Jiovany Soliman, 1951338

Chloe’ Miller, 2031648

EE/CSE 371 Lab 1 Report: Parking Lot Occupancy Counter

October 6, 2023

**Design Procedure**

In this lab, we were tasked with building a Parking Lot Occupancy Counter system. This system uses two photosensor inputs to determine when a car has entered or exited the parking lot and displays the current number of cars in the lot on the 7-segment HEX displays. The displays will state “CLEAR0” when the parking lot is empty, and “FULL16” when the parking lot has reached maximum capacity. This system only accounts for cars entering and exiting the lot and completely disregards pedestrians passing through.

We used UW LabsLand for demonstration of lab functionality.

**Task #1:**

For this Parking Lot Occupancy Counter system, we first decided to create a block diagram of the overall system according to the lab specifications. By creating a block diagram as shown in Figure 1, we were able to see how the modules were connected and what inputs and outputs were required for each, allowing us to split up the lab work effectively.

**A diagram of a car wiring

Description automatically generated**

Figure 1: Block diagram of overall Parking Lot Occupancy system.

**Task #2:**

After identifying how the high-level functionality of the system should function, we started implementing the car detection finite state machine. The most important feature of the car detection module is its ability to detect two specific sequences from the outer and inner photosensors for entering and exiting cars. An equally important feature of this module is its ability to disregard all other sequences from the photosensors, such as the sequence for pedestrians passing through. If an entering or exiting sequence is identified, the car detection module needs to signal that a car has either entered or exited the parking lot, or else the signals need to remain unaffected. The number of cars in the parking lot is indicated on the HEX displays.

We used a Moore machine to design the car detection module. Although Moore machines react slower to inputs than Mealy machines, the Parking Lot Occupancy system is assumed to be a slow system in which the inputs are not changing rapidly. Additionally, Moore machines are much easier to design, which was advantageous when trying to design an FSM that can track two sequences while ignoring all others. Figure 2 shows that 8 states were required to enable the Moore machine to track the enter and exit sequences, as well as restart the sequence tracking when any inputs did not match so that the enter and exit signals remained unaffected.

**A diagram of a diagram

Description automatically generated**

Figure 2: Moore FSM state diagram of the car detection module.

**Task #3:**

After implementing the car detection FSM, we decided to implement the car counter module using a Mealy FSM. This machine has inputs for incrementing and decrementing the count of cars in the parking lot, plus it drives the 7-segment HEX displays. The most important features of the car counter module are its ability to accurately display the current number of cars in the parking lot on the seven-segment HEX displays in response to the enter and exit signals, and display when the lot is either empty or full. When the lot reaches maximum capacity (16 cars), the car counter module needs to display “FULL16,” and when the lot is empty (0 cars), it needs to display “CLEAR0.”

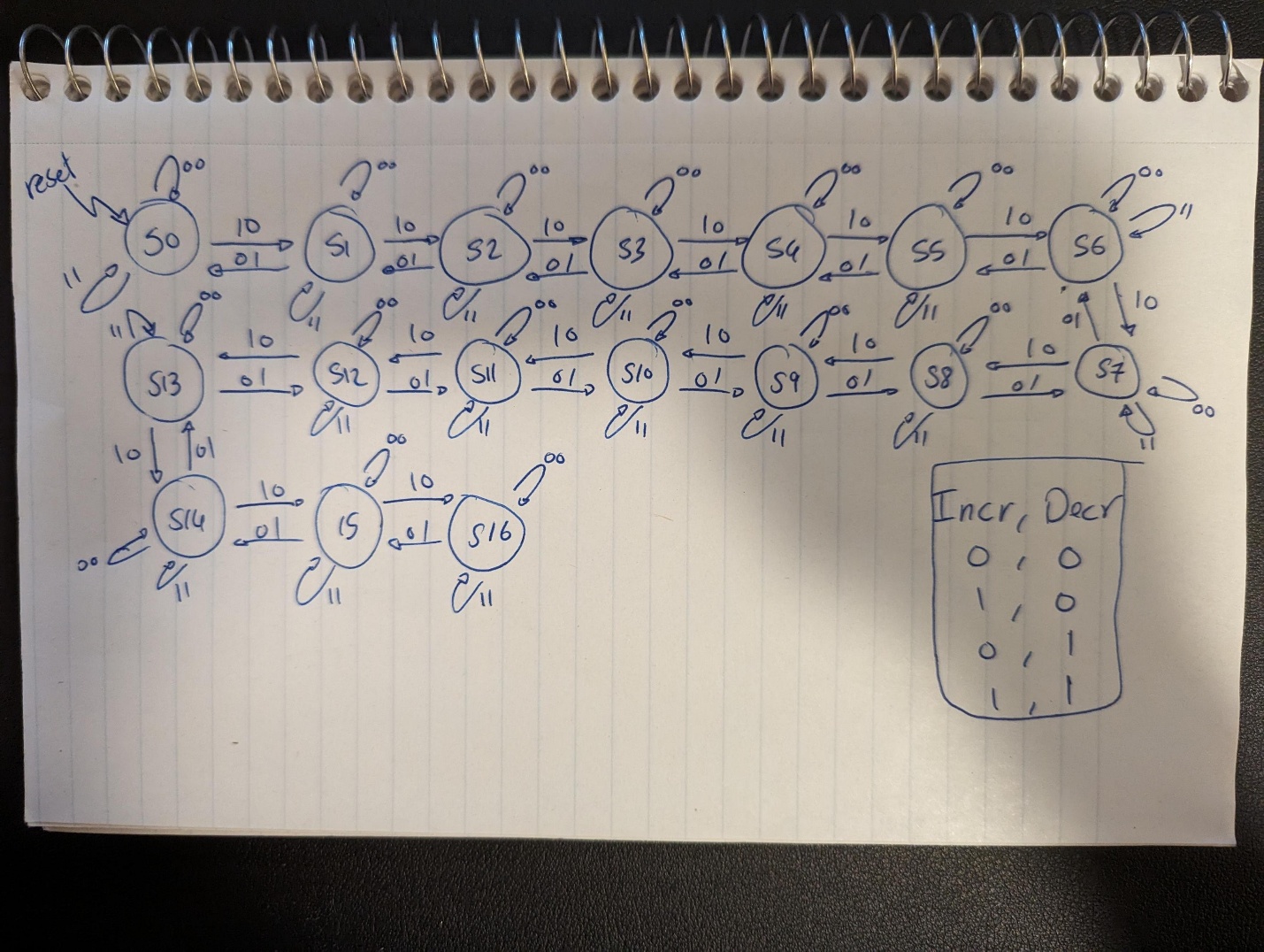


Figure 3: Mealy FSM diagram for the car counter module.

**Task #4:**

After successfully completing the implementation of the sub-modules and ensuring their functionality, we started implementing the parking lot occupancy module and the DE1\_SoC module. The parking lot occupancy module simply connects the car detection and car counter modules together and connects their inputs and outputs to the off-board input switch signals and on-board seven-segment HEX displays. The DE1\_SoC Top-Entity module is responsible for using the V\_GPIO expansion header to physically wire the switches (inner sensor, outer sensor, and reset) to their corresponding LEDs and inputs of the parking lot occupancy module.

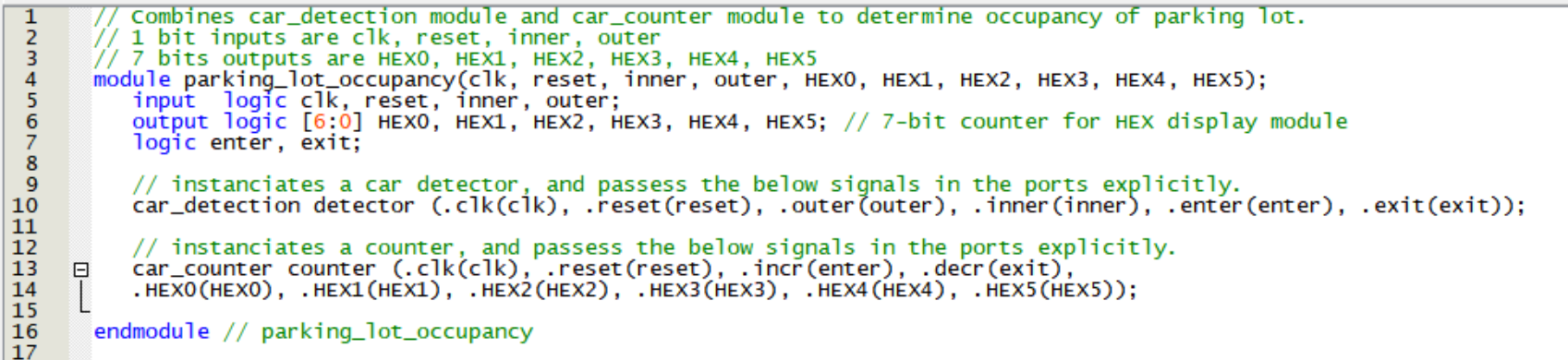
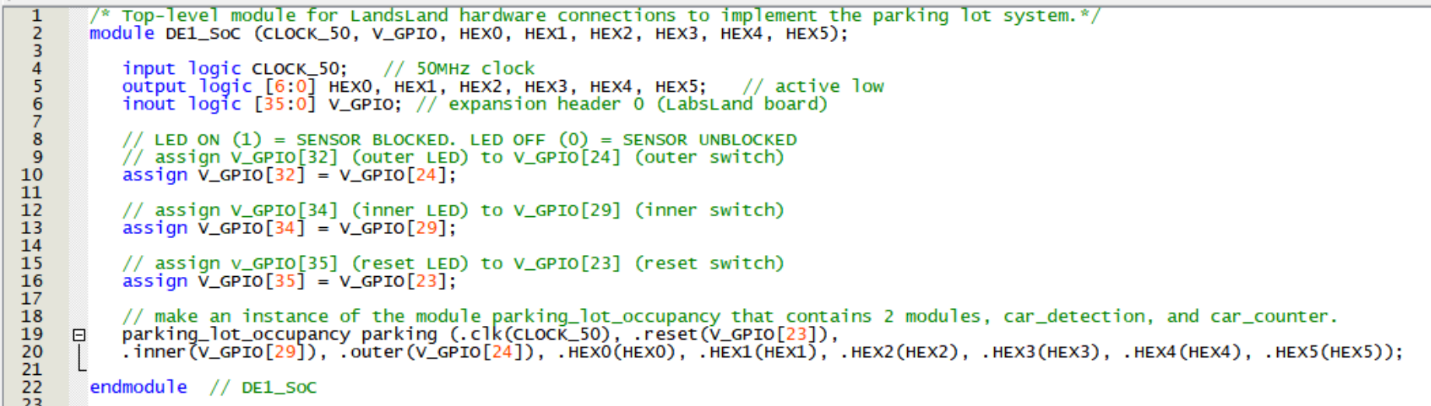


Figure 4: Snippet of SystemVerilog parking\_lot\_occupancy module.

 Figure 5: Snippet of SystemVerilog DE1\_SoC Top-Level Entity module.

**Results**

Our completed design of the Parking Lot Occupancy system meets the requirements of Lab 1. Whenever a car enters the parking lot, the counter is incremented by 1. Similarly, when a car leaves the parking lot, the counter is decremented by 1. The counter is always within the range of 0-16, with “CLEAR0” and “FULL16” displayed for 0 cars and 16 cars, respectively. The Parking Lot Occupancy system ignores pedestrians and other photosensor sequence inputs that do not indicate a car has entered or exited the lot.

To verify that the car detection module was functioning properly, we created a testbench that passes an entering sequence, exiting sequence, a sequence that keeps the module in the initial state S0, and a pedestrian sequence, and analyzed the behavior. Figure 6 shows how the car detection module transitioned between the present state (ps) and next state (ns) based on the sequence of inner and outer signals, and when the enter and exit sequences were successfully identified.

Between 0 ps and 886 ps the exit sequence was tested, which takes the car detection module from the initial state S0 to the exit sequence end state S6. Testing this sequence slowly allowed us to verify that the module will remain in the same state for several clock cycles. When the module reaches state S6, the exit signal is true for one clock cycle before the module returns to the initial state to track another sequence.

From 886 ps to 1485 ps, the enter sequence was tested, and took the car detection module from S0 to the enter sequence end state S5. When the module reaches S5, the enter signal is true for one clock cycle before the module returns to S0 again.

Finally, the car detection module is then tested with sequences that are not to be identified as the entering or exiting sequence. From 1485 ps to 2323 ps, we test to make sure that from S0, the module only begins to track a sequence once both inner and outer are false (indicating that both photosensors are unblocked, which is state S1). And from 2323 ps to 3269 ps, a pedestrian sequence is tested. For both tests, we see that neither the enter nor the exit signals registered as true, indicating that our car detection module was functioning properly.

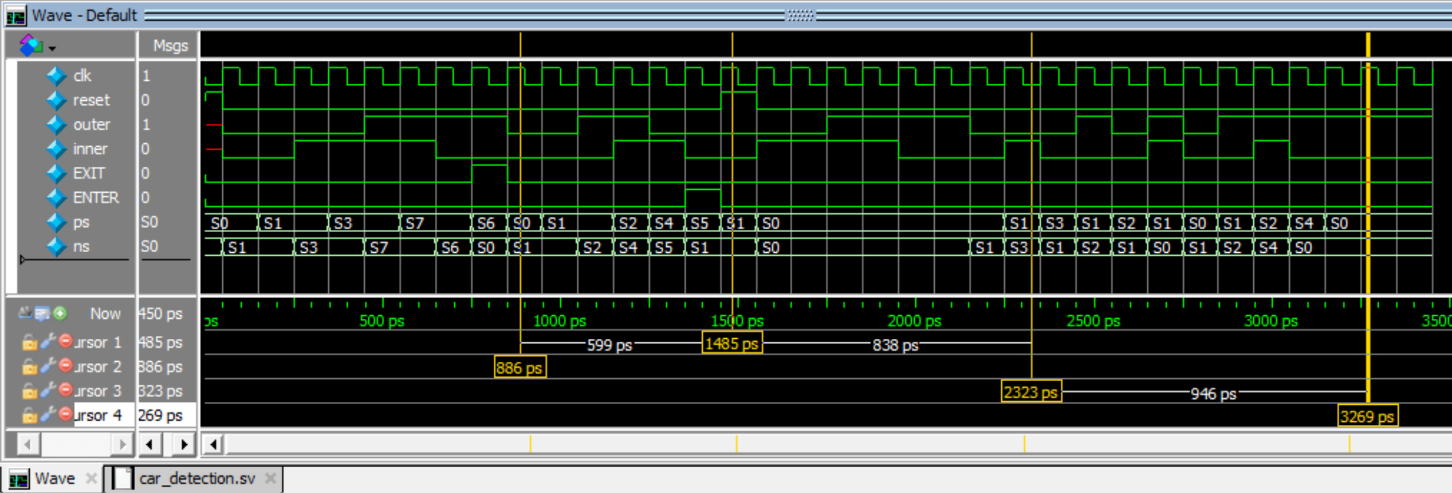


Figure 6: The ModelSim waveform for the car detection module.

**DE1\_SOC**

The goal of the top-level simulation is to verify that the interconnections and instantiations of the various sub-modules are done correctly and accurately. The top-level waveform shown below in Figure 7 clearly demonstrates the changing states (ps) of the car detection module FSM and the changing state of the car counter, reflected by the changing states of the seven-segment displays (HEX0 - HEX5). The input is performed by using the *V\_GPIO* off-board input switches and simulated by using the *V\_GPIO\_in* as provided in the GPIO\_Guide document.

We start the simulation by initializing the intermediate logic signals that will be used to drive the simulated input signals. We then simulate an oscillating clock signal called clk which continues for the entire duration of the simulation. We then initialize the individual pin directions that we’re using in *V\_GPIO*. In our device under test, we’re using *V\_**GPIO[24], V\_**GPIO[29],* and *V\_**GPIO[23]* as inputs for the outer sensor, inner sensor, and reset switch, respectively, and *V\_**GPIO[32], V\_**GPIO[34],* and *V\_**GPIO[35]* as outputs to their corresponding LEDs.

We start our simulated sequence of events by simulating an entering car. This is done by first sending a logic HIGH signal to the outer sensor and holding that value while a logic HIGH signal is sent to the inner sensor, then by sending a logic LOW to the outer sensor and holding it at that value while a logic LOW is sent to the inner sensor. This constitutes the car entering sequence. We then simulate a reset switch press by sending a logic HIGH to the reset and then a Logic LOW. This shows that reset capability is functioning properly. We then simulate the car going in sequence 16 more times, ensuring that all the states in the counter are functional.

Then we simulate an exiting car. This is done by first sending a logic HIGH signal to the inner sensor and holding that value while a logic HIGH signal is sent to the outer sensor, then by sending a logic LOW to the inner sensor and holding it at that value while a logic LOW is sent to the outer sensor. We then simulate the car exiting sequence 16 more times, ensuring that the decrement behavior is working properly. Finally, we simulate an additional sequence which tests for a pedestrian passing through the sensors. This is done by only having the inner and outer sensors at logic LOW then logic HIGH one at a time. The waveform shown below reflects the proper behavior of the top-level module.

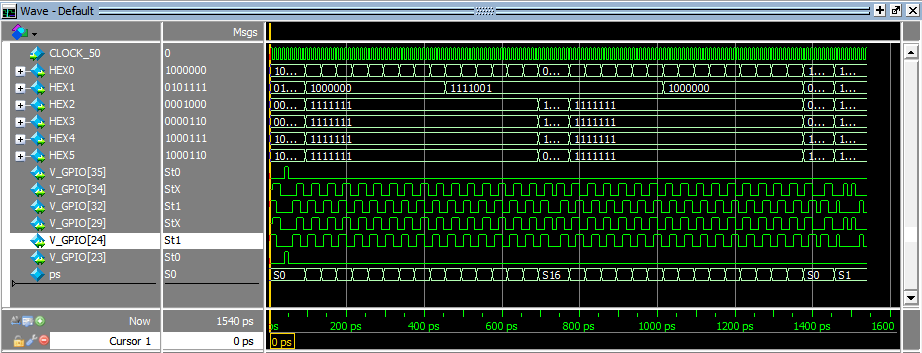


Figure 7: The ModelSim waveform for the DE1\_SoC top-level module.

**Flow Summary**

**A screenshot of a computer

Description automatically generated**

Figure 8: The ModelSim Flow Summary of the compilation of the DE1\_SoC module.

**Experience Report**

Creating the Parking Lot Occupancy system was moderately difficult. We initially encountered issues with implementing the V\_GPIO header correctly to connect off-board switches and LEDs, and the virtual breadboard was very finicky at times. The breadboard and the short allocated testing time meant that it took longer to test our code in real time. Using GitHub for easy collaboration of shared files made code developing and debugging more efficient. It also helped to reference example code using the V\_GPIO header when implementing it into our own code. For the next lab we would like to get a head start and aim to work together live more frequently to reduce debugging time. Additionally, using GitHub’s full functionality for version control may help make our code development process even smoother.

The estimated total time working on this lab was 24 hours broken as follows:

* Reading: 1 hours
* Planning: 1 hours
* Design: 3 hours
* Coding: 11 hours
* Testing: 4 hours
* Debugging: 4 hours.