stall_i)
.....    pipe.EX1.stall();
```

```
.....// pipeline controller
.....plc.s_flush_i = s_flush;
.....plc.s_id_clear_i = s_id_clear;
.....plc.s_ex_stall_i = s_ex_stall;
.....plc.s_ex_clear_i = s_ex_clear;
.....plc.s_ex1_stall_i = s_ex1_stall;
.....plc.s_ex1_clear_i = s_ex1_clear;
```

```
.....// EX1 stage
.....s_ex1_stall = ex1_stall;
.....s_ex1_clear = false;
```

# Siemens findings at Codasip (2/2)

- Illegal instruction exceptions not raised
- Illegal CSR counter increment
- Legal instructions treated as illegal
- Wrong settings of floating-point flags, memory accesses, and program counter
- Storing the wrong address in mtval in case of a misaligned memory access exception.



# Conclusion

- Consistency checks and “Super” linting can find errors earlier
- OneSpin Processor Verification App allows a very efficient, additional way to verify
- Fixing issues earlier in the verification process than standard UVM
- Shorter verification times than the standard functional based approach
- Good complement for any verification methodology



# Questions