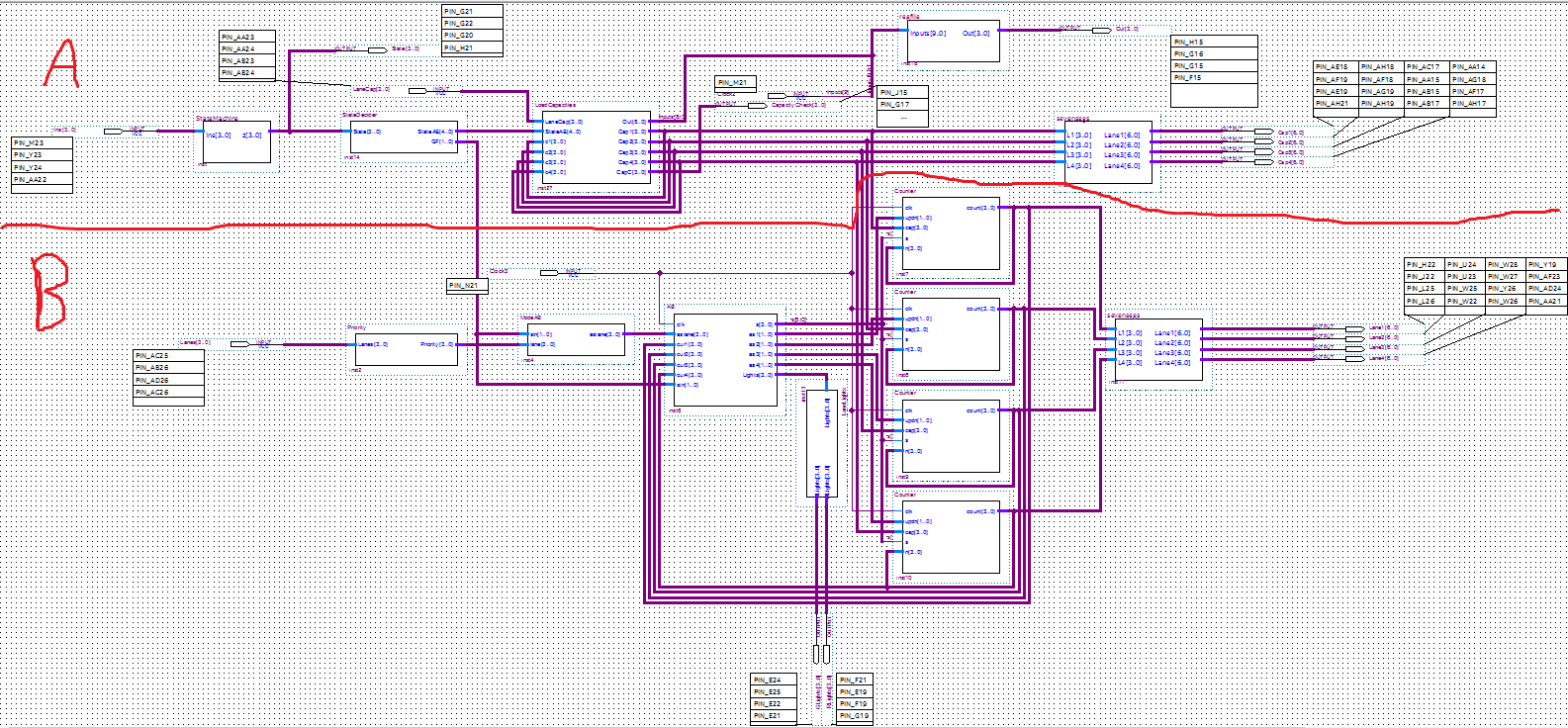
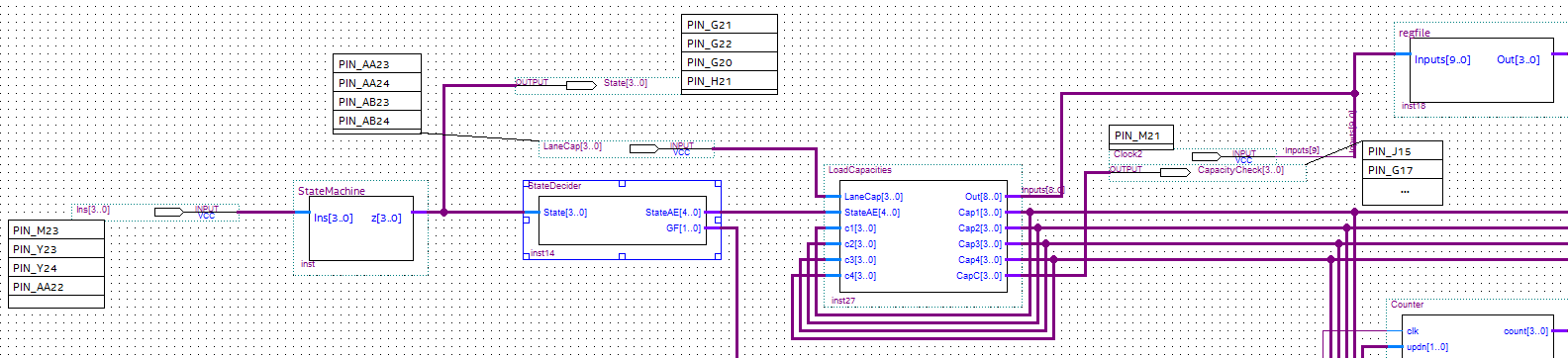
Final Project Report

For my final project I chose to do option 3 which was the traffic lane fairness system. Included in this report is a picture of what the completed circuit looks like and also an in-depth explanation of each block and what they do to make the circuit function correctly.

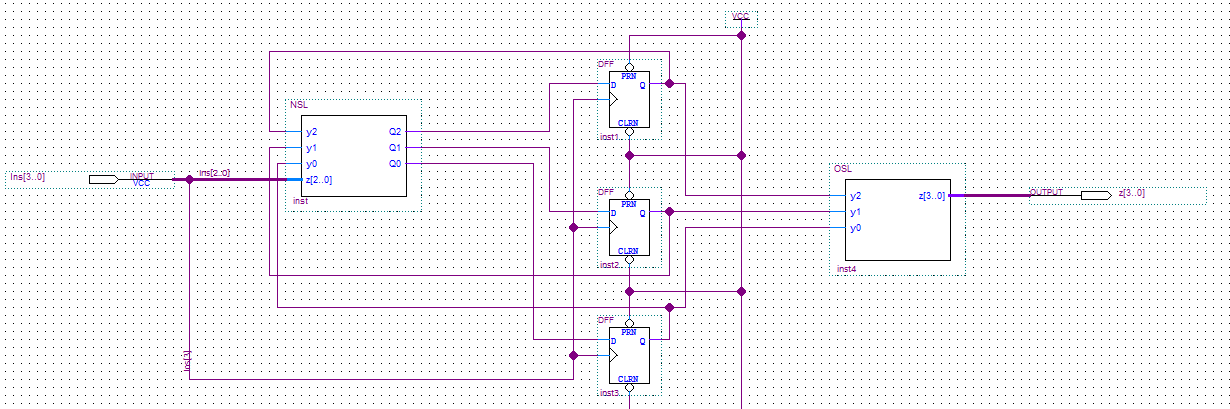
**** **Top Level View:**

This is what the whole circuit looks like when it is all put together. Due to the size of the circuit, I have split it into two halves, A and B, and I will go over each aspect of each half.

**** **Part A**

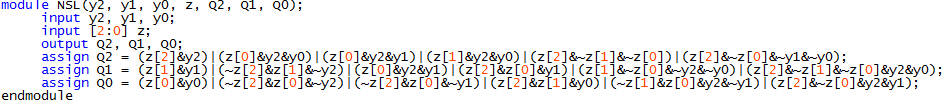
This half of the circuit includes the State machine, a block that decides what State the program is in, a block that detects that State and then depending on the State, loads in the capacities for the specified lane.

**State Machine**

****

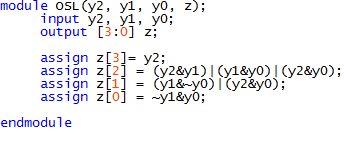
In the Part A snippet, you will see that the far-left block is called “StateMachine”. This picture up above is what that block is. What this State machine does is it reads in the values from the input switches and then goes into the block called “NSL”, or next State logic, which also reads the data being outputted by the flip flops (a.k.a. the current State). The NSL block then outputs three variables, Q2, Q1, and Q0. These are your next State variables. Then when you press the button for the clock, that value is then loaded into your flip flops and that becomes your current State. Those outputs also go into the block called “OSL”, or out State logic, which takes in the values outputted by the flip flops and then generates the output of the State you are currently in.

**NSL**

****

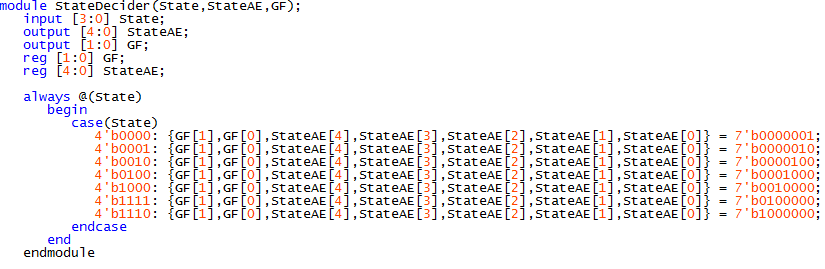
This is the code that is in the NSL block. Like I Stated earlier, it takes in the input from the toggle switches, z, and the current output from the flip flops, y2, y1, and y0, and then outputs what the next State is based on that next State variable’s Boolean expression.

**OSL**

****

This is the code that is in the OSL block. Like I Stated earlier, it takes the inputs currently coming out of the flip flops, y2, y1, and y0, and then outputs what the State is depending on that output variable’s Boolean expression.

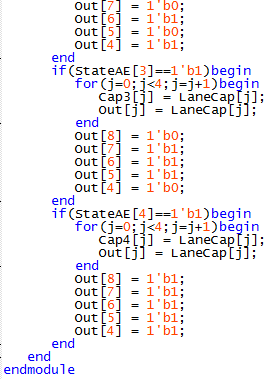
**State Decider**



Now to the block to the right of the State machine block which is called “StateDecider”. What this block does is it checks the output from the State machine, which is called State, and then sets StateAE and GF accordingly. The reason that the outputs are one hot encoded is because this makes it a lot easier to check what State it’s in. For example, when the current State is A, the State machine will output “0000”. This block then recognizes that and sets GF to 00, because it is not State G or F. The order of StateAE goes EDCBA, so when it recognizes the 0000 output from the State machine it sets StateAE to 00001, which means that the program is currently in State A. StateAE is then used in the next block called “LoadCapacities” because States B-E are for loading capacities and lane A separates States B-E and F-G (will show later). GF is then used in the ModeAB block that is in part B of this report.

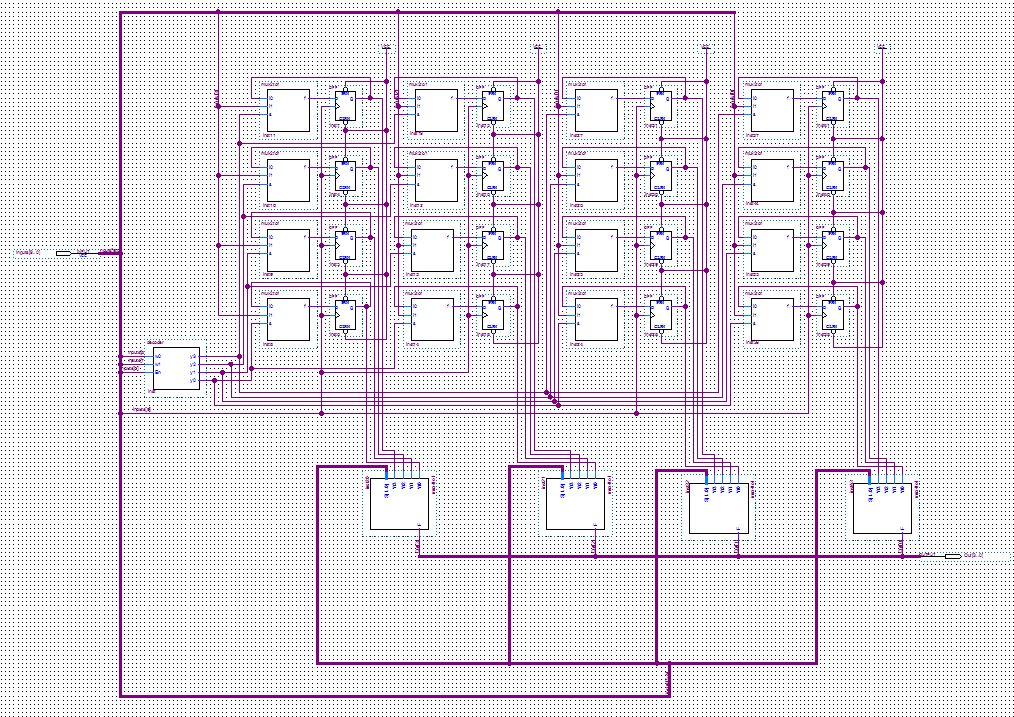
**Load Capacities**





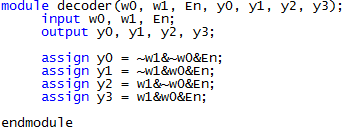
In the next block of part A called “LoadCapacities”, it first checks to see if any of the capacities for any lane is currently zero. If a lane does have a current capacity of zero, then a one is assigned to the appropriate spot in CapC, which means capacity check. For example, if lanes one and two had a capacity of zero, then CapC would become 0011. This outputs 0011 to four read lights which will light up telling the user that they need to add a capacity for lanes one and two. This is done by taking the outputs from the capacity lanes and plugging them back into the block. Then it begins checking StateAE, which we learned from the StateDecider block is an array that tells us what State we are in. If the zeroth element of StateAE is one, that means we are in State A and that means that we do nothing. Therefore, we set every bit in the Out array to 0 because the Out array goes directly to the register file and if every bit is zero, mainly the enable and writing bits, then nothing is changed in the register file. If it is not in State A, then we have to add the value being read in from LaneCap, which is the given capacity given by the user by using four toggle switches, into the capacity array for the correct lane. Also, you have to write this value to the correct spot in the register file and I also set the read values to the same value to make sure it is added correctly, and the value read is outputted on lights. If you are in Mode B, you are writing to the first lane so you set the writing bits to 00 and you use the array cap1, 01 and cap2 for State C, 10 and cap3 for State D, and 11 and cap4 for State E.

**Register File**

****

This is the register file, it takes a ten-bit input, four for the new input capacity, two for the writing bits, one for the enable, one for the clock, and two for the read bits. In the register file is a 2-4 decoder. What this does is it take the 2-bit write value and deicides which register to write to.

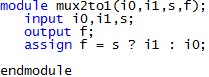
**Decoder**



What this code does is it reads in the bits for the write values and the enable. The output values, y3, y2, y1, and y0 are one hot encoded. Each of those values then controls a specific register, y3 is for the lane 4 register, y2 for lane3, y1 for lane 2, and y0 for lane 1. If the write bits were 00, the output would be 0001, which means you are writing to the first register. The reason you AND every output with the En input is because if En is 0, then that means you are not changing any of the registers.

The output from the decoder then goes into the 2-1 multiplexers which then decide if you are going to keep the value in the register the same or change it to the input values.

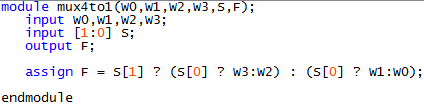
**2-1 Multiplexer**

****

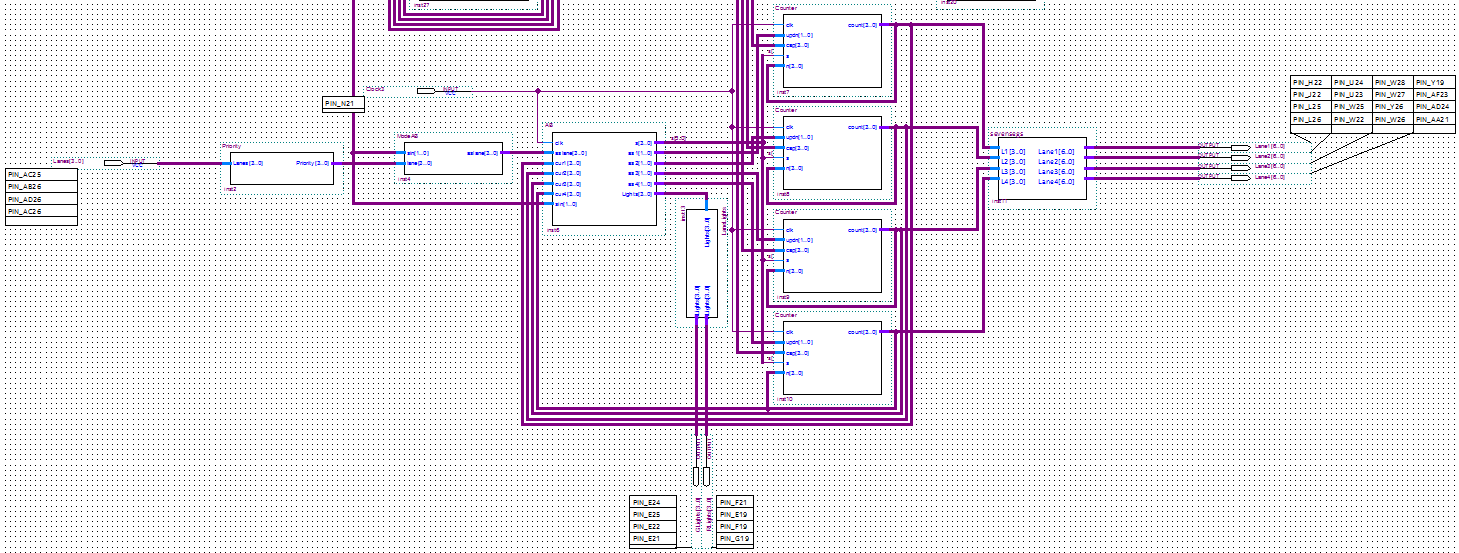
What this code does is it reads in the two values i0 and i1. i0 is the bit currently being outputted by the flip flop and i1 is the new value. s is the value that would is outputted from the decoder. If that value is one, it selects the new bit to be put in the flip flop, otherwise it keeps the same bit.

If you want to be able to tell what the capacity of each lane is you simply just have to read the value that you want and the 4-1 multiplexers output the value of that lane.

**4-1 Multiplexer**

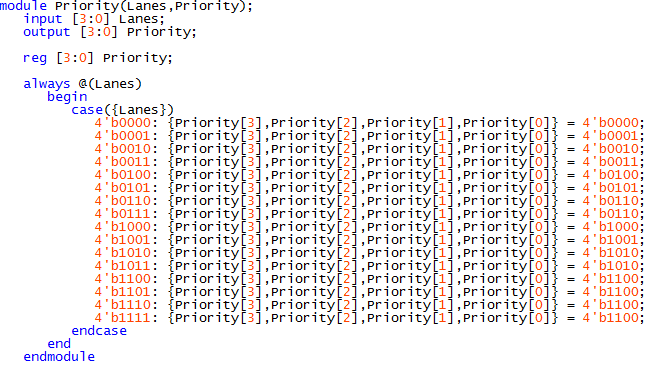
****

There are four of these 4-1 multiplexer blocks. Each will output one bit from one lane. So, w0 is the value from lane 1, w1 is the value from lane 2, w2 is the value from lane 3, and w3 is the value from lane 4. This code checks to see which lane to read from, which is done by the array S. If S is 10, then that means you are checking the side that is between w3 and w2, and since the first element of S is zero, that means you are reading from lane 3, which matches what I said above in the paragraph. The four one-bit outputs from these four 4-1 multiplexers are then bused into one single output and then displayed by red lights and also by the seven segment displays (the seven segment displays part will be shown in part B.

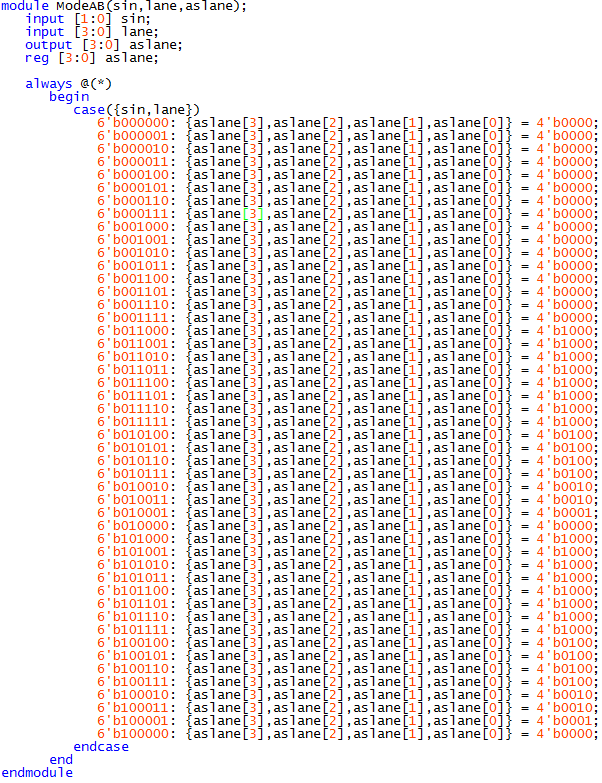
 **Part B**

This is the second half of the circuit. This half is just for Modes A and B (a.k.a. States F and G).

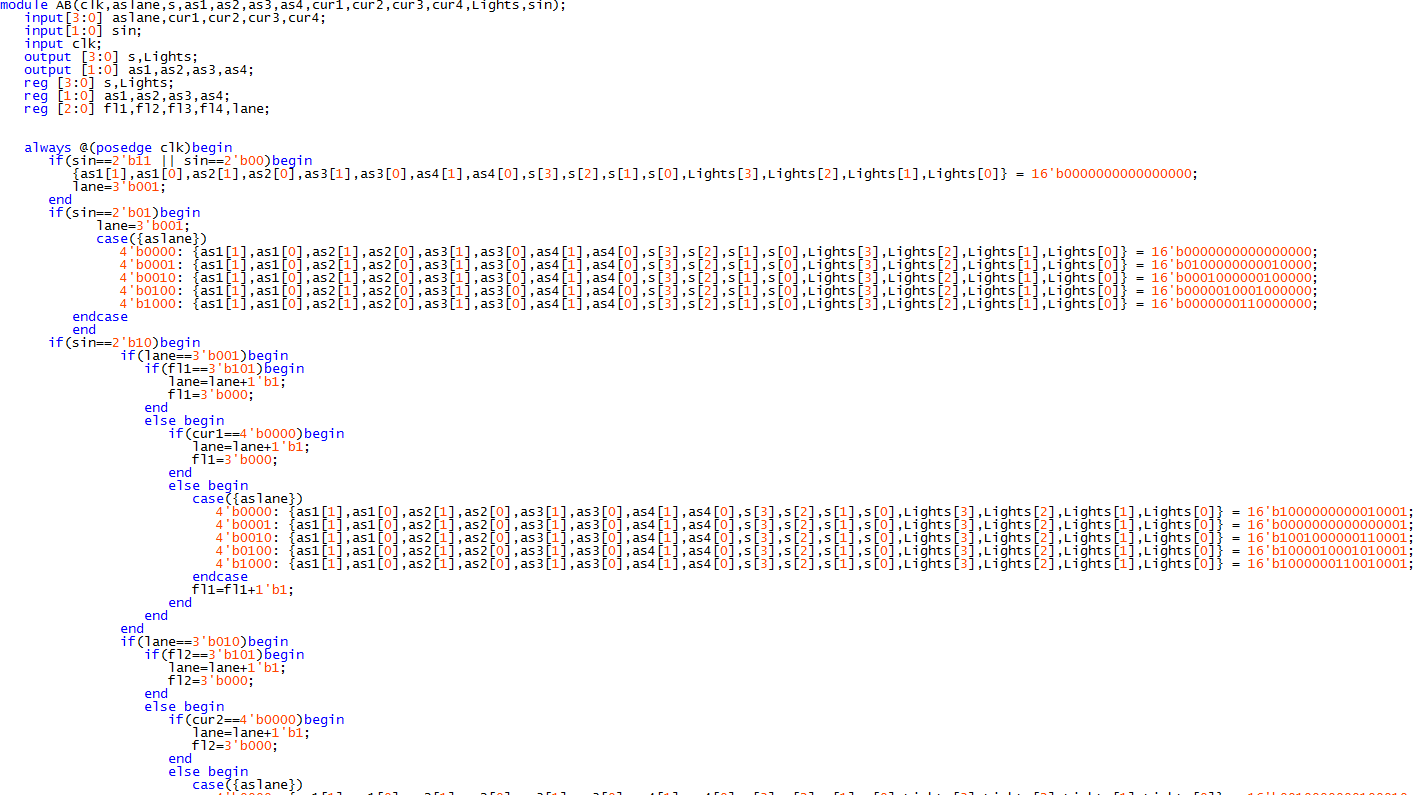
**Priority**

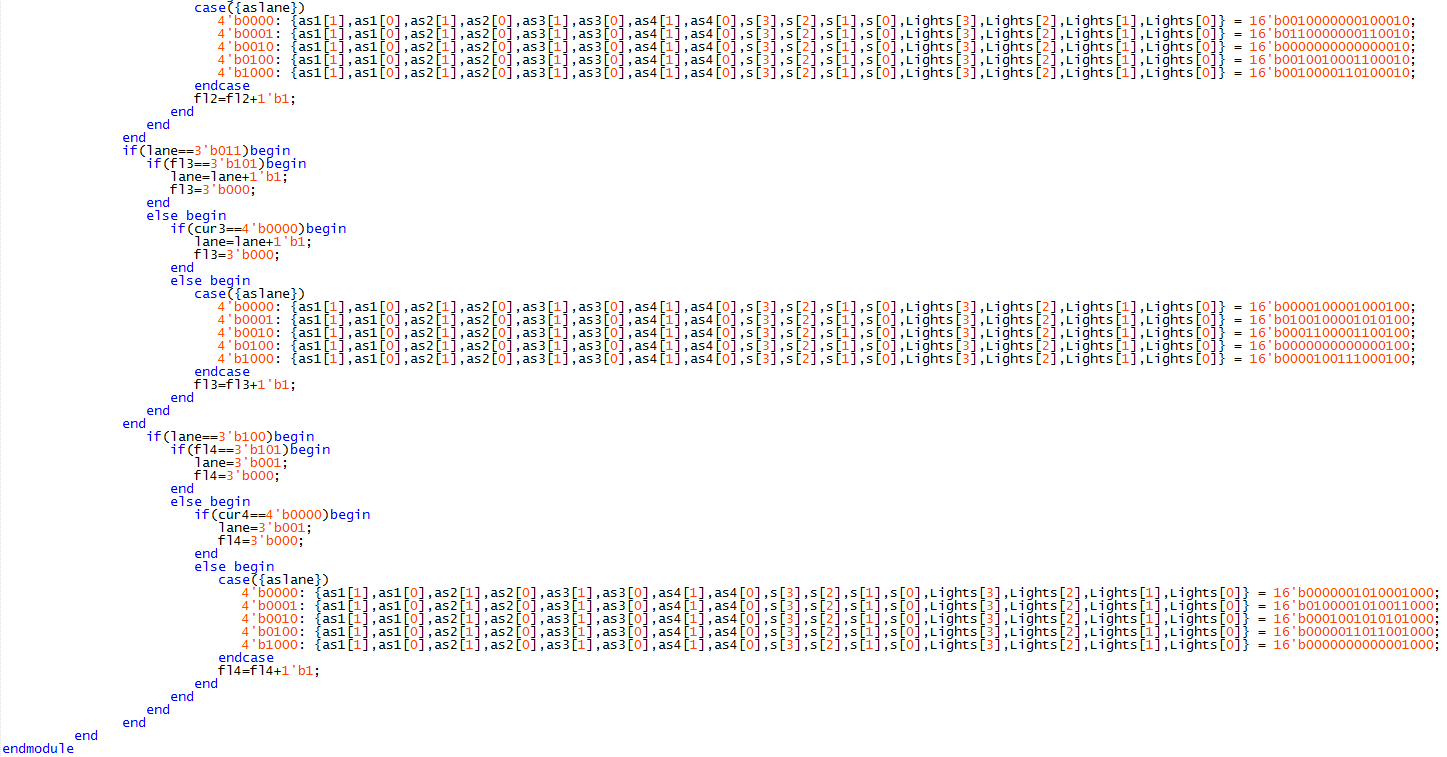
****

This is the first block on the left called “Priority”. What this block does is it sets a priority on the lanes so that if multiple lanes’ switches are up then it only picks at max two of them. The priority I have chosen is based off of rank with lane 4 > lane 3 > lane 2 > lane 1. So, if the Lanes value was 1110, then the priority of those lanes would be 1100. This output then gets put into the next block on the next page.

** Mode AB**

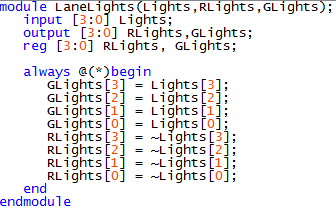
What this block does is almost the exact same as the Priority block and as I am typing this now, I now see there is no need for the Priority block whatsoever but it’s too late now. Anyway, the main functions of this code are to see what State it is in, the six-bit number starts off with 01, which is sin, or select in, which is the GF array from the StateDecider block, if it is in State F, or Mode A, 10 if it is in State G, or Mode B, and 00 if it is in any other State. If it is not in State F or G, it sets aslane, or add and subtract lane, to 0000 which means that nothing will be added or subtracted from any lane. If it is in State F or G however, it then sets aslane to a prioritized value and unlike the previous, useless, priority block, only one lane can be outputted but it is still based on the same rank. For example, if the most significant bit of lane is 1, then aslane will automatically be 1000, unless it is not in State G or F. aslane is then outputted and then used as the input in the next block which will be shown on the next page.

** AB**

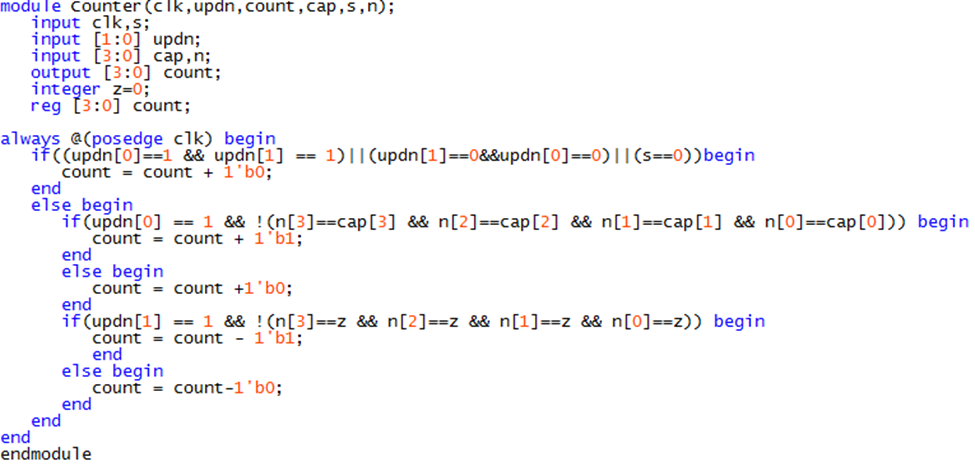
****

What this code does is checks if it is in State G, Mode A, or State F, Mode B, or neither. This is checked by sin, or select in, which is the GF output from the StateDecider block. If it is not in Mode A or B, it outputs a 16-bit number, twelve of which go to the four counters, and the other four go to the Lights array. Out of those first twelve, the first eight of those decide whether you will add or subtract, hence the names as1-4. If those values are 00 or 11 the counters will not do anything. The next four are select lines for the counters, hence the names s1-4. If the counters select line is zero, it will not do anything because it is not activated. Lights is set to 0000 because that means that no lane has a green light so none of the green lights will light up, but all the red ones will. If it is in Mode A, then you are allowed to add traffic to one lane while its light is red, and you are not at the capacity. If you reach the capacity, then you are not able to add any more cars to that lane. For example, if the aslane is 1000, then you will make as1-3 all 00 since you aren’t using them, and then you make s 1000 since you are only using the counter for lane four. For any lane, Lights will be 0000 because all lanes have a red light in Mode A. If you are in Mode B, that means that you will start at lane one. Lane one then has a green light and the number of cars will decrement from lane 1 for a maximum of five clock cycles, this is controlled by the always @(posedge clk) Statement towards the top of the code, or until the lane reaches zero cars. Then, once one of these conditions is met, it moves on to lane two and so on and so forth. The lane value is kept track by an array called “lane” and each time it moves from one lane to the next you add one to it, except in lane four where you set it back to 001. Each lane has an array called “fl1-4”, fl means five light, and it works the same as lane. Each clock cycle you spend on that lane you add one to it and if it reaches five you move on to the next lane. For example, if aslane is 0000 and you switch to Mode B, you then want to decrement from lane one and do nothing with any other lane. Therefore, you set as1 to 10 because 10 means subtract, and 01 means add, and then you set as2-4 to 00 and s to 0001, since you only want the first lane’s counter to be activated. Lastly, you set Lights to 0001. This will cause the far-right red light to turn off and the far light green light to turn on. Another example is if aslane is 1000 and you switch to Mode B. Then, as1 is set to 20 and as2-3 are set to 00 and as4 is set to 01 since you are adding to that lane. Then s is set to 1001 since the first and fourth lane are activated, and Lights still remains 0001 since lane one is the only lane with a green light. At any point if the current lane equals aslane (aslane is 0010 and you are on lane 2) nothing happens.

**Lane Lights**

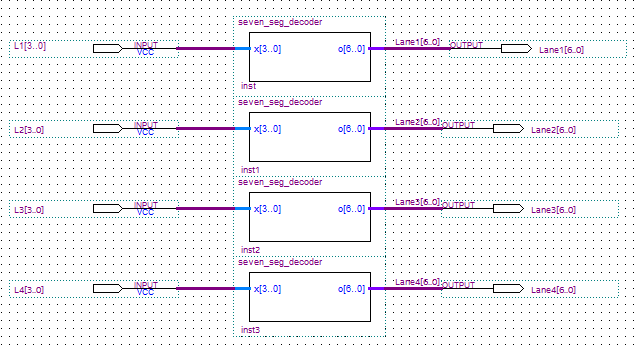
****

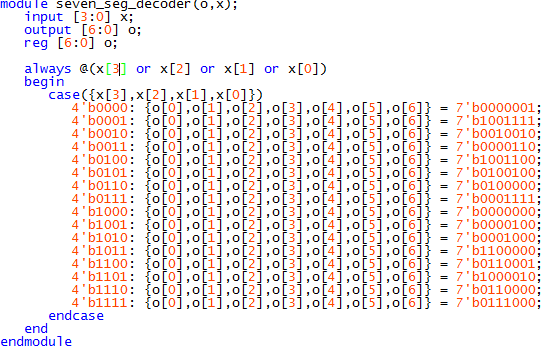
All LaneLights does is it takes in the Lights array from the AB block and outputs them so that the red lights are the opposite of the green lights.

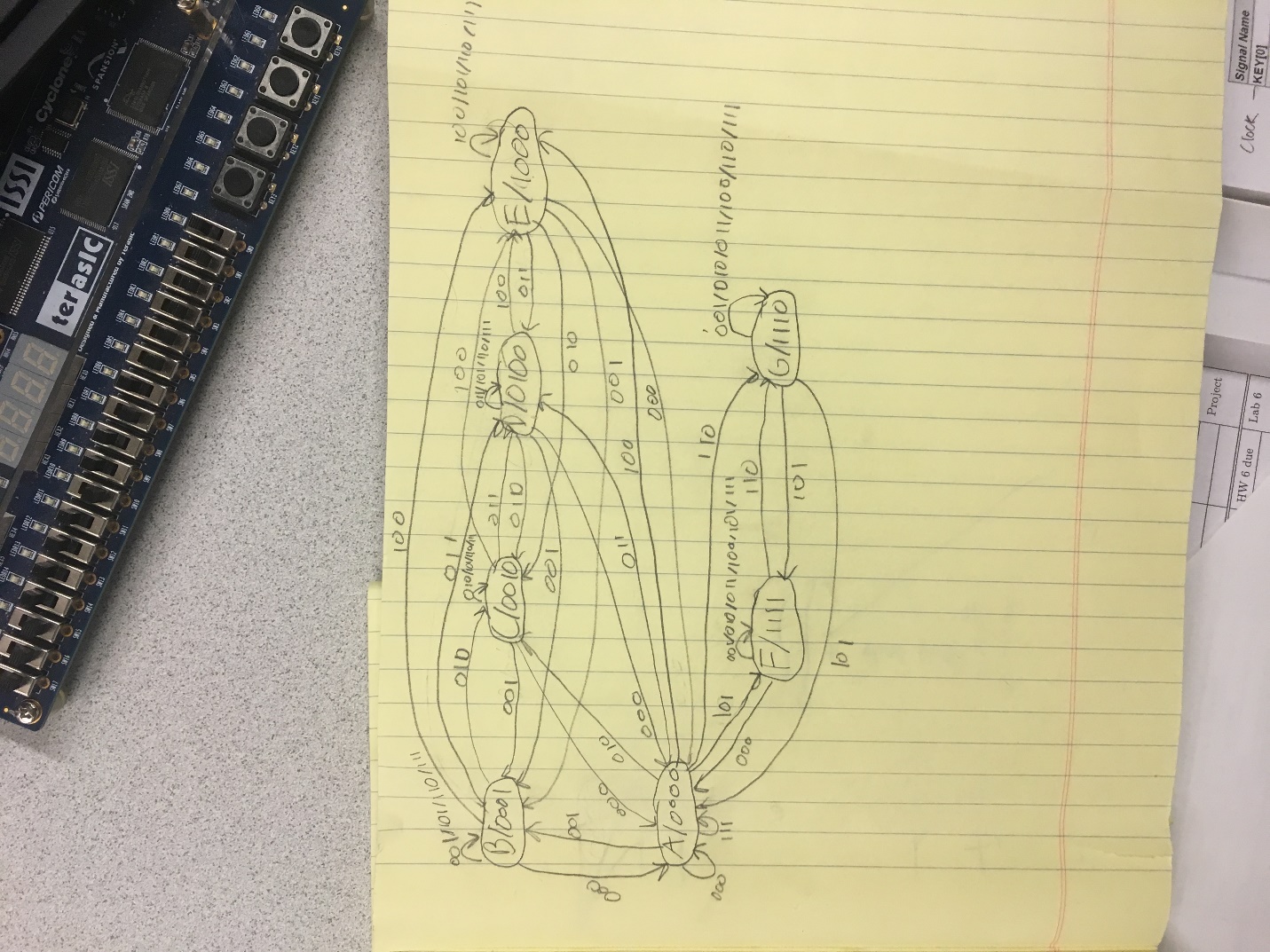
**** **Counter**

What Counter does is it takes n, which is the value of count plugged back into the counter, and first checks to see if updn, which means add or subtract, is 11 or 00 or if s is zero. If any of these are true, the counter will not do anything. If those aren’t true however, it checks to see if updn is 01 and that n is not at the capacity of that lane. If it is not at the capacity, you add one, otherwise you add zero. Then it checks to see if updn is 10 and that n is not at zero. If not, the counter decrements the cars in the lane by one, otherwise it does nothing.

**Seven Segs**

****

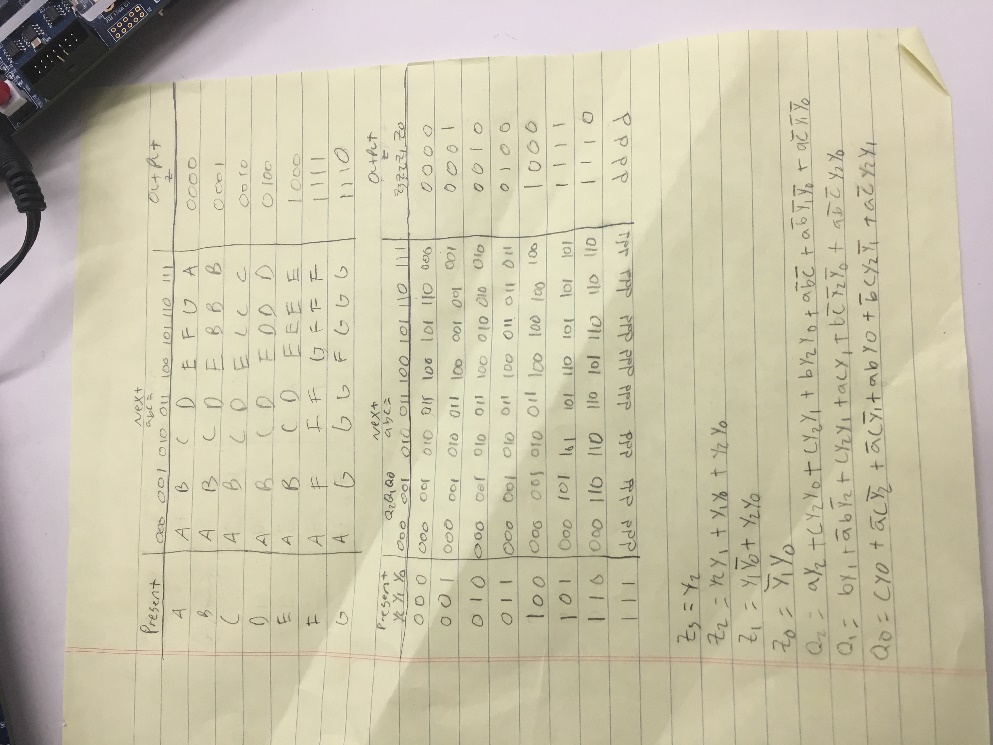
****

**** The values of the counter are then outputted into a block called “SevenSegs”, which is a block of four seven segment decoders. This then takes the output of the counter and displays them as a decimal number on the hex displays.

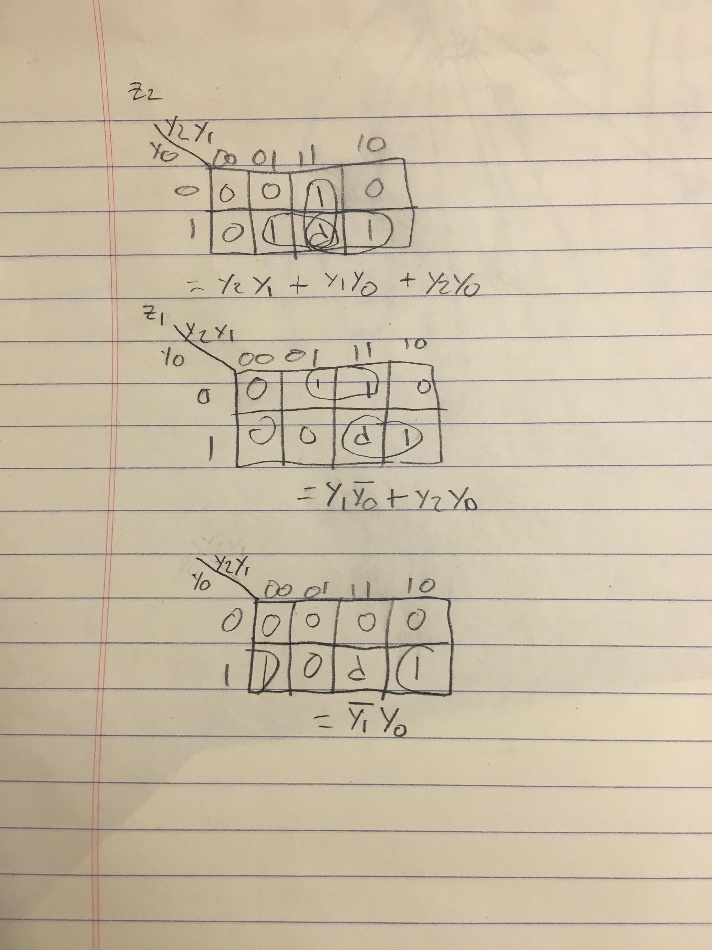
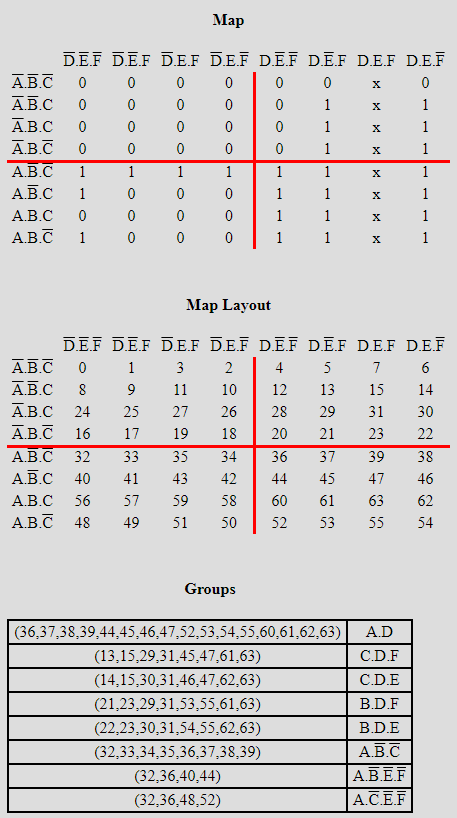
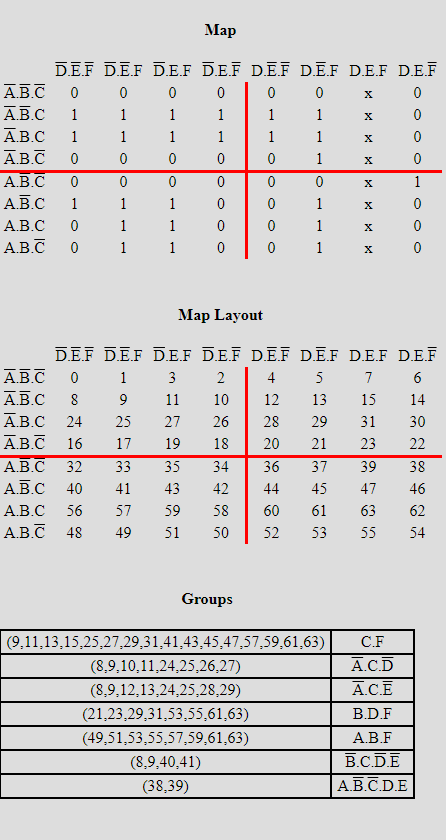
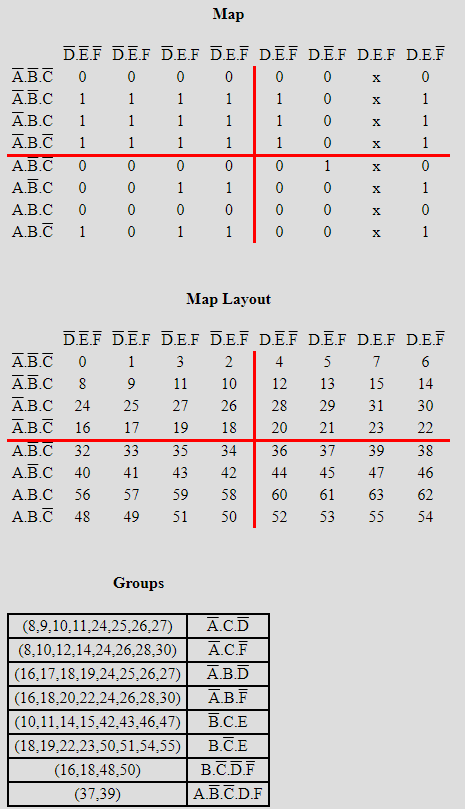
**State Machine Drawing**

As you can see from the State, State A does nothing. Its main purpose is to separate the lane capacity loading States and Mode A and B. Once you go to B, C, D, E and you load in a lane’s capacity, you can then go to any lane you want. Once you are done, you go to State A and then you can choose whether you want to go to Mode A or B. From this you can get the State table and the State assigned table on the next page.

**State Tables**

****

On the sheet you can see the Boolean expressions for the next State variables and the output variables. Those can be found by simplifying these K-Maps:



List of Block Elements I had to Design:

* OSL
* NSL
* StateMachine
* StateDecider
* LoadCapacities
* RegFile
* 2to1Mux
* 4-1Mux
* Decoder
* Priority (Useless)
* ModeAB
* AB
* Counter
* SevenSegs
* Seven\_Seg\_Decoder
* LaneLights

Test Case

At first I will go to State B by setting SW17-SW15 to 001 and then use the button KEY0, which is used to switch between all State, to switch from State A to B. Then, LEDG7-LEDG4 will light up 0001 showing that it is in State B, this State will load the capacity for lane one. In the beginning, LEDR10-LEDR7 will output 1111 because all lanes currently have a capacity of zero which cannot happen, HEX3-HEX0 will all show zero because no lanes have any cars in them, LEDR3-LEDR0 will light up 1111 because all cars have red lights, and LEDG3-LEDG0 will light up 0000 because no cars have green lights. For lane one, I will use SW14-SW11 to load in the value 1101, or 13, and then LEDR7 will turn off and HEX4 will show ‘d’. I will also then use the KEY1 button to send this value to the register file and LEDR17-LEDR14 will light up 1101 because this is the value currently in the register for lane one. Then I will go to State C, which loads the capacity for lane two by setting SW17-SW15 to 010. LEDG7-LEDG4 will then light up 0010 and then I will set SW14-SW11 to 1001 to set the capacity of lane two to nine. LEDR8 will then turn off and HEX5 will show ‘9’ and then I will use the KEY1 button to send this to the register file and LEDR17-LEDR14 will show 1001. Then by setting SW17-SW15 to 011 I will go to State D, which will load in the capacity for lane 3. LEDG7-LEDG4 will then light up 0100. I will set SW14-SW11 to 1111 to set the capacity of lane 3 to 15 and then LEDR9 will turn off and HEX6 will light up ‘F’ and after hitting the KEY1 button, LEDR17-LEDR14 will light up 1111. Then I will set SW17-SW15 to 100 to go to State E, which sets the capacity for lane 4 and LEDG7-LEDG4 will light up 1000. I will then use SW14-SW11 to 0111 to set the capacity of lane 4 to 7. HEX7 will show ‘7’, LEDR10 will turn off, and after hitting the KEY1 button LEDR17-LEDR14 will show 0111. Then I will set SW17-SW15 to 000 to go back to State A and LEDG7-LEDG4 will light up 0000. Then I will set SW17-SW15 to 110 to go to State G to show that it can go from State A to State G, LEDG7-LEDG4 will output 1110. Then I will set SW17-SW15 to 100 to go to State F, which is Mode A, and LEDG7-LEDG4 will light up 1111 and LEDR3-LEDR0 will lights up 1111 because in Mode A all lanes have red lights. I will then set SW8-SW5 to 1000 to add cars to lane 4. Using the button KEY2, I will add 7 cars to this lane and then attempt to add more cars to it by continuing to hit this button but HEX3 will continue to show ‘7’ because the capacity of this lane is 7. Then I will set SW8-SW5 to 0100 to add cars to lane 3. Using the button KEY2, I will add 8 cars to this lane and HEX2 will show ‘8’. Then I will set SW8-SW5 to 0010 to add cars to lane 2. Using the button KEY2, I will add 9 cars to this lane and HEX1will show ‘9’. Then I will set SW8-SW5 to 0001 to add cars to lane 1. Using the button KEY2, I will add 10 cars to this lane and HEX0 will show ‘A’. Then I will set SW17-SW15 to 100 and switch to State G, which is Mode B. Then I will hit KEY2 and LEDR3-LEDR0 will light up 1110 because now lane 1 has a green light and LEDG3-LEDG0 will light up 0001. Then, while I hit KEY2, the value in lane one will decrease by one and HEX0 will show this happening. After I hit the KEY2 button five times, it will switch over to lane two and lane two will have a green light. Consequently, LEDR3-LEDR0 will light up 1101 since lane 2 has a green light and LEDG3-LEDG0 will light up 0010. Then, while lane 2 has a green light, I will set SW8-SW5 to 0001 and then while I hit the KEY2 button, it will decrease the amount of cars in lane 2, shown on HEX1, by one but also increase the amount of cars in lane 1 by one. After I hit the KEY2 button five times, it will switch over to lane 3 and LEDR3-LEDR0 will output 1011 because now lane 3 has a green light and LEDG3-LEDG0 will output 0100. I will set SW8-SW5 to 0000 so that no lanes get cars added and I will again hit KEY2 five times and the decreasing value of cars in lane 3 will be shown on HEX2. Then it will switch to lane 4 and LEDR3-LEDR0 will output 0111 since lane 4 has a green light and LEDG3-LEDG0 will output 1000, the decreasing value of cars in lane 4 will be shown on HEX3. After I hit the KEY2 button five times, it will go back to lane one. I will continue hitting KEY2 button until I get to lane 2 to show that after I reach a value of zero for current cars in that lane and it will automatically switch to the next lane and I will do the same until all lanes are at 0. Then I will set SW17-SW15 to 000 and hit the KEY1 button and then hit the KEY2 button. LEDG7-LEDG4 and LEDG3-LEDG0 will output 0000 and LEDR3-LEDR0 will output 1111.