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

<table>
<thead>
<tr>
<th>formal</th>
<th>traditional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cover</td>
<td>Simulation</td>
</tr>
<tr>
<td>Bounded Model Check</td>
<td>Unit test</td>
</tr>
<tr>
<td>k-Induction</td>
<td>Test fixture?</td>
</tr>
<tr>
<td>Exhaustive search</td>
<td>random test cases</td>
</tr>
<tr>
<td>synthesizable test-bench</td>
<td>can be procedural</td>
</tr>
<tr>
<td>“assume” inputs</td>
<td>test vectors</td>
</tr>
<tr>
<td>“assert” outputs</td>
<td>“assert” outputs</td>
</tr>
</tbody>
</table>
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:
  $$I(s_0)P(s_0) \land T(s_0, s_1)\overline{I(s_1)}P(s_1) \land \cdots \land T(s_{k-1}, s_k)\overline{I(s_k)}P(s_k)$$

- \(k \leftarrow 0\)
- base case: no path from initial state leads to a bad state in \(k\) steps
- 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
Single register with feedback
Registered output with internal state
Registered output with enable
Verifying a flip-flop

Cesar Strauss

Introduction to Formal Verification of Digital Circuits
Complete flip-flop with input and enable
module simple(input clk);

reg r = 0;

always @(posedge clk)
    r <= r;

`ifdef FORMAL
always @*
    assert(!r);
`endif
[options]
mode prove
depth 1

[engines]
smtbmc yices

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

[files]
simple.v
\$ \texttt{sby simple.sby}

\textit{induction}: Trying induction in step 1..
\textit{induction}: Trying induction in step 0..
\textit{induction}: Temporal induction successful.
\textit{basecase}: Checking assumptions in step 0..
\textit{basecase}: Checking assertions in step 0..
\textit{basecase}: Status: passed
\textit{summary}: \texttt{engine_0 (smtbmc yices)} returned pass
  for induction
\textit{summary}: \texttt{engine_0 (smtbmc yices)} returned pass
  for basecase
\textit{summary}: successful proof by k-induction.
\textit{DONE (PASS, rc=0)}
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)
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()
summary: engine_0 returned pass for basecase
summary: engine_0 returned FAIL for induction
DONE (UNKNOWN, rc=4)
The End
Thank you
Questions?

- Discussion: http://lists.libre-soc.org
- Libera IRC #libre-soc
- http://libre-soc.org/
- 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