

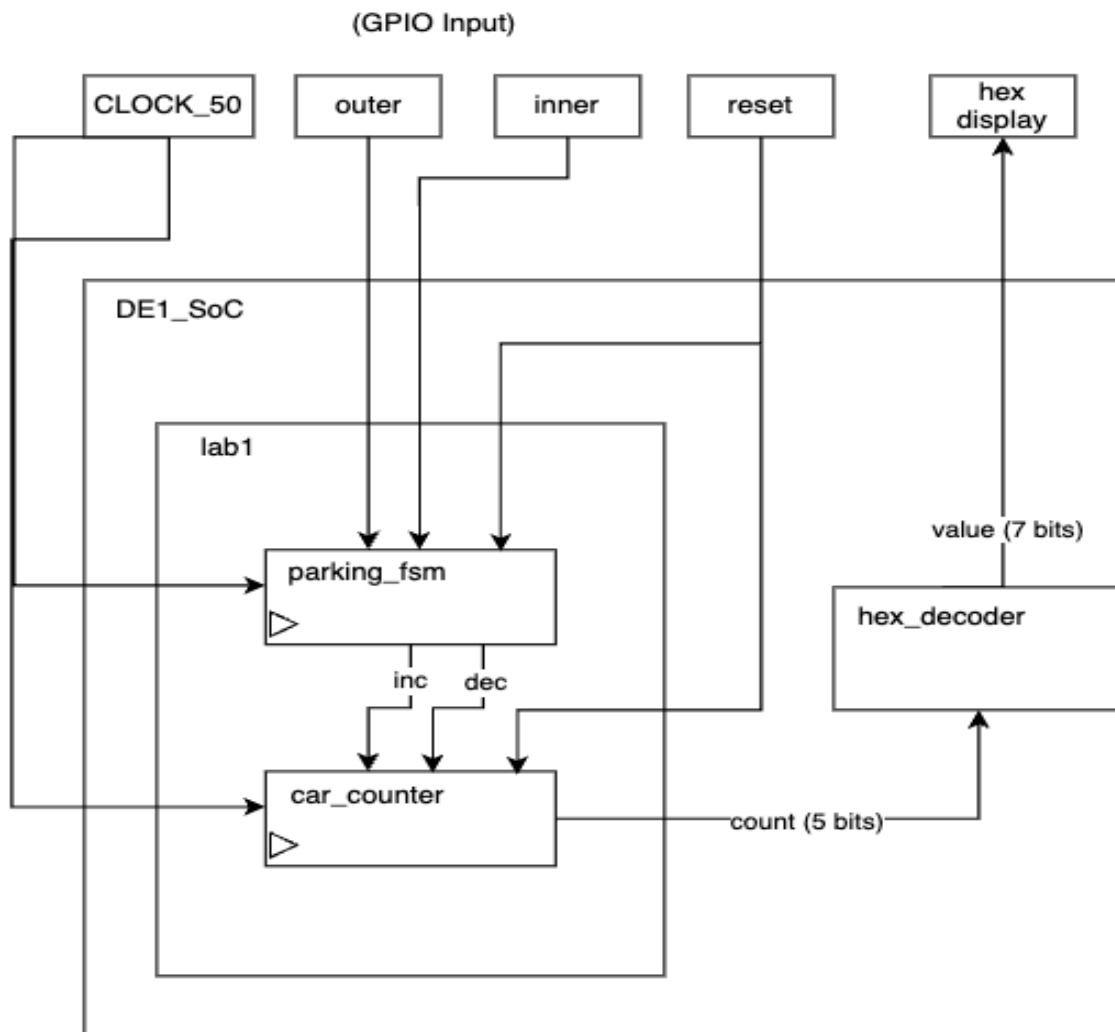
# CSE 371 Lab 1 Report

Elijah Melton, 2164822

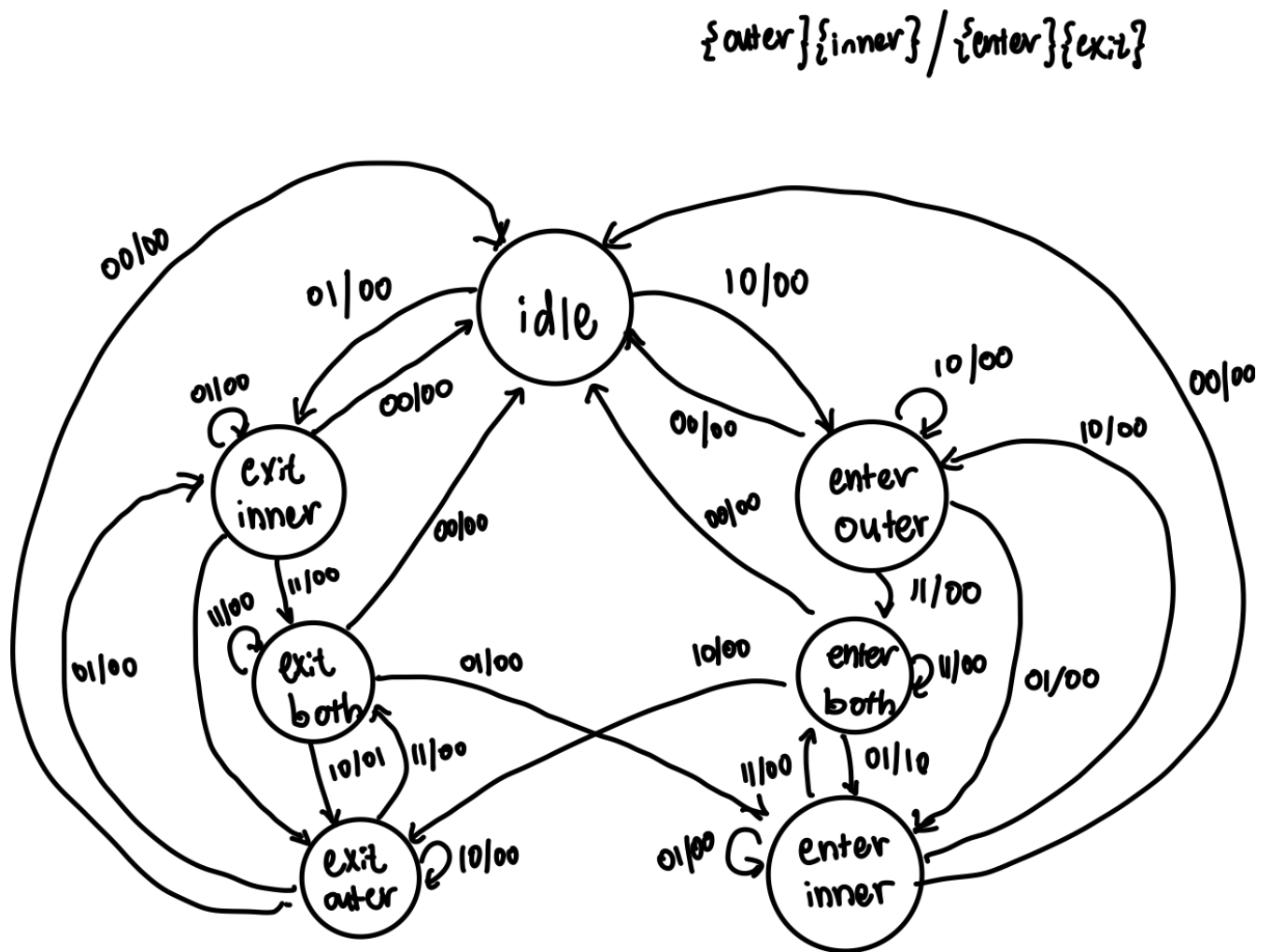
Ankith Tunuguntla, 2234509

## Design Procedure

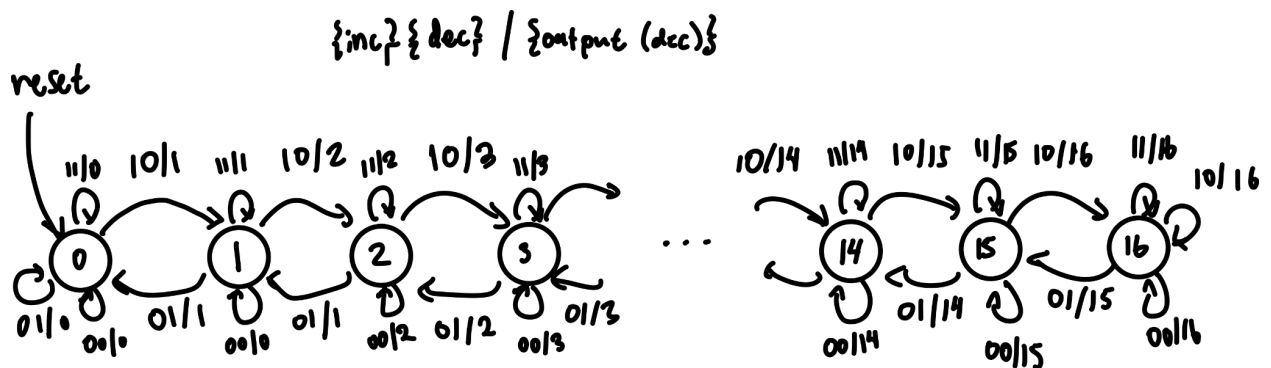
We approached lab 1 by first identifying high level components we needed in our system based on functional requirements. We then implemented the simplest components first, testing them along the way, and then eventually integrating them after we were confident they behaved as intended.



Parking FSM Diagram



Car Counter Diagram



## parking\_fsm

Implements a Mealy FSM that tracks cars entering and exiting the parking lot.

- Uses two sensor inputs (outer and inner)
- Car entry sequence: outer -> both -> inner -> neither
- Car exit sequence: inner -> both -> outer -> neither
- Doesn't count pedestrians, who trigger only one sensor at a time
- Produces enter and exit pulses for a single cycle to trigger counter updates
- Resets to IDLE state when system reset is triggered

## car\_counter

Manages the parking lot's count.

- 5-bit counter supporting counts in range 0 to 16
- Uses two inputs (incr, decr) coming from parking\_fsm
- prevents overflow above 16 and underflow below 0
- synchronous reset to clear the count
- Ignores simultaneous increment and decrement

## hex\_decoder

Converts binary count to input for 7-segment display.

- Takes in 5-bit count (0-16)
- Displays numbers 0-9 on a single display and 10-16 across two displays
- Includes special pattern for "CLEAR" (when count is 0) and "FULL" (when count is 16)
- Uses active-low 7-segment encoding

## lab1

Ties together the application logic modules.

- Intermediate between hardware aware modules and application logic/state management, making testing easy
- Connects the `parking_fsm` to the `car_counter`

## DE1\_SoC

Top-level module handling all hardware interfaces.

- Maps FPGA pins to signals, and then into `lab1` and `hex_decoder`
  - Connects the count output to the `hex_decoder`
  - Connects "photo sensor" inputs to `lab_1`

## Results (TODO)

### Overview

We implemented all of the modules listed in the design section. Most of them were fairly straight forward (e.g. `hex_decoder`, `car_counter`), with the only module containing notable design decisions being `parking_fsm`, since it contains the majority of the logic for the entire system. In particular, we used the following states:

```
enum logic [3:0] {  
    IDLE,           // Both sensors unblocked  
    ENTER_OUTER,    // Outer sensor active, something beginning to enter  
    ENTER_BOTH,     // Both sensors active, car entering  
    ENTER_INNER,    // Inner sensor active, something finishing entering  
    EXIT_INNER,     // Inner sensor active, something beginning to exit  
    EXIT_BOTH,      // Both sensors active, car exiting  
    EXIT_OUTER      // Outer sensor active, something finishing exiting  
} ps, ns;          // Present State and Next State
```

and we conceptualized a car as being one of the exit or enter sequences listed in the design procedure, and all other sequences being explained by pedestrians. Pedestrians can be identified by one of the following sequences:

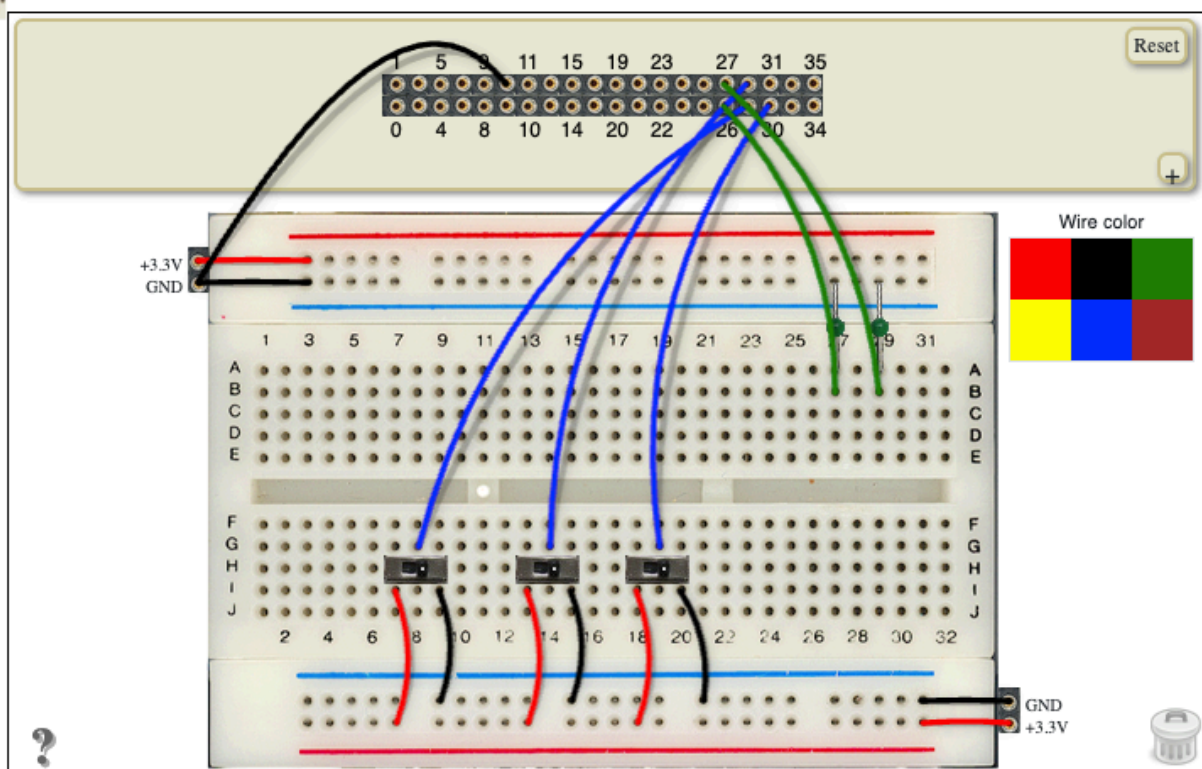
- ENTER\_OUTER -> ENTER\_INNER
- ENTER\_OUTER -> IDLE -> ENTER INNER
- EXIT\_INNER -> EXIT\_OUTER
- EXIT\_INNER -> IDLE -> EXIT\_OUTER

With this being said, we made the design decision to output a `car_enter/car_exit` signal on the transition from `*_BOTH` state, since we could be sure this is a car, and cars aren't able to change direction, so after this state is reached it must be a car either entering or exiting. This means that even though the car is still blocking one of the signals, we are guaranteed that it will continue entering/exiting after this state, so the overall count will be correct.

We used the following GPIO pin mappings:

```
assign V_GPIO[26] = outer;  
assign V_GPIO[27] = inner;  
assign outer = V_GPIO[28];  
assign inner = V_GPIO[29];  
assign reset = V_GPIO[30];
```

And the following breadboard configuration:

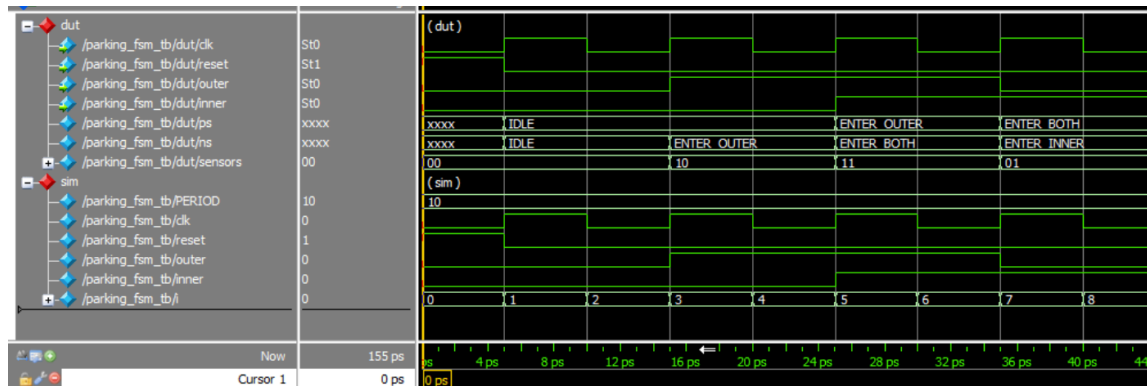
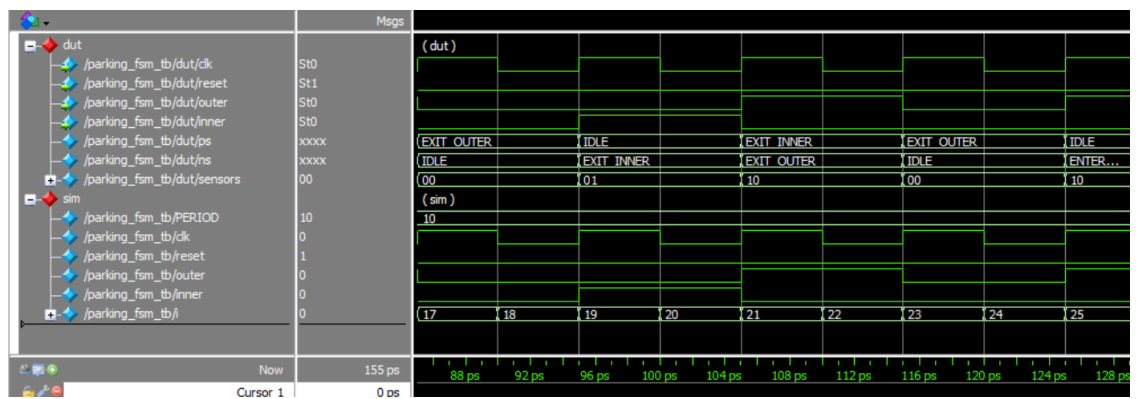
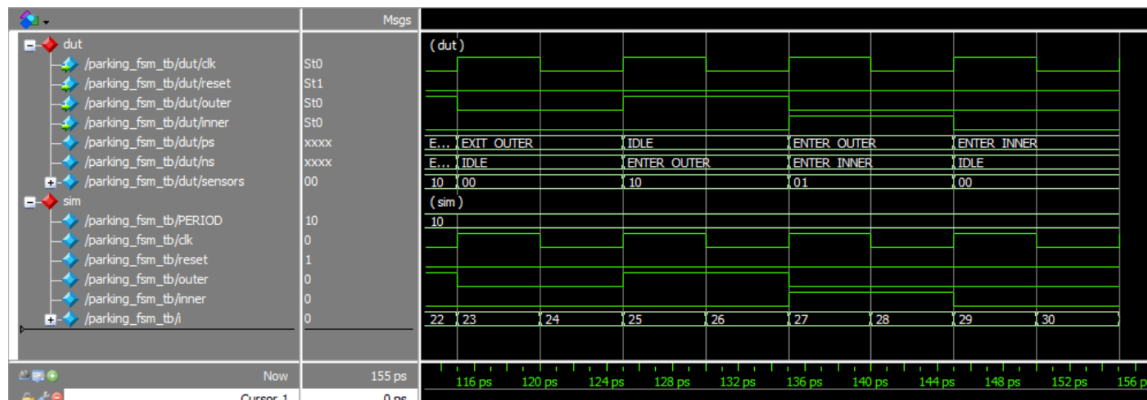
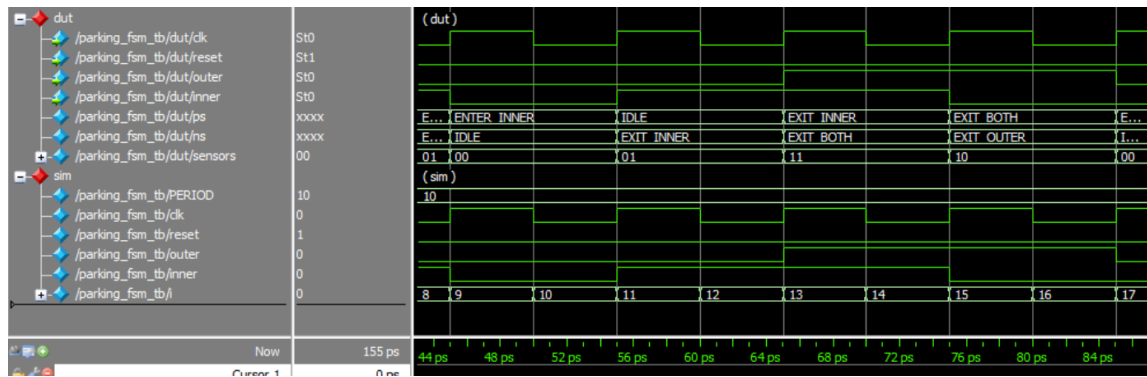


## Parking FSM Simulation

We tested the following transitions, along with reset, and also asserted that the car\_enter and car\_exit signals were active when they should have been

1. IDLE → ENTER\_OUTER (Car enters: outer sensor activated)
2. ENTER\_OUTER → ENTER\_BOTH (Car progressing: both sensors activated)
3. ENTER\_BOTH → ENTER\_INNER (Car progressing: inner sensor only activated)
4. ENTER\_INNER → IDLE (Car fully entered: no sensors activated)
5. IDLE → EXIT\_INNER (Car exits: inner sensor activated)
6. EXIT\_INNER → EXIT\_BOTH (Car progressing: both sensors activated)
7. EXIT\_BOTH → EXIT\_OUTER (Car progressing: outer sensor only activated)
8. EXIT\_OUTER → IDLE (Car fully exited: no sensors activated)
9. IDLE → EXIT\_INNER (Pedestrian exits: inner sensor activated)
10. EXIT\_INNER → EXIT\_OUTER (Pedestrian exiting: outer sensor activated, skipping EXIT\_BOTH)
11. EXIT\_OUTER → IDLE (Pedestrian fully exited: no sensors activated)
12. IDLE → ENTER\_OUTER (Pedestrian enters: outer sensor activated)
13. ENTER\_OUTER → ENTER\_INNER (Pedestrian entering: inner sensor activated, skipping ENTER\_BOTH)
14. ENTER\_INNER → IDLE (Pedestrian fully entered: no sensors activated)

Below, you can see our simulation waves, which correspond directly to the above transitions



## Hex Decoder

Additionally, we tested `hex_decoder` separately to verify we got the expected outputs for every value passed in. We've included this in the report (despite it not being listed in the rubric) because we chose to leave it out of our `lab_1` module, since it is hardware specific. With this approach, we were able to verify all modules worked so that we could be confident they would function properly when integrated together.

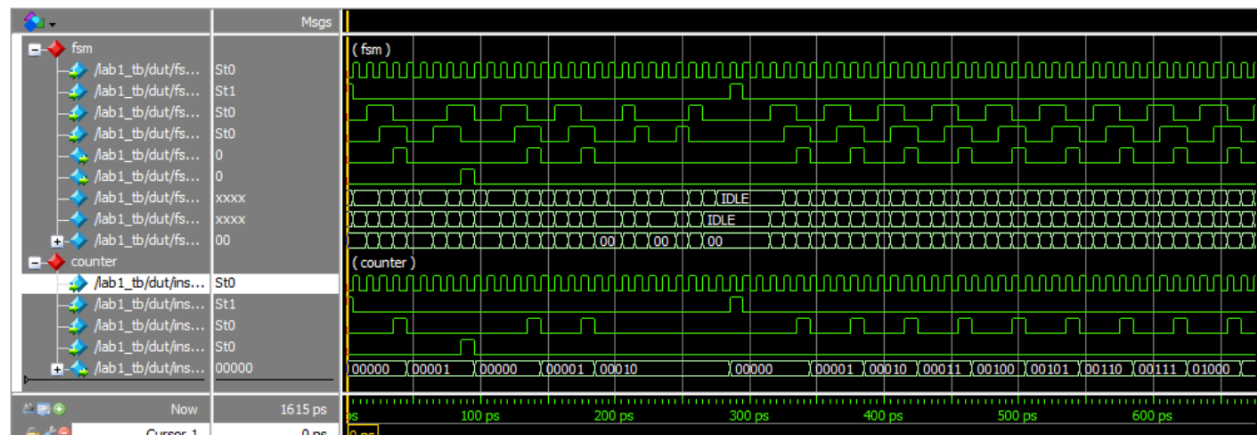
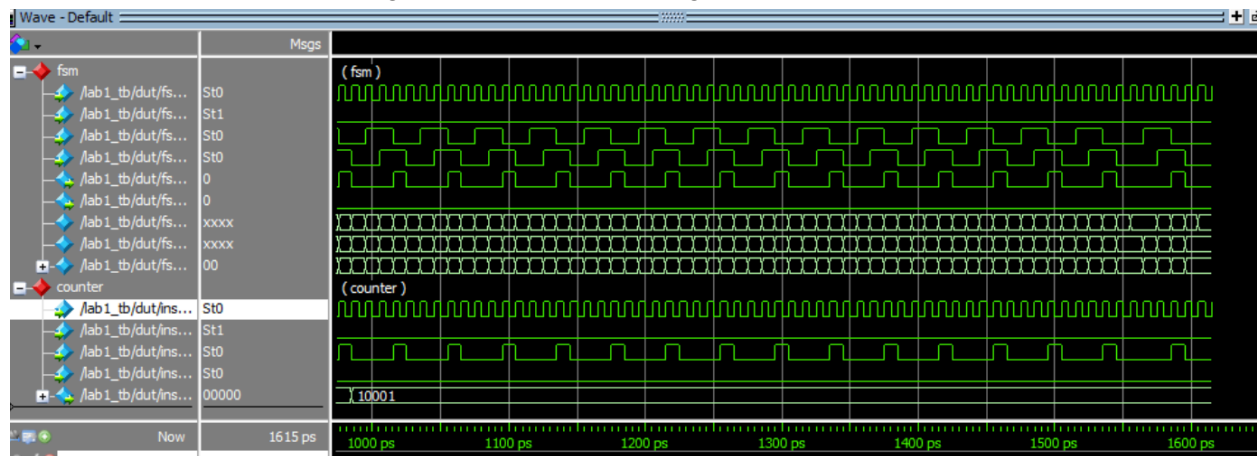


## Lab 1 Simulation

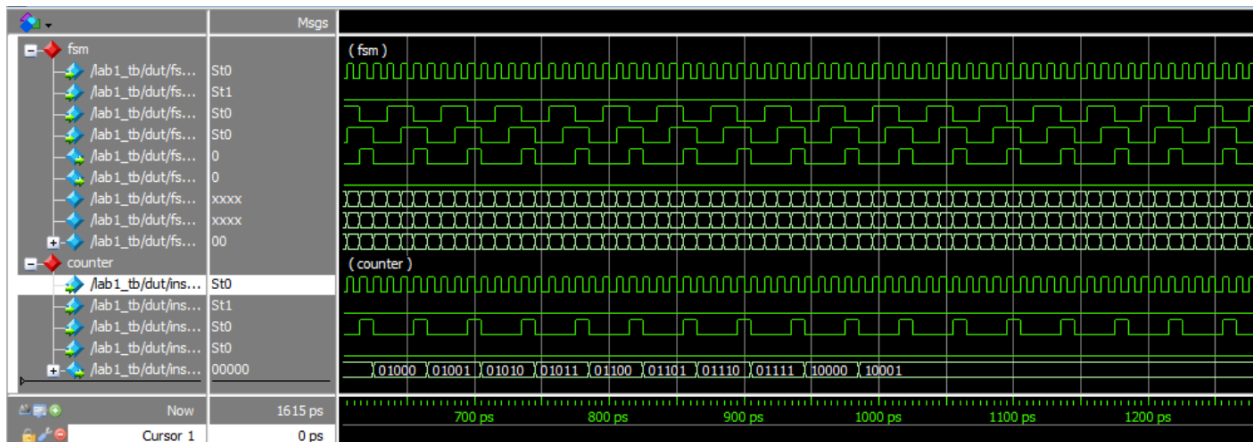
We tested the exact same transitions here as we did the `parking_fms_tb`

1. IDLE → ENTER\_OUTER (Car enters: outer sensor activated)
2. ENTER\_OUTER → ENTER\_BOTH (Car progressing: both sensors activated)
3. ENTER\_BOTH → ENTER\_INNER (Car progressing: inner sensor only activated)
4. ENTER\_INNER → IDLE (Car fully entered: no sensors activated)
5. IDLE → EXIT\_INNER (Car exits: inner sensor activated)
6. EXIT\_INNER → EXIT\_BOTH (Car progressing: both sensors activated)
7. EXIT\_BOTH → EXIT\_OUTER (Car progressing: outer sensor only activated)
8. EXIT\_OUTER → IDLE (Car fully exited: no sensors activated)
9. IDLE → EXIT\_INNER (Pedestrian exits: inner sensor activated)
10. EXIT\_INNER → EXIT\_OUTER (Pedestrian exiting: outer sensor activated, skipping EXIT\_BOTH)
11. EXIT\_OUTER → IDLE (Pedestrian fully exited: no sensors activated)
12. IDLE → ENTER\_OUTER (Pedestrian enters: outer sensor activated)
13. ENTER\_OUTER → ENTER\_INNER (Pedestrian entering: inner sensor activated, skipping ENTER\_BOTH)
14. ENTER\_INNER → IDLE (Pedestrian fully entered: no sensors activated)

With the main difference being we verified counter logic.

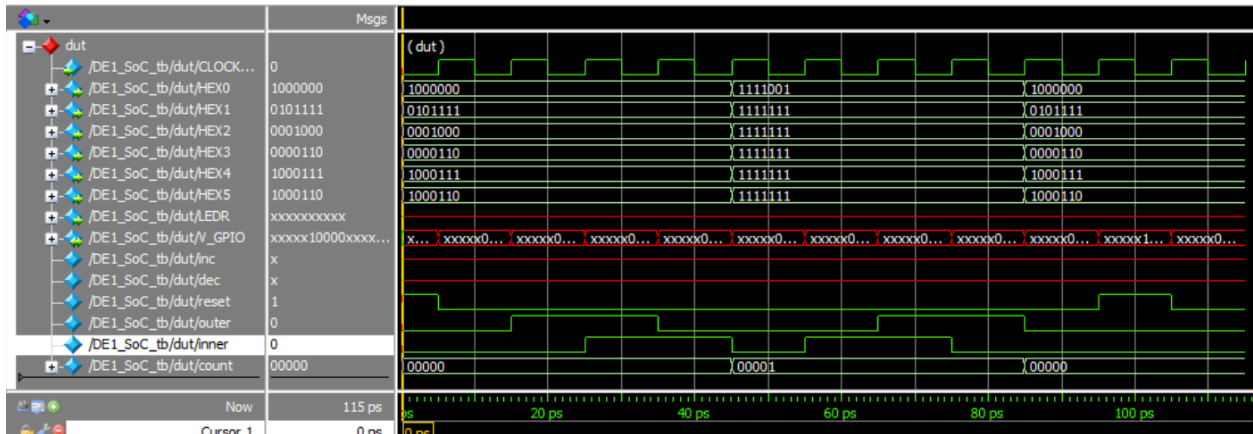







## Top Level Simulation

Since we already verified all functionality in other modules, our actual top level module's testbench was very simple. We simply needed to verify that all the ports were properly connected, particularly that the GPIO inputs were correctly being read, and that the count was correctly being routed to the `hex_decoder`. As you can see in the below simulation, we go through the car enter and exit sequence, and the hex displays change appropriately.



## Flow Summary

Flow Summary	
 <<Filter>>	
Flow Status	Successful - Fri Apr 11 18:51:21 2025
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Editio
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	18 / 32,070 ( < 1 % )
Total registers	8
Total pins	89 / 457 ( 19 % )
Total virtual pins	0
Total block memory bits	0 / 4,065,280 ( 0 % )
Total DSP Blocks	0 / 87 ( 0 % )
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 ( 0 % )
Total DLLs	0 / 4 ( 0 % )
Activate Windows Go to Settings to activate Windows.	

# Experience Report

## Feedback

This lab was a nice review of basic system verilog concepts. The only feedback we have is that it was a little difficult to understand how to hook up the GPIO IO pins, and some additional documentation/resources for understanding *what* as well as *why* would be useful

## Significant Issues

I did not encounter any significant issues in this lab

## Tips/Tricks

- You don't need to re-synthesize every time you want to run static analysis
- assertions and display are very useful for writing test benches/verifying correctness
- Start small and test as you go

## Time Spent

~13 hours