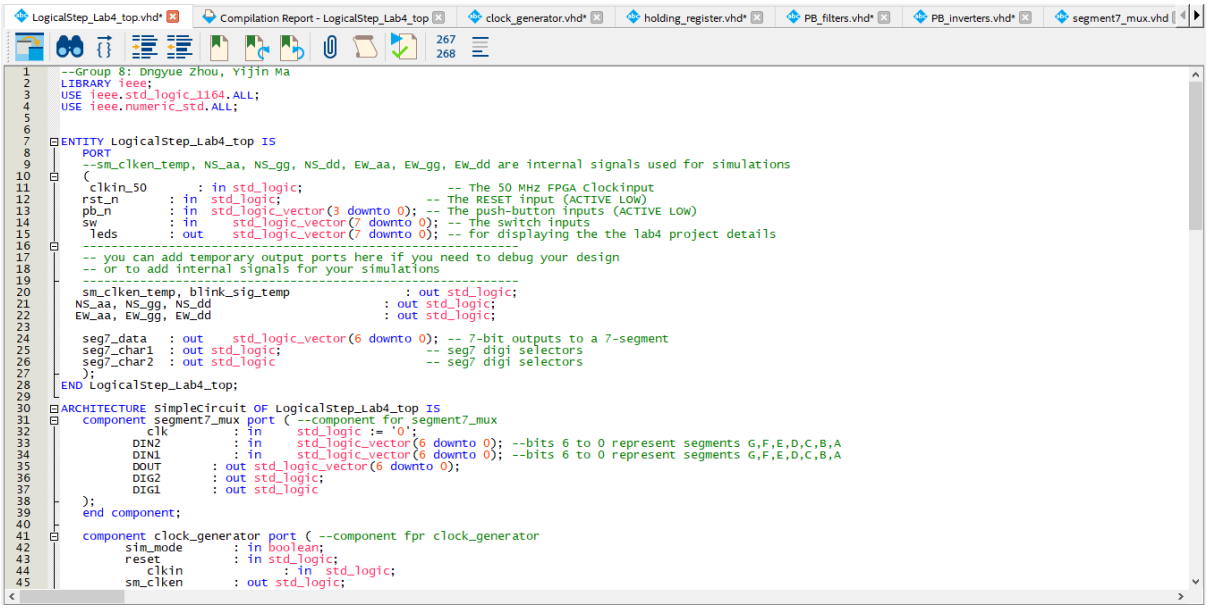
Lab 3 Report

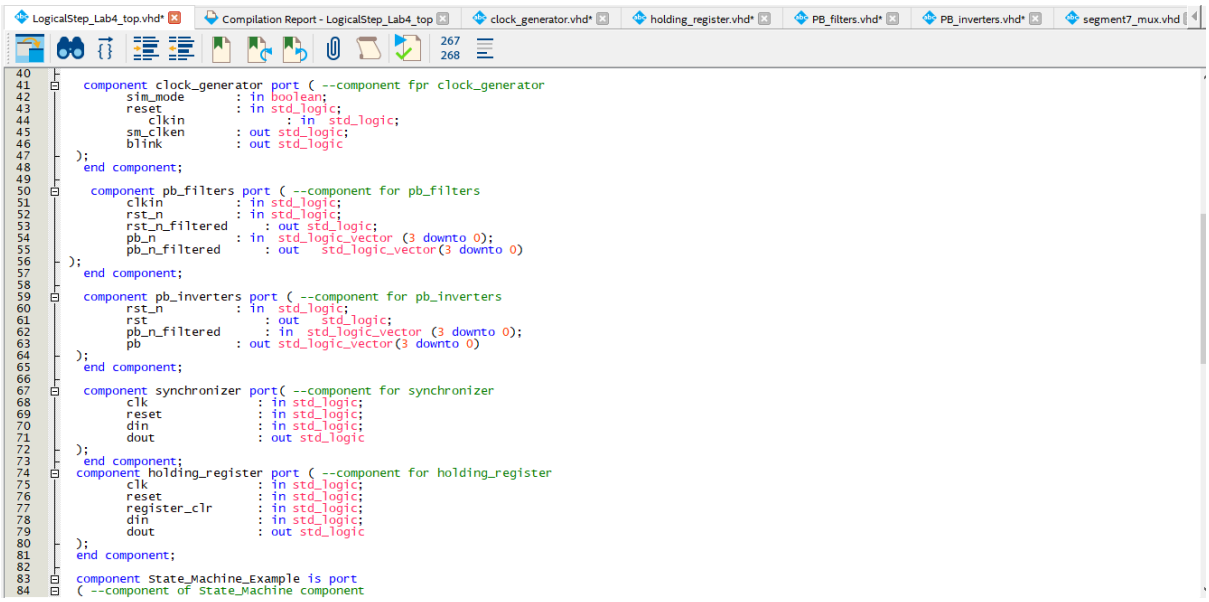
ECE 124
Group 8
Session 201

DongyueZhou 21073014
YijinMa 21070763

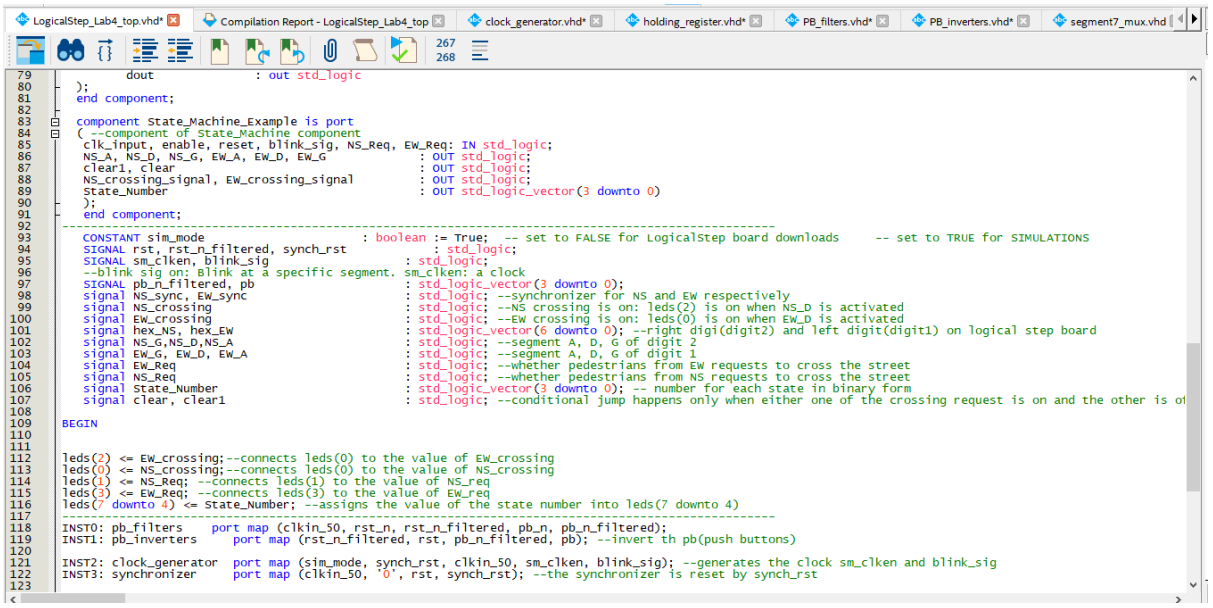
LogicalStep_Lab4_Top:



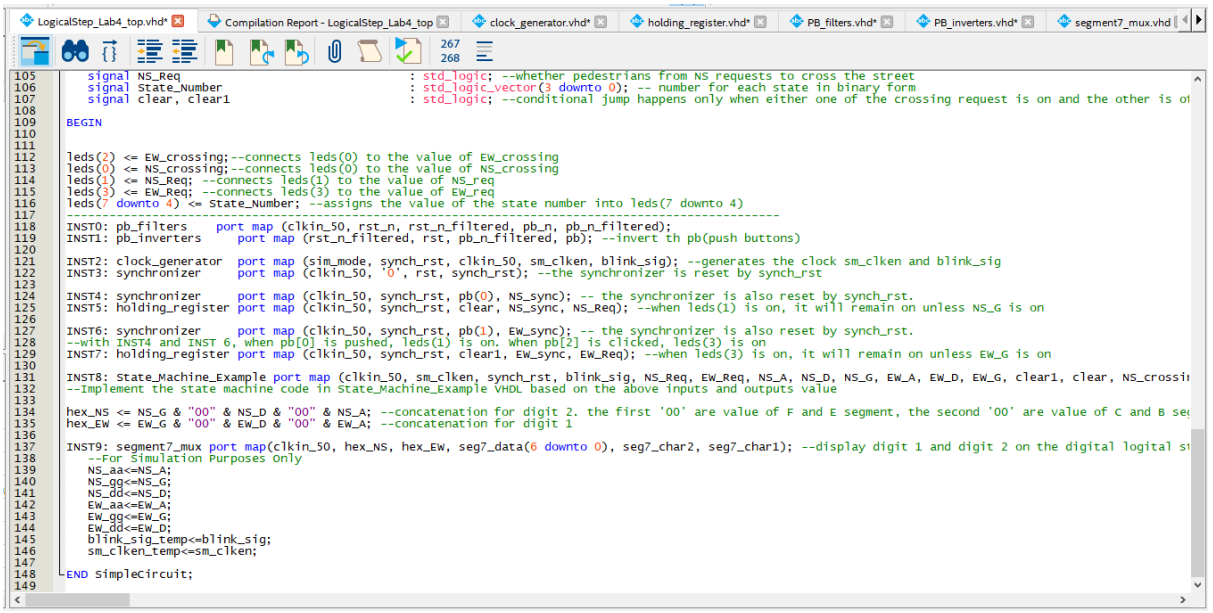
```
--Group 8: Dngyue Zhou, Yijin Ma
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;


ENTITY LogicalStep_Lab4_top IS
   PORT
   --sm_clken_temp, NS_aa, NS_gg, NS_dd, EW_aa, EW_gg, EW_dd are internal signals used for simulations
   (
    clkin_50       : in std_logic;                    -- The 50 MHz FPGA Clockinput
   rst_n       : in  std_logic;                    -- The RESET input (ACTIVE LOW)
   pb_n        : 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 the lab4 project details
   -----------------------------------------------------------
   -- you can add temporary output ports here if you need to debug your design
   -- or to add internal signals for your simulations
   -----------------------------------------------------------
   sm_clken_temp, blink_sig_temp              : out std_logic;
  NS_aa, NS_gg, NS_dd                       : out std_logic;
  EW_aa, EW_gg, EW_dd                       : out std_logic;

   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 segment7_mux port ( --component for segment7_mux
            clk          : in     std_logic := '0';
          DIN2           : in     std_logic_vector(6 downto 0); --bits 6 to 0 represent segments G,F,E,D,C,B,A
          DIN1           : in     std_logic_vector(6 downto 0); --bits 6 to 0 represent segments G,F,E,D,C,B,A
          DOUT       : out std_logic_vector(6 downto 0);
          DIG2       : out std_logic;
          DIG1       : out std_logic
   );
   end component;

   component clock_generator port ( --component fpr clock_generator
         sim_mode       : in boolean;
         reset          : in std_logic;
            clkin            : in  std_logic;
         sm_clken       : out std_logic;
```



```
   component clock_generator port ( --component fpr clock_generator
         sim_mode       : in boolean;
         reset          : in std_logic;
            clkin            : in  std_logic;
         sm_clken       : out std_logic;
         blink          : out std_logic
   );
   end component;

    component pb_filters port ( --component for pb_filters
         clkin          : in std_logic;
         rst_n          : in std_logic;
         rst_n_filtered     : out std_logic;
         pb_n           : in  std_logic_vector (3 downto 0);
         pb_n_filtered      : out   std_logic_vector(3 downto 0)
 );
   end component;

   component pb_inverters port ( --component for pb_inverters
         rst_n          : in  std_logic;
         rst                : out   std_logic;
         pb_n_filtered      : in  std_logic_vector (3 downto 0);
         pb             : out std_logic_vector(3 downto 0)
   );
    end component;

    component synchronizer port( --component for synchronizer
         clk                : in std_logic;
         reset              : in std_logic;
         din                : in std_logic;
         dout               : out std_logic
   );
    end component;
   component holding_register port ( --component for holding_register
         clk                : in std_logic;
         reset              : in std_logic;
         register_clr       : in std_logic;
         din                : in std_logic;
         dout               : out std_logic
   );
   end component;

   component State_Machine_Example is port
   ( --component of State_Machine component
```

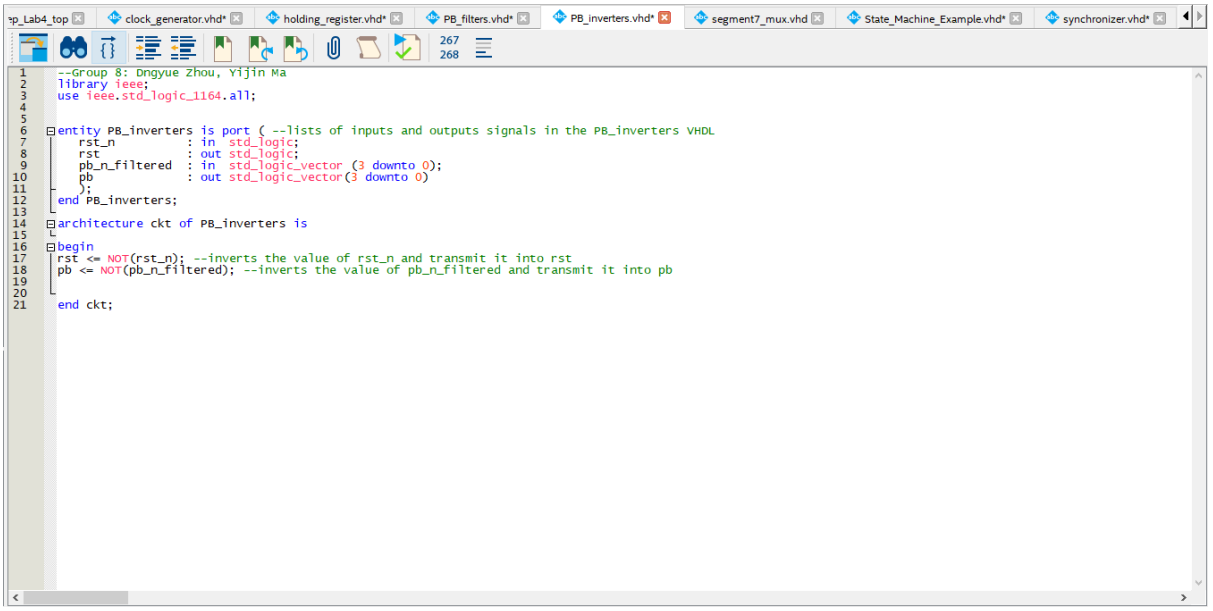


```
79          dout              : out std_logic
80      );
81      end component;
82
83      component State_Machine_Example is port
84      ( --component of State_Machine component
85       clk_input, enable, reset, blink_sig, NS_Req, EW_Req: IN std_logic;
86       NS_A, NS_D, NS_G, EW_A, EW_D, EW_G              : OUT std_logic;
87       clear1, clear                                  : OUT std_logic;
88       NS_crossing_signal, EW_crossing_signal         : OUT std_logic;
89       State_Number                                   : OUT std_logic_vector(3 downto 0)
90       );
91       end component;
92      ------------------------------------------------------------------------------------------
93       CONSTANT sim_mode                       : boolean := True;  -- set to FALSE for LogicalStep board downloads     -- set to TRUE for SIMULATIONS
94       SIGNAL rst, rst_n_filtered, synch_rst           : std_logic;
95       SIGNAL sm_clken, blink_sig                : std_logic;
96       --blink sig on: Blink at a specific segment. sm_clken: a clock
97       SIGNAL pb_n_filtered, pb                        : std_logic_vector(3 downto 0);
98       signal NS_sync, EW_sync                         : std_logic; --synchronizer for NS and EW respectively
99       signal NS_crossing                              : std_logic; --NS crossing is on: leds(2) is on when NS_D is activated
100      signal EW_crossing                              : std_logic; --EW crossing is on: leds(0) is on when EW_D is activated
101      signal hex_NS, hex_EW                           : std_logic_vector(6 downto 0); --right digi(digit2) and left digit(digit1) on logical step board
102      signal NS_G,NS_D,NS_A                           : std_logic; --segment A, D, G of digit 2
103      signal EW_G, EW_D, EW_A                         : std_logic; --segment A, D, G of digit 1
104      signal EW_Req                                   : std_logic; --whether pedestrians from EW requests to cross the street
105      signal NS_Req                                   : std_logic; --whether pedestrians from NS requests to cross the street
106      signal State_Number                             : std_logic_vector(3 downto 0); -- number for each state in binary form
107      signal clear, clear1                            : std_logic; --conditional jump happens only when either one of the crossing request is on and the other is of
108
109     BEGIN
110
111
112     leds(2) <= EW_crossing;--connects leds(0) to the value of EW_crossing
113     leds(0) <= NS_crossing;--connects leds(0) to the value of NS_crossing
114     leds(1) <= NS_Req; --connects leds(1) to the value of NS_req
115     leds(3) <= EW_Req; --connects leds(3) to the value of EW_req
116     leds(7 downto 4) <= State_Number; --assigns the value of the state number into leds(7 downto 4)
117     ------------------------------------------------------------------------------------------
118     INST0: pb_filters    port map (clkin_50, rst_n, rst_n_filtered, pb_n, pb_n_filtered);
119     INST1: pb_inverters      port map (rst_n_filtered, rst, pb_n_filtered, pb); --invert th pb(push buttons)
120
121     INST2: clock_generator   port map (sim_mode, synch_rst, clkin_50, sm_clken, blink_sig); --generates the clock sm_clken and blink_sig
122     INST3: synchronizer      port map (clkin_50, '0', rst, synch_rst); --the synchronizer is reset by synch_rst
123
```



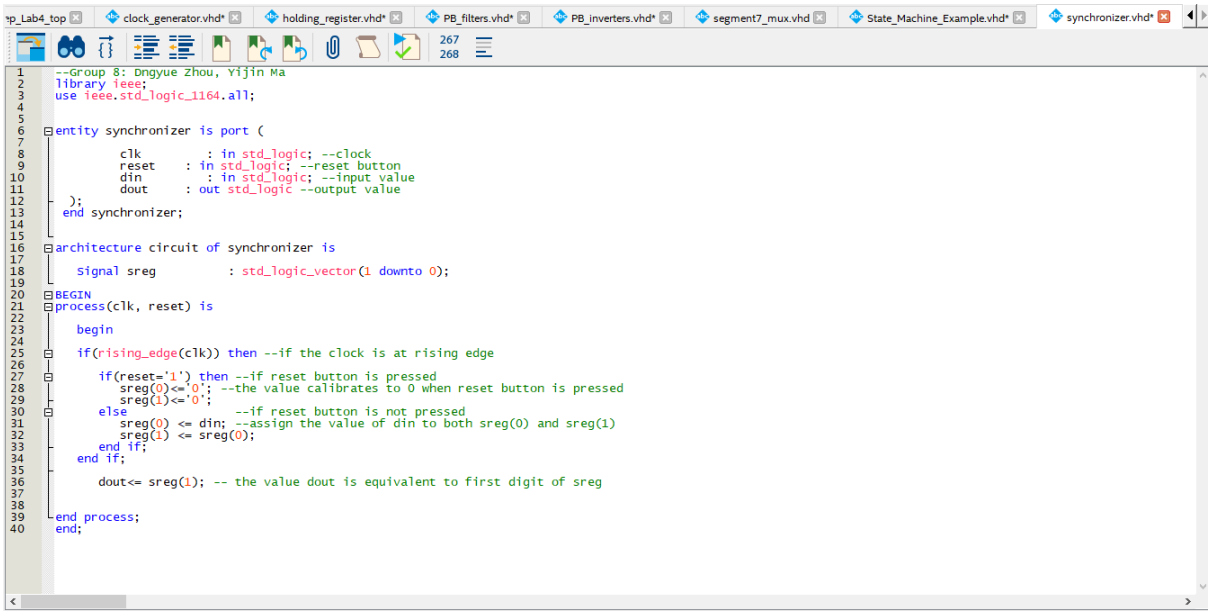
```
105      signal NS_Req                           : std_logic; --whether pedestrians from NS requests to cross the street
106      signal State_Number                     : std_logic_vector(3 downto 0); -- number for each state in binary form
107      signal clear, clear1                    : std_logic; --conditional jump happens only when either one of the crossing request is on and the other is of
108
109     BEGIN
110
111
112     leds(2) <= EW_crossing;--connects leds(0) to the value of EW_crossing
113     leds(0) <= NS_crossing;--connects leds(0) to the value of NS_crossing
114     leds(1) <= NS_Req; --connects leds(1) to the value of NS_req
115     leds(3) <= EW_Req; --connects leds(3) to the value of EW_req
116     leds(7 downto 4) <= State_Number; --assigns the value of the state number into leds(7 downto 4)
117     ------------------------------------------------------------------------------------------
118     INST0: pb_filters    port map (clkin_50, rst_n, rst_n_filtered, pb_n, pb_n_filtered);
119     INST1: pb_inverters      port map (rst_n_filtered, rst, pb_n_filtered, pb); --invert th pb(push buttons)
120
121     INST2: clock_generator   port map (sim_mode, synch_rst, clkin_50, sm_clken, blink_sig); --generates the clock sm_clken and blink_sig
122     INST3: synchronizer      port map (clkin_50, '0', rst, synch_rst); --the synchronizer is reset by synch_rst
123
124     INST4: synchronizer      port map (clkin_50, synch_rst, pb(0), NS_sync); -- the synchronizer is also reset by synch_rst.
125     INST5: holding_register port map (clkin_50, synch_rst, clear, NS_sync, NS_Req); --when leds(1) is on, it will remain on unless NS_G is on
126
127     INST6: synchronizer      port map (clkin_50, synch_rst, pb(1), EW_sync); -- the synchronizer is also reset by synch_rst.
128     --with INST4 and INST 6, when pb[0] is pushed, leds(1) is on. When pb[2] is clicked, leds(3) is on
129     INST7: holding_register port map (clkin_50, synch_rst, clear1, EW_sync, EW_Req); --when leds(3) is on, it will remain on unless EW_G is on
130
131     INST8: State_Machine_Example port map (clkin_50, sm_clken, synch_rst, blink_sig, NS_Req, EW_Req, NS_A, NS_D, NS_G, EW_A, EW_D, EW_G, clear1, clear, NS_crossi
132     --Implement the state machine code in State_Machine_Example VHDL based on the above inputs and outputs value
133
134     hex_NS <= NS_G & "00" & NS_D & "00" & NS_A; --concatenation for digit 2. the first '00' are value of F and E segment, the second '00' are value of C and B se
135     hex_EW <= EW_G & "00" & EW_D & "00" & EW_A; --concatenation for digit 1
136
137     INST9: segment7_mux port map(clkin_50, hex_NS, hex_EW, seg7_data(6 downto 0), seg7_char2, seg7_char1); --display digit 1 and digit 2 on the digital logital st
138        --For Simulation Purposes Only
139        NS_aa<=NS_A;
140        NS_gg<=NS_G;
141        NS_dd<=NS_D;
142        EW_aa<=EW_A;
143        EW_gg<=EW_G;
144        EW_dd<=EW_D;
145        blink_sig_temp<=blink_sig;
146        sm_clken_temp<=sm_clken;
147
148     END SimpleCircuit;
149
```

PB_Inverters



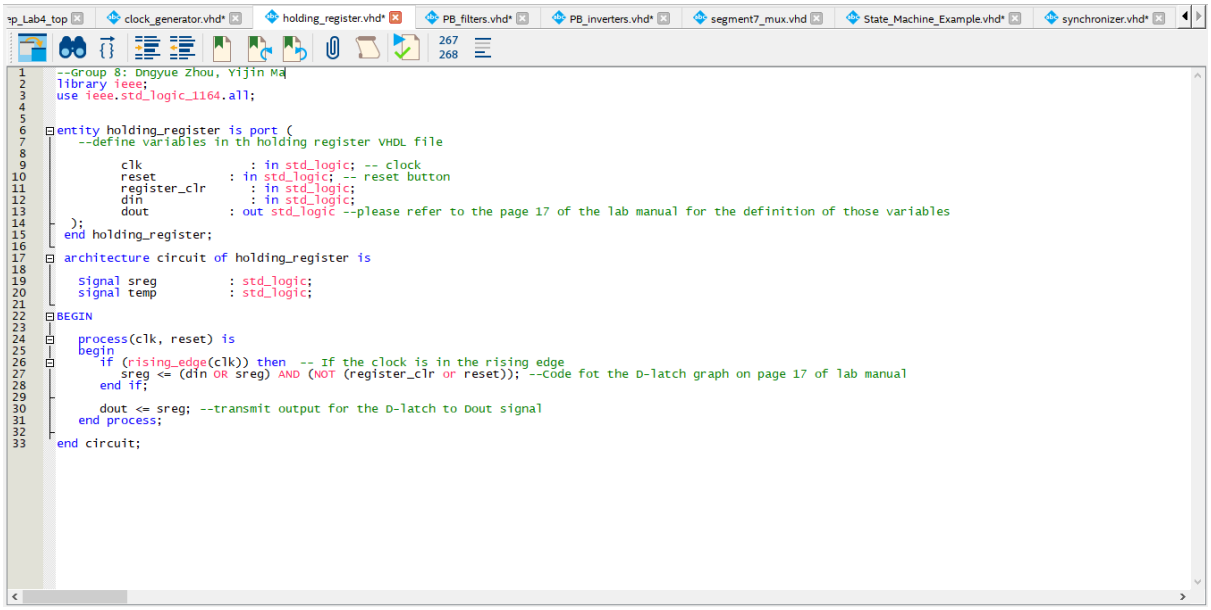
```
--Group 8: Dngyue Zhou, Yijin Ma
library ieee;
use ieee.std_logic_1164.all;

entity PB_inverters is port ( --lists of inputs and outputs signals in the PB_inverters VHDL
    rst_n           : in  std_logic;
    rst             : out std_logic;
    pb_n_filtered   : in  std_logic_vector (3 downto 0);
    pb              : out std_logic_vector(3 downto 0)
    );
end PB_inverters;

architecture ckt of PB_inverters is

begin
rst <= NOT(rst_n); --inverts the value of rst_n and transmit it into rst
pb <= NOT(pb_n_filtered); --inverts the value of pb_n_filtered and transmit it into pb


end ckt;
```

Synchronizer



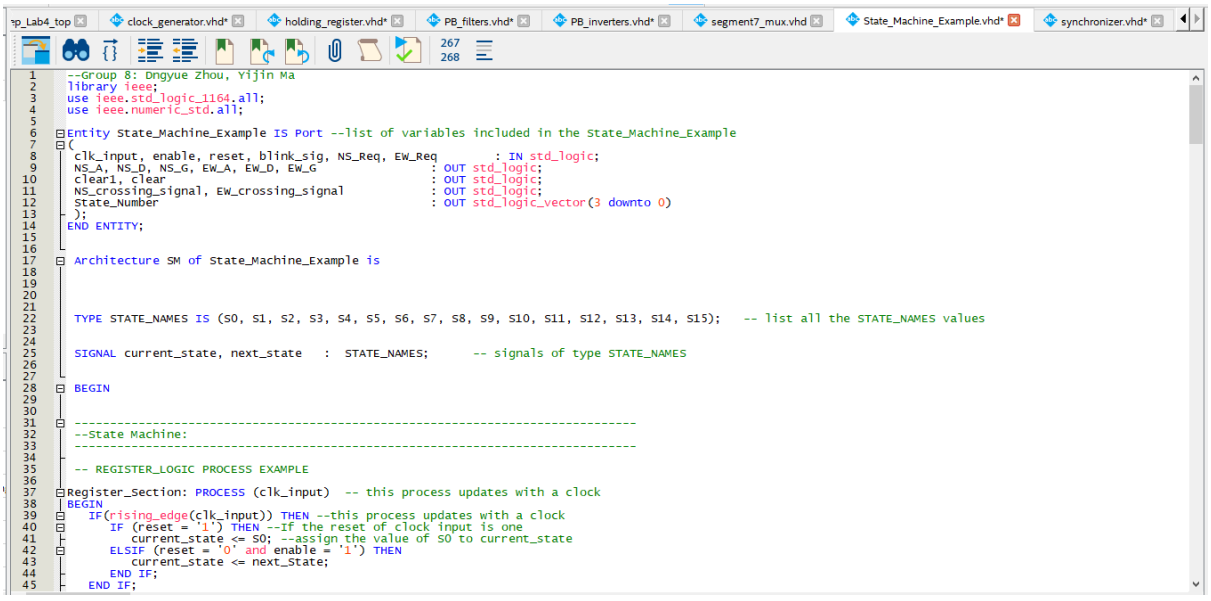
```
--Group 8: Dngyue Zhou, Yijin Ma
library ieee;
use ieee.std_logic_1164.all;


entity synchronizer is port (

        clk         : in std_logic; --clock
        reset    : in std_logic; --reset button
        din         : in std_logic; --input value
        dout     : out std_logic --output value
  );
  end synchronizer;


architecture circuit of synchronizer is

   signal sreg          : std_logic_vector(1 downto 0);

BEGIN
process(clk, reset) is

   begin

   if(rising_edge(clk)) then --if the clock is at rising edge

      if(reset='1') then --if reset button is pressed
         sreg(0)<='0'; --the value calibrates to 0 when reset button is pressed
         sreg(1)<='0';
      else          --if reset button is not pressed
         sreg(0) <= din; --assign the value of din to both sreg(0) and sreg(1)
         sreg(1) <= sreg(0);
      end if;
   end if;

      dout<= sreg(1); -- the value dout is equivalent to first digit of sreg


end process;
end;
```

Holding_register:

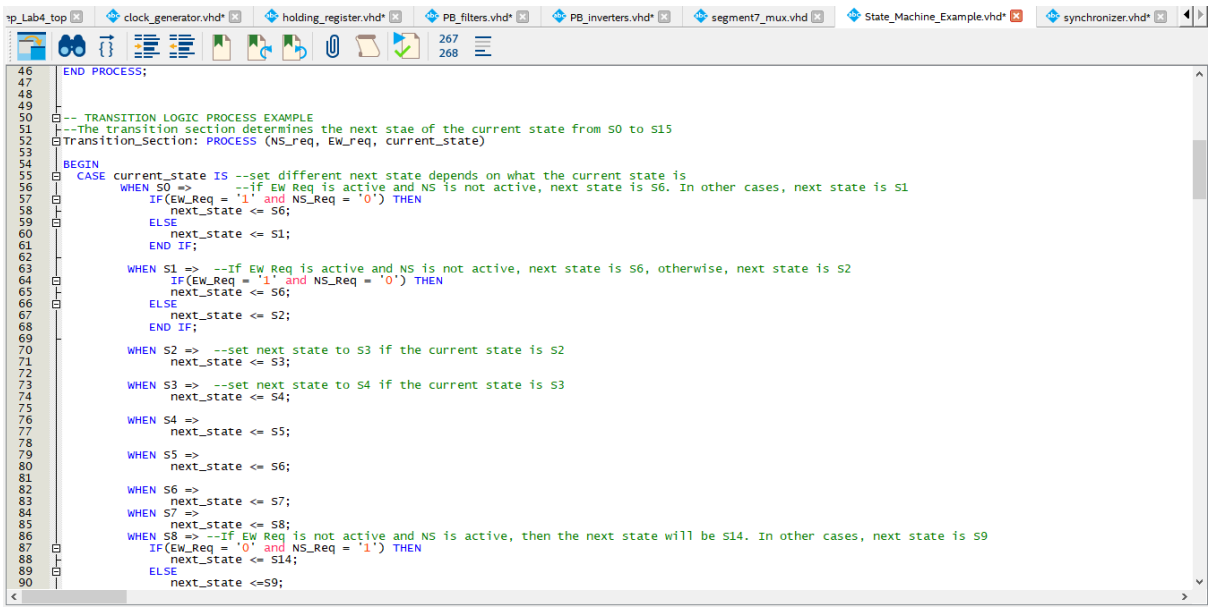


```
--Group 8: Dngyue Zhou, Yijin Ma
library ieee;
use ieee.std_logic_1164.all;

entity holding_register is port (
   --define variables in th holding register VHDL file

         clk             : in std_logic; -- clock
         reset           : in std_logic; -- reset button
         register_clr    : in std_logic;
         din             : in std_logic;
         dout            : out std_logic --please refer to the page 17 of the lab manual for the definition of those variables
 );
end holding_register;

architecture circuit of holding_register is

   signal sreg           : std_logic;
   signal temp           : std_logic;

BEGIN

   process(clk, reset) is
   begin
      if (rising_edge(clk)) then  -- If the clock is in the rising edge
         sreg <= (din OR sreg) AND (NOT (register_clr or reset)); --Code fot the D-latch graph on page 17 of lab manual
      end if;

      dout <= sreg; --transmit output for the D-latch to Dout signal
   end process;

end circuit;
```

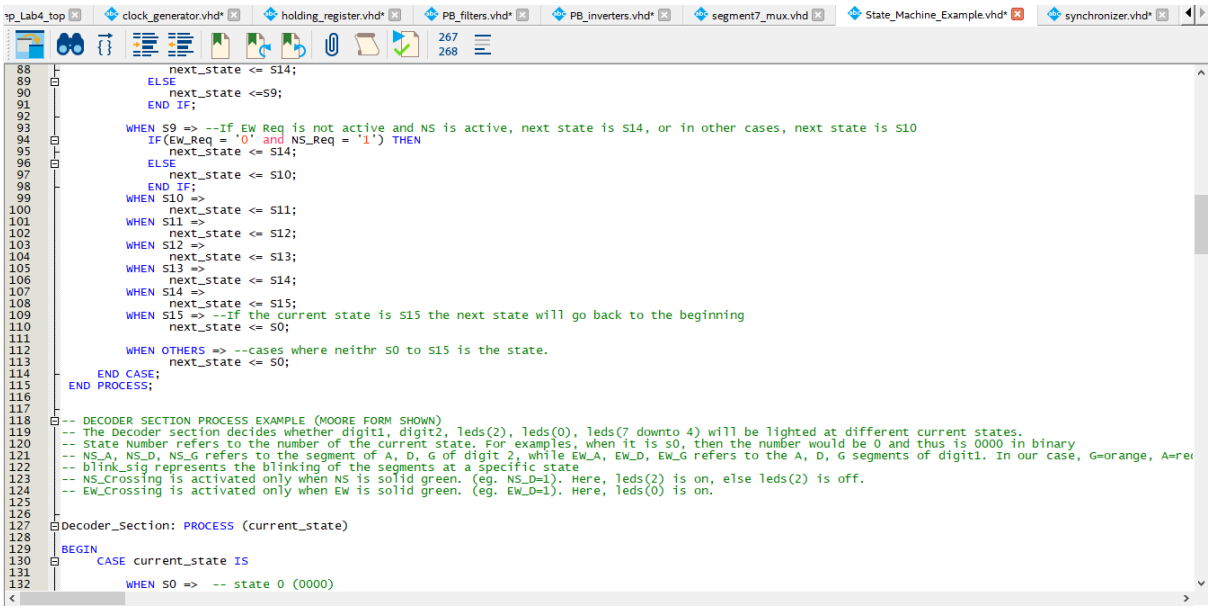
State_Machine_Example



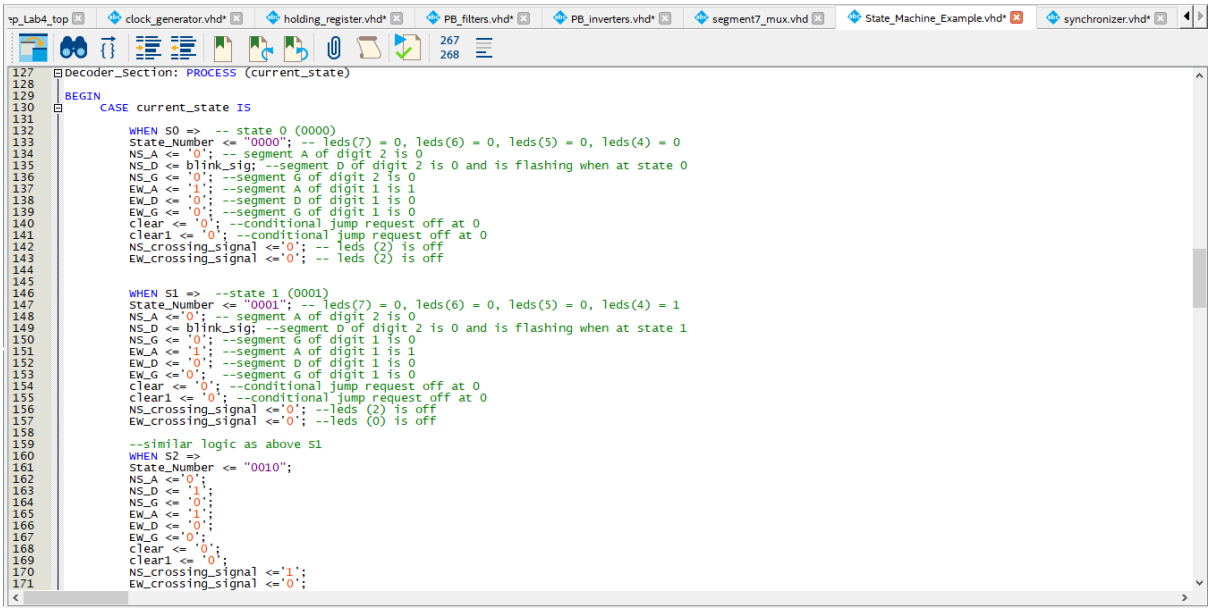
```
--Group 8: Dngyue Zhou, Yijin Ma
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity State_Machine_Example IS Port --list of variables included in the State_Machine_Example
(
 clk_input, enable, reset, blink_sig, NS_Req, EW_Req        : IN std_logic;
 NS_A, NS_D, NS_G, EW_A, EW_D, EW_G                         : OUT std_logic;
 clear1, clear                                              : OUT std_logic;
 NS_crossing_signal, EW_crossing_signal                     : OUT std_logic;
 State_Number                                               : OUT std_logic_vector(3 downto 0)
 );
END ENTITY;

Architecture SM of State_Machine_Example is



TYPE STATE_NAMES IS (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15);   -- list all the STATE_NAMES values

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

BEGIN

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

-- REGISTER_LOGIC PROCESS EXAMPLE

Register_Section: PROCESS (clk_input)  -- this process updates with a clock
BEGIN
   IF(rising_edge(clk_input)) THEN --this process updates with a clock
      IF (reset = '1') THEN --If the reset of clock input is one
         current_state <= S0; --assign the value of S0 to current_state
      ELSIF (reset = '0' and enable = '1') THEN
         current_state <= next_State;
      END IF;
   END IF;
```



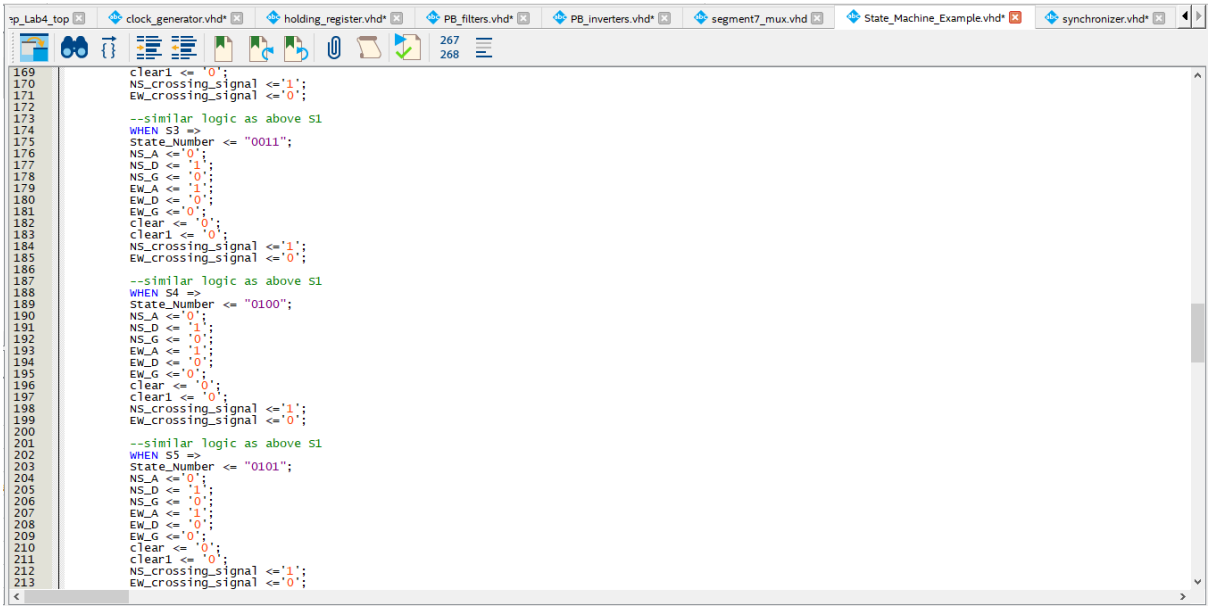
```
END PROCESS;


-- TRANSITION LOGIC PROCESS EXAMPLE
--The transition section determines the next stae of the current state from S0 to S15
Transition_Section: PROCESS (NS_req, EW_req, current_state)

BEGIN
  CASE current_state IS --set different next state depends on what the current state is
        WHEN S0 =>      --if EW Req is active and NS is not active, next state is S6. In other cases, next state is S1
            IF(EW_Req = '1' and NS_Req = '0') THEN
                next_state <= S6;
            ELSE
                next_state <= S1;
            END IF;

        WHEN S1 =>  --If EW Req is active and NS is not active, next state is S6, otherwise, next state is S2
             IF(EW_Req = '1' and NS_Req = '0') THEN
             next_state <= S6;
            ELSE
                next_state <= S2;
            END IF;

        WHEN S2 =>  --set next state to S3 if the current state is S2
                next_state <= S3;

        WHEN S3 =>  --set next state to S4 if the current state is S3
                next_state <= S4;

        WHEN S4 =>
                next_state <= S5;

        WHEN S5 =>
                next_state <= S6;

        WHEN S6 =>
                next_state <= S7;
        WHEN S7 =>
                next_state <= S8;
        WHEN S8 => --If EW Req is not active and NS is active, then the next state will be S14. In other cases, next state is S9
            IF(EW_Req = '0' and NS_Req = '1') THEN
                next_state <= S14;
            ELSE
                next_state <=S9;
```



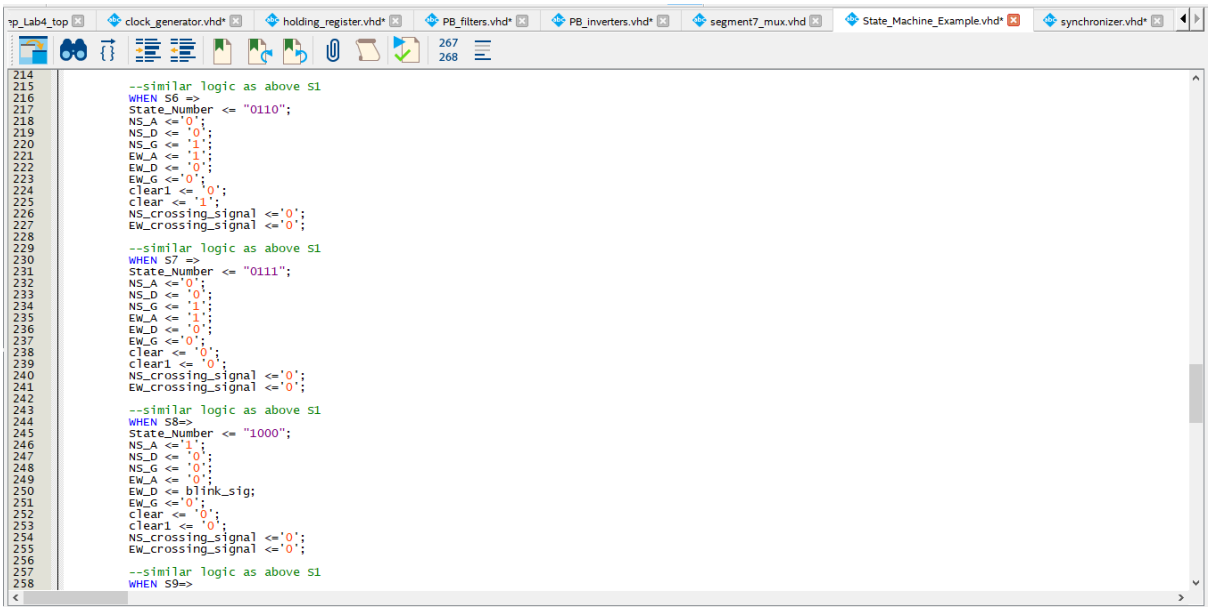
```
                next_state <= S14;
            ELSE
                next_state <=S9;
            END IF;

        WHEN S9 => --If EW Req is not active and NS is active, next state is S14, or in other cases, next state is S10
            IF(EW_Req = '0' and NS_Req = '1') THEN
                next_state <= S14;
            ELSE
                next_state <= S10;
            END IF;
        WHEN S10 =>
                next_state <= S11;
        WHEN S11 =>
                next_state <= S12;
        WHEN S12 =>
                next_state <= S13;
        WHEN S13 =>
                next_state <= S14;
        WHEN S14 =>
                next_state <= S15;
        WHEN S15 => --If the current state is S15 the next state will go back to the beginning
                next_state <= S0;

        WHEN OTHERS => --cases where neithr S0 to S15 is the state.
                next_state <= S0;
    END CASE;
  END PROCESS;


-- DECODER SECTION PROCESS EXAMPLE (MOORE FORM SHOWN)
-- The Decoder section decides whether digit1, digit2, leds(2), leds(0), leds(7 downto 4) will be lighted at different current states.
-- State Number refers to the number of the current state. For examples, when it is s0, then the number would be 0 and thus is 0000 in binary
-- NS_A, NS_D, NS_G refers to the segment of A, D, G of digit 2, while EW_A, EW_D, EW_G refers to the A, D, G segments of digit1. In our case, G=orange, A=re
-- blink_sig represents the blinking of the segments at a specific state
-- NS_Crossing is activated only when NS is solid green. (eg. NS_D=1). Here, leds(2) is on, else leds(2) is off.
-- EW_Crossing is activated only when EW is solid green. (eg. EW_D=1). Here, leds(0) is on.

Decoder_Section: PROCESS (current_state)

BEGIN
    CASE current_state IS

        WHEN S0 =>  -- state 0 (0000)
```



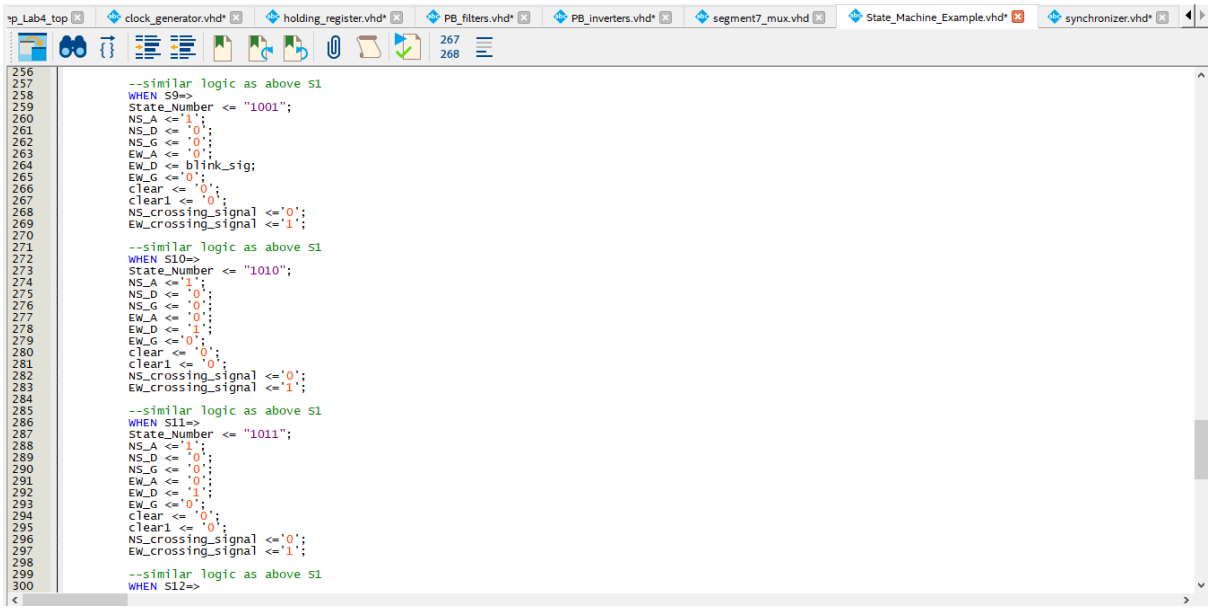
```
Decoder_Section: PROCESS (current_state)

BEGIN
    CASE current_state IS

        WHEN S0 =>  -- state 0 (0000)
        State_Number <= "0000"; -- leds(7) = 0, leds(6) = 0, leds(5) = 0, leds(4) = 0
        NS_A <= '0'; -- segment A of digit 2 is 0
        NS_D <= blink_sig; --segment D of digit 2 is 0 and is flashing when at state 0
        NS_G <= '0'; --segment G of digit 2 is 0
        EW_A <= '1'; --segment A of digit 1 is 1
        EW_D <= '0'; --segment D of digit 1 is 0
        EW_G <= '0'; --segment G of digit 1 is 0
        clear <= '0'; --conditional jump request off at 0
        clear1 <= '0'; --conditional jump request off at 0
        NS_crossing_signal <='0'; -- leds (2) is off
        EW_crossing_signal <='0'; -- leds (2) is off


        WHEN S1 =>  --state 1 (0001)
        State_Number <= "0001"; -- leds(7) = 0, leds(6) = 0, leds(5) = 0, leds(4) = 1
        NS_A <='0'; -- segment A of digit 2 is 0
        NS_D <= blink_sig;  --segment D of digit 2 is 0 and is flashing when at state 1
        NS_G <= '0'; --segment G of digit 1 is 0
        EW_A <= '1'; --segment A of digit 1 is 1
        EW_D <= '0'; --segment D of digit 1 is 0
        EW_G <='0';  --segment G of digit 1 is 0
        clear <= '0'; --conditional jump request off at 0
        clear1 <= '0'; --conditional jump request off at 0
        NS_crossing_signal <='0'; --leds (2) is off
        EW_crossing_signal <='0'; --leds (0) is off

        --similar logic as above S1
        WHEN S2 =>
        State_Number <= "0010";
        NS_A <='0';
        NS_D <= '1';
        NS_G <= '0';
        EW_A <= '1';
        EW_D <= '0';
        EW_G <='0';
        clear <= '0';
        clear1 <= '0';
        NS_crossing_signal <='1';
        EW_crossing_signal <='0';
```



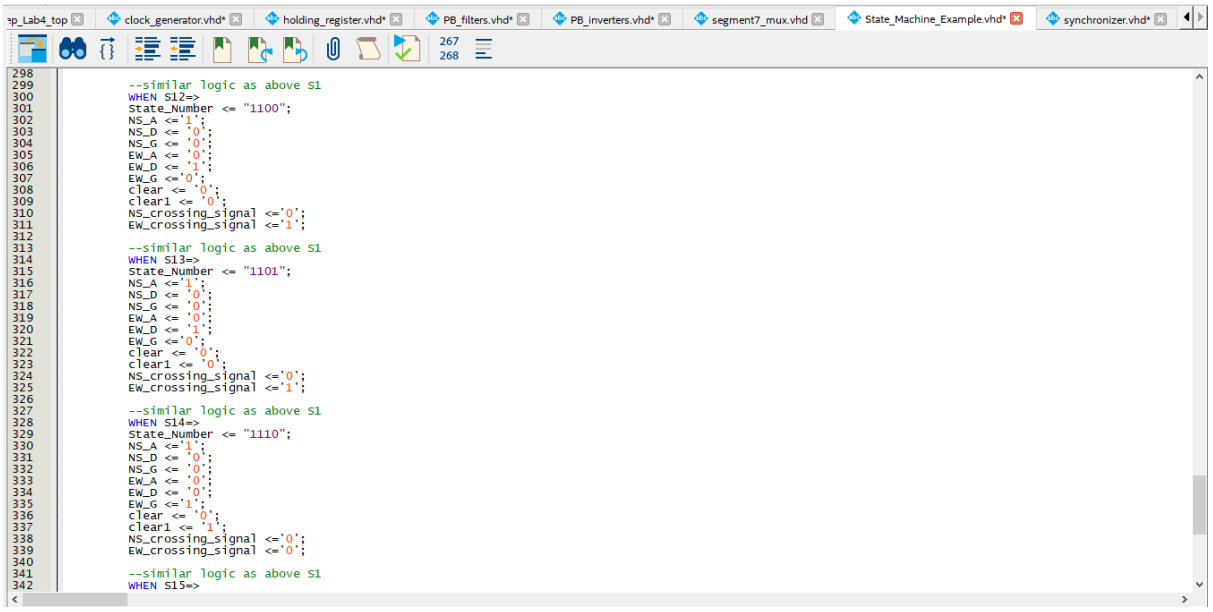
```
169         clear1 <= '0';
170         NS_crossing_signal <='1';
171         EW_crossing_signal <='0';
172
173         --similar logic as above S1
174         WHEN S3 =>
175         State_Number <= "0011";
176         NS_A <='0';
177         NS_D <= '1';
178         NS_G <= '0';
179         EW_A <= '1';
180         EW_D <= '0';
181         EW_G <='0';
182         clear <= '0';
183         clear1 <= '0';
184         NS_crossing_signal <='1';
185         EW_crossing_signal <='0';
186
187         --similar logic as above S1
188         WHEN S4 =>
189         State_Number <= "0100";
190         NS_A <='0';
191         NS_D <= '1';
192         NS_G <= '0';
193         EW_A <= '1';
194         EW_D <= '0';
195         EW_G <='0';
196         clear <= '0';
197         clear1 <= '0';
198         NS_crossing_signal <='1';
199         EW_crossing_signal <='0';
200
201         --similar logic as above S1
202         WHEN S5 =>
203         State_Number <= "0101";
204         NS_A <='0';
205         NS_D <= '1';
206         NS_G <= '0';
207         EW_A <= '1';
208         EW_D <= '0';
209         EW_G <='0';
210         clear <= '0';
211         clear1 <= '0';
212         NS_crossing_signal <='1';
213         EW_crossing_signal <='0';
```



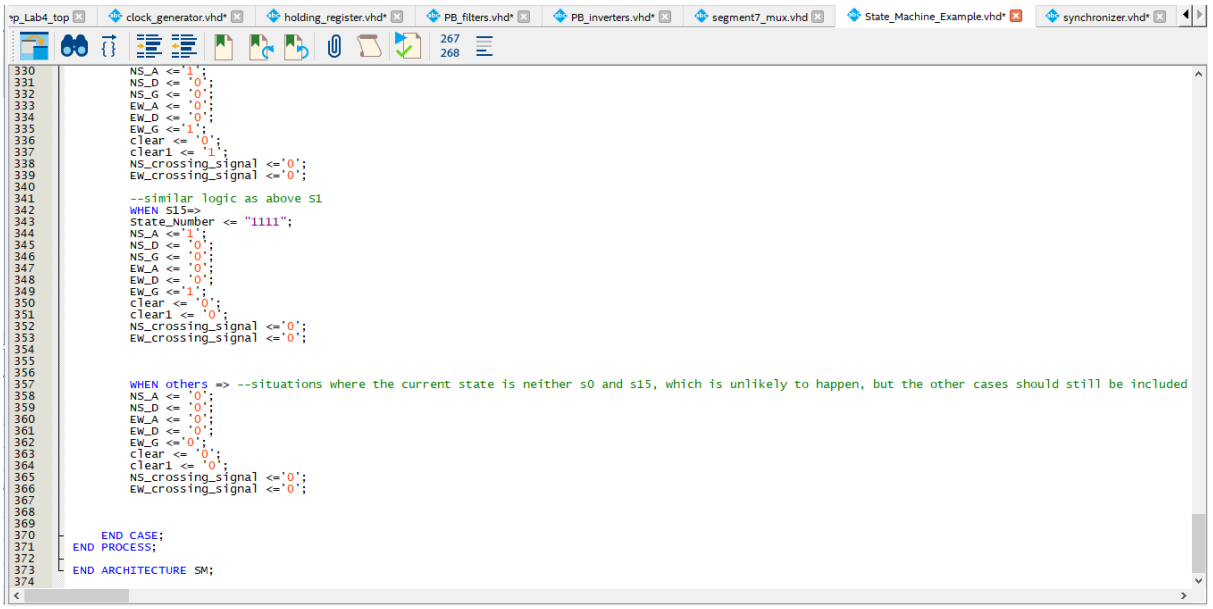
```
214
215         --similar logic as above S1
216         WHEN S6 =>
217         State_Number <= "0110";
218         NS_A <='0';
219         NS_D <= '0';
220         NS_G <= '1';
221         EW_A <= '1';
222         EW_D <= '0';
223         EW_G <='0';
224         clear1 <= '0';
225         clear <= '1';
226         NS_crossing_signal <='0';
227         EW_crossing_signal <='0';
228
229         --similar logic as above S1
230         WHEN S7 =>
231         State_Number <= "0111";
232         NS_A <='0';
233         NS_D <= '0';
234         NS_G <= '1';
235         EW_A <= '1';
236         EW_D <= '0';
237         EW_G <='0';
238         clear <= '0';
239         clear1 <= '0';
240         NS_crossing_signal <='0';
241         EW_crossing_signal <='0';
242
243         --similar logic as above S1
244         WHEN S8=>
245         State_Number <= "1000";
246         NS_A <='1';
247         NS_D <= '0';
248         NS_G <= '0';
249         EW_A <= '0';
250         EW_D <= blink_sig;
251         EW_G <='0';
252         clear <= '0';
253         clear1 <= '0';
254         NS_crossing_signal <='0';
255         EW_crossing_signal <='0';
256
257         --similar logic as above S1
258         WHEN S9=>
```



```
256
257         --similar logic as above S1
258         WHEN S9=>
259         State_Number <= "1001";
260         NS_A <='1';
261         NS_D <= '0';
262         NS_G <= '0';
263         EW_A <= '0';
264         EW_D <= blink_sig;
265         EW_G <='0';
266         clear <= '0';
267         clear1 <= '0';
268         NS_crossing_signal <='0';
269         EW_crossing_signal <='1';
270
271         --similar logic as above S1
272         WHEN S10=>
273         State_Number <= "1010";
274         NS_A <='1';
275         NS_D <= '0';
276         NS_G <= '0';
277         EW_A <= '0';
278         EW_D <= '1';
279         EW_G <='0';
280         clear <= '0';
281         clear1 <= '0';
282         NS_crossing_signal <='0';
283         EW_crossing_signal <='1';
284
285         --similar logic as above S1
286         WHEN S11=>
287         State_Number <= "1011";
288         NS_A <='1';
289         NS_D <= '0';
290         NS_G <= '0';
291         EW_A <= '0';
292         EW_D <= '1';
293         EW_G <='0';
294         clear <= '0';
295         clear1 <= '0';
296         NS_crossing_signal <='0';
297         EW_crossing_signal <='1';
298
299         --similar logic as above S1
300         WHEN S12=>
```

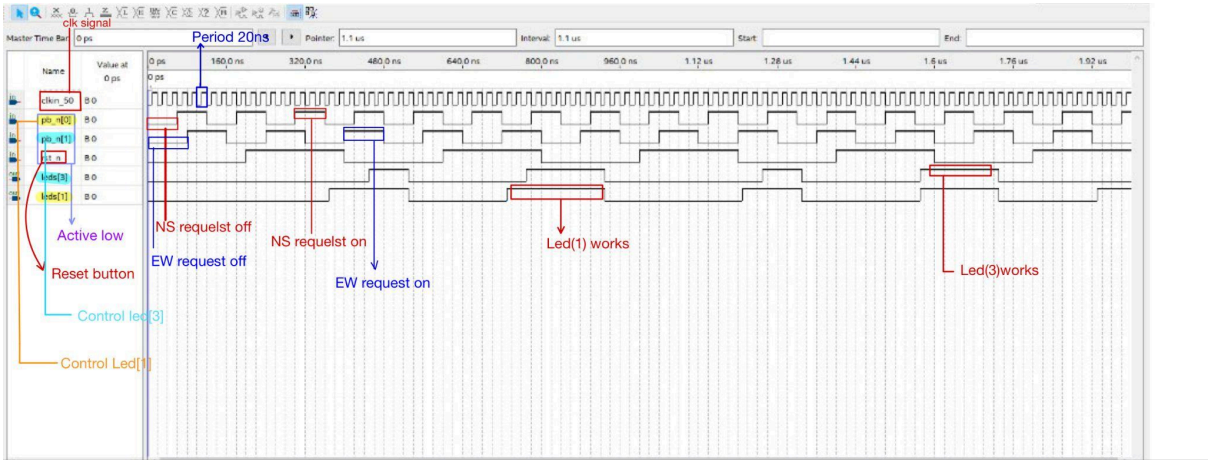


```
298
299         --similar logic as above S1
300         WHEN S12=>
301         State_Number <= "1100";
302         NS_A <='1';
303         NS_D <= '0';
304         NS_G <= '0';
305         EW_A <= '0';
306         EW_D <= '1';
307         EW_G <='0';
308         clear <= '0';
309         clear1 <= '0';
310         NS_crossing_signal <='0';
311         EW_crossing_signal <='1';
312
313         --similar logic as above S1
314         WHEN S13=>
315         State_Number <= "1101";
316         NS_A <='1';
317         NS_D <= '0';
318         NS_G <= '0';
319         EW_A <= '0';
320         EW_D <= '1';
321         EW_G <='0';
322         clear <= '0';
323         clear1 <= '0';
324         NS_crossing_signal <='0';
325         EW_crossing_signal <='1';
326
327         --similar logic as above S1
328         WHEN S14=>
329         State_Number <= "1110";
330         NS_A <='1';
331         NS_D <= '0';
332         NS_G <= '0';
333         EW_A <= '0';
334         EW_D <= '0';
335         EW_G <='1';
336         clear <= '0';
337         clear1 <= '1';
338         NS_crossing_signal <='0';
339         EW_crossing_signal <='0';
340
341         --similar logic as above S1
342         WHEN S15=>
```

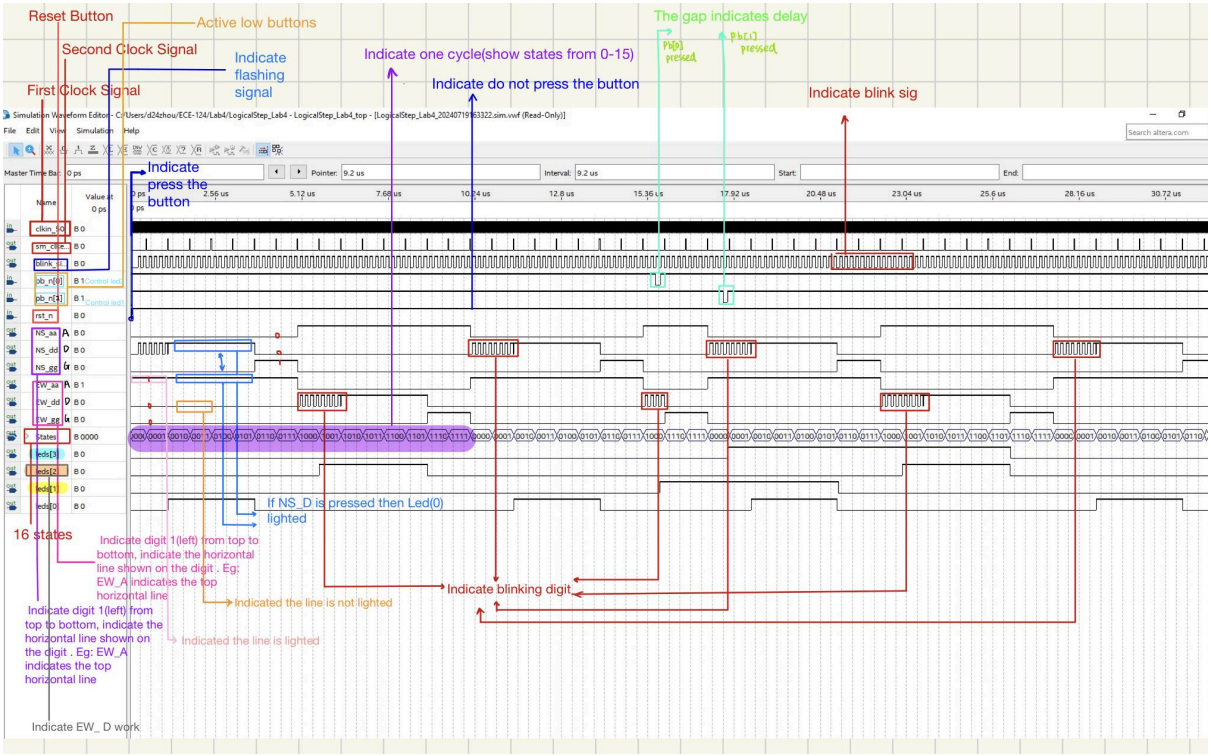


```
330         NS_A <='1';
331         NS_D <= '0';
332         NS_G <= '0';
333         EW_A <= '0';
334         EW_D <= '0';
335         EW_G <='1';
336         clear <= '0';
337         clear1 <= '1';
338         NS_crossing_signal <='0';
339         EW_crossing_signal <='0';
340
341         --similar logic as above S1
342         WHEN S15=>
343         State_Number <= "1111";
344         NS_A <='1';
345         NS_D <= '0';
346         NS_G <= '0';
347         EW_A <= '0';
348         EW_D <= '0';
349         EW_G <='1';
350         clear <= '0';
351         clear1 <= '0';
352         NS_crossing_signal <='0';
353         EW_crossing_signal <='0';
354
355
356
357         WHEN others => --situations where the current state is neither s0 and s15, which is unlikely to happen, but the other cases should still be included
358         NS_A <= '0';
359         NS_D <= '0';
360         EW_A <= '0';
361         EW_D <= '0';
362         EW_G <='0';
363         clear <= '0';
364         clear1 <= '0';
365         NS_crossing_signal <='0';
366         EW_crossing_signal <='0';
367
368
369
370     END CASE;
371 END PROCESS;
372
373 END ARCHITECTURE SM;
374
```

PartB Waveform



PartE Waveform



State_Chart

	Source State	Destination State	Condition
1	S0	S1	(!NS_Req).(!EW_Req).(Register_Section).(!reset) + (NS_Req).(Register_Section).(!reset)
2	S0	S0	(!Register_Section) + (Register_Section).(reset)
3	S0	S6	(!NS_Req).(EW_Req).(Register_Section).(!reset)
4	S1	S1	(!Register_Section).(!reset)
5	S1	S0	(reset)
6	S1	S6	(!NS_Req).(EW_Req).(Register_Section).(!reset)
7	S1	S2	(!NS_Req).(!EW_Req).(Register_Section).(!reset) + (NS_Req).(Register_Section).(!reset)
8	S2	S3	(Register_Section).(!reset)
9	S2	S0	(reset)
10	S2	S2	(!Register_Section).(!reset)
11	S3	S3	(!Register_Section).(!reset)
12	S3	S4	(Register_Section).(!reset)
13	S3	S0	(reset)
14	S4	S4	(!Register_Section).(!reset)
15	S4	S0	(reset)
16	S4	S5	(Register_Section).(!reset)
17	S5	S0	(reset)
18	S5	S6	(Register_Section).(!reset)
19	S5	S5	(!Register_Section).(!reset)
20	S6	S7	(Register_Section).(!reset)
21	S6	S0	(reset)
22	S6	S6	(!Register_Section).(!reset)
23	S7	S7	(!Register_Section).(!reset)
24	S7	S8	(Register_Section).(!reset)
25	S7	S0	(reset)
26	S8	S8	(!Register_Section).(!reset)
27	S8	S14	(NS_Req).(!EW_Req).(Register_Section).(!reset)
28	S8	S0	(reset)
29	S8	S9	(!NS_Req).(Register_Section).(!reset) + (NS_Req).(EW_Req).(Register_Section).(!reset)

29	S8	S9	(!NS_Req).(Register_Section).(!reset) + (NS_Req).(EW_Req).(Register_Section).(!reset)
30	S9	S14	(NS_Req).(!EW_Req).(Register_Section).(!reset)
31	S9	S0	(reset)
32	S9	S9	(!Register_Section).(!reset)
33	S9	S10	(!NS_Req).(Register_Section).(!reset) + (NS_Req).(EW_Req).(Register_Section).(!reset)
34	S10	S11	(Register_Section).(!reset)
35	S10	S0	(reset)
36	S10	S10	(!Register_Section).(!reset)
37	S11	S11	(!Register_Section).(!reset)
38	S11	S0	(reset)
39	S11	S12	(Register_Section).(!reset)
40	S12	S13	(Register_Section).(!reset)
41	S12	S0	(reset)
42	S12	S12	(!Register_Section).(!reset)
43	S13	S13	(!Register_Section).(!reset)
44	S13	S14	(Register_Section).(!reset)
45	S13	S0	(reset)
46	S14	S15	(Register_Section).(!reset)
47	S14	S14	(!Register_Section).(!reset)
48	S14	S0	(reset)
49	S15	S15	(!Register_Section).(!reset)
50	S15	S0	(!Register_Section).(reset) + (Register_Section)

Transitions / Encoding /

State Diagram