Parking Lot Occupancy Counter

CSE 521 FPGA Design	
Winter 2018	
Name of Student: Adeeb Alqahtani	Name of Lab Instructor: Qingquan Sun
Date: March-07-18	Grade:

Abstract

In this lab we are going to construct a circuit that counts the number of cars in a parking structure using FSM methodology. The circuit can detect if the car is entering the parking structure or leaving then the counter should count up or down accordingly.

Objective and Introduction

The purpose of this lab is to understand how to use finite state machine methodology in Verilog to design a digital circuit. Unlike sequential circuits, FSM is used to represent random patterns.

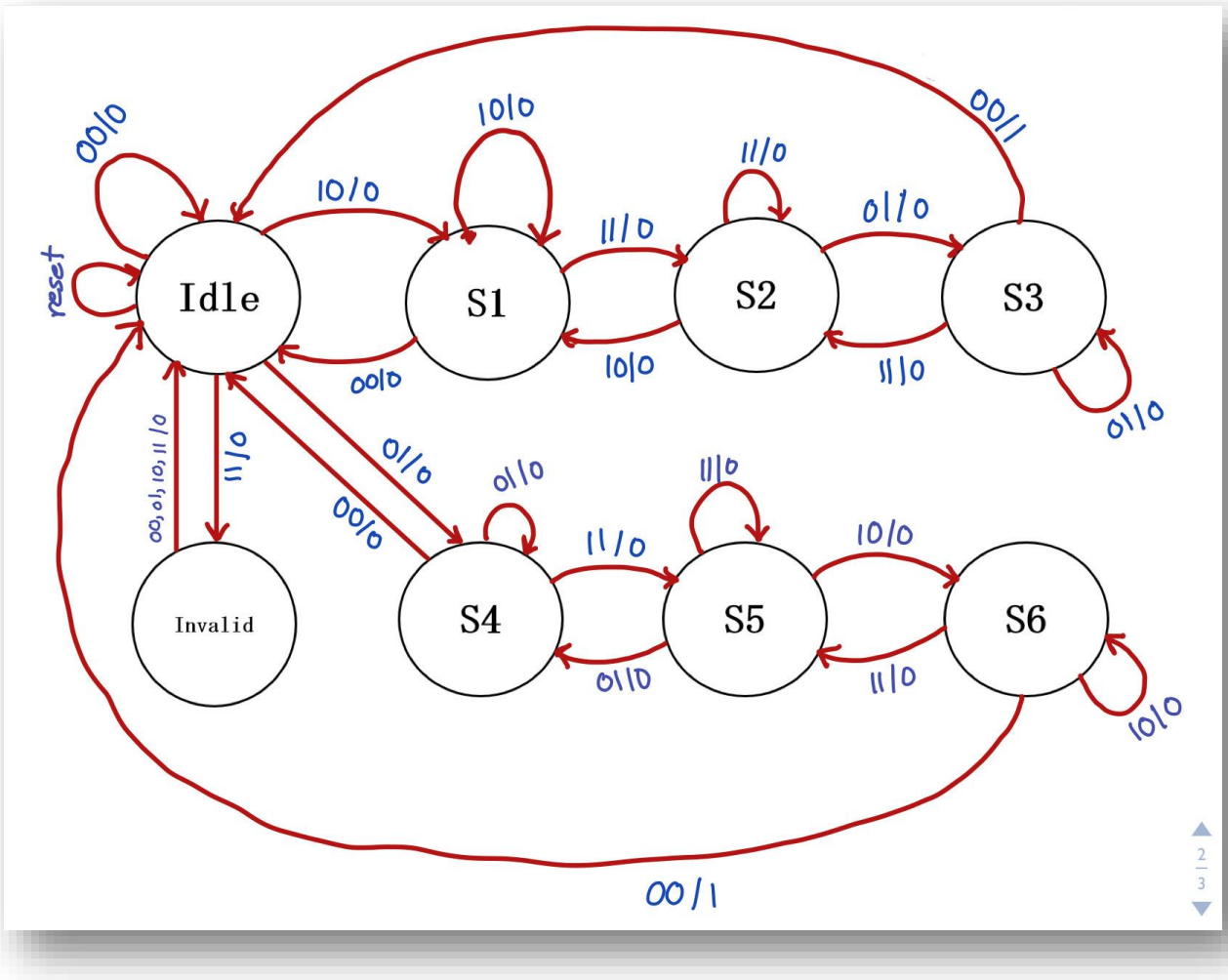
Theory and Experimental Methods

The module designed in this circuit can count the number of existing cars in a parking structure. It detects, using two sensors, if a car is leaving or entering the parking structure. If a car leaves the circuit should decrement 1 from the total number of cars and if a car enters the circuit adds 1 to keep the total number of cars updated.

The pattern of knowing if a car is entering is when the first sensor "a" is triggered, then both sensors a & b are triggered, then sensor "b" is ON and "a" is OFF and finally both sensors are OFF. After that a signal from FSM module is sent to counter to tell it to count 1 up. The opposite pattern is used to detect if a car is leaving. The following table is to show the pattern of leaving or entering of a car.

	A & B	A & B	A & B	A & B	counter
Entering	10	11	01	00	+1
Leaving	01	11	10	00	-1

Moreover, the module constructed is also able to know if a car leaving or entering is backing off and going back to the parking structure or going back to leave the parking structure and not entering. In this case, the circuit will not count up or down because the pattern for entering or leaving was not complete. In addition to that, if a car is on the way of leaving or entering and a pedestrian walk and triggered the other sensor, the circuit should know that it is not the car that is triggering the other sensor. Also, if a pedestrian is walking and triggered the sensors the circuit will also able to know it is pedestrian and not a car.



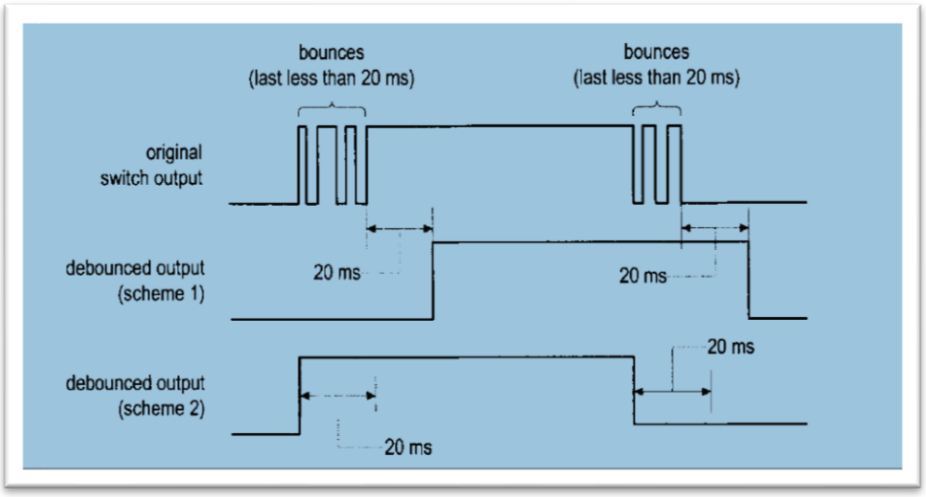
FSM Diagram for parking lot occupancy counter

From the FSM diagram, I’m constructing a mealy machine for this lab that has 8 states. The Verilog code should clearly describe each state and how to transit from state to other.

```
module ParkingFinal(
   input a,
   input b,
   input clk,
   input reset,
   output reg increment,
   output reg decrement
   );

localparam [2:0]
        idle = 3'b000,
        s1 = 3'b001,
        s2 = 3'b010,
        s3 = 3'b011,
        s4 = 3'b100,
        s5 = 3'b101,
        s6 = 3'b110,
        invalid =3'b111;
        //Internal Registers as Temp variabls
        reg[2:0] state_reg,state_next;
        initial begin
        state_reg = 0;       //Initializing state_reg to zero
        state_next = 0;
        end
           // STATE_REGESTER
        always@(posedge clk, posedge reset)
           begin
           if (reset)
              state_reg <= idle;
           else
              state_reg <= state_next;
           end
    always@*
    begin
       increment <= 0 ; // default value for inc going to the clock is 0 unless triggered
       decrement <= 0 ; // default value for dec going to the clock is 0 unless triggered

       case(state_reg)
       //initial state
          idle:
             if(a&~b) // car entering
                state_next <= s1;
             else if (~a&b) //care exiting
                state_next <= s4;
             else if (~a&~b) //No car
                state_next<=idle;
 // if both sensors are triggered at the same time without having a car entring or leaving
 // Impossible to happen but it's for the safety of the system.
             else if (a&b)
                state_next <= invalid;
          s1:
             if(a&b) // a car half way through entring the parking structure.
                state_next <= s2;
             else if(~a&~b) // if a car enters and then back off
                state_next <= idle;
             else if (a&~b) // car enters but not moving or moving slowly
                state_next<=s1;
          s2:
             if(~a&b) // car is alomost entering
                state_next <= s3;
             else if(a&~b)
                state_next <= s1; // backing off!!
             else if(a&b)
                state_next <= s2; //not moving!!
```

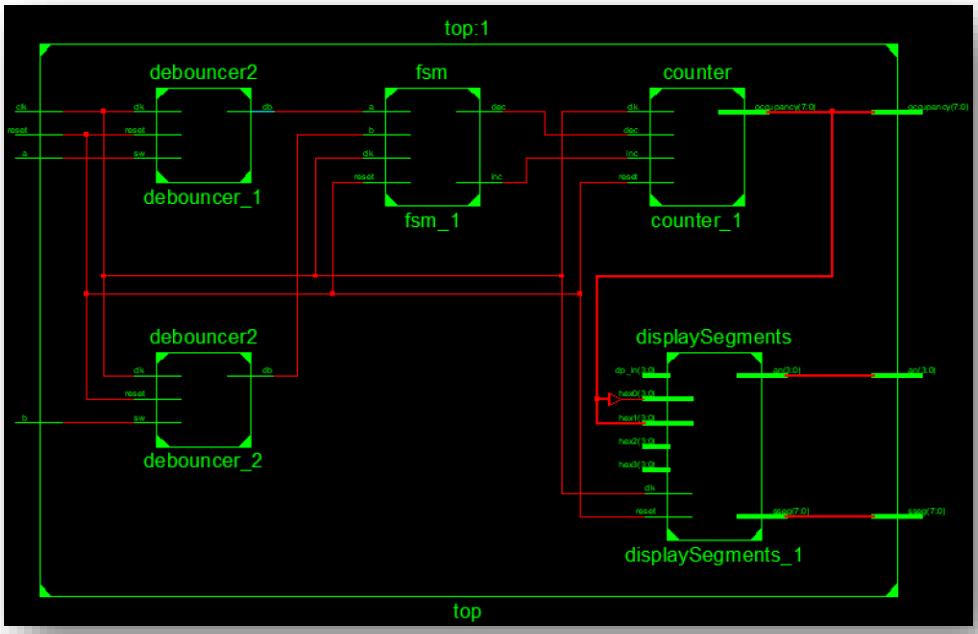
```
s3:
   if(~a&~b) // car entered!!
      begin
         increment <= 1; // incrementing the number of cars in the parking structure.
         state_next <= idle; // starting all over again
      end
   else if(a&b)   // the car is backing off and it is in the middle
      state_next <= s2;
   else if (~a&b) // the car not moving.!!
      state_next <= s3;

s4:
   if(a&b) // the car is half way through leving
      state_next <= s5;
   else if(~a&~b) // the car is backing off and going back the parking structure!!
      state_next <= idle;
   else if (~a&b) // The car not maving!! it triggered sensor b and stopped moving
      state_next <= s4;
s5:
   if(a&~b) // The car is almost leaving the parking structure
      state_next <= s6;
   else if(~a&b) //The car is backing off!!
      state_next <= s4;
   else if (a&b) // The car is half way through leaving but stopped moving
      state_next <= s5;
         s6:
            if(~a&~b) // the car left the parking structure!!
            begin
               decrement <= 1 ;
               state_next<=idle;
            end
            else if(a&b) // going back to the parking structure
               state_next <= s5;
            else if(a&~b) // The car is triggering sensor "a" on the way leaving the parking
                          //but did not leave.Stopped!!
               state_next <= s6;
               invalid:
            state_next<=idle;
       endcase
    end
endmodule
```

Hardware wise, this module will not work properly if implemented to the FPGA board unless the input signals a and b are connected to a debouncing circuit before they get passed to the FSM circuit. The propose for having a debouncing circuit is to make sure the input signal is stable and not fluctuating. Using mechanical switches like push-buttons and slide switches as input signal will result in an unstable signal because the switch when turned ON bounces back and forth for a period before it turns ON or OFF.



A figure from the book compare the original switch output with a debounced one

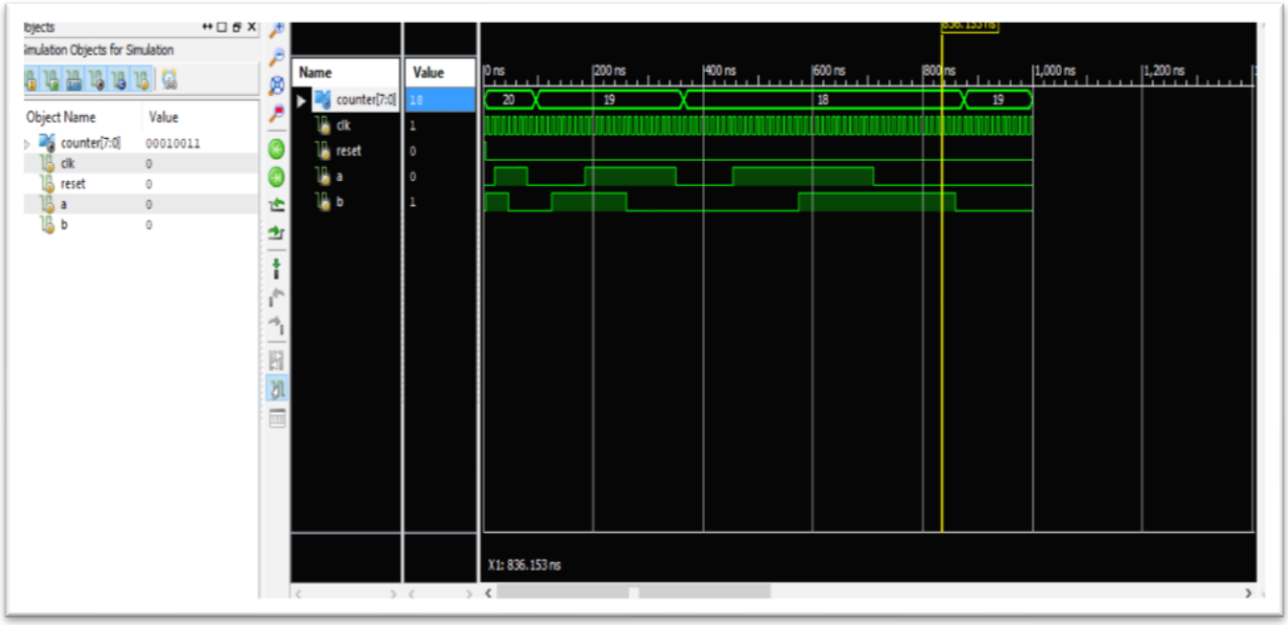
The outputs of the FSM module is connected to the inputs of a two-signal controlled counter that can increment or decrement based on the inputs coming from the FSM module. The output of the counter is displayed using the 4 digits 7-segment display. This time, 2 digits are being used to represent 8-bits binary number in hex. The output counter [7:4] is displayed in hex3 and the output counter [3:0] is displayed in hex2. The other 2 digits can be left unconnected or set to zero. Both ways give the same result since it is low active controlled LED's.



The schematic diagram of the circuit

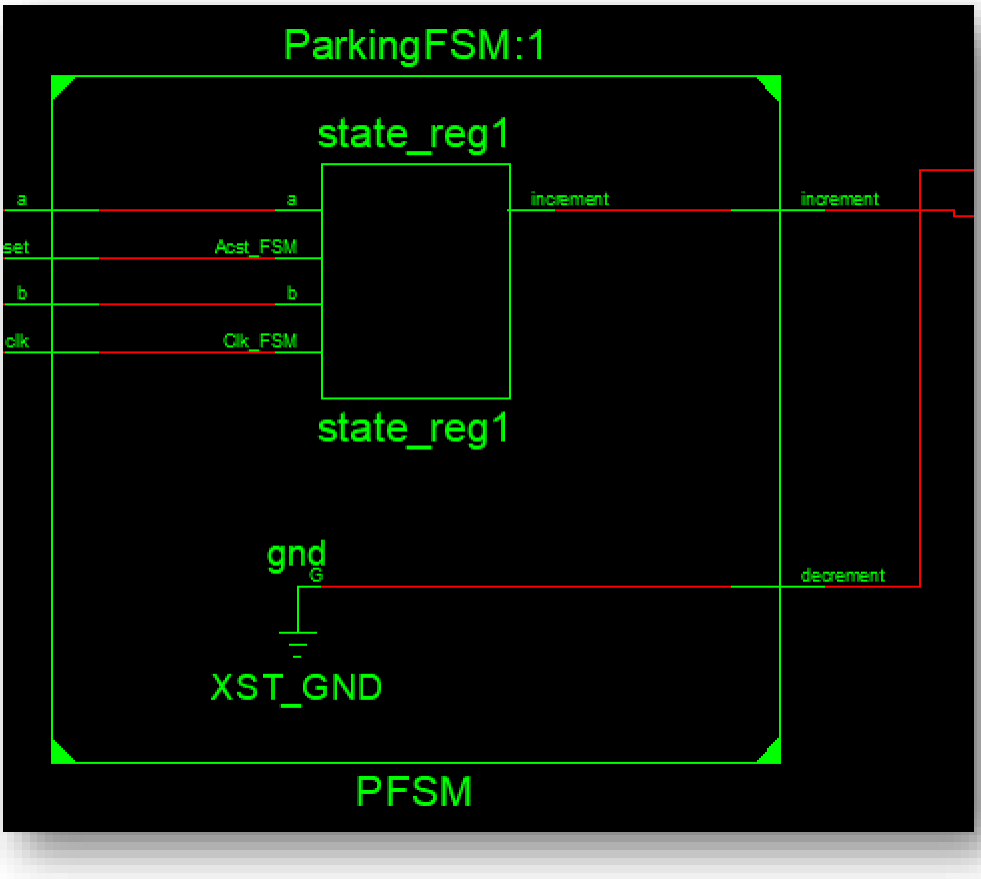
Results and Discussion

I was able to run the simulation for the project and the result came as expected. It is clear from the simulation that the initial number of cars were 20. Two cars left, and one entered making the total number of cars =19.



A figure showing the result of simulating the movements of 3 cars

However, I learned from this lab not to always trust Xilinx ISE. When I implemented the code on the FPGA board for the first time, the counter was counting up but not counting down. In other words, the circuit seemed not to detect cars leaving the parking lot. I did a test bench for the counter and from the simulation it was working fine. The ISE console did not show any error which made the situation even worse. I modified my code so many times, but nothing seemed to work. I had to try other FPGA boards, but I had the same issue.
However, two days later and after going over the schematic diagram, I found that the wire that supposed to tell the counter to count down because a car just left was connected to ground.



A schematic diagram showing decrement wire connected to ground

This was strange because when I run the code in Vivado it would decrement. The solution for this problem was to delete the project and type the code again on another ISE project in JB computer lab and rewrite the code carefully.
Carefully means not to rely on the “check syntax” process. The fix was to

block the state that decrements 1 with begin and end. This was an obvious fix and caused by an incorrect code. However, I'm still not sure why it worked fine on Vivado.
This is the right sentax.

```
always@* begin
       increment <= 0;
       decrement <= 0;
       case(state_reg)
-
-
-
_
s6:
if(~a&~b)              begin
decrement <= 1;
state_next<=idle;
               end
```

Moreover, when I added the other pieces, debouching and multiplexing circuits, another problem came and cause frustration. I thought that adding the debouching and multiplexing circuits was not done properly. The ISE console did not show any error but saying the synthesize fail. However, it turns out that Xilinx64 version causes issues when synthesizing the top module.

The shortcut on my desktop was 32-bit version

and the default project navigator was the 64-bit version. So, when I open project navigator first from my desktop the module runs perfectly, but when I double click on the project file from the project directory it doesn't. So, I made Xilinx32 version as the default program. That fixed the problem and Synthesize-XST now does not fail. This fixed would have saved a lot of time modifying a working code, that I thought was the problem, if discovered earlier.

Conclusions

In this lab I learned how write a Verilog code based on finite state machine diagram. The first step in writing the code is specify your inputs and outputs. The clock signal plays the major part here enabling the transit

between states because FSM is a sequential logic circuit but with random pattern. After that, you need to define your states using “localparam” to give it a values and readable name. For example,

```
localparam [2:0]
                  idle = 3'b000,
                  s1   = 3'b001,
                  s2   = 3'b010;
```

S0, s1 and s3 are states of an FSM. To be able to transit between them, you need to define 2 registers for this specific design, one for state register and the other for the next state logic. Bothe registers must have the same size as “localparam” variables. The outputs could either be wires or registers. I’m using my 2 outputs as registers, so I can assign values to them within the case statement that is used to transit between states. For example:

```
if (~a & ~b) begin
      inc <= 1;
      state_next <= idle;
      end
```

The debouncing circuit worked perfectly with my circuit. I did not need to modify anything, I just connected it with the FSM module using top module. Unlike previous lab assignments, I spend way more time debugging than coding. I have not gone this deep into debugging a code for any course than this lab assignment. Even though it was painful, I learned a lot.