2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 28 - MARCH 3, 2022

Tutorial: Introduction to the 5 levels of RISC-V processor verification

Simon Davidmann and Lee Moore, Imperas Software

imperas

accellera SYSTEMS INITIATIVE

Focus of this tutorial

- RISC-V is changing the options that SoC designers have in their tool kits
- RISC-V means many teams are designing new processors, or modifying source of processors
- For RISC-V anybody can be ‘an architecture licensee’
- And every CPU needs verifying… in detail… (its not like buying in pre-verified IP)
- Many people are new to CPU DV for the first time
  - Traditionally done behind closed doors in commercial/proprietary companies

- This presentation aims to introduce the main approaches of RISC-V CPU DV
- And discusses pros and cons of the different approaches
- Also it introduces the main components needed in any RISC-V processor DV environment

RISC-V processor verification tutorial
Items to be covered, main takeaways

- Introduction to RISC-V
- Overview of the issues when verifying the design of a RISC-V CPU
- Different approaches to verifying a RISC-V CPU
- Main components of a verification testbench
- Introduction to simulators for RISC-V CPUs
- Use of various virtual platform components in verification
- Discussion of different reference model requirements
- Introduction to instruction stream generators
- Status of RISC-V compliance and its relationship to verification
- Pointers to some useful architectural validation test suites
- Understanding a complete SystemVerilog testbench via a detailed walk through demonstration

Introduction to Imperas
Involvement with RISC-V

- Imperas develops simulators, tools, debuggers, modeling technology, and models to help embedded systems developers get their software running…
- …and hardware developers get their designs correct
- 14+ years, self funded, profitable, UK based, team with much EDA (simulators, verification), processors, and embedded experience
- Staff worked in Arm, MIPS, Tensilica, Cadence, Synopsys
- and in verification in EDA on development of Verilog, VCS, SystemVerilog, Verisity and their methodologies
- Started work with customers on RISC-V in 2017
- Contributed to RISC-V compliance since 2018, RISC-V DV since 2019
- Our RISC-V focus is CPU verification
- We provide configurable reference models, the fastest highest quality simulators, advanced development tools and the absolute best solution for RISC-V hardware design verification
- 20+ of the leading RISC-V CPU developers use and rely on Imperas solutions

www.imperas.com www.OVPworld.org

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Components of RISC-V CPU DV environment

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Components of RISC-V CPU DV environment

For each topic and item we will try to introduce, explain, even demo technologies and products that are available – to give you a feeling of current state-of-the-art

We will also introduce and walk through some open source solutions

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Components of RISC-V CPU DV environment

RISC-V History

imperas

- RISC-V (pronounced "risk-five") is an open standard instruction set architecture (ISA) that began in 2010 and is based on established reduced instruction set computer (RISC) principles
- Unlike most other ISA designs, RISC-V is provided under open source licenses that do not require fees to use
- The project began in 2010 at the University of California, Berkeley, but now many current contributors are volunteers not affiliated with the university
- Unlike other academic designs which are typically optimized only for simplicity of exposition, the designers intended that the RISC-V instruction set be usable for practical computers

RISC-V == Freedom…

Freedoms enabled by RISC-V are a huge opportunity

RISC-V == Freedom…

Freedoms enabled by RISC-V are a huge opportunity

Freedoms enabled by RISC-V are a huge challenge for verification

the largest change in the industry since? …

Challenges in RISC-V CPU DV

- Feature selection and choices require serious consideration due to implications of every choice
  - Experienced architecture teams know the costs associated with every feature
    - Every addition dramatically increases (doubles ?) verification & compounds verification complexity
    - Costs of simple added feature can be huge – and unknown to inexperienced teams
    - Adds schedule, resources, quality costs == big risks…
- As of 2021, No off-the-shelf toolkit/products available for DV of processors
  - No EDA vendor has 'RISC-V CPU DV kit' product
  - There are in-house proprietary solutions in CPU developers… Intel, AMD, Arm, …
  - Building your own adds schedule, resources, quality costs – and risks
- Current SoC cost is 50% for HW DV (with CPUs bought in as proven IP)
  - Developing own CPU adds huge DV incremental schedule, resources, quality challenges

Agenda

- Brief Introduction to RISC-V
- Processor DV project timeline
- RISC-V CPU HW DV approaches
- Components of RISC-V CPU DV environment

A CPU HW DV project timeline

- Source/build/hire/allocate the expert team to do the work…
- Focus on what needs to be verified – develop measurement metrics
- Develop Verification Plan (and measurement metrics)
- Determine EDA tools and models and methods to be used
- Simulation choices: open source, commercial, bespoke, SystemVerilog, UVM, FPGA, Emulation, …
- Formal…
- Get tools, verification IP (VIP), testbenches, models in place
- Obtain tests, create tests
- Generate huge number of (constrained random) tests
- Verify, triage and fix issues, continue…
- … continue while measuring until functional and code coverage metrics reached
- Benchmark, soak and integration testing

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Components of RISC-V CPU DV environment

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Components of RISC-V CPU DV environment

During this section, several components will be mentioned… like ISS, and ISG – these may be introduced as we go, or may be discussed in more detail in later sections of this tutorial

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
- Components of RISC-V CPU DV environment

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
- Components of RISC-V CPU DV environment

Note that not all projects have the same requirements, schedule or verification needs – so each project’s DV needs may / will differ

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
- Components of RISC-V CPU DV environment

#0: 'Hello World' DV

- "if I can get a program to run – then my DV is done… right?"
- "my DV challenge is sorted if I can get Linux to boot on my design…"

- Basically this level of DV is where developer feels if they can get their current compilation of their current program to run (through one path) - then their silicon design job is done

- This may be fine for test chips, research, academic, hobbyists, but NOT for products
- This approach is often due to lack of knowledge or interest in quality, …

‘Hello World’ DV

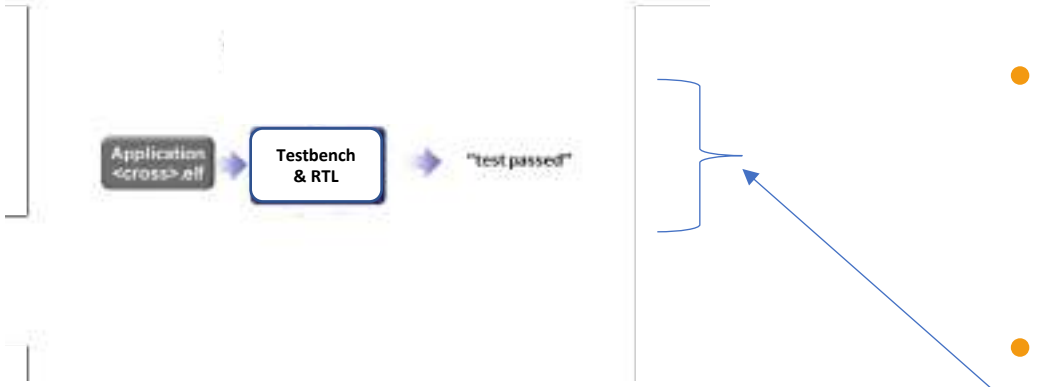
- This requires either an HDL simulator, or an FPGA, or some silicon, and a test harness of some form to allow it to run programs…

- This is not DV!

Agenda

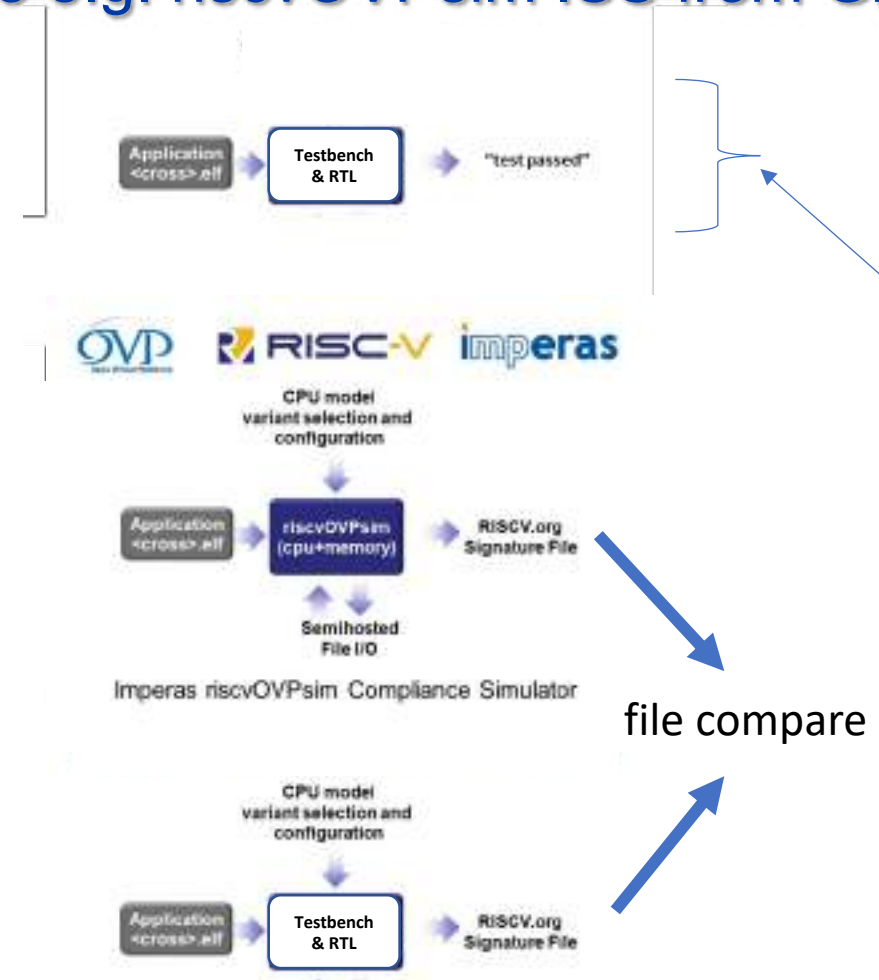
- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
- Components of RISC-V CPU DV environment

#1: Simple (results) check (1a)



- Run RTL DUT in testbench
  - (no real testbench)
  - Just loads & runs the test program
- Each test program checks its results = go/no go test
  - Prints message to log
  - or writes bit to memory

#1: Simple (results) check (1b)
(use e.g. riscvOVPsim ISS from GitHub)



file compare

- Run RTL DUT in testbench
  - (no real testbench)
  - Just loads & runs the test program
- Either
  - Each test program checks its results = go/no go test
    - Prints message to log
    - or writes bit to memory
- Or, then run ISS, write log or signature file
  - Compare/diff file results (afterwards)
  - This is the approach taken by RISCV International for their architectural validation ("compliance tests")

#1: Simple check
(use e.g. riscvOVPsim ISS from GitHub)

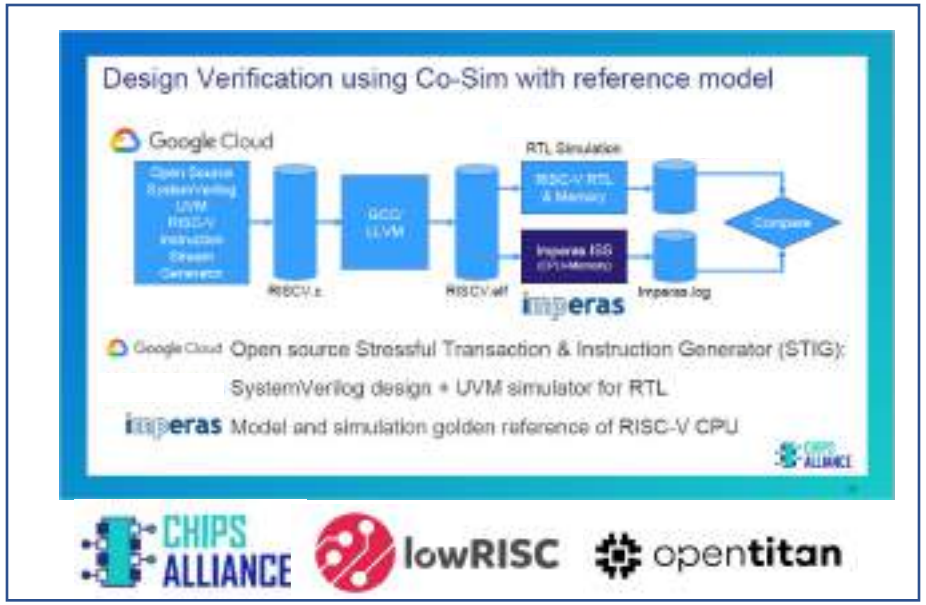
- Summary
  - Very simple, needs basic ISS, and tool chains
    - Free ISS: https://github.com/riscv-ovpsim
    - Free compiler: https://github.com/Imperas/riscv-toolchains
  - Basic bring up
  - Good for simple test runs
  - Basic functionality testing
  - Still need accurate, configurable, version selectable, complete, reference model

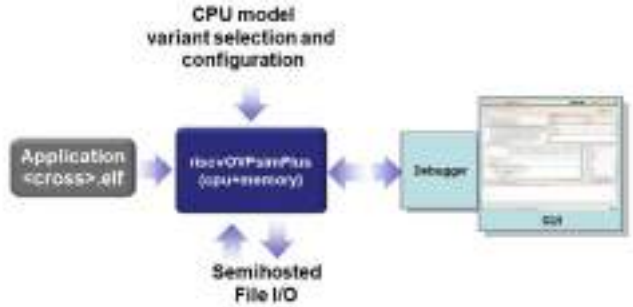
  - Not a robust DV solution

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
- Components of RISC-V CPU DV environment

#2: Entry Level DV: post-sim trace-compare
(use e.g. riscvOVPsimPlus ISS from OVPworld)



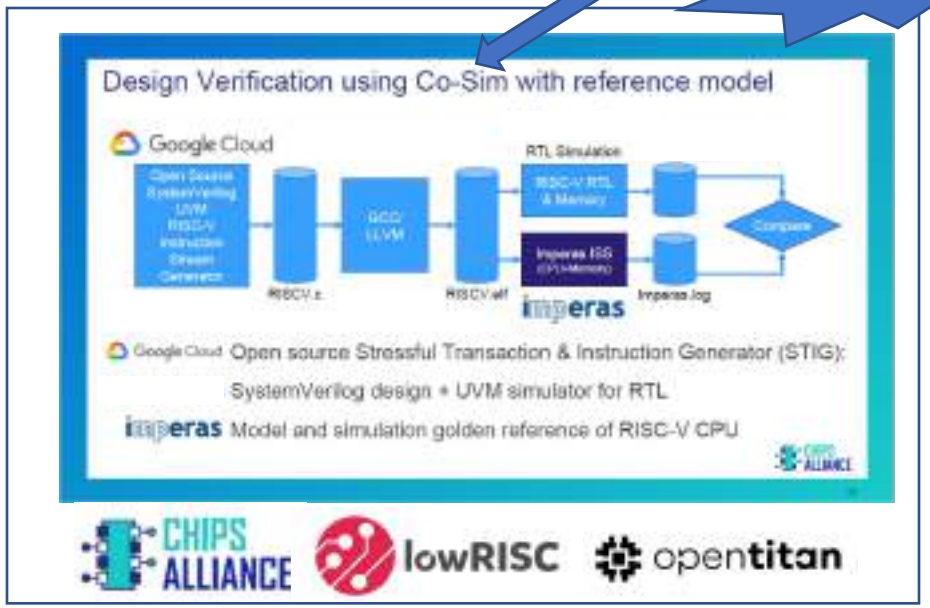


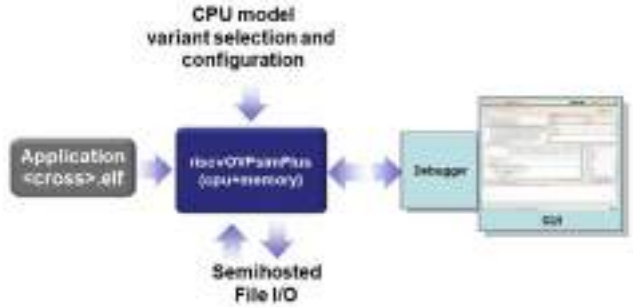
Imperas riscvOVPsimPlus Reference Simulator

- Process
  - use random generator (ISG) to create tests
  - during simulation of ISS write trace log file
  - during simulation of RTL write trace log file
  - at the end of both runs, run logs through compare program to see differences / failures
- ISS: riscvOVPsimPlus includes Trace and GDB interface
  - Free ISS: https://www.ovpworld.org/riscvOVPsimPlus
- ISG: riscv-dv from Google Cloud / Chips Alliance
  - Free ISG: https://github.com/google/riscv-dv

#2: Entry Level DV: post-sim trace-compare
(use e.g. riscvOVPsimPlus ISS from OVPworld)

Note: This is not co-sim…





Imperas riscvOVPsimPlus Reference Simulator

- Process
  - use random generator (ISG) to create tests
  - during simulation of ISS write trace log file
  - during simulation of RTL write trace log file
  - at the end of both runs, run logs through compare program to see differences / failures
- ISS: riscvOVPsimPlus Includes Trace and GDB interface
  - Free ISS: https://www.ovpworld.org/riscvOVPsimPlus
- ISG: riscv-dv from Google Cloud / Chips Alliance
  - Free ISG: https://github.com/google/riscv-dv

#2: Entry Level DV: post-sim trace-compare (use e.g. riscvOVPsimPlus ISS from OVPworld)

Summary

- Compares files created after test runs
- Can be signature, logging, or instruction trace
- Can use random ISG as no need to know expected results…
- Usually the easiest method to implement (dependent on tracing formats)
  - Capture of program flow (monitor the PC)
  - Capture of program data (monitor the Registers)

- Potentially very large data files
- Potential for wasteful execution (if early failure)
- Will not work for on async events, control flow, or hardware real time effects, MP, OoO, multi-issue, …
- Not a robust DV solution for commercial cores

- Can engage with Imperas for licenses of reference models and optional development to add customer own instructions, CSR, behaviors

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
- Components of RISC-V CPU DV environment

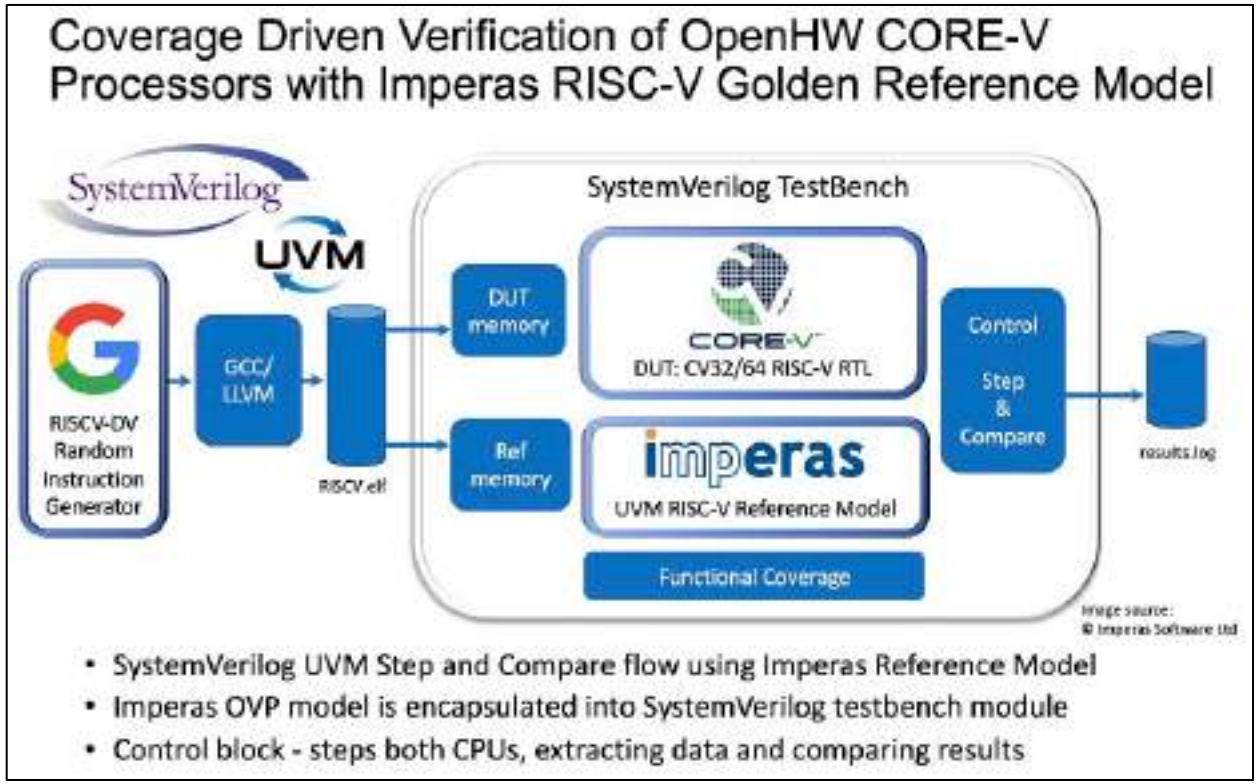
#3: Industrial Quality Sync DV (sync-lock-step-compare)

- Tandem lockstep run – both reference and DUT run together in lock step
- Not very complex to obtain, set up
- Compare PC, CSRs, GPRs, other internal state – instruction by instruction
- No requirement on data saving
- No requirement on known good results in test

- Will not work for async events and control flow , … – it is all about the data flow

- [OpenHW evolved into using async – see later slides]

Example flow:



1st Generation OpenHW flow (1H2020)

#3: Industrial Quality Sync DV (sync-lock-step-compare)

Summary

- Instruction by instruction lockstep comparison (excludes async events)
  - Comparison of execution flow
  - Comparison of program data
  - Comparison of programmers and internal state
- Immediate comparison
  - Allows for debug introspection at point of failure – very powerful
  - Does not waste execution cycles after failure
- Will not work for async events, control flow, or hardware real time effects, …
- Not too hard to develop & set up (depends on RTL DUT tracer features)
- Lock-Step / Compare is by far the best and most efficient approach
  - But does not address async events (see level #4)

- Need to engage with vendors such as Imperas for licenses of reference models and optional development to add customer own instructions, CSR, behaviors

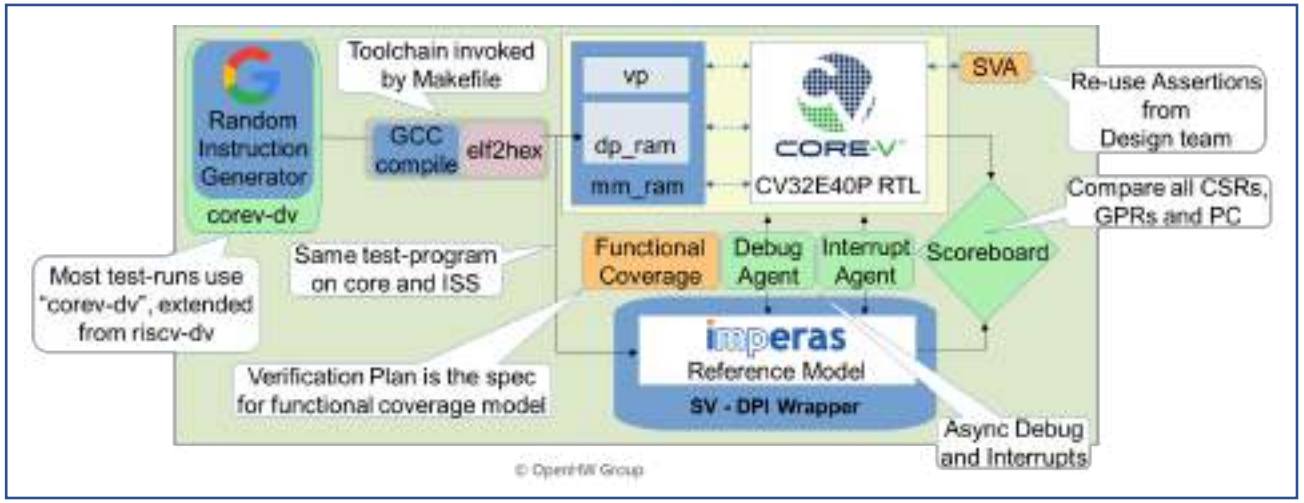
Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
- Components of RISC-V CPU DV environment

#4: Industrial Highest Quality Async DV (async-lock-step-compare)

- Builds on & extends Industrial Quality sync-lock-step-compare DV
- Adds focus on async capabilities
- Depending on design this can include: OoO, MP, debug mode, interrupts, multi-issue, …
  - Example SystemVerilog Components
    - tracer: Reports instructions for checking and register writebacks
    - step_and_compare: Manages the reference model and checks functionality
    - interrupt_assert: Properties for interrupt coverage/checking
    - debug_assert: Properties for debug coverage/checking
- Typically hard, complex, and expensive to get working
  - Challenge is extracting async info from micro-architecture RTL pipeline
    - See latest developments with RVVI and ImperasDV

Example flow:



2nd generation CV32E40P OpenHW flow (2H2020)
(Imperas model encapsulated in SystemVerilog)

#4: Industrial Highest Quality Async DV
(async-lock-step-compare)

Summary

- Instruction by instruction lockstep comparison (**includes async events**)
  - Comparison of execution flow, of program data, of programmers and internal state
- Immediate comparison
  - Allows for debug introspection at point of failure – very powerful
  - Does not waste execution cycles after failure
- Includes focus on async events, control flow, and hardware real time effects
- Can be hard to develop & set up (depends on RTL DUT tracer features and pipeline understanding)
  - See latest development for RVVI and ImperasDV
- Can be expensive in terms of time, resources, licenses and costs a lot per bug found
  - But the bugs are even more expensive if not found early enough…
- Lockstep / Compare is by far the best and most efficient approach (industry ‘gold standard’)

Agenda

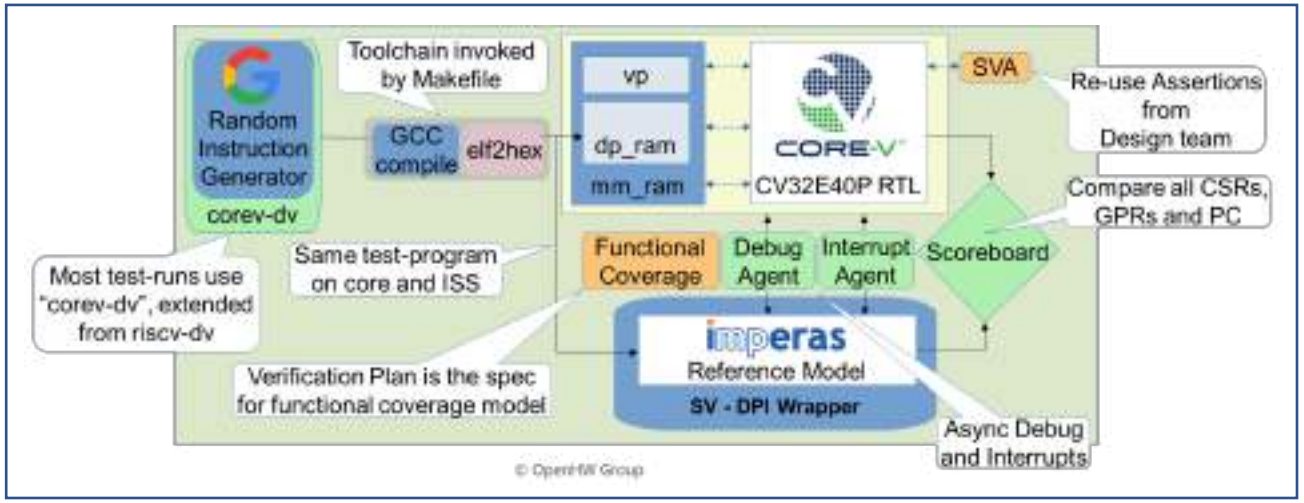
- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
- Components of RISC-V CPU DV environment

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
    - Digression into why we need standards…
- Components of RISC-V CPU DV environment

Challenges moving forward – the need for standards

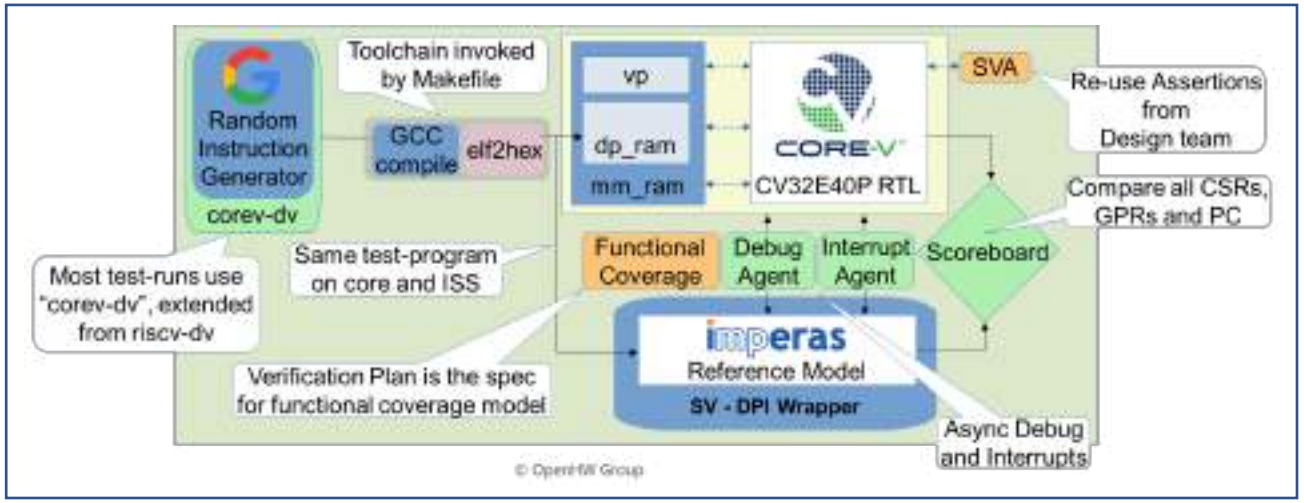
- There are many different components needed:
  - DUT & its encapsulation
    - ‘tracer’ information
    - Control
  - Reference model & its encapsulation
    - Configuration
    - Comparisons
    - Synchronization
    - Asynchronous operations
    - Control
  - Functional coverage measurement & assertions
  - Test bench
    - Configuration
    - Overall control
    - Scoreboarding
    - Reporting / Logging
  - Tests (directed or generated)
    - Program linker scripts and binary file reader
- And each component has different interfaces and requirements on the interfaces



2nd generation CV32E40P OpenHW flow (2H2020)

Challenges moving forward – the need for standards

- There are many different components needed:
  - DUT & its encapsulation
    - 'tracer' information
    - Control
  - Reference model & its encapsulation
    - Configuration
    - Comparisons
    - Synchronization
    - Asynchronous operations
    - Control
  - Functional coverage measurement & assertions
  - Test bench
    - Configuration
    - Overall control
    - Scoreboarding
    - Reporting / Logging
  - Tests (directed or generated)
    - Program linker scripts and binary file reader
- And each component has different interfaces and requirements on the interfaces



2nd generation CV32E40P OpenHW flow (2H2020)

It would be a disaster for RISC-V if every design team had to re-invent everything…

Why need standard interfaces?

- There are many blocks required in DV solutions
  - They all have different interfaces and these interfaces need defining
  - They may come from different developers / suppliers
  - They may be used in different projects and processor configurations / generations
- If standards exist, then verification IP can be created and licensed
- Goals when developing standard interfaces:
  - block re-use
  - common ways to do things
  - quick start-up
  - efficiency

Why adopt a standard?

- You have to use some interfaces
- No need to re-invent on your own – they do not need to be proprietary
- RVVI is an open standard available on GitHub
- RVVI (and its predecessor) have already been flushed out and are in use
- There is no downside to adoption
- Upside to adoption: potentially make use of other tools / code
- What tools / technologies can potentially be (re)used
  - Encapsulation of reference models
  - Test benches & test bench components (including onward connection to reference models)
  - Functional coverage & assertions
  - Log file writer
  - Signature file writer (for RISC-V compliance testing)

RVVI: RISC-V Verification Interface
History

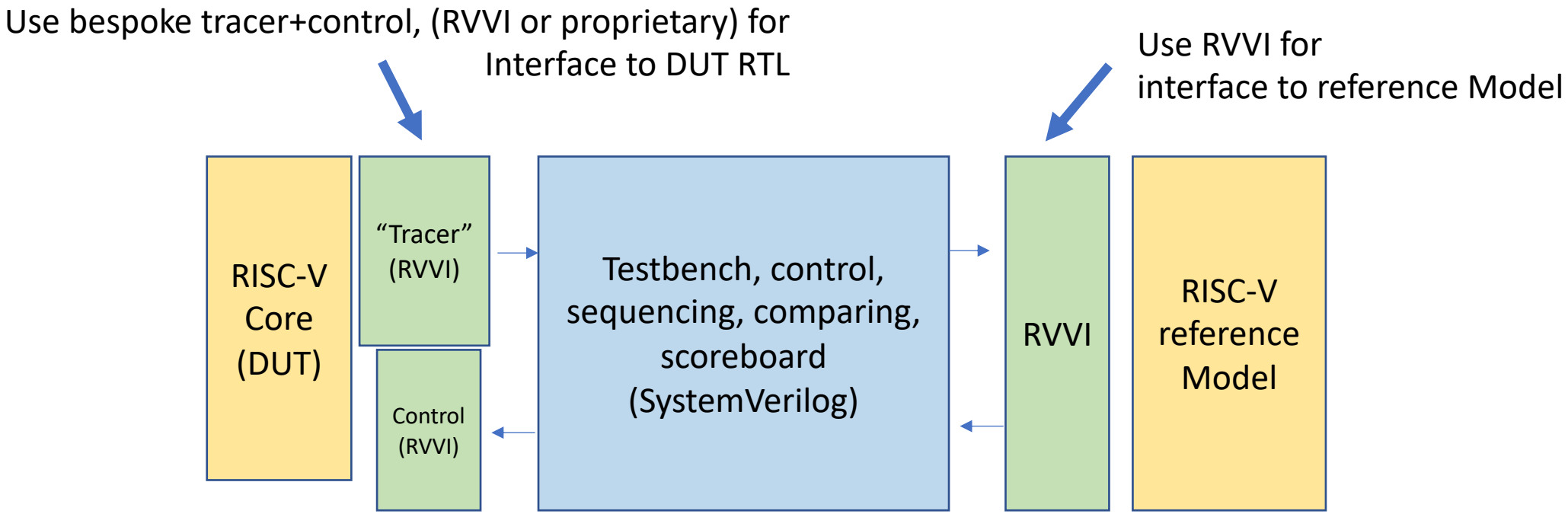
- This RVVI work has evolved from over 2 years experience
  - Imperas, EM Micro, and SiLabs, …, working with several RISC-V verification projects
  - Collaboration with OpenHW Group (https://github.com/openhwgroup/core-v-verif)
    - Re-usable test bench for Core-V range of open-source RISC-V cores

- Also… there was previously the RISC-V Formal Interface (RVFI) – targeting formal tools
  - https://github.com/SymbioticEDA/riscv-formal
    - Interface for providing observation into a running core by streaming what is executing on the core (i.e. the basic trace data / functionality)
  - **For quality RISC-V processor DV more is needed (than RVFI)**
    - **And each user needs to extend it with proprietary extensions (which is not the right approach…)**
  - **The RVVI-VLG interface has some parts very similar to the RVFI**
    - **RVVI-VLG can be thought of as an updated superset of RVFI**

Agenda

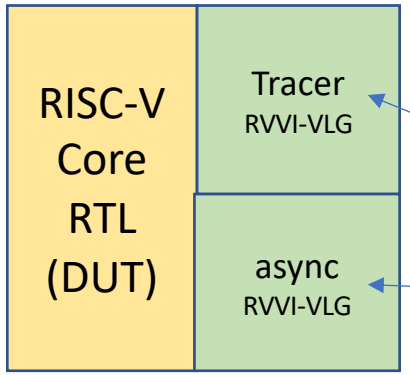
- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
- Components of RISC-V CPU DV environment

#5 Evolving to use developing standards (RVVI)

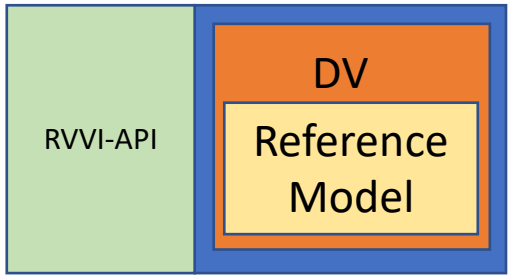


- Focus is on developing standard interfaces between components
  - Allows reuse
  - Allows development of independent VIP
- Two main components to consider
  - DUT
  - Reference model

RVVI: RISC-V Verification Interface (driven by RISC-V DV usage)



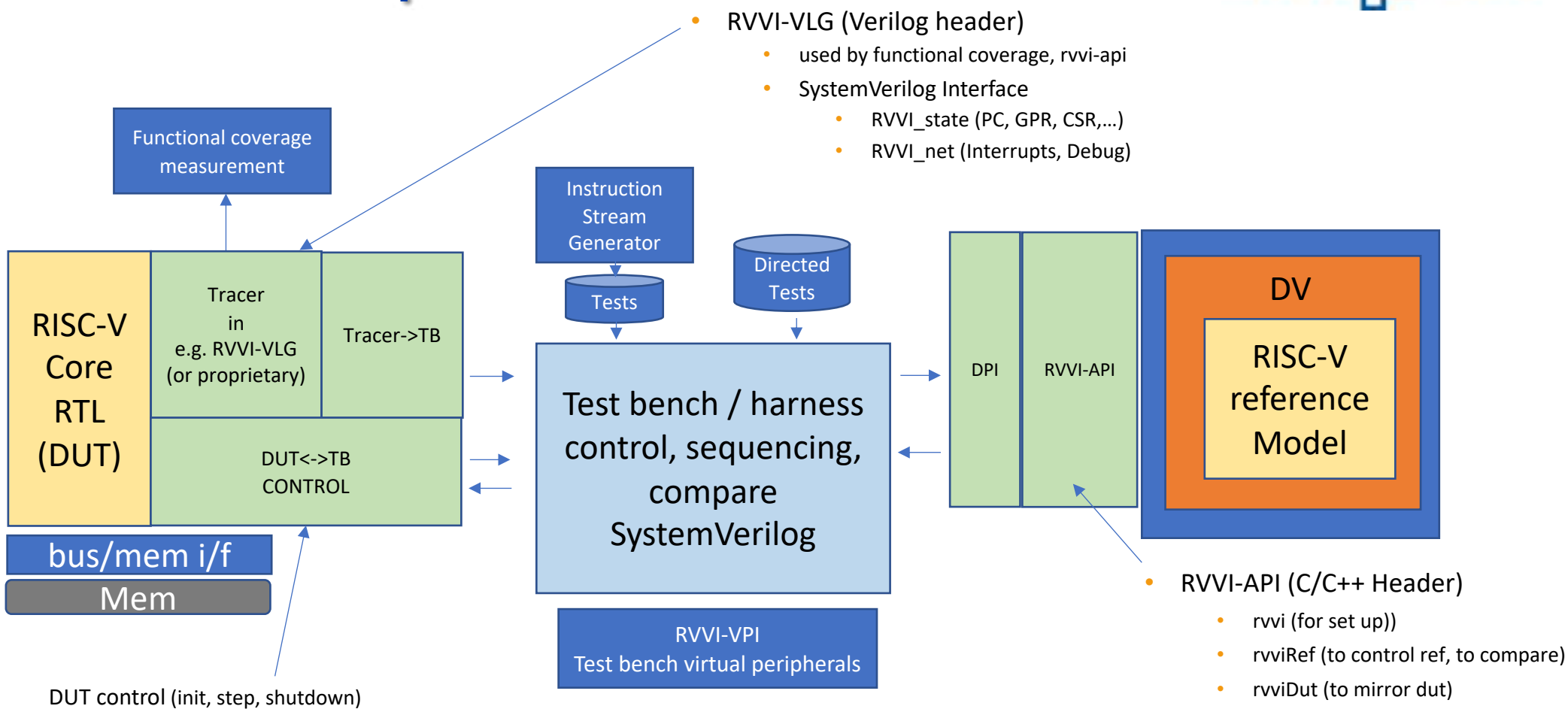
- https://github.com/riscv-verification/RVVI (Public Open Standard)
- RVVI-VLG
  - Verilog DUT interfaces
    - RVVI-VLG state – streaming 'tracer' data
    - RVVI-VLG nets – implementation dependent (Interrupts, Debug)
    - Handles multi-hart, multi-issue, Out-of-Order



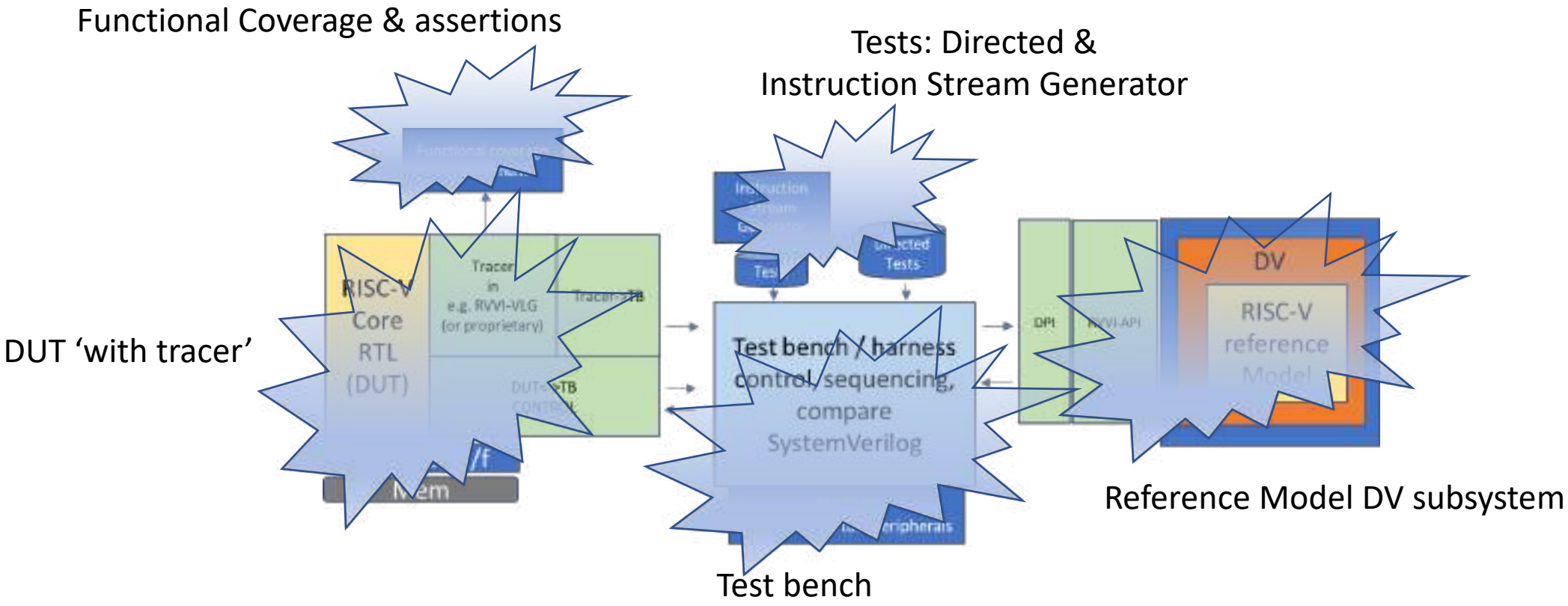
- RVVI-API
  - Controls DV subsystem and reference model
    - C/C++
    - SystemVerilog

- RVVI-VPI (work-in-progress (Feb. 2022))
  - Virtual Peripheral Interfaces
    - timers, interrupts, debug, random, printer/uart, …
  - Verilog and C macros & examples

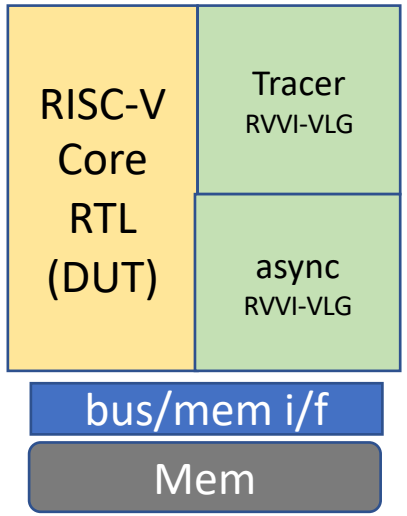
SystemVerilog test bench using RVVI and components



5 main CPU DV components



RTL DUT with ‘tracer’ interface



- The key component – the DUT being tested
  - Includes memory model and bus interfaces
- Requires a ‘tracer’ to provide appropriate data to the test bench
- Requires control interface so test bench can step through events
- Quality of the ‘tracer’ determines the potential capabilities of the DV
- Can be RVVI, bespoke, or bespoke + extensions + control

Tests: Directed & Instruction Stream Generator

(random) Instruction Stream Test Generator

- Generates test programs
- Usually using constrained random approach
- Most often obtain one:
  - Commercial such as Valtrix STING
  - Open source e.g. Google riscv-dv (written in SystemVerilog)

Directed Tests

- Acquire suites of tests
  - Imperas make some available
- May require toolchains like GCC, LLVM for assemblers, linkers

Imperas Test Suites

- When verifying a CPU design - you can never have enough tests...
- Imperas have developed a directed RISC-V test generator, instruction coverage measuring VIP, and a test qualifying mutating fault simulator to provide high quality test suites
- The generated tests suites are targeting architectural compatibility as defined in the RVIA architectural test working group coverage requirements
- There are currently over 50 free test suites, including
  - I,M,C,F,D,B,K,V,P
    - The provided vector test suite is one specific vector engine configuration
- The test suites are provided under an OVP open source license and are available free from: https://github.com/riscv-ovpsim/imperas-riscv-tests

Functional Coverage & assertions

Functional coverage measurement

- Normally written in SystemVerilog using covergroups, coverpoints, and assertions
- Targets specific measurements as required in verification plan
- Connects to RVVI-VLG from ‘tracer’
- Typically includes
  - Standard ISA instruction extension measurement
  - Sequential instruction monitoring for e.g. hazards, etc.
  - Privilege model items such as interrupts, exceptions, debug mode
  - User specifics related to pipeline and micro-architecture
- Requires SystemVerilog simulator

Test Bench / Harness

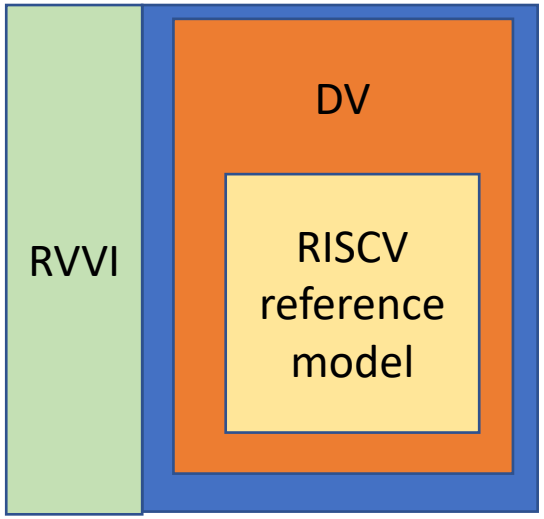
imperas

Test bench / harness
control, sequencing,
compare
SystemVerilog

RVVI-VPI
Test bench virtual peripherals

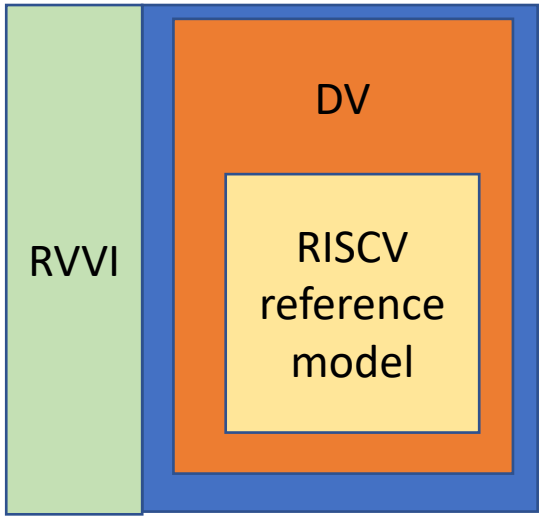
- Instances and connects all the subsystems
- Controls the stepping of events and instructions
- Connects the data & signals between the DUT and reference subsystem synchronizers and comparators
- Can have virtual peripherals such as uart for logging, or timers for asynchronous event / interrupt generation
- Can be in SystemVerilog for DUT in RTL
- Can be in C/C++ for DUT in C based compiled simulator
- Imperas provides templates in C/C++ and SystemVerilog

Reference model DV Subsystem



- Three main components
  - RISC-V reference model
  - DV infrastructure to control model, etc.
  - RVVI interface into test bench / harness
- Responsible for
  - Configuration of model
  - Comparisons between DUT and reference state
  - Synchronization due to pipeline affects
  - Asynchronous operations
  - Control of model

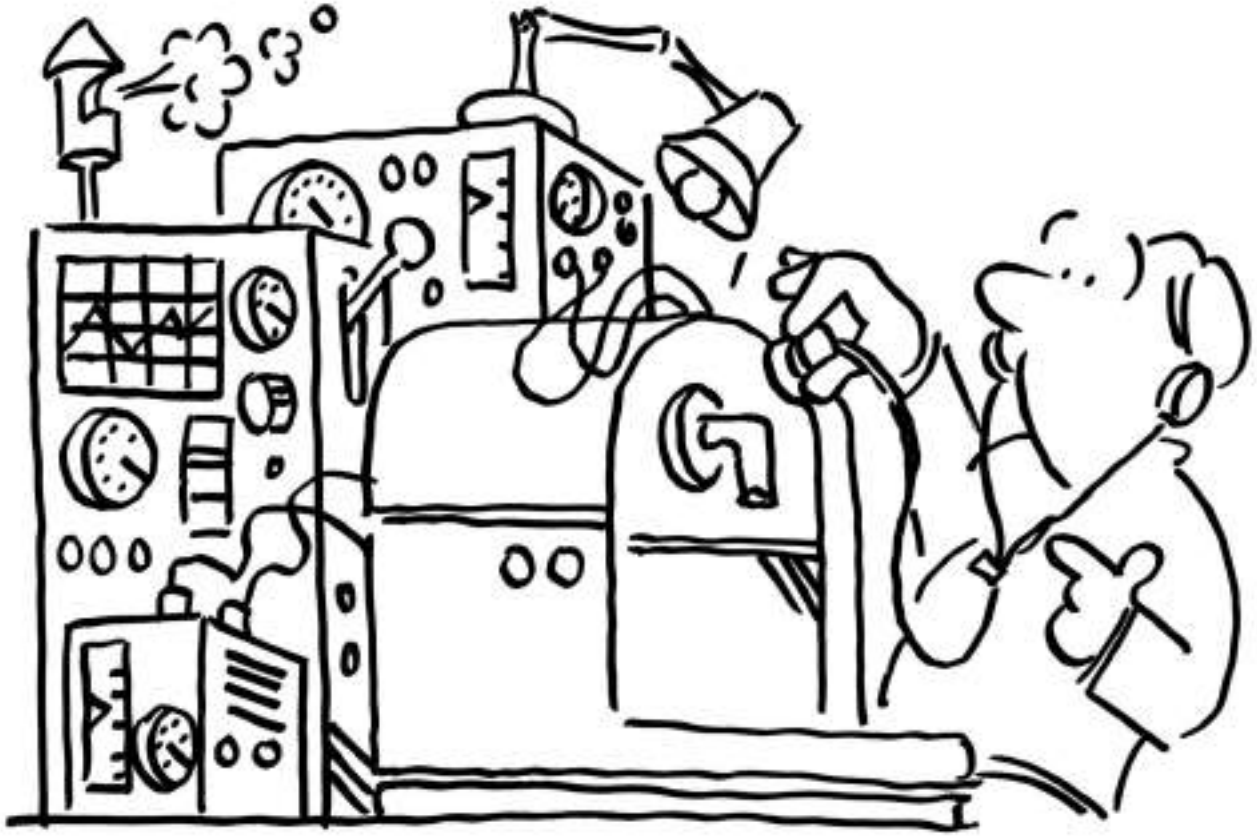
Reference model DV Subsystem



- Three main components
  - RISC-V reference model
  - DV infrastructure to control model, etc.
  - RVVI interface into test bench / harness
- Responsible for
  - Configuration of model
  - Comparisons between DUT and reference state
  - Synchronization due to pipeline affects
  - Asynchronous operations
  - Control of model

Key component is Reference Model

- RISC-V is highly configurable & extendable
  - 200… Questions ?
- So it can get a little …. complicated



Example reference model: Imperas

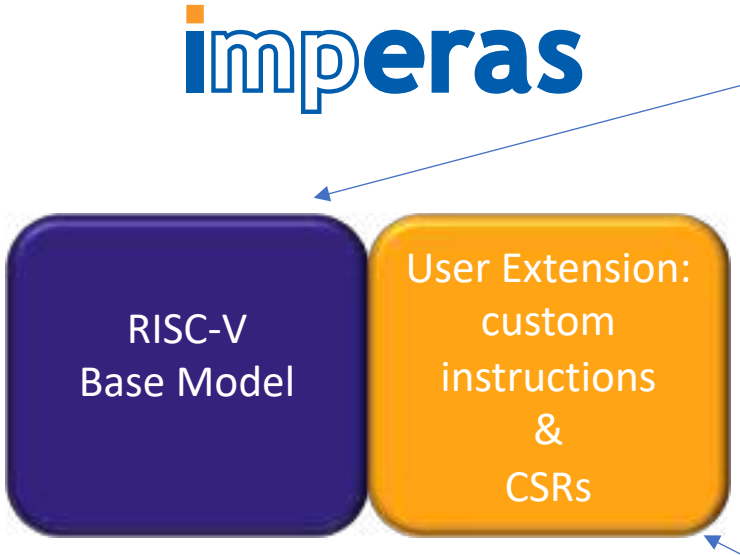
RISC-V Reference Model & Simulator

http://www.imperas.com/riscv

- Imperas provides full RISC-V Specification envelope model
- Industrial quality model /simulator of RISC-V processors for use in compliance, verification and test development
- Complete, fully functional, configurable model / simulator
  - All 32bit and 64bit features of ratified User and Privilege RISC-V specs
    - Unprivileged versions 2.2, 20191213
    - Privilege versions 1.10, 1.11, 1.12
  - Vector extension, versions 0.7.1, 0.8, 0.9, 1.0
  - Bit Manipulation extension, versions 0.91, 0.92. 0.93, 1.0.0
  - Hypervisor version 0.6.1, 1.0.0
  - K-Crypto Scalar version 0.7.1, 1.0.0
  - Debug versions 0.13.2, 0.14, 1.0.0
  - P DSP/SIMD versions 0.5.2, 0.9.6
  - Zicbom, Zicbop, Zicboz, Zmmul, Zfh, Zfinx, Zce
  - Svnapot, Svpbmt, Svinval, Smstaten, Smepmp, …
- Model source included under Apache 2.0 open source license
- Used as reference by :
  - Mellanox/Nvidia, Seagate, NSITEXE/Denso, Google Cloud, Chips Alliance, lowRISC, OpenHW Group, Andes, Valtrix, SiFive, Codasip, MIPS, Nagra/Kudelski, Silicon Labs, RISC-V Compliance Working Group, …

Imperas is used as RISC-V Golden Reference Model

Imperas Model extensibility



- Separate source files and no duplication to ensure easy maintenance
- Imperas or user can develop the extension
- User extension source can be proprietary

Imperas develops and maintains base model

- Base model implements RISC-V specification in full
- Fully configurable to select which ISA extensions
- Fully configurable to select which version of each ISA extension
  - Updated very regularly as ISA extension specification versions change
- Fully configurable for all RISC-V specification options
  - e.g. implemented optional CSRs, read only or read/write bits etc…

Imperas provides methodology to easily extend base model

- Templates to add new instructions
- Code fragments for adding functionality
- 100+ page user guide/reference manual with many examples
  - Includes example extended processor model

Imperas model is architected for easy extension & maintenance

User feedback of Imperas as a reference

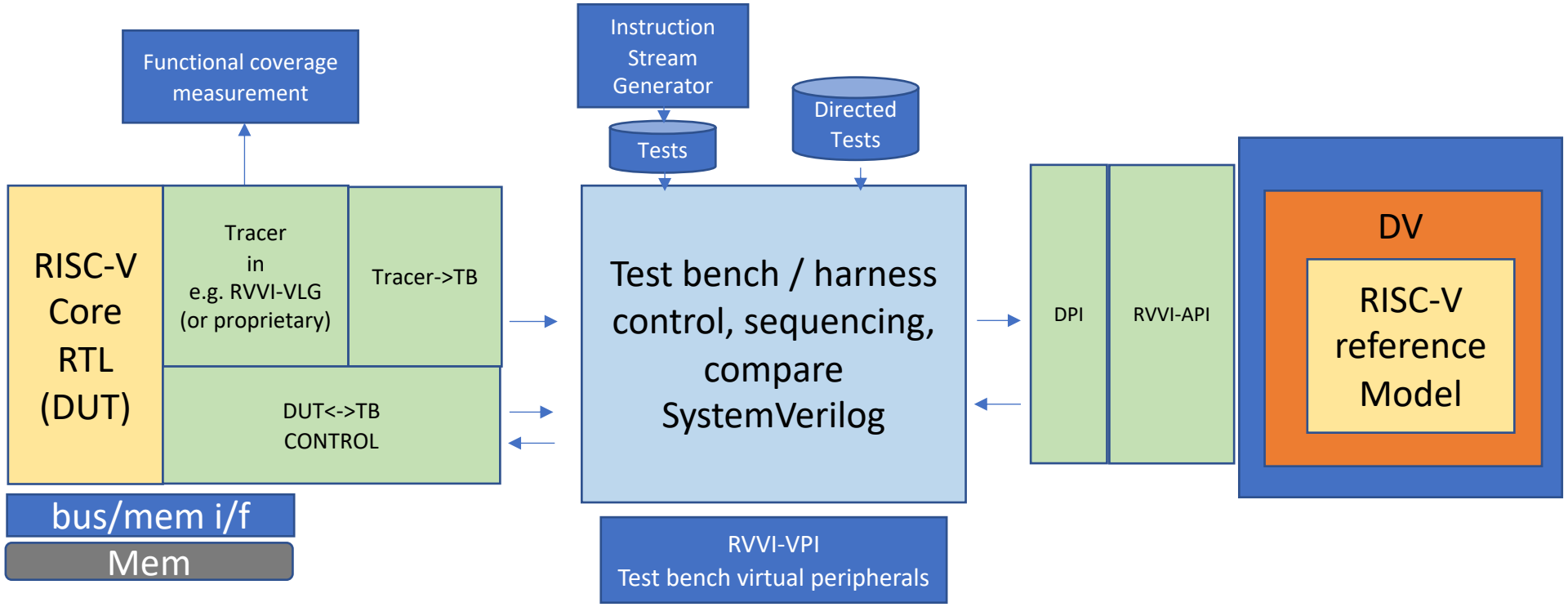
- “Andes is pleased to **certify the Imperas model and simulator** as a reference for the new Vector processor NX27V, and is already actively used by our mutual customers.”
  - **Charlie Hong-Men Su, CTO / EVP at Andes Technology Corp**
- "We have **selected Imperas simulation tools and RISC-V models** for our design verification flow because of the **quality of the models and the ease of use** of the Imperas environment. Imperas reference model of the complete RISC-V specification, the ability to add our custom instructions to the model and their experience with processor RTL DV flows were also important to our decision.“
  - **Shlomit Weiss, Senior Vice President of Silicon Engineering at Mellanox / Nvidia**
- "The OpenHW Group charter is to deliver high quality processor IP cores for our leading commercial members and open source community adoption. Central to this goal, the OpenHW Verification Task Group developed and published a DV test plan and implemented an open engineering-in-progress approach as we complete the verification tasks using the **Imperas golden RISC-V reference model**.“
  - **Rick O’Connor, Founder and CEO at OpenHW Group**
- “Imperas are the pioneers in simulation technology and processor verification for RISC-V. Codasip is very proud of our rigorous approach to verification – **using Imperas as an important part of our quality process** furthers extend our differentiation. The Imperas independence, reputation and technical strength provides our customers with further reassurance in our ‘best in class’ RISC-V processors.”
  - **Philippe Luc, Verification Director Codasip**
- “With this Imperas collaboration, our **mutual customers will benefit from the availability of SiFive qualified models** that are compatible with the mainstream EDA tool flows.”
  - **Phil Dworsky, Director, Strategic Alliances, SiFive**

Agenda

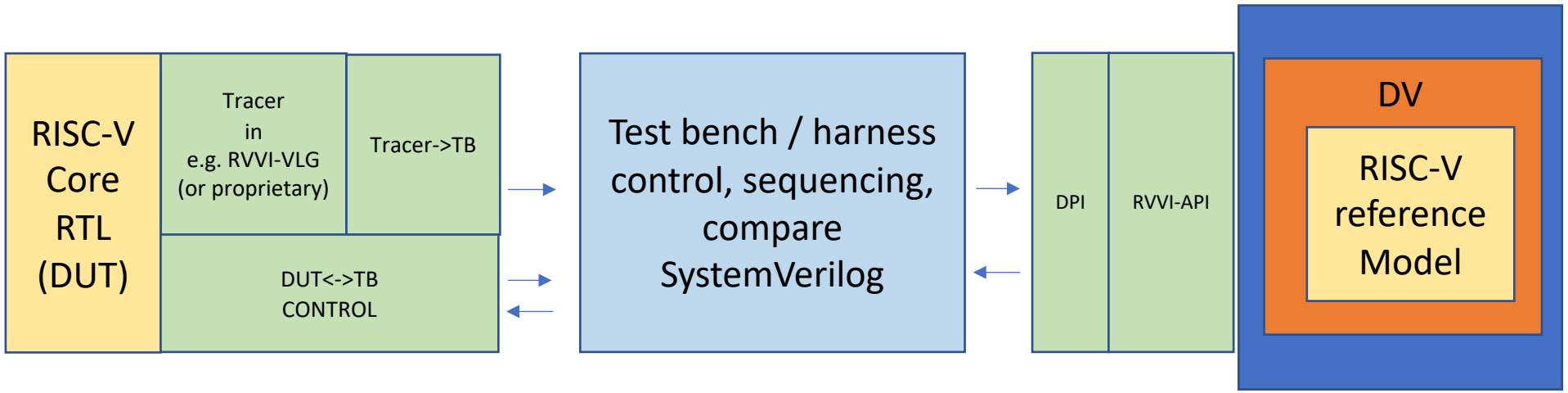
- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
  - #6 using standards based DV products and VIP
    - Drill down into an available commercial RISC-V HW DV verification solution
    - Demonstration of ImperasDV in action on open source RISC-V cores
- Components of RISC-V CPU DV environment

A dedicated RISC-V CPU DV solution: ImperasDV from Imperas

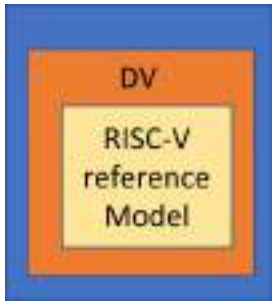
CPU DV test bench components



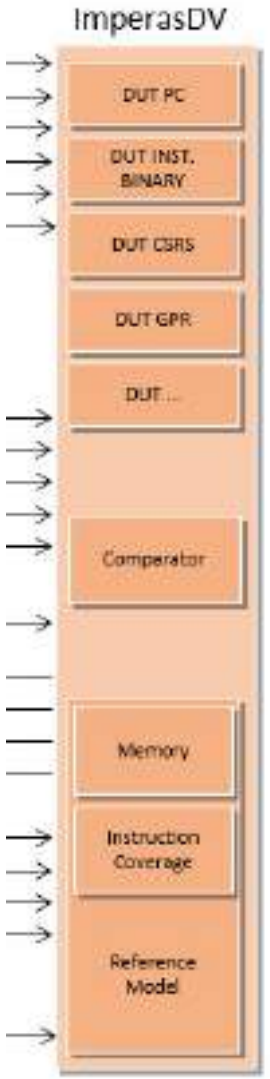
CPU DV test bench components



- Focus on:
  - DUT + 'tracer'
  - Test bench
  - DV subsystem



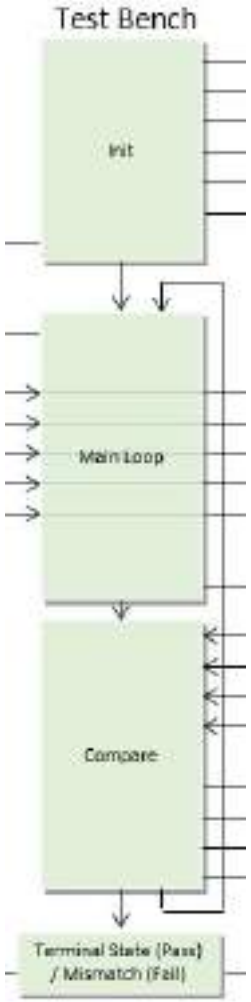
ImperasDV



- Encapsulates the reference model
  - Select model, use variant, configure,
- Includes DUT reference state storage
- Includes synchronization technology
  - Can run sync, async, interrupts, debug, multi-hart
- Includes comparison technology
- Includes Imperas instruction coverage
- Is configurable and traceable
- Can be used in C/C++ or SystemVerilog test bench / harness
  - Uses RVVI-API
- Very simple to use – the ‘smarts’ are built-in

Test bench / harness control, sequencing, compare (SystemVerilog, C or C++)

Test Bench / Harness

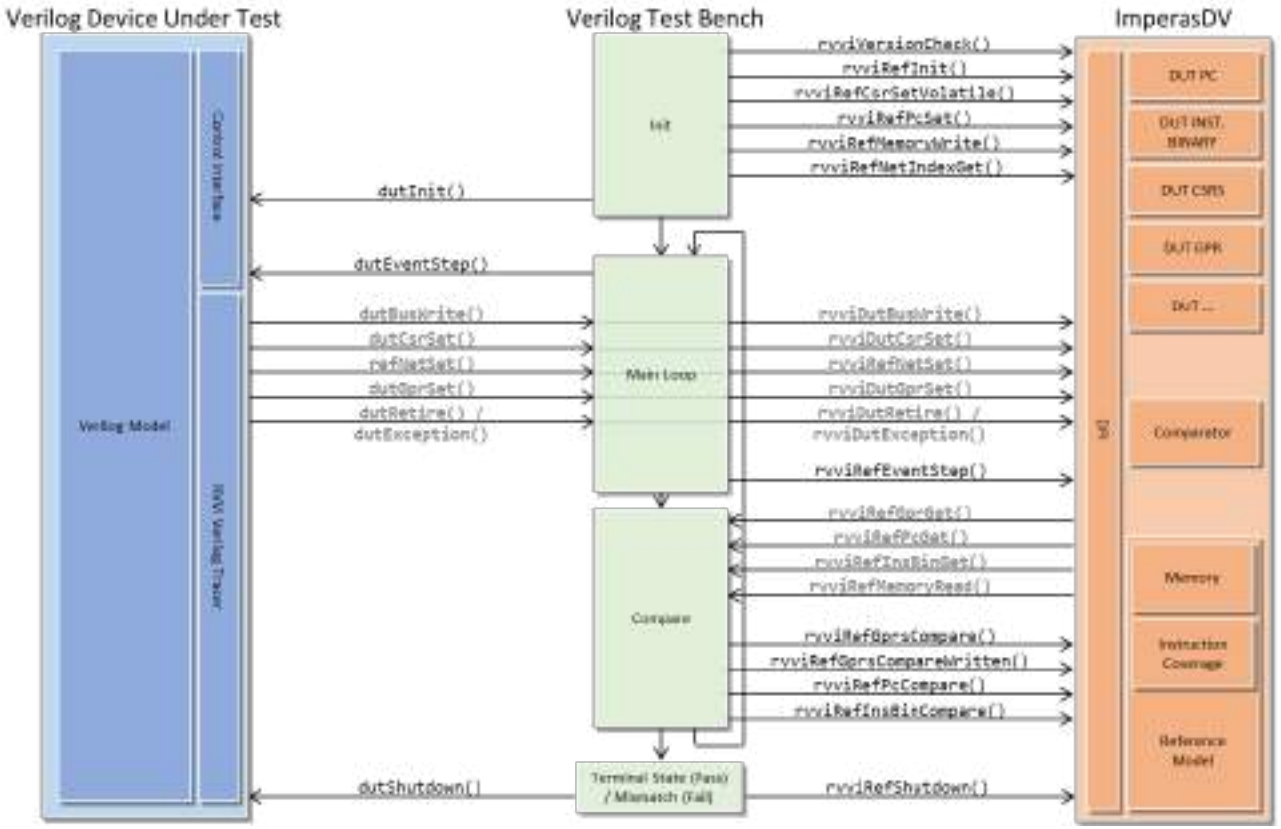


- Instances and connects all the subsystems
- Controls the stepping of events and instructions
- Connects the data & signals between the DUT and reference subsystem synchronizers and comparators

- Can be in SystemVerilog for DUT in RTL
- Can be in C/C++ for DUT in C based compiled simulators

- Imperas provides templates in C/C++ and SystemVerilog
  - Expect users to extend and customize

ImperasDV setup



- Reference model setup
- Configuration of register and memory initialization
- Selection of what to compare (depends on DUT ‘tracer’ capabilities) :
  - PC, GPR, CSR, FPR, VR, decode, net, hart…
- Select capabilities:
  - sync-lock-step-compare or async-lock-step-compare
- Trace and logging set up
- Selection of built-in Imperas instruction coverage
- Choice of DV control options

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Drill down into an available commercial RISC-V HW DV verification solution
- Demonstration of ImperasDV in action on open source RISC-V cores
  - SystemVerilog test bench
- Components of RISC-V CPU DV environment

Demo: ImperasDV
Core: lowRISC Ibex
Simulator: SystemVerilog
DV mode: sync-lock-step-compare

- Overview block diagram from RVVI github
- Walk through C/C++ rvvi.h and in doxygen – introduce APIs: RVVI-VLG
- Walk through tracer code where it converts RVVI-VLG nets to -> RVVI-API
- Walk through SystemVerilog harness
  - Show init, config, main step loop
- Run example - passes
- Run example - fails, show trace, show in eGuiMPD and waveforms
- Show arch test suites
- Show instruction coverage

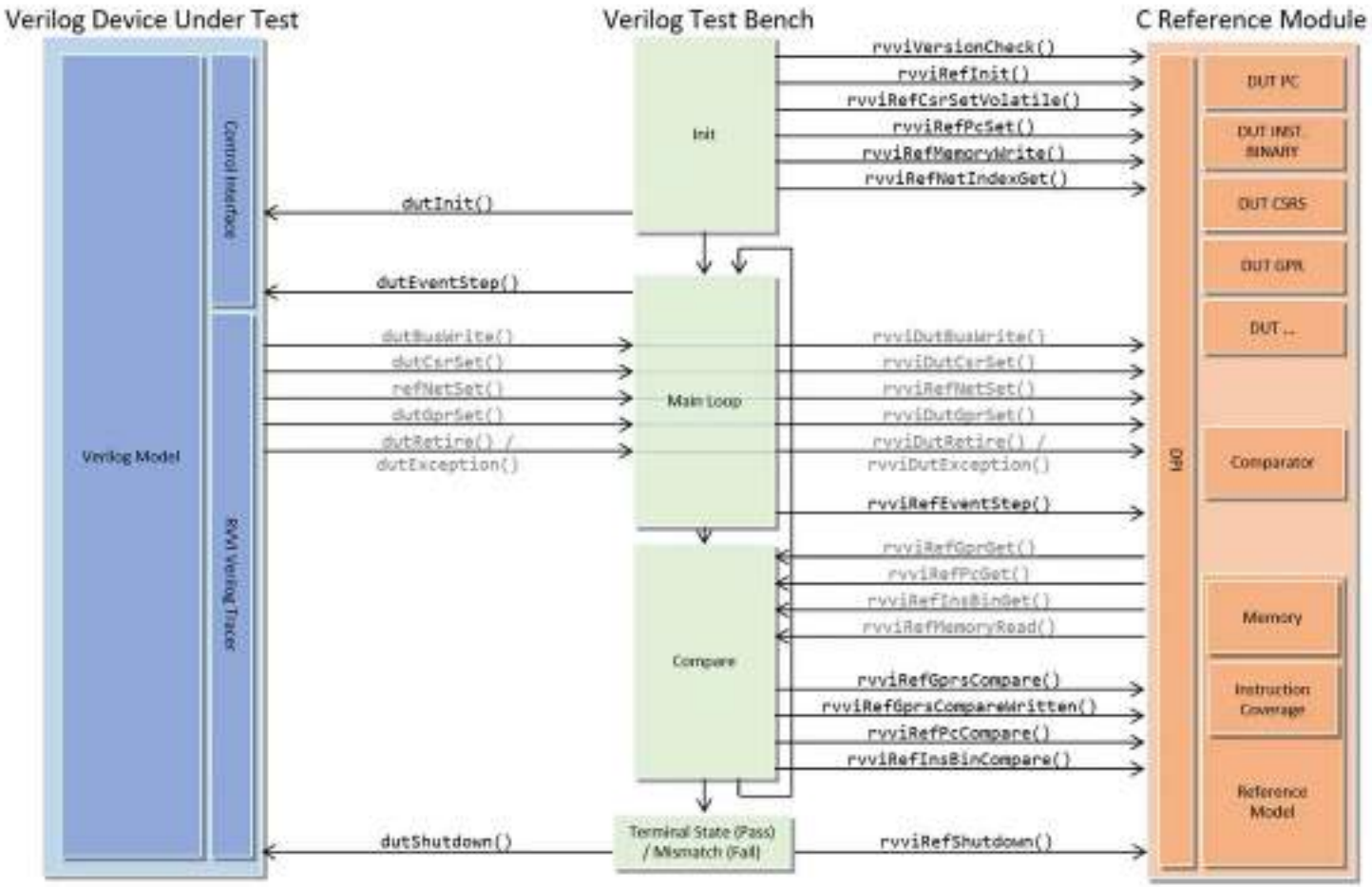
Demonstration / Walkthrough of ImperasDV Commercial RISC-V CPU DV solution

Lee Moore

Agenda

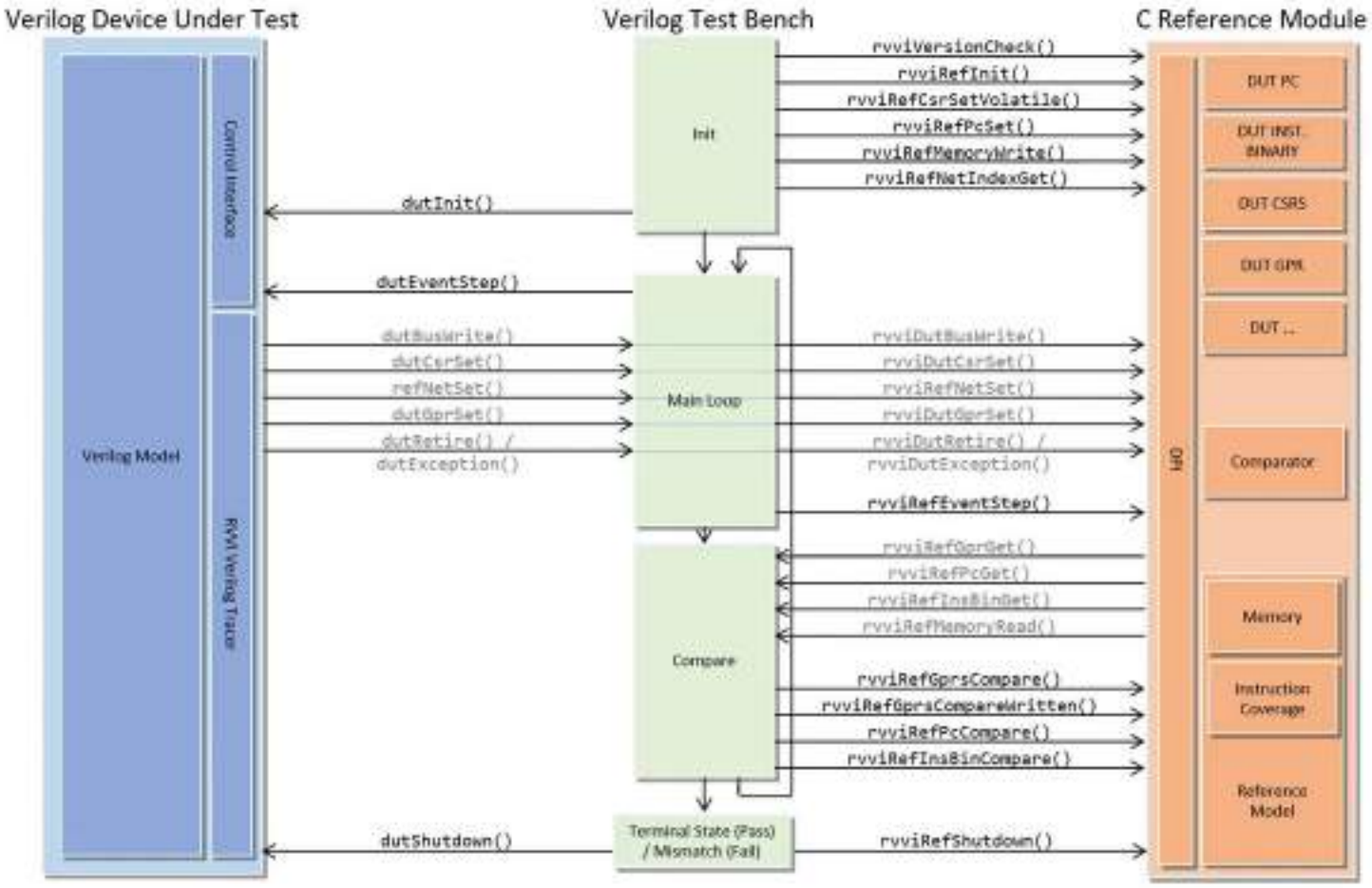
- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Drill down into an available commercial RISC-V HW DV verification solution
- Demonstration of ImperasDV in action on open source RISC-V cores
  - SystemVerilog test bench
- Components of RISC-V CPU DV environment

README.md



own in the center is in charge of coordinating the sequence of events during testing. The DUT shown on the left will

README.md



The test harness shown in the center is in charge of coordinating the sequence of events during testing. The DUT shown on the left will

imperas Local Documentation OVP

RVVI (RISC-V Verification Interface)

mperas Software Ltd. Open Virtual Platforms API Reference documentation.

bool_t	**rvviRefInsBinCompare** (uint32_t hartId) Compare retired instruction bytes between reference and DUT. More...
bool_t	**rvviRefPcCompare** (uint32_t hartId) Compare program counter for the retired instructions between DUT and the the reference model. More...
bool_t	**rvviRefCsrCompare** (uint32_t hartId, uint32_t csrIndex) Compare a CSR value between DUT and the the reference model. More...
bool_t	**rvviRefCsrsCompare** (uint32_t hartId) Compare all CSR values between DUT and the the reference model. More...
bool_t	**rvviRefVrsCompare** (uint32_t hartId) Compare all RVV vector register values between reference and DUT. More...
bool_t	**rvviRefFprsCompare** (uint32_t hartId) Compare all floating point register values between reference and DUT. More...
uint64_t	**rvviRefGprGet** (uint32_t hartId, uint32_t index) Read a GPR value from a hart in the reference model. More...
uint32_t	**rvviRefGprsWrittenGet** (uint32_t hartId) Read a GPR written mask from the last rvviRefEventStep. More...
uint64_t	**rvviRefPcGet** (uint32_t hartId) Return the program counter of a hart in the reference model. More...
uint64_t	**rvviRefCsrGet** (uint32_t hartId, uint32_t index) Read a CSR value from a hart in the reference model. More...
uint64_t	**rvviRefInsBinGet** (uint32_t hartId) Return the binary representation of the previously retired instruction. More...
uint64_t	**rvviRefFprGet** (uint32_t hartId, uint32_t index) Read a floating point register value from a hart in the reference model. More...
void	**rvviRefVrGet** (uint32_t hartId, uint32_t index, void *data, uint32_t size) Read a RVV vector register value from a hart in the reference model. More...
void	**rvviDutBusWrite** (uint32_t hartId, uint64_t address, uint64_t value, uint32_t byteEnableMask) Notify RVVI that the DUT has been written to memory. More...
void	**rvviRefMemoryWrite** (uint32_t hartId, uint64_t address, uint64_t data, uint32_t size) Write data to the reference model physical memory space. More...

workspace - ImperasDV/ibex/designTop/systemverilog/ibex_testbench.sv - Imperas eGui

File Edit Source Refactor Navigate Search Project Run Window Help

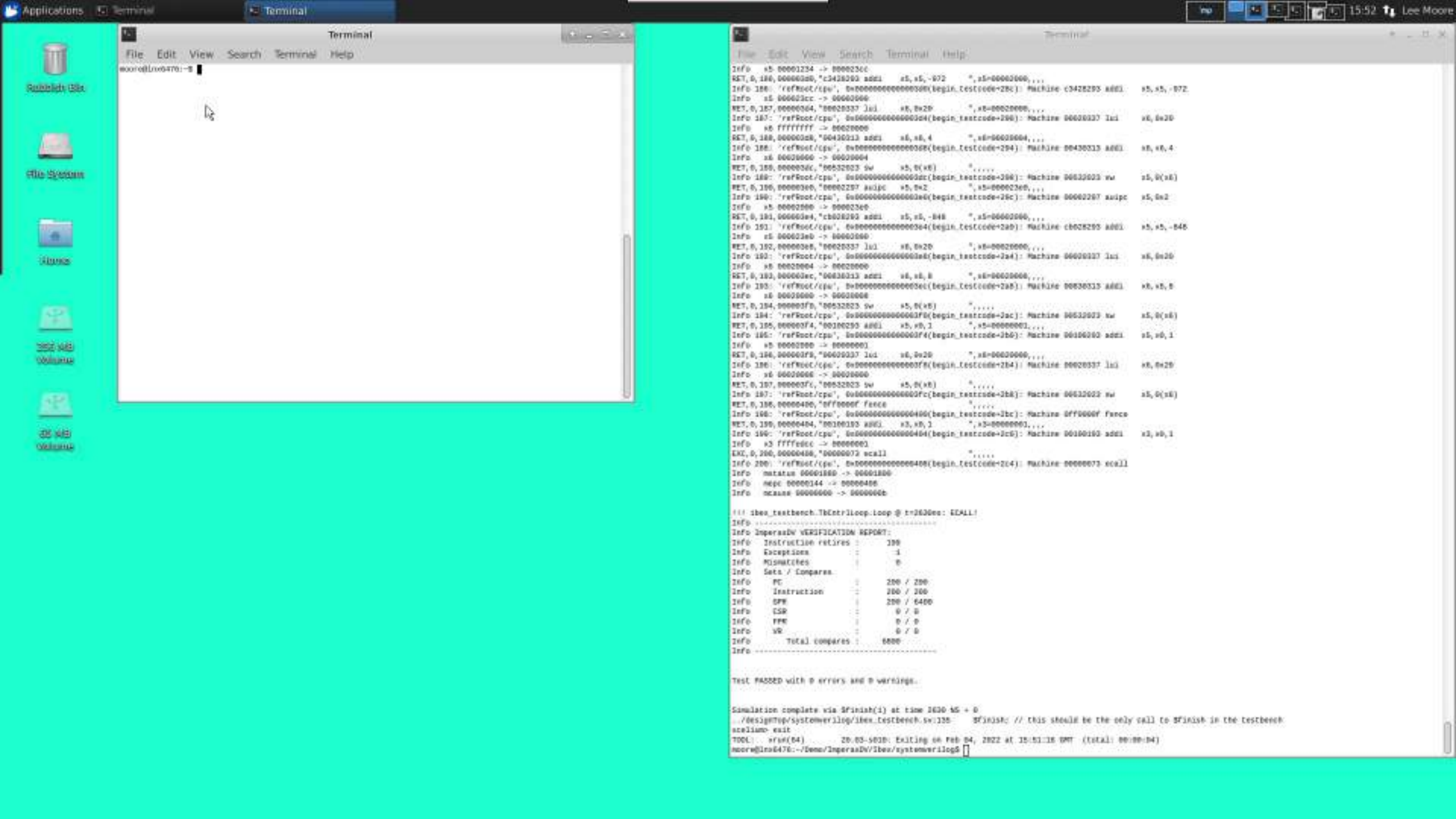
Project Explorer

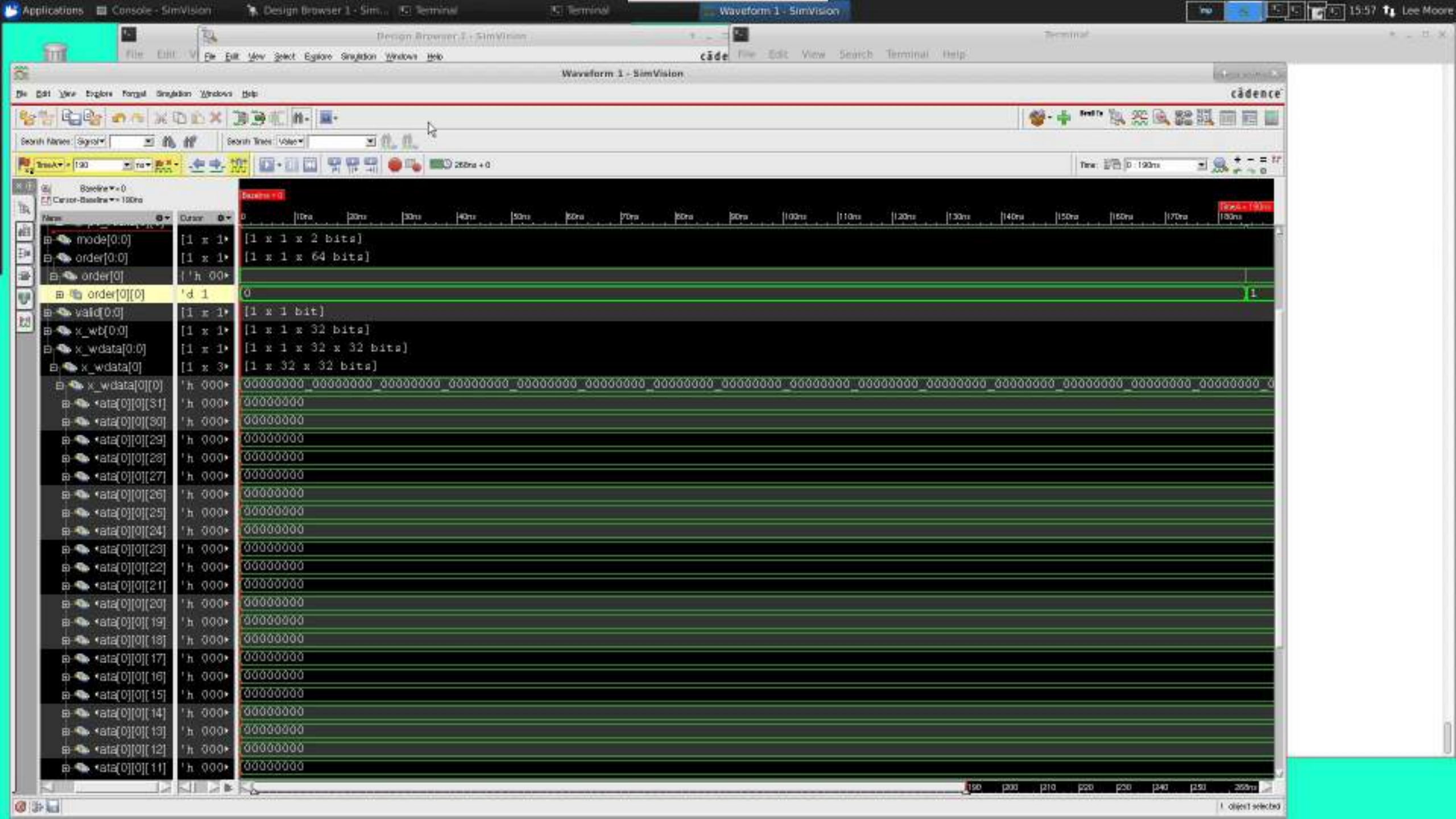
ImperasDV
- ibex
  - designTop
  - rtl
  - systemverilog
  - verilator
- Imperas
  - ImpProprietary
    - source
      - host
        - rvvi
          - vlg2api.sv
          - vlg2log.sv
  - ImpPublic
    - include
      - host
        - rvvi
          - rvvi-api.h
          - rvvi-api.svh
    - source
      - host
        - rvvi
          - rvvi-pkg.sv
          - rvvi-vlg.sv
- Setup_and_Checks
  - Control_Files
  - checkEnvironment.sh
  - defines.QUESTA.Makefile.include
  - defines.TARGETS.include
  - defines.XCELIUM.Makefile.include
  - install.pkg
  - Makefile.common.include
  - README.txt
- tests
- README.txt

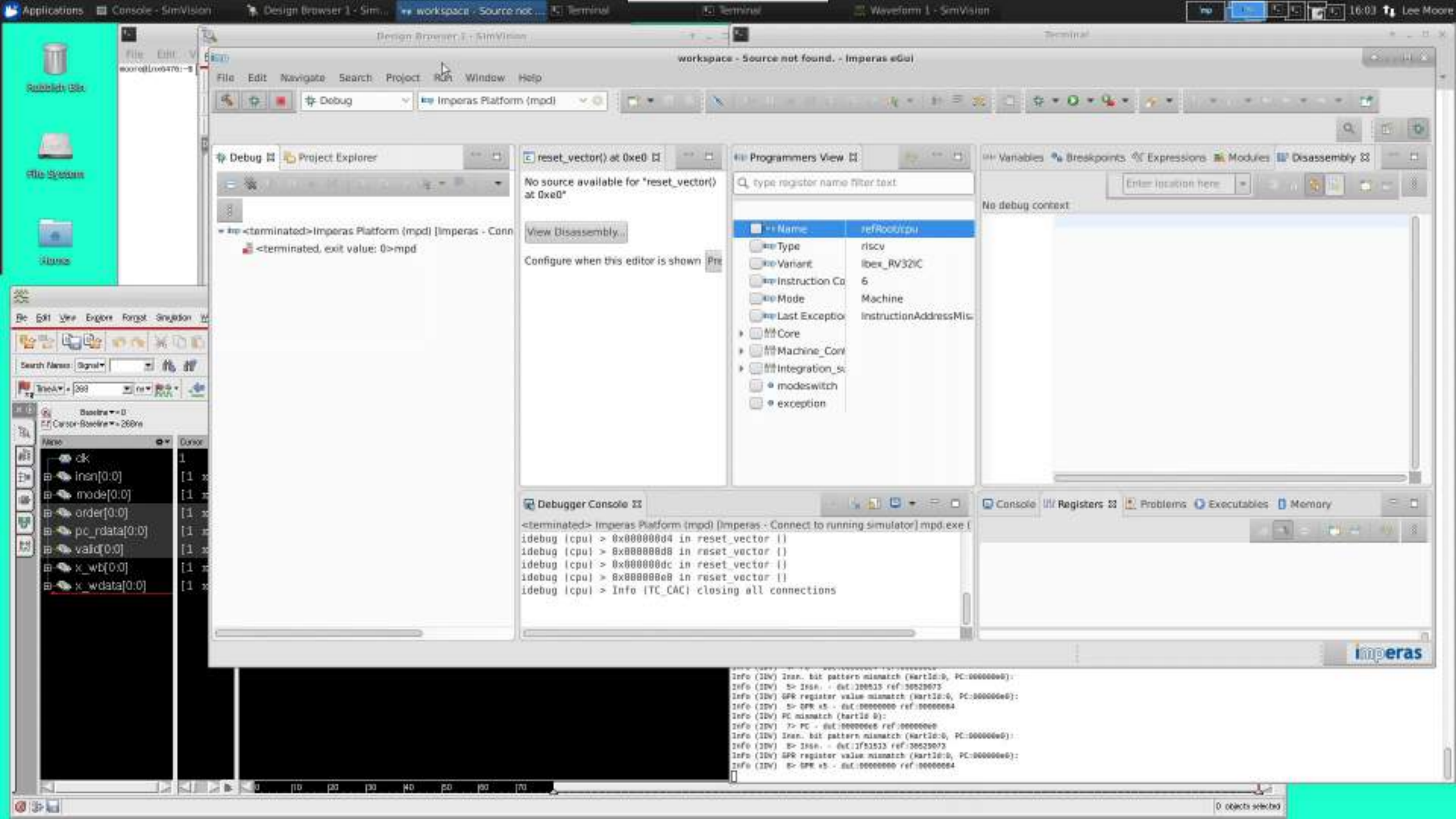
rvvi-api.h | rvvi-api.svh | rvvi-vlg.sv | vlg2api.sv | vlg2log.sv | ibex_rvvi_top.sv | ibex_testbench.sv | ibex_alu.sv

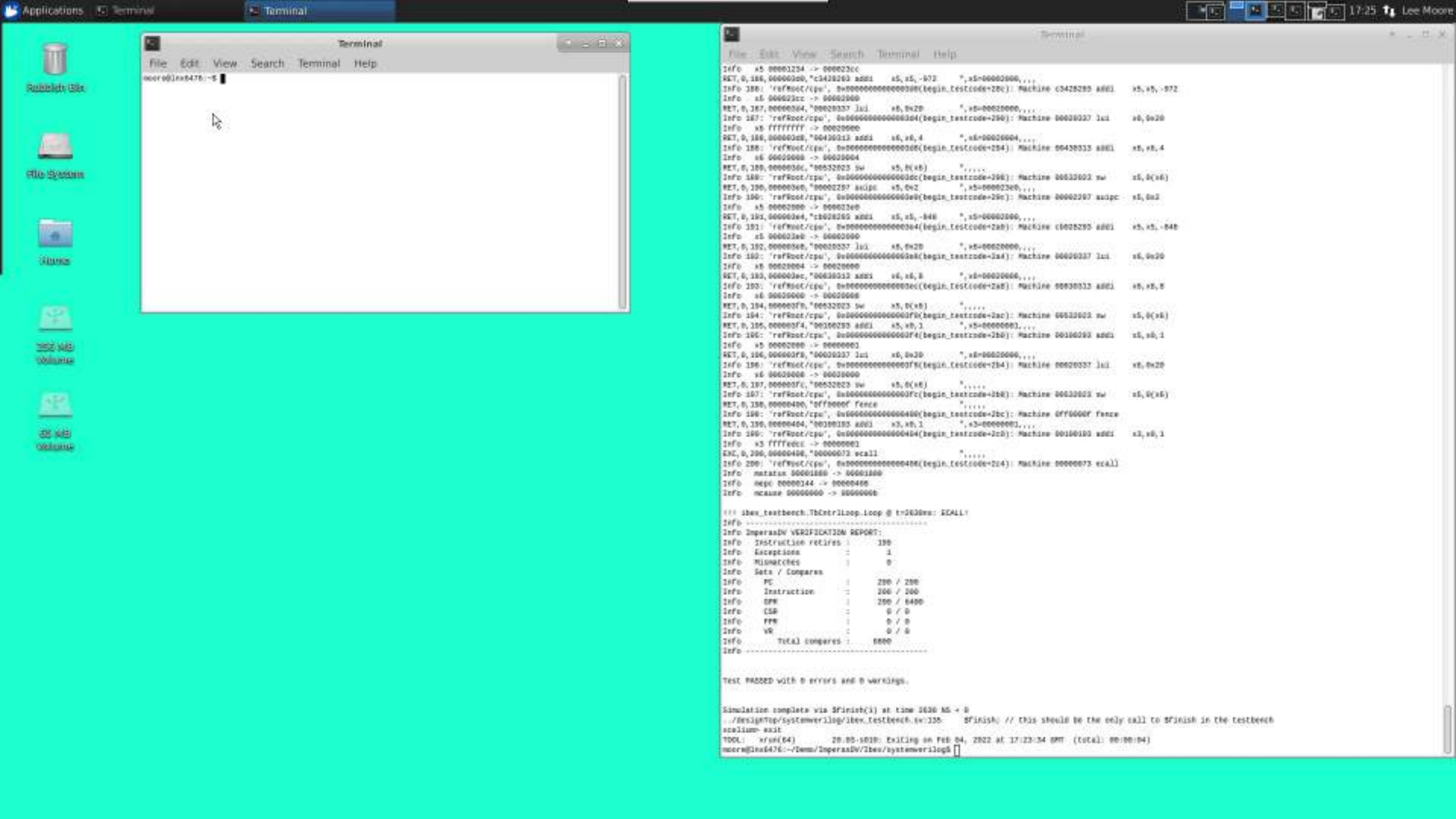
```
129   if (fatal) begin
130     $display(" due to fatal error(s) and %0d non-fatal errors and %0d warnings.\n\n", err_cnt, warn_cnt);
131   end
132   else begin
133     $display(" with %0d errors and %0d warnings.\n\n", err_cnt, warn_cnt);
134   end
135   $finish; // this should be the only call to $finish in the testbench
136 endfunction
137
138 ////////////////////////////////////////////////////////////////////////////////
139 // Testbench control:
140 //    This task implements the step-and-compare loop
141 task TbCntrlLoop;
142   int i;
143   i = 0;
144   // Initialize REF (do this before initializing the DUT)
145   if (!rvviVersionCheck(`RVVI_API_VERSION)) begin
146     $display("%m @ t=%0t: Expecting RVVI API version %0d.", $time, `RVVI_API_VERSION);
147     fatal_out();
148   end
149
150   if (!rvviRefInit("test_program.elf", "lowrisc.ovpworld.org", "Ibex_RV32IC")) begin
151     $display("%m @ t=%0t: rvviRefInit failed", $time);
152     fatal_out();
153   end
154
155   void'(rvviRefCsrSetVolatile(tb_hart_id, 32'h0000_0C00)); // cycle
156   void'(rvviRefCsrSetVolatile(tb_hart_id, 32'h0000_0C02)); // instret
157
158   // Initialize DUT
159   dutInit;
160
161   // Start step-and-compare loop
162   forever begin: Loop
163
164
165     dutStep; // returns on instruction retirement
166     i = i + 1;
167     if (i>100) tb_irq_external = 1;
168
169     inst = rvviRefInsBinGet(0);
170     if (inst == ECALL) begin
171       $display("\n!!! %m @ t=%0t: ECALL!", $time);
172       if (FINISH_ON_ECALL) begin
173         break;
174       end
175     end
176   end // Loop
177
178   dutShutdown();
179   void'(rvviRefShutdown());
180   terminate_sim();
181 endtask // TbCntrlLoop
182
183 ////////////////////////////////////////////////////////////////////////////////
184 // Watchdog timer
185 always @(posedge tb_clk) begin: wdt
186   ++tb_cycles;
187   if (tb_cycles >= TIMEOUT_CYCLES) begin
188     $display("\n\n!!!\n!!! %m @ t=%0t: FATAL: timeout!\n!!!", $time);
189     fatal_out();
190   end
191   if (err_cnt >= MAX_ERRS) begin
192     $display("\n\n!!!\n!!! %m @ t=%0t: FATAL: too many errors!\n!!!", $time);
193     fatal_out();
194   end
195 end // wdt
```

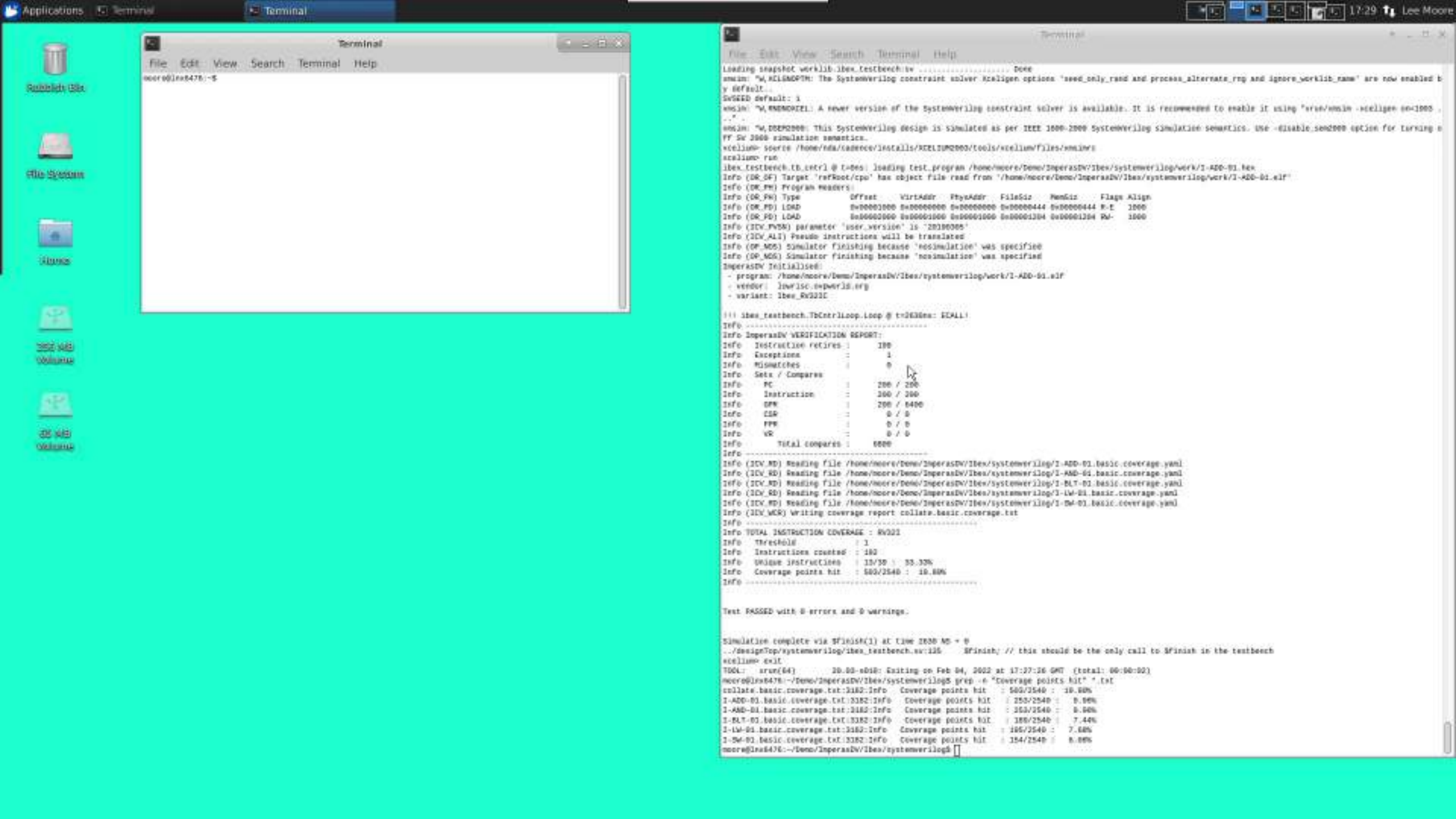
Writable | Insert | 143 : 11 : 4721


Applications
Terminal
Terminal
15:52
Lee Moore
Terminal
File Edit View Search Terminal Help
moore@lnx0470:~$
Terminal
File Edit View Search Terminal Help
Info x5 00001234 -> 000023cc
RET, 0, 186, 000003d0, "c3428293 addi x5, x5, -972 ", x5=00002000,...
Info 186: 'refRoot/cpu', 0x00000000000003d0(begin_testcode+28c): Machine c3428293 addi x5, x5, -972
Info x5 000023cc -> 00002000
RET, 0, 187, 00000364, "00020337 lui x6, 0x20 ", x6=00020000,...
Info 187: 'refRoot/cpu', 0x0000000000000364(begin_testcode+298): Machine 00020337 lui x6, 0x20
Info x6 ffffffff -> 00020000
RET, 0, 188, 000003d8, "00430313 addi x6, x6, 4 ", x6=00020004,...
Info 188: 'refRoot/cpu', 0x00000000000003d8(begin_testcode+294): Machine 00430313 addi x6, x6, 4
Info x6 00020000 -> 00020004
RET, 0, 189, 0000033c, "00532023 sw x5, 0(x6) ", ...
Info 189: 'refRoot/cpu', 0x000000000000033c(begin_testcode+298): Machine 00532023 sw x5, 0(x6)
RET, 0, 190, 00000368, "00002297 auipc x5, 0x2 ", x5=000023e8,...
Info 190: 'refRoot/cpu', 0x0000000000000368(begin_testcode+29c): Machine 00002297 auipc x5, 0x2
Info x5 00002368 -> 000023e8
RET, 0, 191, 000003e4, "cb028293 addi x5, x5, -848 ", x5=00002000,...
Info 191: 'refRoot/cpu', 0x00000000000003e4(begin_testcode+2a0): Machine cb028293 addi x5, x5, -848
Info x5 000023e8 -> 00002000
RET, 0, 192, 000003e8, "00020337 lui x6, 0x20 ", x6=00020000,...
Info 192: 'refRoot/cpu', 0x00000000000003e8(begin_testcode+2a4): Machine 00020337 lui x6, 0x20
Info x6 00020004 -> 00020000
RET, 0, 193, 000003ec, "00830313 addi x6, x6, 8 ", x6=00020008,...
Info 193: 'refRoot/cpu', 0x00000000000003ec(begin_testcode+2a8): Machine 00830313 addi x6, x6, 8
Info x6 00020000 -> 00020008
RET, 0, 194, 000003f0, "00532023 sw x5, 0(x6) ", ...
Info 194: 'refRoot/cpu', 0x00000000000003f0(begin_testcode+2ac): Machine 00532023 sw x5, 0(x6)
RET, 0, 195, 000003f4, "00100293 addi x5, x0, 1 ", x5=00000001,...
Info 195: 'refRoot/cpu', 0x00000000000003f4(begin_testcode+2b0): Machine 00100293 addi x5, x0, 1
Info x5 00002000 -> 00000001
RET, 0, 196, 000003f8, "00020337 lui x6, 0x20 ", x6=00020000,...
Info 196: 'refRoot/cpu', 0x00000000000003f8(begin_testcode+2b4): Machine 00020337 lui x6, 0x20
Info x6 00020008 -> 00020000
RET, 0, 197, 000003fc, "00532023 sw x5, 0(x6) ", ...
Info 197: 'refRoot/cpu', 0x00000000000003fc(begin_testcode+2b8): Machine 00532023 sw x5, 0(x6)
RET, 0, 198, 00000400, "0ff0000f fence ", ...
Info 198: 'refRoot/cpu', 0x0000000000000400(begin_testcode+2bc): Machine 0ff0000f fence
RET, 0, 199, 00000404, "00100193 addi x3, x0, 1 ", x3=00000001,...
Info 199: 'refRoot/cpu', 0x0000000000000404(begin_testcode+2c0): Machine 00100193 addi x3, x0, 1
Info x3 ffffedcc -> 00000001
EXC, 0, 200, 00000408, "00000073 ecall ", ...
Info 200: 'refRoot/cpu', 0x0000000000000408(begin_testcode+2c4): Machine 00000073 ecall
Info mstatus 00001880 -> 00001800
Info mepc 00000144 -> 00000408
Info mcause 00000000 -> 0000000b
!!! ibex_testbench.TbEntrLoop.Loop @ t=2630ns: ECALL!
Info --
Info ImperasDV VERIFICATION REPORT:
Info Instruction retires : 199
Info Exceptions : 1
Info Mismatches : 0
Info Sets / Compares
Info PC : 200 / 200
Info Instruction : 200 / 200
Info GPR : 200 / 6400
Info CSR : 0 / 0
Info FPR : 0 / 0
Info VR : 0 / 0
Info Total compares : 6800
Info --
Test PASSED with 0 errors and 0 warnings.
Simulation complete via $finish(1) at time 2630 NS + 0
../designTop/systemverilog/ibex_testbench.sv:155 $finish; // this should be the only call to $finish in the testbench
xcelium> exit
TOOL: xrun(64) 20.03-s010: Exiting on Feb 04, 2022 at 15:51:16 GMT (total: 00:00:04)
moore@lnx0470:~/Demo/ImperasDV/Ibex/systemverilog$


Waveform 1 - SimVision
mode[0:0]
order[0:0]
order[0]
order[0][0]
valid[0:0]
x_wb[0:0]
x_wdata[0:0]
x_wdata[0]
x_wdata[0][0]
[1 x 1 x 2 bits]
[1 x 1 x 64 bits]
[1 x 1 bit]
[1 x 1 x 32 bits]
[1 x 1 x 32 x 32 bits]
[1 x 32 x 32 bits]


Applications
Console - SimVision
Design Browser 1 - Sim...
workspace - Source not ...
Terminal
Terminal
Waveform 1 - SimVision
16:03
Lee Moore
Rubbish Bin
File System
Home
Design Browser 1 - SimVision
Terminal
workspace - Source not found. - Imperas eGui
File Edit Navigate Search Project Run Window Help
Debug
Imperas Platform (mpd)
Debug
Project Explorer
<terminated>Imperas Platform (mpd) [Imperas - Conn
<terminated, exit value: 0>mpd
reset_vector() at 0xe0
No source available for "reset_vector() at 0xe0"
View Disassembly...
Configure when this editor is shown
Programmers View
type register name filter text
Name
Type
Variant
Instruction Co
Mode
Last Exceptio
Core
Machine_Cont
Integration_s
modeswitch
exception
riscv
Ibex_RV32IC
6
Machine
InstructionAddressMis
Variables
Breakpoints
Expressions
Modules
Disassembly
Enter location here
No debug context
Debugger Console
<terminated> Imperas Platform (mpd) [Imperas - Connect to running simulator] mpd.exe
idebug (cpu) > 0x000000d4 in reset_vector ()
idebug (cpu) > 0x000000d8 in reset_vector ()
idebug (cpu) > 0x000000dc in reset_vector ()
idebug (cpu) > 0x000000e0 in reset_vector ()
idebug (cpu) > Info (TC_CAC) closing all connections
Console
Registers
Problems
Executables
Memory
imperas
clk
insn[0:0]
mode[0:0]
order[0:0]
pc_rdata[0:0]
valid[0:0]
x_wb[0:0]
x_wdata[0:0]
Info (IDV) Insn. bit pattern mismatch (HartId:0, PC:00000000):
Info (IDV) 5> Insn. - dut:100513 ref:30529073
Info (IDV) GPR register value mismatch (HartId:0, PC:00000000):
Info (IDV) 5> GPR x5 - dut:00000000 ref:00000004
Info (IDV) PC mismatch (HartId 0):
Info (IDV) 7> PC - dut:00000000 ref:00000000
Info (IDV) Insn. bit pattern mismatch (HartId:0, PC:00000000):
Info (IDV) 8> Insn. - dut:1f93513 ref:30529073
Info (IDV) GPR register value mismatch (HartId:0, PC:00000000):
Info (IDV) 8> GPR x5 - dut:00000000 ref:00000004
0 objects selected


Applications
Terminal
Terminal
17:25
Lee Moore
Rubbish Bin
File System
Home
Terminal
File Edit View Search Terminal Help
moore@lex6476:~$
Terminal
File Edit View Search Terminal Help
Info: x5 00001234 -> 000023cc
RET,0,186,000003d0,"c3428293 addi x5,x5,-972 ",x5=00002000,...
Info 186: 'refRoot/cpu', 0x00000000000003d0(begin_testcode+2f0): Machine c3428293 addi x5,x5,-972
Info: x5 000023cc -> 00002000
RET,0,187,000003d4,"00020337 lui x6,0x20 ",x6=00020000,...
Info 187: 'refRoot/cpu', 0x00000000000003d4(begin_testcode+2f4): Machine 00020337 lui x6,0x20
Info: x6 ffffffff -> 00020000
RET,0,188,000003d8,"00430313 addi x6,x6,4 ",x6=00020004,...
Info 188: 'refRoot/cpu', 0x00000000000003d8(begin_testcode+2f8): Machine 00430313 addi x6,x6,4
Info: x6 00020000 -> 00020004
RET,0,189,000003dc,"00532023 sw x5,0(x6) ",...
Info 189: 'refRoot/cpu', 0x00000000000003dc(begin_testcode+2fc): Machine 00532023 sw x5,0(x6)
RET,0,190,000003e0,"00002297 auipc x5,0x2 ",x5=000023e0,...
Info 190: 'refRoot/cpu', 0x00000000000003e0(begin_testcode+300): Machine 00002297 auipc x5,0x2
Info: x5 000023e0 -> 000023e0
RET,0,191,000003e4,"c5828293 addi x5,x5,-840 ",x5=00002000,...
Info 191: 'refRoot/cpu', 0x00000000000003e4(begin_testcode+2a0): Machine c5828293 addi x5,x5,-840
Info: x5 000023e0 -> 00002000
RET,0,192,000003e8,"00020337 lui x6,0x20 ",x6=00020000,...
Info 192: 'refRoot/cpu', 0x00000000000003e8(begin_testcode+2a4): Machine 00020337 lui x6,0x20
Info: x6 00020004 -> 00020000
RET,0,193,000003ec,"00630313 addi x6,x6,6 ",x6=00020006,...
Info 193: 'refRoot/cpu', 0x00000000000003ec(begin_testcode+2a8): Machine 00630313 addi x6,x6,6
Info: x6 00020000 -> 00020006
RET,0,194,000003f0,"00532023 sw x5,0(x6) ",...
Info 194: 'refRoot/cpu', 0x00000000000003f0(begin_testcode+2ac): Machine 00532023 sw x5,0(x6)
RET,0,195,000003f4,"00100293 addi x5,x0,1 ",x5=00000001,...
Info 195: 'refRoot/cpu', 0x00000000000003f4(begin_testcode+2b0): Machine 00100293 addi x5,x0,1
Info: x5 00002000 -> 00000001
RET,0,196,000003f8,"00020337 lui x6,0x20 ",x6=00020000,...
Info 196: 'refRoot/cpu', 0x00000000000003f8(begin_testcode+2b4): Machine 00020337 lui x6,0x20
Info: x6 00020000 -> 00020000
RET,0,197,000003fc,"00532023 sw x5,0(x6) ",...
Info 197: 'refRoot/cpu', 0x00000000000003fc(begin_testcode+2b8): Machine 00532023 sw x5,0(x6)
RET,0,198,00000400,"0ff0000f fence ",...
Info 198: 'refRoot/cpu', 0x0000000000000400(begin_testcode+2bc): Machine 0ff0000f fence
RET,0,199,00000404,"00100193 addi x3,x0,1 ",x3=00000001,...
Info 199: 'refRoot/cpu', 0x0000000000000404(begin_testcode+2c0): Machine 00100193 addi x3,x0,1
Info: x3 ffffedcc -> 00000001
EXC,0,200,00000408,"00000073 ecall ",...
Info 200: 'refRoot/cpu', 0x0000000000000408(begin_testcode+2c4): Machine 00000073 ecall
Info: mstatus 00001800 -> 00001800
Info: mepc 00000144 -> 00000408
Info: mcause 00000000 -> 0000000b
!!! ibex_testbench.TbCntrlLoop.loop @ t=3630ns: ECALL!
Info --
Info ImperasDV VERIFICATION REPORT:
Info Instruction retires : 199
Info Exceptions : 1
Info Mismatches : 0
Info Sets / Compares
Info PC : 200 / 200
Info Instruction : 200 / 200
Info GPR : 200 / 6400
Info CSR : 0 / 0
Info FPR : 0 / 0
Info VR : 0 / 0
Info Total compares : 6800
Info --
Test PASSED with 0 errors and 0 warnings.
Simulation complete via $finish(1) at time 3630 NS + 0
../DesignTop/systemverilog/ibex_testbench.sv:155 $finish; // this should be the only call to $finish in the testbench
xcelium> exit
TOOL: xrun(64) 20.09-s010: Exiting on Feb 04, 2022 at 17:23:34 GMT (total: 00:00:04)
moore@lex6476:~/Demo/ImperasDV/Ibex/systemverilog$


Applications
Terminal
Terminal
17:29
Lee Moore
Terminal
File Edit View Search Terminal Help
moore@lex8478:~$
Terminal
File Edit View Search Terminal Help
Loading snapshot worklib.ibex_testbench:sv ..... Done
xmsim: *W,RCLSNOPTM: The SystemVerilog constraint solver Xceligen options 'seed_only_rand and process_alternate_rng and ignore_worklib_name' are now enabled by default.
SvSEED default: 1
xmsim: *W,RNDNOPCEL: A newer version of the SystemVerilog constraint solver is available. It is recommended to enable it using "xrun/xmsim -xceligen on=1005 ..." .
xmsim: *W,DSEM2009: This SystemVerilog design is simulated as per IEEE 1800-2009 SystemVerilog simulation semantics. Use -disable_sem2009 option for turning off SV 2009 simulation semantics.
xcelium> source /home/xda/cadence/installs/XCELIUM2003/tools/xcelium/files/xmsimrc
xcelium> run
ibex_testbench.tb_cntrl @ t=0ns: loading test_program /home/moore/Demo/ImperasDV/Ibex/systemverilog/work/I-ADD-01.hex
Info (OR_OF) Target 'refRoot/cpu' has object file read from '/home/moore/Demo/ImperasDV/Ibex/systemverilog/work/I-ADD-01.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type   Offset   VirtAddr   PhysAddr   FileSiz   MemSiz   Flags Align
Info (OR_PD) LOAD   0x00001000 0x00000000 0x00000000 0x00000444 0x00000444 R-E   1000
Info (OR_PD) LOAD   0x00002000 0x00001000 0x00001000 0x00001304 0x00001304 RW-   1000
Info (ICV_PVSN) parameter 'user_version' is '20190305'
Info (ICV_ALI) Pseudo instructions will be translated
Info (OP_NOS) Simulator finishing because 'nosimulation' was specified
Info (OP_NOS) Simulator finishing because 'nosimulation' was specified
ImperasDV Initialised:
 - program: /home/moore/Demo/ImperasDV/Ibex/systemverilog/work/I-ADD-01.elf
 - vendor: lowrisc.eu#world.org
 - variant: Ibex_RV32IC

!!! ibex_testbench.TbCntrlLoop.Loop @ t=2830ns: FINAL!
Info --
Info ImperasDV VERIFICATION REPORT:
Info   Instruction retires :   198
Info   Exceptions   :   1
Info   Mismatches   :   0
Info   Sets / Compares
Info   PC   :   200 / 200
Info   Instruction   :   200 / 200
Info   GPR   :   200 / 8400
Info   CSR   :   0 / 0
Info   FPR   :   0 / 0
Info   VR   :   0 / 0
Info   Total compares :   8800
Info --
Info (ICV_RD) Reading file /home/moore/Demo/ImperasDV/Ibex/systemverilog/I-ADD-01.basic.coverage.yaml
Info (ICV_RD) Reading file /home/moore/Demo/ImperasDV/Ibex/systemverilog/I-AND-01.basic.coverage.yaml
Info (ICV_RD) Reading file /home/moore/Demo/ImperasDV/Ibex/systemverilog/I-BLT-01.basic.coverage.yaml
Info (ICV_RD) Reading file /home/moore/Demo/ImperasDV/Ibex/systemverilog/I-LW-01.basic.coverage.yaml
Info (ICV_RD) Reading file /home/moore/Demo/ImperasDV/Ibex/systemverilog/I-SW-01.basic.coverage.yaml
Info (ICV_WCR) Writing coverage report collate.basic.coverage.txt
Info --
Info TOTAL INSTRUCTION COVERAGE : RV32I
Info   Threshold   : 1
Info   Instructions counted : 183
Info   Unique instructions : 15/30 : 50.00%
Info   Coverage points hit : 503/2540 : 19.80%
Info --

Test PASSED with 0 errors and 0 warnings.

Simulation complete via $finish(1) at time 2830 NS + 0
../designTop/systemverilog/ibex_testbench.sv:135 $finish; // this should be the only call to $finish in the testbench
xcelium> exit
TOOL: xrun(64)   20.03-s010: Exiting on Feb 04, 2022 at 17:27:26 GMT (total: 00:00:02)
moore@lex8478:~/Demo/ImperasDV/Ibex/systemverilog$ grep -e "Coverage points hit" *.txt
collate.basic.coverage.txt:3182:Info   Coverage points hit   : 503/2540 : 19.80%
I-ADD-01.basic.coverage.txt:3182:Info   Coverage points hit   : 253/2540 :   9.96%
I-AND-01.basic.coverage.txt:3182:Info   Coverage points hit   : 253/2540 :   9.96%
I-BLT-01.basic.coverage.txt:3182:Info   Coverage points hit   : 189/2540 :   7.44%
I-LW-01.basic.coverage.txt:3182:Info   Coverage points hit   : 195/2540 :   7.68%
I-SW-01.basic.coverage.txt:3182:Info   Coverage points hit   : 154/2540 :   6.06%
moore@lex8478:~/Demo/ImperasDV/Ibex/systemverilog$

Demo: ImperasDV
Core: lowRISC Ibex
Simulator: SystemVerilog
DV mode: sync-lock-step-compare

- Overview block diagram from RVVI github
- Walk through C/C++ rvvi.h and in doxygen – introduce APIs: RVVI-VLG
- Walk through tracer code where it converts RVVI-VLG nets to -> RVVI-API
- Walk through SystemVerilog harness
  - Show init, config, main step loop
- Run example - passes
- Run example - fails, show trace, show in eGuiMPD and waveforms
- Show arch test suites
- Show instruction coverage

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Components of RISC-V CPU DV environment

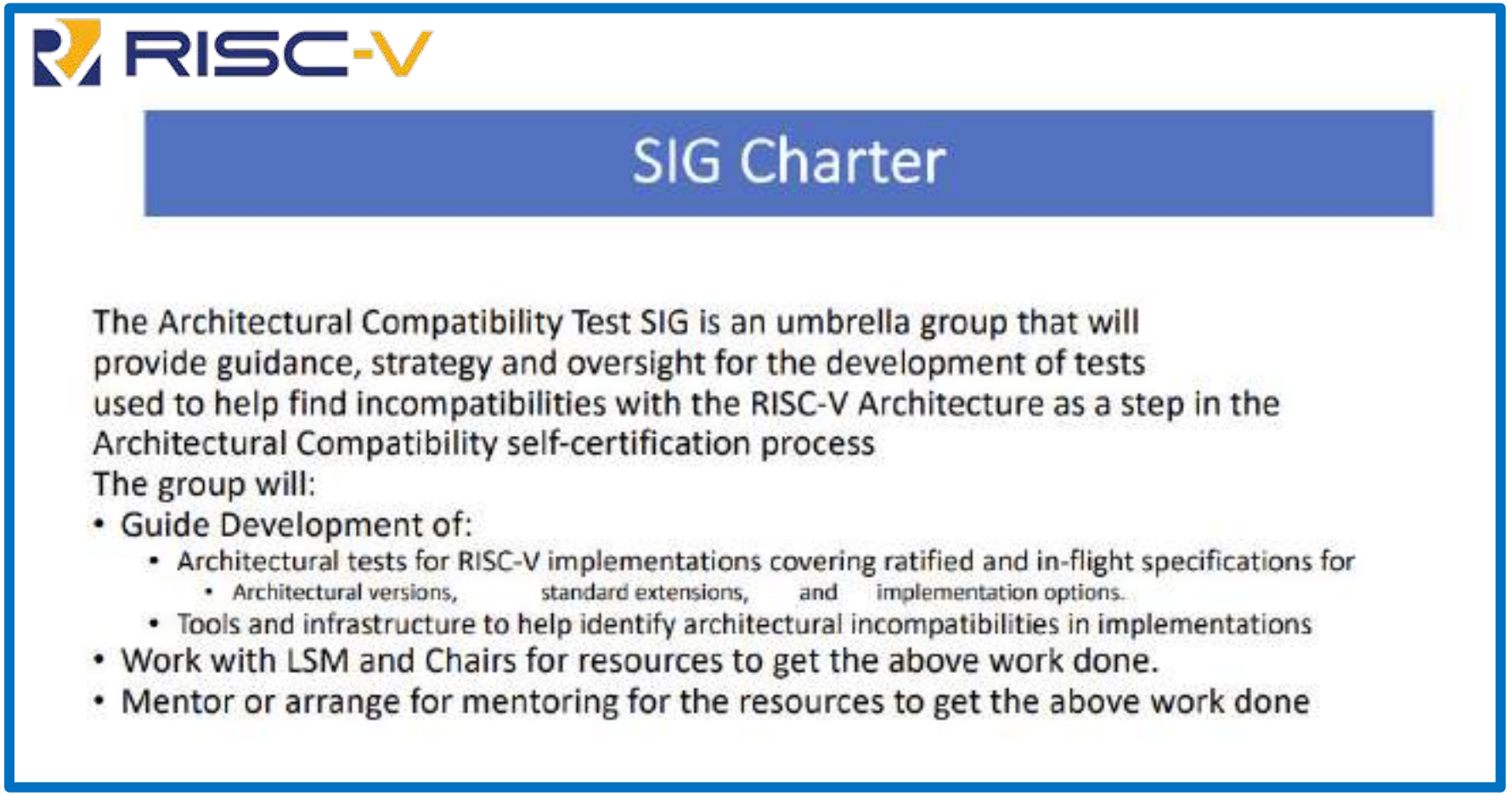
Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Components of RISC-V CPU DV environment
  - Compliance Tests and other Test Suites
  - Instruction Stream Generators
  - Functional Coverage
  - Instruction Set Simulators

RISC-V International's compliance tests

- RISC-V International has been working on compliance testing since 2018
- Status (Feb 2022):
  - Test suites for basic un-priv older ratified extensions I, M, C, etc
  - Simple framework for running DUT and provides signatures for comparison
  - Working on new framework to run in a post simulation signature compare mode
    - Encapsulates sail model, uses yaml configuration, does not provide build in reference signatures
  - Process is self-certification
    - You run the tests on your DUT and declare it is RISC-V compliant

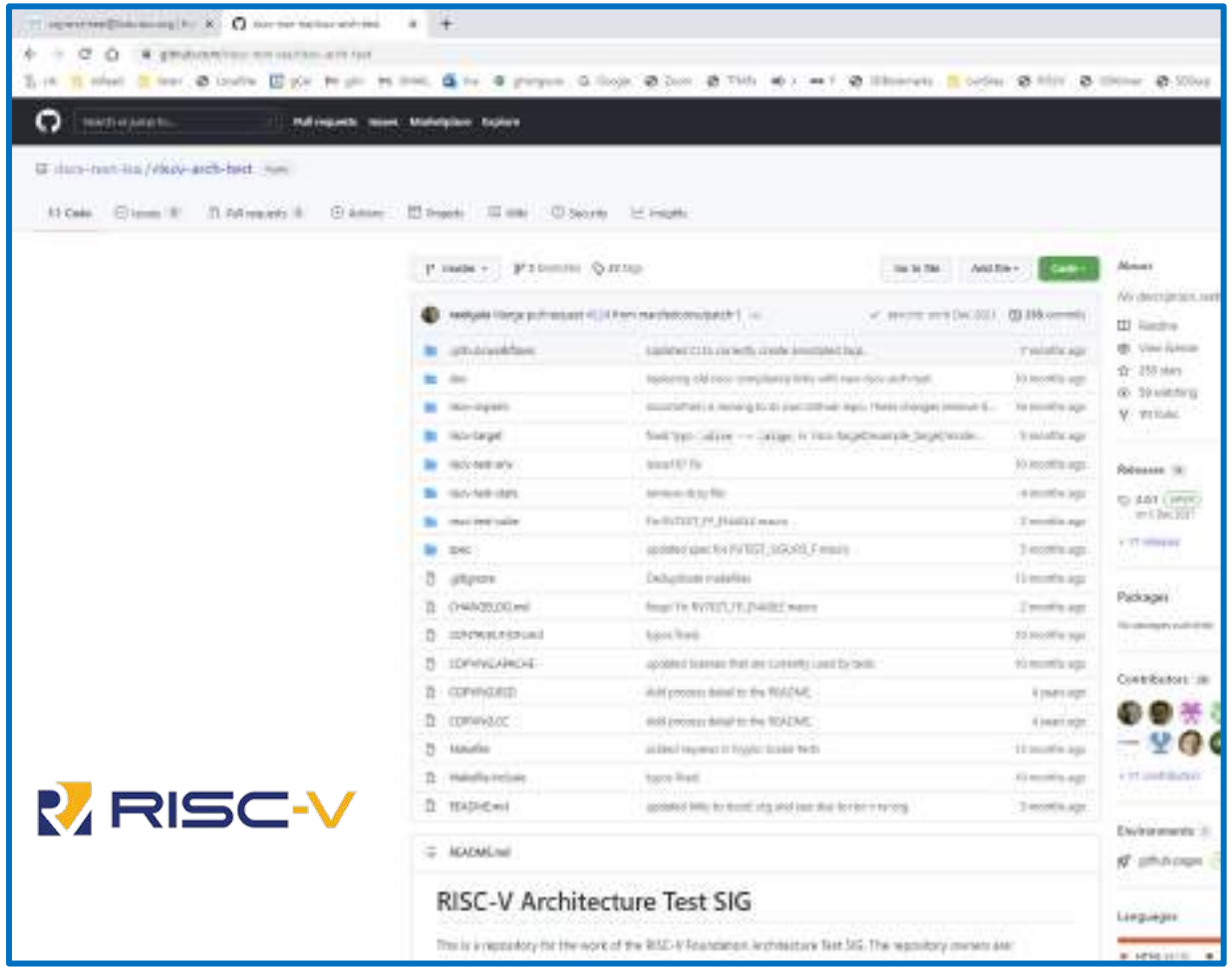
RISC-V International Compliance Special Interest Group - Charter (aka compliance working group)



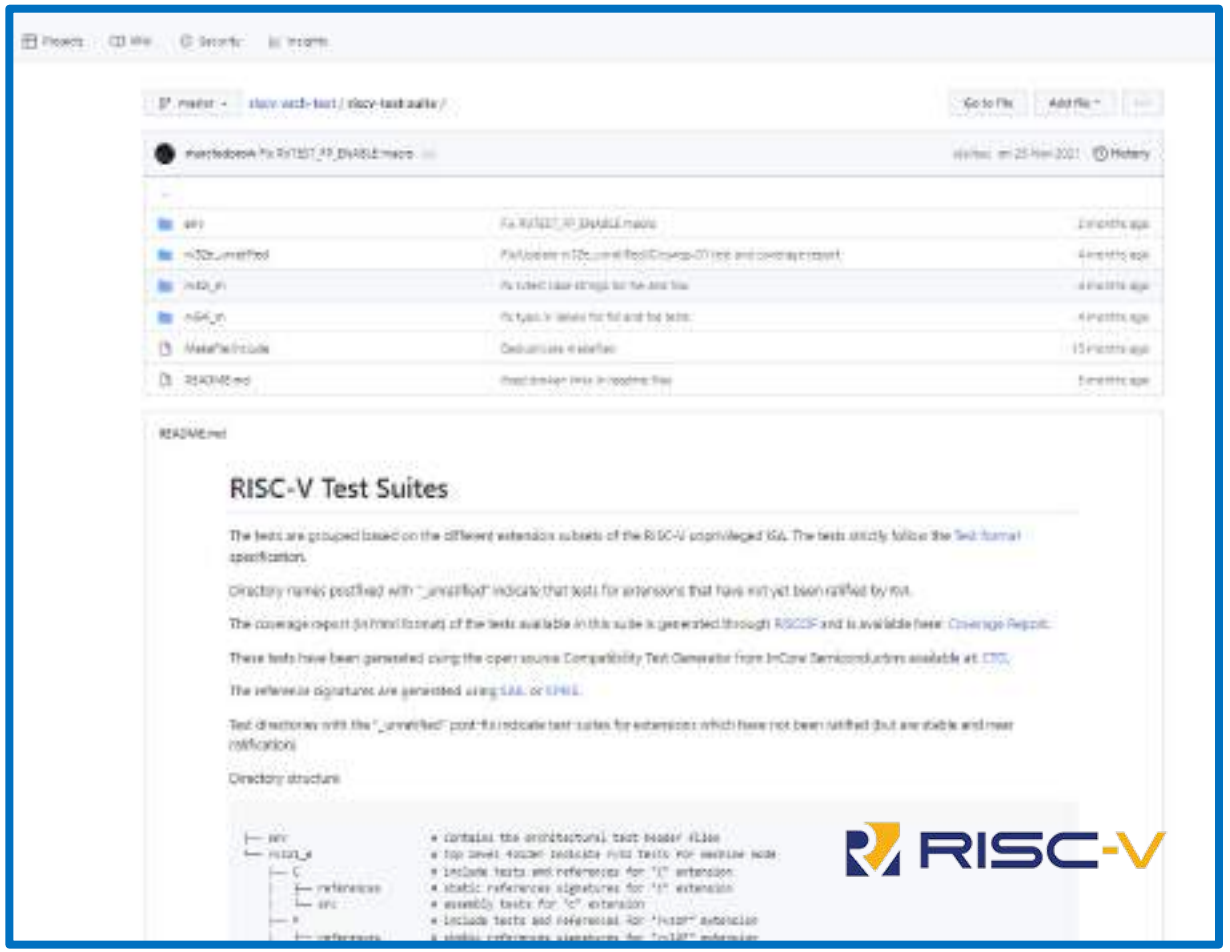
RISC-V International Compliance Special Interest Group (2) (aka compliance working group)

RISC-V International Compliance Special Interest Group (3) – Mailing List (aka compliance working group)

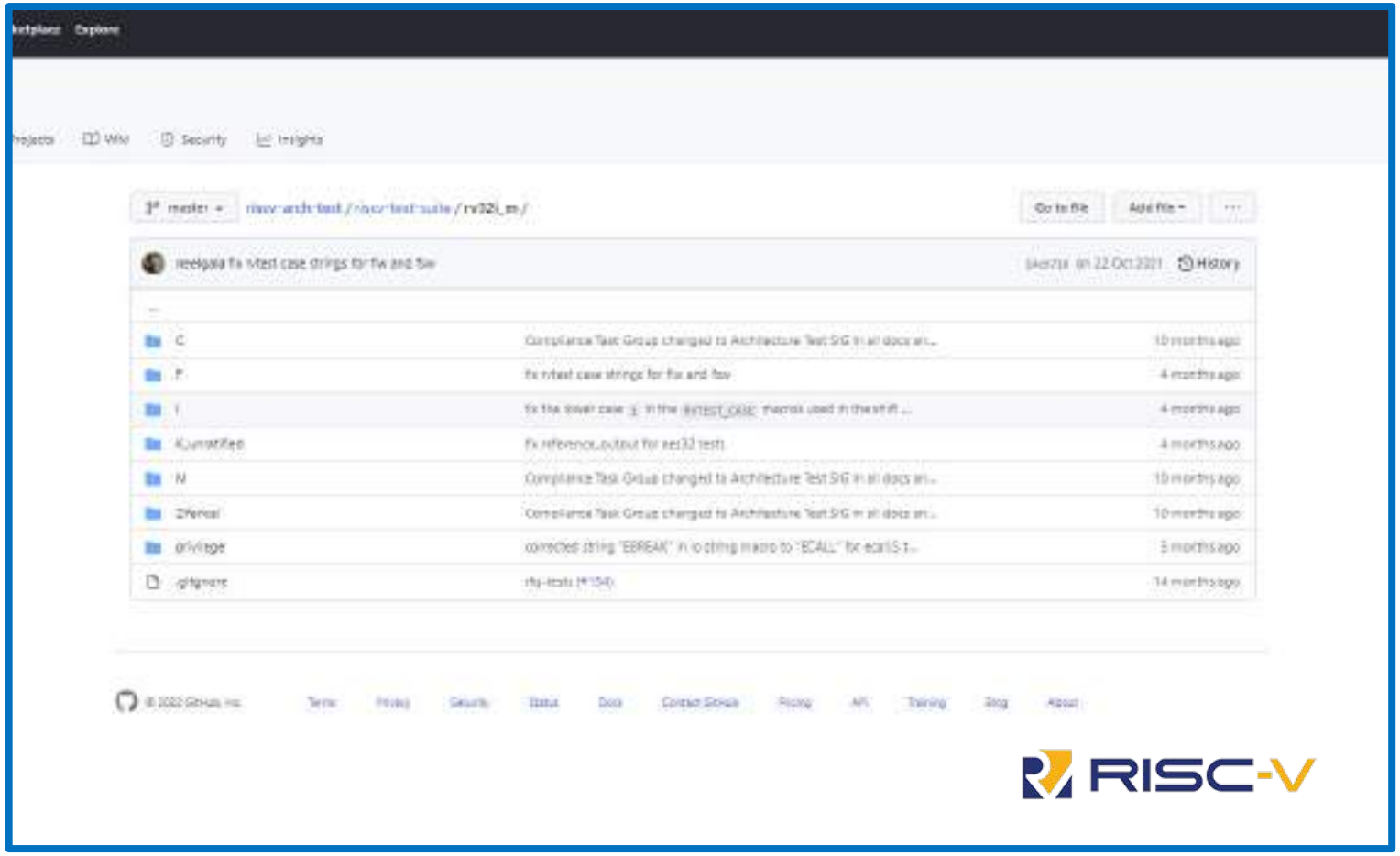
RISC-V International Compliance Special Interest Group (4) - GitHub (aka compliance working group)



RISC-V International Compliance Special Interest Group (5) – GitHub – test suites (aka compliance working group)

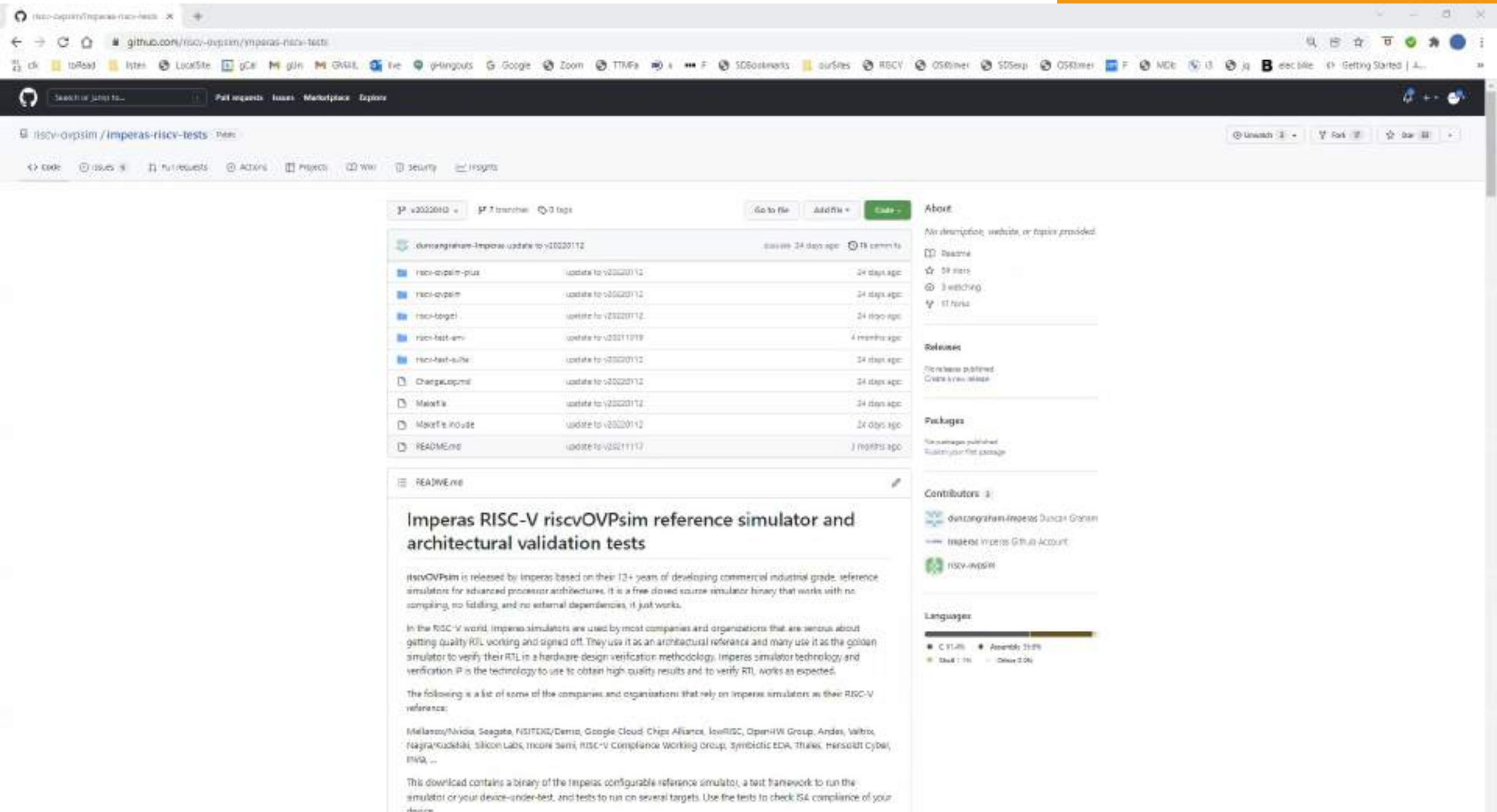


RISC-V International Compliance Special Interest Group (6) – GitHub – test suites (aka compliance working group)

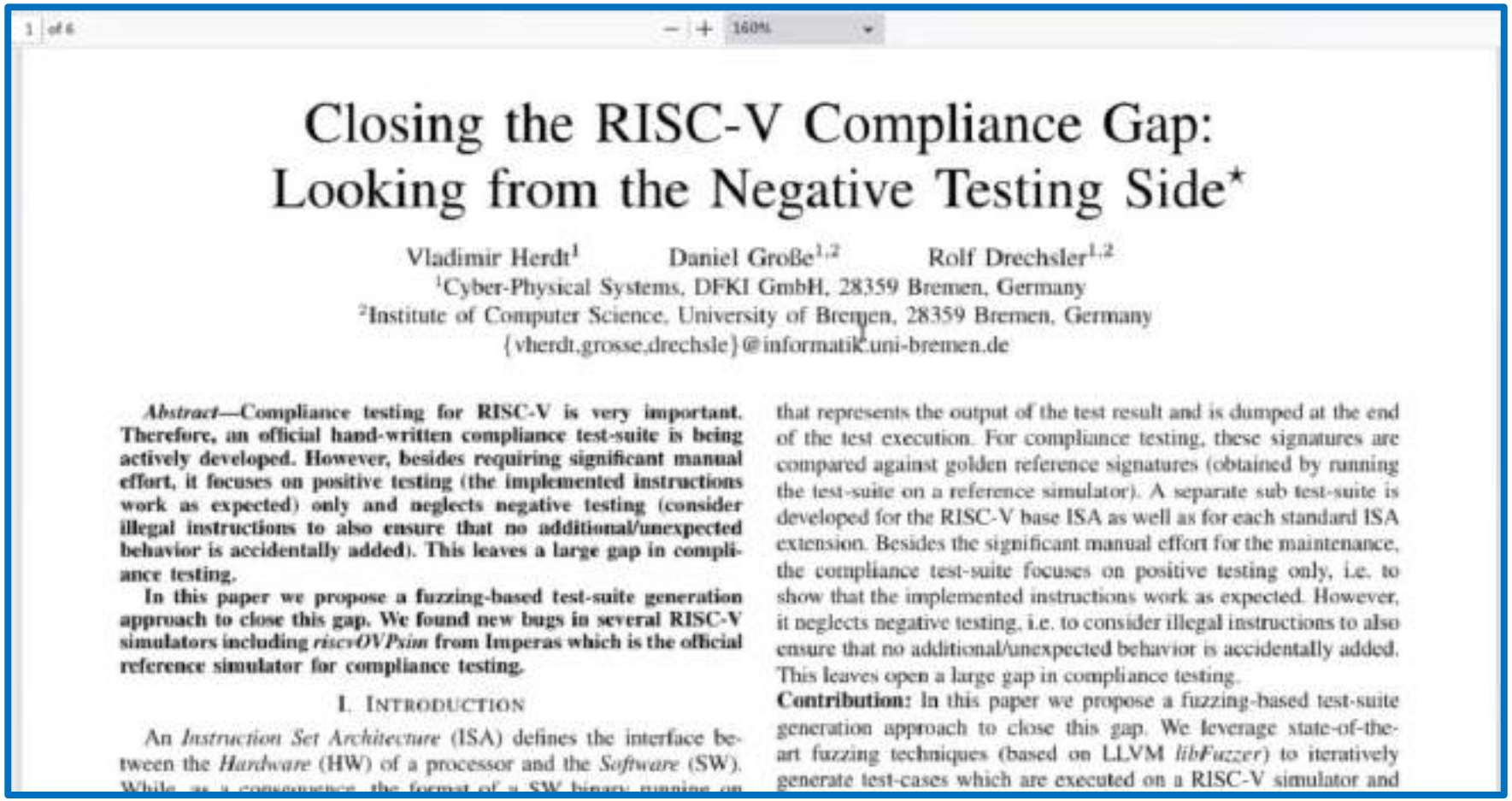


Imperas Test Suites

- When verifying a CPU design - you can never have enough tests...
- Imperas have developed a directed RISC-V test generator, instruction coverage measuring VIP, and a test qualifying mutating fault simulator to provide high quality test suites
- The generated tests suites are targeting architectural compatibility as defined in the RVIA architectural test working group coverage requirements
- There are currently over 50 free test suites, including
  - I,M,C,F,D,B,K,V,P
    - The provided vector test suite is one specific vector engine configuration
- The test suites are provided under an OVP open source license and are available free from: https://github.com/riscv-ovpsim/imperas-riscv-tests


github.com/riscv-ovpsim/imperas-riscv-tests
riscv-ovpsim / imperas-riscv-tests
Imperas RISC-V riscvOVPsim reference simulator and architectural validation tests
riscvOVPsim is released by Imperas based on their 13+ years of developing commercial industrial grade, reference simulators for advanced processor architectures. It is a free closed source simulator binary that works with no compiling, no fiddling, and no external dependencies, it just works.
In the RISC-V world, Imperas simulators are used by most companies and organizations that are serious about getting quality RTL working and signed off. They use it as an architectural reference and many use it as the golden simulator to verify their RTL in a hardware design verification methodology. Imperas simulator technology and verification IP is the technology to use to obtain high quality results and to verify RTL works as expected.
The following is a list of some of the companies and organizations that rely on Imperas simulators as their RISC-V reference:
This download contains a binary of the Imperas configurable reference simulator, a test framework to run the simulator or your device-under-test, and tests to run on several targets. Use the tests to check ISA compliance of your
About
Releases
Packages
Contributors
Languages

Bremen Univ. 'Fuzz' tests



Closing the RISC-V Compliance Gap: Looking from the Negative Testing Side*

Vladimir Herdt[1] Daniel Große[1,2] Rolf Drechsler[1,2]
[1]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[2]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
{vherdt,grosse,drechsle}@informatik.uni-bremen.de

***Abstract*—Compliance testing for RISC-V is very important. Therefore, an official hand-written compliance test-suite is being actively developed. However, besides requiring significant manual effort, it focuses on positive testing (the implemented instructions work as expected) only and neglects negative testing (consider illegal instructions to also ensure that no additional/unexpected behavior is accidentally added). This leaves a large gap in compliance testing.**

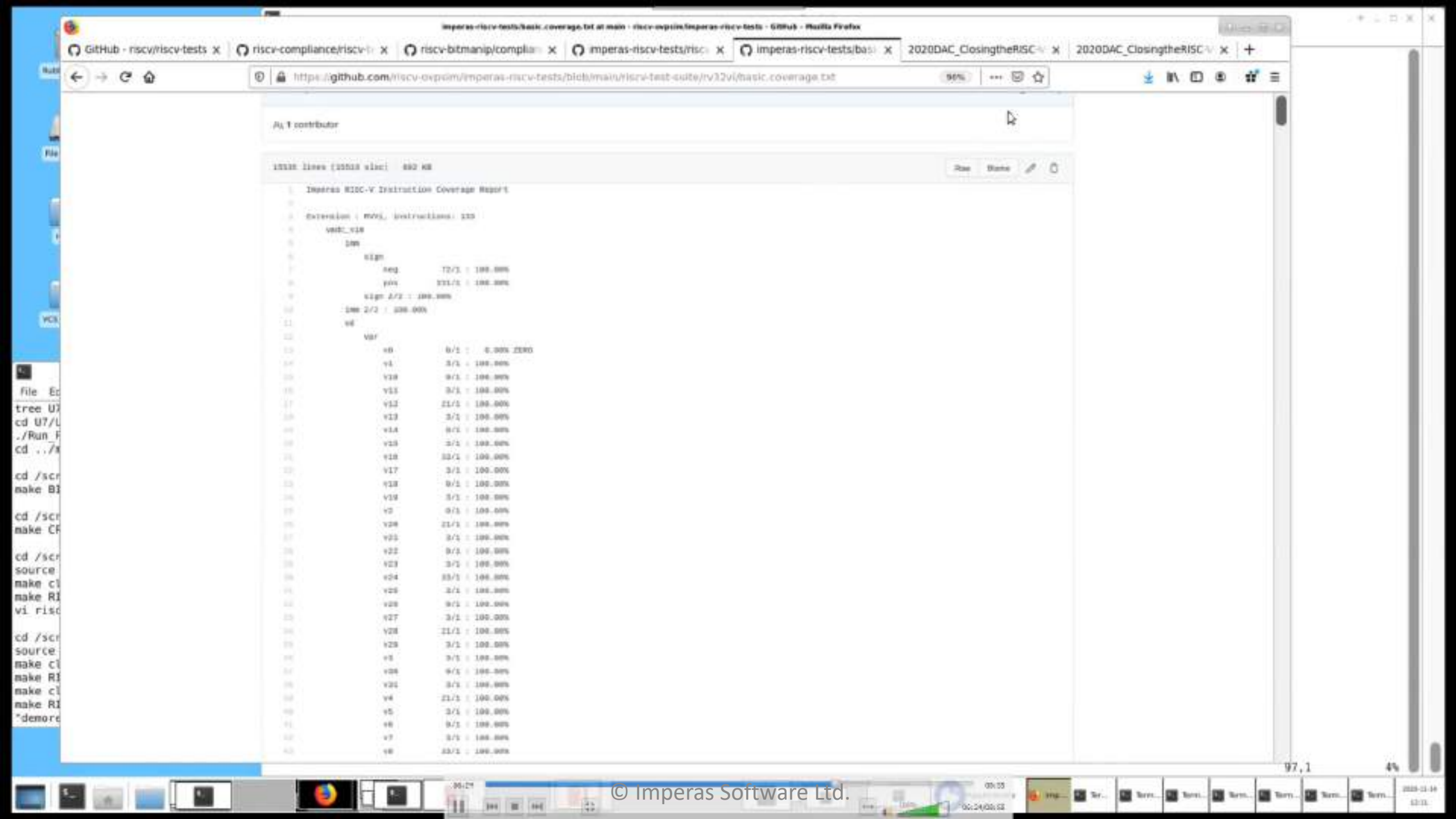
In this paper we propose a fuzzing-based test-suite generation approach to close this gap. We found new bugs in several RISC-V simulators including *riscvOVPsim* from Imperas which is the official reference simulator for compliance testing.

I. Introduction

An *Instruction Set Architecture* (ISA) defines the interface between the *Hardware* (HW) of a processor and the *Software* (SW). While, as a consequence, the format of a SW binary running on

that represents the output of the test result and is dumped at the end of the test execution. For compliance testing, these signatures are compared against golden reference signatures (obtained by running the test-suite on a reference simulator). A separate sub test-suite is developed for the RISC-V base ISA as well as for each standard ISA extension. Besides the significant manual effort for the maintenance, the compliance test-suite focuses on positive testing only, i.e. to show that the implemented instructions work as expected. However, it neglects negative testing, i.e. to consider illegal instructions to also ensure that no additional/unexpected behavior is accidentally added. This leaves open a large gap in compliance testing.

Contribution: In this paper we propose a fuzzing-based test-suite generation approach to close this gap. We leverage state-of-the-art fuzzing techniques (based on LLVM *libFuzzer*) to iteratively generate test-cases which are executed on a RISC-V simulator and



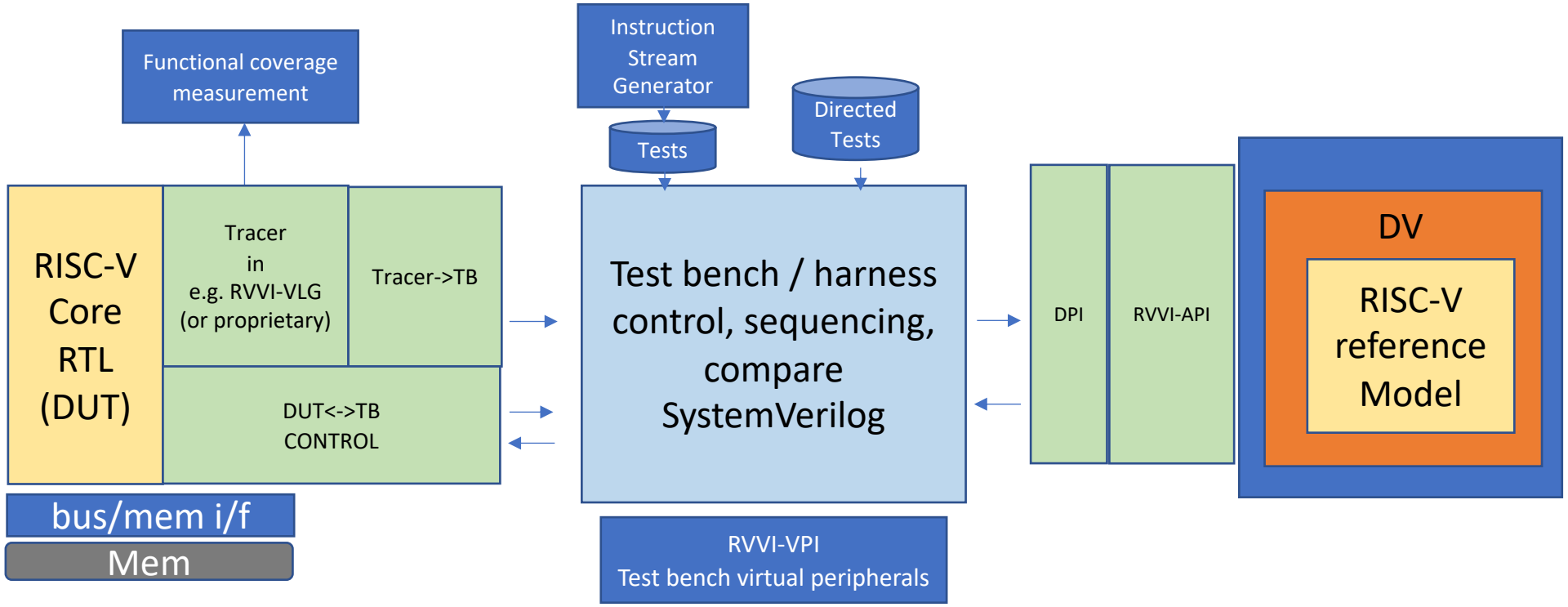
Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Other components of RISC-V CPU DV environments
  - Compliance Tests and other Test Suites
  - Instruction Stream Generators
  - Functional Coverage
  - Instruction Set Simulators

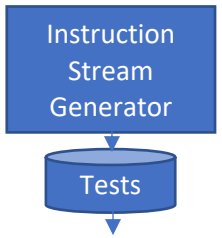
Agenda

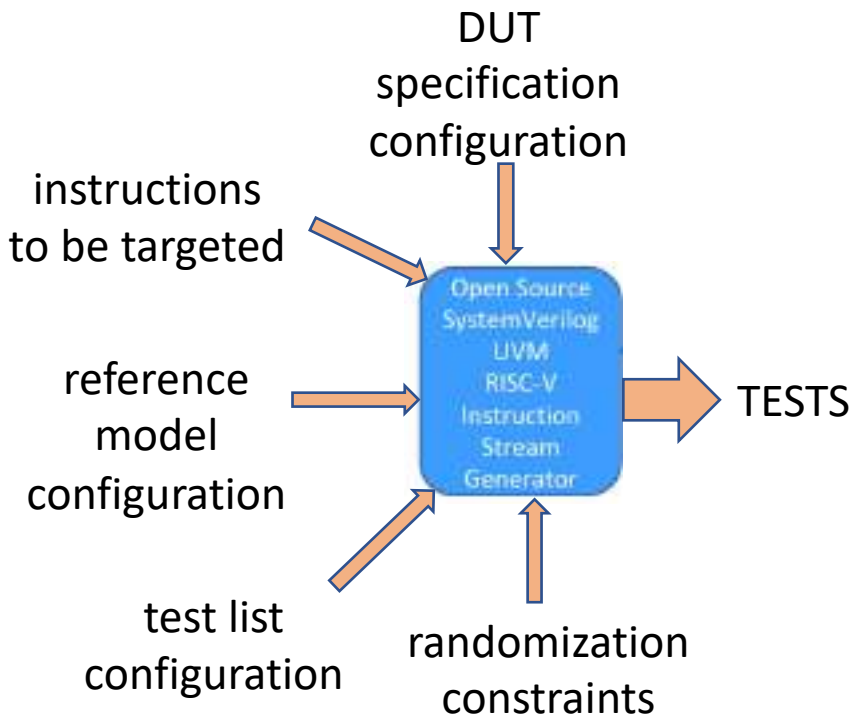
- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Other components of RISC-V CPU DV environments
  - Compliance Tests and other Test Suites
  - Instruction Stream Generators
    - Google Cloud riscv-dv (SystemVerilog open source)
    - OpenHW Group force-riscv (C++ open source)
    - Valtrix STING (commercial)
  - Functional Coverage
  - Instruction Set Simulators

CPU DV test bench components



Constrained Random Instruction Stream Test Generators (ISG)





Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Other components of RISC-V CPU DV environments
  - Compliance Tests and other Test Suites
  - Instruction Stream Generators
    - Google Cloud riscv-dv (SystemVerilog open source)
    - OpenHW Group force-riscv (C++ open source)
    - Valtrix STING (commercial)
  - Functional Coverage
  - Instruction Set Simulators

Key Features

imperas

Google Cloud

01

Randomness

Randomize everything: instruction, ordering, program structure, privileged mode setting, exceptions..

02

Architecture Aware

The generated program should be able to hit the corner cases of the processor architectural features.

03

Performance

The instruction generator should be scalable to generate a large program in a short period of time.

04

Extendability

Easy to add new instruction sequences, custom instruction extension, custom CSR etc.

- From Google Cloud presentation 2019 RISC-V Summit

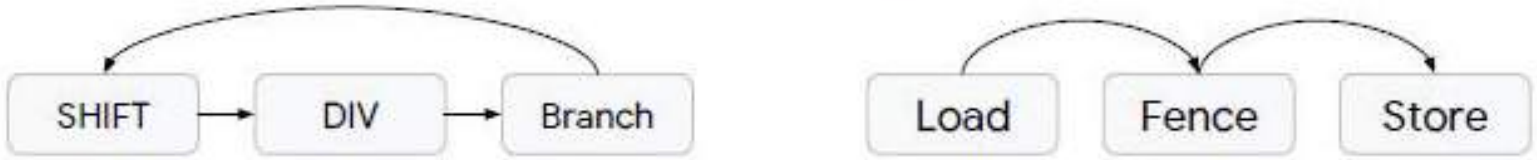
Randomness

Instruction level randomization

Cover all possible operands and immediate values of each instruction
Example: Arithmetic overflow, divide by zero, long branch, exceptions etc.

Sequence level randomization

Maximize the possibility of instruction orders and dependencies

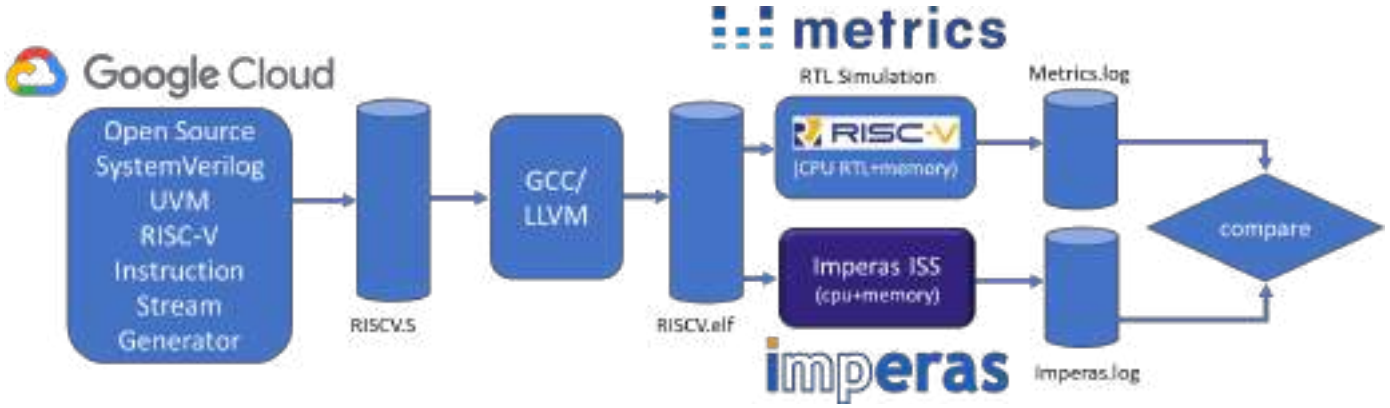


Program level randomization

Random privileged mode setting, page table organization, program calls

Google RISC-V Instruction Stream Generation

- High quality SystemVerilog UVM DV infrastructure
- Open source
- Drives a RISC-V core through corner cases and pushes it to the limit



https://github.com/google/riscv-dv

- Imperas worked on this with Google Cloud & Metrics through 2019-2020
- Uses a post-sim trace-compare methodology
- Uses Imperas riscvOVPsim as the reference

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Other components of RISC-V CPU DV environments
  - Compliance Tests and other Test Suites
  - Instruction Stream Generators
    - Google Cloud riscv-dv (SystemVerilog open source)
    - OpenHW Group force-riscv (C++ open source)
    - Valtrix STING (commercial)
  - Functional Coverage
  - Instruction Set Simulators

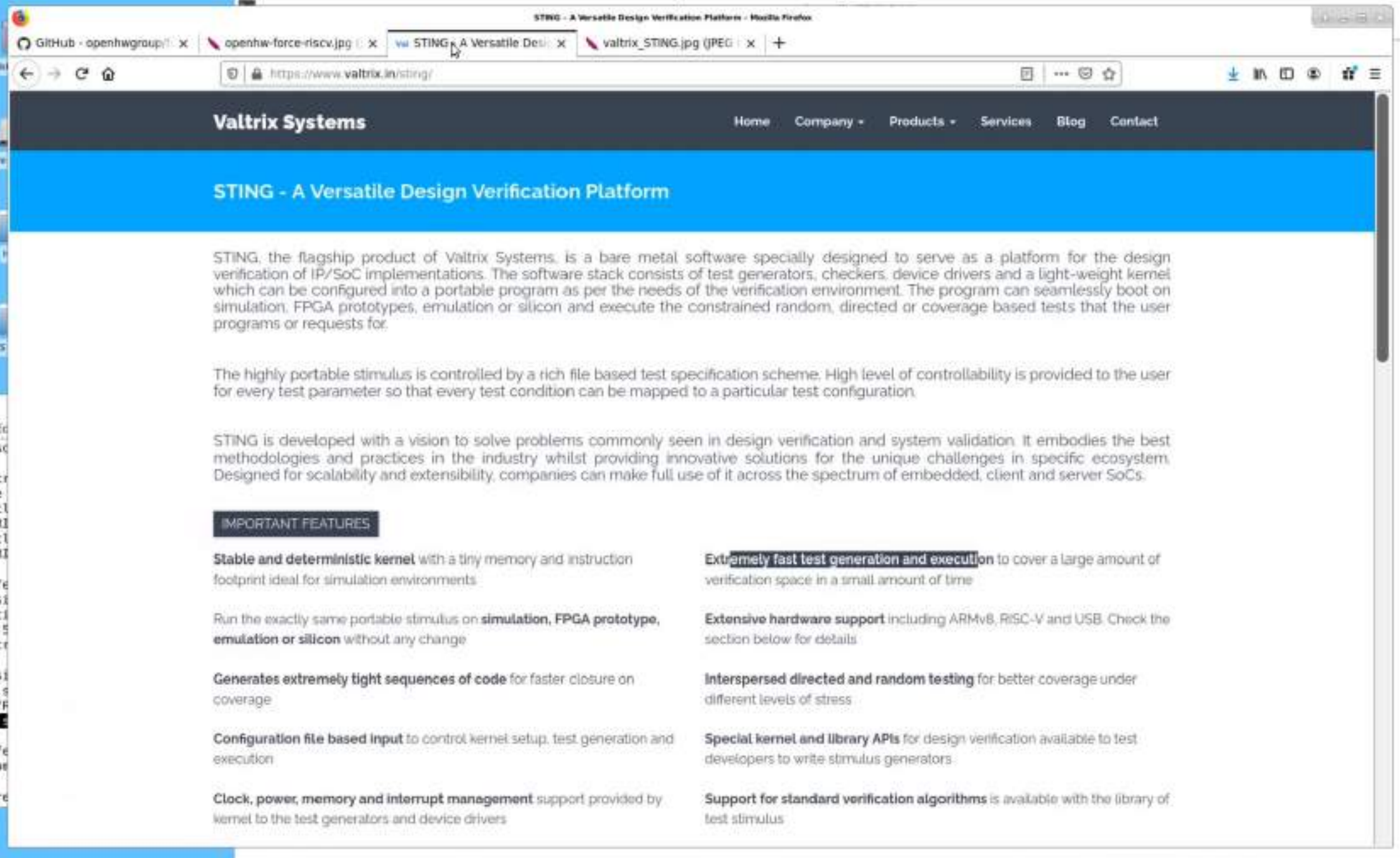
OpenHW Group force-riscv

imperas

- Developed initially by Futurewei
- Open source C++
- https://github.com/openhwgroup/force-riscv
- Initial focus on RV64, recently working on RV32
- Not yet key part of OpenHW flows for core-v cores

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Other components of RISC-V CPU DV environments
  - Compliance Tests and other Test Suites
  - Instruction Stream Generators
    - Google Cloud riscv-dv (SystemVerilog open source)
    - OpenHW Group force-riscv (C++ open source)
    - Valtrix STING (commercial)
  - Functional Coverage
  - Instruction Set Simulators


STING - A Versatile Design Verification Platform - Mozilla Firefox
GitHub - openhwgroup/
openhw-force-riscv.jpg
STING - A Versatile Des
valtrix_STING.jpg (JPEG
https://www.valtrix.in/sting/
Valtrix Systems
Home
Company
Products
Services
Blog
Contact
STING - A Versatile Design Verification Platform
STING, the flagship product of Valtrix Systems, is a bare metal software specially designed to serve as a platform for the design verification of IP/SoC implementations. The software stack consists of test generators, checkers, device drivers and a light-weight kernel which can be configured into a portable program as per the needs of the verification environment. The program can seamlessly boot on simulation, FPGA prototypes, emulation or silicon and execute the constrained random, directed or coverage based tests that the user programs or requests for.
The highly portable stimulus is controlled by a rich file based test specification scheme. High level of controllability is provided to the user for every test parameter so that every test condition can be mapped to a particular test configuration.
STING is developed with a vision to solve problems commonly seen in design verification and system validation. It embodies the best methodologies and practices in the industry whilst providing innovative solutions for the unique challenges in specific ecosystem. Designed for scalability and extensibility, companies can make full use of it across the spectrum of embedded, client and server SoCs.
IMPORTANT FEATURES
Stable and deterministic kernel with a tiny memory and instruction footprint ideal for simulation environments
Run the exactly same portable stimulus on simulation, FPGA prototype, emulation or silicon without any change
Generates extremely tight sequences of code for faster closure on coverage
Configuration file based input to control kernel setup, test generation and execution
Clock, power, memory and interrupt management support provided by kernel to the test generators and device drivers
Extremely fast test generation and execution to cover a large amount of verification space in a small amount of time
Extensive hardware support including ARMv8, RISC-V and USB. Check the section below for details
Interspersed directed and random testing for better coverage under different levels of stress
Special kernel and library APIs for design verification available to test developers to write stimulus generators
Support for standard verification algorithms is available with the library of test stimulus

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Other components of RISC-V CPU DV environments
  - Compliance Tests and other Test Suites
  - Instruction Stream Generators
  - Functional Coverage
  - Instruction Set Simulators

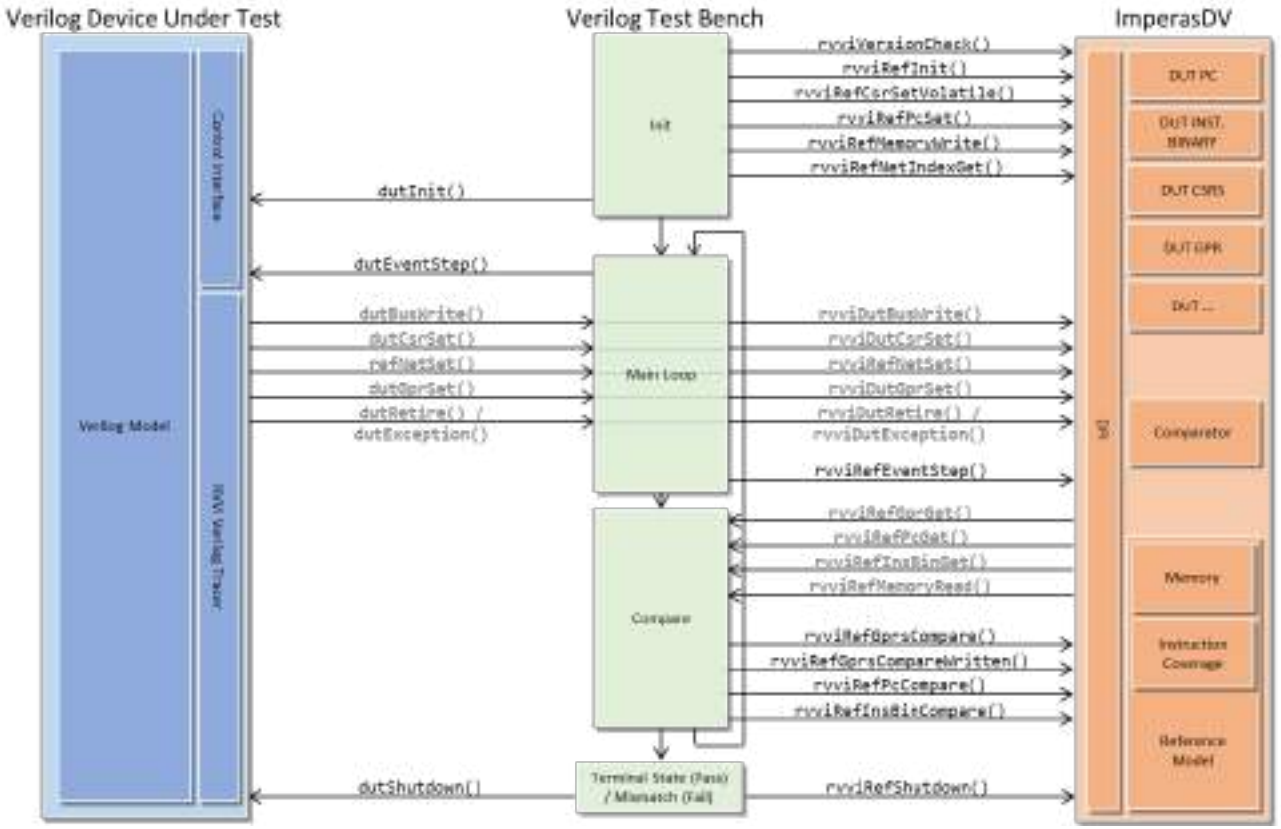
Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Other components of RISC-V CPU DV environments
  - Compliance Tests and other Test Suites
  - Instruction Stream Generators
  - Functional Coverage
    - Imperas Built-in Instruction Coverage
    - SystemVerilog Covergroups, Coverpoints, Assertions
  - Instruction Set Simulators

Imperas Instruction Coverage

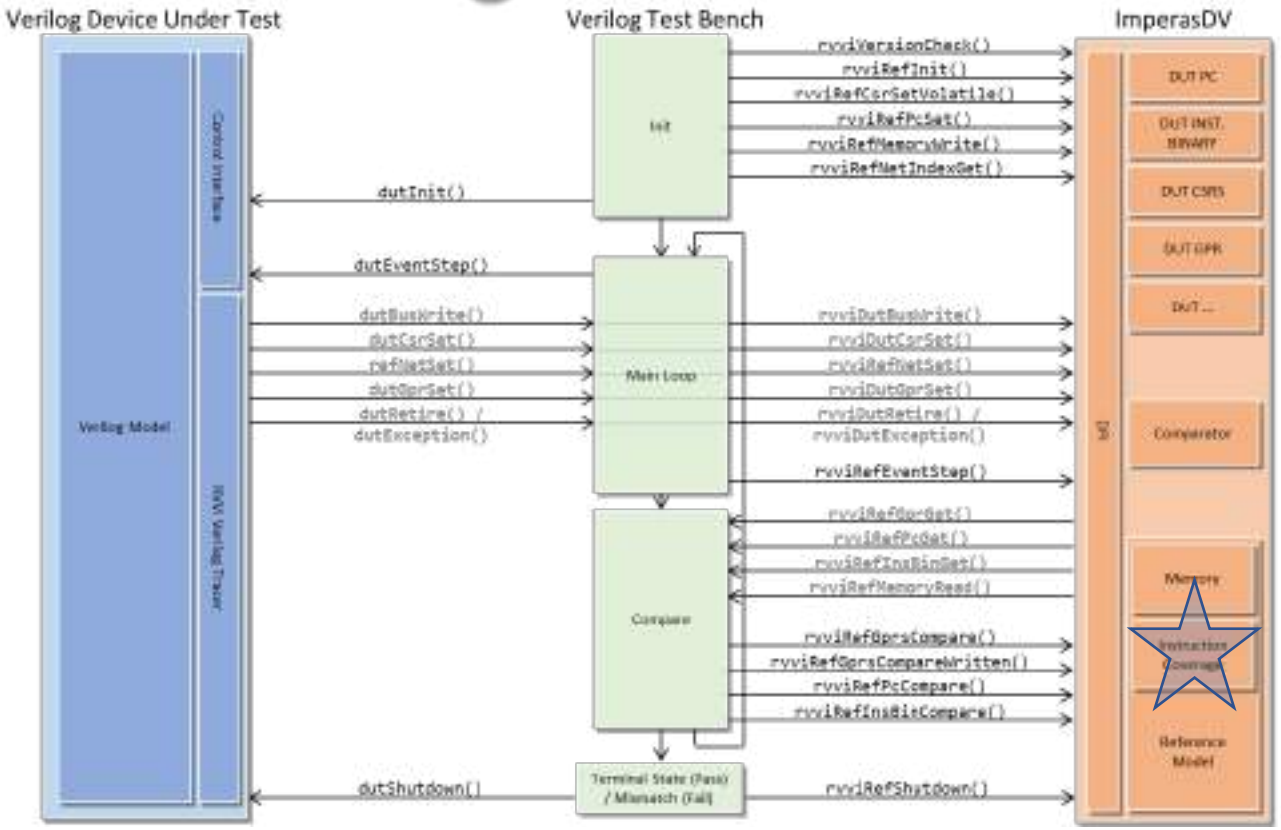
- Built-in as ‘extension library’ to Imperas models
- Works with all Imperas simulators and RISC-V models
- Focus is for measuring ‘architectural validation tests’
  - i.e. measurement of basic architectural operations
  - Not intended for measuring HW DV testing (use SystemVerilog for that)
- Controlled from command line – tailor options for each run
  - Outputs .txt file, and .yaml file of data
- After individual runs, collates data into ‘suite measured coverage’
- Shows what has, and has not, been covered
- Selectable coverage focus
  - --extensions, --instructions
  - --mnemonic, --basic, --extended

ImperasDV Instruction Coverage



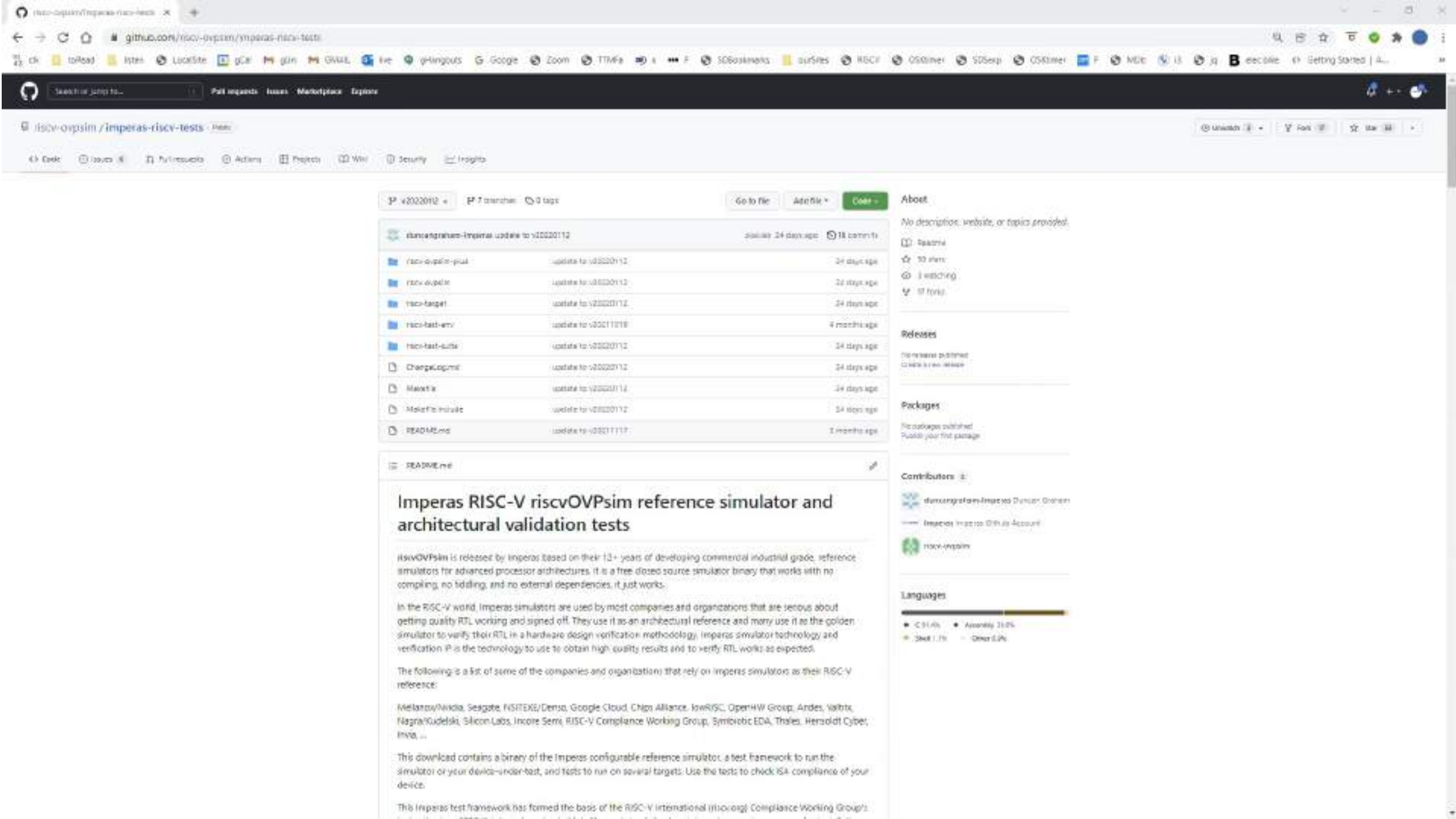
- Reference model setup
- Configuration of register and memory initialization
- Selection of what to compare (depends on DUT ‘tracer’ capabilities) :
  - PC, GPR, CSR, FPR, VR, decode, net, hart…
- Select capabilities:
  - sync-lock-step-compare or async-lock-step-compare
- Trace and logging set up
- Selection of built-in Imperas instruction coverage
- Choice of DV control options

ImperasDV Instruction Coverage



- Reference model setup
- Configuration of register and memory initialization
- Selection of what to compare (depends on DUT 'tracer' capabilities) :
  - PC, GPR, CSR, FPR, VR, decode, net, hart…
- Select capabilities:
  - sync-lock-step-compare or async-lock-step-compare
- Trace and logging set up
- Selection of built-in Imperas instruction coverage
- Choice of DV control options

- ImperasDV includes built-in Imperas Instruction Coverage



riscv-ovpsim / imperas-riscv-tests

Pull requests Issues Marketplace Explore

Code Issues 6 Pull requests Actions Projects Wiki Security Insights

v20220112 · 7 branches · 0 tags

Go to file · Add file · Code

duncangraham-Imperas update to v20220112 — latest 24 days ago · 18 commits

riscv-ovpsim-pua	update to v20220112	24 days ago
riscv-ovpsim	update to v20220112	24 days ago
riscv-target	update to v20220112	24 days ago
riscv-test-env	update to v20211019	4 months ago
riscv-test-suite	update to v20220112	24 days ago
ChangeLog.md	update to v20220112	24 days ago
Makefile	update to v20220112	24 days ago
Makefile.include	update to v20220112	24 days ago
README.md	update to v20211117	3 months ago

README.md

Imperas RISC-V riscvOVPsim reference simulator and architectural validation tests

riscvOVPsim is released by Imperas based on their 12+ years of developing commercial industrial grade, reference simulators for advanced processor architectures. It is a free closed source simulator binary that works with no compiling, no tiddling, and no external dependencies, it just works.

In the RISC-V world, Imperas simulators are used by most companies and organizations that are serious about getting quality RTL working and signed off. They use it as an architectural reference and many use it as the golden simulator to verify their RTL in a hardware design verification methodology. Imperas simulator technology and verification IP is the technology to use to obtain high quality results and to verify RTL works as expected.

The following is a list of some of the companies and organizations that rely on Imperas simulators as their RISC-V reference:

Mellanox/Nvidia, Seagate, NSITEXE/Denso, Google Cloud, Chips Alliance, lowRISC, OpenHW Group, Andes, Vaibhx, Nagra/Kudelski, Silicon Labs, Incore Semi, RISC-V Compliance Working Group, Symbiotic EDA, Thales, Herzoldt Cyber, Inria, ...

This download contains a binary of the Imperas configurable reference simulator, a test framework to run the simulator or your device-under-test, and tests to run on several targets. Use the tests to check ISA compliance of your device.

This Imperas test framework has formed the basis of the RISC-V international (riscv.org) Compliance Working Group's

About

No description, website, or topics provided.

Releases

Packages

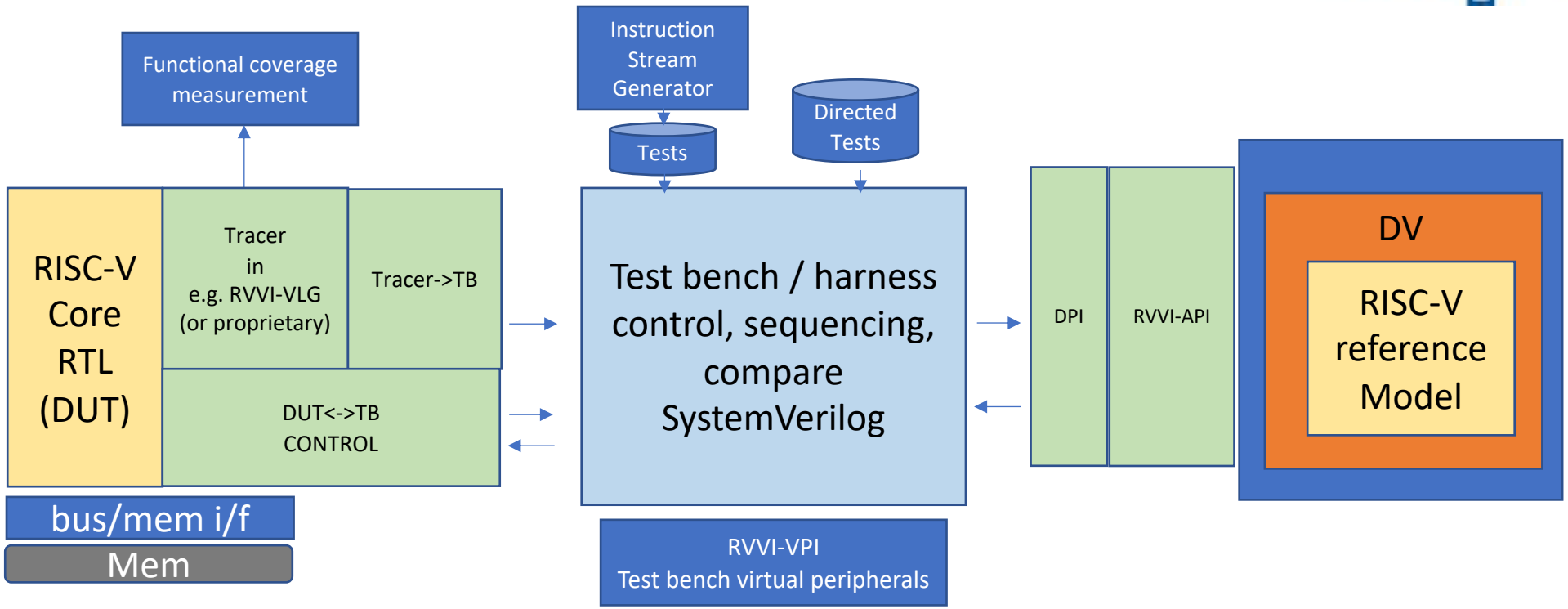
Contributors

Languages

Agenda

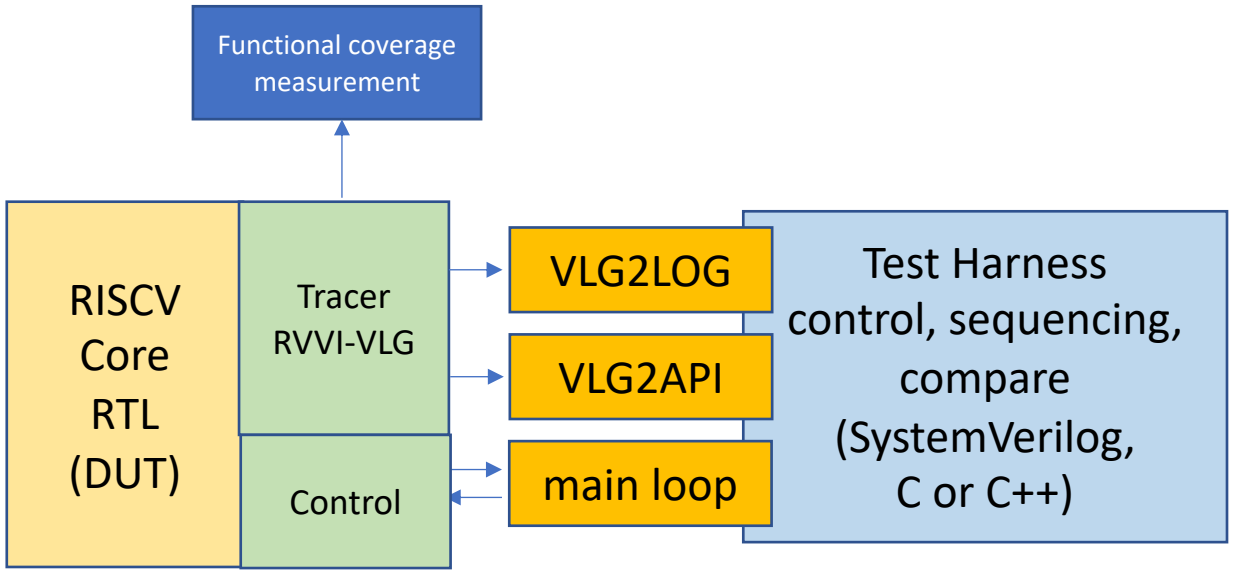
- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Other components of RISC-V CPU DV environments
  - Compliance Tests and other Test Suites
  - Instruction Stream Generators
  - Functional Coverage
    - Imperas Built-in Instruction Coverage
    - SystemVerilog Covergroups, Coverpoints, Assertions
  - Instruction Set Simulators

CPU DV test bench components
SystemVerilog Functional Coverage



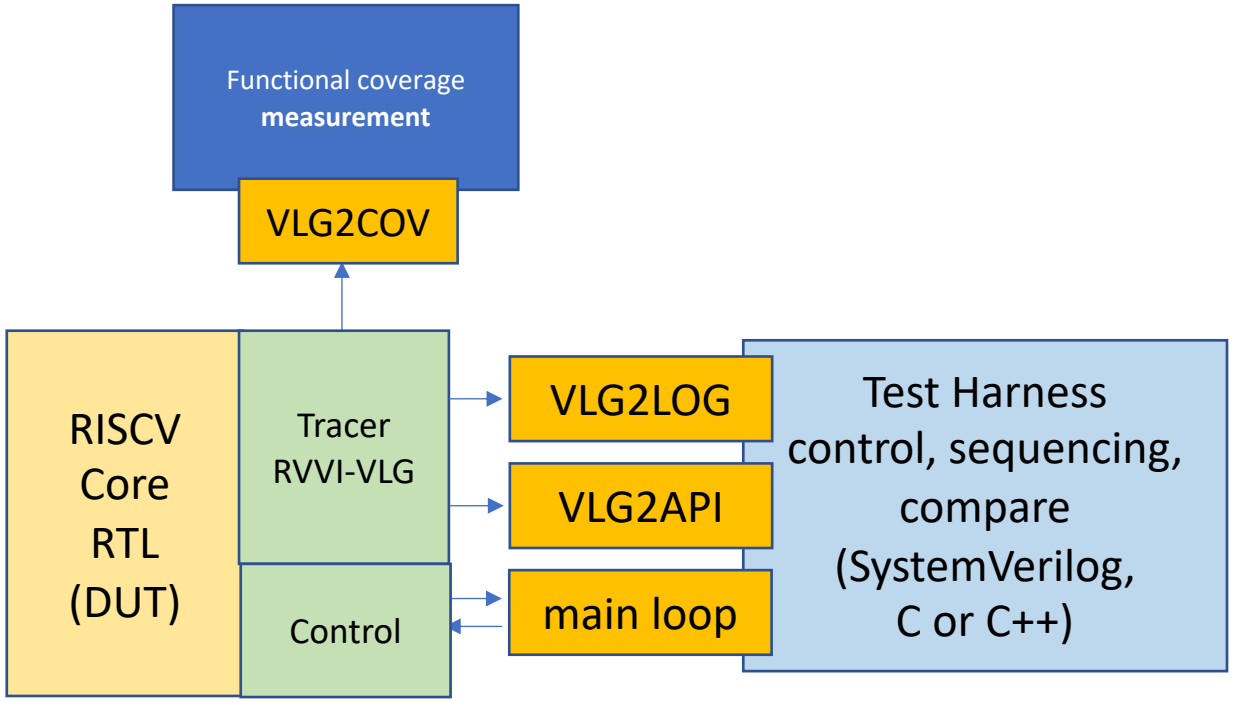
- Recall there needs to functional coverage connected to the 'tracer'
- To measure what the architecture and micro-architecture is doing…

CPU DV test bench components
SystemVerilog Functional Coverage (2)



- And we saw earlier in the detailed ImperasDV walkthrough the connections from the 'tracer' via the RVVI
  - From the standard RVVI-VLG there are reusable source 'clients' that connect the 'tracer' to other components

CPU DV test bench components
SystemVerilog Functional Coverage (3)



- We can connect a functional coverage subsystem in the same way

Connecting RVVI to Covergroups

- VLG2COV connects to ‘tracer’ and is informed when events like an instruction retires or interrupt is taken
- On these events VLG2COV
  - calls ‘sample’ in coverage class
  - Which then calls the appropriate covergroup sample functions

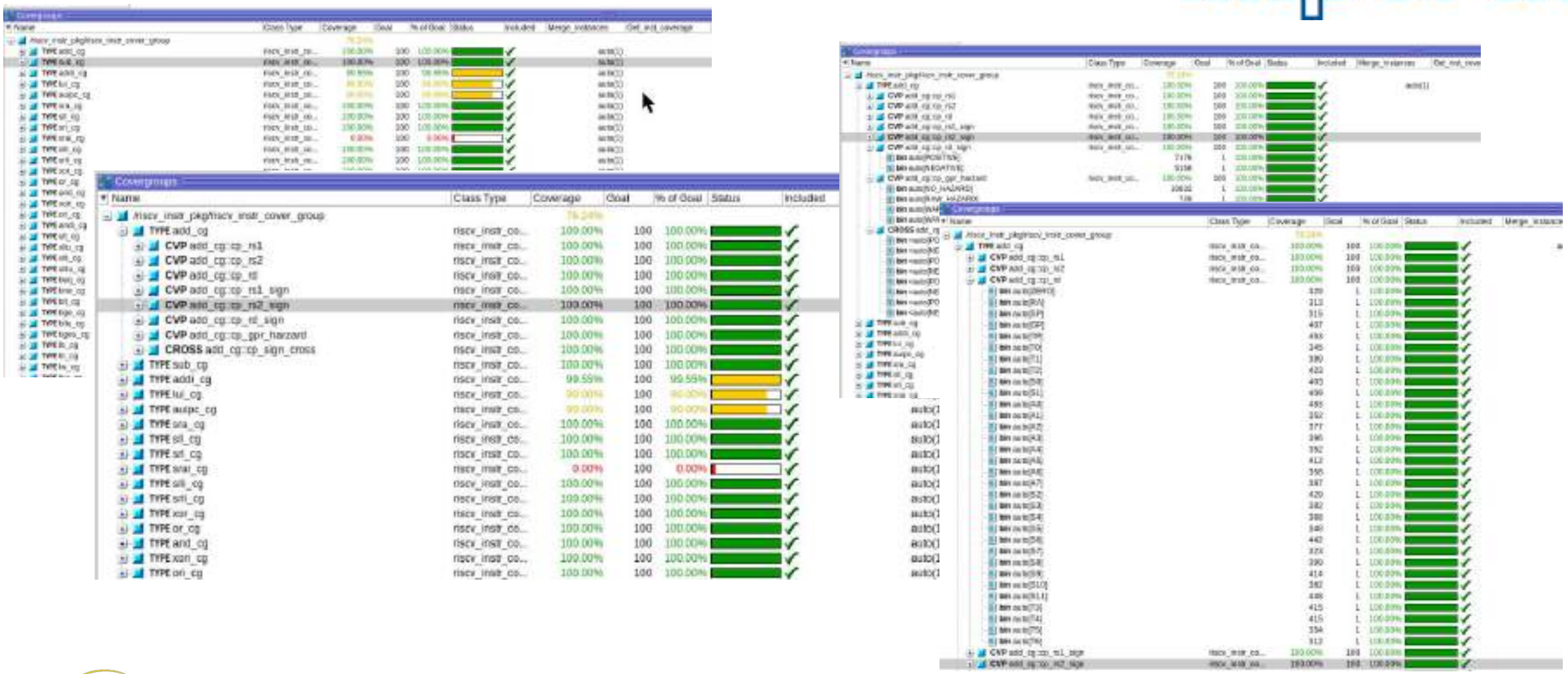
```
  covergroup ins_cg with function sample(string ins_str);
      option.per_instance = 1;
      cp_asm : coverpoint get_asm_enum(ins_str);
  endgroup

  function new();
      ins_cg = new();
  endfunction

  function void sample(input string decode);
      string ins_str, op[4], key, val;
      int num = $sscanf (decode, "%s %s %s %s %s", ins_str, op[0], op[1], op[2], op[3]);
      ins_cg.sample(ins_str);
  endfunction

endclass
```

```
  covergroup sub_cg with function sample(ins_t ins);
      option.per_instance = 1;
      cp_rd    : coverpoint get_gpr_name(ins.ops[0].val, ins.ops[0].key, "sub");
      cp_rs1   : coverpoint get_gpr_name(ins.ops[1].val, ins.ops[1].key, "sub");
      cp_rs2   : coverpoint get_gpr_name(ins.ops[2].val, ins.ops[2].key, "sub");
  endgroup

  covergroup sw_cg with function sample(ins_t ins);
      option.per_instance = 1;
      cp_rs2   : coverpoint get_gpr_name(ins.ops[0].val, ins.ops[0].key, "sw");
      cp_r     : coverpoint get_gpr_name(ins.ops[1].val, ins.ops[1].key, "sw");
      cp_imm   : coverpoint get_imm(ins.ops[2].key, "sw") {
          bins neg  = {[$:-1]};
          bins zero = {0};
          bins pos  = {[1:$]};
      }
  endgroup

  covergroup wfi_cg with function sample(ins_t ins);
      option.per_instance = 1;
      cp_asm   : coverpoint ins.asm == WFI {
          ignore_bins zero = {0};
      }
  endgroup

  covergroup xor_cg with function sample(ins_t ins);
      option.per_instance = 1;
      cp_rd    : coverpoint get_gpr_name(ins.ops[0].val, ins.ops[0].key, "xor");
      cp_rs1   : coverpoint get_gpr_name(ins.ops[1].val, ins.ops[1].key, "xor");
      cp_rs2   : coverpoint get_gpr_name(ins.ops[2].val, ins.ops[2].key, "xor");
  endgroup
```

RVVI Functional Coverage

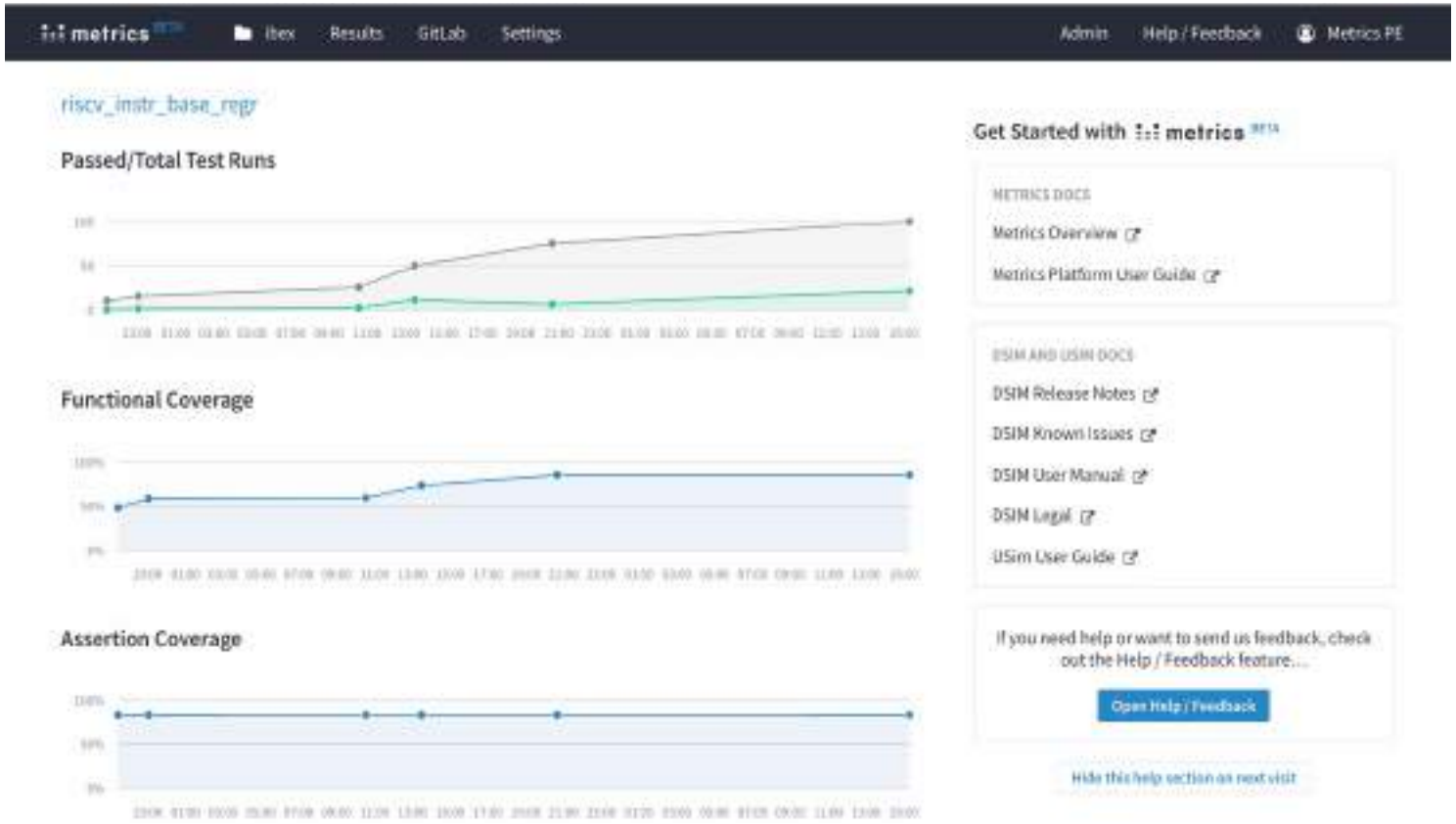
- So the use of a standard interface from DUT ‘tracer’ to testbench means reusable standard components can be developed
  - A key one is a SystemVerilog Functional Coverage sub system
- And for RISC-V standard extensions they can be provided as SystemVerilog source
  - And then extended for DV specifics of the specific micro-architecture
    - Such as pipeline issues, hazards, assertions – design specific items
- Imperas has an example available (Feb2022) and will shortly release others
  - (contact Imperas for more information)

SystemVerilog simulators provide Coverage reports



- Coverage reports from Siemens/Mentor Questa

Metrics: includes top level overview dashboard



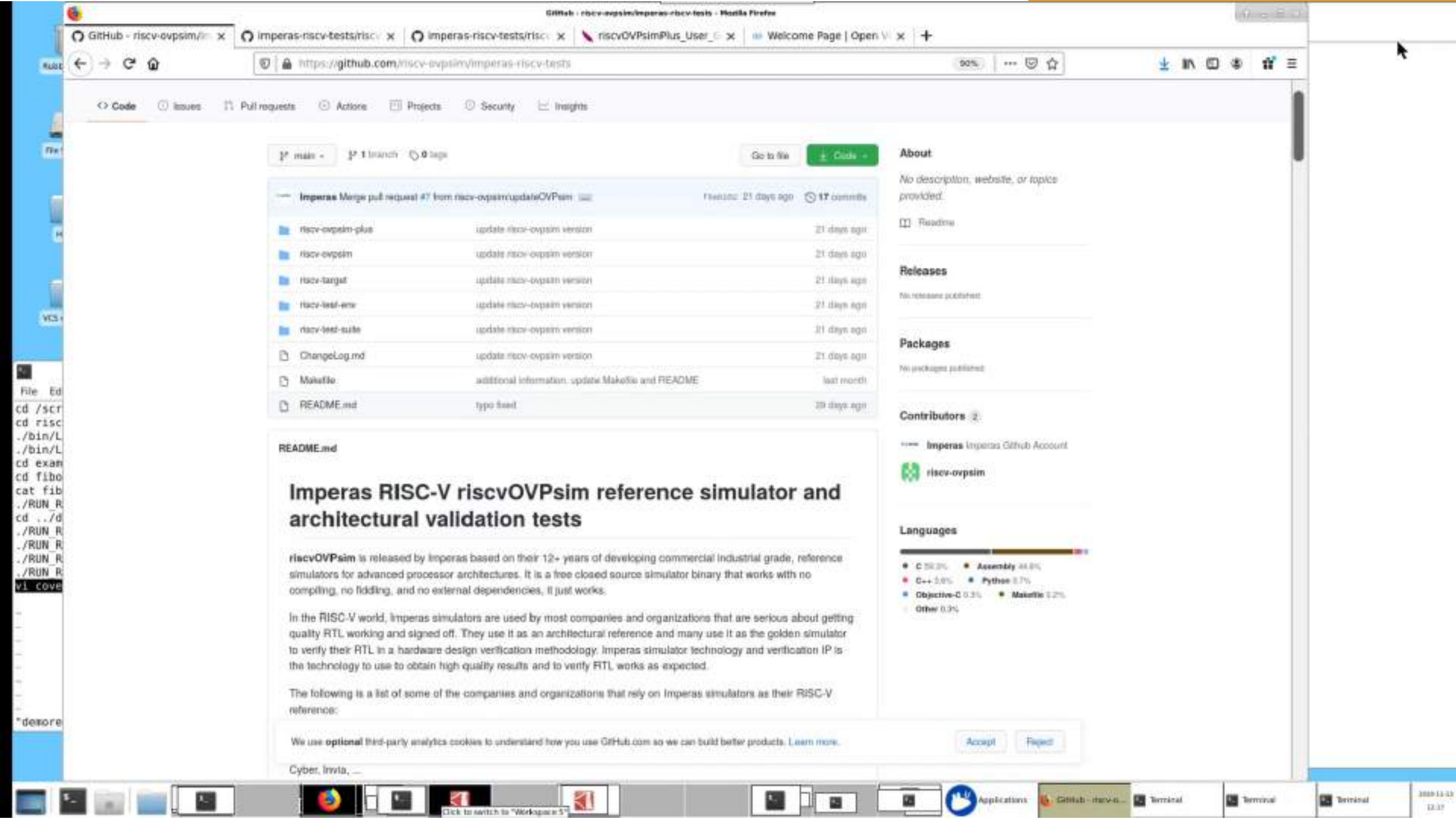
- Allows management overview of status of verification

Agenda

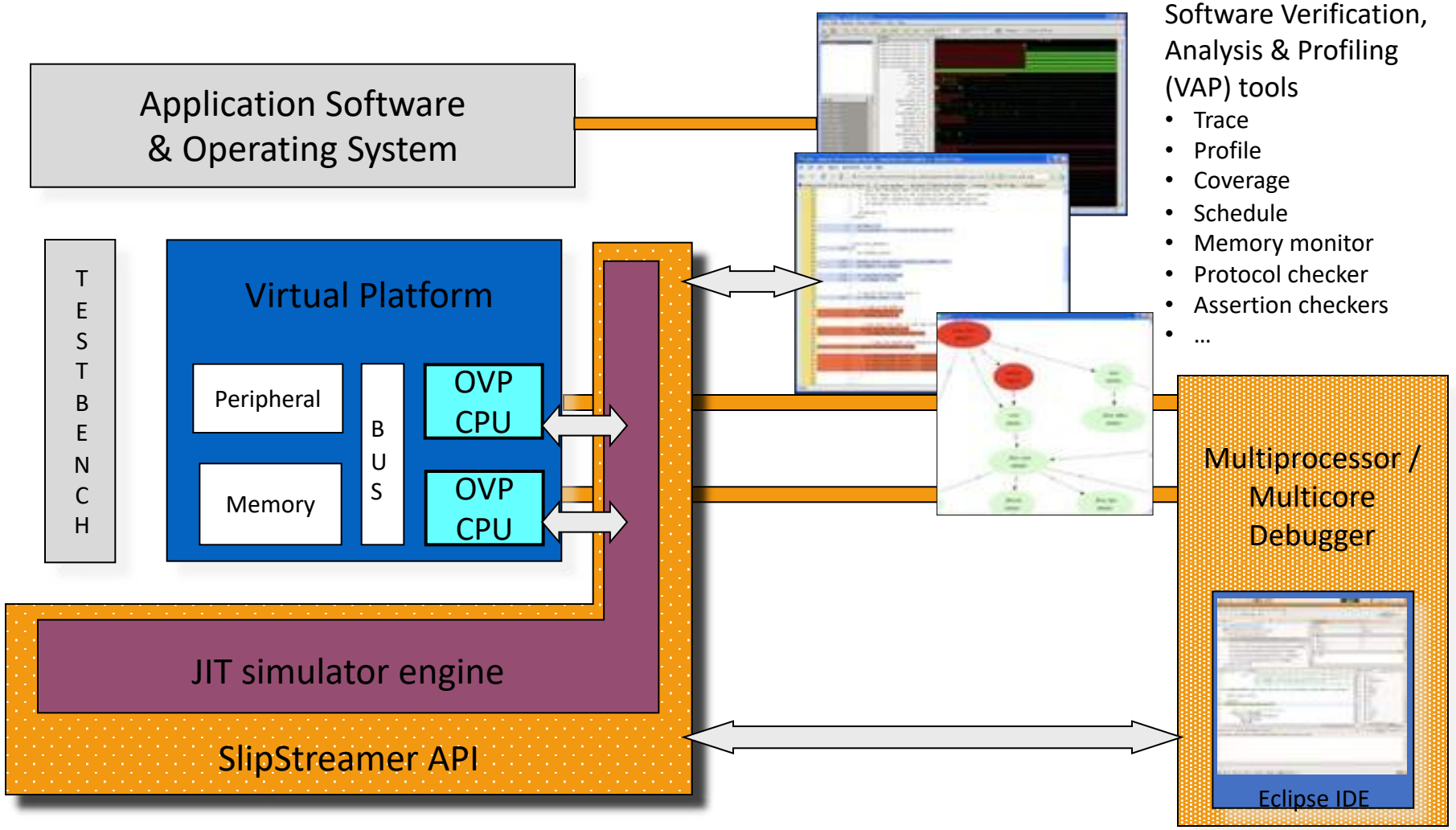
- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Other components of RISC-V CPU DV environments
  - Compliance Tests and other Test Suites
  - Instruction Stream Generators
  - Functional Coverage
  - Instruction Set Simulators

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Other components of RISC-V CPU DV environments
  - Compliance Tests and other Test Suites
  - Instruction Stream Generators
  - Functional Coverage
  - Instruction Set Simulators
    - Free riscvOVPsim (github)
    - Free riscvOVPsimPlus (ovpworld.org)
    - Commercial M*SIM (imperas.com)
    - Commercial M*SDK (imperas.com)
    - Commercial ImperasDV (imperas.com)


GitHub - riscv-ovpsim/imperas-riscv-tests - Mozilla Firefox
https://github.com/riscv-ovpsim/imperas-riscv-tests
Code
Issues
Pull requests
Actions
Projects
Security
Insights
main
1 branch
0 tags
Go to file
Code
Imperas Merge pull request #7 from riscv-ovpsim/updateOVPsim
17 commits
riscv-ovpsim-plus
riscv-ovpsim
riscv-target
riscv-test-env
riscv-test-suite
ChangeLog.md
Makefile
README.md
update riscv-ovpsim version
additional information, update Makefile and README
typo fixed
About
No description, website, or topics provided.
Readme
Releases
No releases published
Packages
No packages published
Contributors 2
Imperas Imperas Github Account
riscv-ovpsim
Languages
README.md
Imperas RISC-V riscvOVPsim reference simulator and architectural validation tests
riscvOVPsim is released by Imperas based on their 12+ years of developing commercial industrial grade, reference simulators for advanced processor architectures. It is a free closed source simulator binary that works with no compiling, no fiddling, and no external dependencies, it just works.
In the RISC-V world, Imperas simulators are used by most companies and organizations that are serious about getting quality RTL working and signed off. They use it as an architectural reference and many use it as the golden simulator to verify their RTL in a hardware design verification methodology. Imperas simulator technology and verification IP is the technology to use to obtain high quality results and to verify RTL works as expected.
The following is a list of some of the companies and organizations that rely on Imperas simulators as their RISC-V reference:
We use optional third-party analytics cookies to understand how you use GitHub.com so we can build better products. Learn more.
Accept
Reject
Cyber, Invia, ...

Imperas Tools for Embedded Software Development, Debug & Test



Software Verification, Analysis & Profiling (VAP) tools

- Trace
- Profile
- Coverage
- Schedule
- Memory monitor
- Protocol checker
- Assertion checkers
- …

Welcome to Imperas | Imperas - Embedded Software Development - Mozilla Firefox

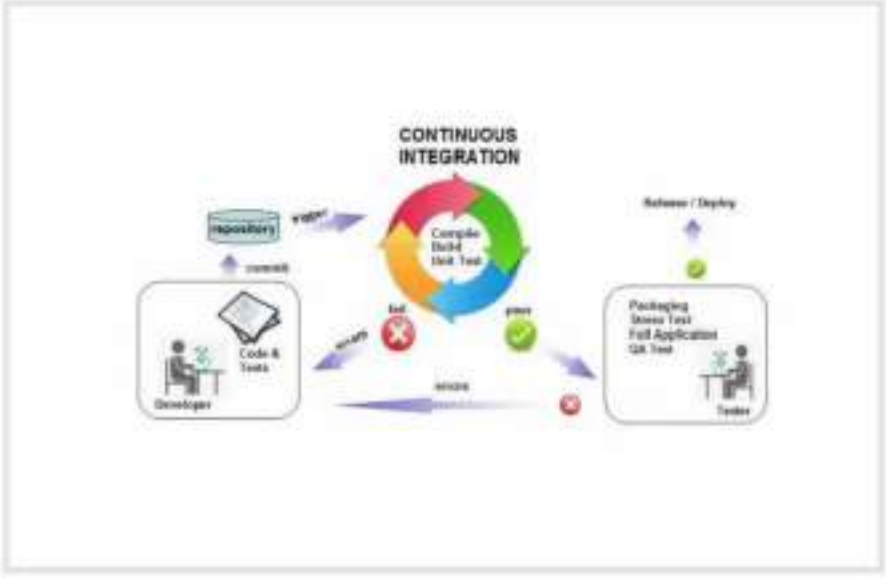
GitHub - riscv-ovpsim | imperas-riscv-tests | imperas-riscv-tests | riscvOVPsimPlus_User | Welcome Page | Open | Welcome to Imperas | Index of /home/simon | Index of /home/simon

https://www.imperas.com

User Login | Contact Us | SEARCH

imperas

Home | Products | Solutions | OVPworld | Resources | News | About Us

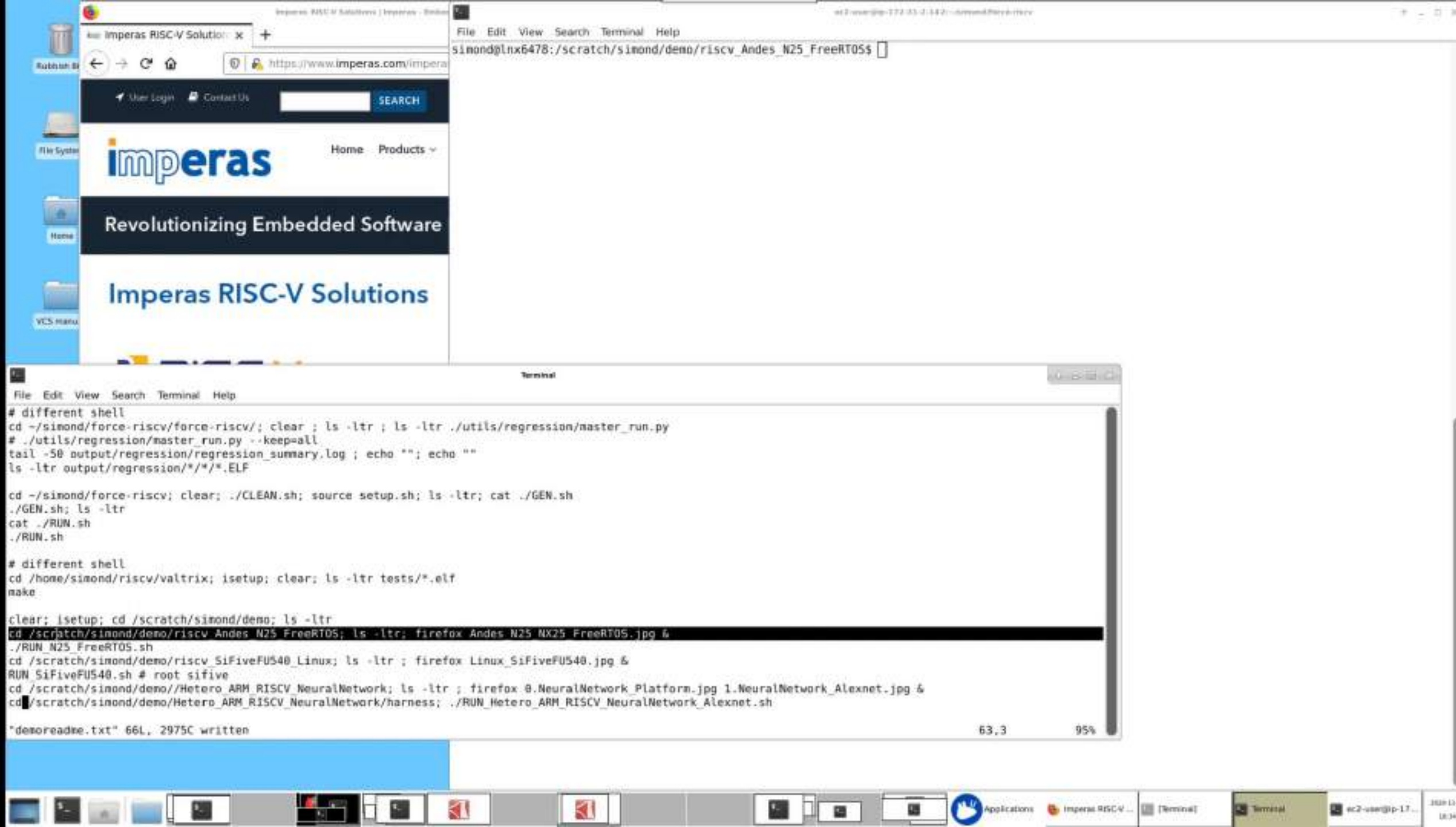


Imperas Continuous Integration

Welcome to Imperas - Revolutionizing Embedded Software Solutions

File Ed
cd /scr
cd risc
./bin/L
./bin/L
cd exan
cd fibo
cat fib
./RUN_R
cd ../d
./RUN_R
./RUN_R
./RUN_R
./RUN_R
vi cove
cd ~/In
cd ~/In
tree U7
cd U7/U
./Run_P
cd ../n
-- INSE

Applications | Welcome to Imp... | Terminal | Terminal | Terminal


Imperas RISC-V Solutions
https://www.imperas.com/impera
User Login
Contact Us
SEARCH
imperas
Home
Products
Revolutionizing Embedded Software
Imperas RISC-V Solutions
File Edit View Search Terminal Help
simond@lnx6478:/scratch/simond/demo/riscv_Andes_N25_FreeRTOS$
Terminal
File Edit View Search Terminal Help
different shell
cd ~/simond/force-riscv/force-riscv/; clear ; ls -ltr ; ls -ltr ./utils/regression/master_run.py
./utils/regression/master_run.py --keep=all
tail -50 output/regression/regression_summary.log ; echo ""; echo ""
ls -ltr output/regression/*/*/*.ELF
cd ~/simond/force-riscv; clear; ./CLEAN.sh; source setup.sh; ls -ltr; cat ./GEN.sh
./GEN.sh; ls -ltr
cat ./RUN.sh
./RUN.sh
different shell
cd /home/simond/riscv/valtrix; isetup; clear; ls -ltr tests/*.elf
make
clear; isetup; cd /scratch/simond/demo; ls -ltr
cd /scratch/simond/demo/riscv_Andes_N25_FreeRTOS; ls -ltr; firefox Andes_N25_NX25_FreeRTOS.jpg &
./RUN_N25_FreeRTOS.sh
cd /scratch/simond/demo/riscv_SiFiveFU540_Linux; ls -ltr ; firefox Linux_SiFiveFU540.jpg &
RUN_SiFiveFU540.sh # root sifive
cd /scratch/simond/demo//Hetero_ARM_RISCV_NeuralNetwork; ls -ltr ; firefox 0.NeuralNetwork_Platform.jpg 1.NeuralNetwork_Alexnet.jpg &
cd /scratch/simond/demo/Hetero_ARM_RISCV_NeuralNetwork/harness; ./RUN_Hetero_ARM_RISCV_NeuralNetwork_Alexnet.sh
"demoreadme.txt" 66L, 2975C written
63,3
95%
Applications
Imperas RISC-V ...
[Terminal]
Terminal
ec2-user@ip-17...

ISS: Summary

- Only mentioned Imperas simulators (as we use daily, our customers rely on them , and we understand their quality)
  - There are are others…
    - There are many open source grad. student project simulators… - go search github…
    - RISC-V has a formal model under development – 'sail'
    - There is also spike from Berkeley – for architectural exploration
    - And there are full system software emulators like qemu
- Free: github: riscvOVPsim.exe
  - Model selection configuration
  - Signature, logs, coverage
  - Includes rv32I tests
  - Useful for compliance tests
- Free: ovpworld.org (needs registration): riscvOVPsimPlus.exe
  - As riscvOVPsimPlus
  - trace, debug
  - Useful for post-sim-trace-compare, e.g. in Google riscv-dv
- Commercial from Imperas – the industry leader in processor based simulation solutions
  - Full range of ISS, virtual platforms, full system emulation, fixed & extendable platforms
  - 50 reference platforms, 250+ peripheral components, 300+ processor models

Agenda

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Other components of RISC-V CPU DV environments
- Summary

- Congratulations… on getting to the end of this tutorial

- if you got this far – and still have some energy…
  - send us an email (info@imperas.com) with
    - Subject: imperasdv at dvcon22 tutorial
    - With: comments on this tutorial
  - and we will send the first 50 of you one of our ImperasDV drinking mugs

We covered…

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
- Components of RISC-V CPU DV environment

We covered (2)

- Brief Introduction to RISC-V
- RISC-V CPU HW DV approaches
  - #0 “hello world” test
  - #1 self checking tests (e.g. Berkeley torture tests pre2018)
  - #2 Post simulation trace log file compare (e.g. Google riscv-dv 2019)
  - #3 sync-lock-step-compare (e.g. CV32E40P in OpenHW 1H 2020)
  - #4 async-lock-step-compare (e.g. CV32E40P in OpenHW 2H H2020)
  - #5 test bench use of standards (RVVI) (e.g. CV32E40X/S in OpenHW 2021)
  - #6 using standards based DV products and VIP (ImperasDV)
    - And had a 35 minute walk through

We covered (3)

- Components of RISC-V CPU DV environment
  - Compliance Tests and other Test Suites
    - RISC-V architectural test suites
    - Imperas architectural test suites
    - Bremen fuzz testing
  - Instruction Stream Generators
    - Google riscv-v
    - OpenHW force-riscv
    - Valtrix STING
  - Functional Coverage
    - Imperas build-in instruction coverage
    - SystemVerilog covergroups and coverpoints
  - Instruction Set Simulators & tools
    - Free riscvOVPsim (github)
    - Free riscvOVPsimPlus (ovpworld.org)
    - Commercial M*SIM (imperas.com)
    - Commercial M*SDK (imperas.com)
    - Commercial ImperasDV (imperas.com)

What we did not talk about…

- SystemVerilog encapsulation of Imperas models (yes we do that)
- Using ISS with RTL emulators
  - Including hybrid simulation (yes we do that)
- Formal tools
  - (no we don't do these…)

Summary

- The open standard ISA of RISC-V offers many design freedoms
  - Many standard extensions and configuration options plus custom instructions
- The key verification requirements are to detect discrepancies with efficient debug
- The open standard RVVI offers a framework for verification reuse with support for both open-source and commercial tools
  - RISC-V Verification Interface
  - https://github.com/riscv-verification/RVVI
- Lockstep / Compare is by far the best and most efficient approach (industry 'gold standard')
  - https://www.imperas.com/imperasdv

Thank You!

info@imperas.com

www.imperas.com

www.imperas.com/ImperasDV

www.OVPworld.org

Questions?