EE 8350 Advanced Verification Methodologies for VLSI Systems
LAB 9

OBJECTIVE: To learn about SystemVerilog Assertions and their implementations.

SystemVerilog Assertion
Often known as SVA, assertions are primarily used to help validate the behavior of a design.

In SystemVerilog there are two kinds of assertions: immediate (assert) and concurrent (assert property).

Immediate Assertions
Immediate assertions are procedural statements and are mainly used in simulation. An assertion is basically a statement that something must be true, similar to the if statement. The difference is that an if statement does not assert that an expression is true, it simply checks that it is true.

Concurrent Assertions
There are properties of a design that must evaluate true globally, and not just during a particular instance in which a line of procedural code executes. Hence, SVA provides support for concurrent assertions, which may be checked outside the execution of a procedural block. That is, the checks are performed concurrently with all other procedural blocks in the verification environment. Concurrent assertions are easily identified by observing the presence of the SVA ***property*** keyword combined with the ***assert*** directive, as we will see in this lab.

Implication operators
An implication refers to a situation where, in order for a behavior to occur, a preceding sequence must have occurred. This preceding sequence in this case is known as the **antecedent**. The succeeding behavior is known as the **consequent**. One way to express this symbolically is as follows:

antecedent |-> *consequent;*

If there is no match of the antecedent sequence expression, implication succeeds vacuously by returning true. If there is a match, for each successful match of the antecedent sequence expression, the consequent sequence expression is separately evaluated, beginning at the end point of the match.

There are two forms of implication: ***overlapped implication*** using the operator |->, and ***non-overlapped implication*** using operator the |=>.

For ***overlapped implication (|->)***, if there is a match in the antecedent sequence expression, then the first element of the consequent sequence expression is evaluated on the ***same clock cycle***.

For ***non-overlapped implication (|=>)***, the first element of the *consequent sequence expression* is evaluated on the ***next clock cycle***.

Value change system functions:

Sl. No.	**Function**	**Description**
1	$rose(expr)	This returns TRUE if the LSB of the expression changed to 1. Otherwise, it returns false.
2	$fell(expr)	This returns TRUE if the LSB of the expression changed to 0. Otherwise, it returns false.
3	$stable(expr)	This returns TRUE if the value of the expression did not change.
4	$changed(expr)	This returns TRUE if the value of the expression changed.
5	$sampled(expr)	This returns the current sampled value of the specified expression. As assertions use sampled values, the use of $sampled in assertions apart from the disable-statement and the action block is redundant.
6	$past(expr, no_clk_cycles)	This returns the sampled value of the specified expression that was present that number of cycles prior to the time of evaluation of $past. The number of cycles must be one or greater, and it defaults to the value 1.

In this lab, we will be writing the SystemVerilog Assertion for the switch module used in the previous lab.

Step 1: In **switch_assertions.sv**, define the assertion property for checking the clock frequency

- Define the assertion property with name clk_freq

  ```
  property clk_freq;
  ```

- Define a local variable to store the current time. Make this assertion trigger at positive edge of clock

  ```
      time cur_time;
      @(posedge clk)
  ```

- Compute the difference in the time between 2 consecutive positive edges of the clock using the non-overlapping operator. End the property.

```
            (1, cur_time=$time) |=> (`PERIOD==$time-cur_time);
endproperty
```

- Assert the assertion property defined above and implement the pass/fail logs for a pass or failure of the assertion.

```
            ap_clk_freq: assert property(clk_freq)
                  $display("Pass: Clock frequency assertion passes at %t",
$time);
            else
                  $display("Fail: Clock frequency assertion fails at %t",
$time);
```

Step 2: In **switch_assertions.sv**, define PERIOD.

- This is the expected time period which is used to compare with the clock time period of the actual clock.

`define PERIOD 10

It is a good practice to keep the assertion module in a separate SystemVerilog file. This module needs to be instantiated in the top module with the required port list.

Step 3: In **top.sv**, we need to instantiate the assertion module:

- Instantiate the switch_assertion module. Make use of the switch_input_if and switch_output_if interfaces for the connections.

```
switch_assertions sw_assert(.clk(clk),
      .ready_in(switch_input_if.ready_in),
      .data_in(switch_input_if.data_in),
      .port_num(switch_input_if.port_num),
      .ready_out_0(switch_output_if[0].ready_out),
      .ready_out_1(switch_output_if[1].ready_out),
      .ready_out_2(switch_output_if[2].ready_out),
      .ready_out_3(switch_output_if[3].ready_out),
      .data_out_0(switch_output_if[0].data_out),
      .data_out_1(switch_output_if[1].data_out),
      .data_out_2(switch_output_if[2].data_out),
      .data_out_3(switch_output_if[3].data_out)
);
```

Step 4: Run the following commands on the command line to execute the simulation. Take a snapshot of the results displayed on the terminal. (The first ten assertion pass/fail logs should be sufficient.)

```
vcs -cm line+cond+fsm -Mupdate +v2k -sverilog -timescale=1ns/10ps +incdir+$UVM_HOME/src $UVM_HOME/src/uvm.sv $UVM_HOME/src/dpi/uvm_dpi.cc -CFLAGS -DVCS top.sv switch_input_interface.sv switch_output_interface.sv switch.v switch_assertions.sv -l compile.log -debug_all
```

```
./simv -cm line+cond+fsm +UVM_TESTNAME=switch_test +ntb_random_seed=<your student ID> +UVM_VERBOSITY=UVM_NONE
```

(Note: Enter your student ID for the random seed value.)

Step 5: In **switch_assertions.sv**, modify the clock period so that the assertion fails.

- Modify the Period to 15, defined in Step 2:

```
`define PERIOD 15
```

Step 6: Run the simulation command and take a snapshot of the results. (The first ten assertion pass/fail logs should be sufficient.)

Step 7: In **switch_assertions.sv**, do the following:

- Comment or disable the previous clock frequency assertion by **commenting the code below:**

```
`define ENABLE_CLK_FREQ_CHECK
```

Step 8: In **switch_assertions.sv**, define an assertion to check if ***data_out_0*** is disabled if ready_out_0 is LOW as defined in the functionality of the design under test.

- Define the assertion property

```
property data_out_0_disable;
        disable iff (!ready_in | ready_out_0)
        @(posedge clk)
        data_out_0 == 0;
endproperty
```

- Assert the assertion property for **data_out_0_disable**

```
ap_data_out_0_dis: assert property(data_out_0_disable)
        $display("Pass: data_out_0 disabled when ready_out_0 is LOW at %t", $time);
else
        $display("Fail: data_out_0 is not disabled when ready_out_0 is LOW at %t", $time);
```

Step 9: Run the simulation command and take a snapshot of the results.

Step 10: In **switch_assertions.sv**, define an assertion to check if the proper ready_out signal is asserted when port_num changes.

- We first define the decoding logic which gives us the expected ready_out signal when port_num changes:

```
always_comb
begin: Expected_ready_out_block
        case(port_num)
                2'b00 : ready_out_exp = 4'b0001;
                2'b01 : ready_out_exp = 4'b0010;
                2'b10 : ready_out_exp = 4'b0100;
                2'b11 : ready_out_exp = 4'b1000;
                default : ready_out_exp = 4'b0000;
        endcase
end
```

- Define the assertion property:

```
property ready_out_0_check;
        disable iff (!ready_in)
         @(posedge clk)
                $rose(ready_out_exp[0]) |-> ##[0:1] $rose(ready_out_0);
endproperty
```

- Assert the assertion property for **ready_out_0_check**:

```
ap_ready_out_0: assert property(ready_out_0_check)
        $display("Pass: ready_out_0 rises when port_num changes to 2'd1
at %t", $time);
else
        $display("Fail: ready_out_0 doesn't rise when port_num changes to
2'd1 at %t", $time);
```

- Similarly, define assertion properties for ready_out_1, ready_out_2 and ready_out_3. Name them as **ready_out_1_check, ready_out_2_check and ready_out_3_check**, respectively. Modify the **ready_out_exp[<x>]** and **ready_out_<x>** to proper values in the assertion body.

- Assert the properties for the above 3 assertions. Name the properties as **ap_ready_out_1, ap_ready_out_2, ap_ready_out_3** respectively. Modify the

ready_out_<x>_check to the proper value. Also modify the two display instructions for **ready_out_<x>_check** to proper values.

Step 11: Run the simulation command and take a snapshot of the results.

Questions:

1. Explain the difference between an Immediate and a concurrent assertion.

2. What is the difference between the two implication operators used in concurrent assertions?

3. In Step 9, you may observe a FAIL report for the **data_out_0_disable assertion**. Is this an actual failure or not? What do you think the reason could be for this failure?
   (Hint: Look at the waveforms to debug this failure.)

4. Write an assertion to check if all ready_out signals (ready_out_0 through ready_out3) are LOW within 1 clk_cycle after ready_in goes LOW. Use the following base code for implementing the assertion. (Only the concurrent assertion expression is required for the report)

```
//ready_out_disable_check
//Define property for ready_out_check
property ready_out_disable_check;
  @(posedge clk)
        /*
        Add your code here…
        …………………………………….
        */
endproperty

//Assert property ready_out_0
ap_ready_out_disable: assert property(ready_out_disable_check)
  $display("Pass: ready_out_disable at %t", $time);
else
  $display("Fail: ready_out_disable at %t", $time);
```

What is to be turned in?

A report (pdf file) consisting of the screenshot of the simulation results described in Steps 4, 6, 9 and 11, as well as the answers to the above four questions.

References used in the preparation of this lab:

1. http://www.verificationguide.com
2. https://www.edaplayground.com
3. http://testbench.in
4. http://www.asic-world.com
5. https://verificationacademy.com
6. http://systemverilog.us/driving_into_wires.pdf
7. http://www.eetimes.com/document.asp?doc_id=1276112
8. http://www.embedded.com/print/4004083
9. https://colorlesscube.com/uvm-guide-for-beginners
10. https://www.doulos.com
11. http://www.chipverify.com/uvm
12. https://vlsi.pro/system-verilog-assertions-sva-system-tasks-and-functions/
13. https://www.verificationguide.com/p/systemverilog-implication-operator.html
14. https://www.project-veripage.com/sva_12.php
15. https://link.springer.com/content/pdf/bbm%3A978-0-387-68398-0%2F1.pdf