October 29-30, 2019

Design and Verification Conference and Exhibition Europe

Munich, Germany

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Open Source Solution *for* RISC-V Verification

Mikhail Chupilko, Alexander Kamkin, Alexander Protsenko

ISP RAS

Ivannikov Institute for System Programming

Russian Academy of Sciences

RISC-V 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)

RISC-V Open Specifications

- https://github.com/mit-plv/riscv-semantics
  by MIT in **Haskell**
- https://github.com/cliffordwolf/riscv-formal
  by Clifford Wolf (Symbiotic EDA) in **SystemVerilog**
- https://github.com/samuelgruetter/riscv-coq
  by Samuel Gruetter (MIT) in **Coq**
- https://github.com/rsnikhil/RISCV_ISA_Formal_Spec_in_BSV
  by Rishiyur Nikhil (Bluespec) in **Bluespec SystemVerilog**
- https://github.com/rsnikhil/RISCV-ISA-Spec
  by Rishiyur Nikhil (Bluespec) in **Haskell**
- https://github.com/SRI-CSL/l3riscv
  by Prashant Mundkur (SRI International) in **L3**
- https://forge.ispras.ru/projects/microtesk-riscv
  by ISP RAS in **nML**

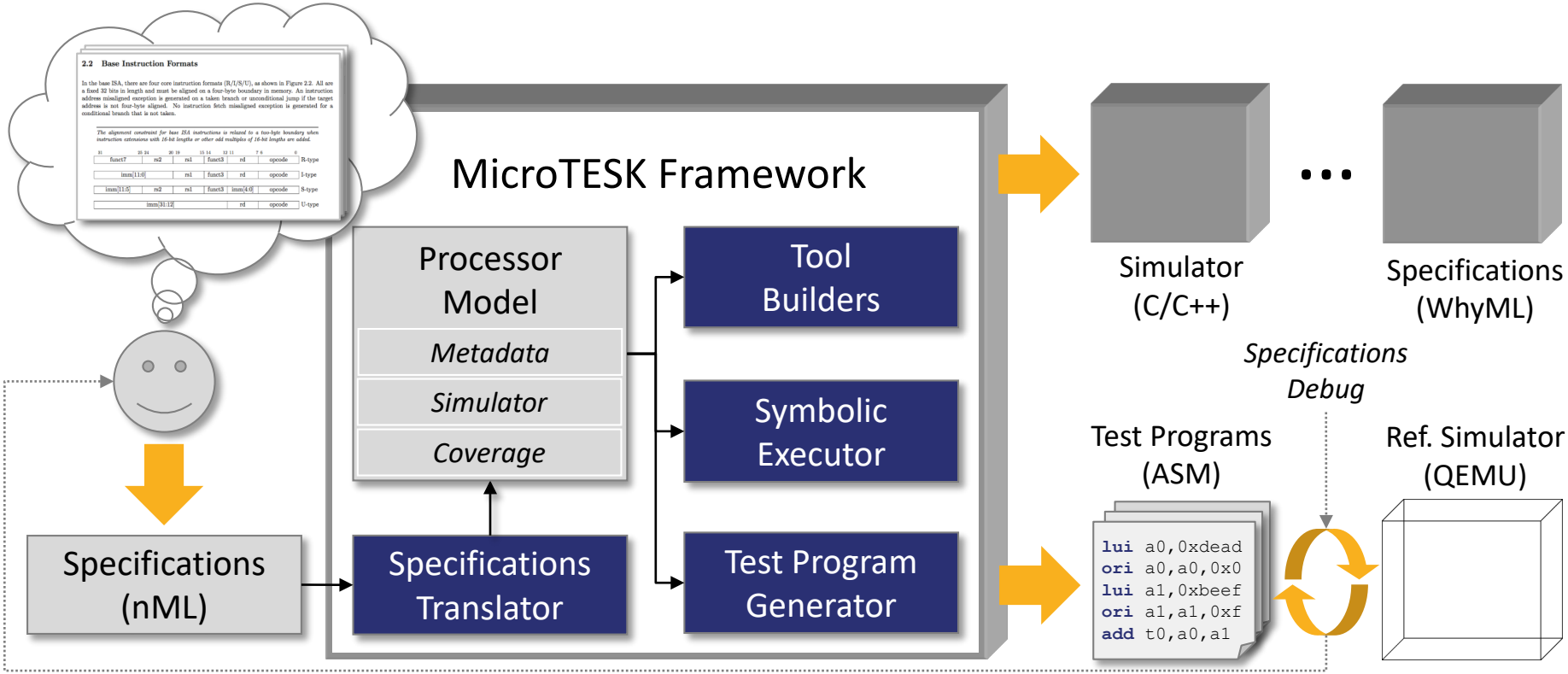
We are here

RISC-V Open Test Suites

- https://github.com/riscv/riscv-tests
  Benchmarks and tests by RISC-V Foundation in **C** and **assembly**
- https://github.com/riscv/riscv-compliance
  Compliance tests by RISC-V Foundation in **assembly**
- https://github.com/ucb-bar/riscv-torture
  Test generator by Yunsup Lee and Henry Cook (UC Berkley) in **Scala**
- https://forge.ispras.ru/projects/microtesk-riscv
  Test generator by us (ISP RAS) in **Java** and **Ruby**
- https://github.com/ucb-bar/riscv-avs
  Tests and test generators by us (ISP RAS) in **assembly** and **Ruby**

We are here

Specification Approach and Framework



ISPRAS https://forge.ispras.ru/projects/microtesk

accellera SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

nML Language Overview

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

```
type WORD    = card(32)
type XWORD   = card(XLEN)
type FLOAT32 = float(23, 8)
```

```
reg XREG [32, XWORD]
reg PC   [XWORD]
```

```
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;
 }
```

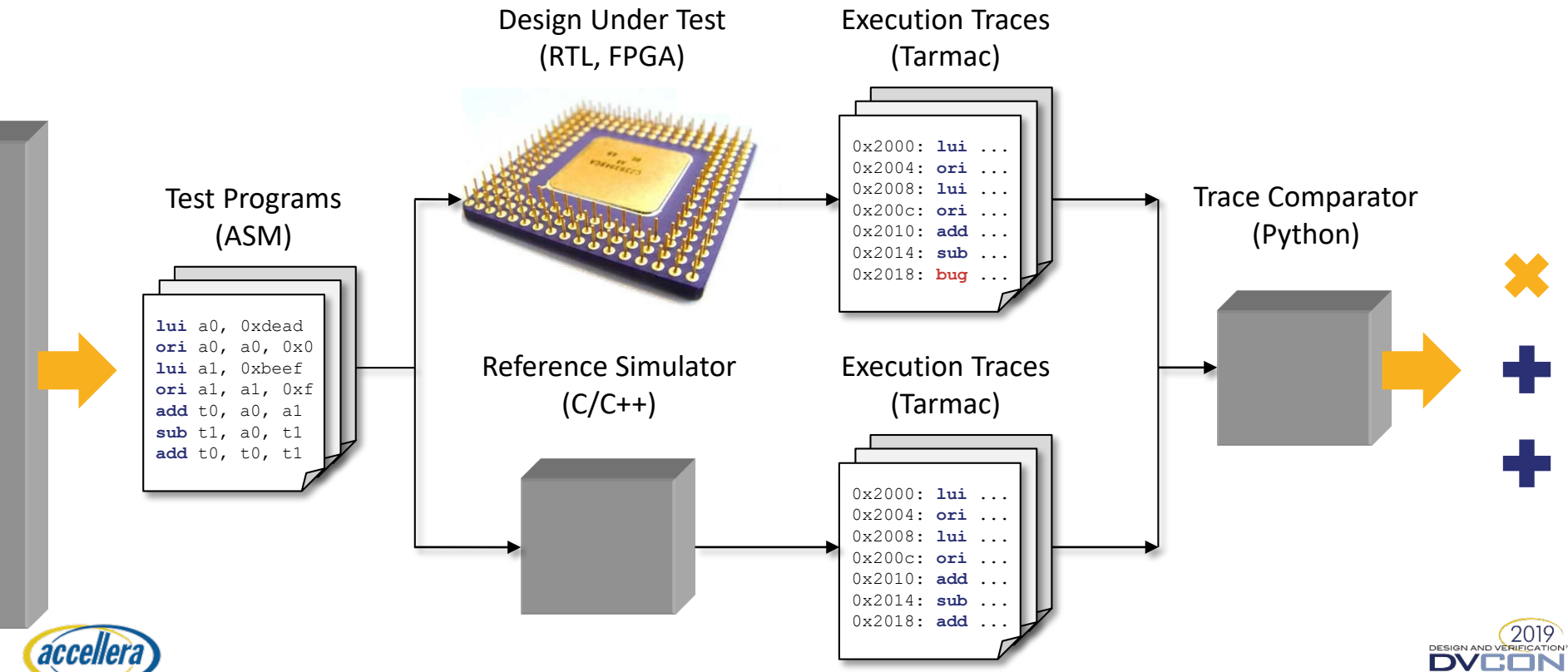
```
mode X (i: card(5)) = XREG[i]
 syntax = format("%s",
  if   i==0 then ZERO().syntax
  elif i==1 then RA().syntax
  elif i==2 then SP().syntax
  elif i==3 then GP().syntax
  elif i==4 then TP().syntax
  elif i>=5 && i<=7 then
    Temp(coerce(card(3), i-5)).syntax
  elif i>=8 && i<=9 then
    Saved(coerce(card(4), 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)
```


Markus Freericks. The nML Machine Description Formalism. TU Berlin, 1991-1993
http://www.i6.in.tum.de/Main/Publications/Freericks1991a.pdf

RISC-V Specifications

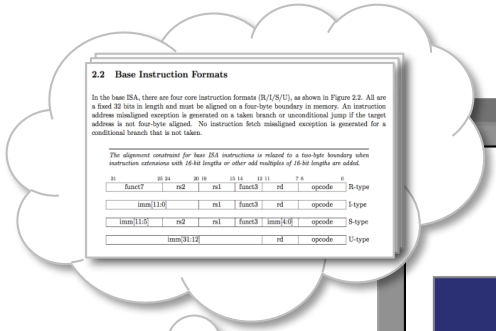
Specifications Characteristic					Value		
RISC-V Instruction Set Manual Vol. I: User-Level ISA (v. 2.2)					**145** pages (**94** pages in chapters 2-18)		
Base	Extension				Base	Extension	In Total
RV32I	M	F	C	P	24 pages	44 pages	68 pages
RV32E	A	D	B	V			
RV64I		Q	J	N	2 pages	24 pages	26 pages
RV128I		L	T				
# Specified Instructions					**230** instructions		
ISA Specifications Size (nML)					**4 900** LOC (w/o comments)		
MMU Specification Size (MMUSL)					**460** LOC (Sv32, Sv39, and Sv48)		

Application 1: Test Program Generation (1/3)



accellera SYSTEMS INITIATIVE™
2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Application 1: Test Program Generation (2/3)



2.2 Base Instruction Formats

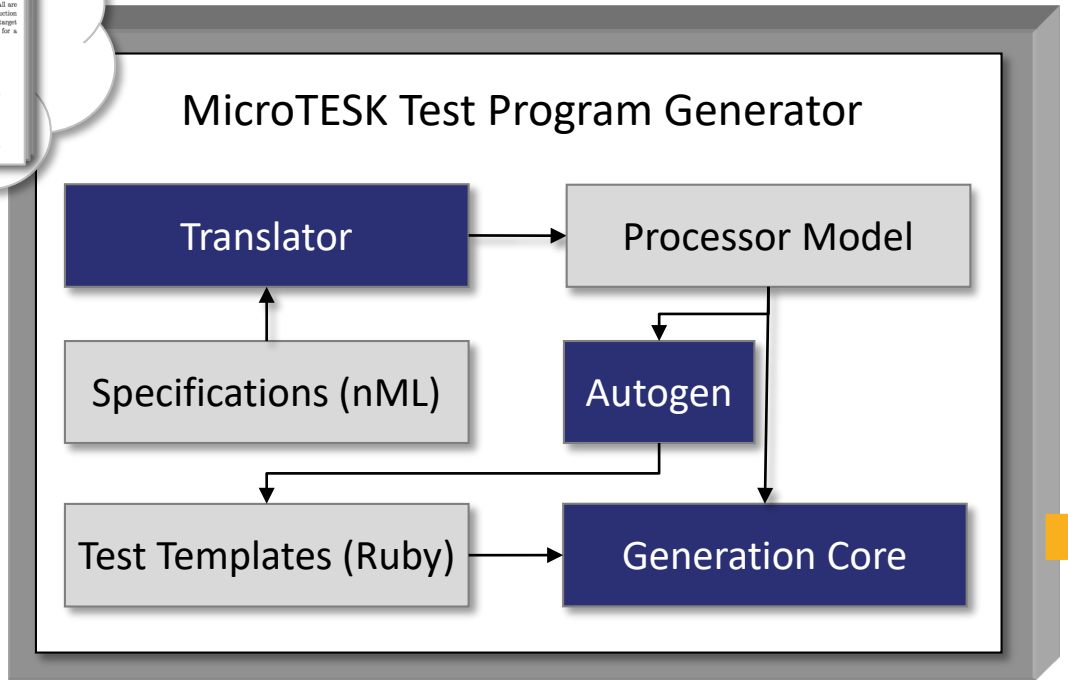
In the base ISA, there are four core instruction formats (R/I/S/U), as shown in Figure 2.2. All are a fixed 32 bits in length and must be aligned on a four-byte boundary in memory. An instruction address misaligned exception is generated on a taken branch or unconditional jump if the target address is not four-byte aligned. No instruction fetch misaligned exception is generated for a conditional branch that is not taken.

The alignment constraint for base ISA instructions is relaxed to a two-byte boundary when instruction extensions with 16-bit lengths or other odd multiples of 16-bit lengths are added.

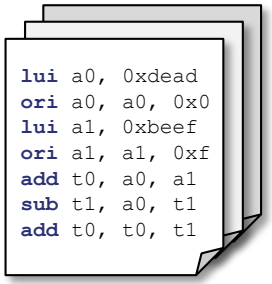
31	25 24	20 19	15 14	12 11	7 6	0	
funct7	rs2	rs1	funct3	rd	opcode		R-type
imm[11:0]		rs1	funct3	rd	opcode		I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode		S-type
imm[31:12]				rd	opcode		U-type

Specification Engineer

Verification Engineer



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
```

ISPRAS https://forge.ispras.ru/projects/microtesk-riscv

accellera SYSTEMS INITIATIVE™
2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Application 1: Test Program Generation (3/3)

```
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
end
```

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

2 x 2 = 4 test cases

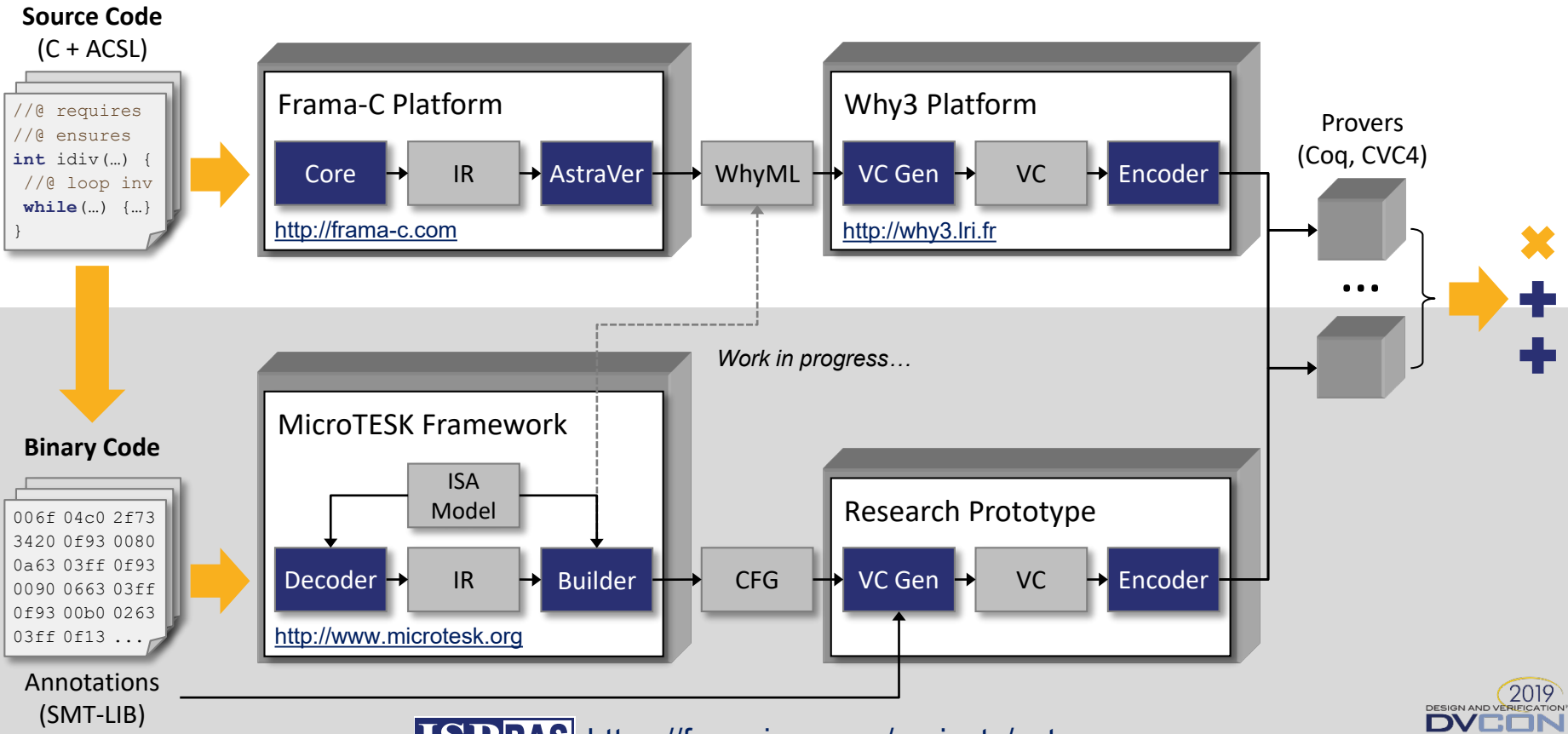
```
# Single Test Case
# Initialization
ori   a7, a7, 0x2d7
slli  a7, a7, 0xb
ori   a7, a7, 0x1
slli  a7, a7, 0xb
ori   a7, a7, 0x3d2
ori   t3, t3, 0x164
slli  t3, t3, 0xb
ori   t3, t3, 0x52b
slli  t3, t3, 0xb
ori   t3, t3, 0x24e

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

Case Study: Syntacore's SCR1 [RV32IMC]

Test Suite	Coverage LOC	Coverage %
riscv-tests & riscv-compliance except `*.c`	1422 / 2922	48%
microtesk-riscv autogen only *not all of them*	1354 / 2922	46%

https://github.com/syntacore/scr1

Application 2: Binary Code Verification (1/2)



ISPRAS https://forge.ispras.ru/projects/astraver

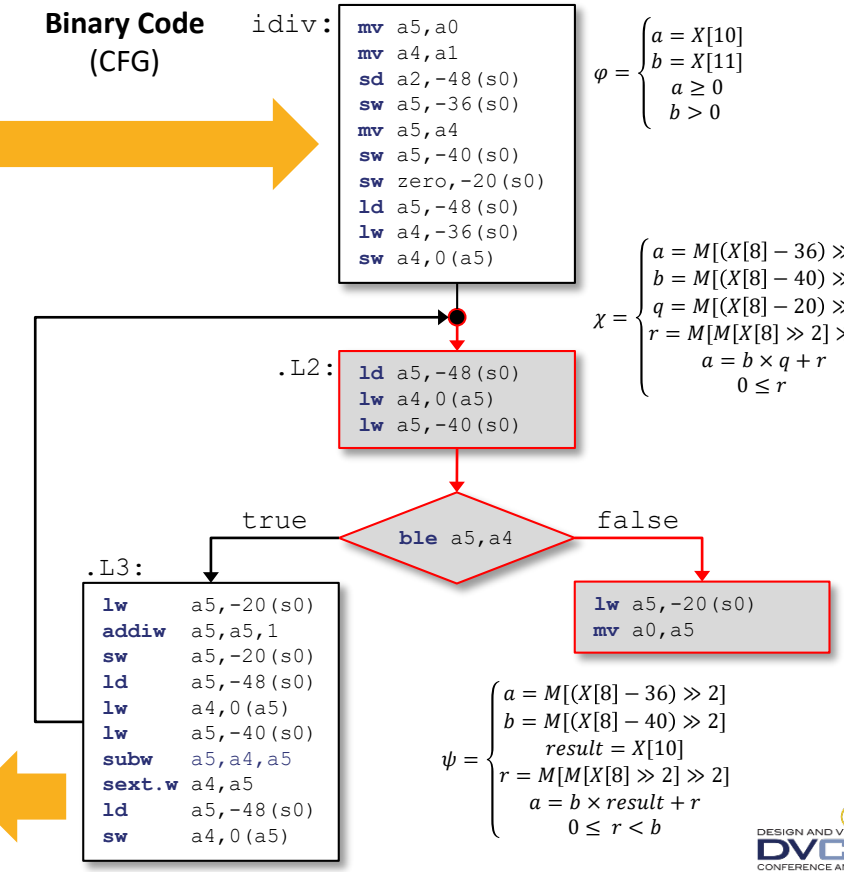
2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Application 2: Binary Code Verification (2/2)

Source Code
(C + ACSL)

```c
//@ requires a >= 0;
//@ requires b > 0;
//@ ensures a == b*\result + *r;
//@ ensures 0 <= *r < b;
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)



$$\varphi = \begin{cases} a = X[10] \\ b = X[11] \\ a \geq 0 \\ b > 0 \end{cases}$$

$$\chi = \begin{cases} a = M[(X[8] - 36) \gg 2] \\ b = M[(X[8] - 40) \gg 2] \\ q = M[(X[8] - 20) \gg 2] \\ r = M[M[X[8] \gg 2] \gg 2] \\ a = b \times q + r \\ 0 \leq r \end{cases}$$

$$\psi = \begin{cases} a = M[(X[8] - 36) \gg 2] \\ b = M[(X[8] - 40) \gg 2] \\ result = X[10] \\ r = M[M[X[8] \gg 2] \gg 2] \\ a = b \times result + r \\ 0 \leq r < b \end{cases}$$

Verification Conditions
(SMT-LIB)

```
(assert (= op0.rd.i!1  #b01111))
(assert (= op0.rs1.i!1 #b01000))
(assert (= op0.imm!1   #b111111010000))
(assert (and                                  ; ld a5, -48(s0)
 (= 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                                  ; mv a0, a5
 (= 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)))
```

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Future Work Directions

- **RISC-V specifications**
  - Specification of subsets (RV128I, Q, L, B, T, J, P, V, and N)
  - Multi-core issues (cache coherency, atomic load/stores, etc.)
- **MicroTESK framework**
  - Online test program generation (post-silicon verification)
  - RTL generation (golden model for equivalence checking)
- **RISC-V architecture validation suite**
  - More test programs and templates (memory, branches, etc.)
  - ISA-level coverage estimation (situations, dependencies, etc.)

MicroTESK

Verification Technology *for* Microprocessors

```
_start:     .org  0x50000
            nop
            adr   x0, page_table
            msr   ttbr0_el1, x0
            movz  x0, #0x8082, LSL #16
            movk  x0, #0x3219, LSL #0
            msr   tcr_el1, x0
            isb   #0
            movz  x0, #0x30c5, LSL #16
            movk  x0, #0x0831, LSL #0
            msr   sctlr_el1, x0
            isb   #0
            movz  x0, #0x0020, LSL #16
            movk  x0, #0x4000, LSL #0
            ldar  x1, [x0], #01
            hlt   #0xf000
```

Thank You!

http://www.microtesk.org

https://github.com/ispras/riscv-avs

forge.ispras.ru/projects/microtesk

Home My page Projects Help

MicroTESK

Overview Activity Roadmap Issues New issue Gantt Calendar News Documents

Overview

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*). The 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 model-based testing in the early design stages when the microprocessor architecture is frequently modified.

The current version of the tool supports *ISA specification* (in nML) 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.