Ain Shams University
Faculty of Engineering
Computer and Systems Department
Spring 2024

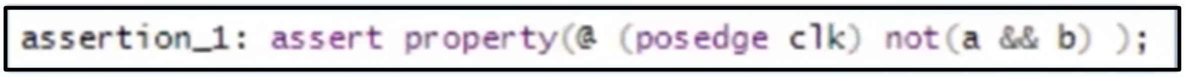
CSE 313s
Selected Topics in Computer Engineering
Sheet 7

1. What is SVA? SVA is a language construct used to define temporal properties of digital circuits.

2. What are the benefits of using assertions??

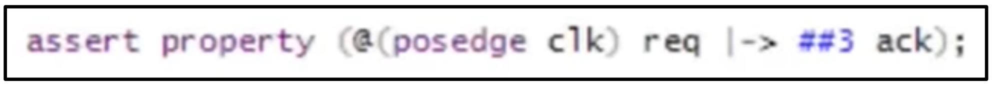
   1. Early Bug Detection
   2. Increased Verification Coverage
   3. Automated Verification
   4. Documentation and Specification
   5. Reuse and Maintainability

3. Write a System Verilog assertion to check the following:
   a and *b* are two signals, which can be active at any time, but should never be active together.



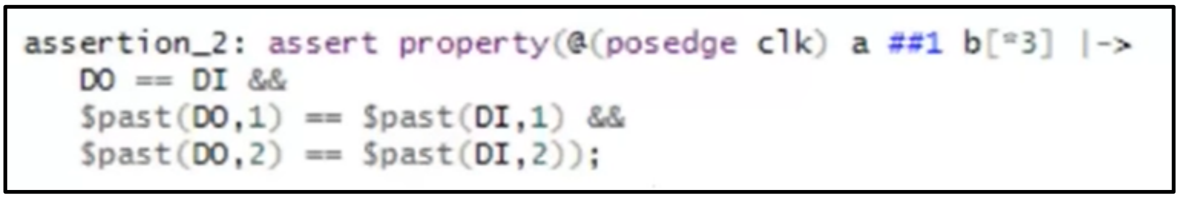
```
assertion_1: assert property(@ (posedge clk) not(a && b) );
```

4. Write a System Verilog assertion to check the following:
   Every time the request *req* goes high, *ack* arrives exactly 3 clocks later



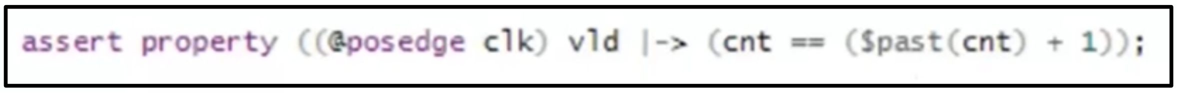
```
assert property (@(posedge clk) req |-> ##3 ack);
```

5. Write a System Verilog assertion to check the following:
   If *a* is high at a clock edge, followed by 3 consecutive cycles in which *b* is high, then in each of the 3 cycles the data output *DO* is equal to the data input *DI*.



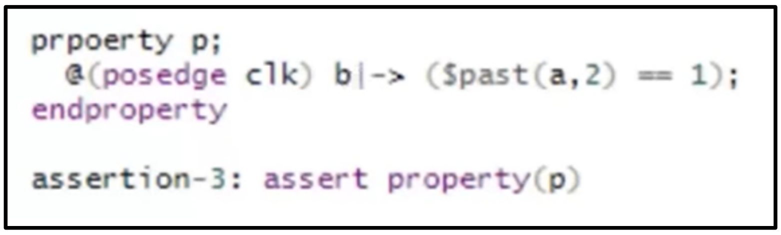
```
assertion_2: assert property(@(posedge clk) a ##1 b[*3] |->
    DO == DI &&
    $past(DO,1) == $past(DI,1) &&
    $past(DO,2) == $past(DI,2));
```

6. Write a System Verilog assertion to check the following:
   Every time the valid signal *vld* is high, the *cnt* is incremented



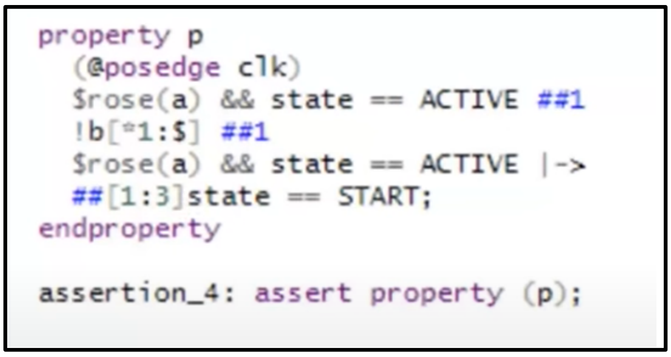
```
assert property ((@posedge clk) vld |-> (cnt == ($past(cnt) + 1));
```

7. Write a System Verilog assertion to check the following:
If *b* is high at a clock edge, then 2 cycles before that, *a* was high.



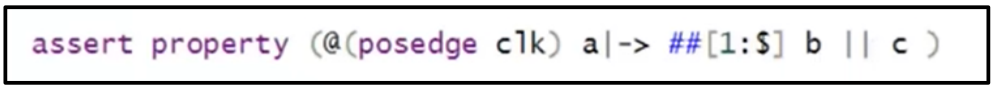
```
prpoerty p;
  @(posedge clk) b|-> ($past(a,2) == 1);
endproperty

assertion-3: assert property(p)
```

8. Write a System Verilog assertion to check the following:
If there are two occurrences of "*a*" rising while state = ACTIVE, and no "*b*" occurs between them, then within 3 cycles of the second rise of "*a*", START must occur.



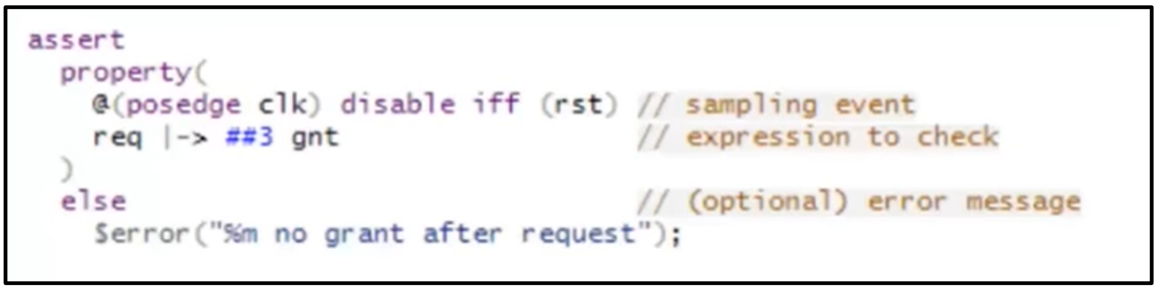
```
property p
  (@posedge clk)
  $rose(a) && state == ACTIVE ##1
  !b[*1:$] ##1
  $rose(a) && state == ACTIVE |->
  ##[1:3]state == START;
endproperty

assertion_4: assert property (p);
```

9. Write a System Verilog assertion to check the following:
Every "*a*" must eventually be acknowledged by "*b*", unless "*c*" appears any time before "*b*" appears.



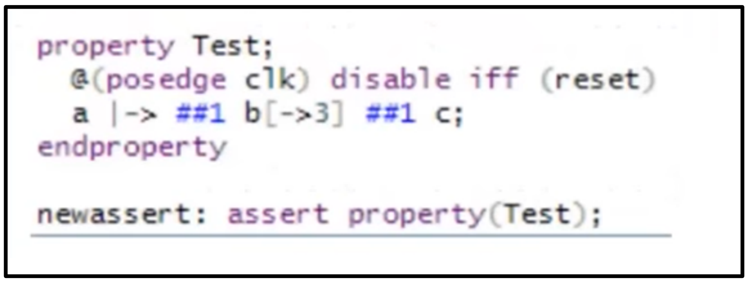
```
assert property (@(posedge clk) a|-> ##[1:$] b || c )
```

10. Write a System Verilog assertion to check the following:
Every time the request *req* goes high, *gnt* arrives exactly 3 clocks later. If this is not achieved an error is reported with the message: "no grant after request".
But this assertion should only be checked if the reset signal, *rst*, is not active.



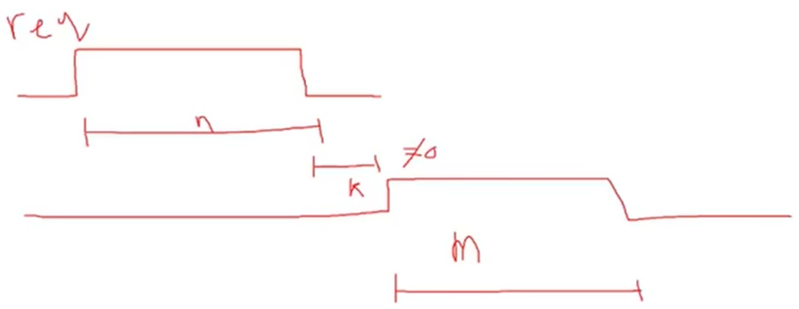
```
assert
  property(
    @(posedge clk) disable iff (rst) // sampling event
    req |-> ##3 gnt                  // expression to check
  )
  else                               // (optional) error message
    $error("%m no grant after request");
```

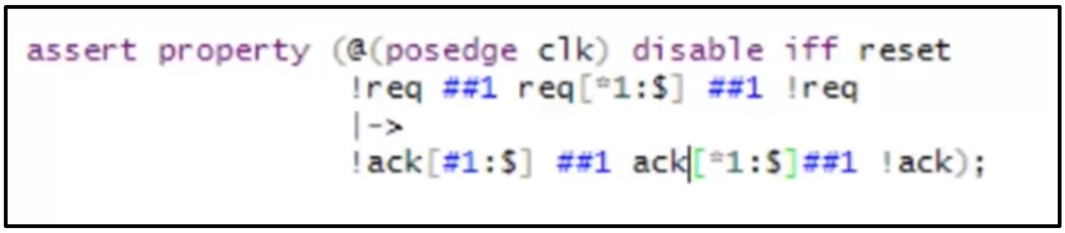
11. Write a System Verilog assertion to check the following:
If a signal "*a*" is high on a given posedge of the clock, the signal "*b*" should be high for 3 clock cycles followed by "*c*" that should be high after "*b*" is high for the third time. During this entire sequence, if reset is detected at any point, the checker will stop.



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

newassert: assert property(Test);
```

12. Write a System Verilog assertion to check the following:
A request "*req*" is high for one or more cycles, then returning to zero, is followed after one or more cycles, by an acknowledge, "*ack*" for one or more cycles before "*ack*" returns to zero. "*ack*" must be zero in the cycle in which "*req*" returns to zero. During this entire sequence, if reset is detected at any point, the checker will stop.





```
assert property (@(posedge clk) disable iff reset
                 !req ##1 req[*1:$] ##1 !req
                 |->
                 !ack[#1:$] ##1 ack[*1:$]##1 !ack);
```