Machine-Readable Specifications of RISC-V ISA

M. Chupilko, A. Kamkin, A. Kotsynyak, A. Protsenko, S. Smolov, A. Tatarnikov

Ivannikov Institute for System Programming
Russian Academy of Sciences
Formalization Motivation

- **High-level synthesis (HLS)**
  - High-level HDL (Chisel, BSV, SystemC) → RTL (Verilog, VHDL)

- **Cross development tools and virtual platforms**
  - ADL (CodAL, nML) → simulator + compiler (Codasip, Imperas)

- **Functional and security verification**
  - ADL + scenarios → assembly (MicroTESK, Genesys-Pro, RAVEN)
  - HLS (Clash) + equivalence checking (Yosys, ABC, JasperGold)
  - Theorem provers and proof assistants (Isabelle/HOL, Coq, PVS)
  - ADL + SW code + properties (MicroTESK + Why3 + SMT)
Formalization Activities

• **Technical activities**
  - SW models: C/C++ (Spike, QEMU, RV8, gem5, Imperas, etc.)
  - ISA-centric ADL: nML (ISP RAS), CodAL (Codasip)
  - High-level HDL: Chisel, Bluespec SystemVerilog

• **Research activities**
  - Functional languages: Haskell (Bluespec, Galois, MIT)
  - ISA-centric ADL: L3 DSL (SRI), nML (ISP RAS)
  - Logic frameworks: Coq (MIT), HOL4 (SRI), Why3 (ISP RAS)
  - ... and possibly more (not familiar with all of them)
Open Specifications

1. [https://github.com/mit-plv/riscv-semantics](https://github.com/mit-plv/riscv-semantics) by MIT in Haskell
2. [https://github.com/cliffordwolf/riscv-formal](https://github.com/cliffordwolf/riscv-formal) by Clifford Wolf (Symbiotic EDA) in SystemVerilog
3. [https://github.com/samuelgruetter/riscv-coq](https://github.com/samuelgruetter/riscv-coq) by Samuel Gruetter (MIT) in Coq
4. [https://github.com/rsnikhil/RISCV_ISA_Formal_Spec_in_BSV](https://github.com/rsnikhil/RISCV_ISA_Formal_Spec_in_BSV) by Rishiyur Nikhil (Bluespec) in Bluespec SystemVerilog
5. [https://github.com/rsnikhil/RISCV-ISA-Spec](https://github.com/rsnikhil/RISCV-ISA-Spec) by Rishiyur Nikhil (Bluespec) in Haskell
6. [https://github.com/SRI-CSL/l3riscv](https://github.com/SRI-CSL/l3riscv) by Prashant Mundkur (SRI International) in L3
MicroTESK Framework

Specifications (nML)

Processor Model
- Metadata
- Simulator
- Coverage

Specifications Translator

Tool Builders

Symbolic Executor

Test Program Generator

Simulator (C/C++)

Specifications (WhyML)

Test Programs (ASM)

Ref. Simulator (QEMU)

https://forge.ispras.ru/projects/microtesk
nML Language Overview

```nml
define RV64I
  let XLEN = 64
  ...
define

let MEMORY_SIZE_IN_WORDS = 2 ** (XLEN - 2)
shared mem MEM[MEMORY_SIZE_IN_WORDS, WORD]

op add (rd: X, rs1: X, rs2: X)
syntax = format("add %s, %s, %s", rd.syntax, rs1.syntax, rs2.syntax)
image = format("0000000%s%s000%s0110011", rs2.image, rs1.image, rd.image)
action = {
  rd = rs1 + rs2;
}
```

```nml
mode X (i: card(5)) = XREG[i]
syntax = format("%s",
  if i==0 then ZERO().syntax
  elif i==1 then RA().syntax
  elif i==2 then SP().syntax
  elif i==3 then GP().syntax
  elif i==4 then TP().syntax
  elif i>=5 && i<=7 then
    Temp(coerce(card(3), i-5)).syntax
  elif i>=8 && i<=9 then
    Saved(coerce(card(4), i-8)).syntax
  elif i>=10 && i<=17 then
    Func(coerce(card(3), i-10)).syntax
  elif i>=18 && i<=27 then
    Saved(coerce(card(4), i-16)).syntax
  else
    Temp(coerce(card(3), i-25)).syntax
image = format("%5s", i)
```
### Specifications Characteristic

<table>
<thead>
<tr>
<th>Base</th>
<th>Extension</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RV32I</td>
<td>M</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>P</td>
</tr>
<tr>
<td>RV32E</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>V</td>
</tr>
<tr>
<td>RV64I</td>
<td>Q</td>
<td>J</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>RV128I</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

**Value**

<table>
<thead>
<tr>
<th>RISC-V Instruction Set Manual Vol. I: User-Level ISA (v. 2.2)</th>
<th>145 pages <em>(94 pages in chapters 2-18)</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>Extension</td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>RV32I M F C P</td>
<td>24 pages</td>
</tr>
<tr>
<td>RV32E A D B V</td>
<td>2 pages</td>
</tr>
</tbody>
</table>

- **# Specified Instructions**: 200+ instructions
- **ISA Specifications Size (nML)**: 4 500 LOC *(w/o comments)*
- **MMU Specification Size (MMU$\text{SL}$)**: 0 LOC *(unspecified)*
Application 1: Test Program Generation (1/3)

Design Under Test (RTL, FPGA)

Test Programs (ASM)
- `lui a0, 0xdead`
- `ori a0, a0, 0x0`
- `lui a1, 0xbeef`
- `ori a1, a1, 0xf`
- `add t0, a0, a1`
- `sub t1, a0, t1`
- `add t0, t0, t1`

Reference Simulator (C/C++)

Execution Traces (Tarmac)
- `0x2000: lui ...
- 0x2004: ori ...
- 0x2008: lui ...
- 0x200c: ori ...
- 0x2010: add ...
- 0x2014: sub ...
- 0x2018: bug ...

Trace Comparator (Python)

Execution Traces (Tarmac)
- `0x2000: lui ...
- 0x2004: ori ...
- 0x2008: lui ...
- 0x200c: ori ...
- 0x2010: add ...
- 0x2014: sub ...
- 0x2018: add ..."
Application 1: Test Program Generation (2/3)

MicroTESK Test Program Generator

Translator

Specifications (nML)

Processor Model

Autogen

Generation Core

Test Programs (Assembly Code)

lui a0, 0xdead
ori a0, a0, 0x0
lui a1, 0xbeef
ori a1, a1, 0xf
add t0, a0, a1
sub t1, a0, t0
add t0, t0, t1

Specification Engineer

Verification Engineer

https://forge.ispras.ru/projects/microtesk-riscv
Application 1: Test Program Generation (3/3)

- Test templates similar to RISC-V Foundation’s Tests
  https://github.com/riscv/riscv-tests

- Test templates similar to RISC-V Torture Test Generator
  https://github.com/ucb-bar/riscv-torture

```python
class MyTemplate < RiscVBaseTemplate
    def run
        block(:combinator => 'product',
                :compositor => 'random') {
            iterate {
                xor x(_), x(_), x(_) do situation(...) end
                lui x(_), _
            }
            iterate {
                and x(_), x(_), x(_)
                or  x(_), x(_), x(_)
            }
        }
    .run
end
```

- Combinatorial Brute Force
- Randomization
- Constraint Solving
- Self Checking

```plaintext
# Single Test Case
# Initialization
ori  a7, a7, 0x2d7
slli a7, a7, 0xb
ori  a7, a7, 0x1
slli a7, a7, 0xb
ori  t3, t3, 0x164
slli t3, t3, 0xb
ori  t3, t3, 0x52b
slli t3, t3, 0xb
ori  t3, t3, 0x24e

# Stimulus
and  s4, a7, a7
xor  s8, s4, t3
```

2 x 2 = 4 test cases

Code is distributed under Apache License, v2.0
Application 2: Binary Code Verification (1/2)

Source Code
(C + ACSL)

//@ requires
//@ ensures
int idiv(…) {…}
//@ loop inv
while(…) {...}

Frama-C Platform
Core → IR → Jessie
http://frama-c.com

Why3 Platform
WhyML → VC Gen → VC → Encoder

Provers
(Coq, CVC4)

Binary Code
006f 04c0 2f73 3420 0f93 0080 0a63 03ff 2f73 0090 0663 03ff 0f93 00b0 0263 03ff 0f13 ...

MicroTESK Framework
Decoder → IR → Builder
ISA Model
http://www.microtesk.org

Research Prototype
VC Gen → VC → Encoder

MicroTESK Framework
Decoder → IR → Builder
ISA Model
http://www.microtesk.org

Annotations
(SMT-LIB)

Work in progress…
Application 2: Binary Code Verification (2/2)

Source Code
(C + ACSL)

```c
int idiv (int a, int b, int *r) {
    int q = 0; *r = a;
    //@ loop invariant (a == b*q + *r);
    //@ loop invariant (0 <= *r)
    while (*r >= b) {
        q++; *r -= b;
    }
    //@ assert (a == q*b + *r);
    //@ assert (0 <= *r < b);
    return q;
}
```

Binary Code
(CFG)

```asm
mv a5,a0
mv a4,a1
sd a2,-48(s0)
sw a5,-36(s0)
mv a5,a4
sw a5,-40(s0)
sw zero,-20(s0)
ld a5,-48(s0)
lw a4,-36(s0)
sw a4,0(a5)
```

Verification Conditions
(SMT-LIB)

```
(assert (= op0.rd.i!1 #b01111))
(assert (= op0.rs1.i!1 #b01000))
(assert (= op0.imm!1 #b11111010000))
(assert (and
  (= op0.rs1!1 (select X!1 op0.rs1.i!1))
  (= addr!1 (bvadd op0.rs1!1 (_ sign_extend 52) op0.imm!1)))
  (= index!1 (bvlshr addr!1 #b0...010)))
  (= dword!9 (select M!1 index!1))
  (= X!2 (store X!1 op2.rd.i!1 dword!9)))
...
(assert (and
  (= X!10 (store X!9 op5.rd.i!1 op5.rd!1))
  (= op5.rs1!1 (select X!9 op5.rs1.i!1))
  (= op5.rd!1 (bvadd op5.rs1!1 (_ sign_extend 52) op5.imm!1)))
```

```
L2:
ld a5,-48(s0)
lw a4,0(a5)
lw a5,-40(a0)
```

```
L3:
lw a5,-20(s0)
addiw a5,a5,1
sw a5,-20(s0)
ld a5,-48(s0)
lw a4,0(a5)
lw a5,-40(s0)
subw a5,a4,a5
sext.w a4,a5
ld a5,-48(s0)
sw a4,0(a5)
```

```text
\[ \psi = \begin{cases} 
  a = X[10] & \text{if true} \\
  b = X[11] & \text{if false} \\
\end{cases} 
\]
```

```
\[ \chi = \begin{cases} 
  a = M[X[8] - 36] \gg 2 & \text{if true} \\
  b = M[X[8] - 40] \gg 2 & \text{if false} \\
  r = M[X[8] \gg 2] \gg 2 & \text{if true} \\
  a = b \times q + r & \text{if false} \\
  \text{true} \leq r < b & \text{if true} \\
\end{cases} 
\]`
Future Work Directions

- **RISC-V specifications**
  - Specification of subsets (RV128I, Q, L, B, T, J, P, V, and N)
  - Specification of MMU (address translation mechanisms)

- **MicroTESK framework**
  - Online test program generation (post-silicon verification)
  - RTL generation (golden model for equivalence checking)

- **Specifications validation**
  - Testing (self checking, co-simulation, coverage analysis)
  - Formal equivalence checking (HLS to Verilog + BMC)
MicroTESK Verification Technology for Microprocessors

http://www.microtesk.org
microtesk-support@ispras.ru

MicroTESK is a reconfigurable (retargetable and extendable) model-based test program generator (TPG) for microprocessors and other programmable devices (such kind of tools are also called instruction stream generators or ISG). This generator is customized with the help of instruction-set architecture (ISA) specifications and configuration files, which describe parameters of the microprocessor subsystems (pipeline, memory and others). The suggested approach eases the model development and makes it possible to apply the modal-based testing in the early design stages when the microprocessor architecture is frequently modified.

The current version of the tool supports ISA specification (in XML) and manual development of test program templates (in Ruby). It also implements lightweight methods for automated test program generation, including random-based and combinatorial techniques. Facilities for describing memory management units and microprocessor pipelines (microarchitectural networks) are under development, and so are the methods for advanced test program generation. The framework is applicable to a wide range of microprocessor architectures including RISC (ARM, MIPS, SPARC, etc.), CISC (x86, etc.), VLIW/EPIC (Elbrus, Itanium, etc.), DSP, GPU, etc.