SYNOPSYS®

# RISC-V Processor Verification Requires the Full Toolbox

Larry Lapides, Exec. Director, RISC-V Tools Bus. Dev.

Verification Futures UK, 2025

# Agenda

- Where and why is RISC-V being used?
- RISC-V processor verification challenges
- The RISC-V processor verification solution: the full toolbox
- Dynamic verification, including test generation and hardware assisted verification
- Formal verification
- Summary

# Agenda

- **Where and why is RISC-V being used?**
- RISC-V processor verification challenges
- The RISC-V processor verification solution: the full toolbox
- Dynamic verification, including test generation and hardware assisted verification
- Formal verification
- Summary

# Where and why is RISC-V being used?

Anyone can design their own processor based on the RISC-V ISA

Modular ISA = choice of which features to include/exclude

Extensibility and freedom to customize at ISA and micro-architectural levels

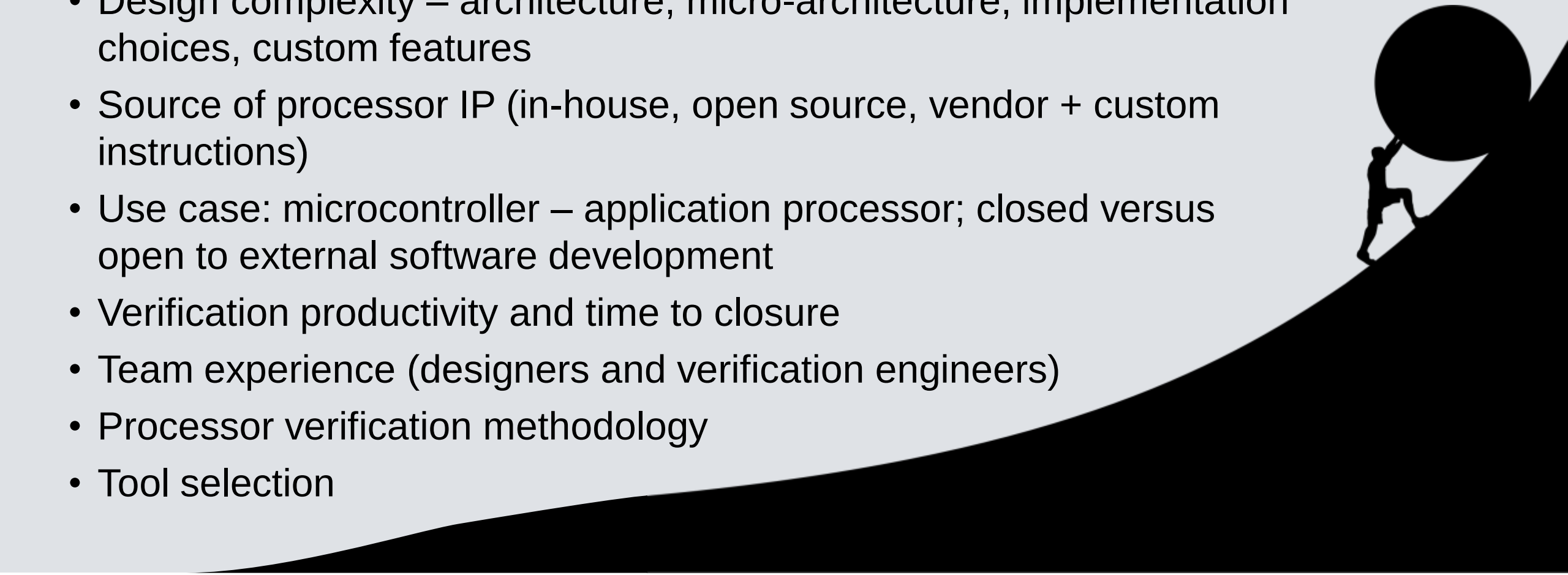
**RISC-V enables the creation of domain-specific differentiated processors**

# Agenda

- Where and why is RISC-V being used?
- **RISC-V processor verification challenges**
- The RISC-V processor verification solution: the full toolbox
- Dynamic verification, including test generation and hardware assisted verification
- Formal verification
- Summary

# Challenges in RISC-V Processor Verification

- Design complexity – architecture, micro-architecture, implementation choices, custom features
- Source of processor IP (in-house, open source, vendor + custom instructions)
- Use case: microcontroller – application processor; closed versus open to external software development
- Verification productivity and time to closure
- Team experience (designers and verification engineers)
- Processor verification methodology
- Tool selection



# RISC-V Processor Verification Process

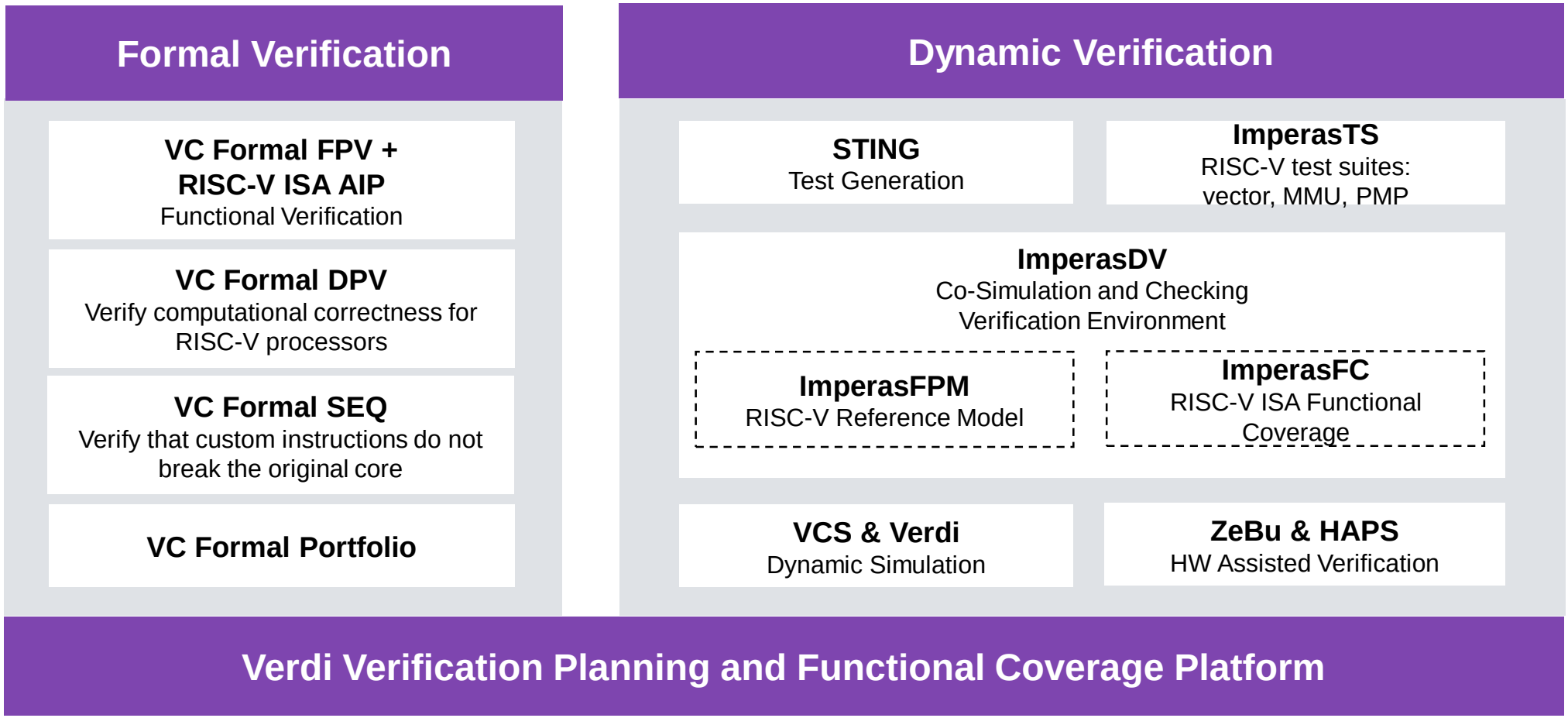
Design verification from unit to SoC

| Design Level | Example | Tool/Methodology |
|---|---|---|
| Unit | Pipeline, FPU | Formal + predefined assertion IP |
| | Security | Formal + predefined security assertion IP |
| Architecture | ISA | Dynamic |
| | | Formal + predefined assertion IP |
| Custom instructions, CSRs | Custom DSP, matrix | Dynamic |
| | | Formal sequential equivalence checking, register verification, datapath validation |
| Processing subsystem | Coherent cache, multi- or many-processor accelerator | Dynamic, especially using hardware assisted verification |
| | | Formal property verification for cache coherence verification |

# Agenda

- Where and why is RISC-V being used?
- RISC-V processor verification challenges
- **The RISC-V processor verification solution: the full toolbox**
- Dynamic verification, including test generation and hardware assisted verification
- Formal verification
- Summary

# The RISC-V Processor Verification Toolbox



# Agenda

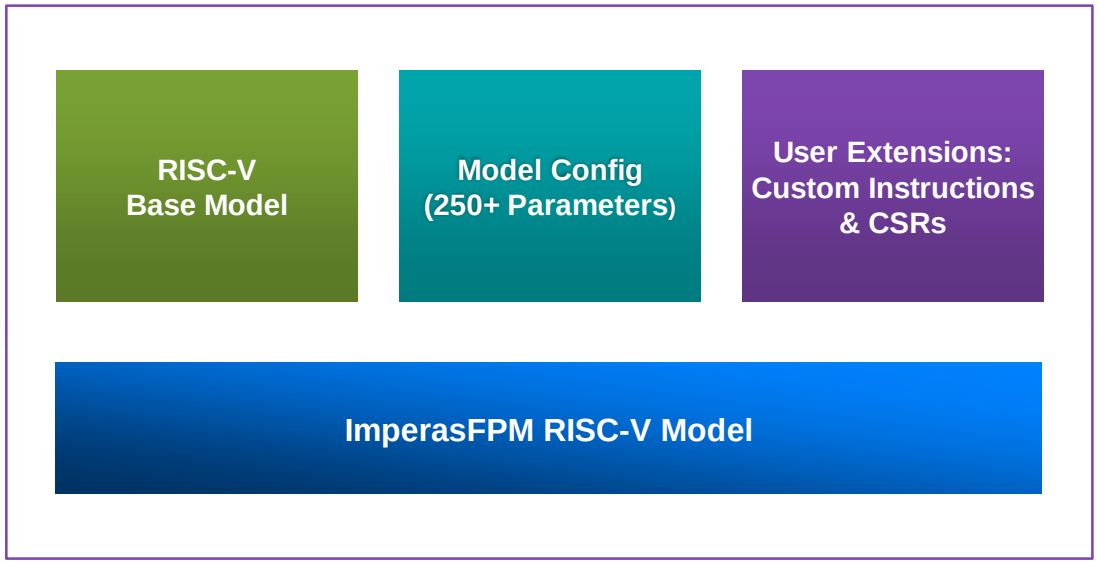
- Where and why is RISC-V being used?
- RISC-V processor verification challenges
- The RISC-V processor verification solution: the full toolbox
- **Dynamic verification, including test generation and hardware assisted verification**
- Formal verification
- Summary

# Fast Processor Models for RISC-V: ImperasFPMs

## **Use Cases**: Software Development and RISC-V Processor Verification

**SW Development w/ Virtual Prototypes**

Shift left software development using the industry leading processor model




**Processor Verification Solution**

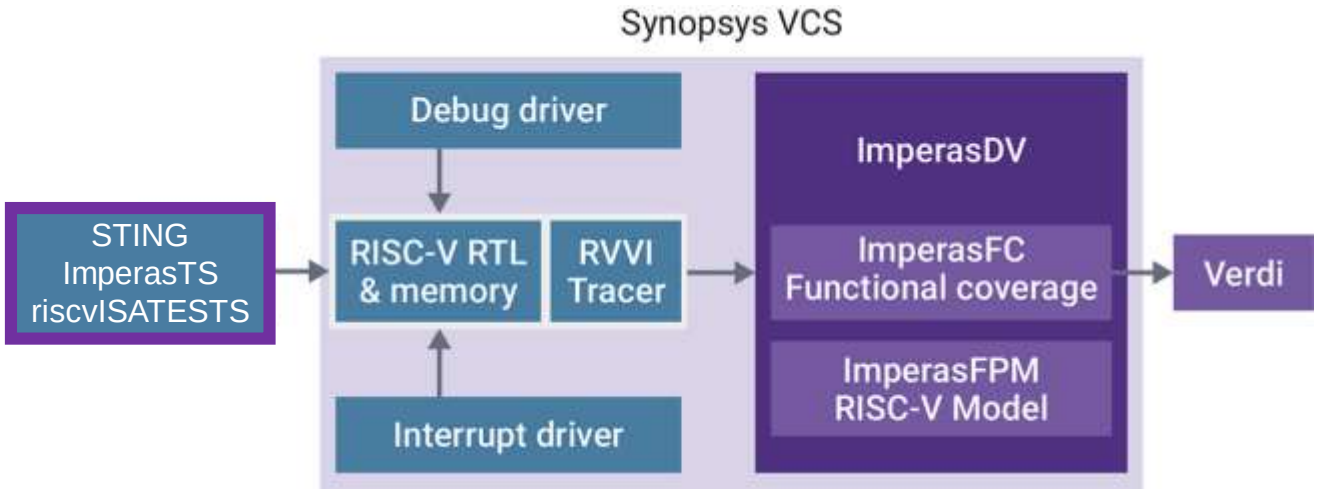
Reference model used for verification of RISC-Vprocessors instantiated in more than 50 SoCs

**Using the same model for both HW & SW verification enables significant reduction in SoC "Bring-Up" time**

# ImperasDV RISC-V Processor Verification Solution

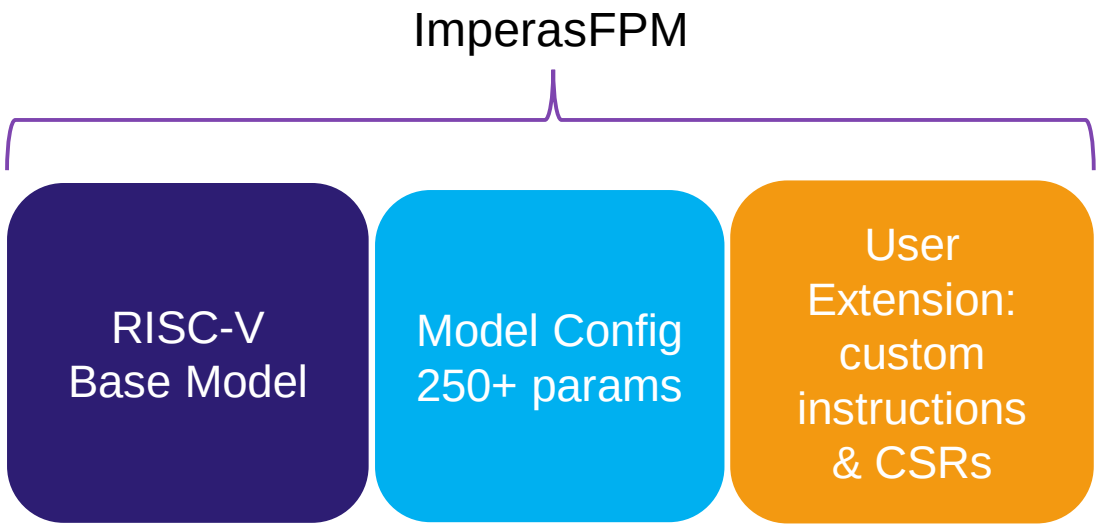
Enabling complete and comprehensive processor verification

- Configurable, extendable RISC-V reference model
- RISC-V RTL simulation in VCS
- Scoreboarding and checking
- Functional coverage output to Verdi




VCS with ImperasDV solution reduces RTL risk and accelerates verification schedule

# ImperasFPMs (Fast Processor Models) for RISC-V




- Base Model implements RISC-V specification in full
- Fully user configurable to select ISA extensions and versions
- Pre-defined configurations and custom instructions for processor IP vendors
- User extensions built in a separate library do not perturb the verified Base Model, help reduce maintenance
- Because every ImperasFPM uses the RISC-V Base Model, and including users of both commercial and free tools, over 150 companies, organizations and universities have used the ImperasFPM

# ImperasFC: SystemVerilog Functional Coverage for RISC-V

- Functional coverage code generation
  - Manual creation would be tedious, time consuming and error prone
  - >100K lines of code
  - Synopsys tools can automatically generate functional coverage code for custom instructions
- Functional coverage is the key verification metric


Machine-readable RISC-V ISA specification

SystemVerilog coverage code generator

https://github.com/riscv-verification/riscvISACOV/tree/v20240124/documentation for list of covered extensions

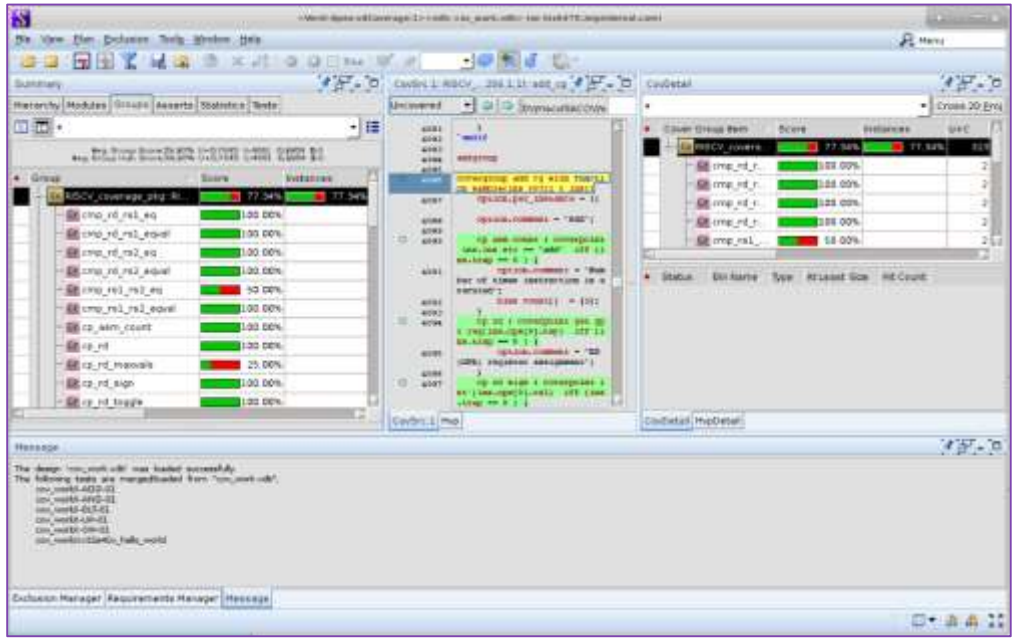
# Integrating ImperasDV with Verdi

## riscvISACOV: RISC-V SystemVerilog Functional Coverage: RV32I

ISA Extension: RV32I
Specification: I Base Integer Instruction Set
Version: 2.1
XLEN: 32

Instructions: 37
Covergroups: 37
Coverpoints total: 438
Coverpoints Compliance Basic: 204
Coverpoints Compliance Extended: 234

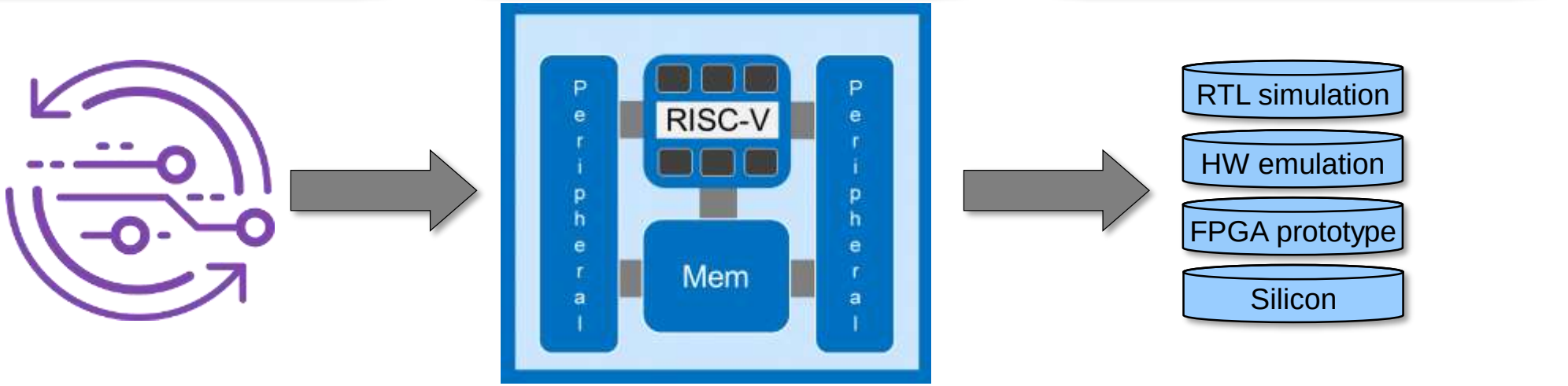
| Extension | Subset | Instruction | Covergroup | Coverpoint | Coverpoint Description | Coverpoint Level |
|---|---|---|---|---|---|---|
| RV32I | | addi | addi_cg | cp_asm_count | Number of times instruction is executed | Compliance Basic |
| | | | | cp_rd | RD (GPR) register assignment | Compliance Basic |
| | | | | cp_rd_sign | RD (GPR) sign of value | Compliance Basic |
| | | | | cp_rs1 | RS1 (GPR) register assignment | Compliance Basic |
| | | | | cp_rs1_sign | RS1 (GPR) sign of value | Compliance Basic |



- Auto-generated documentation in markdown and csv formats for inclusion in Verification Plans
- Functional coverage data is reported in verification tools such as Verdi

# STING Generates Tests for RISC-V Processors and Systems

**STING** | **RISC-V design** | **Target Platforms**




Generate **constrained random, directed and graph-based tests**

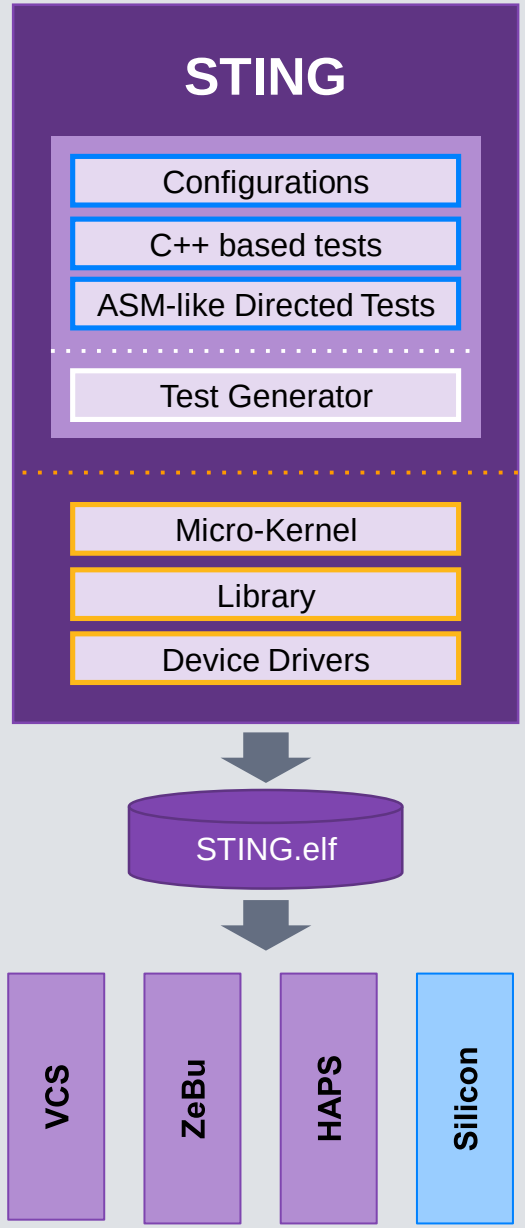
Verify **RISC-V ISA, custom extensions, multi-hart, memory coherence, concurrency**

Test on **multiple platforms** including silicon devices

# STING

## Preventing bug escapes for complex RISC-V designs

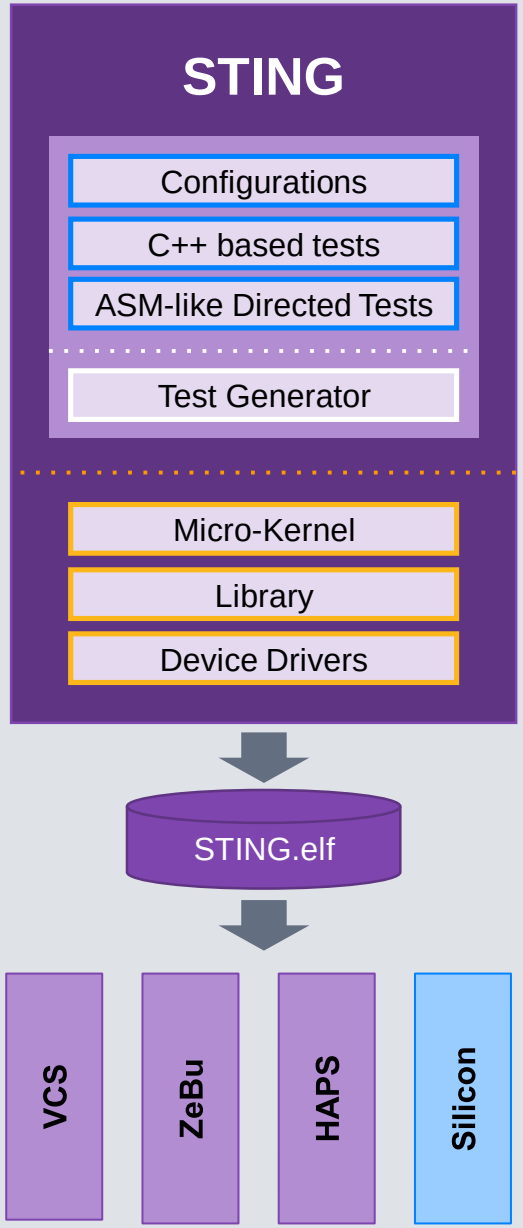
- ★ Bare metal tool using a software driven methodology for RISC-V design verification
- ★ Integrates several test generation methodologies to give the best verification throughput
- ★ Highly scalable and quick test generation; Compatible with any system configuration/memory map; IoT/embedded to server class; MP-ready
- ★ Self-checking architecturally correct stimulus portable across simulation, emulation, FPGA and silicon



# STING

Preventing bug escapes for complex RISC-V designs

- ★ Complete support for 32-bit and 64-bit RISC-V base integer extensions along with all standard ratified extensions and several un-ratified ones
- ★ Comprehensive coverage of privilege specification - MMU, PMP, PMA, Hypervisor, Supervisor, CSRs; Ready for RVA22 and RVA23 profiles
- ★ Used for architectural compliance and functional testing of several proprietary and open source RTL designs and models

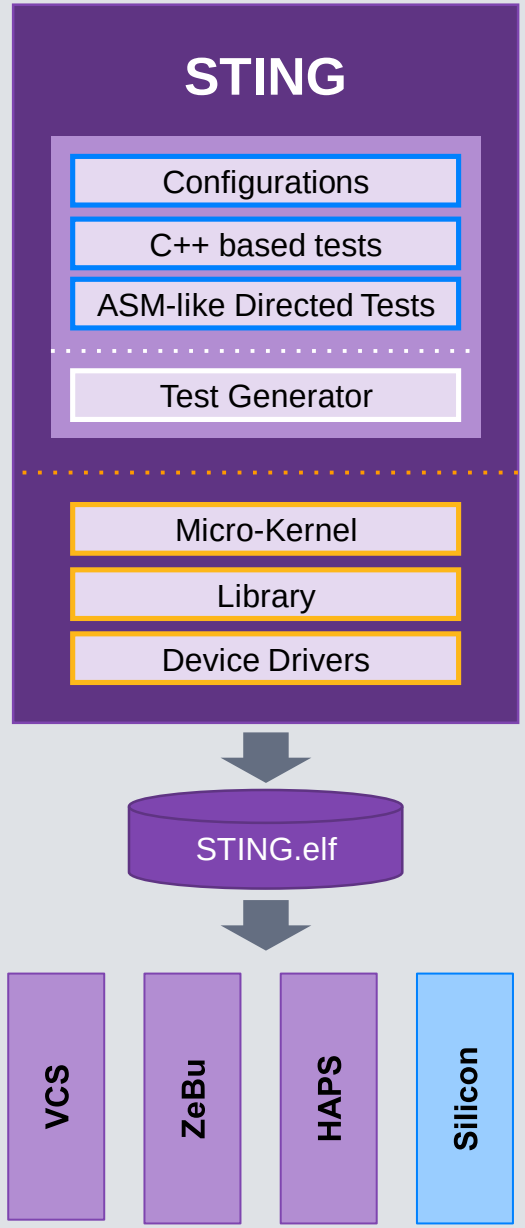


# STING

Preventing bug escapes for complex RISC-V designs

- Self-checking test generation for RISC-V
- Addressing single CPU and complex many core SoC designs
- Generates constrained-random, directed stimulus, and combinations of the two
- Portable across simulation, emulation, prototyping and silicon
- Full support for the RISC-V ISA specification
- Extensible to custom instructions and peripheral devices

- ImperasDV adds comprehensive checking and functional coverage when STING output is used in VCS



# STING - Use Cases

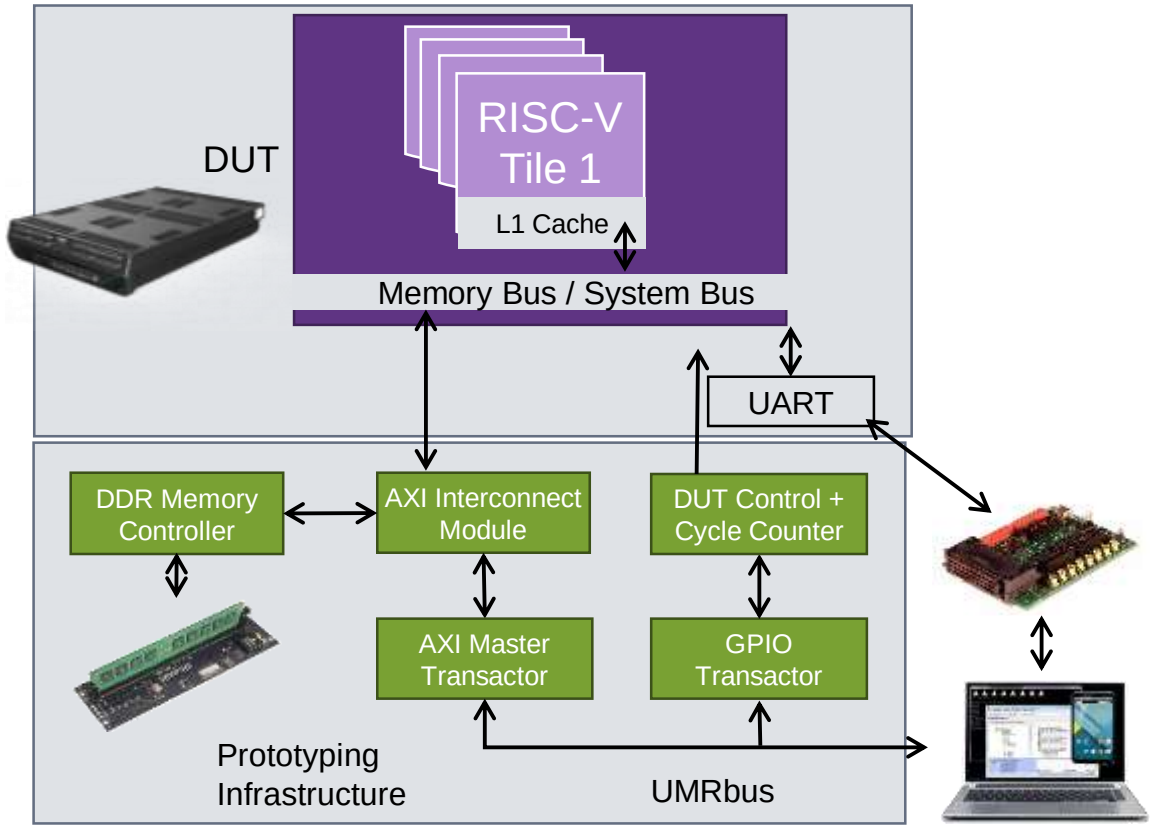
- ★ Verifying the functionality and architectural compliance of RISC-V extensions (several of which are not ratified)
- ★ Sweeping through several CPU configurations for RISC-V core vendor companies
- ★ Security extensions - WorldGuard, PMP, Smepmp
- ★ Privilege specification - Machine, Supervisor, User and Hypervisor extensions, MMU, PMP, PMA
- ★ Testing multicore systems with device interactions
- ★ Specialized workloads for branch, load stores, floating point, memory ordering, forward progress, caches

# STING - Bugs Found

- *"**Deadlock condition** existed when a TLB Miss for an older load/store instruction waits for its page-table-walk which cannot complete because newer stores have been issued and filled up certain miss-handling buffers in the load/store unit. This was uncovered by STING exercising streams of loads/stores with virtual memory enabled."*
- *"Design had an **optimization issue** to convert a conditional branch over a single instruction into a predicated operation. There was a corner case bug in the implementation of this logic which used to cause **register corruption** when the 2 instructions (a "branch" and a "move") were separated by a pipeline flush in some scenarios."*
- *"**Page table walk returning incorrect address translations** due to a bug in flushing of newer instructions when an older flush was taking place in the same cycle."*
- *"**Stall condition** when a multiply instruction was in progress converted caught a pipeline issue in multiply unit that converted one type of multiply to another type of multiply."*
- *"Back-to-back divides preceded by a long-latency memory bus read **caused the second divide to hang.**"*
- *"STING **found a lot of nuances with floating-point rounding modes** and signaling/quiet NaNs. RISC-V has some quirks particularly with respect to the sNaN/qNaN handling."*
- *"After tuning for our cache configuration, STING **did a good job stressing the cache controller.** Made sure a **lot of cache conflicts** were occurring. We support multiple outstanding misses so it found things like window conditions when one outstanding miss was getting filled and a request to that same line was being handled by the LSU. Windows around when data is available vs. directory state through the pipeline. Some windows that led to a cache line getting fetched and filled with "old" data while a write-back with new data was in progress."*
- *"Few privileged **CSRs were getting sign-extended incorrectly** for some of the trap exceptions."*
- *"**Unexpected execution** of instructions and trap exceptions **in the shadow of branch**"*
- *"Issues with fence.i implementation resulting in **incorrect execution of self modifying code sequences**"*
- *"Not found directly by STING: but much of the testing is built upon running STING tests while applying external stimulus of various forms. Debug, interrupts, etc. **Having sufficiently interesting code being executed by STING while that external stimulus was on-going help find a number of good issues.**"*
- *"**PMP execute check wrong for the grain prior to a valid grain.** The problem occurs when attempting to execute from a PMP grain just prior to a configured PMP region. The defect lets the checks for the prior grain use the configuration of the next grain, which can cause exceptions to falsely fire, or falsely not fire."*
- *"Corner-case hang requiring a combination of: - Completed but uncommitted loads or partial stores in the LSQ. - A dram slave request hits the LSU, matching one of the entries from #1 - An outstanding device request."*
- *"**1 cycle window where wrong instruction text was serviced from the fetch buffer on a backwards branch,** relating to a specific case where an instruction cache line boundary is being crossed on the fetch buffer ingest side."*
- *"**FSM in the DCACHE not cleaning the state correctly** for consecutive custom instructions that cancels each other."*
- *"DCACHE not incrementing the "free entries" counter which leads to a counter leak that could **potentially block the core from doing any memory operation**"*
- *"**Thread 0 starves Thread 1** (of the same core) when both threads are using the same resources (VPU, ALU) and one of the threads is doing a long latency operation (e.g. div)."*
- *"**Livelock in the shared ICACHE** due to a bad LRU implementation."*
- *"**LRAM clock gating triggered too early** and caused some writes to be lost."*
- *"A pipeline optimization for multiplication operations results into a **deadlock condition**'*

# Hardware Accelerates Verification

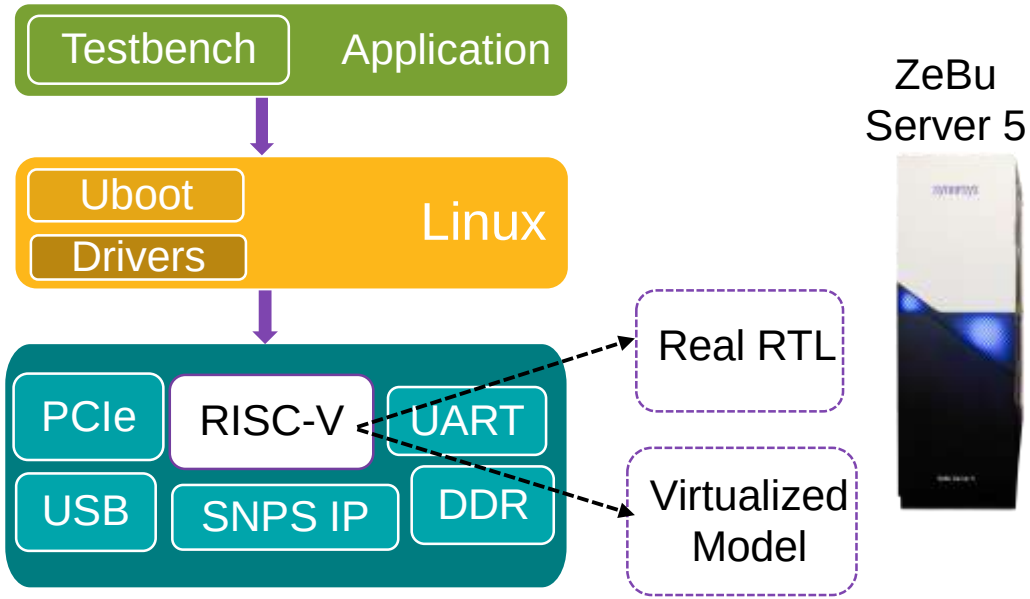
## HAPS Prototyping Solution




HW/SW debug with real ASIC | Unified RTL debug with Verdi |

4 RISC-V embedded cores running at 100MHz in one FPGA

## ZeBu Emulation




Emulation Bring-Up and Software Stack

# Agenda

- Where and why is RISC-V being used?
- RISC-V processor verification challenges
- The RISC-V processor verification solution: the full toolbox
- Dynamic verification, including test generation and hardware assisted verification
- **Formal verification**
- Summary

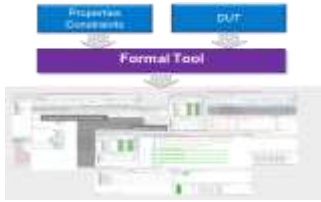
# Formal Verification: VC Formal

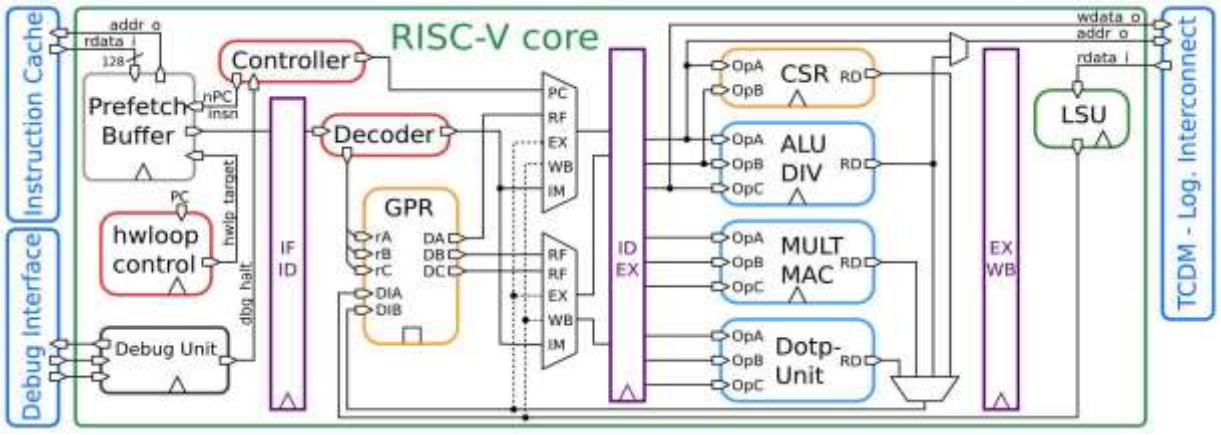
- Formal verification provides exhaustive proof of correct behavior
- Excellent tool for unit-level DV
  - Can get started early, even with design engineers
  - Unit-level includes pipeline, floating point unit, load/store unit, …
- RISC-V ISA Assertion IP (AIP) available to enable early use of VC Formal
- VC Formal Apps improve verification efficiency of many tasks
  - Register verification, datapath validation, connectivity checking, security verification …

VC Formal Apps

# RISC-V Core Unit Verification Task Examples Using Formal Property Verification

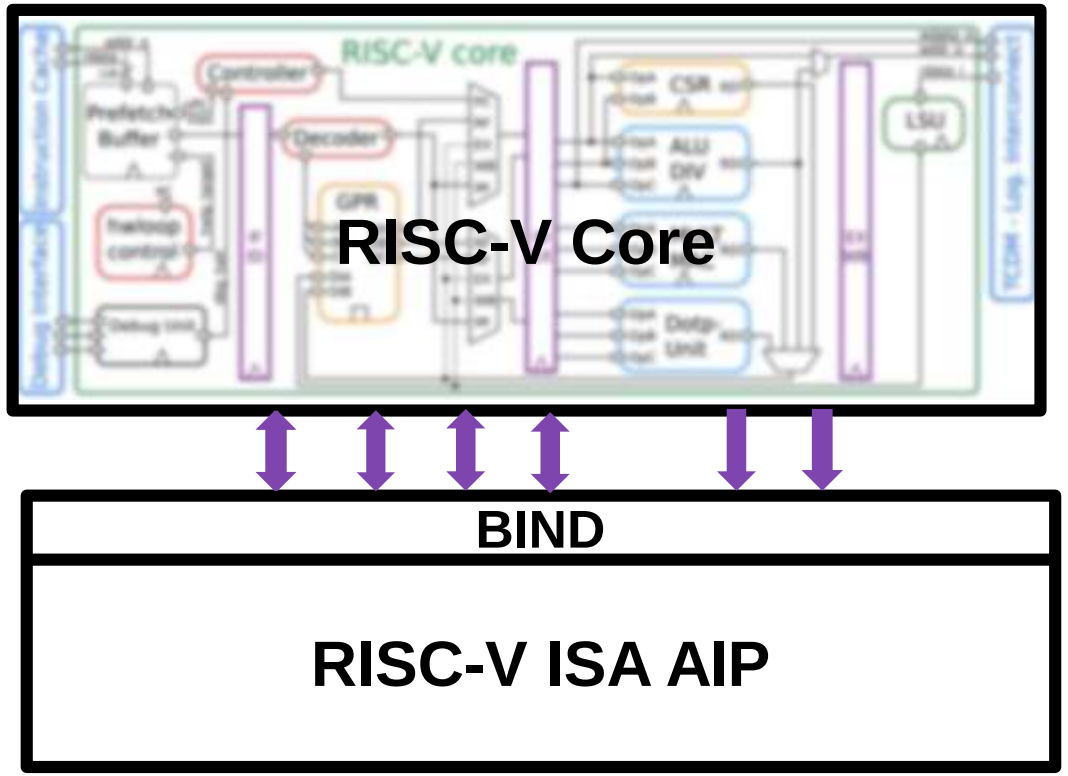

- Prefetch Buffer:
  - Redirect/Clear from various components: BPU/EX etc. should cause proper action and in a priority order
  - Instruction Cache
  - Direction/Target Prediction
  - Branch Target Buffer
  - Wake: Detecting a ready instruction
  - Dispatch: Need to select (oldest woken-up instruction first)
  - Resolve Dependencies
- Decoder
  - Check for undefined instructions
  - Fusing check if 2 or 3 instructions can be used together
- Execution (ALU):
  - Simple ALU functions
  - Bypass Functionality Checking
  - Misprediction should lead to redirect; Correct prediction should result in completion
- Load/Store Unit (LSU):
  - Load addr should be sent before Load data
  - Store addr should be sent before store data
  - Load/Store Functionality
- Pipeline
  - Control logic




Source: https://www.semanticscholar.org/paper/Near-Threshold-RISC-V-Core-With-DSP-Extensions-for-GautschiSchiavone/47f8ce7e0f0f64d0707a13c83c32c30959aa64d5/figure/6

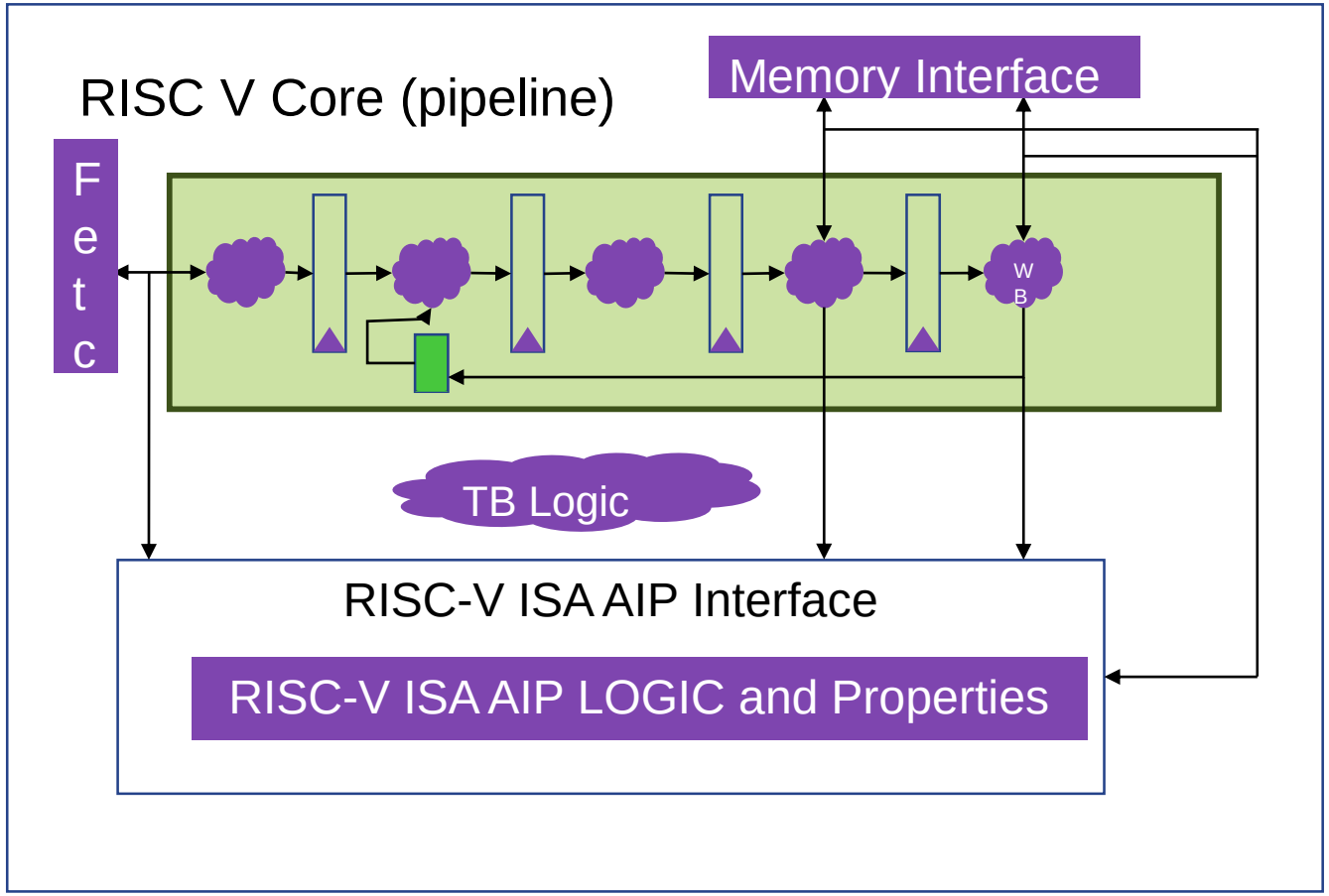
# VC Formal RISC-V AIP for Exhaustive ISA Verification




## Benefits of RISC-V ISA AIP Formal Verification

- Formal exhaustively tests all possible RISC-V instruction scenarios
- Availability of RISC-V ISA AIP reduces debug turn-around-time
- RISC-V ISA AIP validates instruction execution control and base-ISA data path
  - For complex math operations (MUL/DIV), will need DPV verification to ensure datapath correctness
- RISC-V ISA AIP can be used for multiple configurations and cores
- Verification quality and confidence are high

# Formal RISC-V ISA AIP Applied to Design




- RISC-V ISA AIP needs minimal access to a small number of points around the pipeline to observe certain events
- Interfaces to bind to RISC-V ISA AIP
  - Instruction Fetch Interface
  - Data Memory Interface
  - Instruction Retire Interface
  - Register Write Interface
- Testbench logic
  - DUT signal expressions to bind to RISC-ISA AIP interfaces
  - DUT-specific constraints
- RISC-V ISA AIP offers all the properties and policies for checking the instruction architecture

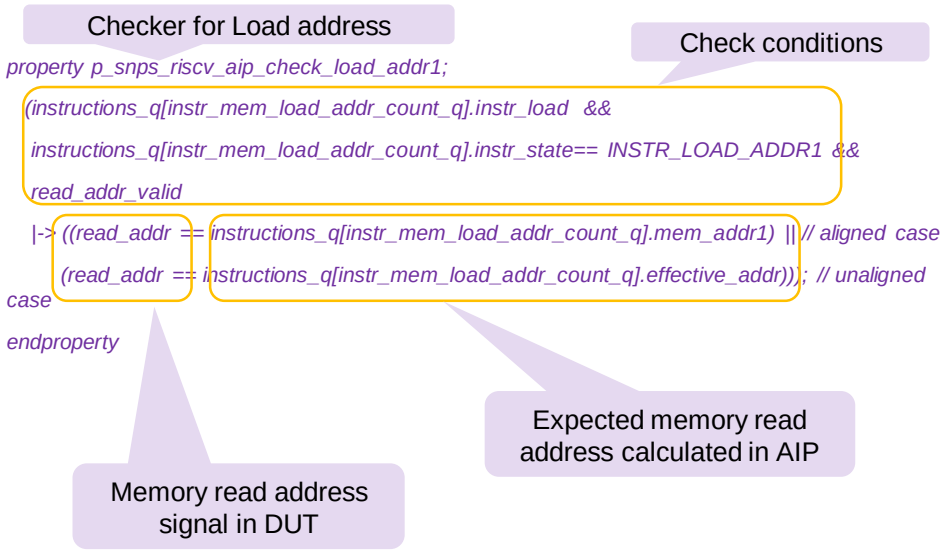
# Verification of the RISC-V ISA AIP

- The RISC-V AIP itself is verified using formal against a testplan extracted from the ISA spec
- The testbench contains a CPU core target to act as reference for our interpretation of the architecture
  - Any failures due to difference in interpretation would need resolving with the vendor
- Targets we have verified the AIP against in the past include:
  - OpenHW CV32E40P and CV32E40X
  - OpenHW CVA6 (as both 32bit and 64bit variants)
  - Internal Synopsys core
  - Internal Synopsys core models
  - Ibex
  - SweRV EH2
  - …

# RISC-V ISA AIP Checks

- Register write address (RD) check
- Register write data (Result) check – for the integer pipeline
- Load/Store (aligned/misaligned) address/data check
- RAW- Read after Write hazard check
- WAW- Write after Write hazard check
- Instruction fetch address check at fetch interface
- Race condition check on register file write interface with multiple ports
- AIP internal data structure pointer checks (for ease of debug)
- Check for legal config parameter values
- Highly configurable constraints model to match design

- Property example:



# RISC-V ISA AIP Functionality – Some Details

- Fetch address checks work around branch prediction by ensuring that all addresses come from what's possible within the program
  - RISC-V ISA AIP keeps track of all control flow instructions
- Instructions are only ready to be checked after all of their input operands are "available"
  - RISC-V ISA AIP keeps track of all hazards between instructions
  - We cope with multi-issue/out of order execution by precomputing results in decode (when possible) and by preserving program order
- Forwarding is implicitly checked because all source operands are known in advance from within the AIP
  - RISC-V ISA AIP keeps track of the most up to date value of all registers

# Examples of Bugs Found With RISC-V Formal AIP

| Bug description | FV runtime | Likely to find in simulation |
|---|---|---|
| Simultaneous writes to same destination register from stalled LOAD_FP retiring out of order with subsequent OP_FP | ~20 min | Low |
| RV32F LOAD_FP unexpectedly writing 64-bit floating point values to FP register file when core is configured as 64bit integer pipeline (RV64I) with RV32F – core overrides RV32F and instantiates 64bit FP pipeline (config bug) | ~20 min | High |
| A power optimization problem where inadvertent multiple register writes were seen for stalled or unaligned load | ~2 min | Med |
| Core fully executes instruction that was not requested and updates the integer register file – instr_read_valid without first instr_fetch_valid. Although protocol is violated, core does not protect the pipeline (expose security hole) | ~1 min | Med |

# Agenda

- Where and why is RISC-V being used?
- RISC-V processor verification challenges
- The RISC-V processor verification solution: the full toolbox
- Dynamic verification, including test generation and hardware assisted verification
- Formal verification
- **Summary**

SYNOPSYS®

# Thank You

Learn more at www.synopsys.com/RISC-V