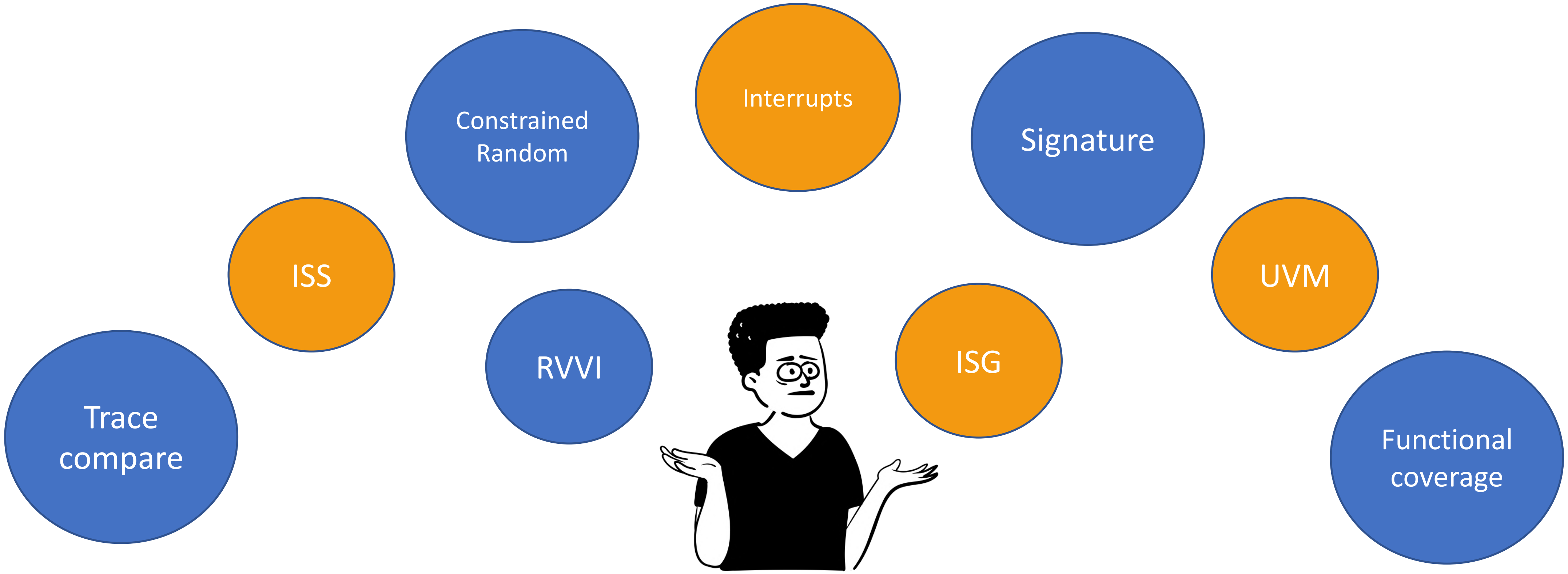
An introduction to RISC-V processor verification techniques

Lee Moore moore@imperas.com
Aimee Sutton aimees@imperas.com

@ImperasSoftware

RISC-V Design Verification



Agenda

- **RISC-V Design Verification challenges**
- RISC-V design verification techniques
- Techniques from ASIC/SoC DV
- How to choose the right technique?

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 the ASIC/SoC verification are insufficient
- New methods are required
  - Take advantage of what has worked in the ASIC world
  - Add to it and adapt for RISC-V

Agenda

- RISC-V Design Verification challenges
- **RISC-V design verification techniques**
  - Post-simulation trace file compare
  - Self-checking tests and Signatures
  - Step-and-compare
  - Step-and-compare with asynchronous events
  - Verification IP using RVVI
  - Demo video
- Techniques from ASIC/SoC DV
- How to choose the right technique?

Post-simulation trace file compare

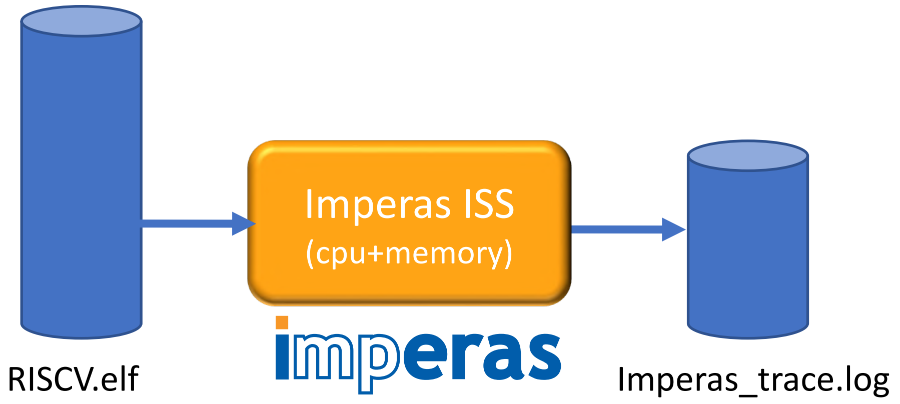
- Components
  - Test programs
    - Can be generated by an ISG – Instruction Stream Generator
  - Instruction Set Simulator (ISS)
  - DUT and Tracer
  - RTL simulator
  - Comparison script

Test programs

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

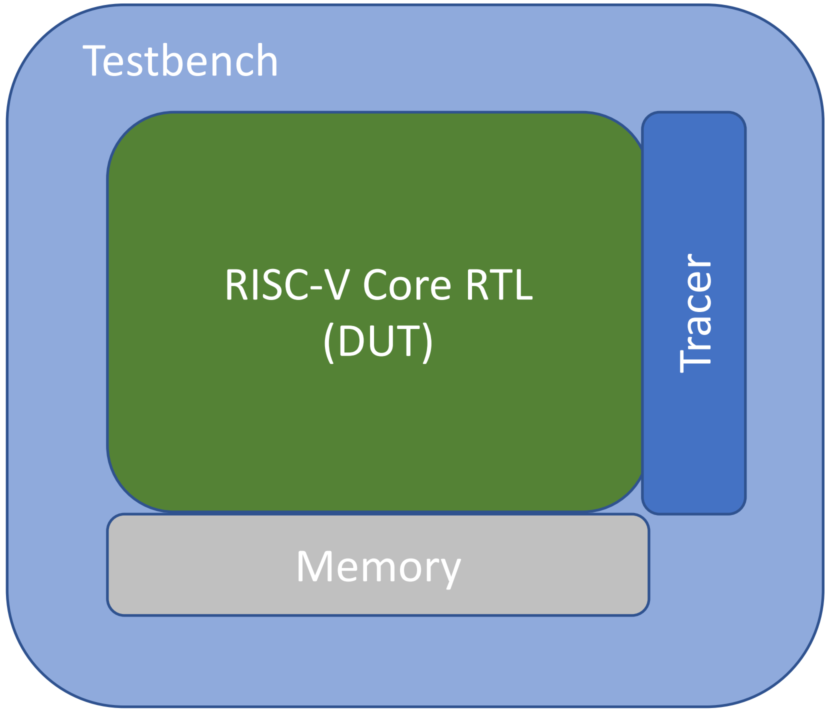
Instruction Set Simulators

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

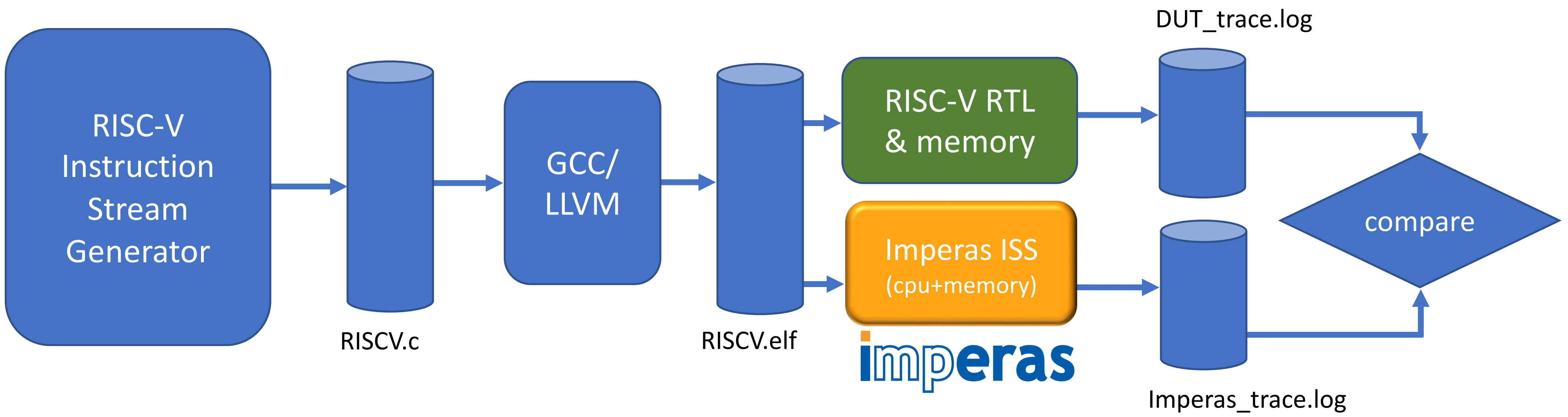


DUT + Tracer

- DUT
  - 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



Trace compare: Process



- Run random generator (ISG) to create tests
- Simulate using ISS; write trace log file
- Simulate using RTL; write trace log file
- Run compare program to see differences / failures

Trace compare: Pros and Cons

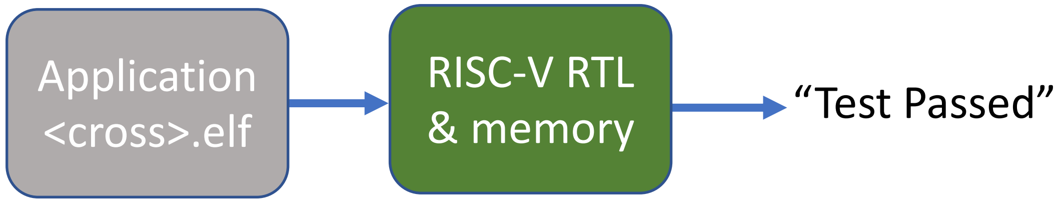
- Pros:
  - Availability of generic RISC-V simulators (e.g. riscvOVPsimPlus from Imperas)
  - Simple to set up and use
- Cons:
  - Incompatible trace formats
  - Must run RTL simulation to the end
  - Cannot debug live
  - Difficult to verify asynchronous events (e.g. interrupts, debug requests)
  - Not a comprehensive DV strategy

Agenda

- Background: RISC-V Design Verification challenges
- **RISC-V design verification techniques**
  - Post-simulation trace file compare
  - Self-checking tests and Signatures
  - Step-and-compare
  - Step-and-compare with asynchronous events
  - Verification IP using RVVI
  - Demo video
- Techniques from ASIC/SoC DV
- How to choose the right technique?

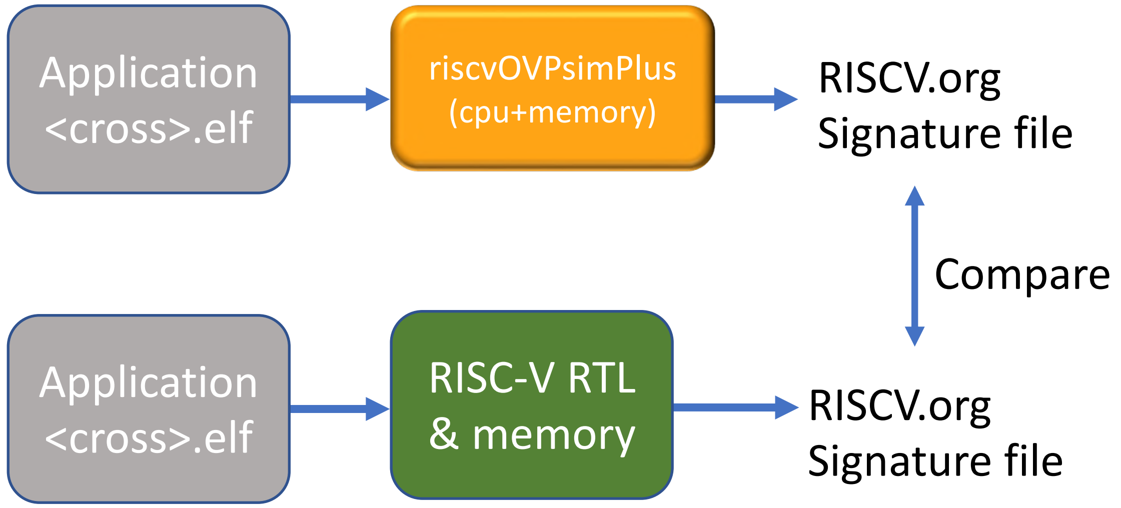
Self-checking tests

- 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



Signature comparison

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



Self-checking tests & Signatures: 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 compliance tests freely available
- Cons:
  - Directed tests cover a subset of processor functionality
  - Not a complete DV strategy

Agenda

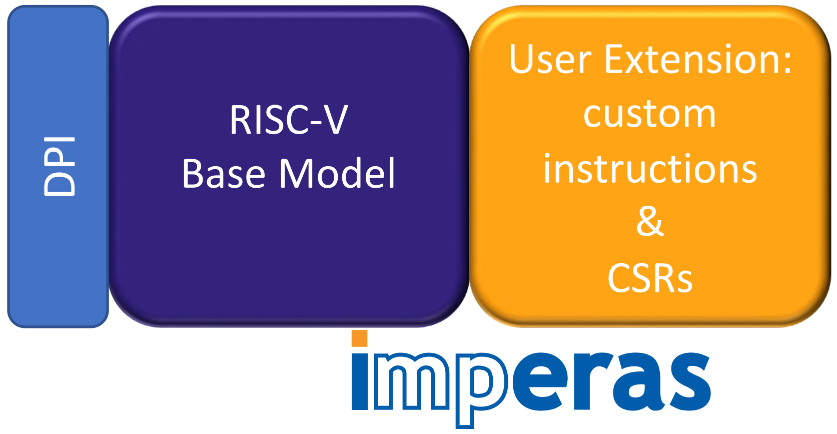
- RISC-V Design Verification challenges
- **RISC-V design verification techniques**
  - Post-simulation trace file compare
  - Self-checking tests and Signatures
  - Step-and-compare
  - Step-and-compare with asynchronous events
  - Verification IP using RVVI
  - Demo video
- Techniques from ASIC/SoC DV
- How to choose the right technique?

Step and compare

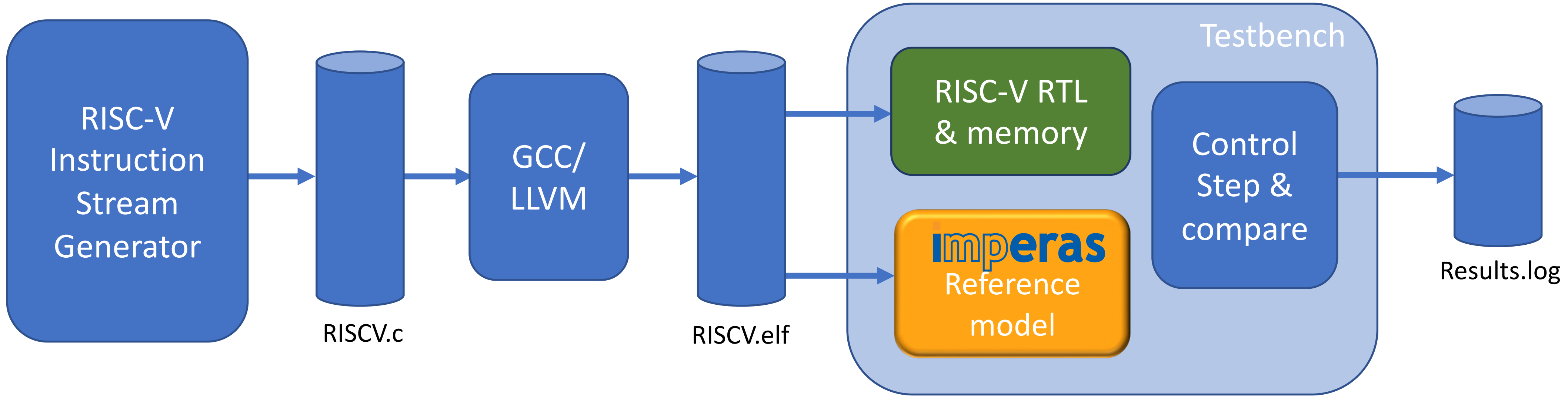
- Components
  - Test programs (can be generated by an ISG)
  - **Processor reference model**
  - DUT and tracer
  - RTL simulator
  - **Step-and-compare logic**

Processor reference model

- Reference model requirements:
  - Configurable to select RISC-V ISA extensions
  - Ability to add customizations (e.g. instructions, CSRs)
  - Can run in lock-step with the RTL simulator (co-sim)
  - Ability to “step” reference model at significant events (retire, trap)
  - Functions to query state of model for comparison



Step and compare: Process



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

Step-and-compare: 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

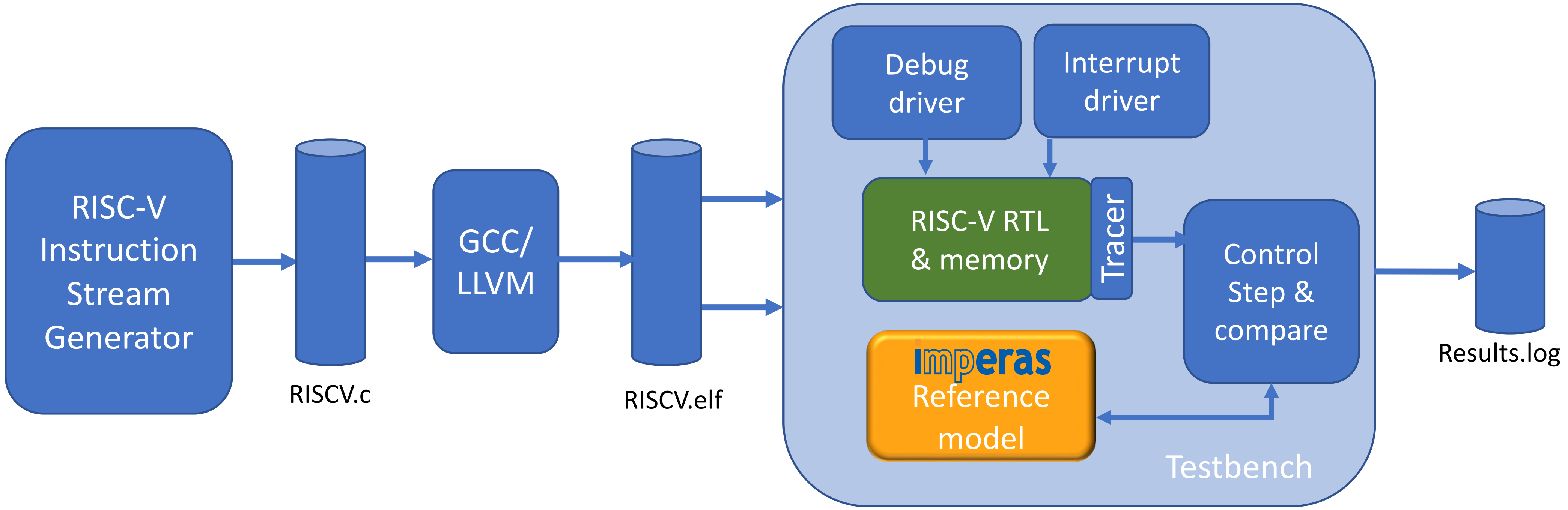
Agenda

- Background: RISC-V Design Verification challenges
- **RISC-V design verification techniques**
  - Post-simulation trace file compare
  - Self-checking tests and Signatures
  - Step-and-compare
  - Step-and-compare with asynchronous events
  - Verification IP using RVVI
  - Demo video
- Techniques from ASIC/SoC DV
- How to choose the right technique?

Step and compare + Async

- Components
  - Test programs (can be generated by an ISG)
  - Processor reference model
  - DUT and tracer
  - RTL simulator
  - **Asynchronous event drivers** (e.g. UVM agents)
  - **Step-and-compare logic +**

Step and compare + Async: Process



- Asynchronous events are driven into the DUT
- Step and compare logic informs reference model about async events

Async step and compare: Pros and Cons

- Pros:
  - All the benefits of step-and-compare
  - Responds to asynchronous events
- Cons:
  - Step-and-compare logic can be fragile and error prone
  - Implementation of async event handling is not reusable
  - Async events not connected to the reference - can conceal bugs
  - Significant effort to implement and maintain

Agenda

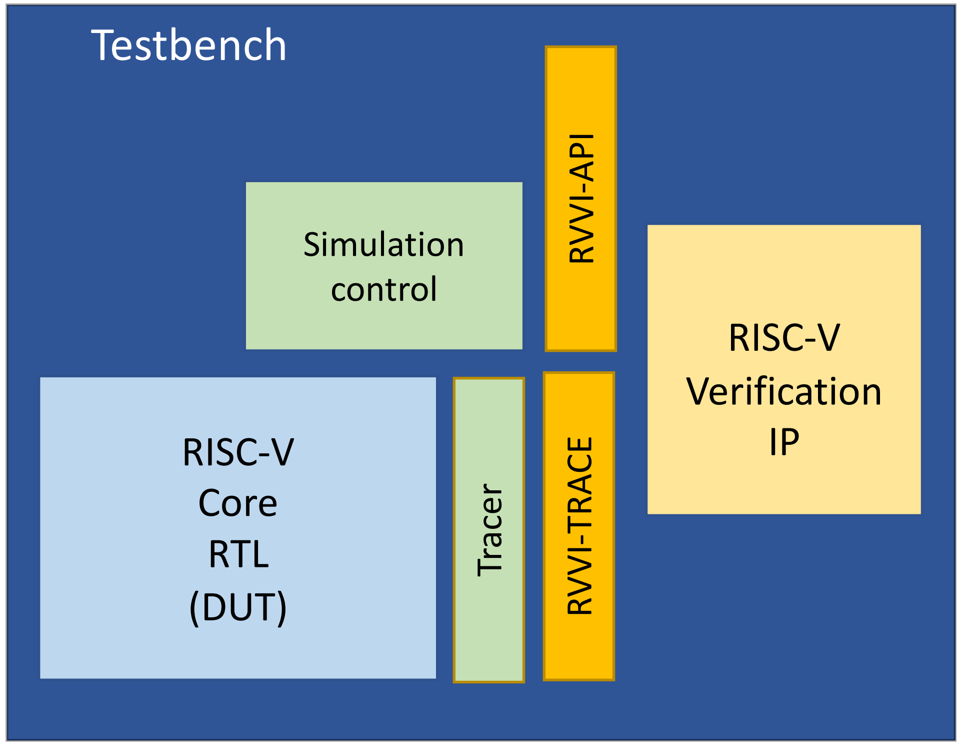
- Background: RISC-V Design Verification challenges
- **RISC-V design verification techniques**
  - Post-simulation trace file compare
  - Self-checking tests and Signatures
  - Step-and-compare
  - Step-and-compare with asynchronous events
  - Verification IP using RVVI
  - Demo video
- Techniques from ASIC/SoC DV
- How to choose the right technique?

RISC-V Processor VIP

- Requirements:
  - Configurable, extendable RISC-V processor reference model
  - Standard interface to receive tracer data
  - Standard way to receive asynchronous events
  - 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

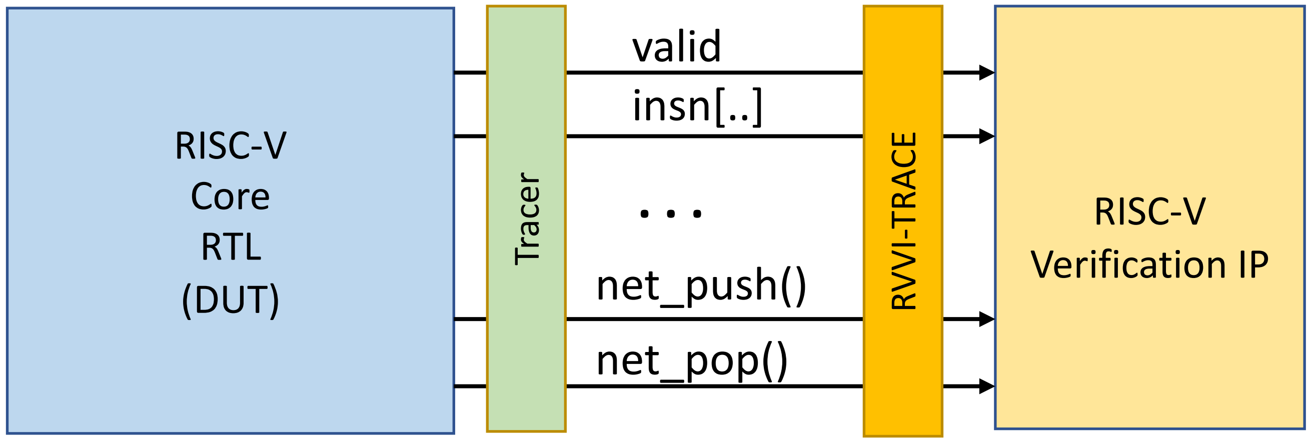
Standard interface: RVVI

- RVVI = RISC-V Verification Interface
  - https://github.com/riscv-verification/RVVI
- Work has evolved over 2 years
  - Imperas, EM Micro, SiLabs, OpenHW
- Standardize communication between testbench and RISC-V VIP
- Two parts:
  - **RVVI-TRACE**: signal level interface to RISC-V VIP
  - **RVVI-API**: function level interface to RISC-V VIP



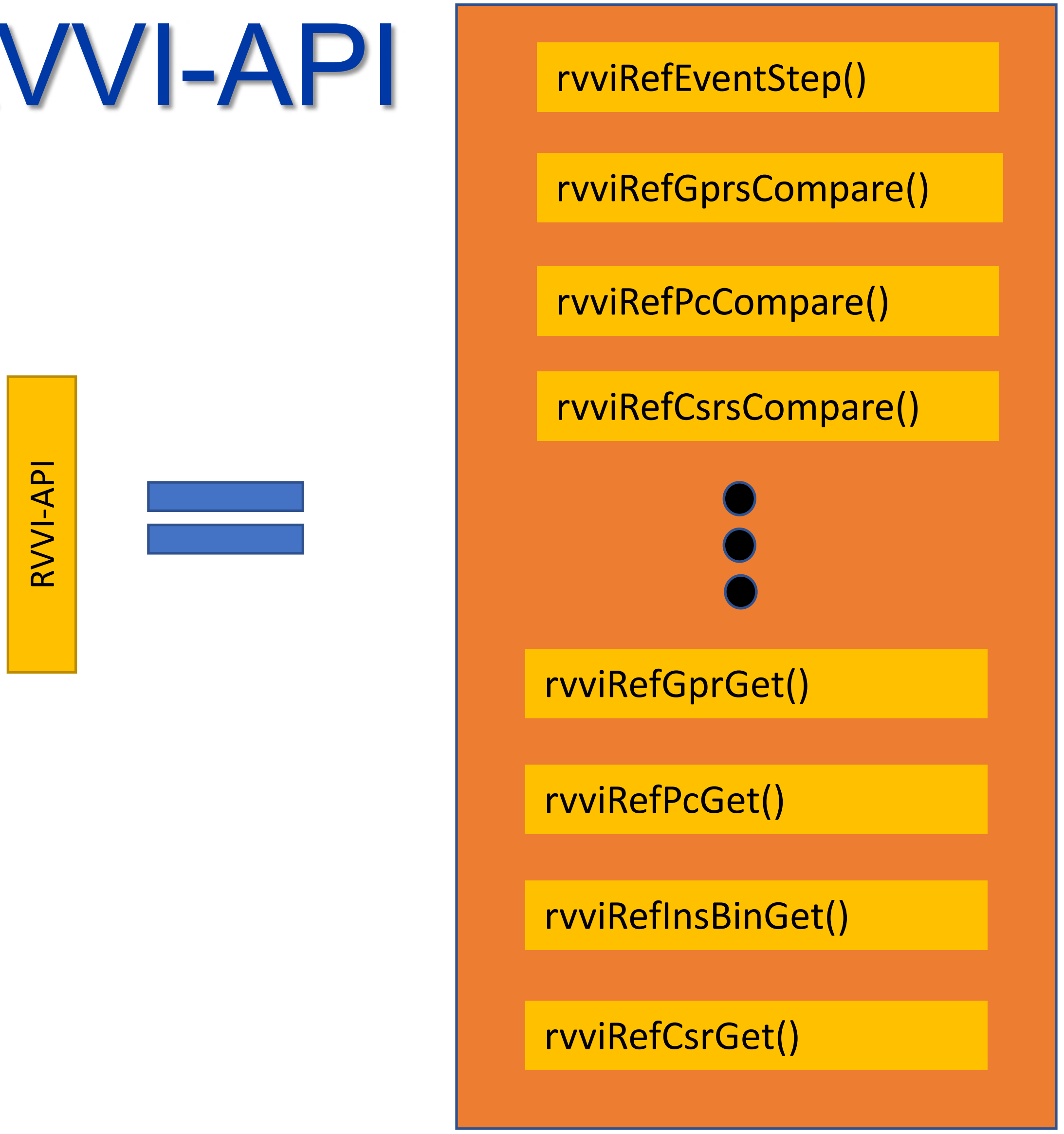
RVVI-TRACE

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



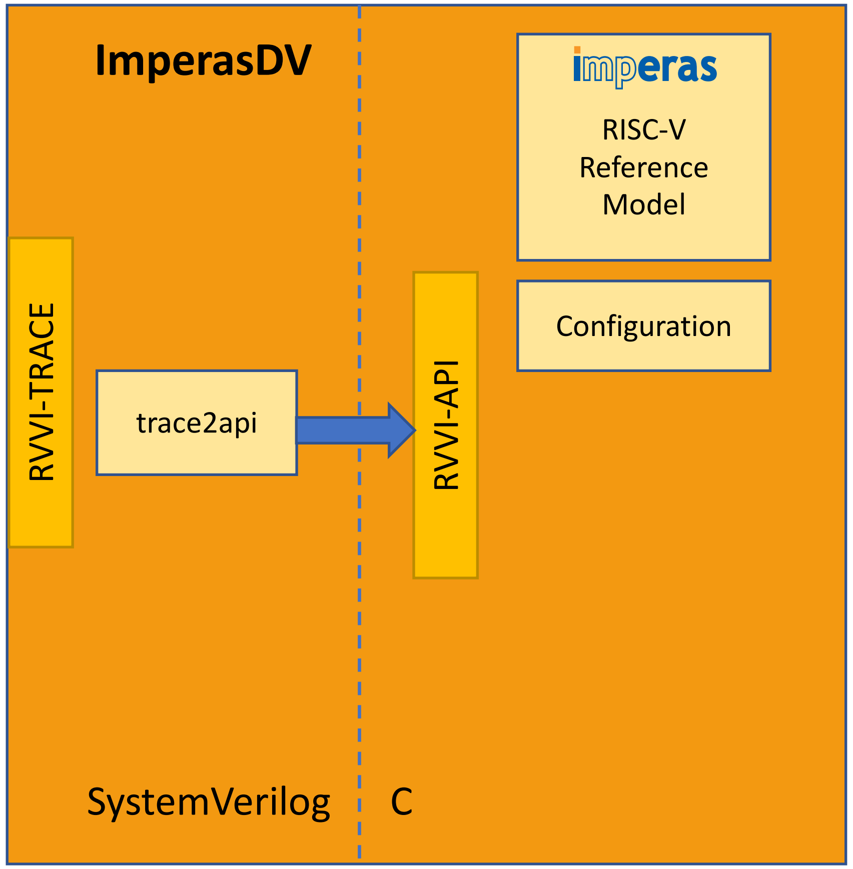
- https://github.com/riscv-verification/RVVI/tree/main/RVVI-VLG

RVVI-API

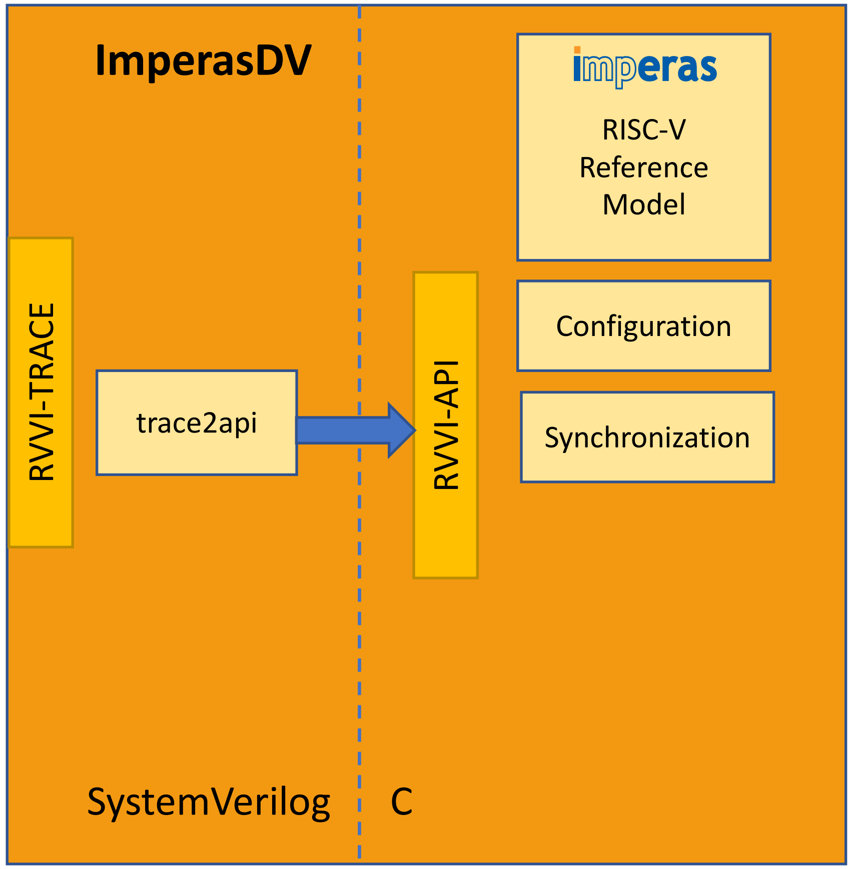


- Standard functions that RISC-V processor VIPs need to implement
- Supports a step-and-compare methodology
- C and SystemVerilog versions available
- https://github.com/riscv-verification/RVVI/blob/main/include/host/rvvi/rvvi-api.h

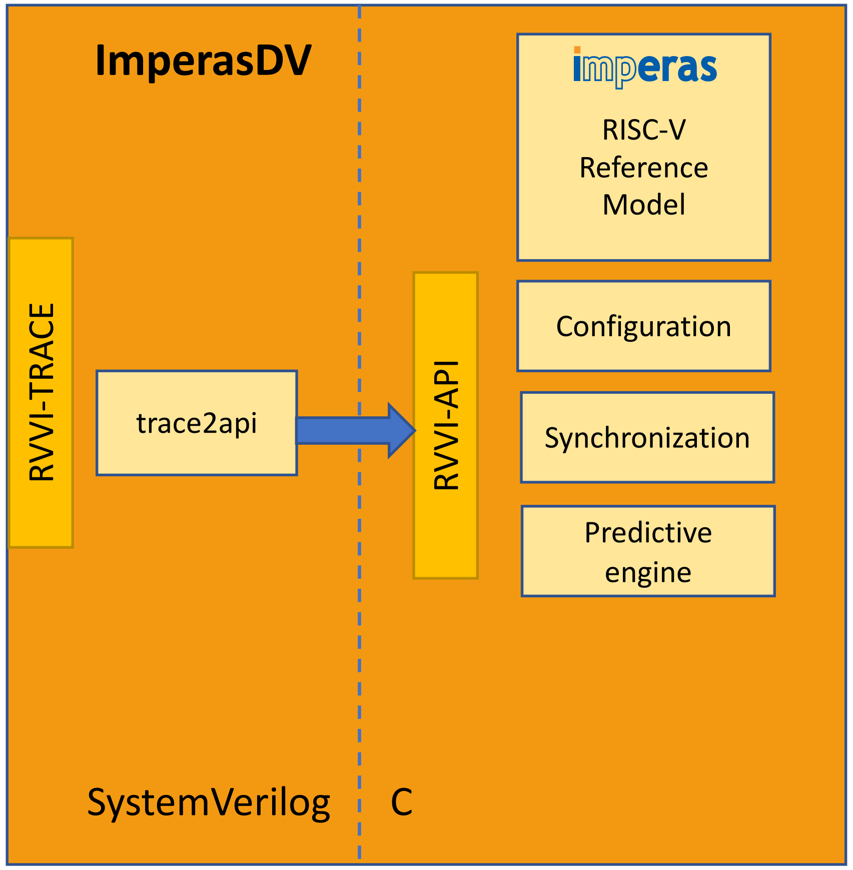
ImperasDV components
Configurable reference



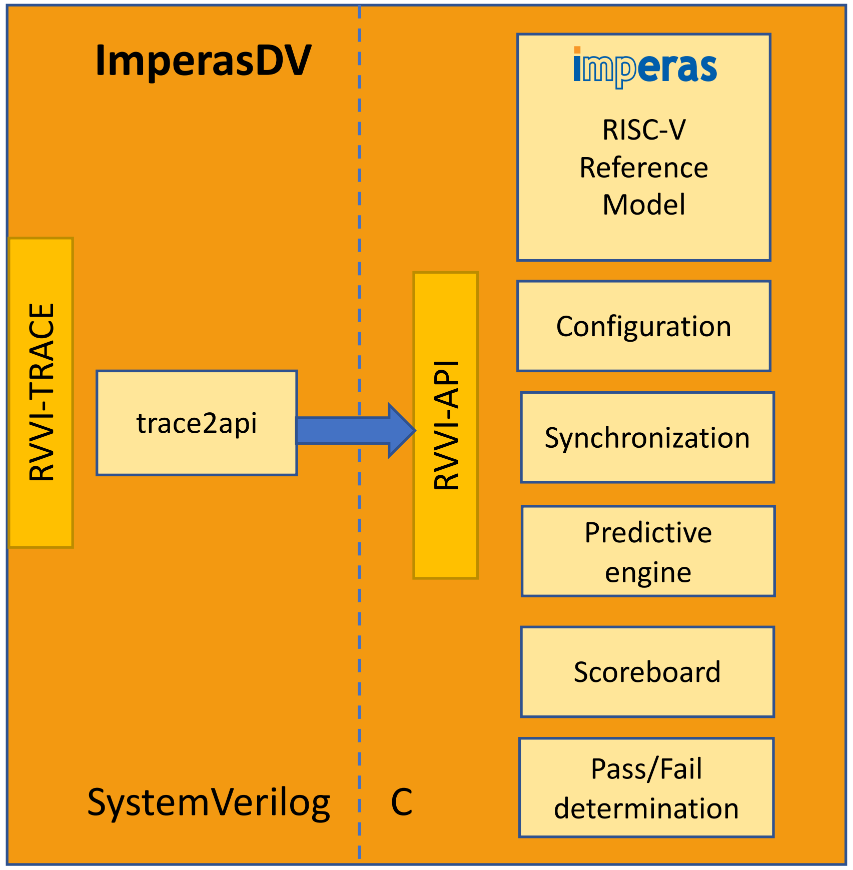
ImperasDV components
Control and Introspection



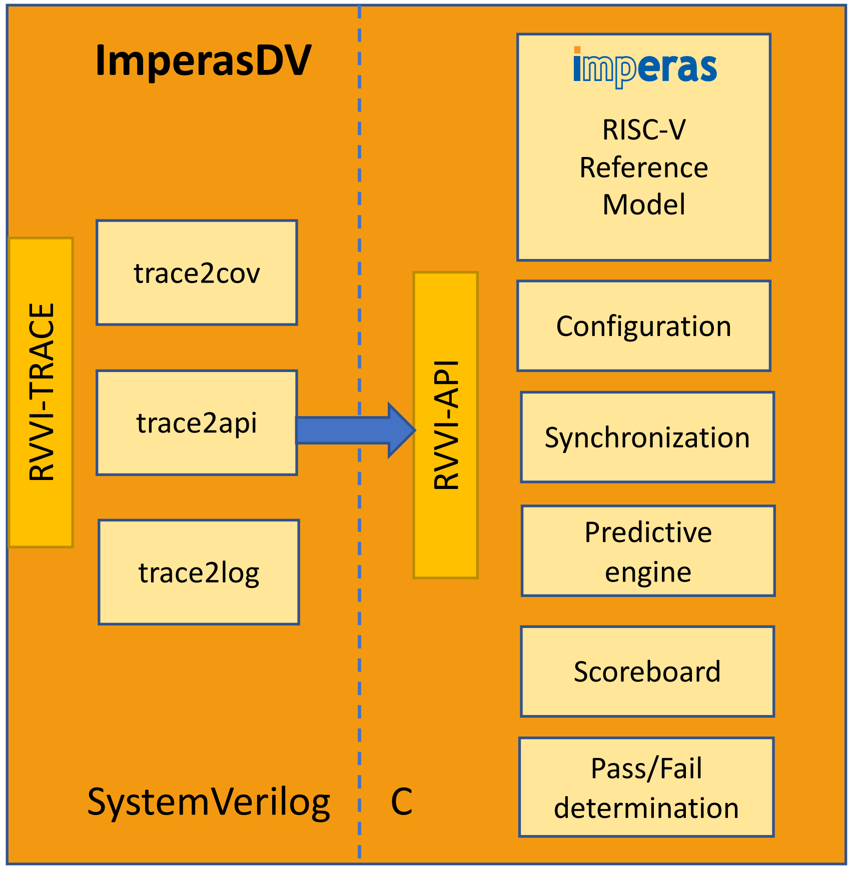
ImperasDV components
Asynchronous events



ImperasDV compoents
Comparison

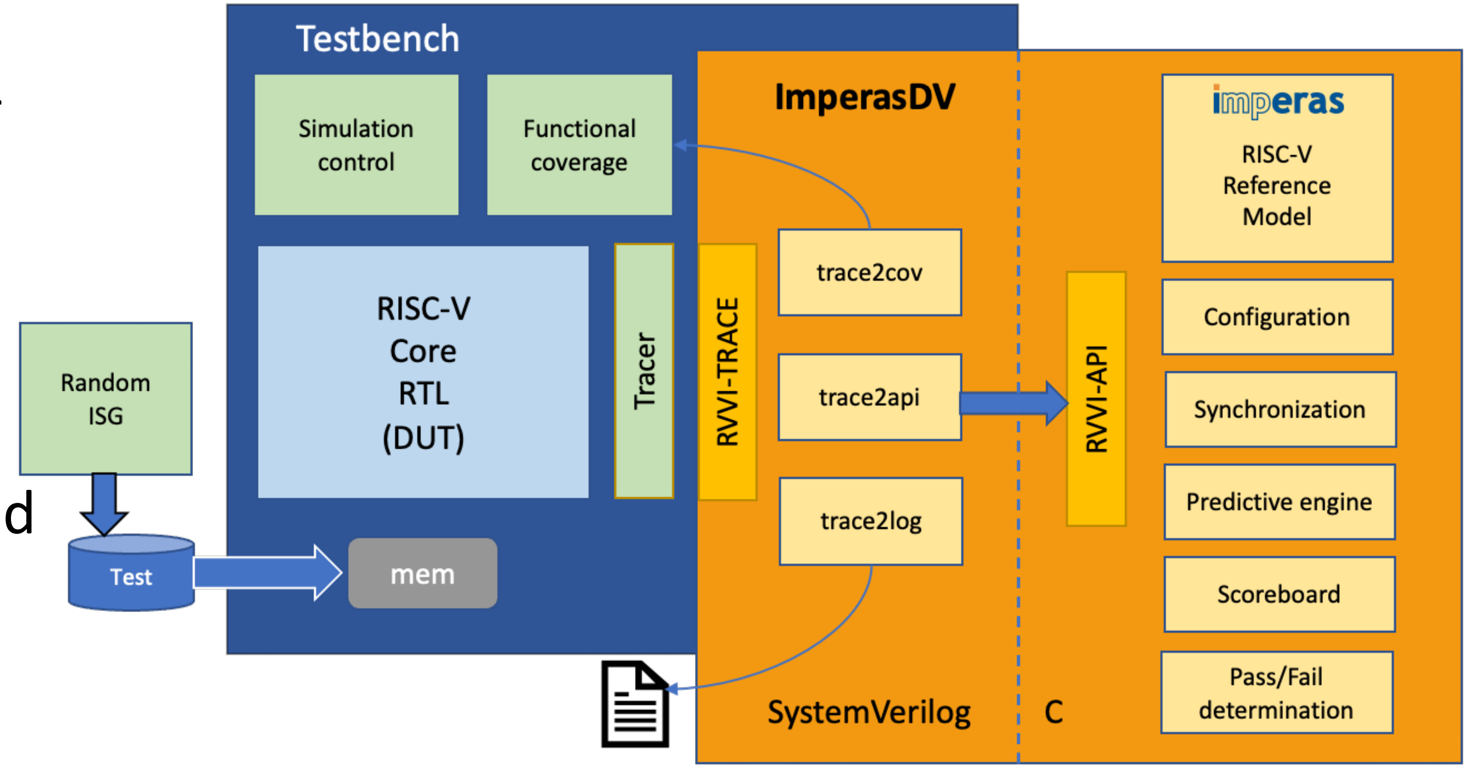


ImperasDV components
Coverage and logging



Verification IP + 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



Verification IP using RVVI

- Pros:
  - Errors are flagged immediately
  - Finds synchronous and asynchronous bugs
  - Checking is done for you
  - 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

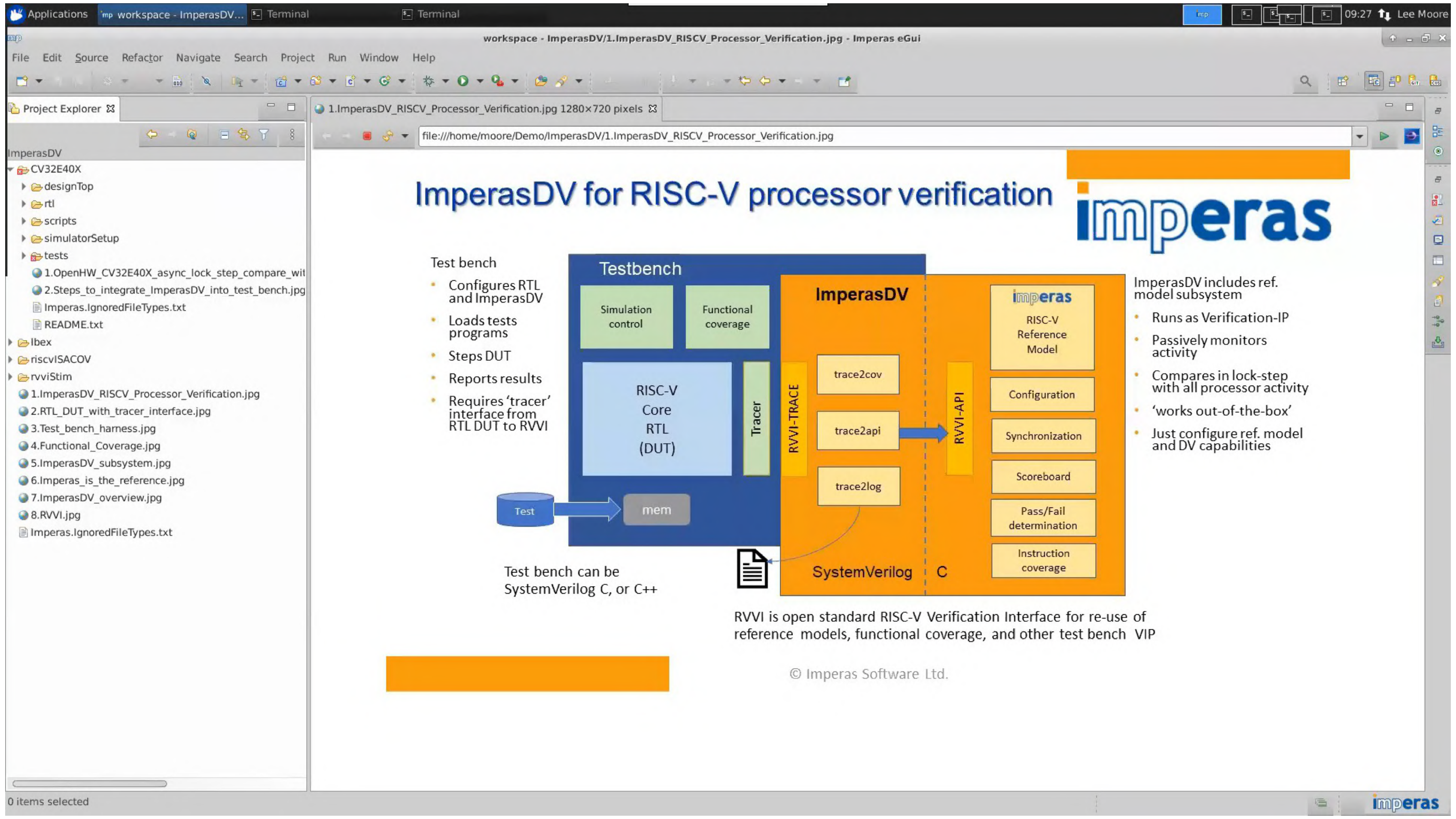
- Background: RISC-V Design Verification challenges
- **RISC-V design verification techniques**
  - Post-simulation trace file compare
  - Self-checking tests
  - Step-and-compare
  - Step-and-compare with asynchronous events
  - Verification IP using RVVI
  - Demo video
- Techniques from ASIC/SoC DV
- How to choose the right technique?

Demonstration

- DUT: OpenHW Group CV32E40X RISC-V processor
  - Simulation: passing test
  - Simulation: failing test
  - Simulation: asynchronous event bug

VIDEO: Passing test

- 1:22



Applications
workspace - ImperasDV...
Terminal
Terminal
09:27
Lee Moore
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
0 items selected

```
ImperasDV
  CV32E40X
    designTop
    rtl
    scripts
    simulatorSetup
    tests
      1.OpenHW_CV32E40X_async_lock_step_compare_wit
      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
```

ImperasDV for RISC-V processor verification

Test bench
- Configures RTL and ImperasDV
- Loads tests programs
- Steps DUT
- Reports results
- Requires 'tracer' interface from RTL DUT to RVVI

Test bench can be SystemVerilog C, or C++

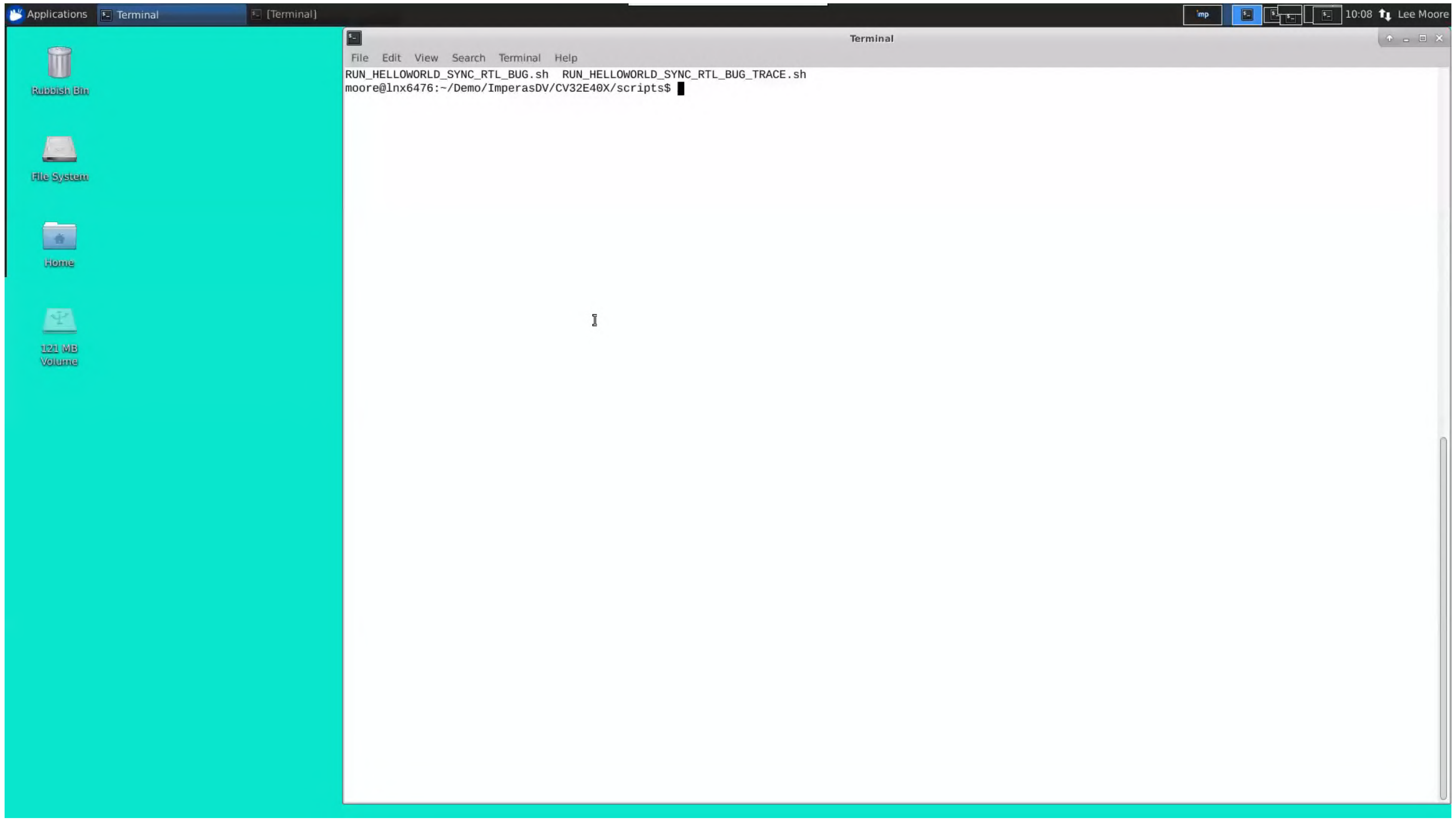
ImperasDV includes ref. model subsystem
- Runs as Verification-IP
- Passively monitors activity
- Compares in lock-step with all processor activity
- 'works out-of-the-box'
- Just configure ref. model and DV capabilities

RVVI is open standard RISC-V Verification Interface for re-use of reference models, functional coverage, and other test bench VIP

© Imperas Software Ltd.

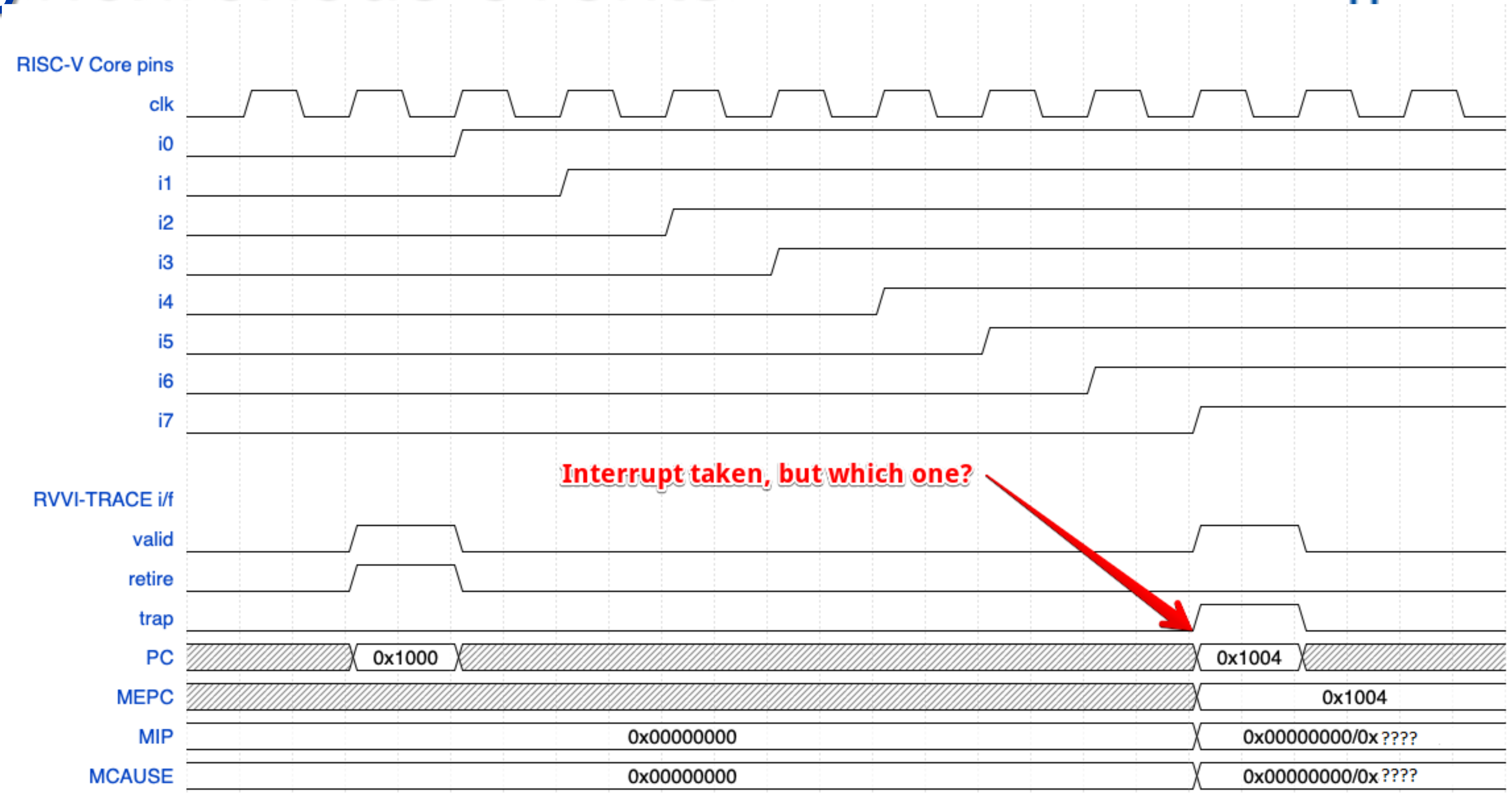
VIDEO: Failing test

- 2:59



```
RUN_HELLOWORLD_SYNC_RTL_BUG.sh  RUN_HELLOWORLD_SYNC_RTL_BUG_TRACE.sh
moore@lnx6476:~/Demo/ImperasDV/CV32E40X/scripts$
```

Asynchronous events



VIDEO: Asynchronous

- 4:38

Agenda

- Background: RISC-V Design Verification challenges
- RISC-V design verification techniques
- **Techniques from ASIC/SoC DV**
  - Verification planning
  - Functional coverage
  - Assertions
- How to choose the right technique?

Verification planning

- Start with the end in mind. What are your verification goals?
  - Capture them in a plan
- How you will measure that they have been met?
  - Directed test, coverpoint, assertion?
  - Capture this in the plan too
- Metric-driven verification is popular
  - Common metrics: code coverage, functional coverage, all tests passing, no new bugs found for a period of time
- Sample open source verification plans:
  - https://github.com/openhwgroup/core-v-verif/tree/master/cv32e40p/docs/VerifPlans

Functional coverage

- Define goals for verification
- Measure that goals are achieved
- Measure the effectiveness of constrained-random stimulus
- Requires EDA tools to capture, merge, display coverage results
- Requires many simulations to achieve coverage closure
- Industry-standard best practice for ASIC/SoC

RISC-V Functional Coverage

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

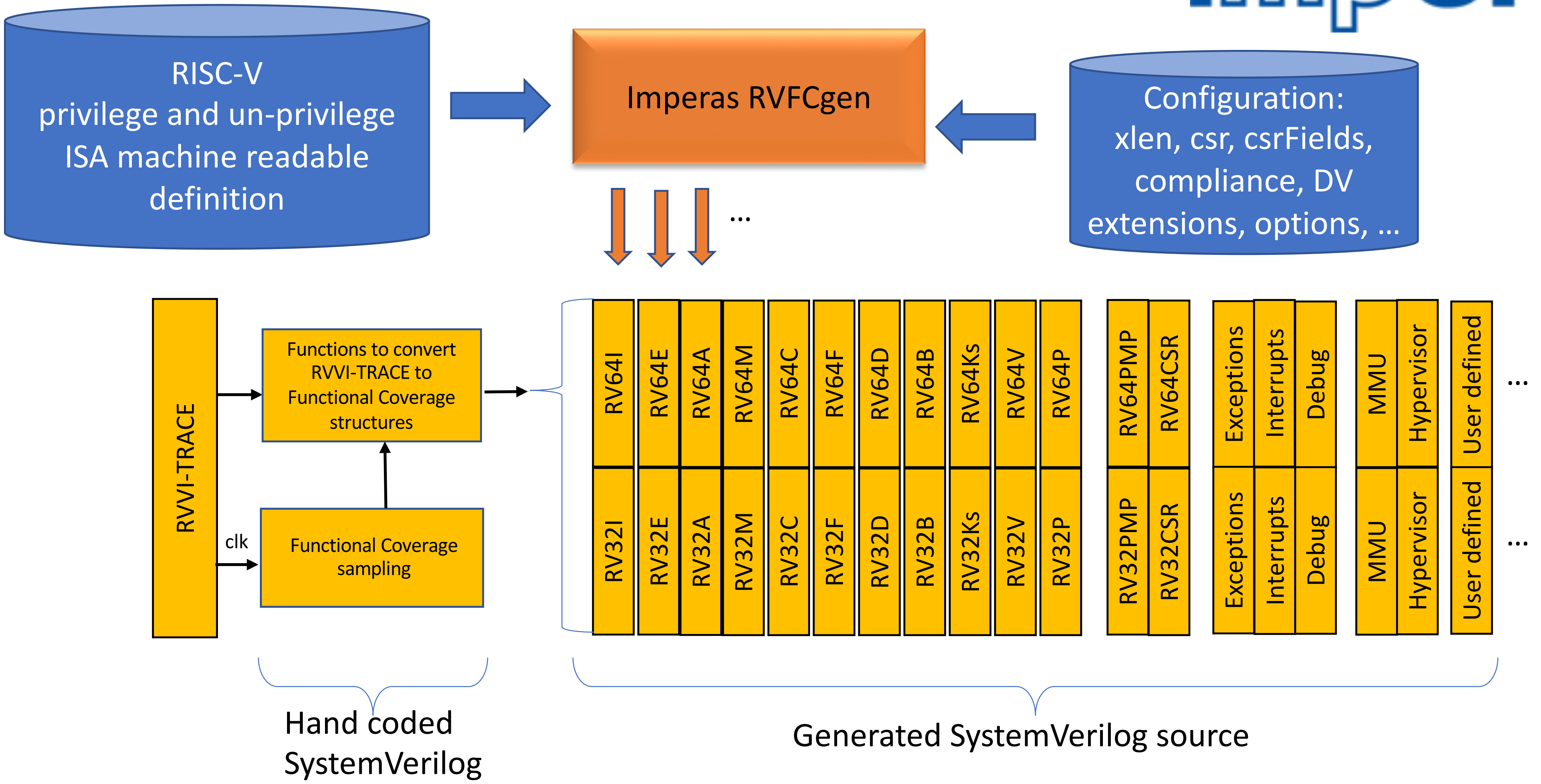
- Standard ISA architectural features
  - unpriv. ISA items: mainly instructions, their operands, their values

  => 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)

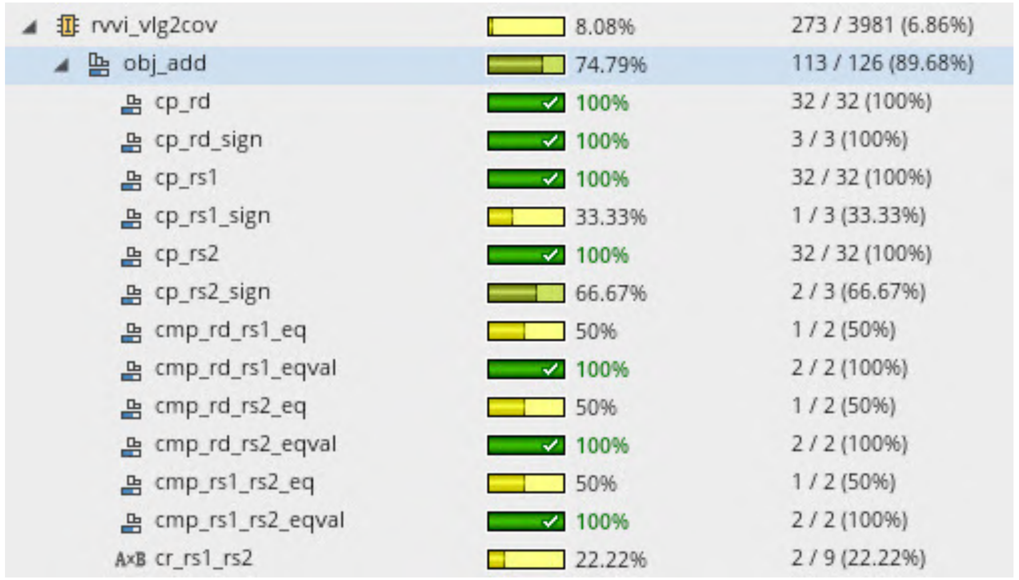
- There are many different instructions in the RV64 extensions:
  - Integer: 56, Maths: 13, Compressed: 30, FP-Single: 30, FP-Double: 32
  - Vector: 356, Bitmanip: 47 Krypto-scalar: 85
  - P-DSP: 318
  - For RV64 that is 967 instructions…
- Each instruction needs SystemVerilog covergroups and coverpoints
  - 10-40 lines of SystemVerilog for each instruction
- 10,000-40,000++ lines of code to be written
  - Not design or core specific

Generating functional coverage source files from machine readable ISA definition



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



rvvi_vlg2cov	8.08%	273 / 3981 (6.86%)
obj_add	74.79%	113 / 126 (89.68%)
cp_rd	✓ 100%	32 / 32 (100%)
cp_rd_sign	✓ 100%	3 / 3 (100%)
cp_rs1	✓ 100%	32 / 32 (100%)
cp_rs1_sign	33.33%	1 / 3 (33.33%)
cp_rs2	✓ 100%	32 / 32 (100%)
cp_rs2_sign	66.67%	2 / 3 (66.67%)
cmp_rd_rs1_eq	50%	1 / 2 (50%)
cmp_rd_rs1_eqval	✓ 100%	2 / 2 (100%)
cmp_rd_rs2_eq	50%	1 / 2 (50%)
cmp_rd_rs2_eqval	✓ 100%	2 / 2 (100%)
cmp_rs1_rs2_eq	50%	1 / 2 (50%)
cmp_rs1_rs2_eqval	✓ 100%	2 / 2 (100%)
cr_rs1_rs2	22.22%	2 / 9 (22.22%)

Assertions

- Popular languages: SVA (SystemVerilog), PSL
- Concurrent assertions
  - Rules to check behaviour over time
  - Can be used to verify micro-architectural details
  - Can be written by RTL designers
  - Can be reused in formal verification
- “Cover” properties contribute to functional coverage

Agenda

- RISC-V Design Verification challenges
- RISC-V design verification techniques
- Techniques from ASIC/SoC DV
- **How to choose the right technique?**
    - DUT considerations
    - Technology questions
    - Hybrid methodologies

DUT considerations affecting verification method

- Is this a new design?
- Have you started from a commercial IP core?
  - What is the magnitude of your change?
  - What does your IP vendor recommend for verification?
- Are you using or modifying an open-source core?
  - Can you find evidence of verification done to date?
  - Can you reuse or build upon existing DV infrastructure?
- What is your goal?
  - Research project, sell/provide IP, tape out
- What is your requirement for reuse?
  - Across teams, future projects, etc.

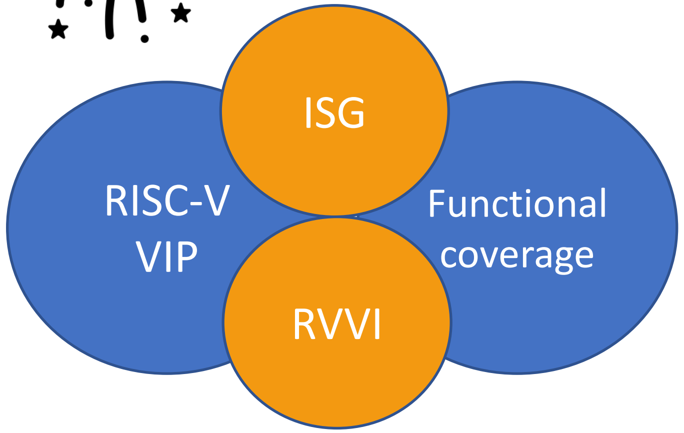
Technology considerations affecting verification method

- Verification language
  - SystemVerilog, VHDL, C/C++, Python?
  - Some methodologies only available in a certain language
    - E.g. functional coverage (SV), UVM (SV, Python), OSVVM (VHDL)
- UVM
  - Widely adopted and industry proven
  - Good body of knowledge / online resources available
  - Strengths: virtual sequences, configuration database, messaging
  - Weaknesses: limited choice of RTL simulation tools, heavy-weight solution
- Build it yourself, use open-source, or use Verification IP?
  - Cost of VIP licenses vs cost of time and effort to build

Hybrid methodologies

- Post-simulation trace file compare + VIP
  - Use trace file compare for ISA / unprivileged tests
  - Use verification IP for complex scenarios:
    - Sync and Async exceptions
    - Corner cases
  - Make sure to combine functional coverage results
- Pros: can save on license costs
- Cons: effort required to build, maintain, and co-ordinate two separate verification environments

Thank you



- Any questions?

- Lee Moore (moore@imperas.com)
- Aimee Sutton (aimees@imperas.com)

imperas

RISC-V SUMMIT