

VHDL Top File

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4
5  ENTITY LogicalStep_Lab5_top IS
6  PORT
7  (
8      clk_50      : in  std_logic;           -- The 50 MHz FPGA Clockinput
9      rst_n       : in  std_logic;           -- The RESET input (ACTIVE LOW)
10     pb          : in  std_logic_vector(3 downto 0); -- The push-button inputs (ACTIVE LOW)
11     sw          : in  std_logic_vector(7 downto 0); -- The switch inputs
12     leds        : out std_logic_vector(7 downto 0); -- for displaying the switch content
13     seg7_data    : out std_logic_vector(6 downto 0); -- 7-bit outputs to a 7-segment
14     seg7_char1   : out std_logic;           -- seg7 digi selectors
15     seg7_char2   : out std_logic;           -- seg7 digi selectors
16 );
17 END LogicalStep_Lab5_top;
18
19 ARCHITECTURE SimpleCircuit OF LogicalStep_Lab5_top IS
20
21     component cycle_generator port (
22         clk_in      : in  std_logic;
23         rst_n       : in  std_logic;
24         modulo       : in  integer;
25         strobe_out   : out std_logic;
26         full_cycle_out : out std_logic
27     );
28     end component;
29
30     component segment7_mux port (
31         clk          : in  std_logic := '0';
32         DIN2         : in  std_logic_vector(6 downto 0);
33         DIN1         : in  std_logic_vector(6 downto 0);
34         DOUT         : out std_logic_vector(6 downto 0);
35         DIG2         : out std_logic;
36         DIG1         : out std_logic
37     );
38     end component;
39
40     component Lab5_Moore_SM port (
41         clk_input, rst_n      : IN std_logic;
42         night_mode           : IN std_logic;
43         reduced_mode         : IN std_logic;
44         ns_latch, ew_latch   : IN std_logic;
45         current_value        : OUT std_logic_vector(4 downto 0)
46     );
47     end component;
48
49     component synchronizer is port(
50         input      : in std_logic;
51         rst_n      : in std_logic;
52         clk_input   : in std_logic;
53         output      : out std_logic
54     );
55     end component;
56
57     component traffic_latch is port(
58         sync_in     : in std_logic;
59         clear       : in std_logic;
60         clk_input    : in std_logic;
61         enable      : in std_logic;
62         rst_n       : in std_logic;
63         output      : out std_logic
64     );
65     end component;
66
67     -----
68     CONSTANT SIM : boolean := TRUE ;
69
70     CONSTANT CNTR1_modulo : integer := 25000000; -- modulo count for 1Hz cycle generator 1 with 50Mhz clocking input
71     CONSTANT CNTR2_modulo : integer := 5000000;  -- modulo count for 5Hz cycle generator 2 with 50Mhz clocking input
72     CONSTANT CNTR1_modulo_sim : integer := 199;  -- modulo count for cycle generator 1 during simulation
73     CONSTANT CNTR2_modulo_sim : integer := 39;    -- modulo count for cycle generator 2 during simulation
74
75     SIGNAL CNTR1_modulo_value : integer ; -- modulo count for cycle generator 1
76     SIGNAL CNTR2_modulo_value : integer ; -- modulo count for cycle generator 2
77
78     SIGNAL clken1, clken2 : STD_LOGIC; -- clock enables 1 & 2
79
80     SIGNAL strobe1, strobe2 : std_logic; -- strobes 1 & 2 with each one being 50% Duty Cycle
81
82     SIGNAL SM_output : std_logic_vector(4 downto 0); -- signal for what state the LTC is at
83
84     SIGNAL seg7_A, seg7_B : STD_LOGIC_VECTOR(6 downto 0); -- signals for inputs into seg7_mux.
85
86     SIGNAL sync_out1, sync_out2 : std_logic;
87
88     SIGNAL clearNS, clearEW : std_logic;
89
90     SIGNAL NS_out, EW_out : std_logic;
91
92     BEGIN
93
94     -----
95

```

```

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 "0111", -- reset latch at state 15
            '0' when others;

```

```

with SM_output select
  clearEW <= '1' when "0011", -- 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 WITH SM_output select
135 -- the NS TLC
136 -- red 1, amber 7, green 4
137 seg7_A <=
138 ----- G flash
139 "000" & strobe2 & "000" when "00000", -- strobe2 is the 5Hz strobe signal
140 "000" & strobe2 & "000" when "00001",
141 ----- G solid
142 "0001000" when "00010",
143 "0001000" when "00011",
144 "0001000" when "00100",
145 "0001000" when "00101",
146 ----- A solid
147 "1000000" when "00110",
148 "1000000" when "00111",
149 ----- R solid
150 "0000001" when "01000",
151 "0000001" when "01001",
152 "0000001" when "01010",
153 "0000001" when "01011",
154 "0000001" when "01100",
155 "0000001" when "01101",
156 "0000001" when "01110",
157 "0000001" when "01111",
158 ----- night mode (NS is green)
159 "0001000" when "10000",
160 ----- reduced mode (NS is slow flash A)
161 strobe1 & "000000" when others;
162
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

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  Entity Lab5_Moore_SM IS Port
6  (
7      clk_input, rst_n          : IN std_logic;
8      night_mode                : IN std_logic;
9      reduced_mode               : IN std_logic;
10     ns_latch, ew_latch         : IN std_logic;
11     current_value              : OUT std_logic_vector(4 downto 0)
12 );
13 END ENTITY;
14
15 Architecture MSM of Lab5_Moore_SM is
16     TYPE STATE_NAMES IS (S0, S1, S2, S3, S4, S5, S6, S7,
17                           S8, S9, S10, S11, S12, S13, S14, S15, S16, S17); -- list all the STATES
18
19     signal current_state,next_state : STATE_NAMES; -- signals of type STATE_NAMES
20     signal super_night_mode : std_logic;
21
22 BEGIN
23
24     -----
25     --State Machine:
26     -----
27
28     -- REGISTER LOGIC PROCESS
29     -- add clock and any related inputs for state machine register section into Sensitivity List
30
31     Register_Section: PROCESS (clk_input, rst_n,next_state) -- this process synchronizes the activity to a clock
32     BEGIN
33         IF (rst_n = '0') THEN
34             current_state <= S0;
35         ELSIF(rising_edge(clk_input)) THEN
36             current_state <= next_state;
37         ELSE
38             current_state <= current_state;
39         END IF;
40     END PROCESS;
41
42     PROCESS(night_mode, reduced_mode)
43     BEGIN
44         if (night_mode = '1') and (reduced_mode = '1') then
45             super_night_mode <= '0';
46         else
47             super_night_mode <= night_mode;
48         end if;
49     END PROCESS;
50
51     -- TRANSITION LOGIC PROCESS (to be combinational only)
52     -- add all transition inputs for state machine into Transition section Sensitivity List
53     -- make sure that all conditional statement options are complete otherwise VHDL will infer LATCHES.
54
55     Transition_Section: PROCESS (current_state)
56     BEGIN
57         -- 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
58         -- SE monitors latch during S0 to 4; go to S6, then at S7, send latch clear
59
60         BEGIN
61             CASE current_state IS
62                 -- part D MSM
63                 WHEN S0 =>
64                     if (ew_latch = '1') THEN
65                         next_state <= S6;
66                     else
67                         next_state <= S1;
68                     end if;
69                 WHEN S1 =>
70                     if (ew_latch = '1') THEN
71                         next_state <= S6;
72                     else
73                         next_state <= S2;
74                     end if;
75                 WHEN S2 =>
76                     if (ew_latch = '1') THEN
77                         next_state <= S6;
78                     else
79                         next_state <= S3;
80                     end if;
81                 WHEN S3 =>
82                     if (ew_latch = '1') THEN
83                         next_state <= S6;
84                     else
85                         next_state <= S4;
86                     end if;
87                 WHEN S4 =>
88                     if (ew_latch = '1') THEN
89                         next_state <= S6;
90                     else
91                         next_state <= S5;
92                     end if;
93                 WHEN S5 =>
94                     next_state <= S6;
95                 WHEN S6 =>
96                     next_state <= S7;
97                 ...

```

```

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";
196     WHEN S11 =>
197         current_value <= "01011";
198     WHEN S12 =>
199         current_value <= "01100";
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

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4
5  Entity cycle_generator IS port (
6      clk_in      : in std_logic;
7      rst_n       : in std_logic;
8      modulo      : in integer;
9      strobe_out   : out std_logic;
10     full_cycle_out : out std_logic
11 );
12 end entity;
13
14 ARCHITECTURE counter OF cycle_generator IS
15
16     SIGNAL bin_counter      : UNSIGNED(31 DOWNTO 0);
17     SIGNAL terminal_count    : std_logic;
18     SIGNAL half_cycle, full_cycle : std_logic;
19     SIGNAL strobe            : std_logic;
20
21 BEGIN
22
23     half_cycle <= terminal_count; -- outputs a 1 once a period; otherwise 0
24     strobe_out <= strobe;         -- toggle every period
25     full_cycle_out <= full_cycle; -- toggle every 2 periods
26
27
28 MODULE_COUNTING: PROCESS(clkin, rst_n) IS
29     BEGIN
30
31         IF (rst_n = '0') THEN
32             bin_counter <= to_unsigned(modulo,32);
33             terminal_count <= '0';
34
35         ELSIF (rising_edge(clkin)) THEN -- binary counter decrements on rising clock edge.
36
37             IF(bin_counter = 0) THEN
38                 -- when bin_counter reaches 0
39                 bin_counter <= to_unsigned(modulo,32); -- reload the (converted integer to 32 bit unsigned signal type) modulo value
40                 terminal_count <= '1'; -- and output a terminal_count signal
41             ELSE
42                 bin_counter <= bin_counter - 1;
43                 terminal_count <= '0';
44             END IF;
45         END IF;
46     END PROCESS;
47
48 Strobe_gen: PROCESS(clkin, rst_n) IS -- Strobe is with 50% duty cycle
49     BEGIN
50         IF (rst_n = '0') THEN
51             strobe <= '0';
52
53         ELSIF (rising_edge(clkin)) THEN
54
55             IF (half_cycle = '1') and (strobe = '1') THEN
56                 strobe <= '0';
57             ELSIF (half_cycle = '1' and strobe = '0') THEN
58                 strobe <= '1';
59             END IF;
60         END IF;
61     END PROCESS;
62
63
64 CLKEN_GEN: PROCESS(clkin, rst_n) IS -- full_cycle is one "clkin" cycle in duration and occurs once for every two occurrences of half_cycle
65     BEGIN
66         IF (rst_n = '0') THEN
67             full_cycle <= '0';
68
69         ELSIF (rising_edge(clkin)) THEN
70             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
71                 full_cycle <= '0';
72             ELSIF (half_cycle = '1') and (strobe = '1') and (full_cycle = '0') THEN
73                 full_cycle <= '1';
74             END IF;
75         END IF;
76     END IF;
77 END PROCESS;
78
79 END Architecture;
80
81
```

D Flip Flop

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity d_flip_flop is port(
6      clk_input  : in std_logic;  -- 50Mhz Clock
7      rst_n      : in std_logic;
8      D          : in std_logic;  --input from pb (can be metastable)
9      Q          : out std_logic -- temp output (not metastable)
10 );
11 end d_flip_flop;
12
13 ARCHITECTURE behaviour OF d_flip_flop IS
14 begin
15
16 PROCESS (clk_input) is
17 begin
18     if rising_edge(clk_input) then -- by definition of a D flip flop
19         if (rst_n = '0') then
20             Q <= '0';
21         else
22             Q <= D;
23         end if;
24     end if;
25 end process;
26
27 END ARCHITECTURE behaviour;
```

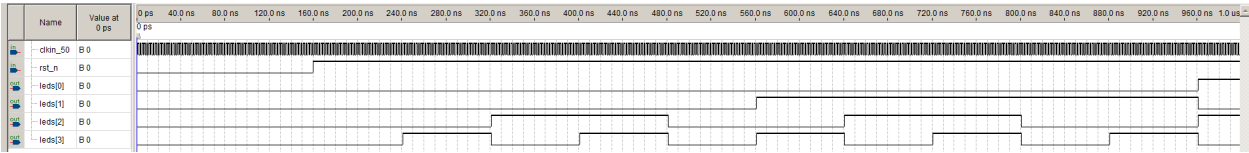
Enabled D Flip Flop

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity en_dff is port(
6      clk_input  : in std_logic;  -- 50Mhz Clock
7      enable     : in std_logic;  -- enable
8      rst_n      : in std_logic;  -- reset
9      D          : in std_logic;  --input from pb (can be metastable)
10     Q          : out std_logic -- temp output (not metastable)
11 );
12 end en_dff;
13
14 ARCHITECTURE behaviour OF en_dff IS
15 begin
16
17 PROCESS (clk_input) is
18 begin
19     if rising_edge(clk_input) then -- by definition of a D flip flop
20         if (rst_n = '0') or (enable = '1') then -- Q = 0 of reset or clock not enabled
21             Q <= '0';
22         else
23             Q <= D;
24         end if;
25     end if;
26 end process;
27
28 END ARCHITECTURE behaviour;
```

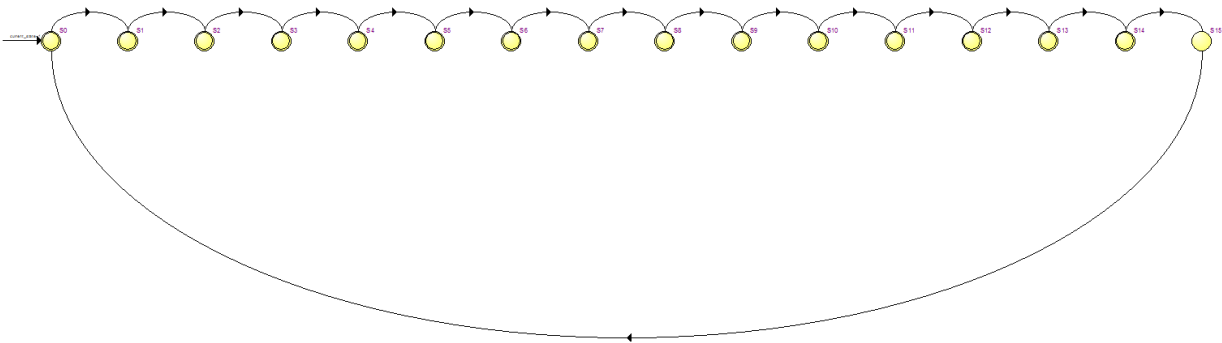

Synchronizer

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity synchronizer is port(
6      input      : in std_logic;
7      rst_n      : in std_logic;
8      clk_input  : in std_logic;
9      output     : out std_logic
10 );
11 end synchronizer;
12
13
14 ARCHITECTURE behaviour OF synchronizer IS
15
16     component d_flip_flop is port(
17         clk_input  : in std_logic;
18         rst_n      : in std_logic;
19         D           : in std_logic;
20         Q           : out std_logic
21     );
22     end component;
23
24     signal temp: std_logic;
25
26     begin
27         DFF1: d_flip_flop port map(clk_input, rst_n, input, temp); -- slave flip flop
28         DFF2: d_flip_flop port map(clk_input, rst_n, temp, output); -- master flip flop
29     |
30 END ARCHITECTURE behaviour;
```

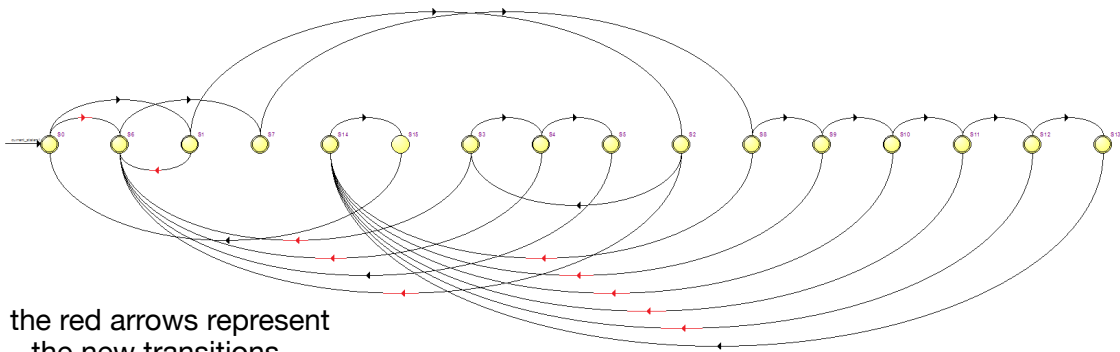
Simulation from Part A



State Diagram from Part B

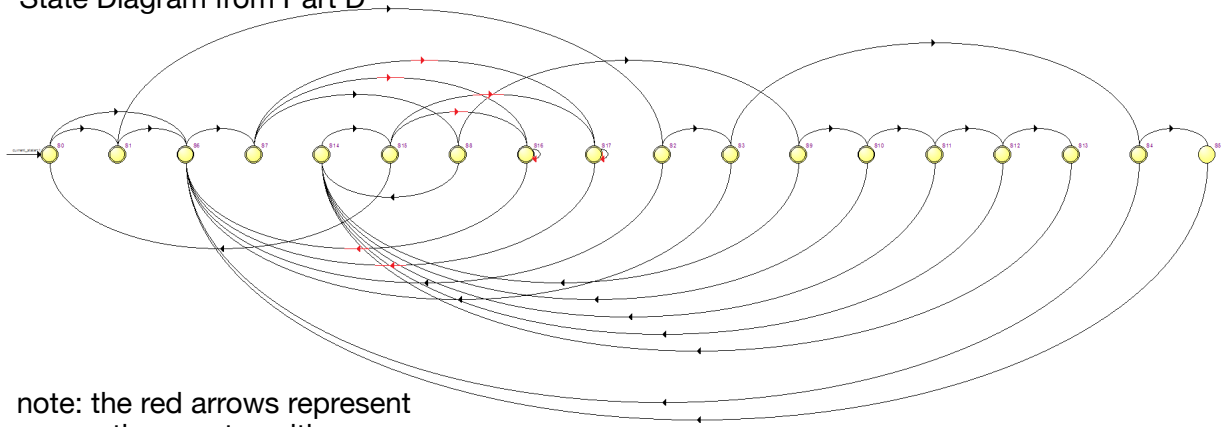


State Diagram from Part C



note: the red arrows represent the new transitions

State Diagram from Part D



note: the red arrows represent
the new transitions