Digital Design and Computer Architecture (252-0028-00L), Spring 2023

Optional HW 2: Sequential Logic and Verilog

Instructor: Prof. Onur Mutlu
TAs: Juan Gomez Luna, Mohammad Sadrosadati, Mohammed Alser, Ataberk Olgun, Giray Yaglikci, Can Firtina, Geraldo De Oliveira Junior, Rahul Bera, Konstantinos Kanellopoulos, Nika Mansouri Ghiasi, Nisa Bostancı, Rakesh Nadig, Joel Lindegger, İsmail Emir Yüksel, Haocong Luo, Yahya Can Tuğrul, Julien Eudine

Released: Wednesday, March 15, 2023

1 Verilog (I)

Please answer the following three questions about Verilog.

(a) Does the following code result in a D Flip-Flop with a synchronous active-low reset? Please explain your answer.

```
module mem (input clk, input reset, input [1:0] d, output reg [1:0] q);
always @ (posedge clk or negedge reset)
    begin
      if (!reset) q <= 0;
      else q <= d;
    end
endmodule
```

(b) Does the following code result in a sequential circuit or a combinational circuit? Please explain your answer.

```
module Mask (input [1:0] data_in, input mask, output reg [1:0] data_out);
always @ (*)
    begin
        data_out[1] = data_in[1];
        if (mask)
            data_out[0] = 0;
    end
endmodule
```

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

```
module fulladd(input a, b, c, output reg s, c_out);
   assign s = a^b;
   assign c_out = (a & b) | (b & c) & (c & a);
endmodule

module top ( input wire [5:0] instr, input wire op, output z);

  reg[1:0] r1, r2;
  wire [3:0] w1, w2;

  fulladd FA1 (.a(instr[0]), .b(instr[1]), .c(instr[2]),
                             .c_out(r1[1]), .z(r1[0]));
  fulladd FA2 (.a(instr[3]), .b(instr[4]), .c(instr[5]),
                             .z(r2[0]), .c_out(r2[1]));

  assign z = r1 | op;
  assign w1 = r1 + 1;
  assign w2 = r2 << 1;
  assign op = r1 ^ r2;

endmodule
```

2 Verilog (II)

Please answer the following four questions about Verilog.

(a) Does the following code result in a D Flip-Flop with asynchronous reset? Please explain why.

```
module dff (input clk, input reset, input [3:0] d, output reg [3:0] q);
always @ (posedge clk)
    begin
      if (reset == 0) q <= 0;
      else q <= d;
    end
endmodule
```

(b) Does the following code result in a sequential circuit or a combinational circuit? Explain why.

```
module concat (input clk, input data_in1, input data_in2,
                                  output reg [1:0] data_out);
 always @ (posedge clk, data_in1, data_in2)
   if (data_in1 > data_in2)
      data_out = {data_in1, data_in2};
   else
      data_out = {data_in2, data_in1};
endmodule
```

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

```
module Inn3r ( input [3:0] d, input op, output s);
  assign s = op ? (d[1:0] - d[3:2]) :
                         (d[3:2] + d[1:0]);
endmodule

module top ( input wire [6:0] instr, input wire op, output reg z);

  reg[1:0] r1, r2, r3;
  wire [3:0] w1, w2;

  Inn3r i0 (.instr(instr[1:0]), .op(instr[6]), .z(r1) );
  Inn3r i1 (.instr(instr[3:2]), .op(instr[0]), .z(r2) );

  assign z = r1 | r2;
  assign w1 = r1 + 1;
  assign w2 = r2 << 1;

  top t (.instr({w1, w2, w1==w2}), .op(z), .z(r3));

  assign op = r1 ^ r2 ^ r3;

endmodule
```

(d) Does the following code correctly implement a counter that counts down from 10 to 1 (e.g., 10, 9, 8, ..., 2, 1, 10, 9, ...)? If so, say "Correct". If not, correct the code with minimal modification.

```
module the_final_count_down (clk, count);
  wire clk;
  reg[3:0] count = 10;
  reg[3:0] count_next;

  always @ * begin
    count_next <= count;
    if(count != 1)
      count_next <= count_next - 1;
    else
      count_next <= 1;
  end


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

Answer with concise explanation:

(e) Which of the combinational logic blocks does the following verilog code implement?

```
module mystery(input select, input enable, output result);
        wire [3:0] result;
        wire [1:0] select;
        wire enable;

        assign result = enable << (select);
endmodule
```

3 Verilog (III)

Please answer the following questions about Verilog.

3.1 Blocking vs. Non-Blocking Assignments

(a) What is the difference between a blocking and a non-blocking assignment?

3.2 Verilog Synthesis

For each circuit that results from the following code segments, select and write all applicable relevant words from the **word bank** below. If there are any syntactical or semantic issues in a code sequence, list them all in the code blocks' corresponding answer box.

word bank: *asynchronous, synchronous, active-low, active-high, reset, D Flip-Flop, sequential, combinational, inferred latch, trimmed signal, multiple drivers, race condition, tri-state logic*

(a) Code Block 1

```
module A (input clk, input rst, input [2:0] d, output wire [1:0] q)
  always @ (posedge clk or negedge rst) begin
        if (!rst) q <= 0;
        else q <= d;
  end
endmodule
```

(b) Code Block 2

```
module B (input clk, input rst, input [1:0] d, input m, output reg [1:0] q);
  always @ (*) begin
        q[1] = d[1];
        if (m)
          q[0] = 0;
  end
endmodule
```

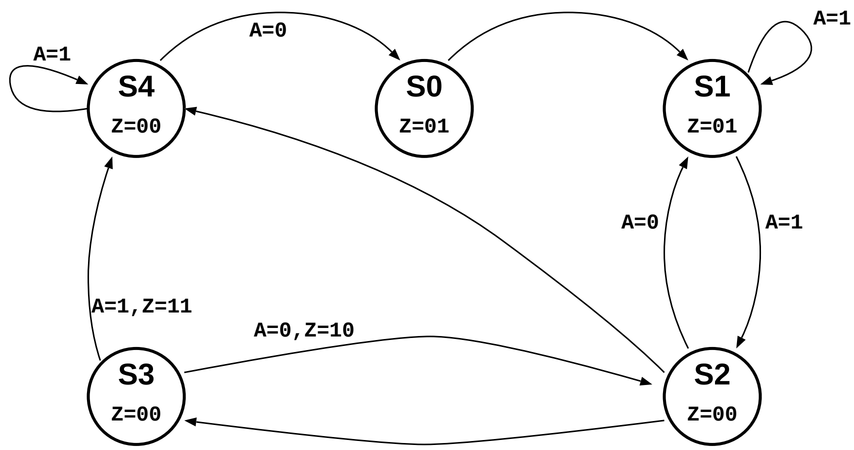
(c) Code Block 3

```
module C (input clk, input rst, input d, output reg q);
  always @ (posedge clk or negedge rst) begin
        if (!rst) q <= 0;
        else q <= d;
  end
endmodule
```

4 Finite State Machines (I)

This question has three parts.

(a) An engineer has designed a deterministic finite state machine with a one-bit input (A) and a two-bit output (Z). He started the design by drawing the following state transition diagram:



Although the exact functionality of the FSM is not known to you, there are **at least three mistakes** in this diagram. Please list **all** the mistakes.

(b) After learning from his mistakes, your colleague has proceeded to write the following Verilog code for a much better (and **different**) FSM. The code has been verified for syntax errors and found to be OK.

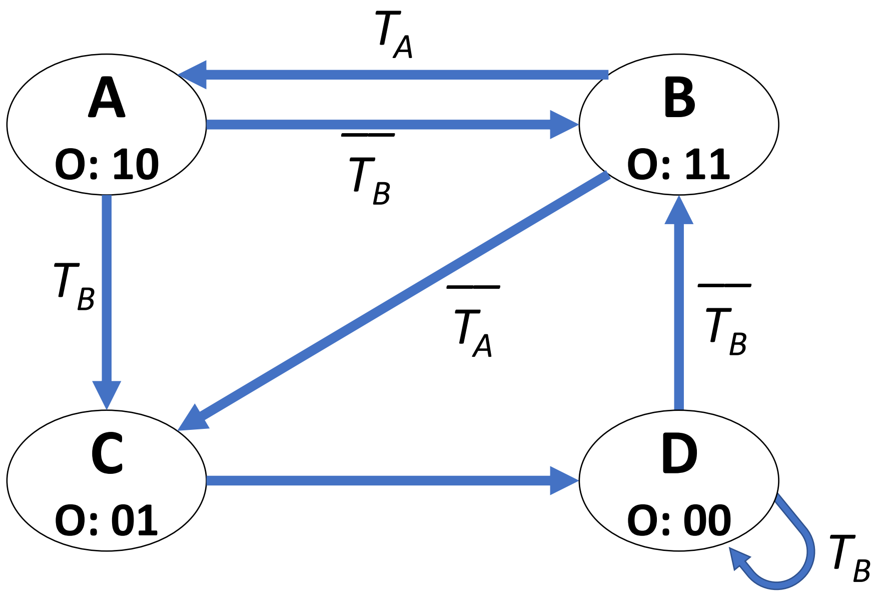
```
module fsm (input CLK, RST, A, output [1:0] Z);

  reg [2:0] nextState, presentState;

  parameter start   = 3'b000;
  parameter flash1 = 3'b010;
  parameter flash2 = 3'b011;
  parameter prepare   = 3'b100;
  parameter recovery = 3'b110;
  parameter error = 3'b111;

  always @ (posedge CLK, posedge RST)
     if (RST)  presentState <= start;
     else      presentState <= nextState;

  assign Z = (presentState == recovery) ? 2'b11 :
             (presentState == error)    ? 2'b11 :
             (presentState == flash1)   ? 2'b01 :
             (presentState == flash2)   ? 2'b10 : 2'b00;

  always @ (presentState, A)
    case (presentState)
      start    : nextState <= prepare;
      prepare  : if (A) nextState <= flash1;
      flash1   : if (A) nextState <= flash2;
                 else   nextState <= recovery;
      flash2   : if (A) nextState <= flash1;
                 else   nextState <= recovery;
      recovery : if (A) nextState <= prepare;
                 else   nextState <= error;
      error    : if (~A) nextState <=start;
      default  : nextState <= presentState;
    endcase

endmodule
```

Draw a proper state transition diagram that corresponds to the FSM described in this Verilog code.

(c) Is the FSM described by the previous Verilog code a Moore or a Mealy FSM? Why?

5 Finite State Machines (II)

You are given the following FSM with two one-bit input signals (T_A and T_B) and one two-bit output signal (O). You need to implement this FSM, but you are unsure about how you should encode the states. Answer the following questions to get a better sense of the FSM and how the three different types of state encoding we dicussed in the lecture (i.e., one-hot, binary, output) will affect the implementation.



(a) There is one critical component of an FSM that is *missing* in this diagram. Please write what is missing in the answer box below.

(b) What kind of an FSM is this?

(c) List one major advantage of each type of state encoding below.

- One-hot encoding
- Binary encoding
- Output encoding

(d) Fully describe the FSM with equations given that the states are encoded with **one-hot** encoding. Assign state encodings such that numerical values of states increase monotonically for states A through D while using the **minimum** possible number of bits to represent the states with one-hot encoding. Indicate the values you assign to each state and simplify all equations:

(e) Fully describe the FSM with equations given that the states are encoded with **binary** encoding. Assign state encodings such that numerical values of states increase monotonically for states A through D while using the **minimum** possible number of bits to represent the states with binary encoding. Indicate the values you assign to each state and simplify all equations:

(f) Fully describe the FSM with equations given that the states are encoded with **output** encoding. Use the **minimum** possible number of bits to represent the states with output encoding. Indicate the values you assign to each state and simplify all equations:

(g) Assume the following conditions:

- We can only implement our FSM with 2-input AND gates, 2-input OR gates, and D flip-flops.
- 2-input AND gates and 2-input OR gates occupy the *same* area.
- D flip-flops occupy 3x the area of 2-input AND gates.

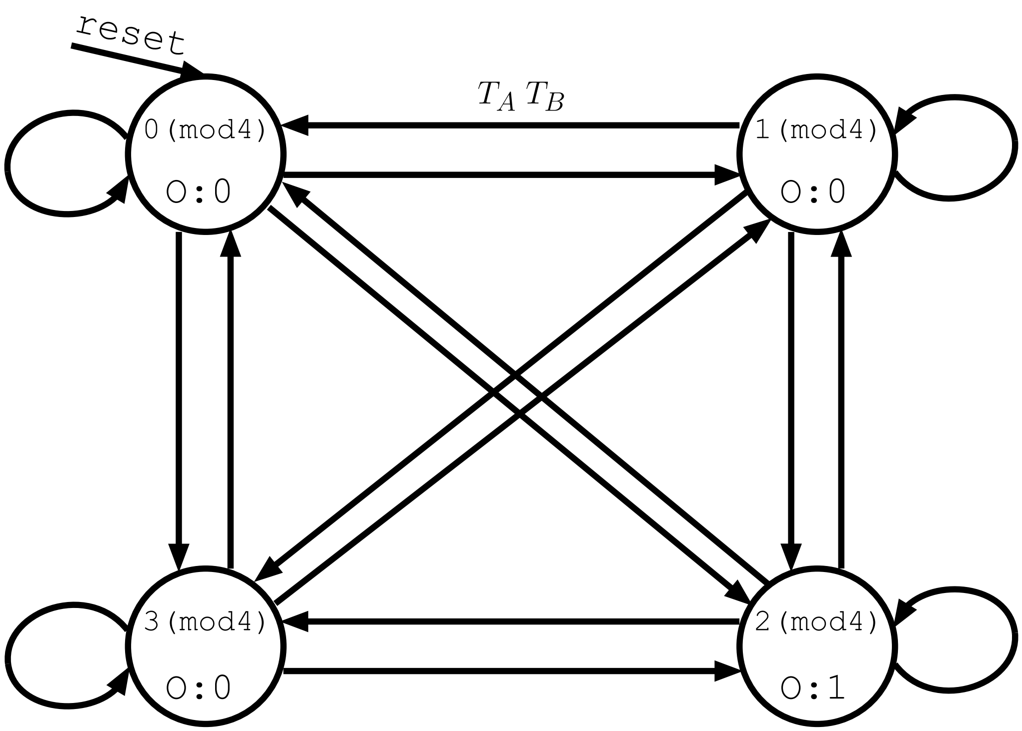
Which state-encoding do you choose to implement in order to minimize the total area of this FSM?

6 Finite State Machines (III)

You are given two *one-bit* input signals (T_A and T_B) and one *one-bit* output signal (O) for the following modular equation: $2N(T_A) + N(T_B) \equiv 2 \pmod 4$. In this modular equation, $N(T_A)$ and $N(T_B)$ represent the **total number of times** the inputs T_A and T_B are high (i.e., logic 1) at each positive clock edge, respectively. The one-bit output signal, O, is set to 1 when the modular equation is satisfied (i.e., $2N(T_A) + N(T_B) \equiv 2 \pmod 4$), and 0 otherwise. An example that sets $O = 1$ at the end of the fourth cycle would be:

- (1^{st} cycle) $T_A = 0$ ($N(T_A) = 0$), $T_B = 0$ ($N(T_B) = 0$), $2N(T_A) + N(T_B) \equiv 0 \pmod 4 \Rightarrow O = 0$
- (2^{nd} cycle) $T_A = 1$ ($N(T_A) = 1$), $T_B = 1$ ($N(T_B) = 1$), $2N(T_A) + N(T_B) \equiv 3 \pmod 4 \Rightarrow O = 0$
- (3^{rd} cycle) $T_A = 1$ ($N(T_A) = 2$), $T_B = 0$ ($N(T_B) = 1$), $2N(T_A) + N(T_B) \equiv 1 \pmod 4 \Rightarrow O = 0$
- (4^{th} cycle) $T_A = 0$ ($N(T_A) = 2$), $T_B = 1$ ($N(T_B) = 2$), $2N(T_A) + N(T_B) \equiv 2 \pmod 4 \Rightarrow O = 1$

(a) You are given a partial **Moore** machine state transition diagram that corresponds to the modular equation described above. However, the input labels of most of the transitions are still missing in this diagram. Please label the transitions with the correct inputs so that the FSM correctly implements the above specification.



(b) Describe the FSM with Boolean equations assuming that the states are encoded with **one-hot encoding**. Assign state encodings while using the **minimum** possible number of bits to represent the states. Please indicate the values you assign to each state.

(c) Describe the FSM with Boolean equations assuming that the states are encoded with **binary encoding** (i.e., fully encoding). Assign state encodings while using the **minimum** possible number of bits to represent the states. Please indicate the values you assign to each state.

(d) Consider an implementation of the FSM assuming that the states are encoded with **output encoding**. What is the **minimum** number of bits required to encode the states with output encoding?

7 Finite State Machines (IV)

7.1 Mealy Machine and Moore Machine

Figure 1 depicts a Mealy state machine corresponding to a digital circuit design that receives *one input* and produces *one output*. All state transitions in the diagram are labelled with the corresponding *input/output* values. Answer the following questions for this state diagram.

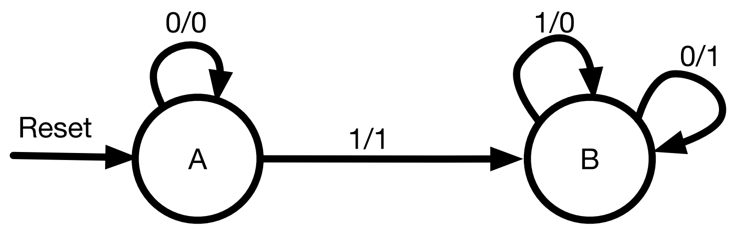


Figure 1: A Mealy Machine.

(a) Can this Mealy machine be converted to its equivalent Moore machine? If so, please convert it to its equivalent Moore machine. Draw the converted Moore machine. If not, explain why it is not possible.

(b) Assume the state machine in Figure 1 is used to process binary numbers, from their least significant bit to their most significant bit. You observe an output bit stream from this FSM, as shown in Figure 2.

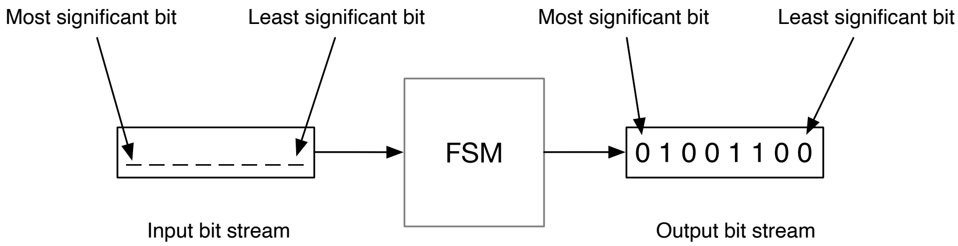


Figure 2: Usage of the Mealy machine to process a bit stream.

What was the input bit stream supplied to this FSM? Show your work.

7.2 Designing an FSM

Design a Moore finite state machine (FSM) with one input and one output. The input provides an unsigned binary number in a bit-serial manner, from the most-significant bit to the least-significant bit. The output should be logic-1 in a clock cycle if the provided input made up of all the bits received so far is divisible by 3 (i.e., [the input number] `mod` $3 = 0$). (Hint: Recall that the output depends only on the current state in a Moore FSM.)

Below are some example bit-streams that should output logic-1.

- 11
- 110
- 1001
- 1100
- 1111

Draw the state diagram and explain why it works. Your state machine should use as few states as possible and each state should have a comprehensive definition.