EE 371 Lab1 Report: Parking Lot

Ziyi Huang

ECE, Embedded System

University of Washington

4/9/2019

The purpose of this lab is to practice finite state machine development. In this lab, a finite state machine for a parking lot is designed and modeled with a DE1-SoC board. The design is written in Verilog, simulated with ModelSim, and synthesized with Quartus. The development procedure, results, and reflections are described in the rest of the report.

Procedure

To build the system, I first designed a state diagram to describe the system and then a block diagram to module out the necessary functionalities. The state diagram and the block diagram are reproduced in Figure 1 and Figure 2, respectively.

I decided to change the count of the cars only when one of the two specific sequence of sensor blocking happened, i.e. 00->10->11->01 for increment and 00->01->11->10 for decrement. This decision resulted in 7 states and 2 separate paths in my finite state machine. I also decided to only transit states when valid inputs were given. But I could only do this because we could assume cars were only able to move in one direction. Overall, this implementation significantly simplified the system, as the routes were mostly linear and the output only changed at 1 edge per route. Another implementation I tried involved 4 states and the edges made cycles within them. It

generated confusing edges and output transitions, which got worse when I tried to synthesize the hardware. It cost me a lot of time debugging.

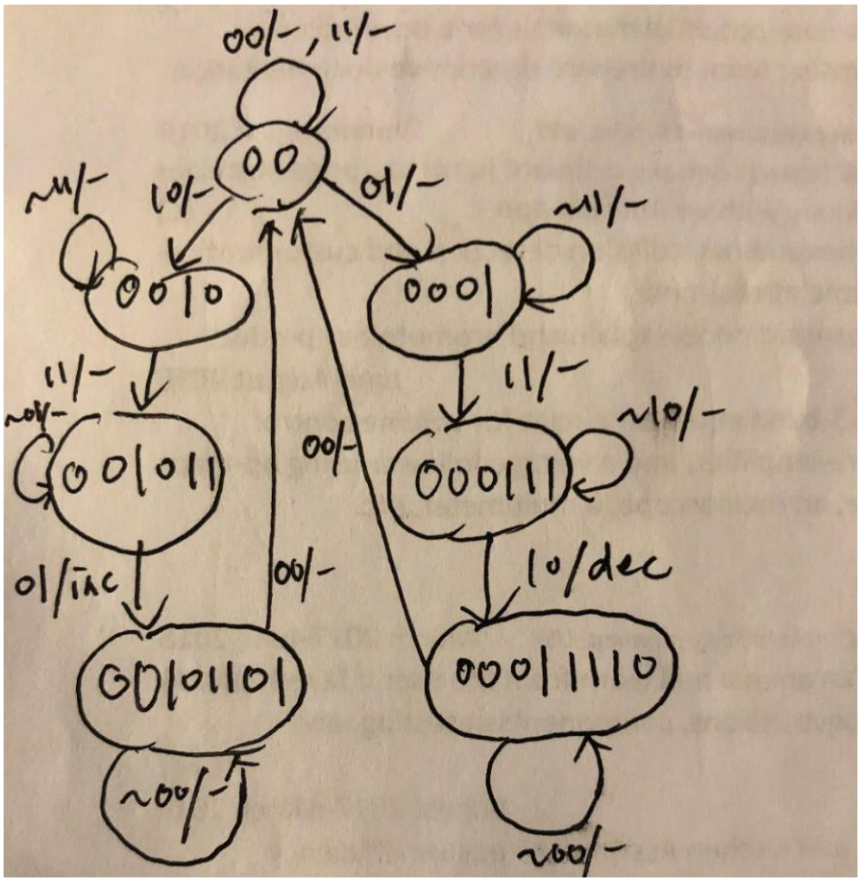

Figure 1: State Diagram for the Parking Lot

The final system consisted of 4 modules – a top-level driver, a finite state machine, a counter, and a display control. The top-level driver translated the hardware elements to symbols in the design so that the codes could be more readable. The rest of the modules handled 1 task each, making the logic flow easy to follow and the codes easy to debug. In addition, since the HEX control had a simply logic but many cases, making it an individual unit reduced the length and complexity in the other two units. In fact, the HEX control unit contains more than 50 lines while the other units contain only

20~30 lines.

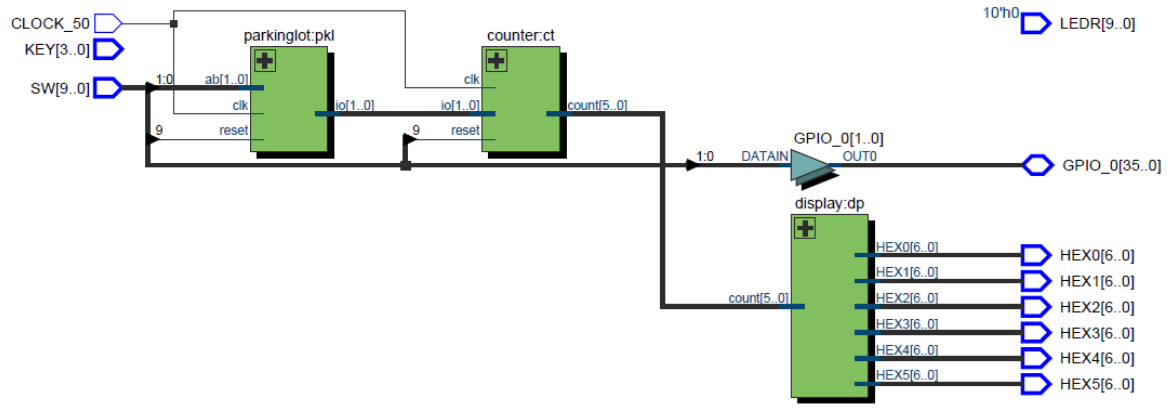

Figure 2: Schematic of the System

Results

The major results produced are simulation waveforms and demo-able hardware. In this section, I will first discuss the simulation waveforms of the top-level driver unit and the counter units then the demo results of the hardware. I would not discuss the finite state machine simulation as it was included in the top-level driver unit.

The simulation of the top-level driver unit is presented in Figure 3. I majorly tested the hardware connections and the state transitions. The *ab* signal in the waveform are actually *SW[1:0]* on the DE1-SoC. I first tested the *enter* route and then the *exit* route to see the count change. For each state, I first supplied all inputs that were expected to not provide state transition and then the input that did. As shown in Figure 3, the states only changed after some stall cycles, which verified that the states

transit correctly. The output control was also verified as it only provided increment or decrement at one edge per route.

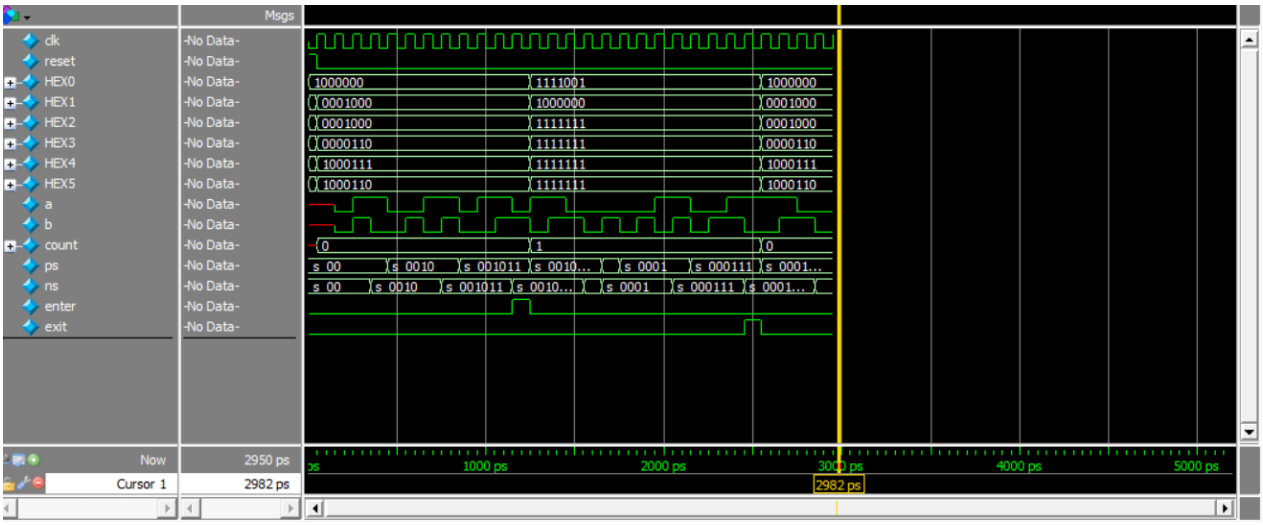

Figure 3: Top-level Driver Simulation Waveform

The simulation for the counter is presented in Figure 4. The goal of this test was to verify that this unit correctly responded to the inc and dec signals, which in this case were equivalent to enter and exit, respectively. Also, it was expected to verify that the count never got lower than 0 or surpassed 25. I supplied the enter and exit signals for 29 and 30 cycles, respectively, so that theoretically the counter increased to 29 and

decreased to -1. The fact that the counter first increased but limited to 25 and then decreased and limited to 0 verified that the counter functioned correctly.

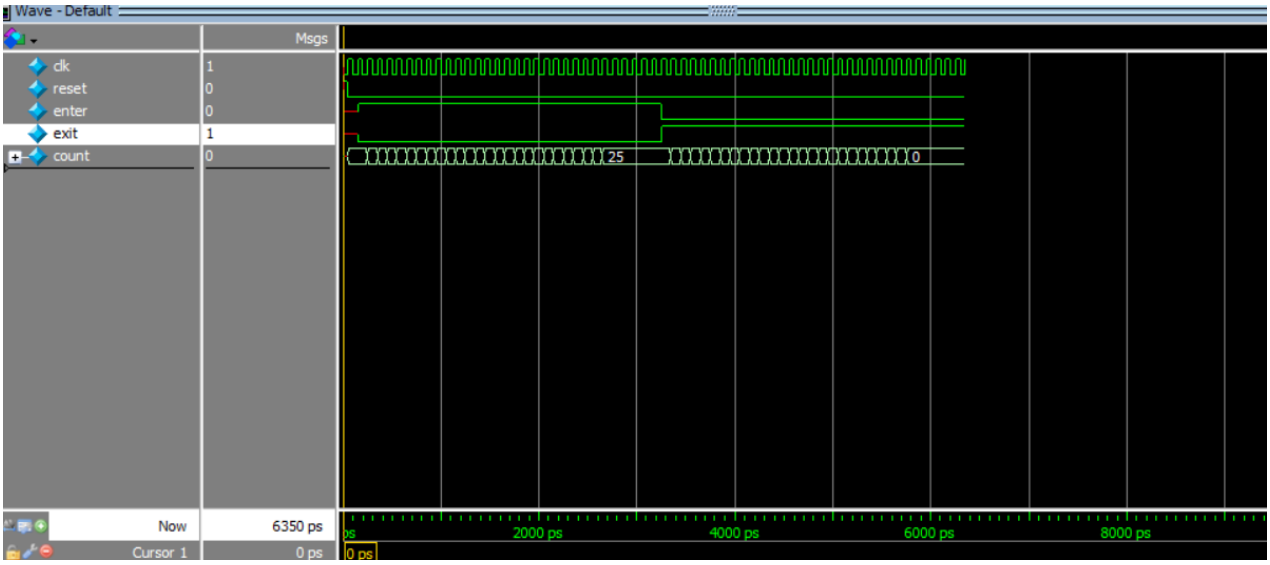

Figure 4: Counter Simulation Waveform

The simulation for the HEX display is presented in Figure 5. The goal of this test was to verify the letters and numbers displays behaved correctly. I supplied a for loop that increased count by 1 each time. The fact that HEX0 changed every increment and the HEX1 changed every 10 increments verified that the number displays behaved correctly. The fact that the higher HEXes displayed things at 0 and 25 while displayed nothing anywhere else verified that the letter displays behaved correctly. To verify that the displays were written correctly, I synthesized a clock divider and switches-controlled enter and exit signals to the hardware. By doing so, I verified that the numbers and letters looked correct.

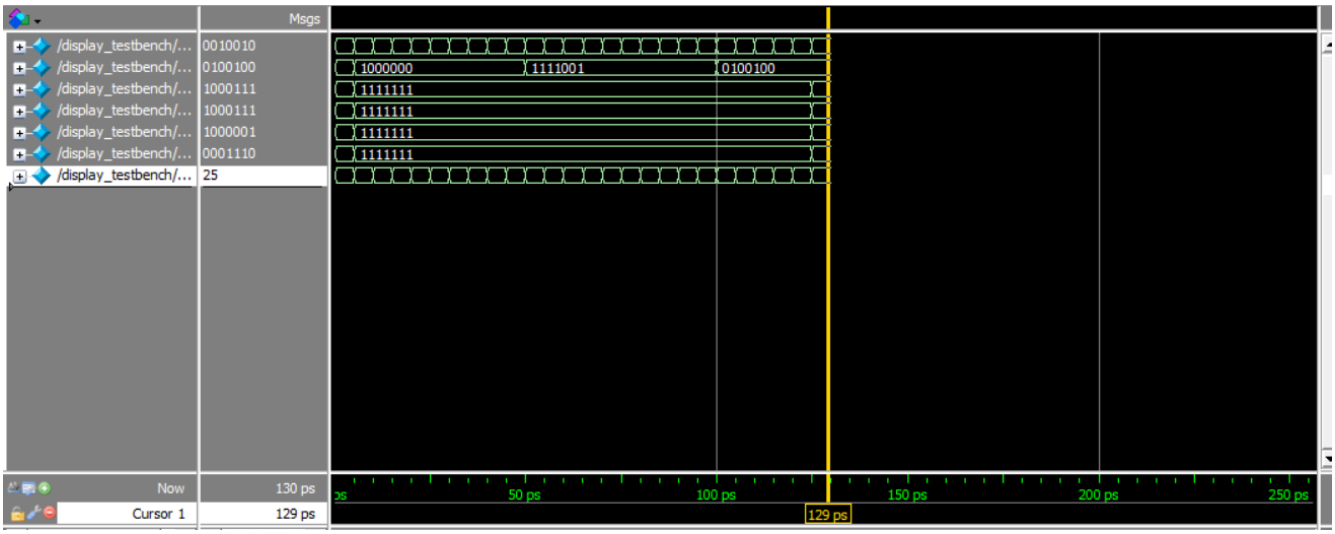

Figure 5: HEX Display Simulation Waveform

Overall, the resulted hardware satisfied the lab requirement. When the values of SW [1:0] followed the sequence 00->10->11->01, the count increased by 1; when the values followed the sequence 00->01->11-10, the count decreased by 1. The count never surpassed 25 while displayed FULL25 at 25 and never got lower than 0 while displayed CLEAR0. When SW[1] equaled 1, representing sensor a blocked, the red off-board LED turned on; when SW[0] equaled 1, representing sensor b blocked, the green off-board LED turned on/

Problem Faced & Feedback

I spent 10 hours in total developing this lab. The time distribution was:

- 5% reading & planning
- 10% designing
- 15% coding
- 50% debugging
- 20% testing

My major difficulty was caused by the confusing 4-state design that I came up with initially. The simulation seemed to be correct but the system broke when synthesized to

the hardware. I spent 7 hours debugging it before I switched to the 7-state design and used 30 minutes to finish the lab. I learn that when a design problem becomes tricky and intriguing, step back and tried a simpler design. Also, simplifying a system is way important than trying to cover all cases. Overall, I like this lab. I only hope the failure experience of this time could help me get through the future labs more easily.