CSE 140 Final Exam (Version A)

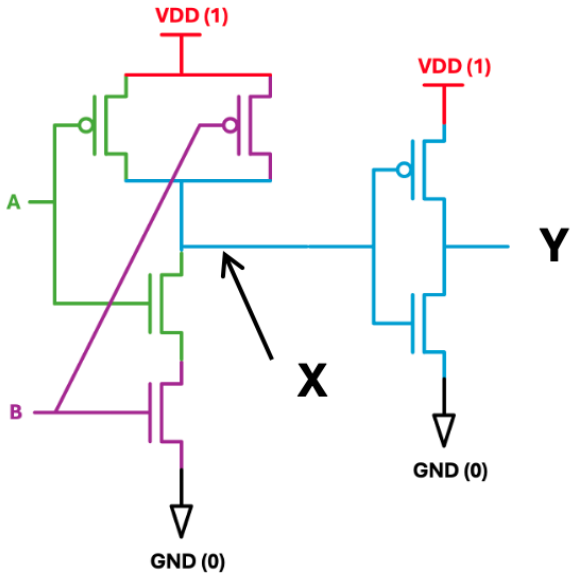
Name: _____ **PID:** _____

Please record all your answers on the bubble sheet. Exam policies:

- You are allowed to use one letter-size, double-sided cheat sheet.
- This exam is closed book. No phones or calculators allowed.
- You can use the back of the question papers as scratch sheets.
- We do not take questions during the exam.
- If you think a question is wrong or unclear, write your reasoning down on the back of the bubble sheet. We will consider that and may give points to everyone.
- Some terms are explicitly defined in the exam to ensure consistent terminology. However, most of these terminologies have already been covered in class.
- There are 55 questions. Each question is worth 2 points. There's only one correct option to each question except questions 14 and 15. You will only earn points with question 14 and 15 if your selections match the answers exactly.
- You will be graded out of 100 points. (10 bonus points!)
- Please write down you name, PID, and select the version number on the bubble sheet.

========== Section A: Logic gates and transistors ==========

This is a logic gate in CMOS.



1. If inputs A=0, B=1, specify the logic levels at X and Y.

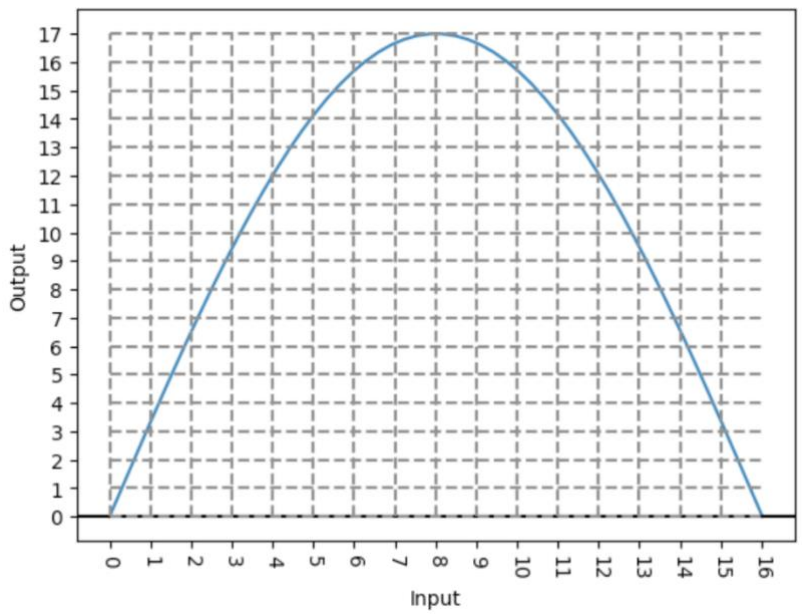
A. X=0, Y=0 B. X=0, Y=1 C. X=1, Y=0 D. X=1, Y=1

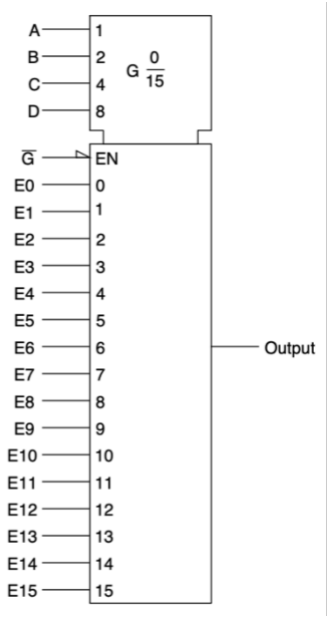
2. What logic gate is this, where Y = gate(A, B)?

A. AND gate B. NAND gate C. OR gate D. NOR gate

========== Section B: Look-up Tables and K maps ==========

Sometimes it is easier to make a look-up-table (LUT) than it is to directly compute the function. The following graph on the left plots the first half-period of a sinusoidal wave. The input is restricted to the integer values from 0 through 16, inclusive. For each input, the corresponding output is rounded to the nearest integer. Assume that both the input and output are represented as unsigned binary numbers. Answer questions 3-5.





On the top right is an example of a 4-input, 1-output LUT. It has 16 data lines. By configuring the data lines E0-E15, the LUT can represent any truth table. A, B, C, D are the inputs, and each data line is one line in the truth table.

3. What is the minimum number of bits required to represent all possible input values in the waveform?
A. 3 B. 4 C. 5 D. 6

4. What is the minimum number of bits required to represent all possible output values in the waveform?
A. 3 B. 4 C. 5 D. 6

5. The LUT has M input bits and N output bits. How many data lines does this LUT have in total?
A. $2^M \cdot N$ B. $M \cdot N$ C. $2^M \cdot 2^N$ D. $M \cdot 2^N$

DC \ BA	00	01	11	10
00	1	0	0	1
01	1	1	1	0
11	1	1	1	0
10	1	0	0	1

6. What is the minimum-literal SOP expression for the above K-map?
A. `Y = A'C' + AC + A'B'C`
B. `Y = A'C' + AC + A'B'`
C. `Y = AC' + A'C + A'B'`
D. `Y = A'C' + AC + B'CD`

========== Section C: Fixed Point Arithmetic ==========

For a fixed-point binary number, the **integer code** is the signed integer obtained by interpreting its bit pattern as a two's-complement integer. The relation between the integer code and its real value is:

$$\text{Real value} = \text{Integer code} \times \text{Quantization scale}$$

For example, consider the signed fixed-point number `4'b0101`, with quantization scale of 0.25 (2 fractional bits, 2 integer bits including the sign bit). Its real value is 1.25 and its integer code is 5. All fixed-point values in this section are signed and use two's-complement representation.

Consider the input vector:

$$x = [1.25, -0.75, 1.50, 0.50],$$

and the weight vectors:

$$w_0 = [0.50, 0.50, 0.25, 1.50] \qquad w_1 = [1.50, -0.25, 0.50, 0.25]$$

All elements of x, w_0 and w_1 are represented using a 4-bit signed fixed-point format with 2 integer bits (including the sign bit) and 2 fractional bits. We refer to this format as "**input format**".

Define inner products as:

$$y_i = x \cdot w_i, \quad i = 0,1$$

Each inner product y_i is initially stored using an 8-bit signed fixed-point format with 4 integer bits (including the sign bit) and 4 fractional bits. We refer to this format as "**accumulator format**".

7. How is the number **-0.75** represented as 4-bit binary with input format? Hint: Two's complement.

A. 4'b0000 B. 4'b0011 C. 4'b1001 D. 4'b1101

8. Which option correctly gives (1) the representable range of input format and (2) the quantization scale of the accumulator format?

A. Range of real number is [−2.00, +1.75]; the accumulator format has scale 0.0625

B. Range of real number is [−2.00, +2.00]; the accumulator format has scale 0.0625

C. Range of real number is [−1.75, +2.00]; the accumulator format has scale 0.125

D. Range of real number is [−2.00, +1.75]; the accumulator format has scale 0.125

Now we need to re-quantize y_i back to the input quantization format. Answer Question 9 and 10.

9. Assume the integer code of y_0 under accumulator format is 22 (Binary: `8'b00010110`). What are the re-quantization results given by truncation and RNE (round to the nearest even) policies?

A. Truncation: `4'b0101`, RNE: `4'b0110` B. Both give `4'b0101`

C. Truncation: `4'b0110`, RNE: `4'b0101` D. Both give `4'b0110`

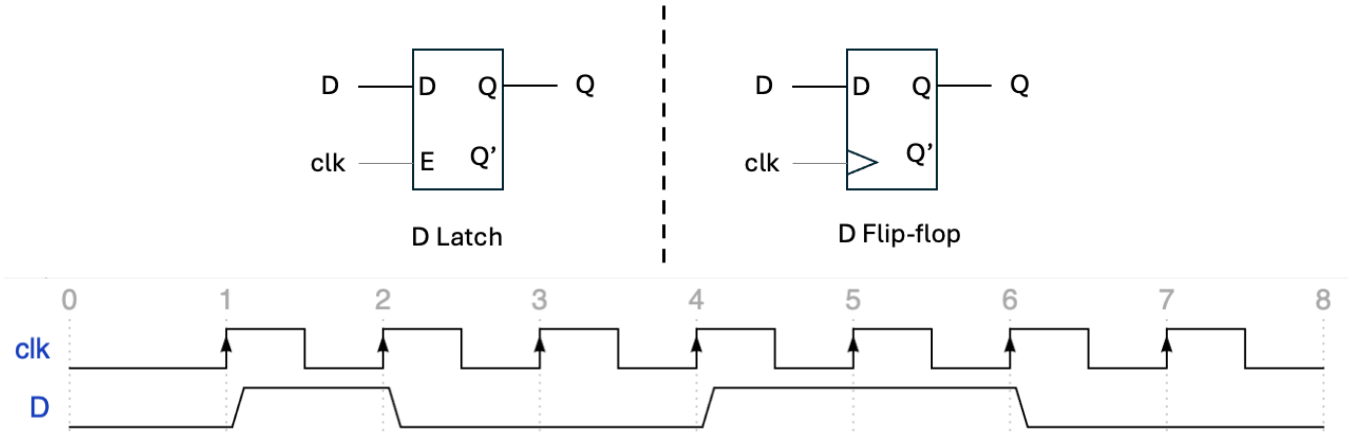
10. Assume the integer code of y_1 under accumulator format is 47 (Binary: `8'b00101111`). After applying RNE to the fractional bits, to handle overflow, which pair of results is produced by clamping and wrap-around overflow?

A. Clamp: `4'b0111`, Wrap: `4'b1011` B. Clamp: `4'b1100`, Wrap: `4'b0111`

C. Clamp: `4'b0111`, Wrap: `4'b0111` D. Clamp: `4'b0111`, Wrap: `4'b1100`

========== Section D: Latch and Flip-flop ==========

We apply the following waveform to the D-latch and D flip-flop. The clock signal is connected to the E port of D-latch. Assume the input D is registered, and it will only change right after a rising clock edge. The latch is transparent while clk is **high**, and the flip-flop captures D on the **rising** clock edge. Compare the different behavior of D-latch and D flip flop under the same input sequence in questions 11 and 12.



Find the values of Q right before the falling clock edge at **t = 1.5, 3.5, 5.5,** and **7.5**.

11. What are the values of Q for the D **latch**?

A. $Q = 0, 0, 1, 0$ B. $Q = 1, 0, 1, 0$ C. $Q = 1, 1, 1, 0$ D. $Q = 1, 0, 1, 1$

12. What are the values of Q for the D **flip flop**?

A. $Q = 0, 1, 1, 1$ B. $Q = 1, 0, 1, 0$ C. $Q = 0, 0, 0, 1$ D. $Q = 0, 0, 1, 0$

Below is the code for a simple shift register from the lecture. Assume `q` starts as `4'b0000`, and for the next 5 rising `clk` edges, `rstn` is high, and the input `s_in` is `1, 0, 1, 1, 1` before each rising `clk` edge.

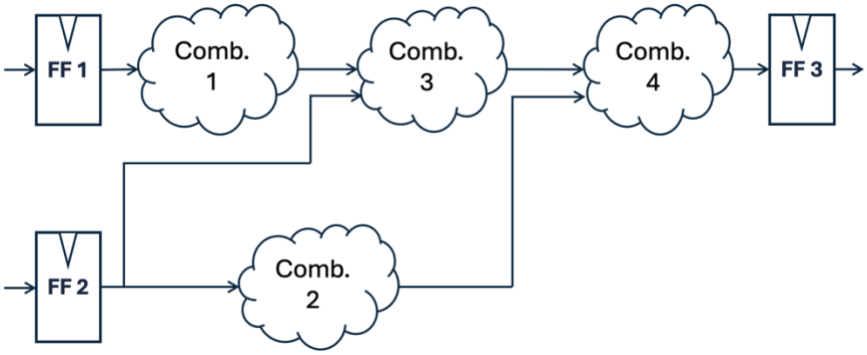
```
module shift_register (
   input  logic       clk, rstn, s_in,
   output logic [3:0] q
);
  always_ff @(posedge clk) begin
    if (!rstn) q <= '0;
    else     q <= {q[2:0], s_in};
  end
endmodule
```

13. What is the value of `q` right after it settles after the 5th rising clock edge?

A. `4'b0111` B. `4'b1011` C. `4'b1110` D. `4'b0001`

========== Section E: Timing Analysis ==========

Consider the following digital circuit. All flip-flops share a single clock. Assume each combinational block's propagation delay is the same from either input to its output. The propagation delay (t_{pd}) and contamination delay (t_{cd}) of the combinational circuits are given in the table.



Block	t_{pd}	t_{cd}
Comb 1	0.55 ns	0.25 ns
Comb 2	1.15 ns	0.40 ns
Comb 3	0.45 ns	0.10 ns
Comb 4	0.80 ns	0.30 ns

Both FF1 and FF2 have the maximum clock-to-Q delay of: $t_{pcq} = 0.30$ ns, and the minimum clock-to-Q delay of $t_{ccq} = 0.05$ ns. FF3 has the following requirements: $t_{\text{setup}} = 0.60$ ns, $t_{\text{hold}} = 0.50$ ns.
Ignore the clock skew, wire delay, setup and hold uncertainty. Answer Questions 14 to 19.

14. What is the combinational path with the maximum propagation delay? Select **all** blocks on the path.
A. Comb. 1 B. Comb. 2 C. Comb. 3 D. Comb. 4

15. What is the combinational path with minimum contamination delay? Select **all** blocks on the path.
A. Comb. 1 B. Comb. 2 C. Comb. 3 D. Comb. 4

16. If the target clock period is 2.90 ns, what is the worst setup slack? Is there setup violation?
A. 0.65 ns; No B. 0.20 ns; No C. 0.05 ns; No D. -0.95 ns; Yes

17. What is the worst hold slack? Is there hold violation? Hold violation is independent of the clock period.
A. -0.10 ns; Yes B. -0.05 ns; Yes C. 0.20 ns; No D. 0.25 ns; No

18. Suppose we have a hold time violation. Among the 4 options, where should we insert buffers? A buffer does not change the value of the input but adds delay.
A. Between FF1 and Comb 1. B. Between Comb 1 and Comb 3
C. Between FF2 and Comb 3, after the fanout node. D. Between Comb 2 and Comb 4

19. What should the minimum contamination delay of the buffer be so that we can have a positive hold slack of 0.10 ns?
A. 0.05 ns B. 0.15 ns C. 0.20 ns D. 0.55 ns

========== Section F: Finite State Machine ==========

Part 1: Sequence Detector

You will design a Moore finite state machine that watches a serial input `X`. One bit of `X` arrives at every rising clock edge. The output `Y` is asserted (`Y = 1`) whenever the last five bits received are `5'b10101`, where the leftmost bit of the pattern is the oldest bit and the rightmost bit is the most recent bit.

Detection is **overlapping** after a match is found. The machine does **not** restart from scratch. Bits that are already part of the pattern just found may be reused as the beginning of the next match.

The machine starts in the reset state `S0`. Use the following state naming convention for every question: `Sk` means the last `k` bits received match the first `k` bits of the pattern `10101` and no longer prefix matches.

Examples:
`S0` = nothing matched,
`S1` = 1 matched,
`S2` = `10` matched,
`S3` = `101` matched,
`S4` = `1010` matched,
and `S5` = the full pattern `10101` matched, output asserted as high.

Answer questions 20 - 24

20. Assume the input sequence is (old to new): `0101010110101` (one input per cycle). Then how many times the output is asserted as high? Hint: remember, the detection is overlapping.
A. 1 B. 2 C. 3 D. 4

21. Assuming the current state is `S0`, what should the input be, so that the next state is `S1`?
A. 0 B. 1

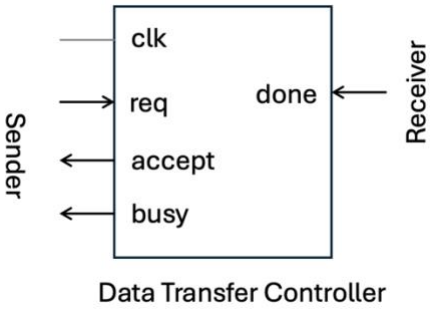
22. Assume the current state is S3, and the input is 1, what should the next state be?
A. `S0` B. `S1` C. `S4` D. `S5`

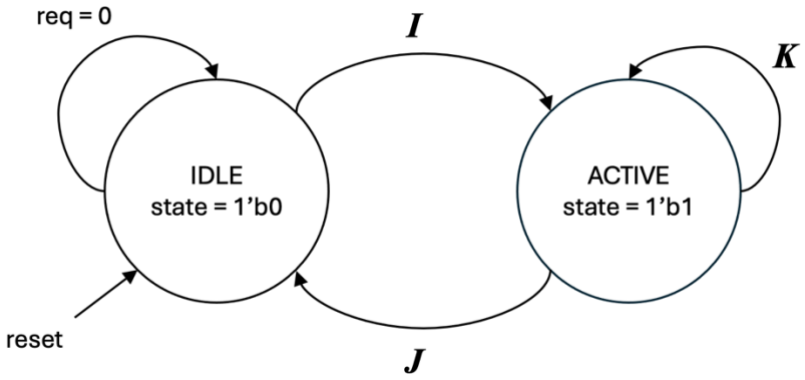
23. Assume the current state is S5, and the input is 0, what should the next state be?
A. `S0` B. `S2` C. `S4` D. `S5`

24. Assume the current state is S4, and the input is 0, what should the next state be?
A. `S0` B. `S1` C. `S2` D. `S5`

Part 2: Data Transfer Controller (questions 25 to 30)

Consider a data transfer controller sitting between a sender and a receiver. This is a simplified model: we assume the receiver is always available, and we omit the data channels.





The controller has 2 states held in a flip flop.

- `IDLE`: no transfer is currently active
- `ACTIVE`: a transfer is currently in progress

The Inputs (both are synchronous to clk):

- `req = 1`: (from sender) requests that a new transfer begin.
- `done = 1`: (from receiver) signals that the active transfer is completed.

The Outputs:

- `accept = 1`: (to sender) indicates that the current request is accepted.
- `busy = 1`: (to sender) notifies an active transfer is currently in progress.

Required behavior:

1. `busy = 0` while the controller is `IDLE`; `busy = 1` when state is `ACTIVE`.
2. The controller stays in `IDLE` until `req = 1`. On that rising edge its next state is `ACTIVE`.
3. The controller stays in `ACTIVE` until `done = 1`; on that rising edge its next state is `IDLE`.
4. The controller asserts `accept = 1` only when it is in `IDLE` and `req = 1`, and it does so as soon as `req = 1`. Therefore, at the next rising clock edge the sender sees `accept = 1` while it is still driving `req = 1`. This is the handshake telling both sides the request succeeded.
5. `accept = 0` whenever the controller is in `ACTIVE`.

25. Should the output accept and busy be implemented as a Mealy output or a Moore output?
A. Mealy, Moore B. Moore, Moore C. Moore, Mealy D. Mealy, Mealy

Complete the state transition conditions **I**, **J** and **K**. For example, the transition condition from IDLE to IDLE is `req == 0`, and the value of done is don't care, thus it's not listed.

26. Find state transition condition **I**.
A. `done==1` B. `req==1` C. `req==1 and done==0` D. `req==1 and done==1`

27. Find state transition condition **J**.
A. `req==0` B. `done==1 and req==1` C. `done==1 and req==0` D. `done == 1`

28. Find state transition condition **K**.
A. `done==0` B. `done==0 and req==0` C. `busy == 1` D. `req == 1`

Under the current specification, when a transfer finishes while the sender is already waiting with `req = 1` for its next transfer, the controller must return to `IDLE` before it can accept that waiting request. One clock cycle is wasted on every pair of back-to-back transfers.

We want to optimize the spec and the state machine so that the controller can accept a waiting request as early as possible. Suppose on a rising clock edge both `req` and `done` are high, this indicates a new transfer is available while the current transfer is already finished. Under this condition we should accept this new transaction and stay in `ACTIVE` instead of rejecting it and return to `IDLE`.

To do this, we need to modify the conditions of **J**, **K** and the outputs. Answer questions 29 and 30.

29. Find **K** after our optimization. Hint: **K** must cover 2 cases (finished and unfinished transaction), what happens when `done == 0`?

A. `done == 0 and req == 0`
B. `done == 1 and req == 1`
C. `req == 1`
D. `done == 0 or req == 1`

30. What should the output `accept` be after our optimization? "·" means "and", and "+" means "or".

A. `accept = (req == 1) · (state == IDLE)`
B. `accept = (req == 1) · (done == 1);`
C. `accept = (req == 1);`
D. `accept = (req == 1) · ((state == IDLE) + (done == 1));`

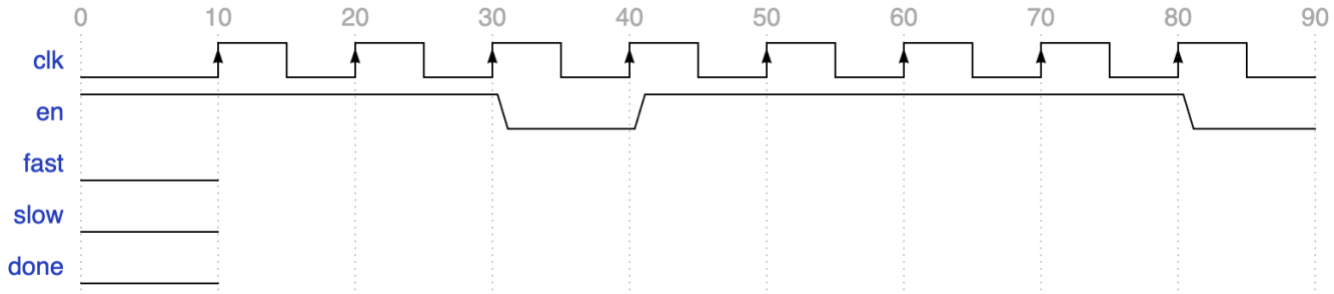
========== Section G: SystemVerilog ==========

Assume signals contain only `0` and `1` unless a question explicitly concerns an uninitialized value. Values at a stated time are taken after all events at that time have settled. Multi-bit waveform values are hexadecimal.

For Questions 31–37, consider this nested counter:

```
logic [1:0] fast, slow;
logic       done;
always_ff @(posedge clk or negedge rstn)
  if (!rstn) {fast, slow, done} <= '0;
  else begin
    done <= 1'b0;
    if (en) begin
      if (fast == 2) begin
        fast <= '0;
        if (slow == 1) begin
          slow <= '0;
          done <= 1'b1;
        end else begin
          slow <= slow + 1'b1;
        end
      end else begin
        fast <= fast + 1'b1;
      end
    end
  end
```

Asynchronous reset has completed before the first active edge. The first post-reset rising edge is at 10 ns, and rising edges occur every 10 ns.



31. Which statement correctly describes the behavior of `done`?
A. Once asserted, `done` stays high until reset.
B. `done=1` only during the clock cycle after `en && fast==2 && slow==1`; otherwise, `done=0`.
C. `done` toggles whenever `fast==2`, regardless of `en` and `slow`.
D. `done` changes immediately when `fast` or `slow` changes between clock edges.

32. Under what condition does `slow` increment?
A. `en && fast==2 && slow!=1` B. `en && fast!=2` C. `!en && fast==2` D. `en && slow==1`

33. Under what condition does `fast` increment?
A. `en && fast==2` B. `en && slow!=1` C. `!en && fast!=2` D. `en && fast!=2`

34. What is `(fast, slow, done)` at 25 ns?
A. `(2,0,0)` B. `(0,1,0)` C. `(2,1,0)` D. `(0,0,1)`

35. What is `(fast, slow, done)` at 75 ns?

A. `(2,1,0)` B. `(0,1,1)` C. `(1,0,1)` D. `(0,0,1)`

36. If `en` = high and clk period is 10 ns, how much time passes between consecutive `done=1` pulses?

A. 20 ns B. 30 ns C. 60 ns D. 80 ns

37. A student moves `done <= 1'b0;` inside the `if (en)` block. What can happen if `en` becomes zero immediately after `done` is asserted?

A. `done` remains high instead of producing a one-cycle pulse.
B. `fast` increments twice.
C. `slow` can never increment.
D. Reset becomes active high.

For Questions 38–40, consider this maximum-finding tree with a register after every three reduction levels and after the final level. Assume `N` is always a power of two.
For this example, **`N=64, DEPTH=$clog2(N)=6`**.

```
localparam DEPTH = $clog2(N);
for (stage = 0; stage < DEPTH; stage++) begin : g_stage
  for (node = 0; node < N >> (stage + 1); node++) begin : g_node
    logic [W-1:0] larger, left, right;
    always_comb begin
      left = tree[stage][2*node]; right = tree[stage][2*node+1];
      larger = left > right ? left : right;
    end
    if (((stage + 1) % 3 == 0) || (stage + 1 == DEPTH)) begin
      always_ff @(posedge clk or negedge rstn)
        if (!rstn)    tree[stage+1][node] <= '0;
        else if (cen) tree[stage+1][node] <= larger;
    end else begin
      always_comb tree[stage+1][node] = larger;
    end
  end
end
```

38. At a node, `left=8'hFE=(-2)` and `right=8'h03=(3)`. What change fixes the comparison?

A. Replace `>` with `>=`
B. Compare `$signed(left) > $signed(right)`.
C. Compare only bit 0 of each value
D. Cast both operands with `$unsigned`.

39. When `cen=1` continuously, what are the latency and steady-state throughput of this tree?

A. Latency 1 cycle; one result every cycle
B. Latency 6 cycles; one result every 6 cycles
C. Latency 2 cycles; one result every 2 cycles
D. Latency 2 cycles; one result every cycle

40. A registered `m_valid` is added alongside `y`. Which expression for the shared clock enable makes the output behave like an AXI-Stream ready/valid interface by freezing the tree only while a valid output is blocked?

A. `cen = m_valid && !m_ready;`
B. `cen = m_valid || m_ready;`
C. `cen = !(m_valid && !m_ready);`
D. `cen = m_ready;`

Use this student's implementation of `y = min(x*x + a, MAX)` for questions 41–45.

```
01 module square_clamp #(parameter W=4, MAX=20)(
02   input  logic clk, rstn,
03   input  logic signed [W-1:0] x, a,
04   output logic signed [2*W:0] y
05 );
06   logic signed [2*W-1:0] square;
07   logic signed [2*W:0]   sum;
08   always_ff @(posedge clk or negedge rstn)
09     if (!rstn) square = '0;
10     else       square <= $signed(x) * $signed(x);
11   always_comb begin
12     sum <= $signed(square) + $signed(a);
13     if (sum > MAX) y = MAX;
14   end
15 endmodule
```

41. Which lines use assignment operators inconsistent with the RTL coding conventions for `always_ff` and `always_comb`?

A. Line 9 only B. Line 12 only C. Lines 9 and 12 D. Neither line

42. What is the pipeline-alignment bug?

A. x should be delayed and squared twice. B. a of this input is added to previous input's square.
C. a is wider than `sum`. D. a cannot be used in combinational logic.

43. Which change fixes the pipeline-alignment bug?

A. Clear a in `always_comb`. B. Replace a with x.
C. Update a on the falling clock edge. D. Register `a_q <= a,` then add `a_q` to `square`.

44. What is wrong with line 13?

A. It should use <= instead of =. B. It uses a procedural `if` instead of a generate `if`.
C. The missing `else` can infer a latch. D. It uses a generate `if` instead of a procedural `if`.

45. What should be added after line 13 to complete the upper clamp?

A. `else y = MAX;` B. `else y = '0;` C. `else sum = y;` D. `else y = sum;`

46. Why is UART called asynchronous while AXI-Stream-style ready/valid communication is synchronous?

A. UART has no data signal, while AXI has no control signals.
B. UART endpoints do not share a clock and use an agreed baud rate and framing; ready/valid signals are sampled using a shared clock.
C. UART is always faster than AXI.
D. AXI does not use rising clock edges.

47. Which pairing best matches the communication links discussed in class?

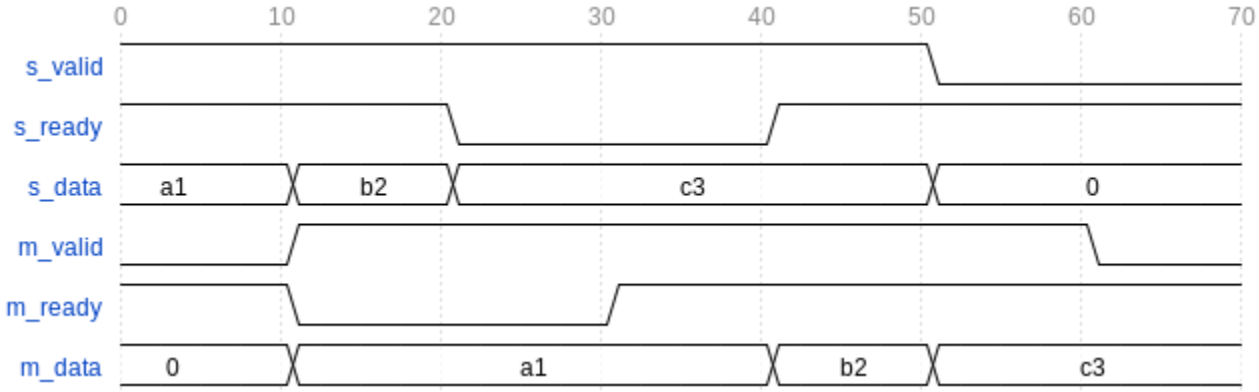
A. CPU-GPU IP: UART; microcontroller-PC: AXI B. Both links must use UART.
C. CPU-GPU IP: AXI; microcontroller-PC: UART D. Both links must use AXI.

========== Section H: Ready/Valid Flow Control ==========

Use this skid-buffer state-transition logic for Questions 48–51:

```
enum logic [1:0] {EMPTY=0, PARTIAL=1, FULL=2} state, next;
always_comb begin
  next = state;
  unique case (state)
    EMPTY  : if (s_valid)                    next = PARTIAL;
    PARTIAL: if      (!m_ready && s_valid) next = FULL;
             else if (m_ready && !s_valid) next = EMPTY;
    FULL   : if (m_ready)                    next = PARTIAL;
  endcase
end
always_ff @(posedge clk) state <= !rstn ? EMPTY : next;
```

Reset has completed before the first rising edge at 10 ns. Each numbered time is a rising edge. Transfers use signal values immediately before the edge; registered outputs may change just after it. `PARTIAL` means only the output is occupied; `FULL` means the output and skid register are occupied.



48. At which rising edges are input items accepted?
A. 10 ns only B. 10 ns and 50 ns C. 10 ns, 20 ns, and 50 ns D. 10 ns, 20 ns, 40 ns, and 50 ns

49. At which rising edges is the output under backpressure (`m_valid && !m_ready`)?
A. 10 ns and 20 ns B. 20 ns and 30 ns C. 30 ns and 40 ns D. 40 ns and 50 ns

50. At the 20 ns edge, which item is captured in the skid register, and what is the combinational value of `next` immediately before the edge?
A. `A1`; `next=PARTIAL` B. `B2`; `next=FULL` C. `C3`; `next=FULL` D. No item; `next=EMPTY`

51. Which expression identifies an output stall?
A. `m_valid&&!m_ready` B. `m_valid&&m_ready` C. `s_valid&&s_ready` D. `!m_valid&&!m_ready`

========== Section I: CPU ==========

Use this shortened version of the course CPU for Questions 52–55.

Bits	[15:8]	[7:4]	[3:0]
Meaning	addr	i_reg	opcode

```
enum logic [3:0] {LOAD=0, STORE=1, JNZ=6} opcode;
logic [7:0]  pc, addr;
logic [3:0]  i_reg;
logic [15:0] regs [16];
always_comb begin
  imem_addr = pc;
  {addr, i_reg, opcode} = imem_rdata;
  dmem_addr  = addr;
  dmem_wdata = regs[i_reg];
  dmem_wen   = opcode == STORE;
end
always_ff @(posedge clk)
  if (reset) begin
    pc <= '0;
    for (int i=0; i<16; i++) regs[i] <= '0;
  end else begin
    pc <= pc + 1'b1;
    case (opcode)
      LOAD: regs[i_reg] <= dmem_rdata;
      JNZ : if (regs[i_reg] != '0) pc <= addr;
    endcase
  end
```

52. What does instruction `16'h1230` do?

A. Store `r3` into data-mem address `8'h12`.
B. Load `8'h12` directly into `r3`.
C. Jump to instruction address `8'h12`.
D. Load data from data-mem address `8'h12` into `r3`.

53. What hardware implements `dmem_wdata = regs[i_reg]` for sixteen 16-bit registers?

A. A 4-to-16 demultiplexer
B. A 16-to-4 priority encoder
C. A 16-to-1 multiplexer with a 16-bit data path
D. A 16-bit counter

54. What hardware converts `i_reg` and `opcode==LOAD` into sixteen individual register write enables?

A. A 16-to-1 multiplexer
B. A 4-to-16 decoder followed by write-enable gating
C. A barrel shifter
D. A full adder

55. Just before a rising edge, `pc=8'h05`, `opcode=JNZ`, `addr=8'h20`, and `regs[i_reg]=1`. What is `pc` after the edge?

A. `8'h05`
B. `8'h06`
C. `8'h20`
D. `8'h21`