Coverage completeness

Ofer Cohen

Cover what ?

- Spec
  - Expected scenarios
  - Unexpected scenarios
  - Configuration
- Interfaces
- Scoreboard / checkers / assertions
- HDL code coverage
- Designer specific requests

Were should the coverage be placed ?

- Interfaces coverage in monitors.
- Scenarios should be covered in the generation or scoreboard/checkers.
- HDL code coverage is automatically.
- Assertions / checkers coverage should be created automatically.
- Designer specific requests usually using “HILHULIM” should be placed in different file or under “define”

Who should be involved in the coverage definition ?

- Verification designer, based on specification and interfaces definition.
- HDL designer, for implementation-related coverage focus and specific coverage items.
- System architect or other, with zoomed-out point of view, mostly for defining unexpected scenarios.

Completeness killing

- Large crosses.
  - Try to avoid them
  - If Holes remain, try to review the cross using code coverage results, and determine if all items of the cross are truly necessary.
- Match coverage expectation to the Verification environment scope (Don’t expect system level coverage to be the same as block level). Define well what will be covered in which environment.

Completeness killing cont.

- Designers tend to add capabilities to their block in order to make it more robust (behavior in wrong configuration scenarios, neighbor blocks error defense,…) . Sometimes it is wise to avoid checking/covering these capabilities.
- Random environment may not always be easily manipulated to produce directed tests. When writing environment always think how easily it can be manipulated to create specific scenarios.

Coverage completeness flow

(based on simulator only verification environment)

1. Reach 95% + of functional Paths/ features (not cross items or special scenarios)
2. 90% + of checks coverage.
3. Run code coverage and if necessary update functional coverage plan .
4. Reach 100% of functional Paths/ features + 95% of all functional coverage + 100% Code coverage. (or good excuse why not)

Coverage completeness flow cont.

5. Review cross coverage tables with holes according to spec and code coverage results.
6. 6 Define verification end-point, re-calculate risks of uncovered holes and act accordingly.

Keep in mind that in engineering there is nothing as 100% good,
only good enough.

Open issues

- When verifying design using FPGA or other non-simulator environments, code coverage need to be checked only for the “simulator based verification” and this may not easily distinguished.