OpenHW Group

Proven Processor IP

CV32E40P Debug Verification Kickoff

Mike Thompson

mike@openhwgroup.org

Objectives

- Launch the Verification of the Debug Feature for CV32E40P
  - Ideally also portable to CV32E40 and CV64A
- Obtain agreement on the following:
  - Definition of what we are building/verifying.
  - Staffing:
    - How many engineers.
    - Contribution level (days per week).
    - Roles & Skills.
  - How we will manage and track progress.

What Are We Building?

- Currently, the “Device Under Test” is just the Core:
  - This is insufficient to support either Debug Verification or the OpenHW BHAG.
  - SiLabs has proposed a “RISC-V Subsystem”.

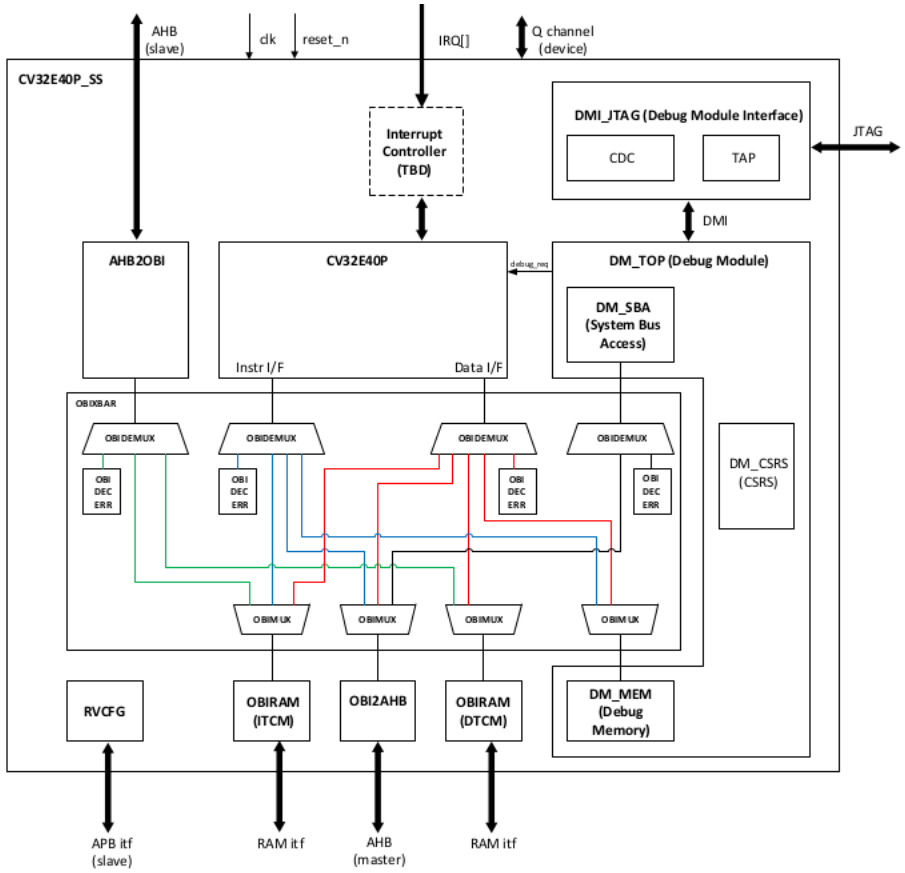


Figure 1 RISC-V subsystem (proposal for OpenHW BHAG)

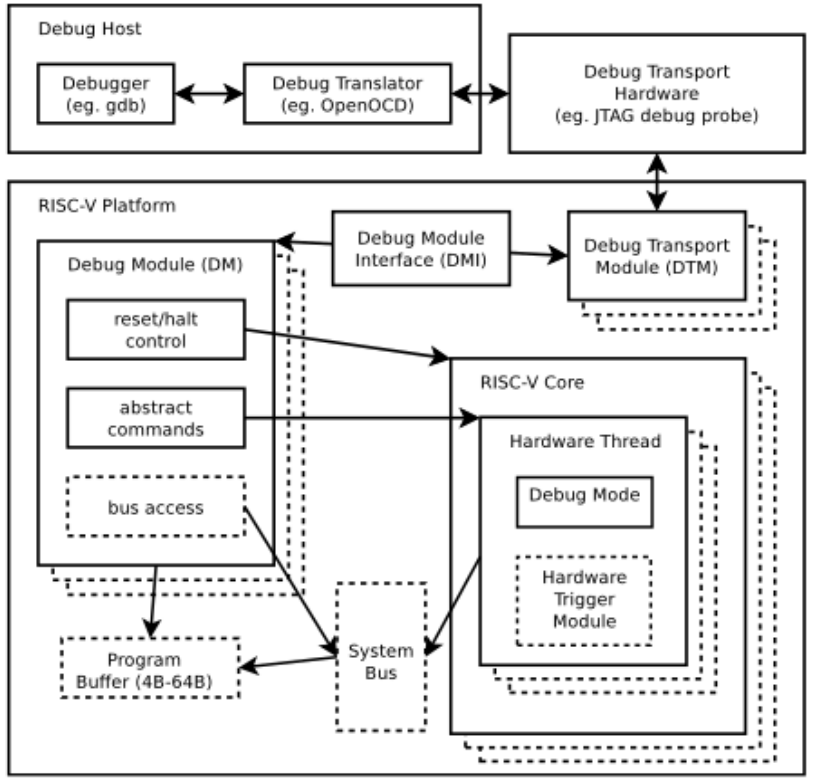


Figure 2.1: RISC-V Debug System Overview

Recommendation

- Define, Implement & Verify a “Device under Test” similar to the “RISC-V Subsystem”:
  - CORE-V Core (CV32E/CV64A).
  - Debug Module and associated Debug Module Interface (from riscv-dbg).
  - Exclude the Debug Transport Module (this is a delta from “RISCV Subsystem”).
  - Define the System Bus:
    - Are we adopting Open Bus Interface?
    - Is Memory considered external to the “RISCV Subsystem”?

  Note: in the meeting it was agreed to verify Debug at the CORE level first, and revisit Debug verification at the Subsystem level later.

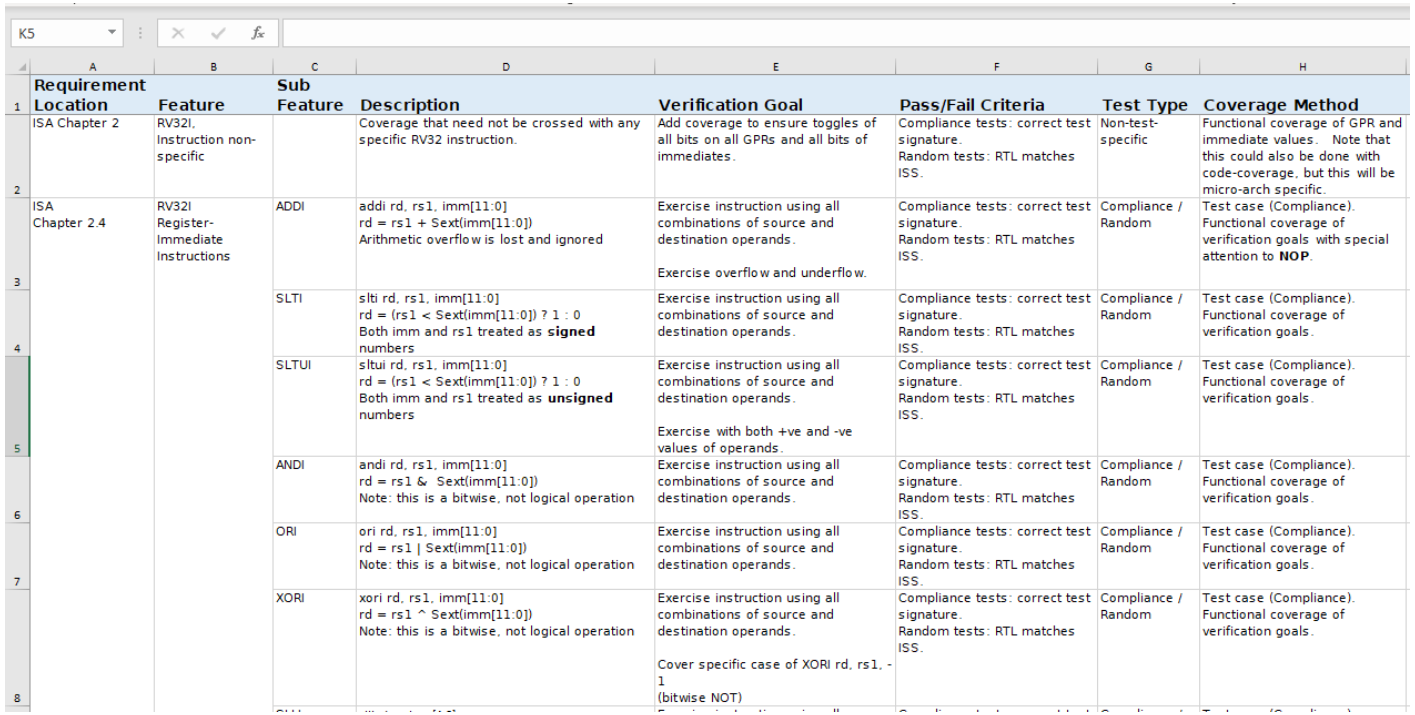
- Verification will be executed in the UVM environment in core-v-verif:
  - DMI to be driven by a (new) purpose-built UVM Agent.

How Big Is this Job?

- We won’t know until we have a few key items in place:
  - Joint understanding of scope.
  - User Manual updates.
  - Verification Plan.
- Verification can start now:
  - Much of the Vplan will come from the RISC-V Foundation Debug Spec.
  - Completion of the Vplan is gated by the User Manual.

Workflow: Vplans

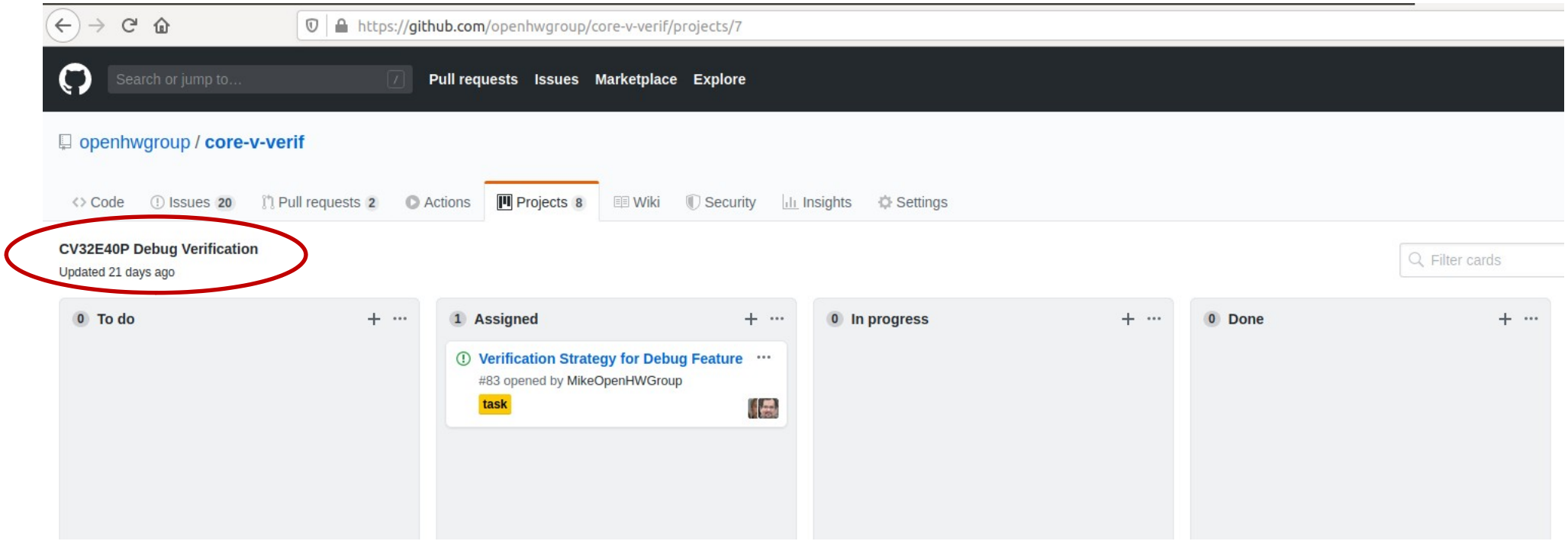
- Template for Verification Plans exists:
  - HOWTO documentation available.
  - Lots of existing examples: currently, ~60% of the Vplan for CV32E40P is captured.



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.

Workflow: Projects & Tasks on GitHub

- Two Projects relavent to this discussion:
  - **CV32E40P Verification Planning** (https://github.com/openhwgroup/core-v-docs/projects/1)
    - Tracks Vplan capture and review (also drives our Design colleagues to finish the UM!).
  - **CV32E40P Debug Verification** (https://github.com/openhwgroup/core-v-verif/projects/7)
    - Tracks Debug verification development and execution (mostly emtpy).



Thank You!

OPENHW GROUP™
PROVEN PROCESSOR IP