Will Snider
wisn3972@colorado.edu

2350 Project 3 report

## Problem Statement

The purpose of this project was to recreate the taillight system that is used in a 1965 Ford Thunderbird. Switches 2 down to 0 control what will be shown along with KEY[1] being used to differentiate between left and right for the turn signals. Hazards are controlled by SW[0], turn signal by SW[1], and the brakes through SW[2]. Hazards will have the highest priority, turning them on regardless of the state of SW[2:1] if SW[0] is up. When both the brake and turn signal are used, the taillights are to still display the turn signal but on the other side, display the brakes. The design is to have two modes controlled by SW[9], the default being to display in accordance with how the switches and buttons are configured, the other being to automate the design going through each state on its own through the use of memory.

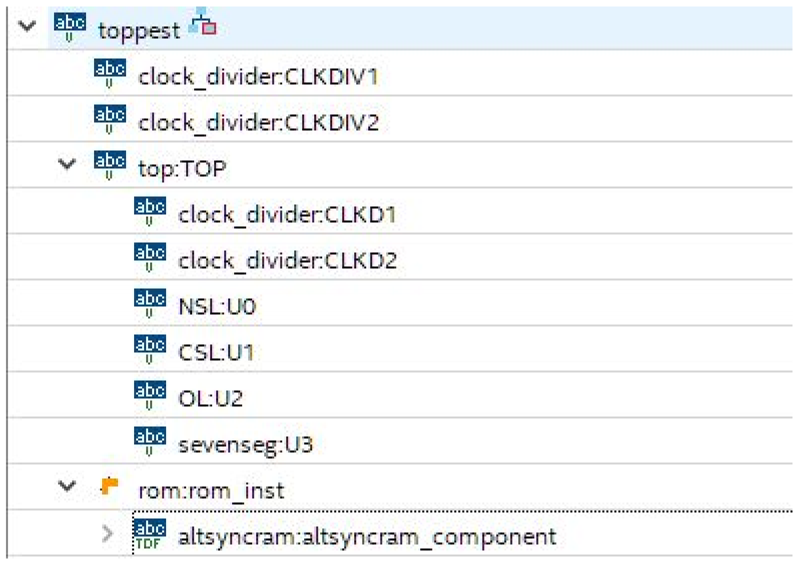
## Theory of operation

The design consists of three main modules, CSL, NSL, and OL. Clock dividers and the seven segment display module were also brought back from previous projects. The sevenseg is used to display the current state the device is in, with the clock dividers being used for various system clocks, a 2Hz clock is used for clocking in the states and the memory, while a 4Hz clock is used to control the speed of the turn signals and hazard lights. Controlling the states in the memory automation part of the design, a 0.4Hz clock is used (ignore the naming of it being pointTwoHzClock in the code). Multiple files were added using the built in Quartus memory wizard, The memory was designed to simply store the inputs required for each state (KEY[1] and SW[2:0]) in separate locations. The 0.4Hz clock simply cycled a reg through each memory address in order to automate the design.

NSL was our Next state logic module, taking our inputs and determining the state we'll be in next. This was done simply by concatenating KEY[1] with SW[2:0]  and using a case statement to determine the next state using the possible inputs. CSL was relatively simple simply setting current state based on the 2Hz clock or resetting to idle in the case of a reset. The theory behind the OL was to have temp variables for each state that update in the background and continue to function regardless of what the current state is, then using a case statement based on our current state to set a final temporary LED variable to a either the single temp variable needed or a combination of multiple such as in the case with brakes and a turn signal. Lastly assigning LEDR to this temp variable.

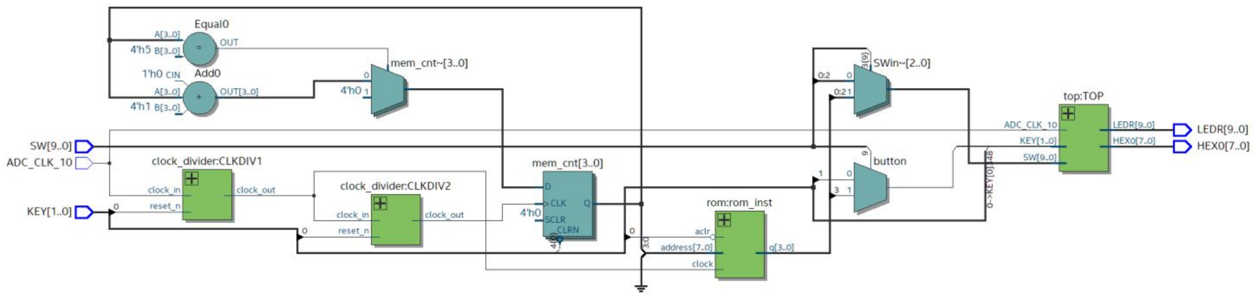
A multiplexer based on the state of SW[9] was used to control the inputs going into the design, either using the memory while high or the physical keys and switches while low.

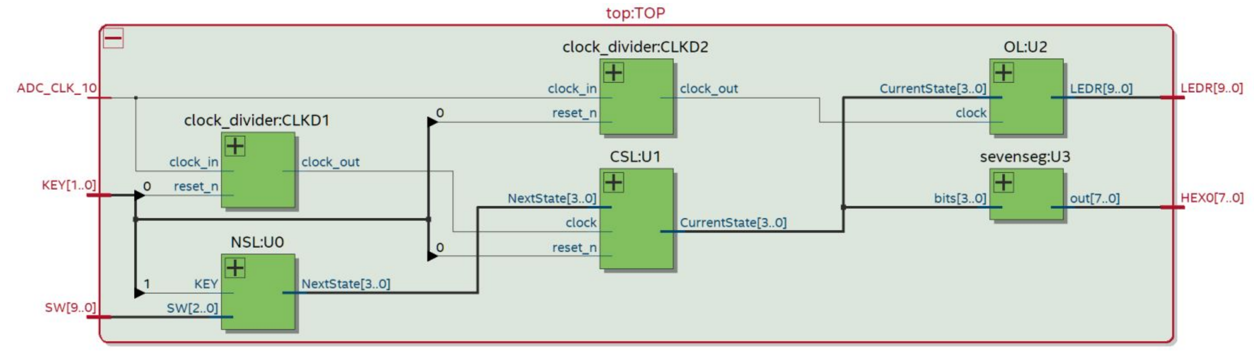
## Hierarchy



toppest
clock_divider:CLKDIV1
clock_divider:CLKDIV2
top:TOP
clock_divider:CLKD1
clock_divider:CLKD2
NSL:U0
CSL:U1
OL:U2
sevenseg:U3
rom:rom_inst
altsyncram:altsyncram_component

## Block diagram

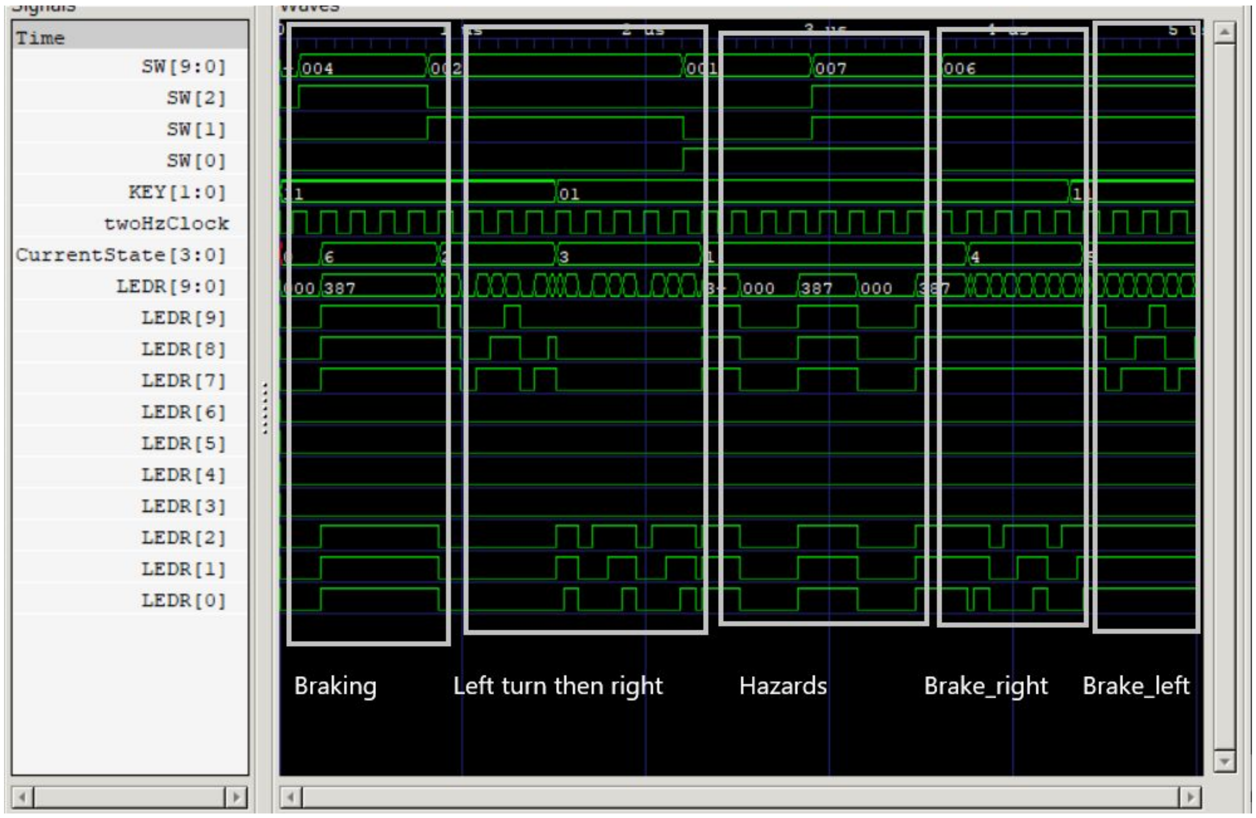




## Testbench Operation

Our testbench tests the design before memory automation was implemented. Starting out with a system reset, and then going through each state we want to test. And making sure they appear as we expect them to.

## GTKwave output



## Summary

To conclude, we were able to recreate the design of the 1965 Ford Thunderbird tail lights and control them through either the board's physical inputs or using memory automation in order to display the possible states. A test bench was created in order to test the design in manual mode with the output shown above.