EE 371, Lab 1
Parking Lot Occupancy Counter

Lab Objectives

This lab is a refresher to the finite state machines that you learned in EE 271 by designing a parking lot occupancy counter.

Laboratory Design Kits

We will use the DE1-SoC development board that you used in EE 271. Please pick up your lab kit from the EE store.

Using Quartus II Software

The Quartus II web edition release 17.0 is preloaded on the machines in the department, and you can do all the work on these PCs. However, if you have a PC of your own that you would like to use, you can install the software for free. To install the software, go to http://fpgasoftware.intel.com/17.0/?edition=lite&platform=windows&download_manager=dlm3 , **choose version 17.0 from the top drop-down menu**, download the free web edition and install it. Note that you will have to register to be able to download the software. The file to download is the 5.8 GB tar file under the "Combined files" tab. Extract the tar file using a program like 7-zip and run the QuartusLiteSetup-17.0-windows.exe file. When it asks for the components to install, make sure you select each of these:

- Quartus Prime Lite Edition (Free)
- Devices: Cyclone V
- ModelSim: Intel FPGA Starter Edition (Free)

When the software is done, make sure to install the USBblaster driver. Run Quartus next, and if asked about licensing just run the software (we use the free version, so no license required).

NOTE: If you have trouble accessing the website to download the software, you can download it from this google drive. (you need to login using your netID)
https://drive.google.com/open?id=1Tl_1G_DFn6Yps0DuEh3yUAr4vXpJ48SF

Task 0: Warm up exercise on Quartus and ModelSim

This task is a refresher on creating a Quartus project, writing a SystemVerilog program and simulating it in ModelSim. You can skip this task if you do not need a refresher.

For this task, follow along a series of video tutorials that will walk you through the steps of creating a full adder using SystemVerilog and simulating the design in ModelSim. For your reference, the source code used in the tutorials is in the appendix of this document.

Please follow the tutorials in the following order:

1. Launch the Quartus Prime software.
2. Create a project from scratch. Please follow the steps in the following videos and use the same project name as in the video
   https://youtu.be/iLbmSTG7bpA

3. Implement the full adder using SystemVerilog and simulate it using ModelSim. Please follow the following video: *Please note that in the video when it refers to compiling the project for the first time, it may give you a compilation error. If you run the video for few more seconds it'll tell you about setting the top level modules so that the program compiles*.
   https://youtu.be/BcvclrqZ2fc

3. Mapping a SystemVerilog design to an FPGA.
   https://youtu.be/mnZt2iNNfp4

5. Loading the design onto the FPGA. Please follow the steps in the following video and test your full adder on the DE1_SoC board and verify that it matches the truth table.
   https://youtu.be/uMxA3VcS3f8

After completing this task, you should create future projects in a similar fashion and use SystemVerilog and ModelSim accordingly.

Assigned Task

Parking Lot Occupancy Counter

Consider a parking lot with a single entry and exit gate. Two pairs of photo sensors are used to monitor the activity of cars, as shown in Figure 1. When an object is between the photo transmitter and the photo receiver, the light is blocked, and the corresponding output is asserted to 1. By monitoring the events of two sensors, we can determine whether a car is entering or exiting, or a pedestrian is passing through. For example, the following sequence indicates that a car enters the lot:

- Initially, both sensors are unblocked (i.e., the a and b signals are "00").
- Sensor a is blocked (i.e., the a and b signals are "10").
- Both sensors are blocked (i.e., the a and b signals are "11").

- Sensor a is unblocked (i.e., the a and b signals are "01").
- Both sensors become unblocked (i.e., the a and b signals are "00").

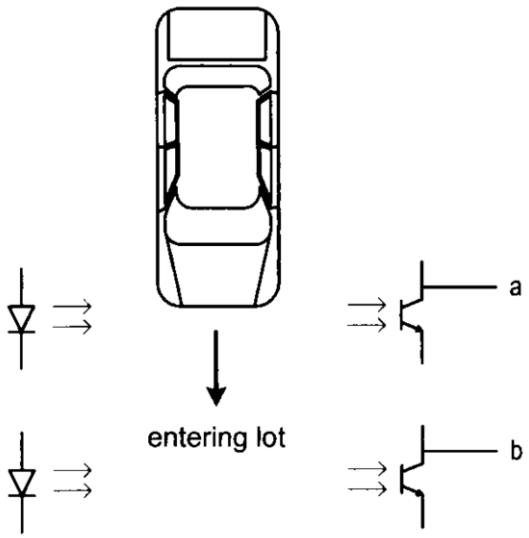

Figure 1 Parking lot sensors

Design a parking lot occupancy counter as follows:

1. Design an FSM with two input signals, a and b, and two output signals, *enter* and *exit*. The *enter* and *exit* signals assert true for one clock cycle when a car or exits the lot, respectively. Derive the SystemVerilog code for the FSM and simulate it in ModelSim. You may assume that cars will not change direction while entering or exiting the parking lot.
2. Design a counter with two control signals, *inc* and *dec*, which increment and decrement the counter when asserted. Assume that the maximum capacity of the parking lot is 25 spots. Derive the SystemVerilog code for the FSM and simulate it in ModelSim.
3. Combine the counter and the FSM and model the parking lot, using push buttons to mimic the two sensor outputs and the seven-segment displays to display the car count. Your system should have the following:
    a. Display the car counts as they enter the parking lot on the seven-segment displays HEX0 and HEX1.
    b. If the counter reaches 25, display the word “FULL” on HEX5-HEX2
    c. As cars exit the lot, the counter decrements and the corresponding number should be displayed on HEX0 and HEX1.
    d. When the lot is empty. Display the word “EMPTY” on HEX5-HEX1 and display the number ‘0’ on HEX0.
    e. Use 2 off-board LEDs to represent the a and b signals. When a is 1, turn on a red LED, and when a is 0, turn off a red LED. Stimulatingly, when b is 1, turn on a green LED, and when b is 0, turn off a green LED.
4. Simulate the system in ModelSim.
5. Download the program to the DE1_SoC board and demonstrate it to your TA.
6. Create a block diagram for your system and include it in your lab report.

Lab Demonstration and Submission Requirements

- Submit a technical lab report that summarizes the procedures and results obtained in the lab. The report should include
  - o Abstract, introduction, procedure(s), results and analysis, and conclusion sections. The procedures and results sections should elaborate on the details of your design and experiments. Include screen shots of your code and simulation of the parking lot occupancy counter.
- In your report, include the block diagram and the state diagrams of the system.
- Submit the SystemVerilog of your parking lot occupancy counter and the ModelSim waveforms of all modules.
- Submit the Flow Summary (produced during compilation) of compiling your system.
- In your report, include the number of hours (estimated) it took to complete this lab, including reading, planning, design, coding, debugging, testing, etc. Everything related to the lab (in total).
- Submit your report and programs to Canvas. No hard copies.

Appendix

Source code used in the videos

1. FullAdder module

```
module fullAdder (A,B, cin, sum, cout);

   input logic A,B, cin;
   output logic sum, cout;

   assign sum = A ^ B ^ cin;
   assign cout = A&B | cin & (A^B);

endmodule

module fullAdder_testbench();

      logic A, B, cin, sum, cout;

      fullAdder dut (A, B, cin, sum, cout);

      initial begin

         A = 0; B= 0; cin =0; #10;
                      cin =1; #10;
                B= 1; cin =0; #10;
                      cin =1; #10;
         A = 1; B= 0; cin =0; #10;
                      cin =1; #10;
                B= 1; cin =0; #10;
                      cin =1; #10;

      $stop;

      end //initial

endmodule
```

2. DE1_Soc module

```
module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);

   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic [9:0] LEDR;
   input logic [3:0] KEY;
   input logic [9:0] SW;

   fullAdder FA (.A(SW[2]), .B(SW[1]), .cin(SW[0]), .sum(LEDR[0]), .cout(LEDR[1]));

   assign HEX0 = 7'b1111111;
   assign HEX1 = 7'b1111111;
   assign HEX2 = 7'b1111111;
   assign HEX3 = 7'b1111111;
   assign HEX4 = 7'b1111111;
   assign HEX5 = 7'b1111111;

endmodule

module DE1_SoC_testbench();

   logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   logic [9:0] LEDR;
   logic [3:0] KEY;
   logic [9:0] SW;

   DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .LEDR, .SW);

   integer i;
   initial begin
   SW[9] = 1'b0;
   SW[8] = 1'b0;
   for (i=0; i<2**8; i++) begin
      SW[7:0] = i; #10;
   end
end

endmodule
```