The Art and Science of Automating Verification Checking

Simon Davidmann (simond@imperas.com)

04-Oct-2022

The Design and Verification Club
(Europe and India)

The Art and Science of Automating Verification Checking

Abstract:

- As with many aspects of modern verification plans, the methodologies and techniques are revised and improved over time to improve efficiency and quality
- Specifications are the reference for all the key requirements from communication protocols, bus standards, and processor ISAs so these are the essential starting point
- Automation starts with the specifications and supports the development of generators for coverage libraries, score boarding, event sequences, and test suites
- Using examples from actual projects, this talk highlights the latest software generators for automating coverage tests
- Key Points:
  - Specifications are the basis of functional verification
  - Automating generators for coverage add efficiency and quality
  - How to apply software techniques to generators for SystemVerilog

There is never enough time

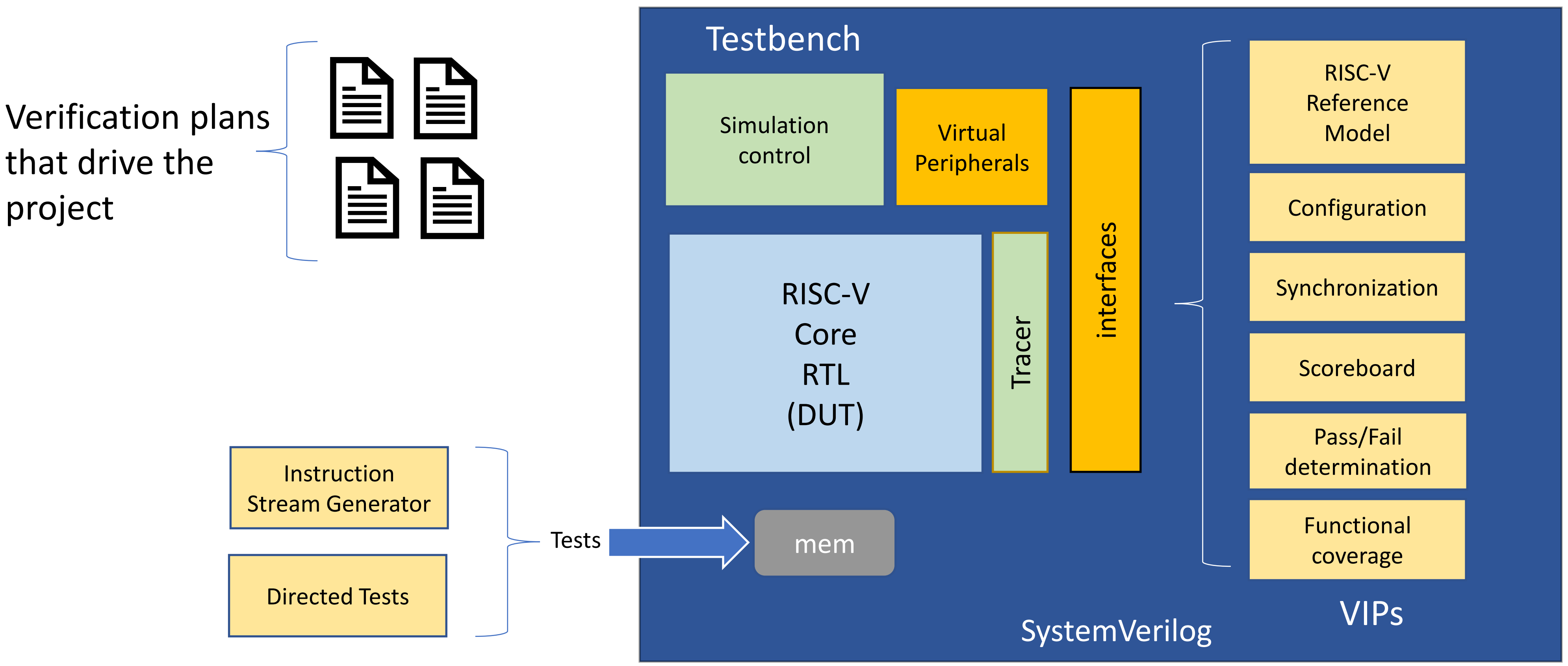
- Current electronic products need more verification and delivery timescales are shrinking
- It is OK saying ‘we can use the infinite resources of the cloud’
  - But… that assumes that all you need to do is run ‘stuff’
- Where does this ‘stuff’ come from…
- Well… you have to create it…

What is needed in verification projects

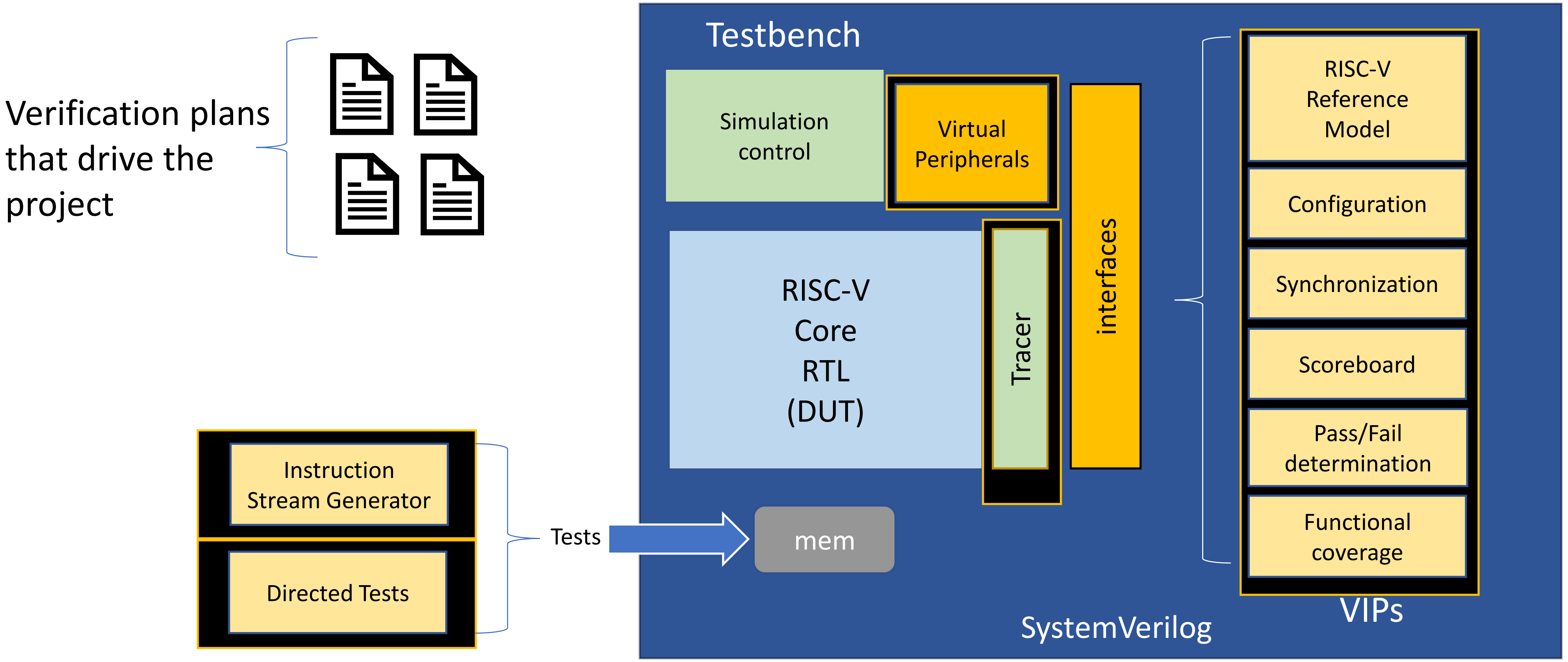
- Create verification plans - that drive the goals of the verification project...
- Develop & validate reference models/predictors and configure them
- Create the scoreboards to record and compare seen behavior / functionality
- Write functional coverage to measure progress vs. verification plan
- Write test benches to run them
- Write & generate test programs to stimulate them

- And then you can just run the ‘stuff’…

Example: RISC-V Processor DV



Example: RISC-V Processor DV



- So all this ‘stuff’ needs creating/acquiring/configuring before you can really start with DV

So where can automation help…

- Develop & validate reference models/predictors and configure them
- Create the scoreboards to record and compare seen behavior / functionality
- Write functional coverage to measure progress vs. verification plan
- Write test benches to run them
- Write & generate test programs to stimulate them

- And then you can just run the ‘stuff’…

So where can automation help…

- Develop & validate reference models/predictors and configure them

Best to get access to silicon proven (commercially) supported configurable reference models designed for full integration e.g. Imperas

- Create the scoreboards to record and compare seen behavior / functionality

★ This can be automated….

- Write functional coverage to measure progress vs. verification plan

★ This can be automated….

- Write test benches to run them

Use standards & high level reusable VIPs and this become easier

- Write & generate test programs to stimulate them

★ This can be mainly automated….

So where can automation help…

- Write functional coverage to measure progress vs. verification plan

★ This can be automated….

Automating the generation of the testbench Functional Coverage measuring components

Agenda:

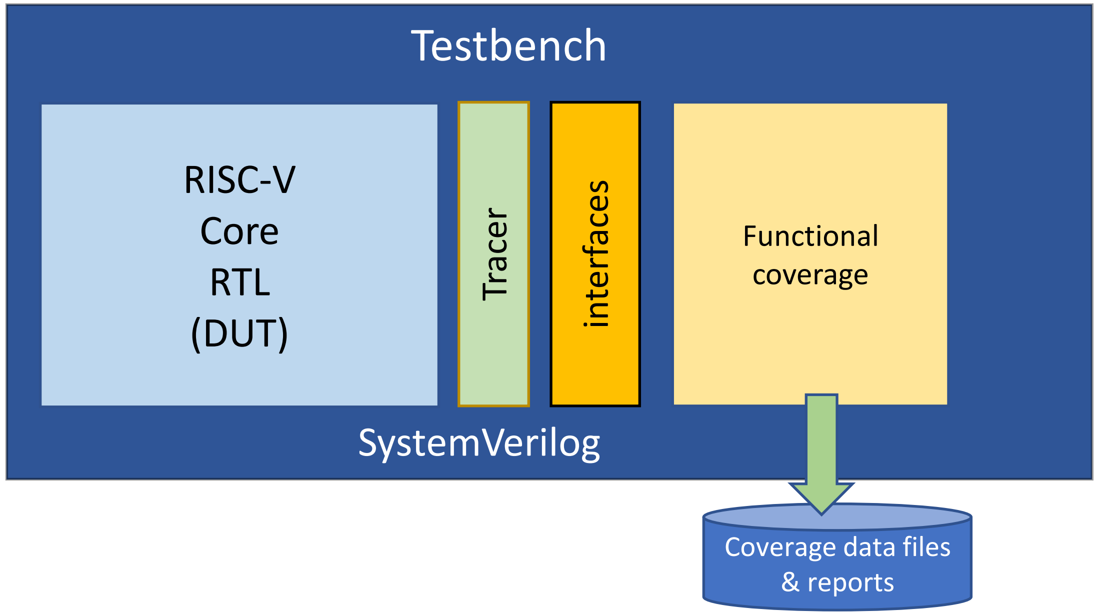
- SystemVerilog functional coverage overview
  - Interfaces, content, tools
- Requirements on the testbench approach
- Requirements for RISC-V processor functional coverage
- Generation flow
- Examples in use

Automating the generation of the testbench Functional Coverage measuring components

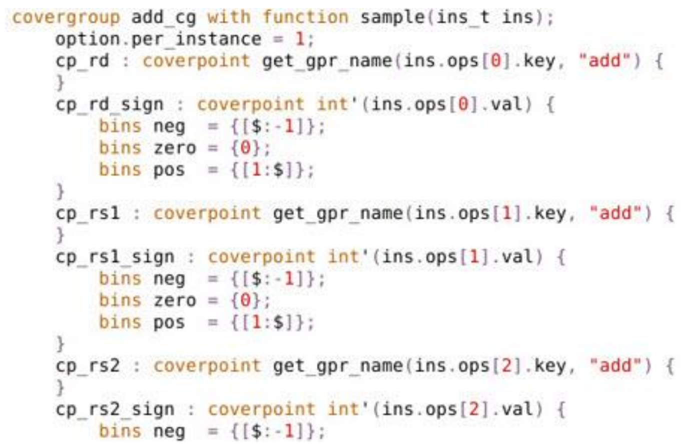
Agenda:

- SystemVerilog functional coverage overview
  - Interfaces, content, tools
- Requirements on the testbench approach
- Requirements for RISC-V processor functional coverage
- Generation flow
- Examples in use

SystemVerilog functional coverage overview

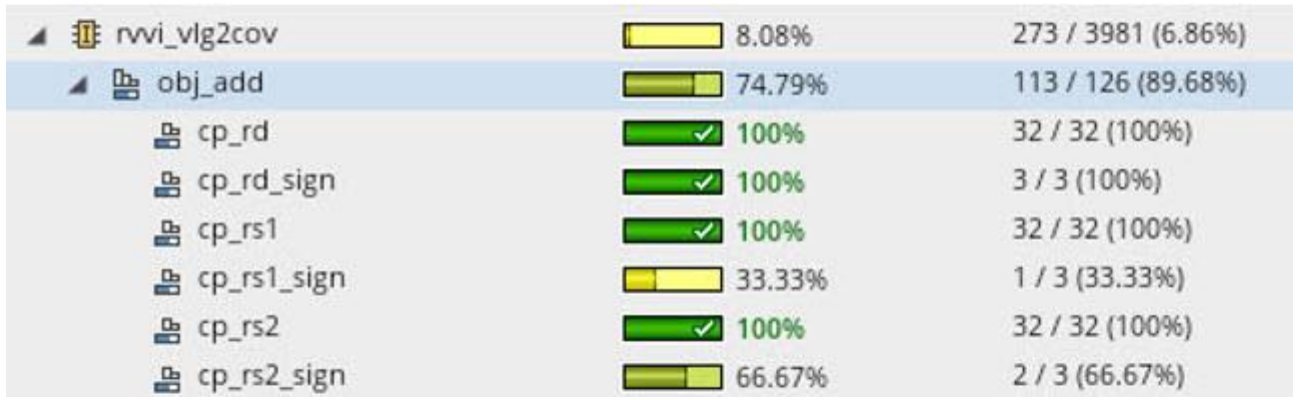


- Coverpoints & covergroups in SystemVerilog source in testbench
- Connects via interfaces into core through ‘tracer’
- Core execution events trigger sampling of coverpoints to collect data on state of core
- SystemVerilog simulator & tools from e.g. Cadence, Synopsys, Siemens EDA, Metrics create coverage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

Automating the generation of the testbench Functional Coverage measuring components

Agenda:

- SystemVerilog functional coverage overview
- Requirements on the testbench approach
- Requirements for RISC-V processor functional coverage
- Generation flow
- Examples in use

Requirements on the testbench approach

- Functional coverage only tells you what was seen in the test bench and DUT
- It does not tell you that the DUT actually did what you expected…
- That is what the reference model, scoreboards, comparators, and other VIPs do

- For the DV to be good (and the coverage to be meaningful) – the test bench has to include appropriate components
- Currently there are several test bench approaches
  - Self test
  - File compare (e.g. signature) (post-simulation)
  - Log trace compare (post-simulation)
  - Lock-step-compare (co-simulation)
    - Synchronous events
    - Asynchronous events
- It is important to use an appropriate test bench for the level of verification required
  - For example, … if you need to test async behavior you need async-lock-step-compare (co-sim) approach…

Automating the generation of the testbench Functional Coverage measuring components

Agenda:

- SystemVerilog functional coverage overview
- Requirements on the testbench approach
- Requirements for RISC-V processor functional coverage
- Generation flow
- Examples in use

Requirements for RISC-V processor functional coverage

- Basic ISA
  - Instructions and un-privilege execution
  - Standard basic processor state and privilege modes
- Customer core design & micro-architectural features
  - Privilege and implementation choices: CSRs, Interrupts, Debug block, …
  - Interrupt events, asynchronous behaviors and debug modes, …
  - Pipeline, multi-issue, multi-hart, Out-of-Order, …
- Quantities…
  - 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 Crypto-scalar: 85
    - P-DSP: 318
    - For RV64 that is 967 instructions…
- And for each instruction you will need to write SystemVerilog covergroups and coverpoints…
  - Maybe 10-40 lines of SystemVerilog for each instruction…

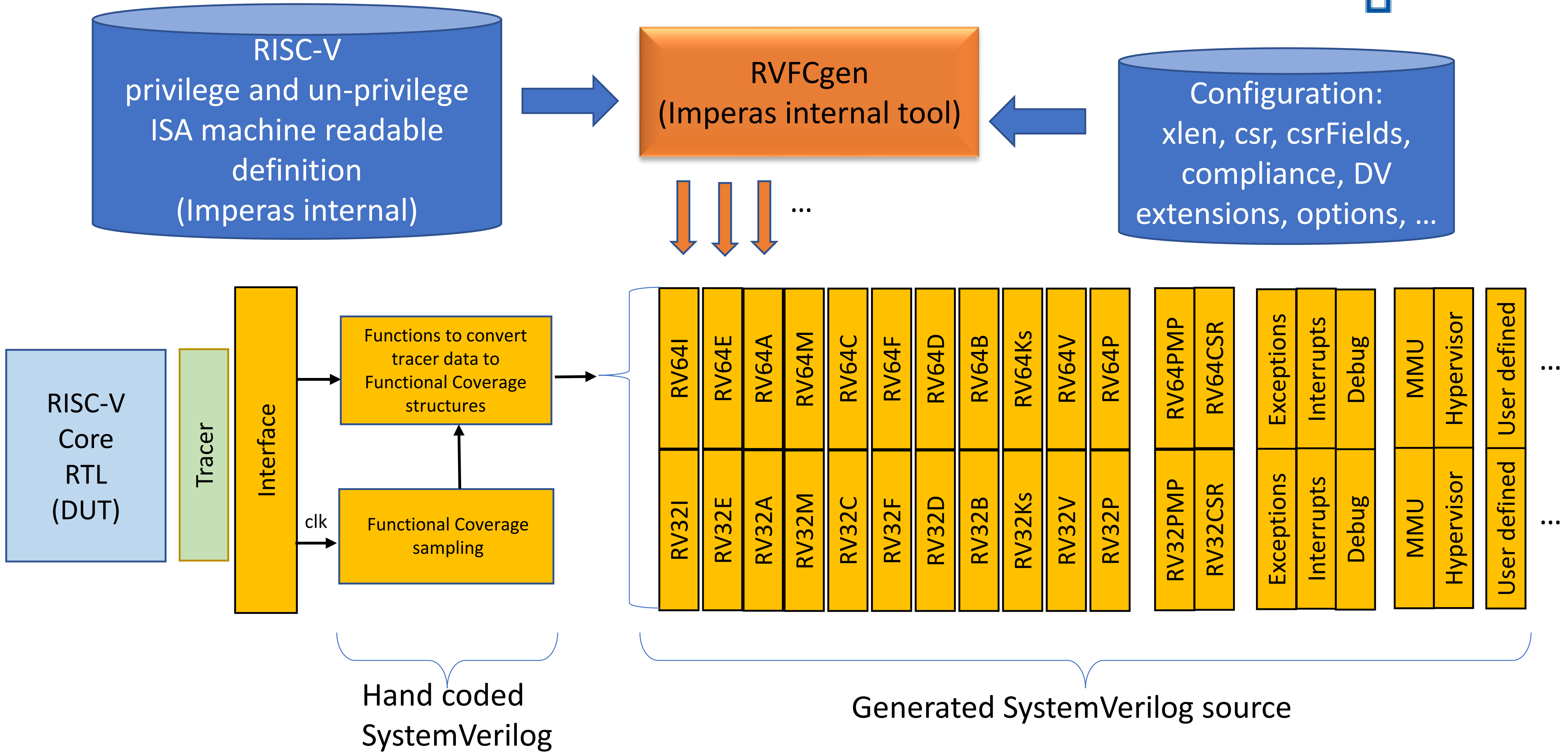
⇒ 10,000-40,000 lines of SystemVerilog code to be written… (and be correct and working…)
  - And that is just for the basic un-privilege mode ISA…

Automating the generation of the testbench Functional Coverage measuring components

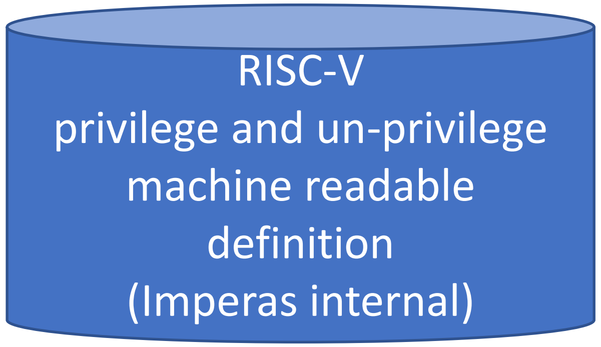
Agenda:

- SystemVerilog functional coverage overview
- Requirements on the testbench approach
- Requirements for RISC-V processor ISA functional coverage
- Generation flow
- Examples in use

Generation flow for creating functional coverage source



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’

Automating the generation of the testbench Functional Coverage measuring components

Agenda:

- SystemVerilog functional coverage overview
- Requirements on the testbench approach
- Requirements for RISC-V processor ISA functional coverage
- Generation flow
- Examples in use

Examples in use

- Internally developed instruction coverage for RISC-V Vector Test Suite project (2018-2019)
  - Contract to develop RISC-V Vector Test Suite – we needed to know its coverage
- OpenHW CV32E40P – used source output of first gen of Imperas RVFCgen – RV32I (2020)
- Evolved as part of RISC-V Architecture Compliance group and targeted their coverage requirements
  - Included with all Imperas simulators as binary (not SystemVerilog)
    - riscvOVPsimPlus, … -> …, ImperasDV
  - Provides coverage of all ratified ISA extensions
- 2022 – 2nd gen
  - Results released as SystemVerilog source in Imperas products for first few ISA subsets
  - Released as open source
  - Interface to core tracer now utilizes open standard RVVI-TRACE interface
    - https://github.com/riscv-verification/RVVI
  - Restructured as UVM class libraries
    - So can be user extended in maintainable way
- New OpenHW VTG ARVM-FunctionalCoverage project
  - Focusing on adding DV requirements

Automating the generation of the testbench Functional Coverage measuring components

Agenda:

- SystemVerilog functional coverage overview
- Requirements on the testbench approach
- Requirements for RISC-V processor ISA functional coverage
- Generation flow
- Examples in use
- Summary

Automating the generation of the testbench Functional Coverage measuring components

Summary:

- Generation of verification components can produce significant benefits to DV teams
- Requires machine readable formats as input to generators
- Project in OpenHW VTG is driving requirements for RISC-V processors

- 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 SystemVerilog Functional Coverage source available in 2022

Thank you

More information:

- info@imperas.com
- www.imperas.com

- RISC-V Verification Interface (RVVI)
  - https://github.com/riscv-verification/RVVI
- OpenHW Group
  - https://www.openhwgroup.org