Open Source Formal Verification with PSL

Coverage

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

July 2025

Coverage

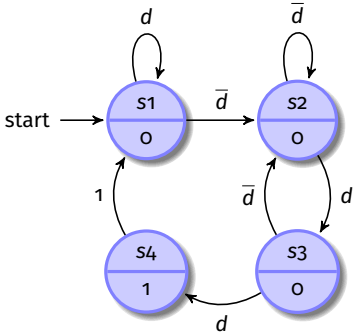
- Coverage is the concept of knowing if we observed a certain number of sequences, states, or values
- Can be applied in:
  - Simulation
  - Formal verification

`cover` directive

- It can be interesting to know if a certain sequence has been observed
- The operator `cover` is just what we need
- Example:
  - `cover {a;b;a} report "Seen";` displays a message when the sequence `a` followed by `b` and then again `a` is observed

State machine - transitions

- If we have access to the internal architecture
- Ensure coverage of expected transitions



Cover allowed transitions

```
cover {state = s1; state = s1};
cover {state = s1; state = s2};
cover {state = s2; state = s2};
cover {state = s2; state = s3};
cover {state = s3; state = s2};
cover {state = s3; state = s4};
cover {state = s4; state = s1};
```

`assert` and coverage

- With SBY, running with the `cover` mode transforms the assertions into coverage points
  - It will cover the left-hand side of implications

Example

```
assert always ({d;d;not d;not d} |-> detected) abort rst;
assert always (a -> b) abort rst;
```

- Will check if the sequence `d;d;not d;not d` is observed, and if `a` went high
- Not particularely useful on unconstrained inputs
- But useful for outputs and constrained inputs
  - Maybe the assumptions were too strong

Coverage - When?

- Coverage is there to know if we observed interesting behaviors or not
- To be used after the design has been proven correct
  - Maybe some assertions did not trigger because their left-hand side was never observed
- With SBY:
  - `sby -yosys "yosys -m ghdl" -f run.sby cover`
  - If it fails, display the covers or assertions that where not covered
    - Useful to label the `cover` directives

A parenthesis for simulations

Coverage in simulation

- Cover statements are also useful in simulation
- Can drive random-based coverage-driven verification
- Can detect that some sequences of events have been observed during simulation

Code coverage (1)

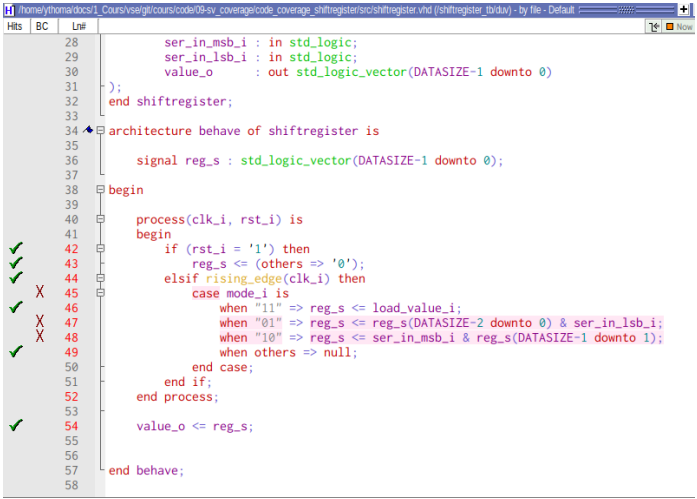
- In simulation, most simulators allow code coverage analysis
  - Have all the lines of code been executed?
  - Have all the conditional branches been observed in both choices?

Questasim

```
vcom +cover myfile.vhd
vsim -coverage work.myfile_tb
```

- Can be visualized in Questasim
- Allows to compute a global coverage over various scenarios and the Verification Run manager

Code coverage (2)



Code coverage (3)

- Goal:
  - 100% of code coverage on the design files
- Do not add the tesbenches

```
if (observed /= expected) then
    report "Huge mistake";   <-- We expect this line not to be reached
end if;
```

- Except if you have testbenches with built-in self-tests
  - Need a way to inject errors in the observed values

Conclusion

- Coverage is key for ensuring verification is complete
  - Well, as complete as possible
- In simulation
  - Activate code coverage
  - Use functional coverage when possible (OSVVM or SV coverage)
- In formal
  - Check coverage to ensure the assertions can be observed
    - Do this when all assertions are checked
    - It does not imply you did check enough
    - There is no real way of being sure we had enough assertions
    - Actually if we implement all the possible properties we could synthesize a module based on the properties
    - Quite a dream for real systems
    - Try to identify the relevant ones to complement your simulation-based verification