2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

The Evolution of RISC-V Processor Verification

Aimee Sutton and Lee Moore, Imperas Software

Mike Thompson, OpenHW Group

imperas

OPENHW GROUP® PROVEN PROCESSOR IP

accellera SYSTEMS INITIATIVE

Agenda

- Introduction to the OpenHW Group
- The first generation CORE-V-VERIF environment
- The second generation CORE-V-VERIF environment
- RISC-V Verification Interface (RVVI)
- The third generation CORE-V-VERIF environment
- Asynchronous event verification
- Areas for future work

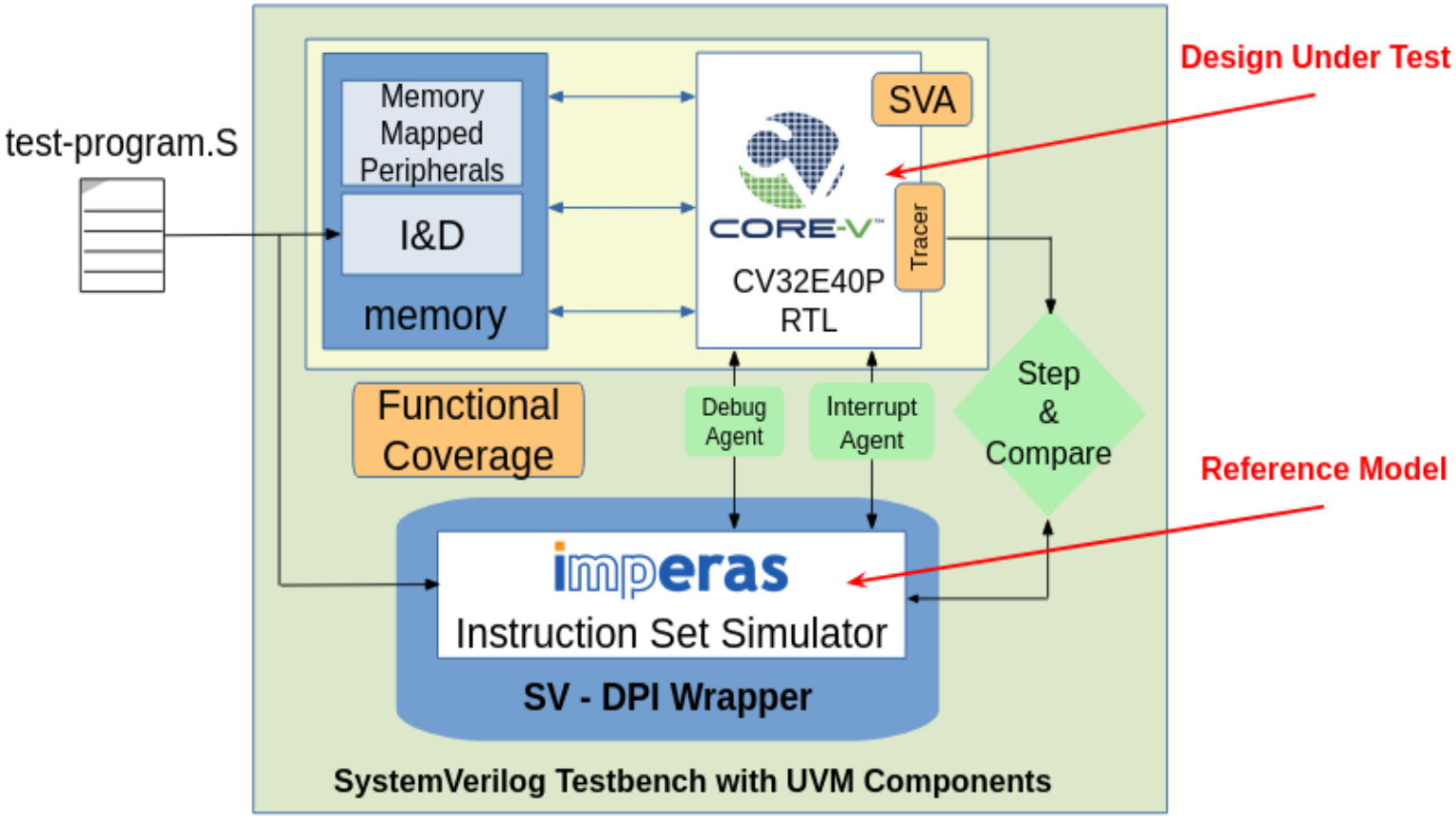
OpenHW Group

- A not-for-profit, global organization registered in Canada and Europe.
  - Almost 100 Members and Partners from all regions of the world.
- Members (corporate & academic) collaborate in the development of open-source cores, related IP, tools and SW.
- Primary focus is the CORE-V Family of open-source RISC-V processors.

CORE-V-VERIF

- A UVM environment for RISC-V processor verification
- Open source and available on GitHub:
  - https://github.com/openhwgroup/core-v-verif
- Developed by the OpenHW Group's Verification Task Group (VTG)
  - Contributions from Silicon Labs, Imperas, EM Microelectronic
  - On-going work by Silicon Labs, Imperas, Thales, Dolphin, NXP, Intrinsix
- Created in 2019; evolving and improving ever since
- In use at many academic and commercial organizations today

First generation CORE-V-VERIF



First generation: operation

- DUT and reference model execute the same program
- As each instruction retires, state variables are compared
  - mismatches reported as errors
  - know as the “Step-and-compare” method
- External interrupts and debug requests sent to both DUT and reference model
  - known as “asynchronous events”

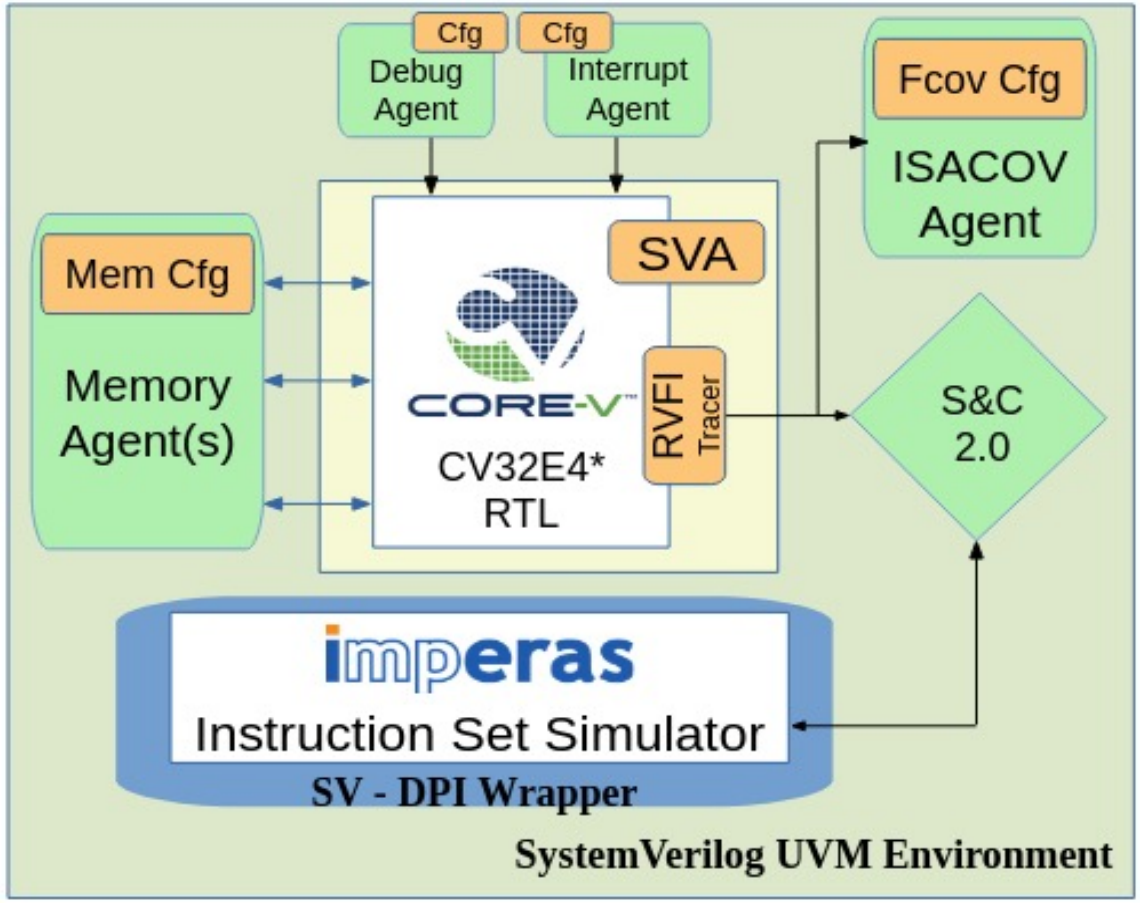
First generation: implementation

- Imperas OVPSim ISS used as reference model
  - saves testbench development effort
  - model operates at a different abstraction level than DUT
  - difficult to handle instructions with side effects
- Tracer was an ad-hoc behavioral model exposing internal state
  - GPR, PC, and CSR values

First generation: challenges

- ISS always receives async events (interrupts, debug requests) at the start of an instruction
  - RTL will observe these at a different time (before, during, or after the same instruction)
  - Result: false-negative comparisons
- Difficult to handle instructions with “side effects”
  - each async event required specific code to model these side effects
  - Result: testbench is prone to bugs and difficult to maintain
- Tracer and step-and-compare logic required frequent updates

Second generation CORE-V-VERIF



Second generation: Improvements

- Ad-hoc tracer from first generation replaced
  - new tracer extends the RISC-V Formal Interface (CORE-V RVFI)
- Async events are connected to the core RTL but not to the ISS
  - Step-and-compare logic simplified (version 2.0)

CORE-V Tracer

- Derived from the RISC-V Formal Interface (RVFI)
- Enhanced for verification
  - Meets the needs of step-and-compare
- Unambiguously identifies instruction retirement
- Unambiguously reports synchronous traps and asynchronous exceptions (interrupts and debug requests)
- Exposes the state of the core (PC, CSRs, GPRs)

Asynchronous events with CORE-V Tracer

- Reference model is not directly connected to asynchronous events
- Tracer monitors and reports these events; informs the reference model to interrupt normal program flow
- Example: debug mode request
  - signals ***rvfi_dbg*** and ***rvfi_dbg_mode*** valid when instruction is retired
  - ***rvfi_dbg*** contains debug cause
  - clear rules specifying how debug entry is identified

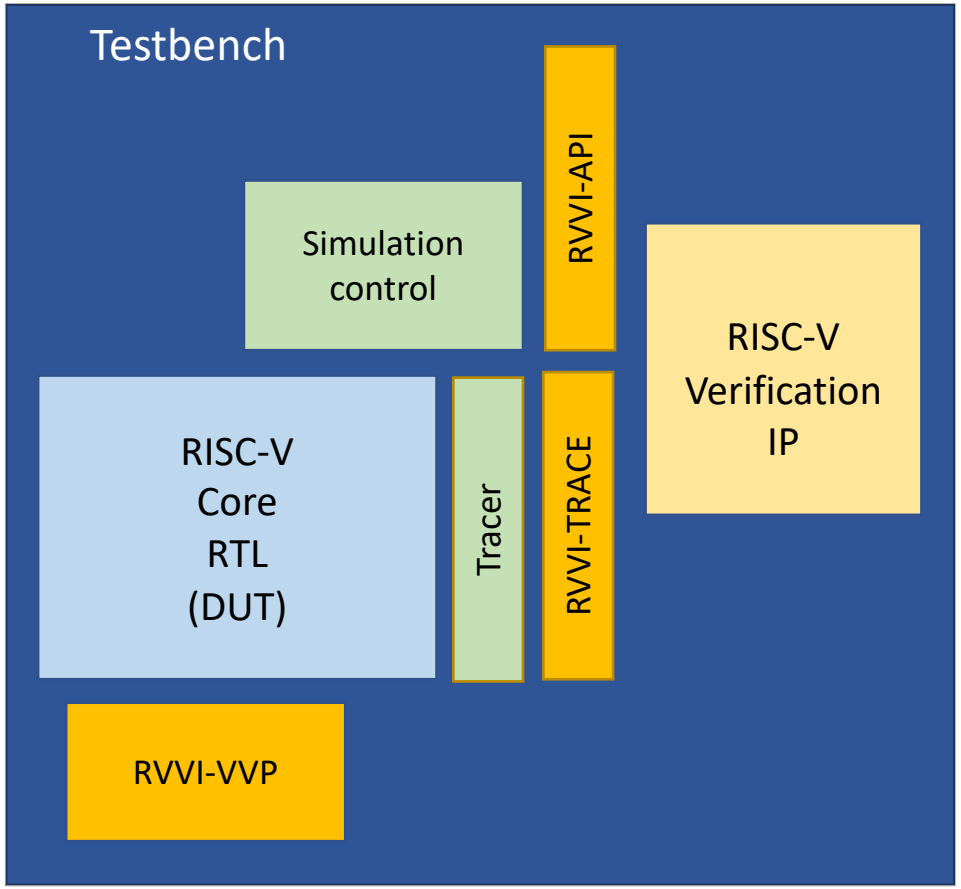
Second generation: challenges

- Asynchronous event handling
  - reference model cannot provide independent verification of async events
  - Example:
    - multiple interrupts are enabled and pending
    - reference model cannot verify the correct one (by priority) was serviced
- CORE-V Tracer additions to support step-and-compare bespoke to particular processor
  - tracer interface would need to change for each new RISC-V core

RISC-V Verification Interface (RVVI)

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

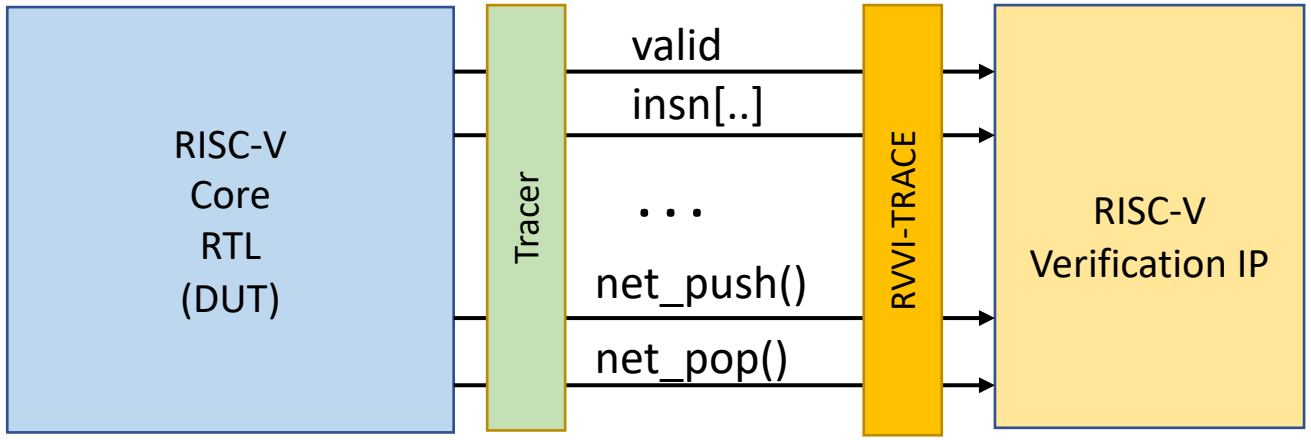
- Common components should have standard interfaces
- Standardize communication between RTL, testbench, and RISC-V VIP
  - **RVVI-TRACE**: signal level interface (tracer) between RTL and testbench
  - **RVVI-API**: function level interface between testbench and RISC-V VIP
  - **RVVI-VVP**: virtual verification peripherals



RVVI-TRACE

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

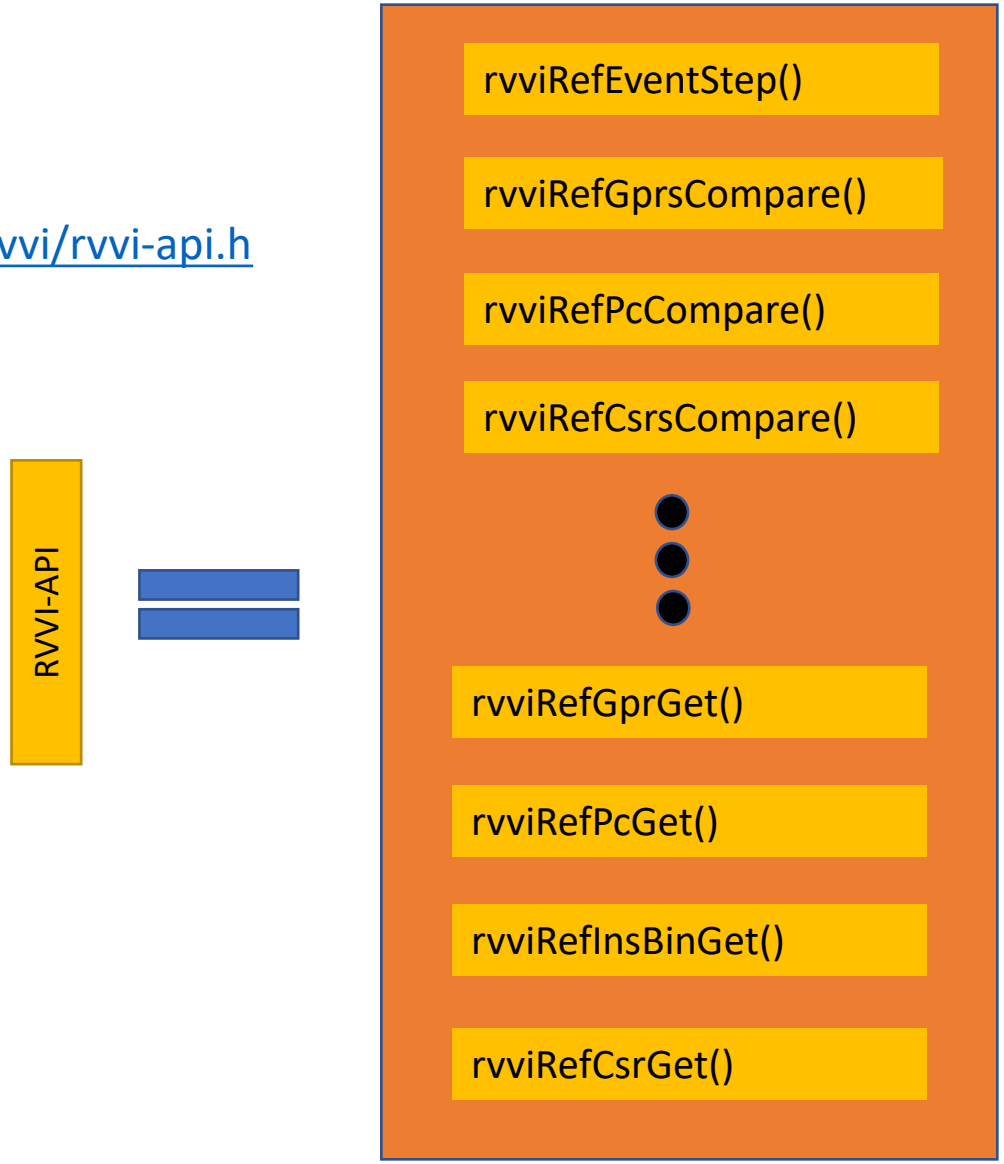
- Generically defines information to be extracted by tracer
- SystemVerilog interface
- Includes functions to handle asynchronous events
  - implements a queue to store multiple net changes during a single interval



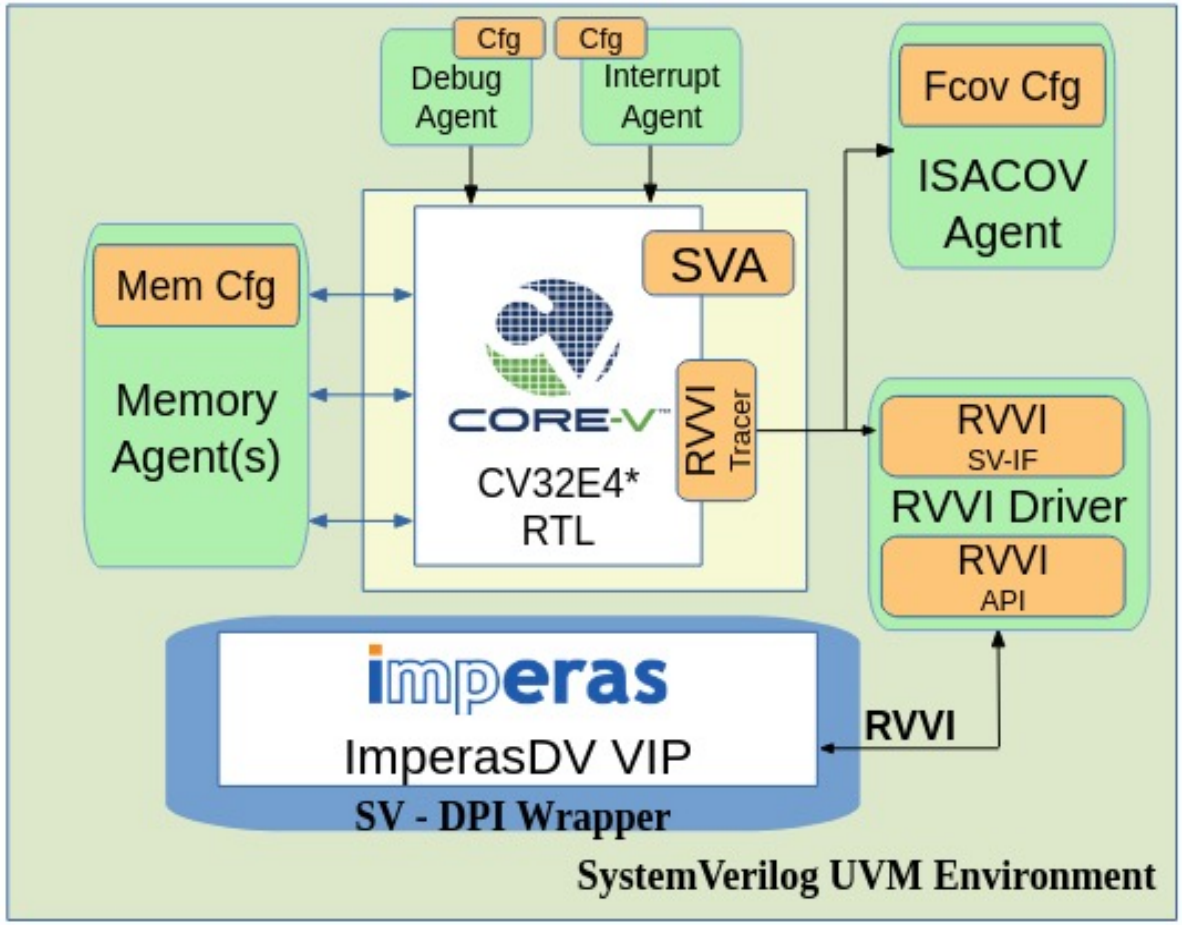
RVVI-API

https://github.com/riscv-verification/RVVI/blob/main/include/host/rvvi/rvvi-api.h

- Standard functions that RISC-V processor VIPs need to implement
- Supports a co-simulation, continuous comparison methodology
- C and SystemVerilog versions available
- Example: functions to marks registers/fields as “volatile”



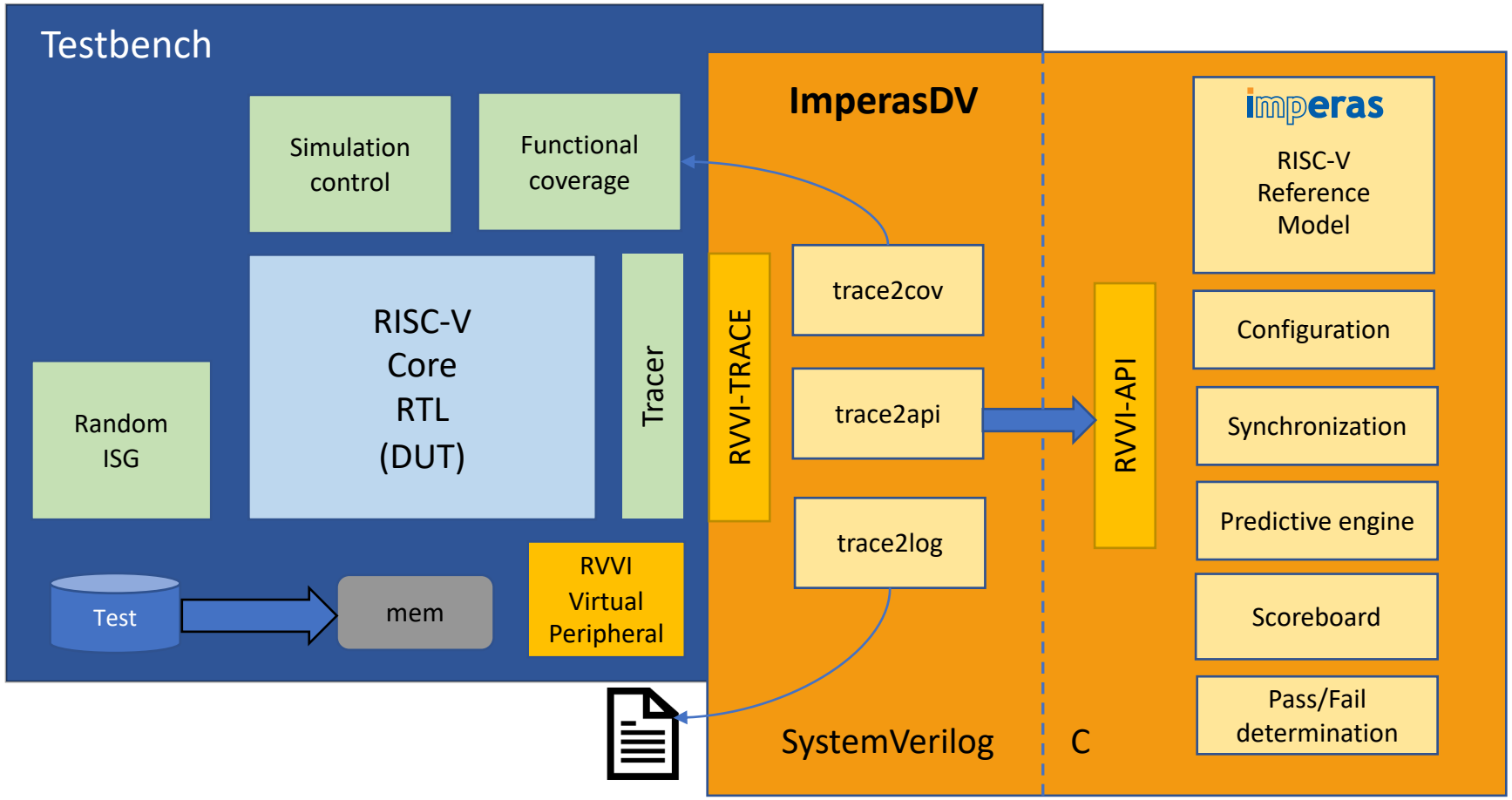
Third generation CORE-V-VERIF



Third generation: Improvements

- CORE-V tracer replaced with an RVVI-compliant tracer
- Imperas Reference model replaced with ImperasDV verification IP (VIP) that incorporates the reference model
- Step-and-compare logic eliminated

RISC-V CPU DV using Verification IP

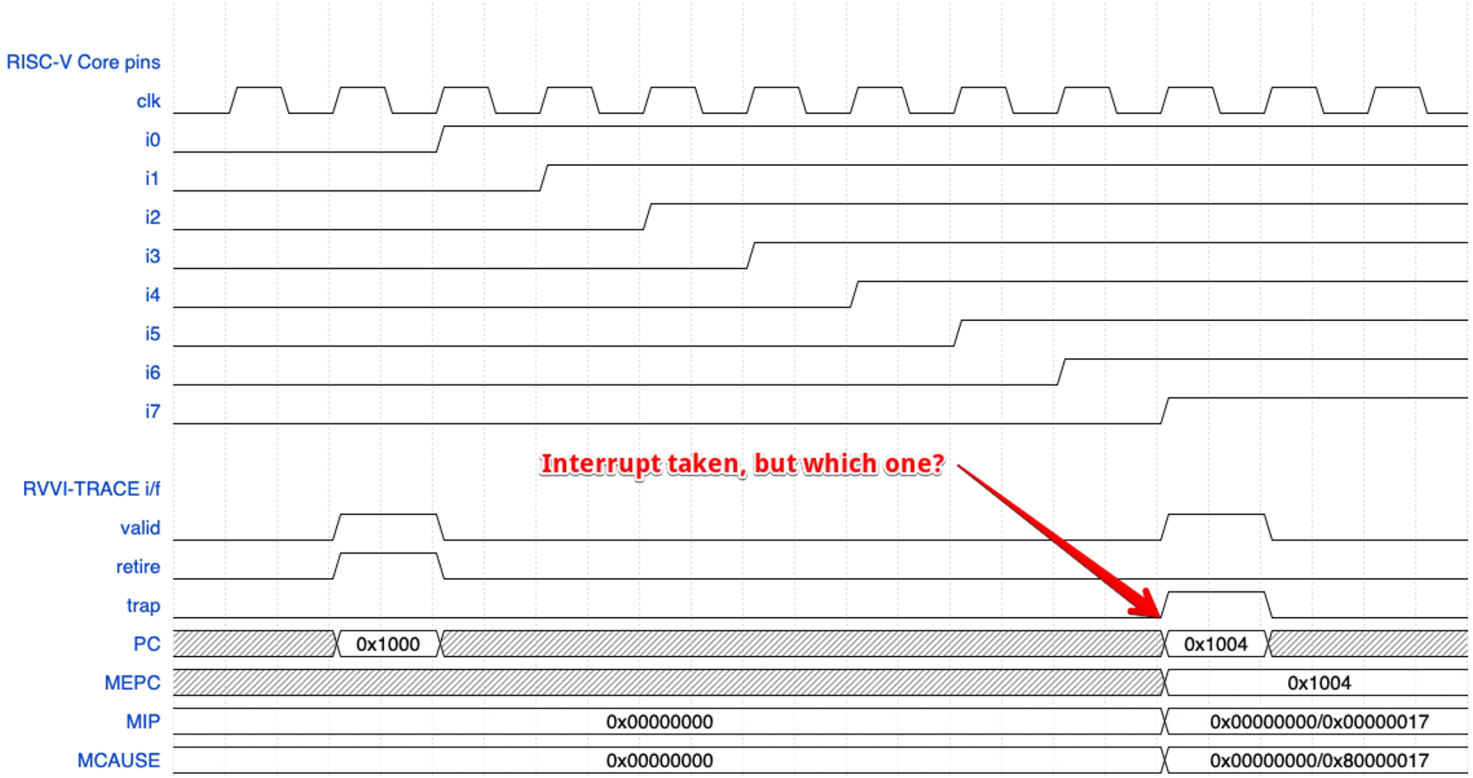


5 components of RISC-V CPU DV

- DUT subsystem with 'tracer'
- Tests: (random) instruction test generator and directed tests
- Functional coverage measurement
- Test bench / harness
- ImperasDV subsystem

- RVVI-TRACE i/f to core tracer
- RVVI-API i/f to verification IP
- RVVI-VVP virtual verification peripherals

Asynchronous events with ImperasDV



- Net changes received as a set
- Predictive engine analyzes all potential legal scenarios
- Produces an error if the DUT's response does not match one of them

Future work

- Incorporate machine-generated functional coverage code into CORE-V-VERIF
  - using data sampled from RVVI-TRACE i/f
- Advanced RISC-V Verification Methodologies (ARVM) projects
  - https://github.com/openhwgroup/programs/tree/master/TGs/verification-task-group/projects

To learn more…

- Active OpenHW Projects
  - https://github.com/openhwgroup/core-v-cores
- The CORE-V-VERIF environment on GitHub
  - https://github.com/openhwgroup/core-v-verif
- The CORE-V-VERIF Quick Start Guide
  - https://docs.openhwgroup.org/projects/core-v-verif/en/latest/quick_start.html
- Imperas RISC-V processor Design Verification solutions
  - https://www.imperas.com/index.php/imperasdv

Questions

- Thank you for attending

Aimee Sutton (aimees@imperas.com)

Lee Moore (moore@imperas.com)

Mike Thompson (mike@openhwgroup.org)