TestRIG: Randomized Testing of RISC-V CPUs using Direct Instruction Injection

Attending authors: Alexandre Joannou[1], Peter Rugg[1], Jonathan Woodruff[1], Franz A. Fuchs[1], Simon W. Moore[1]
1 - University of Cambridge
peter.rugg@cl.cam.ac.uk
Project URL: www.github.com/CTSRD-CHERI/TestRIG

UNIVERSITY OF CAMBRIDGE
Computer Science & Technology

TestRIG

- TestRIG (Testing with Random Instruction Generation) is a testing framework for RISC-V implementations.
- We propose model-based verification: comparing execution between a design-under-test and a model implementation (e.g. Sail RISC-V).
- We use the RISC-V Formal Interface (RVFI) to observe the state changes after each instruction, and novel Direct Instruction Injection (DII) for test injection.
- Growing ecosystem: participants benefit from shared model and test-generation work by adopting the interface, regardless of implementation language.
- Integrated Sail RISC-V golden model, Piccolo, Flute, Toooba, RVBS (Bluespec), Ibex (Verilog), QEMU, and Spike (C).
- Standard interface libraries in a range of languages.

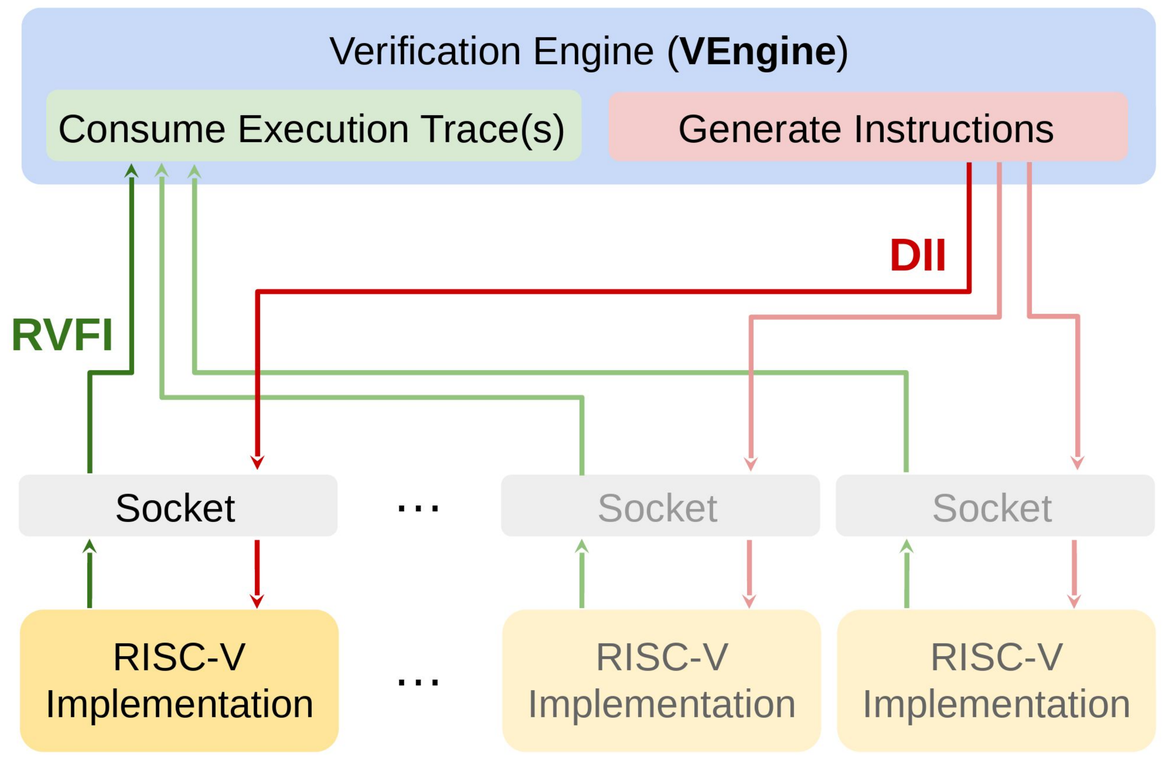

Figure 1. An illustration of the TestRIG ecosystem with a Verification Engine communicating with any two RISC-V implementations over sockets. The VEngine injects instruction sequences and compares the execution traces until it finds a divergence.

RISC-V Formal Interface (RVFI)

- RVFI is an existing trace format for formal verification using symbolic instructions.
- Exposes architectural signals such as the instruction encoding and memory and writeback information.
- An RVFI interface exports internal signals of an RTL design, or internal variables of a simulator or emulator.
- For more complex RTL designs, such as out-of-order microarchitectures, extracting the values may require preserving state for an RVFI report in a commit stage that could not previously access them.
- Extending the superscalar Toooba core for RVFI-DII required two new records in the Reorder Buffer.

```
struct RVFI_DII_Execution {
  // ...
  Bit#(32) rvfi_insn; // Executed instr
  // Register writeback
  Bit#(8) rvfi_rd_addr;   // Written index
  Bit#(64) rvfi_rd_wdata; // Written value
  // Memory
  Bit#(64) rvfi_mem_addr;  // Address
  Bit#(64) rvfi_mem_wdata; // Written data
  Bit#(8) rvfi_mem_wmask;  // Byte enables
  Bit#(64) rvfi_mem_rdata; // Read data
  // ...
}
```

Listing 1. Definition of the RVFI type reported by RISC-V implementations. Not all fields are shown.

Direct Instruction Injection (DII)

- DII specifies the instructions expected in the output trace, separating instructions from memory addresses.
- Requires custom pipeline instrumentation, but enables greatly simplified sequence generation and shrinking.
- Must support resetting implementations between tests, including registers and memory.
- Implementations must rerun an instruction in the stream if it is flushed from the pipeline on mispredict.
- Interfaces provide a standard 8 MiB memory window.

```
struct RVFI_DII_Instruction {
  // Instruction or reset command?
  Bool rvfi_cmd;
  // Time to inject instruction
  Bit#(10) rvfi_time;
  // Instruction word (32/16 bit)
  Bit#(32) rvfi_insn;
}
```

Listing 2. Definition of the DII type exposed to the implementations expressing the injected instruction.

QuickCheck VEngine

- Leverages Haskell's *QuickCheck* library.
- Propositions passed into *QuickCheck* consist of a monadic function that issues instructions to both implementations and compares the reported results.
- *QuickCheck* attempts to disprove the proposition by generating test cases using our generator templates.
- On finding a counterexample, the trace is shrunk to find simpler test cases that still cause divergence.
- Trace serialisation enables a regression testing library without false positives when the specification is changed.
- Assertions ensure the intended behaviour is provoked.
- We have used QCVEngine to test our processors, including the CHERI RISC-V security extension.
- Eases per-instruction bringup.
- Found a security bug in capability compression code.
- It also accelerated debugging of a cache bug when switching Flute configurations (Listing 3).
- Enabled testing for speculative side-channels.

```
lui  x1, 0x40004
slli x1, x1, 1
lui x20, 0x40004 # Value used as data
ori  x3, x1, 1   # A page address
lui  x2, 0x40004
slli x2, x2, 1   # The same page address
lhu  x4, x3[1]   # Load from address
sh  x20, x2[2]   # Store to same address
lhu  x2, x3[1]   # Divergence on reloading
```

Listing 3. The shrunk counterexample produced by TestRIG to demonstrate a cache bug in the Flute processor: cache hits could return the line in the wrong way of a set.

Intelligent shrinking

- DII allows traces to be automatically shrunk without control flow completely changing the test semantics.
- *QuickCheck*'s list shrinking function automatically deletes instructions unnecessary for divergence.
- Additional shrinking strategies include detecting *bypass* instructions that act as register moves.
- Library of candidate shrinks for each RISC-V instruction.
- Resulting counterexamples are much easier to understand, improving debugging experience.

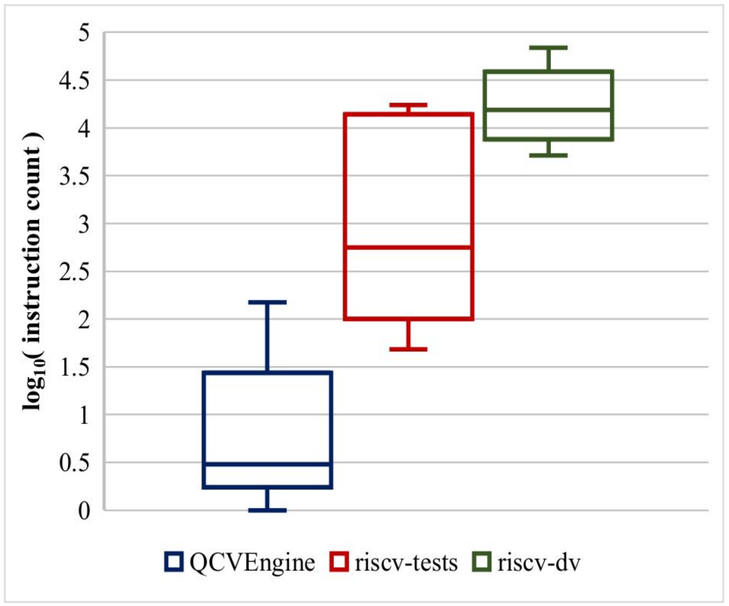

Figure 2. Size complexity of QCVEngine counterexamples for CHERI bringup compared to length of traditional tests.

Recursive templates

- Mechanism for directed-random test generation.
- Can specify static distribution of instructions.
- Use Haskell for *recursive templates* to reach *deep states* in the processor, e.g. setting up page-table mappings, switching privilege modes, etc.
- Compared to static tests, templates are general, allowing for varying addresses, register indices, and even sequences of instructions.
- They can also be composed arbitrarily.
- Our template library achieves good RISC-V architectural coverage (Sail RISC-V branch coverage), similar to the RISC-V test suite and RISCV-DV.

```
surroundWithMemAccess x = random do
  // Randomise fields
  value <- bits 12 // 12 random bits
  shift <- bits 6
  // ...
  return $  storeToAddress regAddr regData
                           offset value
                           shift
        <> x
        <> loadFromAddress regAddr
                           offset regData
```

Listing 4. An example Haskell template to surround a given template with accesses to the same memory address.

In collaboration with SRI International

Paper: Joannou, A., Rugg, P., Woodruff, J., Fuchs, F.A., Van der Maas, M., Naylor, M., Roe, M., Watson, R.N., Neumann, P.G. and Moore, S.W., 2023. Randomized Testing of RISC-V CPUs using Direct Instruction Injection. *IEEE Design & Test*.
This work was supported by DARPA (HR0011-18-C-0016 "ECATS") and Innovate UK ISCF DSbD (105694).