**ASSERTIONS**

**Basic Level**

1. **What are SystemVerilog assertions (SVA)? Why are they used?**
2. **What is the difference between immediate and concurrent assertions?**
3. **Explain the syntax of an immediate assertion.**
4. **What are the key components of a concurrent assertion?**
5. **What is a property in SystemVerilog? How is it different from a sequence?**
6. **How do you disable an assertion during simulation?**
7. **Write an immediate assertion to check if a signal data is always greater than zero.**
8. **What does the disable iff construct do in concurrent assertions?**
9. **Explain the difference between |-> and |=>.**
10. **What is $fatal, $error, and $warning in assertions?**

**Intermediate Level**

1. **What is the role of temporal operators in assertions? Provide examples.**
2. **Explain the ## operator and its significance in assertions.**
3. **What happens if an assertion fails during simulation?**
4. **Write a concurrent assertion to check if signal a is high for two consecutive clock cycles.**
5. **What are the advantages of using assertions in functional verification?**
6. **How would you use assertions to verify an AMBA protocol (e.g., AXI or AHB)?**
7. **Explain the difference between sequence, property, and assert property.**
8. **How do you debug assertion failures in a simulation environment?**
9. **What is the significance of using cover properties with assertions?**
10. **What is the difference between overlapped and non-overlapped sequences?**

**Advanced Level**

1. **How do you integrate assertions in a UVM environment?**
2. **Explain the use of strong and weak operators in concurrent assertions.**
3. **What are sampled values in SVA, and why are they important?**
4. **Write a property to check a handshake protocol where req must be followed by ack within three clock cycles.**
5. **How would you verify low-power designs using assertions (e.g., UPF flow)?**
6. **What is the significance of first\_match and throughout operators?**
7. **Write an assertion to check if a reset signal (rst) remains low for at least five clock cycles after being deasserted.**
8. **Explain the difference between always and eventually temporal operators in SVA.**
9. **Describe the assertion testing flow in formal verification tools like JasperGold or Questa Formal.**
10. **What are checker libraries in SVA, and how are they used?**

**Practical and Scenario-Based Questions**

1. **How do you monitor functional coverage using assertions?**
2. **What are common pitfalls when writing assertions, and how do you avoid them?**
3. **Write a property for an AXI burst transaction that ensures the burst length is not exceeded.**
4. **How would you check for glitches or setup/hold violations using assertions?**
5. **How do you manage assertion failures in regression testing?**
6. **Write an assertion to check that two signals, data and valid, toggle in a specific pattern (e.g., valid -> data).**
7. **Explain the concept of overlapping sequences and provide an example.**
8. **How would you optimize assertions for performance in simulation?**
9. **Describe a scenario where assertions helped identify a complex bug in your project.**
10. **How would you use assertions to verify pipeline stages in a processor design?**

**Basic Scenario-Based Questions**

1. **Clock Stability Check**  
   Write an assertion to ensure that the clock signal does not glitch (e.g., it must remain stable for at least one clock cycle).

**Assert property @(posedge clk) $stable(clk) [\*1:$];**

1. **Reset Signal Assertion**  
   Write an assertion to check that a reset signal remains active for at least two clock cycles after being asserted.

**Assert property @(posedge clk) reset [\*2:$];**

1. **Data Signal Assertion**  
   Write an assertion to ensure that a data signal is not X or Z during normal operation.

**Assert property @(posedge clk) normal\_op |-> !(data == X || data == z);**

1. **Enable Signal Assertion**  
   Write an assertion to verify that the enable signal goes high only after the reset signal has been de-asserted.

Assert property @(posedge clk ) disble iff(reset) Reset |=>$rose(en);

1. **Output Validity**  
   Assert that an output out\_valid is asserted only if in\_ready and data\_valid are both high.

**Assert property @(posedge clk) disable iff (reset) (in\_ready and data\_valid) |-> out\_valid**

**Intermediate Scenario-Based Questions**

1. **Sequence Monitoring**  
   Write an assertion to check that when a signal start is high, a sequence of events (a, then b, then c) occurs within 5 clock cycles.

Sequence seq\_a;

A ##1 b ##1 c;

Endsequence

Property p1;

@(posedge clk) start |-> ##[1:5] seq\_a;

Endproperty

Assert property (p1);

1. **FIFO Overflow Check**  
   Write an assertion to detect and flag a FIFO overflow condition when write\_enable is asserted but the FIFO is full.

**Assert property @(posedge clk) (wr\_en&&fifo\_full) |-> 0;**

**Handshake Protocol**  
Create an assertion to verify a valid handshake where req must be followed by ack within 3 clock cycles.

Property p1 ;

@(posedge clk) req |-> ##[1:3] ack’

Assert property (p1) else $error(“error”);

1. **Bus Idle Assertion**  
   Verify that a bus remains idle (no read or write transactions) when the bus\_enable signal is low.

Property p1;

@(posedge clk) !(en) |-> !(rd || wr) ;

Endproperty

Assert property (p1) else $error(“”);

1. **Power-Up Sequence**  
   Write an assertion to check that the power\_good signal becomes high only after the reset is de-asserted and voltage\_stable is high for at least 3 cycles.

**Property p1**

**@(posedge clk) disable iff(reset) power\_good |-> ##[1:3] voltage\_stable;**

**Endproperty**

**Assert property (p1) else $error(“”);**

**Advanced Scenario-Based Questions**

1. **AXI Protocol Validation**  
   Write assertions to verify the following AXI protocol requirements:
   * ARREADY must be high within two cycles of ARVALID going high.
   * AWVALID and ARVALID must not be high at the same time.

Property p1;

@(posedge clk)

$rose(ARVALID) |-> ##[1:2] ARREADY;

Endpproperty

Assert property (p1) else $error(“ERROR”);

Property p2 ;

@(posedge clk)

!(AWVALID && ARVALID) ;

Endproperty

Assert property (p2) else $error(“ERROR”);

1. Sequence With Disable Condition  
   Write an assertion to monitor a signal data\_out for a specific sequence but disable the check when reset is high.

Property p1;

@(posedge clk) disable iff (reset) data\_out == 8’FF |-> data\_out == 8’AA;

Endproperty

Assert property (p1);

1. **Conditional Sequence**  
   Create an assertion to verify that if signal mode is 1, then signal output toggles within 4 clock cycles after signal input toggles.

**Property p1;**

@(posedge clk)

Mode == 1 |-> ##[1:4] !($stable(output)) |=> ##1 !($stable(input));

Endproperty

Assert property (p1);

1. **Data Integrity Check**  
   Verify that signal data remains constant between valid and ready handshakes.

**Property p1;**

**@(posedge clk) disable iff (reset)**

**(valid && ready) |-> ##1 $stable(data);**

**Endproperty**

**Assert property (P1);**

1. **Read-Write Mutual Exclusivity**  
   Write an assertion to ensure that read\_enable and write\_enable are not high simultaneously.

Assert property @(posedge clk) (!(RD\_EN && WR\_EN));

**Protocol-Specific and Real-World Scenario Questions**

1. **AMBA Protocol**  
   Write an assertion for the AMBA APB protocol to ensure that PREADY is asserted only after PSEL and PENABLE are high.

**Property p1;**

**@(posedge clk) disable iff(reset)**

**(PSEL && PENABLE) |=> PREADY;**

**Endproperty**

**Assert property (p1) else $fatal(“”);**

1. **DDR Timing**  
   Write an assertion to check the timing relationship between read\_enable and data\_out in a DDR interface. Ensure there is a two-cycle delay between them.

Property p1;

@(posedge clk) disable iff(reset)

Rd\_en |=> ##[2] data\_out;

Endproperty

Assert property (p1) else $error(“”);

1. **UART Protocol**  
   Verify that the start\_bit of a UART frame is followed by exactly 8 data bits and 1 stop bit.

Property p1;

@(posedge clk) disable iff (reset)

start\_bit |=>

foreach(data[i])

data[i] ##1;

stop\_bit;

Endproperty

Assert property (p1);

1. **Interrupt Latency**  
   Write an assertion to ensure that an interrupt signal irq is serviced within 5 clock cycles of being asserted.

**Property p1;**

**@(posedge clk) disable iff(reset)**

**Irq |-> ##[1:5] irq;**

**Endproperty**

**Assert property (p1);**

1. **Pipeline Verification**  
   Ensure that a pipeline stage does not overwrite the valid signal until the ready signal from the next stage is asserted.

**Property p1;**

**@(posedge clk) disable iff(reset) valid |-> ##[1:$]ready;**

**Endproperty**

**Assert property(p1)**

**Corner Cases and Debugging Scenario Questions**

1. **Glitch Detection**  
   Write an assertion to flag any glitch on the signal clk (i.e., any transition shorter than one clock period).

**21.Power Management Assertion**  
Ensure that signal clk stops toggling when power\_down is asserted.

Property p1;

@posedge clk disable iff(reset)

power\_down |-> ($stable(clk))

endproperty

assert property(p1);

1. **Deadlock Detection**  
   Write assertions to check for deadlock conditions in a multi-threaded transaction environment.
2. **Out-of-Order Transactions**  
   Create an assertion to verify that transactions on a bus always follow a strict in-order protocol.
3. **Timeout Check**  
   Ensure that a response signal is generated within 10 cycles after a request signal is asserted.

**Req |=> ##[=1:10]resp**

**Debugging and Practical Scenario Questions**

1. **Assertion for Debugging**  
   How would you write an assertion to check that an FSM never enters an illegal state during operation?

**Typedef enum logic [2:0]{**

**IDLE = 3’b000;**

**RUN=3’b001;**

**STOP = 3’b010;**

**} f\_state;**

**F\_state fsm\_state;**

**Property p1;**

**@(posedge clk) disable iff(reset)**

**(Fsm\_state inside {Idle, run, stop});**

**Endproperty**

**Assert property (p1) else ;**

1. **Error Flag Monitoring**  
   Write an assertion to ensure that an error signal is cleared within 2 cycles of being asserted.

**Property p1;**

**@(posedge clk) disable iff(reset)**

**error != ##[1:2] $fell(error);**

**endproperty**

**assert property(p1) else $fatal(“”);**

1. **Multiple Clocks**  
   Verify that a cross-domain signal from clock domain clk1 is synchronized correctly before being used in clk2.

**Property p1;**

**@(posedge clk1) disable iff (reset)**

**Async\_sig |=> ##1 @(posedge clk2) $(stable(async\_signal));**

**endproperty**

1. **Protocol Timeout**  
   Write an assertion to flag a timeout error if ack is not received within 6 cycles of req.

Property p1;

@(posedge clk) disable iff(reset) req |-> ##[1:6] ack;

Endproperty

Assert property (p1) else $error(“Timeout error”);

1. **Branch Misprediction**  
   Write an assertion to check that a branch misprediction in a processor pipeline is correctly handled by flushing the pipeline.

**Exercise 1: Signal Stability**

**Question:**  
Write an assertion to check that a signal clk remains stable during the simulation (no glitches).

Assert property @(posedge clk) $stable(clk) else $error(“glitch”);

**Exercise 2: Active High Signal**

**Question:**  
Write an assertion to ensure that a signal enable is always high during normal operation.

Assert property @(posedge clk) normal\_op |-> en;

**Exercise 3: Range Check**

**Question:**  
Write an assertion to verify that a signal data always lies between 0 and 255.

Assert(data>0 && data<255);

**Exercise 4: Signal Toggle**

**Question:**  
Write an assertion to check that a signal flag toggles its value (changes from 0 to 1 or 1 to 0) every clock cycle.

Assert property @(posedge clk) !($stable(flag)) else $error(“”);

**Exercise 5: Reset Assertion**

**Question:**  
Write an assertion to ensure that a reset signal remains active for at least 3 clock cycles after it is asserted.

Property p1;

@(posedge clk) reset |-> ##[\*3:$] reset

Endproperty

Assert property (p1) else $error(“”);

**Exercise 6: Rising Edge Detection**

**Question:**  
Write an assertion to check that a signal start transitions from low to high ($rose(start)).

Assert property @(posedge clk) $rose(start) else $error(“error”);

**Exercise 7: Mutual Exclusivity**

**Question:**  
Write an assertion to ensure that two signals write\_enable and read\_enable are not high at the same time.

Property p1;

@(posedge clk) !(wr\_en && rd\_en) |-> $error(“”);

Endproperty

Assert property (p1);

**Exercise 8: Sequence Check**

**Question:**  
Write an assertion to verify that after a signal start is asserted, the sequence of signals (a, then b, then c) occurs in order.

Property p1;

@(posedge clk) start |-> (a ##1 b ##1 c);

Endproperty

Assert property (p1) else $error(“”);

**Exercise 9: FIFO Overflow**

**Question:**  
Write an assertion to detect when a write\_enable signal is asserted while the FIFO is full (fifo\_full).

Property p1;

@(posedge clk)

FIFO\_FULL |-> !(WRITE\_ENABLE)

Endproperty

Assert property (p1);

**Exercise 10: Data Validity**

**Question:**  
Write an assertion to ensure that a data signal is never in an unknown state (X or Z) during normal operation.

Property p1;

@(posedge clk) normal\_op |-> !($isunknown(data));

Endproprty

Assert property (p1) else $error(“”);

**Exercise 11: Timing Window**

**Question:**  
Write an assertion to ensure that a signal ready goes high within 3 clock cycles after valid is asserted.

Property p1;

@(posedge clk) valid |-> ##[1:3] $rose(ready);

endProperty

assert property (p1) else $error(“”);

**Exercise 12: One-Hot Encoding**

**Question:**  
Write an assertion to check that out of three signals (s0, s1, s2), only one signal is high at any given time.

Property p1 ;

@(posedge clk) $onehot({s0,s1,s2});

Endproperty

Assert property (p1) else $fatal(“error”);

**Exercise 13: Handshake Protocol**

**Question:**  
Write an assertion to verify that when a req signal is high, an ack signal becomes high in the next clock cycle.

Property p1 ;

@(posedge clk) req |=> $rose(ack);

Endproperty

Assert property (p1)

Else $fatal(“error”);

**Exercise 14: Conditional Assertion**

**Question:**  
Write an assertion to ensure that when mode is high, the signal output is always greater than input.

Property p1;

@(posedge clk) mode |-> (output>input);

Endproperty

Assert property (p1)

Else $fatal(“assertion failed”);

**Exercise 15: Event Count**

**Question:**  
Write an assertion to check that a signal event goes high exactly 5 times during a simulation.

Property p1;

@(posedge clk) $rose(event) [=5];

Endproperty

Assert property (p1)

Else $fatal(“assertion failed”);