Compliance Task Group Call – Minutes

Mon, Dec09 2019 2pm Pacific →Standard← Time

Charter

The Compliance Task Group will

- Develop a framework for RISC-V tests, taking into account approved specifications for:
  - Architectural versions (e.g. RV32I, RV32E, RV64I, RV128I)
  - Standard Extensions (M,A,F,D,Q,L,C,B,J,T,P,V,N)
  - All spec'ed implementation options
    - (incl. MHSU modes, optional CSRs, optional CSR bits)
- Develop a method for selecting and configuring appropriate tests for a RISC-V implementation, taking into account:
  - Platform profile and Execution Environment (EE)
  - Implemented architecture, extensions, and options
- Develop a method to apply the appropriate tests to an implementation and verify that it meets the standard
  - test result signature stored in memory will be compared to a golden model result signature

Adminstrative Pointers

- Chair – Allen Baum allen.baum@esperantotech.com
- Co-chair – Stuart Hoad stuart.hoad@microchip.com
- TG Email tech-compliance@lists.riscv.org
  - Notetakers: please send emails to allen.baum@esperantotech.com
- Meetings -Bi-monthly at 8am Pacific time on 2nd/4th Wednesdays
  - Location is https://zoom.us/j/6213886723
- Documents, calendar, roster, etc. in https://lists.riscv.org/tech-compliance/
  see /documents, /calendars subdirectories
  - https://riscof.readthedocs.io/en/latest/ riscof
  - https://riscv-config.readthedocs.io/en/latest/ config: YAML and WARL spec
- Git repositories
  - https://github.com/riscv/riscv-compliance/
  - https://github.com/rsnikhil/Experimental_RISCV_Feature_Model
  - https://github.com/rsnikhil/Forvis_RISCV-ISA-Spec
  - https://gitlab.com/incoresemi/riscof (Shakti framework)

Attendees

- Allen Baum (Esperanto)

No attendance taken at summit

Meeting Agenda (in order of Priority)

1. State of Compliance (see slide 6)
2. Summit Compliance Announcement (see slide 9)
3. Imperas Compliance status
    1. Rv32i coverage
    2. B-extension test porting and coverage
    3. V-extension tests and coverage & how it stresses the framework
    4. Collaboration with Google on functional coverage
4. RISCOF review:
5. Pull requests (if time permits)
    - 65: Test Format spec: first half reviewed

State of Compliance

- The de-facto compliance framework is the existing GIT repository
- There have been significant updates to this lately
  - a separate config repository that enables precise description of implemented options, including more precise WARL definition
  - more model support
  - more coverage support
- This will be formalized as a v.1 compliance framework
- V.2 is under review
  - updated test spec format, designed to be easier to write compliance tests (under review)
  - updated framework (riscof) (under review)
    - implements the updated test spec format,
    - specifically enables dynamic signatures, instead of static or self checking
  - formalizes WARL description
    - Limits illegal→legal mappings, making formal models and tests easier to write

Discussion

1. Compliance announcement
2. Imperas Coverage status

   See separate slides

3. General comments

   We need more maintainers/reviewers for the repository

   The compliance TG will need to morph into a standing committee after test spec and final framework are ratified in order to add tests for new extensions, new features, and to cover holes found in coverage, and fix spec errata

   There were questions about how to fund additional specs; can we write tools for tests? Leverage the formal spec(s)?

   Coverage should include corner cases; the described coverage does not list those (although proposed coverage draft on slide 12 mentions those). Needs more rigor, in general.

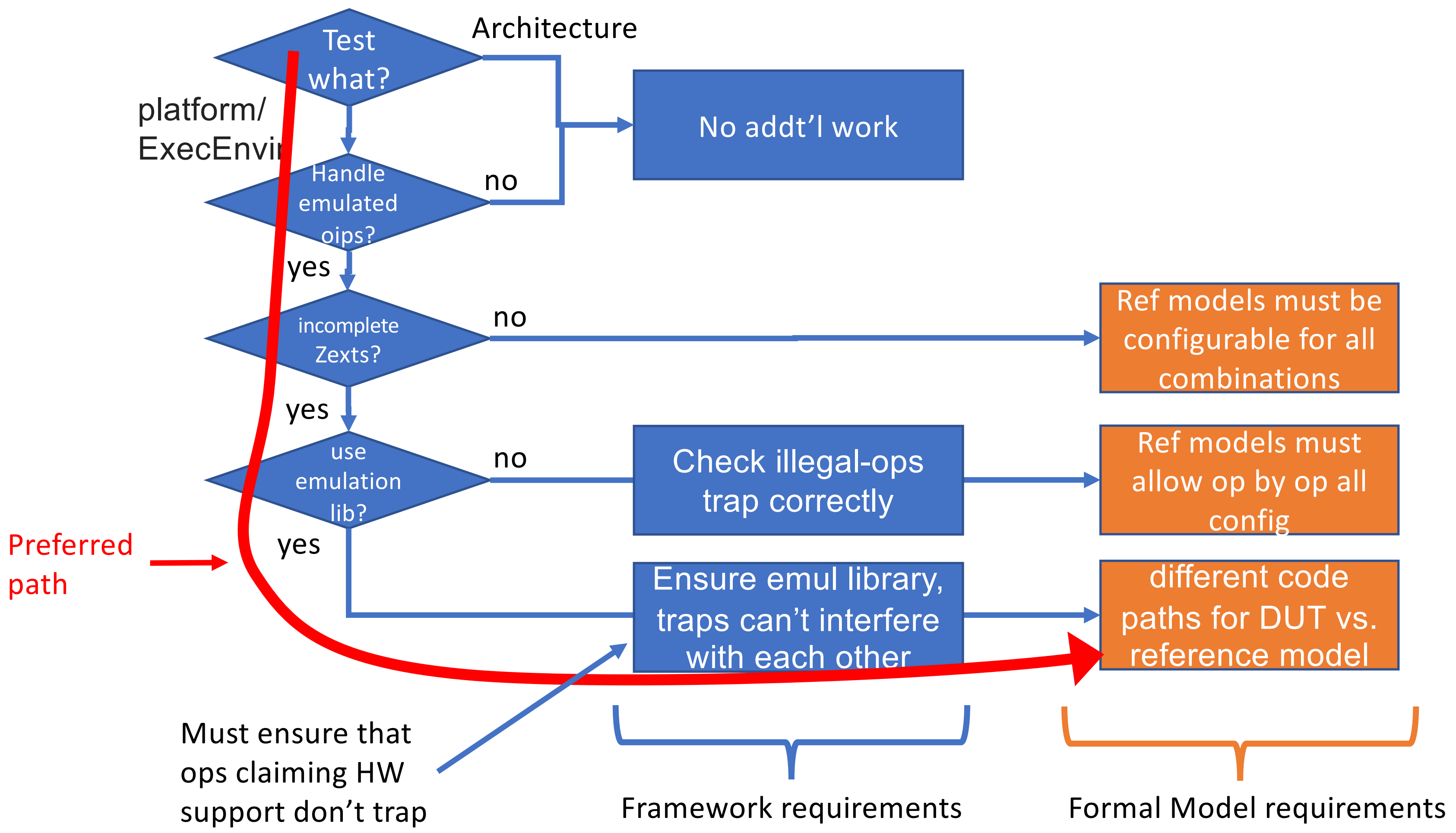
(There were more comments that are not captured here but are in the audio capture)

Decisions & Action Items

Decisions

Action Items

Emulated Op Handling – upcoming decisions


Test what?
Architecture
No addt'l work
platform/ ExecEnvir
Handle emulated oips?
no
yes
incomplete Zexts?
no
Ref models must be configurable for all combinations
yes
use emulation lib?
no
Check illegal-ops trap correctly
Ref models must allow op by op all config
yes
Preferred path
Ensure emul library, traps can't interfere with each other
different code paths for DUT vs. reference model
Must ensure that ops claiming HW support don't trap
Framework requirements
Formal Model requirements

Testing RISC-V Architectural Compliance

- A v0.1 of the RISC-V Compliance Framework is now available here:
  https://github.com/riscv/riscv-compliance/
  - Compares arbitrary models against a reference signature
    - e.g. your implementation,RISC-V CSAIL formal model, ovpsim, spike, etc.
  - Currently covers RV32IMC and RV64IMC (unprivileged spec) only (see
    - With nearly 100% coverage for RV32
  - Work on V.2 is underway
    - New test specification with Improved macro support for writing tests
    - Can be configured to match much larger choice of architectural options
    - E.g. priv. levels, extensions, HW unaligned access support, CSRs and WARL field implementation
    - Will allow comparison of two arbitrary models, e.g. formal model vs. your implementation

The RISC-V spec allows many (architectural) implementation choices, so

- A new repository has been created to describe implementation configurations that the Framework will use to select & configure tests:
  https://riscv-config.readthedocs.io/en/latest/

Update for Compliance Working Group RISC-V Summit, December 2019

Simon Davidmann, Lee Moore

simond@imperas.com, moore@imperas.com

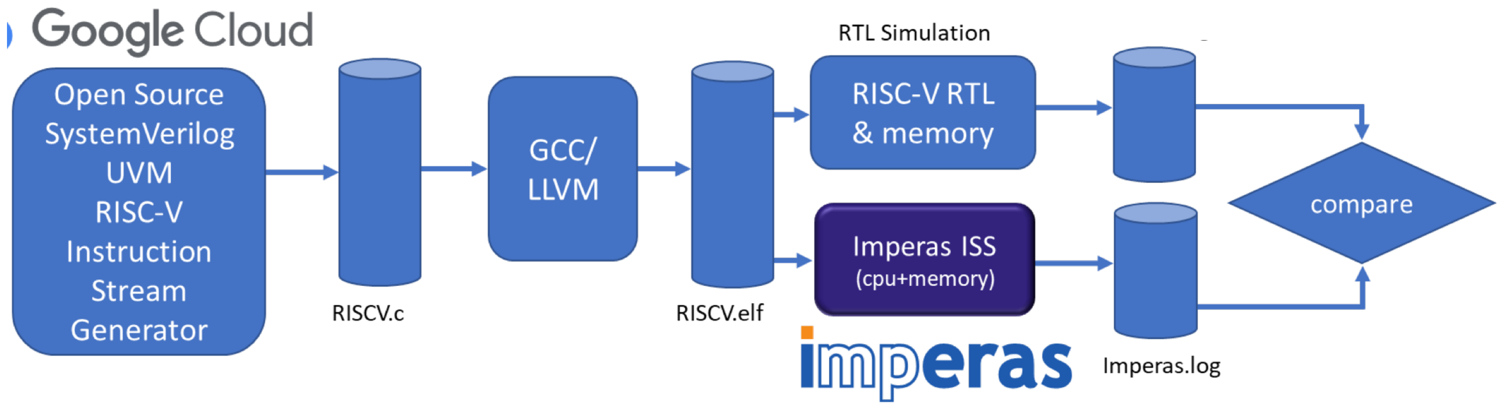
Agenda – Imperas update

- Functional Coverage of RISC-V Base Instructions
- Update to RV32I compliance Suite
- Compliance suite for RISC-V Vector Instruction Base Extension
- Imperas riscvOVPsim – reference model for base, vector & bitmanip instructions & functionality – free on GitHub

Background - Imperas

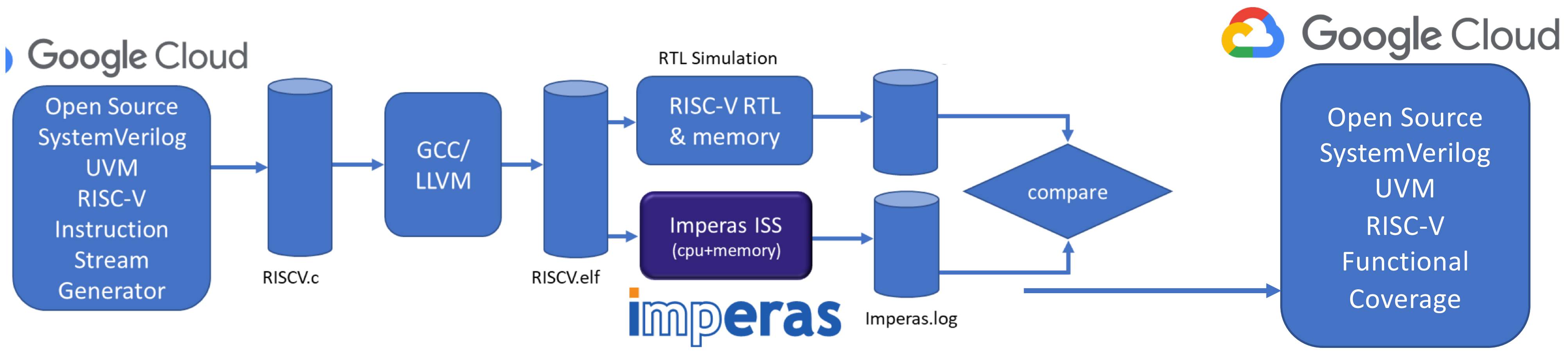
- Provide FREE full envelope model and reference simulator – riscvOVPsim
  - Commercial tool implementing full RISC-V spec
    - Controllable to be precise design choices inc. 32/64 IMAFDC B V
  - As used in RISCV Foundation's Working Groups: Compliance, BitManip
  - Open Source (Apache 2.0) OVP model, available on GitHub
- Models of commercial IP/Silicon
  - Andes N22,N25,NX25,N25F,NX25F,A25,A25F,A25MP,AX25,AX25F,AX25MP, NX27V (new, vectors)
  - MicroSemi CoreRISCV, MiVRV32IMA
  - lowRISC 32EC, 32EMC, 32IC, 32IMC
  - SiFive E20,E21,E24,E31,E34,E51,E76,E75MC,S21,S51,S54,S76,S76MC,U54, U54MC,U74,U74MC
- Example extendable virtual platform kits as source
  - Microsemi, Andes, SiFive
  - Running FreeRTOS, Linux, SMP-Linux, …
- Full professional multicore debugger, advanced Verification, Analysis, Profiling tools
- Methodology and tools to add, verify, and debug custom instructions
- Working in RISC-V Foundation helping drive RISC-V Compliance
- Member of CHIPS Alliance, and OpenHW Group to evolve open source cores

Functional Coverage of RISC-V Instructions



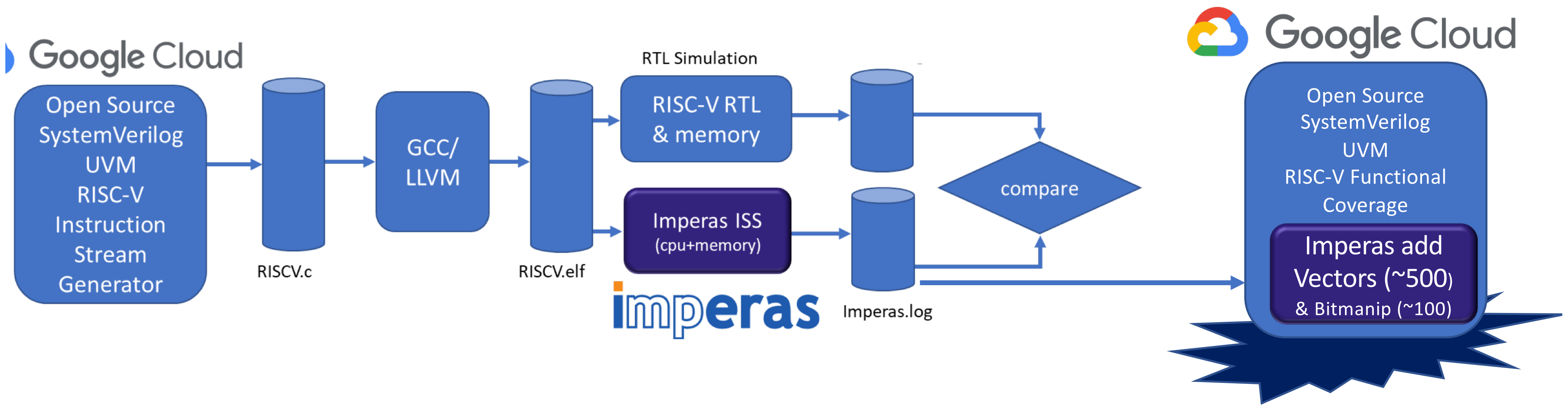
- Presented at DVCon in Munich, Oct 2019
- Google Cloud developed SystemVerilog UVM RISC-V Instruction Stream Generator
- Uses Imperas ISS as reference to compare with RTL trace

Functional Coverage of RISC-V Instructions



- Google have added SystemVerilog UVM Functional Coverage
- Uses SystemVerilog UVM covergroups and tools

Functional Coverage of RISC-V Instructions


Google Cloud
Open Source SystemVerilog UVM RISC-V Instruction Stream Generator
RISCV.c
GCC/ LLVM
RISCV.elf
RTL Simulation
RISC-V RTL & memory
Imperas ISS (cpu+memory)
imperas
Imperas.log
compare
Google Cloud
Open Source SystemVerilog UVM RISC-V Functional Coverage
Imperas add Vectors (~500) & Bitmanip (~100)

- <u>Imperas have added Vector & Bitmanip extension instructions to the Functional Coverage</u>

Update to Compliance Suites

- First thing we did – measure coverage of existing tests
- Then developed directed test generator to create new tests in correct shape
- Initial focus was on RV32I – developed new suite, then on RV64V
- Current status:

• RV32I	(NEW tests) coverage:	97.23% (48 tests)
• RV32M		87.76% (8 tests)
• RV32C		61.25% (25 tests)
• RV64I		51.29% (9 tests)
• RV64M		69.52% (4 tests)
• RV64B	(NEW tests) coverage:	54.12% (100 tests) [work in progress: Symbioticeda]
• RV64V	(NEW tests) coverage:	99.84% (1,512 tests, 70 instructions out of ~500) [expect there to be 10,000 tests…]

Example Cover groups

```
covergroup add_cg with function sample(riscv_instr_cov_item instr) {
        cp_rs1        : coverpoint instr.rs1;
        cp_rs2        : coverpoint instr.rs2;
        cp_rd         : coverpoint instr.rd;
        cp_rs1_sign   : coverpoint instr.rs1_sign;
        cp_rs2_sign   : coverpoint instr.rs2_sign;
        cp_rd_sign    : coverpoint instr.rd_sign;
}
```

Others:

```
    cp_imm_sign    : coverpoint instr.imm_sign;
    cp_branch_hit  : coverpoint instr.branch_hit;
    cp_sign_cross: cross cp_rs1_sign, cp_rs2_sign, cp_rd_sign;
```

- Results of coverage runs are in the GitHub repo next to the test suite src and signatures

Compliance Suite for Vector Instructions

- Developing test suite to ensure models, cores compliant
  - Funded by customers
  - Currently in use to validate customer's RTL and partner's IP
  - Uses Imperas riscvOVPsim as golden reference model
- Uses existing Compliance Working Group Framework for compliance
- Challenge:
  - VLEN,SLEN,ELEN means 90+ possible hardware configs
  - Multiplied by SEW, LMUL, VL means 16 x 0-VL (could be 2048) -> 100s -> 1,000s of variations
    - Where the instructions are polymorphic – i.e. results change based on these dynamic settings
  - => 1,000s of different configurations for the vector engine – needing checking…
  - So to test each instruction needs setting up the vector engine in the golden reference and then dynamically generating the reference result and comparing with the DUT
- Status:
  - Creating configurable compliance test generator to generate tests with known expected results
  - (Nov 2019) Completed: 70 integer arithmetic tests, one HW config, polymorphic over sew, lmul, etc. = 1,512 tests (over 2M ins)
  - Tested for coverage and gets over 99% functional coverage
- Plan:
  - Ongoing work in progress to increase to include all vector instructions
  - Plan to open source as part of Foundation's compliance suites
    - Will need change to Framework to allow test signatures to be dynamic (currently static)

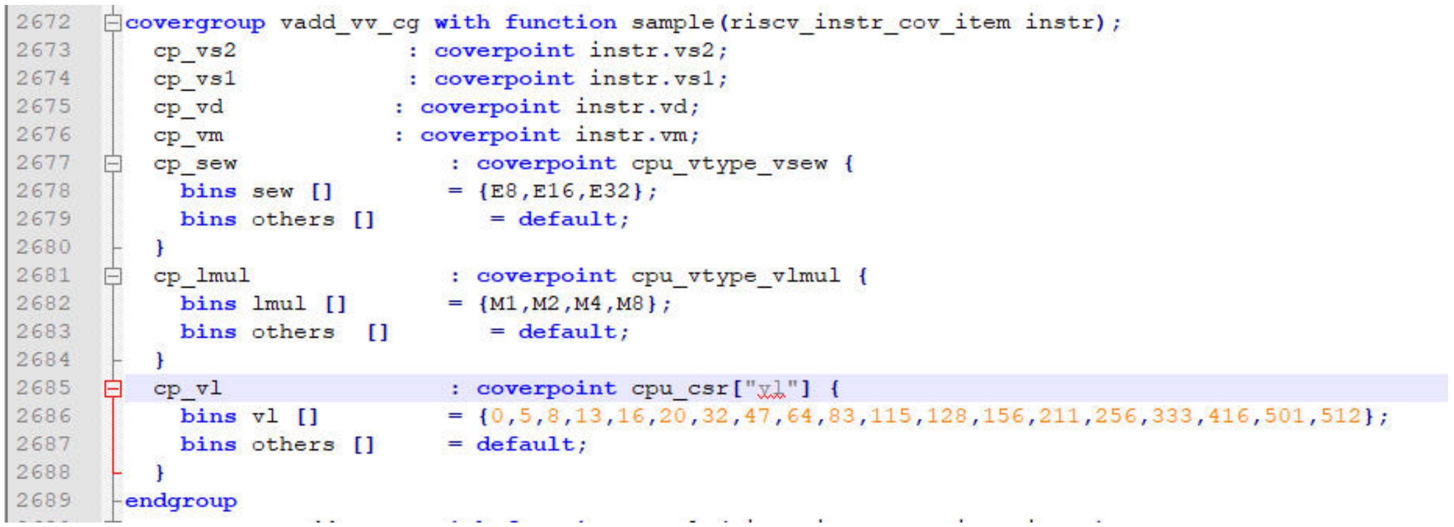
Imperas vector simulator - riscvOVPsim

- Press Release: "Imperas delivers first RISC-V Simulator for new Vector and Bit Manipulation specifications to Lead Customers" – June 2019
- Implements full vector specification (except Zediv)
- (almost) Every week we update simulator to reflect changes that have happened in the specification in last week
- riscvOVPsim – available free from: https://github.com/riscv/riscv-ovpsim
    - Configurable so can select any version of spec on simulator command line for example:
        - 0.7.1-draft-2019060
        - 0.8-draft-20190906
        - 0.8-draft-20191004
        - 0.8-draft-20191117
        - 0.8-draft-20191118
        - Master – always the latest (currently reflects 26 Nov 2019 (commit 6701109))
    - Documentation lists details of each version
    - Changelog is good place to see what has been updated
        - https://github.com/riscv/riscv-ovpsim/blob/master/ChangeLog.md

We need help…

- What should and should not be in the Vector compliance suite?
  - For example: should it be exercising the illegals settings/configurations?
- What functional coverage (coverpoints) should we be measuring?
- …
- Pls send us your thoughts: simond@imperas.com, moore@imperas.com

Coverage group (one vector instruction)



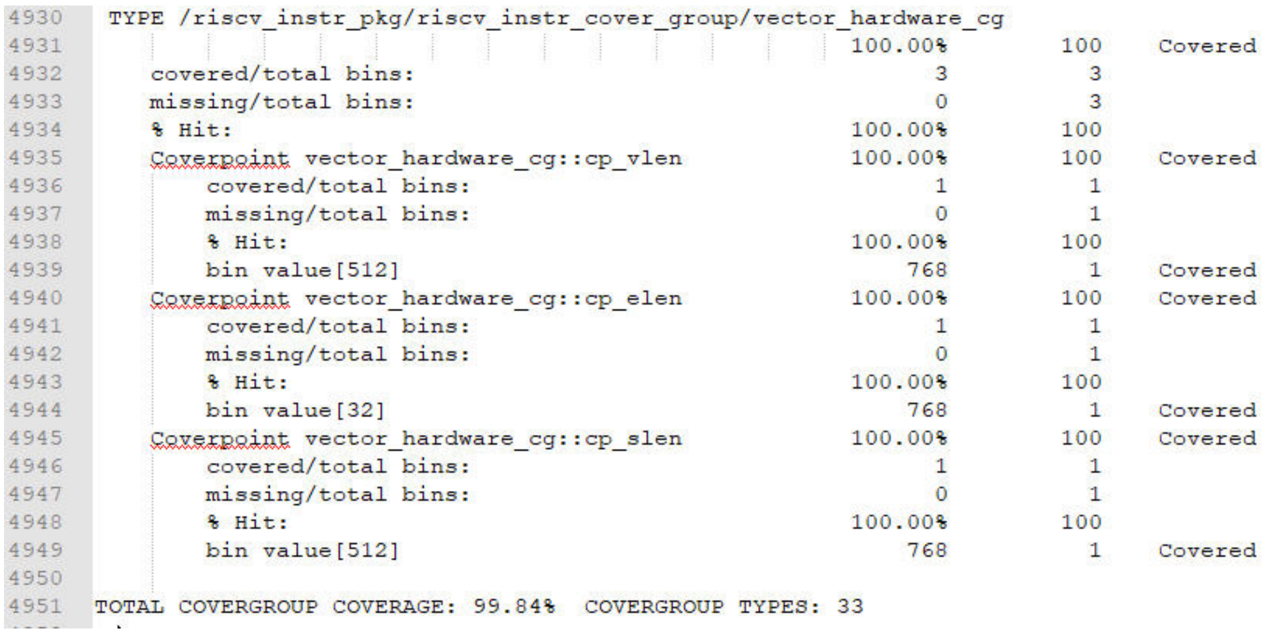
```
covergroup vadd_vv_cg with function sample(riscv_instr_cov_item instr);
  cp_vs2              : coverpoint instr.vs2;
  cp_vs1              : coverpoint instr.vs1;
  cp_vd             : coverpoint instr.vd;
  cp_vm             : coverpoint instr.vm;
  cp_sew                  : coverpoint cpu_vtype_vsew {
    bins sew []           = {E8,E16,E32};
    bins others []           = default;
  }
  cp_lmul                 : coverpoint cpu_vtype_vlmul {
    bins lmul []          = {M1,M2,M4,M8};
    bins others  []          = default;
  }
  cp_vl                   : coverpoint cpu_csr["vl"] {
    bins vl []            = {0,5,8,13,16,20,32,47,64,83,115,128,156,211,256,333,416,501,512};
    bins others []        = default;
  }
endgroup
```

- Complex cross coverage still needs to be added
  - Between sew, lmul, vm, and vl

Current instructions tested

- vmv_v_v,vsaddu_vi,vnmsub_vx,vxor_vx,vand_vv,vdivu_vv,vdivu_vx,vmul_vv,
- vrem_vv,vand_vi,vmxnor_mm,vsaddu_vv,vmulhsu_vv,vdiv_vx,vadd_vi,
- vmacc_vx,vmax_vv,vmin_vx,vmul_vx,vssubu_vv,vdiv_vv,vmax_vx,
- vmulhu_vx,vmxor_mm,vadd_vx,vmin_vv,vmulhsu_vx,vor_vx,
- vmand_mm,vsub_vx,vsub_vv,vmaxu_vx,vmor_mm,vrem_vx,
- vmv_v_i,vssubu_vx,vand_vx,vmnor_mm,vssub_vv,vxor_vi,vmornot_mm,
- vadd_vv,vsadd_vi,vmulhu_vv,vmerge_vim,vmadd_vx,
- vnmsac_vx,vmaxu_vv,vnmsub_vv,vxor_vv,vmandnot_mm,vmerge_vxm,
- vmulh_vv,vmv_v_x,vssub_vx,vor_vv,vnmsac_vv,vmacc_vv,vminu_vv,
- vsaddu_vx,vmerge_vvm,vsadd_vv,vsadd_vx,vmnand_mm,vminu_vx,
- vremu_vx,vremu_vv,vmulh_vx,vmadd_vv,vor_vi

Coverage Results



```
TYPE /riscv_instr_pkg/riscv_instr_cover_group/vector_hardware_cg
                                                     100.00%        100     Covered
   covered/total bins:                                     3          3
   missing/total bins:                                     0          3
   % Hit:                                            100.00%        100
   Coverpoint vector_hardware_cg::cp_vlen            100.00%        100     Covered
       covered/total bins:                                 1          1
       missing/total bins:                                 0          1
       % Hit:                                        100.00%        100
       bin value[512]                                    768          1     Covered
   Coverpoint vector_hardware_cg::cp_elen            100.00%        100     Covered
       covered/total bins:                                 1          1
       missing/total bins:                                 0          1
       % Hit:                                        100.00%        100
       bin value[32]                                     768          1     Covered
   Coverpoint vector_hardware_cg::cp_slen            100.00%        100     Covered
       covered/total bins:                                 1          1
       missing/total bins:                                 0          1
       % Hit:                                        100.00%        100
       bin value[512]                                    768          1     Covered

TOTAL COVERGROUP COVERAGE: 99.84%  COVERGROUP TYPES: 33
```

Backup from previous discussions

Future Discussions

1. Coverage metric (slide 12)
2. WARL definition – (slides 13-16)
   https://riscv-config.readthedocs.io/en/latest/yaml-specs.html#warl-field-restriction-proposal
3. Email discussion#5 nonconforming extension support

Draft Test Coverage Proposal (unpriv)

Classes of things we want to test for

- Decode
  - Immediate – test all bits in either polarity will affect output
  - Register specifiers – test that changing any bit will affect output, ensure all regs are tested
  - Variations – test values of opcodes suffixes that have any string after a "." in its opcode
- Register combinations
  - Destructive (dest = either src) and non-destructive
  - Non-updating (i.e., targeting X0), or non-supplying (X0 as an input)
  - All registers (or immediate bit) should be used per instruction *<u>category</u>*
- Special and exception cases
  - Explicitly defined (e.g. shifts>=XLEN & RD=X0
  - Implicitly defined – corner cases
    - Maximal and minimal inputs, or creating maximal outputs
    - Inputs that special case outputs (mostly FP cases, also. shiftamt>=XLEN)
    - Outputs crossing value boundary (e.g. address cross word/page/superpage/VA boundary, FP crossing exponent boundary)

proposed
<u>coverage & categories</u>

Arith[I],	*W1/0,crys*
Logical[I],	*W1/0*
Shift[I],	*W1/0/msk,+*
Auipc,Lui,	
Ld,St,	*W1/0, bndXing*
Br,	*W1/0, bndXing*
Jmp ,	*W1/0, bndXing*
Ebreak/ Ecall	

W1/0= walking 1/0
BndXing=: boundary crossing

This works for 32i base ops – what do we need to add for priv modes? Mem model? Sequential Dependencies? Other extensions?

Need a review of existing (non-RISC-V) compliance specs

RISCV-CONFIG

- Examples & definitions
  - https://github.com/riscv/riscv-config/tree/master/examples
  - https://github.com/riscv/riscv-config/tree/master/riscv_config/schemas
  - https://github.com/riscv/riscv-compliance/tree/master/riscv-ovpsim/config-yaml/examples
- Validator
  - https://github.com/riscv/riscv-config/blob/master/riscv_config/checker.py
- Example integration of converter (OVPsim)
  - https://github.com/riscv/riscv-compliance/tree/master/riscv-ovpsim/config-yaml
- WARL, YAML
  - https://riscv-config.readthedocs.io/en/latest/

RISCV-CONFIG WARL Syntax

WARL: {optional items in curly braces}

- dependency_fields: [list] – use this when legal/illegal values depend on other fields (in list)
- legal: [<warl-string>{,<warl-string>*}]
- wr_illegal: [<warl-string>{,<warl-string>*}] -> update_mode

where <warl-string> is either "&" separated list of rangehi:rangelo lists

{*[dependency_value] ->*} field-name1[bit#hi:bit#lo] in [legal-range-list]

*{ & field-name2[bit#hi:bit#lo] in [legal-range] }**

or "&" separated list of bitmasks

{*[dependency_value] ->*} field-name1[bit#hi:bit#lo] bitmask [mask, fixval]

{ & field-name2[bit#hi:bit#lo] bitmask [mask, fixval] }*

(can't mix ranges and bitmasks)

RISCV-CONFIG WARL Example1

```
#        When base of mtvec depends on the mode field.
WARL:
 dependency_fields: [mtvec::mode]
 legal:
  - "[0] -> base[29:0] in [0x20000000, 0x20004000]”                  # can take only 2 fixed values when mode==0.
  - "[1] -> base[29:6] in [0x00000:0xF00000] & base[5:0] in [0x00]" # 256 byte aligned when mode==1
 wr_illegal:
  - "[0] -> unchanged”
  - "[1] wr_val in [0x2000000:0x4000000] -> 0x2000000”             # predefined value if write value is in this range
  - "[1] wr_val in [0x4000001:0x3FFFFFFF] -> unchanged”            # predefined value  if write value is this range

#       When base of mtvec depends on the mode field. Using bitmask instead of range
WARL:
 dependency_fields: [mtvec::mode]
 legal:
  - "[0] -> base[29:0] in         [0x20000000, 0x20004000]"         # can take only 2 fixed values when mode==0.
  - "[1] -> base[29:0] bitmask [0x3FFFFFC0,  0x00000000]”          # 256 byte aligned when mode==1
 wr_illegal:
  - "[0] -> unchanged”                                              # no illegal for bitmask defined legal strings.

„
```

RISCV-CONFIG WARL Example2

```
# no dependencies. Mode field of mtvec can take only 2 legal values using range-descriptor
WARL:
 dependency_fields:
 legal:
  - "mode[1:0] in [0x0:0x1]                          # Range of 0 to 1 (inclusive)"
 wr_illegal:                                        # default to 0 if not a legal value
  - "0x00"

# no dependencies. using single-value-descriptors
WARL:
 dependency_fields:
 legal:
  - "mode[1:0] in [0x0,0x1]                          # also Range of 0 to 1 (inclusive)"
 wr_illegal:
  - "0x00”
```

- "[1] wr_val in [0x2000000:0x4000000] -> 0x2000000 & wr_val in [0x4000001:0x3FFFFFFF] -> **unchanged**