Part 1: users manual

Operation:

a. Turning on the board:
   First power on the board and start the program, in the initial state you will see two green lines on the top and bottom rows with nothing in between and no red light representing the frog.

   In order to start the game you will need to pick an appropriate speed by setting the switches 7:0 to represent a binary number that determines the slowness. I.e. a larger binary number makes the cars slower. I recommend only having switch 7 high to start, the slowest possible setting is with all the switches 0-7 high and the fastest possible is the onboard clock speed achieved by having all switches low. (many very fast speeds are not practical for gaming purposes, the lowest slowness looks just like a solid green wall, no one has ever beat the game without at least switch 6 high)

   If you want to adjust the settings mid game refer to part c, mid game tampering can cause unexpected behavior.

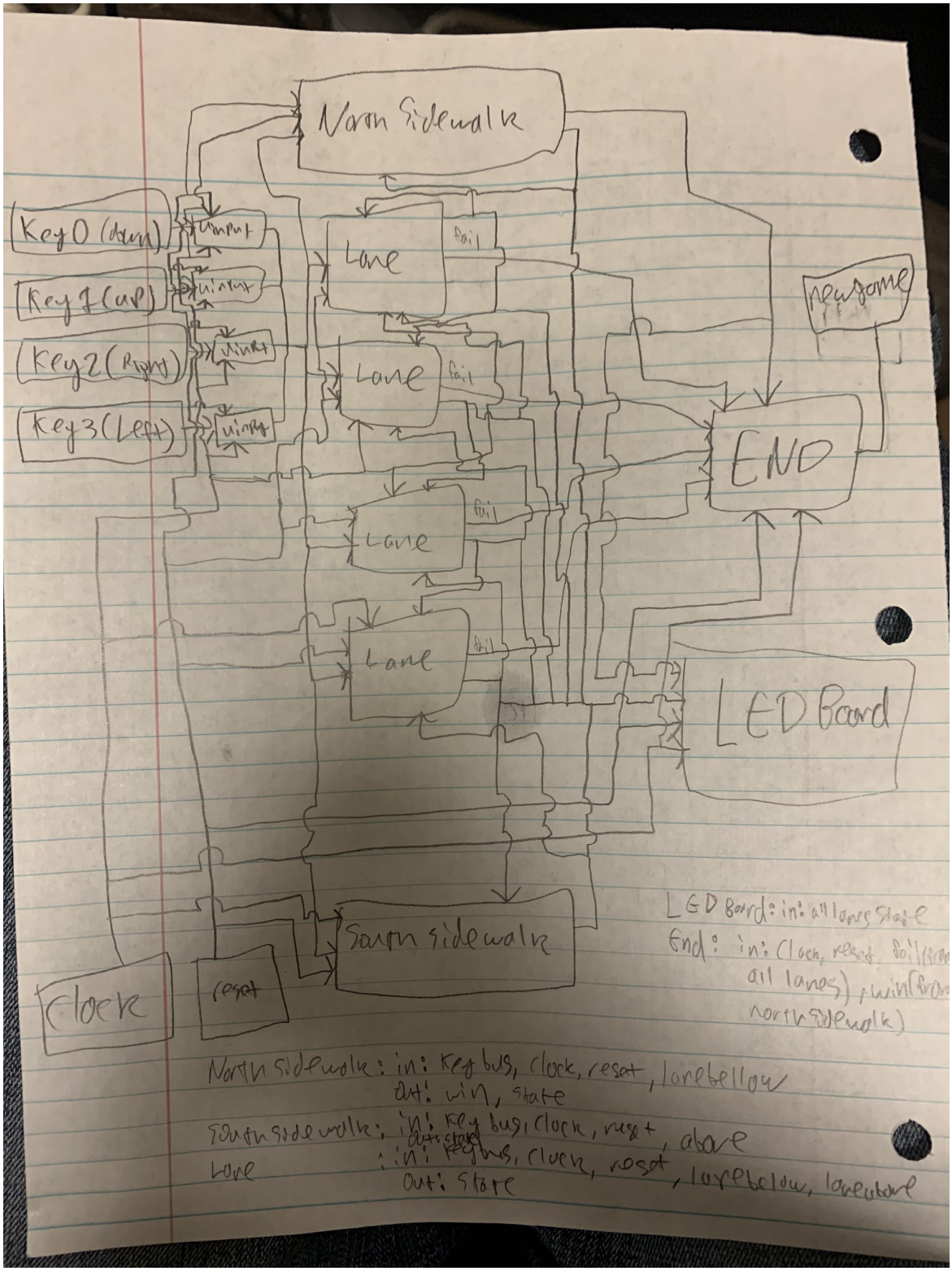
   Now that switches 0-7 are set flick switch 9 up, you should see green stationary dots appear on the board along with one red dot for the frog, the game begins when you lower switch 9. Switch 8 is a separate reset for the led board, it is not normally used, if switch 8 is high the board will turn off, but the game will still progress.

b. Playing the game
   Keys 0-3 control the frogs movement Key 3 moves left, key 2 moves right, key 1 moves Up and key 0 moves back. If the frog collides with any of the green cars then it becomes Roadkill and the game will display the message “dead” a new game can be start by Pressing a button. If the frog hops onto the north sidewalk the game is won and the message “pass” is displayed.

   There are three portions of the level to complete divided by rest portions on lane4 and lane9 (0 based lane indexing), the last portion is designed to force the player to move horizontally with the cars instead of just waiting for a gap in the cars to line up vertically.

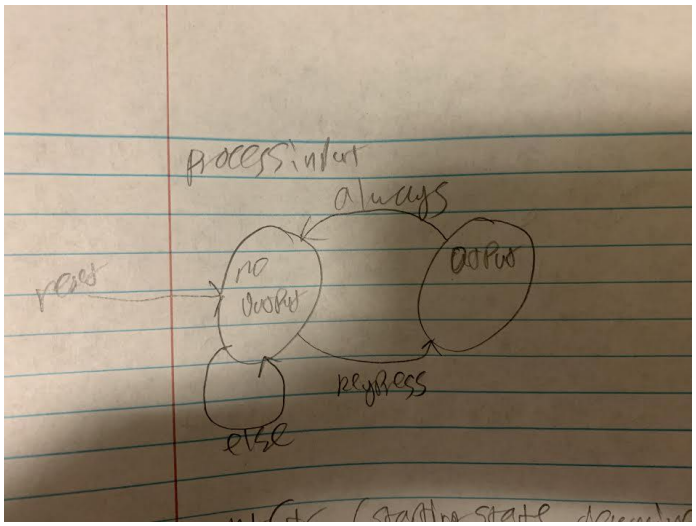
c. Adjusting the settings
   If you want to adjust the speed part way through the game, slowing the game down should be possible though sometimes leads to unexpected behavior like cars disappearing. Speeding the game up is unsafe and may cause the game to cease updating, though the frog will still be able to move. To avoid these issues it is recommended to flip switch 9 up before adjusting the speed, and then flick switch 9 back down to start a game with the new settings.

Block diagram:
(the new game output of the end module is also an input to all of the lanes, but it would be too cluttered to draw)



Module descriptions:

The processInput modules are similar to the ones used in the tug of war lab that allow us to only take one input per button press.
State Diagram:

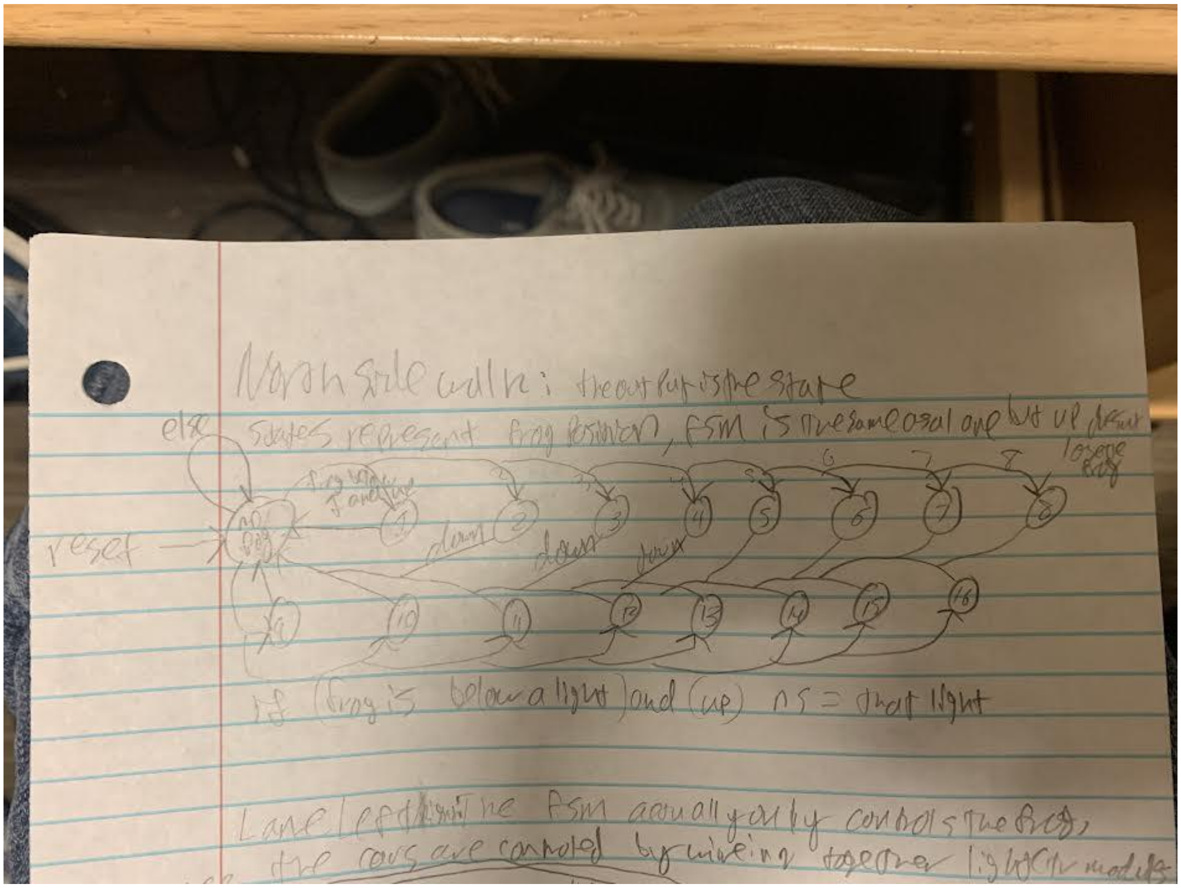


The LEDDriver module is starter code given to us, I do not claim to understand what black magic it employs

The LEDArray module above called the LEDBoard takes all the rows of output from the state of the lights in each lane and puts them into a matrix for the LEDDriver to use.

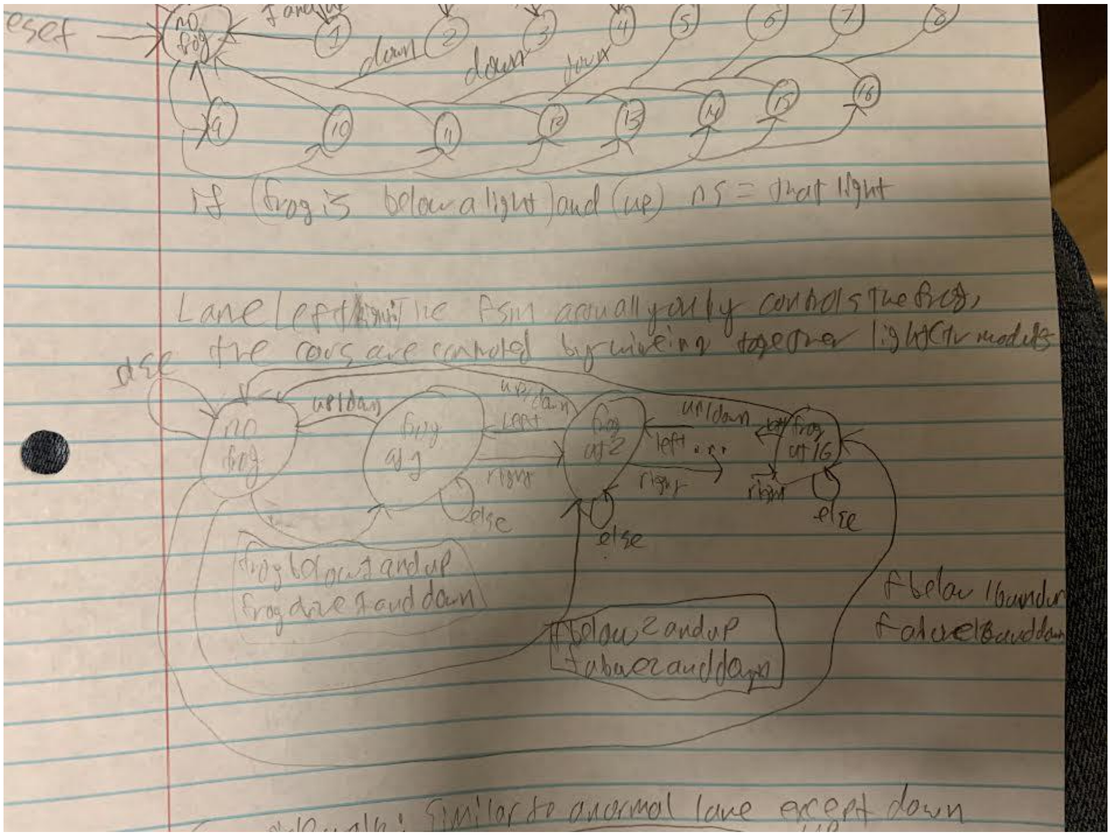
The clock_divider is just stock code for a simple clock divider, not much to say here except it allows us to get slower clocks if we want them.

The northSideWalk module controls the lighting for the top line its fsm just uses the 16 bit binary number representing the red lights as its state and whenever this state is not equal to zero it means a frog made it into the sidewalk and the game is won.
State Diagram:



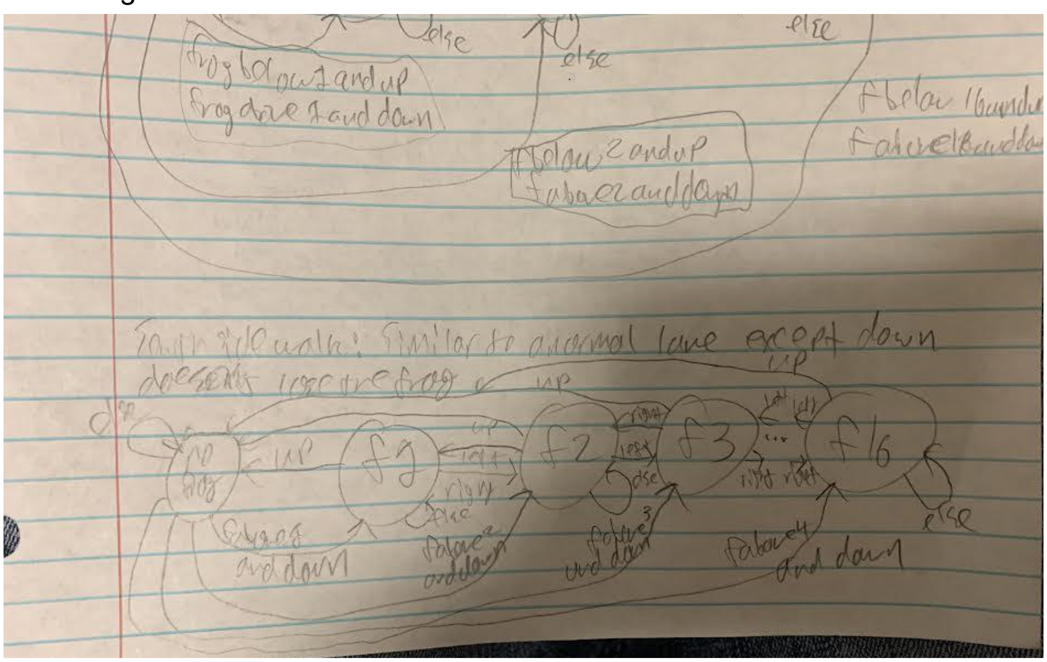
The laneLeft and laneRight modules are nearly identical except the lights update in opposite directions. They control the logic for the present state of the lights in a lane. The possible states represent the positions of the frog on the row, if there is a frog and a car in the same position a lose signal is outputted.

State Diagram:

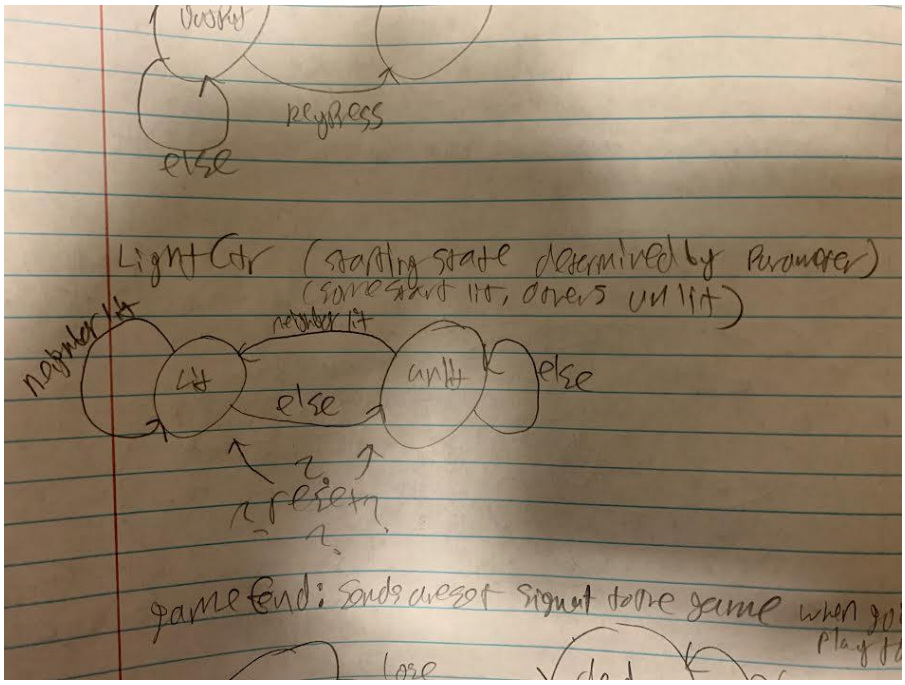


The southSideWalk module is similar to the north, it controls the frog position in its module and not much else, it is simpler because it doesnt need to output any win or lose signals

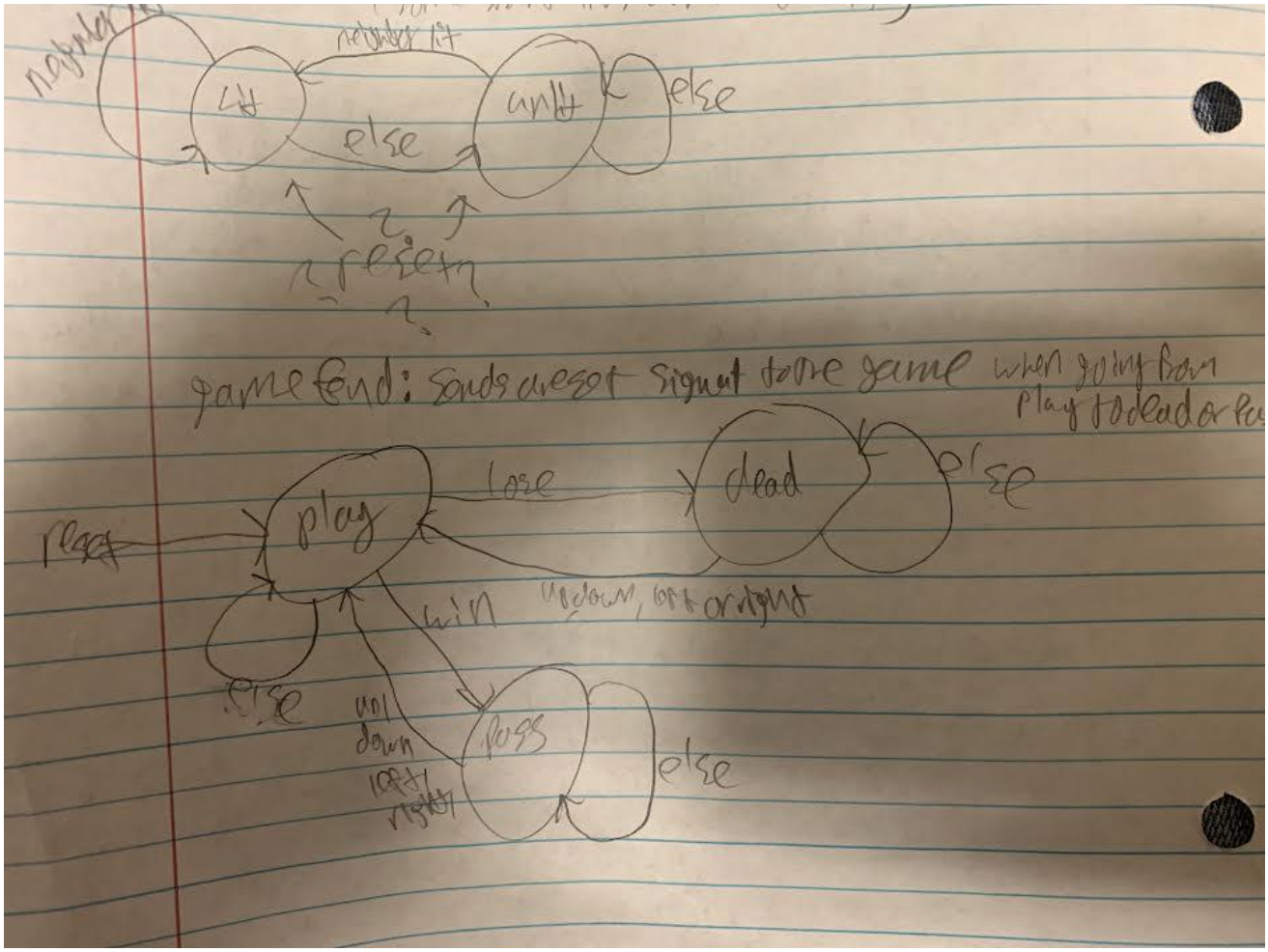
State Diagram:



The lightCtr module controls whether an individual light should be green, it is wired up so that if the light before it is lit then it outputs lit on the next clock cycle, all of the lights are wired in lines so that they loop around the edge of the board.
State Diagram:



The gameEnd module controls the logic for when the board should be reset and what it should output to the hex display. It only has 3 states, one for each possible hex output, dead, pass, or play.
State Diagram:



Testing:
The key was being able to break the work up into simple modules that it was easy to figure out the expected behavior of. Once I was sure that my lightCtr module worked properly and only stayed lit for one clock cycle after the input was lit, it was easy to link 16 of these together in a line to control the cars in a lane, the model sim test benches were crucial at this step of the design process because I couldn't yet see lights on a screen, once I got to individual lanes however I was able to debug mostly through interacting with the board, after lots of thought I finally realized I would need two separate modules for left moving and right moving lanes, I tried really hard to get one generic module that would take an input for whether it was a left moving or right moving lane, but I gave up since this way is simpler.

Part 2: Hours spent: approximately 15 hours total including goofing off with the game.