Lab 5: Garage Occupancy Counter

Sign-Off Sheet

Student 1: Margaret Earnest **ECE mailbox:** _____

Student 2: Jonathan Lopez **ECE mailbox:** _____

YOU ARE RESPONSIBLE TO COMPLETE ALL THE ASSIGNMENTS IN THE CHECK LIST BELOW IN ORDER TO GET FULL CREDIT FOR THE LAB…

Check List	
Assignments	**TA Sign-off**
Pre-Lab (**MUST** be completed before the start of the lab) 1. Counter 2. FSM Car Entering/Exiting 3. Simulation Read Lab Write-up	
Lab part Create Source File for the following Modules: i. BCD7SEG ii. Slow Clock iii. 2-bit Counter iv. 4 to 1 Mux v. Decoder2-to-4 vi. BinarytoBCD vii. 8-bit-Counter viii. Debounce ix. FSM_Enter x. FSM_Exit xi. Top Module (put the system together)	
Project File(s) Upload on CANVAS (Import all files in one folder, zip it, upload it) 1. Upload Sign-off Sheet 2. Upload your project files (zip the folder), name your file as: lastname_labn_D'20 **Present your work** 3. Record a 2-3 minute video showing that all parts of your lab functioned properly. **OR** Show your work to TAs for the sign-off over Zoom. Upload the Writing Assignment (Debugging and Troubleshooting errors FAQ's) – OPTIONAL	

Due Date: 05/13/2020

Both Students MUST be present at Sign-off for any and all parts!!

Prelab 5: Garage Occupancy Counter

1. Objective

The objective of this lab is to design an occupancy counter to display how many cars are currently parked in the garage. Fig. 1 shows a typical garage entrance with sensors on both entrance and exit gates.

Figure 1. Garage entrance and exit with photo sensors

2. Detecting Vehicle Entering the Garage

The basic design is to use a pair of photo sensors. When an object is between the optical transmit and optical receiver, the light is blocked and the corresponding output is asserted to '1'. Otherwise, the output is '0'. As shown in Fig. 1, we can determine if a car has entered the garage by monitoring the events (orders) of these 2 sensors.

The sequence can be described as the following:

a) Initially, both sensor are unblocked (i.e., **a** and **b** signals are "00")
b) Sensor **a** is blocked (signals are "10")
c) Sensor **b** is blocked (signals are "01")
d) Both sensors are unblocked ("00")

Although it is not required for this lab design, you may consider the question: What if the car is very long (a limousine) or very short (a smart)?

3. Detecting Vehicle Exiting the Garage

The sequence will be the opposite as that of entering:

a) Initially, both sensor are unblocked (i.e., **c** and **d** signals are "00")
b) Sensor **d** is blocked (signals are "01")
c) Sensor **c** is blocked (signals are "10")
d) Both sensors are unblocked ("00")

Although it is not required for this lab design, you may consider the questions: What if the car is very long (a limousine) or very short (a smart)? Could a pair of sensors be "11" or "00" in between steps b) and c)?

IMPOTANT!

When you press a button, there is a chance that the button will not simply go from open to close. Since a button is a mechanical device, the contacts can bounce. For a short period after the button is pressed the value you read from an IO pin may toggle between 0 and 1 a few times before settling on the actual value. To debounce a button, you just need to require that for a button to register as being pressed, it must look like its being pressed for a set amount of time. In this case, being pressed is when the value of the button is 1. If you read enough 1's in a row it is safe to assume that the button has stopped bouncing and you can register one button press. If you fail to do this and you are using the button to increment a counter, then the counter may increase by more than 1 per button press since it will appear that each bounce was a separate press.

4. Prelab Tasks

A. Design an 8-bit counter: The counter has two inputs (inc and dec). When inc = 1, the counter is incremented by 1. When dec = 1, the counter is decremented by 1. The block diagram is shown in Fig. 3. Write the Verilog code and test it through Vivado simulation.

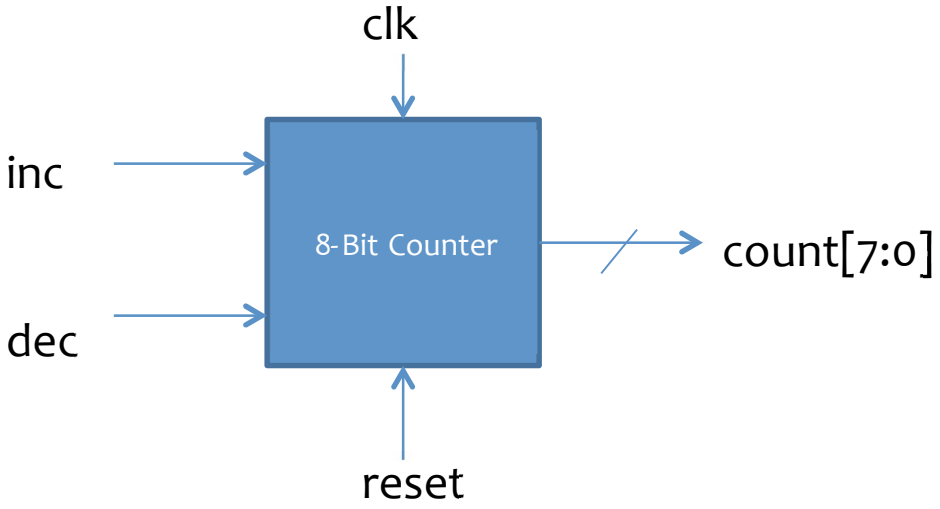

Figure 3. Block diagram of an 8 -bit counter

```
module counter(
input   clk,                    //clock
input   reset,                  //reset
input   increment,              //increment
input   decrement,              //decrement
output [7:0] current_count          //count, max count is 255 (8-bit)
);
reg [7:0] current_count = 0;        //initially setting counter to zero

always @(posedge clk) begin             //when positive edge of the clock arrives
        if(reset)
                current_count <= 0;
```

```
        else if(increment)
                current_count <= current_count + 1;          //increment
        else if(decrement)
                current_count <= current_count - 1; //decrement
        else
                current_count <= current_count;
end
        assign count = current_count;

endmodule
```

B. Design a finite state machine (FSM) to detect a car entering the garage: the FSM generates a pulse of signal (inc = 1) for one clock cycle for each car entry.

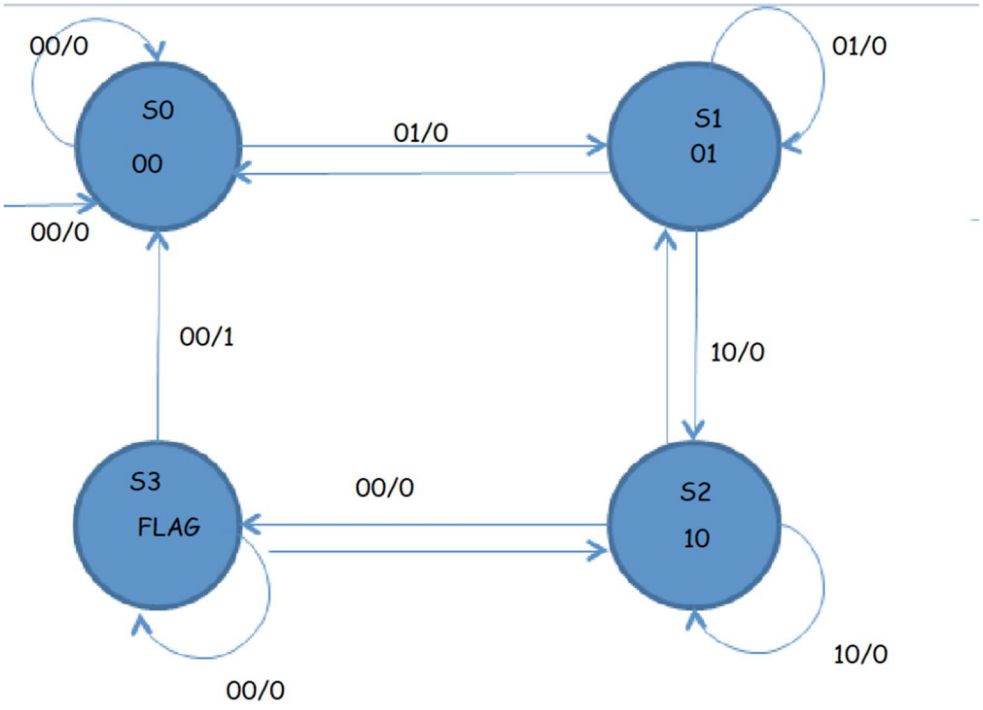

Figure 4. An example of the FSM for detecting the car entering the garage

There are many different ways to design such as FSM. You can design in a different way. Test your design using a simulation testbench in Vivado.

```
module FSM(
   input clk,
   input reset,
   input sensor_1,
   input sensor_2,
   output count_flag);
```

```
   localparam S00 = 0, S01 = 1, S10 = 2, FLAG = 3;

   reg [1:0] current_state = 0;                  // Current State
   reg [1:0] next_state = 0;                     // Next State
   reg set_flag = 1;                             // Flag for when to count

//----------------------------------------------------------------------------
// Next state sequential logic
//----------------------------------------------------------------------------

   always @(posedge clk) begin
      if (reset) current_state <= S00;
else current_state <= next_state;
   end

//----------------------------------------------------------------------------
// Next state combinational logic
//----------------------------------------------------------------------------

   always @(current_state, sensor_1, sensor_2)
   begin
      case(current_state)
         S00: begin
                  if (sensor_1 & ~sensor_2)
                           next_state = S01;
                  else
                           next_state = S00;
         end
         S01: begin
                  if (~sensor_1 & sensor_2)
                           next_state = S10;
                  else
                           next_state = S01;
         end
         S10: begin
                  if(~sensor_1 & ~sensor_2)
                           next_state = FLAG;
                  else
                           next_state = S10;
         end
         FLAG: begin
                  next_state = S00;
```

```
        end
        default: begin   // Implied-latch free implementation
          next_state = S00;
        end
endcase
end

//-------------------------------------------------------------------------------
// Combinational output logic for each state
//-------------------------------------------------------------------------------

  always @(current_state) begin
case (current_state)
        S00: begin
                set_flag = 0;
  end
        S01: begin
                set_flag = 0;
        end
        S10: begin
                set_flag = 0;
        end
        FLAG: begin
                set_flag = 1;
        end
        default: begin
          set_flag = 0;
       end
endcase
end

//-------------------------------------------------------------------------------
// Output assignment
//-------------------------------------------------------------------------------

assign count_flag = set_flag;
endmodule
```

Similarly, design a finite state machine (FSM) to detect a car exiting the garage: the FSM generates a pulse of signal (dec = 1) for one clock period for each car exit.

5. Prelab Signoff

You can signoff the prelab by demonstrating all the above 3 blocks in Vivado simulation environment.