Introduction to Formal Verification of Digital Circuits

Cesar Strauss

FOSDEM 2024

Why Formal Verification?

- A tool for finding bugs
- Complementary to simulation
- Helps finding corner cases
- . . . triggered by specific sequences of events

Comparison with traditional debugging concepts

formal	traditional
Cover	Simulation
Bounded Model Check	Unit test
k-Induction	Test fixture?
Exhaustive search	random test cases
synthesizable test-bench	can be procedural
“assume” inputs	test vectors
“assert” outputs	“assert” outputs

Workflow

- HDL: includes assertions
- SBY: work plan, drives the process
- Yosys: synthesizes to logic functions:
  - state s: contents of all registers and inputs
  - initial predicate: $I(s)$
  - transition relation $T(s_1, s_2)$
  - assertions: $P(s)$
- yosys-smtbmc: proves correctness or outputs a trace
  - exhaustive search for a path from the initial state to a bad state
  - if not found, the design is correct
  - if found, output an error trace

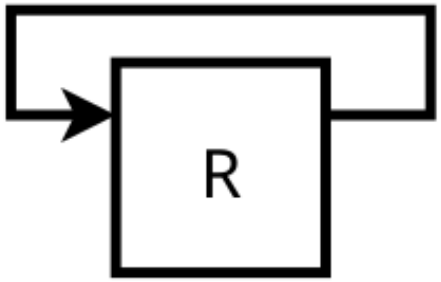
Unbounded inductive proof

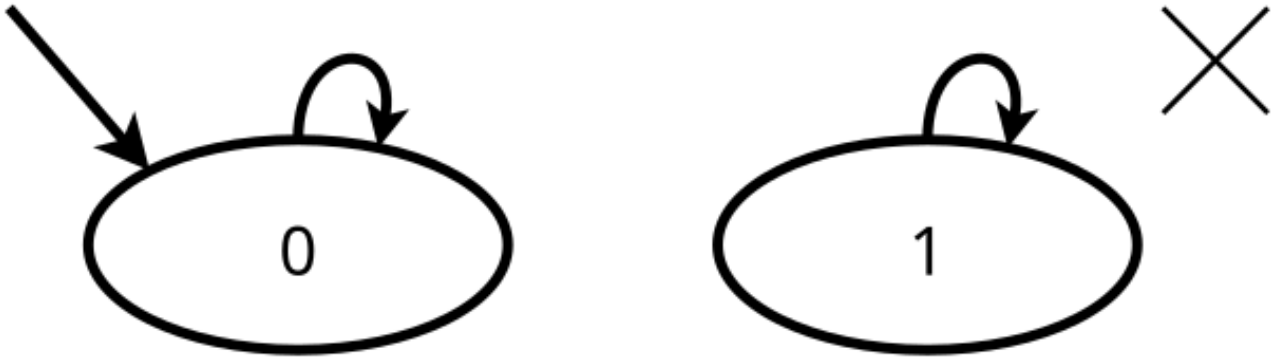
- bad trace:

$$I(s_0)P(s_0) \wedge T(s_0, s_1)P(s_1) \wedge \cdots \wedge T(s_{k-1}, s_k)\overline{P(s_k)}$$

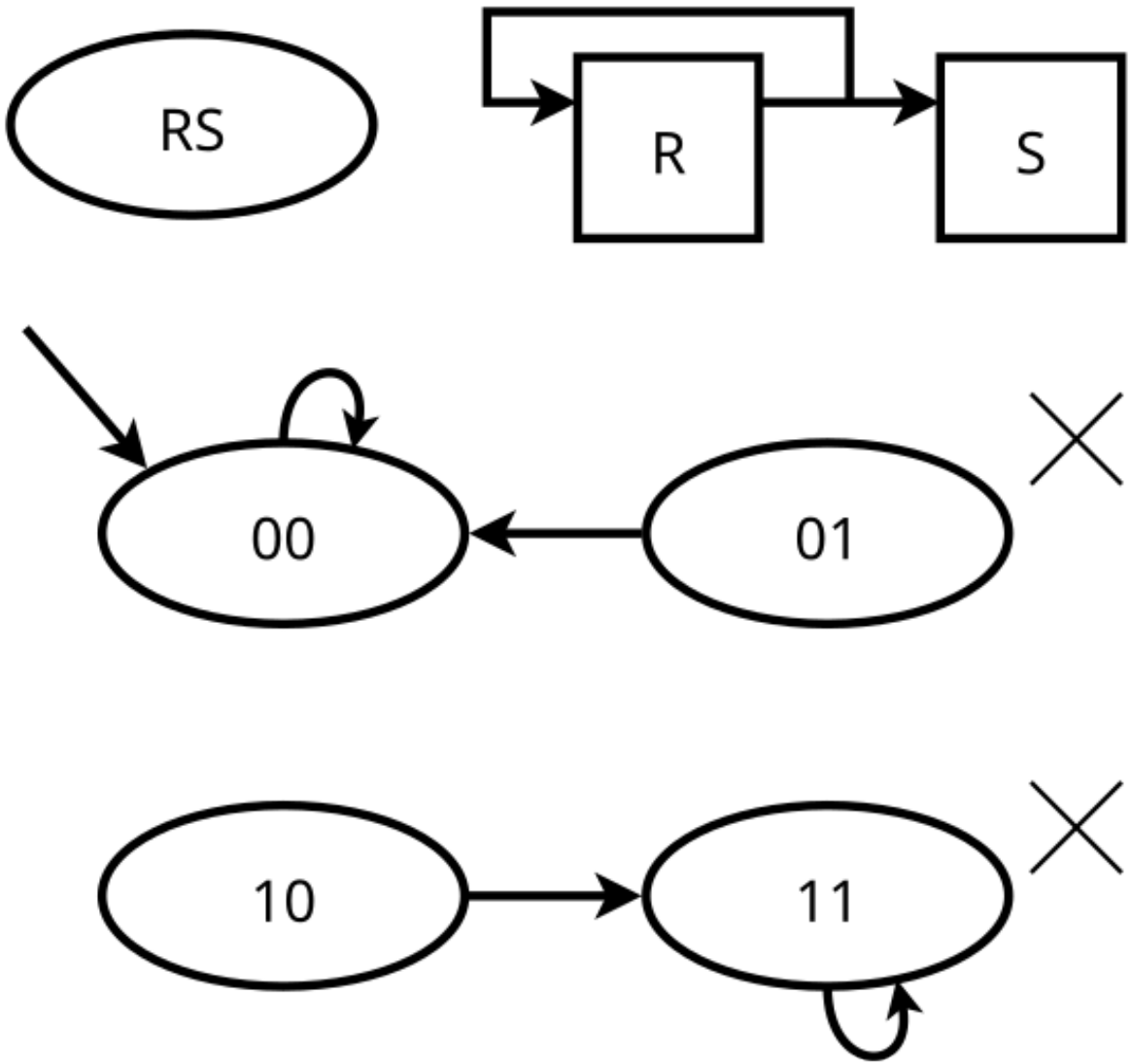
- k $\leftarrow$ 0
- base case: no path from initial state leads to a bad state in k steps
- if base case fails, report the bad trace
- inductive case: no path ending in a bad state can be reached in k+1 steps
- if inductive case fails, $k \leftarrow k + 1$ and repeat
- otherwise, proof is complete, circuit is safe.

Single register with feedback

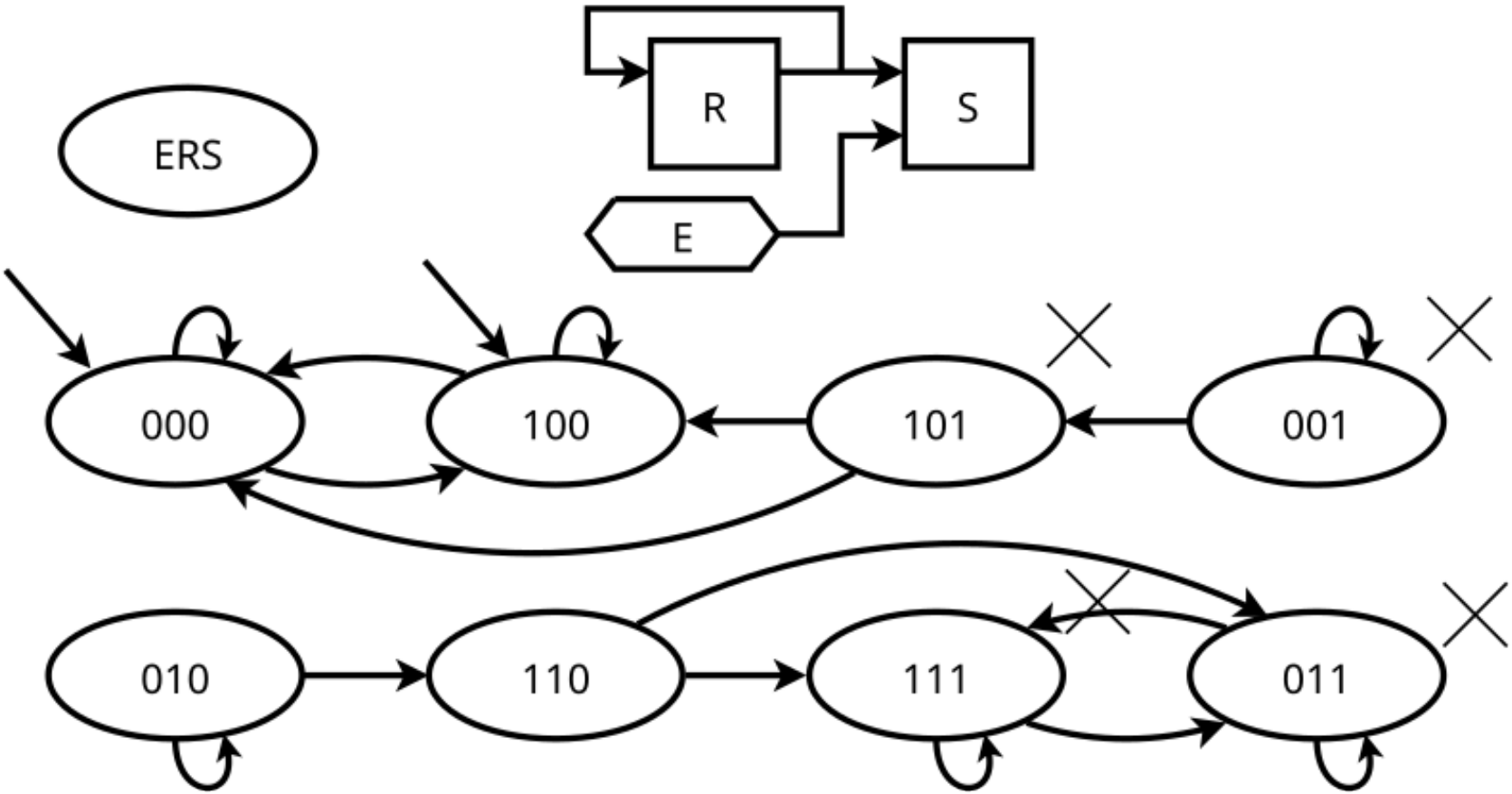




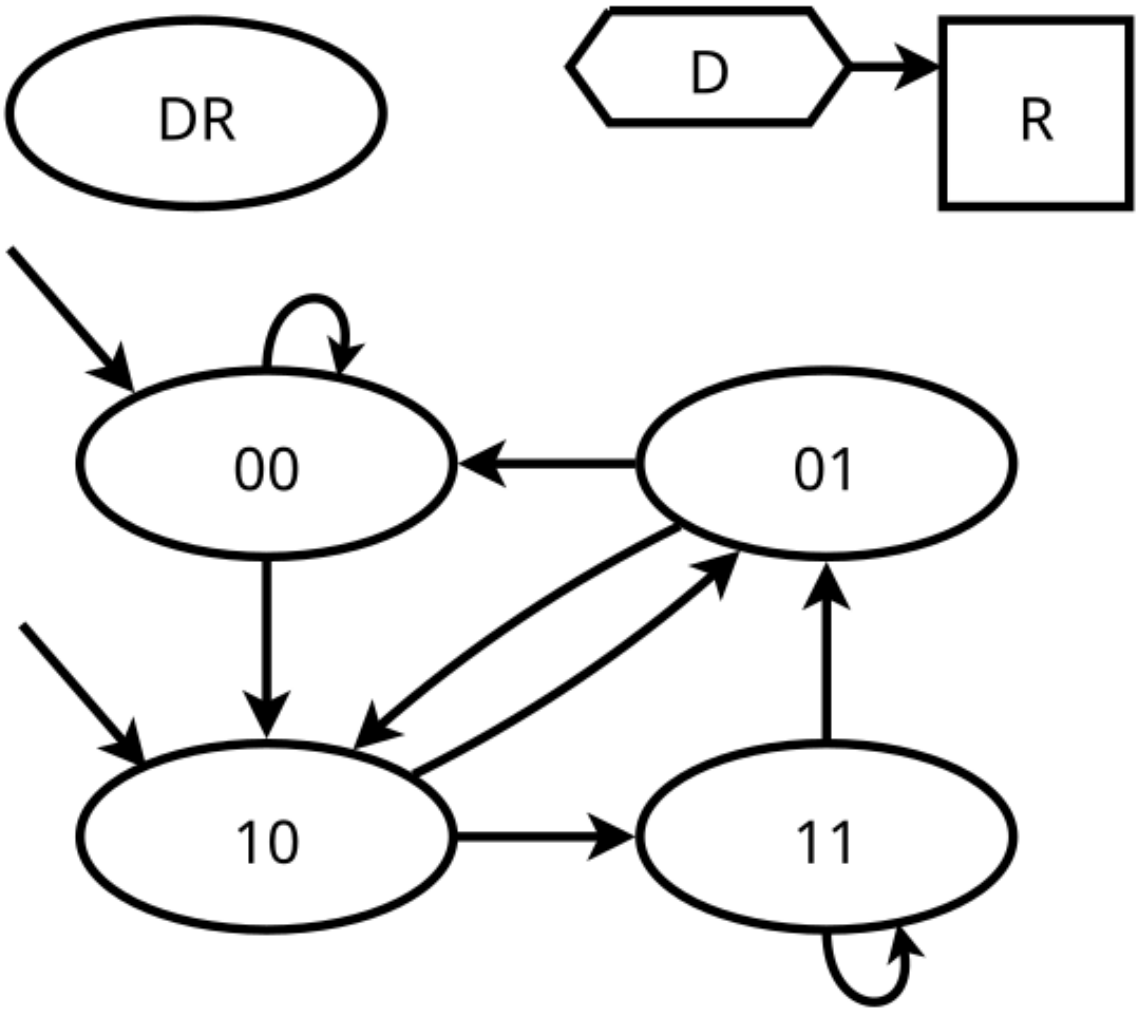
Registered output with internal state



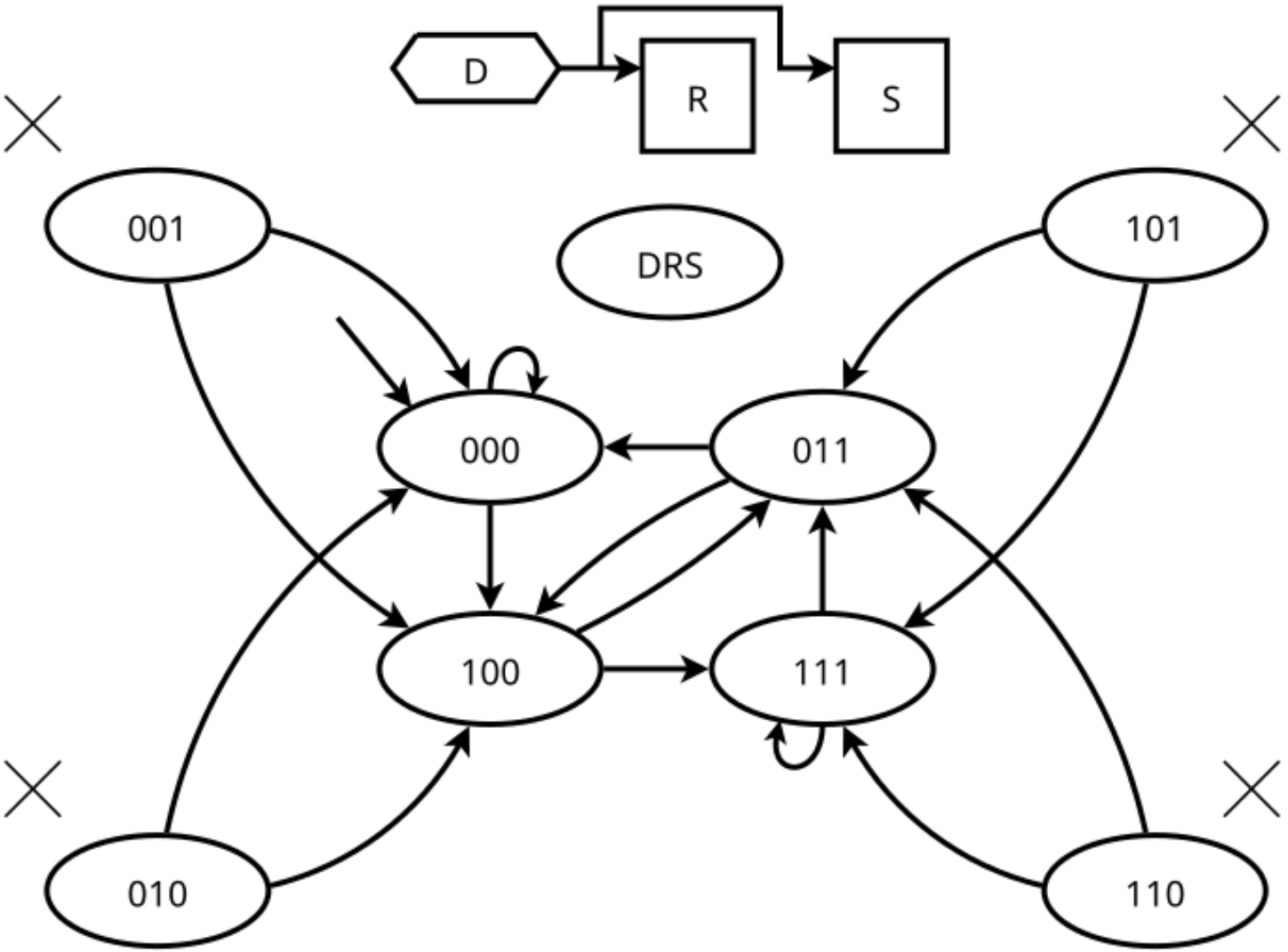
Registered output with enable



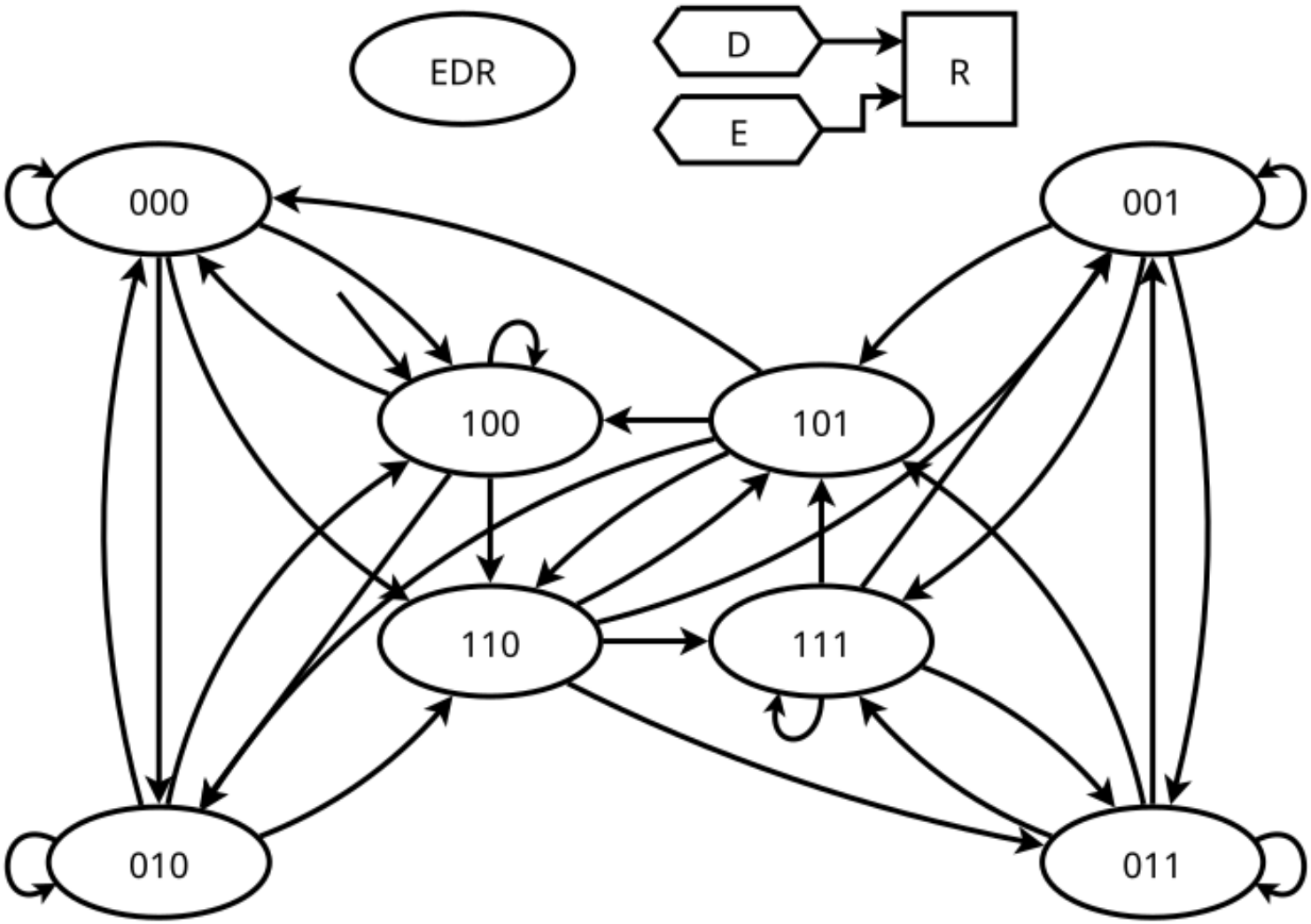
Flip-flop with input



Verifying a flip-flop



Complete flip-flop with input and enable



Code for simple register with feedback

```
module simple(input clk);

reg r = 0;

always @(posedge clk)
    r <= r;

`ifdef FORMAL
always @*
    assert(!r);
`endif
```

SBY drive file

```
[options]
mode prove
depth 1

[engines]
smtbmc yices

[script]
read_verilog -formal simple.v
prep -top simple

[files]
simple.v
```

Output (simplified)

```
$ sby simple.sby

induction: Trying induction in step 1..
induction: Trying induction in step 0..
induction: Temporal induction successful.
basecase: Checking assumptions in step 0..
basecase: Checking assertions in step 0..
basecase: Status: passed
summary: engine_0 (smtbmc yices) returned pass
 for induction
summary: engine_0 (smtbmc yices) returned pass
 for basecase
summary: successful proof by k-induction.
DONE (PASS, rc=0)
```

Flip flop with enable (1/2)

```
from nmigen.asserts import Assert, Assume, Past
from nmutil.formaltest import FHDLTestCase
from nmigen import Signal, Module
import unittest

class Formal(FHDLTestCase):
    def test_enable(self):
        m = Module()
        r1 = Signal()
        r2 = Signal()
        s = Signal()
        en = Signal()
        m.d.sync += [r2.eq(r1), r1.eq(r2)]
        with m.If(en):
            m.d.sync += s.eq(r1 & r2)
```

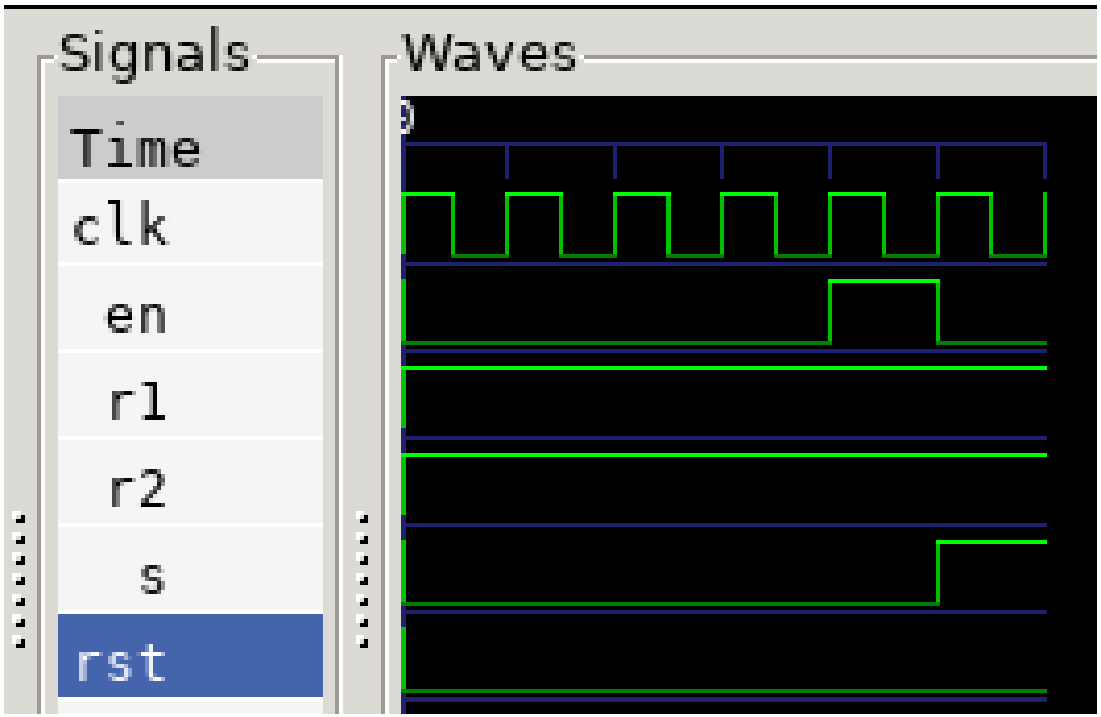
Flip flop with enable (2/2)

```
        m.d.comb += Assert(~s)
        m.d.sync += Assume(Past(en) | en)
        m.d.comb += Assert(~r1 & ~r2)
        self.assertFormal(m, mode="prove", depth=5)
```

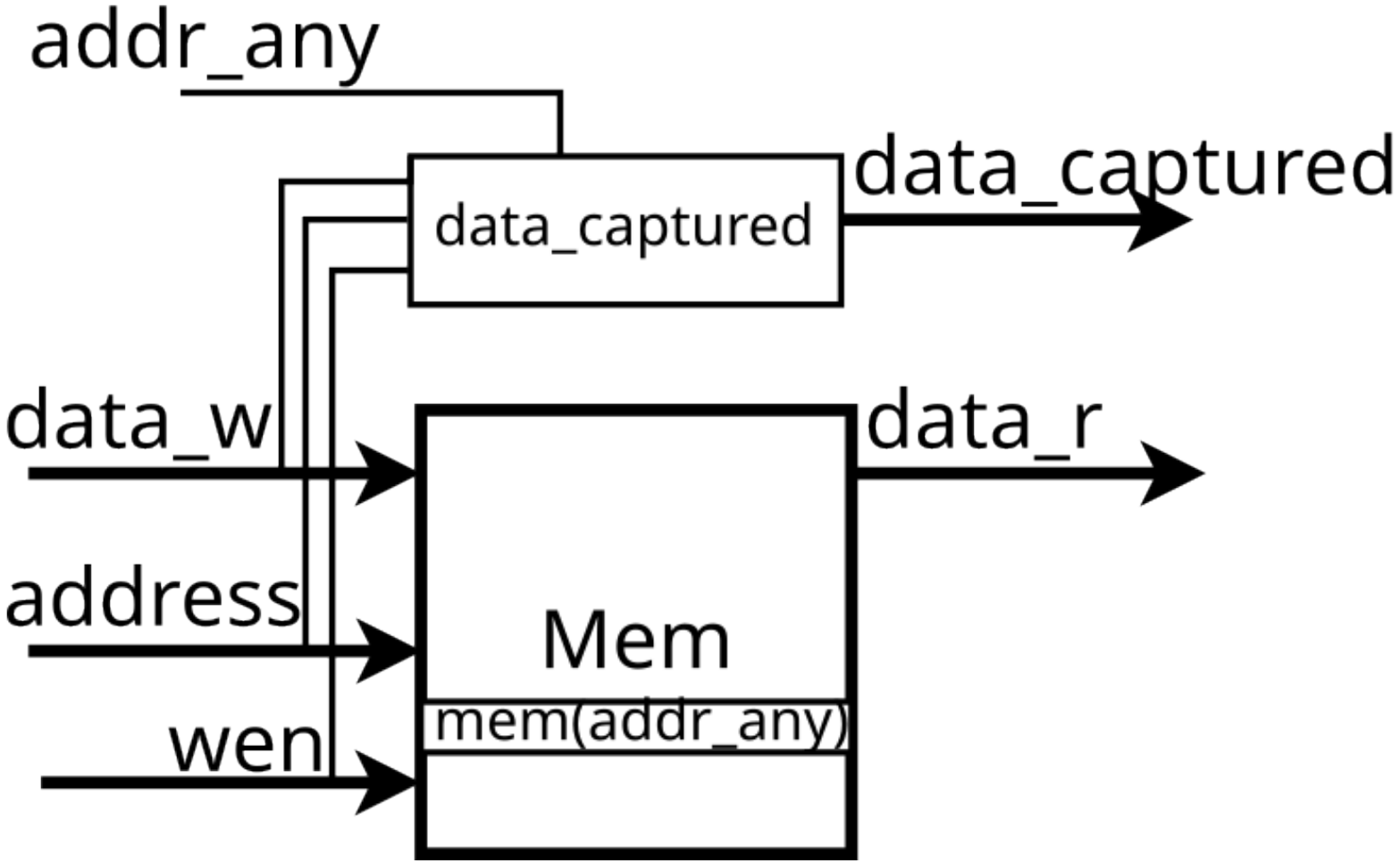
```
if __name__ == '__main__':
    unittest.main()
```

Induction failure example

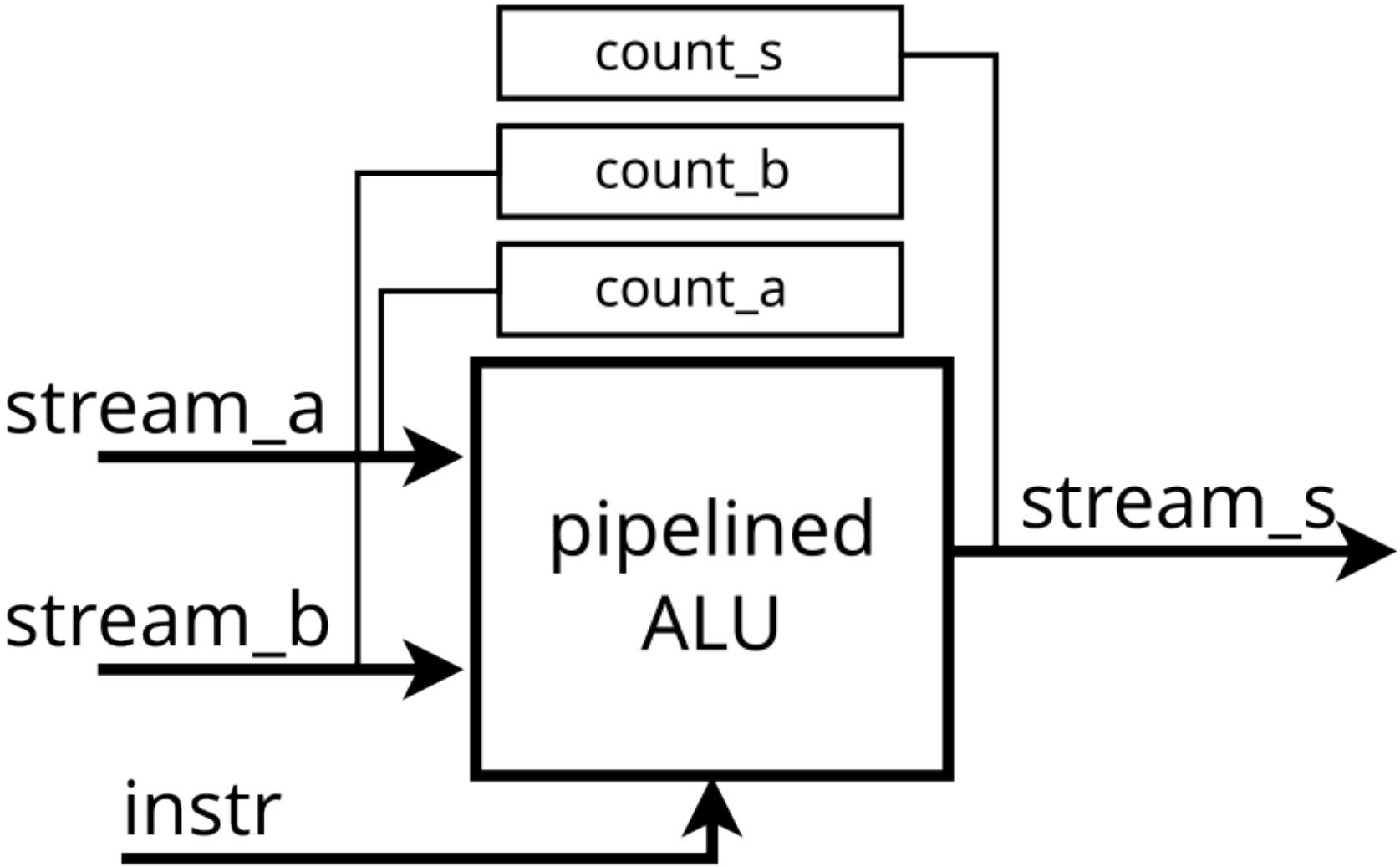
```
summary: engine_0 returned pass for basecase
summary: engine_0 returned FAIL for induction
DONE (UNKNOWN, rc=4)
```



Verifying memories with a “victim address”

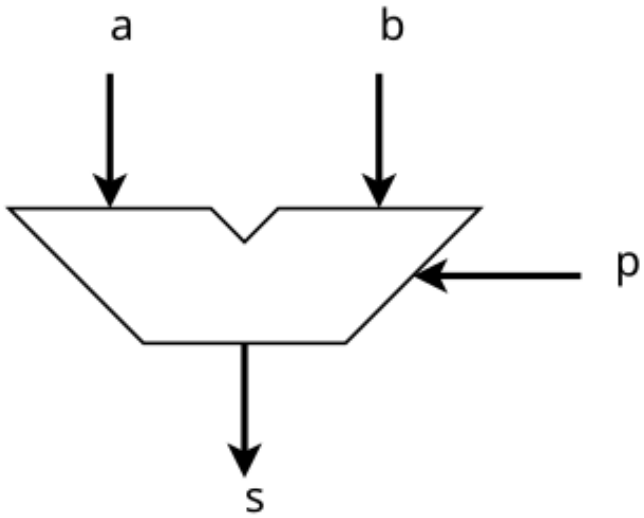


Verifying streams with transaction counters



Dynamic SIMD

```
exp-a    : ....0....0....0....  1x 32-bit
exp-a    : ....0....0....1....  1x 24-bit plus 1x 8-bit
exp-a    : ....0....1....0....  2x 16-bit
...
...
exp-a    : ....1....1....0....  2x 8-bit, 1x 16-bit
exp-a    : ....1....1....1....  4x 8-bit
```



The End

Thank you

Questions?

- Discussion: http://lists.libre-soc.org
- Libera IRC #libre-soc
- http://libre-soc.org/
- https://libre-soc.org/resources/
- http://nlnet.nl/entrust
- https://libre-soc.org/nlnet_2022_ongoing/
- https://libre-soc.org/nlnet/#faq
- https://git.libre-soc.org/?p=soc.git;a=tree;f=src/soc/experiment/formal;hb=HEAD