Structural VHDL Design Top File

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

ENTITY LogicalStep_Lab4_top IS
  PORT(
    clkin_50    : in  std_logic;
    rst_n       : in  std_logic;
    pb          : in  std_logic_vector(3 downto 0);
    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_Lab4_top;

ARCHITECTURE SimpleCircuit OF LogicalStep_Lab4_top IS
  component compx4 port (
    a0, b0, a1, b1, a2, b2, a3, b3   : in std_logic;
    a_less_b, a_equal_b, a_more_b    : out std_logic
    );
  end component;

  component Moore_SM port (
    clk_input, rst_n               : IN std_logic;
    MORE, EQUAL, LESS            : IN std_logic;
    current_value                : OUT std_logic_vector(3 downto 0)
    );
    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 SevenSegment port (
      hex       :  in  std_logic_vector(3 downto 0);   -- digit 1 shows state number in hex of state machine; if they don't match; it changes
      sevenseg  :  out std_logic_vector(6 downto 0)    -- target is sw [3..0] and displayed on right digit
      );
    end component;

  ------------------------------------------------------------------------------------------------
    CONSTANT SIM                    : boolean := FALSE;  -- set to TRUE for simulation runs otherwise keep at 0.
    CONSTANT  CLK_DIV_SIZE          : INTEGER := 24;  -- size of vectors for the counters

    signal   Main_CLK               : STD_LOGIC;         -- main clock to drive sequencing of State Machine

    signal   bin_counter            : UNSIGNED(CLK_DIV_SIZE-1 downto 0); -- := to_unsigned(0,CLK_DIV_SIZE); -- reset binary counter to zero

    signal   Simple_States          : std_logic_vector(7 downto 4);
    signal   Left0_Right1           : std_logic;

    signal more                     : std_logic; -- for 4-bit comparator
    signal less                     : std_logic; -- for 4-bit comparator
    signal equal                    : std_logic; -- for 4-bit comparator

    signal current_value            : std_logic_vector(3 downto 0);
    signal target_value             : std_logic_vector(3 downto 0);

    signal seg7_A                   : std_logic_vector(6 downto 0); -- left display
    signal seg7_B                   : std_logic_vector(6 downto 0); -- right display
```

```vhdl
-------------------------------------------------------------------------------------------------------------
BEGIN

-- CLOCKING GENERATOR WHICH DIVIDES THE INPUT CLOCK DOWN TO A LOWER FREQUENCY

BinCLK: PROCESS(clkin_50, rst_n) is
   BEGIN
      IF (rising_edge(clkin_50)) THEN -- binary counter increments on rising clock edge
         bin_counter <= bin_counter + 1;
      END IF;
   END PROCESS;

clock_Source:
           Main_Clk <=
           clkin_50 when sim = TRUE else               -- for simulations only
           std_logic(bin_counter(23));                 -- for real FPGA operation

-------------------------------------------------------------------------------------------------
Left0_Right1 <= pb(0);              -- switch direction of led(7..4)
target_value <= sw(3 downto 0);     -- target value based on switches

leds(7 downto 4) <= Simple_States;  -- incrementing/decrementing counter
leds(3) <= Main_Clk;                -- flashing LED at speed of Main_Clk
leds(2) <= more;                    -- count up led
leds(1) <= equal;                   -- count done led
leds(0) <= less;                    -- count down led




INST0: compx4 port map(sw(0), current_value(0), sw(1), current_value(1),
                       sw(2), current_value(2), sw(3), current_value(3),
                       more, equal, less); --passes in target value and current value to comparator
INST1: Moore_SM port map(Main_Clk, rst_n, more, equal, less, current_value); -- uses slowed down clock
INST2: SevenSegment port map(target_value, seg7_A);
INST3: SevenSegment port map(current_value, seg7_B);
INST4: segment7_mux port map(clkin_50, seg7_A, seg7_B, seg7_data, seg7_char2, seg7_char1); -- uses fast clock

-------------------------------------------------------
process (Main_Clk, rst_n) is
begin
   if (rst_n = '0') then
         Simple_States <= "1000";
   elsif (rising_edge(Main_Clk)) then
         if (Left0_right1 = '1') then -- TRUE for RIGHT shift
            Simple_States (7 downto 4) <= Simple_States(4) & Simple_States(7 downto 5); --includes wrap around of shift registers bits
         else
            Simple_States (7 downto 4) <= Simple_States(6 downto 4) & Simple_States(7); --includes wrap around of shift registers bits
         end if;
   end if;
end process;

END SimpleCircuit;
```

MSM VHDL file

```
 1   library ieee;
 2   use ieee.std_logic_1164.all;
 3   use ieee.numeric_std.all;
 4
 5   Entity Moore_SM IS Port
 6   (
 7    clk_input, rst_n              : IN std_logic;
 8    MORE, EQUAL, LESS             : IN std_logic;
 9    current_value                 : OUT std_logic_vector(3 downto 0)
10   );
11   END ENTITY;
12
13   Architecture MSM of Moore_SM is
14     TYPE STATE_NAMES IS (S0, S1, S2, S3, S4, S5, S6, S7,
15                          S8, S9, S10, S11, S12, S13, S14, S15); -- list all the STATES
16
17
18     signal current_state,next_state   :  STATE_NAMES;   -- signals of type STATE_NAMES
19
20   BEGIN
21   --------------------------------------------------------------------------------
22   --State Machine:
23   --------------------------------------------------------------------------------
24
25   -- REGISTER LOGIC PROCESS
26   -- add clock and any related inputs for state machine register section into Sensitivity List
27
28   Register_Section: PROCESS (clk_input, rst_n,next_state)  -- this process synchronizes the activity to a clock
29   BEGIN
30      IF (rst_n = '0') THEN
31         current_state <= S0;
32     ELSIF(rising_edge(clk_input)) THEN
33         current_state <=next_state;
34              ELSE
35         current_state <= current_state;
36              END IF;
37   END PROCESS;
38
39   -- TRANSTION LOGIC PROCESS (to be combinational only)
40   -- add all transition inputs for state machine into Transition section Sensitivity List
41   -- make sure that all conditional statement options are complete otherwise VHDL will infer LATCHES.
42
43   Transition_Section: PROCESS (MORE, LESS, EQUAL, current_state)
44
45   BEGIN -- based on comparator output, assign state to next_state
46        CASE current_state IS
47         WHEN S0 =>
48           IF (MORE ='1') THEN
49             next_state <= S1;
50           ELSE
51             next_state <= S0;
52           END IF;
53
54         WHEN S1 =>
55           IF (MORE ='1') THEN
56             next_state <= S2;
57           ELSIF (LESS ='1') THEN
58             next_state <= S0;
59           ELSE
60             next_state <= S1;
61           END IF;
62
63         WHEN S2 =>
64           IF (MORE ='1') THEN
65             next_state <= S3;
66           ELSIF (LESS ='1') THEN
67             next_state <= S1;
68           ELSE
```

```
69          next_state <= S2;
70        END IF;
71
72      WHEN S3 =>
73        IF (MORE ='1') THEN
74          next_state <= S4;
75        ELSIF (LESS ='1') THEN
76          next_state <= S2;
77        ELSE
78          next_state <= S3;
79        END IF;
80
81      WHEN S4 =>
82        IF (MORE ='1') THEN
83          next_state <= S5;
84        ELSIF (LESS ='1') THEN
85          next_state <= S3;
86        ELSE
87          next_state <= S4;
88        END IF;
89
90      WHEN S5 =>
91        IF (MORE ='1') THEN
92          next_state <= S6;
93        ELSIF (LESS ='1') THEN
94          next_state <= S4;
95        ELSE
96          next_state <= S5;
97        END IF;
98
99      WHEN S6 =>
100       IF (MORE ='1') THEN
101         next_state <= S7;
102       ELSIF (LESS ='1') THEN
103         next_state <= S5;
104       ELSE
105         next_state <= S6;
106       END IF;
107
108     WHEN S7 =>
109       IF (MORE ='1') THEN
110         next_state <= S8;
111       ELSIF (LESS ='1') THEN
112         next_state <= S6;
113       ELSE
114         next_state <= S7;
115       END IF;
116
117     WHEN S8 =>
118       IF (MORE ='1') THEN
119         next_state <= S9;
120       ELSIF (LESS ='1') THEN
121         next_state <= S7;
122       ELSE
123         next_state <= S8;
124       END IF;
125
126     WHEN S9 =>
127       IF (MORE ='1') THEN
128         next_state <= S10;
129       ELSIF (LESS ='1') THEN
130         next_state <= S8;
131       ELSE
132         next_state <= S9;
133       END IF;
134
```

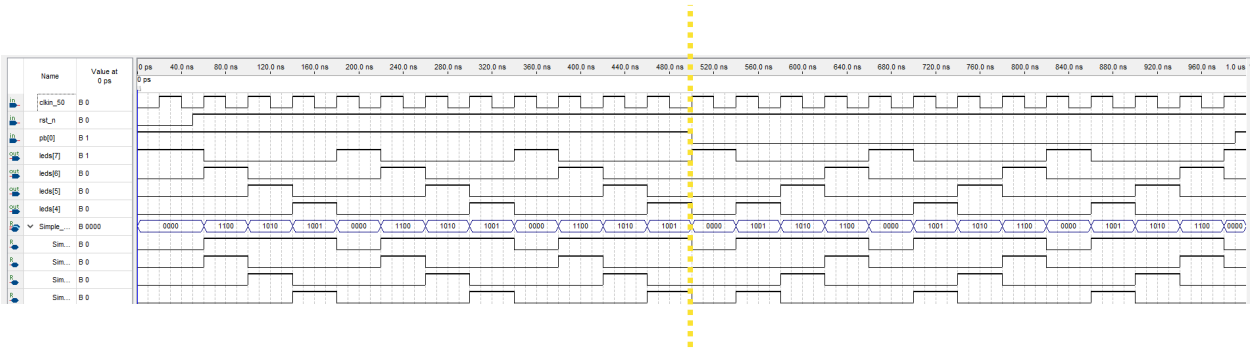
```
      WHEN S10 =>
        IF (MORE ='1') THEN
          next_state <= S11;
        ELSIF (LESS ='1') THEN
          next_state <= S9;
        ELSE
          next_state <= S10;
        END IF;

      WHEN S11 =>
        IF (MORE ='1') THEN
          next_state <= S12;
        ELSIF (LESS ='1') THEN
          next_state <= S10;
        ELSE
          next_state <= S11;
        END IF;

      WHEN S12 =>
        IF (MORE ='1') THEN
          next_state <= S13;
        ELSIF (LESS ='1') THEN
          next_state <= S11;
        ELSE
          next_state <= S12;
        END IF;

      WHEN S13 =>
        IF (MORE ='1') THEN
          next_state <= S14;
        ELSIF (LESS ='1') THEN
          next_state <= S12;
        ELSE
          next_state <= S13;
        END IF;

      WHEN S14 =>
        IF (MORE ='1') THEN
          next_state <= S15;
        ELSIF (LESS ='1') THEN
          next_state <= S13;
        ELSE
          next_state <= S14;
        END IF;

      WHEN S15 =>
        IF (LESS ='1') THEN
          next_state <= S14;
        ELSE
          next_state <= S15;
        END IF;

      WHEN others =>
          next_state <= S0;
      END CASE;

END PROCESS;

Decoder_Section: PROCESS(current_state)

BEGIN -- based on current state, assign 4-bit value to current-value
  CASE current_state IS
    WHEN S0 =>
      current_value <= "0000";
    WHEN S1 =>
      current_value <= "0001";
```

```
200         current_value <= "0001";
201       WHEN S2 =>
202         current_value <= "0010";
203       WHEN S3 =>
204         current_value <= "0011";
205       WHEN S4 =>
206         current_value <= "0100";
207       WHEN S5 =>
208         current_value <= "0101";
209       WHEN S6 =>
210         current_value <= "0110";
211       WHEN S7 =>
212         current_value <= "0111";
213       WHEN S8 =>
214         current_value <= "1000";
215       WHEN S9 =>
216         current_value <= "1001";
217       WHEN S10 =>
218         current_value <= "1010";
219       WHEN S11 =>
220         current_value <= "1011";
221       WHEN S12 =>
222         current_value <= "1100";
223       WHEN S13 =>
224         current_value <= "1101";
225       WHEN S14 =>
226         current_value <= "1110";
227       WHEN S15 =>
228         current_value <= "1111";
229     END CASE;
230   END PROCESS;
231
232   END ARCHITECTURE MSM;
```

Resource Utilization

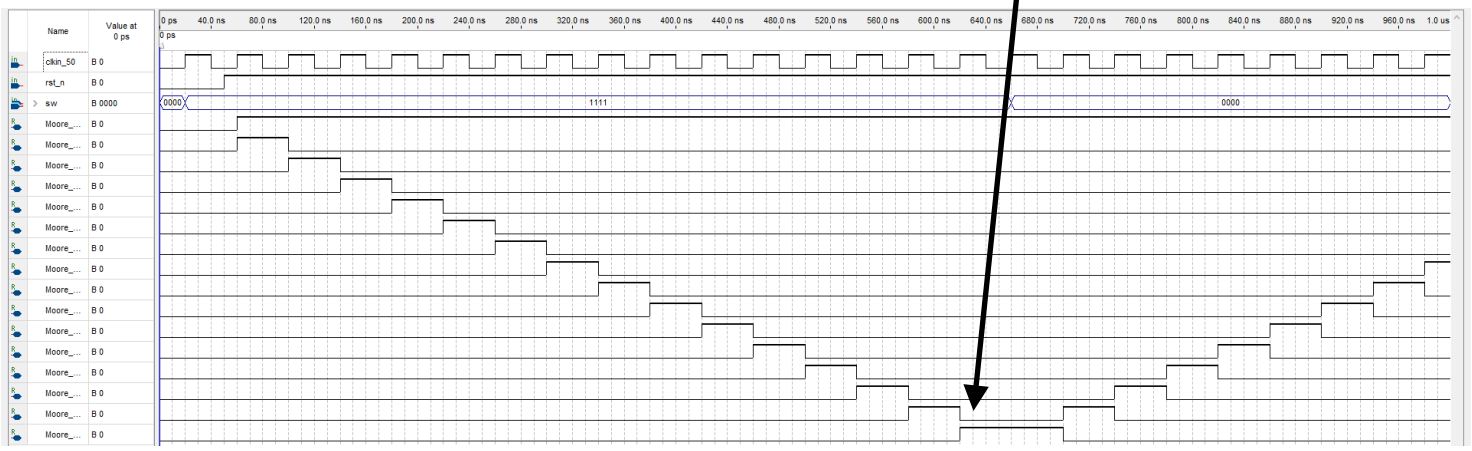
Flow Status	Successful - Thu Jun 29 23:53:04 2017
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Standard Edition
Revision Name	LogicalStep_Lab4_top
Top-level Entity Name	LogicalStep_Lab4_top
Family	MAX 10
Device	10M08SAE144C8G
Timing Models	Final
Total logic elements	103 / 8,064 (1 %)
Total combinational functions	103 / 8,064 (1 %)
Dedicated logic registers	44 / 8,064 (< 1 %)
Total registers	44
Total pins	31 / 101 (31 %)
Total virtual pins	0
Total memory bits	0 / 387,072 (0 %)
Embedded Multiplier 9-bit elements	0 / 48 (0 %)
Total PLLs	0 / 1 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 1 (0 %)

Simulation of 4 stage Shift Register operating in both directions



When pb[0] is pressed, the direction of the shift register changes, as we can see from the change in direction when pb[0] goes from 1 to 0.

Simulation of Moore State Machine counting over entire range to Max target value



At every rising edge of the clock, the state machine will count either down or up to the max target value, based on the input from the sw.

State Diagram