

System Verilog Assertions

LAB Material

Ashok B. Mehta

<http://www.defineview.com>

© 2006-2013

Copyright Notice

Copyright Notice

© 2006-2013

The material in this training guide is copyrighted by DefineView Consulting/Ashok B. Mehta of Los Gatos, California. All rights reserved. No material from this guide may be duplicated or transmitted by any means or in any form without the express written permission of Ashok B. Mehta

DefineView Consulting

<http://www.defineview.com>

Ashok B. Mehta

501 Pine Wood Lane

Los Gatos, CA 95032

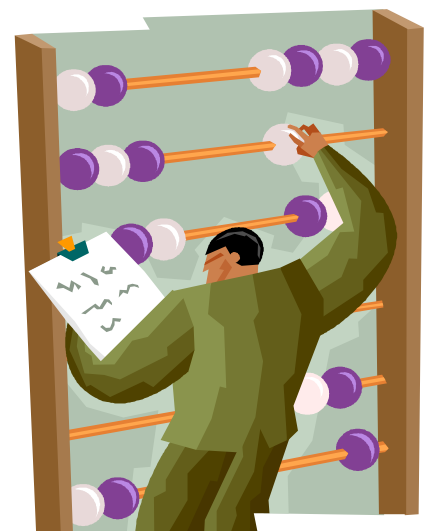
(408) 309-1556

Email: ashok@defineview.com

Verilog is a registered trademark of Cadence Design Systems, San Jose, California.

Lab 4 ...

a way to count ...



LAB 4 : COUNTER

LAB Overview

A simple UP/DOWN COUNTER design is presented. Counter assertions deployed directly at the source can greatly reduce the time to debug since these assertions will point to the exact cause of a Counter error without the need for extensive back-tracing debug when design fails.

LAB Objectives

1. *You will learn use of sampled value functions.*
2. *Alternate ways of modeling an assertion.*

LAB Design Under Test (DUT)

A simple UP/DOWN COUNTER design is presented as the DUT.

- *) The counter has 8 bit data input and 8 bit data output*
- *) When ld_cnt_ is asserted (active Low), data_in is loaded and output to data_out*
- *) When count_enb (active High) is enabled (high) and
 - *) updn_cnt is high, data_out = data_out+1;*
 - *) updn_cnt is log, data_out = data_out-1;**
- *) When count_enb is LOW, data_out = data_out;*

LAB 4 : COUNTER

LAB: Database

FILES:

1. *counter.v :: Verilog RTL for a simple counter.*
2. *counter_property.sv :: SVA file for counter properties*
This is the file in which you will add your assertions.
3. *test_counter.sv :: Testbench for the counter.*
Note the use of 'bind' in this testbench.

LAB: Assertions to Code

Code assertions to check for the following conditions in the 'counter' design.

CHECK # 1. Check that when 'rst_' is asserted (==0) that data_out == 8'b0

CHECK # 2. Check that if ld_cnt_ is deasserted (==1) and count_enb is not enabled (==0) that data_out HOLDS it's previous value.

Disable this property if rst is low.

CHECK # 3. Check that if ld_cnt_ is deasserted (==1) and count_enb is enabled (==1) that if updn_cnt==1 the count goes UP and if updn_cnt==0 the count goes DOWN.

Disable this property if rst is low.

LAB 4 : COUNTER

LAB: How to compile/simulate - step by step instructions

Follow the steps below to add your assertion for each check.

Then compile/simulate with each of your assertions and see that your results match with those stored in the ./solution directory

Here's step by step instructions...

1. % cd <myDir>/SVA_LAB/LAB4

2. First run the design without any bugs introduced in it.

% run_nobugs

- This will create the file test_counter_nobugs.log
- Study this log to familiarize yourself with how the counter works.

The remaining flow of the exercise is such that when you run any of the following steps, a specific bug is introduced in the design that your assertion should catch.

3. % vi counter_property.sv

- Look for `ifdef check1
- Remove the 'DUMMY' property and code your property as specified above for CHECK #1
- Save the file and run the following simulation.

% run_check1

- If you have coded the property correct, you should see a failure for the CHECK #1 specified above.
- Simulation will create test_counter_check1.log
- Compare test_counter_check1.log with ./solution/test_counter_check1.log and see if your results match with the log in the .solution directory.
- If your results don't match revisit your property and repeat step 3.

CONTINUED ➔

LAB 4 : COUNTER

LAB: How to compile/simulate - step by step instructions

4. % vi counter_property.sv
- Look for `ifdef check2
 - Remove the 'DUMMY' property and code your property as specified above for CHECK #2
 - Save the file and run the following simulation.

% run_check2

- If you have coded the property correct, you should see a failure for the CHECK #2 specified above.
 - Simulation will create test_counter_check2.log
 - Compare test_counter_check2.log with .solution/test_counter_check2.log and see if your results match with the log in the .solution directory.
 - If your results don't match revisit your property and repeat step 4.

5. % vi counter_property.sv
- Look for `ifdef check3
 - Remove the 'DUMMY' property and code your property as specified above for CHECK #3
 - Save the file and run the following simulation.

% run_check3

- If you have coded the property correct, you should see a failure for the CHECK #3 specified above.
 - Simulation will create test_counter_check3.log
 - Compare test_counter_check3.log with .solution/test_counter_check3.log and see if your results match with the log in the .solution directory.
 - If your results don't match revisit your property and repeat step 4.

DONE... CONGRATULATIONS