ASSERTIONS SHORT NOTES

Bala Murali Krishna

www.linkedin.com/in/bala-murali-krishna-vlsi-dv

Assertions

Example of how to write an assertion.

```
property property_name;
        @(posedge clk) $rose(a) |-> b;
Endproperty

assertion_name : assert property(property_name)
                    else $error ("error statement");
```

1) ## →
   Syntax : a ##2 b
   Explanation : If "a" is high in a cycle after two clock cycles, signal "b" has to be asserted high.

2) |-> (overlapped implication same cycle)
   Syntax : a |-> b
   Explanation : If the left hand side condition ("a"==1) of the implication "|->" (called "antecedent") is true, check the right hand side (called "consequent") condition ("b" == 1) in the same cycle.

3) |=> (non-overlapped implication next cycle)
   Syntax : a |=> b
   Explanation : If the left hand side condition ("a"==1) of the implication "|->" (called "antecedent") is true, check the right hand side (called "consequent") condition ("b" == 1) in the next cycle.

4) |-> ##[min_delay_range : max_delay_range]
   Syntax : a |->[5:7] b
   Explanation : when signal "a" is asserted high, within given latency ranges it checks assertion of signal "b".

5) $rose
   Syntax : a |-> $rose(b)
   Explanation : $rose checks when signal "a" asserts high, in the same cycle it also detects the positive edge on signal "b".

6) $fell
   Syntax : a |-> $fell(b)
   Explanation : $fell checks when signal "a" asserts high, in the same cycle, it also detects negative edge on signal "b".

7) $stable

Syntax : a |-> $stable(b)

Explanation : $stable checks when signal “a” asserts high, it checks for no change in signal “b”. It means, signal “b” should stay as it was in the previous cycle.

8) $past

Syntax : a |-> $past(b, 2)

Explanation : $past checks when signal “a” asserts high, it checks signal “b” was high before 2 clock cycles.

If the second argument is not specified, then by default, $past checks for the signal value in the immediate previous cycle.

9) Disable iff

Syntax :

```
property disable_iff_p;
  disable iff(reset)
  @(posedge clk) a |-> b;
endproperty

disable_iff_chk: assert property (disable_iff_p);
```

Explanation : Property *disable_iff_p*, remains disabled if signal “reset” is asserted high. If reset is not asserted high, then it checks if signal “a” is asserted high, then in the same cycle, signal “b” should also be asserted high.

10) Signal_name[*n] (consecutive repetition operator)

Syntax : $rose(a) |-> b[*6]

Explanation : Above syntax checks when the positive edge is detected on signal “a”, check from next clock onwards, signal “b” is asserted high continuously for 6 clock cycles.

11) [-> n] (go to repetition operator no consecutive)

Syntax : signal_name1 [->n] ##1 signal_name2

Explanation : It specifies that an expression matches the number of times specified, not necessarily in the consecutive clock cycles. The matches can be intermittent.

The key requirement of a “go to” repeat is that the last match of the expression (which is checked for repetition) should happen in the clock cycle before the end of the entire sequence match.

Example : $rose(a) |->b [->3] ##1 c

To make assertion pass for this after a has been go high b should high for 3 cycles nonconsecutively and will be followed by c.

!a a !b !b b !b !b b !b !b !b b c

12) [=n] (Nonconsecutive repetition operator):

Syntax : signal_name1 [=n] ##1 signal_name2

Explanation :

```
property non_consecutive_repetition_p;
   @(posedge clk) $rose(a) |-> b[=3] ##1 c;
endproperty

non_consecutive_repetition_chk : assert property (non_consecutive_repetition_p);
```

Property non_consecutive_repeatation_p check, when the positive edge of signal “a” is detected, check for signal “b” to be high continuously or intermittently for 3 clock cycles, followed by signal “c “ to be high in any cycle after that while signal “b” remains low.

!a a !b !b b !b !b b !b !b !b b !b !b !b c

13) If else inside property

Syntax :

```
property if_else_p;
@(posedge clk) $rose(d) |=>
  if(b)
  (c[->2] ##1 d)
  else
  (a[->2] ##1 c);
endproperty

if_else_chk: assert property (if_else_p);
```

14) $onehot

Syntax : $onehot(signal_name)

Explanation : $onehot checks the number of ones must be one in “signal_name” for each cycle.

$onehot(~signal_name) checks for number of zeros must be one.

15) Throughout

Syntax : (signal_a throughout signal_b)

Explanation : signal a should be continuously till signal_b goes low.

If signal_b is low when signal_a goes high, then also this assertion passes as it check only when signal_a goes low signal is low or not(only checks at starting an ending of signal b)

Assertion expects signal_a to be high in the last cycle where signal_b goes low.

16) Until & until_with

Syntax : (signal_a until signal_b)

Explanation : $rose(a) |=> (b until c)

$rose(a) |=> (b until_with c)

Property until_with checks for:

When the positive edge of signal “a” is detected, assertion checks for signal “b” to be high continuously until signal “c” goes low.

Property until checks for :

When the positive edge of signal “a” is detected, assertion checks for signal “b” to be high continuously until one cycle before signal “c” goes low.

17) Within

Syntax : sequence1 within sequence 2;

Explanation : This means that seq1 happens within the start and completion of seq2. The starting matching point of seq2 must happen before the starting matching point of seq1. The ending matching point of seq1 must happen before the ending matching point of seq2.

18) $isunknown

Syntax : $isunknow(signal_a)

Explanation : checks for signal_a should be unknown at least for 1 clock.

19) Indefinite timing window [1:$]

Syntax : a[*1:$]

Explanation : a should high minimum of 1 clock cycle and maximum is unlimited

20) $assertkill : turnoff assertion during simulation time