Compliance Task Group Call – Minutes

Weds, June 12, 2019 8am Pacific →Daylight← Time

(summaries on slides 5,6,7)

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 membership- Sue Leininger sue@riscv.org
  - Send email to her - you must have a lists.riscv.org login
- TG Email tech-compliance@lists.riscv.org
  - Notetakers: please send emails to allen.baum@esperantotech.com
- Meetings -Bi-monthly at 9am 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
- 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 Technologies)
- Simon Davidmann (Imperas)
- Deborah Soung (Sifive)
- Grant Martin (Cadence)
- Iilja Stepanov (Syntacore)
- Josh Scheid (Ventana)
- Marek Masarik (Codasip)
- Milan Skala (Codasip)
- Premsyl Vaclavik (Codasip)
- Neel Gals (IIT)
- S Pawan Kumar (IIT)

Meeting Agenda (in order of Priority)

1. Next Steps (Priorities) (slides 9..14)
2. Pull Request review;
   - See new TestSpecFormat (dir structure, new macros, allowing absolute addr) (slides 13,16-19)
   - Other "To Be Discussed" sections should be reviewed
3. RISCOV Framework presentation (includes WARL configuration)
4. Test Coverage discussion (see slide 14)
   - Questions from last meeting: SOC vs architecture coverage
     - e.g. how do have adequate coverage for platform limited features (memory map)
     - ARM includes memory maps in AVS tests
     - How did Microchip deal with this?
   - Testing eBreak without ending test: do we need to? (eBreak expects special noop before/after inst for debug)
   - Instruction categories & coverage types (e.g. all,walking0/1,)

Discussion

Pulls (no discussion)

1. Next Steps – Priorities
   - Current makefile framework, test suites, OVPSIM OK for beta
   - Simon will provide video and simple docs on how to use it
     - Biggest issue is having a precompiled toolchain
   - What does “release” mean?
     - When all pieces in place, declare 45 day comment period, after which board can declare it “official”
     - Official means that developers can use it to apply for trademark use
   - 1.0 release will have rv32/64i_, rv32/64i_m, rv32/64i_c, maybe a,f,d, but with limited coverage
     - Note that all existing tests in repository have been converted to signature style, but are not comprehensive
2. IIT Shakti project RISCOF framework presentation
   - YAML everywhere (device, platform, environment, and test configurations)
   - No GUI – may be one to generate configuration YAML file
   - has been run on rv32i[mc] tests, rv32i[afd] coming
     - Many tests rewritten to use proposed macros
     - Macros enable auto-generation of “Common compliance test pool reference document”
       - YAML format
   - WARL definition is being implemented
     - Merging of distinct/range types ongoing

Find a better name ! e.g. “Test Reference DB”

3. Test Spec
   - No disagreements voiced about removing prohibition of absolute addresses, test directory restructuring, new standard macros (see use in RISCOF)
4. Test Coverage – not covered (pun intended)

Conclusions & Action Items

Decisions

- We will shoot for a Beta release of rv32i[_,m,c] by end of Q3
- Change TestSpecFormat
  - Remove absolute address prohibition from test spec
  - add 3 new standard macros,
  - change to new test directory structure
  - Test conditions centralized in a common file
  - Many clarifications

Action Items

- Make video example of using current framework (Simon)
- Get updated test spec into repository (Allen)
- RISCOV respository is https://gitlab.com/incoresemi/riscof

Backup for discussions

Next Steps I: a beta release

Goal: By EO 3rd Quarter, have beta compliance test suite announced

What is it?

A well documented procedure for using an existing framework, existing tests, and a existing model to test an RV32i implementation

What do we need to get there?

1. Framework
    - A. Choose an existing framework: *e.g. existing makefile framework*
    - B. Cookbook for running compliance suite under framework (examples)\
        include sample code for user-provided macros under execution environment
2. Test Suite
    - Use the existing tests in the repository (so rv32i, rv32im, rv32ic
3. Golden model
    - Use OVPSIM for now, but provide hooks for others

Next Steps II: v1.00 release

Next Goal: V. 1.0 test suite announced

What do we need to get there?

1. Framework
    - A. Extend and document YAML spec
    - B. Cookbook for running compliance suite under framework (examples)
2. Test Suite
    - A. Tests (rv32/64i_, rv32/64i_m, rv32/64i_c,)
        - i. review and ratify Test Format Spec (pull request 57?)
        - ii. Re-write (some) tests to meet spec.
            - e.g self checking tests
        - iii. Re-organize repository to match spec
        - iv. Add more tests (to meet coverage)
    - B. Coverage spec
        - i. Define and ratify coverage spec
        - ii. Develop tool for measuring coverage according to spec
    - C. Common compliance test pool reference document ←find a shorter name!
        - i. define and ratify spec for its format (could be expanded version of TestFormatSpec Sec. 4.4)
        - ii. Develop tool for extracting basic info from tests
3. Golden model (not under our control)

1A., 1B., 2Ai. Framework, Cookbook

1A. presentation to follow (see also slide 19)

1B. Need a volunteer to update doc/README.adoc with a concrete example and a volunteer to validate that the steps work

2Ai. Everyone needs to read and comment on the draft (attached)

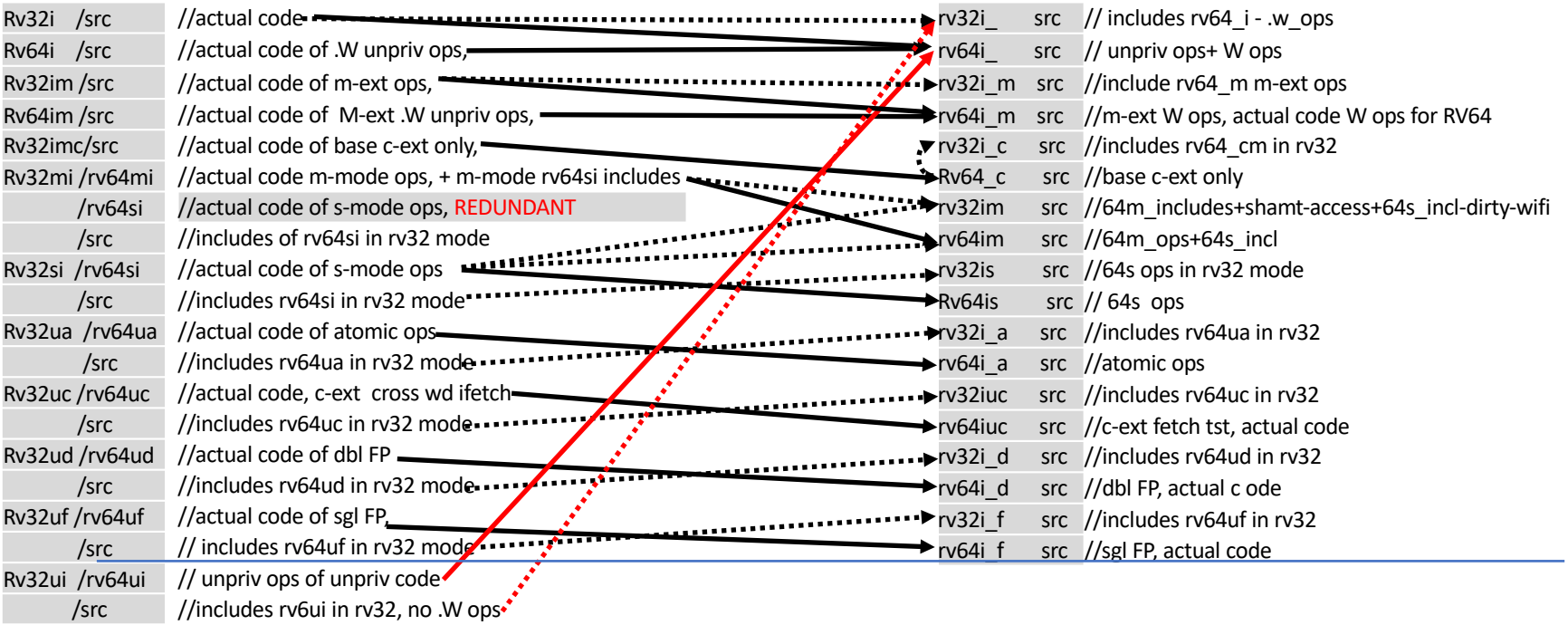
(see also issues in slides 17-18)

2Aii. Example Macro Use

```
RVTEST_IO_WRITE_STR(x31, "# Test part A1 - val 0 w/ 0, 1, -1, MIN, MAX reg values\n");

# Addresses for test data and results
la x1, test_A1_data
la x2, test_A1_res

# Load testdata
lw x3, 0(x1)

# Register initialization
li x4, 0
li x5, 1
li x6, -1
li x7, 0x7FFFFFFF
li x8, 0x80000000

# Test
add x4, x3, x4
add x5, x3, x5
add x6, x3, x6
add x7, x3, x7
add x8, x3, x8

# Store results
sw x3, 0(x2)
sw x4, 4(x2)
sw x5, 8(x2)
sw x6, 12(x2)
sw x7, 16(x2)
sw x8, 20(x2)

// Assert
RVTEST_IO_CHECK()
RVTEST_IO_ASSERT_GPR_EQ(x2, x3, 0x00000000)
RVTEST_IO_ASSERT_GPR_EQ(x2, x4, 0x00000000)
RVTEST_IO_ASSERT_GPR_EQ(x2, x5, 0x00000001)
RVTEST_IO_ASSERT_GPR_EQ(x2, x6, 0xFFFFFFFF)
RVTEST_IO_ASSERT_GPR_EQ(x2, x7, 0x7FFFFFFF)
RVTEST_IO_ASSERT_GPR_EQ(x2, x8, 0x80000000)

RVTEST_IO_WRITE_STR(x31, "# Test part A1 - Complete\n");
```

```
RVTEST_CASE("A1","val 0 w/ 0, 1, -1, MIN, MAX reg values\n", TRUE);

// Addresses for test data and results
RV_TEST_SIGBASE(x2, test_A1_res)
la x1, test_A1_data

lw x3, 0(x1)     // Load testdata

// Register initialization
li x4, 0
li x5, 1
li x6, -1
li x7, 0x7FFFFFFF
li x8, 0x80000000

// Test
add x4, x3, x4
add x5, x3, x5
add x6, x3, x6
add x7, x3, x7
add x8, x3, x8

RVTEST_IO_CHECK()       //store and assert results
RVTEST_SIGUPDATE(x2, x3, 0x00000000)
RVTEST_SIGUPDATE(x2, x4, 0x00000000)
RVTEST_SIGUPDATE(x2, x5, 0x00000001)
RVTEST_SIGUPDATE(x2, x6, 0xFFFFFFFF)
RVTEST_SIGUPDATE(x2, x7, 0x7FFFFFFF)
RVTEST_SIGUPDATE(x2, x8, 0x80000000)

RVTEST_IO_WRITE_STR(x31, "# Test part A1 - Complete\n");
```

2Aiii Test Suite Directory Reorg+Rename



What other collateral needs to be updated?

2Bi. Instruction Test Coverage

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 *category*
- 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 coverage & categories	
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	

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

2Bii. Tools

2Bii. Tool for measuring coverage according to spec

expanded version of Imperas tool?

2Ci. Common compliance test pool reference document spec

could be expanded version of TestFormatSpec Sec. 4.4

2Cii. tool for extracting basic info from test suit directory

a script that extracts strings in macros, or

a macro implementation that updates the docs when macro is run

Name (from repository)	Title	Description (from RVTEST_CASE macro)	Requirement
rv32i_-ADD- 01.S#A1	Instruction ADD test	RV32I Base Integer Instruction Set, Version 2.0	True
rv32i_-C.ADD-01.S	Compressed Instruction ADD test	RV32I Compressed Base Integer Instruction Set, Version 2.0	C_extension
rv32i_-MISALIGN_JMP-01.S	Compressed Instruction ADD test	RV32I Base Integer Instruction Set, Version 2.0	~C_extension
rv32i_m-MUL-01.S	Instruction MUL test	RV32I Base M-extension Instruction Set, Version 2.0	M_extension

Standard test structure & macros

to make writing tests easier

- RVTEST_SIGBASE(**sig_base**,val) – defines base register **sig_base** used to store signature values, initializes it, and clears **sig_offset**
  - (e.g .set SIG_BASE val; .set SIGOFF 0; la SIG_BASE, val)
  - Could this change during test? (e.g. generate a move from old to new base? Or just reset sig_offset and start using new base)?
- RVTEST_CASE([name], reason_str ,cond_str)
  - This updates the test_section variable & lays out requirements for each test section;
    - If empty name, then condition applies to all following test cases
  - Framework will remove the section if conditions unmet
  - Do we need Test_case_end? E.g. can Test_case_start end the previous part & start a new one
  - Also need a way to pass values of options into code – here or elsewhere?
- RVTEST_SIGUPDATE(srcreg [, assertion_val])
  - Store srcreg into **sig_offset**(**sig_base**) and increments **sig_offset**
    - ST srcreg, SIG_OFF(SIG_BASE); .set SIG_OFF SIG_OFF +XLEN/8
  - Optional assertion value; Framework uses it to build a signature file, and will insert checking/printing code for debug if separately enabled

RVTEST_SIGBASE() and RVTEST_SIGUPDATE each replace a single instruction, and perform framework housekeeping; RVTEST_CASE is used by framework and only performs housekeeping

Directory structure / Test naming

- Last meeting: decided to keep hierarchical directory
- Framework should use a unified database that lists each test and the conditions under which it should be executed
  - E.g. mode, extension supported address widths, CSRs implemented, etc
  - Slightly redundant with directory structure,testnames, but more flexible
- Directory format is RV[**32i** | **32e** | **64i**][**MSU_**]{**extension**}
  - **U** tests should not touch privileged resources, nor trap
  - _ tests can be executed in any mode with identical results
  - **M/S** tests may simply be U tests with an environment that causes a trap
  - Compressed ops should be located in the appropriate extension directory; database will label it to only be executed if C-ext is implemented& enabled

Absolute Addresses in tests

Compliance tests are intended to store a signature.
Self-checking test compares signature to a "correct" value, and branches to a success or failure label.

Spec requirement: Tests shall not store absolute addresses from the program in the signature.

This is somewhat in conflict with the directive that test not be self-checking:
Any test with PC dependent results needs to be specially written to avoid an absolute address.
E.g.: tests that check JAL[R] return address, or the TVAL or EPC of a trap.
Absolute addresses must be converted to relative address by subtracting
- an absolute address in the code (e.g. the entry point, data declaration), or
- a previously saved PC in a register (e.g. from an AUIPC)

This simply adds code to avoid branching to separate success/fail labels.

- what is the rationale for the prohibition on absolute addresses?
- what would break if we stored absolute addresses

My opinion:
- serves no useful purpose, makes tests longer, complicates development, hides some error classes.
- Possible downside: would not be possible to store universal signatures (for all platform configs),
(so always run test on both the golden model and the device under test and compare.

Is that a good enough reason give the drawbacks?

WARL field Restriction Proposal (updated)

2 Modes with variants:

- Bitmask(Mask, FixedVal)
  - Mask[n]==1 means bit[n] is RW; Mask==0 means bit[n]==FixedVal[n]
- Range(Mode{*see table*}, {LegalList = Rng,Rng…,Rng})
  - Where Rng = upper..lower, or value

BitMask: (WriteVal & Mask) |(FixedVal & ~Mask)

Range : if Write is not an element of LegalList, then depending on Mode either

Unchd: Current value
Largest: max(LegalList)
Smallest: min(LegalList)
NextUp: ceiling(LegalList) //next larger element of list or largest
NextDn: floor(LegalList) //next smaller element of list or smallest
NearUp: ceiling(LegalList) //pick closest element of list, ties go up
NearDn: floor(LegalList) //pick closest element of list, ties go down
BadAddr: out of range flips MSB of field

Range		Legal		Legal				Legal	
Mode	<1	1	2	3	4..5	6	7..8	9	>9
Largest	1	1	9	3	9	9	9	9	9
Smallest			1		1	1	1		
NextUp			3		9	9	9		
NextDn			1		3	3	3		
NearUp			3		3	9	9		
NearDn			1		3	3	9		
UnChgd	curr Val		curr Val		curr Val	curr Val	curr Val		curr Val
BadAddr	~MSB		~MSB +2		~MSB +4..5	~MSB +6	~MSB 7..8		~MSB +val

Note: These restrictions apply to fields that do not have architectural requirements, but that are completely implementation defined, i.e. their final value does not depend on any other state variable in the implementation.

Q: should subfields of a WARL field be independently defined to enable alignment restrictions?