1. Write an assertion check to make sure that a signal is high for a minimum of 2 cycles and a maximum of 6 cycles.

```
property p_sig_up_2to6_cycles;
    @(posedge clk)
        disable iff(!rst);
    $rose(valid) |-> signal_a[*2:6];
endproperty

p_sig_up_2to6_cycles_A:
 assert property(p_sig_up_2to6_cycles)
     $info("p_sig_up_2to6_cycles Passed");
 else $error("Assertion p_sig_up_2to6_cycles Failed");
```

2. Are following assertions equivalent:
   @(posedge clk) req |=> ##2 $rose(ack);
   @(posedge clk) req |-> ##3 $rose(ack);

```
"Answer": Both are equal, will reflect on the same cycle.
```

3. For a synchronous FIFO of depth = 16, write an assertion for the following scenarios. Assume a clock signal(clk), write and read enable signals, full flag and a word counter signal.
   a. If the word count is >15, FIFO full flag set.
   b. If the word count is 15 and a new write operation happens without a simultaneous read, then the FIFO full flag is set.

```
"[a]" property p_fifo_full0;
       @(posedge clk)
          disable iff(!rst);
      (counter > 15) |-> fifo_full;
      endproperty

      p_fifo_full0_A:
        assert property(p_fifo_full0)
        else $error("ASSERTION p_fifo_full0 FAILED");
```

- **Augustin JK**

```
"[b]" property p_fifo_full1;
        @(posedge clk)
            disable iff(!rst);
        ((counter == 15) && write && !read) |-> fifo_full;
      endproperty

      p_fifo_full1_A:
        assert property(p_fifo_full1)
        else $error("ASSERTION p_fifo_full1_A FAILED");
```

4. Write an assertion checker to make sure that an output signal never goes X?

```
property p_to_check_unknown;
  @(posedge clk)
    disable iff(!rst);
        $rose(en) |-> !($isunknown(out));
endproperty

p_to_check_unknown_A:
    assert property(p_to_check_unknown)
        else $error("ASSERTION p_to_check_unknown_A FAILED");
```

5. Write an assertion to make sure that a 5-bit grant signal only has one bit set at any time?

```
logic[4:0] grant;

property p_one_bit_set;
    @(posedge clk)
        disable iff(!rst);
    $rose(req) |-> ($onehot(grant));
endproperty

p_one_bit_set_A:
    assert property(p_one_bit_set)
        else $error("ASSERTION p_one_bit_set_A FAILED");
```

- **AUGUSTIN JK**

6. Write an assertion which checks that once a valid request is asserted by the master, the arbiter provides a grant within 2 to 5 clock cycles.

```
property p_req_to_grant;
    @(posedge clk)
        disable iff(!rst);
    $rose(req) |-> ##[2:5] grant;
endproperty

p_req_to_grant_A:
    assert property(p_req_to_grant)
        else $error("ASSERTION p_req_to_grant_A FAILED");
```

7. How can you disable an assertion during active reset time?

```
"For ACTIVE HIGH:"
    disable iff(reset);

"For ACTIVE LOW:"
    disable iff(!reset);
```

8. How can all assertion be turned off during simulation (with active assertions)?

```
"USING:" $assertkill;
```

9. As long as signal_a is up, signal_b should not be asserted. Write an assertion.

```
property p_siga_up_sigb_down;
    @(posedge clk)
        disable iff(!rst);
    signal_a |-> !(signal_b);
endproperty

p_siga_up_sigb_down_A:
    assert property(p_siga_up_sigb_down)
        else $error("ASSERTION p_siga_up_sigb_down_A FAILED");
```

- **AUGUSTIN JK**

10. The signal_a is a pulse; it can only be asserted for one cycle, and must be deasserted in the next cycle.

```
property p_siga_up_for_onecycle;
    @(posedge clk)
        disable iff(!rst);
    $rose(signal_a) ##1 $fell(signal_a);
endproperty

p_siga_up_for_onecycle_A:
    assert property(p_siga_up_for_onecycle)
        else $error("ASSERTION p_siga_up_for_onecycle_A FAILED");
```

11. Signal_a and signal_b can only be asserted together for one cycle; in the next cycle, at least one of them must be deasserted.

```
property p_siga_sigb_up_for_onecycle;
    $rose(signal_a && signal_b)  ##1 !(signal_a or signal_b);
endproperty

p_siga_sigb_up_for_onecycle_A:
    assert property(p_siga_sigb_up_for_onecycle)
        else $error("ASSERTION p_siga_sigb_up_for_onecycle_A FAILED");
```

12. When signal_a is asserted, signal_b must be asserted, and must remain up until one of the signals signal_c or signal_d is asserted.

```
property p_sigb_up_until_sigc_or_sigd;
    @(posedge clk)
        disable iff(!rst);
    $rose(signal_a) |-> (signal_b until (signal_c or signal_d));
endproperty

p_sigb_up_until_sigc_or_sigd_A:
    assert property(p_sigb_up_until_sigc_or_sigd)
        else $error("ASSERTION p_sigb_up_until_sigc_or_sigd_A FAILED");
```

- AUGUSTIN JK

13. After signal_a is asserted, signal_b must be deasserted, and must stay down until the next signal_a.

```
property p_sigb_down_until_next_siga;
    @(posedge clk)
        disable iff(!rst);
    $rose(signal_a)  |-> (!(signal_b) until signal_a);
endproperty

p_sigb_down_until_next_siga_A:
    assert property(p_sigb_down_until_next_siga)
        else $error("ASSERTION p_sigb_down_until_next_siga_A FAILED");
```

14. If signal_a is received while signal_b is inactive, then on the next cycle signal_c must be inactive, and signal_b must be asserted.

```
sequence s_sigc_down_sigb_up;
    (!(signal_c) and signal_b)
endsequence

property p_siga_u_sigb_d_next_sigc_d_sigb_u;
    @(posedge clk)
        disable iff(!rst);
    (!(signal_b) && signal_a) |-> s_sigc_down_and_sigb_up;
endproperty

p_siga_u_sigb_d_next_sigc_d_sigb_u_A:
    assert property(p_siga_u_sigb_d_next_sigc_d_sigb_u)
        else $error("ASSERTION p_siga_u_sigb_d_next_sigc_d_sigb_u_A FAILED");
```

15. signal_a must not be asserted together with signal_b or with signal_c.

```
property p_siga_d_within_sigb_or_sigc;
    @(posedge clk)
        disable iff(!rst);
    (!(signal_a) within (signal_b or signal_c));
endproperty

p_siga_d_within_sigb_or_sigc_A:
    assert property(p_siga_d_within_sigb_or_sigc)
        else $error("ASSERTION p_siga_d_within_sigb_or_sigc_A FAILED");
```

- **AUGUSTIN JK**

16. In a RESP operation, request must be true immediately, grant must be true 3 clock cycles later, followed by request being false, and then grant being false.

```
property p_req_grant;
    @(posedge clk)
        disable iff(!rst);
    (state == RESP) |-> req ##3 grant ##1 !req ##1 !grant;
  endproperty

p_req_grant_A:
    assert property(p_req_grant)
        else $error("ASSERTION p_req_grant_A FAILED");
```

17. Request must true at the current cycle; grant must become true sometime between 1 cycle after request and the end of time.

```
request ##[1:$] grant;
 (or)
request ##[+] grant;
```

18. Req must eventually be followed by ack, which must be followed 1 cycle later by done.

```
property p_req_ack;
    @(posedge clk)
        $rose(req) |-> ##[1:$] ack;
   endproperty

property p_ack_done;
    @(posedge clk)
        $rose(ack) |-> ##1 done;
endproperty

p_req_ack_A:
    assert property(p_req_ack)
        else $error("ASSERTION p_req_ack_A FAILED");
p_ack_done_A:
    assert property(p_ack_done)
        else $error("ASSERTION p_ack_done_A FAILED");
```

- **AUGUSTIN JK**

19. The active-low reset must be low for at least 6 clock cycles.

```
property p_rstN_6;
    @(posedge clk)
        !resetN[*6];
endproperty

p_rstN_6_A:
    assert property(p_rstN_6)
        else $error("ASSERTION p_rstN_6_A FAILED");
```

20. Enable must remain true throughout the entire ack to done sequence.

```
property p_en_throughout_acktodone;
    @(posedge clk)
        disable iff(!rst);
    (en) throughout (ack && done);
endproperty

p_en_throughout_acktodone_A:
    assert property(p_en_throughout_acktodone)
        else $error("ASSERTION p_en_throughout_acktodone_A FAILED");
```

21. Write an assertion for glitch detection.

```
property p_glitch_detection;
    @(posedge clk)
        disable iff(!rst);
    $fell(en) |=> $stable(data);
endproperty

p_glitch_detection_A:
    assert property(p_glitch_detection)
        else $error("ASSERTION p_glitch_detection_A FAILED");
```

22. If signal_a is active, then signal_b was active 3 cycles ago.

```
property p_siga_up_sigb_up_past_3;
    @(posedge clk)
        disable iff(!reset);
    signal_a && $past(signal_b, 3);
endproperty

p_siga_up_sigb_up_past_3_A:
    assert property(p_siga_up_sigb_up_past_3)
    else $error("ASSERTION p_siga_up_sigb_up_past_3_A FAILED");
```

23. If the state machine reaches active1 state, it will eventually reach active2 state.

```
property p_active1_eventually_active2;
    @(posedge clk)
        disable iff(!rst);
    (STATE == active1) |-> ##[*] (STATE == active2);
endproperty

p_active1_eventually_active2_A:
    assert property(p_active1_eventually_active2)
    else $error("ASSERTION p_active1_eventually_active2_A FAILED");
```

24. Write an assertion: A high for 5 cycles and B high after 4 continuous highs of A and finally both A and B are high?

```
____|====|_____|======================|______  A

________________________________|=====|______  B
```

```
property p_sigA_up_5cycles_sigB_up_within_sigA;
    @(posedge clk)
        disable iff(!rst);
    (##4 $rose(sigB)) within ($rose(sigA)[*5]);
endproperty

p_sigA_up_5cycles_sigB_up_within_sigA_A:
    assert property(p_sigA_up_5cycles_sigB_up_within_sigA)
    else $error("ASSERTION p_sigA_up_5cycles_sigB_up_within_sigA_A FAILED");
```

- AUGUSTIN JK

25. Write an assertion: On rose of a, wait for rose of b or c. If b comes first, then d should be 1. If c comes first d should be zero.

```
sequence seq1;
    ($rose(b) || $rose (c)) [->1];
endsequence

sequence seq2;
    ((c && !d) || (b && d));
endsequence

property p_a_up_seq1_seq2;
    @(posedge clk)
        disable iff(!rst);
    $rose(a) |-> seq1 ##0 seq2;
endproperty

p_a_up_seq1_seq2_A:
    assert property(p_a_up_seq1_seq2)
    else $error("Assertion p_a_up_seq1_seq2_A FAILED");
```

THANK YOU 😊

- AUGUSTIN JK