Exams/2013 q2bfsm

Consider a finite state machine that is used to control some type of motor. The FSM has inputs *x* and *y*, which come from the motor, and produces outputs *f* and *g*, which control the motor. There is also a clock input called *clk* and a reset input called *resetn*.

The FSM has to work as follows. As long as the reset input is asserted, the FSM stays in a beginning state, called state *A*. When the reset signal is de-asserted, then after the next clock edge the FSM has to set the output *f* to 1 for one clock cycle. **Then**, the FSM has to monitor the *x* input. When *x* has produced the values 1, 0, 1 in three successive clock cycles, then *g* should be set to 1 on the following clock cycle. While maintaining *g* = 1 the FSM has to monitor the *y* input. If *y* has the value 1 within at most two clock cycles, then the FSM should maintain *g* = 1 permanently (that is, until reset). But if *y* does not become 1 within two clock cycles, then the FSM should set *g* = 0 permanently (until reset).

(The original exam question asked for a state diagram only. But here, implement the FSM.)

```
module top_module (
    input clk,
    input resetn,    // active-low synchronous reset
    input x,
    input y,
    output f,
    output g
);
    parameter [3:0] A    = 4'd0; //reset state
    parameter [3:0] F    = 4'd1;
    parameter [3:0] X    = 4'd2; //monitor x

    parameter [3:0] F1   = 4'd3;
    parameter [3:0] F10  = 4'd4;

    parameter [3:0] F101 = 4'd5;//monitor y
    parameter [3:0] Y1   = 4'd6; //monitor y

    parameter [3:0] G1 = 4'd7;
    parameter [3:0] G0 = 4'd8;

    reg [3:0] curr_state;
    reg [3:0] next_state;
    always @ (posedge clk )begin
        if(~resetn)begin
            curr_state <= A;
        end
        else begin
            curr_state <= next_state;
        end
    end
```

```
  always @ (*) begin
     case(curr_state)
        A    : next_state = F;
        F    : next_state = X;
        X    : next_state =  x? F1  : X; //monitor x

        F1  : next_state = ~x? F10  : F1; //recieve 1 is the next is  0?
        F10: next_state =   x? F101:  X; //recieve 10 is the next is 1?

        F101:next_state =  y? G1  : Y1; //monitor y
        Y1    :next_state =  y? G1  : G0;//monitor y

        G1  :next_state = G1;
        G0  :next_state = G0;
        default: next_state = A;
     endcase
  end

  assign f = curr_state == F;

  assign g = (curr_state == G1)|(curr_state == F101)|(curr_state == Y1);

endmodule
```