2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

imperas Google Cloud metrics

Rolling the dice with random instructions is the safe bet on RISC-V verification

Simon Davidmann and Lee Moore - Imperas Software Ltd.

Richard Ho and Tao Liu - Google LLC.

Doug Letcher and Aimee Sutton - Metrics Technologies Inc.

accellera SYSTEMS INITIATIVE

3rd March 2020

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- New challenges posed by new opportunities
- Goals of Testing
  - Differences between RISC-V Compliance and Design Verification
- Verification of RISC-V
  - Compliance Testing
  - Directed Testing
  - Constrained Random Testing (Instruction Stream Generation)
- Components of a simulation based verification flow
  - Instruction stream generators
  - Reference implementations
  - Use of Cloud resources
- Key Issue – Reference Comparison (step/compare verification)
- Case Study / Results

accellera SYSTEMS INITIATIVE

RISC-V presents new challenges

- RISC-V is a new ISA – an open standard ISA
  - Managed by the non-profit RISC-V Foundation (riscv.org)
  - This means any designer can build a processor implementation
    - (Feb 2020 – there are almost 100 RTL designs including open source and proprietary)
- Traditionally
  - processor IP comes from, and is maintained ISA owner
  - is single sourced
  - comes fully verified and compliant to that specific ISA
  - all users need to do is to verify using integration tests
  - there is no ‘standard’ approach and there are few available tools for processor verification
- The RISC-V industry / eco-system needs to adopt its best practises for hardware verification and adapt them to processor verification

Goals of Testing

- Need to be clear what focus of testing is
  - Architecture
    - ISA Definition
  - Micro-Architecture
    - In-Order, Out-Of-Order, Simple-Scalar, Super-Scalar, Transactional Memory, Branch Predictors, …
- Both of these are very different
  - One is about ISA specification
  - Other is about details of a specific implementation
  - This is the difference between ‘Compliance’ and Design Verification

- In the RISC-V Foundation, ‘Compliance’ testing is checking the device works within the envelope of the agreed specification
  - i.e. “have you read and understood the specification”
  - Compliance testing is not a full hardware verification…

Agenda

- New challenges posed by new opportunities
- Goals of Testing
  - Differences between RISC-V Compliance and Design Verification
- Verification of RISC-V
  - Compliance Testing
  - Directed Testing
  - Constrained Random Testing (Instruction Stream Generation)
- Components of a simulation based verification flow
  - Instruction stream generators
  - Reference implementations
  - Use of Cloud resources
- Key Issue – Reference Comparison (step/compare verification)
- Case Study / Results

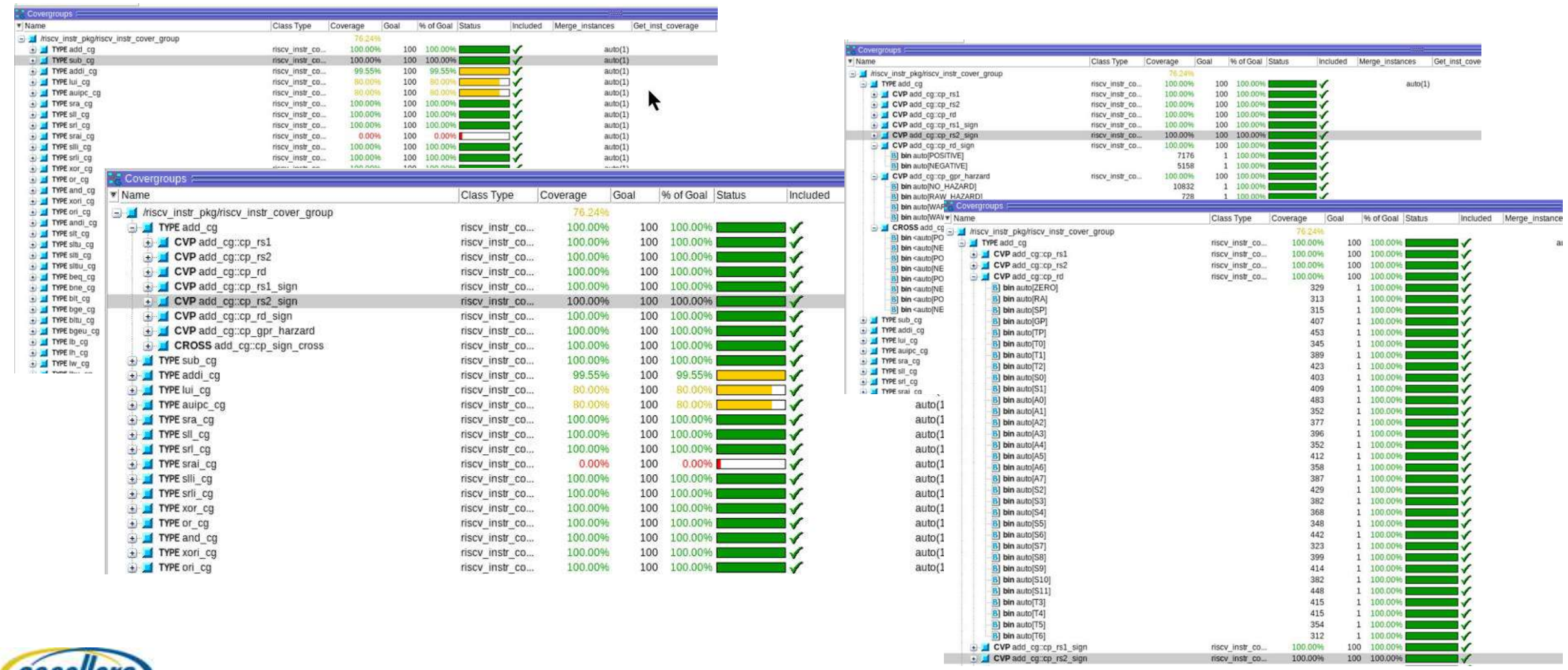
Compliance Testing

- The device works within the envelope of the agreed specifications
  - Have you read and understood the specification
- Testing of the instructions should
  - Attempt to use all registers as source and destination (not combinations)
  - Attempt to operate on all bits which compose the immediate values (1 / 0)
  - Capture a signature in memory region indicating the test result
    - Based upon a particular hardware configuration
  - Compare the signature against a known good reference
    - Static (pre defined signature extraction)
    - Dynamic (runtime generation from YAML configured reference)

Compliance Testing (2)

- Testing of the instructions should NOT
  - Attempt to stress all possible aspects of functional verification, eg
    - All possible combinations of instruction parameters (2-in, 1-out = 32,768)
    - All possible data values
  - Attempt to expose possible micro-architectural aspects
  - Attempt to exercise behaviour which generates an exception
    - Illegal instructions (unsupported extensions)
      - (*) Do not test for missing M instructions in context of RV32I
    - Illegal conditions (misaligned fetch, load, store)

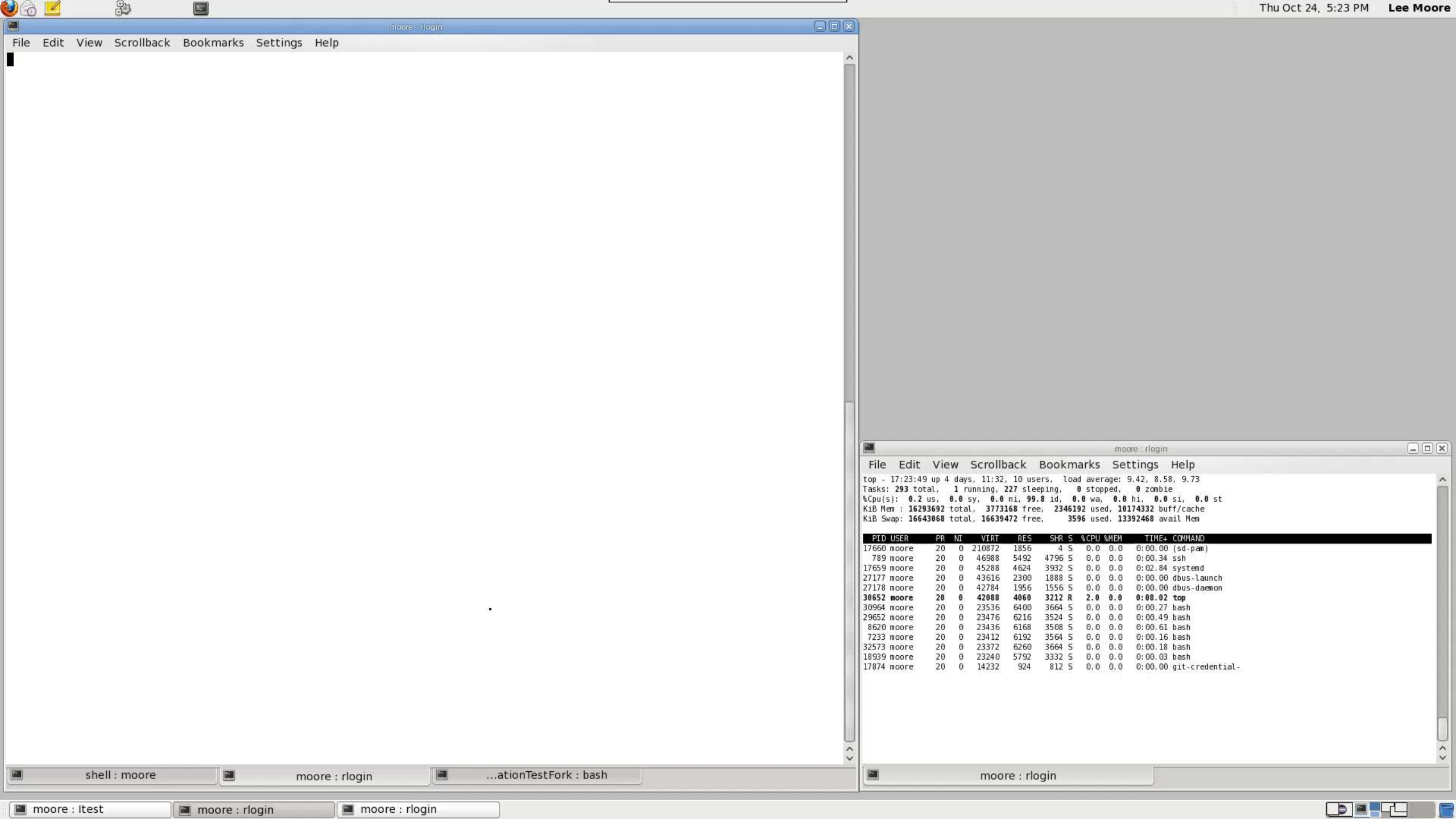
Compliance Testing – Test Qualification
Function Coverage



Coverage images from Mentor Questa SystemVerilog UVM Simulator

Compliance Testing (3)

- Test Qualification
  - Functional Coverage analysis
  - Mutation Fault Simulation - Testing analysis (Imperas work in progress)
    - Provides Decode Coverage
      - Sees if observe changes on all bits of legal decodes


Thu Oct 24, 5:23 PM
Lee Moore
moore : rlogin
File Edit View Scrollback Bookmarks Settings Help
shell : moore
moore : rlogin
...ationTestFork : bash
moore : rlogin
File Edit View Scrollback Bookmarks Settings Help
top - 17:23:49 up 4 days, 11:32, 10 users, load average: 9.42, 8.58, 9.73
Tasks: 293 total, 1 running, 227 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.2 us, 0.0 sy, 0.0 ni, 99.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 16293692 total, 3773168 free, 2346192 used, 10174332 buff/cache
KiB Swap: 16643068 total, 16639472 free, 3596 used. 13392468 avail Mem
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
17660 moore 20 0 210872 1856 4 S 0.0 0.0 0:00.00 (sd-pam)
789 moore 20 0 46988 5492 4796 S 0.0 0.0 0:00.34 ssh
17659 moore 20 0 45288 4624 3932 S 0.0 0.0 0:02.84 systemd
27177 moore 20 0 43616 2300 1888 S 0.0 0.0 0:00.00 dbus-launch
27178 moore 20 0 42784 1956 1556 S 0.0 0.0 0:00.00 dbus-daemon
30652 moore 20 0 42088 4060 3212 R 2.0 0.0 0:08.02 top
30964 moore 20 0 23536 6400 3664 S 0.0 0.0 0:00.27 bash
29652 moore 20 0 23476 6216 3524 S 0.0 0.0 0:00.49 bash
8620 moore 20 0 23436 6168 3508 S 0.0 0.0 0:00.61 bash
7233 moore 20 0 23412 6192 3564 S 0.0 0.0 0:00.16 bash
32573 moore 20 0 23372 6260 3664 S 0.0 0.0 0:00.18 bash
18939 moore 20 0 23240 5792 3332 S 0.0 0.0 0:00.03 bash
17874 moore 20 0 14232 924 812 S 0.0 0.0 0:00.00 git-credential-
moore : rlogin
moore : ltest
moore : rlogin
moore : rlogin

Compliance Testing (4)

- Test Qualification
  - Functional Coverage analysis
  - Mutation Fault Simulation - Testing analysis (Imperas work in progress)
    - Provides Decode Coverage
      - Sees if observe changes on all bits of legal decodes
    - Verified against RV32I test suite
      - 48 hand coded directed tests (average 150 instructions each)
      - https://github.com/riscv/riscv-compliance/tree/master/riscv-test-suite/rv32i/src
    - Decode Coverage data from the Imperas tool
      - ran 478,390 simulations in 308 secs

Compliance Testing (5)

- Compliance RV32I Base Instruction Testing
  - November/12/2019 – 48 tests
- Compliance RV64V Vector instruction Testing (Imperas work in progress)
  - February/2020 – ~6,000 tests

- RISCV-V compliance suites are still a work in progress

Agenda

- New challenges posed by new opportunities
- Goals of Testing
  - Differences between RISC-V Compliance and Design Verification
- Verification of RISC-V
  - Compliance Testing
  - Directed Testing
  - Constrained Random Testing (Instruction Stream Generation)
- Components of a simulation based verification flow
  - Instruction stream generators
  - Reference implementations
  - Use of Cloud resources
- Key Issue – Reference Comparison (step/compare verification)
- Case Study / Results

Directed Testing

- Test Encoded Self Checking
- Reference Comparison Checking

Directed Testing – Test Encoded

- Tests are written with expected behaviour encoded
- Tests can introspect the state and (self) diagnose faults

```
// Device Under Test
int a = 4; int b = 5;
int c = a + b;
assert(c == 9); // report error if result is not as expected
```

Directed Testing – Reference Comparison

- Tests are written without predicting the result
- A reference is consulted for the correct value

```
// Device Under Test
int a = 4; int b = 5;
int c = a + b;
// c == ?
```

```
// Reference
int Ra = 4; int Rb = 5;
int Rc = Ra + Rb;
// Rc == 9
```

```
assert(c == Rc) // external (@runtime or post-processed)
```

Agenda

- New challenges posed by new opportunities
- Goals of Testing
  - Differences between RISC-V Compliance and Design Verification
- Verification of RISC-V
  - Compliance Testing
  - Directed Testing
  - Constrained Random Testing (Instruction Stream Generation)
- Components of a simulation based verification flow
  - Instruction stream generators
  - Reference implementations
  - Use of Cloud resources
- Key Issue – Reference Comparison (step/compare verification)
- Case Study / Results

Constrained Random Testing

- Generate random streams of instructions
- Generator given guidance to target specific instruction types and values
  - Many constraints required to get legal instruction sequences
- No predicted results, relies upon reference

Agenda

- New challenges posed by new opportunities
- Goals of Testing
  - Differences between RISC-V Compliance and Design Verification
- Verification of RISC-V
  - Compliance Testing
  - Directed Testing
  - Constrained Random Testing (Instruction Stream Generation)
- Components of a simulation based verification flow
  - Instruction stream generators
  - Reference implementations
  - Use of Cloud resources
- Key Issue – Reference Comparison (step/compare verification)
- Case Study / Results

Previous open source RISC-V processor verification solutions

Verification is one of the key challenges of modern processor development.

riscv-tests

Assembly unit test

A simple test framework focused on sanity testing the basic functionality of each RISC-V instruction. It's a very good starting point to find basic implementation issues.

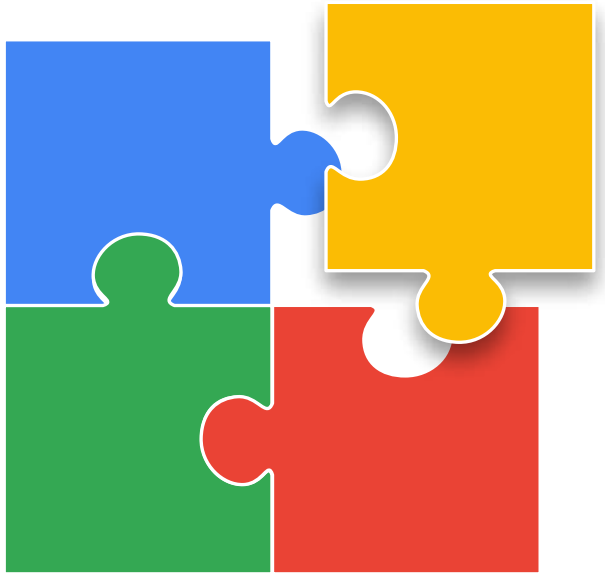
riscv-torture

Scala-based RISC-V assembly generator

Provides a good mix of hand-written sequences. Supports most RISC-V ISA extensions which makes it very attractive. Simple program structure and fixed privileged mode setting.

Many missing pieces

- Complex branch structure
- MMU stress testing
- Exception scenarios
- Compressed instruction support
- Full privileged mode operation verification
- Coverage model
- …



Motivation

Build a high quality open DV infrastructure that can be adopted and enhanced by DV engineers to improve the verification quality of RISC-V processors.

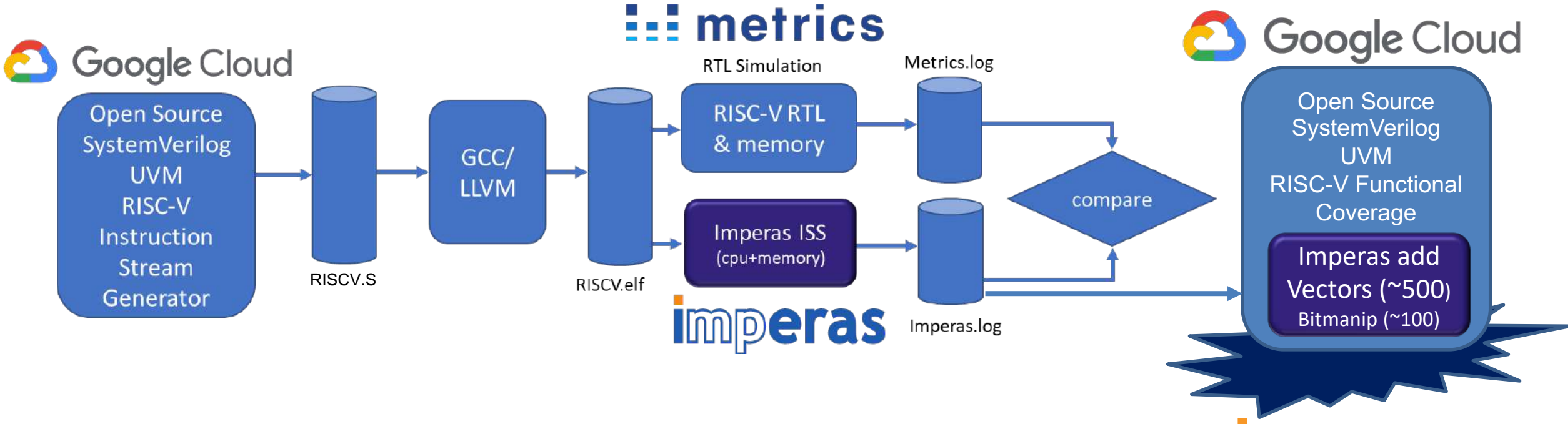
Google RISC-V Instruction Stream Generation

- High quality SystemVerilog UVM DV infrastructure
- Open source (Apache 2.0)
- Drives a RISC-V core through corner cases and pushes it to the limit
- Requires reference and DUT to generate instruction trace disassembly
- Traces compared as post-process (neutral CSV format)
- Can compare values **and** program flow
  - dependant upon target capability
- Provides coverage for test quality, and to aid guidance

Open Source
SystemVerilog
UVM
RISC-V
Instruction
Stream
Generator

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

Constrained Random Testing



- Google: open source riscv-dv instruction stream generator
- Metrics : SystemVerilog design + UVM simulator for RTL
- Imperas: model and simulation golden reference of RISC-V CPU

- Imperas have added Vector and Bitmanip extension instructions to the Functional Coverage

(not yet publicly released)

Agenda

- New challenges posed by new opportunities
- Goals of Testing
  - Differences between RISC-V Compliance and Design Verification
- Verification of RISC-V
  - Compliance Testing
  - Directed Testing
  - Constrained Random Testing (Instruction Stream Generation)
- Components of a simulation based verification flow
  - Instruction stream generators
  - Reference implementations
  - Use of Cloud resources
- Key Issue – Reference Comparison (step/compare verification)
- Case Study / Results

Imperas RISC-V Reference ISS

imperas

RISC-V Reference Model & Simulator

http://www.imperas.com/riscv
https://github.com/riscv/riscv-ovpsim

- Full RISC-V Specification envelope model
- Industrial quality model and simulator of RISC-V processors for use in compliance, verification and test development
- Complete, fully functional, configurable simulator
  - All 32bit and 64bit features of ratified User and Privilege RISC-V specs
    - Vector extension, configurable, versions 0.7.1, 0.8, 0.9 draft
    - Bit Manipulation extension, version 0.91, 0.92. 0.93 draft
  - Model source included under Apache 2.0 open source license
- Used as golden reference in RISC-V Foundations' Compliance Suite and Bit Manipulation group
- Extendibility: easy for user to extend with new instructions and functionality

- In use as reference with customers for RTL DV, for example:
  - "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

ANDES TECHNOLOGY

Agenda

- New challenges posed by new opportunities
- Goals of Testing
  - Differences between RISC-V Compliance and Design Verification
- Verification of RISC-V
  - Compliance Testing
  - Directed Testing
  - Constrained Random Testing (Instruction Stream Generation)
- Components of a simulation based verification flow
  - Instruction stream generators
  - Reference implementations
  - Use of Cloud resources
- Key Issue – Reference Comparison (step/compare verification)
- Case Study / Results

Metrics cloud based solution

- Capacity requirement for simulation are not a constant over a project
  - The additional processor verification requirements only increase this need for peak capacity
  - Cloud resources address this need
- Metrics:
  - Complete SystemVerilog IEEE 1800-2012 compliant simulator including UVM
  - Includes all the standard features of a modern SystemVerilog simulator including debug, APIs, language and testbench support
  - Simulates the testbench, the RTL design, and the populates the coverage models

metrics

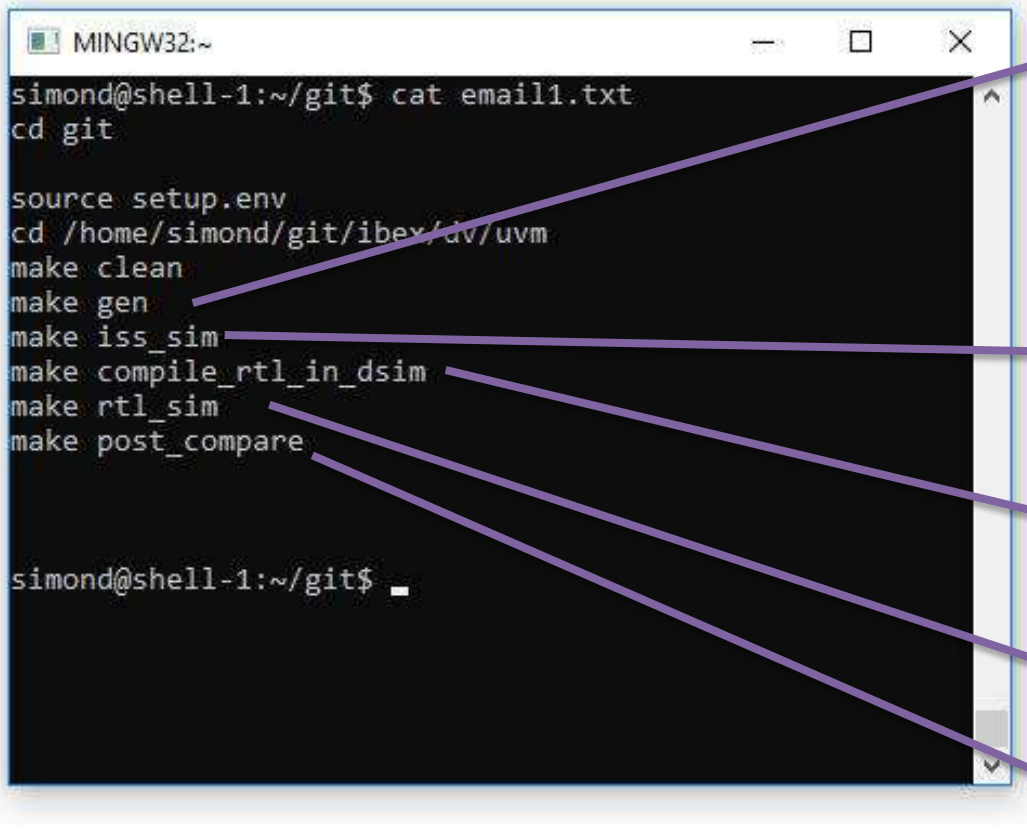
SystemVerilog
UVM
Testbench

RTL
RISC-V CPU

SystemVerilog
UVM
Coverage

https://metrics.ca/

ISG DV Flow is controlled by Makefile and bash scripts and includes python scripts

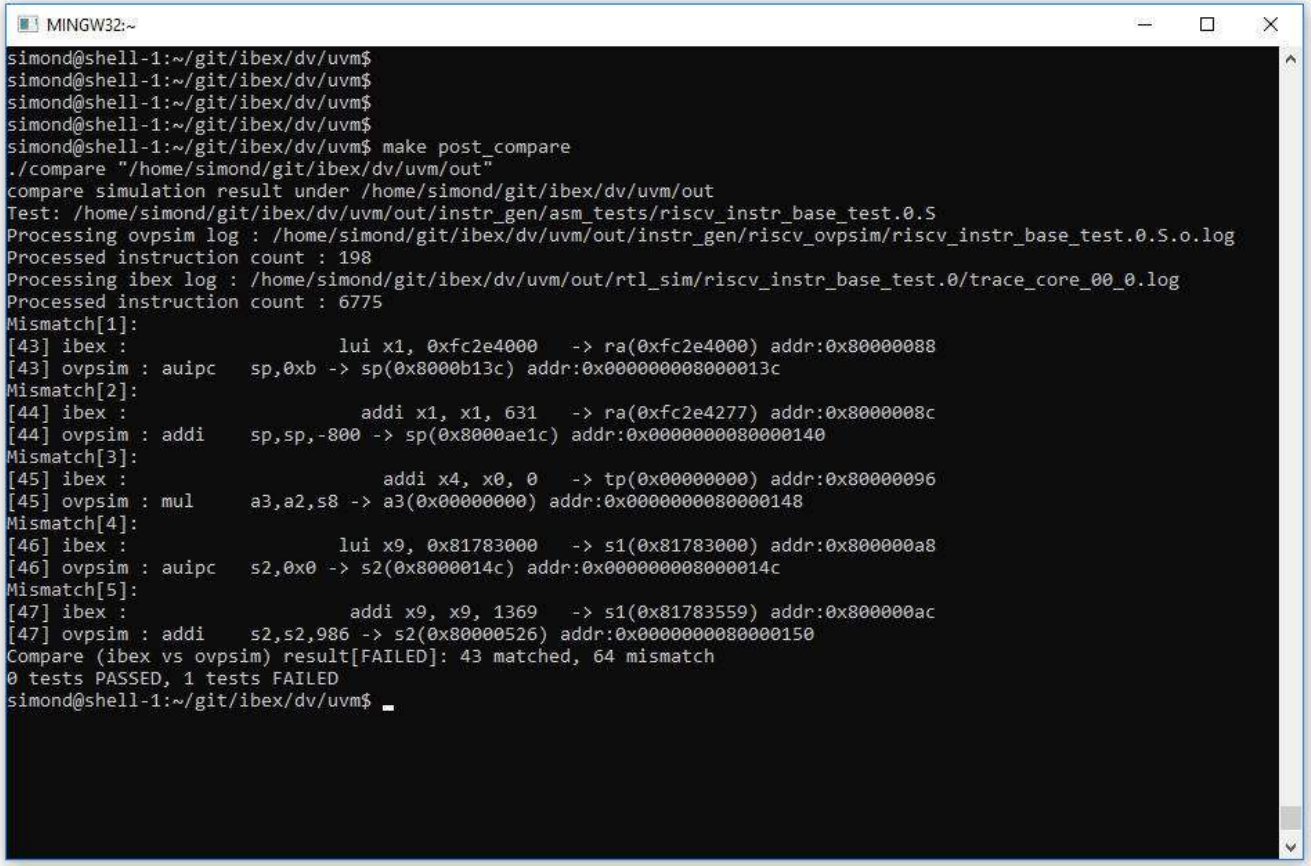


- Compile up SystemVerilog UVM test generator and run it
  - can easily set how many tests to create each run
  - Creates .S files that are then converted to .o
- Run the Imperas ISS to generate reference results
- Compile the SystemVerilog RTL of ibex core and testbench
- Run RTL simulation & record RTL results
- Post-processor run logs and compare

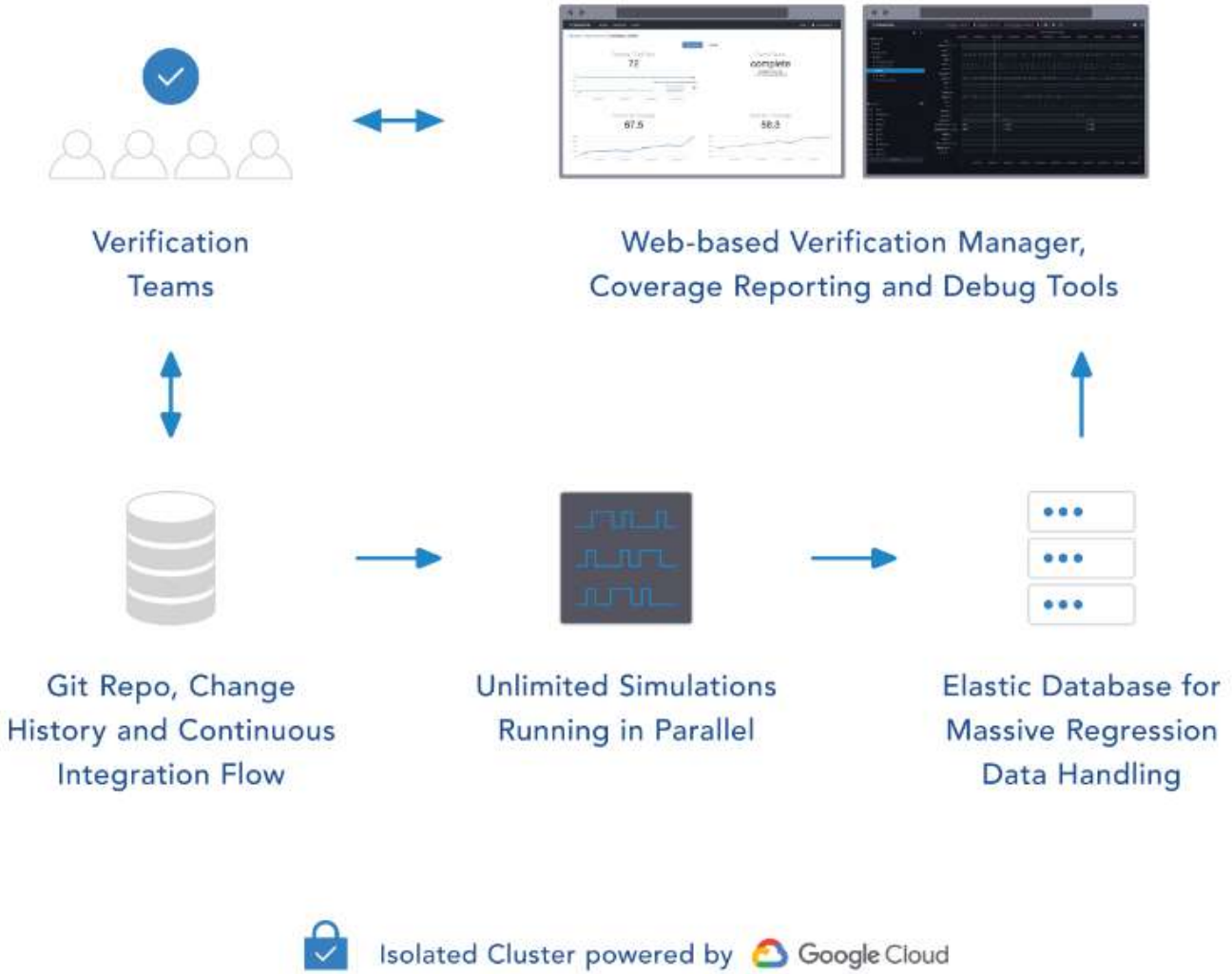
- With Metrics – you get ssh access to shell as if PC was on your desk

And results are simple pass, or detailed fail

- Example of detailed fail:
  - Shows mis-matching instructions
    - Configured here to show 5
  - Full traces etc are kept for review
  - Can dump full VCD for detailed waveform analysis

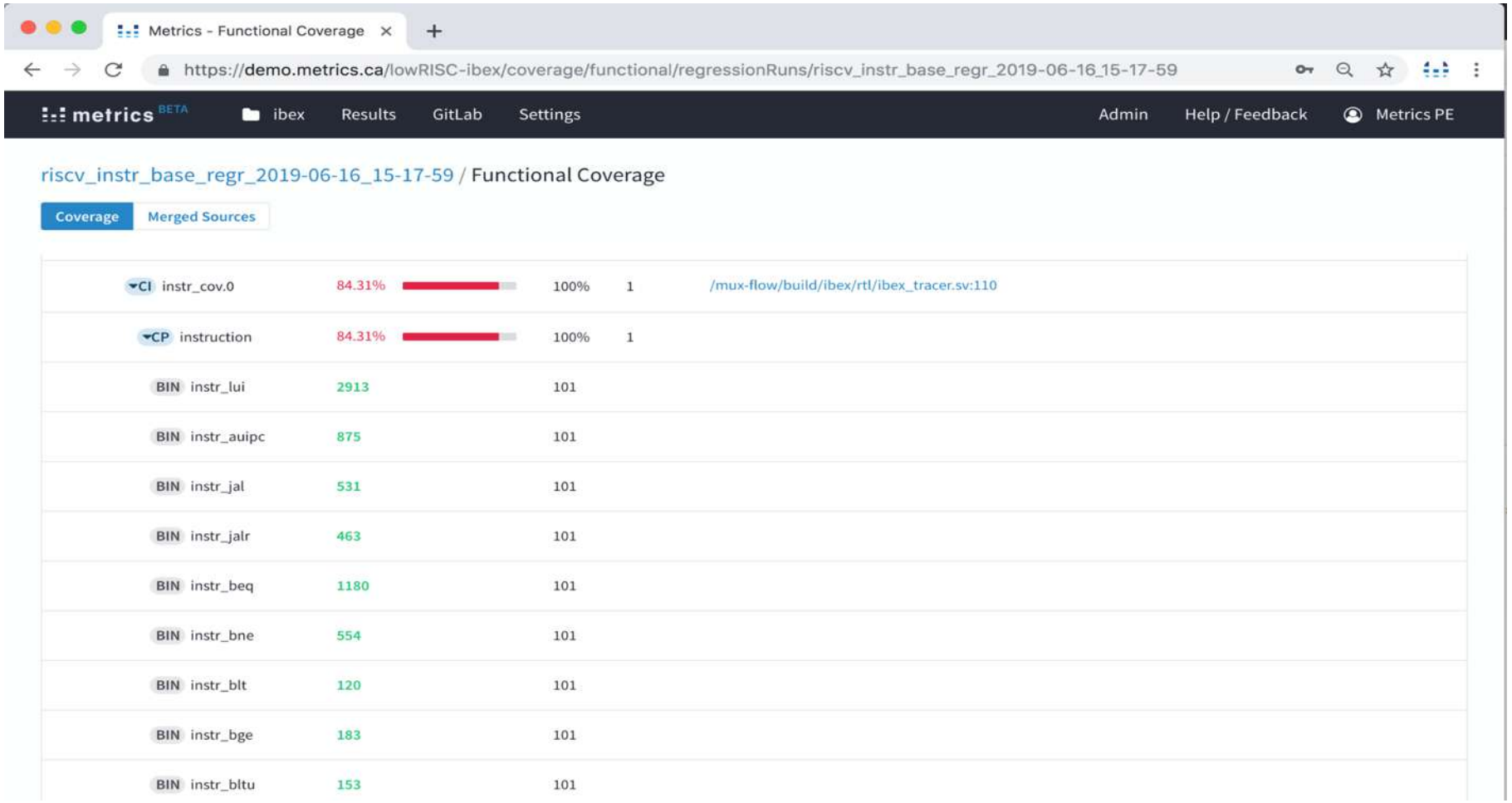


Metrics Cloud Platform makes it all much simpler…



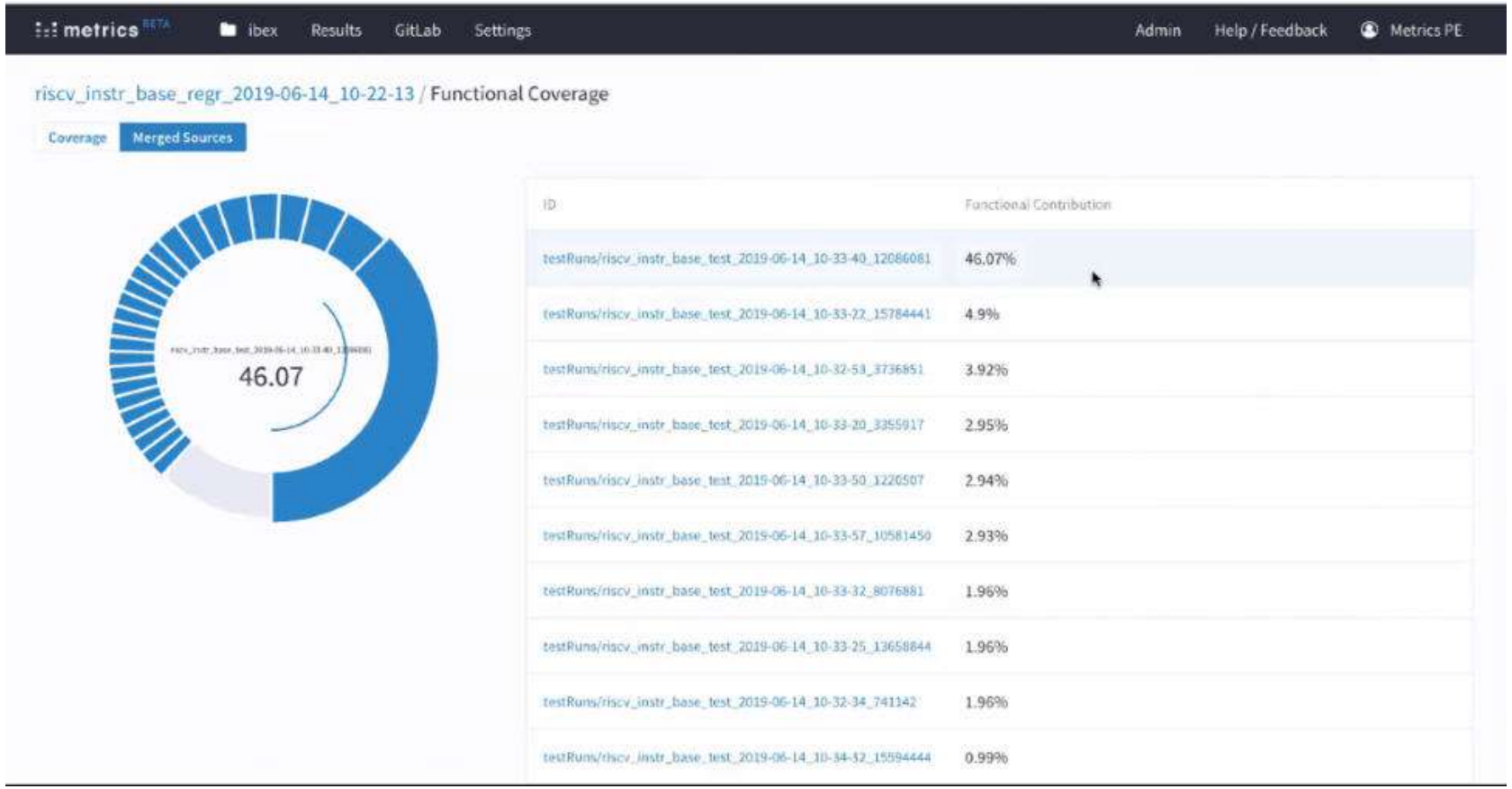
- Complete solution for DV

Metrics: can show functional coverage

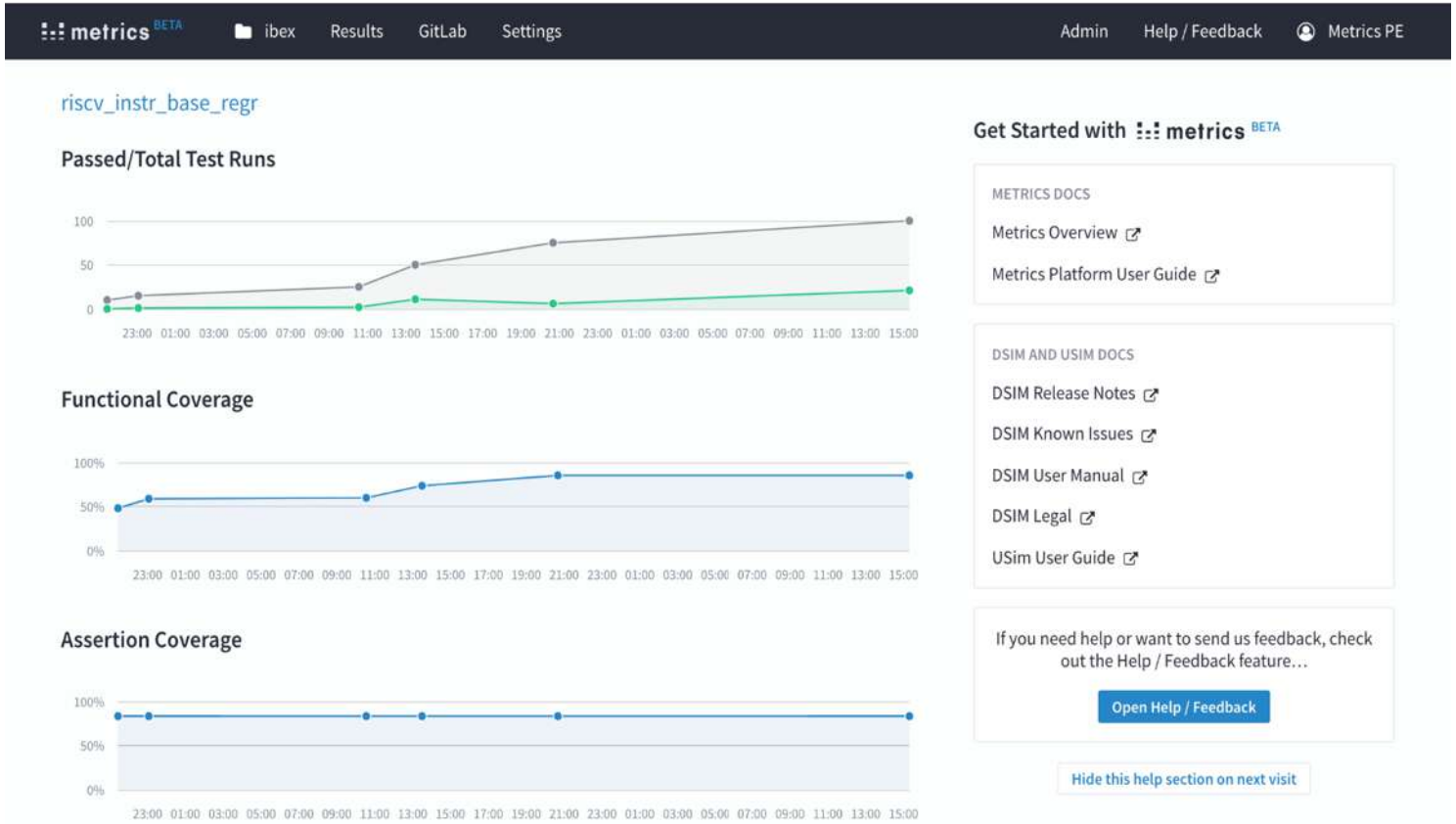


- Uses SystemVerilog covergroups etc.

Metrics: can even see detailed contribution of each test including functional coverage



Metrics: includes top level overview dashboard



- Allows management overview of status of verification

Agenda

- New challenges posed by new opportunities
- Goals of Testing
  - Differences between RISC-V Compliance and Design Verification
- Verification of RISC-V
  - Compliance Testing
  - Directed Testing
  - Constrained Random Testing (Instruction Stream Generation)
- Components of a simulation based verification flow
  - Instruction stream generators
  - Reference implementations
  - Use of Cloud resources
- Key Issue – Reference Comparison (step/compare verification)
- Case Study / Results

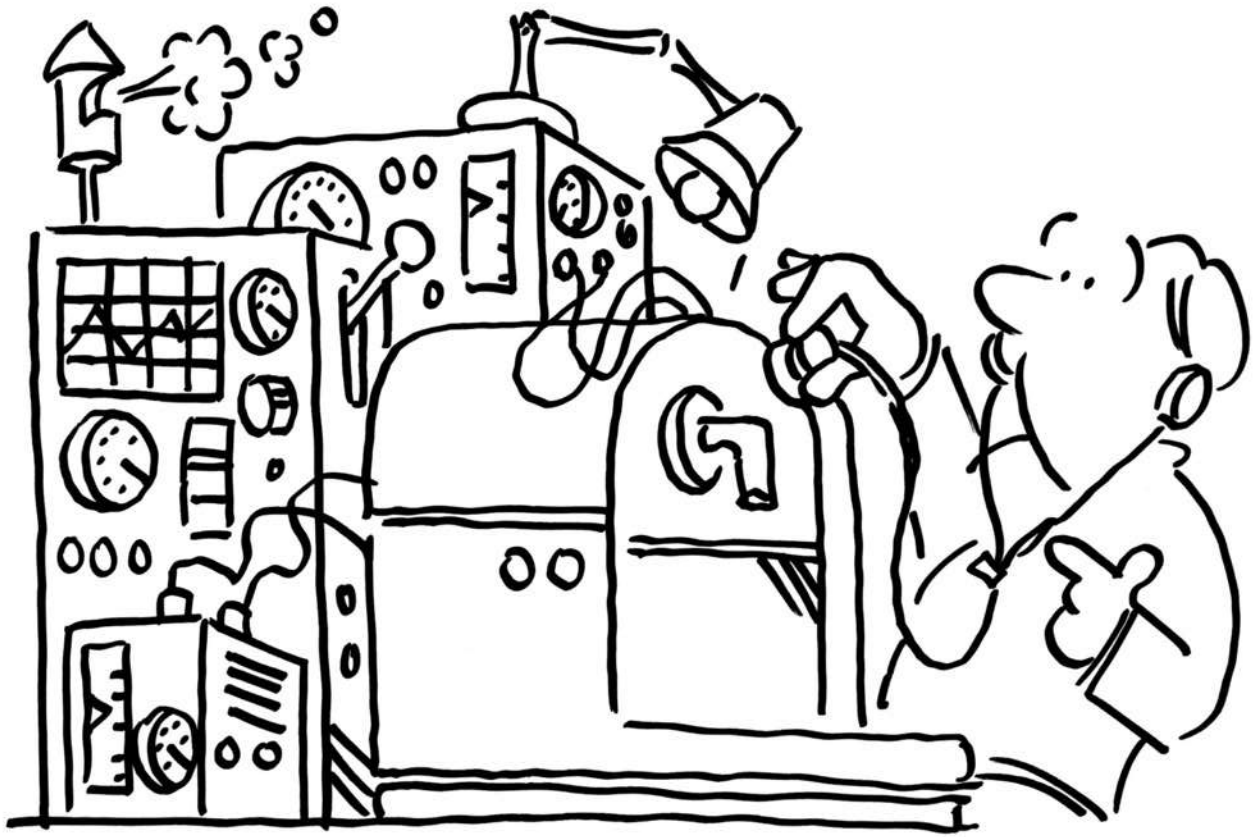
Key Issue – Reference Comparison

- One thing compliance, directed, random have in common...
  - Is a need for a reference implementation to compare with
- So why do I need a reference as part of my verification ?
  - Comparison for the observed behavior
  - Covering all possible aspects of the ISA envelope
- And – it needs to represent exact your design and architecture:
  - XLEN
  - Vectors: VLEN, SLEN, ELEN, (version: 0.7.1, 0,8, 0.9 Draft, …)
  - Bit Manipulation (version: 0.9, 0.91, 0.92, …)
  - Custom Extensions
  - M+U (No S)
  - Hardware LSU Misalignment Support (no exception)
  - CSR: MTVEC ReadOnly
  - …

RISC-V Reference choices

- RISC-V is highly configurable
- So it can get a little …. complicated

- 60… Questions ?



Imperas RISC-V Reference ISS

imperas

RISC-V Reference Model & Simulator

http://www.imperas.com/riscv
https://github.com/riscv/riscv-ovpsim

- Full RISC-V Specification envelope model
- Industrial quality model and simulator of RISC-V processors for use in compliance, verification and test development
- Complete, fully functional, configurable simulator
  - All 32bit and 64bit features of ratified User and Privilege RISC-V specs
    - Vector extension, configurable, versions 0.7.1, 0.8, 0.9 draft
    - Bit Manipulation extension, version 0.91, 0.92. 0.93 draft
  - Model source included under Apache 2.0 open source license
- Used as golden reference in RISC-V Foundations' Compliance Suite and Bit Manipulation group
- Extendibility: easy for user to extend with new instructions and functionality
- In use as reference with customers for RTL DV, for example:
  - "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

ANDES TECHNOLOGY

Comparison Modes

- Post-process of data between DUT and Reference
- DUT and Reference Encapsulation

Comparison Modes
Post-process of data

- Usually the easiest method to implement (dependent on tracing formats)
  - Capture of program flow (monitor the PC)
  - Capture of program data (monitor the Registers, Memory)
- Potentially very large data files
- Potential for wasteful execution (early failure)

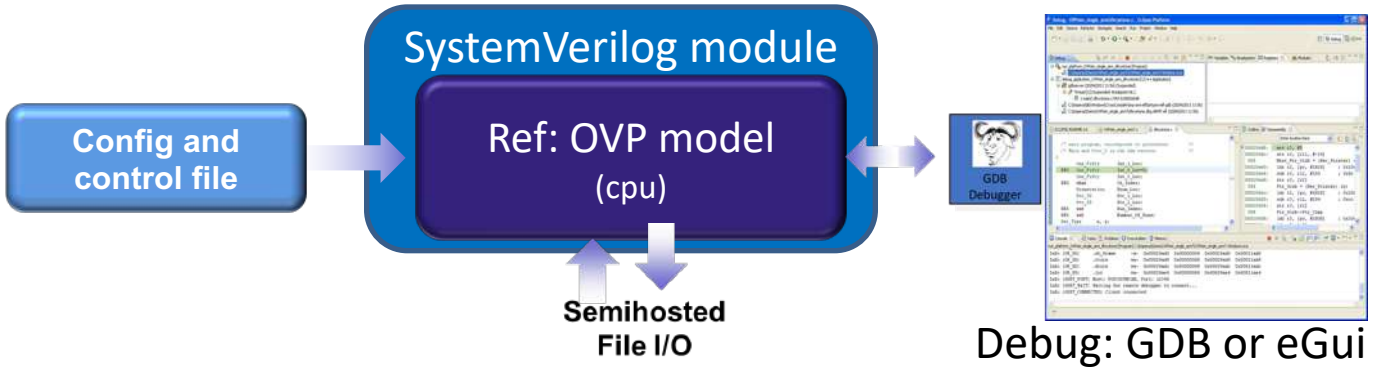
Comparison Modes
Reference Encapsulation

- Instruction by instruction lockstep comparison
  - Comparison of execution flow
  - Comparison of program data
- Immediate comparison
  - Allows for debug introspection at point of failure – very powerful
  - Does not waste execution cycles after failure

Reference Encapsulation

- Imperas OVP simulators can act as a simulation Master
- Imperas OVP simulators can act as a simulation Slave
  - Encapsulated into SystemC/TLM
  - Encapsulated into SystemVerilog via DPI (Direct Procedural Interface)

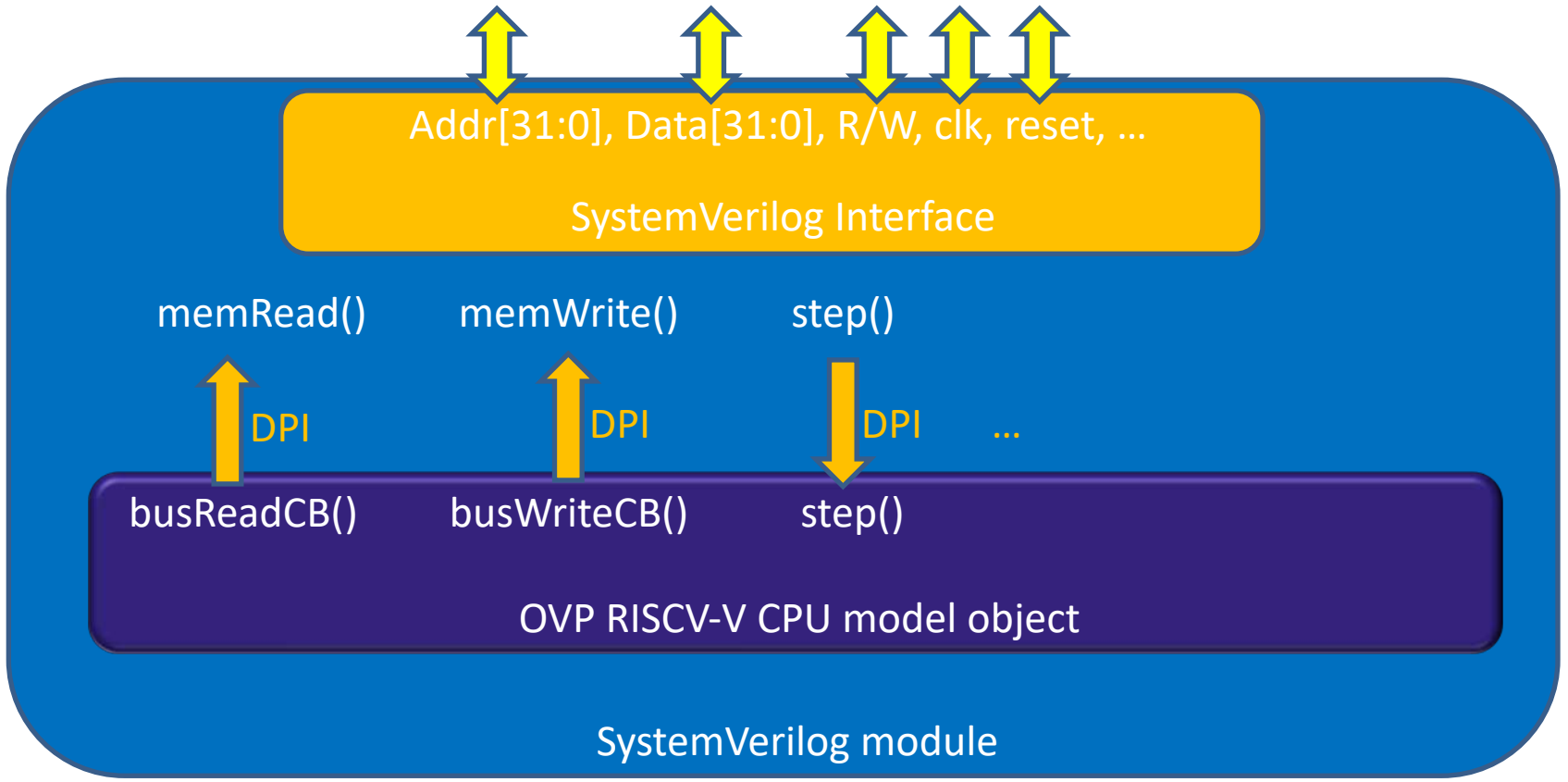
Imperas OVP model in SystemVerilog



Debug: GDB or eGui Eclipse

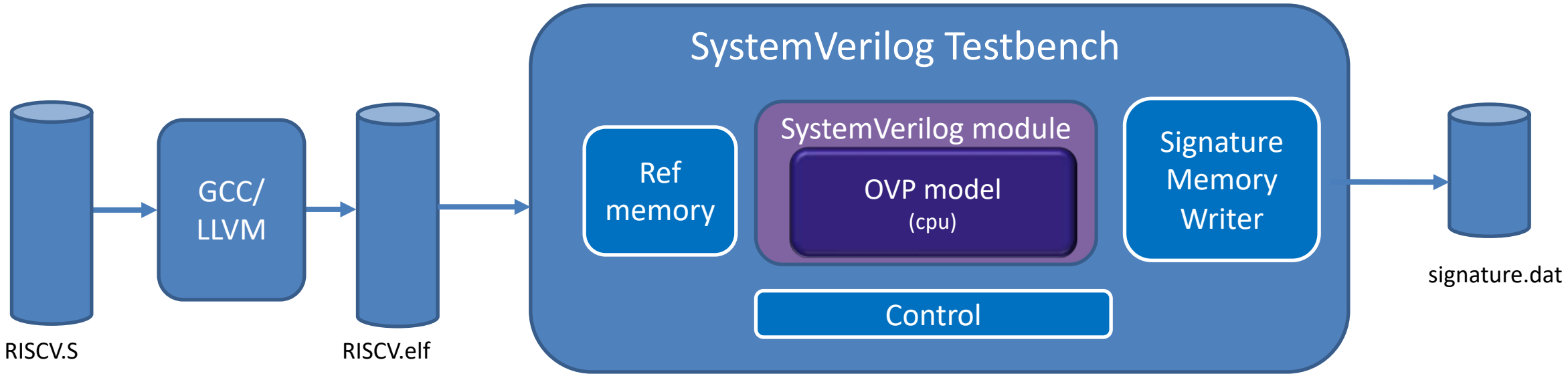
- OVP model is encapsulated into a SystemVerilog module
- Like riscvOVPsim ISS it is an envelope model of RISC-V Foundation's stand ISA and ISA extensions (RV32/64 IMAFDC + B + V)
  - Includes variants for all standard configurations
- Single processor, external everything…
  - Memory etc all in SystemVerilog
- Interfaces being: reset, step, address bus, data bus, interrupts, etc.,…
- Like riscvOVPsim it has full trace and logging capabilities
- Does work with a side-port for GDB or Eclipse eGui debug or Imperas Multi Processor debugger (eGui MPD)

OVP model (encapsulation)



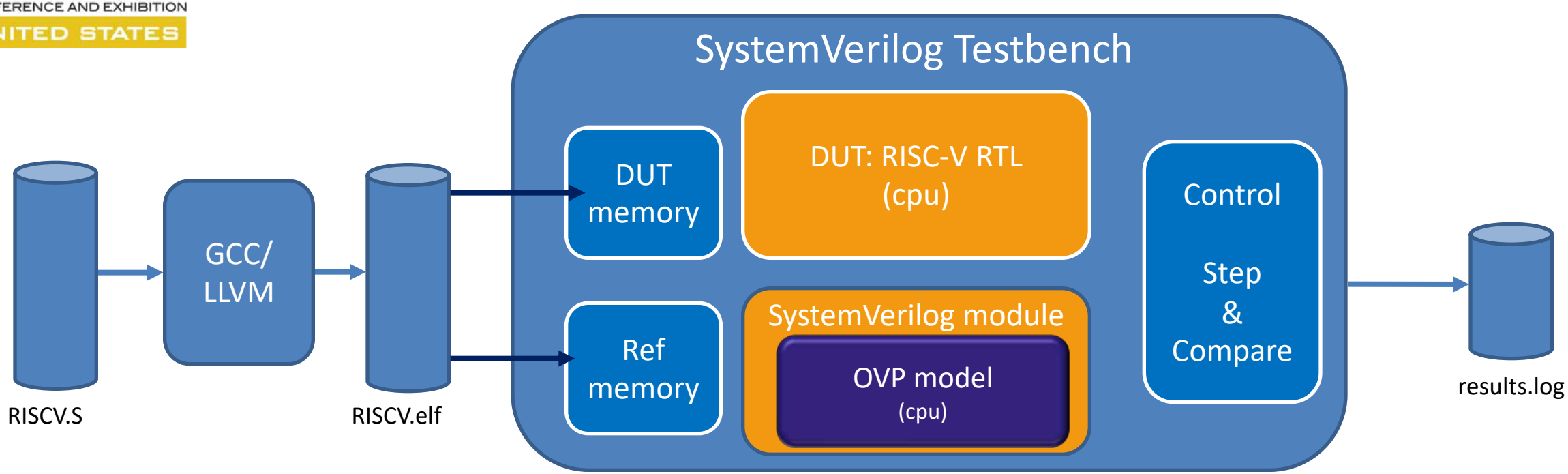
- OVP model is a binary shared object of a single core RISC-V CPU model
- Encapsulated into a SystemVerilog module, using SystemVerilog DPI
- Instanced in SystemVerilog design or testbench like any module

Encapsulated OVP model Running Compliance Suite



- OVP model is encapsulated into SystemVerilog module as target in Compliance Framework
  - Loads .elf file and runs compliance test program – for each test in the compliance suite – generating signature
- RISCV-V Compliance Suite framework controls target and collates signatures and compares with golden reference
- Shows how easily SystemVerilog RTL can be used as target for compliance testing
- User creates similar testbench for user CPU

OVP model - Step and Compare



- OVP model is encapsulated into SystemVerilog module
- Interfaces being: reset, clk, address bus, data bus, interrupts, registers, etc.,…
- Testbench loads .elf program into both memories, resets CPUs (RTL and OVP model)
- Steps CPUs, extracting data, and comparing
  - There is no stored log file – test log data is dynamic and requires two targets to be run and compared

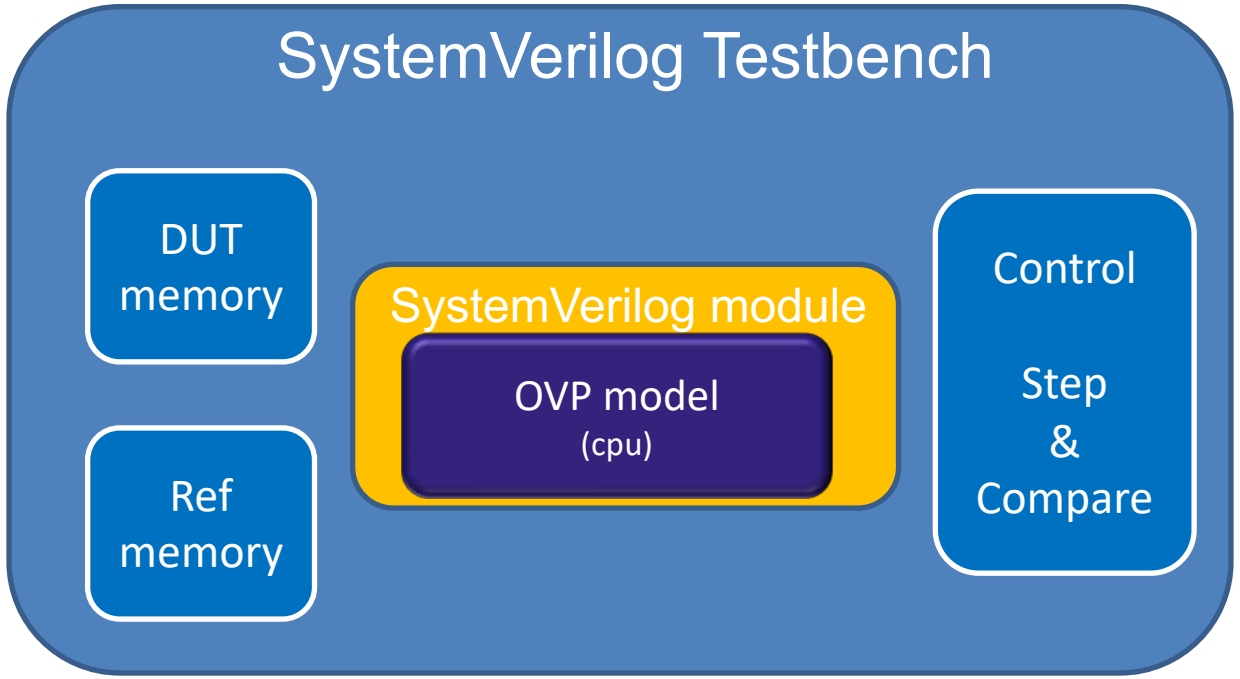
Reference Comparison

- Instruction Retire / PC Compare
  - Compare the program flow during execution
  - Dependent upon the data causing a program flow divergence (branch, jump, exception)
  - Does not detect data flow differences
  - Least invasive regarding detailed knowledge or extraction of the RTL values

Reference Comparison (2)

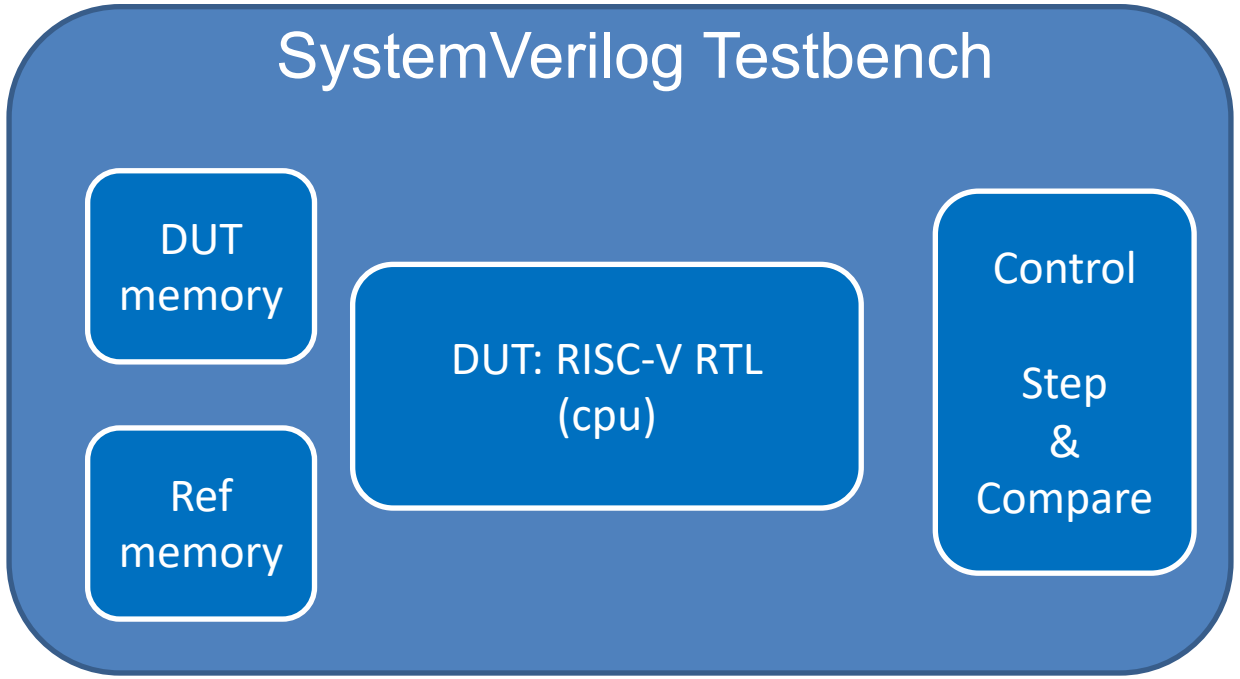
- Instruction Retire / PC, WB, LD, ST Compare
  - Compare the program flow during execution (PC)
  - Compare the registers GPR, FPR, VEC, CSR
  - Immediate detection of divergence due to control and/or data
  - Will require detailed knowledge and extraction of the RTL values

Expert modes of verification – Hot Swapping (RTL)



- Execute a long boot sequence using a Fast Processor model
- e.g., boot Linux to login prompt, and about to run user application

Expert modes of verification – Hot Swapping (RTL) (2)

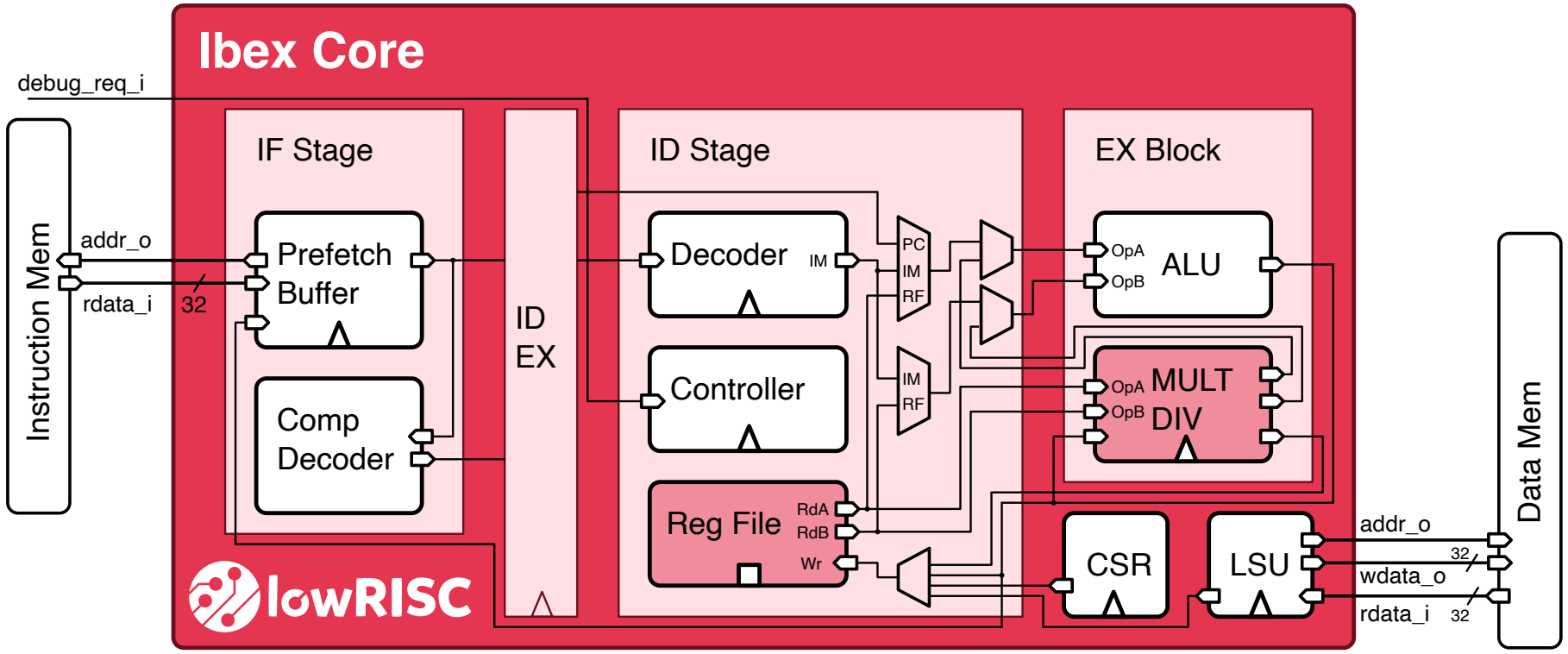


- At the call to system exec() of the user application, hot swap the much slower RTL representation of the core
- Using the OVP API's the entire machine state can be extracted, and applied to the RTL
- (H/W Accelerator)

Agenda

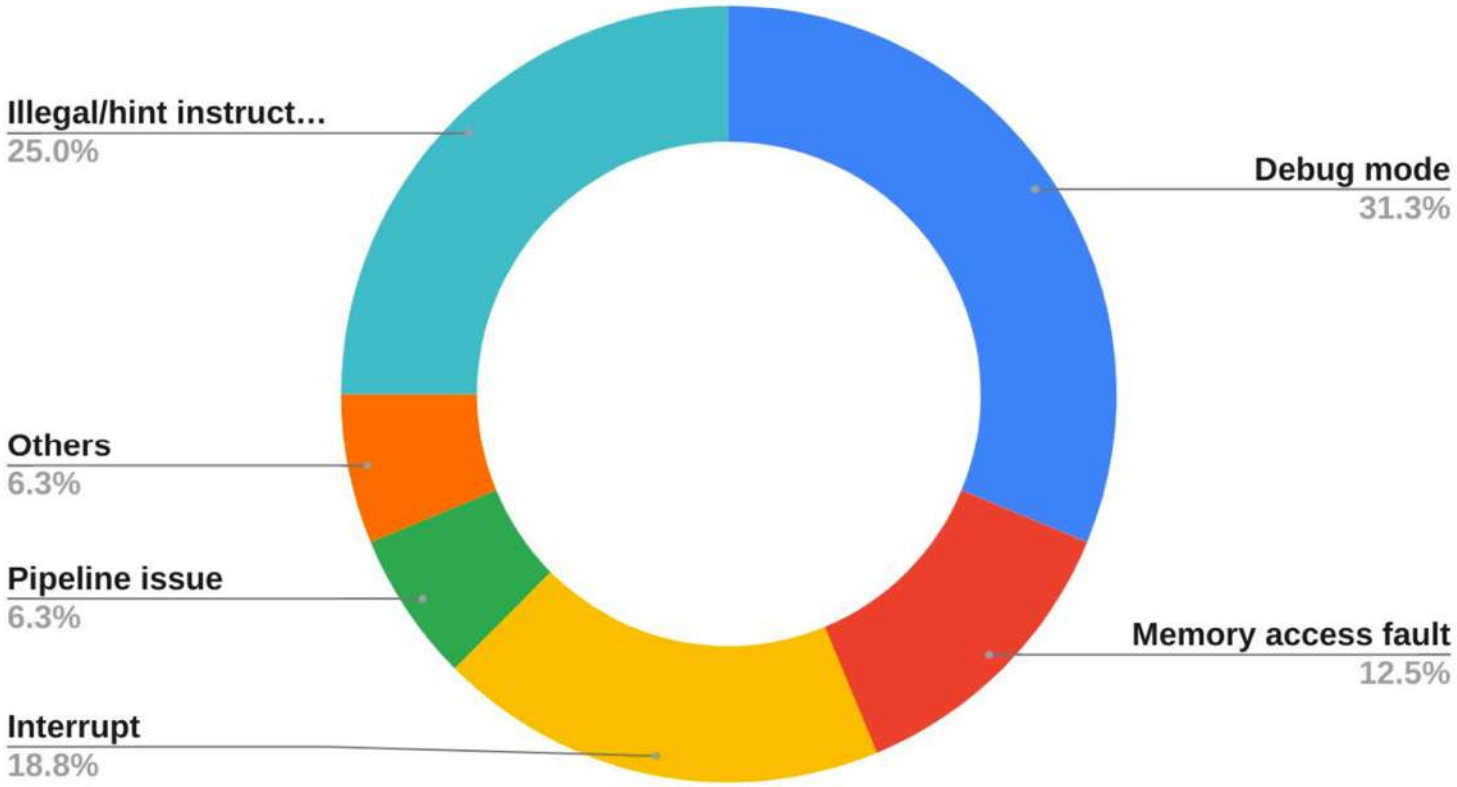
- New challenges posed by new opportunities
- Goals of Testing
  - Differences between RISC-V Compliance and Design Verification
- Verification of RISC-V
  - Compliance Testing
  - Directed Testing
  - Constrained Random Testing (Instruction Stream Generation)
- Components of a simulation based verification flow
  - Instruction stream generators
  - Reference implementations
  - Use of Cloud resources
- Key Issue – Reference Comparison (step/compare verification)
- Case Study / Results

lowRISC Ibex



- Ibex is a small 32 bit RISC-V CPU core (RV32IMC/EMC) with a two stage pipeline, previously known as zero-risky (PULP)
- https://github.com/lowRISC/ibex

Case study: Ibex core verification

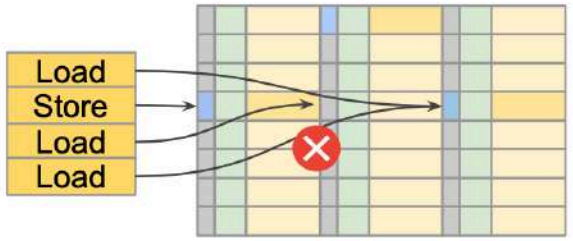


Categories of found bugs

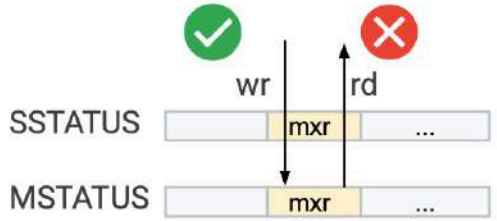
Google Cloud

- Using Random Instruction Stream Generator approach

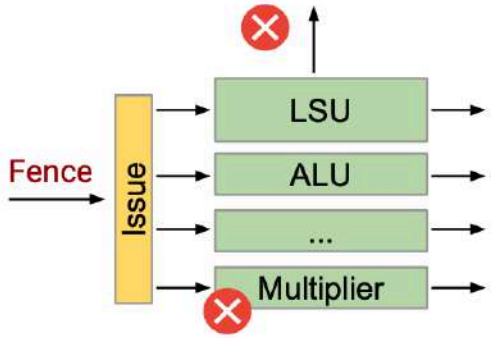
Bugs found



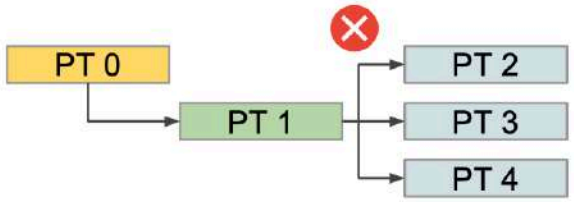
Cache line access racing



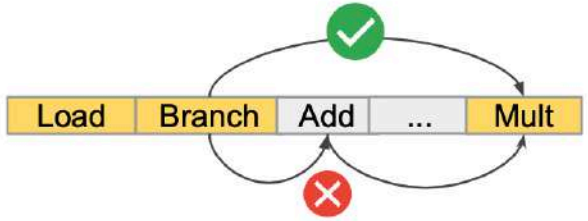
privileged CSR access



FENCE operation failure



page fault handling



Incorrect branch execution

ALU corner case bug

Google Cloud

- Using Random Instruction Stream Generator approach

Agenda

- New challenges posed by new opportunities
- Goals of Testing
  - Differences between RISC-V Compliance and Design Verification
- Verification of RISC-V
  - Compliance Testing
  - Directed Testing
  - Constrained Random Testing (Instruction Stream Generation)
- Components of a simulation based verification flow
  - Instruction stream generators
  - Reference implementations
  - Use of Cloud resources
- Key Issue – Reference Comparison (step/compare verification)
- Case Study / Results
- Conclusions

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Conclusions

- Including a RISC-V processor in your design means much more verification is needed
  - Compliance Testing, Directed Testing, Instruction Stream Generation
- Current ‘gold standard’ approaches such as SystemVerilog UVM, functional coverage and constrained random generators are needed to be adopted
- It is essential to adopt a quality, configurable, proven RISC-V reference
- For efficient verification reference model encapsulation and run-time step/compare is needed
- Solutions are available: e.g. collaboration between Imperas, Google, Metrics

imperas Google Cloud metrics

accellera SYSTEMS INITIATIVE

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Thank You

- Visit https://www.imperas.com/riscv and https://www.ovpworld.org/riscv for more information

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

- https://github.com/lowRISC/ibex

- RISC-V Foundation Compliance Suite, includes riscvOVPsim is available at:
  - https://github.com/riscv/riscv-compliance

Simon Davidmann
Imperas Software Ltd.
info@imperas.com

accellera SYSTEMS INITIATIVE