

2 Verilog [30 points]

Please answer the following four questions about Verilog.

- (a) [6 points] Does the following code result in a sequential circuit or a combinational circuit? Explain why.

```

1 module concat (input clk, input data_in1, input data_in2,
2                 output reg [1:0] data_out);
3     always @ (posedge clk, data_in1)
4         if (data_in1)
5             data_out = {data_in1, data_in2};
6         else if (data_in2)
7             data_out = {data_in2, data_in1};
8 endmodule

```

Answer and concise explanation:

Sequential circuit.

Explanation.

This code results in a sequential circuit because `data_in2` is *not* in the sensitivity list, and thus a latch is inferred for `data_out`.

- (b) [6 points] In the following code, the input `clk` is a clock signal. What is the hexadecimal value of the output `c` right after the third positive edge of `clk` if initially `c = 8'hE3` and `a = 4'd8` and `b = 4'o2` during the entire time?

```

1 module mod1 (input clk, input [3:0] a, input [3:0] b, output reg [7:0] c);
2     always @ (posedge clk)
3         begin
4             c <= {c, &a, |b};
5             c[0] <= ^c[7:6];
6         end
7 endmodule

```

Please answer below. Show your work.

8'hC4.

Explanation.

Cycle 1: $c \leftarrow \{c, \&a, |b\} \rightarrow c \leftarrow \{1110_0011, 0, 1\} \rightarrow c \leftarrow \{1000_1101\}$

$c[0] \leftarrow \wedge c[7:6] \rightarrow c[0] \leftarrow \wedge \{11\} \rightarrow c[0] \leftarrow 0$

At the first positive edge of *clk*, $c = 8'b1000_1100$

Cycle 2: $c \leftarrow \{c, \&a, |b\} \rightarrow c \leftarrow \{1000_1100, 0, 1\} \rightarrow c \leftarrow \{0011_0001\}$

$c[0] \leftarrow \wedge c[7:6] \rightarrow c[0] \leftarrow \wedge \{10\} \rightarrow c[0] \leftarrow 1$

At the second positive edge of *clk*, $c = 8'b0011_0001$

Cycle 3: $c \leftarrow \{c, \&a, |b\} \rightarrow c \leftarrow \{0011_0001, 0, 1\} \rightarrow c \leftarrow \{1100_0101\}$

$c[0] \leftarrow \wedge c[7:6] \rightarrow c[0] \leftarrow \wedge \{00\} \rightarrow c[0] \leftarrow 0$

At the third positive edge of *clk*, $c = 8'b1100_0100 \rightarrow c = 8'hC4$

Note that since the assignments to *c* are non-blocking, $c[7:6]$ in line 5 is not affected by the assignment to *c* in line 4 in the same cycle.

- (c) [6 points] Is the following code syntactically correct? If not, please explain the mistake(s) and how to fix it/them.

```
1 module lnn3r ( input [3:0] d, input op, output [1:0] s );
2   assign s = op ? (d[1:0] - d[3:2]) :
3                 (d[3:2] + d[1:0]);
4 endmodule
5
6 module top ( input wire [6:0] instr, input wire op, output reg z );
7
8   reg [1:0] r1, r2;
9
10  lnn3r i0 (.instr(instr[1:0]), .op(instr[7]), .z(r1) );
11  lnn3r i1 (.instr(instr[3:2]), .op(instr[0]), .z(r2) );
12  assign z = r1 | r2;
13
14 endmodule
```

Answer and concise explanation:

The code is not syntactically correct.

Explanation.

- Module names cannot start with a number → 'lnn3r' is not a legal module name.
- The output signal 'z' has to be declared as a 'wire' but not 'reg'.
- 'r1' and 'r2' has to be declared as 'wire's.
- The module 'lnn3r' does not have ports named 'instr' and 'z'. Those need to be changed to 'd' and 's', respectively.

- (d) [6 points] Does the following code correctly implement a counter that counts from 1 to 11 by increments of 2 (e.g., 1, 3, 5, 7, 9, 11, 1, 3, ...)? If so, say "Correct". If not, correct the code with minimal modification.

```
1 module odd_counter (clk, count);
2     wire clk;
3     reg[2:0] count;
4     reg[2:0] count_next;
5
6     always@*
7     begin
8         count_next = count;
9         if(count != 11)
10            count_next = count_next + 2;
11        else
12            count_next <= 1;
13    end
14
15    always@(posedge clk)
16        count <= count_next;
17 endmodule
```

Answer and concise explanation:

No, the implementation is not correct.

Explanation.

The correct implementation:

```
module odd_counter (clk, count);
    wire clk;
    reg[3:0] count = 1;
    reg[3:0] count_next;

    always@* begin
        count_next = count;
        if(count != 11)
            count_next += 2;
        else
            count_next = 1;
    end

    always@(posedge clk)
        count <= count_next;
endmodule
```

- (e) [6 points] Does the following code correctly instantiate a 4-bit adder? If so, say "Correct". If not, correct the code with minimal modification.

```
1 module adder(input a, input b, input c, output sum, output carry);
2   assign sum = a ^ b ^ c;
3   assign carry = (a&b) | (b&c) | (c&a);
4   endmodule
5
6
7 module adder_4bits(input [3:0] a, input [3:0] b, output [3:0] sum, carry);
8   wire [2:0] s;
9
10  adder u0 (a[0],b[0],1'b0,sum[0],s[0]);
11  adder u1 (a[1],s[0],b[1],sum[1],s[1]);
12  adder u2 (a[2],s[1],b[2],sum[2],s[2]);
13  adder u3 (a[3],s[2],b[3],sum[3],carry);
14  endmodule
```

Yes.

Explanation: Even though the wire *s* is swapped with the input *b*, the final computation produced by the module *adder* is still going to be correct since the *or* and *and* operations are commutative.