`rtlv`: push-button verification of software on hardware

CARRV 2021

Noah Moroze
Anish Athalye
Frans Kaashoek
Nickolai Zeldovich

MIT CSAIL

Formal verification

- *Proves* that a system satisfies a certain property
  - Distinct from traditional “verification” or testing
- Increases confidence in security and correctness

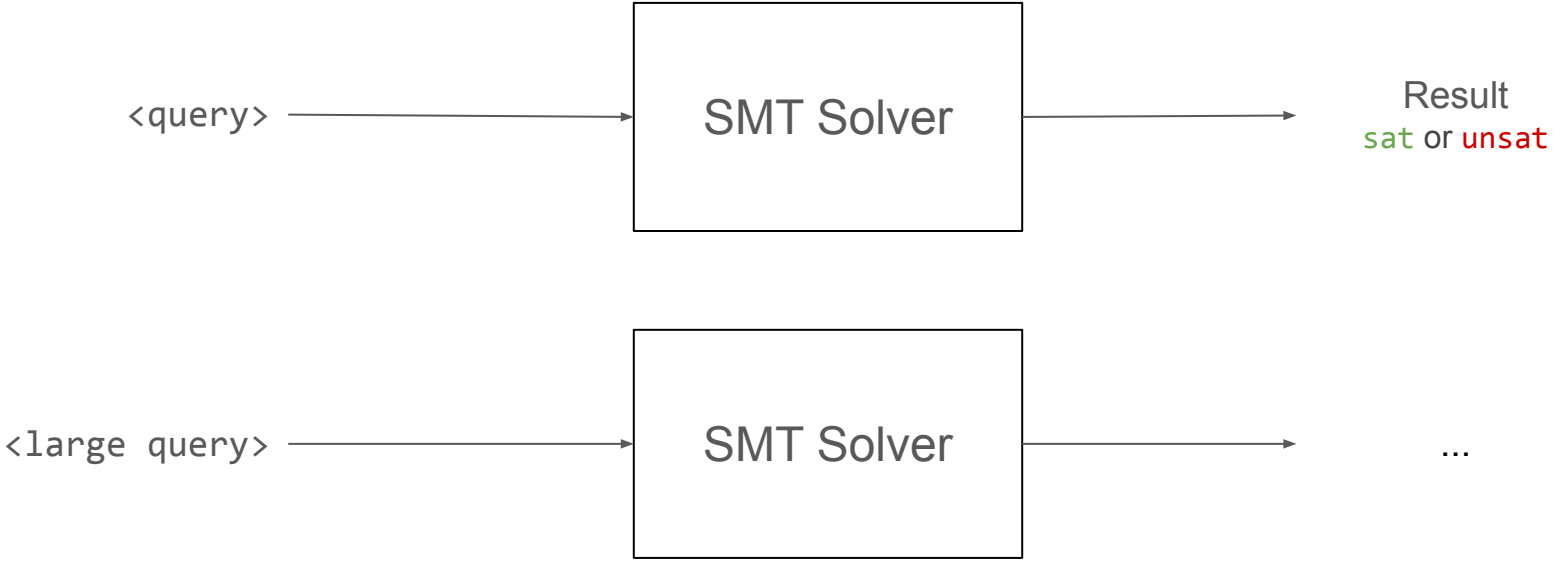
Current verification approach

- SymbiYosys is a popular open-source hardware formal verification tool
- Developers express properties using SystemVerilog Assertion (SVA) constructs

```
module counter (
    input clk,
    input rst_n,
    output reg [1:0] out
);

 always @(posedge clk) begin
   if (!rst_n || out == 2'd2) begin
     out <= 2'd0;
   end else begin
     out <= out + 1'b1;
   end
 end

 assert(out != 2'd3);

endmodule
```

SMT-based formal verification

- Circuit logic and property encoded into SMT query, evaluated by SMT solver
- Complex queries can be a performance bottleneck
  - Solver may take a long time, or time out completely!



Limitation: reasoning about software on hardware

- Want to prove that a CPU's state is cleared by boot code (*deterministic start*)
- Requires reasoning about execution of software on hardware
- SymbiYosys inherently produces complex SMT queries when reasoning about long executions
- Cannot be done by existing tools: we need a new approach!

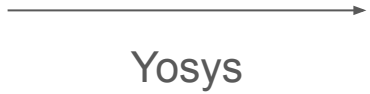
`rtlv`

- `rtlv` is an approach for verifying circuits that enables reasoning about many cycles of execution
- It achieves this by:
  - Using symbolic execution to generate more efficient verification queries
  - Supporting *circuit-agnostic property checkers* that enable disciplined performance optimizations

Generating efficient verification queries

SymbiYosys' SMT query generation

```
module foo (
    input clk,
    input rst_n,
    output reg out
);

 always @(posedge clk) begin
   if (!rst_n) begin
     out <= 1'b0;
   end else begin
     out <= ~out;
   end
 end

endmodule
```




```
foo_t(state, next_state):
  return
    next_state.out ==
    (rst_n ? !state.out : 0)
```

Verilog **SMT-LIB Relation**

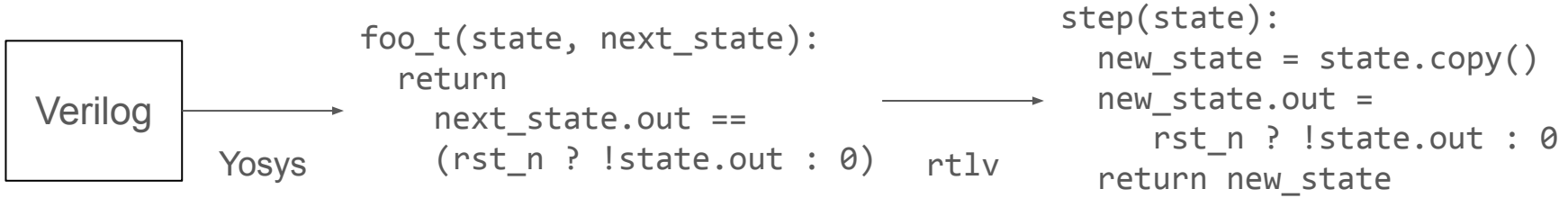
SymbiYosys' SMT query generation

- Want to prove that a property *P(s)* holds for some state *s*
- Construct *n* symbolic circuit states $s_0, s_1, \ldots s_n$
- Constrain s_0 to be a possible initial state ($I(s_0)$ is true)
- Can use transition relation $T(s_a, s_b)$ to encode execution in query

$$\text{assert}(I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{n-1}, s_n) \wedge \neg P(s_n))$$

SMT solvers are bad at reasoning about unrolled complex execution

rtlv approach



SMT-LIB relation

Step function

Generating SMT query through symbolic execution

$$\text{assume}(I(s_0))$$

$$s_1 = \text{step}(s_0, \text{rst}=1)$$

$$s_2 = \text{step}(s_1, \text{rst}=0)$$

$$\dots$$

$$s_n = \text{step}(s_{n-1}, \text{rst}=0)$$

$$\text{assert}(\neg P(s_n))$$

Execution modelled by Rosette's symbolic execution system

Query does not encode execution logic, and therefore does not inherently scale with execution length

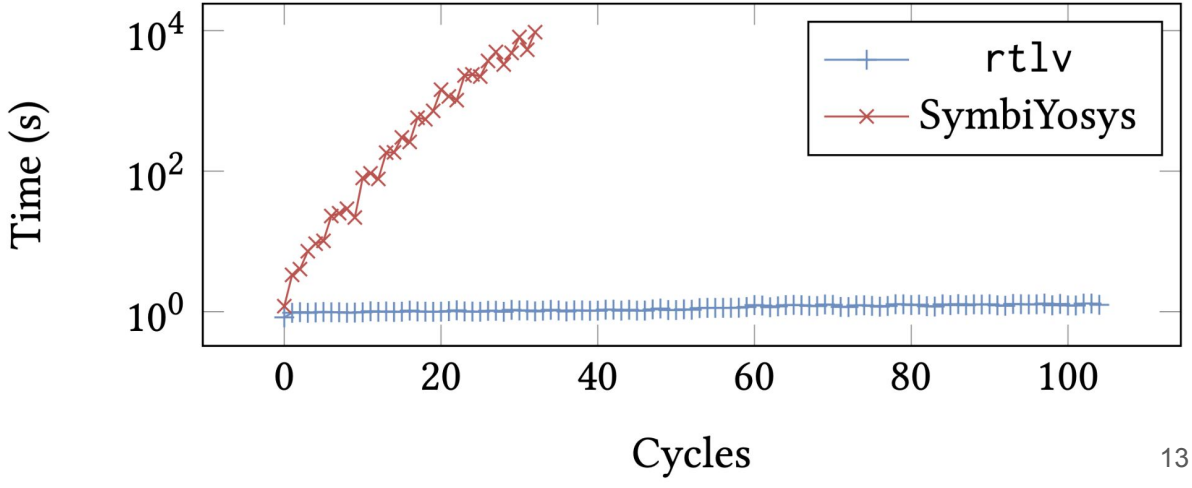
Good applications

- Although final query no longer inherently scales linearly with n, query is only fast if s_n is small
- Some applications are better for this:
  - Lots of concrete state (e.g. known code with concrete control flow)
  - Deterministic start is a great example!

SymbiYosys SMT solver call times out

- Experiment: verify deterministic start for PicoRV32 CPU with SymbiYosys and `rtlv`
- Boot code takes 104 cycles to execute
- SymbiYosys times out after 12 hours, `rtlv` finishes in 1.3 seconds
- Problem goes from intractable to fast!

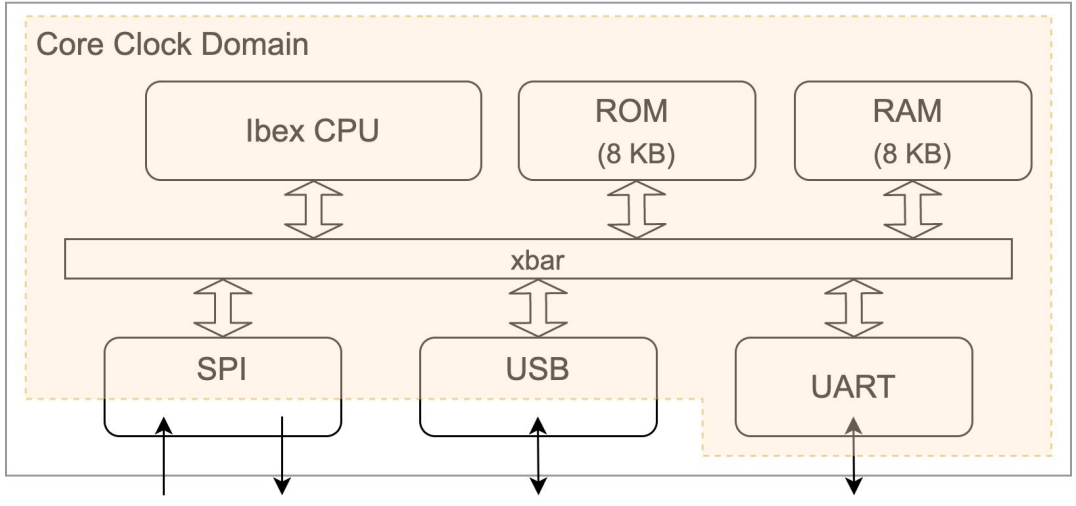
Partial deterministic start experiment



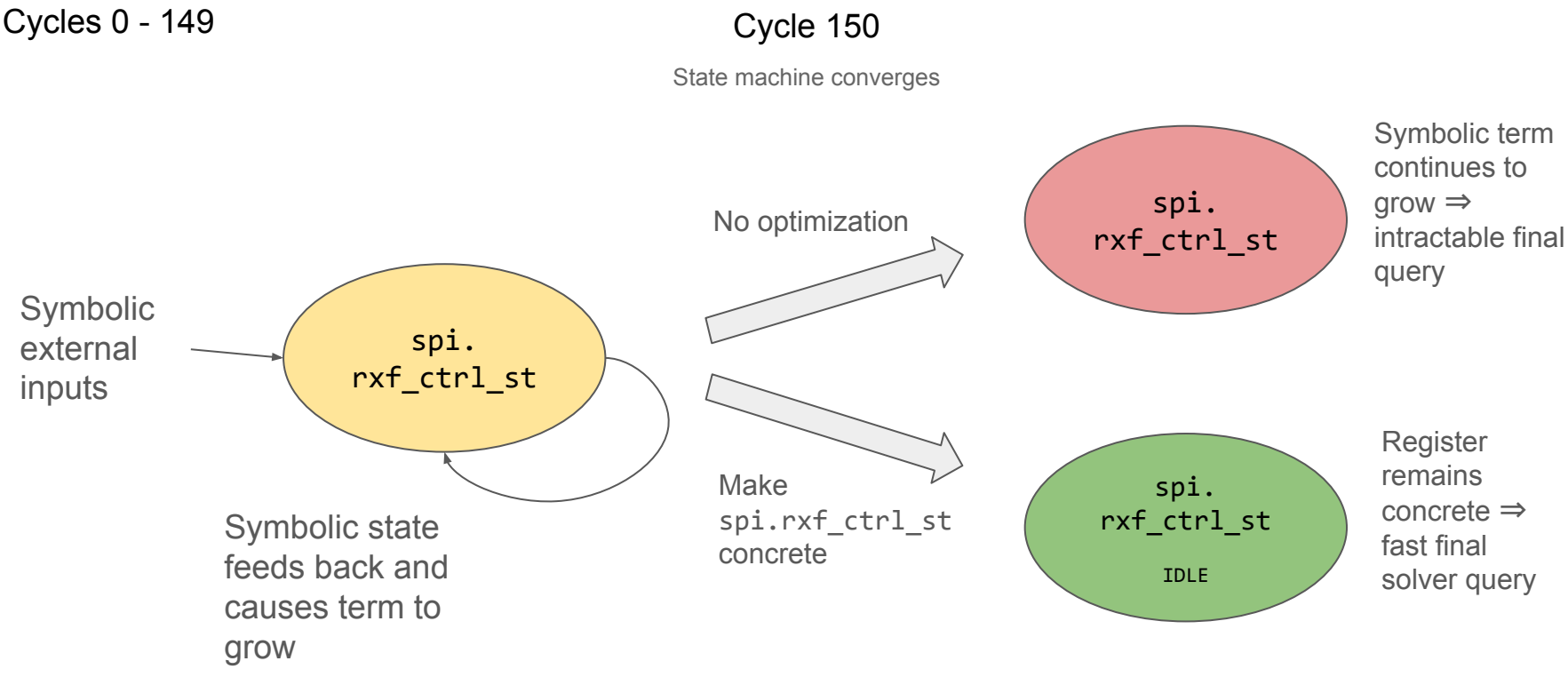
Optimizing verification of complex hardware using property checkers

Case Study: MicroTitan

- Subset of OpenTitan RISC-V SoC
- Hardware complexity:
  4,300 FFs (~3x PicoRV32)
- Boot code length:
  24,516 cycles (~235x)



Scaling problem: SPI state machine



Circuit-agnostic property checkers

- `rtlv/shiva` is *sound*: changing the list of hints cannot make the verification tool incorrectly return 'true'
- Ensured through auxiliary solver queries
  - E.g. `concretize` hint uses SMT solver to determine if field evaluates to concrete value
- Existing tools do not support this approach, since they don't provide direct access to solver

Results

- Verification runtime: 100 minutes
- Violations found:
  - SPI TX leak
  - SPI RX leak
  - USB buffer leak
  - Reset line leak
- Contributed back to OpenTitan: https://github.com/lowRISC/opentitan/pull/2420

Limitations

- Circuit features
  - No latches
  - No async structures
- Complexity of HW/SW
  - Expect we can verify 100s of lines of C on microcontroller
  - Do not expect that current system can verify full OS

Summary

- `rtlv` is an approach for verifying long executions on hardware
- Two key ideas:
  - Symbolically execute circuit using Rosette to generate efficient solver queries
  - Develop circuit-agnostic property checkers to optimize and scale verification in a disciplined way
- `rtlv` enables you to verify properties that can't otherwise be verified using open and accessible tools!

Thank you!

- PicoRV32 comparison case study: https://github.com/anishathalye/deterministic-start-benchmark
- MicroTitan case study: https://github.com/nmoroze/kronos
  - Blog post: https://noahmoroze.com/kronos.html
  - Thesis: https://pdos.csail.mit.edu/papers/moroze-meng.pdf