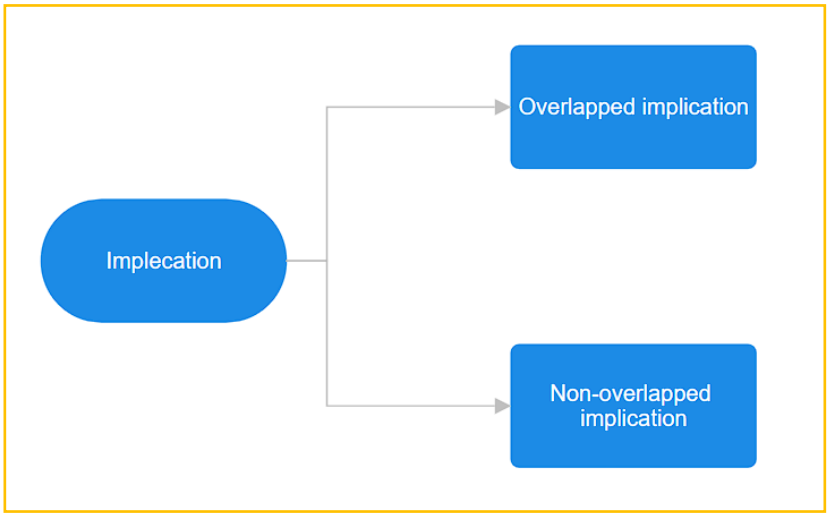
DAY-79
#100DAYSRTL

"System Verilog:-Assertions(Part-1)"

"Implication Operator":-

```
sequence seq;
  @(posedge clk) a ##2 b;
endsequence
```

- In the above sequence, we can observe that the sequence starts on every positive edge of the clock and it looks for "a" to be high on every positive clock edge. If the signal "a" is not high on any given positive clock edge, an error is issued by the checker.
- If we want the sequence to be checked only after "a" is high, this can be achieved by using the implication operator.
- The implication is equivalent to an if-then structure. The left-hand side of the implication is called the **"antecedent"** and the right-hand side is called the **"consequent"**
- The antecedent is the gating condition. If the antecedent succeeds, then the consequent is evaluated.
- The implication construct can be used only with property definitions. It cannot be used in sequences.



“Overlapped Implication”:-

- The overlapped implication is denoted by the symbol |->.
- If there is a match on the antecedent, then the consequent expression is evaluated in the same clock cycle.
- Below property checks that, if signal “a” is high on a given positive clock edge, then signal “b” should also be high on the same clock edge.

```
property p;
  @(posedge clk) a |-> b;
endproperty
a: assert property(p);
```

“Code Practising”:-

```
module asertion_ex;
  bit clk,a,b;
  always #5 clk = ~clk;
  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) a |-> b;
  endproperty
  a_1: assert property(p);
  initial begin
    $dumpfile("dump.vcd"); $dumpvars;
  end
endmodule
```

“Result”:-

```
Error: ASRT_0005 testbench.sv(16): Assertion "a_1" FAILED at time: 35ns, scope: asertion_ex, start-time: 35ns
```

“Non-Overlapped Implication”:-

- The non-overlapped implication is denoted by the symbol |=>.
- If there is a match on the antecedent, then the consequent expression is evaluated in the next clock cycle.
- Below property checks that, if signal “a” is high on a given positive clock edge, then signal “b” should be high on the next clock edge.

```
property p;
  @(posedge clk) a |=> b;
endproperty
a: assert property(p);
```

“Code Practising”:-

```
module asertion_ex;
  bit clk,a,b;
  always #5 clk = ~clk;
  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) a |-> b;
  endproperty
  a_1: assert property(p);
  initial begin
    $dumpfile("dump.vcd"); $dumpvars;
  end
endmodule
```

“Result”:-

```
Error: ASRT_0005 testbench.sv(16): Assertion "a_1" FAILED at time: 25ns, scope: asertion_ex, start-time: 15ns
Error: ASRT_0005 testbench.sv(16): Assertion "a_1" FAILED at time: 45ns, scope: asertion_ex, start-time: 35ns
```

“Implication with a sequence”:-

The below property checks that, if the sequence seq_1 is true on a given positive edge of the clock, then start checking the seq_2 (“d” should be low, 2 clock cycles after seq_1 is true).

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

sequence seq_2;
  ##2 !d;
endsequence

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

“Timing Windows in Assertions”:-

Below property checks that, if signal “a” is high on a given positive clock edge, then within 1 to 4 clock cycles, the signal “b” should be high.

```
property p;
  @(posedge clk) a |-> ##[1:4] b;
endproperty
a: assert property(p);
```

“Repetation Operators”:-

```
property p;
  @(posedge clk) a |-> ##1 b ##1 b ##1 b;
endproperty
a: assert property(p);
```

Using repetation operator we can write it as

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

Examples:-

a ##1 b [*3] ##1 c	// Equiv. to **a ##1 b ##1 b ##1 b ##1 c**
(a ##2 b) [*2]	// Equiv. to **(a ##2 b ##1 a ##2 b)**

(a ##2 b)[*1:3]	// Equiv. to **(a ##2 b)**
	// or **(a ##2 b ##1 a ##2 b)**
	// or **(a ##2 b ##1 a ##2 b ##1 a ##2 b)**

“go to repetition”:-

the go-to repetition operator is used to specify that a signal will match the number of times specified not necessarily on a continuous clock cycle

Eg:-

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

- 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