DAY-80
#100DAYSRTL

"System Verilog:-Assertions(Part-2)"

"Assertions Methods":-

System Verilog has some inbuild methods for the purpose of assertions

- **$rose:-**It returns true if the least significant bit of the signal changed to 1. Otherwise, it returns false
  - **Eg:-** Sequence seq_rose checks that the signal "a" transitions to a value of 1 on every positive edge of the clock. If the transition does not occur, the assertion will fail

```
sequence seq_rose;
  @(posedge clk) $rose(a);
endsequence
```

- **$fell:-** It returns true if the least significant bit of the signal changed to 0. Otherwise, it returns false
  - **Eg:-**Sequence seq_fell checks that the signal "a" transitions to a value of 0 on every positive edge of the clock. If the transition does not occur, the assertion will fail.

```
sequence seq_fell;
  @(posedge clk) $fell(a);
endsequence
```

- **$stable:-** It returns true if the value of the expression don't change. Otherwise, it returns false
  - **Eg:-** Sequence seq_stable checks that the signal "a" is stable on every positive edge of the clock. If there is any transition occurs, the assertion will fail

```
sequence seq_fell;
  @(posedge clk) $fell(a);
endsequence
```

- **$past:-**It provides the value of the signal from the previous clock cycle
  - **Eg:-** Property checks that, in the given positive clock edge, if the “b” is high, then 2 cycles before that, a was high.

```
property p;
  @(posedge clk) b |-> ($past(a,2) == 1);
endproperty
a: assert property(p);
```

“Code Practising”:-

```
module asertion_ex;
  bit clk,a,b;

  always #5 clk = ~clk; //clock generation

  //generating 'a'
  initial begin
        a=1; b=1;
    #15 a=0; b=0;
    #10 a=1; b=0;
    #10 a=0; b=0;
    #10 a=1; b=1;
    #10;
    $finish;
  end

  property p;
    @(posedge clk) b |-> ($past(a,2) == 1);
  endproperty

  //calling assert property
  a_1: assert property(p);

  //wave dump
  initial begin
    $dumpfile("dump.vcd"); $dumpvars;
  end
endmodule
```

- **$past construct with clock gating:-** The $past construct can be used with a gating signal on a given clock edge, the gating signal has to be true even before checking for the consequent condition.
  - **Eg:-** Property checks that, in the given positive clock edge, if the “b” is high, then 2 cycles before that, a was high only if the gating signal "c' is valid on any given positive edge of the clock.

```
Property p;
  @(posedge clk) b |-> ($past(a,2,c) == 1);
endproperty
a: assert property(p);
```

- **$onehot:-** checks that only one bit of the expression can be high on any given clock edge.

```
assert property( @(posedge clk) $onehot(state) );
```

- **$onehot:-**It checks only one bit of the expression can be high or none of the bits can be high on any given clock edge.

```
assert property( @(posedge clk) $onehot0(state) );
```

- **$isunknown:-** It checks if any bit of the expression is X or Z.

```
assert property( @(posedge clk) $isunknown(bus) ) ;
```

- **$countones:-** It counts the number of bits that are high in a signal

```
assert property( @(posedge clk) $countones(bus)> 1 );
```

- **Disable iff:-** In certain design conditions, we don't want to proceed with the check if some condition is true. this can be achieved by using disable iff.
  - **Eg:-** property checks that, if the signal “a” is high on a given posedge of the clock, the signal “b” should be high for 3 clock cycles followed by “c” should be high after ”b” is high for the third time. During this entire sequence, if reset is detected high at any point, the checker will stop

```
property p;
  @(posedge clk)
  disable iff (reset) a |-> ##1 b[->3] ##1 c;
endproperty
a: assert property(p);
```

- **Ended:-** while concatenating the sequences, the ending point of the sequence can be used as a synchronization point. This is expressed by attaching the keyword "ended" to a sequence name.
  - **Eg:-**property checks that, sequence seq_1 and SEQ_2 match with a delay of 2 clock cycles in between them. the end point of the sequences does the synchronization

```
sequence seq_1;
 (a && b) ##1 c;
endsequence

sequence seq_2;
  d ##[4:6] e;
endsequence

property p;
  @(posedge clk) seq_1.ended |-> ##2 seq_2.ended;
endpeoperty
a: assert property(p);
```