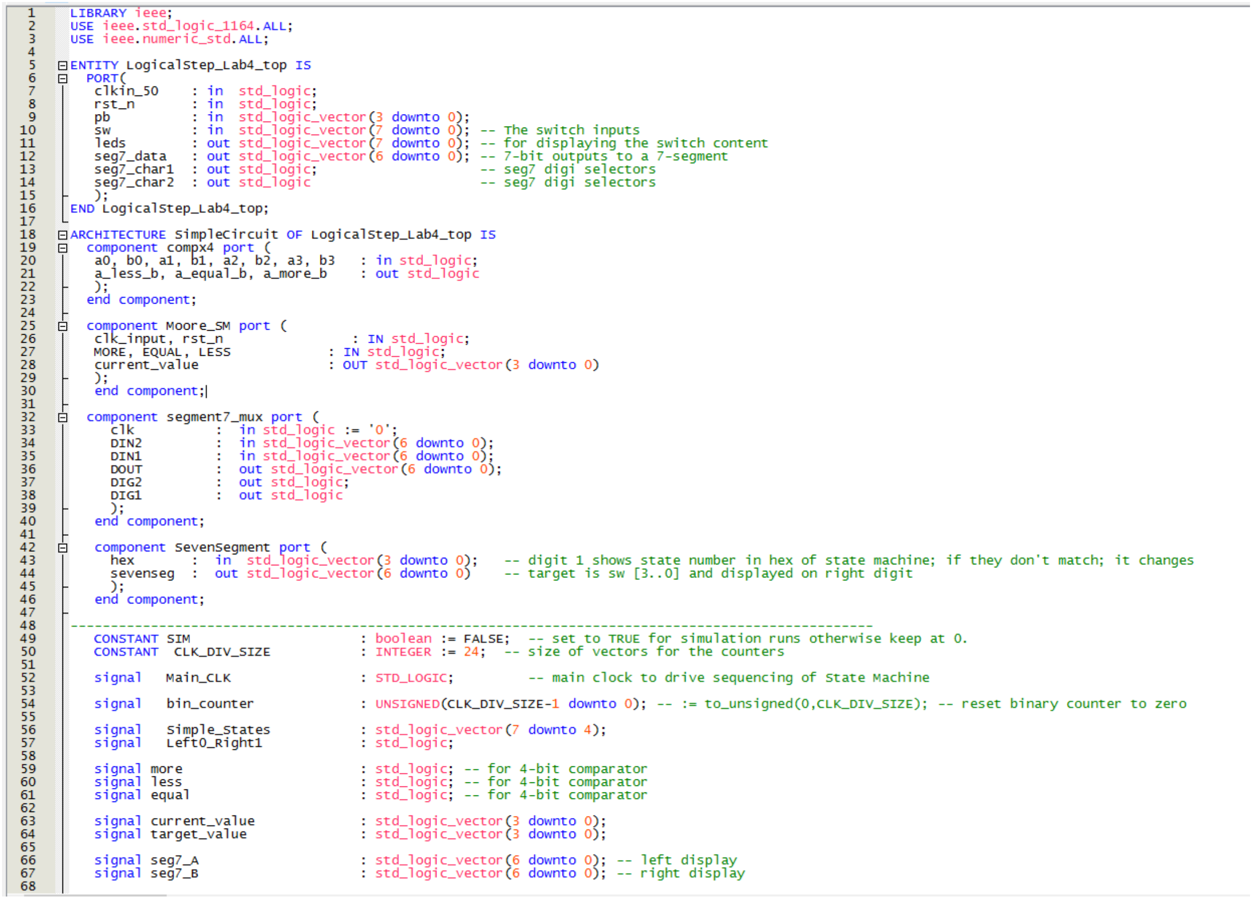
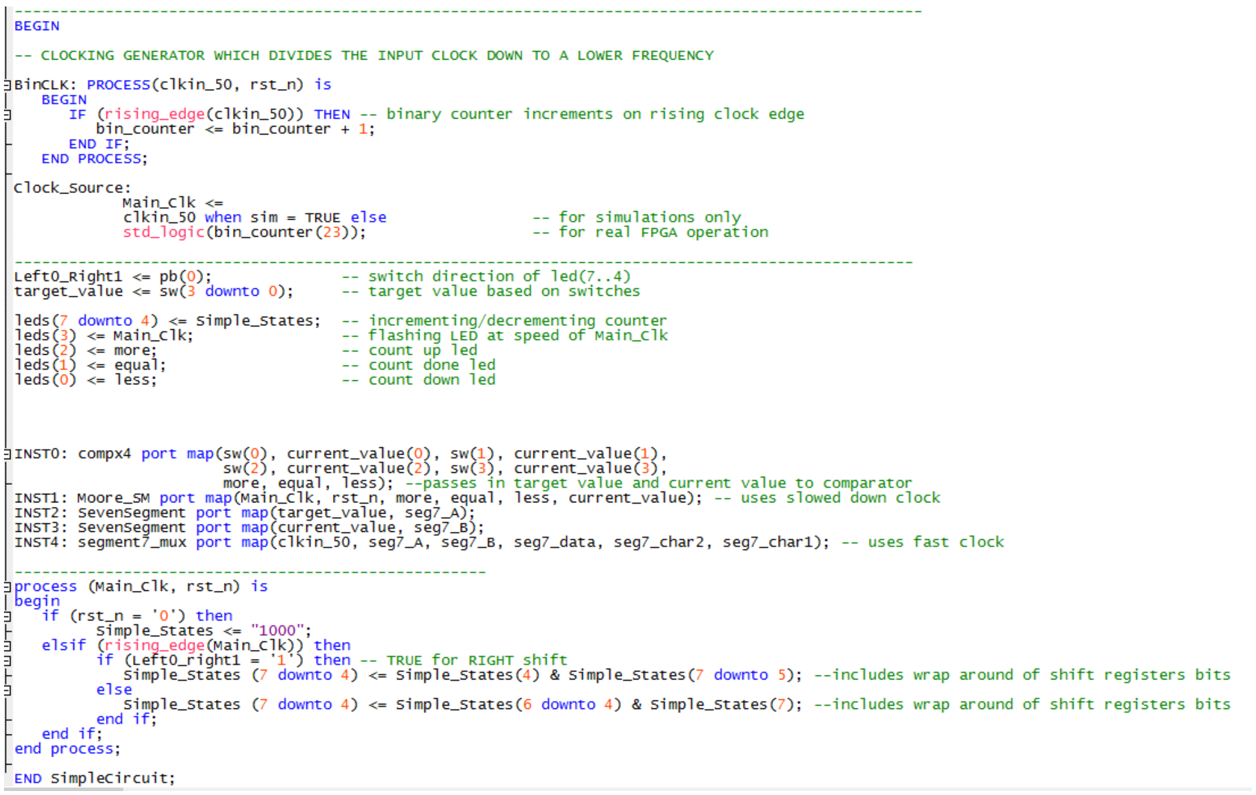
Structural VHDL Design Top File

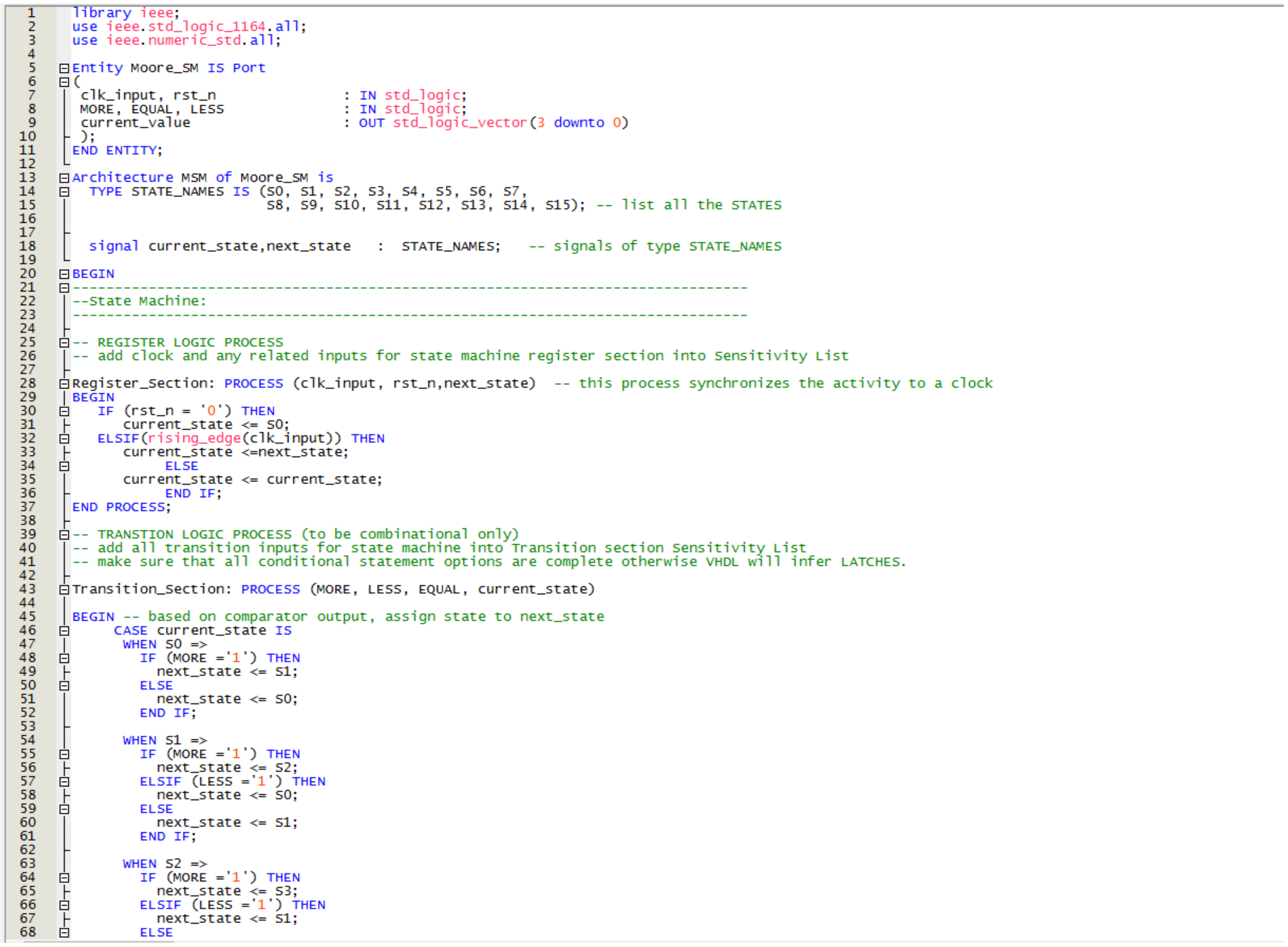


```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

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
```

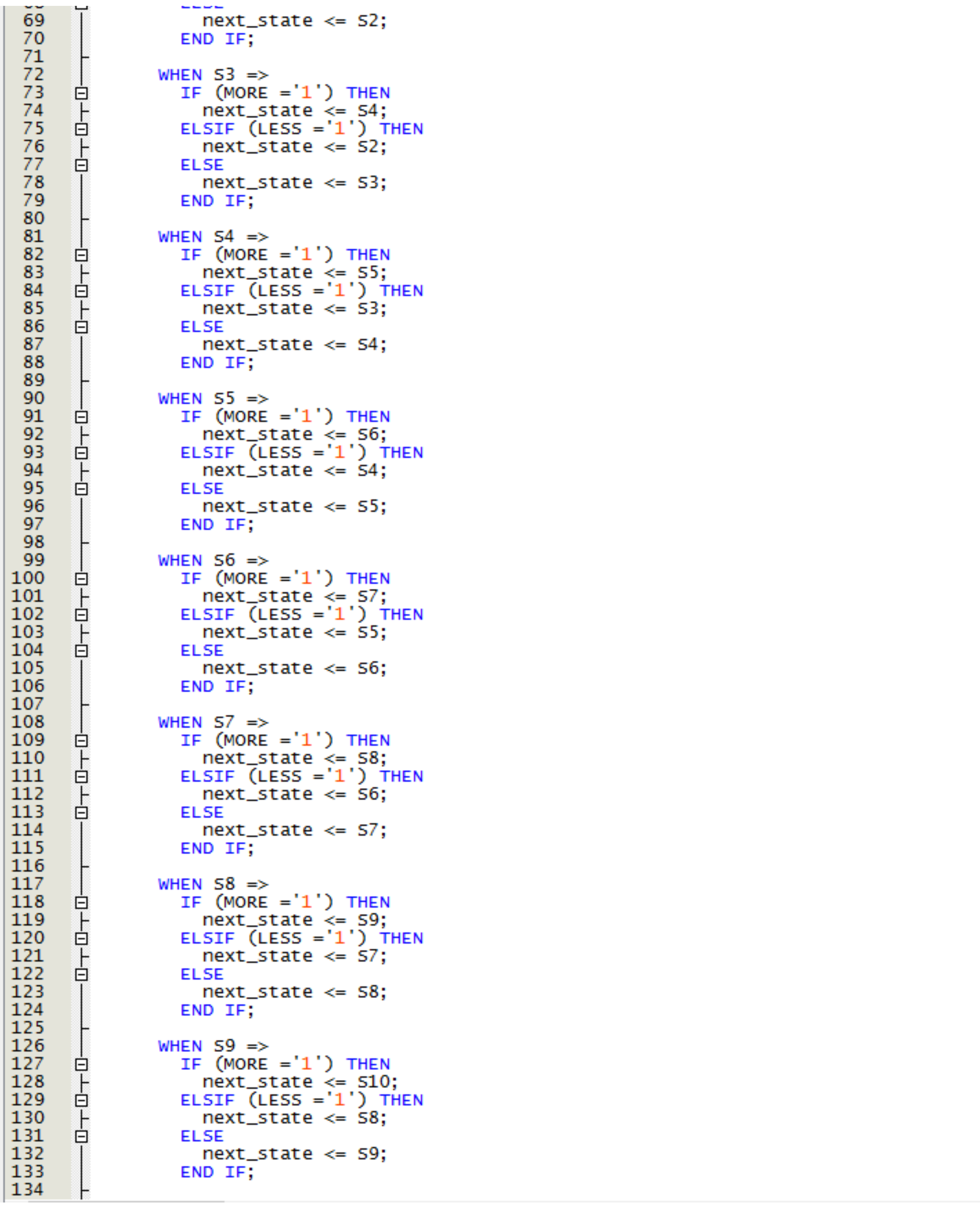


```vhdl
----------------------------------------------------------------------------------------------------
BEGIN

-- 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;
```

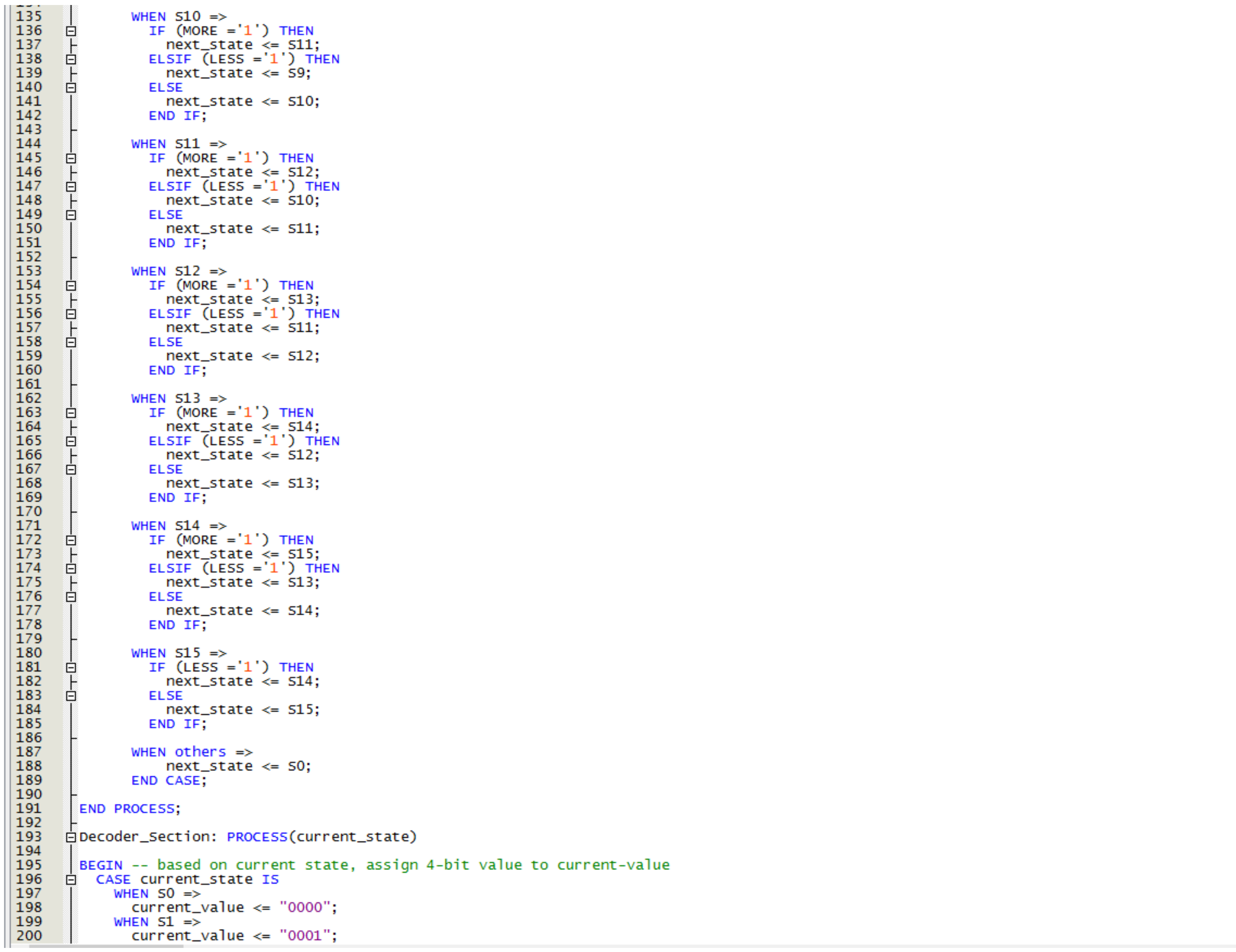
MSM VHDL file



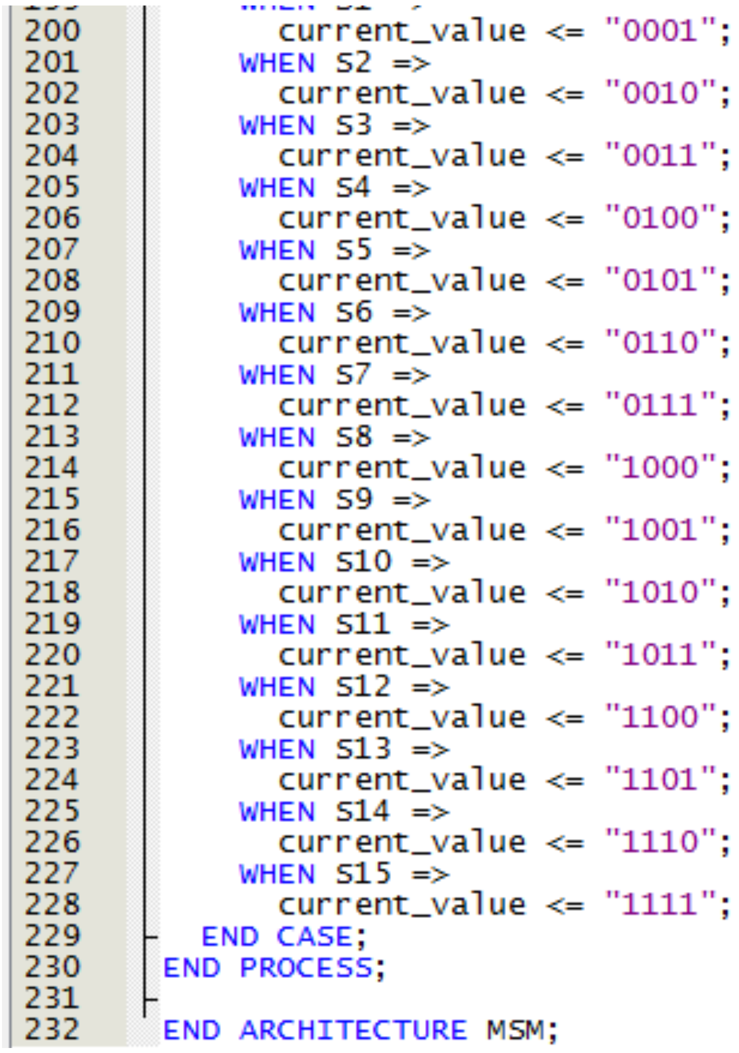
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

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 -- based on comparator output, assign state to next_state
     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
```



```
          next_state <= S2;
        END IF;

      WHEN S3 =>
        IF (MORE ='1') THEN
          next_state <= S4;
        ELSIF (LESS ='1') THEN
          next_state <= S2;
        ELSE
          next_state <= S3;
        END IF;

      WHEN S4 =>
        IF (MORE ='1') THEN
          next_state <= S5;
        ELSIF (LESS ='1') THEN
          next_state <= S3;
        ELSE
          next_state <= S4;
        END IF;

      WHEN S5 =>
        IF (MORE ='1') THEN
          next_state <= S6;
        ELSIF (LESS ='1') THEN
          next_state <= S4;
        ELSE
          next_state <= S5;
        END IF;

      WHEN S6 =>
        IF (MORE ='1') THEN
          next_state <= S7;
        ELSIF (LESS ='1') THEN
          next_state <= S5;
        ELSE
          next_state <= S6;
        END IF;

      WHEN S7 =>
        IF (MORE ='1') THEN
          next_state <= S8;
        ELSIF (LESS ='1') THEN
          next_state <= S6;
        ELSE
          next_state <= S7;
        END IF;

      WHEN S8 =>
        IF (MORE ='1') THEN
          next_state <= S9;
        ELSIF (LESS ='1') THEN
          next_state <= S7;
        ELSE
          next_state <= S8;
        END IF;

      WHEN S9 =>
        IF (MORE ='1') THEN
          next_state <= S10;
        ELSIF (LESS ='1') THEN
          next_state <= S8;
        ELSE
          next_state <= S9;
        END IF;

```

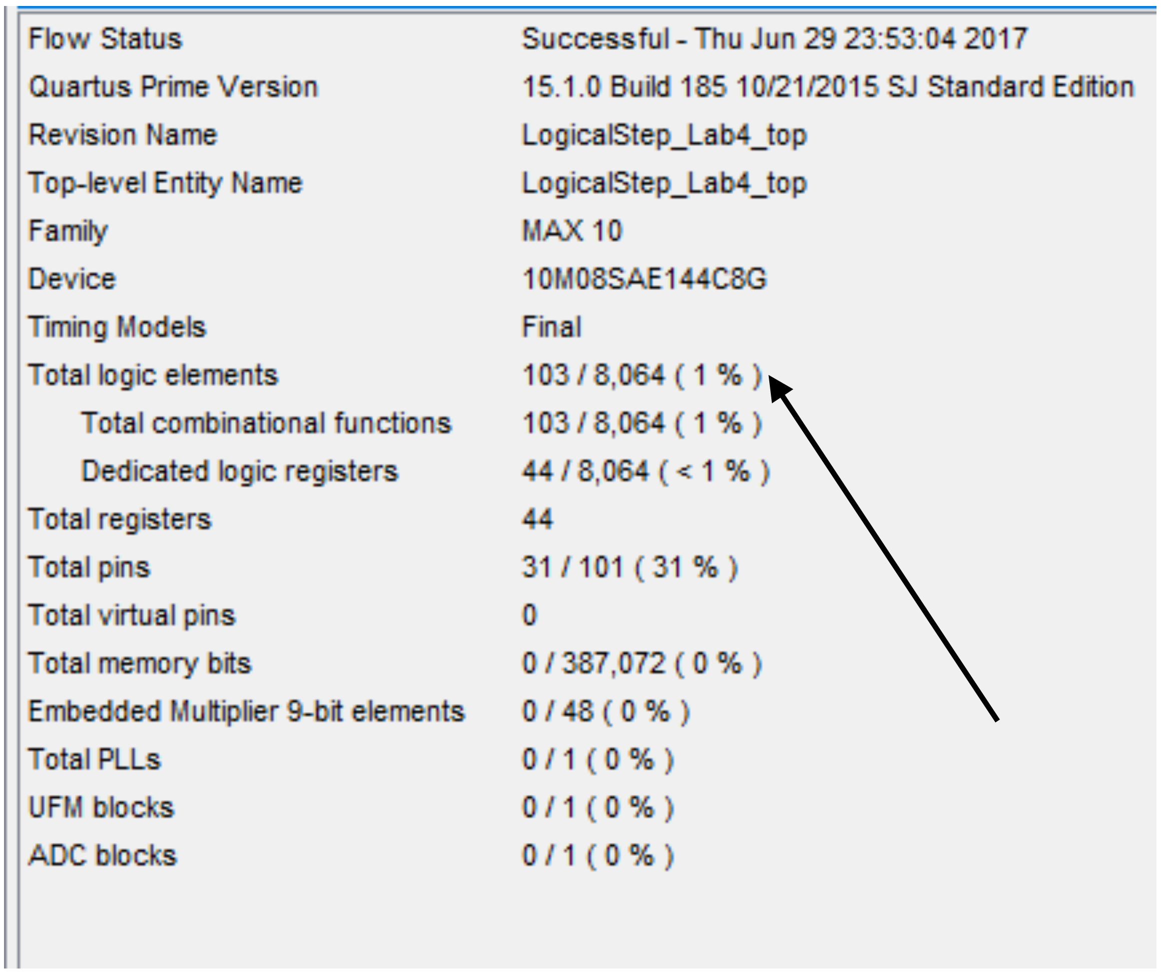


```vhdl
      WHEN S10 =>
        IF (MORE ='1') THEN
          next_state <= S11;
        ELSIF (LESS ='1') THEN
          next_state <= S9;
        ELSE
          next_state <= S10;
        END IF;

      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";
```



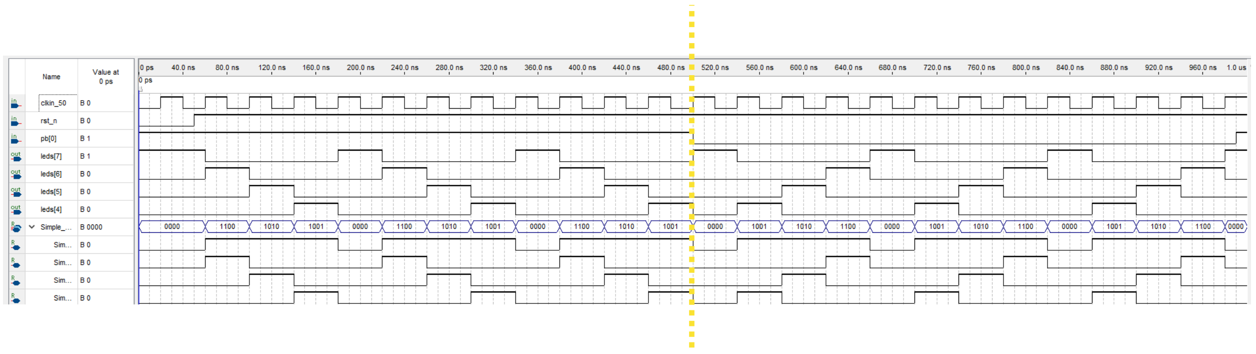
```
      current_value <= "0001";
    WHEN S2 =>
      current_value <= "0010";
    WHEN S3 =>
      current_value <= "0011";
    WHEN S4 =>
      current_value <= "0100";
    WHEN S5 =>
      current_value <= "0101";
    WHEN S6 =>
      current_value <= "0110";
    WHEN S7 =>
      current_value <= "0111";
    WHEN S8 =>
      current_value <= "1000";
    WHEN S9 =>
      current_value <= "1001";
    WHEN S10 =>
      current_value <= "1010";
    WHEN S11 =>
      current_value <= "1011";
    WHEN S12 =>
      current_value <= "1100";
    WHEN S13 =>
      current_value <= "1101";
    WHEN S14 =>
      current_value <= "1110";
    WHEN S15 =>
      current_value <= "1111";
  END CASE;
END PROCESS;

END ARCHITECTURE MSM;
```

Resource Utilization



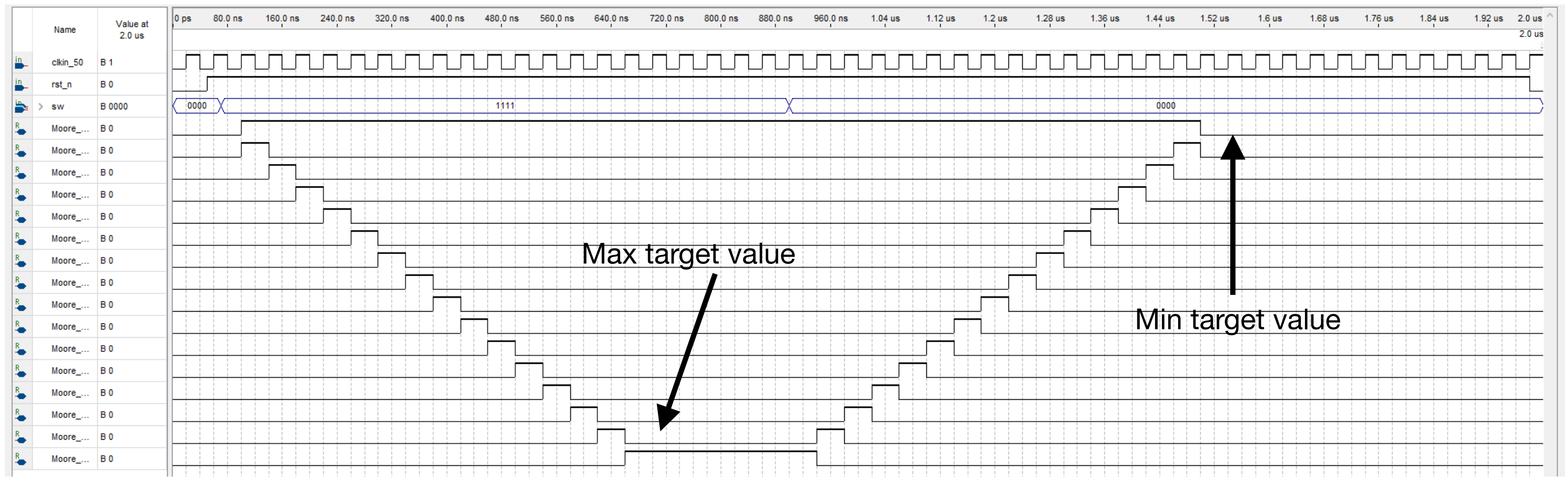
Flow Status	Successful - Thu Jun 29 23:53:04 2017
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Standard Edition
Revision Name	LogicalStep_Lab4_top
Top-level Entity Name	LogicalStep_Lab4_top
Family	MAX 10
Device	10M08SAE144C8G
Timing Models	Final
Total logic elements	103 / 8,064 (1 %)
Total combinational 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



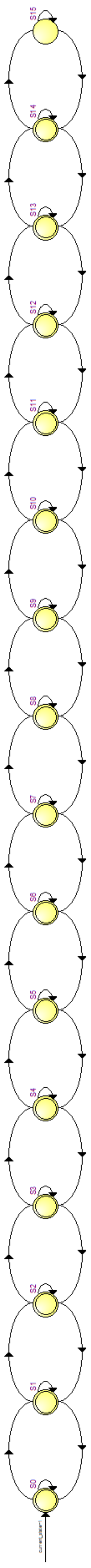
When pb[0] is pressed, the direction of the shift register changes, as we can see from the change in direction when pb[0] goes from 1 to 0.

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



At every rising edge of the clock, the state machine will count up to the max target value (1111), which is the input from the sw. Then, the sw input changes to the min target value (0000) and it counts all the way down again.

State Diagram


S15
S14
S13
S12
S11
S10
S9
S8
S7
S6
S5
S4
S3
S2
S1
S0
current_state~1