

## 2 Verilog [60 points]

### 2.1 Complete the Verilog code [30 points]

For each numbered blank ①–⑤ in the following Verilog code, **mark the choice below** (i.e., one of options A, B, C, D) that makes the Verilog module operate as described in the comments. The resulting code must have correct syntax.

```

1 module my_module (input clk, input rst,
2   input[15:0] idata, input[1:0] op, ①[31:0] odata);
3
4 ② nval = 32'd0; // defining a 32-bit signal with an initial value of 0
5
6 always@* begin
7   case (op)
8     2'b00:
9       nval = odata + idata; // when 'op' is decimal 0, add 'idata' to
10                          // 'odata' and assign the result to 'nval'
11     2'b01:
12       nval = odata - idata; // when 'op' is decimal 1, subtract 'idata'
13                          // from 'odata' and assign the result to 'nval'
14     2'b10:
15       nval = idata; // when 'op' is decimal 2, assign 'idata' to 'nval'
16     ③:
17       nval = 0; // when 'op' is decimal 3, assign 0 to 'nval'
18   endcase
19 end
20
21 // executing the following always block on the rising edge of 'clk'
22 always@ (posedge clk) begin
23   if (rst)
24     ④ // resetting 'odata' to 0 for the next cycle
25   else
26     ⑤ // assigning 'nval' to 'odata' for the next cycle
27   end
28 endmodule

```

Provide your choice for each blank ①–⑤ below:

- |    |                          |                         |                   |                   |
|----|--------------------------|-------------------------|-------------------|-------------------|
| ①: | A. output                | B. output reg           | C. output wire    | D. input reg      |
| ②: | A. reg[31:0]             | B. input[31:0]          | C. wire[31:0]     | D. int[31:0]      |
| ③: | A. 2'b3                  | B. 3'b3                 | C. 2'h11          | D. default        |
| ④: | A. assign odata <= 0;    | B. assign odata = 0;    | C. odata == 0;    | D. odata <= 0;    |
| ⑤: | A. assign odata <= nval; | B. assign odata = nval; | C. odata == nval; | D. odata <= nval; |

**Explanation.**

- ①: `odata` must be declared as an output signal since values are assigned to it in the second `always` block. It cannot be an input signal since inputs are read-only signals and no assignments are allowed to them. `odata` must be also declared as `reg` since the assignments are made inside an `always` block.
- ②: `nval` must be declared as `reg[31:0]` since values are assigned to it inside the first `always` block.
- ③: `default` is a correct choice since all other cases for a 2-bit values (i.e., `2'b00`, `2'b01`, and `2'b10`) are defined in the `case` statement. The other choices are not correct since they do not properly specify the value of 3. For example, in `2'b3`, the problem is that 3 is not a valid binary digit but `2'b` must be followed by a 2-bit binary value.
- ④: Choices with `assign` are not valid since the `assign` keyword cannot be used in an `always` block. Choice C does not specify an assignment operator but an equality comparison, hence it is not a valid choice either. The correct choice is D, which assigns 0 to `odata` using non-blocking assignment operator.
- ⑤: The correct choice is D due to the same reasons as in ④.

## 2.2 What Does This Code Do? [30 points]

You are given a Verilog code that you are asked to analyze and find out what it does.

```
1  module my_module2 (input clk, output[1:0] out);
2
3      reg state = 1'b0;
4      reg[1:0] my_reg = 0;
5
6      always@(posedge clk) begin
7          state <= &out ? ~state : state;
8      end
9
10     always@(posedge clk) begin
11         case(state)
12             1'b0: begin
13                 my_reg <= my_reg + 1;
14             end
15             1'b1: begin
16                 my_reg <= my_reg - 1;
17             end
18         endcase
19     end
20
21     assign out = my_reg;
22 endmodule
```

Show the values (as unsigned decimal numbers) that the out signal takes, starting from the initial state of the module, for 16 consecutive clock (i.e., clk) cycles. Explain your answer briefly.

out is equal to 0, 1, 2, 3, 0, 3, 2, 3, 0, 3, 2, 3, 0, 3, 2, 3 in the first 16 clock cycles.

### Explanation.

The module either increments or decrements my\_reg depending on the state. When state is equal to 0, my\_reg is incremented by 1 and otherwise decremented by 1. The value of my\_reg is directly assigned to the out signal, and both signals are 2-bit wide.

my\_reg and state are both initially 0. Therefore, in subsequent cycles, my\_reg gets incremented until it reaches 3. During the next cycle, a new value for state is being computed (i.e., the inverse of state as ~state). However, since the new value of the state is not updated until the next positive edge of the clk, the second always block reads state as 0, and thus my\_reg gets incremented again to become 0 (the maximum value a 2-bit register can represent is 3 and incrementing my\_reg one more time makes it 0).

During the next cycle, state is 1 and my\_reg is decremented back to 3. Since my\_reg (and thus out) being 3 inverts state, state becomes 0 in the subsequent cycle and my\_reg becomes 2 during the positive edge of clk when state is inverted. Then, my\_reg gets incremented to 3 and 0 in the next consecutive cycles. Because state remains as 0 or 1 for two consecutive cycles and then gets inverted, the values of my\_reg forever repeat the sequence of (0, 3, 2, 3, 0, 3, 2, 3, ...).