Do You Know What Your Assertions Are up To?
A New Approach to Safety Critical Verification

Lee C. Smith
Agenda

• DO-254, Traceability and Coverage Overview
• Assertions Introduction
• Constrained Random Environment
• Challenges with Combining These Three
• Solving the Problem
• Questions and Discussion
DO-254

• A set of guidelines created to address the concerns of design errors in the use of complex electronic hardware in safety-critical aircraft applications
• Covers planning, design, verification and certification
• Requires more documentation and process
• Traceability and coverage are key to success
Basic DO-254 Flow & Traceability

Requirements
- SRD
- RD

Definition
- DD
- VC

Implementation
- Code
- VP

Close
- VR

Requirements Document
Verification Cases
Verification Procedures
Verification Results

Testbench code
Hardware test
Coverage

• All requirements must have verification cases
  – Known as functional coverage
  – Cases need to include corner and out-of-bounds conditions (Robustness)
• Simulation must achieve 100% code coverage
• Hardware test needs to be ‘adequate’
  – Requirements need to trace to a hardware test
  – Requirements that don’t have a hardware test need to have rationale
Assertions

• Typically used for signal relationship testing
• Has a syntactically dense language
  – Hard for reviewer to understand and follow
  – Prone to mistakes
• Defaults to generating a message only when a failure occurs

```verilog
assert property(@(posedge clk)
   (i_rst_f === 1'b0) |-> ##[0:3] (i_led === 'b01))
```
Constrained Random Environment

• Ideally, has a single test
  – Test generates random stimulus
  – Random stimulus is constrained to useful scenarios
  – Test is run many, many times
  – Results for a given verification case could be in any one of the test runs
  – Results are merged into a single database

• Not a directed test environment
  – Directed tests can be created for corner conditions
Challenges

• DO-254 requires traceability to results
• Assertions don’t generate anything unless there’s a failure
• Constrained random environments have the results... somewhere
Solving the Problem: Positive Results

- Add passing messages to the log file for assertions

```verilog
class Example{
  prop property(@(posedge clk)
     (i_rst_f === 1'b0) |-> ##[0:3] (i_led === 'b01))
  begin
    `ASSERT_MSG("PASSED: %0s","LED is 1 when i_rst_f is asserted");
  end
}
```
Solving the Problem: Results Traceability

- Create a script to search through all of the log files
  - Identify which test run has the assertion result
  - Capture the sim time when the result was logged

Where in that big pile is the assertion result?
Solving the Problem: Assertion Validation (problems)

• Printing a PASS message shows the assertion ran and passed, but doesn’t really show that it worked correctly
• A reviewer needs to look at the signals to validate the assertion code
• Because this is signal-based, there’s an expectation that the results used for certification credit will show the signals
• Creating the signal snapshots is tedious and time-consuming
Solving the Problem: Assertion Validation (solution)

• Create a script that takes the simulation result and sim time information and generates a screen shot of the appropriate assertion-related signals
  – Scrub log files for passing message and time stamp
  – Open the associated wave file in a waveform viewer
  – Add the assertion and its signals
  – Zoom the view to slightly before the assertion start to slightly after the assertion passes
  – Capture a screenshot and save to a file
Questions & Discussion