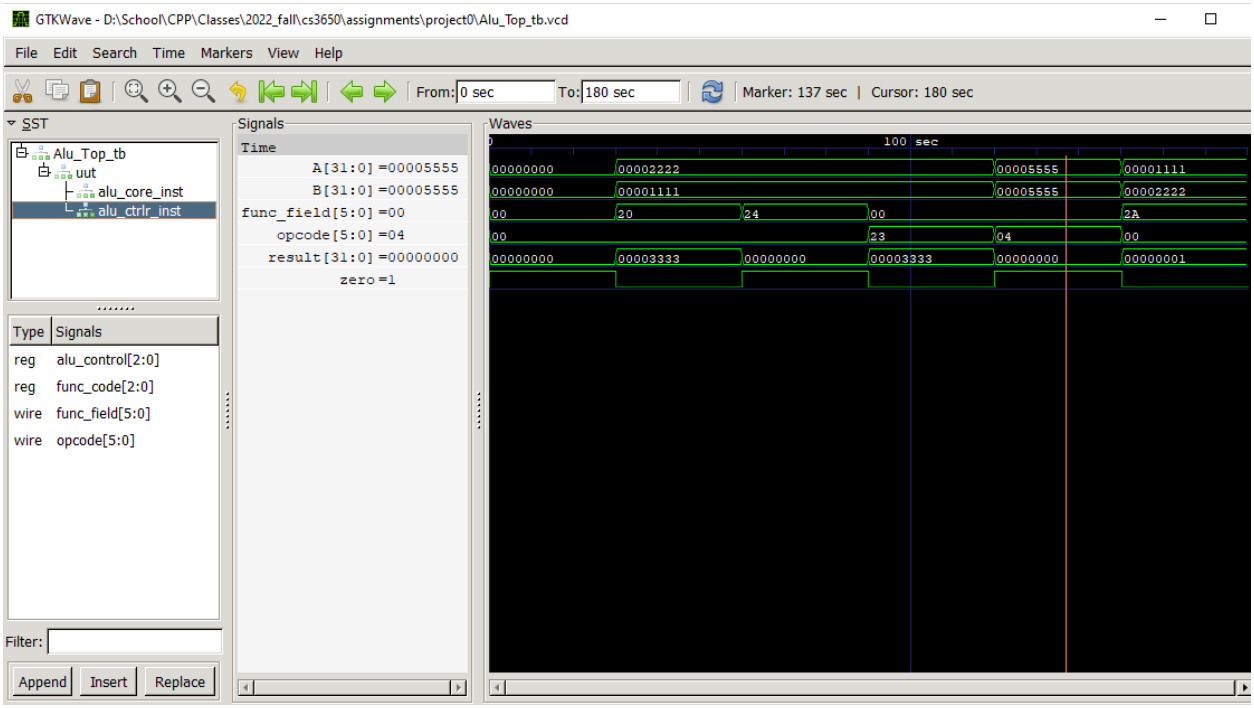
Project0: ALU Wave

Notes of learning:

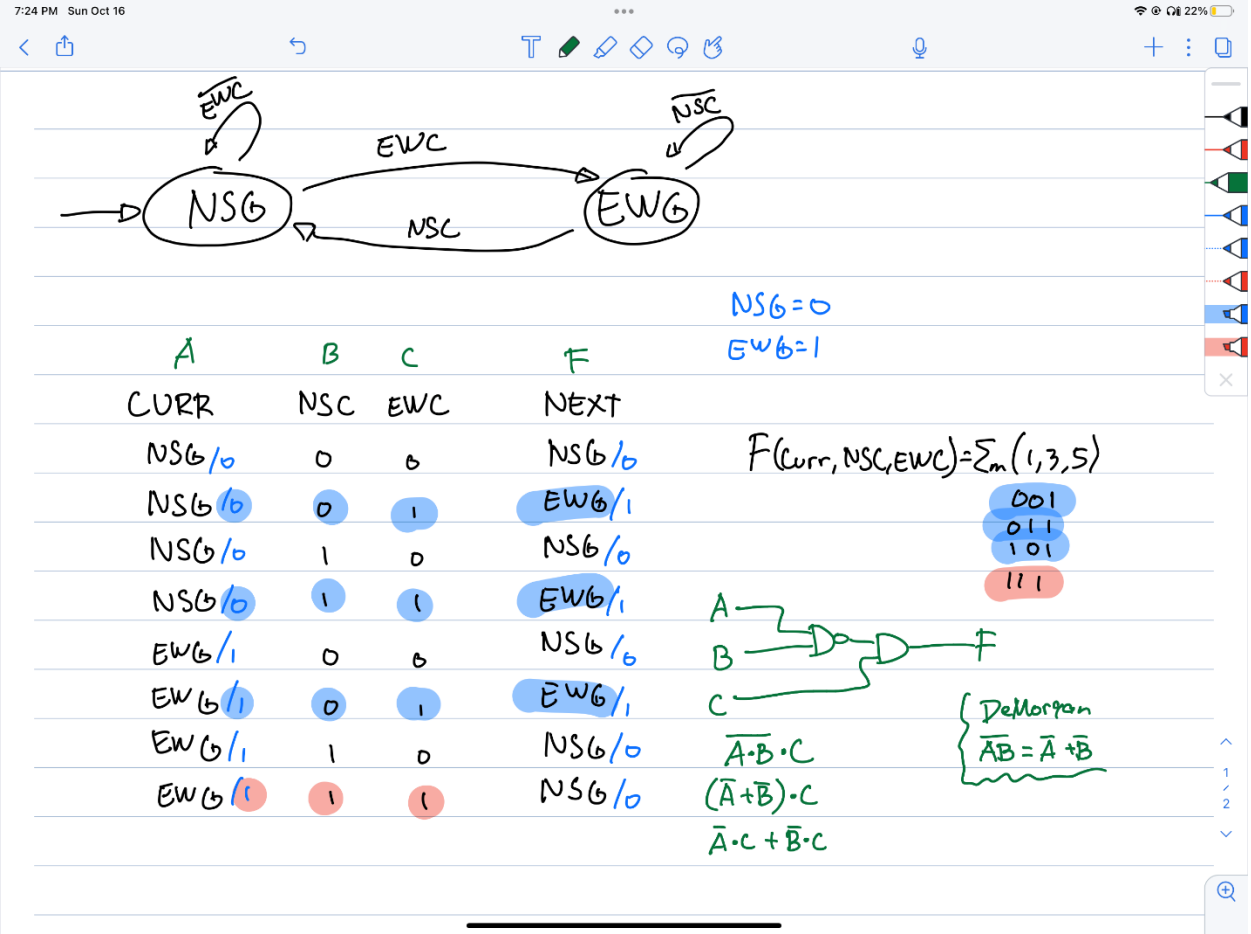
- A module is a block of Verilog code that implements certain functionality. Modules can be embedded within other modules, and a higher-level module can communicate with is lower-level modules using their input and output ports. A module represents a design unit that implements specific behavioral characteristics and will get converted into a digital circuit during synthesis.
    - When initializing a model, it is recommended to name it accordingly to the Verilog file.
        - Ex: Alu_Contro.v -> module Alu_Control ()
    - The inputs and outputs variables are first put inside the parenthesis of module ()
        - Ex: module Alu_Control(opcode, func_field, alu_control);
    - Each module has to be closed with “endmodule”
- Input is a Verilog keyword for input.
    - Ex: input [5:0] opcode; // [5:0] means that it is 6 bits
- Output is a Verilog keyword for output.
    - Ex: output reg [2:0] alu_control; // [2:0] means that it is 3 bits
- Reg keyword in Verilog is used for describing logic.
    - For left hand side of signals assigned inside in always blocks
- Wire keyword in Verilog is a wire represents a physical wire in a circuit and is used to connect gates or modules. The value of a wire can be read, nut not assigned tom in a function or block.
    - For left hand side of signals assigned outside always blocks.
- During the behavioral model simulation, all the flows defined by the always and initial statements start together at simulation time zero.
    - The always statements are executed repetitively.
    - In always @ (*), (*) means “build the sensitivity list for me”.
- Alu_Control module is the signal which is sent to the ALU core to indicate what type of operation is to be performed.
    - Control signal for the ALU which determines the type of operation to be performed by looking at either the opcode for I-Type or function field for R-Type instructions.
- Alu_Core module performs the desired operation from Alu_Control.
- Alu_Top module instantiates and connects both the Alu_Control and Alu_Core modules.
- To store the output in waveform, we must include the following code inside initial begin in Alu_Top module
    - $dumpfile("Alu_Top_tb.vcd");
    - $dumpvars(0, Alu_Top_tb);
- To run the simulation, open the terminal and run the following commands:
    - iverilog -o Alu_Top_tb.vvp Alu_Top_tb.v // synthesizing the test bench module to .vvp from the .v
    - vvp Alu_Top_tb.vvp // to run the simulation
    - gtkwave // view the wave form
    - // open the .vcd file in GTKWave

Screenshot of the waves:



EXTRA CREDIT SUBMISSION

We try to make the traffic light logic on Verilog. The logic we use is based on the book. Here is the sketch of the FSM with the truth table, which then we converted into logical statement.




We can decide which lights that we turn on based on 3 variables: CURRent state, NSC, EWC. We declared that output 0 is NSG and 1 is EWG. This binary is represented the same in CURR.

We ended with expression $\overline{A}\cdot C+\overline{B}\cdot C$ represents the output NEXT.

We assigned NEXT with the expression above.

hello.v | hello_tb.v | traffic.v | traffic_tb.v

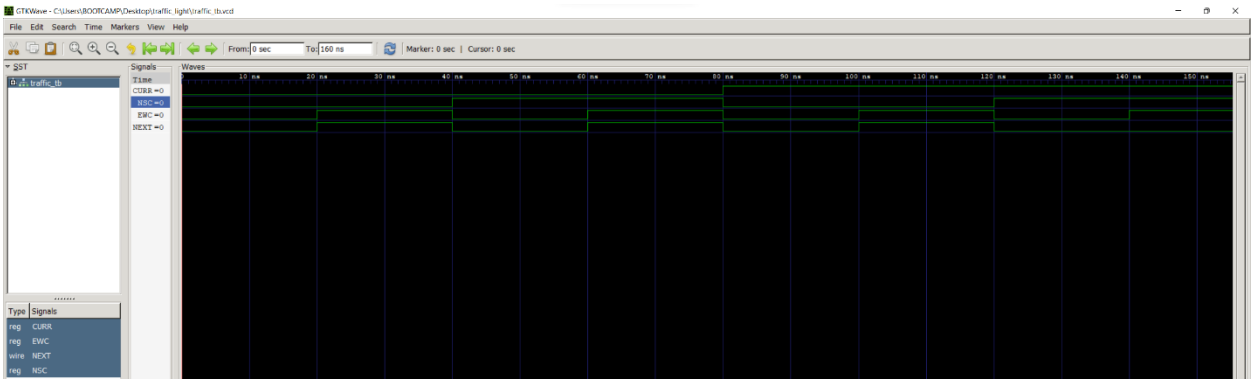
```
1  module traffic (CURR, NSC, EWC, NEXT);
2
3      input CURR, NSC, EWC;
4      output NEXT;
5
6      assign NEXT = ~CURR & EWC | ~NSC & EWC;
7
8  endmodule
```

We made the test file of all of the combination of input, and using advantage of combining input as single decimal number.

hello.v | hello_tb.v | traffic.v | traffic_tb.v

```
1  `timescale 1ns / 1ns
2  `include "traffic.v"
3
4  module traffic_tb;
5
6  reg CURR, NSC, EWC;
7  wire NEXT;
8
9  traffic uut(CURR, NSC, EWC, NEXT);
10
11 initial begin
12
13     $dumpfile("traffic_tb.vcd");
14     $dumpvars(0, traffic_tb);
15
16     {CURR, NSC, EWC} = 3'd0;    #20;
17     {CURR, NSC, EWC} = 3'd1;    #20;
18     {CURR, NSC, EWC} = 3'd2;    #20;
19     {CURR, NSC, EWC} = 3'd3;    #20;
20     {CURR, NSC, EWC} = 3'd4;    #20;
21     {CURR, NSC, EWC} = 3'd5;    #20;
22     {CURR, NSC, EWC} = 3'd6;    #20;
23     {CURR, NSC, EWC} = 3'd7;    #20;
24
25     $display("Test complete");
26
27 end
28
29 endmodule
```

We process and display the test file output using gtkwave and the results are what we expected.




Source used during this project:

[Verilog HDL - Installing and Testing Icarus Verilog + GTKWave]

[Simple Combinational Logic Design in Verilog]