RISC-V Processor Verification: Case Study

DVCon 2021

A. Maymon, S. Harari
Nvidia Networking

L. Moore, L. Lapides
Imperas Software Ltd.
Agenda

- RISC-V processor DV problem
- Verification flow overview
- RISC-V reference models
- Simple step-and-compare flow
- Complex step-and-compare flow
- Results
- Conclusion
Agenda

• RISC-V processor DV problem
• Verification flow overview
• RISC-V reference models
• Simple step-and-compare flow
• Complex step-and-compare flow
• Results
• Conclusion
## RISC-V Progression

<table>
<thead>
<tr>
<th>Year Range</th>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-2016</td>
<td>ISA definition, test chips</td>
<td>tests</td>
</tr>
<tr>
<td></td>
<td>Hardware</td>
<td>Software</td>
</tr>
<tr>
<td>2017-2018</td>
<td>Proof of concept SoCs; “minion” processors for power management, communications, ...</td>
<td>Bare metal software</td>
</tr>
<tr>
<td>2019-2020</td>
<td>IoT SoCs; MCUs</td>
<td>RTOS, firmware</td>
</tr>
<tr>
<td>2021-</td>
<td>Application processors; AI SoCs</td>
<td>Linux, drivers; AI compilers</td>
</tr>
</tbody>
</table>

Increasing hardware and software complexity requires use of best known methods for processor and SoC architecture, implementation, design verification, software development.
The RISC-V Processor Design Verification (DV) Problem

- Arm processor IP
  - \( \sim 10^{15} \) verification cycles per processor (10,000 simulators running constantly for 1 year)
  - Verification of interface between NoC and processor
  - 1,000s of SoC designs successfully produced
- Similar stories for ARC, MIPS, Tensilica, ...

- RISC-V IP questions
  - How well verified is an individual processor (from processor IP vendor, open source, self-built)?
  - How well verified is interface between NoC and processor?
  - How to deal with custom instructions?
Agenda

• RISC-V processor DV problem
• Verification flow overview
• RISC-V reference models
• Simple step-and-compare flow
• Complex step-and-compare flow
• Results
• Conclusion
Previous Work on RISC-V DV Uses Trace Comparison Flow

- 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

- This flow supports only simple instruction test; cannot support asynchronous events including interrupts and Debug mode
- Trace compare is done post-simulation => easy to get started however inefficient use of simulation resources

- Google Cloud
- Open Source SystemVerilog UVM RISC-V Instruction Stream Generator
- RISCV.S
- GCC/LLVM
- RISCV.elf
- RISC-V RTL & memory
- Imperas ISS (cpu+memory)
- Metrics.log
- Open Source SystemVerilog UVM RISC-V Functional Coverage
- Imperas add Vectors (~500) Bitmanip (~100)
Some Previous Work Used
Step-and-Compare DV Flow

- OVP RISC-V 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
Step-and-Compare Requires
Encapsulation of the Reference Model
in SystemVerilog

- The OVP model is a binary shared object of a RISC-V CPU model
- Encapsulated into a SystemVerilog module, using SystemVerilog DPI
- Instanced in SystemVerilog testbench like any module
Parallel Verification Flows Were Used for this Project

- Simple Step-and-Compare
  - Cadence Xcelium RTL simulator
  - Cadence Specman verification environment
  - Imperas riscvOVPsim instruction set simulator (RISC-V reference model and simulator combined)
  - Used for verification of basic instruction functionality

- Complex Step-and-Compare
  - Cadence Xcelium RTL simulator
  - Cadence Specman verification environment
  - Open Virtual Platforms (OVP) RISC-V reference model, including support for custom instructions
  - Imperas M*DEV simulator
  - Used for verification of asynchronous events, Debug mode, ...

Using parallel flows achieved optimal use of verification resources at the lowest cost.
Agenda

• RISC-V processor DV problem
• Verification flow overview
• RISC-V reference models
  • Simple step-and-compare flow
  • Complex step-and-compare flow
• Results
• Conclusion
RISC-V Reference Model Requirements

- Model the ISA, including all versions of the ratified spec, and stable unratified extensions
- Easily update and configure the model for the next project
- User-extendable for custom instructions, registers, interrupts/events, ...
- Model actual processor IP, e.g. Andes, SiFive, OpenHW CV32E40P, SweRV, ...
- Well-defined test process 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.
- Interface to architecture exploration tools including extensibility to timing estimation

- Most RISC-V ISSs can meet one or two of these requirements
- Imperas models and simulators were built to satisfy these requirements, and matured through usage on non-RISC-V ISAs over the last 10+ years
Components of Open Virtual Platforms (OVP) Fast Processor Models

OVP models are open source and free

- Models are built in C using OVP APIs; APIs are supported by OVPsim and Imperas simulators
- All models have both C and SystemC/TLM2 native interfaces
- Available under the Apache 2.0 open source license
- Require an Imperas simulator license to run
- 1 simulator license is all that is needed for multi-core and many-core platforms

http://www.ovpworld.org/info_riscv
OVP Library of RISC-V Fast Processor Models

- Existing Imperas Open Virtual Platforms (OVP) Fast Processor Models of ...
  - Generic or envelope models of RV32/64 IMAFDCEVBHK M/S/U privilege modes
  - Andes cores: A(X)25, N(X)25, N(X)25F, 27-series including NX27V, ...
  - SiFive cores: SiFive Series 2, Series 3 (e.g. E31), Series 5 (e.g. E51, U54), Series 7
  - OpenHW CV32E40P

- Custom features – instructions, registers, interrupts/events – easily added by user or by Imperas
  - New features are added in side file so as not to perturb the verified model
  - Custom instructions can be analyzed for effectiveness using instruction coverage, profiling tools
  - Custom interrupts and events can be added to all spec flows and CSRs with priority and order consideration

- 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
  - Mutation testing used to check quality of test suites
  - Additional testing by processor IP vendors to validate models
RISC-V OVP Reference Model
Configurability

- Industrial quality, free ISS / reference model for instruction testing
  - [https://www.ovpworld.org/info_riscv](https://www.ovpworld.org/info_riscv)
- Model is built using Open Virtual Platforms (OVP) APIs
- Implements full RISC-V envelope
  - Configurable for all features and spec versions

<table>
<thead>
<tr>
<th>Parameter Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification version</td>
<td>Privilege spec version 1.10, 1.11, 1.12</td>
</tr>
<tr>
<td></td>
<td>Debug configuration spec version 0.13.4, 0.14</td>
</tr>
<tr>
<td></td>
<td>Bit manipulation spec version 0.90, 0.91, 0.92, 0.93</td>
</tr>
<tr>
<td></td>
<td>Vector spec version 0.71, 0.8, 0.9, 1.0</td>
</tr>
<tr>
<td>Simple parameters</td>
<td>MISA subsets 32/64 I, M, A, C, F, D, B, V, H, K, …</td>
</tr>
<tr>
<td></td>
<td>Misaligned Code/Data access behavior</td>
</tr>
<tr>
<td></td>
<td>CSR field/behavior configuration</td>
</tr>
<tr>
<td></td>
<td>CLINT configuration</td>
</tr>
<tr>
<td></td>
<td>CLIC configuration</td>
</tr>
<tr>
<td></td>
<td>DEBUG configuration</td>
</tr>
</tbody>
</table>
Flow to Add Custom Instructions

Characterize C Application
- Instruction Accurate Simulation
- Trace / Debug
- Timing Simulation
- Function Timing / Profiling

Develop New Custom Instructions
- Design Instructions
- Add to Application
- Add to Model
- Add Timing

Characterize New Instructions in Application
- Instruction Accurate Simulation
- Trace / Debug
- Timing Simulation
- Function Timing / Profiling

Release & Deploy
- Check RISC-V Compliance
- Use as reference for RTL Design Verification
- Use in Imperas/OVP Platforms, EPKs
  - Heterogeneous / Homogeneous
  - Multi-core, Many-core
- Imperas Multi-Processor Debug, VAP tools
- Port OS, RTOS (Linux, FreeRTOS...)
- Use in many simulation envs (incl. SystemC)
- Deliver to end users

Optimize & Document model
- Instruction Coverage
- Line Coverage
- Instruction Performance
- Generate PDF model doc
Adding Custom Instructions to Model Using Extension Library

- Use standard Open Virtual Platforms (OVP) instruction modeling APIs to add new instructions (and optional state) as *new extension library*
- Easy to extend decode table, add efficient behavioral JIT code
- Compile and link model extension library
New custom instructions, new additional state registers

New custom instructions in trace disassembly
Agenda

• RISC-V processor DV problem
• Verification flow overview
• RISC-V reference models
• Simple step-and-compare flow
• Complex step-and-compare flow
• Results
• Conclusion
Simple Step-and-Compare Flow

- Goal is to test basic instructions
- riscVOPsim reference simulator is run separately from RTL simulator
  - Results are embedded in the Specman scoreboard
  - Tests are then self-checking
- Provides partial testing of multi-hart as tests and expected results can be generated on a per-hart basis
- Similar to flow with Google riscv-dv shown earlier
  - Specman test environment with scoreboard used instead of post-simulation trace compare
  - Added support for bit manipulation instructions to riscv-dv
riscvOVPsim Reference Model

- Industrial quality, free ISS / reference model for instruction testing
  - https://www.ovpworld.org/info_riscv
- Model is built using Open Virtual Platforms (OVP) APIs
- Implements full RISC-V envelope
  - Configurable for all features and spec versions
Adding Support for Bit Manipulation Instructions to riscv-dv Instruction Stream Generator

- Add support for instructions
- Add support for test generation

```verbatim
function bit [6:0] get_opcode();
case (instr_name) inside
    ANDN, ORN, XNOR, GORC, SLO, SRO, ROL, ROR, SBCLR, SBSET, SBINV, SBEXT,
    GREV: get_opcode = 7'b0110011;
    SLOI, SROI, RORI, SBCLRI, SBSETI, SBINVI, SBEXTI, GORCI, GREVI, CMIX, CMOV,
    FSL: get_opcode = 7'b0010011;
...
default: get_opcode = super.get_opcode();
endcase
endfunction
```

```verbatim
class riscv_b_instr extends riscv_instr;
rand riscv_reg_t rs3;
bit has_rs3 = 1'b0;
`

function new(string name = "");
    super.new(name);
endfunction
```

```verbatim
virtual function void set_rand_mode();
    super.set_rand_mode();
    has_rs3 = 1'b0;
    case (format) inside
        R_FORMAT: begin
            if (instr_name inside {CLZW, CTZW, PCNTW, SEXT_B, SEXT_H, CLZ, CTZ, PCNT, BMATFLIP,
                CRC32_B, CRC32_H, CRC32_W, CRC32C_B, CRC32C_H, CRC32C_W, CRC32_D,
                CRC32C_D}) begin
                has_rs2 = 1'b0;
            end
        end
        R4_FORMAT: begin
            has_imm = 1'b0;
            has_rs3 = 1'b1;
        end
        I_FORMAT: begin
            has_rs2 = 1'b0;
            if (instr_name inside {FSRI, FSRIW}) begin
                has_rs3 = 1'b1;
            end
        end
    endcase
endfunction
```
Agenda

- RISC-V processor DV problem
- Verification flow overview
- RISC-V reference models
- Simple step-and-compare flow
  - Complex step-and-compare flow
- Results
- Conclusion
• Reference model and simulator (Imperas M*DEV simulator) are run in parallel with RTL simulator (Cadence Xcelium)
• Reference model and RTL Device Under Test (DUT) are stepped in parallel
• This flow enables sync of the RTL DUT and reference model for interrupts and both asynchronous and synchronous events
• Effectively the same as the SystemVerilog encapsulation
Interactive Co-Debug is Enabled by the Step-and-Compare Environment

Interactive co-debug, especially valuable for asynchronous events and to investigate compare failures.

Cadence Xcelium
SystemVerilog simulation

Imperas M*SDK
RISC-V reference model simulation
Agenda

• RISC-V processor DV problem
• Verification flow overview
• RISC-V reference models
• Simple step-and-compare flow
• Complex step-and-compare flow
• Results
• Conclusion
RISC-V Processor IP

- RV64IMACBNSU
- 64-bit RISC-V core with extensions including
  - Integer
  - Multiply
  - Atomic
  - Compressed
  - Bit manipulation
  - Machine/supervisor/user modes
  - Debug mode
- Custom instructions
- Custom interrupt/event support (e.g. error events)
Instruction Functional Coverage
Results from Simple Flow

ISA coverage report for IMACB plus custom commands and non-supported spec (e.g. F as an illegal command)

ISA coverage report for branch jump sequences: loops, backward, forward branches.

ISA coverage report for multicycle sequential insertion.
### Function Coverage Results

<table>
<thead>
<tr>
<th>Average Grade</th>
<th>Covered Grade</th>
<th>Goal</th>
<th>Weight</th>
<th>Uncovered Bins</th>
<th>Excluded Bins</th>
<th>Total Bins</th>
<th>Item</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>62.50%</td>
<td>62.50% (10/16)</td>
<td>n/a</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>16</td>
<td>CoverPoint</td>
<td>CoverMemReq.RequestAddress</td>
</tr>
<tr>
<td>100.00%</td>
<td>100.00% (3/3)</td>
<td>n/a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>CoverPoint</td>
<td>CoverMemReq.RequestOpcode</td>
</tr>
<tr>
<td>100.00%</td>
<td>100.00% (16/16)</td>
<td>n/a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>CoverPoint</td>
<td>CoverMemReq.RequestThreadId</td>
</tr>
<tr>
<td>100.00%</td>
<td>100.00% (3/3)</td>
<td>n/a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>CoverPoint</td>
<td>CoverMemReq.RequestAgent</td>
</tr>
<tr>
<td>71.43%</td>
<td>71.43% (5/7)</td>
<td>n/a</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>CoverPoint</td>
<td>CoverMemReq.RequestSize</td>
</tr>
<tr>
<td>100.00%</td>
<td>100.00% (128/128)</td>
<td>n/a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>128</td>
<td>CoverPoint</td>
<td>CoverMemReq.RequestData</td>
</tr>
<tr>
<td>100.00%</td>
<td>100.00% (16/16)</td>
<td>n/a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>CoverPoint</td>
<td>CoverMemReq.RequestWriteByteEnable</td>
</tr>
<tr>
<td>100.00%</td>
<td>100.00% (48/48)</td>
<td>n/a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>48</td>
<td>Cross</td>
<td>CoverMemReq.Cross__RequestThreadId__RequestAgent</td>
</tr>
</tbody>
</table>

- Functional coverage results from complex flow with step-and-compare
Agenda

• RISC-V processor DV problem
• Verification flow overview
• RISC-V reference models
• Simple step-and-compare flow
• Complex step-and-compare flow
• Results
• Conclusion
Conclusions

- Robust, comprehensive, best-known-methods approach is needed for RISC-V processor DV
- Step-and-compare methodology is necessary for DV of asynchronous events
- Key pieces of the environment include
  - High quality RISC-V reference model
  - Ability to introspect from the testbench into both the RTL DUT and the reference model
  - Functional coverage metrics are important, as always with DV
Thank you