OpenHW Group

Proven Processor IP

Standardization of CORE-V-VERIF

Mike Thompson

mike@openhwgroup.org

OPENHW GROUP™
PROVEN PROCESSOR IP

© OpenHW Group

Background

- CORE-V-VERIF is now almost two years old!
  - Initial commit was 2019-11-29.

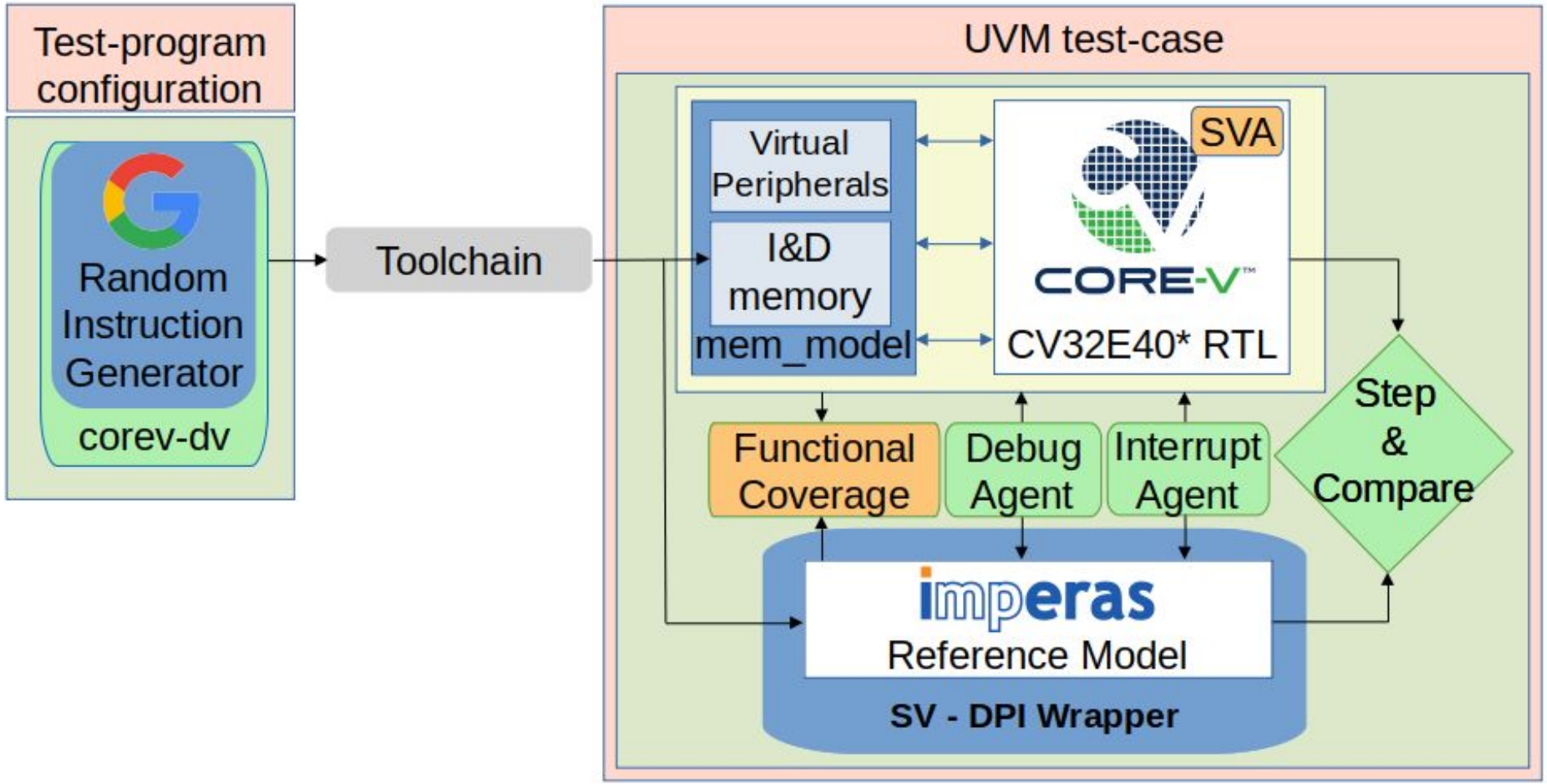
- Envisioned as a unified verification environment for all CORE-V cores.
  - Implemented reality is a unified set of core-level environments plus common library for UVM components and scriptware.
  - Currently supports CV32E40P, E40X, E40S and CVA6.
  - New cores coming online soon: CV32E40Pv2 and E41P.
  - The E40Pv2 will include verification of the F (and D?) ISA extensions.

- Need to standardize our approach to make it simpler for new projects to stand-up their cores in CORE-V-VERIF environment.

Objectives

- Articulate the CORE-V-VERIF philosophy and strategy for core-level verification:
  - Standardization to make it simpler for new projects to stand-up their cores in the CORE-V-VERIF environment.
- Get some feedback from the OpenHW Community.
- Articulate an “end goal” for a “universal CORE-V verification environment”.
- Socialize ongoing standardization activities:
  - Improved documentation.
  - Steve: Improved compile, simulation and regression flow.
  - Standardized UVM components.
  - Coding standards.
  - Linting.

CORE-V-VERIF Implementation as of 2021-03



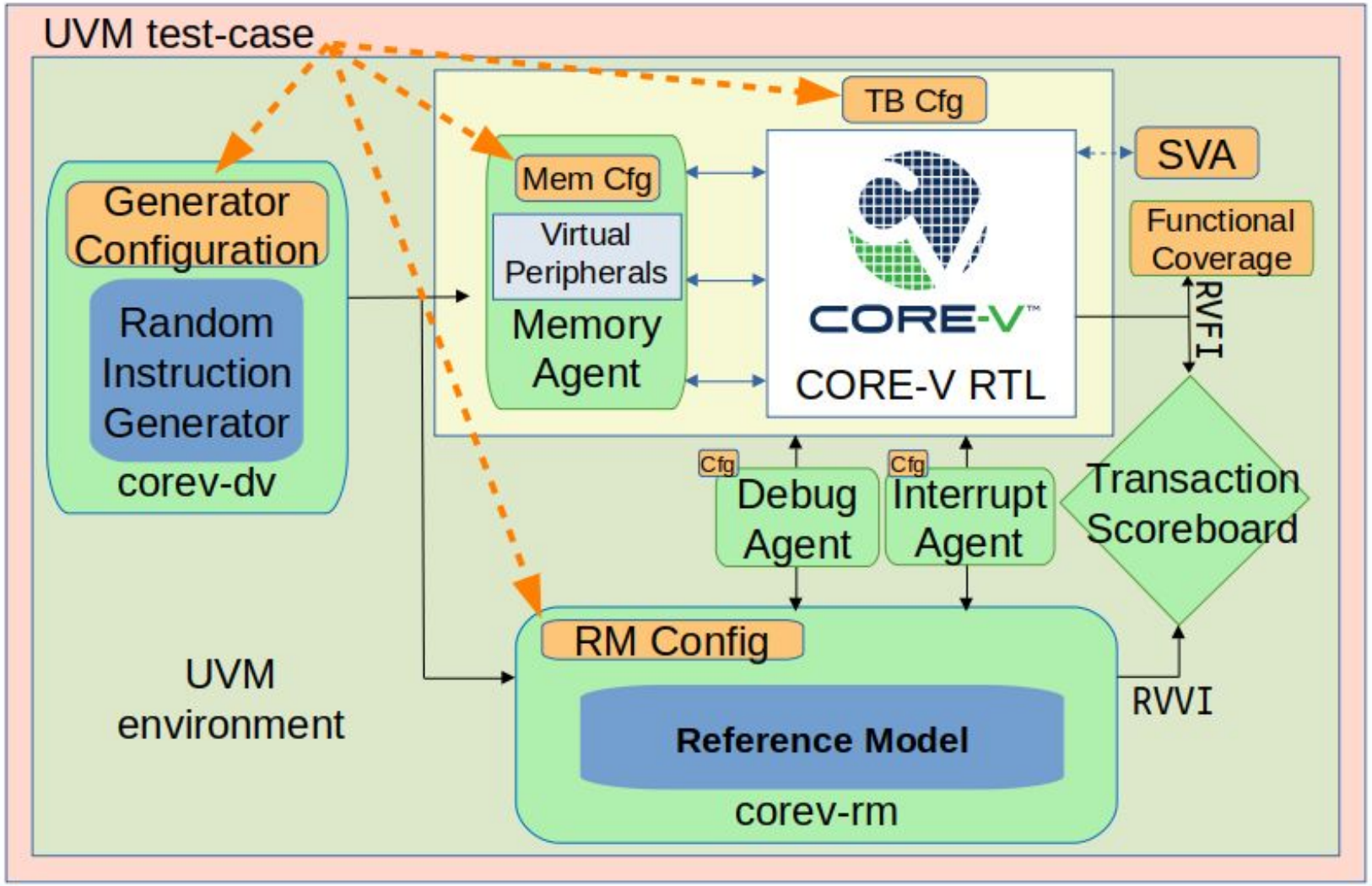
Deficiencies in CORE-V-VERIF
(1 of 2)

- Both Core RTL and RM presents non-standard, “core-specific” instruction trace interfaces to the environment:
  - Integrating a new core and/or RM involves a lot of re-work.
- Fixed Reference Model:
  - Changing the RM is a “forklift” operation that impacts step-and-compare function and functional coverage collection.
- ISA Functional coverage data sourced by Reference Model:
  - Should be sourced by RVFI to allow for operation independent of RM.
- Support cost of step-and-Compare functionality is high:
  - Requires throttling of the core clock (always confuses the Designers!).
  - Changes to cycle-timing behavior of core often changes cycle-timing behavior of core’s tracer interface - this makes the environment ‘brittle’.
  - Not future proof: it’s unclear if step-and-compare can support out-of-order cores.

Deficiencies in CORE-V-VERIF
(2 of 2)

- Only supports a single generator: riscv-dv
  - OpenHW does not have direct influence over future direction of riscv-dv.
- Generator is not a UVM component controllable by UVM run-flow:
  - Must be run in a standalone ‘simulation’ before running the core environment.
  - Control knobs for randomization are static.
- Disjoint control of configuration and stimulus:
  - Lack single-point-of-control of test-program, testbench configuration and UVM testcase.
- Static Memory Model:
  - Fixed Address range.
  - No support for PMA or Virtual Memory.
- Most Assertions are embedded in the RTL:
  - Makes them invisible to the DV plan.

The Goal: a universal UVM environment for CORE-V cores



Features of “universal” core-v-verif

- Single UVM environment:
  - UVM Testcase has control of all components.
- **Corev-dv** for random instruction generator:
  - Wrapper for either **riscv-dv** or **FORCE-RISCV**.
  - “Generator Configuration” provides single test-specification interface to UVM testcases.
- Eliminate need for Toolchain:
  - Generators emit machine code directly.
  - Will maintain “legacy programmers interface” to Toolchain.
- **Corev-rm** for the core’s reference model:
  - Wrapper for either **Imperas OVPsim ISS** or **Spike** (others possible).
  - “RM Cfg” provides single test-specification interface to UVM testcases.
- Standardize “tracer” interfaces:
  - **RVFI** for cores.
  - **RVVI** for reference models.
- Functional Coverage model independent of Reference Model.
- Transaction Scoreboarding of retired instructions:
  - May retain step-and-compare functionality.
  - Eases support for out-of-order cores.
- New “Memory Agent” component:
  - Improved randomization of memory-to-core transactions.
  - Supports rich set of virtual peripherals.
  - Supports testcase constrainable memory interface parameters, PMA, etc.

Progress towards the “universal” environment

(as of 2021-09-24)

- Standardize “tracer” interfaces in place for CV32E40X and CV32E40S:
  - **RVFI** for cores.
  - **RVVI** for reference models.
- ISA Functional Coverage model is now totally independent of Reference Model.
  - Reusable, configurable ISA functional coverage model also in place.
- Transaction Scoreboarding of retired instructions:
  - Retains step-and-compare functionality.
  - Out-of-order checking not yet in place (not yet needed).
- OBI Memory UVM Agent:
  - Generates transactions for ISA coverage.
  - Improved randomization of memory-to-core transactions.
  - Supports rich set of virtual peripherals.
  - Supports constrainable memory interface parameters, PMA, etc.

OPENHW GROUP™
PROVEN PROCESSOR IP

Improved Documentation

- CORE-V-VERIF documentation today is “OK”, but not “good”.
- Lot of *in-place* documentation in the forms of READMEs.
- Excellent DVplans.
- The so-called Verification Strategy needs a forklift upgrade.

Coding Style Guidelines

- Coding style guidelines have multiple purposes:
  - Avoidance of poorly supported coding constructs.
  - Standardized look-and-feel which can make it easier to understand intent.
  - Improved maintainability of the code base.

- There are very few publicly available coding guidelines for DV code:
  - E.g. IBEX guidelines are mostly applicable to RTL code.

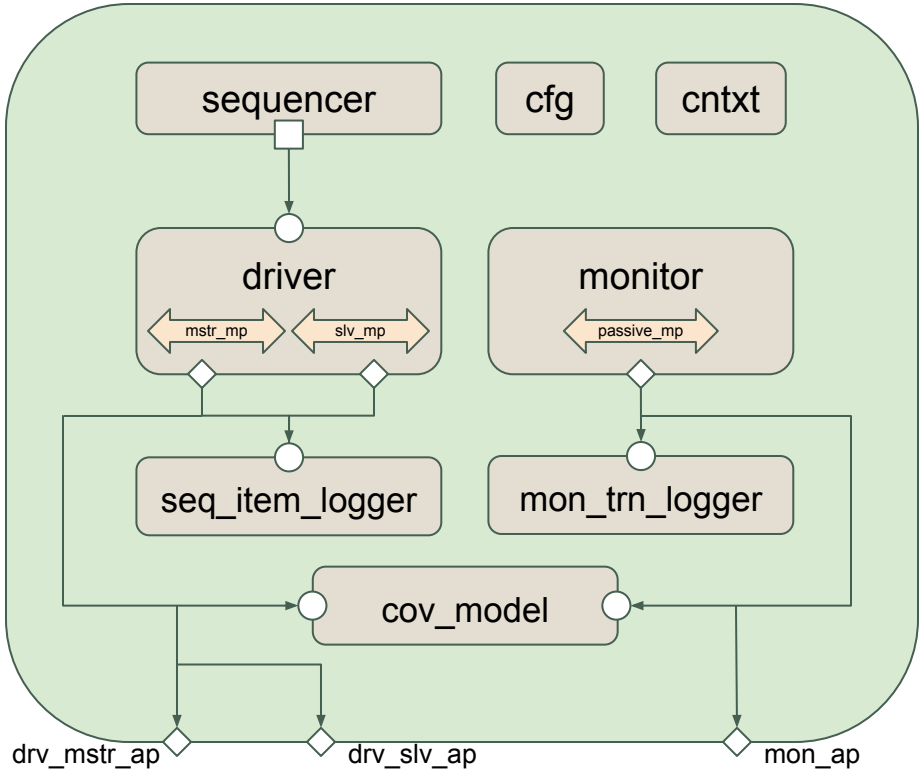
- Currently reviewing the coding guidelines in “Advanced UVM” by Brian Hunter.
  - Not sure if Hunter will allow these to be open-sourced.
  - If you are aware of others, please post something on the TWG: Verification channel on MatterMost.

Standardizing UVM Components

- Many teams that develop UVM verification environments create standardized templates for generating individual UVM components.
  - Templates produce “complete” components that are ready to instantiate in a working environment and will compile out-of-the-box.
  - Component-specific behavior is human added as needed.
- The newly developed obi_memory_agent was based on Moore.io templates and generation scripts.
  - Moore.io is an open-source project of Datum Technology Corporation.

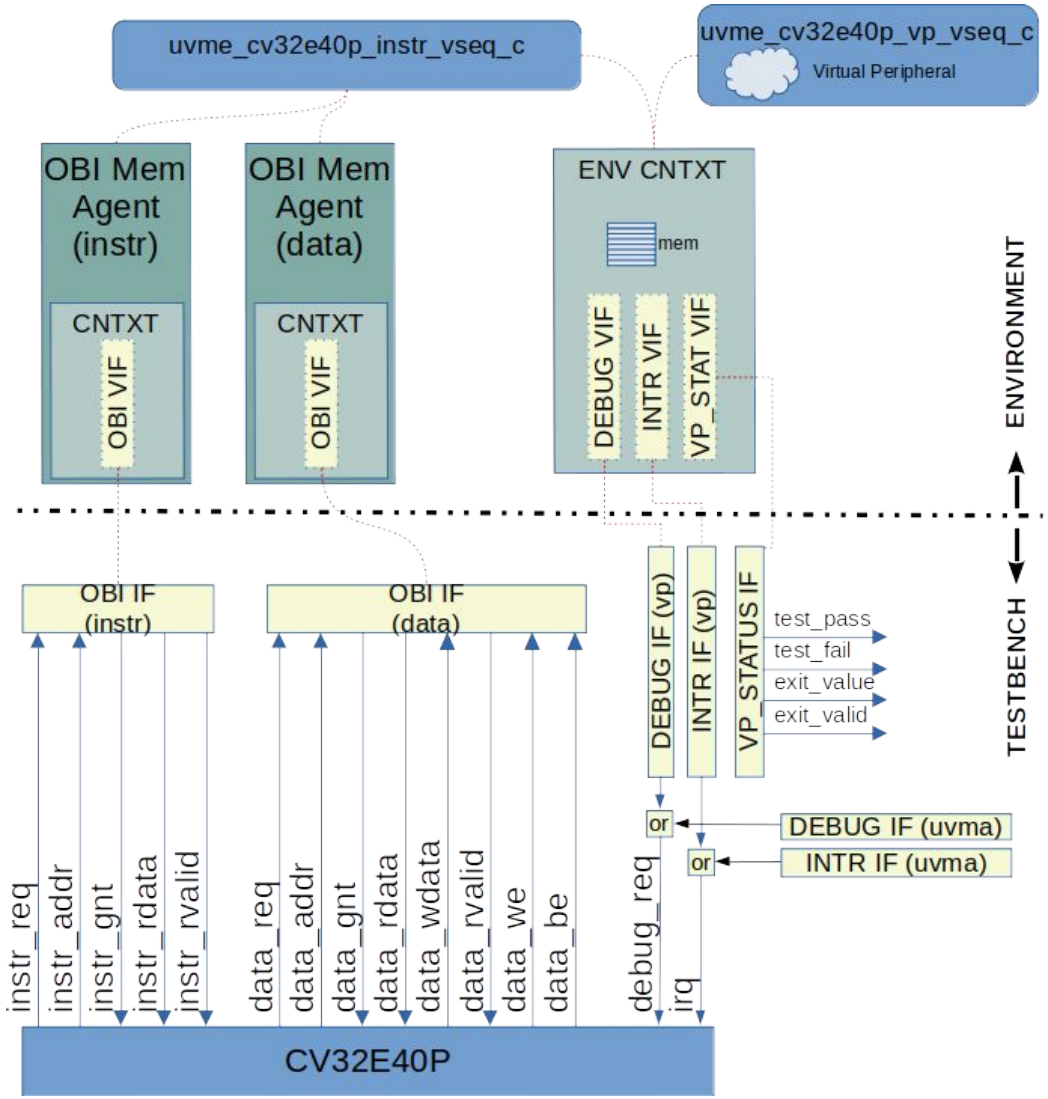
Standardizing UVM Components:
Structure of a Moore.io UVM Agent

- Recognizable UVM Agent.
- Unique features:
  - Context object.
  - Driver and Monitor use interface modports (must be implemented by the SV Interface).
- Common structure:
  - Can lead to lots of unimplemented classes (e.g. loggers)



Standardizing UVM Components:
OBI_MEMORY_AGENT in CV32E40P ENV

- Two instances of obi_memory_agent:
  - Instruction bus (read only)
  - Data bus (read/write)
- Agent Context has OBI VIF member which is ***set()*** in the test bench and ***get()*** in the agent.
- Environment context members:
  - VIFs for debug, interrupt and status.
  - Sparse memory model for all memory segments (instruction, data, debug, etc.)
- As much as is practical, operation of agents is determined by sequences.
  - Agents are “as dumb as possible”.



Linting

- Next presentation...

Thank You