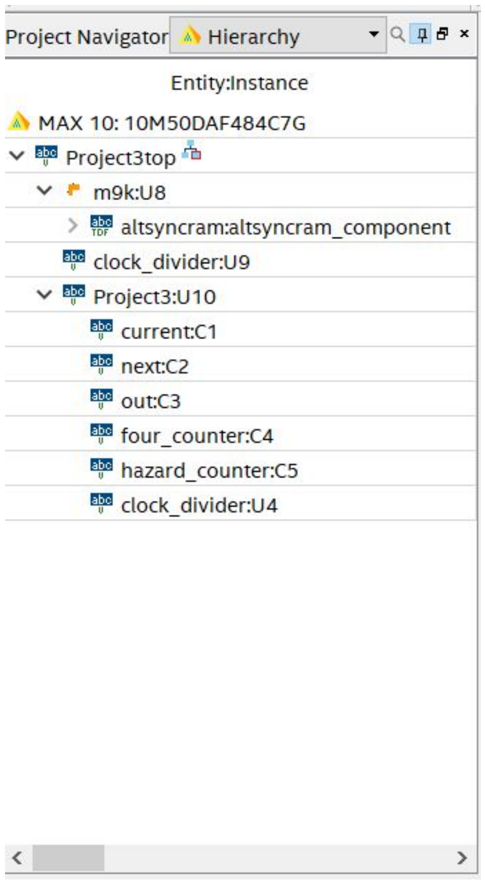
Project 3 Report - Sonal Tamrakar

For this project, we are designing finite state machines which will simulate tail lights of a 1965 Ford Thunderbird. The state machines will be operated through the KEY[1:0] and the SW[9:0] from our DE-10 board. There will be a total of six states, for which each state results in a certain behavior of the taillights. We'll be viewing the taillights and the finite states work through the uses of our HEX[0] display and the ten LED[9:0]. The six states are going to be manually operated. There is also another section of this project where we automate the individual states, not requiring us to manipulate the KEYs and switches. The taillights will be automated through a memory block which we're going to make in Quartus.

Before starting on our output logic, we first made our next state and out current state logic. In order to be organized, we separated all the three, next logic, output logic and current logic into separate verilog files which we later instantiated in our top module. The next state logic consisted of case statements in which if and else-if statements were used to redirect what next state we want to be in. In our current state logic module, we inserted our highest prioritized condition which was if KEY[0] was pressed, it would stop any running state and blank the displays as well as turn the LEDs off. In order to enter a state, certain conditions have to be met like the state of our switches and keys. Our second prioritized state are the hazard lights. If the conditions for hazard lights are met, it will go to State 1 and start blinking the LEDR[9:7] and LEDR[2:0]. SW[0] is the hazard enable and disable switch. SW[1] is the signal enable switch, and depending on the state of KEY[1], it will either blink LEDR[9:0] or LEDR[2:0] in a manner that simulates right or left turn signals. Our brake lights are operated through SW[2] and can also work alongside with our turn signals. State 2 is brake lights with left turn signals and State 3 is brake lights with right turn signals. State 4 and 5 are the normal turn signals. And finally, State 6 are the brake lights. In order to view the operation easily, we used a 5Hz clock by dividing our 10mHz clock. In our output logic, we set the outputs of the switches and the leds. Some outputs needed a counter, which we created separate .v files for. In our Project3.v file, we instantiated all the mentioned designs to manually operate the taillights.

For the second part of the project, we automated the design by using the M9k memory block inside quartus. Through the memory block, the design will spend approximately 5 seconds in 5 of the 6 states, excluding the brakes. It will first be off for 5 seconds, then the hazard lights will blink for 5 seconds, followed by the two turn signals, then finally the two turn signals with brakes on and go back to being off. It will run in a continuous loop. The automated design will run at a frequency of 0.2 Hz so that it is easy to see it operate. The automated and the manual blocks will be operated through a multiplexer based on the state of SW[9] in a new top module.

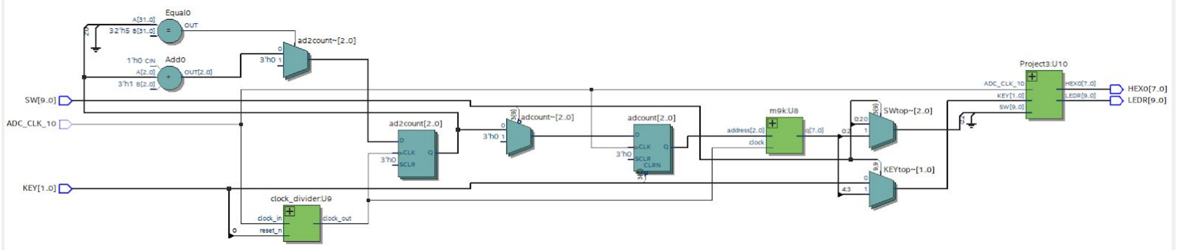
Hierarchy of source files:


Project Navigator | Hierarchy

Entity:Instance

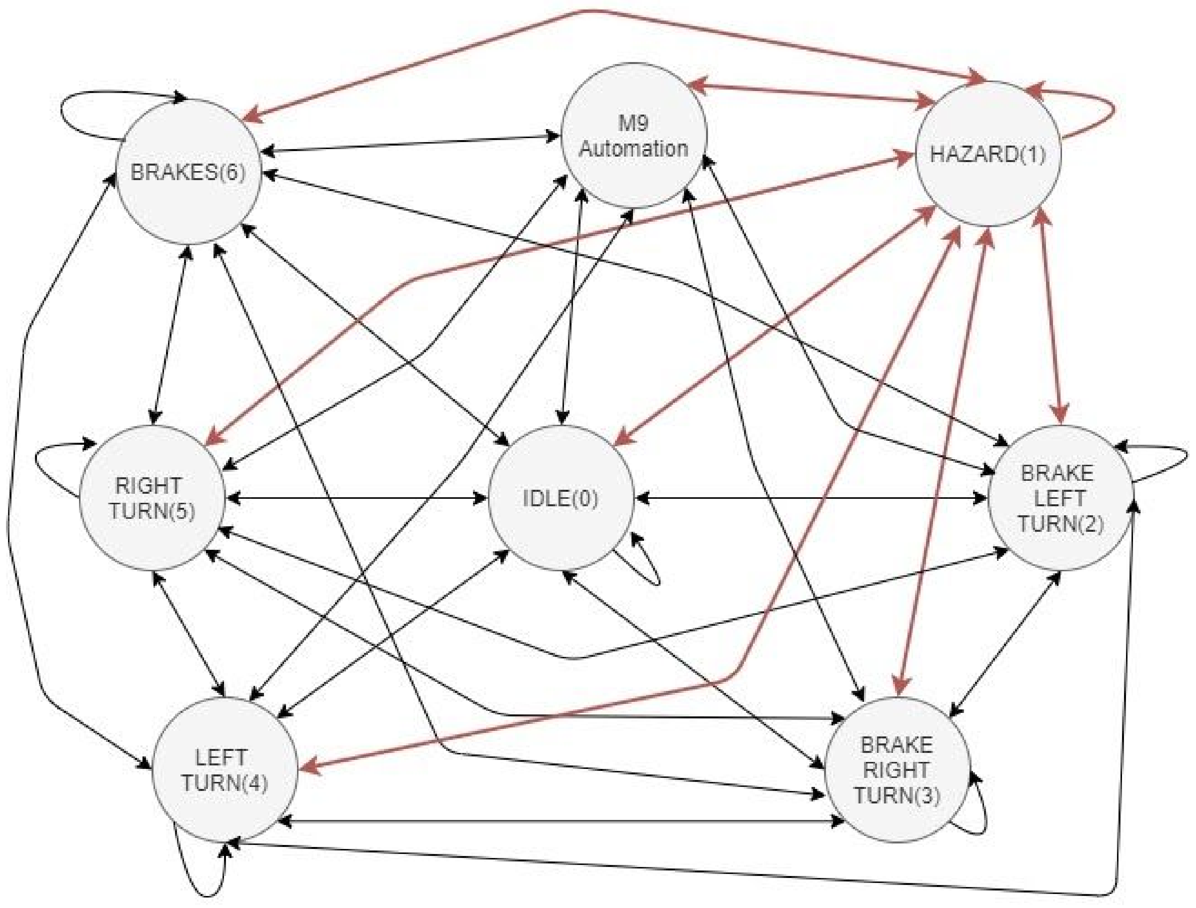
MAX 10: 10M50DAF484C7G
- Project3top
  - m9k:U8
    - altsyncram:altsyncram_component
  - clock_divider:U9
  - Project3:U10
    - current:C1
    - next:C2
    - out:C3
    - four_counter:C4
    - hazard_counter:C5
    - clock_divider:U4

RTL Viewer:

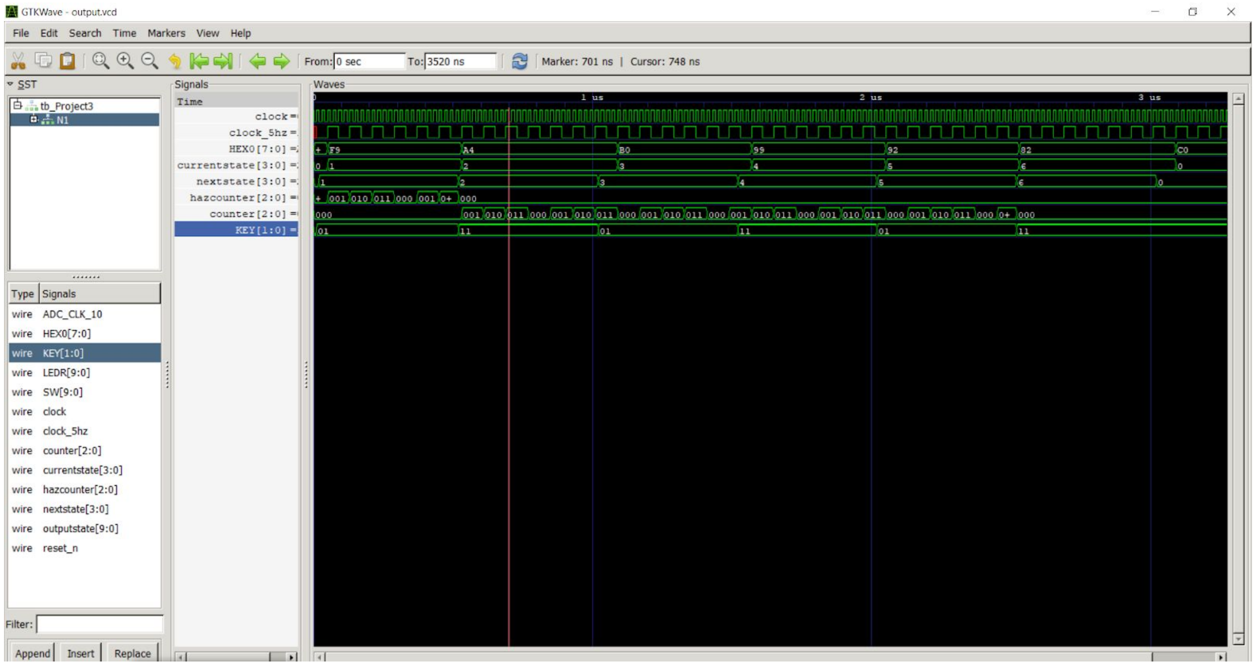


A .pdf form of the RTL viewer is provided in the ProjectReport folder.

State Bubble Diagram:

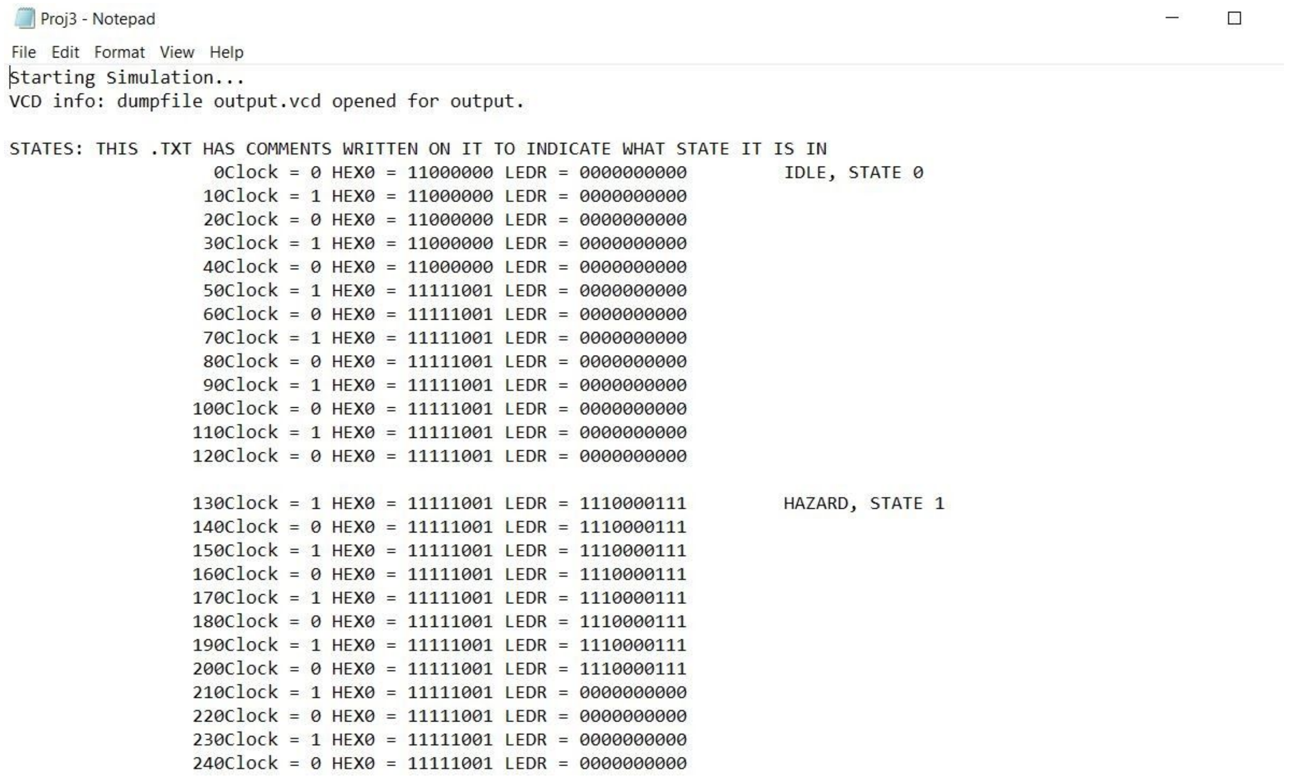


Description of testbench operation:



From: 0 sec To: 3520 ns Marker: 701 ns | Cursor: 748 ns

```
wire ADC_CLK_10
wire HEX0[7:0]
wire KEY[1:0]
wire LEDR[9:0]
wire SW[9:0]
wire clock
wire clock_5hz
wire counter[2:0]
wire currentstate[3:0]
wire hazcounter[2:0]
wire nextstate[3:0]
wire outputstate[9:0]
wire reset_n
```



```
Starting Simulation...
VCD info: dumpfile output.vcd opened for output.

STATES: THIS .TXT HAS COMMENTS WRITTEN ON IT TO INDICATE WHAT STATE IT IS IN
                   0Clock = 0 HEX0 = 11000000 LEDR = 0000000000        IDLE, STATE 0
                  10Clock = 1 HEX0 = 11000000 LEDR = 0000000000
                  20Clock = 0 HEX0 = 11000000 LEDR = 0000000000
                  30Clock = 1 HEX0 = 11000000 LEDR = 0000000000
                  40Clock = 0 HEX0 = 11000000 LEDR = 0000000000
                  50Clock = 1 HEX0 = 11111001 LEDR = 0000000000
                  60Clock = 0 HEX0 = 11111001 LEDR = 0000000000
                  70Clock = 1 HEX0 = 11111001 LEDR = 0000000000
                  80Clock = 0 HEX0 = 11111001 LEDR = 0000000000
                  90Clock = 1 HEX0 = 11111001 LEDR = 0000000000
                 100Clock = 0 HEX0 = 11111001 LEDR = 0000000000
                 110Clock = 1 HEX0 = 11111001 LEDR = 0000000000
                 120Clock = 0 HEX0 = 11111001 LEDR = 0000000000

                 130Clock = 1 HEX0 = 11111001 LEDR = 1110000111        HAZARD, STATE 1
                 140Clock = 0 HEX0 = 11111001 LEDR = 1110000111
                 150Clock = 1 HEX0 = 11111001 LEDR = 1110000111
                 160Clock = 0 HEX0 = 11111001 LEDR = 1110000111
                 170Clock = 1 HEX0 = 11111001 LEDR = 1110000111
                 180Clock = 0 HEX0 = 11111001 LEDR = 1110000111
                 190Clock = 1 HEX0 = 11111001 LEDR = 1110000111
                 200Clock = 0 HEX0 = 11111001 LEDR = 1110000111
                 210Clock = 1 HEX0 = 11111001 LEDR = 0000000000
                 220Clock = 0 HEX0 = 11111001 LEDR = 0000000000
                 230Clock = 1 HEX0 = 11111001 LEDR = 0000000000
                 240Clock = 0 HEX0 = 11111001 LEDR = 0000000000
```

The GTKwave has signals such as the clock, out 5hz divided clock, HEX0 which shows what state we're currently in, the current and the next state, and some of the counters we used. A part of the output file shows the first 240 cycles. The full notepad text file is in the Testbench Output folder. The text displays the changing HEX0 values depending on the state. For reference, we attached the binary to seven segment hex .v file to understand the HEX0 values on the output.txt better. Comments are made in the .txt files which clearly labels which state it is in. The text file also displays the changing behaviour of the LEDs.

Summary:

The manual part of the project went smoothly. We did have some issues making the counter for the LEDs for the turn signals, but ended up figuring it out. The testbench wasn't working properly at start because it wasn't using the right clock, but we eventually got it to work. The automation part of kind of rough as we had to make separate counters for the addresses. As of now, the manual part of the project works as expected. There's around a 5 second transition. When SW[9] is first turned up, it'll stay in State 0 for about 5 seconds then move on to the hazard state. The automation works as expected but sometimes when SW[9] is turned up, at the very beginning, instead of starting at State 0 (all off),it sometimes starts operating at a different state but does run in a continuous loop. Other than that, the project went great and through it, got a much better understanding of state machines and memory blocks.