

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 5 ...

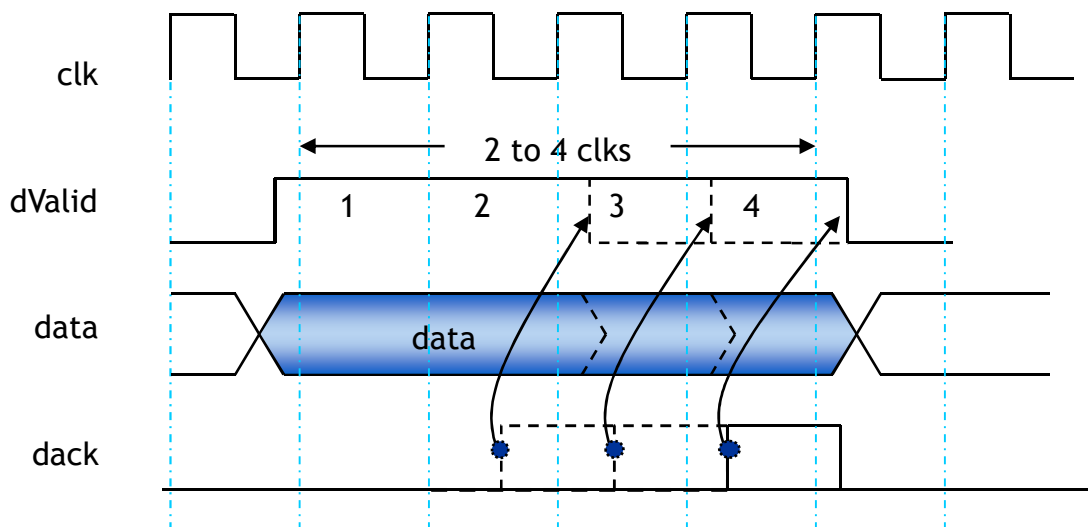
follow the protocol ...

LAB 5 : BUS PROTOCOL

LAB Overview

Specification for a simple data transfer protocol.

- dValid must remain asserted for minimum of 2 clocks but no more than 4 clocks.
 - 'data' must be known when 'dValid' is High.
 - 'dack' going high signifies that target have accepted data and that master must de-assert 'dValid' the clock after 'dack' goes high.
- Note that since data must be valid for minimum 2 cycles, that 'dack' cannot go High for at least 1 clock after the transfer starts (i.e. after the rising edge of 'dValid') and that it must not remain low for more than 3 clocks (because data must transfer in max 4 clocks).*



LAB 5 : BUS PROTOCOL

LAB Objectives

Bus interfaces are common to any design and this lab will show you how to model assertions for common bus protocol specification.

You will learn

- 1. Modeling temporal domain assertions for bus interface type logic.*
- 2. Reinforce understanding of Edge sensitive and sampled value functions, consecutive repetition, boolean expressions, etc.*

LAB: Database

FILES:

- 1. bus_protocol.v :: bus_protocol module that drive a simple bus protocol*
- 2. bus_protocol_property.sv :: SVA file for bus_protocol assertions.
Note that this file is only an empty module shell.
You will add properties that meet the specification described above.*
- 3. test_bus_protocol.sv :: Testbench for the bus_protocol module.
Note the use of 'bind' in this testbench.*

LAB 5 : BUS PROTOCOL

LAB: Assertions to Code

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

CHECK # 1. Check that once dValid goes high that it is consecutively asserted (high) for minimum 2 and maximum 4 clocks

CHECK # 2. Check that data is not unknown and remains stable after dValid goes high and until dAck goes high.

CHECK # 3. Check that 'dAck' and 'dValid' relationship is maintained to complete the data transfer.

In other words,

'dack' going high signifies that target have accepted data and that master must de-assert 'dValid' the clock after 'dack' goes high.

Note that since data must be valid for minimum 2 cycles, that 'dack' cannot go High for at least 1 clock after the transfer starts (i.e. after the rising edge of 'dValid') and that it must not remain low for more than 3 clocks (because data must transfer in max 4 clocks).

LAB 5 : BUS PROTOCOL

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/LAB5`

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

`% run_nobugs`

- This will create the file `test_bus_protocol_nobugs.log`
 - Study this log to familiarize yourself with how the bus_protocol 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 bus_protocol_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_bus_protocol_check1.log`
 - Compare `test_bus_protocol_check1.log` with `./solution/test_bus_protocol_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 5 : BUS PROTOCOL

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

4. % vi bus_protocol_property.sv

- Look for `ifdef check2
- Remove the 'DUMMY' property and code your property as specified 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.
 - Simulation will create test_bus_protocol_check2.log
 - Compare test_bus_protocol_check2.log with .solution/test_bus_protocol_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 bus_protocol_property.sv

- Look for `ifdef check3
 - Remove the 'DUMMY' property and code your property as specified 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.
 - Simulation will create test_bus_protocol_check3.log
- Compare test_bus_protocol_check3.log with .solution/test_bus_protocol_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 5.

CONTINUED ➔

LAB 5 : BUS PROTOCOL

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

6. This step is to simply run the design with ALL the bugs introduced and all the assertions fired.

`% run_checkall`

- If you have coded all your properties correct, you should see all of the failures specifically to learn how more than one design bug could be present at a given time.
 - Simulation will create test_bus_protocol_checkall.log
 - Compare test_bus_protocol_checkall.log with .solution/test_bus_protocol_checkall.log and see if your results match with the log in the .solution directory.
 - If your results don't match, one of your steps 3 or 4 or 5 did not complete correct.

DONE... CONGRATULATIONS