VHDL Top File

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY LogicalStep_Lab5_top IS
    PORT
    (
    clkin_50    : in  std_logic;                    -- The 50 MHz FPGA Clockinput
    rst_n       : in  std_logic;                    -- The RESET input (ACTIVE LOW)
    pb          : in  std_logic_vector(3 downto 0); -- The push-button inputs (ACTIVE LOW)
    sw          : in  std_logic_vector(7 downto 0); -- The switch inputs
    leds        : out std_logic_vector(7 downto 0); -- for displaying the switch content
    seg7_data   : out std_logic_vector(6 downto 0); -- 7-bit outputs to a 7-segment
    seg7_char1  : out std_logic;                    -- seg7 digi selectors
    seg7_char2  : out std_logic                     -- seg7 digi selectors
    );
END LogicalStep_Lab5_top;

ARCHITECTURE SimpleCircuit OF LogicalStep_Lab5_top IS

    component cycle_generator port (
           clkin            : in  std_logic;
           rst_n            : in  std_logic;
           modulo           : in  integer;
           strobe_out       : out std_logic;
           full_cycle_out   : out std_logic
  );
    end component;

    component segment7_mux port (
           clk        : in  std_logic := '0';
           DIN2       : in  std_logic_vector(6 downto 0);
           DIN1       : in  std_logic_vector(6 downto 0);
           DOUT       : out std_logic_vector(6 downto 0);
           DIG2       : out std_logic;
           DIG1       : out std_logic
    );
    end component;

    component Lab5_Moore_SM port (
       clk_input, rst_n                : IN std_logic;
       night_mode                      : IN std_logic;
       reduced_mode                    : IN std_logic;
       ns_latch, ew_latch              : IN std_logic;
       current_value                   : OUT std_logic_vector(4 downto 0)
    );
    end component;

    component synchronizer is port(
    input       : in std_logic;
    rst_n       : in std_logic;
    clk_input   : in std_logic;
    output      : out std_logic
    );
    end component;

    component traffic_latch is port(
    sync_in     : in std_logic;
    clear       : in std_logic;
    clk_input   : in std_logic;
    enable      : in std_logic;
    rst_n       : in std_logic;
    output      : out std_logic
    );
    end component;

-------------------------------------------------------------------------------------------------
    CONSTANT SIM                    :  boolean := TRUE ;

    CONSTANT CNTR1_modulo           :  integer := 25000000;       -- modulo count for 1Hz cycle generator 1 with 50Mhz clocking input
    CONSTANT CNTR2_modulo           :  integer := 5000000;        -- modulo count for 5Hz cycle generator 2 with 50Mhz clocking input
    CONSTANT CNTR1_modulo_sim       :  integer := 199;            -- modulo count for cycle generator 1 during simulation
    CONSTANT CNTR2_modulo_sim       :  integer :=  39;            -- modulo count for cycle generator 2 during simulation

    SIGNAL CNTR1_modulo_value       :  integer ;                  -- modulo count for cycle generator 1
    SIGNAL CNTR2_modulo_value       :  integer ;                  -- modulo count for cycle generator 2

    SIGNAL clken1,clken2            :  STD_LOGIC;                 -- clock enables 1 & 2

    SIGNAL strobe1, strobe2         :  std_logic;                 -- strobes 1 & 2 with each one being 50% Duty Cycle

    SIGNAL SM_output                :  std_logic_vector(4 downto 0); -- signal for what state the LTC is at

    SIGNAL seg7_A, seg7_B           :  STD_LOGIC_VECTOR(6 downto 0); -- signals for inputs into seg7_mux.

    SIGNAL sync_out1, sync_out2     : std_logic;

    SIGNAL clearNS, clearEW         : std_logic;

    SIGNAL NS_out, EW_out           : std_logic;

BEGIN
-------------------------------------------------------------------------------------------------
```

```
MODULO_1_SELECTION:  CNTR1_modulo_value <= CNTR1_modulo when SIM = FALSE else CNTR1_modulo_sim;

MODULO_2_SELECTION:  CNTR2_modulo_value <= CNTR2_modulo when SIM = FALSE else CNTR2_modulo_sim;


-------------------------------------------------------------------------------------------------------
-- Component Hook-up:

GEN1:    cycle_generator port map(clkin_50, rst_n, CNTR1_modulo_value, strobe1, clken1);
GEN2:    cycle_generator port map(clkin_50, rst_n, CNTR2_modulo_value, strobe2, clken2);

SM: Lab5_Moore_SM port map(strobe1, rst_n, sw(0), sw(1), NS_out, EW_out, SM_output);
MUX: segment7_mux port map(clkin_50, seg7_A, seg7_B, seg7_data, seg7_char1, seg7_char2);

SYNC1: synchronizer port map(not pb(0), rst_n, clkin_50, sync_out1);
SYNC2: synchronizer port map(not pb(1), rst_n, clkin_50, sync_out2);

LATCH1: traffic_latch port map(sync_out1, clearNS, clkin_50, clken2, rst_n, NS_out);
LATCH2: traffic_latch port map(sync_out2, clearEW, clkin_50, clken2, rst_n, EW_out);

leds(1 downto 0) <= Strobe1 & Strobe2;
leds(6 downto 2) <= SM_output;
leds(7) <= NS_out or EW_out;

with SM_output select
   clearNS <= '1' when "01111", -- reset latch at state 15
              '0' when others;
with SM_output select
   clearEW <= '1' when "00111", -- reset latch at state 7
              '0' when others;


-- used for simulations
-- leds(0) <= clken1;
-- leds(1) <= Strobe1;
-- leds(2) <= clken2;
-- leds(3) <= Strobe2;
-- leds(7 downto 4) <= State Machine state numbers
134
135   WITH SM_output select
136      -- the NS TLC
137      -- red 1, amber 7, green 4
138      seg7_A <=
139         ---------- G flash
140         "000" & strobe2 & "000" when "00000", -- strobe2 is the 5Hz strobe signal
141         "000" & strobe2 & "000" when "00001",
142         ---------- G solid
143         "0001000" when "00010",
144         "0001000" when "00011",
145         "0001000" when "00100",
146         "0001000" when "00101",
147         ---------- A solid
148         "1000000" when "00110",
149         "1000000" when "00111",
150         ---------- R solid
151         "0000001" when "01000",
152         "0000001" when "01001",
153         "0000001" when "01010",
154         "0000001" when "01011",
155         "0000001" when "01100",
156         "0000001" when "01101",
157         "0000001" when "01110",
158         "0000001" when "01111",
159         ---------- night mode (NS is green)
160         "0001000" when "10000",
161         ---------- reduced mode (NS is slow flash A)
162         strobe1 & "000000" when others;
163
```

```
164   WITH SM_output select
165      -- the EW TLC
166      seg7_B <=
167         ---------- R solid
168         "0000001" when "00000",
169         "0000001" when "00001",
170         "0000001" when "00010",
171         "0000001" when "00011",
172         "0000001" when "00100",
173         "0000001" when "00101",
174         "0000001" when "00110",
175         "0000001" when "00111",
176         ---------- G flash
177         "000" & strobe2 & "000" when "01000",
178         "000" & strobe2 & "000" when "01001",
179         ---------- G solid
180         "0001000" when "01010",
181         "0001000" when "01011",
182         "0001000" when "01100",
183         "0001000" when "01101",
184         ---------- A solid
185         "1000000" when "01110",
186         "1000000" when "01111",
187         ---------- night mode (EW is red)
188         "0000001" when "10000",
189         ---------- reduced mode (NS is slow flash R)
190         "000000" & strobe1 when others;
191
192   END SimpleCircuit;
193
```

Moore SM

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

Entity Lab5_Moore_SM IS Port
(
 clk_input, rst_n              : IN std_logic;
 night_mode                    : IN std_logic;
 reduced_mode                  : IN std_logic;
 ns_latch, ew_latch            : IN std_logic;
 current_value                 : OUT std_logic_vector(4 downto 0)
 );
 END ENTITY;

Architecture MSM of Lab5_Moore_SM is
  TYPE STATE_NAMES IS (S0, S1, S2, S3, S4, S5, S6, S7,
                       S8, S9, S10, S11, S12, S13, S14, S15, S16, S17); -- list all the STATES


   signal current_state,next_state   :  STATE_NAMES;   -- signals of type STATE_NAMES
   signal super_night_mode : std_logic;

BEGIN
-------------------------------------------------------------------------------
 --State Machine:
 -------------------------------------------------------------------------------

-- REGISTER LOGIC PROCESS
 -- add clock and any related inputs for state machine register section into Sensitivity List

Register_Section: PROCESS (clk_input, rst_n,next_state)  -- this process synchronizes the activity to a clock
 BEGIN
   IF (rst_n = '0') THEN
      current_state <= S0;
   ELSIF(rising_edge(clk_input)) THEN
      current_state <=next_state;
         ELSE
      current_state <= current_state;
         END IF;
 END PROCESS;

PROCESS(night_mode, reduced_mode)
 BEGIN
   if (night_mode = '1') and (reduced_mode = '1') then
      super_night_mode <= '0';
   else
      super_night_mode <= night_mode;
   end if;
 END PROCESS;

-- TRANSTION LOGIC PROCESS (to be combinational only)
 -- add all transition inputs for state machine into Transition section Sensitivity List
 -- make sure that all conditional statement options are complete otherwise VHDL will infer LATCHES.

Transition_Section: PROCESS (current_state)

-- NS monitors latch during S8 to 12; if latch output = 1; go to S14 so that theres amber light, at S15, send latch clear to latch
 -- SE monitors latch during S0 to 4; go to S6, then at S7, send latch clear

 BEGIN
     CASE current_state IS
      -- part D MSM
          WHEN S0 =>
         if (ew_latch = '1') THEN
            next_state <= S6;
         else
            next_state <= S1;
         end if;
       WHEN S1 =>
         if (ew_latch = '1') THEN
            next_state <= S6;
         else
            next_state <= S2;
         end if;
       WHEN S2 =>
         if (ew_latch = '1') THEN
            next_state <= S6;
         else
            next_state <= S3;
         end if;
       WHEN S3 =>
         if (ew_latch = '1') THEN
            next_state <= S6;
         else
            next_state <= S4;
         end if;
       WHEN S4 =>
         if (ew_latch = '1') THEN
            next_state <= S6;
         else
            next_state <= S5;
         end if;
       WHEN S5 =>
          next_state <= S6;
       WHEN S6 =>
          next_state <= S7;
```

```
147         WHEN S16 =>
148            if (super_night_mode = '0') then
149               next_state <= S6;
150            else
151               next_state <= S16;
152            end if;
153         WHEN S17 =>
154            if (reduced_mode = '0') then
155               next_state <= S6;
156            else
157               next_state <= S17;
158            end if;
159         WHEN others =>
160             next_state <= S0;
161         END CASE;
162
163
164   END PROCESS;
165
166   Decoder_Section: PROCESS(current_state)
167
168   BEGIN -- based on current state, assign 4-bit value to current-value
169     CASE current_state IS
170       ------------------ G flash
171       WHEN S0 =>
172         current_value <= "00000";
173       WHEN S1 =>
174         current_value <= "00001";
175       ------------------ G solid
176       WHEN S2 =>
177         current_value <= "00100";
178       WHEN S3 =>
179         current_value <= "00011";
180       WHEN S4 =>
181         current_value <= "00100";
182       WHEN S5 =>
183         current_value <= "00101";
184       ------------------ A solid
185       WHEN S6 =>
186         current_value <= "00110";
187       WHEN S7 =>
188         current_value <= "00111";
189       ------------------ R solid
190       WHEN S8 =>
191         current_value <= "01000";
192       WHEN S9 =>
193         current_value <= "01001";
194       WHEN S10 =>
195         current_value <= "01010";
```

```
200         WHEN S15 =>
201            current_value <= "01111";
202         WHEN S16 =>
203            current_value <= "10000";
204         WHEN S17 =>
205            current_value <= "10001";
206     END CASE;
207   END PROCESS;
208
209   END ARCHITECTURE MSM;
```

Cycle Generator

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

Entity cycle_generator IS port (
          clkin            : in  std_logic;
          rst_n            : in  std_logic;
          modulo           : in  integer;
          strobe_out       : out std_logic;
          full_cycle_out   : out std_logic
   );
end entity;

ARCHITECTURE counter OF cycle_generator IS

    SIGNAL bin_counter               : UNSIGNED(31 DOWNTO 0);
    SIGNAL terminal_count            : std_logic;
    SIGNAL half_cycle, full_cycle    : std_logic;
    SIGNAL strobe                    : std_logic;

   BEGIN

    half_cycle <= terminal_count; -- outputs a 1 once a period; otherwise 0
    strobe_out <= strobe;         -- toggle every period
    full_cycle_out <= full_cycle; -- toggle every 2 periods


MODULO_COUNTING: PROCESS(clkin, rst_n) IS
    BEGIN

     IF (rst_n = '0') THEN
        bin_counter <= to_unsigned(modulo,32);
        terminal_count <= '0';

     ELSIF (rising_edge(clkin)) THEN               -- binary counter decrements on rising clock edge.

         IF(bin_counter = 0) THEN
                                                   -- when bin_counter reaches 0
            bin_counter <= to_unsigned(modulo,32); -- reload the (converted integer to 32 bit unsigned signal type) modulo value
            terminal_count <= '1';                 -- and output a terminal_count signal
         ELSE
            bin_counter <= bin_counter - 1;
            terminal_count <= '0';
         END IF;
     END IF;
    END PROCESS;


Strobe_gen: PROCESS(clkin, rst_n) IS         -- Strobe is with 50% duty cycle
   BEGIN
   IF (rst_n = '0') THEN
      strobe <= '0';

   ELSIF (rising_edge(clkin)) THEN

      IF (half_cycle = '1') and (strobe = '1') THEN
         strobe <= '0';
      ELSIF (half_cycle = '1' and strobe = '0') THEN
         strobe <= '1';
      END IF;
   END IF;
   END PROCESS;


CLKEN_GEN: PROCESS(clkin, rst_n) IS          -- full_cycle is one "clkin" cycle in duration and occure once for every two occurrences of half_cycle
   BEGIN
   IF (rst_n = '0') THEN
      full_cycle <= '0';

   ELSIF (rising_edge(clkin)) THEN
      IF (half_cycle = '1') and (strobe = '1') and (full_cycle = '1') THEN    --when cycle_count is 1 i.e. every 2nd cycle, set full_cycle to 1
         full_cycle <= '0';
      ELSIF (half_cycle = '1') and (strobe = '1') and (full_cycle = '0') THEN
         full_cycle <= '1';
      END IF;

   END IF;
   END PROCESS;
END Architecture;
```

D Flip Flop

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

entity d_flip_flop is port(
    clk_input   : in std_logic;  -- 50Mhz Clock
    rst_n       : in std_logic;
    D           : in std_logic;  --input from pb (can be metastable)
    Q           : out std_logic -- temp output (not metastable)
);
end d_flip_flop;

ARCHITECTURE behaviour OF d_flip_flop IS
begin

PROCESS (clk_input) is
    begin
        if rising_edge(clk_input) then -- by definition of a D flip flop
            if (rst_n = '0') then
                Q <= '0';
            else
                Q <= D;
            end if;
        end if;
end process;

END ARCHITECTURE behaviour;
```

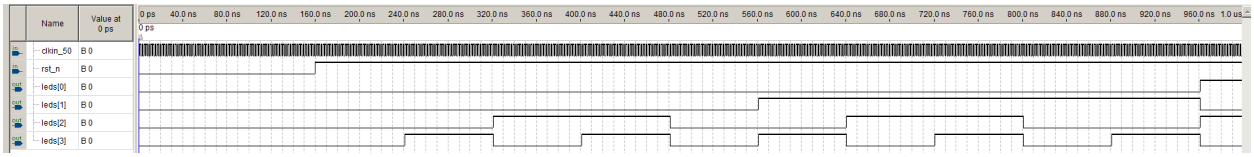
Enabled D Flip Flop

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

entity en_dff is port(
    clk_input   : in std_logic;  -- 50Mhz Clock
    enable      : in std_logic;  -- enable
    rst_n       : in std_logic;  -- reset
    D           : in std_logic;  --input from pb (can be metastable)
    Q           : out std_logic -- temp output (not metastable)
);
end en_dff;

ARCHITECTURE behaviour OF en_dff IS
begin

PROCESS (clk_input) is
    begin
        if rising_edge(clk_input) then -- by definition of a D flip flop
            if (rst_n = '0') or (enable = '1') then -- Q = 0 of reset or clock not enabled
                Q <= '0';
            else
                Q <= D;
            end if;
        end if;
end process;

END ARCHITECTURE behaviour;
```

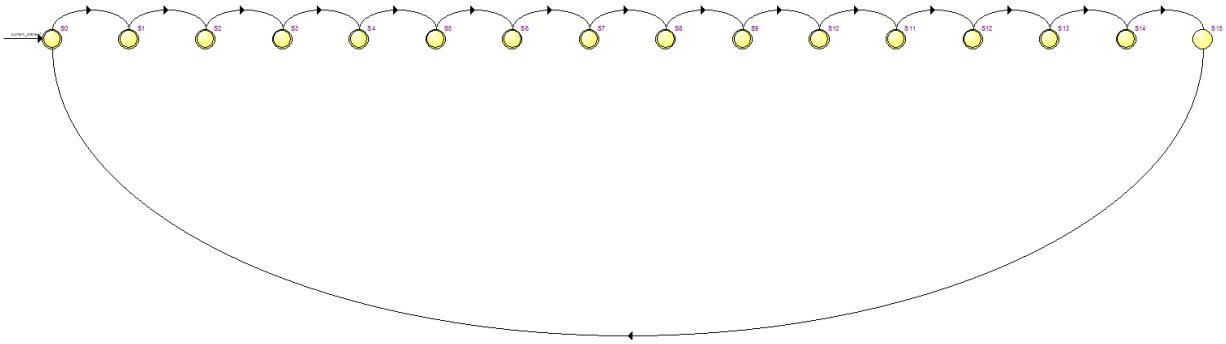
Synchronizer

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

entity synchronizer is port(
    input       : in std_logic;
    rst_n       : in std_logic;
    clk_input   : in std_logic;
    output      : out std_logic
);
end synchronizer;

ARCHITECTURE behaviour OF synchronizer IS

component d_flip_flop is port(
    clk_input   : in std_logic;
    rst_n       : in std_logic;
    D           : in std_logic;
    Q           : out std_logic
    );
end component;

signal temp: std_logic;

begin
    DFF1: d_flip_flop port map(clk_input, rst_n, input, temp); -- slave flip flop
    DFF2: d_flip_flop port map(clk_input, rst_n, temp, output); -- master flip flop

END ARCHITECTURE behaviour;

```

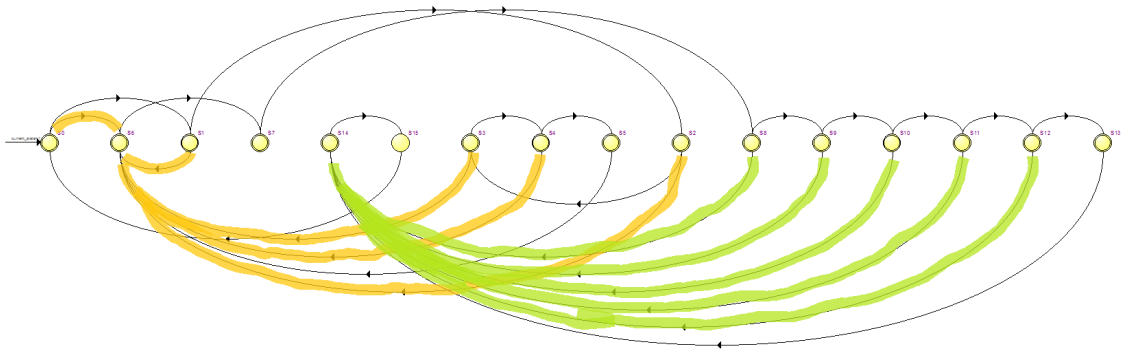
Simulation from Part A



State Diagram from Part B



State Diagram from Part C



State Diagram from Part D