UVM-based RISC-V processor verification platform

Tao Liu, Richard Ho, Udi Jonnalagadda
Agenda

- Motivation
- What makes a good instruction generator
- Random instruction generation flow
- RTL and ISS co-simulation flow
- Benchmark
- Future work
Open source RISC-V processor verification solutions

**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.

Verification is one of the key challenges of modern processor development.
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.
Why SV/UVM

**SystemVerilog (SV)**
Most popular verification language, provides great features like constrained random, coverage groups etc.

**Universal Verification Methodology (UVM)**
Most prevalent verification framework in the industry

We want to build something with the industry standard verification language and framework which most DV engineers can easily understand and extend.
What makes a good instruction generator

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

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

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

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

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

**Sequence level randomization**
Maximize the possibility of instruction orders and dependencies

**Program level randomization**
Random privileged mode setting, page table organization, program calls
Instruction randomization

Easy part

**Arithmetic**: ADD, SUB, LUI, MUL, DIV ...
**Shift**: SLLI, SRL, SRLI, SRAI ...
**Logical**: XOR, OR, AND, ANDI ...
**Compare**: SLTI, SLT, SLTU ...
**Others**: FENCE, SFENCE, EBREAK ...

Randomize each instruction individually with bias towards corner cases. (overflow, underflow, compressed instruction)

Tricky part

**Branch / jump instruction**
- Need a valid branch/jump target
- Avoid infinite loop

**Load/store/jump instruction**
- Need an additional instruction to setup the base address
- The calculated address should be a valid location

**CSR instruction**
- Avoid randomly changing the privileged state
- Result checking could be a challenge as the privileged CSR behavior could be implementation-specific.
Load/store instruction generation

Basic load/store instruction

A basic load/store instruction needs additional instruction to setup the base address (rs1)

```
lw x11, x10, 20
```
Forward branch
Randomly pick a forward target
Avoid step into the atomic instruction stream

Backward branch
A dedicated instruction stream to properly setup loop structure, make sure the loop exit condition can be triggered

- Init loop counter register
- Init loop limit register
- ... ...
- Update loop counter register
- ... ...
- Backward branch instruction
Call stack randomization

Avoid loop function call

Generate call stack in a tree structure

Avoid loop function call

Generate call stack in a tree structure

Main

Sub1
Sub2
Sub3
Sub4
Sub5
Sub6
Sub7
Subn
Page table randomization

Example: SV39 page table randomization (with exception injection)
Architecture aware

01 Branch prediction

02 MMU (TLB, Cache etc)
Architecture aware

03 Issue, execute, commit

It's not just a random stream of instructions, it should be designed to effectively verify the architectural features of the processor.
Generator flow

- Generate program header
  - Privileged mode setup
  - Page table randomization
  - Initialization routine
  - Generate main/sub programs
  - Branch target assignment

- Generate data/stack section
  - Generate page tables
  - Generate intr/trap handler
  - Test completion section
  - Call stack randomization
  - Apply directed instructions
All instructions and data are located in continuous physical address space, and are mapped to the virtual address space through page table.
RTL & ISS co-simulation flow

- UVM random instruction generator
- RISC-V assembly
- RISC-V compiler
- Link script
- RISC-V ISS (spike)
- ISS sim log
- Log compare
- RISC-V Processor DUT
- Memory model
- RTL sim log
- RTL simulation environment
- objcopy
- RISC-V plain binary
Complete feature list

**Supported ISA**
RV32IMC, RV64IMC

**Supported privileged mode**
User mode, supervisor mode, machine mode

**Supported spec version**
User level spec 2.20, privileged mode spec 1.10

**Supported RTL simulator**
VCS, Incisive

#### Test suite
- Basic arithmetic instruction test
- Random instruction test
- MMU stress test
- HW/SW interrupt test
- Page table exception test
- Branch/jump instruction stress test
- Interrupt/trap delegation test
- Privileged CSR test
Benchmark flow

**Processor candidates**

**Pulpino RI5CY:**
4 stages, RV32-IMC, DSP extension

**Pulpino Ariane:**
6-stage, RV64-IMC, single issue, in-order, support M/S/U privileged levels

**Merlin:**
Open Source RV32I[C] CPU

**ISS simulator**

**Spike**

**Benchmark metrics**

Bug hunting capability, test coverage
Flow integration effort, performance
Cache line access racing

Bugs found

Load
Store
Load
Load

PT 0
PT 1
PT 2
PT 3
PT 4

 Load
 Branch
 Add
 ... Mult

 LSU
 ALU
 ... Multiplier

Invalid branch execution

SSTATUS
wr
rd
MSTATUS
mxr
...

FENCE operation failure

PT 1
PT 2
PT 3
PT 4

Privileged CSR access

Multiplier

Incorrect branch execution

ALU corner case bug

Google Cloud
Build it together.

It’s just the beginning ...
More instruction extensions support F/V/A
Performance verification suite
Security verification
Coverage model
...

Planned release date: 01/2019
Please sign up Google group “riscv-dv” for further update
https://groups.google.com/forum/#!forum/riscv-dv
Reference

- riscv-tests
- riscv-torture
- Ariane core specification
- RI5CY core specification
- Merline core specification
- UVM (Universal Verification Methodology)