2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

# Planning for RISC-V Success Verification Planning and Functional Coverage

Duncan Graham, Aimee Sutton, Simon Davidmann, Imperas Software

Pascal Gouedo, Xavier Aubert, Yoann Pruvost, Dolphin Design

DOLPHIN DESIGN

imperas

accellera SYSTEMS INITIATIVE

# CV32E40Pv2: Industry + Open-Source collaboration

- CV32E40P:
  - 32-bit in-order RISC-V core maintained by OpenHW Group
  - originated as RI5CY (ETH Zurich)
  - Reached verification closure (TRL-5) in 2020
  - PULP instructions were removed in the first version
- CV32E40Pv2:
  - Dolphin Design joins OpenHW Group to continue development
  - PULP instructions are included in v2
  - Imperas provides verification IP and functional coverage to achieve TRL-5

## CV32E40P V2 Verification Challenges

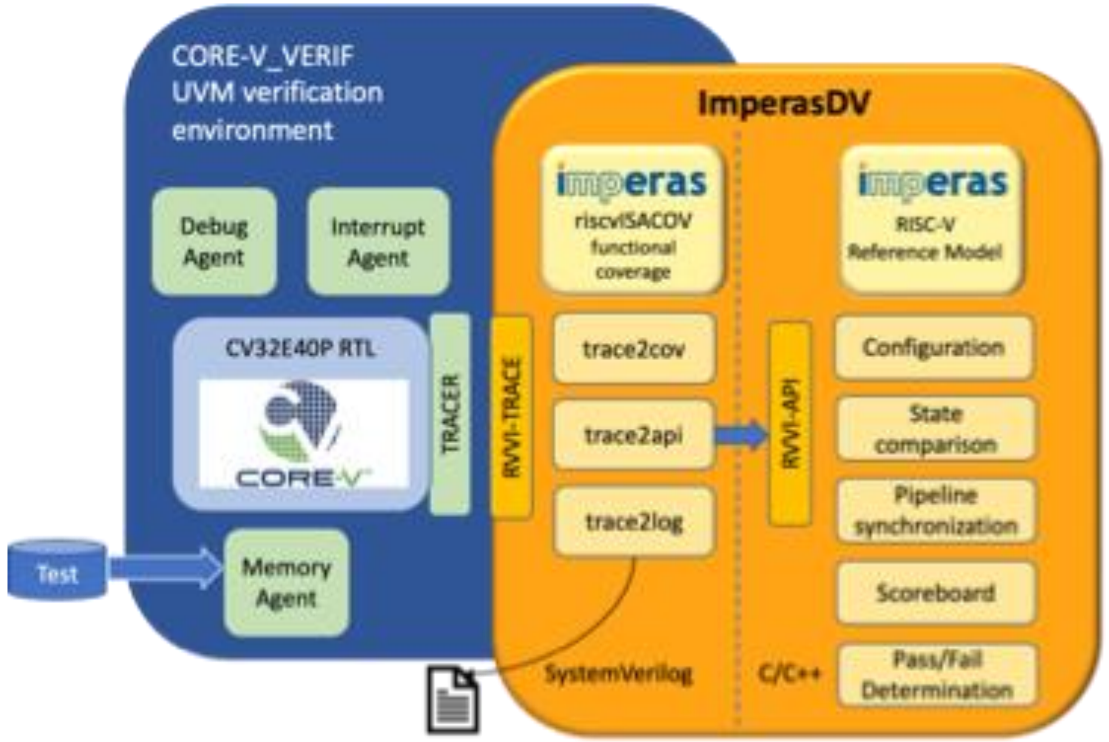
- 300 New instructions (PULP) to be verified
- 120 existing instructions from V1 must be re-verified
- Dual approach to verification: formal and functional simulation
- Formal: architectural compliance
- Functional: complex features, e.g. hardware loops, interrupts, debug mode

# Verification Planning

- For a RISC-V processor, a major source of test stimulus is the program running on the core
  - An infinite amount of stimulus is possible
- Verification plan is required to capture goals and determine when testing is complete
  - Functional coverage is a key metric to measuring those goals
- Verification plan completion is required to achieve TRL-5
- CV32E40P v2 verification plan:
  - https://github.com/openhwgroup/core-v-verif/tree/cv32e40p/dev/cv32e40p/docs/VerifPlans

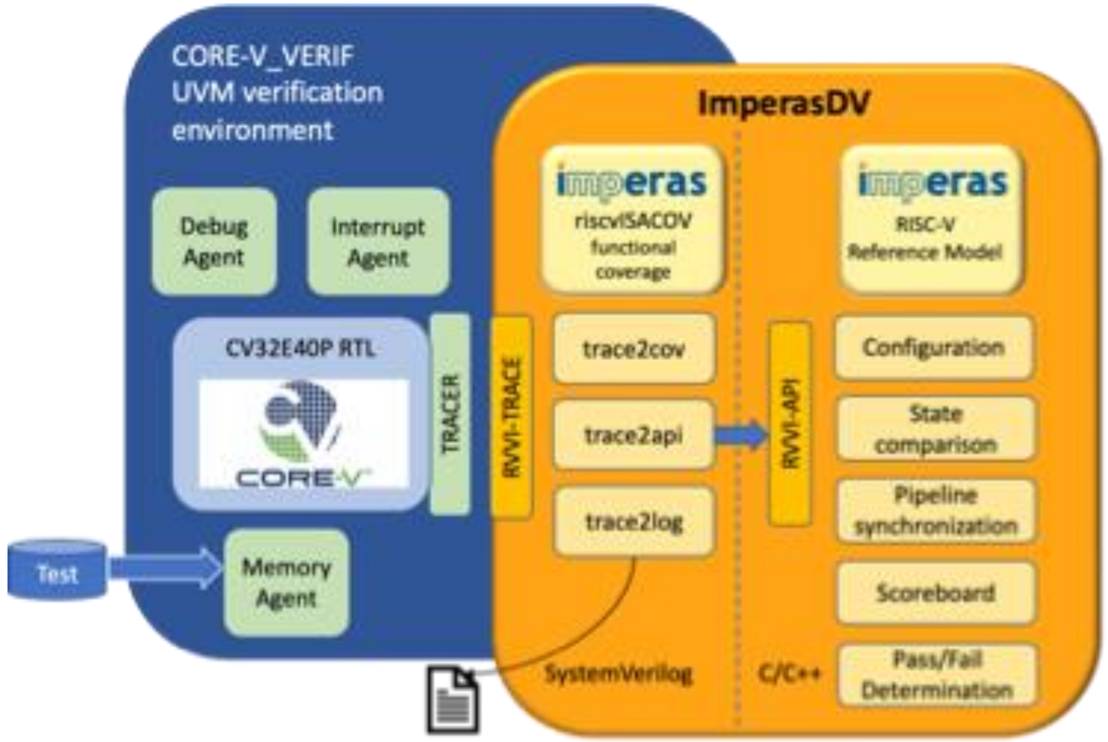
# Simulation Verification Environment

- CV32E40P v1 testbench replaced by a new one using ImperasDV
- Same program is run on both RTL and processor reference model
- Internal state of RTL and async events sent to ImperasDV using RVVI-TRACE
- Internal state of RTL and ref continuously compared as each instruction retires
- ImperasDV flags mismatches immediately



# Simulation Verification Environment (2)

- Functional coverage model (riscvISACOV) provided by ImperasDV
- Coverage is sampled using RVVI-TRACE data from RTL
- Coverage model designed to be extensible by the user

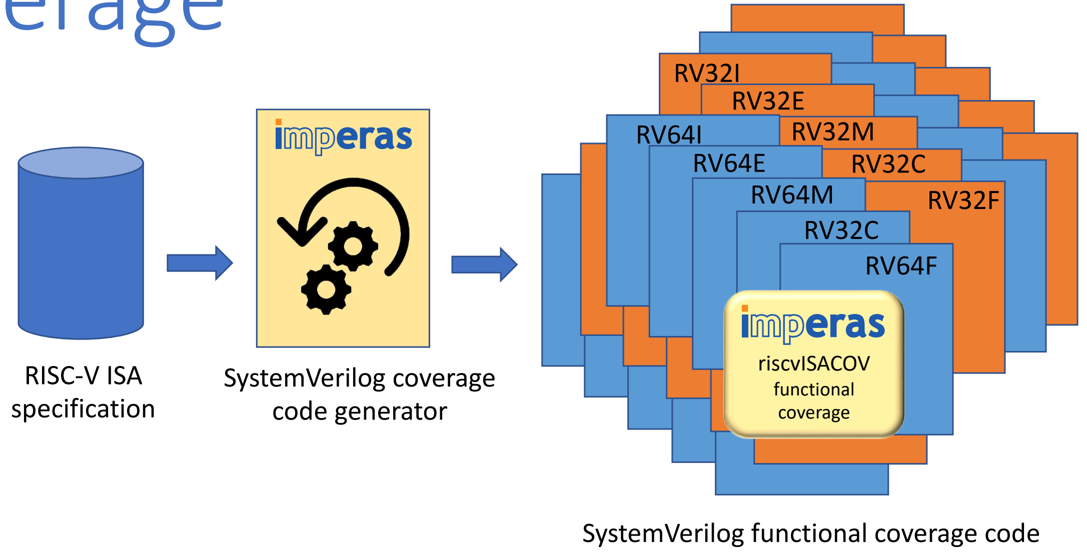


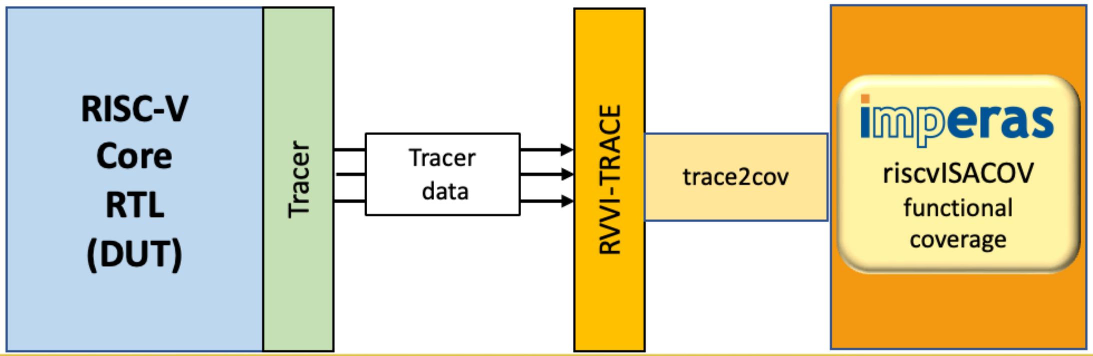
## Functional coverage challenges

- CV32E40P v2 implements I, M, (F or Zfinx), C extensions
  - total 120 instructions
- Each instruction requires 10-30 lines of SystemVerilog code to define coverage model
  - that’s 3600 lines of code
  - tedious and error-prone to create by hand
- 300 PULP instructions would require additional ~9000 lines of code
  - automation is required

# riscvISACOV
## SystemVerilog functional coverage

- A machine-readable version of the RISC-V ISA is used to auto generate reusable covergroups and coverpoints
  - Coverage defined by the RISC-V specification, not design specific
- Provides coverage of instructions, their operands and values, illegal instructions, data hazards
- Generated coverage is modular: include only the extensions which are implemented
- A similar paradigm was used to generate coverage for the PULP instructions


RISC-V ISA specification
imperas
SystemVerilog coverage code generator
RV32I
RV32E
RV64I
RV32M
RV64E
RV32C
RV64M
RV32F
RV32C
RV64F
imperas
riscvISACOV functional coverage
SystemVerilog functional coverage code



## Example: Hardware Loops

- Hardware loops make executing a piece of code multiple times possible without the overhead of branches penalty or updating a counter.
- Hardware loops involve zero stall cycles for jumping to the first instruction of a loop.
- A hardware loop is defined by
  - its **start address** (pointing to the first instruction in the loop)
  - its **end address** (pointing to the instruction just after the last one executed by the loop)
  - a **counter** that is decremented every time the last instruction of the loop body is executed.

programs/custom/pulp_hardware_loop/pulp_hardware_loop.S

```
#test1 hardware loops programmed using
#        immediates starti, endi and counti instructions
test1:
   li x17, 0
   li x18, 0
   .balign 4
   cv.counti   1, 10
   cv.endi     1, endO_1
   cv.starti   1, startO_1
   cv.endi     0, endZ_1
   cv.starti   0, startZ_1
   cv.counti   0, 10
startO_1:
startZ_1:
   addi x17, x17, 1
   addi x17, x17, 1
   addi x17, x17, 1
endZ_1:
   cv.counti 0, 10
   addi x18, x18, 2
endO_1:
```

## Hardware Loop Instructions

- Eight different instructions can be used in hardware loops
- CV.START instruction is used to set the start address of the loop relative to the current value of the program counter (PC)
  - Uses a register to hold the offset from the current PC value
  - This is used to load the loop start address register

| Requirement Location | Feature | Sub Feature | Feature Description | Verification Goal | Pass/Fail Criteria | Test Type | Coverage Method |
|---|---|---|---|---|---|---|---|
| CV32E40P User Manual - Chapter 18.3 | Hardware Loops Instructions | CV.START | cv.start L, rs1<br><br>lpstart[L] = PC + rs1<br>Loads the lpstart[L] CSR register with current PC value plus an unsigned integer | Register operands<br><br>All possible rs1 registers are used. | Check against RM | Constrained-Random | Functional Coverage |
| | | | | coverage:<br><br>All bits of rs1 are toggled | Check against RM | Constrained-Random | Functional Coverage |

# riscvISACOV and Hardware Loops

- riscvISACOV includes documentation / verification plan information
- csv and markdown formats for easy inclusion in verification plans

| Extension | Subset | Instruction | Description | Covergroup | Coverpoint | Coverpoint Description | Coverage Level |
|---|---|---|---|---|---|---|---|
| XPULPV2 | RVXPULPV2 | cv.start | | cv_start_cg | | | |
| | | | | | cp_asm_count | Number of times instruction is executed | Compliance Basic |
| | | | | | cp_L | HW Loop L | Compliance Basic |
| | | | | | cp_rs1 | RS1 (GPR) register assignment | Compliance Basic |
| | | | | | cp_rs1_sign | RS1 (GPR) sign of value | Compliance Basic |
| | | | | | cp_rs1_toggle | RS1 Toggle bits | Compliance Extended |
| | | | | | cp_rs1_maxvals | RS1 Max values | Compliance Extended |

Note: more coverpoints than specified in the CV32e40P v2 verification plan!

## Coverage Levels

- Enables the user to select a level of coverage depending on maturity of design
- Design is immature and undergoing frequent change:
  - compliance basic can provide insight into effectiveness of test stimulus
- Design is stable
  - compliance extended provides goals for verification closure

# CV32E40P PULP directed test with coverage

```
graham@lnx6474 uvmt $
```

## Summary and conclusion

- CV32E40P v2 faced several design verification challenges
  - 300 new PULP instructions
  - re-verification of v1 core
  - limited tool and human resources
- Success is achieved using
  - verification planning
  - RISC-V processor verification IP
  - Machine-generated functional coverage

## Questions

- Thank you
- Duncan Graham (graham@imperas.com)