MicroTESK: Test Program Generation and Binary Code Deductive Analysis

Alexander Kamkin (kamkin@ispras.ru)

Ivannikov Institute for System Programming
Russian Academy of Sciences
Established in 1994 from two departments of Institute for Cybernetics Problems of the RAS
Division of Mathematical Sciences of the RAS
200+ researchers and software engineers including approximately 15 ScDs and 60 PhDs
Many employees work as professors and lecturers in leading universities: MSU, MIPT, and HSE
Microprocessor Verification Group consists of 7+ researchers and engineers including 2 PhDs
http://www.ispras.ru/en
Formalization Motivation

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

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

- **Functional and security verification**
  - ADL + scenarios $\rightarrow$ 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

- [https://github.com/mit-plv/riscv-semantics](https://github.com/mit-plv/riscv-semantics)
  by MIT in Haskell
- [https://github.com/cliffordwolf/riscv-formal](https://github.com/cliffordwolf/riscv-formal)
  by Clifford Wolf (Symbiotic EDA) in SystemVerilog
- [https://github.com/samuelgruetter/riscv-coq](https://github.com/samuelgruetter/riscv-coq)
  by Samuel Gruetter (MIT) in Coq
- [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
- [https://github.com/rsnikhil/RISCV-ISA-Spec](https://github.com/rsnikhil/RISCV-ISA-Spec)
  by Rishiyur Nikhil (Bluespec) in Haskell
- [https://github.com/SRI-CSL/l3riscv](https://github.com/SRI-CSL/l3riscv)
  by Prashant Mundkur (SRI International) in L3
  by ISP RAS in nML
Specification Approach and Framework

MicroTESK Framework

Specifications (nML)

Specifications Translator

Processor Model
- Metadata
- Simulator
- Coverage

Test Program Generator

Tool Builders

Symbolic Executor

Simulator (C/C++)

Test Programs (ASM)

Specifications (WhyML)

Ref. Simulator (QEMU)

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

```ml
#ifdef RV64I
  let XLEN = 64
...
#else
...
#endif

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;
}
```

```ml
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)
```

```ml
type WORD = card(32)
type XWORD = card(XLEN)
type FLOAT32 = float(23, 8)
reg XREG [32, XWORD]
reg PC [XWORD]
```
## Specifications

<table>
<thead>
<tr>
<th>Specifications Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC-V Instruction Set Manual Vol. I: User-Level ISA (v. 2.2)</td>
<td>145 pages (94 pages in chapters 2-18)</td>
</tr>
<tr>
<td>Base</td>
<td>Extension</td>
</tr>
<tr>
<td>RV32I</td>
<td>M</td>
</tr>
<tr>
<td>RV32E</td>
<td>A</td>
</tr>
<tr>
<td>RV64I</td>
<td>Q</td>
</tr>
<tr>
<td>RV128I</td>
<td>L</td>
</tr>
<tr>
<td># Specified Instructions</td>
<td>200+ instructions</td>
</tr>
<tr>
<td>ISA Specifications Size (nML)</td>
<td>4 500 LOC (w/o comments)</td>
</tr>
<tr>
<td>MMU Specification Size (MMuSL)</td>
<td>0 LOC (unspecified)</td>
</tr>
</tbody>
</table>
Application 1: Test Program Generation (1/3)

Test Programs (ASM)

Test Programs (ASM)

Design Under Test (RTL, FPGA)

lui a0, 0xdead
ori a0, a0, 0x0
lui a1, Oxbeef
ori a1, a1, Oxf
add t0, a0, a1
sub t1, a0, t1
add t0, t0, t1

Reference Simulator (C/C++)

0x2000: lui ...
0x2004: ori ...
0x2008: lui ...
0x200c: ori ...
0x2010: add ...
0x2014: sub ...
0x2018: bug ...

Execution Traces (Tarmac)

0x2000: lui ...
0x2004: ori ...
0x2008: lui ...
0x200c: ori ...
0x2010: add ...
0x2014: sub ...
0x2018: add ...

Trace Comparator (Python)
Application 1: Test Program Generation (2/3)

MicroTESK Test Program Generator

- **Translator**
- **Specifications (nML)**
- **Test Templates (Ruby)**
- **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, t1
  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
  
  [GitHub link]

- Test templates similar to RISC-V Torture Test Generator
  
  [GitHub link]
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

Binary Code
006f 04c0 2f73
3420 0f93 0080
0a63 03ff 0f93
0009 0663 03ff
0f93 00b0 0263
03ff 0ff3 ...

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

Provers
(Probes, CVC4)

Research Prototype
VC Gen ➔ VC ➔ Encoder
Work in progress...

Annotations
(SMT-LIB)
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)

```plaintext
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)
```

Verifcation Conditions (SMT-LIB)

```plaintext
(a = X[10])
b = X[11]
(a >= 0)
b > 0
```

```plaintext
(a = M[X[8] - 36] >> 2]
b = M[X[8] - 40] >> 2]
q = M[X[8] - 20] >> 2]
r = M[M[X[8] >> 2] >> 2]
(a = b * q + r)
0 <= r
```

```plaintext
_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)
```

```plaintext
(a = X[10])
b = X[11]
(a >= 0)
b > 0
```

```plaintext
(a = M[X[8] - 36] >> 2]
b = M[X[8] - 40] >> 2]
q = M[X[8] - 20] >> 2]
r = M[M[X[8] >> 2] >> 2]
(a = b * q + r)
0 <= r
```

13
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)
ISP RAS OPEN: December 5-6, 2019