EE/CSE 371:
Design of Digital Circuits and Systems

Lab6: System Verilog in the Real World

Lab Objectives

In lab 1, you were tasked on creating a finite state machine to design a parking lot occupancy counter. Simple GPIO pins and a virtual breadboard were used to emulate the parking lot sensors. In the real world and in industry, you'll find that your SystemVerilog designs will interface with complex sensors and scenarios to create larger systems. In this lab, we will re-introduce the parking lot occupancy, but instead of using simple breadboard components, we will be simulating an actual, 3D parking lot, while also integrating the concepts learned throughout the quarter.

Task 1: Parking Lot Breadboard

Take your implementation for lab 1, and demonstrate your functionality onto the *new* breadboard remote lab interface. You will need to change GPIO_0 reference to V_GPIO in your top-level module, as well as wire it to different GPIO pin numbers.

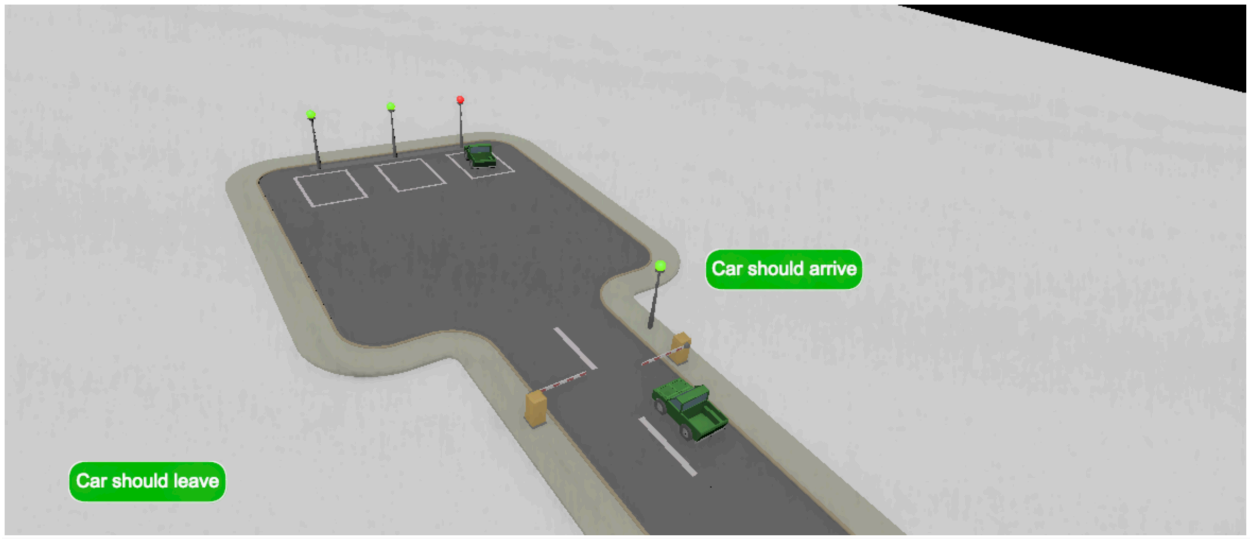
Note: The user interface for the breadboard and the FPGA input and output pins have changed. Please read "Updated GPIO Guide.pdf" available in the Files section on Canvas.

This task is graded on completion only.

Task 2: Parking Lot 3D Simulation

Consider the parking lot again with a single entry and exit gate. For simplicity, this parking lot can hold a maximum of 3 cars. One car can enter the parking lot, and one car can leave the parking lot at a time. The user will control when a car will arrive at the entrance gate through a button on the 3D simulation. When a car arrives, the rest of the functionality of the car is autonomous: the car will park at the parking spot of its choosing. Similarly, the user will control when a car will leave the parking lot. The entering or exiting car will wait at the entrance/exit gate until the gate opens, which is controllable by asserting the appropriate GPIO pin.

Read "3D Parking Lot Guide.pdf" available in the Files section on Canvas for a guide on the UI of the 3D Parking Lot.



User Interface of the 3D Parking Lot Simulator

Car Simulation and GPIO Mapping

FPGA Inputs:

- Presence Parking 1: Lets you know if there is a car in parking spot 1. V_GPIO[28]
- Presence Parking 2: Lets you know if there is a car in parking spot 2. V_GPIO[29]
- Presence Parking 3: Lets you know if there is a car in parking spot 3. V_GPIO[30]
- Presence at Entrance Gate: Lets you know if there is a car waiting at the entrance. V_GPIO[23]
- Presence at Exit Gate: Lets you know if there is a car waiting at the exit. V_GPIO[24]

FPGA Outputs:

- LED Parking 1: Lets you change LED color of parking spot 1 (0 = green, 1 = red). V_GPIO[26]
- LED Parking 2: Lets you change LED color of parking spot 2 (0 = green, 1 = red). V_GPIO[27]
- LED Parking 3: Lets you change LED color of parking spot 3 (0 = green, 1 = red). V_GPIO[32]
- LED Full: Lets you change LED color of the full indicator LED (0 = green, 1 = red). V_GPIO[34]
- Open Entrance Gate: Opens the entrance gate. When you send a 1, the gate will stay open until a car enters the lot. V_GPIO[31]
- Open Exit Gate: Opens the exit gate. When you send a 1, the gate will stay open until a car leaves the lot. V_GPIO[33]

Rush Hour

Pretend this parking lot is for a restaurant that has an 8 hour work day. Pressing KEY[0] will increment the work day by 1 hour. Many cars can enter and/or leave the parking lot in a given hour. We start the day at hour 0, and go up to hour 7.

The end of rush hour for this parking lot is signaled when there are 3 cars in the parking lot (i.e. the parking lot is full), and then at some point in that work day, all 3 cars leave (i.e. the parking lot is empty). For this lab, we want to know at which time in the work day (hours 0 to 7) marks the end of the *first* rush hour (i.e. when there are 3 cars, at what time does the last car leave the parking lot). Construct an FSM and ASMD to find rush hour (using separate control and datapath modules).

Valid end of rush hour sequences include:

3 ➔ 2 ➔ 1 ➔ 0

3 ➔ 2 ➔ 3 ➔ 2 ➔ 1 ➔ 0

3 ➔ 2 ➔ 1 ➔ 2 ➔ 1 ➔ 0

3 ➔ 2 ➔ 1 ➔ 2 ➔ 1 ➔ 2 ➔ 1 ➔ 0

Etc.

At the end of the work day (i.e. HEX5 shows "7", and then you increment the hour one more time), display on HEX4 which hour marked the end of rush hour, and display on HEX3 which hour marked the start of rush hour. If no end of rush hour occurred during that day, display a "-" instead for both HEX displays.

Car Tracking and RAM

In addition to Rush Hour, we would like to know how many total number of cars have entered the parking lot at each given business hour. Using a dual-port 8x16 RAM (separate channels and addresses for read and write), save in each address the total number of cars that have entered the parking lot. Here, the address of the RAM will signify the business hour. At the end of the 8^{th} hour, use a counter that cycles through the 8 different addresses of your RAM and reads the data, cycling through each address every 1 second, similar to Task 2 of Lab 2. Display that RAM data on HEX1 and the RAM address at HEX2. On the 8^{th} hour, the RAM address counter does not need to start at 0, but will need to cycle from address 7 back down to 0.

You can create your own dual-port RAM or use the IP catalog.

Example scenario:

Business Hour	Car Leaving or Entering
0	1 car enters, no cars leave
1	No cars enter, 1 car leaves
2	2 cars enter, no cars leave
3	No cars enter, no cars leave
4	No cars enter, no cars leave
5	1 car enters, no cars leave
6	No cars enter, no cars leave
7	No cars enter, 3 cars leave

After this 8th hour, HEX4 should display "7" to signify that the end of rush hour. HEX3 should display "5" to signal the start of rush hour. HEX1 and HEX2 will cycle through the RAM displaying the following:

HEX2 (RAM Addr)	HEX1 (RAM Data)
0	1
1	1
2	3
3	3
4	3
5	4
6	4
7	4

Design Requirements

Your design *must* follow these requirements:

- Your design must autonomously open the entrance gate, but only if there are parking spaces available and a car presence at the entrance gate is felt.
- Your design must autonomously open the exit gate, but only if an exiting car presence is felt.
- If a car goes into a designated parking lot space, the LED for that space must automatically be illuminated red to indicate that it is occupied.
- If a parking lot space is vacant, the LED for that space must automatically be illuminated green to indicate that it is currently vacant.
- If the parking lot is full, display the word "full" on the HEX displays HEX3 – HEX0. All other HEX displays must be blank, except HEX5 (see below). The parking lot "full LED" must also illuminate red.

- If the parking lot is not full, display the number of remaining parking lot spaces on HEX0. All other HEX displays must be blank, except HEX5 (see below). The parking lot "full LED" must illuminate green.
- Press KEY[0] to advance the work day hour by 1 hour.
- Show on HEX5 the current hour in the work day (0-7).
- At the end of the 8th hour (i.e. HEX5 shows "7", and then you increment the hour one more time), signal on HEX4 which hour marked the end of rush hour, and on HEX3, which hour marked the start of rush hour. See the "Rush Hour" section for an explanation on what constitutes the end of rush hour. All other HEX displays (except HEX2 and HEX1, see bullet requirement below) must be blank. Mark "-" on HEX4 and HEX3 if there were none for that day. If there are multiple, show the *first* time that it happens.
- Also at the end of the 8th hour, (i.e. HEX5 shows "7", and then you increment the hour one more time), cycle through the data in your 8x16 RAM on HEX1 (which indicates the total number of cars that have entered the parking lot by that specific hour in the work day) and the RAM address on HEX2 (which indicates the work day hour). Cycle through this at roughly 1 second, and the address will need to count up to 7 and wrap back to 0.
- Use SW[9] to reset the work day back to 0, and any FSM/ASMD used.

Lab Demonstration and Submission Requirements

- Record a video to demo the two tasks. You will need to demonstrate the soundness of your design by executing the design on the FPGA. Moreover, you are expected to utilize testbench simulations to demonstrate that your modules work as expected.
- Write a Lab Report, as framed by the Lab Report Outline document on Canvas. Comment your code. Follow commenting guidelines as discussed in the Commenting Code document on Canvas. Submit your lab report as a pdf file and all of your SystemVerilog files on Canvas.
- You do not need to re-simulate Task 1. But document the changes that you made from your original lab1 in your Lab Report (i.e. the different GPIO configurations, etc.).