OpenHW Group

Proven Processor IP

CORE-V Verification Overview and Status

Mike Thompson

mike@openhwgroup.org

Objectives

- Introduce of CORE-V verification to INVIA team:
  - Its mostly about CV32E40P at this time.
- Introduce the OpenHW Group workflow.
- Discuss future plans:
  - Phase 2 and 3 of CV32E40P verification
  - CV64A verification
  - Challenges
- Discuss INVIA Contribution.

Overall Status (1 of 2)

- On-going documentation:
  - Verification Strategy: mature, updates pending for debug, interrupt and CV64A.
  - Verification Plans (test plans): ~60% complete – not yet reviewed.
- SV/UVM verification environment for CV32E40P is in place:
  - Expect to integrate Instruction Set Simulator this week.
  - Next steps:
    - Google random instruction generator
    - Functional coverage model
  - Medium term:
    - Debug and Interrupts

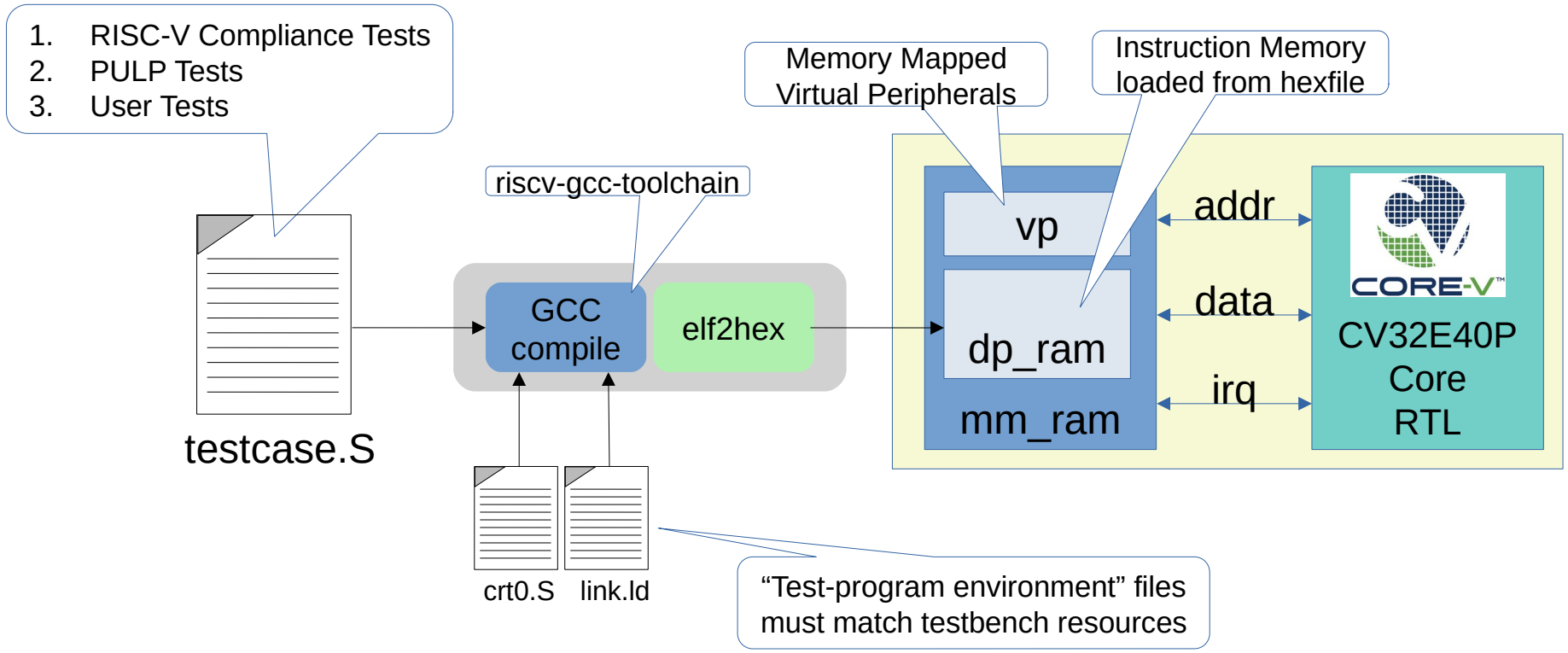
Overall Status (2 of 2)

- Formal Verification:
  - Formal Verification strategy captured in Verification Strategy.
  - Virtual Machine created for OneSpin.
  - Effort has stalled for lack of human resource.
- FPGA Prototyping:
  - A little early to start.
  - Should be a collaborative effort with the hardware and software task groups.
  - May be superseded by proposed “Soft Core” projects.
  - Bluespec has expressed interest.
- CV64A:
  - No significant progress.
- Miscellaneous:
  - Conducting a trial of SymbioticEDA MCY.
  - Verif-ai would like to use CV32E to trial its coverage closure tool.

CV32E40P Verification

- CV32E40P is almost a direct copy of PULP RI5CY.
  - Repository has been moved (not cloned or forked) to OpenHW.
  - Future CV32E40 planned (replace PULP instructions with RV ISA).
- OpenHW verification goals:
  - deploy industrial-grade SV/UVM verification for all CORE-V cores.
  - complete verification of both CV32E and CV64A cores.
- Up to this point, work has focused almost exclusively on CV32E.
- Many members of the PULP community objected to the loss of the “Verilator” testbench:
  - As a result, we maintain it under the name “core” testbench.

CORE Testbench



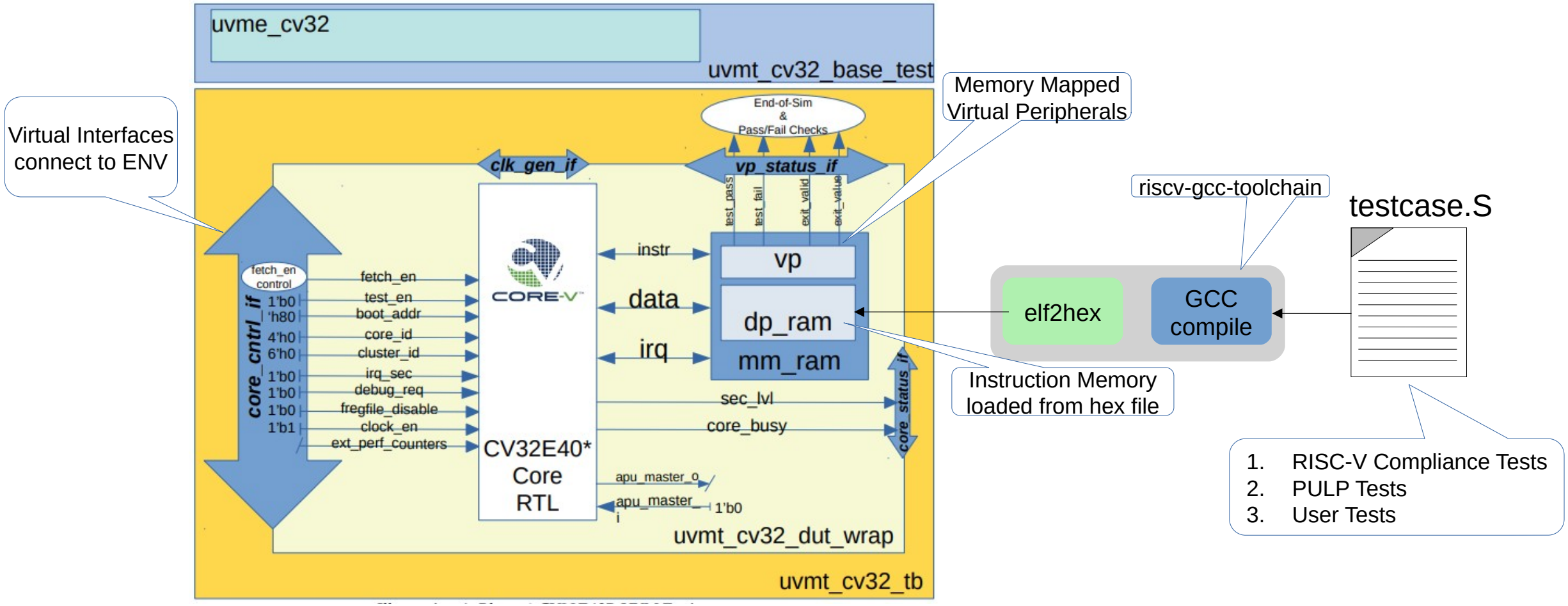
Status of CORE Testbench as of 2020-05-12

- Basic functionality supported:
  - Integrated with CV32E40P
  - Default target runs hello-world test using Verilator
  - Can run all inherited RI5CY testcases

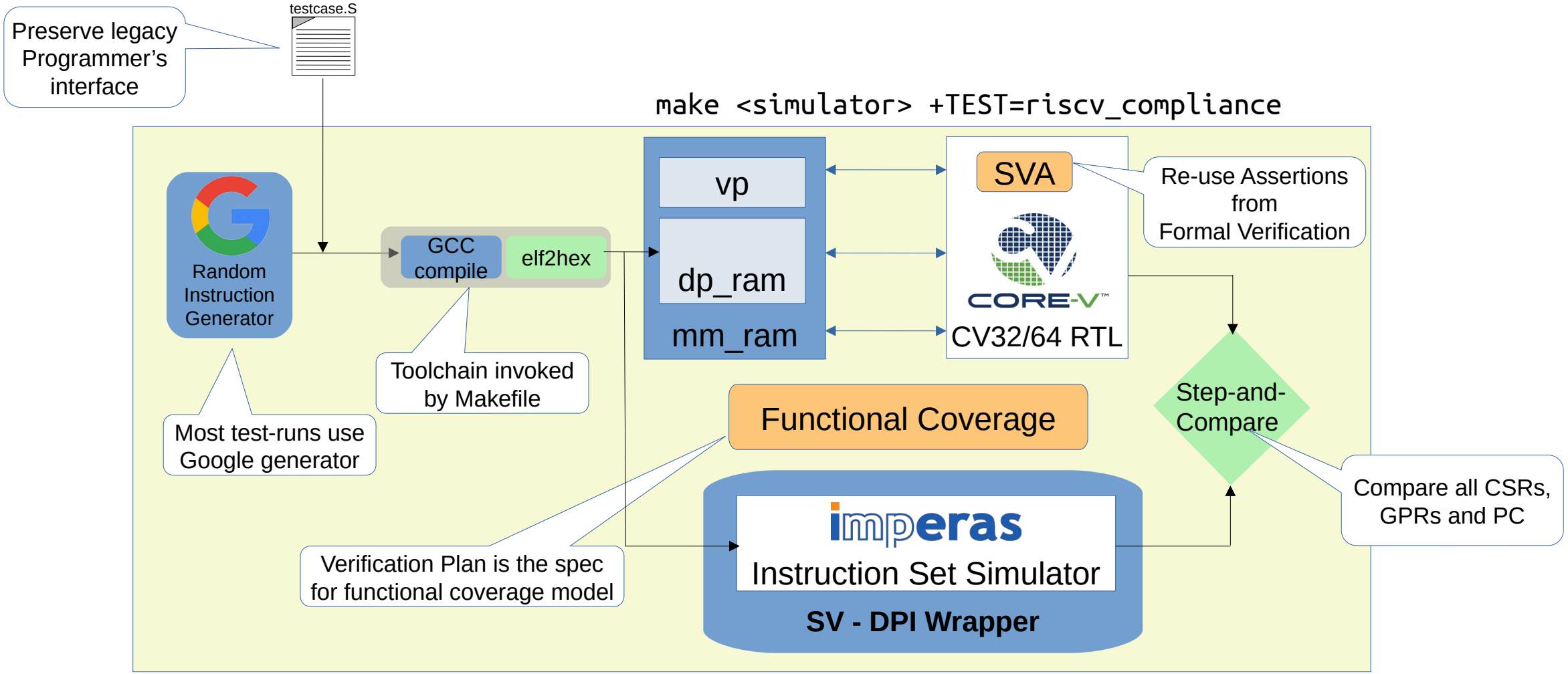
- Makefile for CORE testbench not much changed from February time-frame:
  - Common targets and variables centralized in Common.mk
  - Enforce use of specific branch/hash of the RTL.

- Purpose is to maintain simple environment for simple tests:
    - Designer-level testing
    - Demonstration platform
    - Not for regressions or coverage closure.

Phase 1 UVMT_CV32 Verification Environment



Planned UVM Environment for CV32



Status of UVMT_CV32 as of 2020-05-12

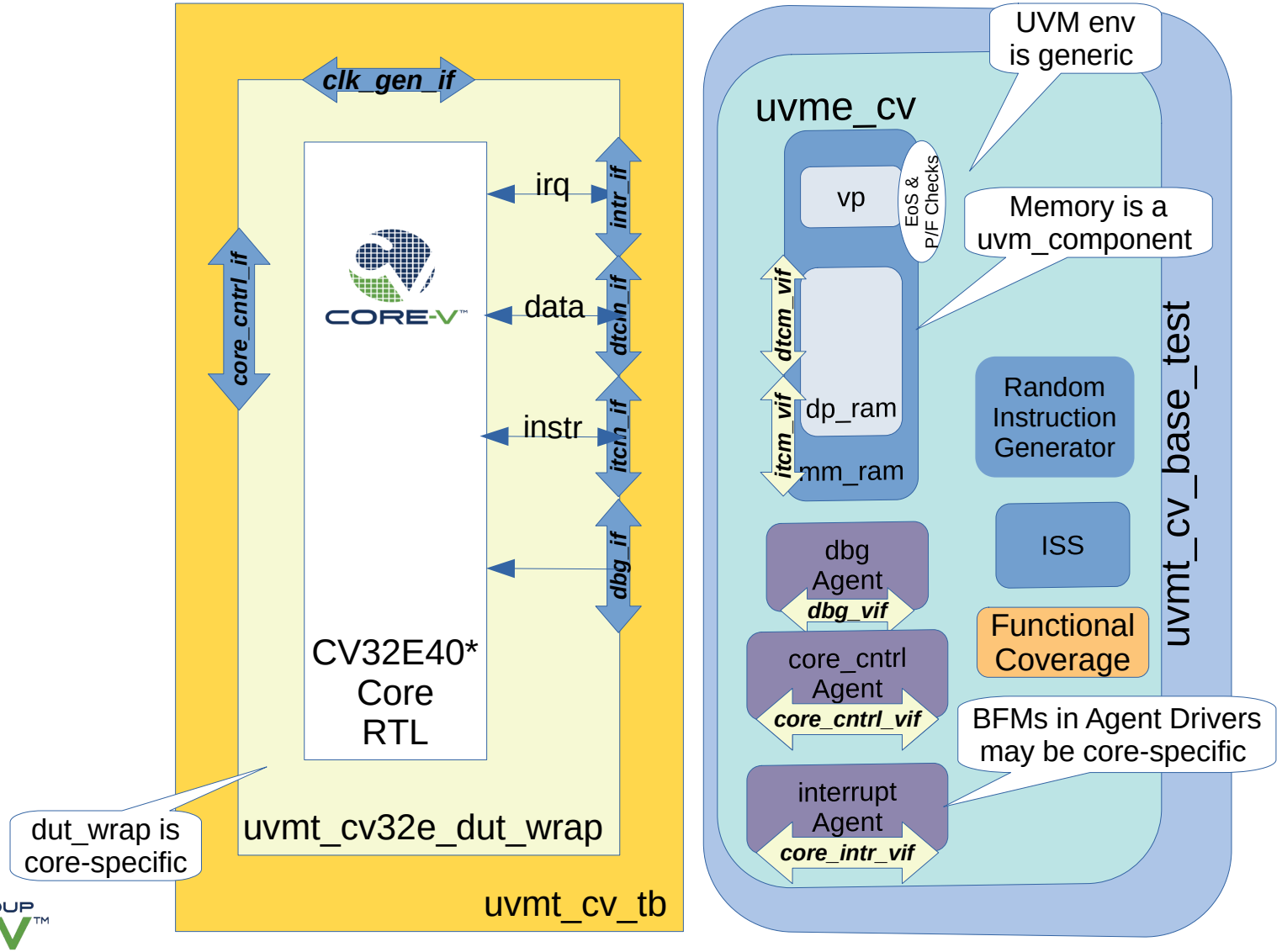
- Phase 1 development complete
  - Integrated with CV32E40P
  - Properly clone RTL from source GitHub Repos and uses Designer Manifest.
  - Can run all inherited RI5CY testcases (two known failures)

- Makefiles UVMT_CV32 have been significantly updated:
  - Enforce use of specific branch/hash of the RTL.
  - Common targets and variables centralized in Common.mk
  - Moved simulator-specific Make targets into separate files

- Phase 2 development of UVM environment on-going:
  - Integration of ISS (largely complete)
  - Ready to integrate Google Instruction Generator (riscv-dv)

- Software issues are starting to become a problem:
  - Need to standardize on a supported compiler/assembler toolchain
  - Need to define and implement runtime environment (crt0.S, link.ld, etc.)

Generic CORE-V UVM Environment



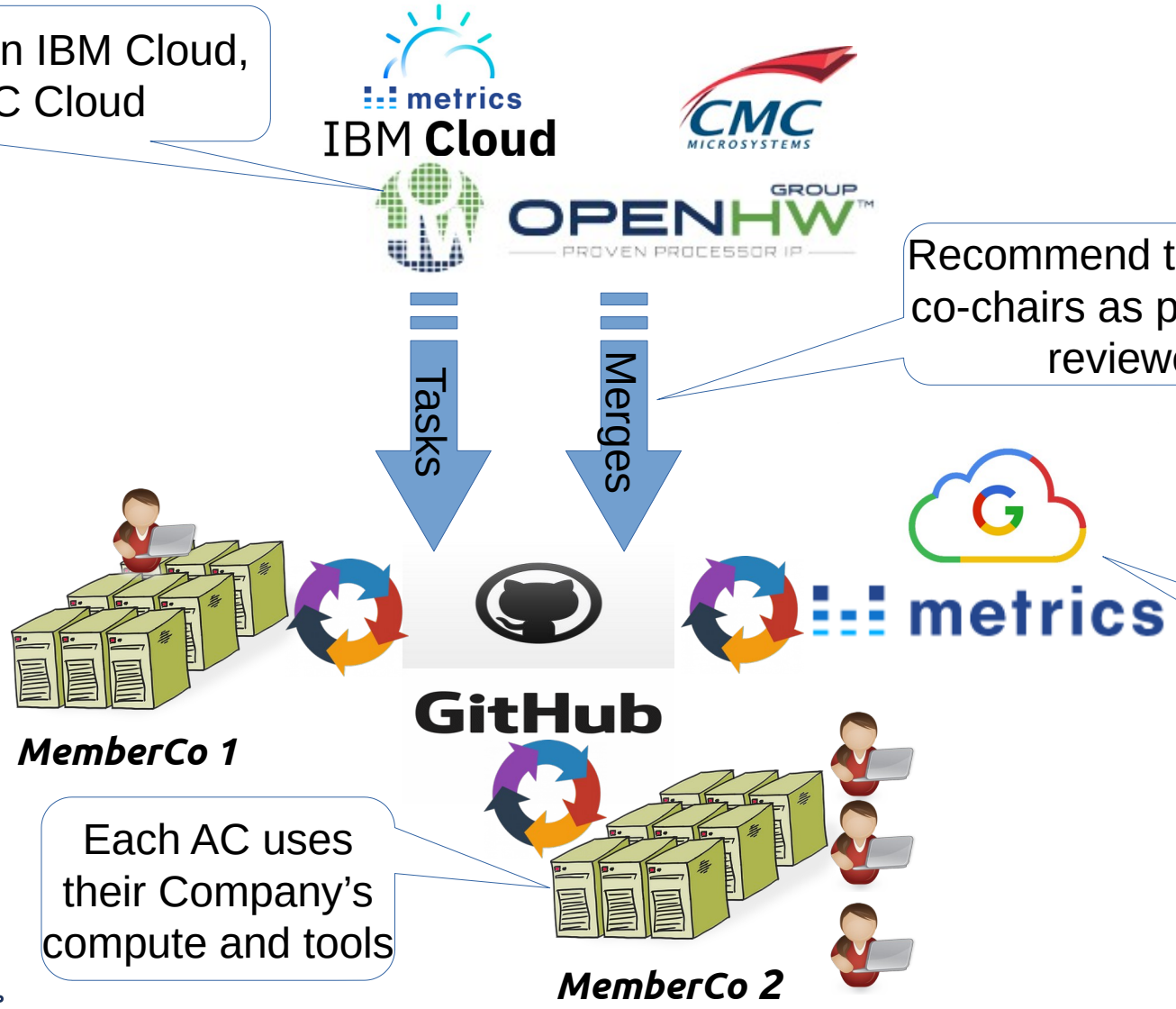
Generic ENV is *Proposal*, not a *Plan*

- The idea of a Generic verification environment for CORE-V verification needs to be evaluated against several criteria:
  - Overhead of development vs. benefit of re-use
  - Buy-in by OpenHW members

- Many open questions:
  - Does the concept of a generic env imply that we standardize on a single generator and ISS?
  - How to provide support for core-specific coverage, etc.

Workflow: Task Assignment, Execution and CI

OpenHW uses ***dsim*** on IBM Cloud, and ***xrun*** on CMC Cloud



Recommend to add VTG co-chairs as pull-request reviewers.

Metrics CI regressions use ***dsim*** on Google Cloud

Each AC uses their Company's compute and tools

On-going Challenges

- The design specification (known as the ‘user_manual’) is incomplete and falling behind
  - No human resource has been allocated to work on this.
  - Design changes to the RTL are happening without prior updates and review of the design specification.
  - At this point, the effort to complete the **verification plans** is stalled until the design specification is completed.

- Use of EDA tools:
  - Some members have been voiced concerns about OpenHW use of commercial SystemVerilog simulators.
    - The verification environment is coded in UVM and cannot be compiled/simulated by open source tools.
    - To alleviate this issue, I agreed to support a “core” testbench for CV32E40P that can be run with Verilator. (In hind-sight, I view this as a mistake.)

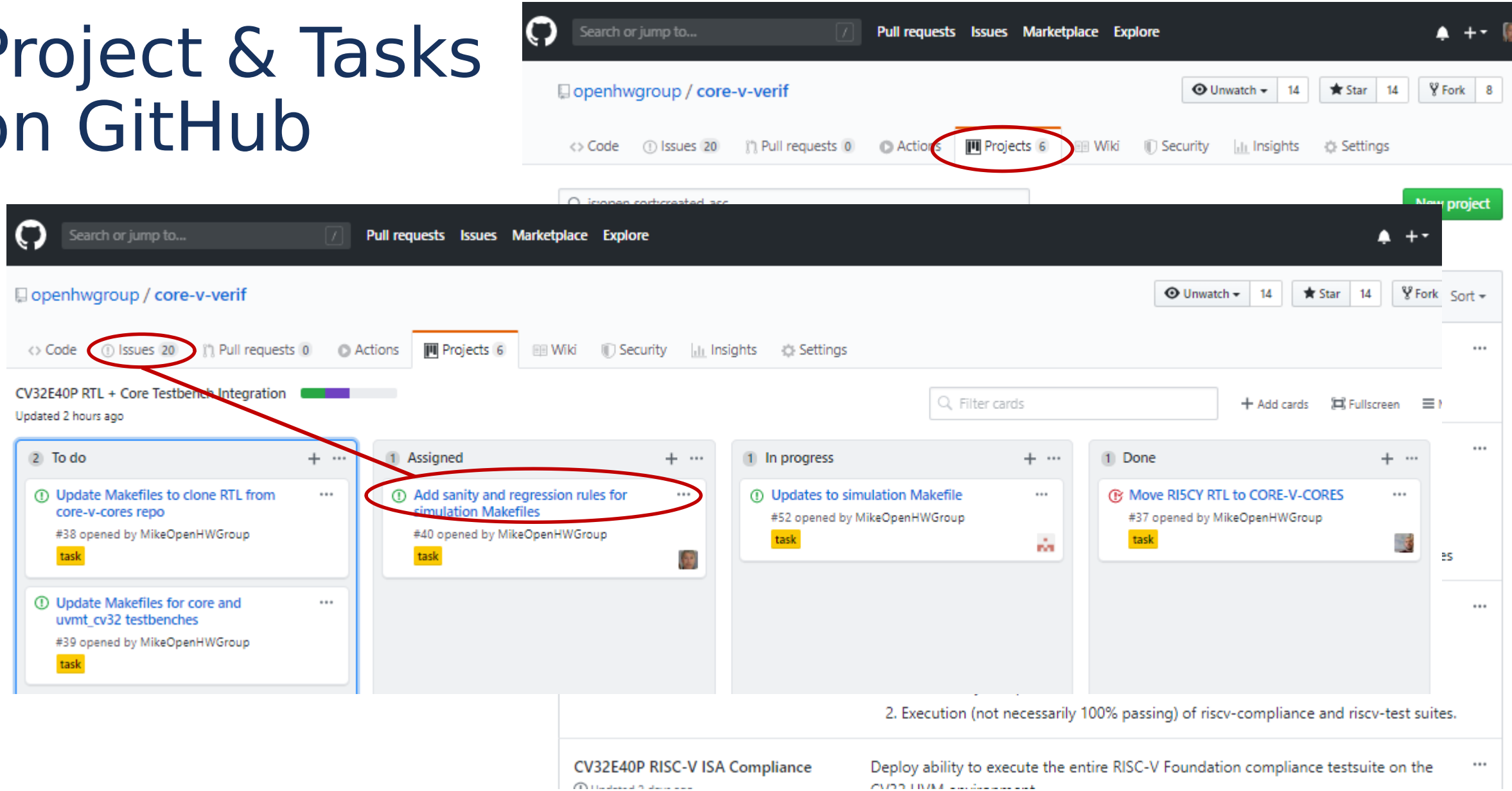
- The Verification Strategy calls for the use of the OVPsim Instruction Set Simulator (ISS)
  - This is commercial Verification IP sold by Imperas.
  - **All users of the CORE-V verification environments will need a license from Imperas.**
  - These will be provided free-of-charge by Imperas to OpenHW members.

- Toolchain
  - The CORE-V verification environments depend on a RISC-V GCC GNU compiler/assembler known as the “toolchain”.
  - Verification needs support from the Software Task Group:
    - The one supported by the RISC-V foundation does not support Xpulp instructions used by CV32E40P.
    - The PULP toolchain does not support some pseudo-instructions required by the RISC-V Compliance testsuite.

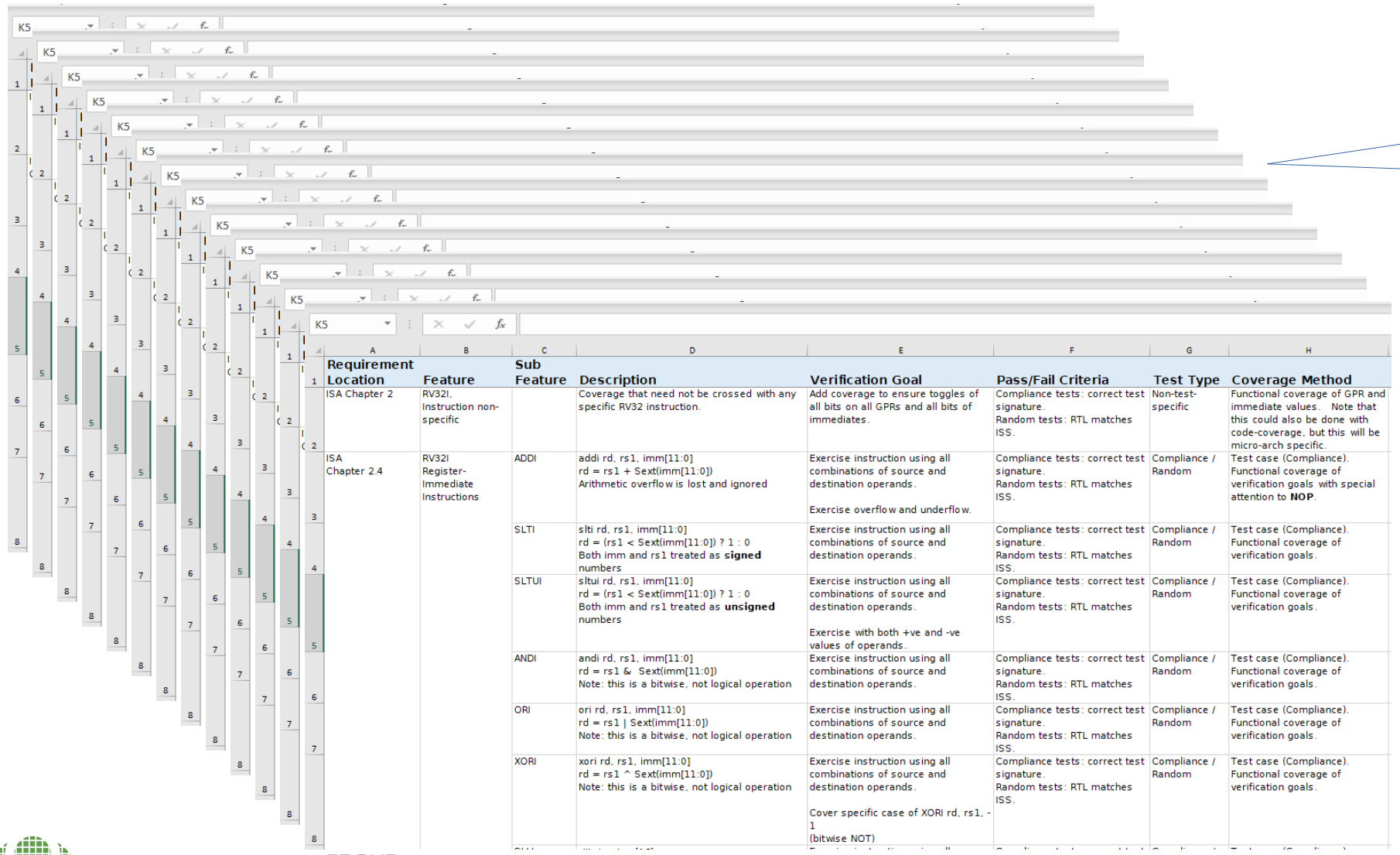
Thank You!

Background Slides

Project & Tasks on GitHub



Workflow: Verification Plans



Requirement Location	Feature	Sub Feature	Description	Verification Goal	Pass/Fail Criteria	Test Type	Coverage Method
ISA Chapter 2	RV32I, Instruction non-specific		Coverage that need not be crossed with any specific RV32 instruction.	Add coverage to ensure toggles of all bits on all GPRs and all bits of immediates.	Compliance tests: correct test signature. Random tests: RTL matches ISS.	Non-test-specific	Functional coverage of GPR and immediate values. Note that this could also be done with code-coverage, but this will be micro-arch specific.
ISA Chapter 2.4	RV32I Register-Immediate Instructions	ADDI	addi rd, rs1, imm[11:0] rd = rs1 + Sext(imm[11:0]) Arithmetic overflow is lost and ignored	Exercise instruction using all combinations of source and destination operands. Exercise overflow and underflow.	Compliance tests: correct test signature. Random tests: RTL matches ISS.	Compliance / Random	Test case (Compliance). Functional coverage of verification goals with special attention to **NOP**.
		SLTI	slti rd, rs1, imm[11:0] rd = (rs1 < Sext(imm[11:0]) ? 1 : 0 Both imm and rs1 treated as **signed** numbers	Exercise instruction using all combinations of source and destination operands.	Compliance tests: correct test signature. Random tests: RTL matches ISS.	Compliance / Random	Test case (Compliance). Functional coverage of verification goals.
		SLTUI	sltui rd, rs1, imm[11:0] rd = (rs1 < Sext(imm[11:0]) ? 1 : 0 Both imm and rs1 treated as **unsigned** numbers	Exercise instruction using all combinations of source and destination operands. Exercise with both +ve and -ve values of operands.	Compliance tests: correct test signature. Random tests: RTL matches ISS.	Compliance / Random	Test case (Compliance). Functional coverage of verification goals.
		ANDI	andi rd, rs1, imm[11:0] rd = rs1 & Sext(imm[11:0]) Note: this is a bitwise, not logical operation	Exercise instruction using all combinations of source and destination operands.	Compliance tests: correct test signature. Random tests: RTL matches ISS.	Compliance / Random	Test case (Compliance). Functional coverage of verification goals.
		ORI	ori rd, rs1, imm[11:0] rd = rs1 | Sext(imm[11:0]) Note: this is a bitwise, not logical operation	Exercise instruction using all combinations of source and destination operands.	Compliance tests: correct test signature. Random tests: RTL matches ISS.	Compliance / Random	Test case (Compliance). Functional coverage of verification goals.
		XORI	xori rd, rs1, imm[11:0] rd = rs1 ^ Sext(imm[11:0]) Note: this is a bitwise, not logical operation	Exercise instruction using all combinations of source and destination operands. Cover specific case of XORI rd, rs1, -1 (bitwise NOT)	Compliance tests: correct test signature. Random tests: RTL matches ISS.	Compliance / Random	Test case (Compliance). Functional coverage of verification goals.

Hierarchically organized by Feature.

One Spreadsheet per Feature