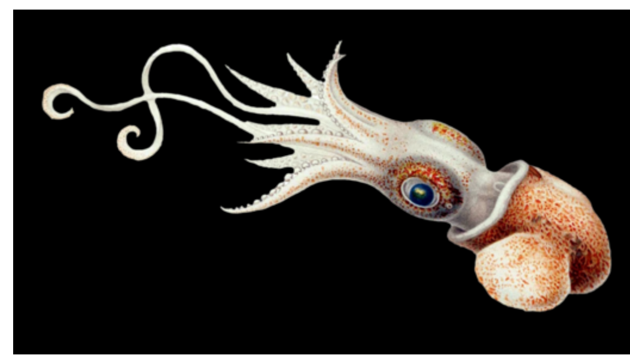


The Essence of BlueSpec
A Core Language for Rule-Based Hardware Design

Thomas Bourgeat, Clément Pit-Claudel, Adam Chlipala, and Arvind | MIT CSAIL

Abstract

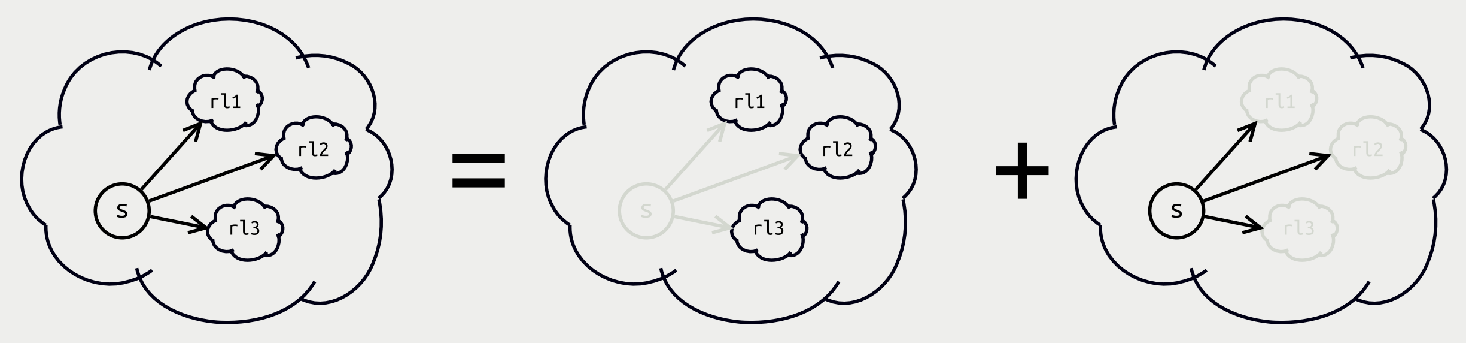
Bluespec, a high-hevel HDL, offers a simple concurrency model that enables functional reasoning without compromising performance.

Unfortunately, its cost model is hard to formalize: performance depends on user hints and static analysis of conflicts within a design.

We present Kôika, a Bluespec derivative that gives direct control over *scheduling* decisions that determine performance, while using dynamic analysis to avoid concurrency anomalies.

Our implementation includes formal semantics, mechanized theorems, and a verified compiler.

Overview



State function = Rules + Explicit schedule

Contributions

- Core calculus for rule-based language amenable to formal reasoning about functionality and performance
- Cycle-accurate operational semantics
- Proof of one-rule-at-a-time abstraction
- Full Coq mechanization of the semantics
- Verified compilation to RTL
- Performance case-study of a pipelined processor

Motivation

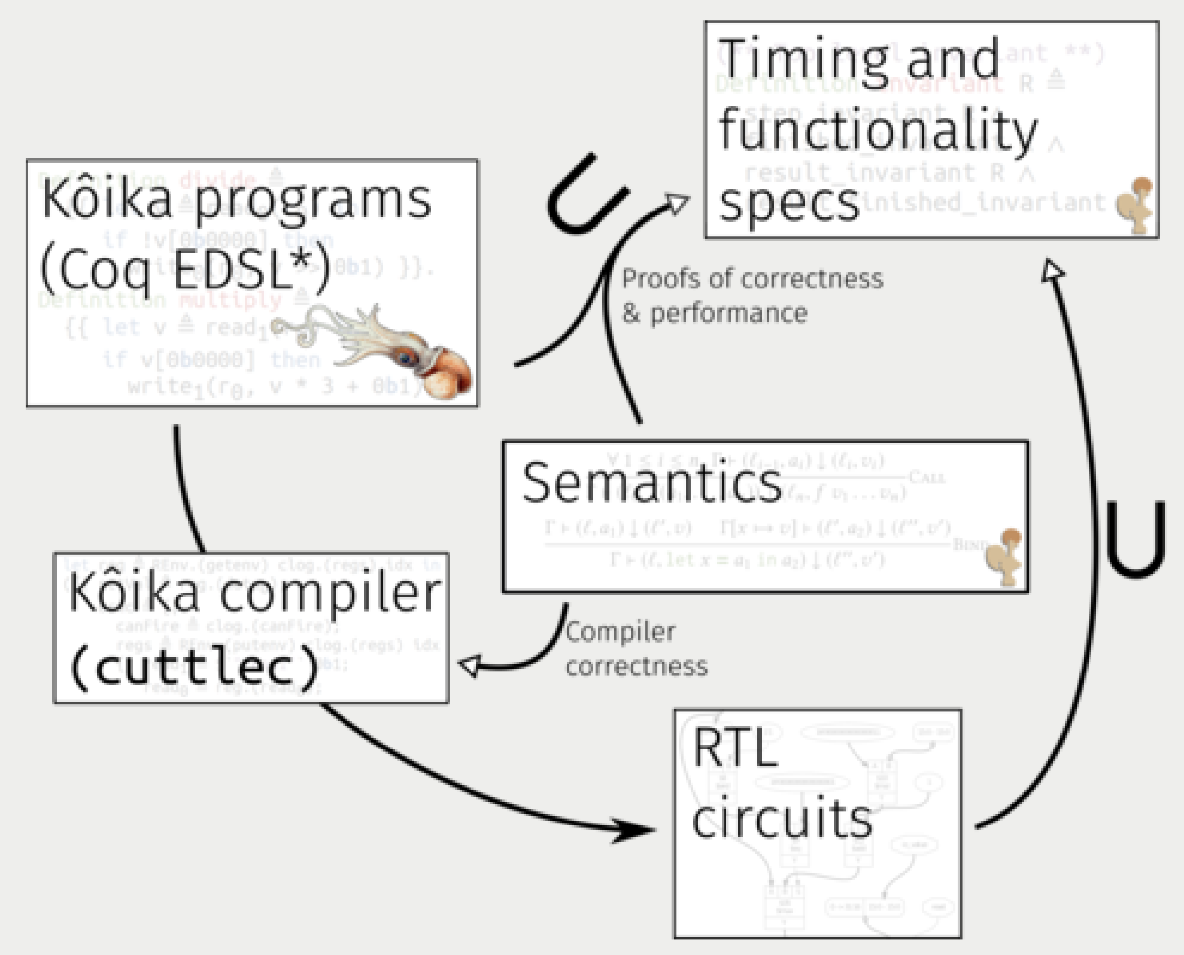
HW spec = **Functional spec** (ORAAT) + **Performance spec** (Not in previous work)

$$\text{in} \to f \to \boxed{\;\;|\;\;|\;\;} \to g \to \text{out}$$

Functional spec: out = $g(f(\text{in}))$
Performance spec: $\text{out}[t+2] = g(f(\text{in}[t]))$

Precise semantics allows for *performance* proofs

Our design


Kôika programs (Coq EDSL*)
Timing and functionality specs
Proofs of correctness & performance
Semantics
Kôika compiler (cuttlec)
Compiler correctness
RTL circuits

The Kôika EDSL

```
rule divide =                   rule multiply =
  let v = r.rd_0() in             let v = r.rd_1() in
  if iseven(v) then               if isodd(v) then
    r.wr_0(v >> 1)                  r.wr_1(3 * v + 1)

schedule collatz = [divide; multiply]
```

```
rule swap =                     rule dyn_abort =
  s.wr_0(r.rd_0());               if r.rd_0() == 0 then
  r.wr_0(s.rd_0())                  t.wr_0(0b1);
                                  if s.rd_0() == 0 then
                                    t.wr_0(0b1)
```

Core language

Registers *r* Variables *x* External functions *f* Constants *b*

Ports *p* ∈ {0, 1}

Actions *a* ::=

```
$b   $x   $f($a, …, $a)   skip   let $x := $a in $a
$r.rd_$p()   $r.wr_$p($a)   if $a then $a else $a
…
```

Semantics

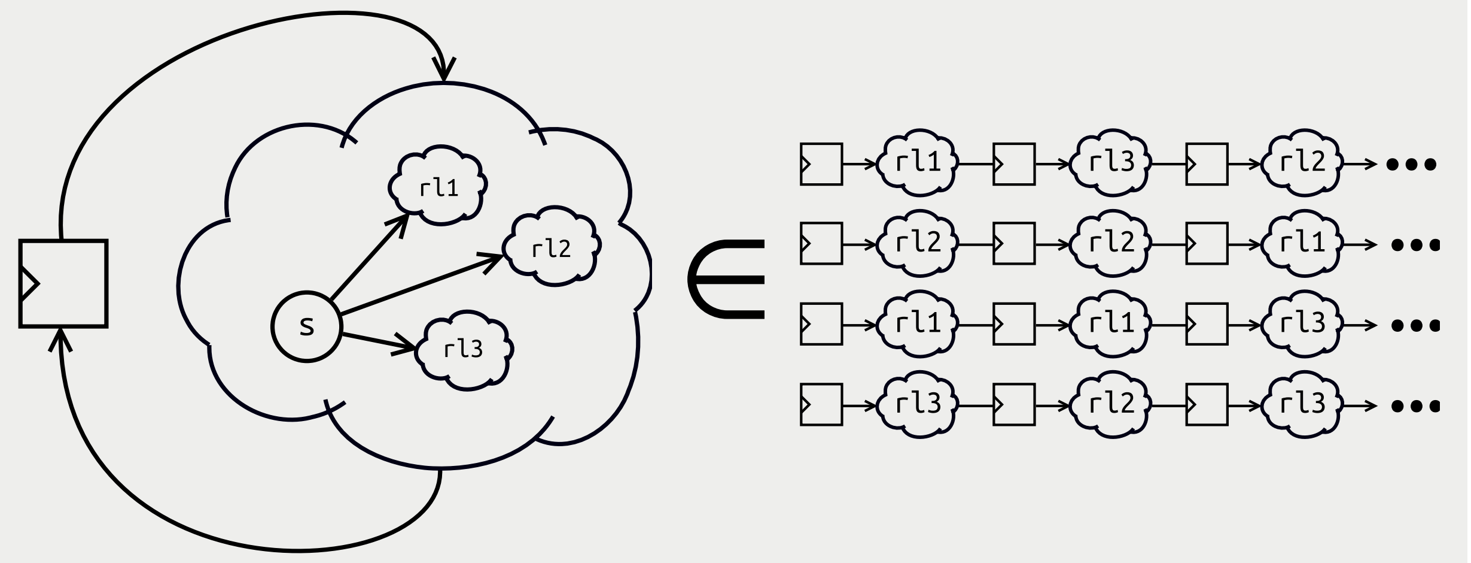
$$\frac{(\mathsf{wr}_1, r, *) \notin L \qquad (\mathsf{wr}_0, r, *) \notin L}{\Gamma \vdash (\ell, r.\mathtt{rd}_0 \downarrow (\ell \mathbin{+\!\!+} [(\mathsf{rd}_0, r)], \mathcal{R}[r])}\,\mathrm{RD}_0$$

$$\frac{\Gamma \vdash (\ell, a) \downarrow (\ell', v) \qquad (\mathsf{wr}_1, r, *) \notin L \mathbin{+\!\!+} \ell'}{\Gamma \vdash (\ell, r.\mathtt{wr}_1\,()) \downarrow (\ell' \mathbin{+\!\!+} [(\mathsf{wr}_1, r, v)], \mathsf{tt})}\,\mathrm{WR}_1$$

Rich specifications serve as a contract between hardware designers and compiler writers and enable verified designs and verified compilation.

One-rule-at-a-time abstraction

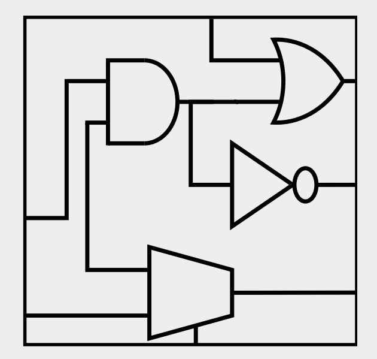
```
Theorem OneRuleAtATime init schedule:
  exists rules ⊂ schedule,
    interp_s init schedule =
    foldl interp_r init rules.
```



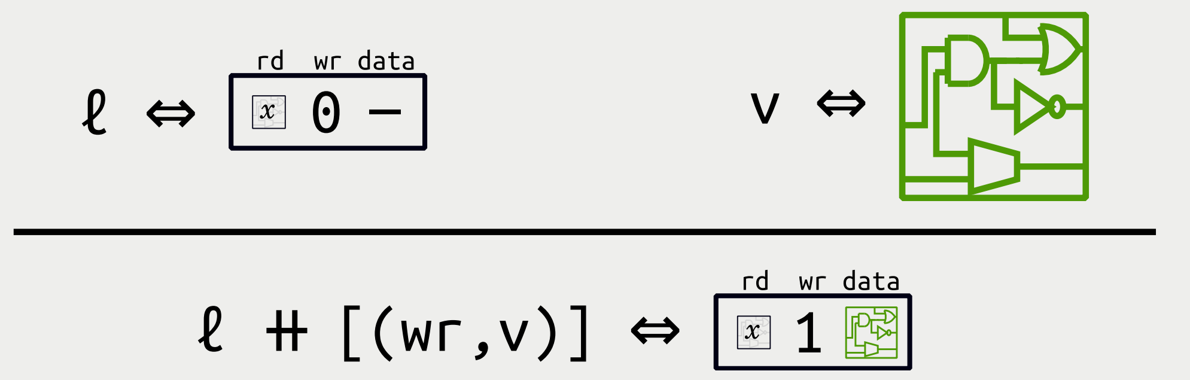
ORAAT property enables local, modular reasoning

Proof mechanization enables semantics exploration

Compilation to RTL



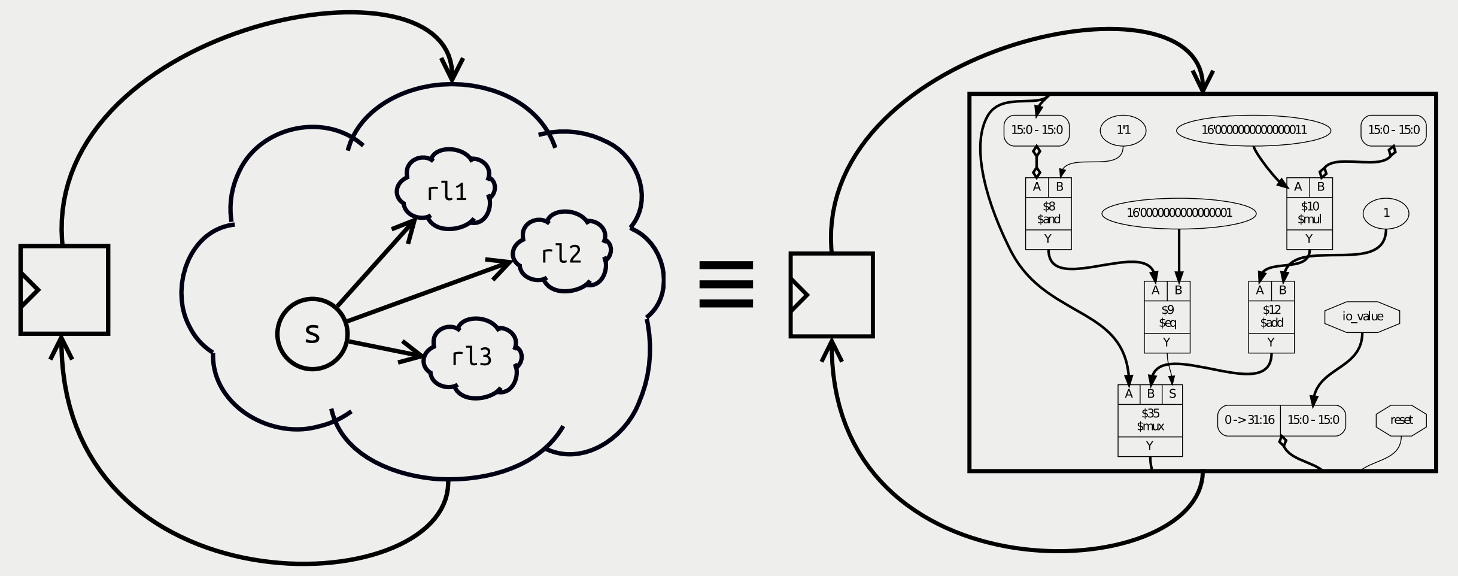
Sequential semantics
↓
Concurrent RTL with deferred checks



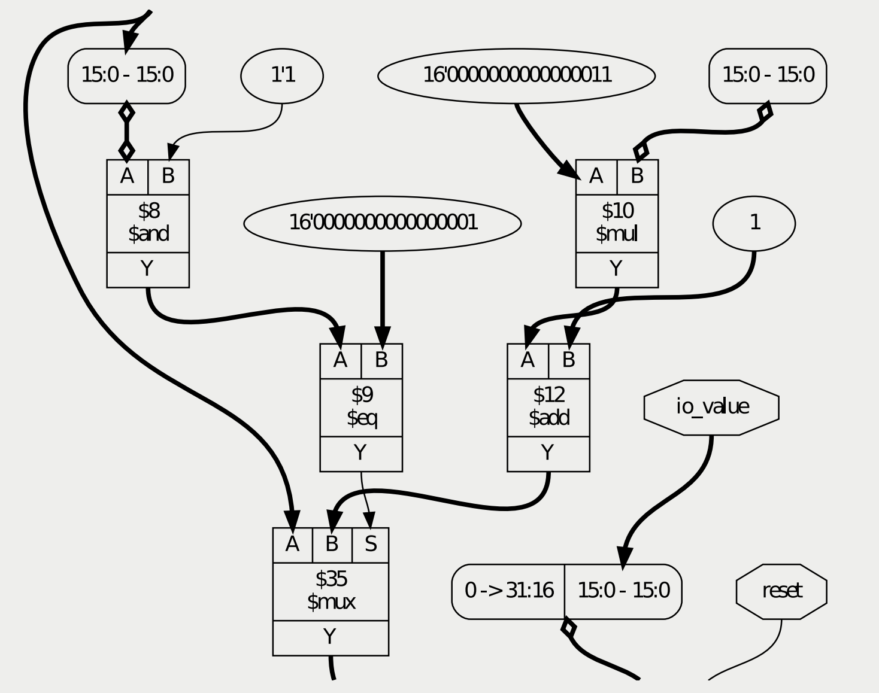
Verified circuit optimizations (constant propagation, mux elimination, partial evaluation) reduce the overhead of dynamic scheduling & conflict resolution.

Compiler correctness

```
Theorem CompilerCorrectness init schedule:
  interp_s   init schedule =
  interp_rtl init (compile schedule).
```



Output and downstream synthesis



- Kôika outputs code in a safe subset of Verilog.
- Downstream tools perform further optimizations and can generate FPGA and ASICs designs.

Hardware artifacts so far

- Arithmetic-pipeline performance proof (on paper)
- Kôika port of a simple BSV RISCV core (most of RV32i&e, critical path and area overhead ≤ 10%)

```
18:30:21 /home/clement/documents/mit/plv/koika/examples/rv
$ _objects/rv32i.v/obj_dir.opt/Vtop +VMH=tests/_build/rv32i/unit/pldi.vmh -1
```

Next steps

- Kôika-level simulation, leveraging high-level structure for performance
- Verification of performance (pipelining behavior) and timing properties of the RISCV core
- Multi-core systems & enclaves, with proofs of safety from timing side-channels
- Further language design, including native modularity