RISC-V Architectural Functional Verification

Hamza Jamal (TL), Ahlyssa Santillana (S), Corey Hickson (J), Jordan Carlin (J, Sp),
Marina Bellido Rodriguez (J, Sp), Roman De Santos (J, Sp), Vikram Krishna (J, Fa)

Advisor: Prof. David Harris
Liaisons: Umer Shahid, Huda Sajjad (Fall), Fatima Saleem

Table of Contents

Project Statement

“The 2024-2025 HMC 10xEngineers Clinic team will develop a suite of RISC-V architectural functional verification test plans, coverpoints, and tests. This includes maintaining a complete list of features to be tested, creating covergroups that adhere to these test plans, and writing the tests to exercise all coverpoints in the defined covergroups. This project will develop open source coverage for all architectural features.”

Project Description

- Collaboration between 10xEngineers, Habib University, UET Lahore, and HMC
- Developing an open-source RISC-V Architectural Functional Verification model with accompanying test suites
- Demonstrating these tools on CORE-V Wally, which is attempting to reach Technology Readiness Level 5
    - 100% functional coverage
    - 100% code coverage

RISC-V Overview

- ISA (instruction set architecture)
  - ISA defines computer core using functions (called “instructions”)
  - E.g. `add rd, rs1, rs2` for adding 2 integers *** describe priv/unpriv
- RISC-V ISA comes in 2 parts
  - Unprivileged: arithmetic, memory operations, branches, jumps
  - Privileged: exceptions, interrupts, access control
- Highly configurable

RISC-V Overview

- Open-source architecture
  - Completely free
    - No licensing fees
    - No usage restrictions
  - 2 major proprietary architectures dominate market (ARM, x86)
    - ARM: Mobile, Desktops
    - x86: Desktops/Servers
  - RISC-V can be competitive in both spaces

Why Do We Verify?

- We want to catch bugs in the design process, not after production
    - Pentium FDIV bug (1994)

- Chip tapeout is expensive and time-consuming
    - 3nm mask costs upwards of $40M
    - Fabrication takes 1-3 months

- Simulate chip functionality before production
    - Test development is expensive, but more reusable
    - Still requires ~50% of total project budget

Impact of Our Work

- No free, comprehensive verification suite for RISC-V currently exists
  - Proprietary suites are available, but costs can be prohibitive
  - Official riscv-arch-test suite is still insufficient for most applications
  - Verification costs are largest bottleneck for development

- Wally is a viable core for industry and academia
  - Accompanied by a very detailed textbook
  - Can be used in commercial applications

Overview of Functional Verification

Verification Methodology

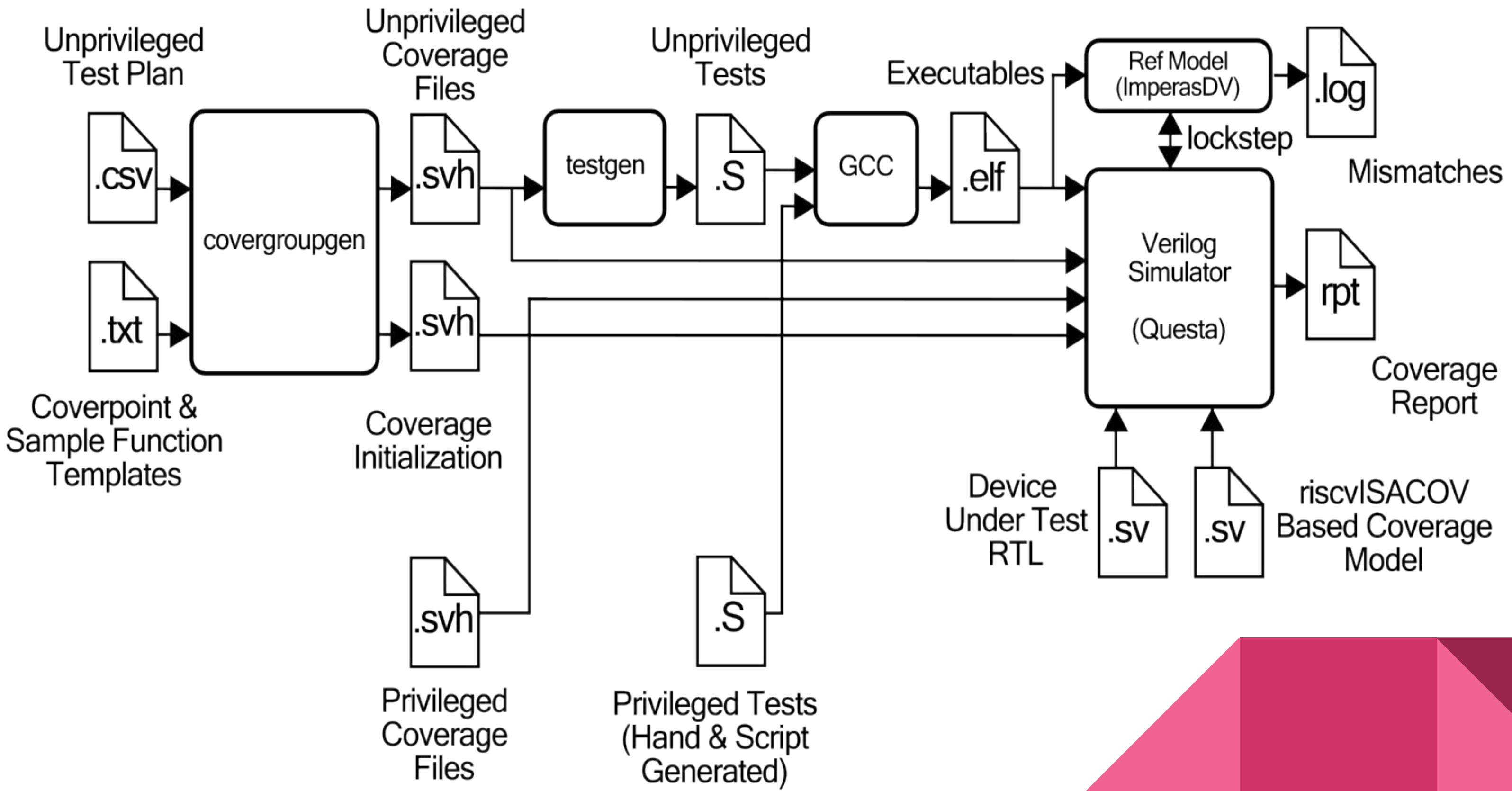
- Functional Verification
  - Unfeasible to test all possible instructions
    - Testing add instruction alone has over ~2^128 different combinations
  - Test a reasonable subset for add instruction
    - Should use every register as a source and destination
    - Corner cases and random values for sources
    - Verify the output of this subset
  - Generate tests for add
    - Coverpoints record what desired states were met by the tests

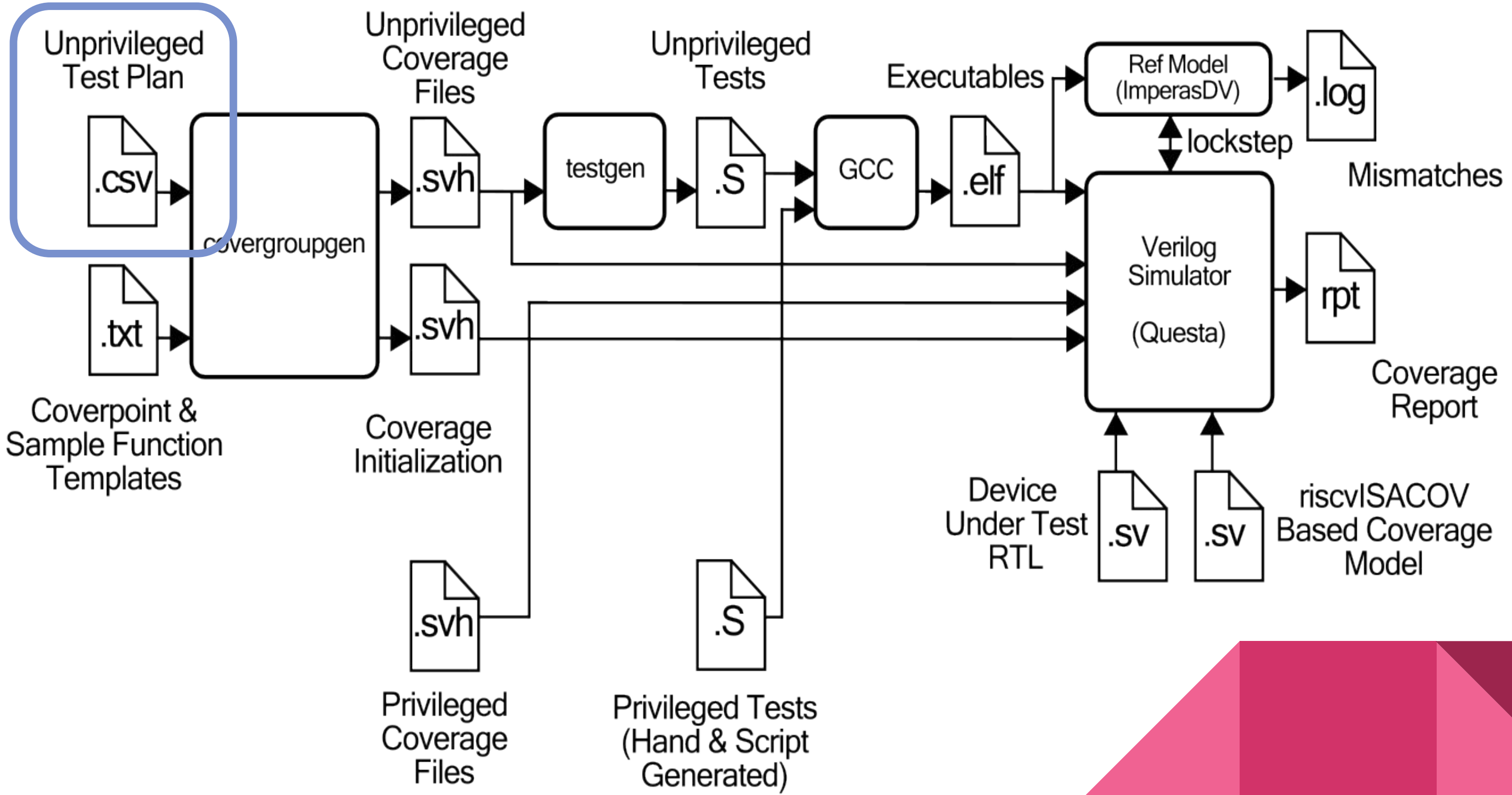
Lockstep Simulation

- Runs test files that exercises all desired states on DUT and RM
- Compares architectural state after each instruction completely executes
- Simulation halts if a mismatch occurs
- Produces a coverage report to show the percentage of desired architectural state we exercised (which coverpoints were hit)

Testing Tool Flow (Unprivileged)

Tool Flow


Unprivileged Test Plan
.csv
.txt
Coverpoint & Sample Function Templates
covergroupgen
Unprivileged Coverage Files
.svh
.svh
Coverage Initialization
testgen
Unprivileged Tests
.S
GCC
Executables
.elf
Ref Model (ImperasDV)
.log
lockstep
Mismatches
Verilog Simulator (Questa)
rpt
Coverage Report
Device Under Test RTL
.sv
.sv
riscvISACOV Based Coverage Model
.svh
Privileged Coverage Files
.S
Privileged Tests (Hand & Script Generated)

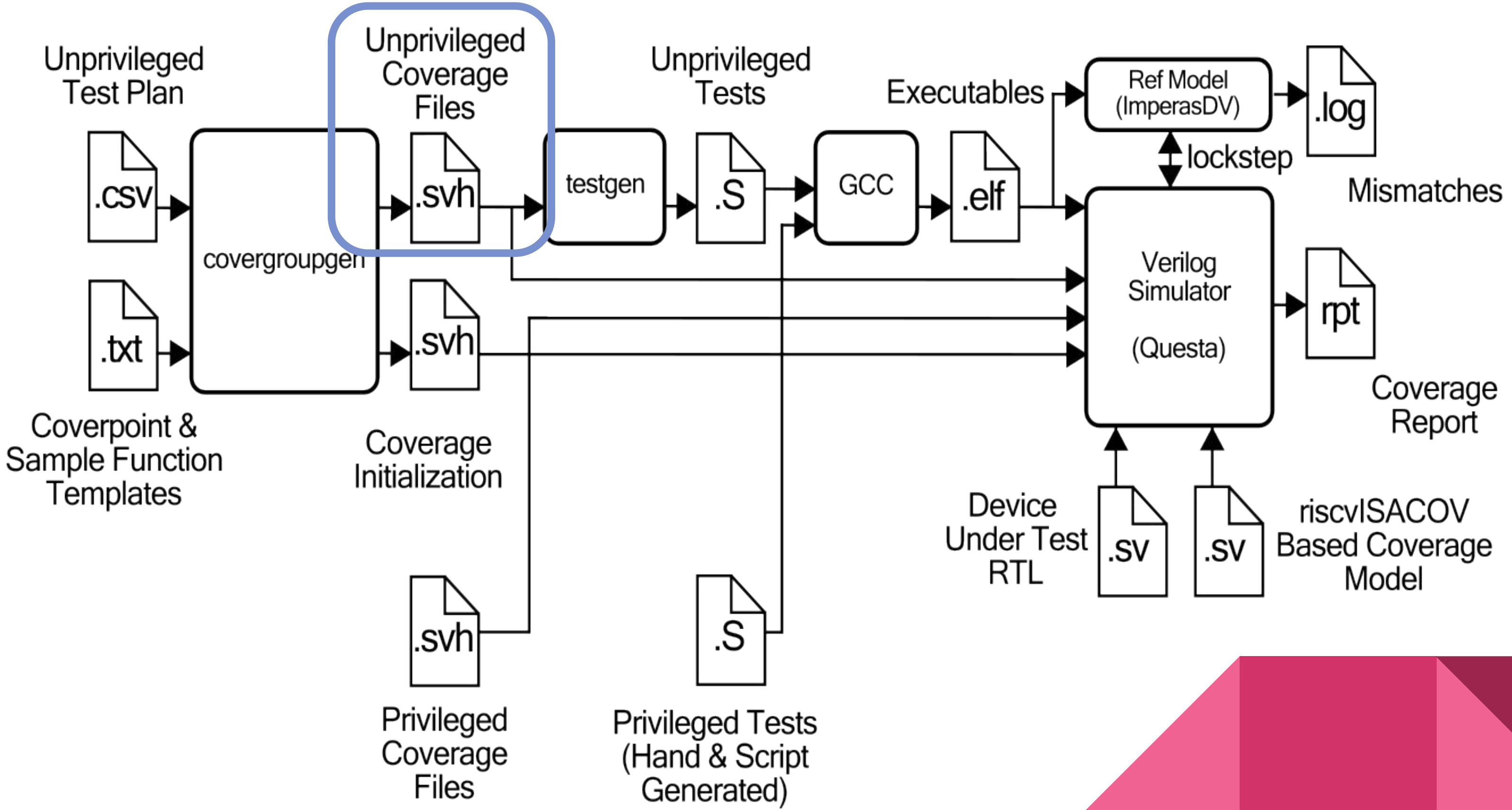

Unprivileged Test Plan
.csv
.txt
Coverpoint & Sample Function Templates
covergroupgen
Unprivileged Coverage Files
.svh
.svh
Coverage Initialization
testgen
Unprivileged Tests
.S
GCC
Executables
.elf
Ref Model (ImperasDV)
.log
lockstep
Mismatches
Verilog Simulator (Questa)
rpt
Coverage Report
Device Under Test RTL
.sv
.sv
riscvISACOV Based Coverage Model
.svh
Privileged Coverage Files
.S
Privileged Tests (Hand & Script Generated)

Example Unprivileged Test Plan - I

Instruction	Type	cp_asm_count	cp_rd	cp_rs1	cp_rs2	cp_rd_corners	cp_rs1_corners	cp_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		

Example Unprivileged Test Plan - I

Instruction	Type	cp_asm_count	cp_rd	cp_rs1	cp_rs2	cp_rd_corners	cp_rs1_corners	cp_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		

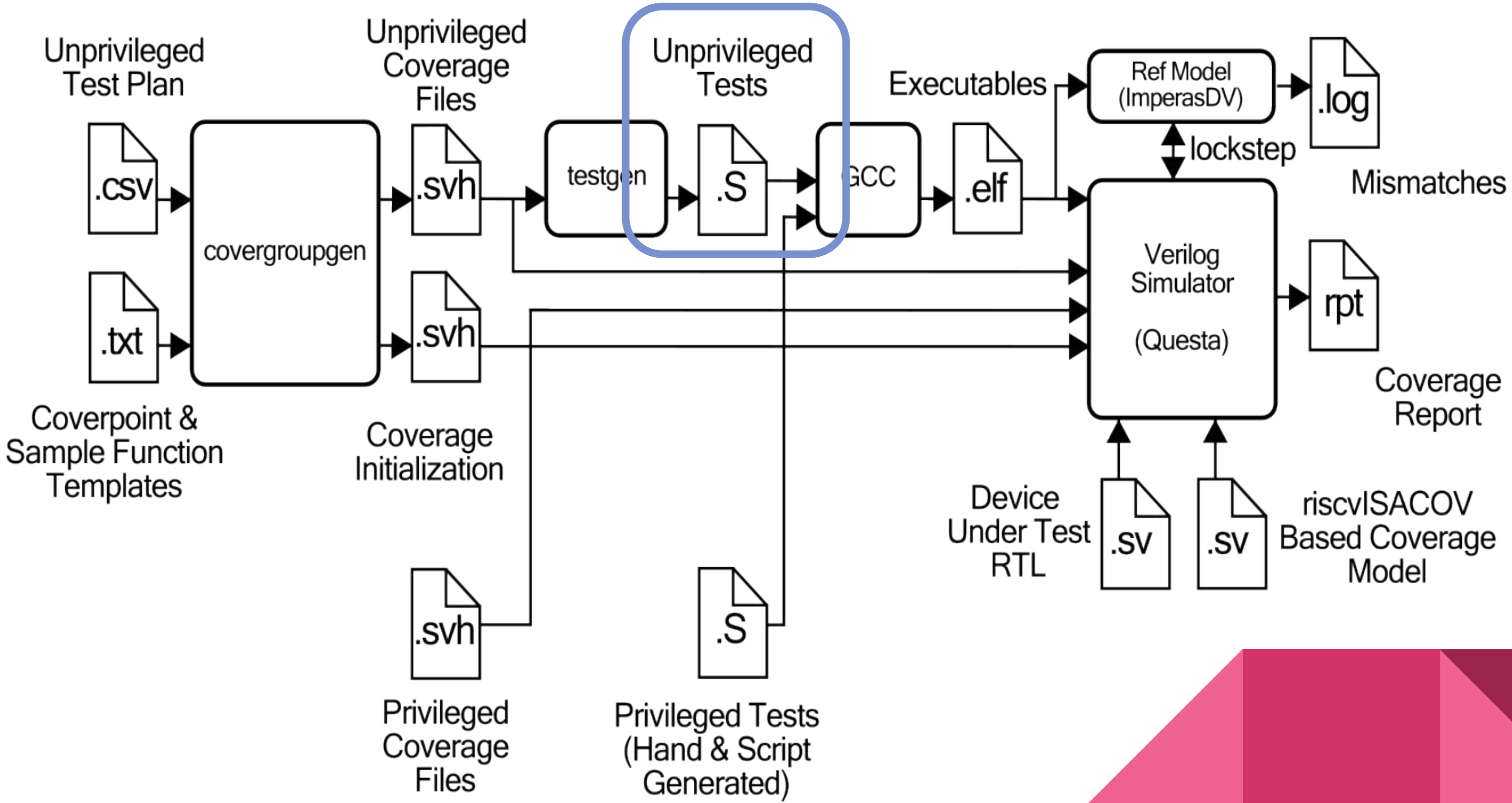

Unprivileged Test Plan
.csv
.txt
Coverpoint & Sample Function Templates
covergroupgen
Unprivileged Coverage Files
.svh
.svh
Coverage Initialization
testgen
Unprivileged Tests
.S
GCC
Executables
.elf
Ref Model (ImperasDV)
.log
lockstep
Mismatches
Verilog Simulator (Questa)
rpt
Coverage Report
.svh
Privileged Coverage Files
.S
Privileged Tests (Hand & Script Generated)
Device Under Test RTL
.sv
.sv
riscvISACOV Based Coverage Model

Unprivileged Coverpoints

```
covergroup I_add_cg with function sample(ins_t ins);

    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 walkodd =  {32'b10101010101010101010101010101010};
        wildcard bins walkeven = {32'b01010101010101010101010101010101};
        wildcard bins random   = {32'b01011011101111001000100001110111};
     }
                                   ...
endgroup
```


Unprivileged Test Plan
.csv
.txt
Coverpoint & Sample Function Templates
covergroupgen
Unprivileged Coverage Files
.svh
.svh
Coverage Initialization
testgen
Unprivileged Tests
.S
GCC
Executables
.elf
Ref Model (ImperasDV)
.log
lockstep
Mismatches
Verilog Simulator (Questa)
rpt
Coverage Report
Device Under Test RTL
.sv
.sv
riscvISACOV Based Coverage Model
.svh
Privileged Coverage Files
.S
Privileged Tests (Hand & Script Generated)

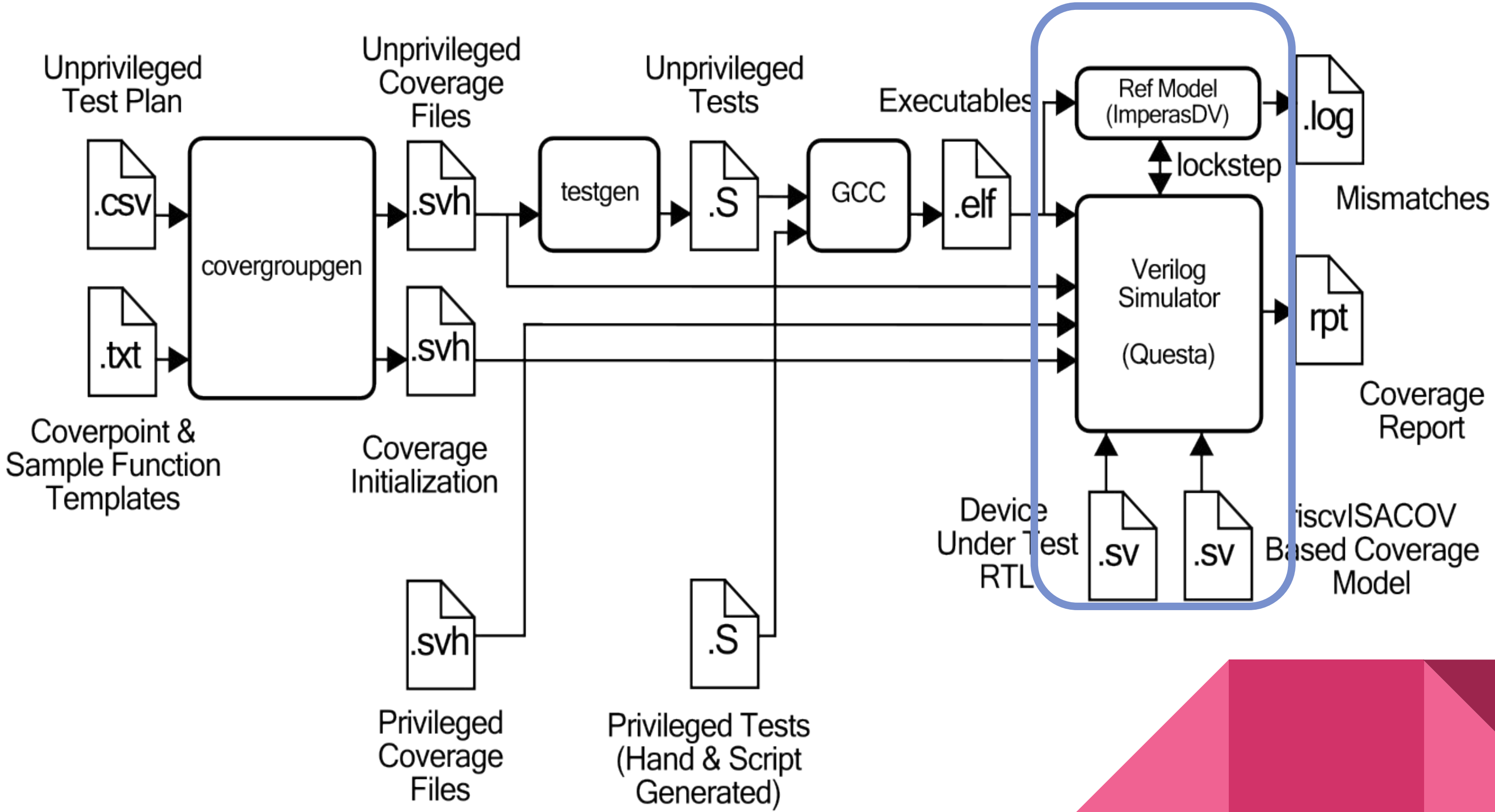
Unprivileged Tests - WALLY-COV-add.s

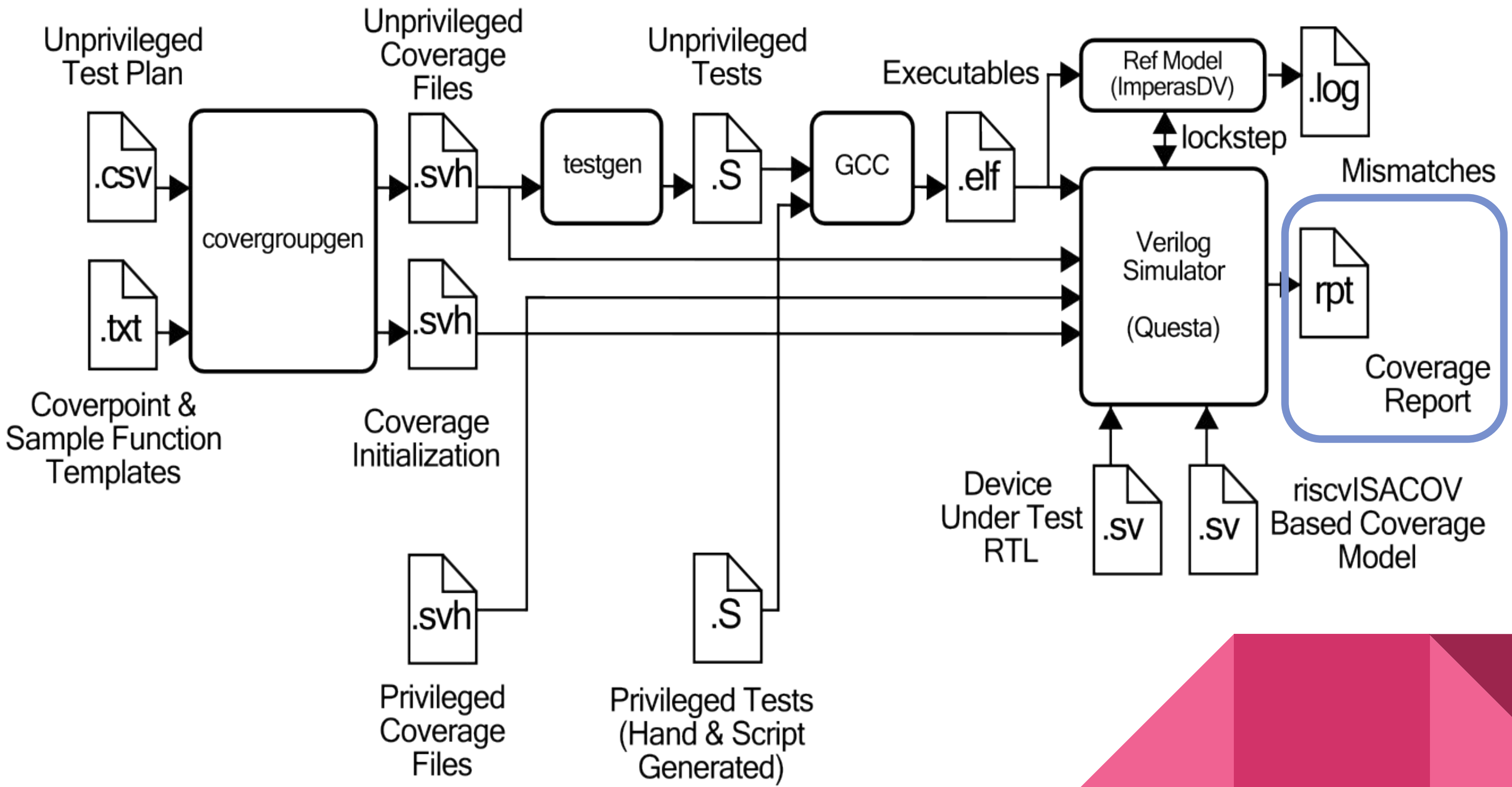
```
# Testcase cp_rd (Test destination rd = x0)
li x15, 0×2df19c49bf9d2b53 # initialize rs1 to a random value
li x3, 0×e21482b7cfc50d6b # initialize rs2 to a random value
add x0, x15, x3 # perform operation
RVTEST_SIGUPD(x5, x0)

# Testcase cp_rd (Test destination rd = x1)
li x24, 0×214606929e470dd7 # initialize rs1 to a random value
li x23, 0×2391fecd330ad053 # initialize rs2 to a random value
add x1, x24, x23 # perform operation
RVTEST_SIGUPD(x5, x1)

 …

# 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 Test Plan
.csv
.txt
Coverpoint & Sample Function Templates
covergroupgen
Unprivileged Coverage Files
.svh
.svh
Coverage Initialization
testgen
Unprivileged Tests
.S
GCC
Executables
.elf
Ref Model (ImperasDV)
lockstep
Verilog Simulator (Questa)
.log
Mismatches
rpt
Coverage Report
Device Under Test RTL
.sv
.sv
riscvISACOV Based Coverage Model
.svh
Privileged Coverage Files
.S
Privileged Tests (Hand & Script Generated)


Unprivileged Test Plan
.csv
.txt
Coverpoint & Sample Function Templates
covergroupgen
Unprivileged Coverage Files
.svh
.svh
Coverage Initialization
testgen
Unprivileged Tests
.S
GCC
Executables
.elf
Ref Model (ImperasDV)
.log
lockstep
Verilog Simulator (Questa)
Mismatches
rpt
Coverage Report
Device Under Test RTL
.sv
.sv
riscvISACOV Based Coverage Model
.svh
Privileged Coverage Files
.S
Privileged Tests (Hand & Script Generated)

Example Report - add covergroup in RV32

```
TYPE /RISCV_coverage_pkg/RISCV_coverage__1/I_add_cg
                                                     100.00%        100           -     Covered
    covered/total bins:                                  401        401           -
    missing/total bins:                                    0        401           -
    Coverpoint cp_asm_count                          100.00%        100           -     Covered
        covered/total bins:                                1          1           -
        missing/total bins:                                0          1           -
        bin count[1]                                     567          1           -     Covered
    Coverpoint cp_rd                                 100.00%        100           -     Covered
        covered/total bins:                               32         32           -
        missing/total bins:                                0         32           -
        bin auto[x0]                                       5          1           -     Covered
        bin auto[x1]                                      24          1           -     Covered
        bin auto[x2]                                      15          1           -     Covered
        bin auto[x3]                                      18          1           -     Covered
        bin auto[x4]                                      19          1           -     Covered
        bin auto[x5]                                      23          1           -     Covered
        bin auto[x6]                                      13          1           -     Covered
        bin auto[x7]                                      15          1           -     Covered
        bin auto[x8]                                      16          1           -     Covered
        bin auto[x9]                                      17          1           -     Covered
        bin auto[x10]                                     21          1           -     Covered
        bin auto[x11]                                     15          1           -     Covered
        bin auto[x12]                                     20          1           -     Covered
        bin auto[x13]                                     22          1           -     Covered
        bin auto[x14]                                     22          1           -     Covered
                                                       ...
```

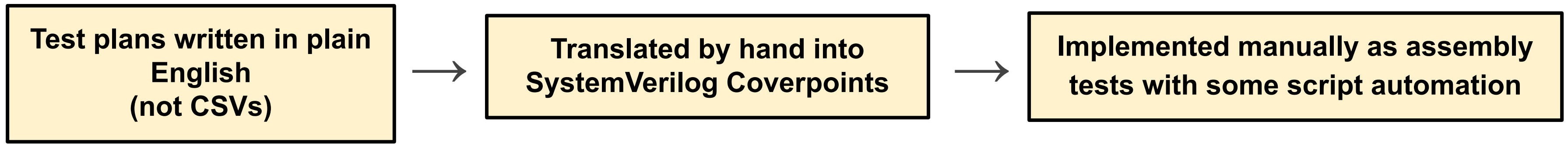
Example Report - add covergroup in RV32

```
TYPE /RISCV_coverage_pkg/RISCV_coverage__1/I_add_cg
                                                     100.00%        100          -    Covered
    covered/total bins:                                  401        401          -
    missing/total bins:                                    0        401          -
    Coverpoint cp_asm_count                          100.00%        100          -    Covered
        covered/total bins:                                1          1          -
        missing/total bins:                                0          1          -
        bin count[1]                                     567          1          -    Covered
    Coverpoint cp_rd                                 100.00%        100          -    Covered
        covered/total bins:                               32         32          -
        missing/total bins:                                0         32          -
        bin auto[x0]                                       5          1          -    Covered
        bin auto[x1]                                      24          1          -    Covered
        bin auto[x2]                                      15          1          -    Covered
        bin auto[x3]                                      18          1          -    Covered
        bin auto[x4]                                      19          1          -    Covered
        bin auto[x5]                                      23          1          -    Covered
        bin auto[x6]                                      13          1          -    Covered
        bin auto[x7]                                      15          1          -    Covered
        bin auto[x8]                                      16          1          -    Covered
        bin auto[x9]                                      17          1          -    Covered
        bin auto[x10]                                     21          1          -    Covered
        bin auto[x11]                                     15          1          -    Covered
        bin auto[x12]                                     20          1          -    Covered
        bin auto[x13]                                     22          1          -    Covered
        bin auto[x14]                                     22          1          -    Covered
                                                       ...
```

Testing Tool Flow (Privileged)

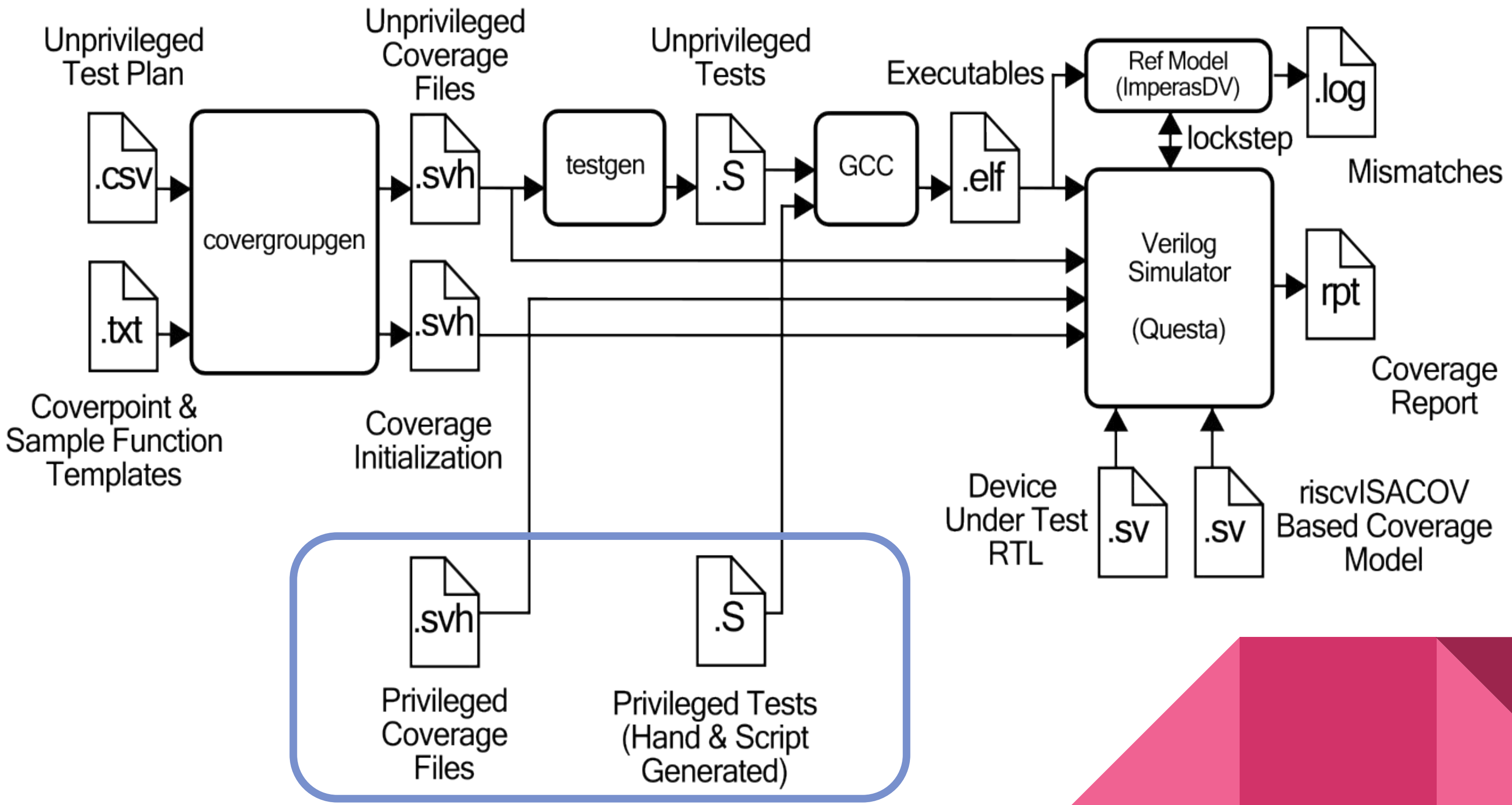
Privileged Testing: Major Differences

- Unprivileged: Testing is uniform across instructions due to a consistent structure
  - Instructions from many features can all be tested with just 1 test generator function
  - Unprivileged instructions implement relatively simple operations (arithmetic, load/store)
- Privileged: Requires specific tests for unique architectural states
  - Features can rarely be tested by running a just single instruction


Test plans written in plain English (not CSVs)
Translated by hand into SystemVerilog Coverpoints
Implemented manually as assembly tests with some script automation

Example Privileged Test Plan

Coverpoint	Covergroup	Sub-feature	Coverpoint Description	Test Hints
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	Only throws an exception if C is not supported.
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	No exception
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	Instruction at multiple of 4. Offset is an odd multiple of 2.
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.	



Unprivileged Test Plan
.csv
.txt
Coverpoint & Sample Function Templates
covergroupgen
Unprivileged Coverage Files
.svh
.svh
Coverage Initialization
testgen
Unprivileged Tests
.S
GCC
Executables
.elf
Ref Model (ImperasDV)
.log
lockstep
Mismatches
Verilog Simulator (Questa)
rpt
Coverage Report
Device Under Test RTL
.sv
.sv
riscvISACOV Based Coverage Model
.svh
Privileged Coverage Files
.S
Privileged Tests (Hand & Script Generated)

Privileged Coverpoints

```
`define COVER_EXCEPTIONSM
covergroup ExceptionsM_exceptions_cg with function sample(ins_t ins);

 priv_mode_m: coverpoint ins.prev.mode {
   bins M_mode = {2'b11};
 }

illegalops: coverpoint ins.current.insn {
       bins zeros = {'0};
       bins ones  = {'1};
 }

 cp_illegal_instruction:  cross illegalops, priv_mode_m;
endgroup
```

Example Test

```
///////////////////////////////////////////
// ExceptionsM.S
///////////////////////////////////////////
#include "WALLY-init-lib.h"

main:
  /////////////////////////////////
  //cp_illegal_instruction
  /////////////////////////////////
  //ExceptionsInstr.S tests all other illegal instructions exhaustively

  // Attempt to execute illegal instructions
  .word 0x00000000
  .word 0xFFFFFFFF
```

Results

Features: RVA22S64 & 32-bit equivalents

Feature	Test kLOC
I: Integer Base	81
M: Multiply/Divide	37
A: Atomics	21
F/D/Zfh/Zfa: Floating Point	2,284
Zb*: Bit Manipulation	80
Zk*: Cryptography	13
Zc*: Compressed	14
Zicsr: Control/Status Registers	1.9

Feature	Test kLOC
Zicond: Conditional Ops	4
Endianness	1.5
Exceptions	3
Interrupts	4
Virtual Memory	18
PMPs	WIP (UET)
Zicntr/Zihpm	1.9
Ssstrict	72

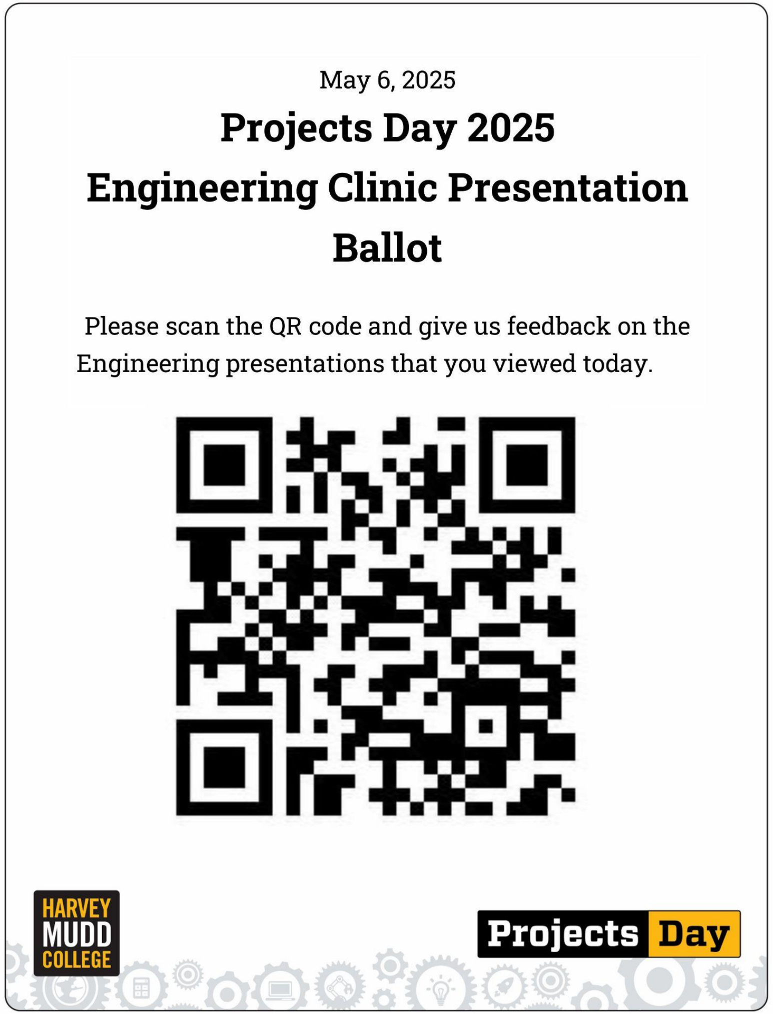
Bugs found in CORE-V Wally

- **`fround`** used the wrong shift amount for signed inputs
  - Flaw in signed to unsigned conversion logic
  - Coverage hole in existing test suites
- **`mstatus`** illegal control bits written in certain configurations
- Illegal instructions and CSR accesses did not consistently trap
  - Not all possible classes of input were tested by existing suites
  - Certain configurations of CVW never detected illegal instructions

Infrastructure Development

- Initial version relied on many proprietary tools
  - Questa Simulator, ImperasDV Reference Model, coverage framework proprietary components
- Shift towards more open source components
  - Functional Verification infrastructure now fully open source and self-contained
  - Plans for self-checking using open source reference model (Sail)
  - Long term: functional coverage collection with Verilator
- Integration with riscv-arch-test
  - Incorporating current standard for easy reuse by other processors
    - Potential for use in certification

Questions?



May 6, 2025

Projects Day 2025

Engineering Clinic Presentation Ballot

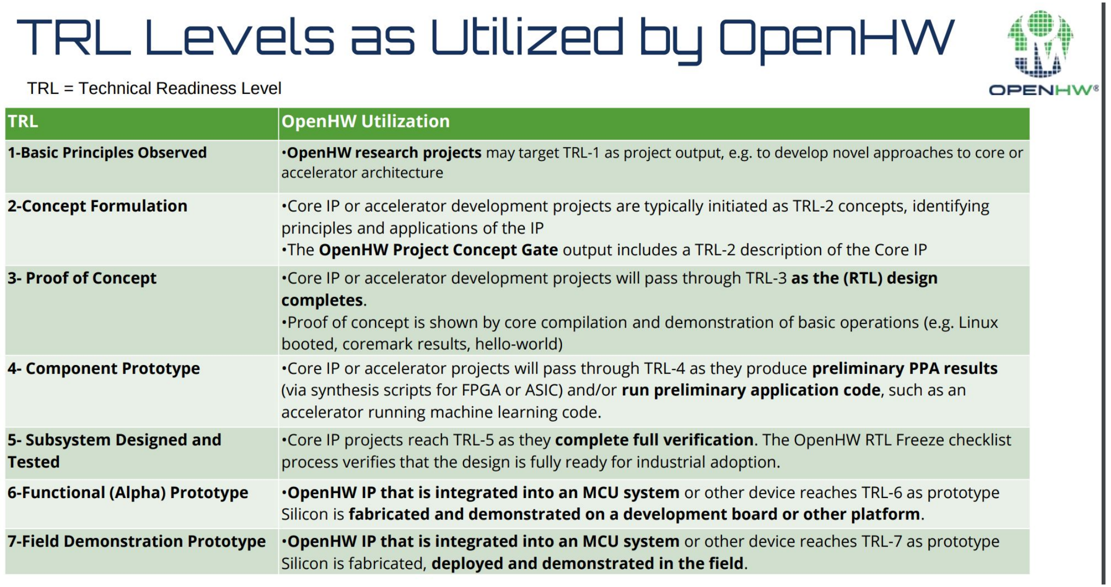
Please scan the QR code and give us feedback on the Engineering presentations that you viewed today.

Unprivileged Coverpoints

Coverpoint	Definition
cp_asm_count	Instruction used at least once
cp_r{s1/s2/d}	Use all 32 possible registers as source or destination
cp_rs{1/2}_corners cr_rs1_rs2_corners	0, 1, 2, MAX-1, MAX, -1, -2, MIN, MIN+1, walking 1s, random value Cross-product of two sources
cmp_rd_rs{1/2}_equalval	Registers have the same value
cmp_rd_rs{1/2} cmp_rs1_rs2 cmp_rd_rs1_rs2	Use same register as a source and/or destination
cp_offset	Branch offsets: positive and negative over range
cp_imm_corners cr_rs1_imm_corners	0, 1, 2, 3, 2047, 2049, 4095, 4096, walking 1s And cross-product of rs1 and immediate corners
cp_uimm	Unsigned immediate 0…XLEN-1 for shifts, Zbs
cp_gpr_hazard	RAW, WAR, WAW

Technology Readiness Level

- Developed by NASA
- Quantifies the maturity of technologies
- Used by many space agencies around the world



TRL Levels as Utilized by OpenHW

TRL = Technical Readiness Level

TRL	OpenHW Utilization
1-Basic Principles Observed	•**OpenHW research projects** may target TRL-1 as project output, e.g. to develop novel approaches to core or accelerator architecture
2-Concept Formulation	•Core IP or accelerator development projects are typically initiated as TRL-2 concepts, identifying principles and applications of the IP •The **OpenHW Project Concept Gate** output includes a TRL-2 description of the Core IP
3- Proof of Concept	•Core IP or accelerator development projects will pass through TRL-3 **as the (RTL) design completes**. •Proof of concept is shown by core compilation and demonstration of basic operations (e.g. Linux booted, coremark results, hello-world)
4- Component Prototype	•Core IP or accelerator projects will pass through TRL-4 as they produce **preliminary PPA results** (via synthesis scripts for FPGA or ASIC) and/or **run preliminary application code**, such as an accelerator running machine learning code.
5- Subsystem Designed and Tested	•Core IP projects reach TRL-5 as they **complete full verification**. The OpenHW RTL Freeze checklist process verifies that the design is fully ready for industrial adoption.
6-Functional (Alpha) Prototype	•**OpenHW IP that is integrated into an MCU system** or other device reaches TRL-6 as prototype Silicon is **fabricated and demonstrated on a development board or other platform**.
7-Field Demonstration Prototype	•**OpenHW IP that is integrated into an MCU system** or other device reaches TRL-7 as prototype Silicon is fabricated, **deployed and demonstrated in the field**.

Scope of Privileged Testing

- **Zicsr{M/S/U/F}**
  - Write every value to every distinct field of every implemented CSR
  - Test all privileged instructions
  - Exercise every illegal CSR
- **Exceptions{M/S/U/F/Zc/Zicbo/Sstvecd}**
  - Cause every type of exception for every reason in every mode
  - Vectored exceptions
  - Exercise every category of illegal instruction
- **Interrupts{M/S/U/Sstc}**
  - Interrupt generation, enabling, delegation, priority, mstatus impact in every priv mode
- **VM{SV32/39/48}**
  - All virtual memory features
  - satp fields, mstatus fields related to VM
  - sfence.vma, svinval
- **PMP{M/S/U}**
  - All PMP features for each register
- **Endian{M/S/U}**
  - Loads, stores of every size in every privilege mode crossed with every endianness choice
- **Zicntr{M/S/U}**
  - Counter/HPM access in every mode

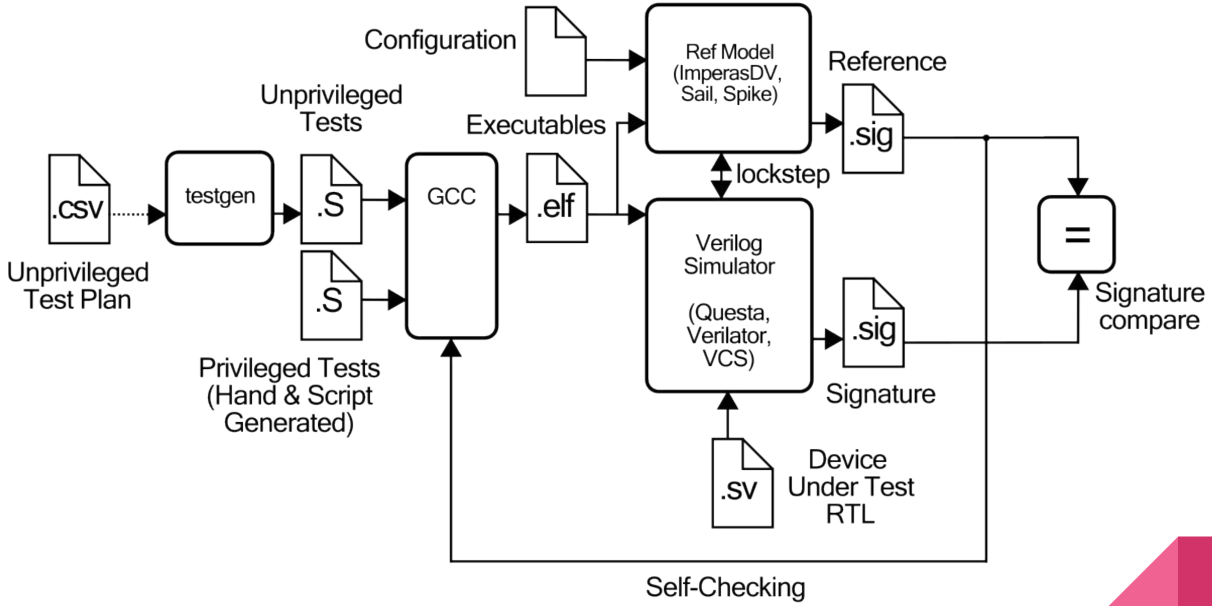
Privilege in RISC-V: Ensuring Security & Stability

Unprivileged mode -> programs have unrestricted access to all memory and features

2 problems:

- Faulty programs -> System crashes
- Malicious programs -> Security breaches

Integration with riscv-arch-test



Trap Handler (WALLY-init-lib.S) Key Features

- **Privilege Control:** ecall to switch modes or terminate tests
- **Exception Handling:** Return from interrupts & exceptions
- **Security Setup:** PMP, timer, and trap vector configuration
- **Illegal Instruction Testing:** Fast, exhaustive validation of illegal instructions and unimplemented CSRs

Notes:

- Extension of Imperas open-source RV32I riscvISACOV
  - https://github.com/riscv-verification/riscvISACOV
- All work is open-source (Solderpad/Apache License)
  - Hosted by OpenHW Group
  - https://github.com/openhwgroup/cvw-arch-verif
- Presently relies on commercial tools
  - Siemens Questa for functional coverage simulation (not needed after coverage proven)
  - Synopsys ImperasDV reference model via RVVI
- Full Open Source Path envisioned
  - Sail is working on lockstep over RVVI
  - Verilator is gradually adding coverage features (unnecessary to rerun coverage)