Felix Pawlowski

ECEN 2350

Fall 2020

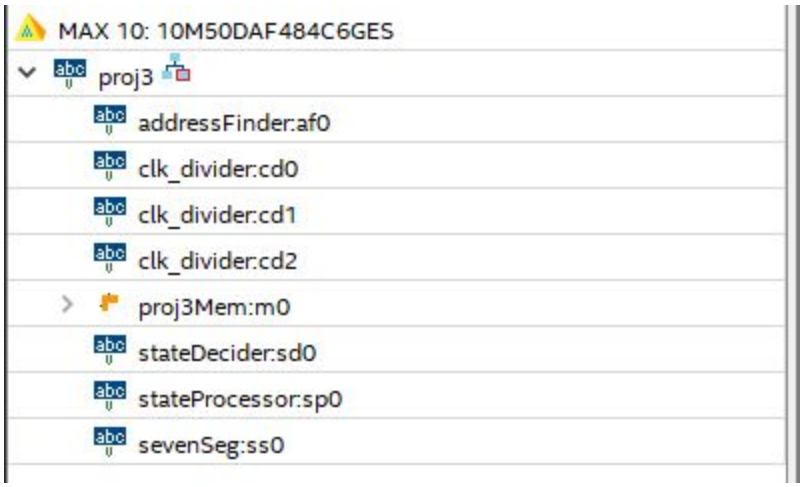
The purpose of this project is to emulate the functioning of the tail lights of a 1965 Ford Thunderbird. In addition to this I must use ROM to create the equivalent of a demo mode for my circuit.

My theory of operation is very simple. I first feed inputs from my switches and buttons into a module called stateDecider. This module converts these inputs into a 4 bit wide binary number. Each bit represents the state of the machine. Bit 0 represents whether or not the turn signal is on, bit 1 represents whether the turn is a left or right turn, bit 2 represents the status of the brake pedal, and bit 3 represents the status of the hazards. While the first 3 bit choices were fairly arbitrary, the final one is very important. For all values greater than or equal to 4'b1000, the hazards are on. The state as determined by the state decider is output to a variable.

From there we do two things with the output from the state decider. The first is to display this convenient 4 bit wide value on one of the hexadecimal displays. The second is to feed it to a second module named stateProcessor. This module does two important things, it uses a clock to time the blinking of hazards and turn signals and it lights up LEDs based on the state of the machine. For the left and right turn signals a counter counts up from zero to two and resets. The value of this number is then parsed into the correct state of the LEDs assigned to a turn signal. For the hazards a variable oscillates between zero and one and depending on the value the lights are either on or off. Another key functionality of this module is the non-clocked illuminating of LEDs. I found through trial and error that making the LEDs light up instantaneously worked more reliably. Another key issue my code addresses is that the state of the button to decide

whether a turn is left or right is irrelevant when the turn signal is off. The state of the button is ignored whenever the turn signal is off and functions the same in both cases.

For the memory I created a 4 bit wide 8 bit deep ROM. In that ROM I stored all 7 states plus an additional hazard state. Thanks to quartus magic a variable is fed the value from the memory at the currently selected address. I use a module named addressFinder to cycle through every possible address at a fairly slow rate such that full functionality is displayed. Finally the status of switch nine is checked. If it's on the output from the memory is fed to the stateProcessor, if it's off the output from stateDecider is fed to the stateProcessor.



The testbench is probably the simplest part of the code. I just simulate switch inputs for all seven possible states and feed it a clock. One key step was ensuring the timing was such that the simulation would show all the information needed. Otherwise I simulated a clock.

(labeled screenshots for every state are in the folder, I do not feel including the details here is incredibly important beyond that my simulation was successful)

I was entirely successful. The main issues I encountered were with the stateProcessor as I alluded to earlier in this report. I was having trouble illuminating the LEDs correctly for the turn signals and brakes. My solution was to create two always blocks, one that was tied to the clock and one that was asynchronous. This solved my problem along with minor adjustments to syntax. The second major issue I encountered was in making the memory. This was solved by consulting Dr. Robinson for help. Once I made the memory block my code successfully executed after syntax fixes. Road bumps aside the only aspect of my code that doesn't function ideally is the demo mode where it sits in the hazard state for two cycles instead of one.