A Modern Fable:
The Lost Art of Processor Verification

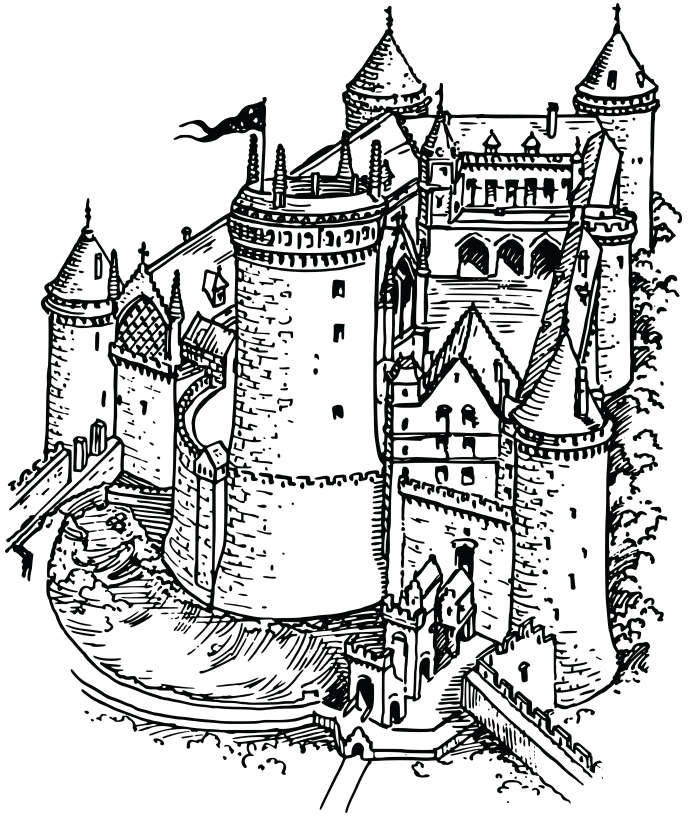
Verification Futures – Austin

Larry Lapides

14 September 2023

RISC-V Is Why We Are All Worried About Processor Verification

- RISC-V is taking over the processor world, except for x86
  - Yes, that includes Arm
- RISC-V processor customization means that every RISC-V developer needs to verify the RISC-V processor
- Lost art? Processor IP vendors guard their verification methodology and details more closely than the IP itself
  - With the verification flow, someone could reverse engineer a high quality processor
  - There are few public details about x86, Arm or Apple processor verification



Agenda

- RISC-V and processor verification
- RISC-V processor models
- RISC-V processor verification methodology
- Processor verification success
- Summary

Agenda

- **RISC-V and processor verification**
- RISC-V processor models
- RISC-V processor verification methodology
- Processor verification success
- Summary

RISC-V Freedom Enables Domain Specific Processing

- **Who**: RISC-V users include traditional semiconductor companies, and embedded systems companies now practicing vertical integration by developing their own SoCs
- **What**: RISC-V is an open instruction set architecture (ISA), it is not a processor implementation
- **Where**: RISC-V is growing in market segments where x86 (PCs, data centers) and Arm (mobile) architectures are not dominant
  - Small microcontrollers for SoC management, replacing proprietary cores
  - Verticals such as IoT and automotive
  - Horizontal markets such as security and AI/ML
  - Deep embedded applications
- **When**: RISC-V processors are now used in over 30% of SoCs
- **Why**: The freedom of the open ISA enables users to develop *differentiated* domain specific processors and processing systems

Keys to RISC-V SoC Success

1) Processor IP source
   - Processor IP vendor
   - Open source IP
   - Build it yourself
2) Processor verification
3) Software porting, development, bring up, test

- All 3 areas need to account for the addition of custom features to the processor (*because everyone adds custom features to the processor*)

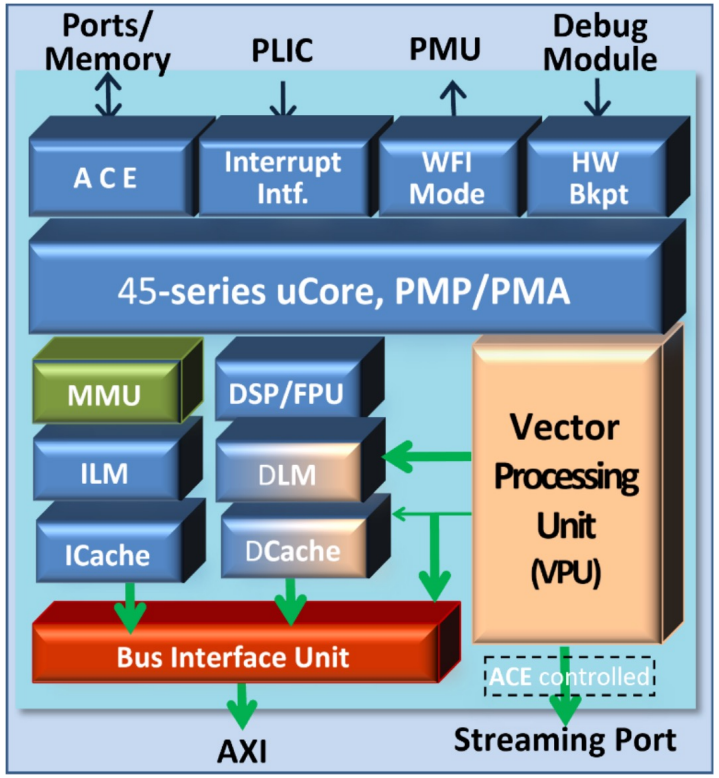
Keys to RISC-V SoC Success

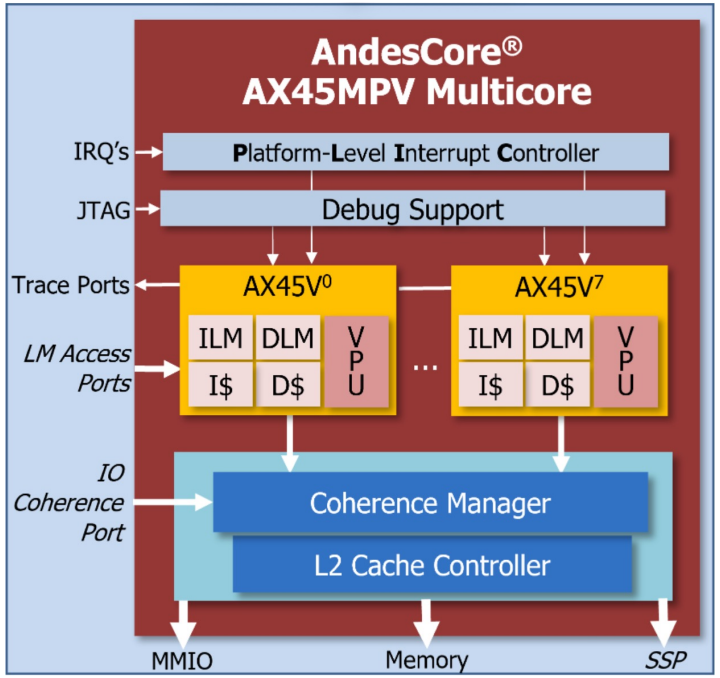
1) Processor IP
   - Processor IP vendor
   - Open source IP
   - Build it yourself
2) **Processor verification**
3) Software porting, development, bring up, test

- Users of all 3 types of processor IP need to account for the addition of custom features to the processor (*because everyone adds custom features to the processor*)
- Success in processor verification requires a high-quality model of the processor
- Success in processor verification requires innovative technologies and methodologies – ***the lost art of processor verification***

RISC-V Processor Complexity

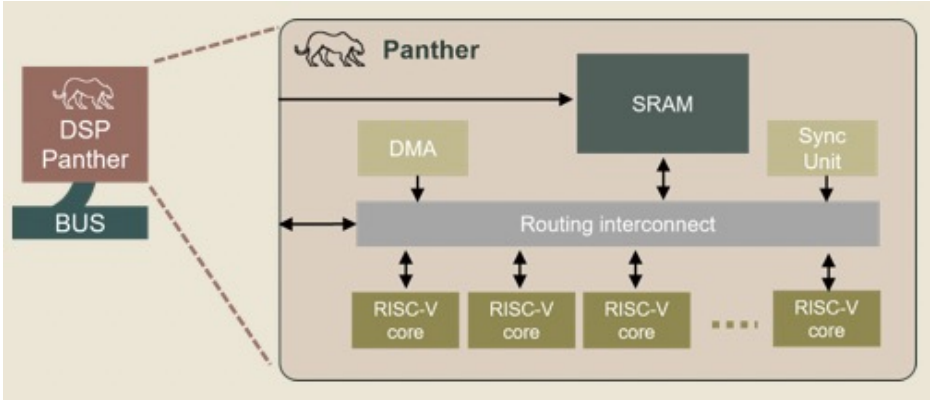
- RISC-V is a modular instruction set architecture
- Any extension (functional group of instructions, e.g. atomics, compressed, floating point, vector) can be added to the base processor
- Then add in interrupts, privilege modes, Debug mode, multi-hart (multi-core), etc. and it gets complex
- Then processor DV, tool chain development and other software development is needed



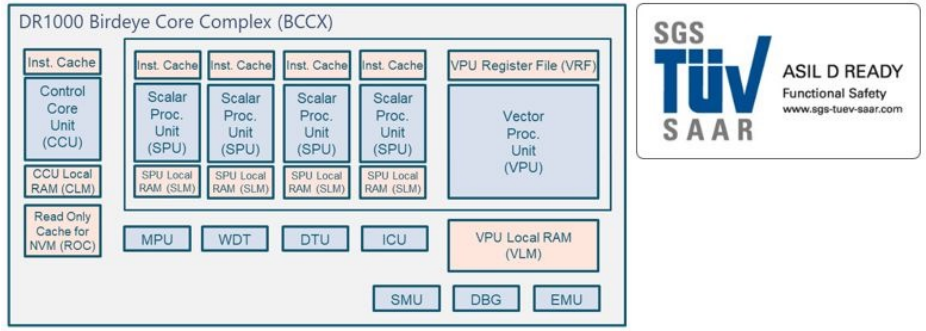


RISC-V Processing Subsystems Compound DV Issues

- Multi-processor subsystems are commonly being developed using RISC-V cores
- Application areas include DSP, AI/ML and packet processing
- This adds complexity to both the DV and software development tasks



Dolphin Design “Panther” DSP



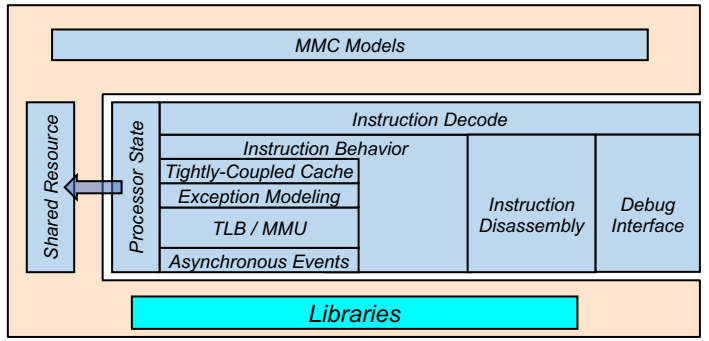
NSITEXE Data Flow Processor

Agenda

- RISC-V and processor verification
- **RISC-V processor models**
- RISC-V processor verification methodology
- Processor verification success
- Summary

RISC-V Model Requirements: Quality, Configurability, Interfaces

- Model the ISA, including all versions of the ratified spec, and stable unratified extensions
- Model other behavioral components, e.g. interrupt controllers
- Easily update and configure the model(s) for the next project
- User-extendable for custom instructions, registers, …
- Model actual processor IP, e.g. Andes, MIPS, NSITEXE, OpenHW, SiFive, …
- Well-defined test process – for the model! – including coverage metrics
- Interface to other simulators, e.g. SystemVerilog, SystemC, Imperas virtual platform simulators
- Interface to software debug tools, e.g. GDB/Eclipse, Imperas MPD
- Interface to software analysis tools including access to processor internal state, etc.

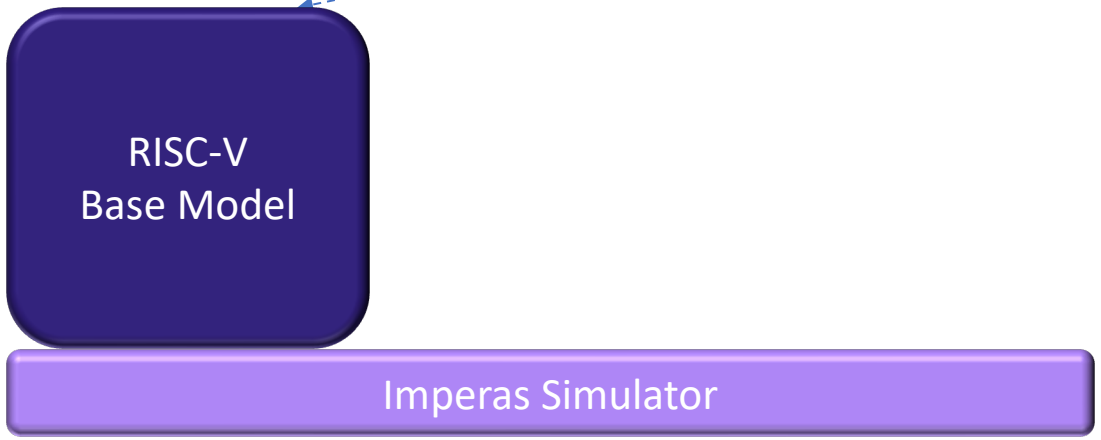


- Imperas models and simulators were built to satisfy these requirements, and matured through usage on non-RISC-V ISAs over the last 15+ years

Imperas OVP RISC-V Fast Processor Models

- Use cases
  - Architecture analysis, including (especially) custom instructions
  - Software development, debug and test
  - Processor and SoC verification
- Existing Imperas Open Virtual Platforms (OVP) Fast Processor Models of …
  - Generic or envelope models of RV32/64 IMAFDCEVBHKPZ* M/S/U privilege modes
  - Models of processor IP vendors: Andes, MIPS, NSITEXE, OpenHW, SiFive, …
  - Models for developers building their own processor
- Custom instructions easily added by user or by Imperas
  - New instructions are added in a side file so as not to perturb the verified model
  - Custom instructions are analyzed for effectiveness
- Models are built using Test Driven Development (TDD) methodology
  - Tests are built at the same time as features are added
  - Continuous Integration (CI) test flow used
  - > 15,000 tests for models + simulator
  - Additional testing by processor IP vendors to validate models

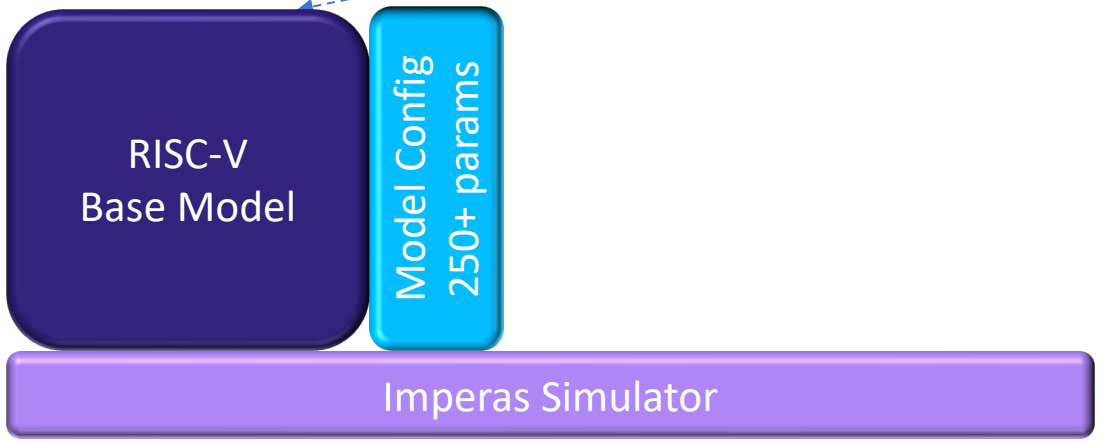
Model + Simulator Architecture (1)



Imperas develops and maintains Base Model

- Base Model implements RISC-V specification in full
- Base Model built using Test Driven Development methodology
- Built using public APIs matured over 15 different ISAs
- Simulator is separate from the model; supports the modeling APIs

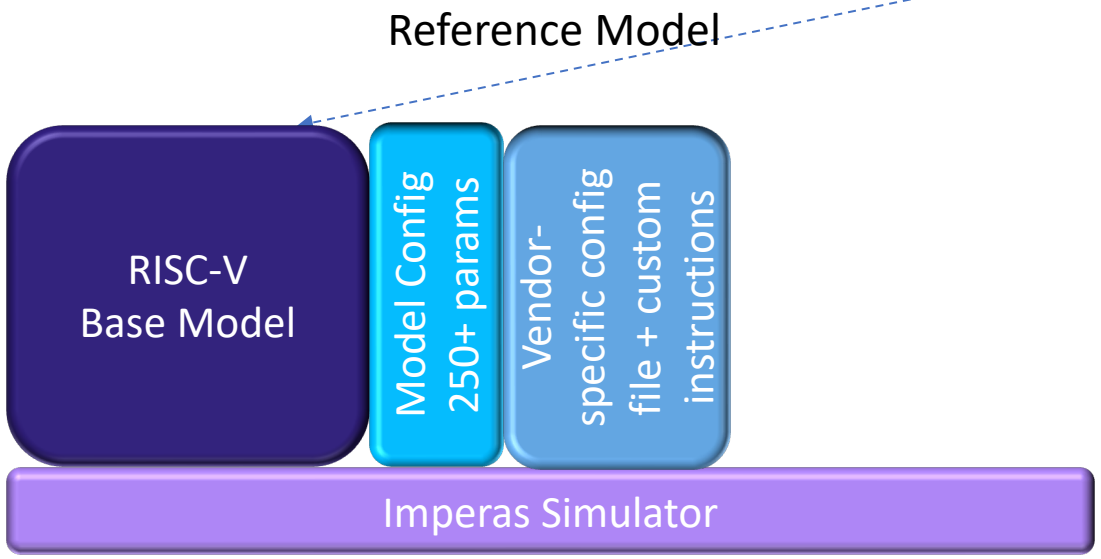
Model + Simulator Architecture (2)



Imperas develops and maintains Base Model

- Base Model implements RISC-V specification in full
- Base Model built using Test Driven Development methodology
- Built using public APIs matured over 15 different ISAs
- Simulator is separate from the model; supports the modeling APIs
- Fully user configurable to select which ISA extensions
- Fully user configurable to select which version of each ISA extension

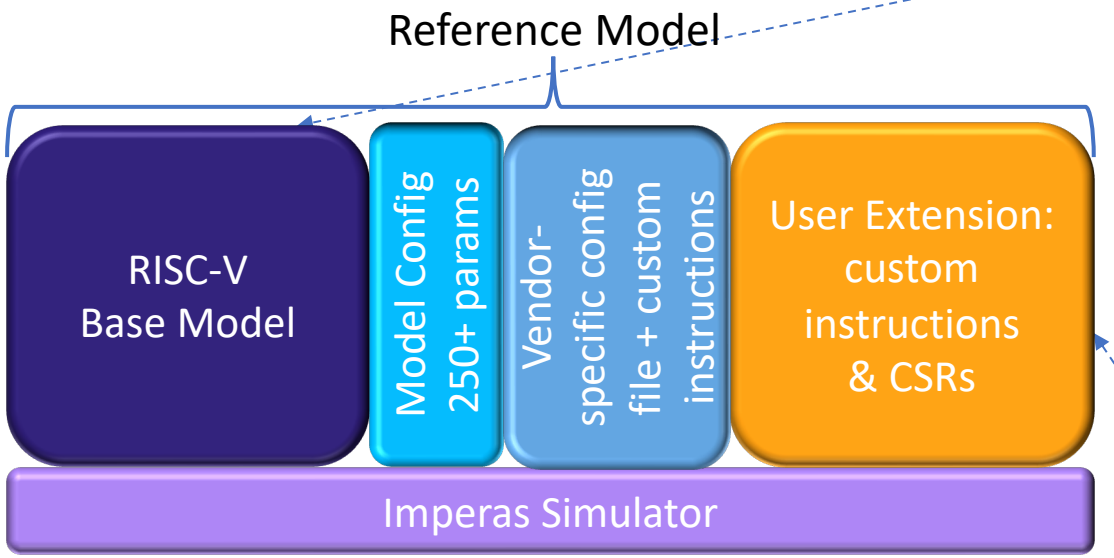
Model + Simulator Architecture (3)



Imperas develops and maintains Base Model

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

Model + Simulator Architecture (4)



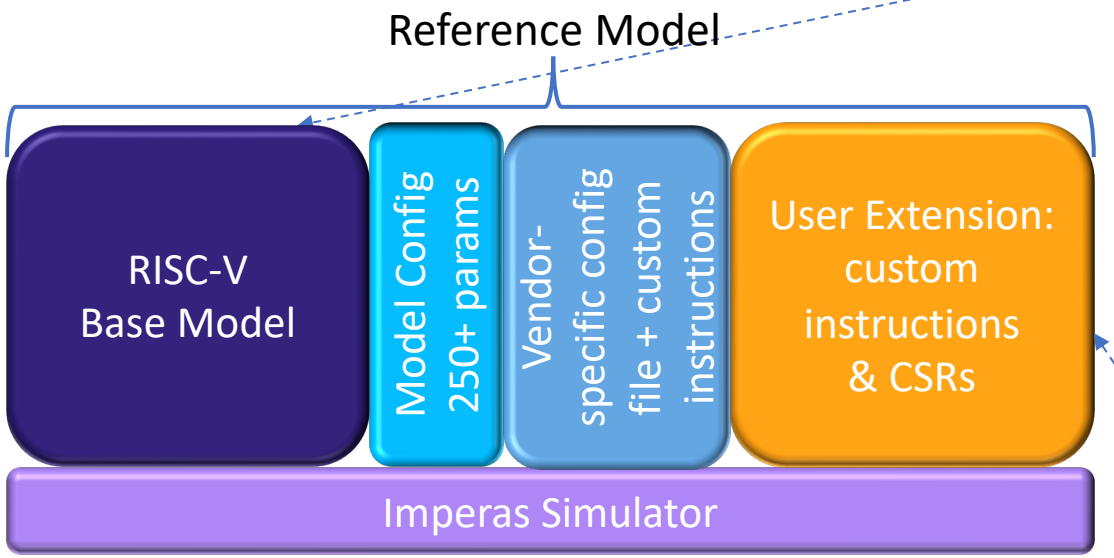
Imperas develops and maintains Base Model

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

Imperas provides methodology to easily extend base model

- Custom instructions added using same APIs as in Base Model
- Separate source files and no duplication to ensure easy maintenance
- 100+ page user guide/reference manual with many examples
- User extension source can be proprietary (Apache 2.0 open source license)

Model + Simulator Architecture (5)



Imperas develops and maintains Base Model

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

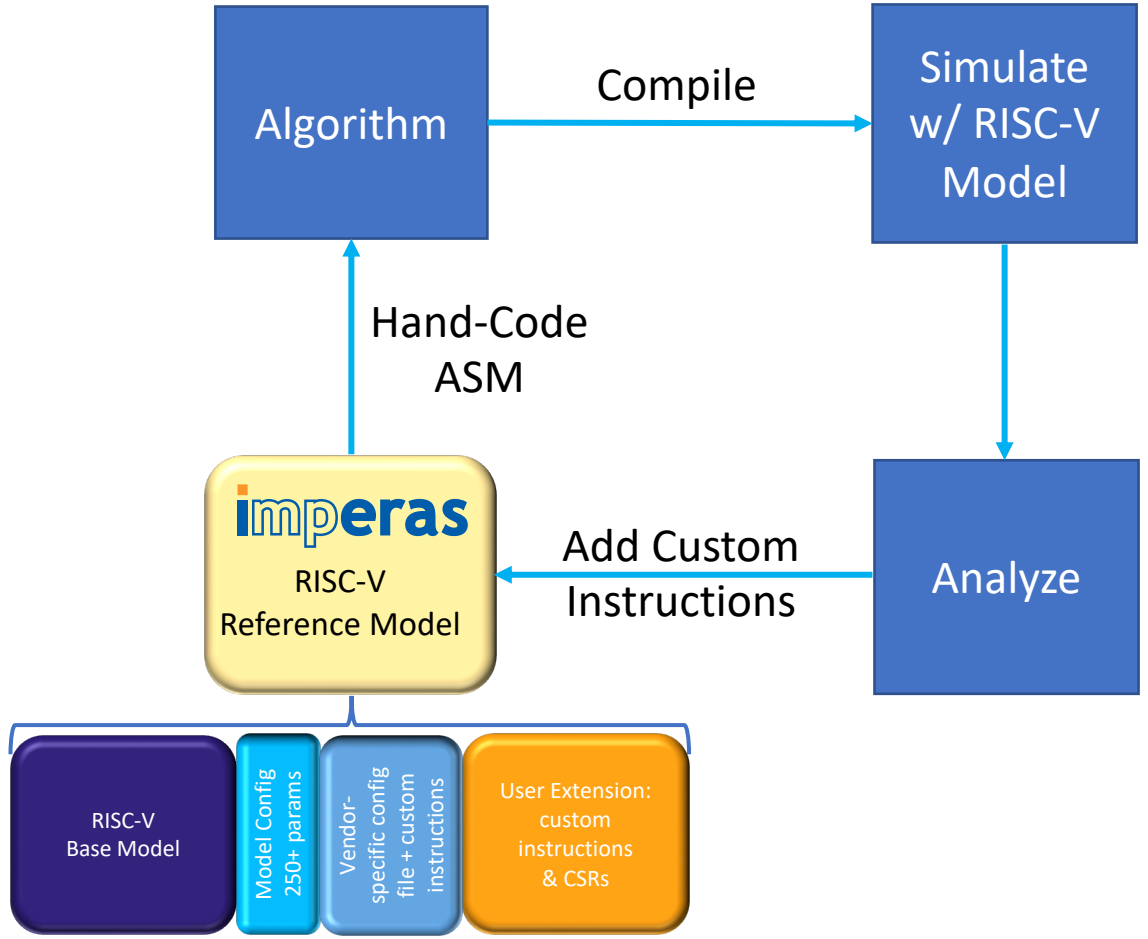
Imperas provides methodology to easily extend base model

- Custom instructions added using same APIs as in Base Model
- Separate source files and no duplication to ensure easy maintenance
- 100+ page user guide/reference manual with many examples
- User extension source can be proprietary (Apache 2.0 open source license)

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

Models Drive Customization

- Custom instructions are added to optimize a specific application or set of applications within a domain
- Models let you explore quickly
  - Much faster to develop than RTL
  - Better profiling information available
  - Easier to debug software
- Methodology
  - Start by characterizing the application to be optimized
  - Then add the custom instructions, evaluate, and iterate



Agenda

- RISC-V and processor verification
- RISC-V processor models
- **RISC-V processor verification methodology**
- Processor verification success
- Summary

5 Levels of RISC-V Processor DV Methodology

1) Asynchronous continuous compare
2) Synchronous step-and-compare
3) Post-simulation trace log file compare
4) Self-checking tests (e.g. Berkeley torture tests pre-2018)
5) Hello World

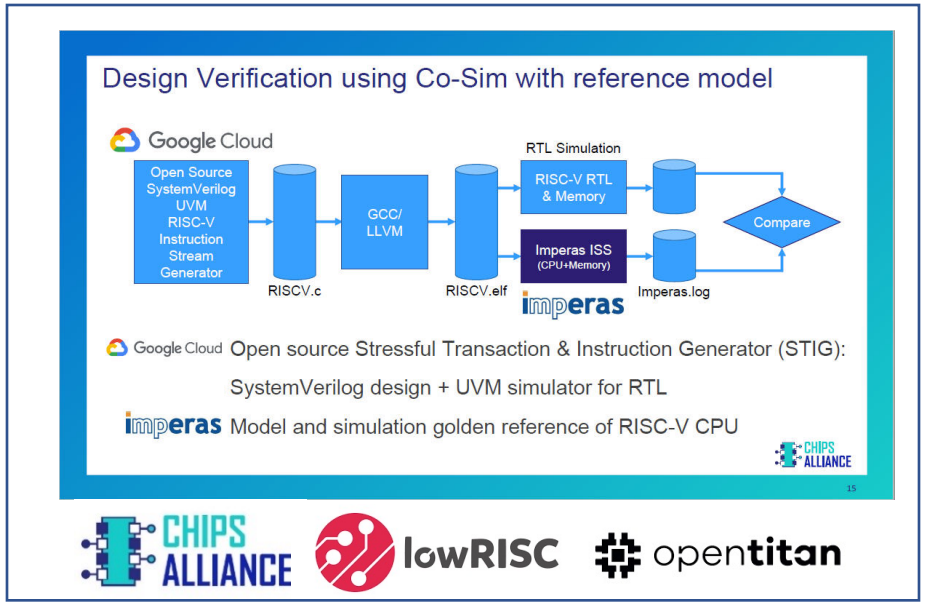
Quality

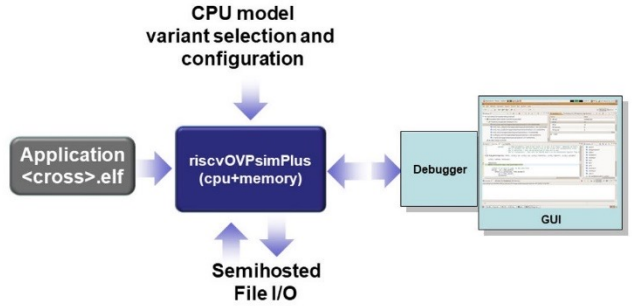
5 Levels of RISC-V Processor DV Methodology

1) **Asynchronous continuous compare**
2) Synchronous step-and-compare
3) **Post-simulation trace log file compare**
4) Self-checking tests (e.g. Berkeley torture tests pre-2018)
5) Hello World

Quality

3) Post-Simulation Trace Log File Compare (Entry Level DV)





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

Post-sim Trace Compare: Pros and Cons

- Pros:
  - Availability of quality RISC-V simulators (e.g. riscvOVPsimPlus from Imperas)
  - Simple to set up and use
- Cons:
  - Must run RTL simulation to the end
  - Cannot debug live
  - Incompatible trace formats (between RTL, ISS, …)
  - Easy to skip instructions, and only compare selected few
  - Difficult to verify asynchronous events (e.g. interrupts, debug requests)
  - Not a comprehensive DV strategy

➢ Post-sim trace compare is widely used

➢ Most effective as a complementary methodology to asynchronous continuous compare

riscvOVPsimPlus / riscvISATESTS: Commercial Users

imperas

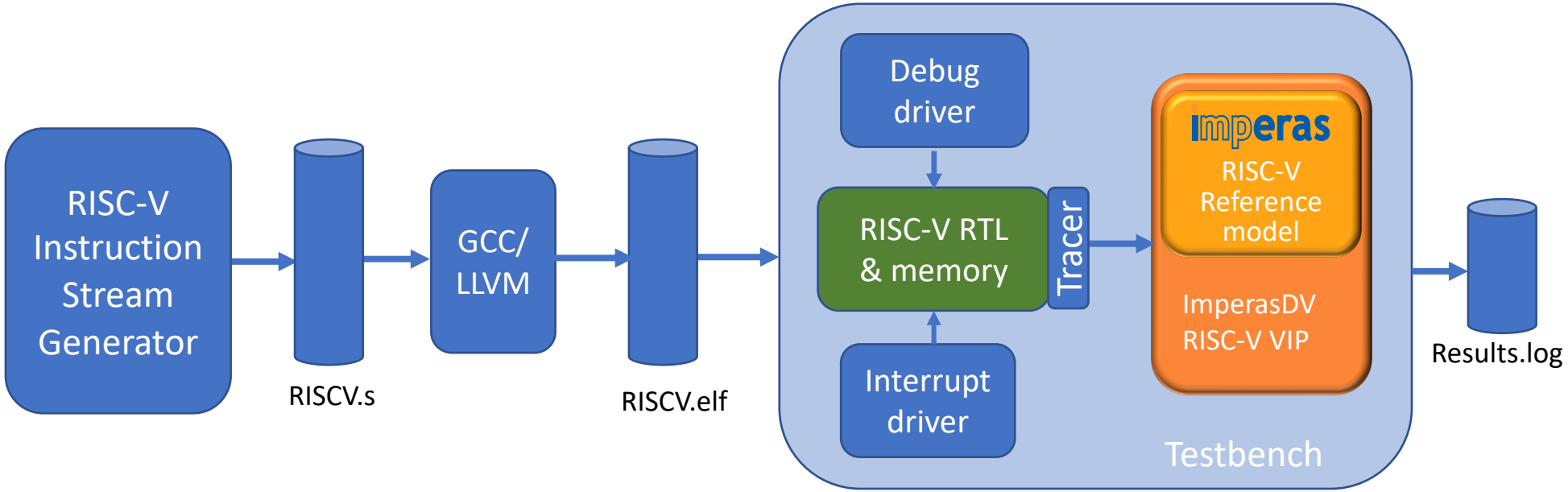
Downloaders from OVPworld of riscvOVPsimPlus / riscvISATESTS (21-feb-2023)

riscvOVPsimPlus / riscvISATESTS: University & Research Lab Users

imperas

Downloaders from OVPworld of riscvOVPsimPlus / riscvISATESTS (21-feb-2023)

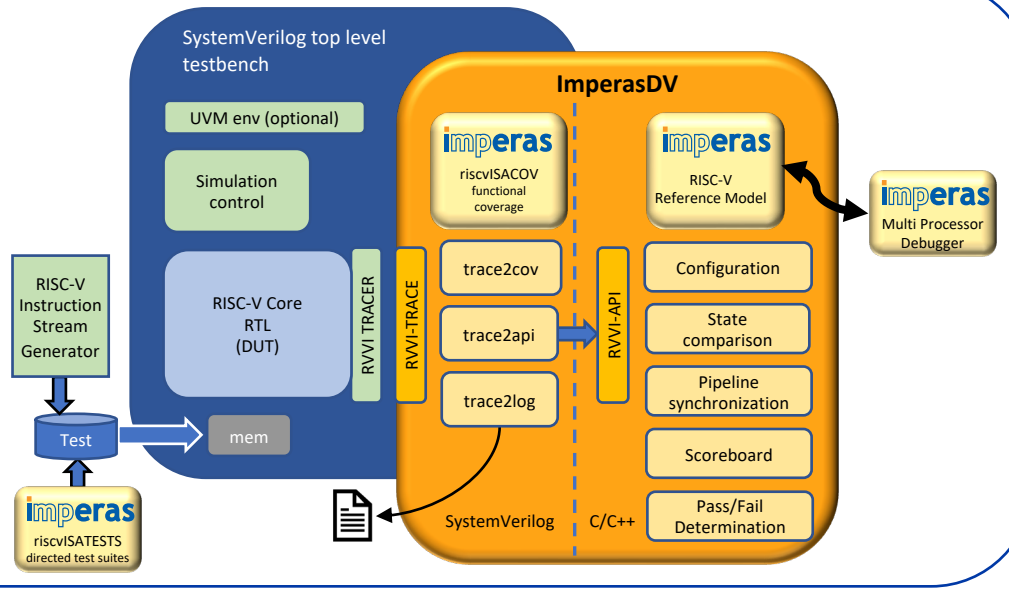
1) Async Continuous Compare
(Highest Quality DV Methodology)



- RTL and reference model are run in “lock-step” in the same simulation
- Internal state of the two is continuously compared
- Asynchronous events are driven into the DUT
- Tracer informs reference model about async events
- Verification IP handles async events, scoreboarding, comparison, pass/fail

ImperasDV Components

- Reference model needed for comparison of correct behavior
- Verification IP provides ease of use, saves time and resources
- RVVI standard provides communication between test bench and reference model subsystem
- riscvISACOV: functional coverage modules
- Test suites: riscvISATESTS, directed test suites for difficult extensions
- MultiProcessor Debugger (MPD) enables RTL-reference model co-debug



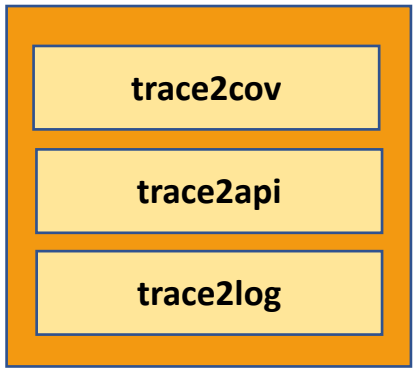
- Feature selection and design choices require serious consideration due to implications of every decision
  - Every addition dramatically compounds verification complexity
  - Adds schedule, resources, quality costs == big risks
- Before 2021, no off-the-shelf toolkit/products available for DV of processors … then came ImperasDV
- ***ImperasDV, with async continuous compare methodology, is needed to support features such as interrupts, privilege modes, Debug mode, multi-hart, multi-issue and OoO pipeline, …***

Async Continuous Compare Pros and Cons

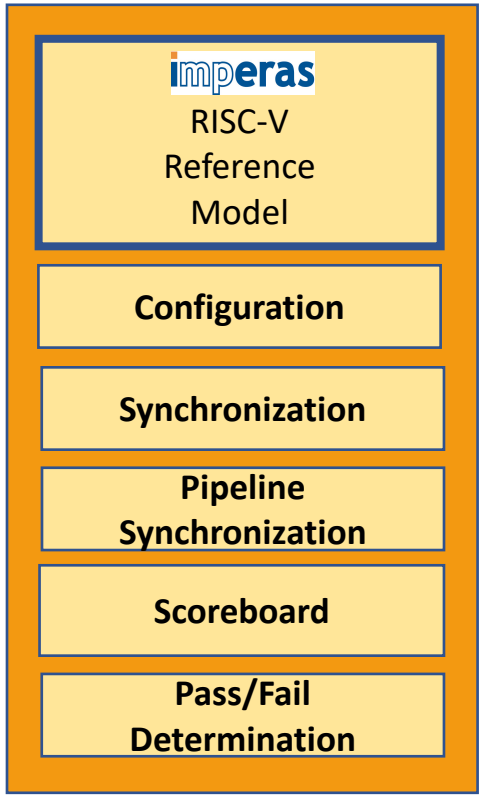
- Pros:
  - Instruction by instruction comparison
  - Comparison of execution flow, program data, internal state
  - Errors are flagged immediately – no runaway simulations
  - Detects synchronous and asynchronous bugs
  - Checking is done for you
  - Verification IP (VIP) is reusable across different core DV projects
  - Ease of use
  - Training, documentation, and support

- Cons:
  - Cost of VIP licenses

ImperasDV: Verification IP



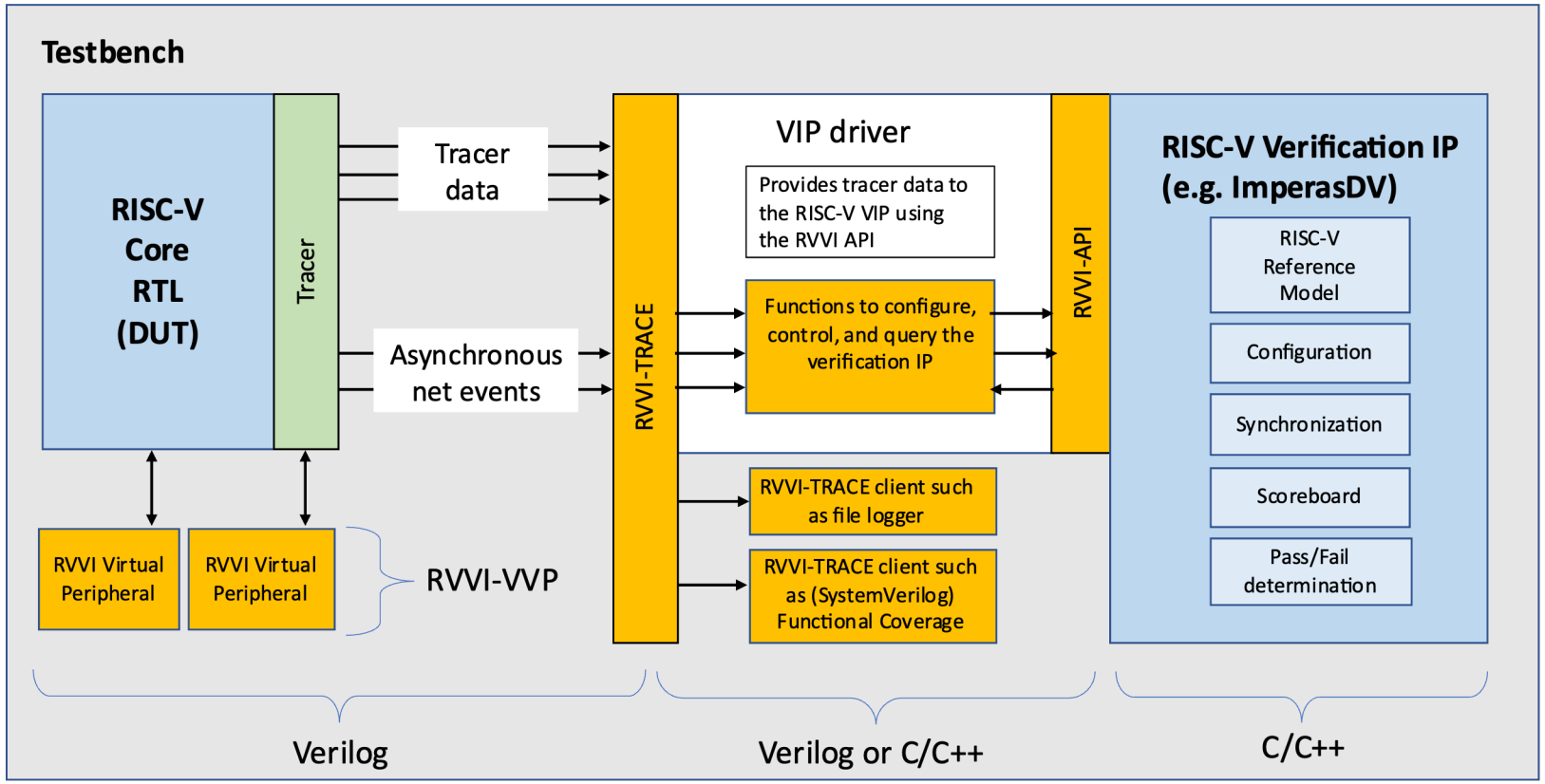
- Data prep for functional coverage
- Data movement from SystemVerilog to C verification IP
- Logging data



- Reference model encapsulation
- Includes DUT reference state storage
- Includes synchronization technology
  - Can run sync, async, interrupts, debug, multi-hart
- Pipeline synchronization is key for asynchronous event DV
- Includes comparison technology
  - Comparisons are done on *DUT/Reference Model processor events*; enables DV of multi-issue and OoO pipeline processors

RISC-V Verification Interface (RVVI)

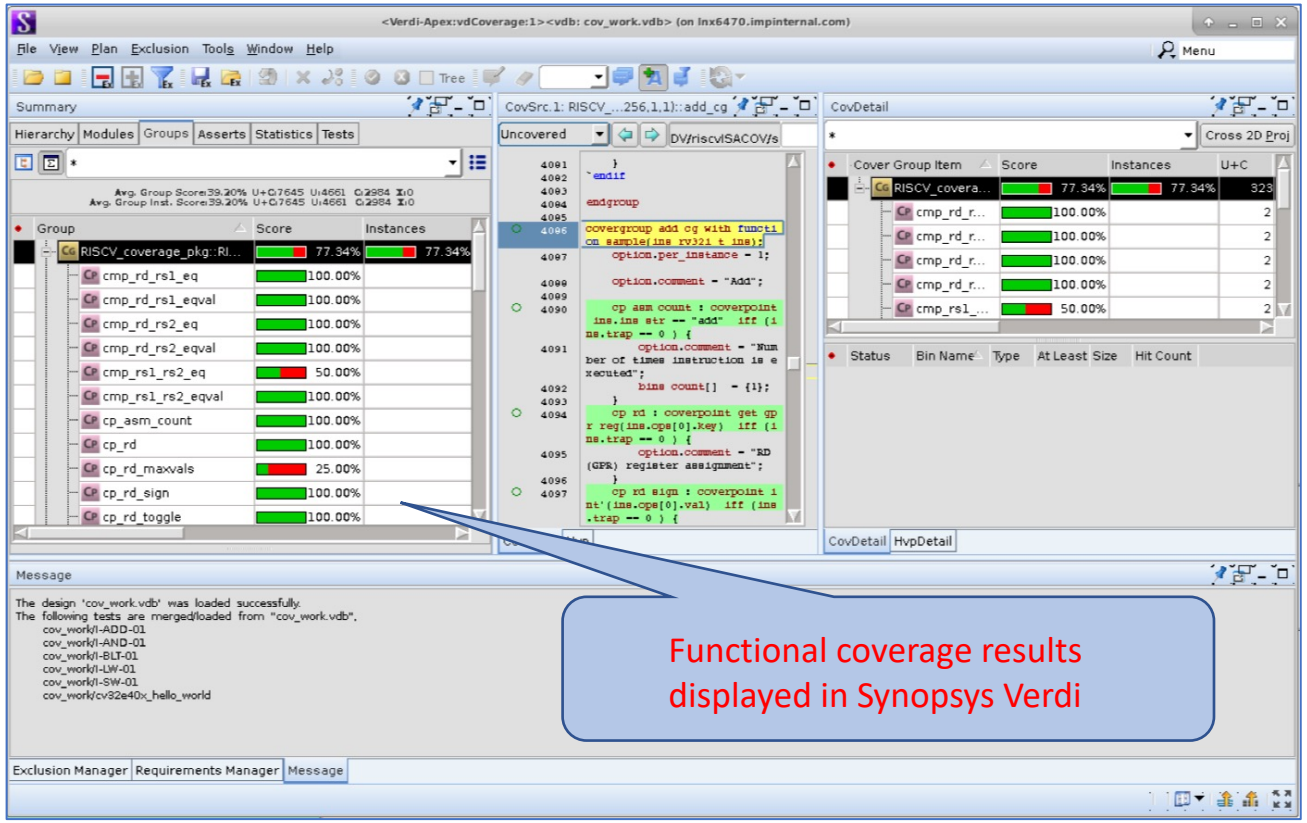
- Standardize communication between DUT, testbench, and RISC-V VIP
  - **RVVI-TRACE**: provides tracer data to RISC-V VIP
  - **RVVI-API**: function level interface to RISC-V VIP
  - **RVVI-VVP**: virtual peripherals
- Collaborative work has evolved over 3 years
  - Imperas, EM Micro, SiLabs, OpenHW Group



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

Functional Coverage is a Key Verification Metric ***Especially True for Processors***

```
covergroup add_cg with function sample(ins_t ins);
    option.per_instance = 1;
    cp_rd : coverpoint get_gpr_name(ins.ops[0].key, "add") {
    }
    cp_rd_sign : coverpoint int'(ins.ops[0].val) {
        bins neg  = {[$:-1]};
        bins zero = {0};
        bins pos  = {[1:$]};
    }
    cp_rs1 : coverpoint get_gpr_name(ins.ops[1].key, "add") {
    }
    cp_rs1_sign : coverpoint int'(ins.ops[1].val) {
        bins neg  = {[$:-1]};
        bins zero = {0};
        bins pos  = {[1:$]};
    }
    cp_rs2 : coverpoint get_gpr_name(ins.ops[2].key, "add") {
    }
    cp_rs2_sign : coverpoint int'(ins.ops[2].val) {
        bins neg  = {[$:-1]};
        bins zero = {0};
        bins pos  = {[1:$]};
    }
    cr_rs1_rs2 : cross cp_rs1_sign, cp_rs2_sign ;
    cmp_rd_rs1_eq : coverpoint ins.ops[0].key == ins.ops[1].key {
        bins rd_eq_rs1 = {1};
        bins rd_ne_rs1 = {0};
    }
    cmp_rd_rs1_eqval : coverpoint int'(ins.ops[0].val) == int'(ins.ops[1].val) {
        bins rd_eqval_rs1 = {1};
        bins rd_neval_rs1 = {0};
    }
    cmp_rd_rs2_eq : coverpoint ins.ops[0].key == ins.ops[1].key {
        bins rd_eq_rs2 = {1};
        bins rd_ne_rs2 = {0};
    }
    cmp_rd_rs2_eqval : coverpoint int'(ins.ops[0].val) == int'(ins.ops[1].val) {
        bins rd_eqval_rs2 = {1};
        bins rd_neval_rs2 = {0};
    }
    cmp_rs1_rs2_eq : coverpoint ins.ops[0].key == ins.ops[1].key {
        bins rs1_eq_rs2 = {1};
        bins rs1_ne_rs2 = {0};
    }
    cmp_rs1_rs2_eqval : coverpoint int'(ins.ops[0].val) == int'(ins.ops[1].val) {
        bins rs1_eqval_rs2 = {1};
        bins rs1_neval_rs2 = {0};
    }
endgroup
```



riscvISACOV (and ImperasDV) works with any SystemVerilog simulator

RISC-V Functional Coverage

For a processor there are different types of functional coverage required:

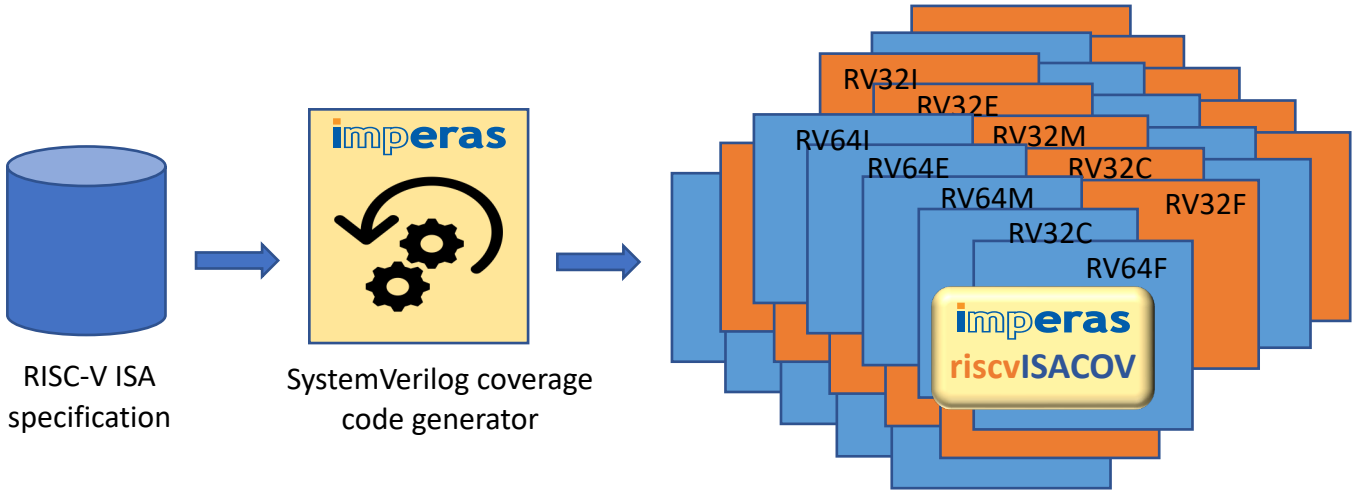
- Standard ISA architectural features
  - unpriv. ISA items: mainly instructions, their operands, their values

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

Functional Coverage of RISC-V Instructions: Scope

- There are many different instructions in the RV64 extensions:
  - Integer: 56, Maths: 13, Compressed: 30, FP-Single: 30, FP-Double: 32
  - Vector: 356, Bitmanip: 47 Krypto-scalar: 85
  - P-DSP: 318
  - For RV64 that is 967 instructions…
- Each instruction needs SystemVerilog covergroups and coverpoints
  - 10-40 lines of SystemVerilog for each instruction
- 10,000-40,000++ lines of code to be written
  - Not design or core specific

riscvISACOV Is Automatically Generated SystemVerilog Functional Coverage



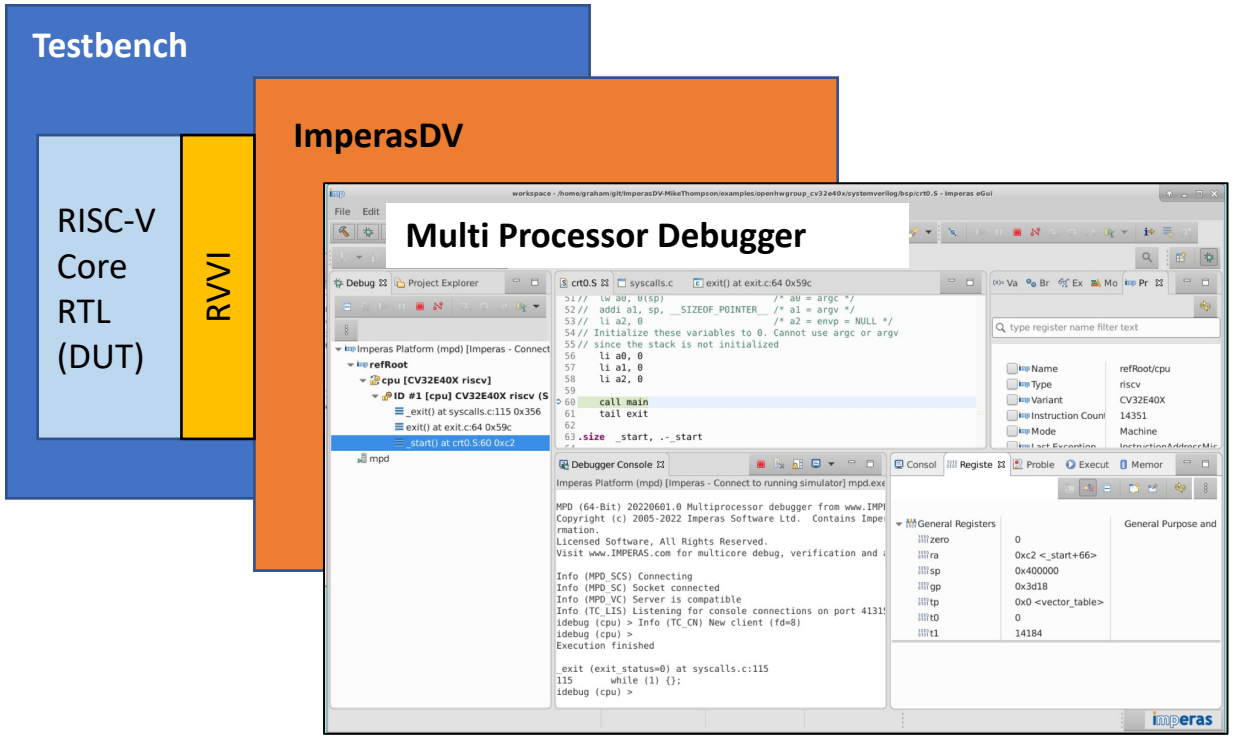
- riscvISACOV provides functional coverage of Instructions and operands
- Roadmap includes CSRs and data hazards
- Imperas tools can automatically generate functional coverage code for custom instructions

Test Stimuli

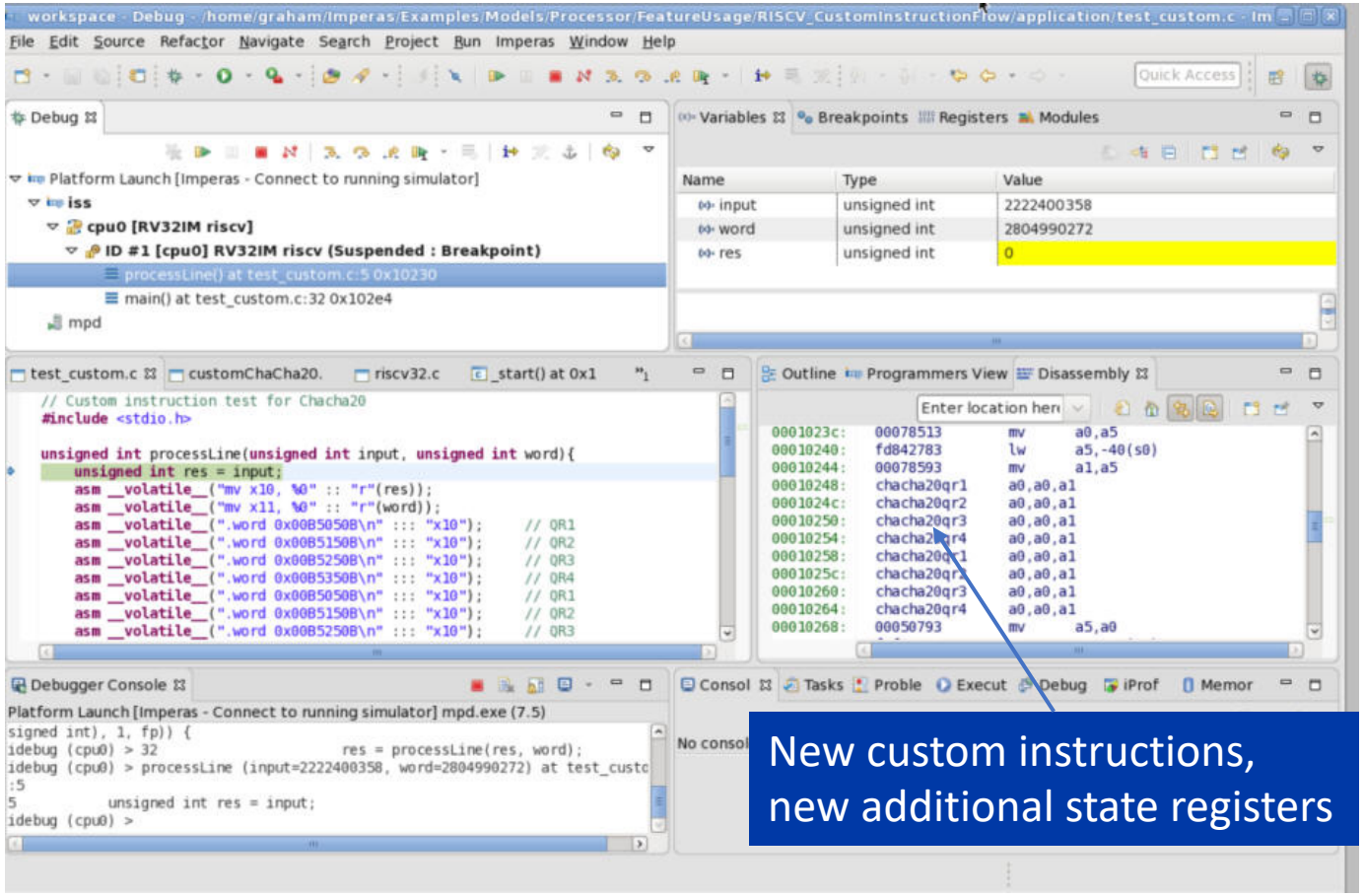
- Instruction Stream Generator (ISG) and/or directed tests
- ISG generates test programs using constrained random approach
  - Most often obtain the ISG:
    - Commercial such as Valtrix STING
    - Open source such as Google riscv-dv
  - Require toolchains like GCC, LLVM for assemblers, linkers
  - Require functional coverage so that you know what you've tested!
- Directed tests
  - Imperas have developed a directed RISC-V test generator, instruction coverage verification IP and a mutating fault simulator (for test qualification) 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
  - Free Imperas architectural validation test suites (50+), including RV32/64 I, M, C, F, D, B, K, V, P
    - https://github.com/riscv-ovpsim/imperas-riscv-tests
  - Imperas commercial directed test suites for vector extension, protected memory components
    - Can support any RISC-V vector or PMP configuration; the user selects the configuration and Imperas generates the test suite

ImperasDV: Debug with MPD

- Imperas MPD is an Eclipse based debug tool
- Can debug using source line or instruction level
- See new custom instructions and any new additional state registers
- Break at the first mismatch, debug SW and RTL concurrently

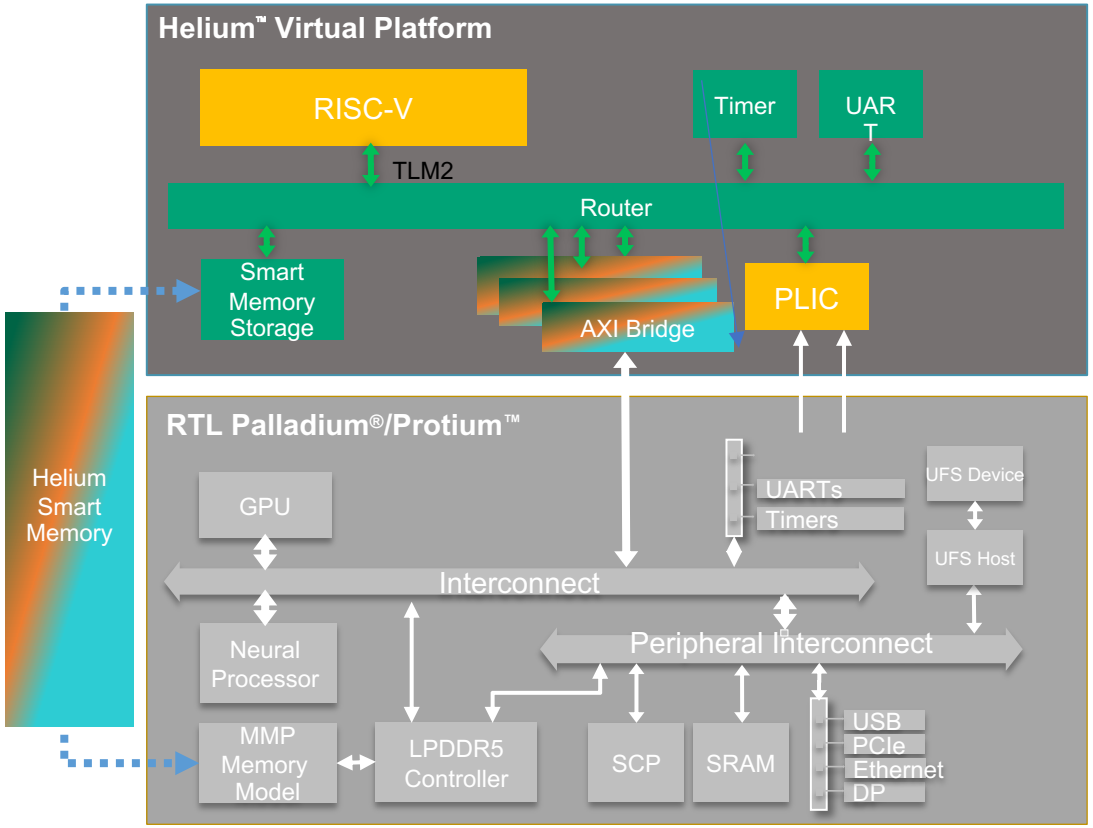


Software Debug and Analysis Tools Automatically Work With the Custom Instructions



Hybrid Simulation-Emulation for HW-SW Co-Verification

- It takes more than just the RTL simulator for comprehensive processor verification
- An additional tool is the hardware emulator, e.g. Cadence Palladium
- The interface to Palladium is via the Cadence Helium SystemC simulator

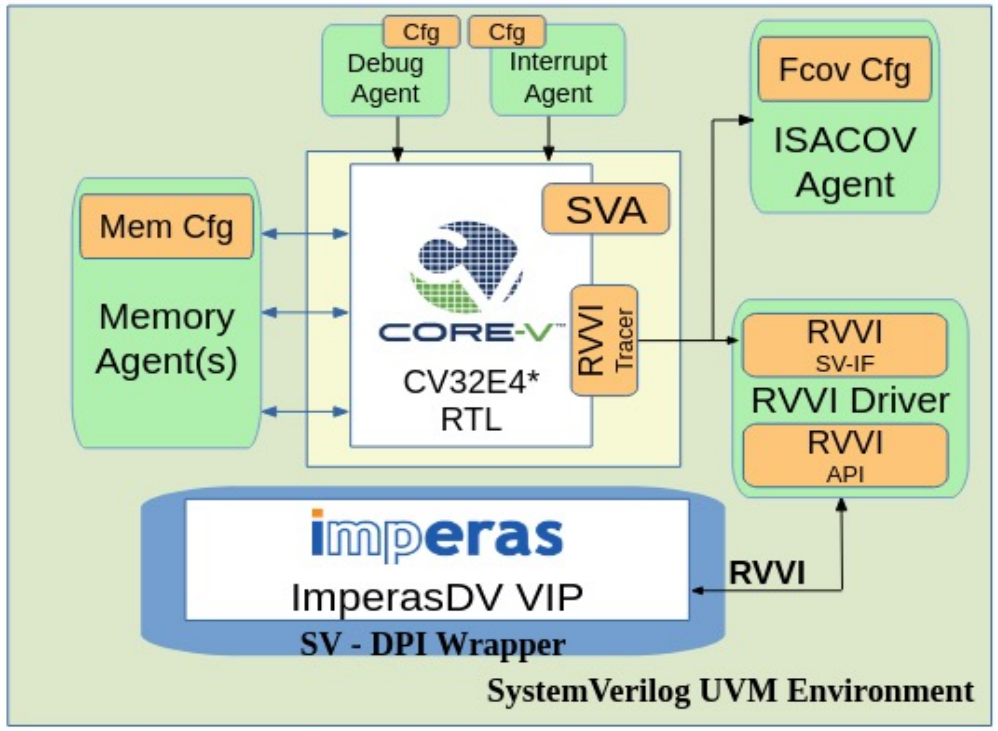


Agenda

- RISC-V and processor verification
- RISC-V processor models
- RISC-V processor verification methodology
- **Processor verification success**
- Summary

RISC-V Processor DV Results

- OpenHW Core-V-Verif
  - OpenHW users now on 3rd generation of Core-V-Verif DV flow
  - CV32E40P successfully taped out
  - CV32E40Pv2, CV32E40S, CV32E40X, CV32E20 using this flow today; all expected to complete DV this year
  - OpenHW “Wally” core
- Other successful DV projects using Imperas include Codasip, MIPS, Nagra, NSITEXE, Nvidia Networking, Skyworks, …



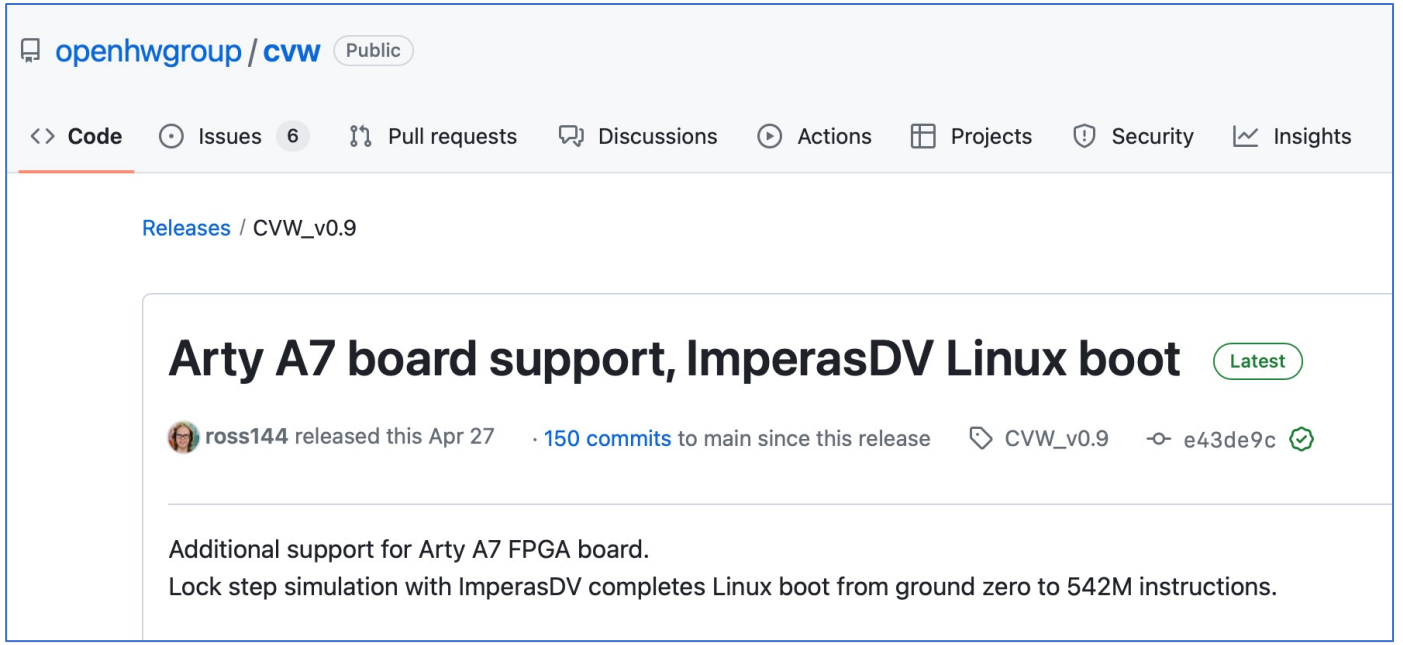
Case Study: Wally RISC-V Core

- Configurable core:
  - RV32I, RV32E, RV64I, RV64E
  - A, C, F, D, M extensions, privileged modes, CSRs
  - MMU/TLB virtual memory, caches
- Developed at Harvey Mudd College / Oklahoma State University
  - Focus: high quality core for processor architecture education
  - Now in OpenHW as CORE-V Wally (https://github.com/openhwgroup/cvw)

- Status in January 2023 – before starting to use RVVI + ImperasDV for verification:
  - Passing all RISC-V International compliance tests, Imperas compatibility tests
    - Using Compliance Level post-simulation signature file compare
  - Boots Linux

Wally + RVVI – Status (July 2023)

- RVVI Tracer + testbench integration: 3 days of effort
- Results:
  - 20+ bugs found in simulation almost immediately using ImperasDV and riscv-dv
  - Reached Linux prompt with continuous checking: 2 days of simulation
  - One bug found just *after* the Linux command prompt (!)
  - Functional coverage achieved by booting Linux: covergroups 37% (bins 3%)
- Future work:
  - Boot Linux with co-sim using hardware assisted verification
  - Achieve 100% functional coverage using constrained-random tests

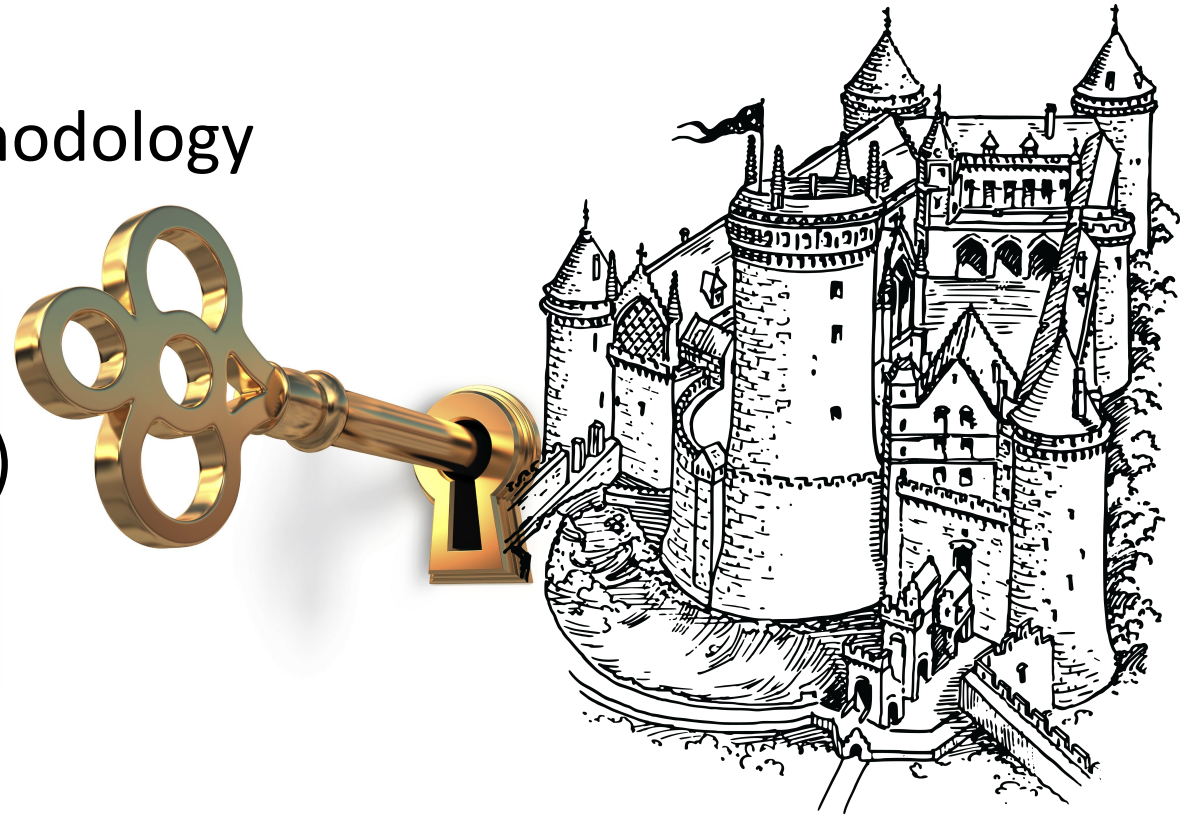


Agenda

- RISC-V and processor verification
- RISC-V processor models
- RISC-V processor verification methodology
- Processor verification success
- **Summary**

Processor Verification: The Key to the RISC-V Castle

- High quality reference models
- Asynchronous continuous compare methodology
- Verification IP
- Verification standards (RVVI)
- Verification metrics (functional coverage)



Thank you

Larry Lapides

LarryL@imperas.com