RISC-V Architectural Functional Verification

David Harris[1], Jordan Carlin[1], Corey Hickson[1], Larry Lapides[2], Lee Moore[2], Huda Sajjad[3], Umer Shahid[3], Aimee Sutton[2], Mike Thompson[4], Rose Thompson[5], Muhammad Zain[6]

[1]Harvey Mudd College, [2]Synopsys, [3]10xEngineers, [4]OpenHW Foundation, [5]Oklahoma State University, [6]UET Lahore

https://github.com/openhwgroup/cvw-arch-verif/

Methodology

- Scope is architectural functional verification
  - Does not cover implementation specific features (caches, branch prediction, etc.)
- Coverage model draws on the Synopsys open-source riscvISACOV covergroup and sampling methodology
- Tests run on a Device Under Test (DUT) communicating with the ImperasDV reference model via the RISC-V Verification Interface (RVVI).
- Testbench collects functional coverage while the reference model checks that the DUT demonstrates correct behavior.

Lockstep Simulation with ImperasDV

- Tests run on DUT and ImperasDV reference model simultaneously
- ImperasDV and the DUT communicate over RVVI, checking that the DUT demonstrates correct behavior
- Lockstep eliminates the burden of generating self-checking tests and the risk of false-positives

Testing Flow

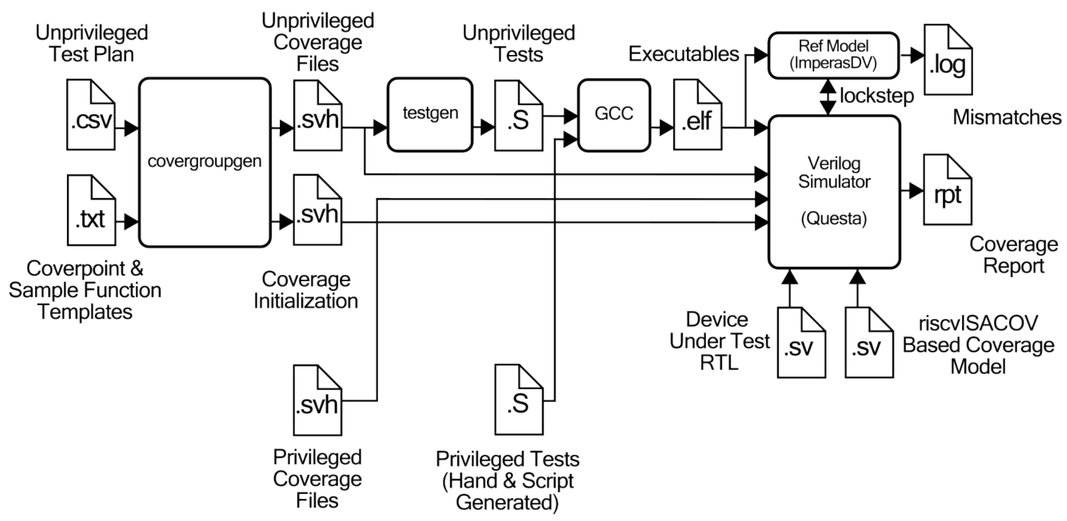

Fig. 1 Architectural functional verification flow

Test Plans

Instruction	Type	cp_asm_count	cp_rd	cp_rs1	cp_rs2	cp_rd_corners	cp_rs1_corners	cr_rs1_rs2_corners
lw	L	x	x	nx0		x		
sw	S	x		nx0	x			
addi	I	x	x	x		x	x	
add	R	x	x	x	x	x	x	x
sub	R	x	x	x	x	x	x	x
beq	B	x		x	x		x	x
jalr	JR	x	x	nx0				
lui	U	x	x			lui		

Fig. 2 Unprivileged test plan

coverpoint	covergroup	Sub Feature	Coverpoint Description
cp_instr_adr_misaligned_branch	exceptionsm	Instruction Address Misaligned	Branch taken to an address that is an odd multiple of 2 PC[1] = 0 and imm[1]=1
cp_instr_adr_misaligned_branch_nottaken	exceptionsm	Instruction Address Misaligned	Each type of branch not taken to an address that is an odd multiple of 2
cp_instr_adr_misaligned_jal	exceptionsm	Instruction Address Misaligned	jal to an address that is an odd multiple of 2 PC[1] = 0 and imm[1]=1
cp_instr_access_fault	exceptionsm	Instruction Access Fault	Instruction access fault is raised
cp_illegal_instruction	exceptionsm	Illegal Instruction	Executing instruction 0x00000000 and 0xFFFFFFFF throws an illegal instruction exception.

Fig. 3 Privileged test plan

Covergroups

```
covergroup I_add_cg with function sample(ins_t ins);
    option.per_instance = 0;
                                ...
    cp_rd : coverpoint ins.get_gpr_reg(ins.current.rd)  iff (ins.trap == 0 )  {
        //RD register assignment
    }
                                ...
    cp_rs1_corners : coverpoint unsigned'(ins.current.rs1_val)  iff (ins.trap == 0 )  {
        wildcard bins zero     = {0};
        wildcard bins one      = {32'b00000000000000000000000000000001};
        wildcard bins two      = {32'b00000000000000000000000000000010};
        wildcard bins min      = {32'b10000000000000000000000000000000};
        wildcard bins minp1    = {32'b10000000000000000000000000000001};
        wildcard bins max      = {32'b01111111111111111111111111111111};
        wildcard bins maxm1    = {32'b01111111111111111111111111111110};
        wildcard bins ones     = {32'b11111111111111111111111111111111};
        wildcard bins onesm1   = {32'b11111111111111111111111111111110};
        wildcard bins alteven  = {32'b10101010101010101010101010101010};
        wildcard bins altodd   = {32'b01010101010101010101010101010101};
        wildcard bins random   = {32'b01011011101111001000100001110111};
    }
                                ...
endgroup
```

- Unprivileged covergroups generated by covergroupgen.py script
- Privileged covergroups handwritten from English test plans

Tests

```
# Testcase cp_rd (Test destination rd = x1)
li x15, 0×2df19c49bf9d2b53 # initialize rs1 to a random value
li x3, 0×e21482b7cfc50d6b # initialize rs2 to a random value
add x1, x15, x3 # perform operation
RVTEST_SIGUPD(x5, x1)

# Testcase cp_rd (Test destination rd = x2)
li x24, 0×214606929e470dd7 # initialize rs1 to a random value
li x23, 0×2391fecd330ad053 # initialize rs2 to a random value
add x2, x24, x23 # perform operation
RVTEST_SIGUPD(x5, x2)

...

# Testcase cp_rd (Test destination rd = x31)
li x19, 0×ea92d50933697752 # initialize rs1 to a random value
li x22, 0×1d3e035ce1739ac5 # initialize rs2 to a random value
add x31, x19, x22 # perform operation
RVTEST_SIGUPD(x1, x31)
```

- Unprivileged tests generated by testgen.py script
- Privileged tests handwritten from English test plans

Results

Feature	Coverpoints	RV64 Test kLOC	Percent Coverage
		Unprivileged	
I	468	81	100%
M	252	38	100%
A	244	21	100%
Zc{a,b,d,f}	233	14	100%
F, D, Zf{h/a}	1332	2284	100%
Zb{a,b,c,s}	672	80	100%
Zkn	292	13	100%
Zicond	28	4	100%
Zicbo*	in progress	in progress	
		Privileged	
Zicsr	187	1.6	100%
Zicntr	39	1.9	100%
Exceptions	249	3	100%
Interrupts	187	4	100%
Endian	130	1.5	100%
PMP	In Progress	N/A	N/A
Virtual Mem	249	18	Sv32 & Sv39 at 100%

Fig. 4 Size of coverage files, lines of test code, and percentage of coverpoints covered by tests

- The tests, covergroups, and lockstep simulation methodology has been demonstrated on the OpenHW Foundation's CORE-V Wally core
- Lockstep simulation with ImperasDV has uncovered 9 bugs that were not detected by riscv-arch-test and other custom test suites
  - fround bad shift in some situations
  - fmv untested and produced garbage
  - certain illegal instructions and CSRs did not trap

Future Work

The teams intends to modify the testing flow to become compatible with the riscv-arch-test framework and to fully integrate the tests into the riscv-arch-test repository. Additionally, the teams plan to add covergroups and tests for the IBM floating point verification framework.