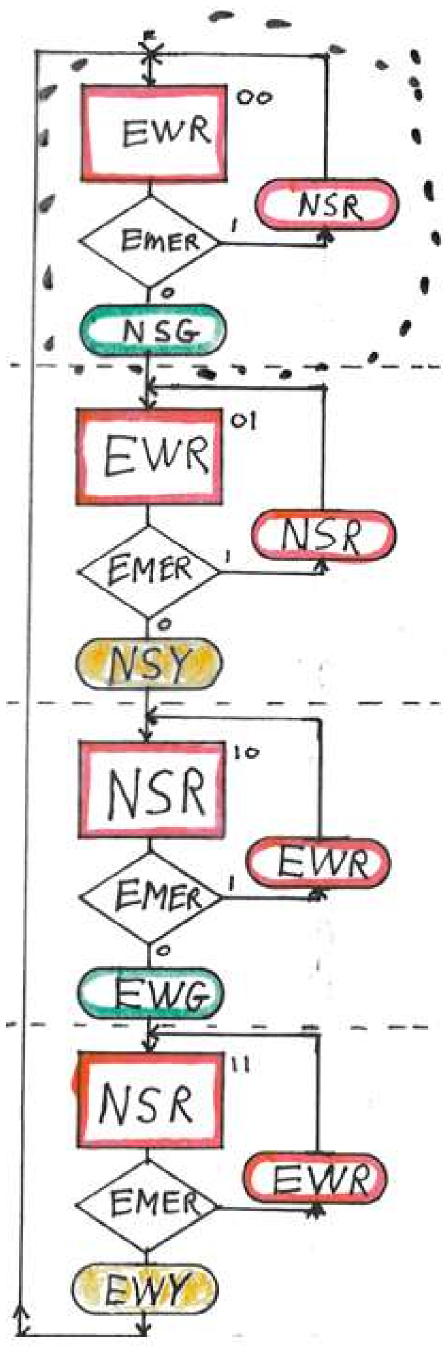
Lab 6 – FPGA Finite State Machine (Mealy Traffic Light)

Below is the Algorithmic State Machine (ASM) chart for a traffic light in the USA at a North-South (NS) street intersection with an East-West (EW) street. Traffic light controller receives emergency radio signal from ambulance that turns both the NS and EW lights red until the ambulance passes through intersection and turns off signal.



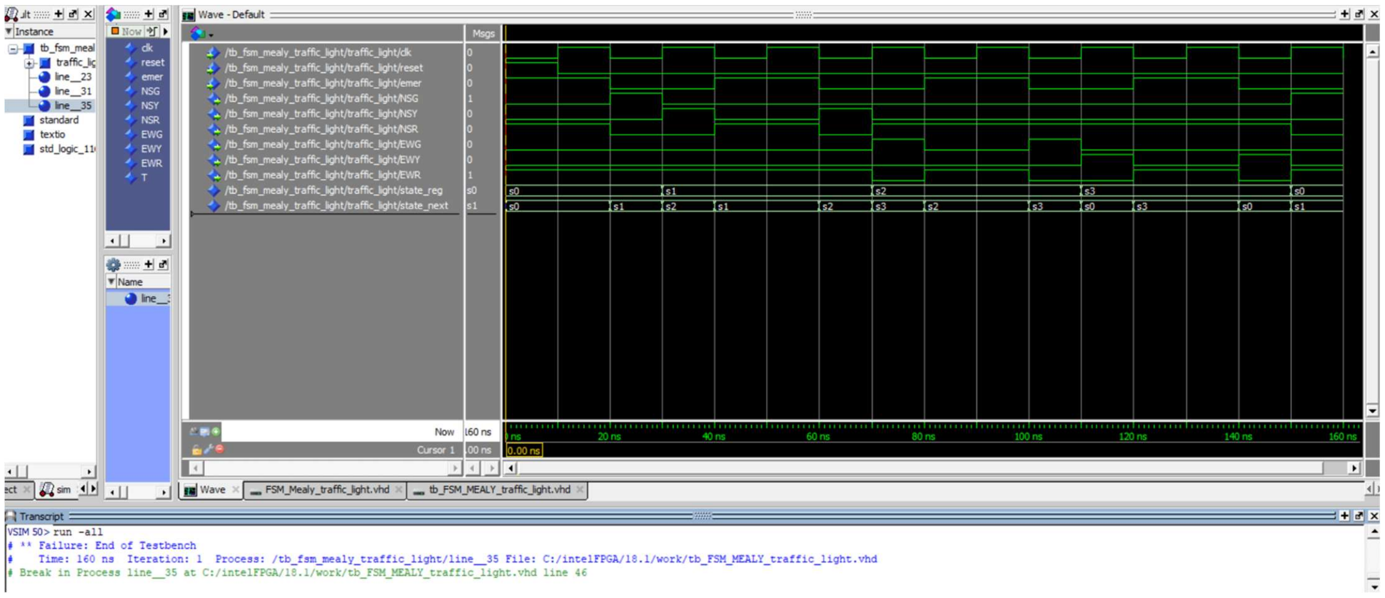
Part 1 of 2 MODELSIM

FSM Mealy Traffic Light – must have 4 states and match the ASM chart exactly. The async reset input should be active high. The emergency input should be active-high (a logic 1 activates the emergency). Create this state machine in ModelSim VHDL using **Chu's 2-segment finite state machine method in my "MCI_VHDL Language_Sakai Summer 2024 Part 2 Sequential and FSM" power point slides 49-53**. (Do **not** make the FSM from a schematic of flip flops and a ROM or gates.) *Use default output assignments so you don't have any inferred latches. To make the code match the ASM chart syntax (easier to write and read), define default values (0) for the lights to be (off) so that your state machine code only shows values for when you turn on the lights. Also define the default value for state_next to be state_reg (stay in the same state).*

Then write a sequential test bench for the FSM (**modify the testbench in my *"*MCI_VHDL Language_Sakai Summer 2024 Part 2 Sequential and FSM*"* power point slides 40-47*).*** Run the test bench in **ModelSim.** Load your test bench file and the FSM component file into ModelSim. *If the testbench can't find your FSM component file, copy the FSM and FSM testbench vhd files to ModelSim's default work directory. Make sure the instantiation work.name matches the component's entity name.* After the testbench and component files compile successfully, start the simulator and select the testbench vhd file.

When you ADD WAVE, right-click on your **component** (not the testbench) so that all signals (includes the state and next state) are displayed (this also helps with debugging). Test all the states and all the ASM chart paths. *Test the emergency path first so that you stay in the same state, then test the normal path.* Do this for each state in order inside one FOR LOOP that handles all four states. Inside the FOR LOOP, set Emer high, wait on a falling edge, then set it low and wait on another falling edge. Do this for all four states. NOTE: (Do not put a For Loop around the clock generation code. It already runs forever.) Simulate with RUN ALL. Stop the simulation with the *assert false, severity failure* commands. Before you run, set the simulation time-step to 20 nS instead of 100 pS. Right-click on the waveform window and select "Zoom Full" from the pop-up menu to see the waveforms better. The simulation should take 160 ns to go through all the paths in the ASM chart. Drag the cursor through the waveforms to see the output pin responses to the test bench input signals.

Submit your vhd design file and your test bench vhd file.
Attach a screenshot of your waveforms. Your output waveform should look similar to the screenshot below. Notice that the current and next states are shown and labeled S0, S1, S2, S3. Make sure you test all 4 states for normal operation, and also test the EMERGENCY input in all four states.



```
VSIM 50> run -all
# ** Failure: End of Testbench
#    Time: 160 ns  Iteration: 1  Process: /tb_fsm_mealy_traffic_light/line__35 File: C:/intelFPGA/18.1/work/tb_FSM_MEALY_traffic_light.vhd
# Break in Process line__35 at C:/intelFPGA/18.1/work/tb_FSM_MEALY_traffic_light.vhd line 46
```

Part 2 of 2 Altera QUARTUS Prime Lite

After your design simulates correctly, create a new project in Quartus and add your FSM traffic light vhd file (you may need to copy it from the ModelSim working directory to the Quartus working directory). When the vhd file is open, make it the top-level file (click "set as top-level entity" on the Project Menu). Compile your design.

NOTE: The directions below could be slightly different than your version of Quartus, but they will be close enough. Select the menu: TOOLS/NETLIST VIEWERS. Select State Machine Viewer to look at the bubble ASM chart. Then click the Encoding tab at the bottom of the window to see the binary state numbering (Quartus defaults to "one- hot" numbering). Select the menu: TOOLS/NETLIST VIEWERS and look at the circuit with the RTL viewer. Also look at the schematic of the resulting circuit using the Quartus Technology Map Viewer – Post Fitting. (You can Zoom In/Out with the View menu.) In the Technology Map, right-click on the combinational blocks and select properties (or use the menu View/Properties). Expand the property windows (wider) located on the left side of the screen to see the logic gates inside the block (not always visible). Select the different tabs on the bottom of the windows such as FAN IN, TRUTH TABLE, EQUATIONS, etc.

Exit the Viewer, and go to the main menu PROCESSING/Compilation Report, expand Timing Analyzer/Slow 1200 mV 0C Model, select *Fmax, Setup Summary and Hold Summary* to see the circuit max clock frequency, setup and hold times (in nS) from input pin to output pin. Red numbers mean requirement was not met at high clock frequencies (we did not constrain the timing, so Quartus could not optimize it). Typically, the "gates" are only about 1 ns each, but the internal signal paths can also have delays of several ns.

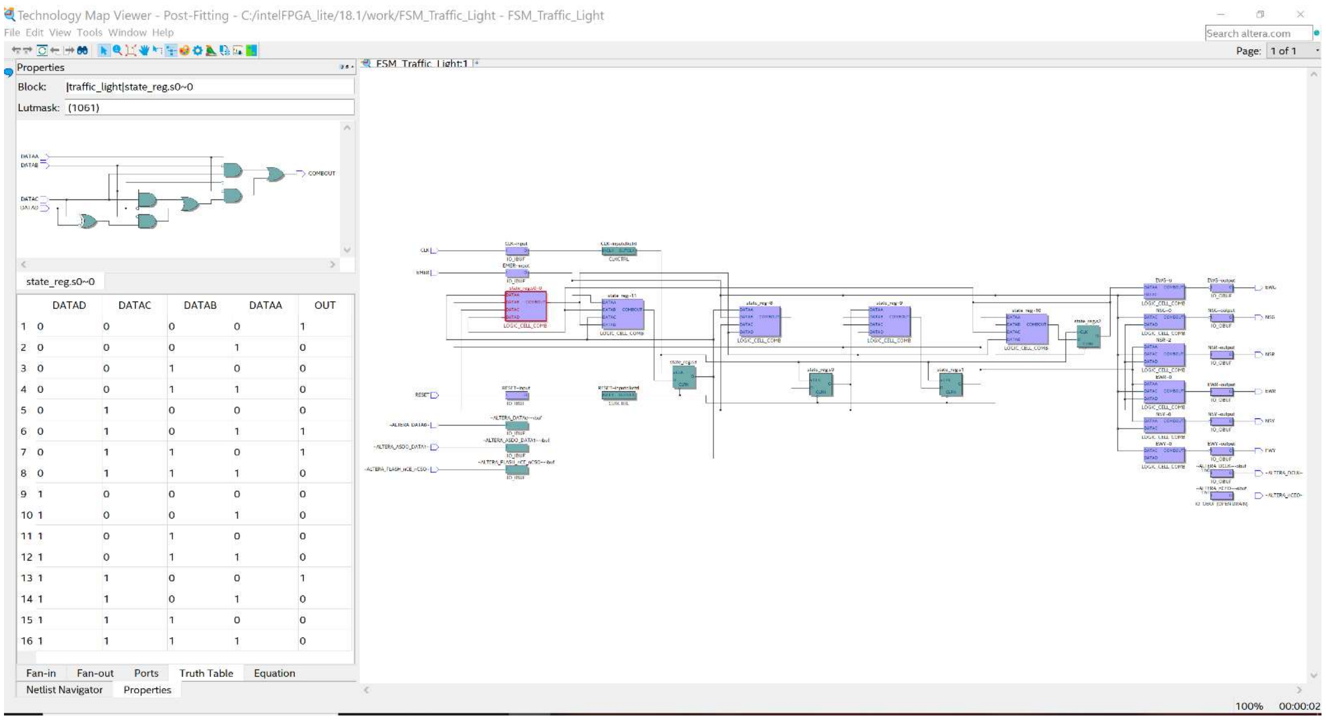
Implement your Traffic Light on your FPGA board. Use 6 of the seven segment leds for the traffic lights as follows: NSR-HEX0[0], NSY-HEX0[6], NSG-HEX0[3], EWR-HEX1[0], EWY-HEX1[6], EWG-HEX1[3]. **The seven-seg display is active low, so change your design file to use zeroes to turn on the lights, and change the default values to one (off).**

Use switch 0 for the reset. *Note: Using a push button for your reset pin will constantly reset your FSM if you use an active-high reset in your VHDL code because the buttons are active-low (use switch 0 instead).* Use switch 1 for the emergency input. **Use pushbutton 3, i.e., KEY[3], for the FSM clock (connect it to your FSM clock input pin). (You will generate each clock pulse by pushing the button.)** When choosing Family & Device Settings, note that the Altera boards DE1, DE2-115 and DE10 have different FPGA chips and pinouts. You can read the device family and chip number off of the FPGA chip on your board.

1. **To prevent damage the board, choose from the menu bar Assignments > Device > "Device and Pin Options…" button > Unused Pins > select "As input tri-stated" > select OK. If all your red leds are faintly lit, then you did not do this correctly.**

Submit a screenshot of the Technology Map for with a property window open showing the gates inside a combinational block. Your image should be similar to the one just below, but the combinatorial circuits will be different because my code is slightly different than this lab assignment.

Demo your traffic light working on the board. Test emergency in all four states.


Technology Map Viewer - Post-Fitting - C:/intelFPGA_lite/18.1/work/FSM_Traffic_Light - FSM_Traffic_Light

Properties

Block: |traffic_light|state_reg.s0~0

Lutmask: (1061)

state_reg.s0~0

	DATAD	DATAC	DATAB	DATAA	OUT
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	1	0	0
4	0	0	1	1	0
5	0	1	0	0	0
6	0	1	0	1	1
7	0	1	1	0	1
8	0	1	1	1	0
9	1	0	0	0	0
10	1	0	0	1	0
11	1	0	1	0	0
12	1	0	1	1	0
13	1	1	0	0	1
14	1	1	0	1	0
15	1	1	1	0	0
16	1	1	1	1	0

```
library ieee;
use ieee.std_logic_1164.all;
--use ieee.numeric_std.all;

entity TrafficLightFSM is
  port (
    i_clk       : in  std_logic;
    i_reset     : in  std_logic;  -- asynchronous, active-high
    i_emergency : in  std_logic;

    o_NS_R : out std_logic;
    o_NS_Y : out std_logic;
    o_NS_G : out std_logic;
    o_EW_R : out std_logic;
    o_EW_Y : out std_logic;
    o_EW_G : out std_logic
  );
end TrafficLightFSM;

architecture TwoSegment of TrafficLightFSM is

  ---------------------------------------------------------------------
  -- State Declaration
```

```
    --------------------------------------------------------------------
    type t_state is (S0, S1, S2, S3);
    signal r_state_reg, r_state_next : t_state;

begin

    --------------------------------------------------------------------
    -- Segment 1: State Register
    --------------------------------------------------------------------
    p_state_register : process(i_clk, i_reset)
    begin
        if i_reset = '1' then
            r_state_reg <= S0;
        elsif (i_clk'event and i_clk = '1') then
            r_state_reg <= r_state_next;
        end if;
    end process;

    --------------------------------------------------------------------
    -- Segment 2: Next-State and Output Logic
    --------------------------------------------------------------------
    p_next_state_logic : process(r_state_reg, i_emergency)
    begin
```

```
-- Defaults
r_state_next <= r_state_reg;
o_NS_R <= '1'; o_NS_Y <= '1'; o_NS_G <= '1';
o_EW_R <= '1'; o_EW_Y <= '1'; o_EW_G <= '1';

case r_state_reg is
  when S0 =>
    o_EW_R <= '0';
    if i_emergency = '1' then
          o_NS_R <= '0';
    else
          o_NS_G <= '0';
      r_state_next <= S1;
    end if;

  when S1 =>
    o_EW_R <= '0';
    if i_emergency = '1' then
          o_NS_R <= '0';
    else
          o_NS_Y <= '0';
      r_state_next <= S2;
    end if;
```

```
        when S2 =>
          o_NS_R <= '0';
          if i_emergency = '1' then
            o_EW_R <= '0';
          else
                o_EW_G <= '0';
            r_state_next <= S3;
          end if;

        when S3 =>
          o_NS_R <= '0';
          if i_emergency = '1' then
            o_EW_R <= '0';
          else
                o_EW_Y <= '0';
            r_state_next <= S0;
          end if;
      end case;
    end process;

end TwoSegment;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_TrafficLightFSM is
end tb_TrafficLightFSM;

architecture behavior of tb_TrafficLightFSM is

  component TrafficLightFSM
    port (
      i_clk       : in  std_logic;
      i_reset     : in  std_logic;
      i_emergency : in  std_logic;

      o_NS_R      : out std_logic;
      o_NS_Y      : out std_logic;
      o_NS_G      : out std_logic;
      o_EW_R      : out std_logic;
      o_EW_Y      : out std_logic;
      o_EW_G      : out std_logic
    );
  end component;
```

```
    -- Testbench signals
    signal i_clk       : std_logic := '0';
    signal i_reset     : std_logic := '0';
    signal i_emergency : std_logic := '0';

    signal o_NS_R, o_NS_Y, o_NS_G : std_logic;
    signal o_EW_R, o_EW_Y, o_EW_G : std_logic;

    constant c_CLK_PERIOD : time := 20 ns;

begin

    -- Instantiate DUT
    uut : TrafficLightFSM
        port map (
            i_clk       => i_clk,
            i_reset     => i_reset,
            i_emergency  => i_emergency,
            o_NS_R      => o_NS_R,
            o_NS_Y      => o_NS_Y,
            o_NS_G      => o_NS_G,
            o_EW_R      => o_EW_R,
            o_EW_Y      => o_EW_Y,
            o_EW_G       => o_EW_G
        );

    -- Clock process
    p_clk : process
    begin
        while true loop
            i_clk <= '0';
            wait for c_CLK_PERIOD / 2;
            i_clk <= '1';
            wait for c_CLK_PERIOD / 2;
        end loop;
    end process;

    -- Stimulus
    process
      begin

        i_reset <= '1';
        wait until (i_clk'event and i_clk = '0');
        i_reset <= '0';
        wait until (i_clk'event and i_clk = '0');
```

```
        i_emergency <= '1';
        wait until (i_clk'event and i_clk = '0');
        i_emergency <= '0';
        wait until (i_clk'event and i_clk = '0');

           i_emergency <= '1';
        wait until (i_clk'event and i_clk = '0');
        i_emergency <= '0';
        wait until (i_clk'event and i_clk = '0');

           i_emergency <= '1';
        wait until (i_clk'event and i_clk = '0');
        i_emergency <= '0';
        wait until (i_clk'event and i_clk = '0');



        -- Stop simulation
        wait for c_CLK_PERIOD;
        assert false report "Simulation finished." severity failure;
    end process;

end behavior;
```