➔ Assertions

1. Assert that req is followed by ack within 3 cycles.

```
module rakesh();
bit req,ack;
bit clk;
always #5 clk = ~clk;
property cycle();
@(posedge clk) req |-> ##[1:3] ack;
endproperty
prop: assert property (cycle);
initial begin
 clk=0;
 #5;
 clk=1;
 req=0;ack=0;
 #10;
 req=1;ack=0;
 #10;
 req=0;ack=1;
 #10;
 req=0;ack=1;
 #10;
 req=0; ack=1;
 end
 endmodule
```

2. Ensure reset is deasserted within 5 cycles of power-up

```
 property  reset_pro();
 @(posedge clk ) power_up |-> ##[1:5] !reset;
 endproperty
 ass:assert property (reset_pro)
  $display("assertion pass");
  else
  $error ("Assertion failed: Reset not deasserted within 5 cycles of power-up");
```

3) Assert that wvalid stays high for at most 2 cycles without wready

```
property  valid_ready();
  @(posedge clk ) wvalid && !wready |-> ##[1:2] !wvalid;
  endproperty
  ass:assert property (valid_ready)
   $display("assertion pass");
   else
   $error ("Assertion failed:wvalid  not stays high for at most 2 cycles without wready");
```

4) Ensure FIFO never underflows (read without empty)

```
property underflow;
@(posedge clk); (rd_en && empty ) |->  ##0 0;
endproperty

ass: assert property(underflow);
 $display("assertion PASS");
 else
 $error("assertion fail : read with empty");
```

5) Assert that no transaction occurs during reset

```
property  no_tranction();
@(posedge clk)
  !reset |-> (!(valid && ready) ##2 addr )
 // !reset |-> !(wr_en || rd_en)
  endproperty

  assse: assert property (no_tranction)
  $display("assertion pass");
  else
  $error("assertiion fail: that transaction occurs during reset");
```

6) Assert burst transfers complete with burst_len cycles.

Let's assume these signals:
burst_start ➔ signal that goes high to indicate start of a burst.
transfer_valid ➔ goes high each cycle when a valid transfer is happening.
burst_len ➔ number of cycles the burst should last (can be a parameter or signal).
clk and active-low reset_n are the clock and reset

```
property burst_completes_correctly;
  @(posedge clk)
    disable iff (!reset_n)
    burst_start |-> ##1
      (transfer_valid[*burst_len]) ##1
      (!transfer_valid);
endproperty

assert property (p_burst_completes_correctly)
  else $error("Burst transfer did not complete in burst_len cycles!");
```

7) Check that write happens only when enable is high

```
property enable_write();
@(posedge clk)
wr_en |-> enable;
```

```
endproperty

assert property (enable_write)
else $error (" write not  happens only when enable is high");
```

8. Assert a transaction ID is not repeated consecutively

```
// Assertion: ID must not repeat consecutively

property id_not_repeated;
  @(posedge clk)
    disable iff (!reset_n)
    txn_valid |-> (txn_id != prev_id);
endproperty

assert property (id_not_repeated)
  else $error("Transaction ID repeated consecutively!");
```

9) Below 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);
```

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

$past (signal_name, number of clock cycles, gating signal)

10) Ensure address in a write is aligned (addr % 4 == 0)

```
property aligned_addr;
@(posedge clk)
write_valid |-> (addr % 4 ==0);
endproperty

asser: assert  property (aligned_addr);
```

Here, write_valid is just a placeholder signal name that should indicate when a write operation is happening. Basically:
write_valid == 1 → means a write is getting triggered this cycle.
Only then, the assertion checks if the addr is aligned.

10.Assert signal done is only asserted after start

```
sequence s_start_seen;
  start ##1 1[*0:$];  // start happens, then any number of cycles
endsequence

property done_after_start;
  @(posedge clk) disable iff (!reset_n)
    done |-> s_start_seen;
endproperty

assert property (done_after_start)
  else $error("DONE asserted without prior START!");
```

Why not write [*0:$] directly?
SystemVerilog requires a Boolean expression before [*].
So this:
systemverilog
Copy
Edit
[*0:$] // ❌ Syntax error! Needs an expression before repetition.
is invalid.
But this:
systemverilog
Copy
Edit
1[*0:$] // ☑ Correct. 1 is always true, so it repeats "any number of cycles".
is valid and is the standard way to say "idle cycles" (don't care what's happening).

"Idle cycle" = a clock tick where your protocol or logic is not actively doing a key operation that you care about in your assertion.

12.Assert that clock frequency doesn’t skip (no 2 rising edges too close)

```
property p_no_clock_glitch;
  time clk_period_min = 5ns; // Set your minimum safe clock period here
  @(posedge clk)
    $stable(clk) ##1 $rose(clk) |->
      ($realtime - $past($realtime)) >= clk_period_min;
endproperty

assert property (p_no_clock_glitch)
  else $error("Clock period too short! Clock edges too close.");
```

Each clock rising edge must be separated by at least 5ns (or your expected period).

13.Check that packet size is always less than max allowed size.

```
parameter int MAX_PKT_SIZE = 1024;
property p_pkt_size_check;
```

```
  @(posedge clk) disable iff (!reset_n)
    pkt_valid |-> (pkt_size <= MAX_PKT_SIZE);
endproperty

assert property (p_pkt_size_check)
  else $error("Packet size exceeds maximum allowed size!");
```

On every pkt_valid (meaning a packet is being sent),
We check ➔ pkt_size must be less than or equal to MAX_PKT_SIZE.
☑ If any packet is too big ➔ assertion will fire.

14.Ensure grant signal is not active for more than 10 cycles

```
property no_high();
 @(posedge clk )
 grant |-> grant [*1:10] ##1 !grant;
 endproperty

 assert property (no_high);
```

15.Assert temperature never exceeds 120°C

```
property  temparature_prop();
@(posedge clk)
temparature <= 120;
endproperty
assert property (temparature);
```

16.Ensure data_valid is only high when data_ready is also high

```
property pro();
 @(posedge clk)
 data_valid |-> data_valid;
 endproperty
 assert property  (pro);
```

17.Assert a valid handshake occurs before any data is transferred

```
 property handshake_trns();
 @(posedge clk)
  (valid && ready ) |-> $changed(data);
 endproperty
```

valid && ready → handshake happens.
|-> → implies that
$changed(data) → data must change when handshake happens (so data transfer is not happening randomly without handshake).

18.Check that a signal toggles at least once every 20 cycles

```
property  toggle();
@(posedge clk)!$stable(sig) within [1:20];
endproperty

assert property (toggle);
```

19) t flip_flop assertion INLINE METHOD

```
module top(clk,rst,t,q,qbar);
input rst,clk,t;
output reg q;
output qbar;
always @ (posedge clk) begin
    if (rst==1)
            q <= 1'b0;
           else if ( t == 0)
            q<= 1'b0;
            else
            q <= ~q;

end

property p1;
@(posedge clk) !t |=> q==$past(q,1);
endproperty

property p2;
@(posedge clk) t |=> ~(q==$past(q,1));
endproperty

case1: assert property (p1)
      $error("assertion pass");
                  else
                  $error("assertion fail");
case2: assert property (p2)
     $error("assrtion pass");
            else
            $error("assrtion fail");
assign qbar = ~q;
endmodule
```

20) JK FLIP FLOP ASSERTIONS I WROTE ASSERTIONS IN SEPARATE BLCOK AND IN TB I BIND IT

```
module jk_ff(clk,rst,j,k,q,qbar);
```

```
input clk,rst,j,k;
output reg q;
output qbar;
always@(posedge clk) begin
  if (rst)
     q <= 1'b0;
            else  begin
              case(j&&k)
                        2'b00 : q <= q;
                        2'b01 : q <= 1'b0;
                        2'b00 : q <= 1'b1;
                        2'b00 : q <= ~q;
                                default q = 1'bx;
                                endcase
             end
           end
           assign qbar = ~q;
endmodule

module assrt(input j,k,clk,rst,q);

property rest;
 @(posedge clk) rst |=> !q;
 endproperty

 property no_change;
   @(posedge clk)
           !j && !k |=> q == $past(q,1);
 endproperty

 property reset;
 @(posedge clk) !j && k |=> !q;
 endproperty

 property set ;
 @(posedge clk) j && !k |=> q;
 endproperty

 property toggle;
  @(posedge clk) j && k |=> q == ~($past(q,1));
 endproperty

    res_t    : assert property (rest);
    nochange : assert property (no_change);
    r_ty     : assert property (reset);
             r_ty     : assert property (reset);
    to_l     : assert property (toggle);
```

```
endmodule

 module top();
    reg clk,rst,j,k;
          wire q, qbar;
           jk_ff d1(.clk(clk),
                .rst(rst),
                                .j(j),
                                .k(k),
                                .q(q),
                                .qbar(qbar)
                               );
//BINDING THE ASSERION HERE                                bind jk_ff assrt r1 (     .clk(clk),
                                                      .rst(rst),
                                                            .j(j),
                                                            .k(k),
                                                            .q(q)
                                                            //.qbar(qbar)
                                                                  );
          initial begin
      clk = 0;
                  #10
                  rst=1;

                  forever #5 clk=~clk;
          end

          initial begin
            #10
            rst = 0;
                  #10
                  j=1'b0;k=1'b0;
                  #10
                  j=1'b0;k=1'b1;
                  #10
                  j=1'b1;k=1'b0;
                  #10
                  j=1'b1;k=1'b1;
          end

          initial begin
             #100 $finish;
                  end

endmodule
```

21) write a assertion for d filp flop

```
property d_ff_reset_con();
  @(posedge clk) reset |-> !q;
endproperty

property no_change();'
  @(posedge clk) !d |=> d==$past(q,1);
endproperty

property d_1_condition();
 @(posedge clk ) d |=> q==d;
endproperty

        d1: assert property(d_ff_reset_con);
         d2:assert property (no_change);
           d3: assert property (d_1_condition);
```

21) if asignal data rises to ahigh value with in 5 clk cycles after the reset siganl deassetrted

```
  property rakeh_pro();
   @(posedge clk) (reset==0) |-> ##[1:5] (data==1);
  endproperty

p4: assert property (rakesh_prop);
```

22) sigan a is high then make sure once it is asserted then it need not to be assserted again

```
 property rakes_prop();
  @(posedge clk) $rose(a) |=>  ##[0:$] !$rose(a);
 endproperty
p3: assert property (rakesh_prop)
```

23) Valid is high with in the 1 to 3 cloclk cycle ready is high

```
 property rakesh_pro();
 @(posedge clk) disable iff (rst)
  valid |-> ## [1:3] ready;
endproperty
assert property (rakesh_prop)
 $display("assertion is passed");
 else
  $display("assertion is failed");
```

24) Valid is high with in the same cycle ready is high

```
Property rakesh_prop();
@(posedge clk)   disable iff(rst)
valid |-> ready;
```

```
Endproperty
p1:assert property (rakesh_prop);
```

25) Valid is high next cycle ready is high

```
 property rakesh_pro();
   @(posedge clk) disable iff (rst)
  valid |=> ready;
 endproperty
 assert property (rakesh_prop)
   $display("assertion is passed");
   else
     $display("assertion is failed");
```