2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Understanding the RISC-V Verification Ecosystem

Simon Davidmann, Imperas Software

Aimee Sutton, Imperas Software

Lee Moore, Imperas Software

imperas

accellera SYSTEMS INITIATIVE

Not talking about these familiar concepts…

- SystemVerilog simulators, UVM
- Formal
- CI technology
- Hardware assist
- FPGA prototyping
- VHDL
- Virtual platforms
- Verification services companies

=> All very important, but not covered in this talk….

Agenda

- Introduction to Imperas
- Introduction to RISC-V
- RISC-V processor verification challenges
  - Why is RISC-V processor DV so critical?
- RISC-V processor verification environment components
- RISC-V Verification approaches
- RISC-V Verification standards
- RISC-V Verification IP
- Functional coverage for RISC-V processors
- Verification Case studies
  - OpenHW Group CV32E40X processor
  - Wally RISC-V processor
- Summary

Agenda

- **Introduction to Imperas**
- Introduction to RISC-V
- RISC-V processor verification challenges
  - Why is RISC-V processor DV so critical?
- RISC-V processor verification environment components
- RISC-V Verification approaches
- RISC-V Verification standards
- RISC-V Verification IP
- Functional coverage for RISC-V processors
- Verification Case studies
  - OpenHW Group CV32E40X processor
  - Wally RISC-V processor
- Summary

Imperas

- 2008 – developed world class processor modeling & simulation solutions for many ISAs for virtual prototyping and software development
  - A good, growing, and profitable business
- 2016 started looking at RISC-V
- 2018 RISC-V processor developers started using Imperas RISC-V model as reference for their hardware verification
- For last 5 years have been assisting companies with their RISC-V DV needs
- For last 4 years started working collaboratively with free and open source solutions
  - e.g. OpenHW Group open source highly verified industrial quality RISC-V cores
- For last 3 years working on RISC-V verification standards and advanced methodologies
- 2022 Introduced first RISC-V processor DV solution that works out-of-the-box

riscvOVPsimPlus / riscvISATESTS – Commercial firms

Downloaders from OVPworld of riscvOVPsimPlus / riscvISATESTS (21-feb-2023)

riscvOVPsimPlus / riscvISATESTS – Academic, Research & Groups

University of Cape Town
भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati
De La Salle University
Bar-Ilan University
Université de Rennes 1
National Taiwan University
University of Engineering & Technology Lahore
UC Riverside
Ozarks Technical Community College
OpenCores
www.opencores.org
Welearn
Ruppin Academic Center
Stanford University
Carl von Ossietzky Universität Oldenburg
NUST School of Electrical Engineering & Computer Science
ASTRI
Hong Kong Applied Science and Technology Research Institute
香港應用科技研究院
PennState
Oklahoma State University
Universidade Federal de Campina Grande
Orta Doğu Teknik Üniversitesi
Middle East Technical University
Technische Universität München
Sharif University of Technology
STU
University of Illinois Urbana-Champaign
Carnegie Mellon University
Universität Stuttgart
Alexandria University
TAT 東京農工大学
Tokyo University of Agriculture and Technology
Technische Universität Dresden
서울대학교
Cornell University
Harvey Mudd College
TÜBİTAK
Universität Augsburg
Massachusetts Institute of Technology
De La Salle University Manila
Universidade do Minho
RWTH Aachen University
Lund University
OpenHW Group
University of Pittsburgh
NC State University
Barcelona Supercomputing Center
Centro Nacional de Supercomputación
UEH University of Economics Ho Chi Minh City
University of Southampton
UTM Universiti Teknologi Malaysia
ICT 中国科学院计算技术研究所
Institute of Computing Technology, Chinese Academy of Sciences
Vishnu (Autonomous) Institute of Technology
Universität Augsburg University
Hokkaido University
Texas A&M University
AUB American University of Beirut
UCI University of California, Irvine
UEH University
University of Cambridge
Viet Nam National University HCMC
University of Information Technology
UIT-HCM
Nirma University
NAAC Accredited 'A+' Grade
GUC German University in Cairo
الجامعة الألمانية بالقاهرة
서울대학교
Seoul National University
Yikexin Research Institute
中国科学技术大学
University of Science and Technology of China
National Institute of Technology Karnataka, Surathkal
Tampere University

Downloaders from OVPworld of riscvOVPsimPlus / riscvISATESTS (21-feb-2023)

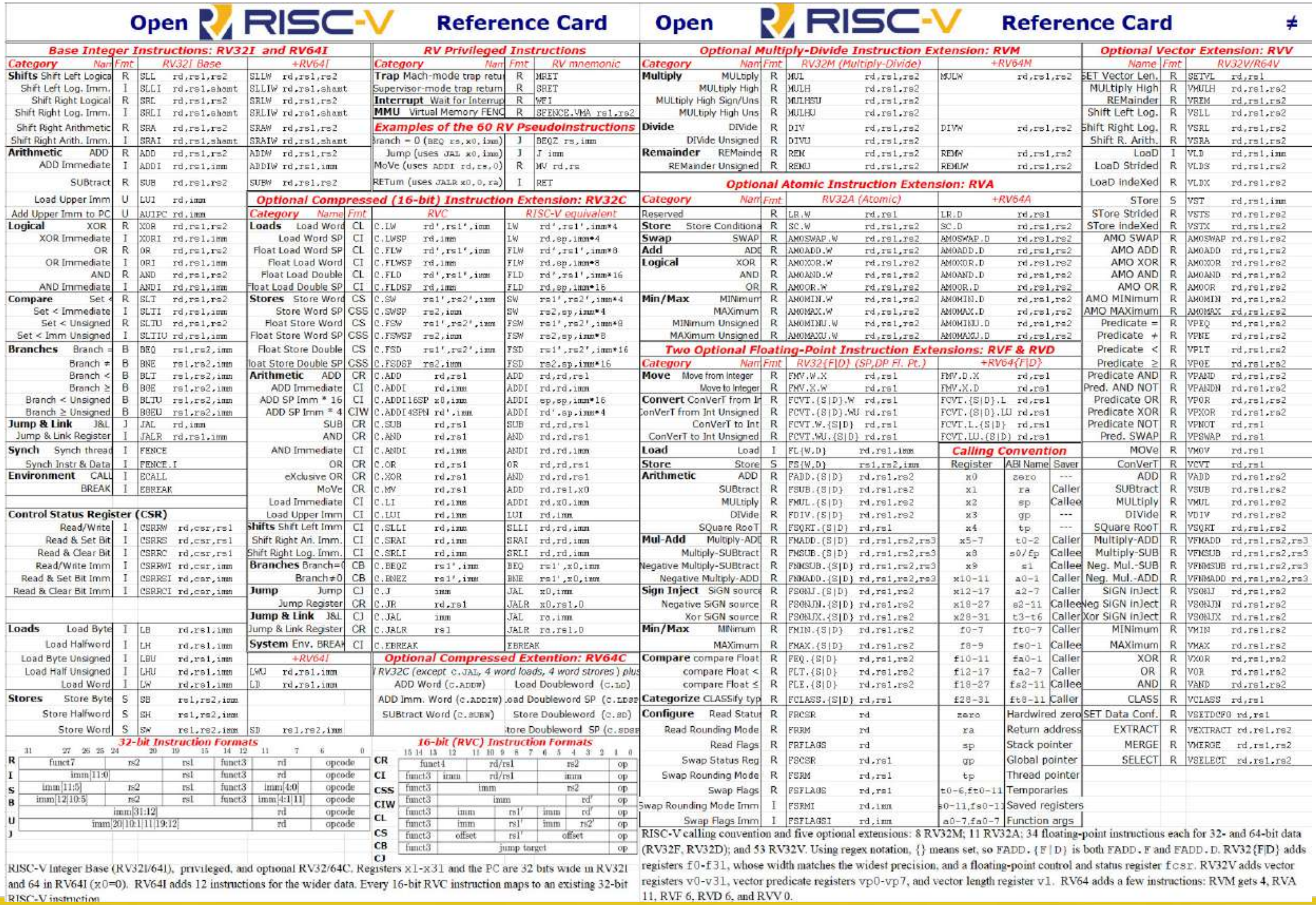
Agenda

- Introduction to Imperas
- **Introduction to RISC-V**
- RISC-V processor verification challenges
  - Why is RISC-V processor DV so critical?
- RISC-V processor verification environment components
- RISC-V Verification approaches
- RISC-V Verification standards
- RISC-V Verification IP
- Functional coverage for RISC-V processors
- Verification Case studies
  - OpenHW Group CV32E40X processor
  - Wally RISC-V processor
- Summary

About RISC-V

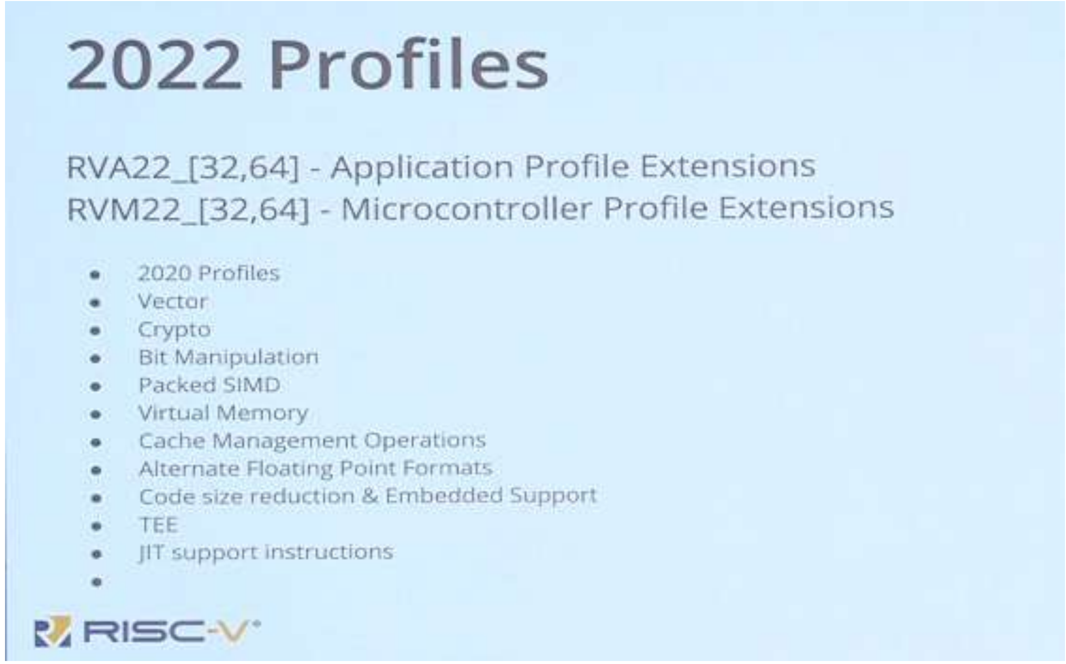
- Developed by researchers at Berkeley in 2010 under Prof. Patterson
- RISC-V is an open standard Instruction Set Architecture (ISA) enabling a new era of processor innovation through open collaboration
- RISC-V International (riscv.org) is the global non-profit home of the RISC-V ISA, related specifications, and stakeholder community
  - 3,000+ RISC-V members across 70+ countries contribute and collaborate to define RISC-V open specifications as well as convene and govern related technical, industry, domain, and special interest groups

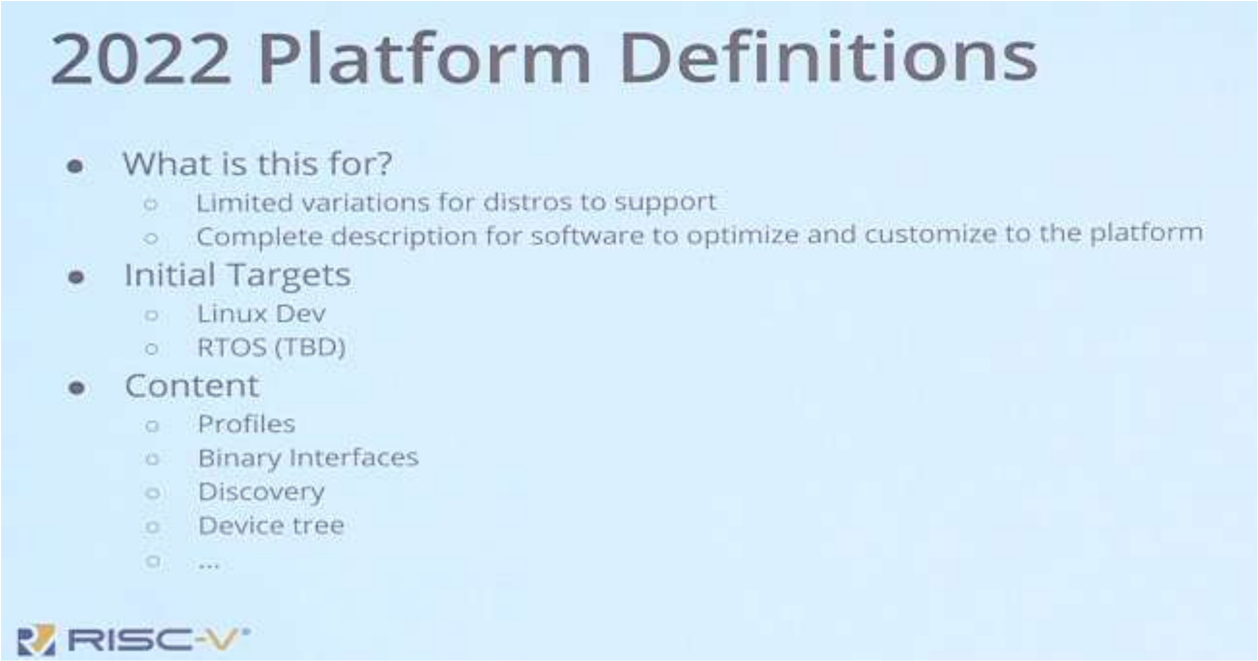
RISC-V Reference card (2018)



Initially 47 instructions,
now over 1,000,
in 70+ ISA extensions

RISC-V Profiles & Platforms





- Ways of grouping the many extensions….

RISC-V Evolving (2022)

Adaptable

Adaptable

Today

- Bases: RV32I, RV64I
- Extensions (70 to date): ACDFHMQV, priv 1.12, SV*, Zb, Zfinx, Crypto Scalar, etc.
- Non-ISA: psABI, SBI, UEFI, Etrace
- Organization: 9 committee, 28 Special Interest Groups (strategy, gap analysis & prioritization), 26 Task Groups (creating specifications)
- Member defined custom extensions (X). for example XVentanaCondOps or V0.7
- No baggage
- Efficient: modular, modern ISA

Tomorrow

- Bases: RV32E, RV64E
- Profiles: RVI20, RVA20, RVA22, RVA23
- Extensions (~30): Crypto Vector, Zc, subsets, etc.
- Non-ISA: ACPI, AP-TEE, IOMMU, IOPMP, Nexus, PLIC, SEE, Security Model, Unified Discovery, Watchdog Timer, CMQRI

~

- Bases: RV128I
- Profiles: RVA24, RVM
- Platforms: OS-A, OS-M
- Extensions: P, Matrix
- Beyond: CHERI, GPU

RISC-V®

Agenda

- Introduction to Imperas
- Introduction to RISC-V
- **RISC-V processor verification challenges**
  - **Why is RISC-V processor DV so critical?**
- RISC-V processor verification environment components
- RISC-V Verification approaches
- RISC-V Verification standards
- RISC-V Verification IP
- Functional coverage for RISC-V processors
- Verification Case studies
  - OpenHW Group CV32E40X processor
  - Wally RISC-V processor
- Summary

The RISC-V Disconnect

RISC-V Core User

Expects core quality to be the same as Arm

RISC-V

RISC-V Core Developer

Unlikely to have resources needed be able to develop all the technologies required to perform the same level of verification as Arm

Putting Processor Verification into Context….

1,000,000,000,000,000

The number of verification cycles Arm uses when verifying an Arm core

- SystemVerilog simulator executes 2,000 cycles / second

  => 15,000 SystemVerilog simulators running for 1 year

- HW emulator or FPGA runs at 1,000,000 cycles / second

  $\Rightarrow$30 years of running needed…

- OK – so this is for high end performance OoO, MP, VM cores (full apps processors)
  - Embedded processors will be an order of magnitude less…

RISC-V Design Verification Challenges

- Processor verification has been a niche discipline
  - Proprietary techniques
- No industry-standard best practices or verification IP
  - Until recently… (stay tuned)
- Techniques from ASIC/SoC verification are insufficient
- New methods are required
  - Take advantage of what has worked in the ASIC world
  - Add to it and enhance for RISC-V

So what is being done in the RISC-V world

- In the RISC-V world, it is unlikely that one company can spend the $ or can hire the people to develop all they need…
  - [Arm relies on ISA / design royalty, Intel relies on silicon sale…]

1) Partnering and Collaboration in non-competitive areas
2) Attracting players into the verification ecosystem to develop needed solutions
3) Building standards to facilitate re-use and efficiency

- If it does not differentiate your product offering / company
  - You can collaborate externally
  - You can license commercial tools

So what have we learnt in last 5 years…
There are many approaches for 'verification' of new processors

- Does a program run? – 'hello world' tests
- Is there simple correct computation? – 'self checking tests'

Simple tests

- Signature checking – 'post simulation signature dump compares'
- Trace log checking – 'post simulation trace file compare'

Compliance

- Basic step and compare co-simulation – 'instruction retire compare'
- Advanced, e.g. commercial solutions – 'async-lock-step-compare'

Verification

- [Note: this discussion is only about dynamic simulation verification – there are of course many excellent commercial formal verification solutions]

Agenda

- Introduction to Imperas
- Introduction to RISC-V
- RISC-V processor verification challenges
  - Why is RISC-V processor DV so critical?
- **RISC-V processor verification environment components**
- RISC-V Verification approaches
- RISC-V Verification standards
- RISC-V Verification IP
- Functional coverage for RISC-V processors
- Verification Case studies
  - OpenHW Group CV32E40X processor
  - Wally RISC-V processor
- Summary

RISC-V processor verification environment components

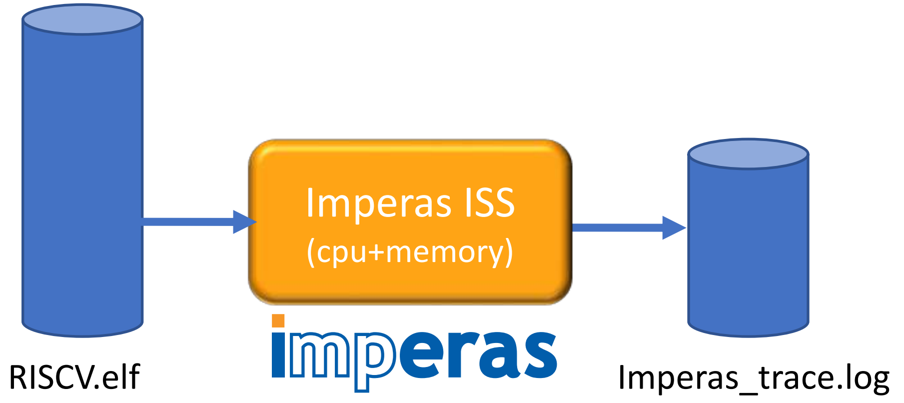
- Test Programs
- Instruction Set Simulators
- DUT + Tracer
- Processor reference model
- Verification IPs

Test programs

- Directed tests
  - Write your own
  - Compliance tests (RISC-V International)
  - Architectural Compatibility test suites (Imperas open source riscvISATESTS)
  - Configurable Commercial test suites (e.g. Imperas PMP and Vector)
  - Other open source, e.g. OpenHW directed test suites (synchronous & asynchronous)
- Instruction stream generators (ISG)
  - Configurable to match processor extensions
  - Open source solutions
    - e.g. riscv-dv (Google / CHIPS Alliance)
  - Commercial solutions
    - e.g. Valtrix STING

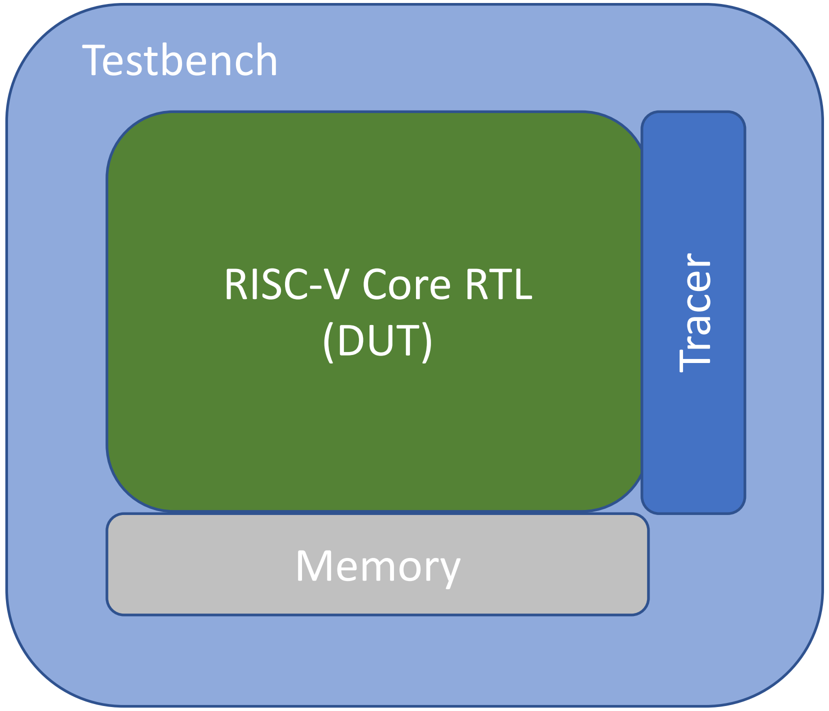
Instruction Set Simulators

- ISS
  - Simulate the execution of a program on a processor
  - Produce a trace file output
  - Open source solutions
  - Commercial/closed-source solutions
    - e.g. riscvOVPsimPlus



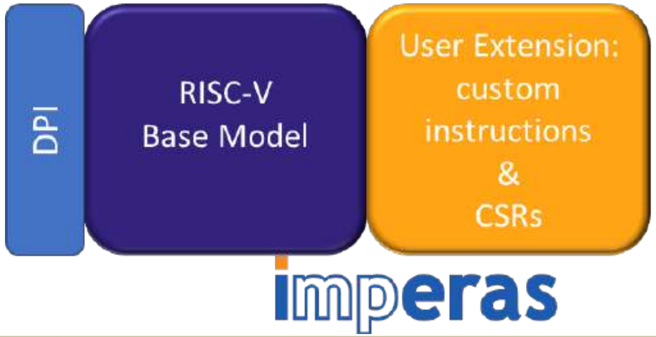
DUT + Tracer

- DUT (Design Under Test)
  - RTL for RISC-V processor
  - Memory model and bus i/f
  - Ability to load test program into memory
- Tracer
  - Extracts information needed for DV
    - e.g. PC, register values
  - Bespoke to particular microarchitecture
  - Often written by processor designers
  - Can use RVVI-TRACE standard

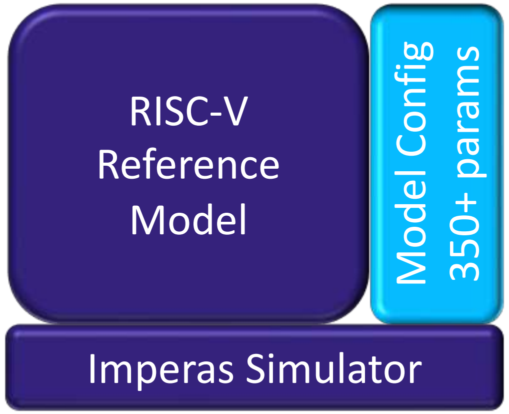


Processor Reference Model

- Reference model requirements:
  - Configurable to select RISC-V ISA extensions
  - Ability to extend / add customizations (e.g. instructions, CSRs)
  - Can run in co-simulation configuration
  - Can be controlled from other simulator
  - Ability to "step" reference model at significant events (retire, trap)
  - Can run in lock-step with the RTL simulator
  - Functions to query state of model for comparison



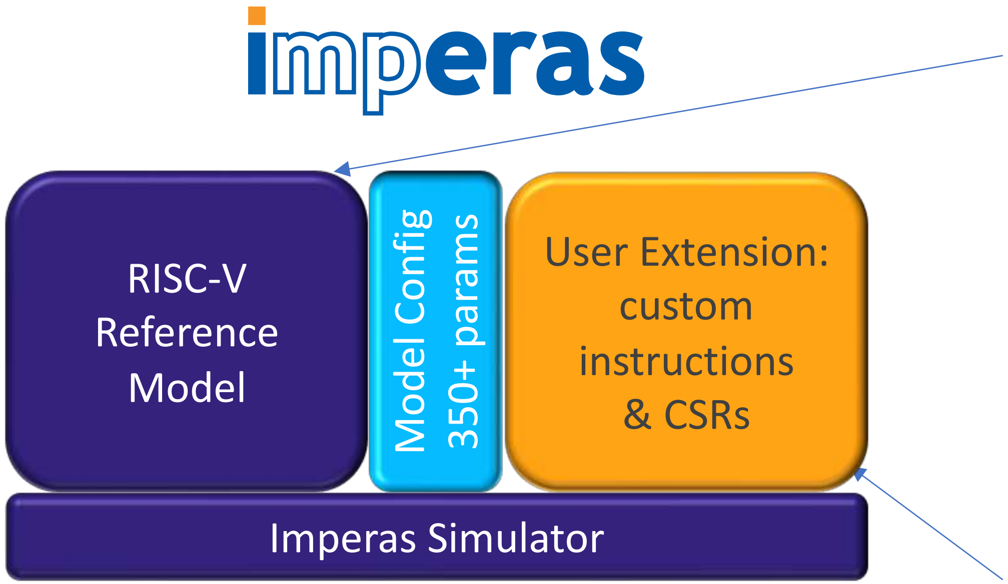
Imperas is used as RISC-V Golden Reference Model



http://www.imperas.com/riscv

- Imperas provides full RISC-V Specification envelope model
- Industrial quality model /simulator of RISC-V processors for use in compliance, verification and test development
- Complete, fully functional, configurable model / simulator
  - All 32bit and 64bit features of ratified User and Privilege RISC-V specs
  - Vector extension, versions 0.7.1, 0.8, 0.9, 1.0
  - Bit Manipulation extension, versions 0.90, 0.92, 0.93, 0.94, 1.0.0
  - Hypervisor version 0.6.1, 1.0
  - Debug versions 0.13.2, 0.14, 1.0.0
  - K - Crypto Scalar version 0.7.1, 1.0.0
  - K - Crypto Vector version 0.3.0
  - P - DSP versions 0.5.2, 0.9.6
- Model source included under Apache 2.0 open source license

Imperas RISC-V reference model



- Separate source files and no duplication to ensure easy maintenance
- Imperas or user can develop the extension
- User extension source can be proprietary

- Imperas develops and maintains base model
  - Base model implements RISC-V specification in full
- Fully user configurable to select required ISA extensions
- Fully user configurable to select which version of each ISA extension
- Imperas provides methodology to easily extend base model
- Imperas model is architected for easy extension & maintenance

Verification IPs

- Requirements:
  - Instance in SystemVerilog test bench
  - Scoreboard
  - Functional Coverage
  - Logger
  - Signature writers
  - Virtual peripherals (for async event generation)
  - Comparators
  - Synchronizers
  - Fault injectors
  - …

Agenda

- Introduction to Imperas
- Introduction to RISC-V
- RISC-V processor verification challenges
  - Why is RISC-V processor DV so critical?
- RISC-V processor verification environment components
- **RISC-V Verification approaches**
- RISC-V Verification standards
- RISC-V Verification IP
- Functional coverage for RISC-V processors
- Verification Case studies
  - OpenHW Group CV32E40X processor
  - Wally RISC-V processor
- Summary

Compliance versus Verification

- Need to be clear what focus of testing is
  - Architecture
    - ISA Definition
  - Micro-Architecture
    - In-Order, Out-Of-Order, Simple-Scalar, Super-Scalar, Transactional Memory, Branch Predictors, …
- These are very different
  - One is about ISA specification
  - Other is about details of a specific implementation
  - This is the difference between “Compliance” and Design Verification

- In the RISC-V Foundation, “Compliance” testing is checking the device works within the envelope of the agreed specification
  - i.e. “have you read and understood the specification”
  - For RISC-V, compliance testing is a very small percentage of full hardware verification…

Many approaches for ‘verification’ (recap)

- Does a program run? – ‘hello world’ tests
- Is there simple correct computation? – ‘self checking tests’

Simple tests

- Signature checking – ‘post simulation signature dump compares’
- Trace log checking – ‘post simulation trace file compare’

Compliance

- Basic step and compare co-simulation – ‘instruction retire compare’
- Advanced, e.g. commercial solutions – ‘async-lock-step-compare’

Verification

- [Note: this discussion is only about dynamic simulation verification – there are of course many excellent commercial formal verification solutions]

RISC-V processor ‘verification’ approaches

- Simple:
  - run program ‘hello world’ tests
  - self checking tests
- Compliance:
  - post simulation signature dump file compare
  - post simulation trace log file compare
- Verification:
  - Basic ‘instruction retire step compare’ co-simulation
  - Quality ‘async lock step compare’ co-simulation

Simple Level Self-Checking Tests

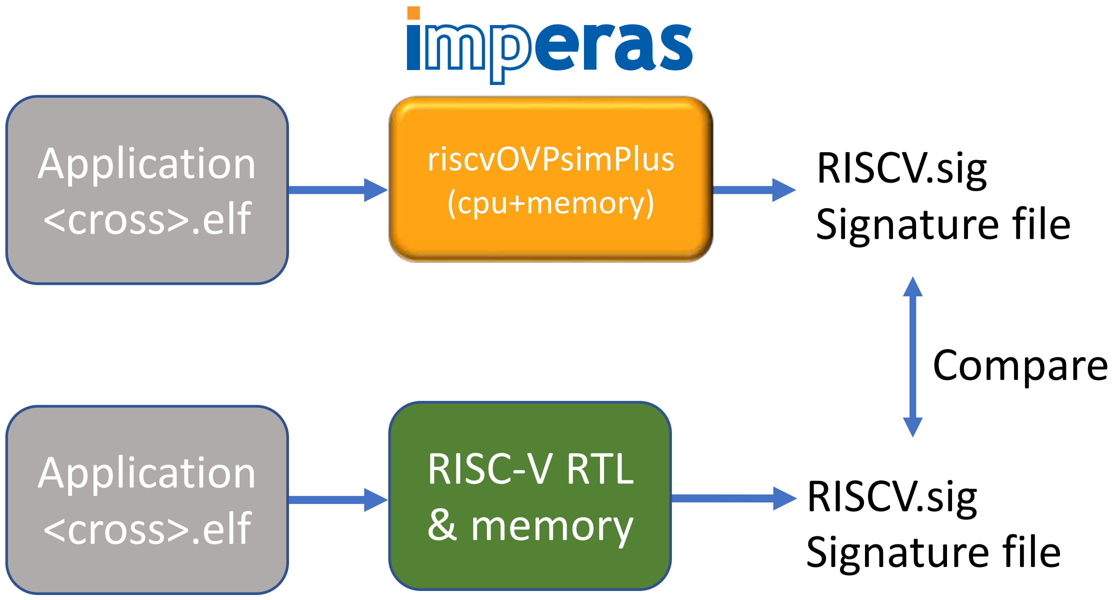
- Components:
  - RISC-V processor (DUT) and test program; optionally ISS
- Process:
  - Each test program checks its results
    - Prints message to log
    - Or writes bit to memory
      - for later reading

Simple Level Self-Checking Tests : Pros and Cons

- Pros:
  - Simple to set up and execute
    - Free ISS: https://github.com/riscv-ovpsim
    - Free compiler: https://github.com/Imperas/riscv-toolchains
  - RISC-V tests freely available, e.g. Berkeley tests
    - https://github.com/riscv-software-src/riscv-tests
- Cons:
  - Simple tests cover a small subset of processor functionality
  - Not a complete DV strategy

Compliance Level
Post-Simulation Signature File Comparison

- Components:
  - RISC-V processor (DUT) and test program
  - ISS + reference model
- Process:
  - Run the test program on the DUT and save the output (signature file)
  - Run ISS + reference model, write signature file
  - Compare / diff file results
  - This is the approach taken by RISC-V International for their architectural validation ("compliance tests")



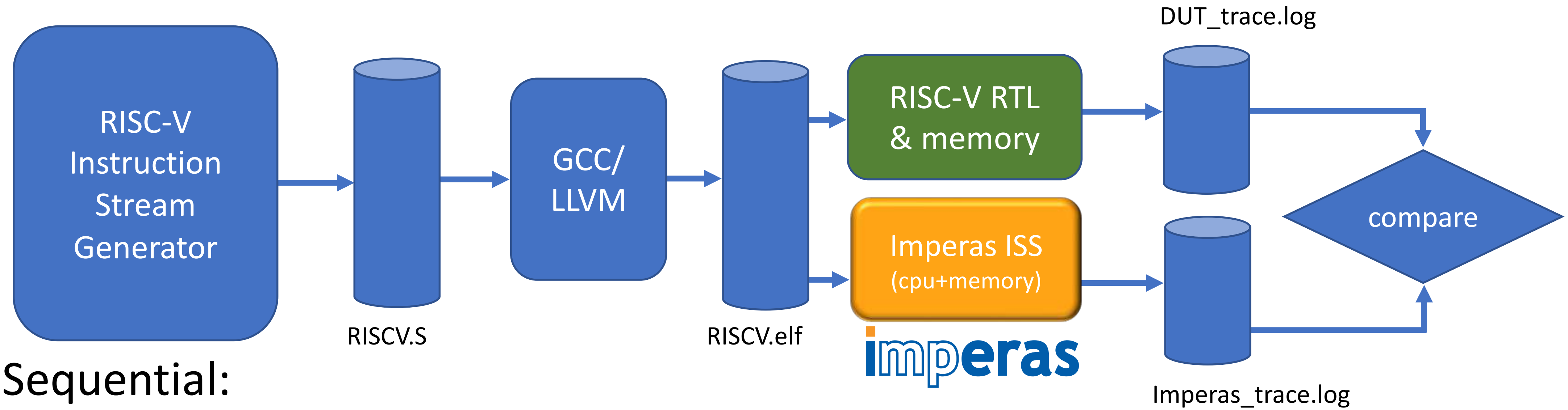
Compliance Level
Post-Sim Signature file compare : Pros and Cons

- Pros:
  - Simple to set up and execute
    - Free ISS: https://github.com/riscv-ovpsim
    - Free compiler: https://github.com/Imperas/riscv-toolchains
  - RISC-V tests & compliance level tests freely available
- Cons:
  - Directed tests cover a subset of processor functionality
  - Easy to have incomplete or wrong info in signatures (misses behaviors)
  - Not a complete DV strategy

Compliance level
Post-Simulation Trace Log File Compare

- Components
  - Test programs
    - Can be generated by an ISG – Instruction Stream Generator
  - Instruction Set Simulator (ISS) + reference model
  - DUT and Tracer
  - RTL simulator
  - Comparison script

Compliance Level
Post-Simulation Trace Log File Compare: Process



Sequential:

1) Run random generator (ISG) to create tests

2) Simulate using ISS; write trace log file

2) Simulate using RTL; write trace log file

3) Run compare program to see differences / failures

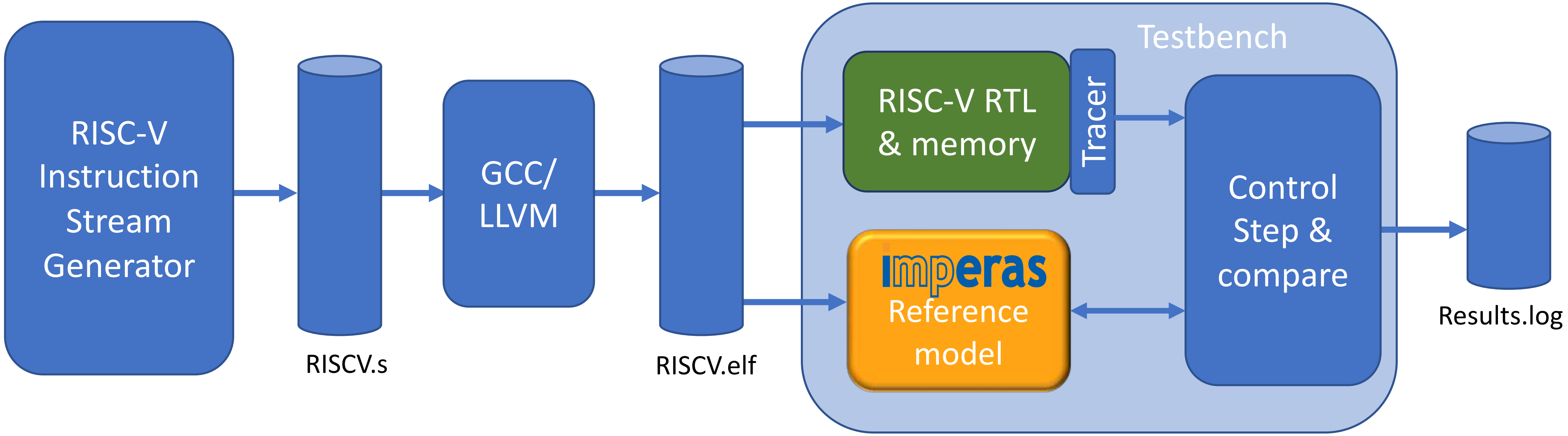
Compliance Level
Post Sim Trace Log File Compare : Pros and Cons

- Pros:
  - Availability of quality RISC-V simulators (e.g. riscvOVPsimPlus from Imperas)
  - Simple to set up and use
- Cons:
  - Must run RTL simulation to the end
  - Cannot debug live
  - Difficult to verify asynchronous events (e.g. interrupts, debug requests)
  - Incompatible trace formats (between RTL, ISS, …)
  - Easy to skip instructions, and only compare selected few
  - Not a comprehensive DV strategy

Verification Level
Sync. Step-And-Compare co-simulation

- Components
  - Test programs (can be compliance, directed, or generated by an ISG)
  - Processor reference model
  - DUT and tracer
  - Step-and-compare logic
  - Comprehensive test bench
  - RTL simulator

Verification Level
Sync. Step-And-Compare co-simulation : Process



- Reference model is encapsulated in a SystemVerilog testbench
- Control block steps both DUT and reference model
- Extracts data from each; compares results on-the-fly
- Differences reported immediately

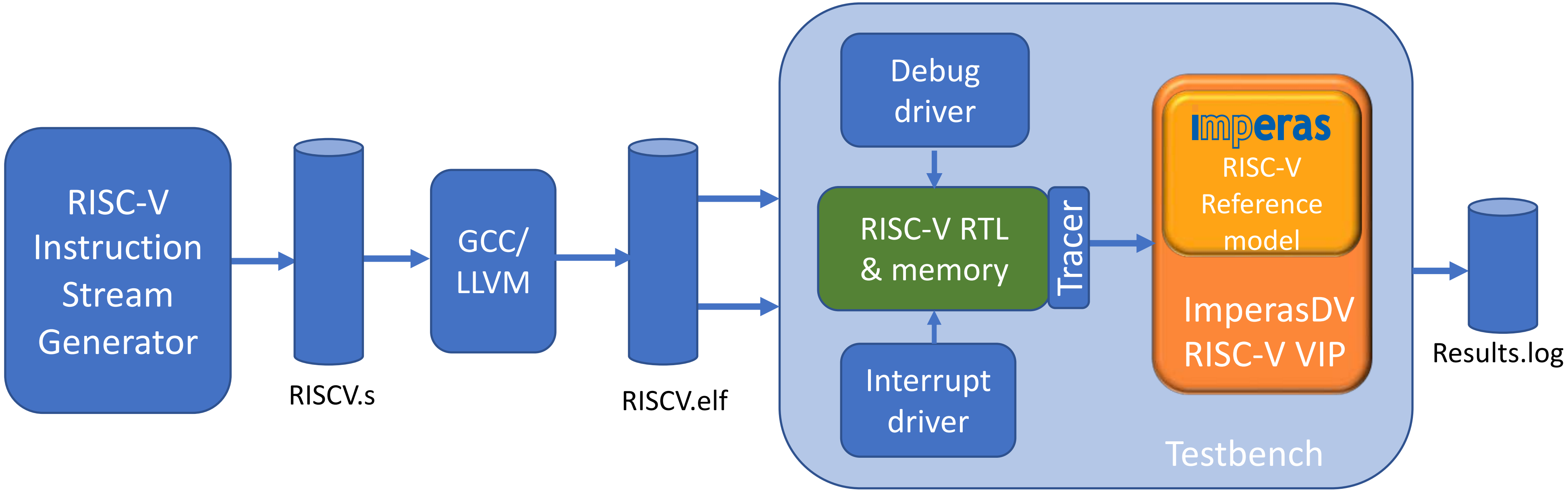
Verification Level
Sync. Step-And-Compare co-sim : Pros and Cons

- Pros:
  - Instruction by instruction lock-step comparison
  - Comparison of execution flow, program data, internal state
  - Errors are flagged immediately – no runaway simulations
  - Detects synchronous bugs
- Cons:
  - Step-and-compare logic can be fragile and error prone
  - Does not easily verify asynchronous events

Verification Level
Async. Step-And-Compare co-simulation

- Components
  - Test programs (can be generated by an ISG)
  - Processor reference model
  - DUT and tracer
  - Asynchronous event drivers (e.g. UVM agents)
  - RISC-V VIP
  - Comprehensive test bench
  - RTL simulator

Verification Level
Async. Step-And-Compare co-simulation: Process



- Asynchronous events are driven into the DUT
- Tracer informs reference model about async events
- Verification IP handles scoreboarding, comparison, coverage, pass/fail

Verification Level
Async. Step-And-Compare co-sim : Pros and Cons

- Pros:
  - All the benefits of sync. step-and-compare
  - Responds to asynchronous events
  - Checking is done for you
  - VIP is reusable across different core DV projects
  - Ease of use
  - Training, documentation, and support

- Cons:
  - Cost of VIP licenses

Verification Levels: Summary

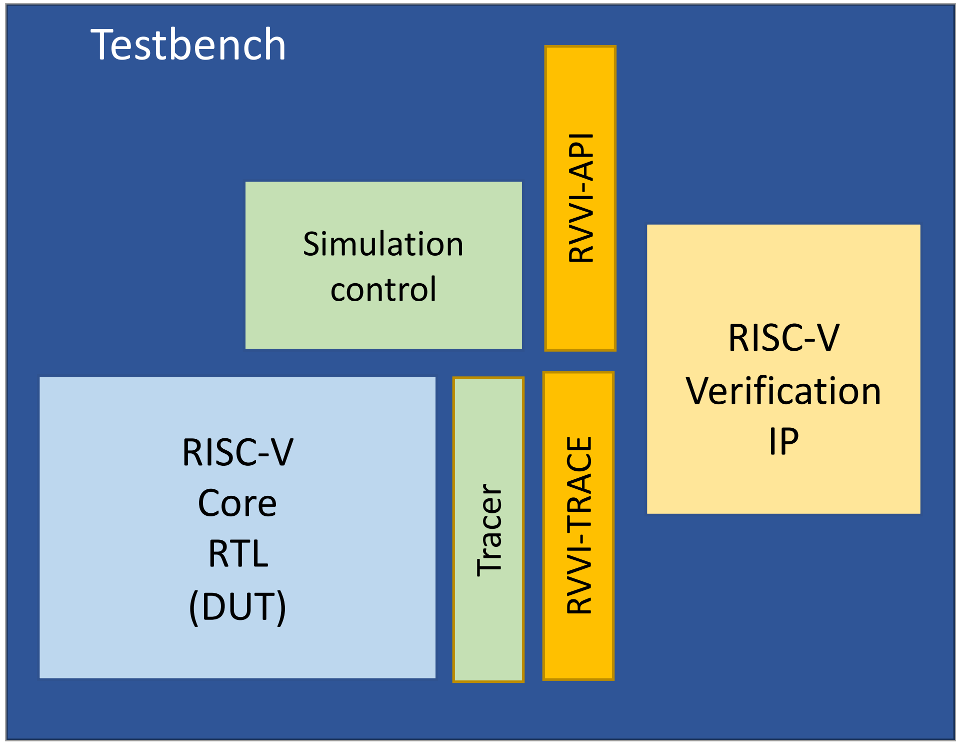
	Check basic functionality (E.g. compliance)	Supports constrained-random stimulus	Simulation ends after specified # of errors	Debug at the point of error	Verifies asynchronous events	Achieves verification closure
Self-checking tests	✓					
Signature file compare	✓					
Post-sim trace file compare	✓	✓				
Synchronous step and compare	✓	✓	✓	✓		
Asynchronous step and compare	✓	✓	✓	✓	✓	✓

Agenda

- Introduction to Imperas
- Introduction to RISC-V
- RISC-V processor verification challenges
  - Why is RISC-V processor DV so critical?
- RISC-V processor verification environment components
- RISC-V Verification approaches
- **RISC-V Verification standards**
- RISC-V Verification IP
- Functional coverage for RISC-V processors
- Verification Case studies
  - OpenHW Group CV32E40X processor
  - Wally RISC-V processor
- Summary

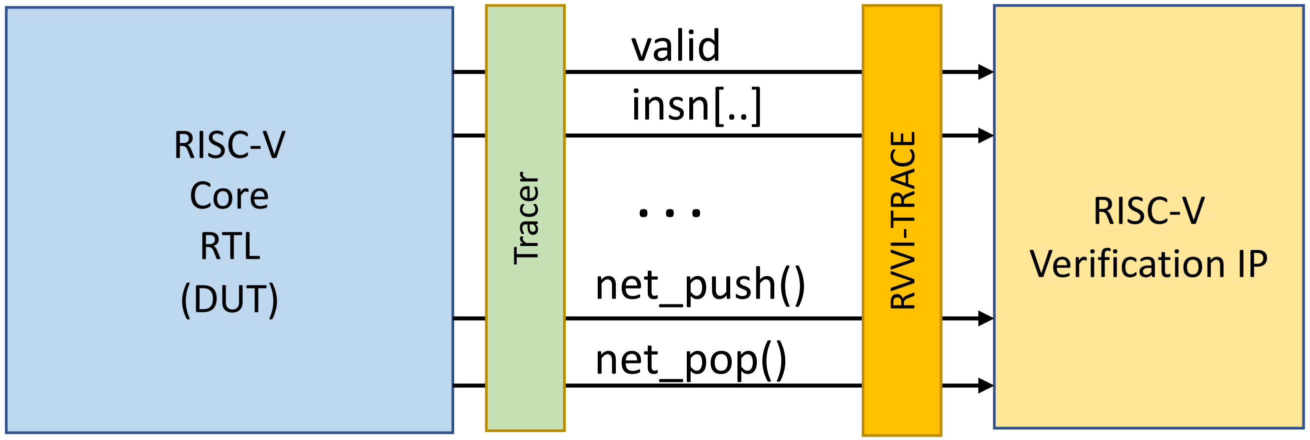
Open Standards
RISC-V Verification Interface: RVVI

- RVVI = RISC-V Verification Interface
  - https://github.com/riscv-verification/RVVI
- Work has evolved over 3 years
  - Imperas, EM Micro, SiLabs, OpenHW
- Standardize communication between DUT, testbench, and RISC-V VIP
- Two parts (currently):
  - **RVVI-TRACE**: signal level interface to RISC-V VIP
  - **RVVI-API**: function level interface to RISC-V VIP



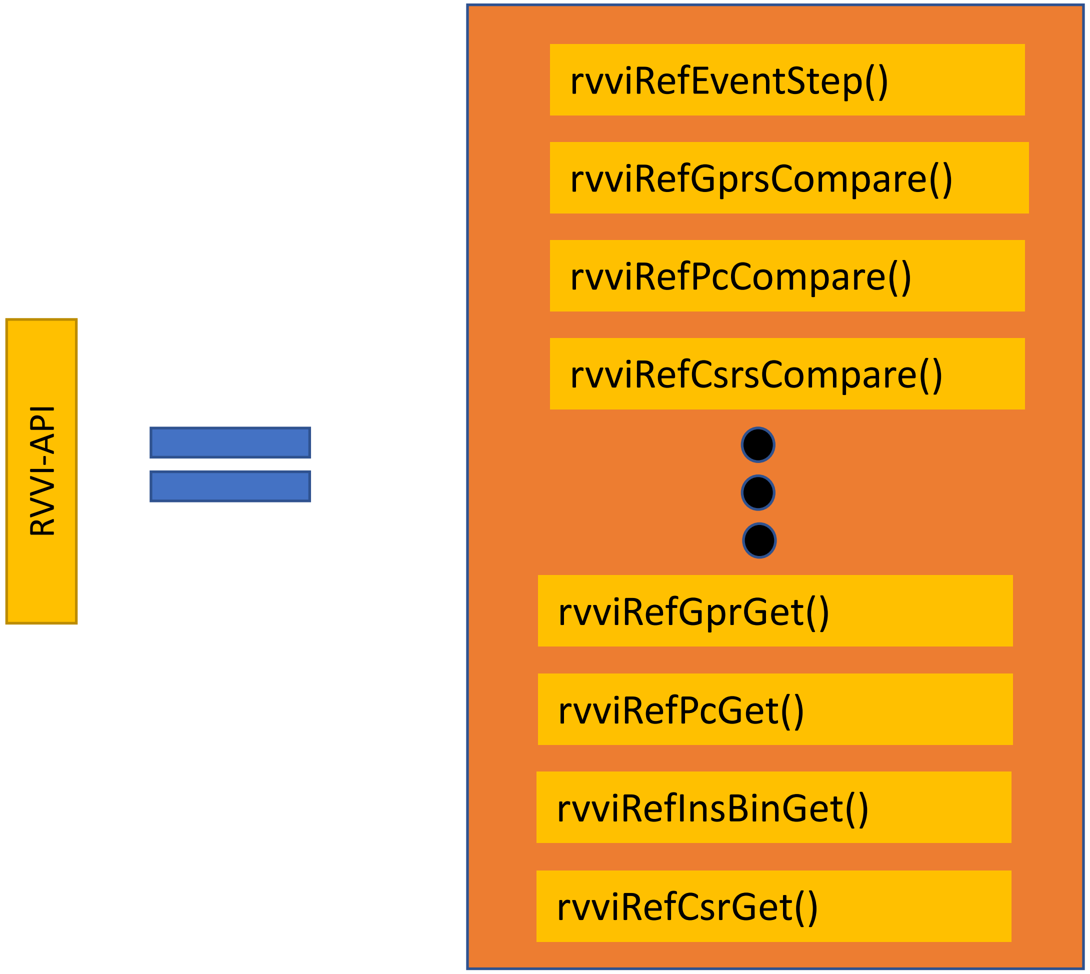
Open Standard: RVVI-TRACE

- Defines information to be extracted by tracer
- SystemVerilog interface
- Includes functions to handle asynchronous events
  - e.g. interrupts, debug requests



https://github.com/riscv-verification/RVVI/tree/main/RVVI-TRACE

Open Standard: RVVI-API



- Standard functions that RISC-V processor VIPs need to implement
- Supports a step-and-compare co-simulation methodology
- C and SystemVerilog versions available
- https://github.com/riscv-verification/RVVI/blob/main/include/host/rvvi/rvvi-api.h

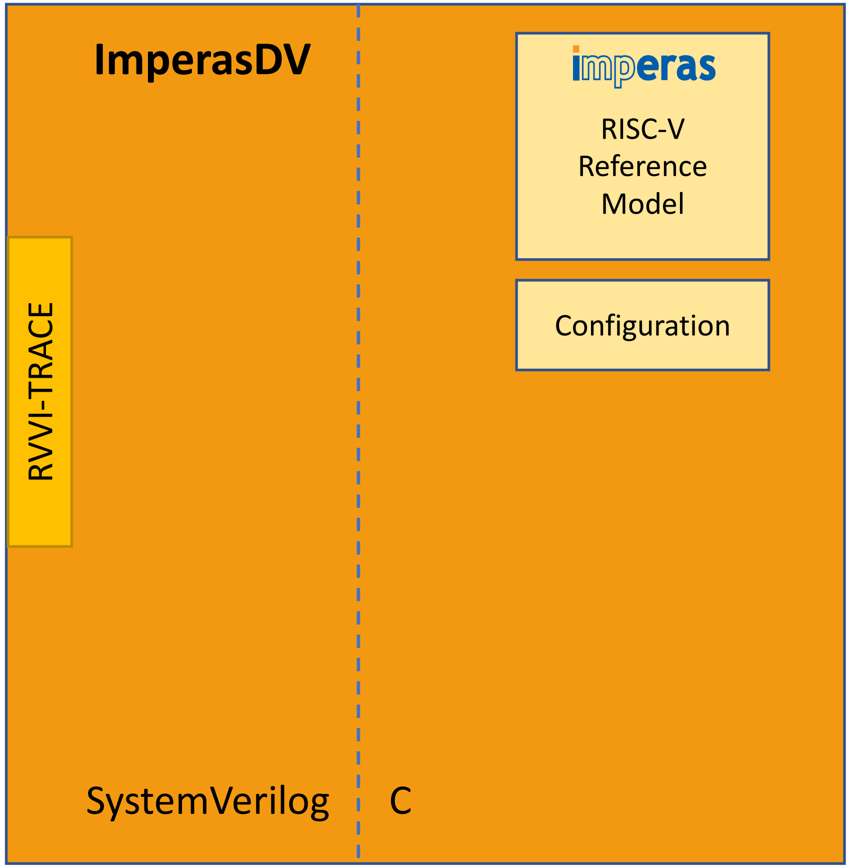
Agenda

- Introduction to Imperas
- Introduction to RISC-V
- RISC-V processor verification challenges
  - Why is RISC-V processor DV so critical?
- RISC-V processor verification environment components
- RISC-V Verification approaches
- **RISC-V Verification standards**
- **RISC-V Verification IP**
- Functional coverage for RISC-V processors
- Verification Case studies
  - OpenHW Group CV32E40X processor
  - Wally RISC-V processor
- Summary

RISC-V Processor VIP

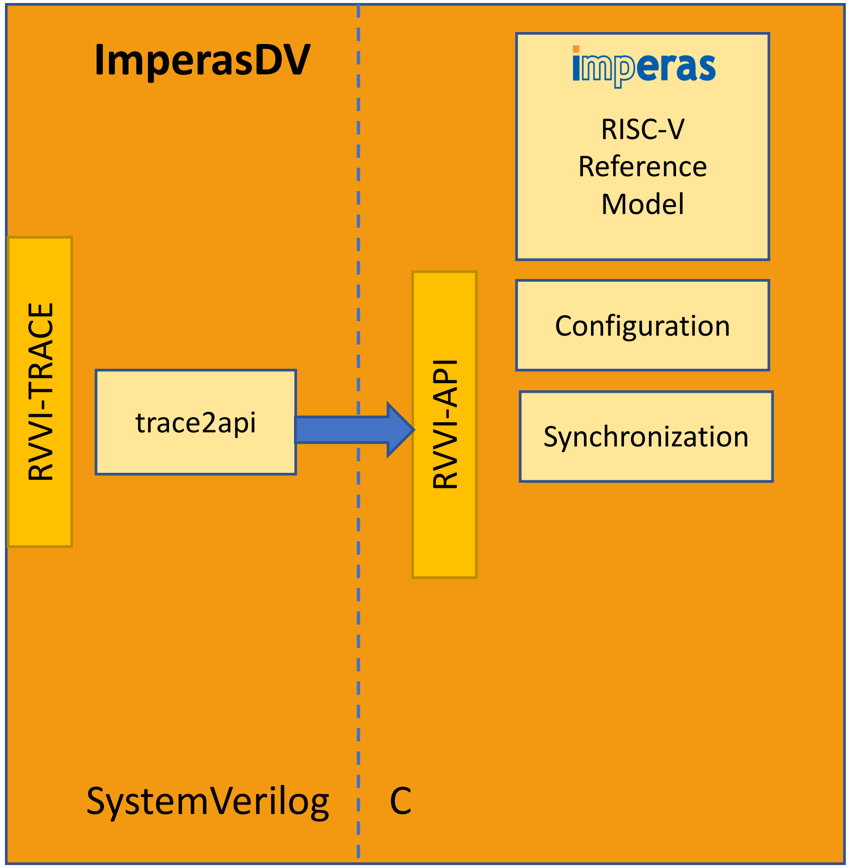
- Requirements:
  - Standard interface to receive tracer data
  - Standard way to receive asynchronous events
  - Configurable, extendable RISC-V processor reference model
  - Methods to configure, control and query the reference model
  - Mechanism to compare DUT state with the reference model and report errors/mismatches
  - A method to verify DUT response to asynchronous events

ImperasDV
Configurable Reference



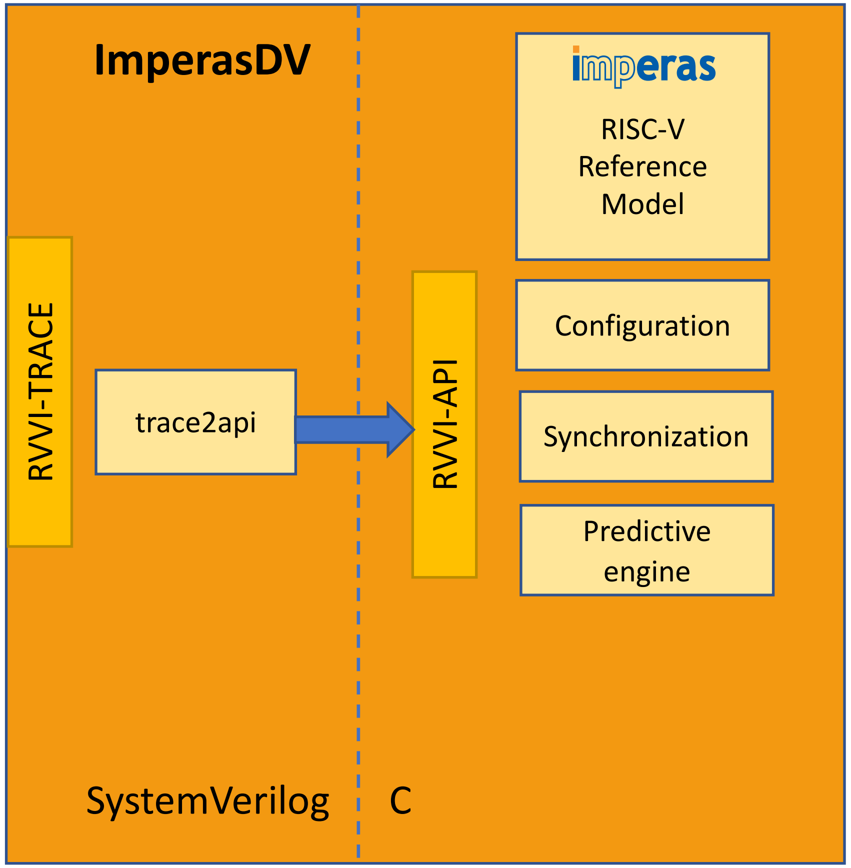
- Imperas configurable reference model
  - Fully user configurable to select required ISA extensions, versions
  - Extensible to match user customizations
- Configuration methods related to memory map (volatile regions) and CSRs

ImperasDV Components
Control and Introspection



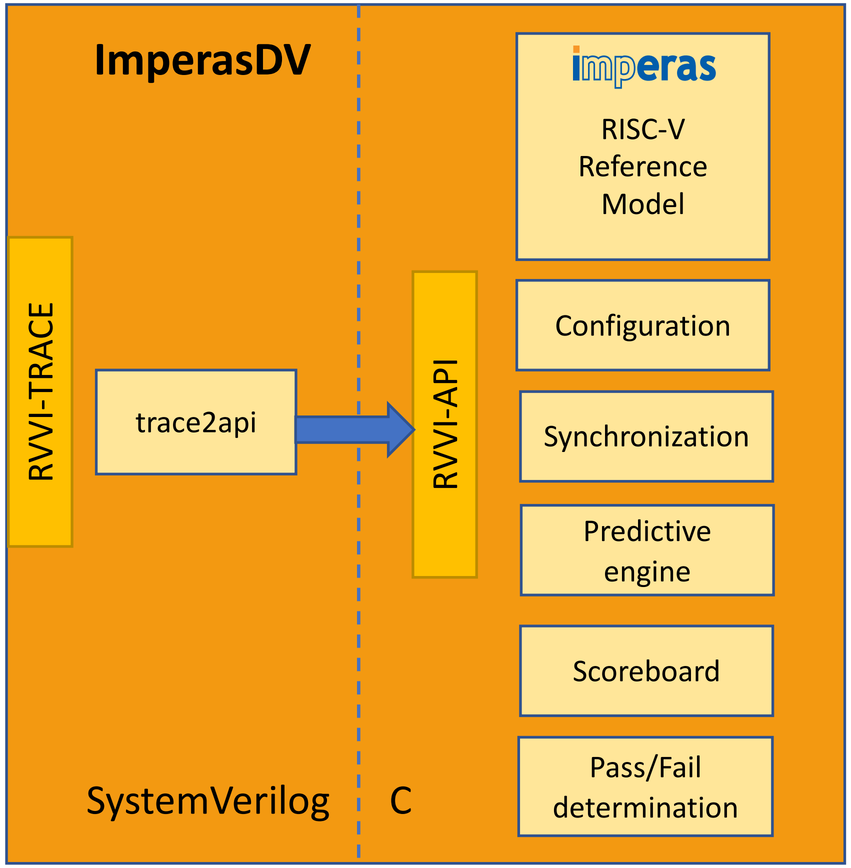
- RVVI-TRACE data is converted into function calls (RVVI-API) which provide DUT state information to the reference model
- Synchronization keeps the reference model running in lock-step with the DUT

ImperasDV Components
Asynchronous Events



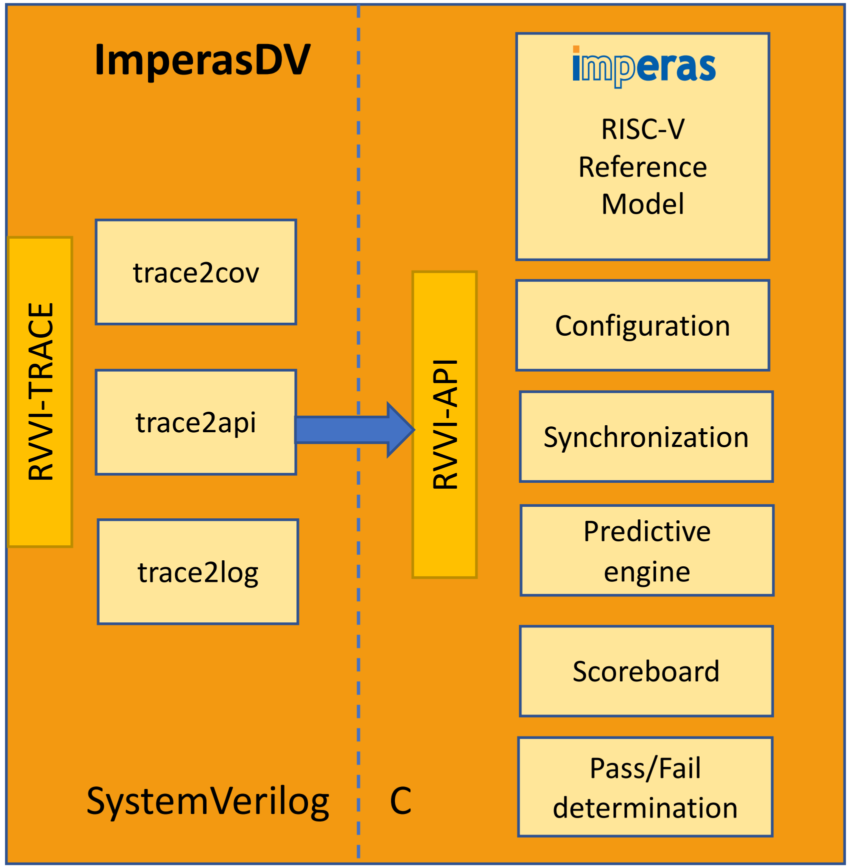
- Predictive engine is notified about asynchronous events via RVVI-API
- Analyzes the current state of the DUT and determines which responses to these events are legal

ImperasDV Components Comparison



- RVVI-API methods invoke comparison between RTL and reference
- Scoreboard keeps track of all passed and failed comparisons

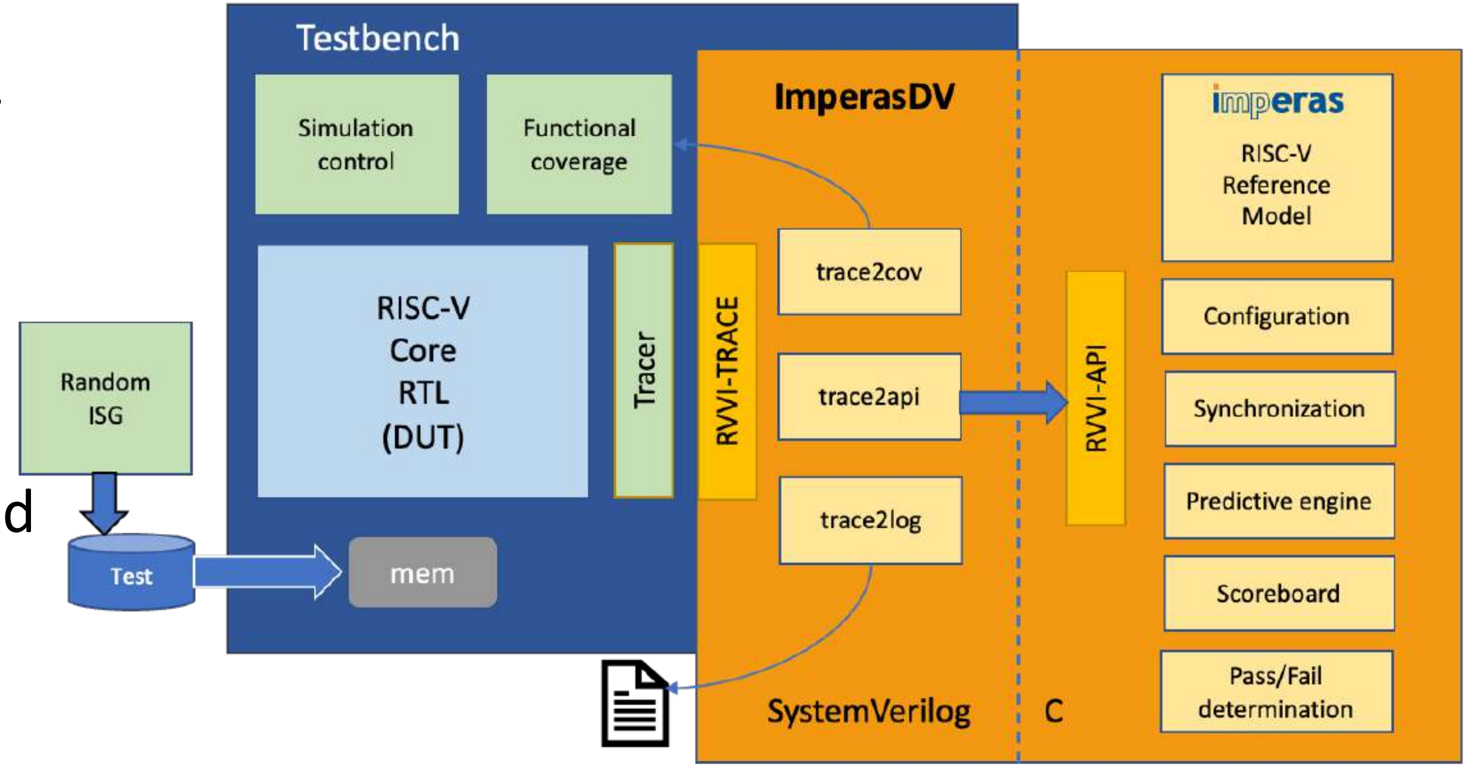
ImperasDV Components
Coverage interface and Logging



- RVVI-TRACE data is used for functional coverage sampling (trace2cov) and to produce detailed logfiles for debug (trace2log)

ImperasDV + RVVI: Process

- Instantiate VIP in a testbench
- Connect tracer using RVVI-TRACE i/f
- DUT and reference model run the same program
- Retire, trap events communicated over RVVI
- Internal state continuously compared
- RVVI-TRACE monitored for async events
- Predictive engine verifies legal scenarios



ImperasDV using RVVI

- Pros:
  - Checks full machine state at every event
  - Sequence checking is done for you
  - Errors are flagged immediately, and in detail
  - Finds synchronous and asynchronous bugs
  - Reusable across different core DV projects
  - Interchangeable due to standard interface (RVVI)
  - Ease of use
  - Training, documentation, and support

- Cons:
  - Cost of VIP licenses

Agenda

- Introduction to Imperas
- Introduction to RISC-V
- RISC-V processor verification challenges
    - Why is RISC-V processor DV so critical?
- RISC-V processor verification environment components
- RISC-V Verification approaches
- RISC-V Verification standards
- RISC-V Verification IP
- **Functional coverage for RISC-V processors**
- Verification Case studies
    - OpenHW Group CV32E40X processor
    - Wally RISC-V processor
- Summary

RISC-V Functional Coverage

For a processor there are different types of functional coverage required:

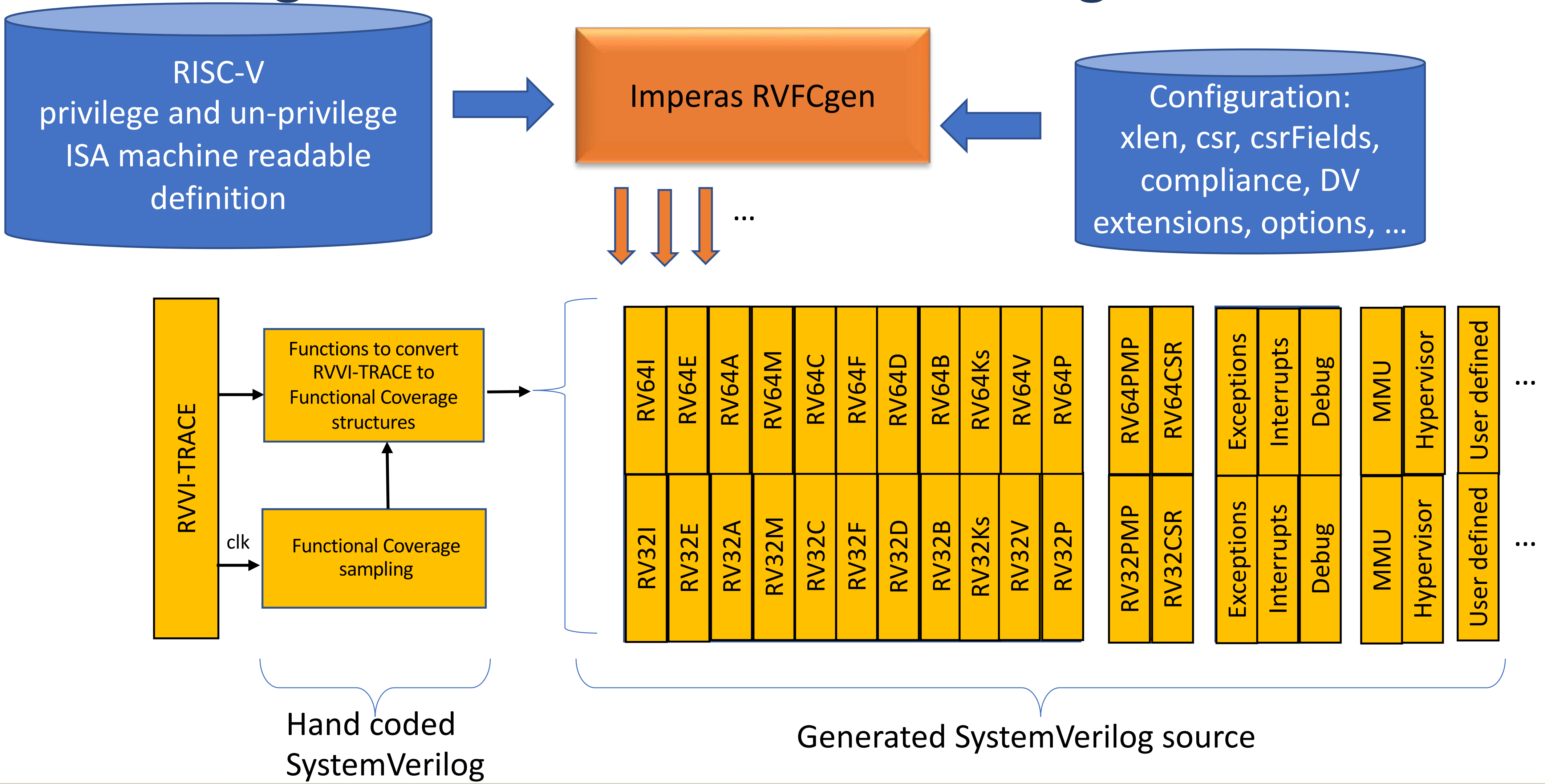
- Standard ISA architectural features
  - unpriv. ISA items: mainly instructions, their operands, their values

  => these are standard and the same for all RISC-V processors – it is the spec…
- Customer core design & micro-architectural features
  - priv. ISA items, CSRs, Interrupts, Debug block, …
  - pipeline, multi-issue, multi-hart, …
  - Custom extensions, CSRs, instructions

RISC-V Instructions (Standard ISA Architectural Feature)

- There are many different instructions in the RV64 extensions:
  - Integer: 56, Maths: 13, Compressed: 30, FP-Single: 30, FP-Double: 32
  - Vector: 356, Bitmanip: 47 Krypto-scalar: 85
  - P-DSP: 318
  - For RV64 that is ~1,000 instructions…
- Each instruction needs SystemVerilog covergroups and coverpoints
  - 10-200+ lines of SystemVerilog for each instruction
- 10,000-100,000++ lines of code to be written
  - Not design or core specific

Machine-generated Functional Coverage



riscvISACOV

https://github.com/riscv-verification/riscvISACOV

- Machine-generated functional coverage code for the RISC-V ISA
  Feb. 2023 status:
  - Extensions covered: 53
  - Instructions covered: 559
  - Covergroups: 559
  - Coverpoints: 5036
- Well documented in markdown
- Includes verification plan information in csv format
- RV32I extension available open source under Apache
- Other extensions available under Imperas Proprietary license

riscvISACOV: Coverage levels

- Compliance basic
  - Essential items to be covered
  - e.g. number of times instruction is executed, register values
- Compliance extended
  - Cross coverage using basic coverpoints
  - e.g. cross floating point register values with rounding modes
- DV Unprivileged basic
  - Essential and cross coverage involving unprivileged mode items
  - e.g. FPU special values for registers

(there are also 3 more comprehensive DV levels - WIP)

riscvISACOV: Documentation and VPlans

- Auto-generated documentation and csv files for inclusion in Verification Plans

```
RV64D,D Standard Extension for Double-Precision Floating-Point,2.2
xlen,64
""
Extension,Subset,Instruction,Description,Covergroup,Coverpoint,Coverpoint Description,Coverage Level,Pass/Fail Criteria,Test Type,Cov
RV64D,,fadd.d,,fadd_d_cg,,,,,,
,,,,,cp_asm_count,Number of times instruction is executed,Compliance Basic,Check against Reference Model,Constrained-Random,Functiona
,,,,,cp_fd,FD (FPR) register assignment,Compliance Basic,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cp_fs1,FS1 (FPR) register assignment,Compliance Basic,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cp_fs2,FS2 (FPR) register assignment,Compliance Basic,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cp_frm,Floating Point FRM (Rounding mode) given as an operand,Compliance Basic,Check against Reference Model,Constrained-Random,
,,,,,cp_csr_fcsr_frm,"Value of fcsr CSR, frm field",Compliance Basic,Check against Reference Model,Constrained-Random,Functional Cove
,,,,,cp_csr_fcsr_fflags,"Value of fcsr CSR, fflags field",Compliance Basic,Check against Reference Model,Constrained-Random,Functiona
,,,,,cp_csr_frm_frm,"Value of frm CSR, frm field",Compliance Basic,Check against Reference Model,Constrained-Random,Functional Covera
,,,,,cp_csr_fflags_fflags,"Value of fflags CSR, fflags field",Compliance Basic,Check against Reference Model,Constrained-Random,Funct
,,,,,cp_fd_vals,FD FPU Special values,DV Un-privileged Basic,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cp_fs1_vals,FS1 FPU Special values,DV Un-privileged Basic,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cp_fs2_vals,FS2 FPU Special values,DV Un-privileged Basic,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cr_fd_frm,FD FRM (ins rounding mode) Cross,Compliance Extended,Check against Reference Model,Constrained-Random,Functional Cover
,,,,,cr_fs1_frm,FS1 FRM (ins rounding mode) Cross,Compliance Extended,Check against Reference Model,Constrained-Random,Functional Cov
,,,,,cr_fs2_frm,FS2 FRM (ins rounding mode) Cross,Compliance Extended,Check against Reference Model,Constrained-Random,Functional Cov
,,,,,cr_fd_vals,FD FPU values Cross,DV Un-privileged Basic,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cr_fs1_vals,FS1 FPU values Cross,DV Un-privileged Basic,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cr_fs2_vals,FS2 FPU values Cross,DV Un-privileged Basic,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cr_fd_fs1,FD FS1 Cross,Compliance Extended,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cr_fd_fs2,FD FS2 Cross,Compliance Extended,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cr_fs1_fs2,FS1 FS2 Cross,Compliance Extended,Check against Reference Model,Constrained-Random,Functional Coverage
RV64D,,fclass.d,,fclass_d_cg,,,,,,
,,,,,cp_asm_count,Number of times instruction is executed,Compliance Basic,Check against Reference Model,Constrained-Random,Functional Coverage
,,,,,cp_rd,RD (GPR) register assignment,Compliance Basic,Check against Reference Model,Constrained-Random,Functional Coverage
```

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

Functional Coverage Examples

- riscvISACOV
  - https://github.com/riscv-verification/riscvISACOV
- OpenHW Group core-v-verif
  - https://github.com/openhwgroup/core-v-verif/tree/master/cv32e40s/env/uvme/cov

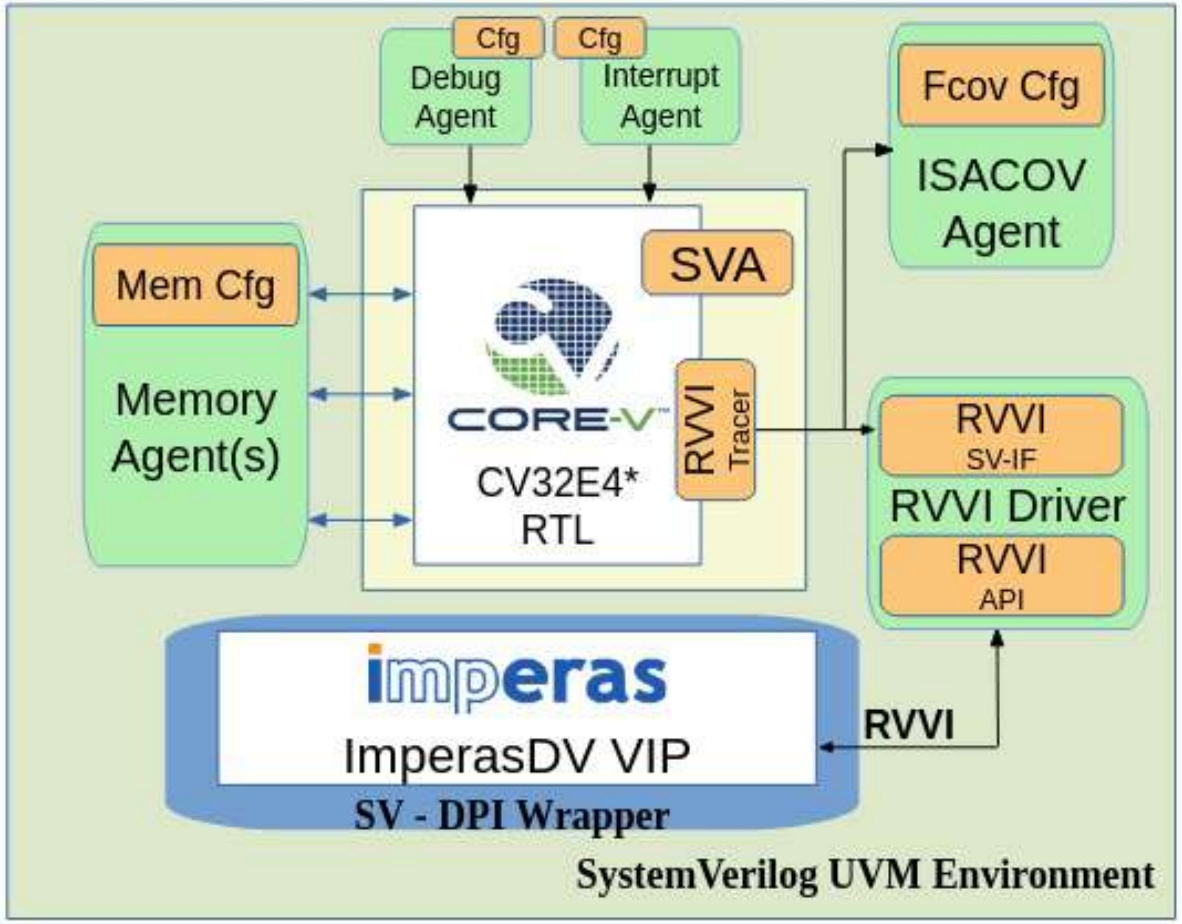
Agenda

- Introduction to Imperas
- Introduction to RISC-V
- RISC-V processor verification challenges
  - Why is RISC-V processor DV so critical?
- RISC-V processor verification environment components
- RISC-V Verification approaches
- RISC-V Verification standards
- RISC-V Verification IP
- Functional coverage for RISC-V processors
- **Verification Case studies**
  - OpenHW Group CV32E40X processor
  - Wally RISC-V processor
- Summary

Verification Case Study – CV32E40X

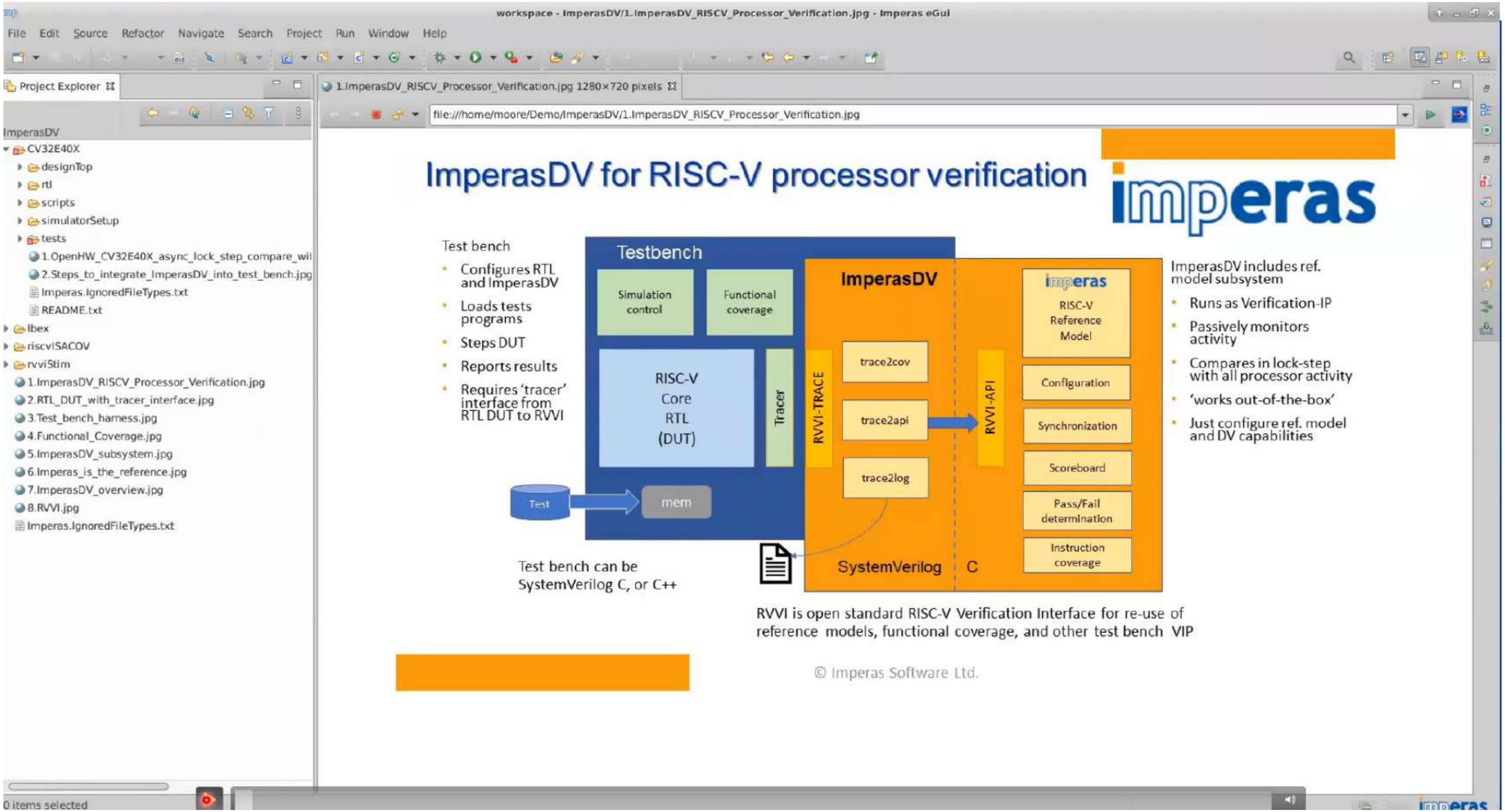
- OpenHW Group CV32E40X RISC-V core
  - 4 stage pipeline, embedded class core:
    - RV32I, RV32E
    - [M|Zmmul][A]Zca_Zcb_Zcmb_Zcmp_Zcmt[Zba_Zbb_Zbs|Zba_Zbb_Zbc_Zbs]ZicntrZihpm ZicsrZifence
    - X interface
  - Evolved from work on the CV32E40P core (originated from Pulp platform)
    - Focus of OpenHW Group is high-quality cores verified to industry standards
  - CORE-V-VERIF environment modified to use ImperasDV in fall 2022

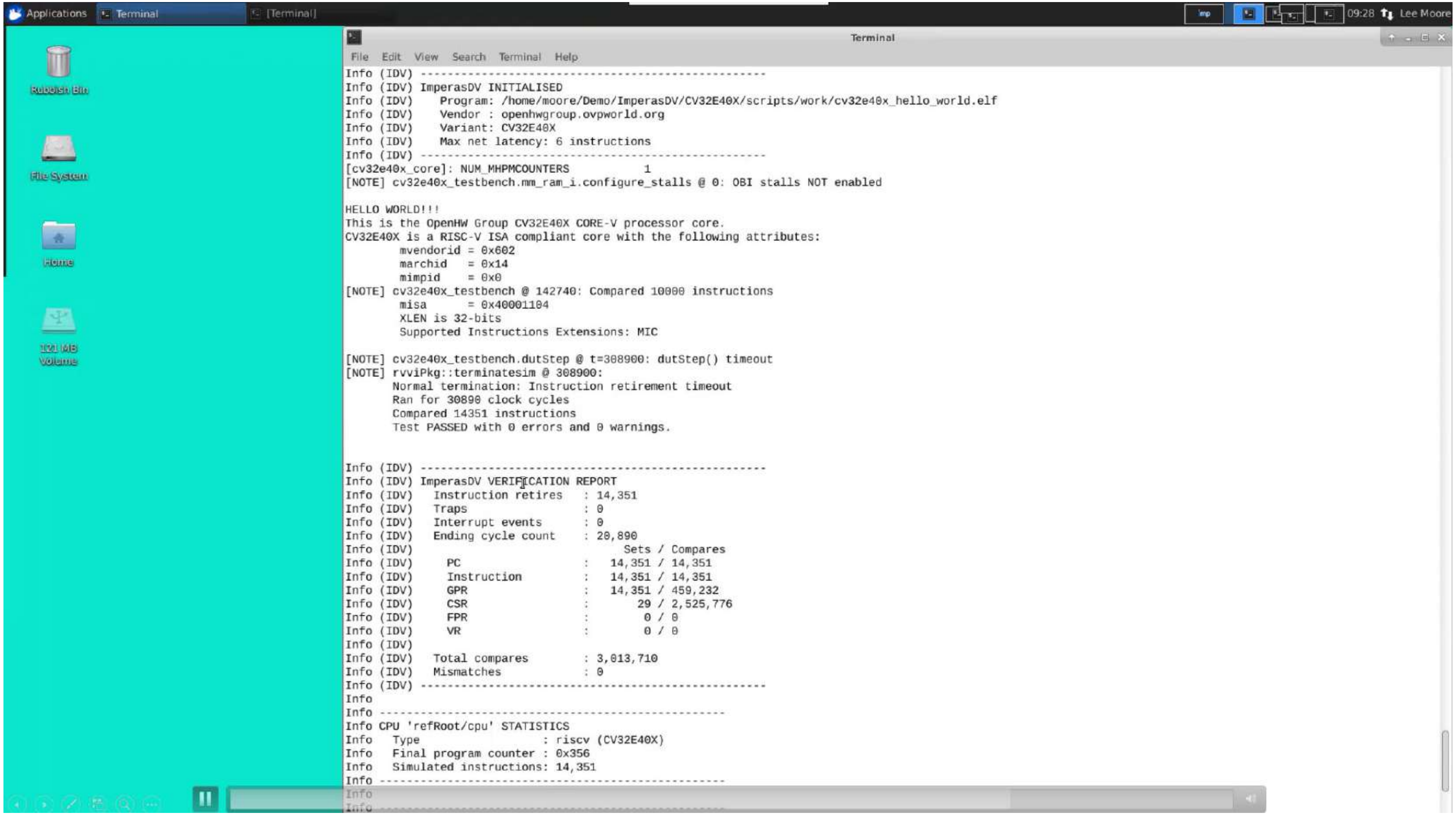
CORE-V-VERIF using ImperasDV



Demonstration

- DUT: OpenHW Group CV32E40X RISC-V processor
  - Simulation: passing test
  - Simulation: failing test
  - Simulation: asynchronous event bug
- Screenshots from the video demonstration now follow


workspace - ImperasDV/1.ImperasDV_RISCV_Processor_Verification.jpg - Imperas eGui
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
1.ImperasDV_RISCV_Processor_Verification.jpg 1280×720 pixels
file:///home/moore/Demo/ImperasDV/1.ImperasDV_RISCV_Processor_Verification.jpg
ImperasDV
CV32E40X
designTop
rtl
scripts
simulatorSetup
tests
1.OpenHW_CV32E40X_async_lock_step_compare_wi
2.Steps_to_integrate_ImperasDV_into_test_bench.jpg
Imperas.IgnoredFileTypes.txt
README.txt
Ibex
riscvISACOV
rvviStim
1.ImperasDV_RISCV_Processor_Verification.jpg
2.RTL_DUT_with_tracer_interface.jpg
3.Test_bench_harness.jpg
4.Functional_Coverage.jpg
5.ImperasDV_subsystem.jpg
6.Imperas_is_the_reference.jpg
7.ImperasDV_overview.jpg
8.RVVI.jpg
Imperas.IgnoredFileTypes.txt
0 items selected
ImperasDV for RISC-V processor verification
imperas
Test bench
Configures RTL and ImperasDV
Loads tests programs
Steps DUT
Reports results
Requires 'tracer' interface from RTL DUT to RVVI
Testbench
Simulation control
Functional coverage
RISC-V Core RTL (DUT)
Tracer
Test
mem
ImperasDV
RVVI-TRACE
trace2cov
trace2api
trace2log
RVVI-API
imperas
RISC-V Reference Model
Configuration
Synchronization
Scoreboard
Pass/Fail determination
Instruction coverage
SystemVerilog
C
ImperasDV includes ref. model subsystem
Runs as Verification-IP
Passively monitors activity
Compares in lock-step with all processor activity
'works out-of-the-box'
Just configure ref. model and DV capabilities
Test bench can be SystemVerilog C, or C++
RVVI is open standard RISC-V Verification Interface for re-use of reference models, functional coverage, and other test bench VIP
© Imperas Software Ltd.


Applications
Terminal
Terminal
File Edit View Search Terminal Help
Info (IDV) --
Info (IDV) ImperasDV INITIALISED
Info (IDV)   Program: /home/moore/Demo/ImperasDV/CV32E40X/scripts/work/cv32e40x_hello_world.elf
Info (IDV)   Vendor : openhwgroup.ovpworld.org
Info (IDV)   Variant: CV32E40X
Info (IDV)   Max net latency: 6 instructions
Info (IDV) --
[cv32e40x_core]: NUM_MHPMCOUNTERS 1
[NOTE] cv32e40x_testbench.mm_ram_i.configure_stalls @ 0: OBI stalls NOT enabled
HELLO WORLD!!!
This is the OpenHW Group CV32E40X CORE-V processor core.
CV32E40X is a RISC-V ISA compliant core with the following attributes:
mvendorid = 0x602
marchid = 0x14
mimpid = 0x0
[NOTE] cv32e40x_testbench @ 142740: Compared 10000 instructions
misa = 0x40001104
XLEN is 32-bits
Supported Instructions Extensions: MIC
[NOTE] cv32e40x_testbench.dutStep @ t=308900: dutStep() timeout
[NOTE] rvviPkg::terminatesim @ 308900:
Normal termination: Instruction retirement timeout
Ran for 30890 clock cycles
Compared 14351 instructions
Test PASSED with 0 errors and 0 warnings.
Info (IDV) --
Info (IDV) ImperasDV VERIFICATION REPORT
Info (IDV)   Instruction retires : 14,351
Info (IDV)   Traps : 0
Info (IDV)   Interrupt events : 0
Info (IDV)   Ending cycle count : 20,890
Info (IDV)   Sets / Compares
Info (IDV)   PC : 14,351 / 14,351
Info (IDV)   Instruction : 14,351 / 14,351
Info (IDV)   GPR : 14,351 / 459,232
Info (IDV)   CSR : 29 / 2,525,776
Info (IDV)   FPR : 0 / 0
Info (IDV)   VR : 0 / 0
Info (IDV)
Info (IDV)   Total compares : 3,013,710
Info (IDV)   Mismatches : 0
Info (IDV) --
Info
Info --
Info CPU 'refRoot/cpu' STATISTICS
Info   Type : riscv (CV32E40X)
Info   Final program counter : 0x356
Info   Simulated instructions: 14,351
Info --

imperas

Applications Terminal [Terminal] 10:19 Lee Moore

Terminal

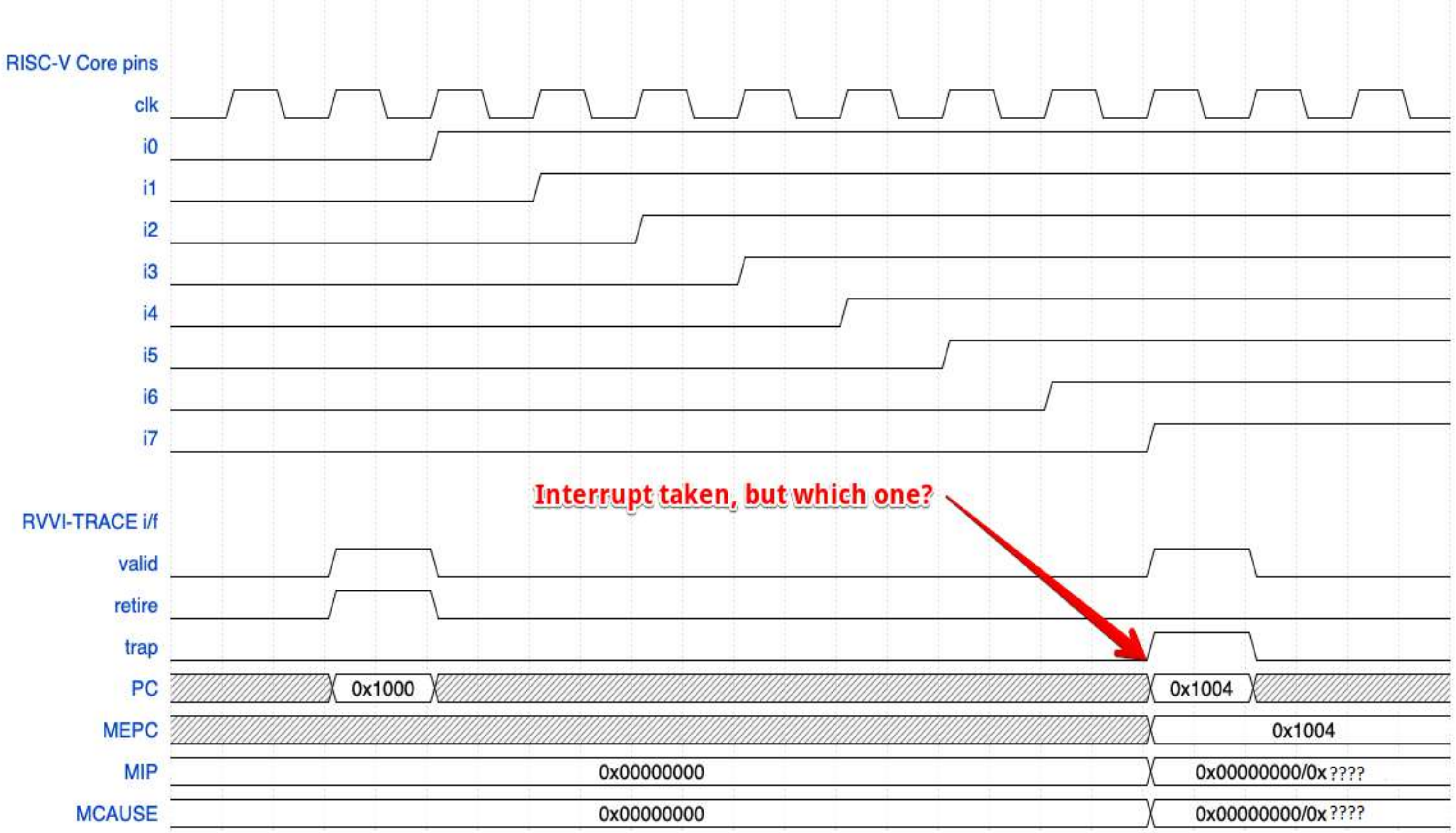
File Edit View Search Terminal Help

```
!!!
      Ran for 37253 clock cycles
      Compared 19336 instructions
      Test FAILED due to fatal error(s) and 5 non-fatal errors and 0 warnings.


Info (IDV) ---------------------------------------------------
Info (IDV) ImperasDV VERIFICATION REPORT
Info (IDV)   Instruction retires    : 19,337
Info (IDV)   Traps                  : 0
Info (IDV)   Interrupt events       : 35
Info (IDV)   Ending cycle count     : 37,253
Info (IDV)                              Sets / Compares
Info (IDV)     PC                   :   19,337 / 19,337
Info (IDV)     Instruction          :   19,337 / 19,337
Info (IDV)     GPR                  :   19,337 / 618,784
Info (IDV)     CSR                  :   75,325 / 3,403,312
Info (IDV)     FPR                  :        0 / 0
Info (IDV)     VR                   :        0 / 0
Info (IDV)
Info (IDV)   Total compares         : 4,060,770
Info (IDV)   Mismatches             : 5
Info (IDV)   PC diverged from DUT   : 5
Info (IDV)   Warnings               : 5
Info (IDV) ---------------------------------------------------
Info (IDV) During execution, the reference model was forced to match DUT state which should be treated as a test failure.
Info (IDV) This feature is only provided to help expose further issues causing the DUT and reference to diverge.
Info
Info ---------------------------------------------------
Info CPU 'refRoot/cpu' STATISTICS
Info   Type                  : riscv (CV32E40X)
Info   Final program counter : 0x1304
Info   Simulated instructions: 19,332
Info ---------------------------------------------------
Info
Info ---------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time        : 0.00 seconds
Info   User time             : 6.38 seconds
Info   System time           : 0.94 seconds
Info   Elapsed time          : 7.49 seconds
Info ---------------------------------------------------

ImperasDVasync finished: Fri Dec  9 10:19:26 2022


ImperasDVasync (64-Bit) v20221209.0 Open Virtual Platform simulator from www.IMPERAS.com.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

Simulation complete via $finish(1) at time 372526 NS + 0
/home/moore/Imperas/ImpProprietary/source/host/rvvi/rvvi-pkg.sv:138     $finish; // this should be the only call to $finish in the testbench
xcelium> exit
TOOL:   xrun(64)        20.03-s010: Exiting on Dec 09, 2022 at 10:19:26 GMT  (total: 00:00:09)
x6476:~/Demo/ImperasDV/CV32E40X/scripts$
```

```
Info 211: 'refRoot/cpu', 0x0000000000001930(_malloc_r+8): Machine 00358493 addi    x9,x11,3
Info   x9 00000138 -> 000001af
[NOTE] cv32e40x_testbench.idv_trace2log @ 2966: RET,0,212,00001934,"ffc4f493 andi    x9,x9,-4        ",x9=000001ac,,,,
Info 212: 'refRoot/cpu', 0x0000000000001934(_malloc_r+c): Machine ffc4f493 andi    x9,x9,-4
Info   x9 000001af -> 000001ac
[NOTE] cv32e40x_testbench.idv_trace2log @ 2976: RET,0,213,00001938,"01212823 sw      x18,16(x2)      ",,,,,
Info 213: 'refRoot/cpu', 0x0000000000001938(_malloc_r+10): Machine 01212823 sw      x18,16(x2)
[NOTE] cv32e40x_testbench.idv_trace2log @ 2986: RET,0,214,0000193c,"00112e23 sw      x1,28(x2)       ",,,,,
Info 214: 'refRoot/cpu', 0x000000000000193c(_malloc_r+14): Machine 00112e23 sw      x1,28(x2)
[NOTE] cv32e40x_testbench.idv_trace2log @ 2996: RET,0,215,00001940,"00812c23 sw      x8,24(x2)       ",,,,,
Info 215: 'refRoot/cpu', 0x0000000000001940(_malloc_r+18): Machine 00812c23 sw      x8,24(x2)
[NOTE] cv32e40x_testbench.idv_trace2log @ 3006: RET,0,216,00001944,"01312623 sw      x19,12(x2)      ",,,,,
Info 216: 'refRoot/cpu', 0x0000000000001944(_malloc_r+1c): Machine 01312623 sw      x19,12(x2)
[NOTE] cv32e40x_testbench.idv_trace2log @ 3016: RET,0,217,00001948,"00848493 addi    x9,x9,8         ",x9=000001b4,,,,
Info 217: 'refRoot/cpu', 0x0000000000001948(_malloc_r+20): Machine 00848493 addi    x9,x9,8
Info   x9 000001ac -> 000001b4
[NOTE] cv32e40x_testbench.idv_trace2log @ 3026: RET,0,218,0000194c,"00c00793 addi    x15,x0,12       ",x15=0000000c,,,,
Info 218: 'refRoot/cpu', 0x000000000000194c(_malloc_r+24): Machine 00c00793 addi    x15,x0,12
Info   x15 00000068 -> 0000000c
[NOTE] cv32e40x_testbench.idv_trace2log @ 3036: RET,0,219,00001950,"00050913 mv      x18,x10         ",x18=00003518,,,,
Info 219: 'refRoot/cpu', 0x0000000000001950(_malloc_r+28): Machine 00050913 mv      x18,x10
Info   x18 00000004 -> 00003518
[NOTE] cv32e40x_testbench.idv_trace2log @ 3046: RET,0,220,00001954,"06f4f263 bgeu    x9,x15,19b8     ",,,,,
Info 220: 'refRoot/cpu', 0x0000000000001954(_malloc_r+2c): Machine 06f4f263 bgeu    x9,x15,19b8
[NOTE] cv32e40x_testbench.idv_trace2log @ 3056: RET,0,221,00001958,"00c00493 addi    x9,x0,12        ",x9=0000000c,,,,
Info 221: 'refRoot/cpu', 0x00000000000019b8(_malloc_r+90): Machine fa04d2e3 bgez    x9,195c
Error (IDV) PC mismatch (HartId:0, PC:0x000019b8 _malloc_r+144):
Info (IDV)  0>
Info (IDV)    . dut:0x00001958 _malloc_r+48
Info (IDV)    . ref:0x000019b8 _malloc_r+144
Error (IDV) Insn. bit pattern mismatch (HartId:0, PC:0x000019b8 _malloc_r+144):
Info (IDV)  1>
Info (IDV)    . dut:00c00493 addi    x9,x0,12
Info (IDV)    . ref:fa04d2e3 bgez    x9,195c
Error (IDV) GPR register value mismatch (HartId:0, PC:0x000019b8 _malloc_r+144):
Info (IDV)  2> GPR x9
Info (IDV)    . dut:0x0000000c
Info (IDV)    . ref:0x000001b4
[ERROR] cv32e40x_testbench.idv_trace2api.state_compare @ 3056: MISMATCH
[NOTE] cv32e40x_testbench.idv_trace2log @ 3066: RET,0,222,0000195c,"06b4e063 bltu    x9,x11,19bc     ",,,,,
Info 222: 'refRoot/cpu', 0x000000000000195c(_malloc_r+34): Machine 06b4e063 bltu    x9,x11,19bc
Error (IDV) GPR register value mismatch (HartId:0, PC:0x0000195c _malloc_r+52):
Info (IDV)  3> GPR x9
Info (IDV)    . dut:0x0000000c
Info (IDV)    . ref:0x000001b4
[ERROR] cv32e40x_testbench.idv_trace2api.state_compare @ 3066: MISMATCH
[NOTE] cv32e40x_testbench.idv_trace2log @ 3096: RET,0,223,000019bc,"00c00793 addi    x15,x0,12       ",x15=0000000c,,,,
Info 223: 'refRoot/cpu', 0x0000000000001960(_malloc_r+38): Machine 00090513 mv      x10,x18
Error (IDV) PC mismatch (HartId:0, PC:0x00001960 _malloc_r+56):
Info (IDV)  4>
Info (IDV)    . dut:0x000019bc _malloc_r+148
Info (IDV)    . ref:0x00001960 _malloc_r+56
Error (IDV) Insn. bit pattern mismatch (HartId:0, PC:0x00001960 _malloc_r+56):
Info (IDV)  5>
```

VIDEO: Asynchronous

- 4:38



```
Info (OR_OF) Target 'refRoot/cpu' has object file read from '/home/moore/Demo/ImperasDV/CV32E40X/scripts/work/interrupt_test.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00001000 0x00000000 0x00000000 0x00004f88 0x00400000 RWE   1000
Info (OR_PD) LOAD           0x00000010 0x00005010 0x00005010 0x00000000 0x003faff0 RW-   1000
Info (OR_PD) LOAD           0x00006000 0x00100000 0x00100000 0x0000008a 0x0000008a R-E   1000
Info (OR_PD) LOAD           0x00000000 0x003fc000 0x003fc000 0x00000000 0x00000080 RW-   1000
Info (IDV) ---------------------------------------------------------
Info (IDV) ImperasDV INITIALISED
Info (IDV)    Program: /home/moore/Demo/ImperasDV/CV32E40X/scripts/work/interrupt_test.elf
Info (IDV)    Vendor : openhwgroup.ovpworld.org
Info (IDV)    Variant: CV32E40X
Info (IDV)    Max net latency: 6 instructions
Info (IDV) ---------------------------------------------------------
[cv32e40x_core]: NUM_MHPMCOUNTERS          1
[NOTE] cv32e40x_testbench.mm_ram_i.configure_stalls @ 0: OBI stalls NOT enabled
TEST 1 - TRIGGER ALL IRQS IN SEQUENCE:
[NOTE] cv32e40x_testbench @ 191240: Compared 10000 instructions
TEST 2 - TRIGGER ALL IRQS AT ONCE:
Error (IDV) The reference model is unable to find a legal set of actions to converge to the provided DUT state.
Info (IDV) Currently applied nets:
Info (IDV)  - MSWInterrupt => 1
Info (IDV)  - MTimerInterrupt => 1
Info (IDV)  - MExternalInterrupt => 1
Info (IDV) Pending net changes:
Info (IDV)  - Group 0:
Info (IDV)  -       MSWInterrupt => 0,  age = 3 cycles
Info (IDV) DUT state change (hartId: 0):
Info (IDV)  - dut.pc                  - 0x0000000c vector_table+12
Info (IDV)  - dut.x0                  - 0x00000000
Info (IDV)  - dut.mstatus             - 0x00001880 SD:0 XS:0(Off) FS:0(Off) MPP:3 VS:0(Off) MPIE:1 MIE:0
Info (IDV)  - dut.mcause              - 0x80000003 Interrupt:1 Code:3(Machine software interrupt)
Info (IDV)  - dut.mcycle              - 0x00008e00
Info (IDV)  - dut.minstret            - 0x0000492a
Info (IDV)  - dut.cycle               - 0x00008e00
Info (IDV)  - dut.instret             - 0x0000492a
Info (IDV) Evaluated progression tree:

 + Net MSWInterrupt => 0x0 { when:37244 }
 |  + Exception { pc:0x1304(test2+74), mcause:0x8000000b, mstatus:0x1880 }
 |  |  + Retire { pc:0x2c(vector_table+44) }
 > dut.pc           - 0x0000000c vector_table+12
   ref.pc           - 0x0000002c vector_table+44
 > dut.mcause       - 0x80000003 Interrupt:1 Code:3(Machine software interrupt)
   ref.mcause       - 0x8000000b Interrupt:1 Code:11(Machine external interrupt)
 + Exception { pc:0x1304(test2+74), mcause:0x8000000b, mstatus:0x1880 }
 |  + Retire { pc:0x2c(vector_table+44) }
 > dut.pc           - 0x0000000c vector_table+12
   ref.pc           - 0x0000002c vector_table+44
 > dut.mcause       - 0x80000003 Interrupt:1 Code:3(Machine software interrupt)
   ref.mcause       - 0x8000000b Interrupt:1 Code:11(Machine external interrupt)
Info (IDV) No evaluated progression caused convergence to DUT state.

Warning (IDV) The reference is being forced to converge to the DUT state, results that follow should be treated with caution.
```

Verification Case Study – HMC/OSU Wally

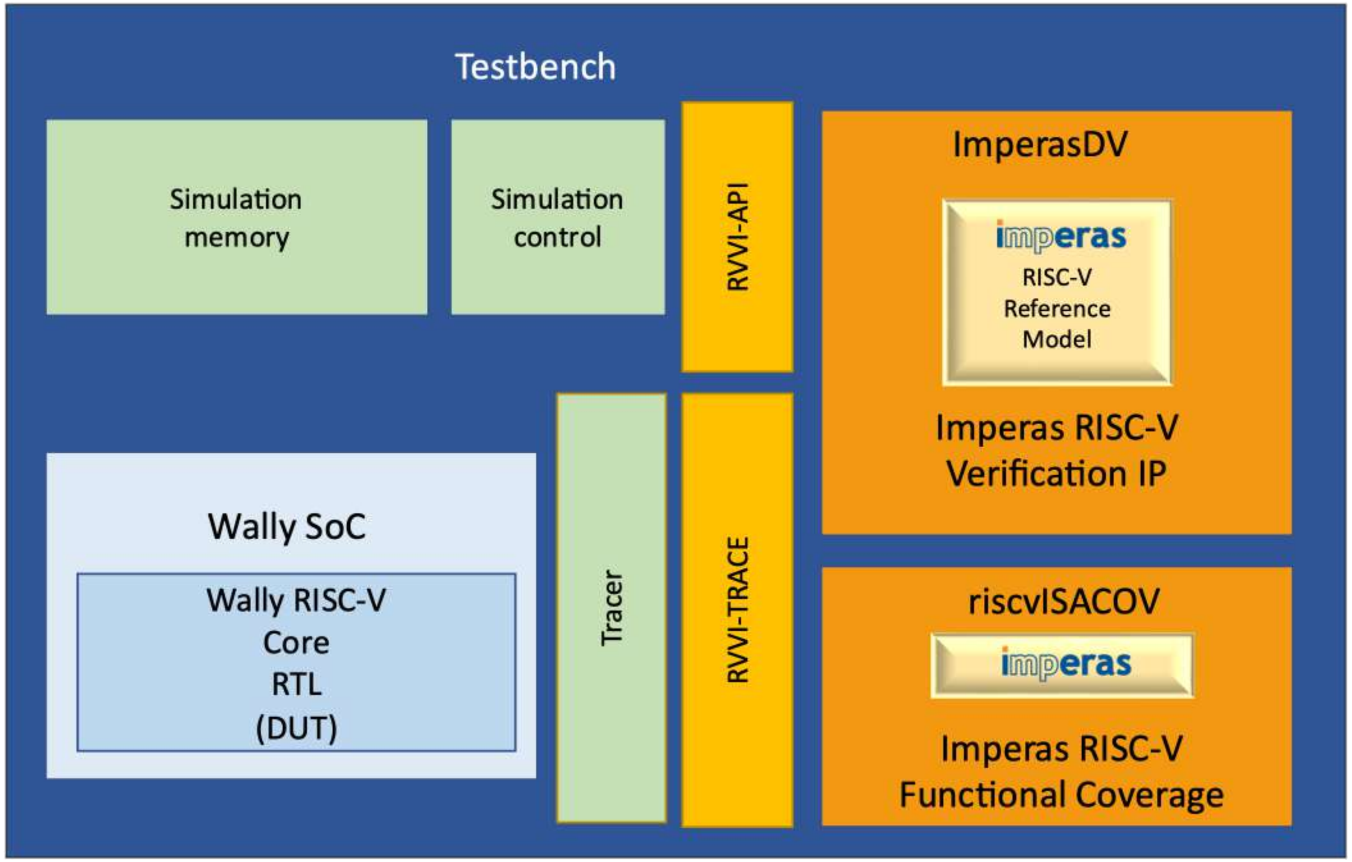
- Overview of the core
- Testbench with RVVI, ImperasDV
- Demonstration runs
- Current status

Verification Case Studies

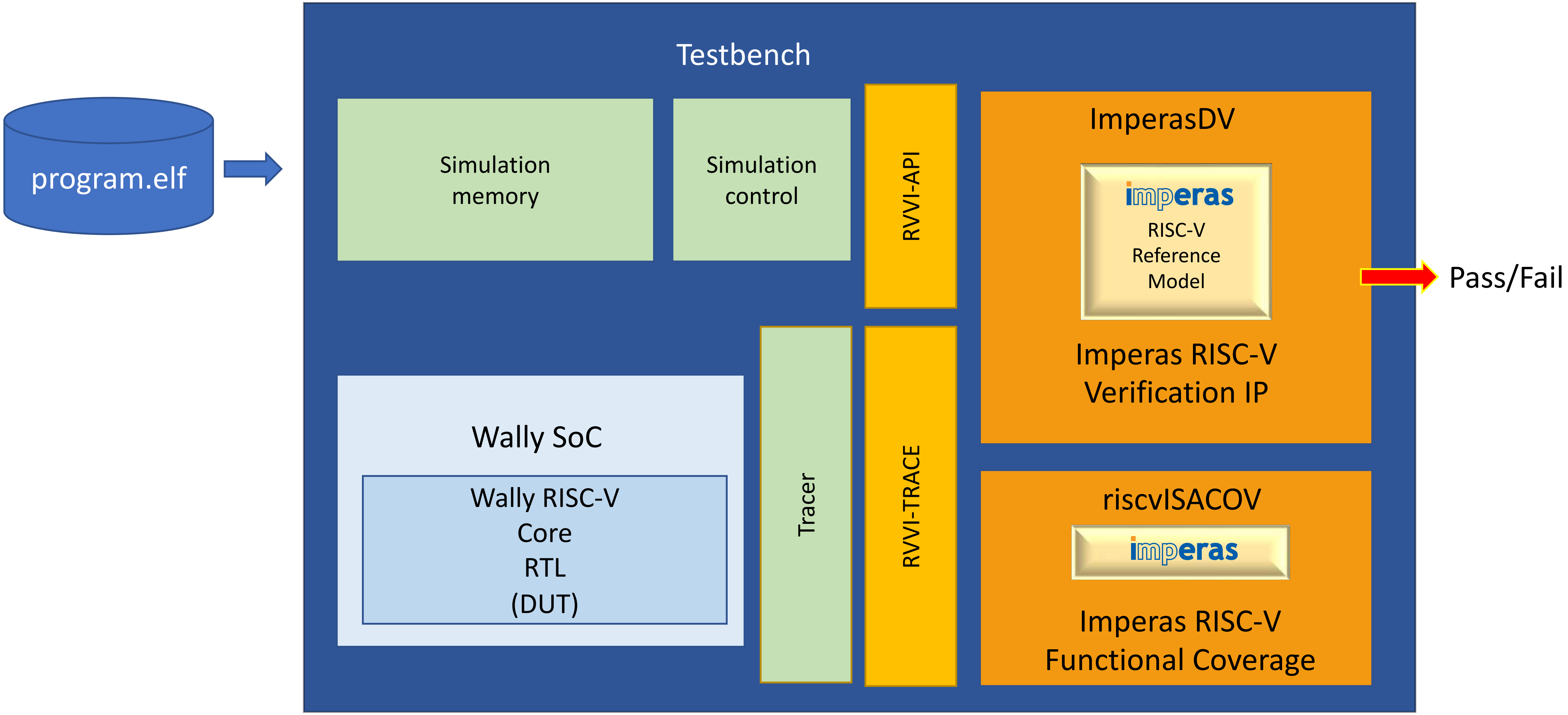
- Wally RISC-V core
  - Configurable core:
    - RV32I, RV32E, RV64I, RV64E
    - A, C, F, D, M extensions, privileged modes, CSRs
    - MMU/TLB virtual memory, caches
  - Developed at Harvey Mudd College / Oklahoma State University
    - Focus is high quality core for processor architecture education
  - Status in January 2023 – before starting to use ImperasDV for verification:
    - passing all RISC-V International compliance tests, Imperas compatibility tests
      - Using Compliance Level post sim signature file compare
    - boots Linux
  - now in OpenHW as CORE-V Wally (https://github.com/openhwgroup/cvw)

Wally + ImperasDV

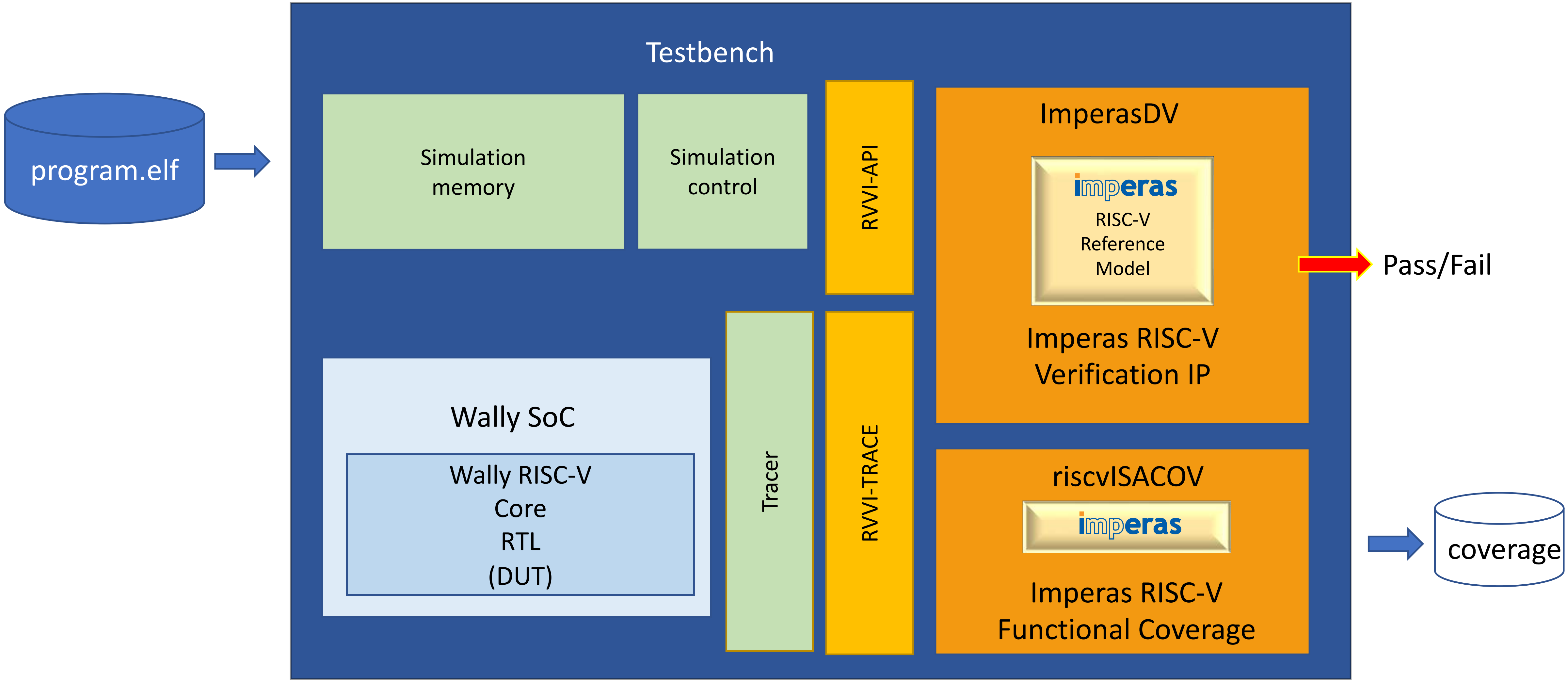
- RVVI Tracer: 1/2 day of effort
- Testbench: 1/2 day of integration
- 2 days effort resolve tracer/integration issues



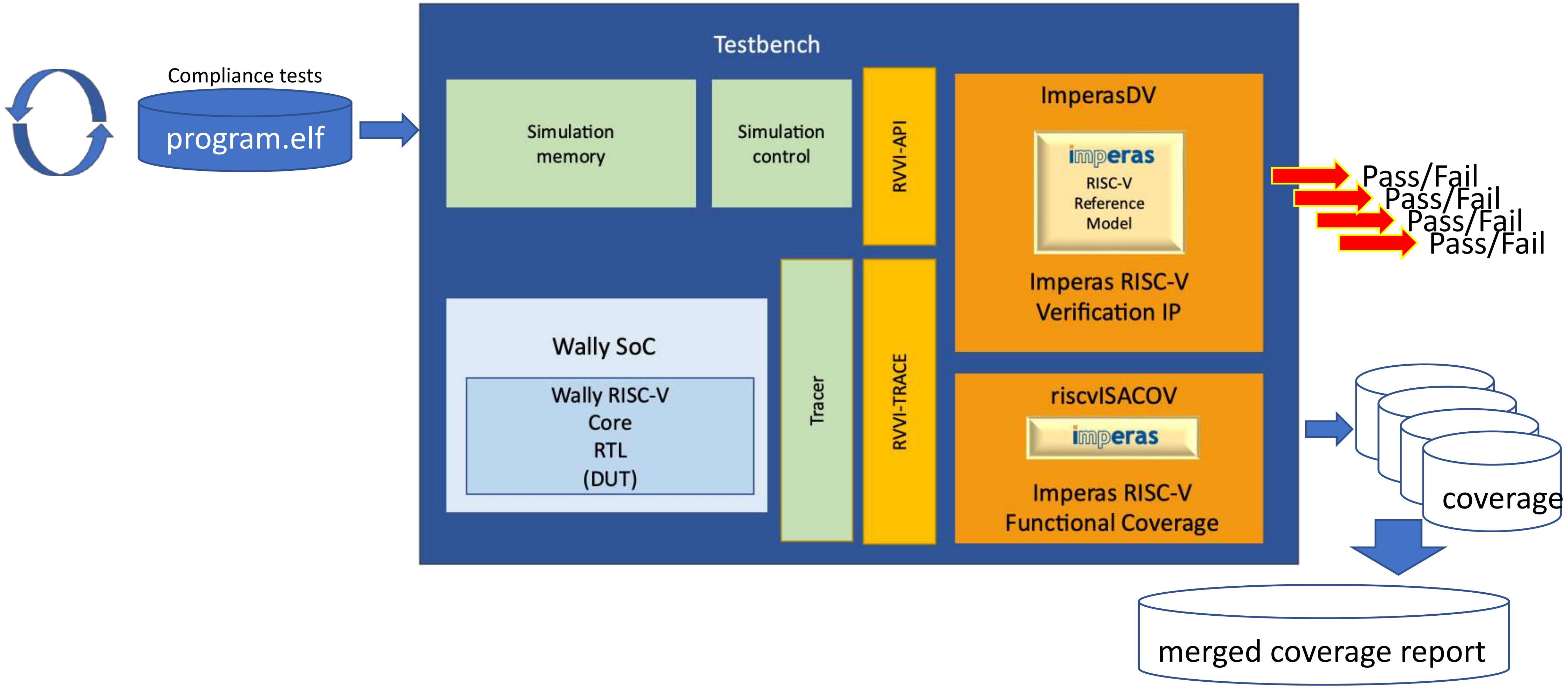
Wally: RVVI, ImperasDV: base use model: verification



Wally: RVVI, ImperasDV: verification with coverage



Wally: RVVI, ImperasDV: verification with compliance suite & merged coverage



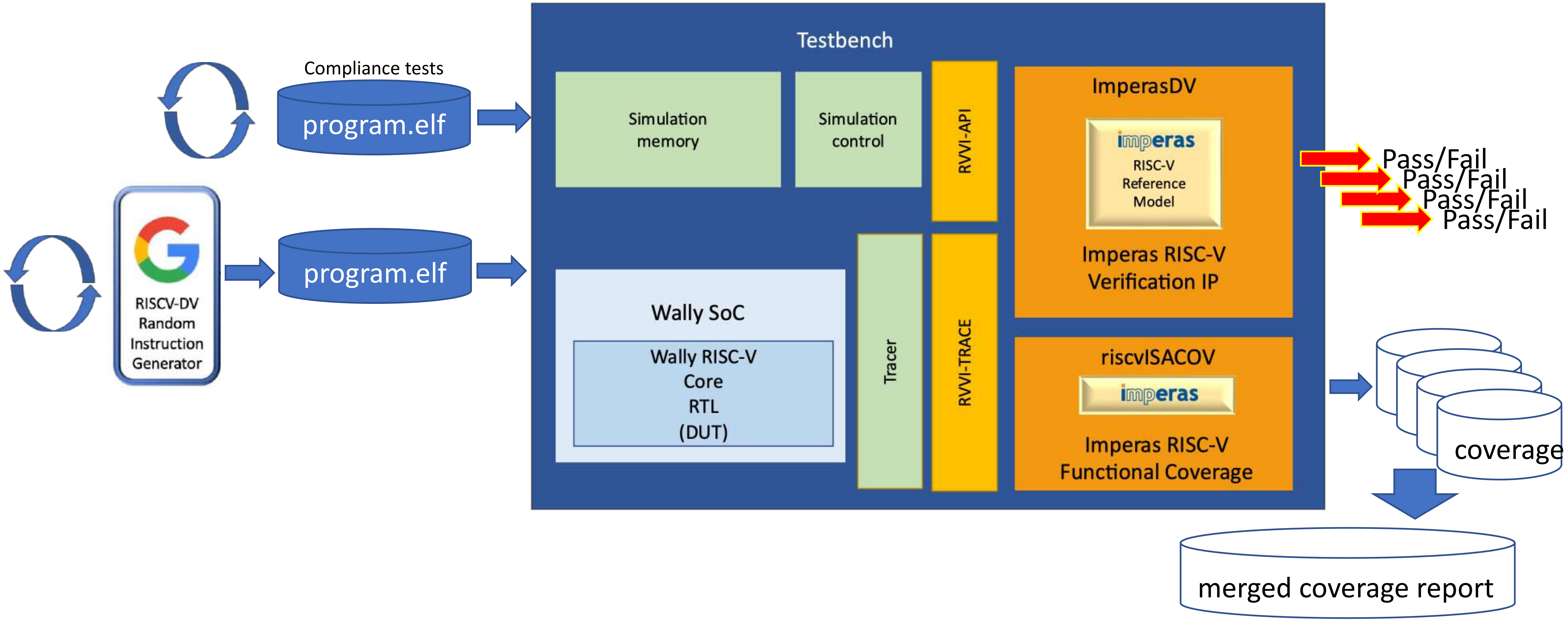
```
simond@lnx6476 wallydemo $ vi cover.ucdb.log
simond@lnx6476 wallydemo $ vi ${IDV}/scripts/cvw/iter-elf.bash
simond@lnx6476 wallydemo $  ${IDV}/scripts/cvw/iter-elf.bash --cover

#
# Running /home/simond/riscv/wallydemo/ImperasDV-OpenHW/scripts/cvw/iter-elf.bash COVERAGE=1
#

running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/mulh-01.elf Test Passed
simond@lnx6476 wallydemo $ vi cover.ucdb.log
simond@lnx6476 wallydemo $ !rm
rm cover.ucdb*
simond@lnx6476 wallydemo $ vi ${IDV}/scripts/cvw/iter-elf.bash
simond@lnx6476 wallydemo $  ${IDV}/scripts/cvw/iter-elf.bash --cover

#
# Running /home/simond/riscv/wallydemo/ImperasDV-OpenHW/scripts/cvw/iter-elf.bash COVERAGE=1
#

running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/REMU-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/DIVU-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/MULHU-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/MULH-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/MUL-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/DIV-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/MULHSU-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/MULW-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/REM-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/REMW-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/REMUW-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/DIVW-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m/M/DIVUW-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/mul-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/divuw-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/rem-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/div-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/mulhsu-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/divw-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/remu-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/remw-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/remuw-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/mulw-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/mulh-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/divu-01.elf Test Passed
running /home/simond/riscv/imperas-riscv-tests/work/rv64i_m-RVI/M/mulhu-01.elf Test Passed
simond@lnx6476 wallydemo $
```

```
Every 2.0s: tail /home/simond/riscv/wallydemo/cover.ucdb.log          lnx6478.impinternal.com: Wed Feb 22 15:2

            bin <neg,pos>                    224        1        -    Covered
            bin <pos,neg>                    244        1        -    Covered
            bin <neg,neg>                     98        1        -    Covered

TOTAL COVERGROUP COVERAGE: 91.82%  COVERGROUP TYPES: 13

Total Coverage By Instance (filtered view): 91.82%

End time: 15:28:16 on Feb 22,2023, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
```

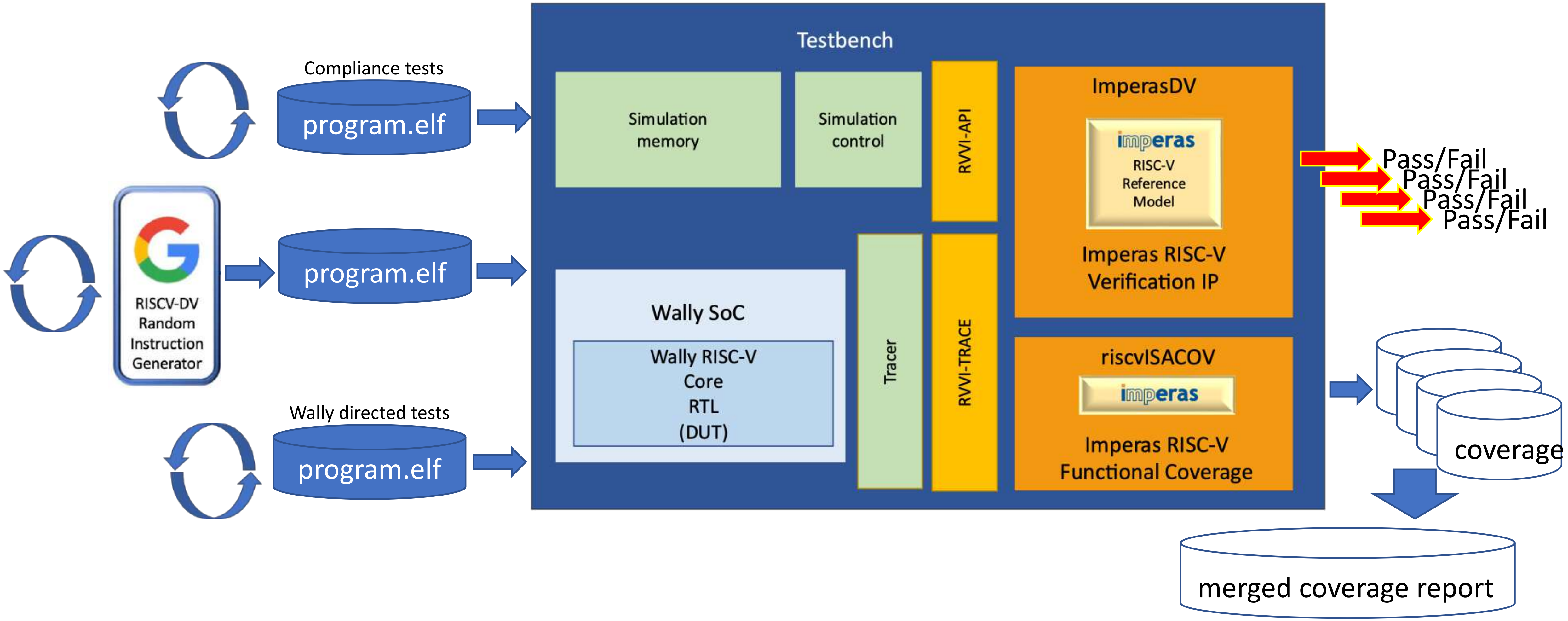
```
x10,0x3333      ",x10=0000000003333000,,,CSRb00(mcycle)=00000000000036c8 CSRb0
# Info 7640: 'refRoot/cpu', 0x00000000800077ac(inst_704): Machine 03333537 lui     x10,0x3333
# Info    MEMX 0x800077ac 0x800077ac 2 3537
# Info    MEMX 0x800077ae 0x800077ae 2 0333
# Info    x10 3333333333333334 -> 0000000003333000
# [NOTE] testbench.idv_trace2log @ 140401: RET,0,7641,800077b0,"3335051b addiw   x10,x10,819    ",x10=0000000003333333,,,CSRb00(mcycle)=00000000000036d3 CSRb0
instret)=0000000000001dd9,
```

imperas

Wally: RVVI, ImperasDV: verification with compliance suites & Google riscv-dv ISG & merged coverage

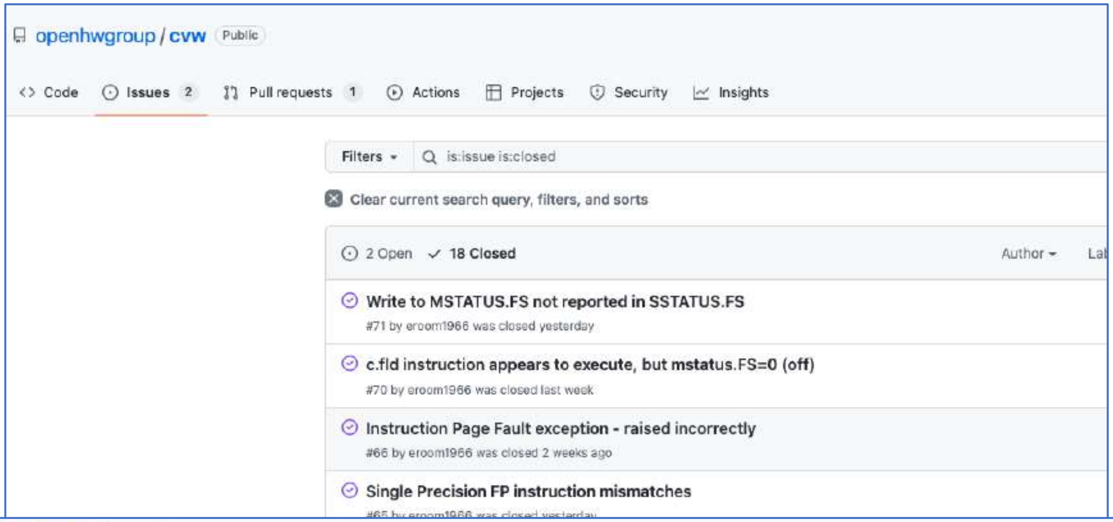


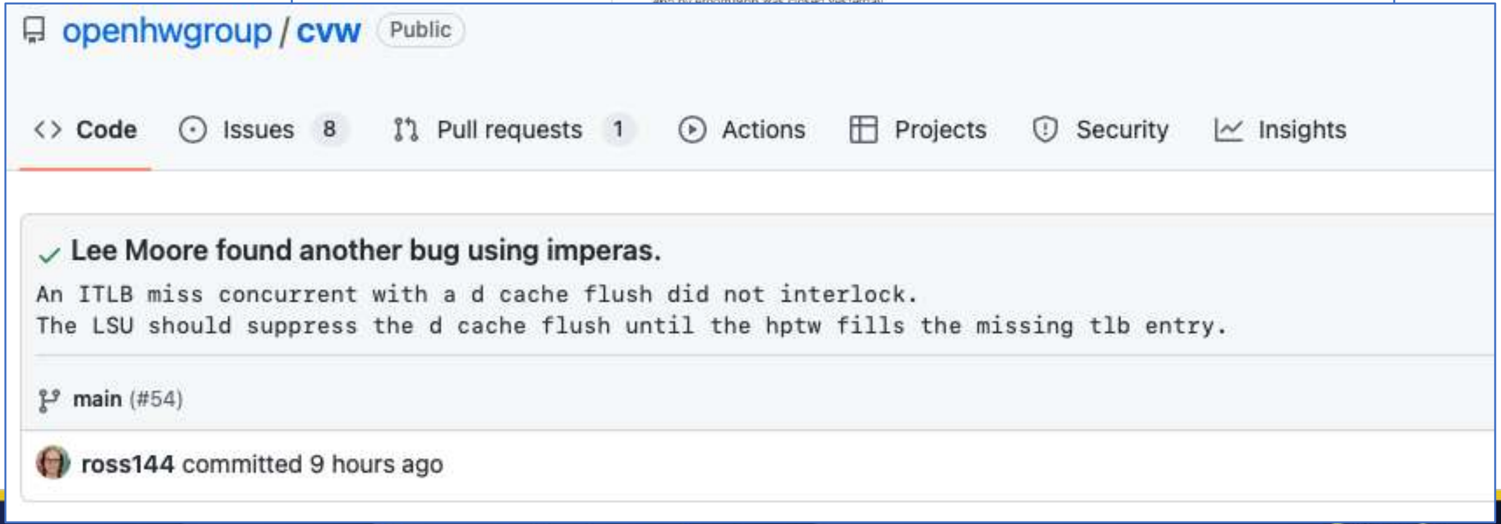
Wally: RVVI, ImperasDV: verification with compliance suites & Google riscv-dv ISG & directed tests & merged coverage



Wally + RVVI + ImperasDV – Status (Feb. 2023)

- RVVI Tracer: 1/2 day of effort
- Testbench: 1/2 day of integration
- 2 days effort resolve tracer/integration
- Results:
    - 20+ bugs found almost immediately
    - With improving functional coverage analysis
- Stimulus: riscv-dv





Agenda

- Introduction to Imperas
- Introduction to RISC-V
- RISC-V processor verification challenges
  - Why is RISC-V processor DV so critical?
- RISC-V processor verification environment components
- RISC-V Verification approaches
- RISC-V Verification standards
- RISC-V Verification IP
- Functional coverage for RISC-V processors
- Verification Case studies
  - OpenHW Group CV32E40X processor
  - Wally RISC-V processor
- **Summary**

Summary

- Processor verification requires unique approaches to ensure the quality of the processor IP
- The verification method chosen will impact the processor’s quality
- Open standards such as RVVI permit efficiency, reuse, and development of RISC-V processor VIP
- The RISC-V ISA is an excellent application for machine-generated functional coverage (e.g. riscvISACOV)
- ImperasDV RISC-V VIP enables a comprehensive processor DV environment that works out of the box

Questions

- Thank you

- Aimee Sutton aimees@imperas.com
- Lee Moore moore@imperas.com
- Simon Davidmann simond@imperas.com