Understanding the RISC-V Verification Ecosystem

Simon Davidmann, Imperas Software
Aimee Sutton, Imperas Software
Lee Moore, Imperas Software
Not talking about these familiar concepts...

• SystemVerilog simulators, UVM
• Formal
• CI technology
• Hardware assist
• FPGA prototyping
• VHDL
• Virtual platforms
• Verification services companies

=> All very important, but not covered in this talk....
Agenda

• Introduction to Imperas
• Introduction to RISC-V
• RISC-V processor verification challenges
  • Why is RISC-V processor DV so critical?
• RISC-V processor verification environment components
• RISC-V Verification approaches
• RISC-V Verification standards
• RISC-V Verification IP
• Functional coverage for RISC-V processors
• Verification Case studies
  • OpenHW Group CV32E40X processor
  • Wally RISC-V processor
• Summary
Agenda

• Introduction to Imperas
  • Introduction to RISC-V
  • RISC-V processor verification challenges
    • Why is RISC-V processor DV so critical?
  • RISC-V processor verification environment components
  • RISC-V Verification approaches
  • RISC-V Verification standards
  • RISC-V Verification IP
  • Functional coverage for RISC-V processors
  • Verification Case studies
    • OpenHW Group CV32E40X processor
    • Wally RISC-V processor
• Summary
Imperas

• 2008 – developed world class processor modeling & simulation solutions for many ISAs for virtual prototyping and software development
  • A good, growing, and profitable business

• 2016 started looking at RISC-V

• 2018 RISC-V processor developers started using Imperas RISC-V model as reference for their hardware verification

• For last 5 years have been assisting companies with their RISC-V DV needs

• For last 4 years started working collaboratively with free and open source solutions
  • e.g. OpenHW Group open source highly verified industrial quality RISC-V cores

• For last 3 years working on RISC-V verification standards and advanced methodologies

• 2022 Introduced first RISC-V processor DV solution that works out-of-the-box
riscvOVPsimPlus / riscvISATESTS – Commercial firms

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

- Introduction to Imperas
- **Introduction to RISC-V**
  - RISC-V processor verification challenges
    - Why is RISC-V processor DV so critical?
  - RISC-V processor verification environment components
  - RISC-V Verification approaches
  - RISC-V Verification standards
  - RISC-V Verification IP
  - Functional coverage for RISC-V processors
  - Verification Case studies
    - OpenHW Group CV32E40X processor
    - Wally RISC-V processor
- Summary
About RISC-V

• Developed by researchers at Berkeley in 2010 under Prof. Patterson
• RISC-V is an open standard Instruction Set Architecture (ISA) enabling a new era of processor innovation through open collaboration
• RISC-V International (riscv.org) is the global non-profit home of the RISC-V ISA, related specifications, and stakeholder community
  • 3,000+ RISC-V members across 70+ countries contribute and collaborate to define RISC-V open specifications as well as convene and govern related technical, industry, domain, and special interest groups
Initially 47 instructions, now over 1,000, in 70+ ISA extensions
RISC-V Profiles & Platforms

2022 Profiles

RVA22_[32,64] - Application Profile Extensions
RVM22_[32,64] - Microcontroller Profile Extensions

- 2022 Profiles
- Vector
- Crypto
- Bit Manipulation
- Packed SIMD
- Virtual Memory
- Cache Management Operations
- Alternate Floating Point Formats
- Code size reduction & Embedded Support
- TEE
- JIT support instructions

2022 Platform Definitions

- What is this for?
  - Limited variations for distros to support
  - Complete description for software to optimize and customize to the platform
- Initial Targets
  - Linux Dev
  - RTOS (TBD)
- Content
  - Profiles
  - Binary Interfaces
  - Discovery
  - Device tree
  - ...

• Ways of grouping the many extensions....
RISC-V Evolving (2022)

Adaptable

Today
- Bases: RV32I, RV64I
- Extensions (70 to date): ACDFHMQV, priv 1.12, SV*, Zb, Zfinx, Crypto Scalar, etc.
- Non-ISA: psABI, SBI, UEFI, Etrace
- Organization: 9 committee, 28 Special Interest Groups (strategy, gap analysis & prioritization), 26 Task Groups (creating specifications)
- Member defined custom extensions (X), for example XVentanaCondOps or V0.7
- No baggage
- Efficient: modular, modern ISA

Tomorrow
- Bases: RV32E, RV64E
- Profiles: RV120, RVA20, RVA22, RVA23
- Extensions (~30): Crypto Vector, Zc, subsets, etc.
- Non-ISA: ACPI, AP-TEE, IOMMU, IOPMP, Nexus, PLIC, SEE, Security Model, Unified Discovery, Watchdog Timer, CMQRI

- Bases: RV128I
- Profiles: RVA24, RVM
- Platforms: OS-A, OS-M
- Extensions: P, Matrix
- Beyond: CHERI, GPU
Agenda

• Introduction to Imperas
• Introduction to RISC-V

• RISC-V processor verification challenges
  • Why is RISC-V processor DV so critical?
  • RISC-V processor verification environment components
  • RISC-V Verification approaches
  • RISC-V Verification standards
  • RISC-V Verification IP
  • Functional coverage for RISC-V processors
  • Verification Case studies
    • OpenHW Group CV32E40X processor
    • Wally RISC-V processor

• Summary
The RISC-V Disconnect

RISC-V Core User

Expects core quality to be the same as Arm

RISC-V Core Developer

Unlikely to have resources needed to be able to develop all the technologies required to perform the same level of verification as Arm
Putting Processor Verification into Context....

1,000,000,000,000

The number of verification cycles Arm uses when verifying an Arm core

- SystemVerilog simulator executes 2,000 cycles / second
  => 15,000 SystemVerilog simulators running for 1 year

- HW emulator or FPGA runs at 1,000,000 cycles / second
  => 30 years of running needed...

- OK – so this is for high end performance OoO, MP, VM cores (full apps processors)
  - Embedded processors will be an order of magnitude less...
RISC-V Design Verification Challenges

• Processor verification has been a niche discipline
  • Proprietary techniques
• No industry-standard best practices or verification IP
  • Until recently... (stay tuned)
• Techniques from ASIC/SoC verification are insufficient
• New methods are required
  • Take advantage of what has worked in the ASIC world
  • Add to it and enhance for RISC-V
So what is being done in the RISC-V world

• In the RISC-V world, it is unlikely that one company can spend the $ or can hire the people to develop all they need...
  • [Arm relies on ISA / design royalty, Intel relies on silicon sale...]

1) Partnering and Collaboration in non-competitive areas
2) Attracting players into the verification ecosystem to develop needed solutions
3) Building standards to facilitate re-use and efficiency

• If it does not differentiate your product offering / company
  • You can collaborate externally
  • You can license commercial tools
So what have we learnt in last 5 years...
There are many approaches for ‘verification’ of new processors

- Does a program run? – ‘hello world’ tests
- Is there simple correct computation? – ‘self checking tests’

- Signature checking – ‘post simulation signature dump compares’
- Trace log checking – ‘post simulation trace file compare’

- Basic step and compare co-simulation – ‘instruction retire compare’
- Advanced, e.g. commercial solutions – ‘async-lock-step-compare’

[Note: this discussion is only about dynamic simulation verification – there are of course many excellent commercial formal verification solutions]
Agenda

• Introduction to Imperas
• Introduction to RISC-V
• RISC-V processor verification challenges
  • Why is RISC-V processor DV so critical?
  • RISC-V processor verification environment components
    • RISC-V Verification approaches
    • RISC-V Verification standards
    • RISC-V Verification IP
    • Functional coverage for RISC-V processors
    • Verification Case studies
      • OpenHW Group CV32E40X processor
      • Wally RISC-V processor
  • Summary
RISC-V processor verification environment components

- Test Programs
- Instruction Set Simulators
- DUT + Tracer
- Processor reference model
- Verification IPs
Test programs

• Directed tests
  • Write your own
  • Compliance tests (RISC-V International)
  • Architectural Compatibility test suites (Imperas open source riscvISATESTS)
  • Configurable Commercial test suites (e.g. Imperas PMP and Vector)
  • Other open source, e.g. OpenHW directed test suites (synchronous & asynchronous)

• Instruction stream generators (ISG)
  • Configurable to match processor extensions
  • Open source solutions
    • e.g. riscv-dv (Google / CHIPS Alliance)
  • Commercial solutions
    • e.g. Valtrix STING
Instruction Set Simulators

- **ISS**
  - Simulate the execution of a program on a processor
  - Produce a trace file output
  - Open source solutions
  - Commercial/closed-source solutions
    - e.g. riscvOVPsimPlus

```
RISCV.elf  Imperas ISS (cpu+memory)  Imperas_trace.log
```
DUT + Tracer

• **DUT (Design Under Test)**
  • RTL for RISC-V processor
  • Memory model and bus i/f
  • Ability to load test program into memory

• **Tracer**
  • Extracts information needed for DV
    • e.g. PC, register values
  • Bespoke to particular microarchitecture
  • Often written by processor designers
  • Can use RVVI-TRACE standard
Processor Reference Model

• Reference model requirements:
  • Configurable to select RISC-V ISA extensions
  • Ability to extend / add customizations (e.g. instructions, CSRs)
  • Can run in co-simulation configuration
  • Can be controlled from other simulator
  • Ability to “step” reference model at significant events (retire, trap)
  • Can run in lock-step with the RTL simulator
  • Functions to query state of model for comparison
Imperas is used as RISC-V Golden Reference Model

- 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
  - Vector extension, versions 0.7.1, 0.8, 0.9, 1.0
  - Bit Manipulation extension, versions 0.90, 0.92, 0.93, 0.94, 1.0.0
  - Hypervisor version 0.6.1, 1.0
  - Debug versions 0.13.2, 0.14, 1.0.0
  - K - Crypto Scalar version 0.7.1, 1.0.0
  - K - Crypto Vector version 0.3.0
  - P - DSP versions 0.5.2, 0.9.6
- Model source included under Apache 2.0 open source license

http://www.imperas.com/riscv
Imperas RISC-V reference model

- Imperas develops and maintains base model
  - Base model implements RISC-V specification in full
- Fully user configurable to select required ISA extensions
- Fully user configurable to select which version of each ISA extension
- Imperas provides methodology to easily extend base model
- Imperas model is architected for easy extension & maintenance

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

• Requirements:
  • Instance in SystemVerilog test bench
  • Scoreboard
  • Functional Coverage
  • Logger
  • Signature writers
  • Virtual peripherals (for async event generation)
  • Comparators
  • Synchronizers
  • Fault injectors
  • ...
Agenda

• Introduction to Imperas
• Introduction to RISC-V
• RISC-V processor verification challenges
  • Why is RISC-V processor DV so critical?
• RISC-V processor verification environment components

• **RISC-V Verification approaches**
  • RISC-V Verification standards
  • RISC-V Verification IP
  • Functional coverage for RISC-V processors
  • Verification Case studies
    • OpenHW Group CV32E40X processor
    • Wally RISC-V processor

• Summary
Compliance versus Verification

- 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, ...

- 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”
  - For RISC-V, compliance testing is a very small percentage of full hardware verification...
Many approaches for ‘verification’ (recap)

- Does a program run? – ‘hello world’ tests
- Is there simple correct computation? – ‘self checking tests’

- Signature checking – ‘post simulation signature dump compares’
- Trace log checking – ‘post simulation trace file compare’

- Basic step and compare co-simulation – ‘instruction retire compare’
- Advanced, e.g. commercial solutions – ‘async-lock-step-compare’

[Note: this discussion is only about dynamic simulation verification – there are of course many excellent commercial formal verification solutions]
RISC-V processor ‘verification’ approaches

• Simple:
  • run program ‘hello world’ tests
  • self checking tests

• Compliance:
  • post simulation signature dump file compare
  • post simulation trace log file compare

• Verification:
  • Basic ‘instruction retire step compare’ co-simulation
  • Quality ‘async lock step compare’ co-simulation
Simple Level
Self-Checking Tests

• Components:
  • RISC-V processor (DUT) and test program; optionally ISS

• Process:
  • Each test program checks its results
    • Prints message to log
    • Or writes bit to memory
      • for later reading

Application <cross>.elf → RISC-V RTL & memory → “Test Passed”
Simple Level
Self-Checking Tests: Pros and Cons

• Pros:
  • Simple to set up and execute
    • Free ISS: https://github.com/riscv-ovpsim
    • Free compiler: https://github.com/Imperas/riscv-toolchains
  • RISC-V tests freely available, e.g. Berkeley tests
    • https://github.com/riscv-software-src/riscv-tests

• Cons:
  • Simple tests cover a small subset of processor functionality
  • Not a complete DV strategy
Compliance Level
Post-Simulation Signature File Comparison

• Components:
  • RISC-V processor (DUT) and test program
  • ISS + reference model

• Process:
  • Run the test program on the DUT and save the output (signature file)
  • Run ISS + reference model, write signature file
  • Compare / diff file results
  • This is the approach taken by RISC-V International for their architectural validation (“compliance tests”)

Application <cross>.elf
RISC-V RTL & memory

riscOVPsimPlus (cpu+memory)

Application <cross>.elf

RISCV.sig Signature file

Compare

RISCV.sig Signature file
Compliance Level
Post-Sim Signature file compare: Pros and Cons

• **Pros:**
  • Simple to set up and execute
    • Free ISS: [https://github.com/riscv-ovpsim](https://github.com/riscv-ovpsim)
    • Free compiler: [https://github.com/Imperas/riscv-toolchains](https://github.com/Imperas/riscv-toolchains)
  • RISC-V tests & compliance level tests freely available

• **Cons:**
  • Directed tests cover a subset of processor functionality
  • Easy to have incomplete or wrong info in signatures (misses behaviors)
  • Not a complete DV strategy
Compliance level
Post-Simulation Trace Log File Compare

- **Components**
  - Test programs
    - Can be generated by an ISG – Instruction Stream Generator
  - Instruction Set Simulator (ISS) + reference model
  - DUT and Tracer
  - RTL simulator
  - Comparison script
Sequential:
1) Run random generator (ISG) to create tests
2) Simulate using ISS; write trace log file
2) Simulate using RTL; write trace log file
3) Run compare program to see differences / failures
Compliance Level
Post Sim Trace Log File 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
  • Difficult to verify asynchronous events (e.g. interrupts, debug requests)
  • Incompatible trace formats (between RTL, ISS, …)
  • Easy to skip instructions, and only compare selected few
  • Not a comprehensive DV strategy
Verification Level
Sync. Step-And-Compare co-simulation

• Components
  • Test programs (can be compliance, directed, or generated by an ISG)
  • Processor reference model
  • DUT and tracer
  • Step-and-compare logic
  • Comprehensive test bench
  • RTL simulator
Verification Level
Sync. Step-And-Compare co-simulation: Process

- Reference model is encapsulated in a SystemVerilog testbench
- Control block steps both DUT and reference model
- Extracts data from each; compares results on-the-fly
- Differences reported immediately
Verification Level
Sync. Step-And-Compare co-sim: Pros and Cons

• Pros:
  • Instruction by instruction lock-step comparison
  • Comparison of execution flow, program data, internal state
  • Errors are flagged immediately – no runaway simulations
  • Detects synchronous bugs

• Cons:
  • Step-and-compare logic can be fragile and error prone
  • Does not easily verify asynchronous events
Verification Level
Async. Step-And-Compare co-simulation

- Components
  - Test programs (can be generated by an ISG)
  - Processor reference model
  - DUT and tracer
  - Asynchronous event drivers (e.g. UVM agents)
  - RISC-V VIP
  - Comprehensive test bench
  - RTL simulator
Verification Level
Async. Step-And-Compare co-simulation: Process

- Asynchronous events are driven into the DUT
- Tracer informs reference model about async events
- Verification IP handles scoreboard, comparison, coverage, pass/fail
Verification Level
Async. Step-And-Compare co-sim: Pros and Cons

• Pros:
  • All the benefits of sync. step-and-compare
  • Responds to asynchronous events
  • Checking is done for you
  • VIP is reusable across different core DV projects
  • Ease of use
  • Training, documentation, and support

• Cons:
  • Cost of VIP licenses
## Verification Levels: Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Check basic functionality (E.g. compliance)</th>
<th>Supports constrained-random stimulus</th>
<th>Simulation ends after specified # of errors</th>
<th>Debug at the point of error</th>
<th>Verifies asynchronous events</th>
<th>Achieves verification closure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-checking tests</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signature file compare</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post-sim trace file compare</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronous step and compare</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Asynchronous step and compare</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Agenda

• Introduction to Imperas
• Introduction to RISC-V
• RISC-V processor verification challenges
  • Why is RISC-V processor DV so critical?
• RISC-V processor verification environment components
• RISC-V Verification approaches

• RISC-V Verification standards
  • RISC-V Verification IP
  • Functional coverage for RISC-V processors
  • Verification Case studies
    • OpenHW Group CV32E40X processor
    • Wally RISC-V processor

• Summary
Open Standards
RISC-V Verification Interface: RVVI

- RVVI = RISC-V Verification Interface
  - [https://github.com/riscv-verification/RVVI](https://github.com/riscv-verification/RVVI)
- Work has evolved over 3 years
  - Imperas, EM Micro, SiLabs, OpenHW
- Standardize communication between DUT, testbench, and RISC-V VIP
- Two parts (currently):
  - **RVVI-TRACE**: signal level interface to RISC-V VIP
  - **RVVI-API**: function level interface to RISC-V VIP
Open Standard: RVVI-TRACE

• Defines information to be extracted by tracer
• SystemVerilog interface
• Includes functions to handle asynchronous events
  • e.g. interrupts, debug requests

https://github.com/riscv-verification/RVVI/tree/main/RVVI-TRACE
Open Standard: RVVI-API

- Standard functions that RISC-V processor VIPs need to implement
- Supports a step-and-compare co-simulation methodology
- C and SystemVerilog versions available
- [https://github.com/riscv-verification/RVVI/blob/main/include/host/rvvi/rvvi-api.h](https://github.com/riscv-verification/RVVI/blob/main/include/host/rvvi/rvvi-api.h)
Agenda

• Introduction to Imperas
• Introduction to RISC-V
• RISC-V processor verification challenges
  • Why is RISC-V processor DV so critical?
• RISC-V processor verification environment components
• RISC-V Verification approaches
• RISC-V Verification standards
• **RISC-V Verification IP**
  • Functional coverage for RISC-V processors
  • Verification Case studies
    • OpenHW Group CV32E40X processor
    • Wally RISC-V processor
• Summary
RISC-V Processor VIP

• Requirements:
  • Standard interface to receive tracer data
  • Standard way to receive asynchronous events
  • Configurable, extendable RISC-V processor reference model
  • Methods to configure, control and query the reference model
  • Mechanism to compare DUT state with the reference model and report errors/mismatches
  • A method to verify DUT response to asynchronous events
ImperasDV
Configurable Reference

• Imperas configurable reference model
  • Fully user configurable to select required ISA extensions, versions
  • Extensible to match user customizations
• Configuration methods related to memory map (volatile regions) and CSRs
ImperasDV Components
Control and Introspection

- RVVI-TRACE data is converted into function calls (RVVI-API) which provide DUT state information to the reference model
- Synchronization keeps the reference model running in lock-step with the DUT
ImperasDV Components
Asynchronous Events

- Predictive engine is notified about asynchronous events via RVVI-API
- Analyzes the current state of the DUT and determines which responses to these events are legal
ImperasDV Components Comparison

- RVVI-API methods invoke comparison between RTL and reference
- Scoreboard keeps track of all passed and failed comparisons
ImperasDV Components
Coverage interface and Logging

- RVVI-TRACE data is used for functional coverage sampling (trace2cov) and to produce detailed logfiles for debug (trace2log)
ImperasDV + RVVI: Process

- Instantiate VIP in a testbench
- Connect tracer using RVVI-TRACE i/f
- DUT and reference model run the same program
- Retire, trap events communicated over RVVI
- Internal state continuously compared
- RVVI-TRACE monitored for async events
- Predictive engine verifies legal scenarios
ImperasDV using RVVI

• Pros:
  • Checks full machine state at every event
  • Sequence checking is done for you
  • Errors are flagged immediately, and in detail
  • Finds synchronous and asynchronous bugs
  • Reusable across different core DV projects
  • Interchangeable due to standard interface (RVVI)
  • Ease of use
  • Training, documentation, and support

• Cons:
  • Cost of VIP licenses
Agenda

• Introduction to Imperas
• Introduction to RISC-V
• RISC-V processor verification challenges
  • Why is RISC-V processor DV so critical?
• RISC-V processor verification environment components
• RISC-V Verification approaches
• RISC-V Verification standards
• RISC-V Verification IP

• Functional coverage for RISC-V processors
  • Verification Case studies
    • OpenHW Group CV32E40X processor
    • Wally RISC-V processor
• Summary
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
RISC-V Instructions (Standard ISA Architectural Feature)

• There are many different instructions in the RV64 extensions:
  • Integer: 56, Maths: 13, Compressed: 30, FP-Single: 30, FP-Double: 32
  • Vector: 356, Bitmanip: 47, Krypto-scalar: 85
  • P-DSP: 318
  • For RV64 that is ~1,000 instructions...

• Each instruction needs SystemVerilog covergroups and coverpoints
  • 10-200+ lines of SystemVerilog for each instruction

• 10,000-100,000++ lines of code to be written
  • Not design or core specific
Machine-generated Functional Coverage

RISC-V privilege and un-privilege ISA machine readable definition

Functions to convert RVVI-TRACE to Functional Coverage structures

Functional Coverage sampling

Hand coded SystemVerilog

RVVI-TRACE

clk

Imperas RVFCgen

Configuration:

xlen, csr, csrFields, compliance, DV extensions, options, ...

Generated SystemVerilog source

Hand coded SystemVerilog

Imperas RVFCgen

Configuration:

xlen, csr, csrFields, compliance, DV extensions, options, ...

Generated SystemVerilog source

Imperas RVFCgen

Configuration:

xlen, csr, csrFields, compliance, DV extensions, options, ...

Generated SystemVerilog source

Imperas RVFCgen

Configuration:

xlen, csr, csrFields, compliance, DV extensions, options, ...

Generated SystemVerilog source

Imperas RVFCgen

Configuration:

xlen, csr, csrFields, compliance, DV extensions, options, ...

Generated SystemVerilog source
riscvISACOV
https://github.com/riscv-verification/riscvISACOV

- Machine-generated functional coverage code for the RISC-V ISA
  Feb. 2023 status:
  - Extensions covered: 53
  - Instructions covered: 559
  - Covergroups: 559
  - Coverpoints: 5036
- Well documented in markdown
- Includes verification plan information in csv format
- RV32I extension available open source under Apache
- Other extensions available under Imperas Proprietary license
riscvISACOV: Coverage levels

• Compliance basic
  • Essential items to be covered
  • e.g. number of times instruction is executed, register values

• Compliance extended
  • Cross coverage using basic coverpoints
  • e.g. cross floating point register values with rounding modes

• DV Unprivileged basic
  • Essential and cross coverage involving unprivileged mode items
  • e.g. FPU special values for registers

(there are also 3 more comprehensive DV levels - WIP)
riscVISAOCOV: Documentation and VPlans

- Auto-generated documentation and csv files for inclusion in Verification Plans

---

**riscVISAOCOV: RISC-V SystemVerilog Functional Coverage: RV32I**

ISA: Extension: RV32I
Specification: I Base Integer Instruction Set
Version: 2.1
XLEN: 32
Instructions: 37
Covergroups: 37
Coverage: Total: 438
Coverage Compliance: Basic: 204
Coverage Compliance: Extended: 234

<table>
<thead>
<tr>
<th>Extension</th>
<th>Subset</th>
<th>Instruction</th>
<th>Covergroup</th>
<th>Coverpoint</th>
<th>Coverage Description</th>
<th>Coverage Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>RV32I</td>
<td>addi</td>
<td>addi_op</td>
<td></td>
<td>cp_axm_count</td>
<td>Number of times instruction is executed</td>
<td>Compliance Basic</td>
</tr>
<tr>
<td></td>
<td>cp_r1d</td>
<td>r1d_g</td>
<td></td>
<td></td>
<td>RD (GPR) register assignment</td>
<td>Compliance Basic</td>
</tr>
<tr>
<td></td>
<td>cp_rs1</td>
<td>rs1_g</td>
<td></td>
<td></td>
<td>RS1 (GPR) register assignment</td>
<td>Compliance Basic</td>
</tr>
<tr>
<td></td>
<td>cp_rs2</td>
<td>rs2_g</td>
<td></td>
<td></td>
<td>RS2 (GPR) register assignment</td>
<td>Compliance Basic</td>
</tr>
</tbody>
</table>

---

**riscVISAOCOV: D Standard Extension for Double-Precision Floating-Point, 2.2**

xlen, 64

extension, subset, instruction, description, covergroup, coverpoint, description, coverage level, pass/fail criteria, test type, coverpoint, flush, spurious, round, trapping, capture, precision.

riscVISAOCOV: Documentation and VPlans

- Auto-generated documentation and csv files for inclusion in Verification Plans

---

**riscVISAOCOV: D Standard Extension for Double-Precision Floating-Point, 2.2**

xlen, 64

extension, subset, instruction, description, covergroup, coverpoint, description, coverage level, pass/fail criteria, test type, coverpoint, flush, spurious, round, trapping, capture, precision.
Functional Coverage Examples

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

• OpenHW Group core-v-verif
  • https://github.com/openhwgroup/core-v-verif/tree/master/cv32e40s/env/uvmecov
Agenda

• Introduction to Imperas
• Introduction to RISC-V
• RISC-V processor verification challenges
  • Why is RISC-V processor DV so critical?
• RISC-V processor verification environment components
• RISC-V Verification approaches
• RISC-V Verification standards
• RISC-V Verification IP
• Functional coverage for RISC-V processors

• Verification Case studies
  • OpenHW Group CV32E40X processor
  • Wally RISC-V processor

• Summary
Verification Case Study – CV32E40X

• OpenHW Group CV32E40X RISC-V core
  • 4 stage pipeline, embedded class core:
    • RV32I, RV32E
      ZicsrZifence
    • X interface
  • Evolved from work on the CV32E40P core (originated from Pulp platform)
    • Focus of OpenHW Group is high-quality cores verified to industry standards
  • CORE-V-VERIF environment modified to use ImperasDV in fall 2022
CORE-V-VERIF using ImperasDV
Demonstration

• DUT: OpenHW Group CV32E40X RISC-V processor
  • Simulation: passing test
  • Simulation: failing test
  • Simulation: asynchronous event bug

• Screenshots from the video demonstration now follow
ImperasDV for RISC-V processor verification

Test bench
- Configures RTL and ImperasDV
- Loads test programs
- Steps DUT
- Reports results
- Requires "tracer" interface from RTL DUT to RVI

RISC-V Core
- RTL (DUT)
- SystemVerilog

Test bench can be SystemVerilog C or C++

RVI is open standard RISC-V Verification Interface for re-use of reference models, functional coverage, and other test bench VIP

© Imperas Software Ltd.
## Terminal Output

```
<table>
<thead>
<tr>
<th>Info</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMV</td>
<td>ImperasDV VERIFICATION REPORT</td>
</tr>
<tr>
<td>IMV</td>
<td>Instruction retire: 10,337</td>
</tr>
<tr>
<td>IMV</td>
<td>Traps: 0</td>
</tr>
<tr>
<td>IMV</td>
<td>Interrupt events: 36</td>
</tr>
<tr>
<td>IMV</td>
<td>Ending cycle count: 37,253</td>
</tr>
<tr>
<td>IMV</td>
<td>PC: 19,337 / 19,337</td>
</tr>
<tr>
<td>IMV</td>
<td>Instruction: 19,337 / 19,337</td>
</tr>
<tr>
<td>IMV</td>
<td>GPR: 19,337 / 618,784</td>
</tr>
<tr>
<td>IMV</td>
<td>CSR: 75,329 / 3,460,312</td>
</tr>
<tr>
<td>IMV</td>
<td>FPR: 0 / 0</td>
</tr>
<tr>
<td>IMV</td>
<td>VR: 0 / 0</td>
</tr>
<tr>
<td>IMV</td>
<td>Total compares: 4,060,770</td>
</tr>
<tr>
<td>IMV</td>
<td>Mismatches: 5</td>
</tr>
<tr>
<td>IMV</td>
<td>Instruction diverged from DUT: 5</td>
</tr>
<tr>
<td>IMV</td>
<td>Warnings: 5</td>
</tr>
</tbody>
</table>

During execution, the reference model was forced to match DUT state which should be treated as a test failure.

This feature is only provided to help expose further issues causing the DUT and reference to diverge.

CPU 'refRoot/cpu' STATISTICS:
- Type: riscv (CV32V4MX)
- Final program counter: 0x3184
- Simulated instructions: 19,332

SIMULATION TIME STATISTICS:
- Simulated time: 0.00 seconds
- User time: 0.00 seconds
- System time: 0.04 seconds
- Elapsed time: 7.49 seconds

ImperasDVasync finished: Fri Dec 9 19:19:26 2022
```
VIDEO: Asynchronous

• 4:38

Interrupt taken, but which one?
Info (ID) Target 'refRoot/cpus' has object file read from '/home/moore/Demo/ImperasaDV/CV2/EX4XX scripts/work/interrupt_test.sif'

Info (ID) Program Headers:

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>VirtAddr</th>
<th>PhysAddr</th>
<th>FileSize</th>
<th>MemSize</th>
<th>Flags</th>
<th>Align</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOADE</td>
<td>0x00000000</td>
<td>0x00008010</td>
<td>0x00008010</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>LOAD</td>
<td>0x00000000</td>
<td>0x00008010</td>
<td>0x00008010</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>LOAD</td>
<td>0x00000000</td>
<td>0x00008010</td>
<td>0x00008010</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>LOAD</td>
<td>0x00000000</td>
<td>0x00008010</td>
<td>0x00008010</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

ImperasaDV INITIALIZED

Info (ID) Program: /home/moore/Demo/ImperasaDV/CV2/EX4XX scripts/work/interrupt_test.sif

Info (ID) Vendor: openlogp.org

Info (ID) Variant: CV2/EX4X

Info (ID) Max net latency: 0 instructions

[cv2/EX4X Coreну]: RUM.MAPCOUNTS = 1

NOTE: cv2/EX4X testsbench_mem_ram_i.c configure.stalls = 0; OB stalls not enabled

TEST 1 - TRIGGER ALL IRQS IN SEQUENCE:

Error (ID) The reference model is unable to find a legal set of actions to converge to the provided DUT state.

Info (ID) Currently applied nets:

Info (ID) - MSGInterrupt => 1

Info (ID) - FLISRInterrupt => 1

Info (ID) - MExternalInterrupt => 1

Info (ID) - Pending net changes:

Info (ID) - Group 0

Info (ID) - MSGInterrupt => 0, age = 3 cycles

Info (ID) - DUT state change (hartid = 0):

Info (ID) - dut.pc = 0x00000000

Info (ID) - dut.v = 0x00000000

Info (ID) - dut.status = 0x00000000

Info (ID) - dut.asize = 0x00000000

Info (ID) - dut.cause = 0x00000000

Info (ID) - dut.cycle = 0x00000000

Info (ID) - dut.insret = 0x00000000

Info (ID) - dut.insret = 0x00000000

Info (ID) - dut.insret = 0x00000000

Info (ID) - dut.insret = 0x00000000

Info (ID) - Evaluated progression tree:

Warning (ID) The reference is being forced to converge to the DUT state. Results that follow should be treated with caution.
Verification Case Study – HMC/OSU Wally

• Overview of the core
• Testbench with RVVI, ImperasDV
• Demonstration runs
• Current status
Verification Case Studies

• 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 is high quality core for processor architecture education
  • Status in January 2023 – before starting to use ImperasDV for verification:
    • passing all RISC-V International compliance tests, Imperas compatibility tests
      • Using Compliance Level post sim signature file compare
    • boots Linux
  • now in OpenHW as CORE-V Wally (https://github.com/openhwgroup/cvw)
Wally + ImperasDV

- RVVI Tracer: 1/2 day of effort
- Testbench: 1/2 day of integration
- 2 days effort resolve tracer/integration issues
Wally: RVVI, ImperasDV: base use model: verification
Wally: RVVI, ImperasDV: verification with coverage

Simulation
memory

Simulation
control

Testbench

RVVI-API

ImperasDV

RISC-V
Reference
Model

Imperas RISC-V
Verification IP

riscvISACOV

Imperas RISC-V
Functional Coverage

Wally SoC

Wally RISC-V
Core
RTL
(DUT)

program.elf

Pass/Fail

coverage
Wally: RVVI, ImperasDV: verification with compliance suite & merged coverage
Wally: RVVI, ImperasDV: verification with compliance suites & Google riscv-dv ISG & merged coverage
Wally: RVVI, ImperasDV: verification with compliance suites & Google riscv-dv ISG & directed tests & merged coverage
Wally + RVVI + ImperasDV – Status (Feb. 2023)

- RVVI Tracer: 1/2 day of effort
- Testbench: 1/2 day of integration
- 2 days effort resolve tracer/integration
- Results:
  - 20+ bugs found almost immediately
  - With improving functional coverage analysis
- Stimulus: riscv-dv
Agenda

• Introduction to Imperas
• Introduction to RISC-V
• RISC-V processor verification challenges
  • Why is RISC-V processor DV so critical?
• RISC-V processor verification environment components
• RISC-V Verification approaches
• RISC-V Verification standards
• RISC-V Verification IP
• Functional coverage for RISC-V processors
• Verification Case studies
  • OpenHW Group CV32E40X processor
  • Wally RISC-V processor

• Summary
Summary

• Processor verification requires unique approaches to ensure the quality of the processor IP
• The verification method chosen will impact the processor’s quality
• Open standards such as RVVI permit efficiency, reuse, and development of RISC-V processor VIP
• The RISC-V ISA is an excellent application for machine-generated functional coverage (e.g. riscvISACOV)
• ImperasDV RISC-V VIP enables a comprehensive processor DV environment that works out of the box
Questions

• Thank you

• Aimee Sutton  aimees@imperas.com
• Lee Moore  moore@imperas.com
• Simon Davidmann  simond@imperas.com