Generation and Configuration of Functional Coverage and Verification IP for RISC-V Processor Verification

Simon Davidmann (simond@imperas.com)

www.imperas.com

28-Nov-2023

The Design and Verification Club

(Europe and India)

Generation and Configuration of Functional Coverage and Verification IP for RISC-V Processor Verification

Abstract:

- The open standard RISC-V instruction set architecture (ISA) offers developers new freedoms and flexibilities to develop domain-specific processors
- RISC-V offers every SoC team the possibility to design an optimized processor, but this also implies that SoC design verification teams will need to address the challenge and complexity of processor verification

- A key metric for design verification (DV) is functional coverage
- Given the number of extensions in the RISC-V ISA, and instructions per extension, just writing the functional coverage modules is challenging.
  - With over 1,000 instructions in the ISA, functional coverage for a fully featured processor could require > 100,000 lines of SystemVerilog
- Writing this by hand is certainly time-consuming and resource intensive, and is vulnerable to errors.
- We report here on a methodology for auto-generation of the functional coverage modules for RISC-V processor DV.

Agenda

- The new world… every team develops a RISC-V based SoC
- The challenges of verifying RISC-V CPUs
  - Scope of DV: domain specific, extensions, individual project customization
- Functional Coverage – a key measurement of progress & quality
- Automation (and riscvISACOV) to the rescue
- Case study with open source CVW core
- Summary

The new world… RISC-V based SoCs

- Why RISC-V, why now?:
  - Changes in commercial/business model with existing ISAs, e.g. MIPS, Arm
  - New electronic products being defined by software requirements
  - Cost, timescales, and complexity are driving collaborative development approaches and open source
- What in RISC-V is so attractive?
  - It gives designers ***freedom***
    - Open standard, Evolving quickly, Configuration and extensibility, Focus and scalability
    - Domain specific processors

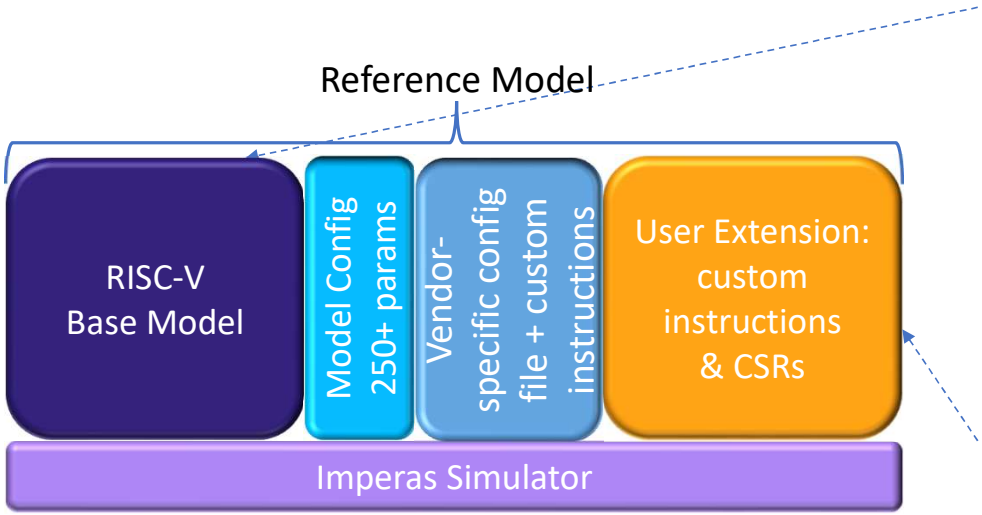
What is the cost of these freedoms?

- Need a highly capable model of CPU
  - Complete (support the full ISA standard), and evolves with standard
  - Fully verified, supported/maintained, adopted by many
  - Controllable, analyzable
  - Methodologies and capabilities for extension & configuration
  - Work with industry solutions (virtual prototypes, debug, verification)
  - High performance: fast speed and low memory footprint => high capacity, throughput
- Very sophisticated tooling needs to be available
  - Software defined electronic products need virtual platforms
- Need much more capable verification for ISA and its configuration / extension
  - Imperas keynote @ RISC-V Summit December 2022
    - Application processor needs ~10**15 verification cycles (Arm data)
      - == 15,000 SW simulator years (SystemVerilog)
      - == 30 HW simulator years (FPGA/accelerator)
    - Custom instructions and state significantly add to the complexity…

ImperasFPM (Fast Processor Models)

- 2008 – Imperas developed world class processor modeling & simulation solutions for many ISAs for virtual prototyping and software development
  - Based on open standard OVP APIs (OVPworld.org)
  - Models written in C, with source available
  - A good, growing, and profitable business
- 2016 Imperas started looking at RISC-V
- Developed first commercially supported RISC-V models
- RISC-V adopters started using Imperas as reference for:
  - System architecture exploration
  - Software development
  - Processor hardware verification

Imperas Fast Processor Models + Simulator Architecture



- ***RISC-V Base Model is used in all Imperas RISC-V processor models***
  - ***By commercial users***
  - ***By academic users***
  - ***By users of the Imperas free ISS riscvOVPsimPlus***
- ***RISC-V Base Model is used by > 150 organizations***

Imperas develops and maintains Base Model

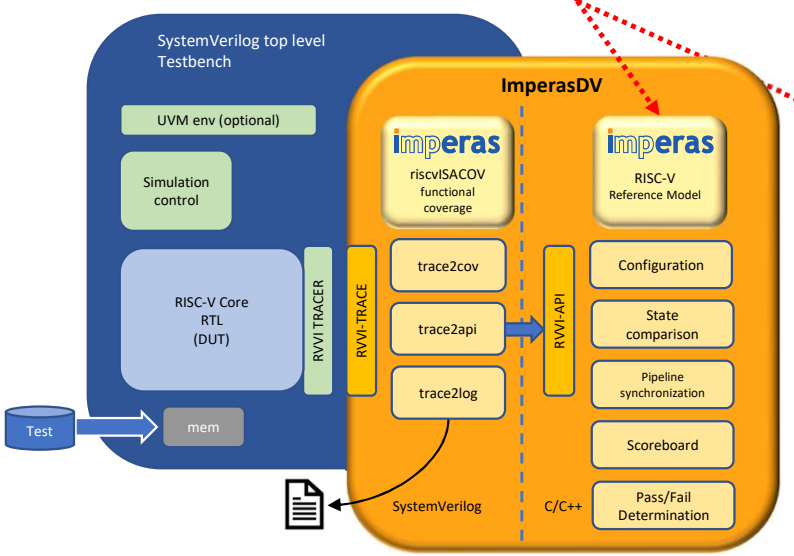
- Base Model implements RISC-V specification in full
- Base Model built using Test Driven Development methodology
- Built using public open standard OVP APIs matured over 15 different ISAs
- Simulator is separate from the model; supports the modelling APIs
- Fully user configurable to select which ISA extensions
- Fully user configurable to select which version of each ISA extension
- For processor IP vendors, have pre-defined configuration plus vendor custom instructions

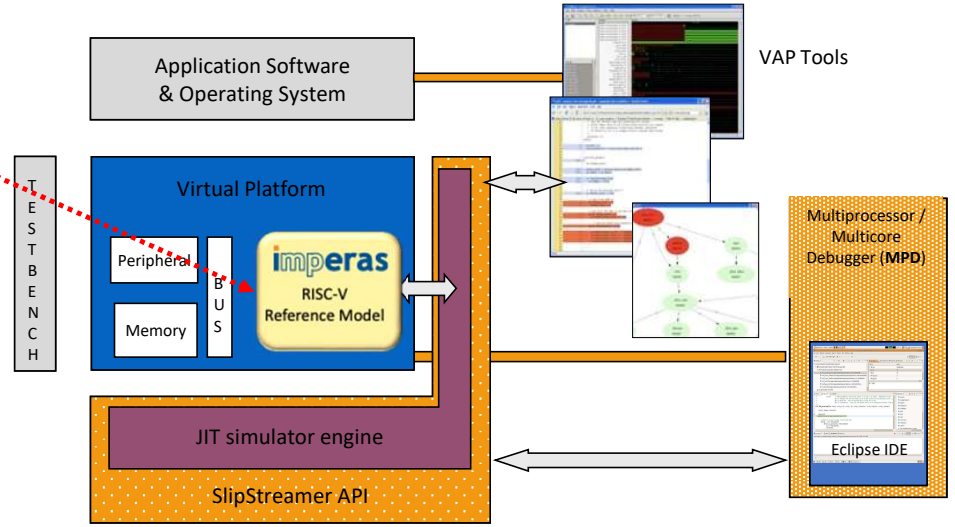
Imperas provides methodology to easily extend base model

- Custom instructions added using same APIs as in Base Model
- Separate source files and no duplication to ensure easy maintenance
- 100+ page user guide/reference manual with many examples
- User extensions source can be proprietary or open source license

Imperas OVP RISC-V Models are used for Processor DV & SW Development

- Base model implements RISC-V specification in full
- Fully user configurable to select which ISA extensions/versions
- Imperas provides methodology to easily extend base model





Agenda

- The new world… every team develops a RISC-V based SoC
- The challenges of verifying RISC-V CPUs
  - Scope of DV: domain specific, extensions, individual project customization
- Functional Coverage – a key measurement of progress & quality
- Automation (and riscvISACOV) to the rescue
- Case study with open source CVW core
- Summary

Verifying a RISC-V CPU

- DV methodology starts with Verification Plan of what is needed
- This includes thousands of specific items to test
- Functional coverage is a mechanism to measure what has been tested
- SystemVerilog language includes testbench syntax for this:
  - Coverpoints: individual items to measure and count into bins
  - Covergroups: collection of related points to measure at sampling events
- SystemVerilog simulators perform the counting and report results

Functional Coverage - Types required for a RISC-V CPU

For a RISC-V CPU there are different types of functional coverage required:

- Standard ISA architectural features
  - unpriv. ISA items: mainly instructions, their operands, their values
  - priv. ISA items, CSRs, Interrupts, Debug block, …

  => these are standard and the 'same' for all RISC-V processors – it is the spec…
- Customer core design & micro-architectural features
  - pipeline, multi-issue, multi-hart, …
  - Custom extensions, CSRs, instructions

  => these are design specific and very customer bespoke – they are proprietary
    - they will be about pipeline etc. specific issues
    - they will include items like pipeline issues/hazards etc…
    - these will always need to be added by customer for each core/design
- Yes, you will need functional coverage for all this…

RISC-V CPU Functional Coverage - In context…

RISC-V Instructions (Standard ISA architectural feature)

- There are many different instructions in the RV64 extensions:
  - Integer: 56, Maths: 13, Compressed: 30, FP-Single: 30, FP-Double: 32
  - Vector: 356, Bitmanip: 47 Krypto-scalar: 85
  - P-DSP: 318
  - For RV64 that is 967 instructions…

- And for each instruction there will need to be written SystemVerilog covergroups and coverpoints…
  - Maybe 10-100 lines of SystemVerilog for each instruction…

⇒10,000-100,000++ lines of code to be written…
  - and be correct and working…
  - For the non-priv. instructions etc…

=> And … It is not design specific, nor specific to your core…

RISC-V CPU Functional Coverage (2)

- Tedious and error-prone to write by hand…
- For each design…
- It is unnecessary to re-develop – it is the same for all ISA compliant cores
- There needs to be collaboration…
- There needs to automation…

=> that is the focus of Imperas' riscvISACOV (https://github.com/riscv-verification/riscvISACOV)

- enable the developing of functional coverage VIPs that can be used for many different core configurations/implementations

Agenda

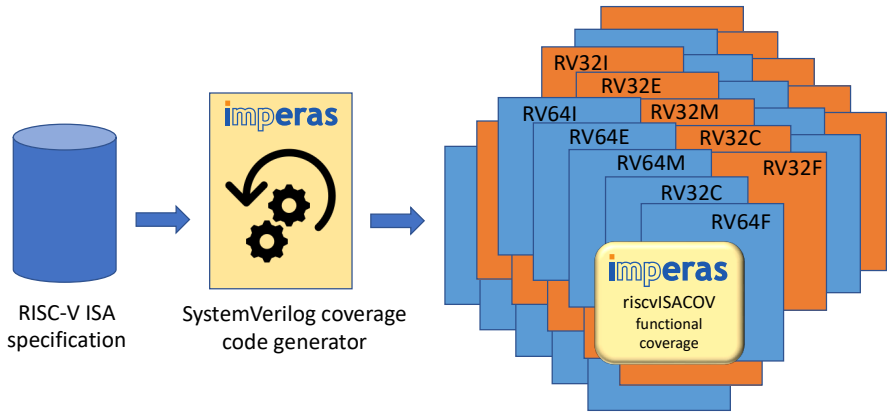
- The new world… every team develops a RISC-V based SoC
- The challenges of verifying RISC-V CPUs
  - Scope of DV: domain specific, extensions, individual project customization
- Functional Coverage – a key measurement of progress & quality
- Automation (and riscvISACOV) to the rescue
- Case study with open source CVW core
- Summary

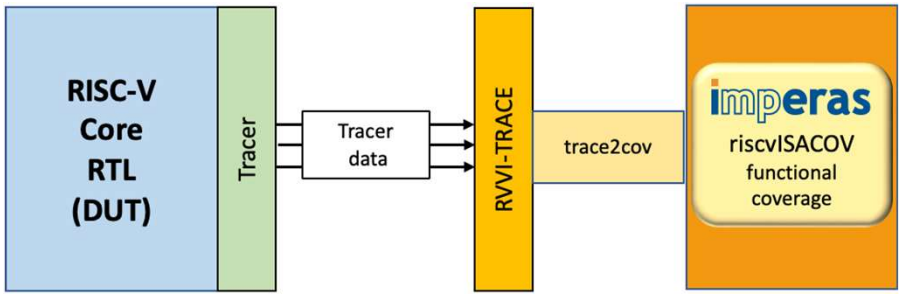
riscvISACOV
SystemVerilog functional coverage

https://github.com/riscv-verification/riscvISACOV

- Use a machine-readable version of the RISC-V ISA to auto generate reusable covergroups and coverpoints
  - 220,000 lines of source
  - 2,100 covergroups, 11,200 coverpoints
  - 136 extension options
  - Hazards, CSRs, CSRcompares, PMP, MMU
  - 6 levels of coverage detail
    - Base compliance -> DV privilege mode extended
- Use open standard RISC-V Verification Interface RVVI-TRACE data to sample the generated covergroups

=> measures how much verification is done





Agenda

- The new world… every team develops a RISC-V based SoC
- The challenges of verifying RISC-V CPUs
  - Scope of DV: domain specific, extensions, individual project customization
- Functional Coverage – a key measurement of progress & quality
- Automation (and riscvISACOV) to the rescue
- Case study with open source CVW core
- Summary

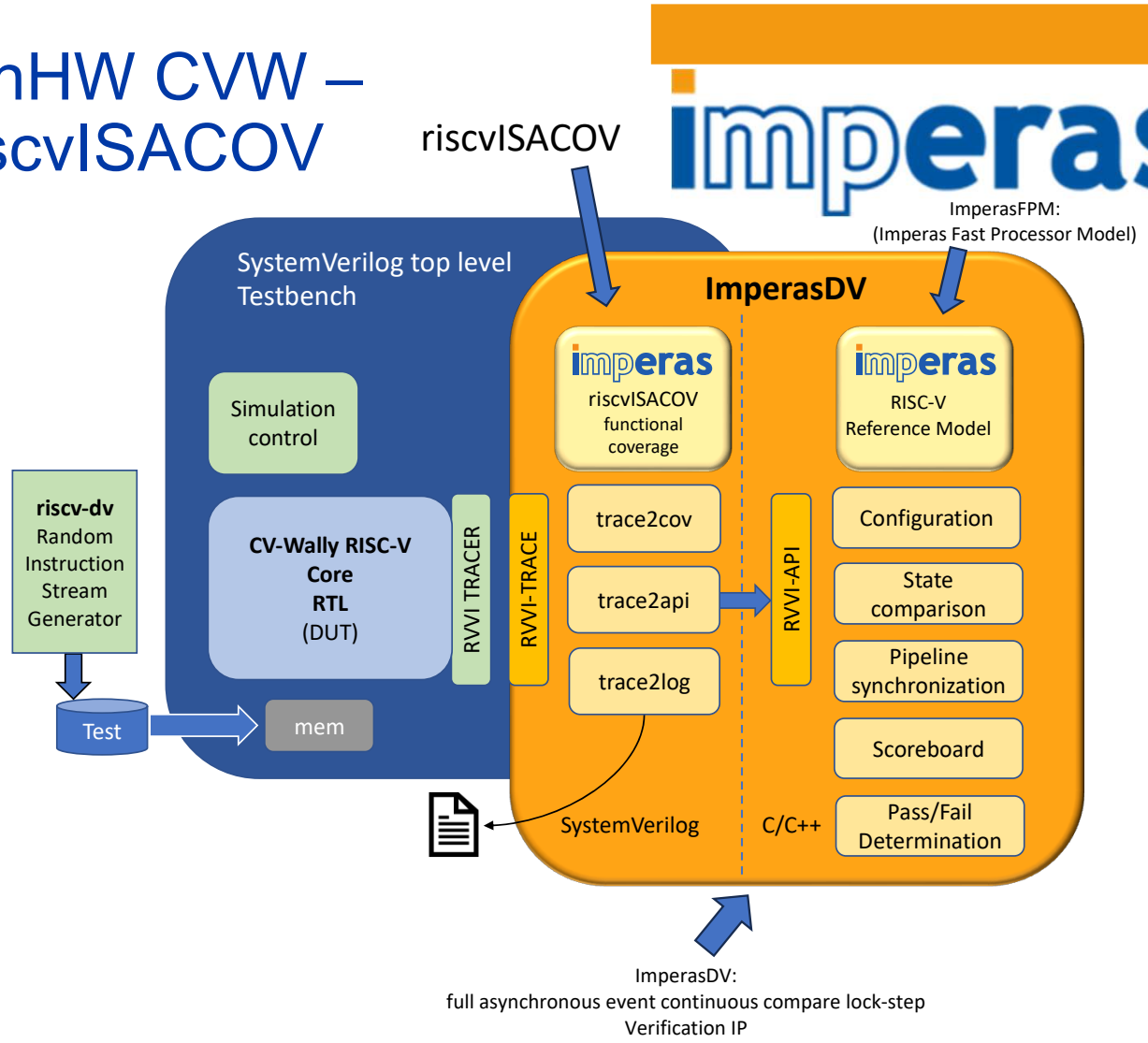
CPU DV example: OpenHW CVW

imperas

- CVW: 5 stage, in-order, single issue RISC-V 32/64 bit application core in new textbook
  - "System-on-Chip Design": Harris, Stine, Thompson, Stine (Elsevier 2025)
- Prior to starting to work with Imperas Fast Processor Model (ImperasFPM) & ImperasDV:
  - RTL passed all RISC-V International Compatibility tests (riscv-arch-tests)
  - RTL passed custom tests, especially for privileged unit
  - 75.2% code coverage per Siemens EDA's Questa
  - Created many more directed tests: Raised code coverage from 75.2% to 91.9%
- Verification methodology: program & memory signature compare
- RTL successfully boots Linux (500M+ instructions), in SystemVerilog simulation, and on an FPGA
- Completeness (riscvISACOV)
  - Boot Linux:
    - 38% of 200 covergroups (instructions)
    - 2% of 1.5M coverpoints/bins
    - (11% of the MMU coverage…)

CPU DV example: OpenHW CVW – using ImperasFPM + riscvISACOV

- Stimulus:
  - Pseudo-random test programs generated by riscv-dv*
- Verification:
  - Same program run on CVW and Imperas reference model
  - Internal state of CVW and reference continuously compared
  - Mismatches reported immediately
  - External interrupts handled by ImperasDV



*https://github.com/chipsalliance/riscv-dv

CPU DV example: OpenHW CVW – results of using Imperas solutions

Github Issue	Fix Pull Request	Description
5	44	fence.i doesn't flush D$ to instruction memory properly
46		Load page fault occurs on misaligned load
47	48	ecall incorrectly increments minstret
49	54	CSRs updated too soon during pipeline stall
50	54	D$ flush not suppressed during hardware page table walk, leading to improper instruction page fault
55-58, 65	93	Single-precision inputs treated as double-precision for NaNs, fsgnjn, fclass
59	128	Decoder does not cause exception for an illegal instruction
66	69	Load miss at same time as spilled instruction fetch with ITLB miss does not wait for load miss to finish
70	74	Floating-point mstatus.FS improperly enabled on reset
106	149	c.fld produces wrong MTVAL (uncompressed not compressed opcode)
120	146	SIE/SIP should be zero unless mideleg bits are set
142	149	scounteren reset to inconsistent value
145	149	Illegal bits in PMPADDR bits 63:54 are not detected
148	170	Interrupt incorrectly taken while I$ and D$ are stalled
203	208	Wrong MEPC on trap when switching to user mode with bad PMP
392	304	During concurrent DTLB and ITLB misses, ITLB entry is corrupted, causing Linux to occasionally crash. Very complex bug.

- ImperasDV async-lock-step continuous compare verification methodology found significant bugs
- Recently used Imperas riscvISATESTS for MMU and found more bugs

Processor Verification Requires Methodology, Models, Tools, Verification IP

- Doing signature compare, or post-simulation trace-compare, is not a sufficiently comprehensive methodology to provide complete functional coverage
- Booting Linux (or running other software) is similarly incomplete
- Success in processor verification requires a high-quality model of the processor
- Success in processor verification requires innovative technologies and methodologies – automation helps scale

Agenda

- The new world… every team develops a RISC-V based SoC
- The challenges of verifying RISC-V CPUs
  - Scope of DV: domain specific, extensions, individual project customization
- Functional Coverage – a key measurement of progress & quality
- Automation (and riscvISACOV) to the rescue
- Case study with open source CVW core
- Summary

Automating the generation of the testbench Functional Coverage measuring components

- Generation of verification components can produce significant benefits to DV teams
- Requires machine readable formats as input to generators

- Imperas has created generation technology and results have been in use in 3 generations since 2020 with OpenHW cores & commercial users
- Imperas roadmap to make more generated RISC-V verification components available in 2024

Summary

- RISC-V freedom enables domain-specific processors
  - It is this freedom, and not the free, that is driving/accelerating RISC-V adoption
- With all the RISC-V implementations, custom instructions and configurations will be used – and will need modeling & verifying
- Verification is an exponential, and endless, problem
  - balance between the “cost of finding bugs” (the verification costs)

  versus
  - the “cost of not finding bugs” (the impact costs of bug escapes)
- Automation (like riscvISACOV) helps build the DV environment that can provide significant time to market, effort, and quality results

Generation and Configuration of Functional Coverage and Verification IP for RISC-V Processor Verification

Simon Davidmann (simond@imperas.com)

www.imperas.com

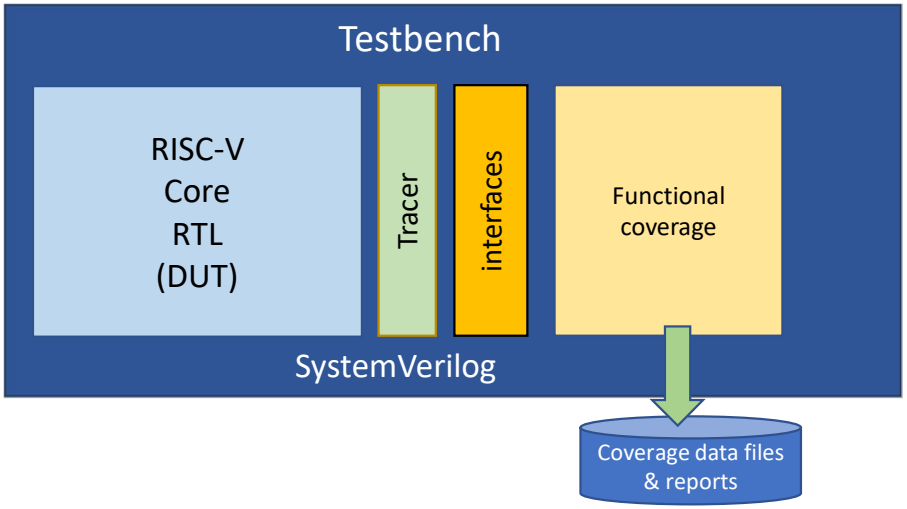
28-Nov-2023

The Design and Verification Club

(Europe and India)

Backup slides

SystemVerilog functional coverage overview



- Coverpoints & covergroups in SystemVerilog source in testbench
- Connects via interfaces into core through 'tracer'
- Core execution events trigger sampling of coverpoints to collect data on state of core
- SystemVerilog simulator & tools from e.g. Cadence, Synopsys, Siemens EDA, Metrics create coverage data files, collate, and then provide coverage reports

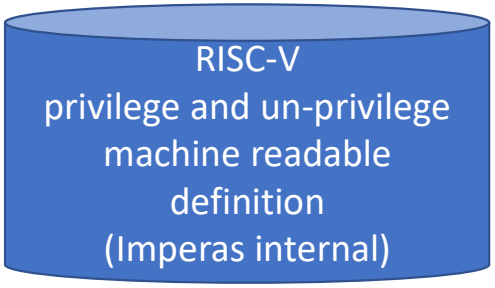
```
covergroup add_cg with function sample(ins_t ins);
    option.per_instance = 1;
    cp_rd : coverpoint get_gpr_name(ins.ops[0].key, "add") {
    }
    cp_rd_sign : coverpoint int'(ins.ops[0].val) {
        bins neg  = {[$:-1]};
        bins zero = {0};
        bins pos  = {[1:$]};
    }
    cp_rs1 : coverpoint get_gpr_name(ins.ops[1].key, "add") {
    }
    cp_rs1_sign : coverpoint int'(ins.ops[1].val) {
        bins neg  = {[$:-1]};
        bins zero = {0};
        bins pos  = {[1:$]};
    }
    cp_rs2 : coverpoint get_gpr_name(ins.ops[2].key, "add") {
    }
    cp_rs2_sign : coverpoint int'(ins.ops[2].val) {
        bins neg  = {[$:-1]};
```

e.g. SystemVerilog covergroups/coverpoints

rvvi_vlg2cov	8.08%	273 / 3981 (6.86%)
obj_add	74.79%	113 / 126 (89.68%)
cp_rd	100%	32 / 32 (100%)
cp_rd_sign	100%	3 / 3 (100%)
cp_rs1	100%	32 / 32 (100%)
cp_rs1_sign	33.33%	1 / 3 (33.33%)
cp_rs2	100%	32 / 32 (100%)
cp_rs2_sign	66.67%	2 / 3 (66.67%)

e.g. Cadence Xcelium IMC coverage report GUI

RISC-V Machine readable definitions



- There are several categories of items that need considering
  - ISA extensions and groups with version revisions
  - Instructions with format and coverage definitions
  - CPU state / Control and Status Registers (CSRs) with field and coverage definitions
  - Interrupt and Debug modes and coverage definitions
- Configurability of the items in these categories
- Context awareness of dynamic settings of these categories
- Specification capability of ‘user choice’, ‘implementation defined’