System Verilog Assertions

LAB Material

Ashok B. Mehta

http://www.defineview.com

Copyright Notice

Copyright Notice

© 2006-2013

DefineView Consulting

http://www.defineview.com

Ashok B. Mehta

501 Pine Wood Lane

Los Gatos, CA 95032

(408) 309-1556

Email: ashok@defineview.com

Lab 1 …

how to 'bind' a design module with it's properties …

LAB 1 : Basic Property

LAB Overview

This LAB is to help you understand how to 'bind' a design module to a property module (which contains assertions for the design module.)

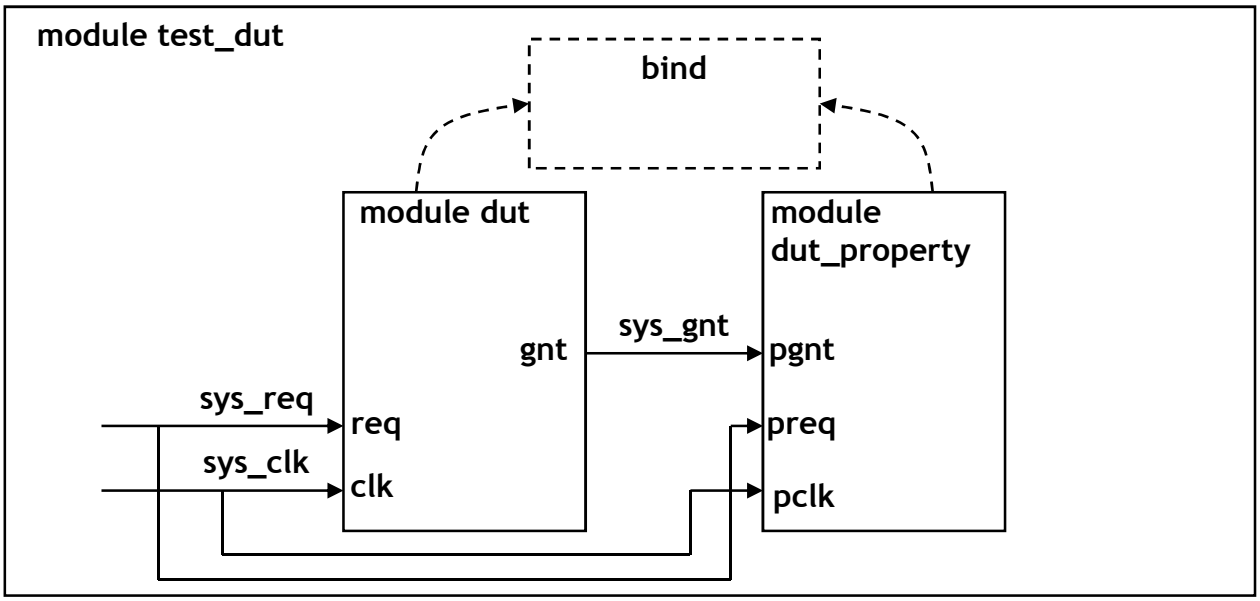
It also highlights how property/sequence behaves with and without an implication and understand the vaccuous pass phenomenon ;-)

LAB Objectives

1. *'bind' a design module to it's property module.*
2. *You will also see how the property can be 'cover'ed and how it can avoid a vacuous pass.*

LAB Design

Here's the block diagram showing connection between 'dut' and 'dut_property' modules. You need to bind these two modules.



LAB 1 : 'bind'ing design/property module

LAB: Database

FILES:

1. *dut.v :: Verilog module that drives a simple req/gnt protocol.*
2. *dut_property.sv :: File that contains 'dut' properties/assertions.*
3. *test_dut.sv :: Testbench for the 'dut'. This is the file in which you'll 'bind' the 'dut' with 'dut_property'*
4. *LAB_QUESTIONS :: This file has questions that you will answer.*

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

1. *% cd <myDir>/SVA_LAB/LAB1*

2. *% vi test_dut.sv*
   Search for "LAB EXERCISE - START"

   Add code after the comments to
   'bind' the design 'dut' with it's property module 'dut_property'

   Save the file.

   Then follow the compile/simulate steps described below.

3. *% run_no_implication*

   This will create test_dut_no_implication.log
   Compare ('diff') your log with the one stored in
   .solution/test_dut_no_implication.log

   If you did a correct bind of 'dut' and 'dut_property', you will see a few PASS/FAIL in the log. If not, you won't see any pass or fail

   -) Study test_dut_no_implication.log
   -) Answer the questions embedded in the file ./LAB_QUESTIONS for +define+no_implication

LAB 1 : 'bind'ing design/property module

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

4. % run_implication

This will create test_dut_implication.log
Compare ('diff') your log with the one stored in
.solution/test_dut_implication.log

If you did a correct bind of 'dut' and 'dut_property', you will see a few PASS/FAIL in the log. If not, you won't see any pass or fail

-) Study test_dut_implication.log
-) Answer the questions embedded in the file ./LAB_QUESTIONS for +define+implication

5. % run_implication_novac

This will create test_dut_implication_novac.log
Compare ('diff') your log with the one stored in
.solution/test_dut_implication_novac.log

If you did a correct bind of 'dut' and 'dut_property', you will see a few PASS/FAIL in the log. If not, you won't see any pass or fail

-) Study test_dut_implication_novac.log
-) Answer the questions embedded in the file ./LAB_QUESTIONS for +define+implication_novac

System Verilog Assertions

LAB Material

Ashok B. Mehta

http://www.defineview.com

Copyright Notice

Copyright Notice

© 2006-2013

DefineView Consulting

http://www.defineview.com

Ashok B. Mehta

501 Pine Wood Lane

Los Gatos, CA 95032

(408) 309-1556

Email: ashok@defineview.com

Lab 2 …

overlapping and

non-overlapping

implication operator …

LAB 2 : Overlap and non-overlap operators

LAB Overview

This example is simply to high light how property behaves with an overlapping implication operator and with a non-overlapping implication operator. It also helps you understand how piplelined threads of a property work

LAB Objectives

1. *You will learn how an overlapping vs. non-overlapping implication operator works.*

2. *You will also learn how multiple pipelined threads work through a property.*

LAB Design Under Test (DUT)

There is no DUT as such in this example. Only a simple property coded with overlapping and non-overlapping operator.

LAB 2 : Overlap and non-overlap operators

LAB: Database

FILES:

1. *test_overlap_nonoverlap.sv :: A simple example showing how to model a property with overlap and non-overlap implication operator.*

2. *LAB_QUESTIONS :: This file has questions that you will answer as pointed out below in compile/simulate section.*

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

1. *% cd <myDir>/SVA_LAB/LAB2*

2. *% run_overlap*
   This will create test_overlap.log
   - Study test_overlap.log
   - Answer the questions embedded in the file ./LAB_QUESTIONS for +define+overlap

3. *% run_nonoverlap*
   This will create test_nonoverlap.log
   - Study test_nonoverlap.log
   - Answer the questions embedded in the file ./LAB_QUESTIONS for +define+nonoverlap

System Verilog Assertions

LAB Material

Ashok B. Mehta

http://www.defineview.com

Copyright Notice

Copyright Notice

© 2006-2013

DefineView Consulting

http://www.defineview.com

Ashok B. Mehta

501 Pine Wood Lane

Los Gatos, CA 95032

(408) 309-1556

Email: ashok@defineview.com

Lab 3 …

good old fifo …

LAB 3 : FIFO

LAB Overview

A simple synchronous FIFO design is presented. FIFOs are some of the most commonly used design elements which require close scrutiny. FIFO assertions deployed directly at the source of a FIFO can greatly reduce the time to debug since these assertions point to the exact instance of fifo where an assertion fires.

LAB Objectives

1. *You will learn how to model various FIFO assertions that will be applicable to most any FIFO.*

2. *You will learn use of boolean expressions and sampled value functions as part of this exercise.*

LAB Design Under Test (DUT)

A simple synchronous FIFO design is presented as the DUT.

- *FIFO is 8 bit wide and 8 deep.*

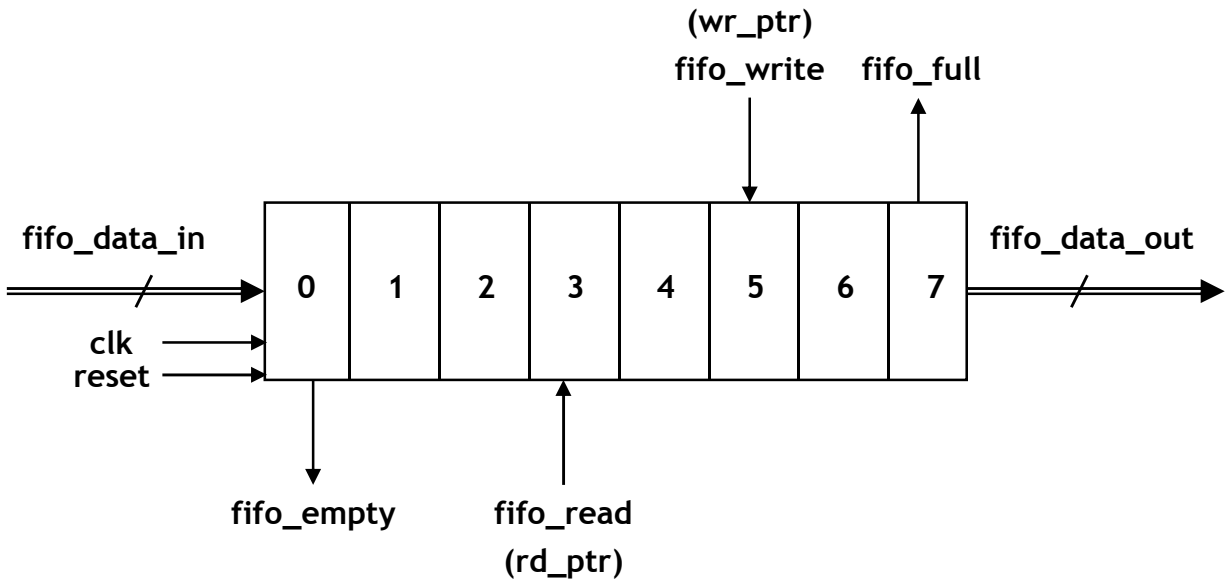
- *FIFO INPUTS*
  fifo_write, fifo_read, clk, rst_ and fifo_data_in[7:0]

- *FIFO OUTPUTS*
  fifo_full, fifo_empty, fifo_data_out[7:0]

LAB 3 : FIFO

FIFO Specs

- *fifo maintains a wr_ptr and a rd_ptr*
    - *wr_ptr increments by 1 everytime a write is posted to the fifo on a fifo_write request*
    - *rd_ptr increments by 1 everytime a read is posted to the fifo on a fifo_read request*
- *fifo maintains a 'cnt' that increments on a write and decrements on a read. It is used to signal fifo_full and fifo_empty conditions as follows*
    - *When fifo 'cnt' is >= 7, fifo_full is asserted*
    - *When fifo 'cnt' is 0, fifo_empty is asserted.*



LAB 3 : FIFO

LAB: Database

FILES:

1. *fifo.v :: Verilog RTL for 'fifo'*

2. *fifo_property.sv :: SVA file for fifo assertion.s*
   This is the file in which you will add your assertions.

3. *test_fifo.sv :: Testbench for the fifo.*
   Note the use of 'bind' in this testbench.

LAB: Assertions to Code

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

CHECK # 1. Check that on reset
rd_ptr=0; wr_ptr=0; cnt=0; fifo_empty=1 and fifo_full=0

CHECK # 2. Check that fifo_empty is asserted when fifo 'cnt' is 0.
Disable this property 'iff (!rst)'

CHECK # 3. Check that fifo_full is asserted any time fifo 'cnt' is greater than 7.
Disable this property 'iff (!rst)'

CHECK # 4. Check that if fifo is full and you attempt to write (but not read) that the wr_ptr does not change.

CHECK # 5. Check that if fifo is empty and you attempt to read (but not write) that the rd_ptr does not change.

CHECK # 6. Write a property to Warn on write to a full fifo

CHECK # 7. Write a property to Warn on read from an empty fifo

LAB 3 : FIFO

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/LAB3

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

   % run_nobugs

   - This will create the file test_fifo_nobugs.log
   - Study this log to familiarize yourself with how the fifo works; when it reaches fifo_full, fifo_empty conditions, etc.

The remaining flow of the exericise 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 fifo_property.sv

   - Look for `ifdef check1
   - Remove the 'DUMMY' property and code your property as specified in the comments
   - 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_fifo_check1.log
   - Compare test_fifo_check1.log with .solution/test_fifo_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 3 : FIFO

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

4. % vi fifo_property.sv

   - Look for `ifdef check2
   - Remove the 'DUMMY' property and code your property as specified in the comments
   - 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_fifo_check2.log
   - Compare test_fifo_check2.log with .solution/test_fifo_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 fifo_property.sv
   - Look for `ifdef check3
   - Remove the 'DUMMY' property and code your property as specified in the comments
   - 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_fifo_check3.log
   - Compare test_fifo_check3.log with .solution/test_fifo_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 3 : FIFO

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

6. % vi fifo_property.sv

- Look for `ifdef check4
- Remove the 'DUMMY' property and code your property as specified in the comments
- save the file and run the following simulation.

% run_check4

- If you have coded the property correct, you should see a failure for the CHECK #4 specified above.
- Simulation will create test_fifo_check4.log
- Compare test_fifo_check4.log with .solution/test_fifo_check4.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 6.

7. % vi fifo_property.sv

- Look for `ifdef check5
- Remove the 'DUMMY' property and code your property as specified in the comments
- save the file and run the following simulation.

% run_check5

- If you have coded the property correct, you should see a failure for the CHECK #5 specified above.
- Simulation will create test_fifo_check5.log
- Compare test_fifo_check5.log with .solution/test_fifo_check5.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 7

CONTINUED ➔

LAB 3 : FIFO

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

8. % vi fifo_property.sv

- Look for `ifdef check6
- Remove the 'DUMMY' property and code your property as specified in the comments
- save the file and run the following simulation.

% run_check6

- If you have coded the property correct, you should see a failure for the CHECK #6 specified above.
- Simulation will create test_fifo_check6.log
- Compare test_fifo_check6.log with .solution/test_fifo_check6.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 8.

9. % vi fifo_property.sv

- Look for `ifdef check7
- Remove the 'DUMMY' property and code your property as specified in the comments
- write the file and run the following simulation.

% run_check7

- If you have coded the property correct, you should see a failure for the CHECK #7 specified above.
- Simulation will create test_fifo_check7.log
- Compare test_fifo_check7.log with .solution/test_fifo_check7.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 9

DONE … CONGRATULATIONS

System Verilog Assertions

LAB Material

Ashok B. Mehta

http://www.defineview.com

Copyright Notice

Copyright Notice

© 2006-2013

DefineView Consulting

http://www.defineview.com

Ashok B. Mehta

501 Pine Wood Lane

Los Gatos, CA 95032

(408) 309-1556

Email: ashok@defineview.com

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 exericise 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

System Verilog Assertions

LAB Material

Ashok B. Mehta

http://www.defineview.com

Copyright Notice

Copyright Notice

© 2006-2013

DefineView Consulting

http://www.defineview.com

Ashok B. Mehta

501 Pine Wood Lane

Los Gatos, CA 95032

(408) 309-1556

Email: ashok@defineview.com

Lab 5 …

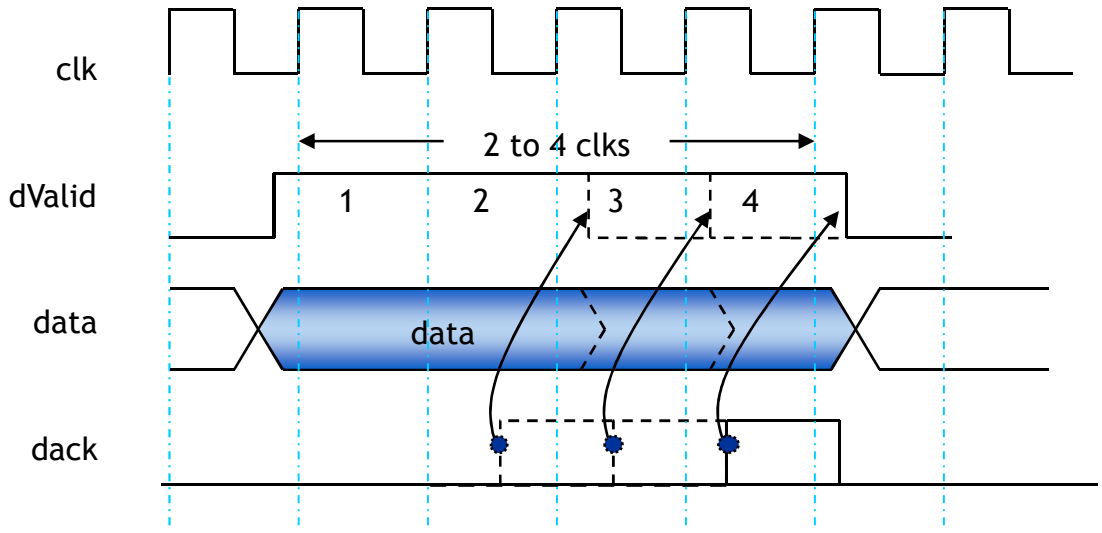
follow the protocol …

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 trasnfer 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 ablve.

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 unknow 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 trasnfer 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 exericise 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

System Verilog Assertions

LAB Material

Ashok B. Mehta

http://www.defineview.com

Copyright Notice

Copyright Notice

© 2006-2013

DefineView Consulting

http://www.defineview.com

Ashok B. Mehta

501 Pine Wood Lane

Los Gatos, CA 95032

(408) 309-1556

Email: ashok@defineview.com

Lab 6 …

PCI Read protocol …

LAB 6: PCI Read Protocol

LAB Overview

A simple system with a PCI Master and PCI Target modules designed to do a simple basic PCI Read operation.

The LAB shows how to derive and write simple but effective assertions for a PCI type bus.

LAB Objectives

1) Learn how to model temporal domain assertions for bus interface type logic.

2) Reinforce understanding of Edge sensitive sampled value functions, consecutive repetition, boolean expressions, etc.

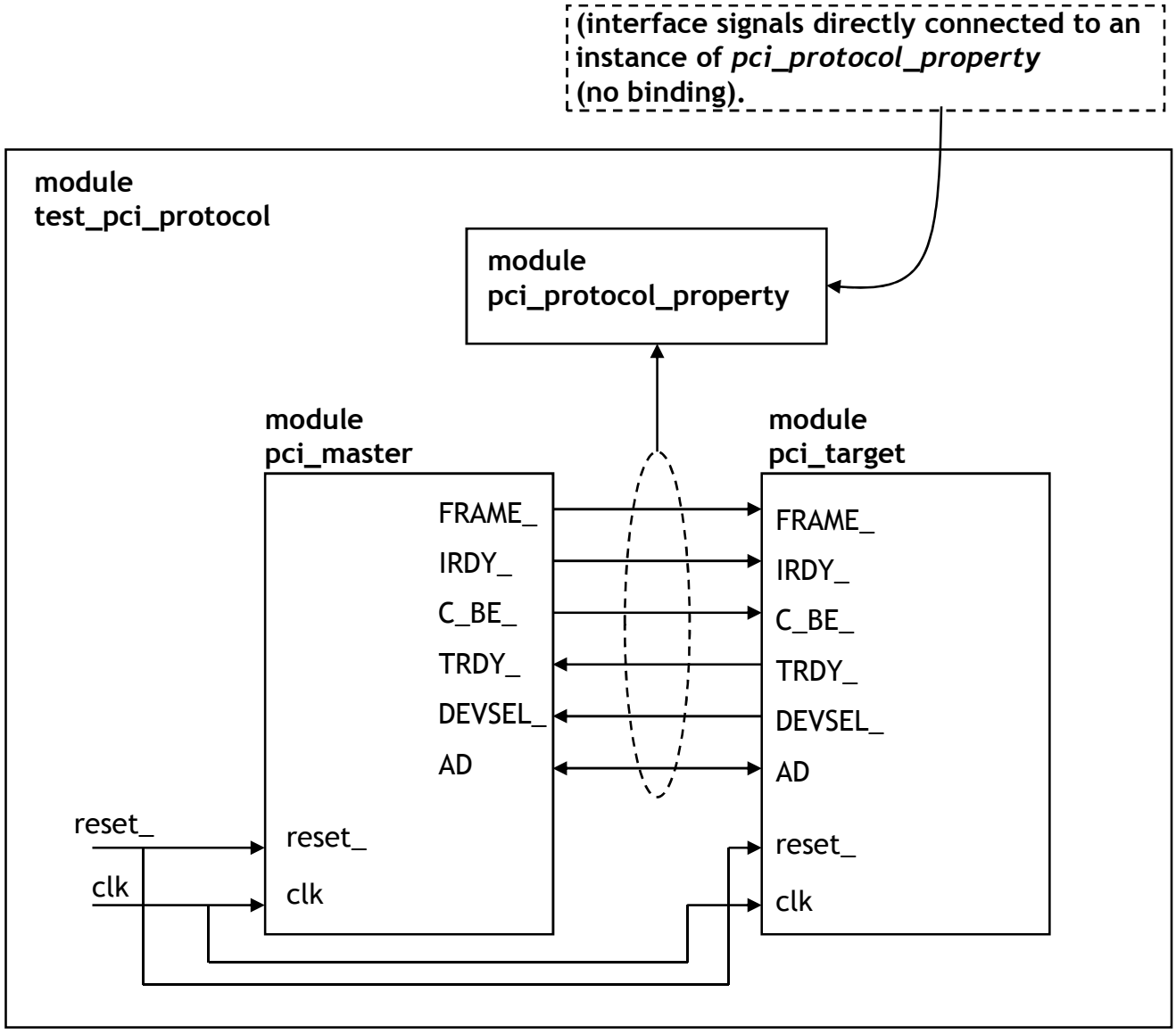
LAB: Database

FILES:

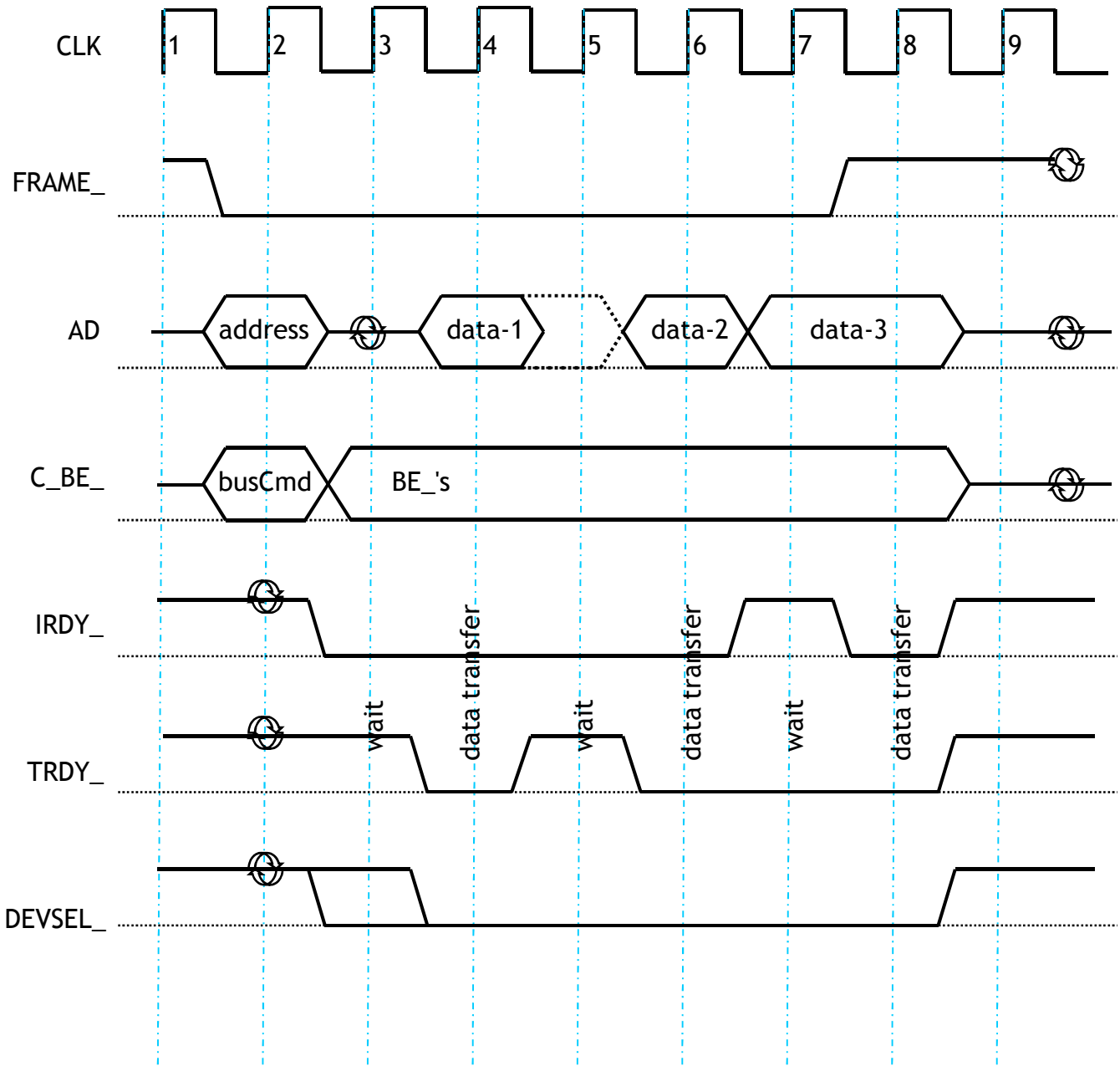
pci_master.v :: A (very) simple PCI Master module driving only a simple Read cycle.
pci_target.v :: A (very) simple PCI Target module responding to a simple Read Cycle.
pci_protocol_property.v :: SVA file for PCI Read cycle assertions.
Note that this file is only an empty module shell.
You will add properties that meet the specification described below.

test_pci_protocol.sv :: Testbench for the pci_protocol module.

LAB 6: PCI System



LAB6: PCI Read Protocol



PCI: Basic Read Protocol Checkers

LAB: Assertions to Code

Property Name	Description
checkPCI_AD_CBE (check1)	On falling edge of FRAME_, AD or C_BE_ bus cannot be unknown
checkPCI_DataPhase (check2)	When both IRDY_ and TRDY_ are asserted, AD or C_BE_ bus cannot be unknown
checkPCI_Frame_Irdy (check3)	FRAME can be de-asserted only if IRDY_ is asserted
checkPCI_trdyDevsel (check4)	TRDY_ can be asserted only if DEVSEL_ is asserted
checkPCI_CBE_during_trx (check5)	Once the cycle starts (i.e. at FRAME_ assertion) C_BE_ cannot float until FRAME_ is de-asserted.

LAB 6 : PCI Read 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/LAB6
% vi pci_protocol_property.sv

Edit this file to add your properties.

Note that DUMMY properties are coded in pci_protocol_property.sv to simply allow the module to compile.

You must remove the DUMMY properties and code correct properties as required above.

2. % vi pci_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_pci_protocol_check1.log
- Compare test_pci_protocol_check1.log with .solution/test_pci_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 this step.

CONTINUED ➔

LAB 6 : PCI Read Protocol

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

3. % vi pci_protocol_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_pci_protocol_check2.log
   - Compare test_pci_protocol_check2.log with .solution/test_pci_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 this step.

4. % vi pci_protocol_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_pci_protocol_check3.log
   - Compare test_pci_protocol_check3.log with .solution/test_pci_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 this step.

CONTINUED ➔

LAB 6 : PCI Read Protocol

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

5. % vi pci_protocol_property.sv
   - Look for `ifdef check4
   - Remove the 'DUMMY' property and code your property as specified above for CHECK #4
   - Save the file and run the following simulation.

   % run_check4
   - If you have coded the property correct, you should see a failure for the CHECK #4 specified above.
   - Simulation will create test_pci_protocol_check4.log
   - Compare test_pci_protocol_check4.log with .solution/test_pci_protocol_check4.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 this step.

6. % vi pci_protocol_property.sv
   - Look for `ifdef check5
   - Remove the 'DUMMY' property and code your property as specified above for CHECK #5
   - Save the file and run the following simulation.

   % run_check5
   - If you have coded the property correct, you should see a failure for the CHECK #5 specified above.
   - Simulation will create test_pci_protocol_check5.log
   - Compare test_pci_protocol_check5.log with .solution/test_pci_protocol_check5.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 this step.

DONE... CONGRATULATIONS