OPENHW GROUP

PROVEN PROCESSOR IP

Ongoing Integration of CVA6 in CORE-V-Verif

mike@openhwgroup.org

2022-02-15

Introduction

- A bit of background information about CORE-V-VERIF:
  - CV32E40P was the first core integrated into CORE-V-VERIF:
    - It started life as the RI5CY “core” testbench.
    - The core testbench was used as the testbench module for the UVM environment.
    - Addition of UVM components and capabilities was added step-by-step.
    - Today, the “core testbench” is no longer used in the UVM environment.
  - The other “E40 cores” were integrated into the UVM environment from the start:
    - New requirements from the E40X and E40S DVplans are driving continued evolution of CORE-V-VERIF.

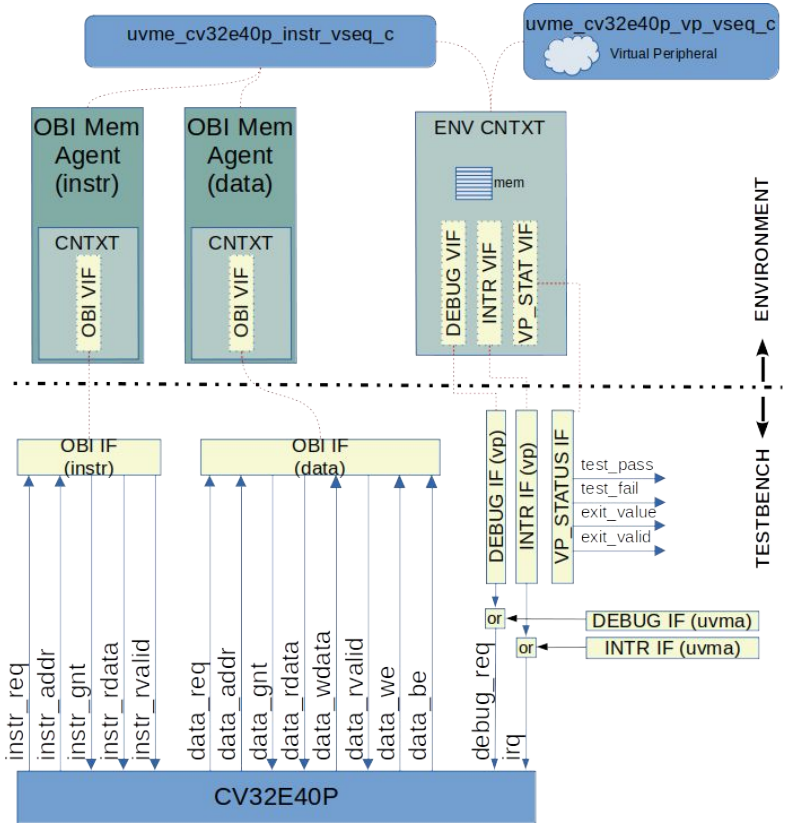
- Which CORE-V-VERIF features currently in use by the E40 cores should be integrated into the CVA6 environment?

Some Suggestions

- Mike’s ideas for CORE-V-VERIF features that may be applicable to CVA6:
  - DVplans!!!
  - Memory, Interrupt and Debug Agents.
  - Common Configuration Class.
  - Test-program Definitions.
  - Modular Makefile structure.
  - On-the-fly Step-and-Compare of PC, GPRs and CSRs between RTL and Spike.
  - ISACOV: functional coverage model for the RISC-V ISA.
- Concepts that CORE-V-VERIF does not yet support that may be useful for CVA6:
  - Handcar API for Spike.
  - Virtual Memory Scoreboard.
- Other verification topics to consider:
  - Verification Checklists (definition of what “verification complete” means).
  - Code coverage strategy (what types, how will exclusions be handled, etc.).

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”.



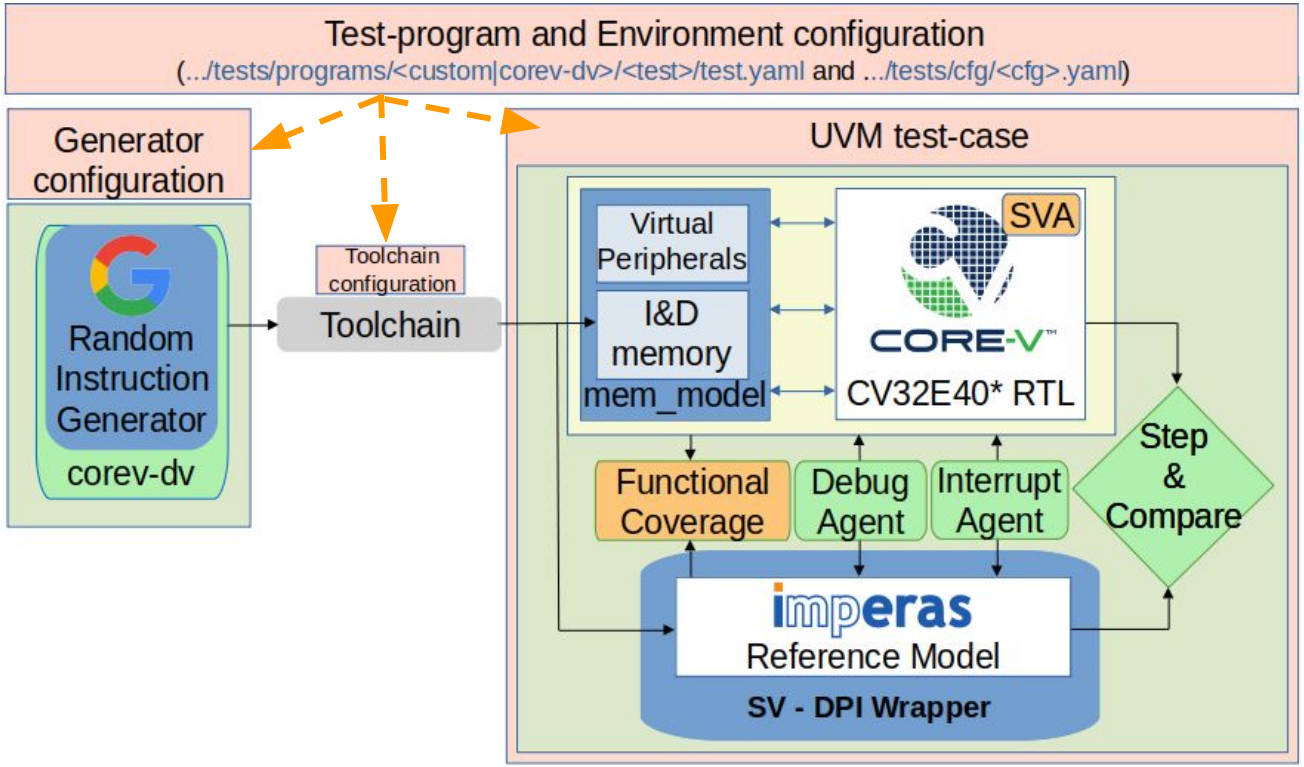
Common Environment Configuration Class

- ISA support (extensions, modes)
- Common switches
- ISS, scoreboard enabled, CSR scoreboard checks
- Common parameters
- Common bootstrap pins

[$(CORE-V-VERIF)/cv32e40x/env/uvme/uvme_cv32e40x_cfg.sv]

```
rand bit                        scoreboarding_enabled;
bit                             disable_all_csr_checks;
bit [CSR_MASK_WL-1:0]           disable_csr_check_mask;
rand bit                        cov_model_enabled;
rand bit                        trn_log_enabled;

// ISS configuration
bit                             use_iss;
string                          iss_control_file = "ovpsim.ic";

// RISC-V ISA Configuration
rand corev_mxl_t                xlen;
rand int unsigned               ilen;

rand bit                        ext_i_supported;
rand bit                        ext_a_supported;
rand bit                        ext_m_supported;
rand bit                        ext_c_supported;
rand bit                        ext_b_supported;
rand bit                        ext_p_supported;
rand bit                        ext_v_supported;
rand bit                        ext_f_supported;
rand bit                        ext_d_supported;
rand bit                        ext_zifencei_supported;
rand bit                        ext_zicsri_supported;

rand bit                        mode_s_supported;
rand bit                        mode_u_supported;

rand bit                        pmp_supported;
rand bit                        debug_supported;

rand bit                        unaligned_access_supported;
rand bit                        unaligned_access_amo_supported;

// Mask of CSR addresses that are not supported in this core
// post_randomize() will adjust this based on extension and mode support
bit [CSR_MASK_WL-1:0]           unsupported_csr_mask;

// Common parameters
int unsigned                    num_mhpmcounters;
uvma_core_cntrl_pma_region_c    pma_regions[];

// Common bootstrap addresses
// The valid bits should be constrained if the bootstrap signal is not valid for this core configuration
rand bit [HART_ID_WL-1:0]       hart_id;
rand bit [MAX_XLEN-1:0]         boot_addr;
rand bit                        boot_addr_valid;      You, 2 days ago • creation of comprehensive core control
rand bit [MAX_XLEN-1:0]         mtvec_addr;
rand bit                        mtvec_addr_valid;
rand bit [MAX_XLEN-1:0]         dm_halt_addr;
rand bit                        dm_halt_addr_valid;
rand bit [MAX_XLEN-1:0]         dm_exception_addr;
rand bit                        dm_exception_addr_valid;
rand bit [MAX_XLEN-1:0]         nmi_addr;
rand bit                        nmi_addr_valid;
```

Common Test-program and Testcase Configuration



Common Test-program and Testcase Configuration

```
# Test definition YAML for corev-dv test generator

# corev-dv generator test
name: corev_rand_interrupt
uvm_test: corev_instr_base_test
description: >
    RISCV-DV generated random interrupt tests
plusargs: >
    +start_idx=0
    +instr_cnt=10000
    +num_of_sub_program=5
    +directed_instr_0=riscv_load_store_rand_instr_stream,4
    +directed_instr_1=riscv_loop_instr,4
    +directed_instr_2=riscv_hazard_instr_stream,4
    +directed_instr_3=riscv_load_store_hazard_instr_stream,4
    +no_fence=1
    +enable_interrupt=1
    +randomize_csr=1
    +boot_mode=m
    +no_csr_instr=1
```

```
# Test definition YAML for random interrupt test

# corev-dv generator test
name: corev_rand_interrupt_<RUN_INDEX>
uvm_test: uvmt_cv32_firmware_test_c
description: >
    Random interrupt generator test
plusargs: >
    +gen_irq_noise
```

Regression Management

- YAML test list format
- Python utility to validates and convert from YAML to proprietary format
  - e.g. Metrics JSON

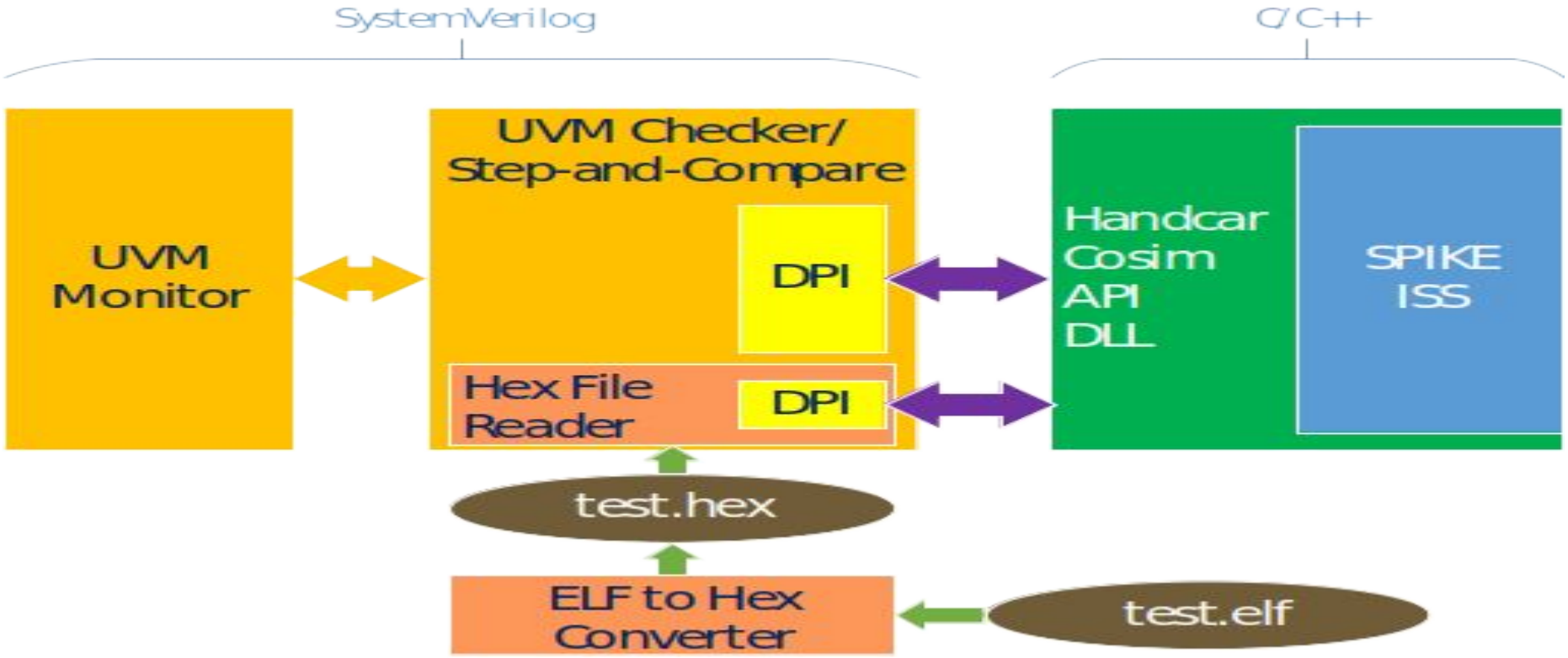
```yaml
# YAML file to specify a regression testlist
---
# Header
list: smoke
description: Basic smoke regression test

# List of builds
builds:
  uvmt_cv32:
    # required: the make command to create the build
    cmd: make corev-dv

# List of tests
tests:
  hello-world:
    # required is required, specifies either uvmt_cv32 or core
    build: uvmt_cv32
    # required: human-readable description of the test
    description: UVM Hello World Test
    # require: make command for the test
    cmd: make hello-world
    # optional: number of simulations to run defaults to 1
    num: 2
    # optional: defaults to empty, skip a simulator (dsim, xrun, vsim, vcs)
    skip_sim: [xrun, vcs]

  riscv_break_test_0:
    build: uvmt_cv32
    description: Static riscv-dv break test
    cmd: make custom CUSTOM_PROG=riscv_break_test_0
    # Set number of runs and fixed seeds
    num: 3
```

Modular Makefiles

- Common Infrastructure across all cores
- Core-specific Repository control
- Simulator-specific make targets and rules.

```
CORE-V-VERIF/
     |
     +--- mk/
     |     +--- Common.mk                    # Common variables and targets
     |     +--- uvmt/
     |           +--- uvmt.mk                # Simulation makefile (includes ../Common.mk and simulator-specific
     |           +--- vcs.mk                 # Synopsys VCS
     |           +--- vsim.mk                # Mentor Questa
     |           +--- dsim.mk                # Metrics dsim
     |           +--- xrun.mk                # Cadance Xcelium
     |           +--- riviera.mk             # Aldec Riviera-PRO
     |           +--- <other_simulators>.mk
     +--- cva6/
           +--- sim/
                 +--- ExternalRepos.mk       # URLs, hashes to external repos (RTL, riscv-dv, etc.)
                 +--- uvmt/
                       +--- Makefile         # includes ../ExternalRepos.mk, ../../../mk/uvmt/uvmt.mk
```

Handcar API for Spike



ISA Functional Coverage

- A unified coverage agent has been developed for RV32:
  - Used by all CV32E4-class cores.
- See $(CORE-V-VERIF)/lib/uvm_agents/uvma_isacov/cov/uvma_isacov_cov_model.sv

What Does “CVA6 Verification Complete” mean?

- The time to define this is *now* - before detailed verification starts.

Category	Item	Sign-off Criteria	Owner	Status
Verification Planning	Top-level Verification Plan	Top-level plan is complete	mike@openhwgroup.org	This is complete. It is the so-called Verification Strategy in core-v-docs.
	Per-functional Category Verification Plan	Completed, reviewed and up-issued per review	mike@openhwgroup.org	Detailed status available in core-v-docs.
	Testbench Cross-reference	Each item in Vplan cross-ref’ed to testcase and/or coverage	mike@openhwgroup.org	75%: - debug Vplan cross-referenced - ISA Vplans cross-referenced - need to cross-reference interrupts
Code Coverage	Block/Statement Coverage	All Blocks covered	steve.richmond@silabs.com	99.84% (Missing 2 lines of illegal instruction decode of the p.clip PULP instruction in non-PULP mode
	100% Conditional Coverage	All conditions covered	steve.richmond@silabs.com	100%
Functional Coverage	All cover-points attributed to a Vplan item	Human review	mike@openhwgroup.org	75% (see above)
	All cover properties attributed to a Vplan item	Human review	mike@openhwgroup.org	75%
	Cover-point coverage	100%	steve.richmond@silabs.com	Holes related to instruction crosses
	Cover-property coverage	100%	steve.richmond@silabs.com	100%%
Regression	Compliance Test Suite	All RISC-V Compliance tests run and passing	mike@openhwgroup.org	Complete.
	CORE-specific Test Suite	All tests run and passing	mike@openhwgroup.org	Largely complete: see previous slides on regression failures

OPENHW GROUP

PROVEN PROCESSOR IP

Thank You