SYNOPSYS®

RISC-V Processor Verification Requires the Full Toolbox

Simon Davidmann, VP Eng., Processor Modeling & Verification

Verification Futures UK, July 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
- Static / 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
- Static / 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

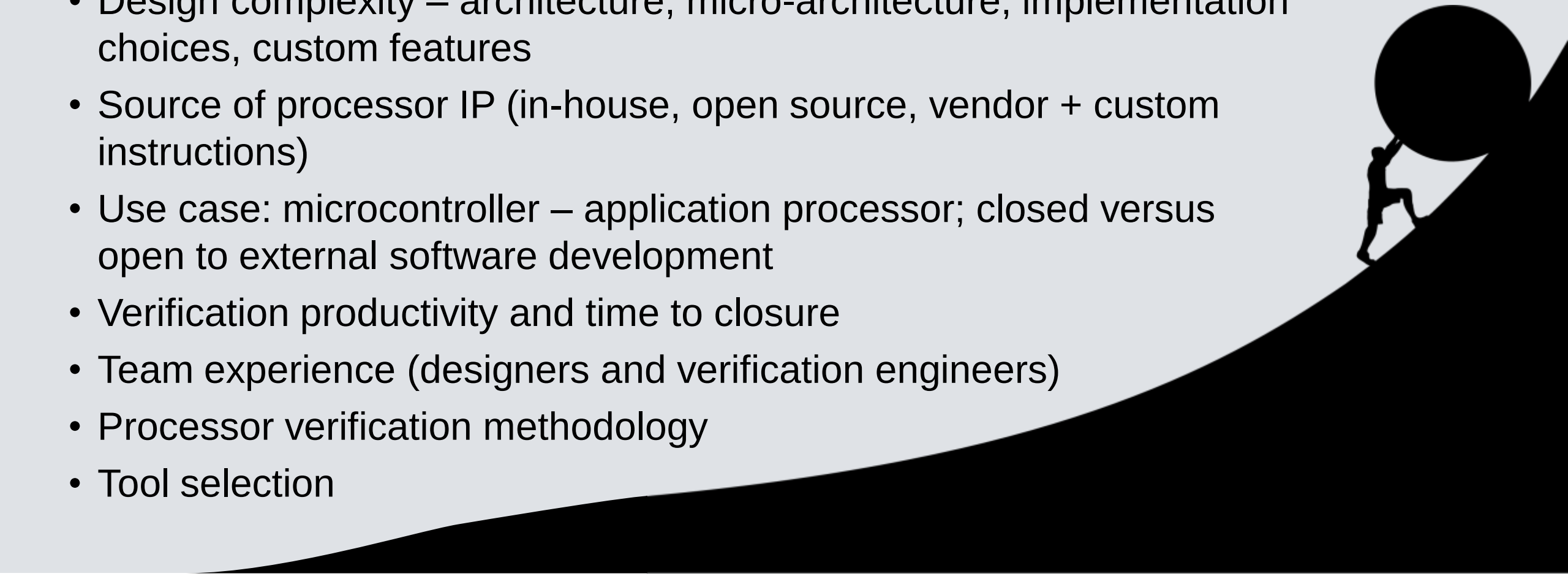
RISC-V enables the creation of domain-specific differentiated processors

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
- Static / Formal verification
- Summary

Challenges in RISC-V Processor Verification

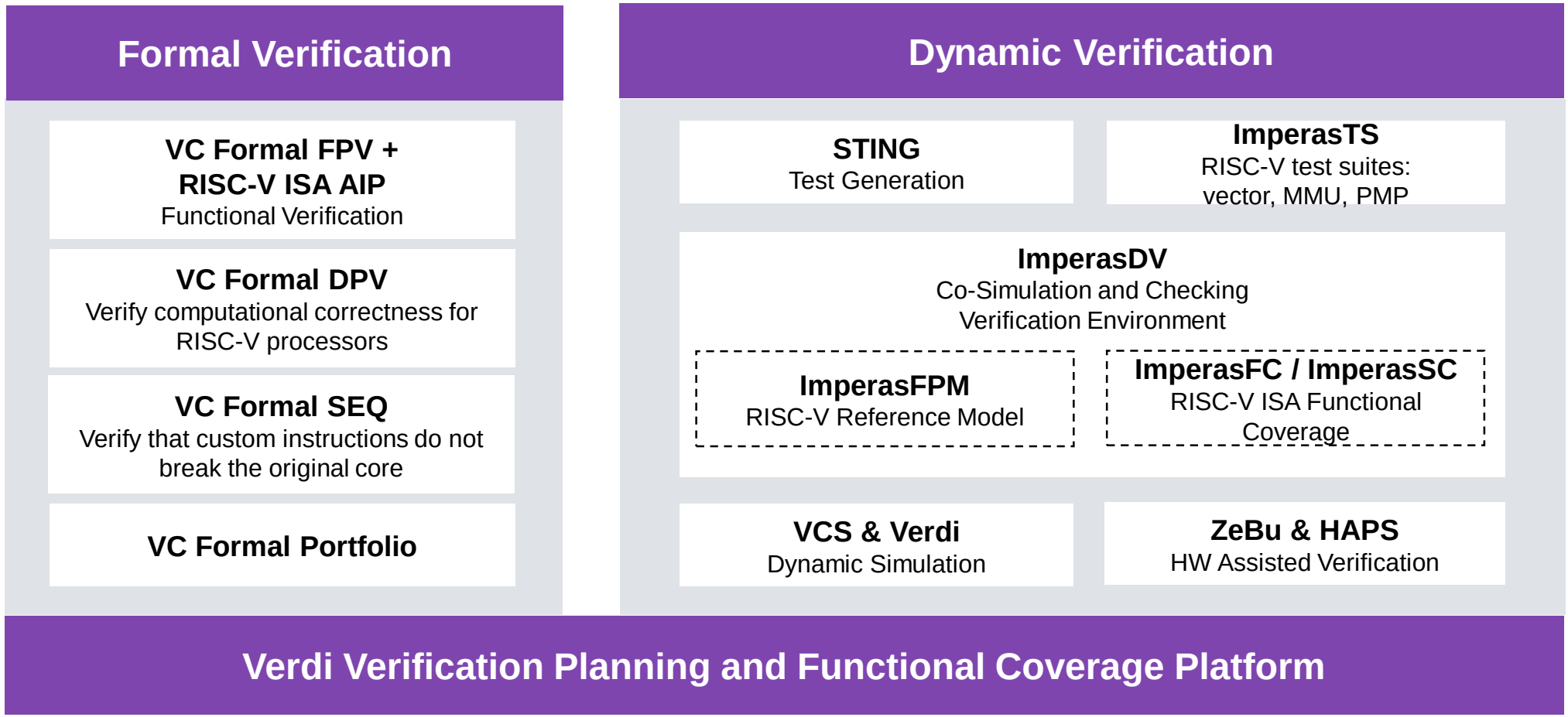
- Design complexity – architecture, micro-architecture, implementation choices, custom features
- Source of processor IP (in-house, open source, vendor + custom instructions)
- Use case: microcontroller – application processor; closed versus open to external software development
- Verification productivity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



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
- Static / Formal verification
- Summary

The Synopsys Processor Verification Toolbox



Tool & Methodology selection table shows how dynamic and formal verification complement each other

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**
- Static / Formal verification
- Summary

Agenda

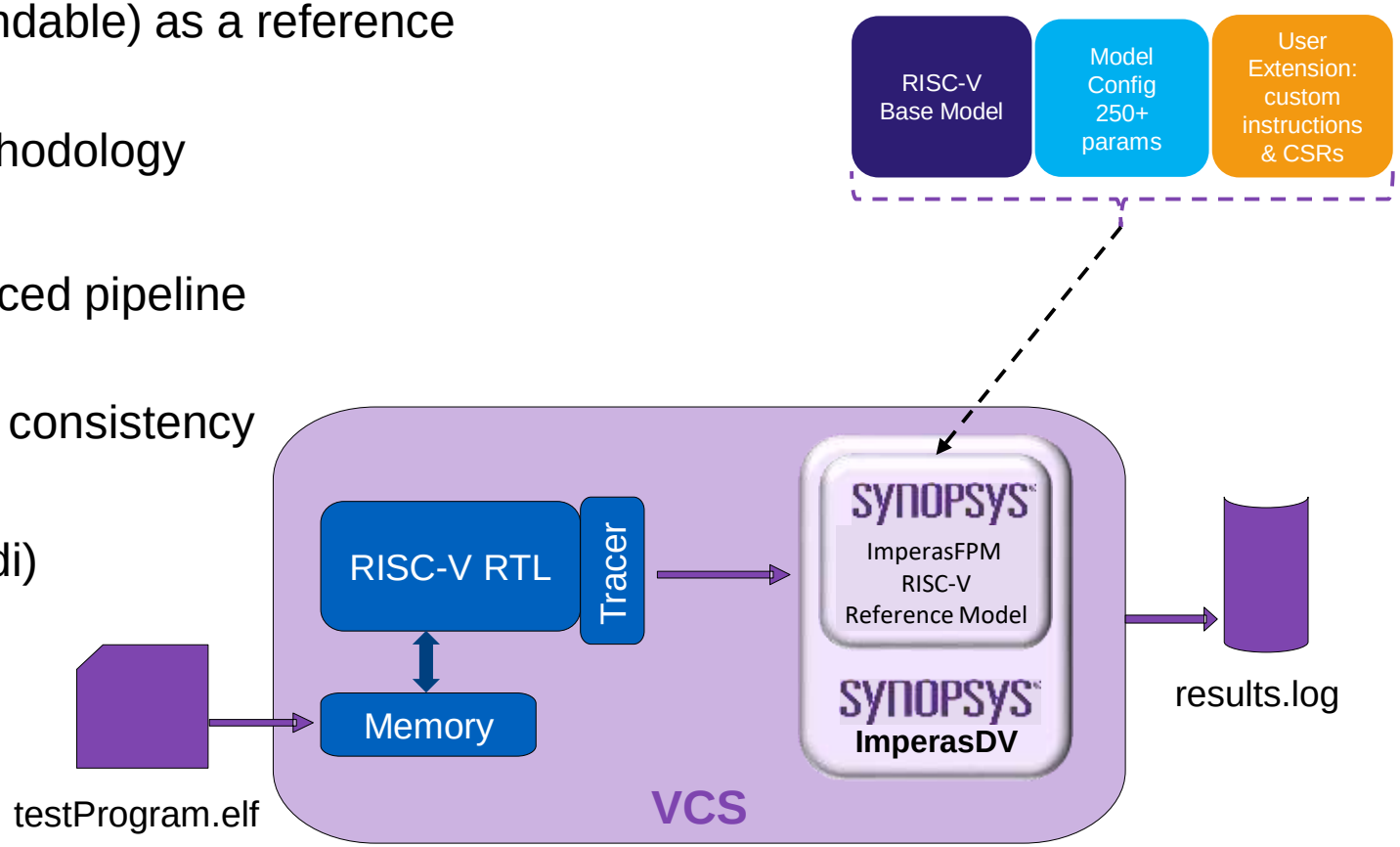
- Dynamic verification
  - ImperasDV, ImperasFPM (reference model)
  - Lock-step continuous compare, async events, weak memory models
  - ImperasFC (functional coverage), ImperasSC (stimulus coverage)
  - ImperasTS (directed test suites for compliance / compatibility)

Agenda

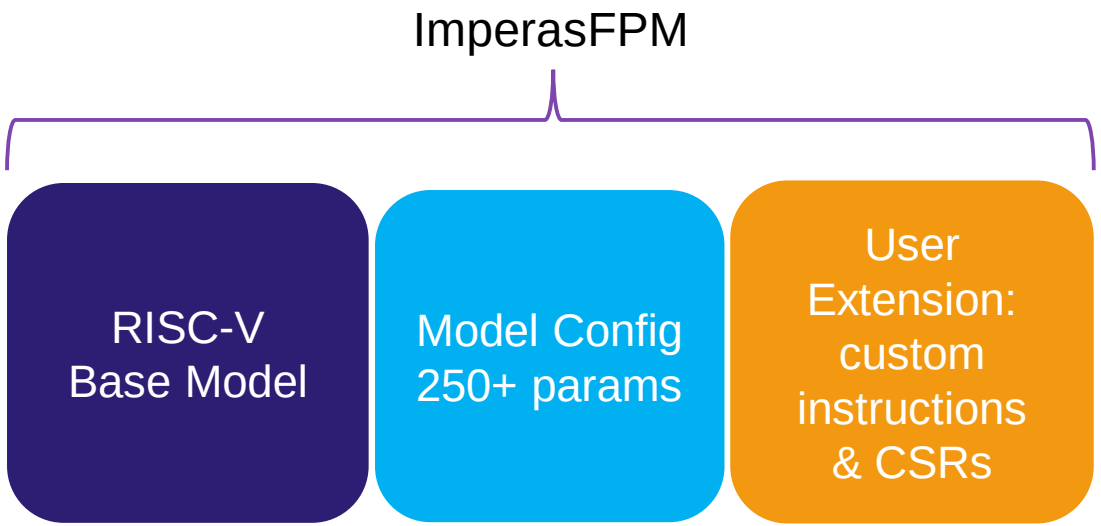
- Dynamic verification
  - **ImperasDV, ImperasFPM (reference model)**
  - Lock-step continuous compare, async events, weak memory models
  - ImperasFC (functional coverage), ImperasSC (stimulus coverage)
  - ImperasTS (directed test suites for compliance / compatibility)

What is ImperasDV?

- A front-end design verification solution for those designing a processor using the RISC-V ISA
- Uses the ImperasFPM (configurable, extendable) as a reference model
- Uses a lock-step continuous compare methodology
- Performs score-boarding and checking
- Verifies asynchronous events using advanced pipeline synchronization technology
- Verifies core with Weak or Strong Memory consistency
  - single- / multi-hart, with / without caches
- Functional coverage (with reporting in Verdi)



ImperasFPMs (Fast Processor Model) 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 source, help reduce maintenance
- Over 150 companies, organizations and universities have used the ImperasFPM
- ImperasFPMs can be used as reference model for verification, and also for software development with virtual prototypes (e.g. Virtualizer)

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

Agenda

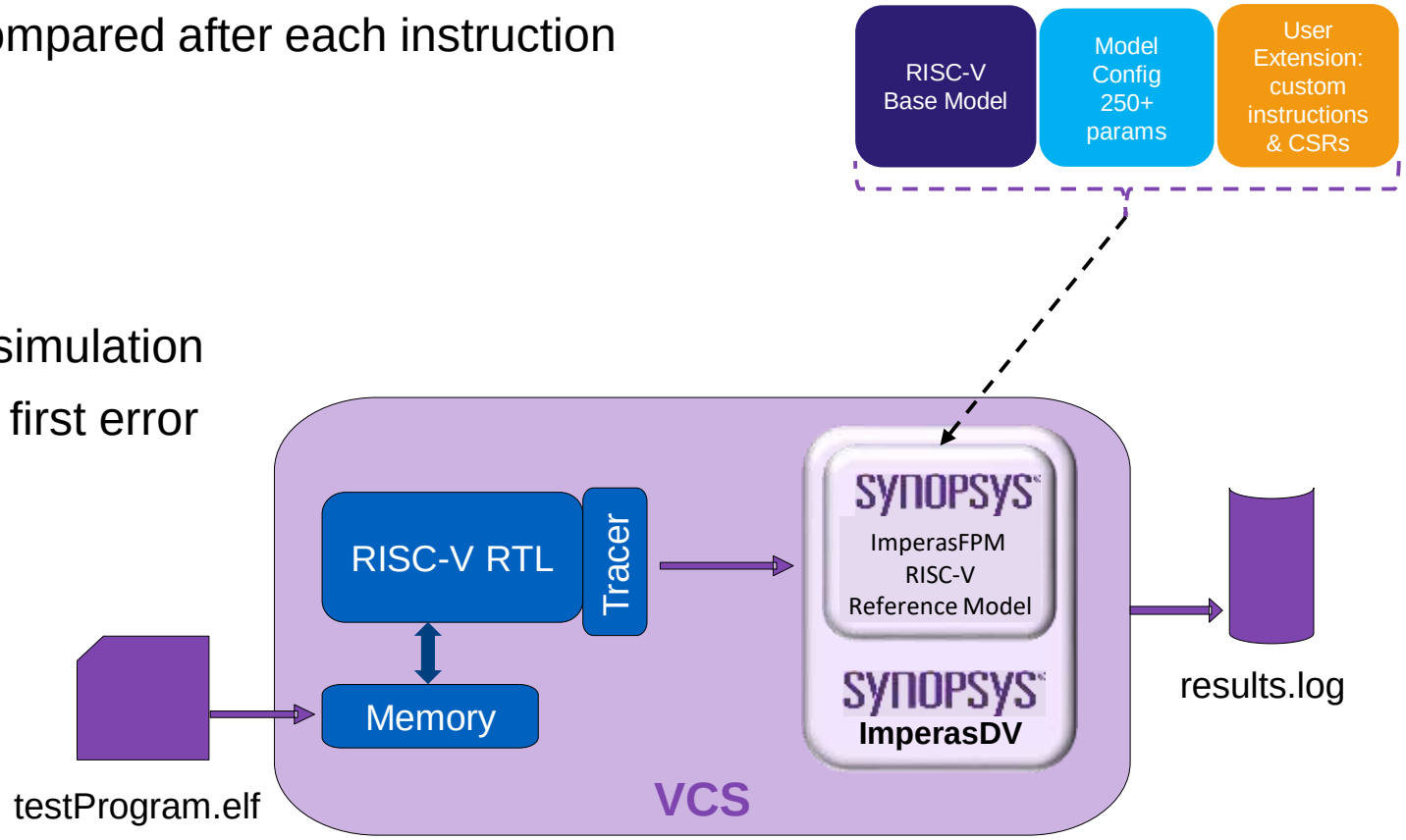
- Dynamic verification
  - ImperasDV, ImperasFPM (reference model)
  - **Lock-step continuous compare, async events, weak memory models**
  - ImperasFC (functional coverage), ImperasSC (stimulus coverage)
  - ImperasTS (directed test suites for compliance / compatibility)

What is Lock-Step Continuous Compare Methodology?

- Co-simulation approach: Reference model and RTL run in lock-step
- Internal state of reference and RTL are compared after each instruction retires

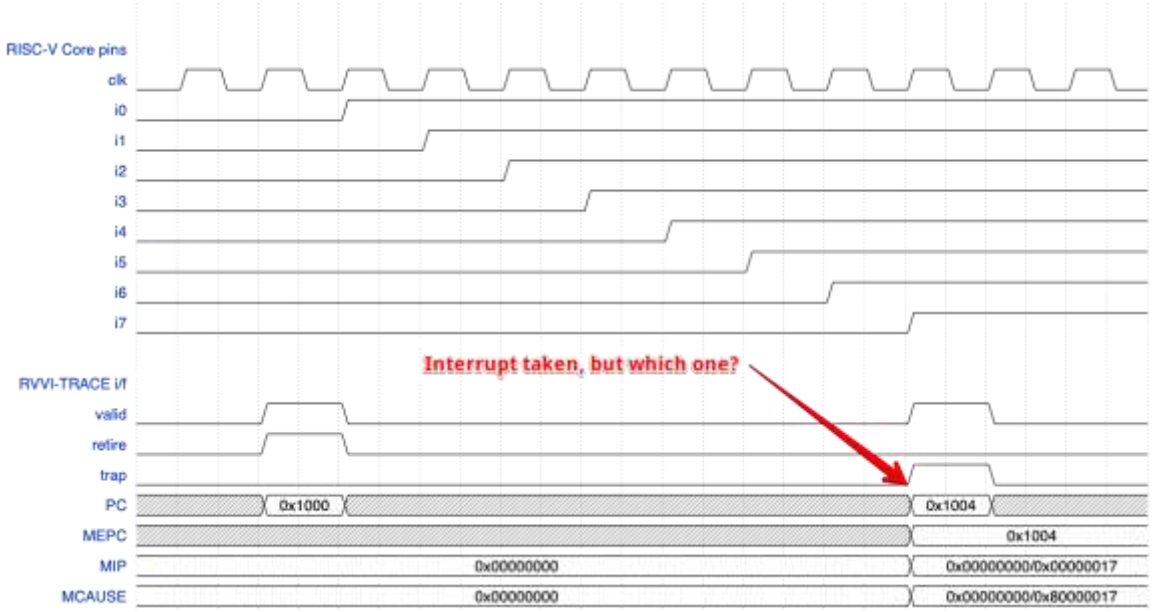
Advantages:

- Bugs reported immediately, not at end of simulation
- No wasted simulation cycles running past first error
- Easier to debug at the point of failure
- Most comprehensive DV methodology



Asynchronous event verification with ImperasDV

- An architectural model cannot predict exactly when an asynchronous event will take effect
- External net changes received as a set from RTL tracer interface (RVVI)
- ImperasDV Pipeline Synchronization technology determines whether RTL DUT response to these events is legal



Asynchronous event verification with ImperasDV

- An architectural model cannot predict exactly when an asynchronous event will take effect
- External net changes received as a set from RTL tracer interface (RVVI)
- ImperasDV Pipeline Synchronization technology determines whether RTL DUT response to these events is legal
- If not, an error is flagged and debug information is provided

```
Info (IDV_RPT) 3 legal scenarios have been evaluated:
Info (IDV_RPT) SCENARIO 1:
Info (IDV_RPT)  +  Exception { pc:0x1304(test2+4a), fetch:'904ff0ef jal     x1,408', 'Machine software in
Info (IDV_RPT)     +  Retire { pc:0xc(vector_table+c), inst:'5ca0006f jal     x0,5d6' }
Info (IDV_RPT)        ! dut.pc           - 0x0000001c vector_table+1c
Info (IDV_RPT)          ref.pc           - 0x0000000c vector_table+c
Info (IDV_RPT)        ! dut.mcause       - 0x80000007 Interrupt:1 Code:7(Machine timer interrupt)
Info (IDV_RPT)          ref.mcause       - 0x80000003 Interrupt:1 Code:3(Machine software interrupt)
Info (IDV_RPT)
Info (IDV_RPT) SCENARIO 2:
Info (IDV_RPT)  +  Exception { pc:0x1304(test2+4a), fetch:'904ff0ef jal     x1,408', 'Machine software in
Info (IDV_RPT)     +  Net MTimerInterrupt => 0 { when:37351 }
Info (IDV_RPT)        +  Retire { pc:0xc(vector_table+c), inst:'5ca0006f jal     x0,5d6' }
Info (IDV_RPT)           ! dut.pc           - 0x0000001c vector_table+1c
Info (IDV_RPT)             ref.pc           - 0x0000000c vector_table+c
Info (IDV_RPT)           ! dut.mcause       - 0x80000007 Interrupt:1 Code:7(Machine timer interrupt)
Info (IDV_RPT)             ref.mcause       - 0x80000003 Interrupt:1 Code:3(Machine software interrupt)
Info (IDV_RPT)
Info (IDV_RPT) SCENARIO 3:
Info (IDV_RPT)  +  Net MTimerInterrupt => 0 { when:37351 }
Info (IDV_RPT)     +  Exception { pc:0x1304(test2+4a), fetch:'904ff0ef jal     x1,408', 'Machine software
...
```

RISC-V System's Memory Consistency

- RISC-V systems can have many cores, harts, caches
- RISC-V ISA defines two memory models: TSO (Total Store Order) and RVWMO (Weak Memory Order)

TSO is strong memory model – and software can assume memory is consistent

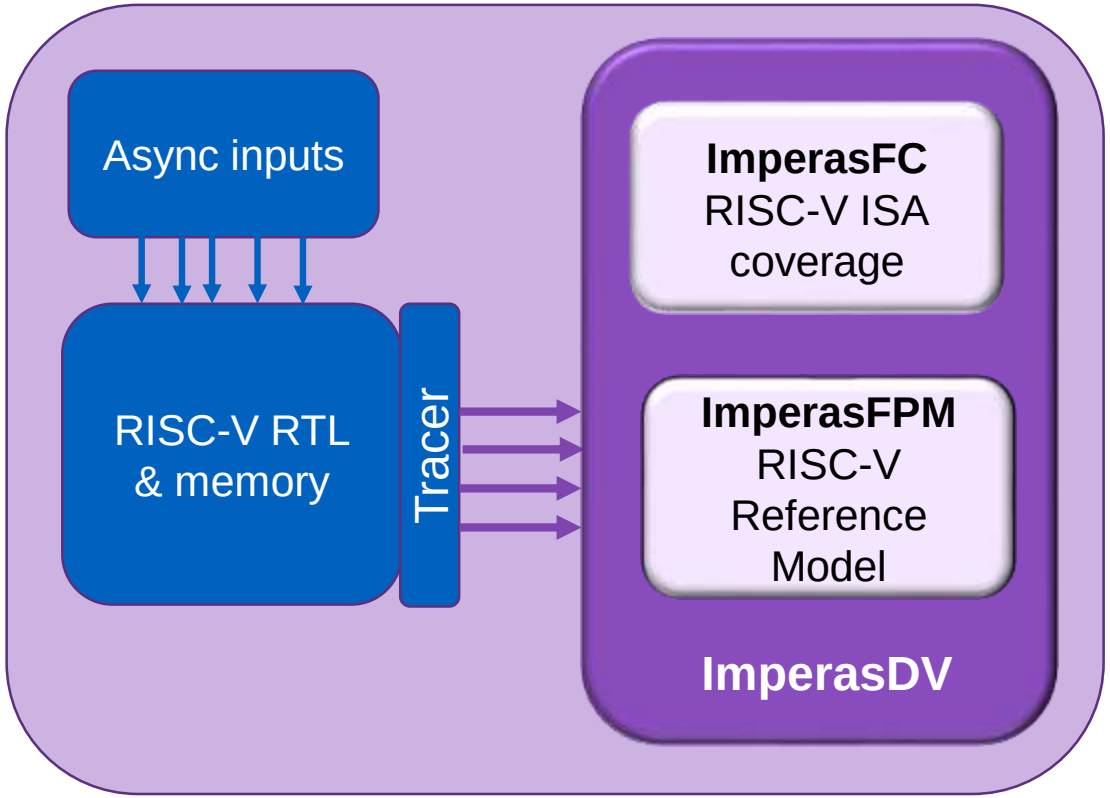
RVWMO

- is weaker model and allows hardware to be designed for simplicity and / or performance
- introduces several modeling and verification challenges compared to stronger memory models like TSO
  - Reordering of memory operations
  - Delayed visibility of writes
  - Complex interactions between multiple cores / harts
  - Explicit synchronization (fences) required to maintain correctness
  - Difficult-to-predict interleaving of memory operations
- allows reordering across the following operations
  - Load → Load
  - Load → Store
  - Store → Load
  - Store → Store
  - Fence enforcement

Verifying cores with multiple harts and caches with ImperasDV

TSO modeled in ImperasFPM reference model with option for RVWMO checking

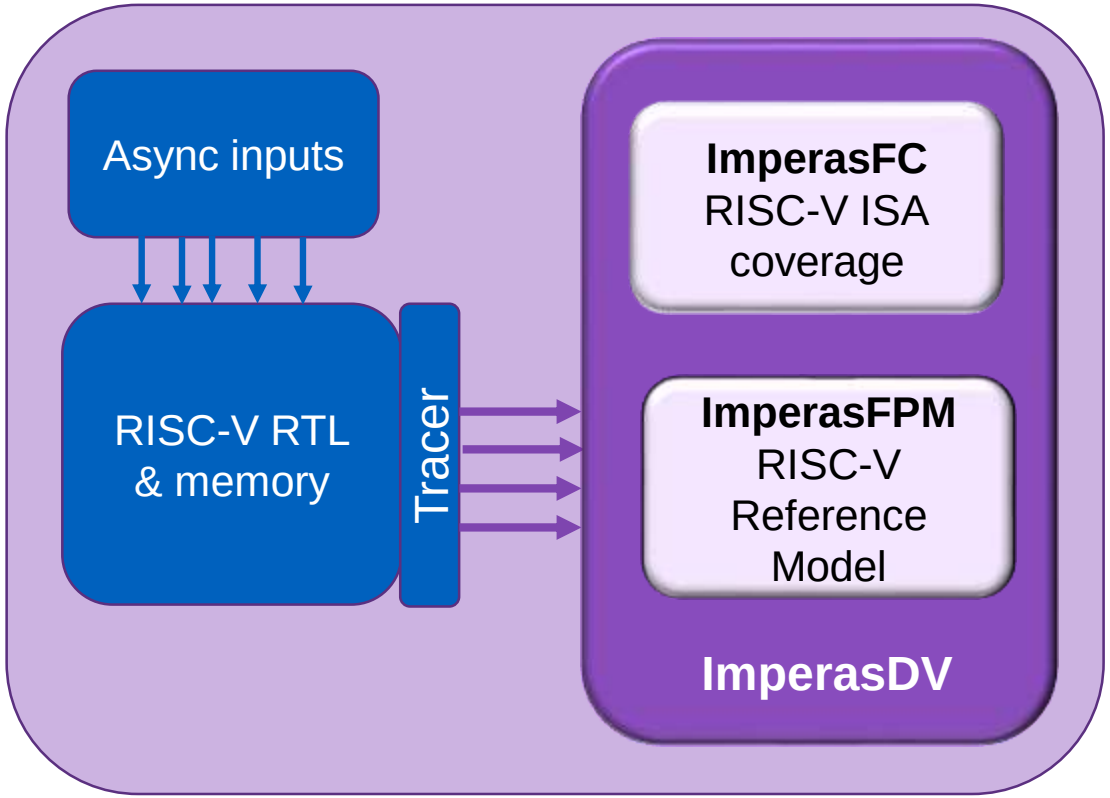
- RVWMO & caches means that memory operations can have many different orderings – all legal
- Tracer notifies ImperasDV when DUT memory is operated on
  - e.g. RTL DUT LD / ST instructions or instruction fetch
  - Indicating which hart and values and when
- For a verification run ImperasDV / ImperasFPM must be configured to select its memory consistency model



Verifying cores with multiple harts and caches with ImperasDV

TSO modeled in ImperasFPM reference model with option for RVWMO checking

- An architectural model of an RVWMO multi-hart device or device with cache(s) cannot predict exactly when a memory operation and its value will take effect
- ImperasDV includes Memory Operation Synchronization technology which determines whether DUT memory operations and values are legal and correct for RVWMO or TSO



Configure ImperasDV / ImperasFPM to use TSO or RVWMO model

Verifying cores with multiple harts and caches with ImperasDV

TSO modeled in ImperasFPM reference model with option for RVWMO checking

- An architectural model of an RVWMO multi-hart device or device with cache(s) cannot predict exactly when a memory operation and its value will take effect
- ImperasDV includes Memory Operation Synchronization technology which determines whether DUT memory operations and values are legal and correct for RVWMO or TSO
- If not, an error is flagged and debug information is provided
- Areas of memory can be excluded from the checking by tagging them ‘volatile’

Example messages

```
Warning (IDV_MSR) Memory difference detected, Memory Synchronizer running...
Info    (IDV_MSRLI) Reverting last instruction
...
Info (IDV_MSF) Memory Synchronizer finished after 6 iterations
Info (IDV_MSMDR)Memory difference resolved, continuing
...
Error (IDV_MSUR) Memory Synchronizer unable to resolve DUT memory load result
Error (IDV_MSNAR) Memory Synchronizer was not able to resolve the mismatch
Info  (IDV_MSRRS) Restoring reference model state
...
```

```
Warning (IDV_MSFID) Fetched instruction bit pattern difference detected,
      Memory Synchronizer running...
...
Info (IDV_MSFIA) Fetched instruction bit pattern accepted, continuing
Info (IDV_MSFP) Fetch of 0001 from 0x000100b2 propagated to reference model
Info (IDV_MSFDR) Fetched instruction difference resolved, continuing
...
Error (IDV_MSNARF) Memory Synchronizer was not able to resolve fetch mismatch
of 0001 from 0x000100b2
...
```

Agenda

- Dynamic verification
  - ImperasDV, ImperasFPM (reference model)
  - Lock-step continuous compare, async events, weak memory models
  - **ImperasFC (functional coverage), ImperasSC (stimulus coverage)**
  - ImperasTS (directed test suites for compliance / compatibility)

ImperasFC: SystemVerilog Functional Coverage for RISC-V

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

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

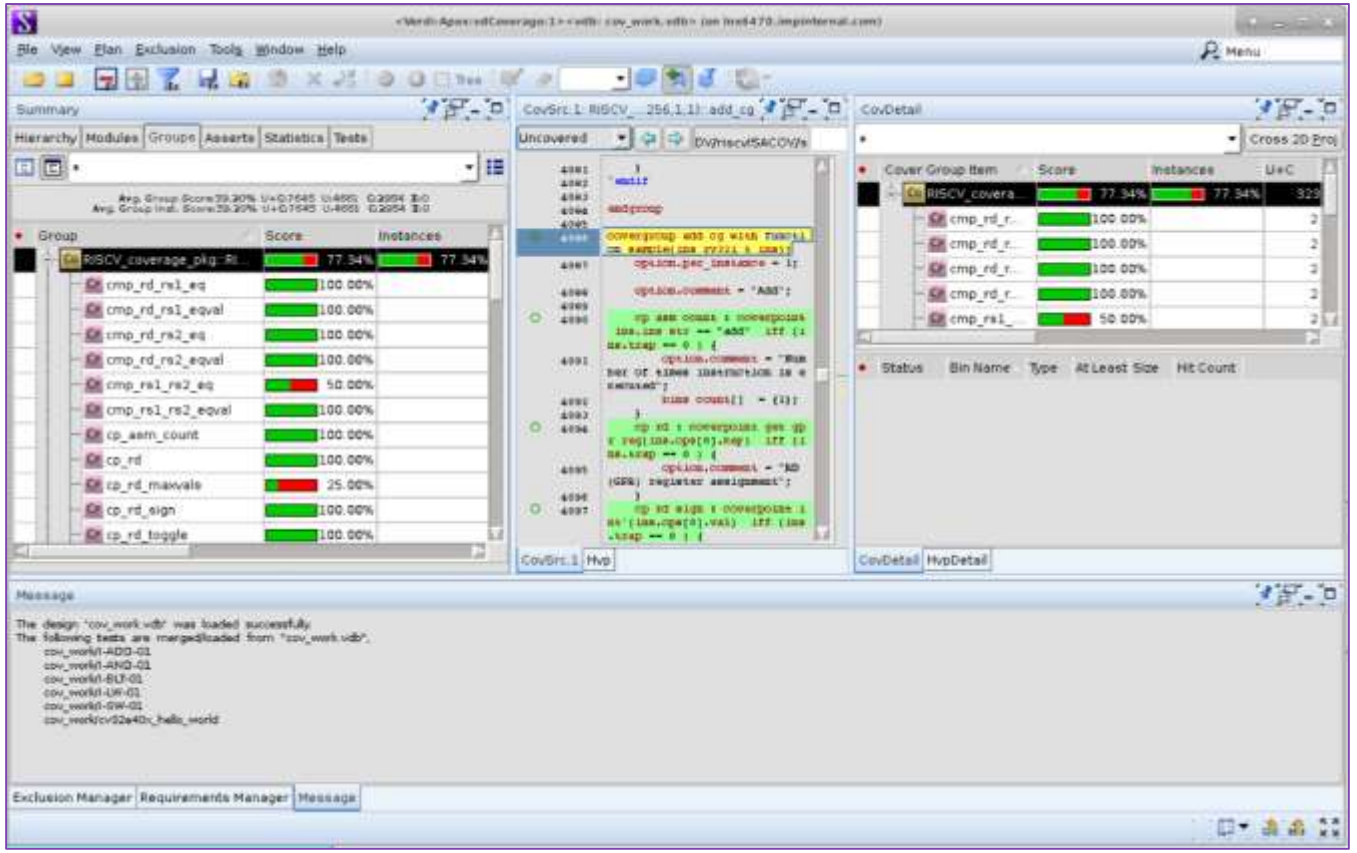
ImperasFC is integrated with ImperasDV / works with Verdi

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

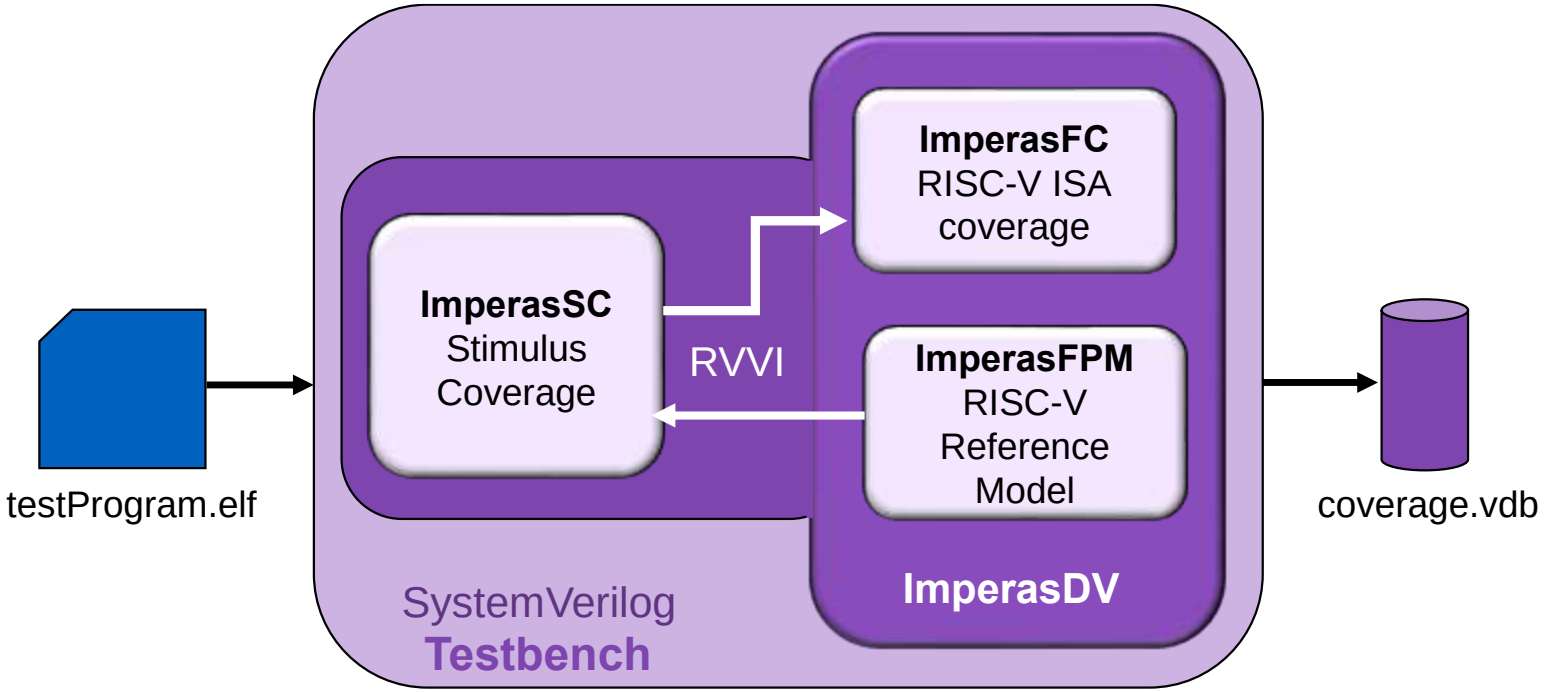
Extension	Subset	Instruction	Covergroup	Coverpoint	Coverpoint Description	Coverpoint Level
RV32I		addi	addi_cg	cp_asm_count	Number of times instruction is executed	Compliance Basic
				cp_rd	RD (GPR) register assignment	Compliance Basic
				cp_rd_sign	RD (GPR) sign of value	Compliance Basic
				cp_rs1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

Start test development early with ImperasSC

Stimulus Coverage measures the impact of test stimulus on functional coverage



- Uses ImperasFPM and ImperasFC
- Shift Left… no RTL required

=> start developing tests and measuring coverage in parallel with RTL development

How it works:

- Tests are run on the ImperasFPM
- RVVI-TRACE data is captured and used to sample functional coverage in ImperasFC
- Use Verdi to merge and analyze coverage results from multiple tests
- RISC-V processor RTL and SystemVerilog tracer are not required

Agenda

- Dynamic verification
  - ImperasDV, ImperasFPM (reference model)
  - Lock-step continuous compare, async events, weak memory models
  - ImperasFC (functional coverage), ImperasSC (stimulus coverage)
  - **ImperasTS (directed test suites for compliance / compatibility)**

ImperasTS (directed Test Suites for compliance / compatibility)

- Directed test suites for architectural validation ("compliance")
- Provided as .S source, Self-checking (includes automatically generated assertions)

Basic fixed ISA – standard RV32, RV64, B, K, … ratified instruction extensions

- 84 Test Suites

Complex, configurable extensions

- Test suites generated to match customer’s core configuration

- Vector (Zv, Zvk)
  - Includes 7 separate test suites
  - Vector crypto (Vk) included
- MMU
  - Supports Sv32, SV39, and SV48 virtual memory systems
  - Separate tests for User and Supervisor modes
- PMP / EPMP
  - Supports 32 bit and 64bit PMP, EPMP
  - Tests are generated to target specific pmpcfg / pmpaddr regions
  - Allows read-only fields and custom reset values in CSRs

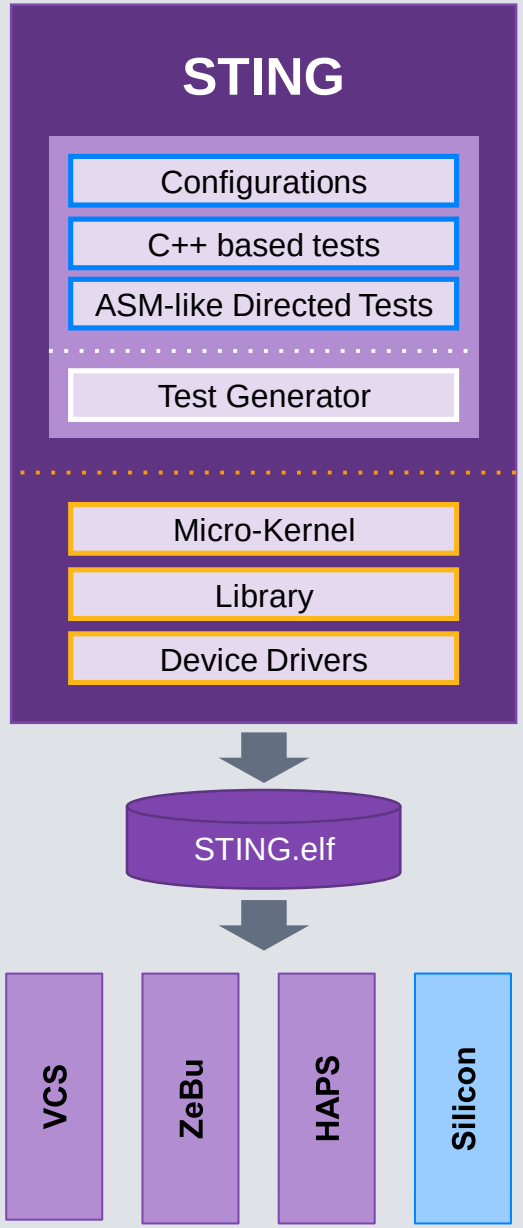
Agenda

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

STING – RISC-V Test Generation

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
- 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



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 users include processor IP vendors and SoC developers building their own RISC-V processor

STING - Bugs Found

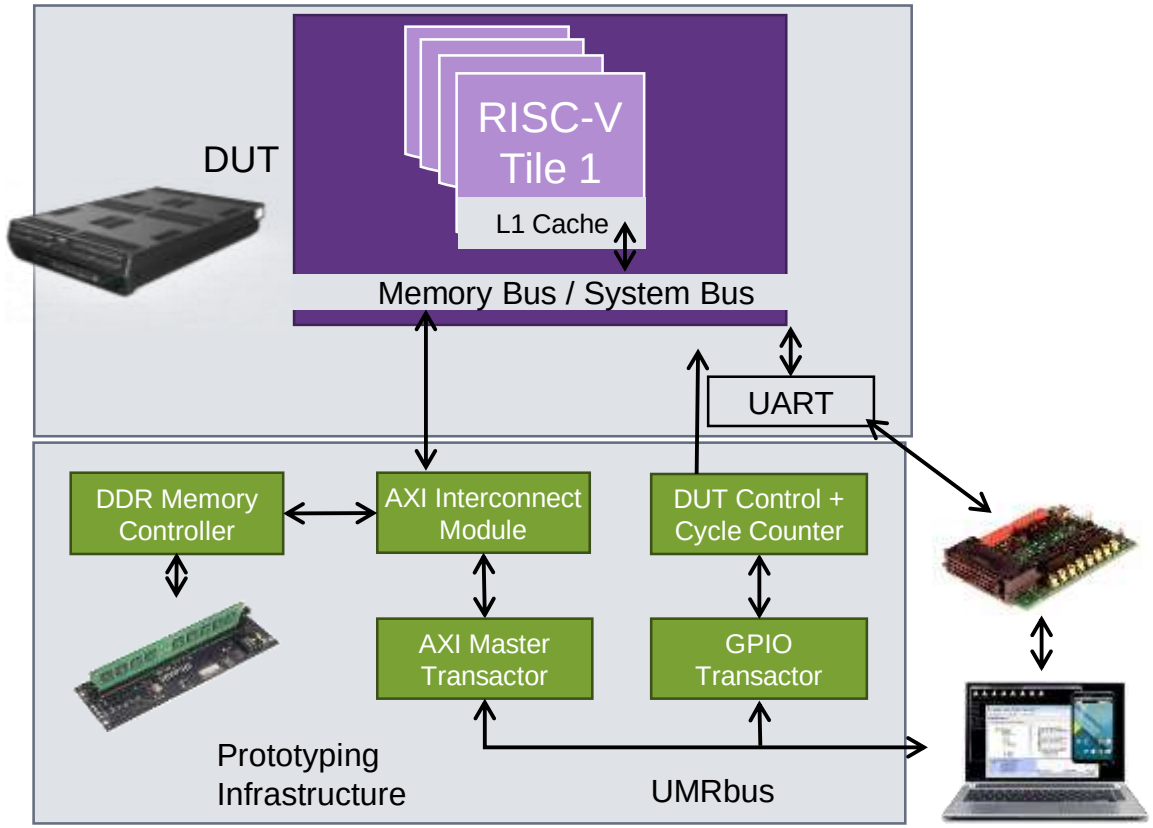
- ★ *"**Deadlock condition** existed when a TLB Miss for an older load/store instruction waits for its page-table-walk which cannot complete because newer stores have been issued and filled up certain miss-handling buffers in the load/store unit. This was uncovered by STING exercising streams of loads/stores 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**'*

Agenda

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

Hardware Accelerates Verification

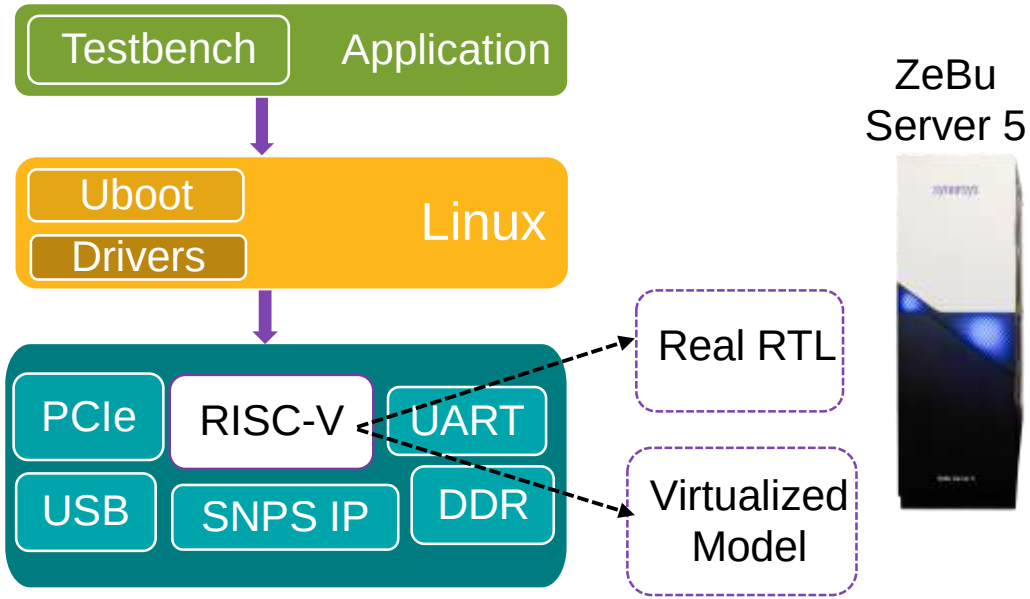
HAPS Prototyping Solution



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
- **Static / Formal verification**
- Summary

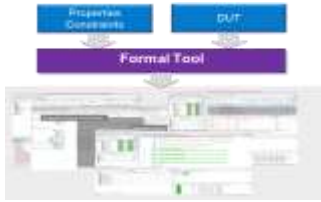
Formal Verification: VC Formal

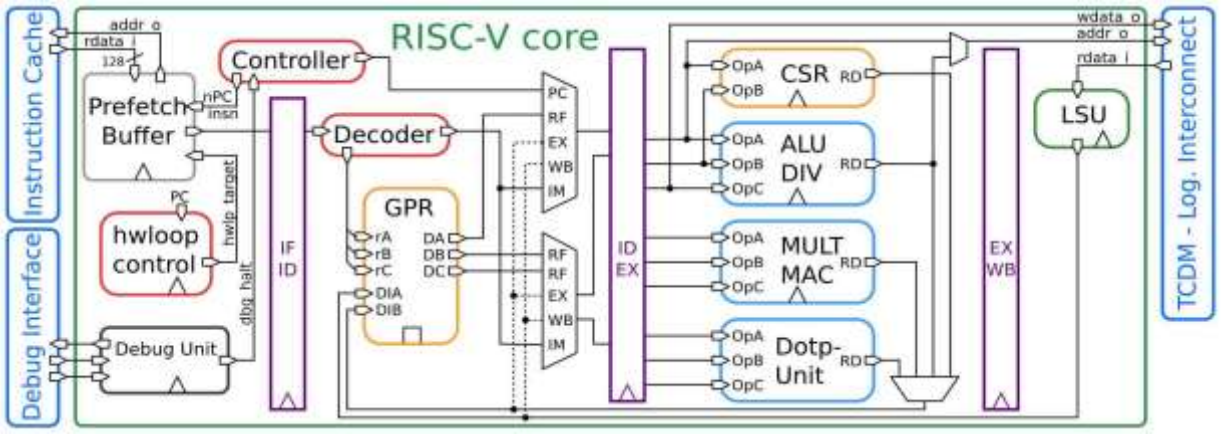
- Formal verification provides exhaustive proof of correct behavior
- Excellent tool for unit-level DV
  - Can get started early, even with design engineers
  - Unit-level includes pipeline, floating point unit, load/store unit, …
- RISC-V ISA Assertion IP (AIP) available to enable early use of VC Formal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

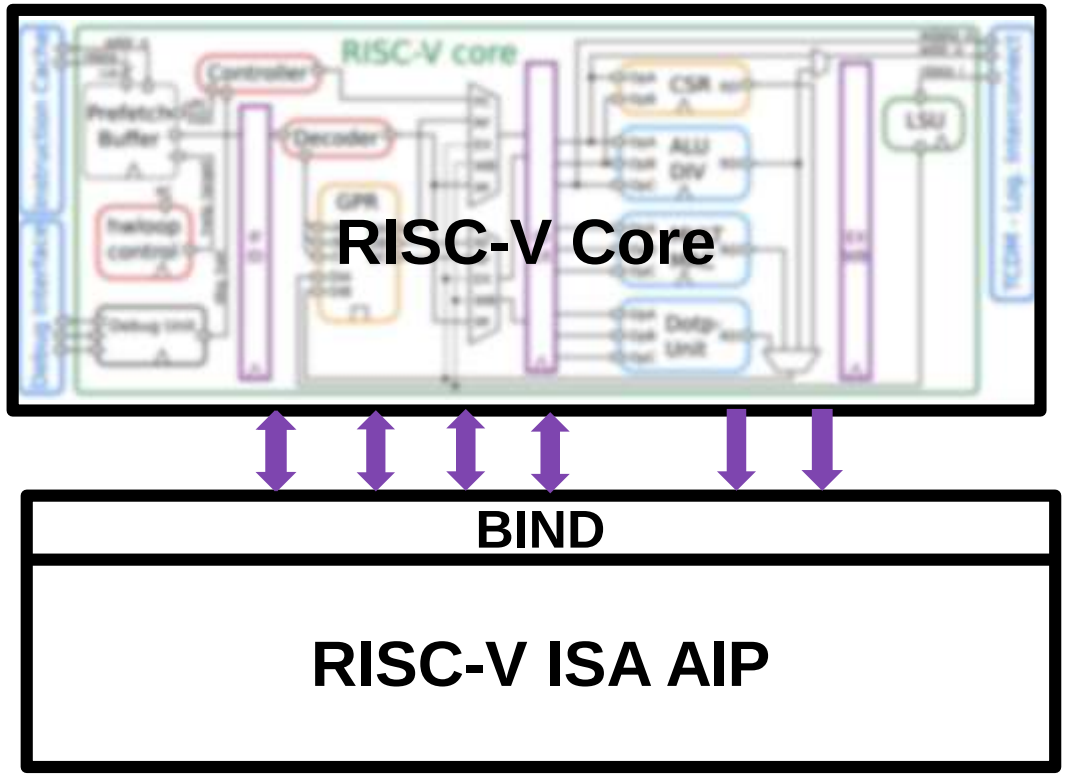
- Prefetch Buffer:
  - Redirect/Clear from various components: BPU/EX etc. should cause proper action and in a priority order
  - Instruction Cache
  - Direction/Target Prediction
  - Branch Target Buffer
  - Wake: Detecting a ready instruction
  - Dispatch: Need to select (oldest woken-up instruction 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

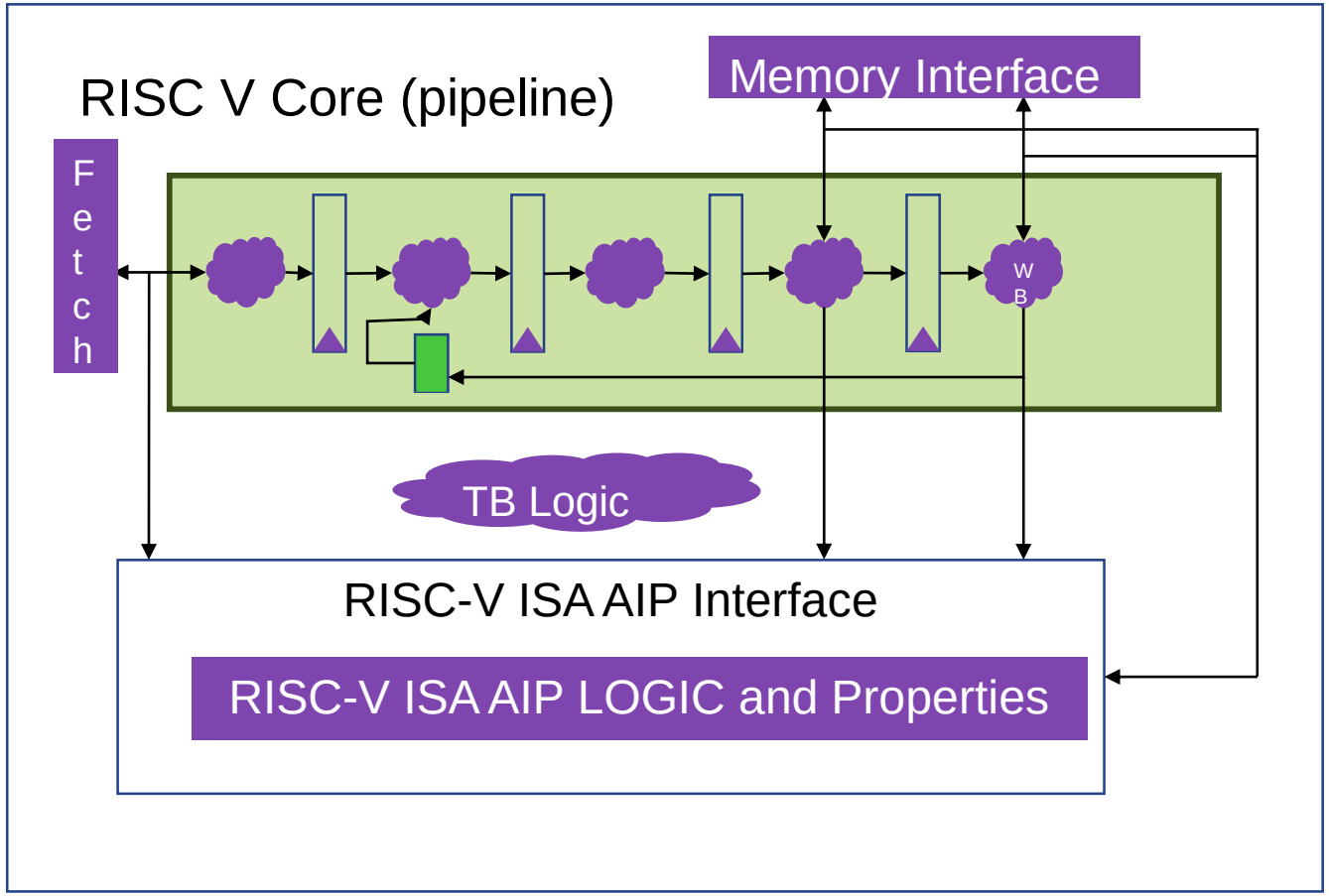
VC Formal RISC-V Assertion IP (AIP) for Exhaustive ISA Verification



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
- 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
  - …

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
- Static / Formal verification
- **Summary**

Summary

- RISC-V processors are coming … are here now!
- RISC-V processor verification is challenging
- The full toolbox is needed for successful verification of RISC-V processors

SYNOPSYS®

Thank You

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