



Towards Trustworthy RISC-V Processors for Safety-Critical Applications

RISC-V Days Tokyo 2022 Spring

SIEMENS

| Agenda

Introduction

RISC-V Processor Core Verification App

Application Examples

Underlying Technology

| Introduction

OneSpin's Industry Proven Solutions

15 years of cutting-edge formal processor verification 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



Formal Verification Applied to the Renesas MCU Design Platform



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

GOMACTech
2019



Enabling High-Quality Processors

OneSpin supports the RISC-V community



Technology / Solution

Processor Core Integrity Solutions*

- Core verification
- Integration verification



Industry Involvement

RISC-V International

OpenHW Group

Scale4Edge Project



User Community

Commercial solution adoption

- AFRL/ Edaptive
- Infineon
- Bosch
- Renesas

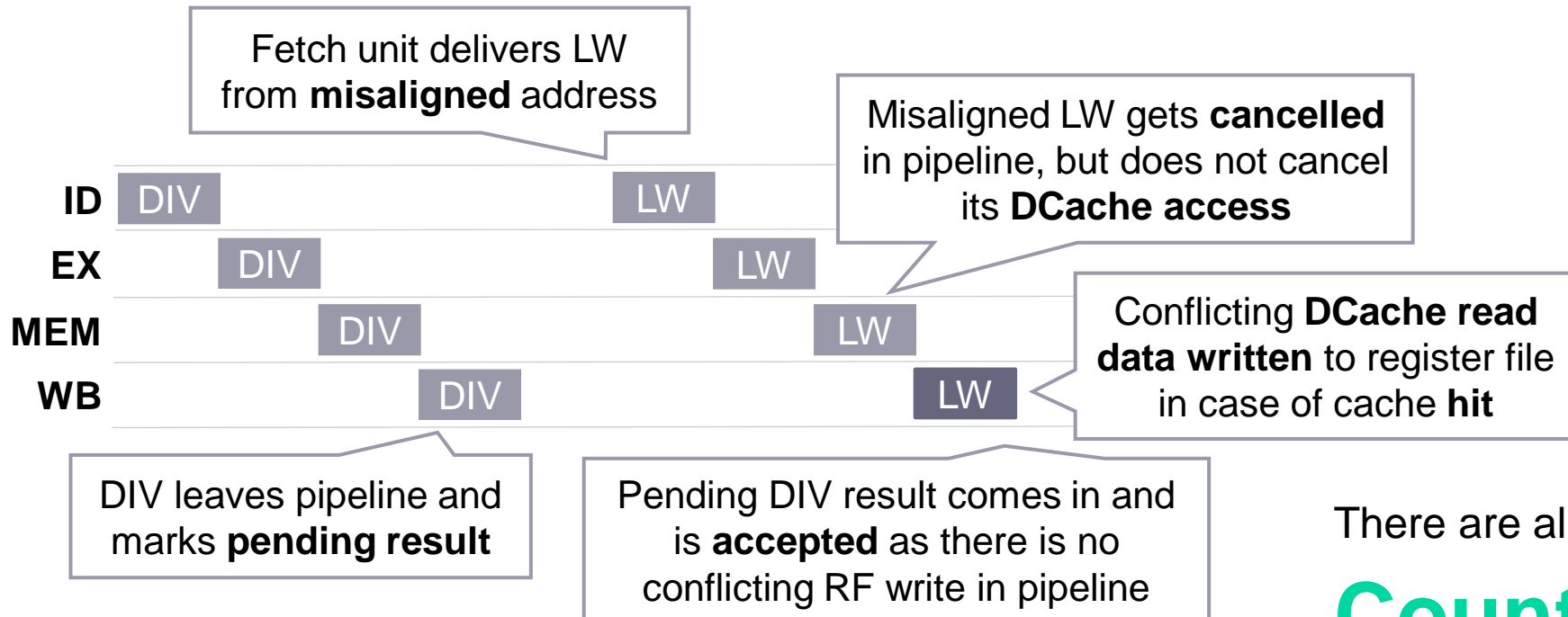


*Ongoing development

Example Bug in Processor Core

DIV result not written back to register file

How do you catch this?



There are almost
Countless
Possibilities



RISC-V Processor Core Verification App

OneSpin's RISC-V Processor Core Verification App

Automate and accelerate processor verification



10x Verification Speed-up

Accelerate coverage closure

Optimized formal engines

Pinpoint bugs - Quick fix check



High Degree of Automation

No writing of functional coverage model

Automatic μ -architecture details extraction

Automatic assertion generation

Built-in disassembler annotation



Exhaustive & Complete Verification

100% functional coverage

Unbounded proofs

No undocumented RTL

Custom extensions support

ISA & privileged ISA compliance

App Flow

1 Automated Design Analysis



2 Provide Core Specific Information



3 Automated Verification Files Generation



4 Retune Core Checker



5 Run Assertions/ Completeness



App GUI

The screenshot shows the onespinn Processor Integrity application interface. At the top, the 'Status' section shows 'Initial' and a 'Merge core specific information' callout points to 'Merge', 'Sync', and 'Clear' buttons. The 'Apps' section on the right includes 'Extract from design', 'Generate assertions', and 'Generate GFV' buttons, with callouts for 'Automated design analysis', 'Generate assertions', and 'Generate completeness plan'. The 'ISA' section displays 'XLEN: 32', 'Extensions: A C D F I M N S U X', and 'Z: Zifencei Zicsr Zfinx Zdinx Zba Zbb Zbc Zbs ...'. Below this is the 'Custom Extensions' section with buttons for 'Instructions', 'Bitfields', 'Registers', 'Register Files', 'CSR Map', and 'CSR Attributes', with a callout for 'Custom/ User extensions'. The 'µ-Architecture' section at the bottom contains fields for 'DUT Module', 'Fetch Interface', 'DUT Instance', 'Data Memory Interface', and various signal names like 'Req. Valid', 'Resp. Ready', etc., with a callout for 'Design micro-architecture information'. At the bottom of this section are buttons for 'Pipeline', 'Mappings', 'Instruction Effects', 'Parameters', and 'Invariants'.

Design ISA information

Custom/ User extensions

Merge core specific information

Automated design analysis

Generate assertions

Generate completeness plan

Design micro-architecture information

Custom Extensions Support

Leverage executable specification to automate verification

Straightforward to add custom instructions

- Table for custom instructions (JSON format)
- Tabular decoding from RISC-V ISA standard
- Execution specified in subset of Sail language*
- Disassembly specified for debugger display

Straightforward to add registers/ register files

Support of multiple source/ destination registers

Automated verification of custom extensions

*<https://github.com/rems-project/sail-riscv>

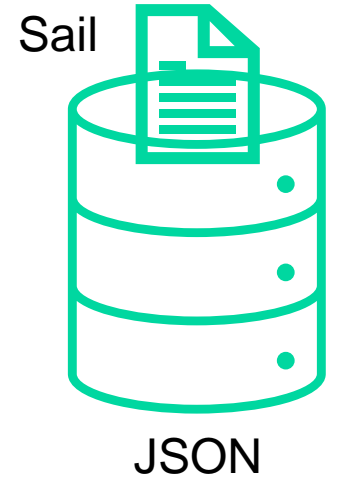
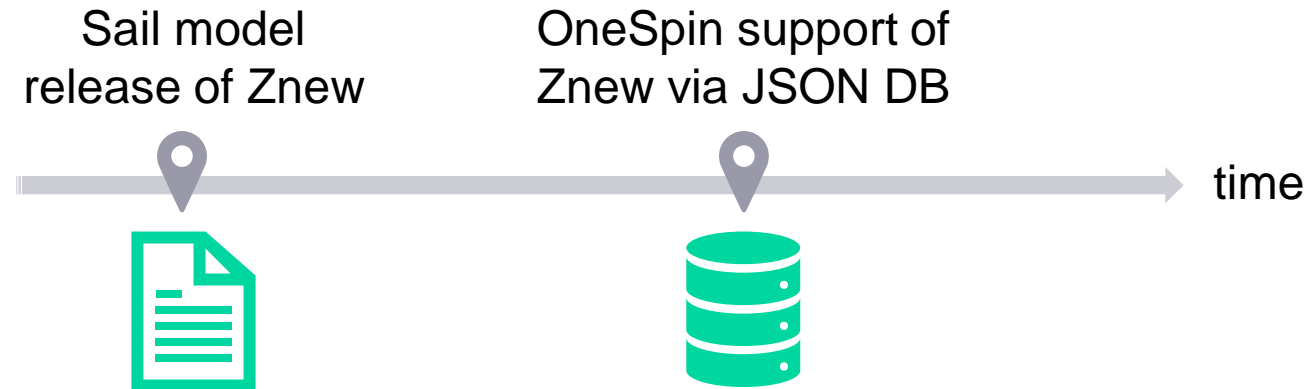
ISA Custom Extensions - Instructions

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},{imm}	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},{imm}	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},{imm}	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
<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(rs
<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]) @
<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
<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

Standard Extension Support

Standard extensions also based on JSON database with Sail execution modeling

- OneSpin support derived from officially released Sail model
- OneSpin support possible after Sail model release
 - Customers drive actual decision
- Recently derived bit manipulation support (Zba, Zbb, Zbc, Zbs) from Sail
- Vector extension to follow after Sail model release



App Assertions

Name	Proof Status	Witness Status	Case Split Status	Validity	Source
! <any stat >	! <any statu >	! <any statu >	! <any statu >	! <any >	! <any >
▶ Constraints					
▼ Properties					
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
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
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
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
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
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
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					

Interrupts

I - Base Integer

M

C

X

Zicsr & Zifencei

Exceptions

RV32IMC
_Zicsr
_Zifencei

27

Assertions

Reference Cards

Assertion Instruction/ Exception/ Interrupt Mapping			
Assertion Name	What is verified? (Instruction/ Exception/ Interrupt)	Extension	XLEN
RESET_a	The behaviour of the pipeline after applying the reset sequence		
BUBBLE_a	NOP (The behaviour of the pipeline when nothing is executed due to stalling or nop operations)		
XCPT_IF_ID_a	Debug instruction address breakpoint, instruction address breakpoint, instruction page fault, Instruction access fault, Illegal instruction, and instruction address misaligned exceptions		
XCPT_MEM_a	Debug Load/Store/AMO address breakpoint, and Load/Store/AMO address breakpoint exceptions		
XCPT_WB_a	Load/Store/AMO address misaligned, Load/Store/AMO page fault and Load/Store/AMO access fault exceptions		
INTR_Handle_a	User/ Supervisor/ Machine software/ timer/ external interrupts as well as the debug interrupts (including single stepping)		
RV32I.ARITH_a	LUI,AUIPC,ADDI,SLTI,SLTIU,XORI,ORI,ANDI,SLLI,SRLI,SRAI,ADD,SUB,SLL,SLT,SLTU,XOR,SRL,SRA,OR,AND	I	32
RV32I.JUMP_a	JAL,JALR	I	32
RV32I.BRANCH_a	BEQ,BNE,BLT,BGE,BLTU,BGEU	I	32
RV32I.MEM_a	LB,LH,LW,LBU,LHU,SB,SH,SW	I	32
RV32I.MEM_MultiAccess_a	LH,LW,LHU,SH,SW	I	32
RV32I.FENCE_a	FENCE	I	32
RV32I.ECALL_a	ECALL (Environment call exception)	I	32
RV32I.EBREAK_BreakPoint_a	EBREAK,C.EBREAK (Environment break exception)	I	32
RV32I.Debug.EBREAK_HaltReq_a	EBREAK,C.EBREAK	I	32
RV32I.Debug.EBREAK_ForcedEntry_a	EBREAK,C.EBREAK	I	32
RV32I.xRET_a	MRET,SRET,URET,DRET		
RV32I.WFI_a	WFI		
RV32I.Supervisor.SFENCE_VMA_a	SFENCE.VMA	S	
RV64I.ARITH_a	ADDIW,SLLIW,SRLIW,SRAIW,ADDW,SUBW,SLLW,SRLW,SRAW	I	64
RV64I.MEM_a	LWU,SD,LD	I	64
RV64I.MEM_MultiAccess_a	LWU,SD,LD	I	64
Z RVZifencei.FENCE_I_a	FENCE.I	Zifencei	32/ 64
Z RVZicsr.CSRx_a	CSRRW,CSRRS,CSRRC,CSRRWI,CSRRSI,CSRRCI	Zicsr	32/ 64
M RV32M.MUL_a	MUL	M	32
M RV32M.MULH_a	MULH	M	32
M RV32M.MULHSU_a	MULHSU	M	32
M RV32M.MULHU_a	MULHU	M	32
M RV32M.DIV_a	DIV	M	32
M RV32M.DIVU_a	DIVU	M	32
M RV32M.REM_a	REM	M	32
M RV32M.REMU_a	REMU	M	32
M RV32M.MUL_a	MUL,MULH,MULHSU,MULHU	M	32
M RV32M.DIV_a	DIV,DIVU,REM,REMU	M	32

VIP Assertion VS Spec Instruction/ Exception Mapping

| Application Examples

Types of Bugs Found

Rocket, CV32E40P, ibex cores

8

Illegal Instruction

7

Misbehaving Pipeline

2

Illegal CSR Update

4

Undocumented Instruction

8

Wrong CSR Update

1

Undocumented CSR

2

Illegal CSR Write

1

Misbehaving Instruction

3

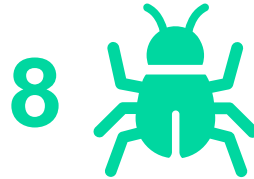
Illegal Counter Update

Application Example

64-Bit Rocket Core


Issues reported*

- #1752 DIV result not written to register file
- #1861 Illegal opcodes replayed
- #1868 Undocumented non-standard instruction
- #1949 Undocumented non-standard CSR
- #2022 DRET instruction is executable outside of debug mode



Performance

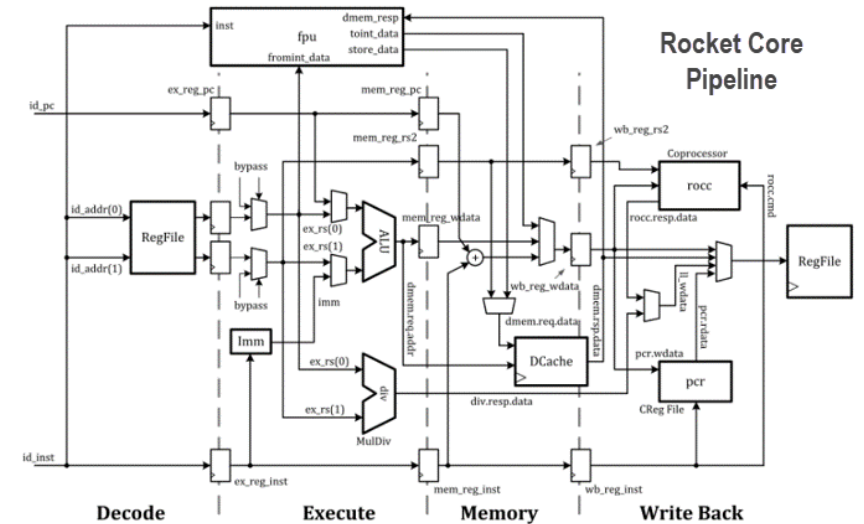
 < 5 days of verification setup

 < 10 minutes run time per property

 < 2 hours total runtime

*<https://github.com/freechipsproject/rocket-chip/issues/>

RV64GC_Zicsr_Zifencei Core



- 5-stage single-issue in-order pipeline
- 39-bit virtual memory address space
- 3 privilege levels
- Out-of-order termination
- Branch prediction & replay mechanism
- Verified and taped-out multiple times

Application Example

32-Bit RI5CY/ CV32E40P Core

Issues reported*

#132 Fetch side exception influences earlier instruction execution

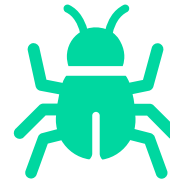
#136, #137, #170 Missed illegal exceptions

#159 Wrong PMP computation

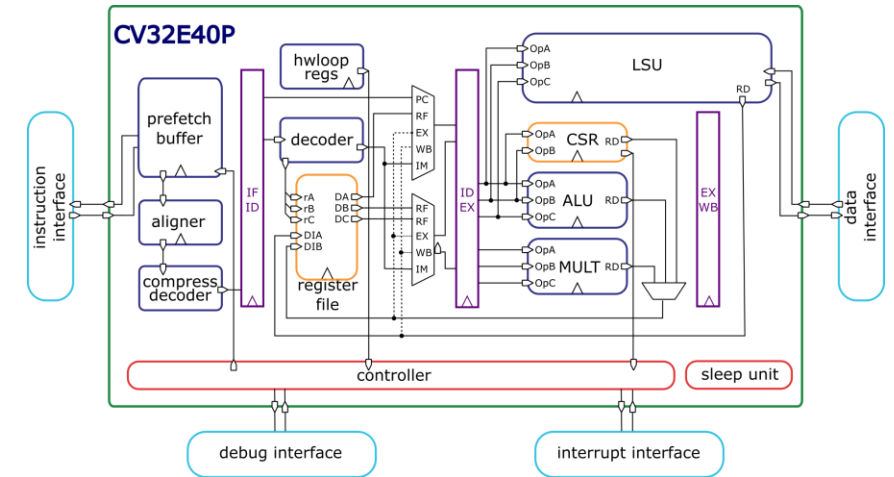
#174 Wrong F instruction result written to register file

#533 Illegal instruction retires

24



RV32IMC[F]_Zicsr_Zifencei[_Xpulp] Core



- 4-stage single-issue in-order pipeline
- Partial support for privileged spec 1.10
 - User Mode & physical memory protection
- Many custom instruction set extensions
 - Post-incrementing load/store instructions
 - Hardware Loops - ALU instructions
 - Multiply Accumulate - SIMD extensions

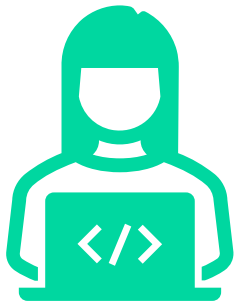
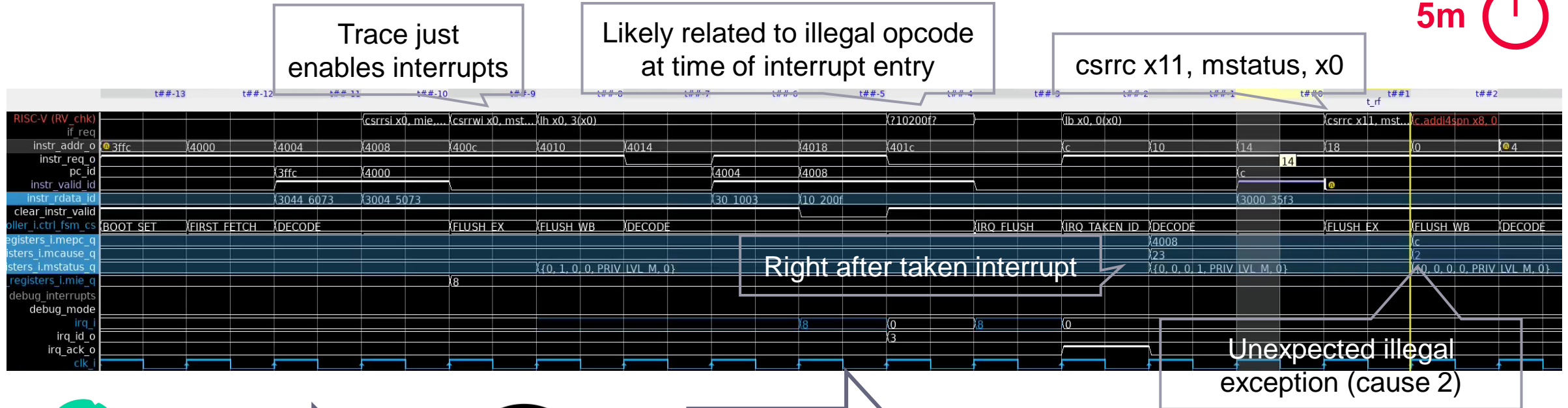
*<https://github.com/openhwgroup/cv32e40p/issues>

CV32E40P Bug Case Study

Systematic bug detection

Assertion CSRx_a verifies complete architecture register update for 6 CSR instructions

5m 



Issue #440



Confirmed and fixed
Closed



Exhaustive Proof

10m 



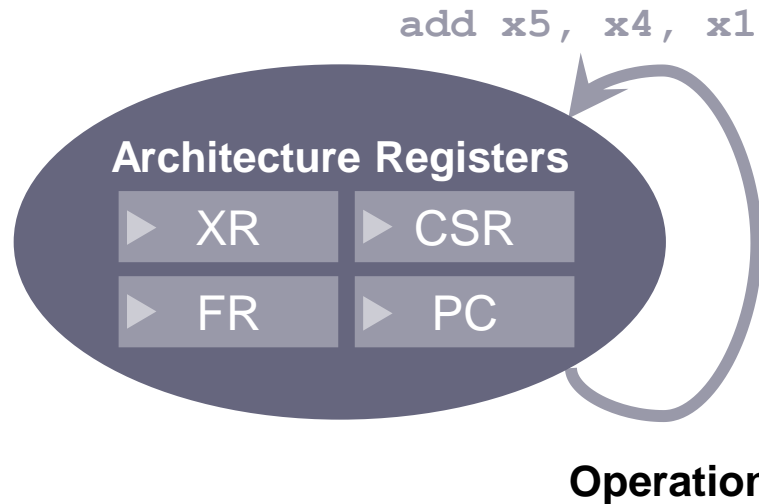
Underlying Technology

Instructions Mapped to Operational Assertions

OneSpin 360's unique modelling principle

Overall specification broken down into list of operations
List of operations forms complete specification

1. Transition **triggered** by some inputs and architecture registers
2. Operation defines **outputs** and new values for architecture registers
3. New values of architecture registers become current values for **next operation**

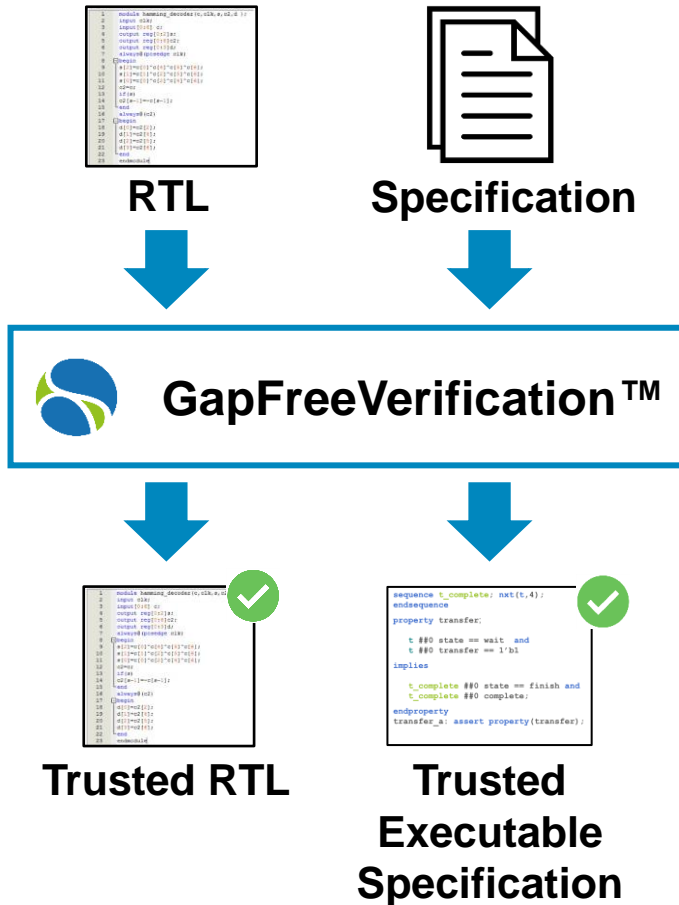


Operational assertions for all operations enable GapFreeVerification

```
property add_p;  
    t_id ##0 start_state and  
    t_id ##0 add_instr and  
    ...  
implies  
    ...  
    t_wb ##1 x5==result and  
    t_wb ##0 end_state;  
endproperty  
  
add_a: assert property(add_p);
```

The tool automatically considers all sequences of operations and all combinations of trigger conditions

Only a Few Operational Assertions for each instruction, exception, interrupt



GapFreeVerification™

Develop executable specification in the form of assertions

Prove that executable specification has no gaps or inconsistencies

Prove that executable specification and RTL are functionally equivalent

Abstraction

Specification, RTL

Scope

New IP, critical IP, untrusted 3PIP

Benefits

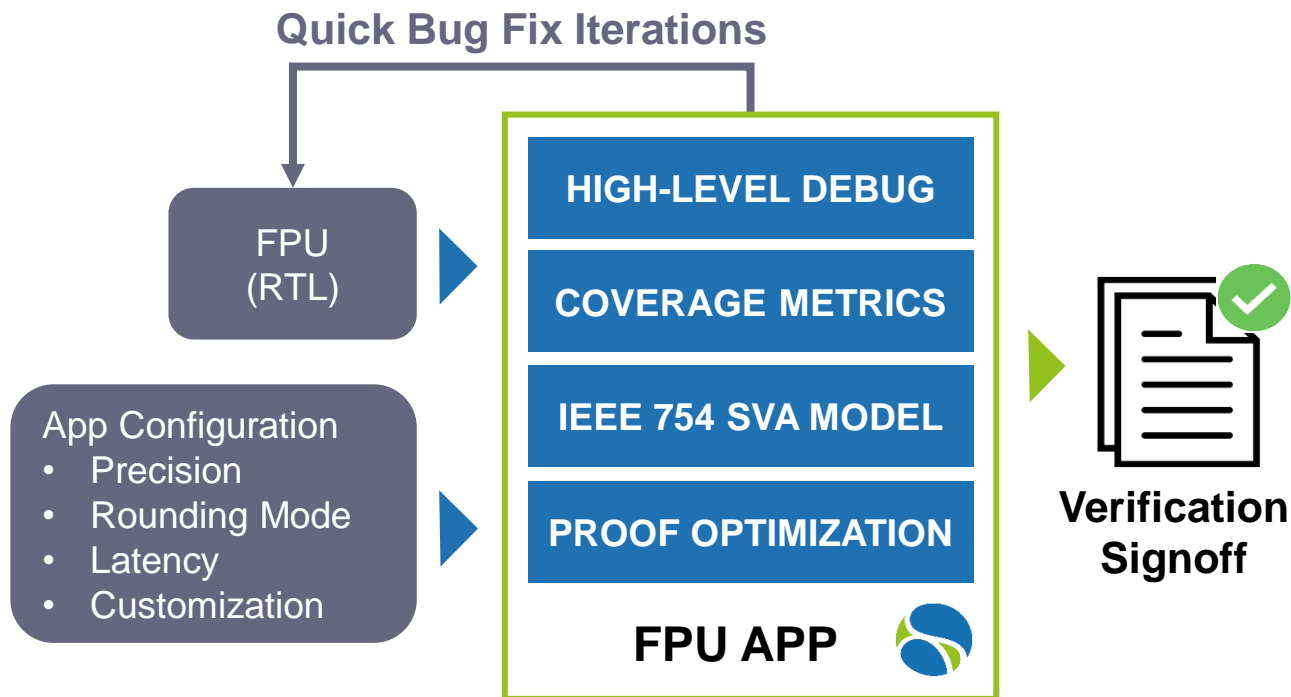
Detects errors and inconsistencies in the specification

Prove 100% equivalence between specification and implementation

- Demonstrate absence of bugs/ Trojans/ ambiguities

RISC-V FPU Verification

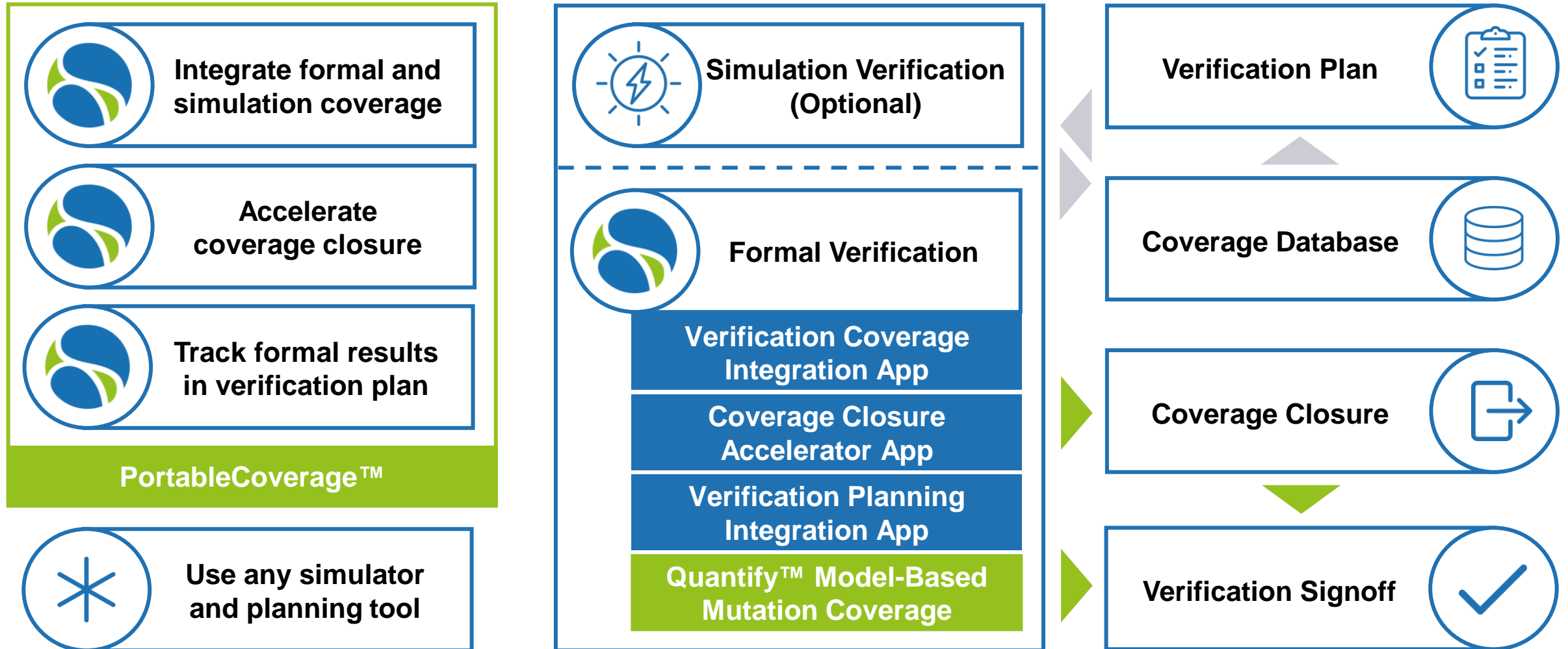
Pre-packaged configuration of OneSpin's FPU App



- Supports half/single/double bfloat16 and custom precisions
- Supports all 5 rounding modes and 5 exceptions flags
- add, sub, mul, fma, abs, neg, min, max
- Conversion and comparison operations
- Parameters to specify ambiguities in the standard
- Easy to model intended deviations from the IEEE-754 standard

Integrate Coverage Results

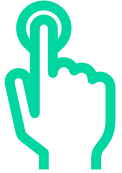
Speed-up coverage closure



OneSpin's RISC-V Processor Core Verification App



>10x improvement
on verification **setup & runtime**



High Degree of Automation
Enabling **automated verification** of generated core from common spec



Exhaustive and Complete
verification leaves **no bugs & exposes vulnerabilities**



Superior & Unique
Leveraging OneSpin's expertise & unique **Interval Property Checking & GapFreeVerification™**



CV32E40P is the first open-source core for high-volume chips verified with the state-of-the-art process required for high-integrity, commercial SoCs. OneSpin is a key contributor. The OneSpin RISC-V Integrity formal verification solution has systematically detected corner-case bugs in the exception logic and pipeline. These issues would only be triggered under rare conditions in the instruction sequence, memory stalls, and CSRs programming. **Constrained-random simulation tests to find these issues would require large investments in development and simulation time. The pinpointing of the issues' root cause was impressive and a massive time-saver in debug time.** The solution also showed almost **zero noise** in detecting real RTL bugs, as opposed to other approaches where the issues reported often lead to fixes in the verification environment.

Steve Richmond - Verification Manager - Central R&D Digital at Silicon Labs

Arjan Bink - Principal Architect - IoT Digital Systems at Silicon Labs

| Contact

Published by Siemens EDA

Salaheddin Hetalani

Field Application Engineer

Nymphenburgerstr. 20a

80335 Munich

Germany

E-mail salaheddin.hetalani@siemens.com

Disclaimer

© Siemens 2022

Subject to changes and errors. The information given in this document only contains general descriptions and/or performance features which may not always specifically reflect those described, or which may undergo modification in the course of further development of the products. The requested performance features are binding only when they are expressly agreed upon in the concluded contract.

All product designations may be trademarks or other rights of Siemens AG, its affiliated companies or other companies whose use by third parties for their own purposes could violate the rights of the respective owner.