Environment Completeness

Akiva – Ace Verification

Content

- How to assure your verification environment is correct/complete.
- Do you have enough checkers?
- Are your stimulus correct?
- Are you flexible enough?

Checking the check's check

- Like in simulation there is always more which could be done but, there is a point of diminishing returns

Solution: Define and rank methods

- Define all of the “checks and balances” we have in place and rank them for greatest ROI

Pre-emptive Methods

- Methodical Definition of Functional coverage
  - Line by line in the Spec (each control, each status bit etc.)
- Methodical Definition of Assertions and architecture checks
  - Are there good guidelines?
  - Checks overlap
- Define functional coverage to overlap
  - Multiple perspectives (RTL, Design)

Review Methods

- Review Definition of Functional Coverage
  - With designer, verification eng., and architect
- Review checks for completeness
- Review coding of the verification environment
- Review coding of functional coverage
- Review coding of assertions and checks

Observation methods

- Assertion Density
  - How many per module/per 1000 lines of code
- Bug evaluation
  - Why wasn't it also caught by architecture, assertion, test check etc.
- Assertion Coverage
  - Should every assertion's primary event be triggered

Duplication Methods

- Have two separate environments for each functionality
  - Keep a firewall between them
  - Examples
    - Block, Full-chip
    - Full-chip and FPGA
- Have two engineers define functional coverage and/or checks for the same module

Other Suggestions?

Let's take a vote

1. Methodical Definition of Functional coverage
2. Methodical Definition of Assertions and architecture checks
3. Define functional coverage to overlap
4. Review Definition of Functional Coverage
5. Review checks for completeness
6. Review coding of the verification environment
7. Review coding of functional coverage
8. Review coding of assertions and checks
9. Assertion Density
10. Bug evaluation
11. Assertion Coverage
12. Have two separate environments for each functionality
13. Have two engineers define functional coverage and/or checks for the same module

Thanks!