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;
leds(1) <= equal;
leds(0) <= less;



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);
INST1: Moore_SM port map(Main_Clk, rst_n, more, equal, less, current_value);
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);

-------------------------------------------------------
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

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

Entity Moore_SM IS Port
(
 clk_input, rst_n              : IN std_logic;
 MORE, EQUAL, LESS             : IN std_logic;
 current_value                 : OUT std_logic_vector(3 downto 0)
 );
END ENTITY;

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

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

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;

-- 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 (MORE, LESS, EQUAL, current_state)

BEGIN
     CASE current_state IS
      WHEN S0 =>
        IF (MORE ='1') THEN
          next_state <= S1;
        ELSE
          next_state <= S0;
        END IF;

      WHEN S1 =>
        IF (MORE ='1') THEN
          next_state <= S2;
        ELSIF (LESS ='1') THEN
          next_state <= S0;
        ELSE
          next_state <= S1;
        END IF;

      WHEN S2 =>
        IF (MORE ='1') THEN
          next_state <= S3;
        ELSIF (LESS ='1') THEN
          next_state <= S1;
        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
```

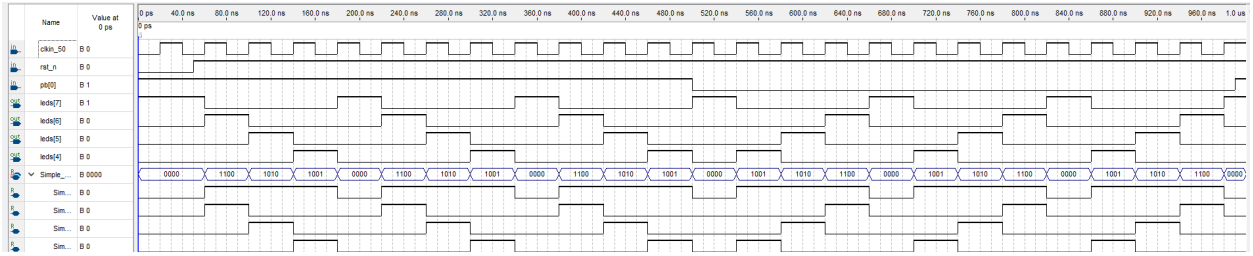
```
135         WHEN S10 =>
136           IF (MORE ='1') THEN
137             next_state <= S11;
138           ELSIF (LESS ='1') THEN
139             next_state <= S9;
140           ELSE
141             next_state <= S10;
142           END IF;
143
144         WHEN S11 =>
145           IF (MORE ='1') THEN
146             next_state <= S12;
147           ELSIF (LESS ='1') THEN
148             next_state <= S10;
149           ELSE
150             next_state <= S11;
151           END IF;
152
153         WHEN S12 =>
154           IF (MORE ='1') THEN
155             next_state <= S13;
156           ELSIF (LESS ='1') THEN
157             next_state <= S11;
158           ELSE
159             next_state <= S12;
160           END IF;
161
162         WHEN S13 =>
163           IF (MORE ='1') THEN
164             next_state <= S14;
165           ELSIF (LESS ='1') THEN
166             next_state <= S12;
167           ELSE
168             next_state <= S13;
169           END IF;
170
171         WHEN S14 =>
172           IF (MORE ='1') THEN
173             next_state <= S15;
174           ELSIF (LESS ='1') THEN
175             next_state <= S13;
176           ELSE
177             next_state <= S14;
178           END IF;
179
180         WHEN S15 =>
181           IF (LESS ='1') THEN
182             next_state <= S14;
183           ELSE
184             next_state <= S15;
185           END IF;
186
187         WHEN others =>
188             next_state <= S0;
189       END CASE;
190
191   END PROCESS;
192
193   Decoder_Section: PROCESS(current_state)
194
195   BEGIN
196     CASE current_state IS
197       WHEN S0 =>
198         current_value <= "0000";
199       WHEN S1 =>
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;
```

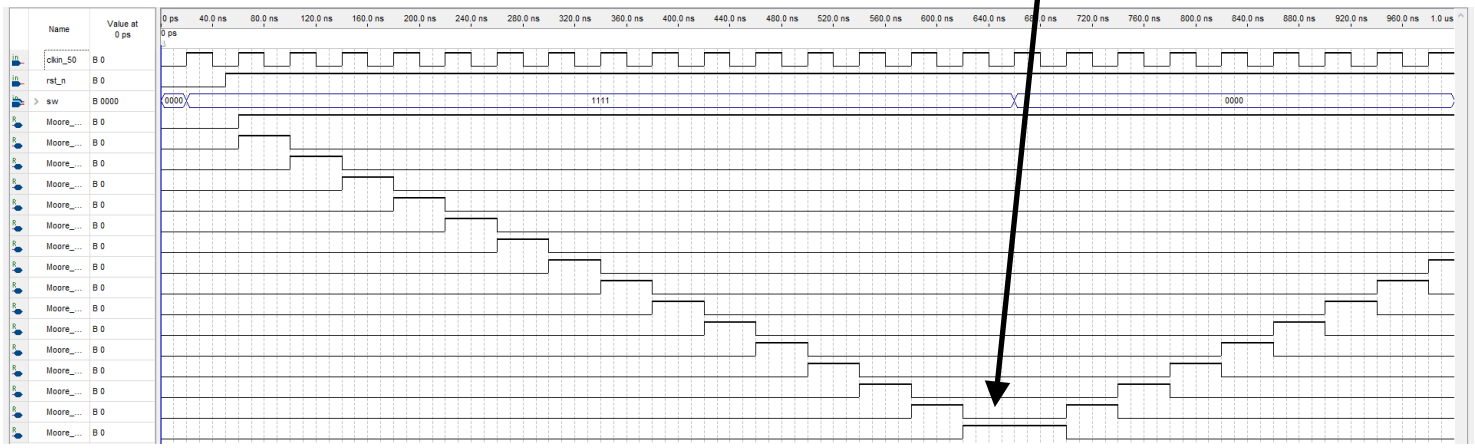
Resource Utilization

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



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



State Diagram