Formalisation of security mechanisms for the RISC-V processor architecture

Matthieu BATY

About

- PhD student at CIDRE (Inria/CentraleSupélec) since january
- Director: Ludovic MÉ
- Advisors:
  - CIDRE: Guillaume HIET, Pierre WILKE
  - ANSSI: Arnaud FONTAINE, Alix TRIEU

"Formalisation of security mechanisms for the RISC-V processor architecture":

- Formal methods
- Hardware security

Motivation

The security of a system depends on the security of all the layers it is built upon.

Why processors?

- Ubiquitous
- Vulnerable (Spectre, Meltdown, SPOILER, Foreshadow, TLBleed, Pentium FDIV, . . .)
- Not a toy example

Why formal methods?

- Show the **absence** of bugs while keeping complexity under control
- Trust

Some basic notions — 1/3

“Formalisation of security mechanisms for **the RISC-V processor architecture**”

RISC-V:

- Instruction set architecture (ISA)
- Open standard
- Emerging technology
- Many open source implementations

Some basic notions — 2/3

“Formalisation of **security mechanisms** for the RISC-V processor architecture”

Security mechanisms:

- Enforce **security properties**
- Example: shadow stack — return address integrity
- Part of the implementation, not of the specification

Some basic notions — 3/3

“**Formalisation** of security mechanisms for the RISC-V processor architecture”

Formal methods:

- Workflow overview:
  - Build a model
  - Define properties about it
  - Prove them
- Exhaustive, unlike most test suites
- Successful in the software world (CompCert, seL4, . . .)

Prior works

- Sail (Cambridge): ISA description language
- Thomas Letan's PhD (CIDRE): x86's System Management Mode, not about microarchitecture
- **Kami and Kôika (MIT), Cava (Google): $\approx$ formal Verilog**
- "Integration Verification across Software and Hardware for a Simple Embedded System", PLDI 2021, A. Erbsen et al.

Hardware development — 1/3

$\boxed{\text{Specification}}$

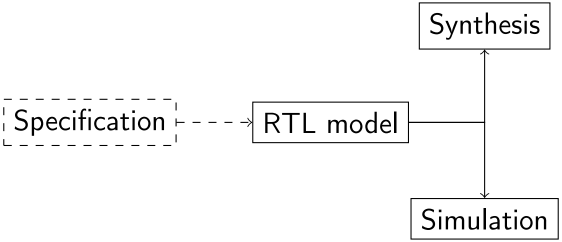
- Set of requirements for implementations
- Plain English

Hardware development — 2/3

$$\begin{tikzcd} \text{Specification} \arrow[r, dashed] & \boxed{\text{RTL model}} \end{tikzcd}$$

- Register Transfer Level
- Verilog, VHDL, Chisel, BlueSpec, . . .
- Architectural decisions
  - L1 cache size?
  - Pipelining?
  - **Security mechanisms**?
  - . . .

Hardware development — 3/3



- Simulation: fast and cheap
- Synthesis:
  - FPGA bitstream, photomasks, ...
  - Physical placement of components/routing

Certified hardware development

Not too common in the industry, generally limited to:

- Model checking
- Proofs about small mechanisms

Many interesting problems:

- Formal specifications
- Proof of an RTL model's adherence to a specification
- **Proof of security properties of an RTL model**
- Preservation of semantics between an RTL description and a physical implementation

Kôika — Overview

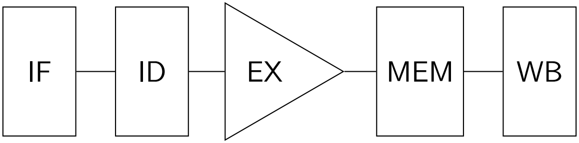
- https://github.com/mit-plv/koika
- "The Essence of BlueSpec", PLDI 2020, Thomas Bourgeat et al. (MIT PLV)
- Formal RTL HDL embedded in Coq
- Inspired by BlueSpec
- In active development
- Includes a basic RISC-V model

Kôika — Workflow

$$\begin{tikzcd}
\text{Evaluation} & & & \text{Simulation} \\
\text{Kôika model (e.g RISC-V)} \arrow[u] \arrow[r, "*"] & \text{Coq circuit} \arrow[r, "*"] & \text{OCaml circuit} \arrow[r] & \text{Verilog} \arrow[u] \arrow[d] \\
 & & & \text{Synthesis}
\end{tikzcd}$$

Kôika — Basics — 1/4

Made with pipelined systems in mind: one rule per stage. Just like BlueSpec!



Cycles and rules:

- During a given cycle each rule might run or not
- Conflicts, scheduling

Kôika — Basics — 2/4

If two rules are in conflict, which one should be prioritized?

- Explicit schedule
- Deterministic semantics

How can two rules communicate?

- Read at the beginning of the cycle, write at the end: not enough for some forms of operand forwarding
- Notion of ports (0 and 1)

Kôika — Basics — 3/4

Example: Collatz sequence

```
rule divide =
  let v = r.rd0 in
  if iseven(v) then
    r.wr0(v >> 1)

rule multiply =
  let v = r.rd1 in
  if isodd(v) then
    r.wr1(3*v + 1)

schedule s = [divide; multiply]
```

From *"The Essence of BlueSpec", PLDI 2020, Thomas Bourgeat et al.*

Kôika — Basics — 4/4

Interacting with the external world: external calls (e.g. for the memory)

- A pure model of each external call is required to evaluate a cycle
- For now, Kôika treats external calls as pure:
  - Not realistic
  - Abusive optimizations

Past work — RISC-V model extension

The RISC-V specification doesn't just describe a fixed ISA:

- base: 32 or 64 bits
- extensions: integer multiplication and division, floating point numbers, atomic instructions, ...

It can be convenient to model a family of processors based on the same architecture.

- Select which base and extensions to use
- Kôika model written indirectly through Coq functions

Past work — Shadow stack

Definition of a simple shadow stack mechanism:

- Added to Kôika's RISC-V model
- Fixed stack size
- Turns processor off in case of misbehavior
- Push when procedure called, pop when procedure returns

Some interesting properties:

- Return address overwritten $\implies$ processor halts
- Overflows or underflows $\implies$ processor halts
- In all other situations, the processor behaves just as it used to before the shadow stack was added

Past work — Proving and semantics

Kôika defines its semantics through a set of interpretation functions, and dependent types appear in them. This lead to several issues:

- Performance problems
- Unfolding of the interpretation function hard to control

Alternative untyped inductive semantics:

- Inductively defined predicate: proof constructors improve control control over the computation
- Solves part of the performance problem
- Proof of equivalence

Ongoing work — Normal form

The models are in a form that is easy to write but hard to reason about. They can be simplified:

- Single rule instead of schedule
- Only write actions guarded by a single condition each
- A bit like an optimization pass in a certified compiler
- Prove equivalence (same state at the end of a cycle)

Future work

Short term:

- Finish the proof
  - Build proof infrastructure
  - Apply to Shadow Stack
- Publish

And then?

- Stick to Kôika? More complex mechanisms?
- Try Cava?
- Yet another formal HDL?

Thank you