Lab 4 Report

ECE 124
Group 8 Session 202

Meghana Reddy Vusirika

Main Files:

- LogicalStep_Lab4_top.vhd - The main file

```
1
2   -- Lab Session 202
3   -- Team #8
4   -- Meghana Reddy Vusirika (21053635) & Okusanya Oluwatise (21054732)
5
6
7   LIBRARY ieee;
8   USE ieee.std_logic_1164.ALL;
9   USE ieee.numeric_std.ALL;
10
11
12  -- Declaring design I/O and including parameters that customize the design of the LogicalStep_Lab4_top file
13
14  entity LogicalStep_Lab4_top is port (
15
16        clkin_50    : in  std_logic;                        -- 1 bit 50 MHz FPGA Clock input
17        rst_n       : in  std_logic;                        -- 1 bit reset input (active low)
18        pb_n        : in  std_logic_vector(3 downto 0);     -- 4 bit push-button inputs (active low)
19        sw          : in  std_logic_vector(7 downto 0);     -- 8 bit switch inputs
20        leds        : out std_logic_vector(7 downto 0);     -- 8 bit output for displaying the the lab4 project details
21
22  -----------------------------------------------------------------------------------
23
24  --    you can add temporary output ports here if you need to debug your design    --
25  --    or to add internal signals for your simulations                             --
26
27  -- sm_clken1    : out std_logic;
28  -- blink_sig1   : out std_logic;
29  -- EW_a         : out std_logic;
30  -- EW_g         : out std_logic;
31  -- EW_d         : out std_logic;
32  -- NS_a         : out std_logic;
33  -- NS_g         : out std_logic;
34  -- NS_d         : out std_logic;
35
36  -----------------------------------------------------------------------------------
37
38     seg7_data   : out std_logic_vector(6 downto 0);    -- 7-bit outputs to a 7-segment
39     seg7_char1  : out std_logic;                       -- seg7 digi selectors
40     seg7_char2  : out std_logic                        -- seg7 digi selectors
41  );
42
43  END LogicalStep_Lab4_top;
44
45  ARCHITECTURE SimpleCircuit OF LogicalStep_Lab4_top IS
46
47     component segment7_mux port (                          -- digit 1 (left) = EW, digit 2 (right) = NS
48
49        clk         : in  std_logic := '0';
50        DIN2        : in  std_logic_vector(6 downto 0);     -- 7 bit input where bits 6 to 0 represent segments G,F,E,D,C,B,A
51        DIN1        : in  std_logic_vector(6 downto 0);     -- 7 bit input where bits 6 to 0 represent segments G,F,E,D,C,B,A
52        DOUT        : out std_logic_vector(6 downto 0);
53        DIG2        : out std_logic;
54        DIG1        : out std_logic
55     );
56
57     end component;
58
59     component clock_generator port (
60
61        sim_mode       : in boolean;
62        reset          : in std_logic;
63        clkin          : in  std_logic;
64        sm_clken       : out std_logic;
65        blink          : out std_logic
66     );
67
68     end component;
69
70     component pb_filters port (
71
72        clkin          : in std_logic;
73        rst_n          : in std_logic;
74        rst_n_filtered : out std_logic;
75        pb_n           : in  std_logic_vector (3 downto 0);
76        pb_n_filtered  : out std_logic_vector(3 downto 0)
77     );
78
79     end component;
```

```
    component pb_inverters port (

       rst_n          : in  std_logic;                      -- 1 bit input from reset button before inverting (active low)
       rst            : out std_logic;                      -- 1 bit output from reset input after inverting (active high)
       pb_n_filtered  : in  std_logic_vector (3 downto 0);  -- 4 bit inputs from push buttons before inverting (active low)
       pb             : out std_logic_vector(3 downto 0)    -- 4 bit outputs from push buttons after inverting (active high)
    );

    end component;

    component synchronizer port(

       clk               : in std_logic;        -- 1 bit clock signal input
       reset             : in std_logic;        -- 1 bit reset button input
       din               : in std_logic;        -- 1 bit input from pb(0), pb(1), or reset button
       dout              : out std_logic        -- 1 bit synchronizer output (to be used in holding register as an input)
    );

    end component;

    component holding_register port (

       clk            : in std_logic;           -- clock signal input
       reset          : in std_logic;           -- reset button input
       register_clr   : in std_logic;           -- crossing signal clear input
       din            : in std_logic;           -- output from synchronizer used as an input here
       dout           : out std_logic           -- output that is held for the state machine
    );

    end component;

    component State_Machine_Moore port (

    --  Inputs  --

    clk_50             : IN std_logic;     -- 1 bit clock input
    clk_input          : IN std_logic;     -- 1 bit clock input
    reset              : IN std_logic;     -- 1 bit reset input
    NS_cross_req       : IN std_logic;     -- 1 bit NS pedestrian request (outputs from holding registor) input
    EW_cross_req       : IN std_logic;     -- 1 bit EW pedestrian request (outputs from holding registor) input
    blnk               : IN std_logic;     -- 1 bit blinking signal (clock generator's output) input

    --  Outputs  --

    outNSred           : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    outNSamber         : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    outNSgreen         : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    outEWred           : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    outEWamber         : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    outEWgreen         : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    NS_clear_crossing  : OUT std_logic;    -- 1 bit EW pedestrian request (outputs from holding registor) input
    EW_clear_crossing  : OUT std_logic;    -- 1 bit EW pedestrian request (outputs from holding registor) input
    NS_cross_display   : OUT std_logic;    -- 1 bit EW pedestrian request (outputs from holding registor) input
    EW_cross_display   : OUT std_logic;    -- 1 bit EW pedestrian request (outputs from holding registor) input

    state_num          : out std_logic_vector(3 downto 0) -- 4 bit current state number input which is to be displayed using leds 7 to 4

    );

    end component;

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

    CONSTANT sim_mode                      : boolean := FALSE;  -- set to FALSE for LogicalStep board downloads and set to TRUE for SIMULATIONS
    SIGNAL rst, rst_n_filtered, synch_rst  : std_logic;
    SIGNAL sm_clken, blink_sig             : std_logic;
    SIGNAL pb_n_filtered, pb               : std_logic_vector(3 downto 0);

    signal NSout : std_logic;           -- 1 bit NS direction synchronizer output & holding registor input (signal)
    signal EWout : std_logic;           -- 1 bit EW direction synchronizer output & holding registor input (signal)

-- signal NS_CROSSING : std_logic;
-- signal EW_CROSSING : std_logic;

    signal NSred, NSamber, NSgreen, EWred, EWamber, EWgreen : std_logic; -- 1 bit NS and EW direction light signals

    signal NS_out_concat    : std_logic_vector(6 downto 0);   -- 1 bit concatenated signal for NS direction to output on 7-segment display (digit 2)
    signal EW_out_concat    : std_logic_vector(6 downto 0);   -- 1 bit concatenated signal for EW direction to output on 7-segment display (digit 1)
    signal NS_HR_pbreq      : std_logic;                      -- 1 bit holding registor output signal to state machine input
    signal EW_HR_pbreq      : std_logic;                      -- 1 bit holding registor output signal to state machine input

    signal NSClear, EWClear : std_logic;                      -- 1 bit holding register clear signals to clear crossing request


BEGIN

   NS_out_concat <= NSamber & "00" & NSgreen & "00" & NSred;  -- concatenatation for NS direction display signal [segments G,F,E,D,C,B,A] (digit 2)
   EW_out_concat <= EWamber & "00" & EWgreen & "00" & EWred;  -- concatenatation for EW direction display signal [segments G,F,E,D,C,B,A] (digit 1)

-- sm_clken1   <=  sm_clken;
-- blink_sig1  <=  blink_sig;
-- EW_a        <=  EWred;
-- EW_g        <=  EWamber;
-- EW_d        <=  EWgreen;
-- NS_a        <=  NSred;
-- NS_g        <=  NSamber;
-- NS_d        <=  NSgreen;

   leds(1) <= NS_HR_pbreq;   -- NS pedestrian crossing request displayed on leds(1)
   leds(3) <= EW_HR_pbreq;   -- EW pedestrian crossing request displayed on leds(3)

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

INST0: pb_filters port map (clkin_50, rst_n, rst_n_filtered, pb_n, pb_n_filtered);                    -- removes any "cross-talk noise glitches" on the pb_n inputs and rst_n
INST1: pb_inverters port map (rst_n_filtered, rst, pb_n_filtered, pb);                                -- converts push button inputs from active low to active high
INST2: synchronizer port map (clkin_50, synch_rst, pb(0), NSout);                                    -- synchronizer for NS direction also reset by synch_rst
INST3: synchronizer port map (clkin_50, synch_rst, pb(1), EWout);                                    -- synchronizer for EW direction
INST4: synchronizer port map (clkin_50, synch_rst, rst, synch_rst);                                  -- synchronizer for reset button
INST5: clock_generator port map (sim_mode, synch_rst, clkin_50, sm_clken, blink_sig);                -- state machine's clock and blink signals are created
INST6: holding_register port map (clkin_50, synch_rst, NSClear, NSout, NS_HR_pbreq);                 -- holding registor for NS direction
INST7: holding_register port map (clkin_50, synch_rst, EWClear, EWout, EW_HR_pbreq);                 -- holding registor for EW direction
INST8: State_Machine_Moore port map(clkin_50, sm_clken, synch_rst, NS_HR_pbreq,EW_HR_pbreq, blink_sig,    -- Moore State Machine
NSred, NSamber, NSgreen, EWred, EWamber, EWgreen, NSClear, EWClear, leds(0), leds(2), leds(7 downto 4));
INST9: segment7_mux port map (clkin_50, NS_out_concat, EW_out_concat, seg7_data, seg7_char2, seg7_char1); -- creates output to display on digits 1 and 2 of the 7-segment display on the FPGA board

END SimpleCircuit;
```

Sub Files:

- synchronizer.vhd

```vhdl
-- Lab Session 202
-- Team #8
-- Meghana Reddy Vusirika (21053635) & Okusanya Oluwatise (21054732)

library ieee;
use ieee.std_logic_1164.all;

entity synchronizer is port (

    clk      : in std_logic;          -- 1 bit clock signal input
    reset    : in std_logic;          -- 1 bit reset button input
    din      : in std_logic;          -- 1 bit input from pb(0), pb(1), or reset button
    dout     : out std_logic          -- 1 bit synchronizer output (to be used in holding register as an input)

);
end synchronizer;

architecture circuit of synchronizer is

Signal sreg : std_logic_vector(1 downto 0);    -- 2 bit temporary signal (as there are 2 flip-flops)
                                               -- sreg(0) = signal that acts as the output of D-flip-flop 1 and input of D-flip-flop 2
                                               -- sreg(1) = signal that acts as the output D-flip-flop 2


BEGIN

PROCESS (clk) is                           -- process updates with a clock

begin

   if(rising_edge(clk)) then               -- when the clock signal is on its rising edge

      sreg(0) <= (not reset and din);      -- when the reset button is not pressed, sreg signal's 0th bit is set to din (input signal)
      sreg(1) <= (not reset and sreg(0));  -- when the reset button is not pressed, sreg signal's 0th bit is shifted to its 1st bit

   end if;
end process;

dout <= sreg(1);                           -- value of dout (output of the synchronizer) is set to the value of the sreg signal's 1st bit

end circuit;
```

- PB_inverters.vhd

```vhdl
-- Lab Session 202
-- Team #8
-- Meghana Reddy Vusirika (21053635) & Okusanya Oluwatise (21054732)

library ieee;
use ieee.std_logic_1164.all;

entity PB_inverters is port (                          -- converts push button inputs from active low to active high

    rst_n          : in  std_logic;                    -- 1 bit input from reset button before inverting (active low)
    rst            : out std_logic;                    -- 1 bit input from reset input after inverting (active high)
    pb_n_filtered  : in  std_logic_vector (3 downto 0);   -- 4 bit inputs from push buttons before inverting (active low)
    pb             : out std_logic_vector(3 downto 0)     -- 4 bit inputs from push buttons after inverting (active high)

);
end PB_inverters;

architecture ckt of PB_inverters is

-- convert active low inputs to active high using the NOT operator

begin

    rst <= not(rst_n);             -- assigning the inverted input of the rst_n to rst
    pb <= not(pb_n_filtered);      -- assigning the inverted input of the rst_n to rst

end ckt;
```

- holding_register.vhd

```
-- Lab Session 202
-- Team #8
-- Meghana Reddy Vusirika (21053635) & Okusanya Oluwatise (21054732)

library ieee;
use ieee.std_logic_1164.all;

entity holding_register is port (

    clk            : in std_logic;        -- 1 bit clock signal input
    reset          : in std_logic;        -- 1 bit reset button input
    register_clr   : in std_logic;        -- 1 bit crossing signal clear input
    din            : in std_logic;        -- 1 bit output from synchronizer used as an input here
    dout           : out std_logic        -- 1 bit output that is held for the state machine

);
end holding_register;

architecture circuit of holding_register is

    Signal sreg  : std_logic;

BEGIN
PROCESS (clk) is                                                -- process updates with a clock

begin

    if(rising_edge(clk)) then                                   -- when the clock signal is on its rising edge

        if (reset = '1') then                                   -- when the reset button is pressed
            sreg <= '0';                                        -- the value of the sreg signal is set to 0 (holding register is cleared)

        else
            sreg <= ((sreg or din) and not (register_clr or reset));    -- value of the combinational logic between sreg, din, and
                                                                        -- register_clr is updated to the sreg signal
        end if;

    end if;

    dout <= sreg;                                               -- value of dout (output of holding register) is set to the value
                                                                -- of the sreg signal (this value is used as an input to the state machine)

end process;
end;
```

- State_Machine_Moore.vhd

```

-- Lab Session 202
-- Team #8
-- Meghana Reddy Vusirika (21053635) & Okusanya Oluwatise (21054732)

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

entity State_Machine_Moore is port (

    --  Inputs  --

    clk_50              : IN std_logic;     -- 1 bit clock input
    clk_input           : IN std_logic;     -- 1 bit clock input
    reset               : IN std_logic;     -- 1 bit reset input
    NS_cross_req        : IN std_logic;     -- 1 bit NS pedestrian request (outputs from holding registor) input
    EW_cross_req        : IN std_logic;     -- 1 bit EW pedestrian request (outputs from holding registor) input
    blnk                : IN std_logic;     -- 1 bit blinking signal (clock generator's output) input

    --  Outputs  --

    outNSred            : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    outNSamber          : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    outNSgreen          : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    outEWred            : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    outEWamber          : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    outEWgreen          : OUT std_logic;    -- 1 bit NS direction lights [red(seg a)] input
    NS_clear_crossing   : OUT std_logic;    -- 1 bit EW pedestrian request (outputs from holding registor) input
    EW_clear_crossing   : OUT std_logic;    -- 1 bit EW pedestrian request (outputs from holding registor) input
    NS_cross_display    : OUT std_logic;    -- 1 bit EW pedestrian request (outputs from holding registor) input
    EW_cross_display    : OUT std_logic;    -- 1 bit EW pedestrian request (outputs from holding registor) input

    state_num           : out std_logic_vector(3 downto 0) -- 4 bit current state number input which is to be displayed using leds 7 to 4
);
end entity;

```

```
Architecture SM of State_Machine_Moore is

 TYPE STATE_NAMES IS (S0, S1, S2, S3, S4, S5, S6, S7, s8, s9, s10, s11, s12, s13, s14, s15);  -- values for the traffic light's 16 states (0-15)

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


BEGIN

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

-- REGISTER_LOGIC PROCESS

Register_Section: PROCESS (clk_input)              -- this process updates with a clock

BEGIN

  IF(rising_edge(clk_50)) THEN                     -- when the clock signal is on its rising edge

     IF (reset = '1') THEN                         -- when the reset button is pressed
        current_state <= S0;                       -- the state jumps to state 0

     ELSIF (reset = '0' and clk_input = '1') THEN  -- when reset button is not pressed and the state machine clock enable signal is active
        current_state <= next_State;               -- the state machine moves on to the next state (current state is updated to the next state value)

     END IF;

  END IF;

END PROCESS;


-- TRANSITION LOGIC PROCESS

Transition_Section: PROCESS (NS_cross_req, EW_cross_req, current_state) -- NS_cross_req = NS (pb(0) pushed, pb(1) not)
                                                                        -- EW_cross_req = EW (pb(1) pushed, pb(0) not)

 BEGIN

  CASE current_state IS

          WHEN S0 =>

          -- state jumps from 0 to 6 if crossing request is made in NS and not in EW (only pb(0) pushed)
          -- state goes to next state (1) otherwise

             IF(NS_cross_req='0' and EW_cross_req='1') THEN
                next_state <= S6;
             ELSE
                next_state <= S1;
             END IF;

          WHEN S1 =>

          -- state jumps from 1 to 6 if crossing request is made in NS and not in EW (only pb(0) pushed)
          -- state goes to next state (2) otherwise

             IF(NS_cross_req='0' and EW_cross_req='1') THEN
                next_state <= S6;
             ELSE
                next_state <= S2;
             END IF;

          WHEN S2 =>                  -- no button is pushed, so go to next state
             next_state <= S3;

          WHEN S3 =>                  -- no button is pushed, so go to next state
             next_state <= S4;

          WHEN S4 =>                  -- no button is pushed, so go to next state
             next_state <= S5;

          WHEN S5 =>                  -- no button is pushed, so go to next state
             next_state <= S6;

          WHEN S6 =>                  -- no button is pushed, so go to next state
                next_state <= S7;

          WHEN S7 =>                  -- no button is pushed, so go to next state
                next_state <= S8;

          WHEN S8 =>

          -- state jumps from 8 to 14 if crossing request is made in EW and not in NS (only pb(1) pushed)
          -- state goes to next state (9) otherwise

             IF(NS_cross_req='1' and EW_cross_req='0') THEN
                next_state <= S14;
             ELSE
                next_state <= S9;
             END IF;
```

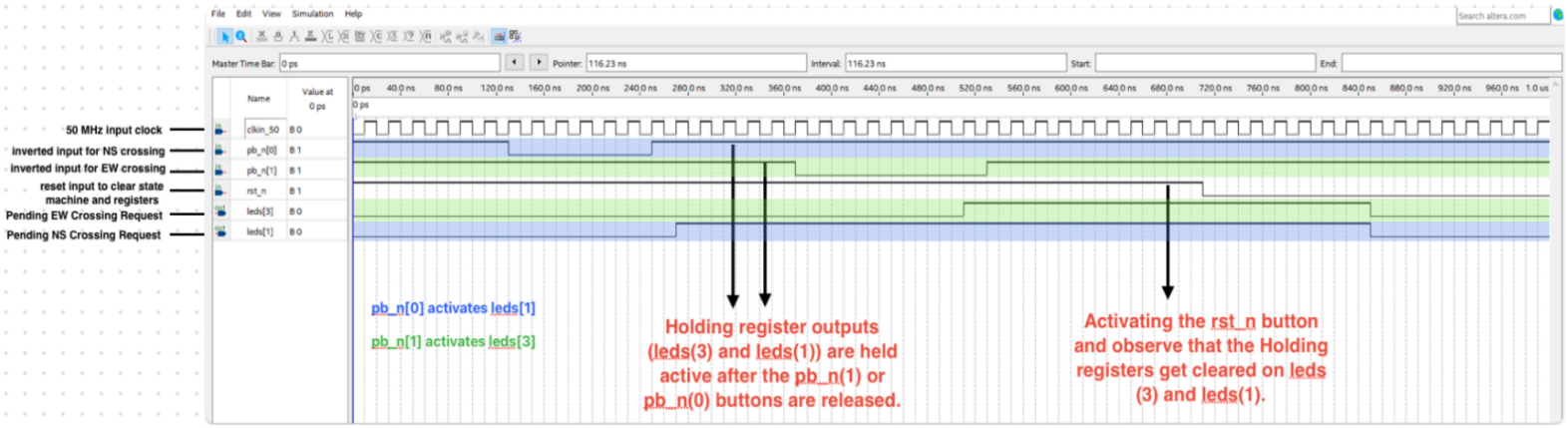
```
        WHEN S9 =>

        -- state jumps from 9 to 14 if crossing request is made in EW and not in NS (only pb(1) pushed)
        -- state goes to next state (10) otherwise

           IF(NS_cross_req='1' and EW_cross_req='0') THEN
              next_state <= S14;
           ELSE
              next_state <= S10;
           END IF;

        WHEN S10 =>                --no button is pushed, so go to next state
           next_state <= S11;

        WHEN S11 =>                --no button is pushed, so go to next state
           next_state <= S12;

        WHEN S12 =>                --no button is pushed, so go to next state
           next_state <= S13;

        WHEN S13 =>                --no button is pushed, so go to next state
           next_state <= S14;

        WHEN S14 =>                --no button is pushed, so go to next state
           next_state <= S15;

        WHEN S15 =>                --no button is pushed, so go to next state (goes back to state 0)
           next_state <= S0;

     END CASE;

 END PROCESS;


-- DECODER SECTION PROCESS


Decoder_Section: PROCESS (current_state, blnk)
-- outNSred    =  NS direction red light (seg a)
-- outNSamber  =  NS direction amber light (seg g)
-- outNSgreen  =  NS direction green light (seg d)
-- outEWred    =  EW direction red light (seg d)
-- outEWamber  =  EW direction amber light (seg d)
-- outEWgreen  =  EW direction green light (seg d)

BEGIN

     CASE current_state IS

         WHEN S0 =>            -- NS blinking green light; EW red light
         outNSred <= '0';
         outNSamber <= '0';
         outNSgreen <= blnk;
         outEWred <= '1';
         outEWamber <= '0';
         outEWgreen <= '0';
         state_num <= "0000";
         NS_clear_crossing <= '0';
         EW_clear_crossing <= '0';
         NS_cross_display <= '0';
         EW_cross_display <= '0';

         WHEN S1 =>            -- NS blinking green light; EW red light
         outNSred <= '0';
         outNSamber <= '0';
         outNSgreen <= blnk;
         outEWred <= '1';
         outEWamber <= '0';
         outEWgreen <= '0';
         state_num <= "0001";
         NS_clear_crossing <= '0';
         EW_clear_crossing <= '0';
         NS_cross_display <= '0';
         EW_cross_display <= '0';

         WHEN S2 =>            -- NS green light; EW red light
         outNSred <= '0';
         outNSamber <= '0';
         outNSgreen <= '1';
         outEWred <= '1';
         outEWamber <= '0';
         outEWgreen <= '0';
         state_num <= "0010";
         NS_clear_crossing <= '0';
         EW_clear_crossing <= '0';
         NS_cross_display <= '1';   -- NS crossing display: leds(0) turns on
         EW_cross_display <= '0';
```

```
226
227         WHEN S3 =>            -- NS green light; EW red light
228         outNSred <= '0';
229         outNSamber <= '0';
230         outNSgreen <= '1';
231         outEWred <= '1';
232         outEWamber <= '0';
233         outEWgreen <= '0';
234         state_num <= "0011";
235         NS_clear_crossing <= '0';
236         EW_clear_crossing <= '0';
237         NS_cross_display <= '1';   -- NS crossing display: leds(0) turns on
238         EW_cross_display <= '0';
239
240         WHEN S4 =>            -- NS green light; EW red light
241         outNSred <= '0';
242         outNSamber <= '0';
243         outNSgreen <= '1';
244         outEWred <= '1';
245         outEWamber <= '0';
246         outEWgreen <= '0';
247         state_num <= "0100";
248         NS_clear_crossing <= '0';
249         EW_clear_crossing <= '0';
250         NS_cross_display <= '1';   -- NS crossing display: leds(0) turns on
251         EW_cross_display <= '0';
252
253         WHEN S5 =>            --NS green light; EW red light
254         outNSred <= '0';
255         outNSamber <= '0';
256         outNSgreen <= '1';
257         outEWred <= '1';
258         outEWamber <= '0';
259         outEWgreen <= '0';
260         state_num <= "0101";
261         NS_clear_crossing <= '0';
262         EW_clear_crossing <= '0';
263         NS_cross_display <= '1';   -- NS crossing display: leds(0) turns on
264         EW_cross_display <= '0';
265
266         WHEN S6 =>            -- NS amber light; EW red light
267         outNSred <= '0';
268         outNSamber <= '1';
269         outNSgreen <= '0';
270         outEWred <= '1';
271         outEWamber <= '0';
272         outEWgreen <= '0';
273         state_num <= "0110";
274         NS_clear_crossing <= '1';  -- NS crossing display: leds(0) turns off (cleared)
275         EW_clear_crossing <= '0';
276         NS_cross_display <= '0';
277         EW_cross_display <= '0';
278
279         WHEN S7 =>            -- NS amber light; EW red light
280         outNSred <= '0';
281         outNSamber <= '1';
282         outNSgreen <= '0';
283         outEWred <= '1';
284         outEWamber <= '0';
285         outEWgreen <= '0';
286         state_num <= "0111";
287         NS_clear_crossing <= '0';
288         EW_clear_crossing <= '0';
289         NS_cross_display <= '0';
290         EW_cross_display <= '0';
291
292         WHEN S8 =>            -- NS red light; EW blinking green light
293         outNSred <= '1';
294         outNSamber <= '0';
295         outNSgreen <= '0';
296         outEWred <= '0';
297         outEWamber <= '0';
298         outEWgreen <= blnk;
299         state_num <= "1000";
300         NS_clear_crossing <= '0';
301         EW_clear_crossing <= '0';
302         NS_cross_display <= '0';
303         EW_cross_display <= '0';
304
305         WHEN S9 =>            -- NS red light; EW blinking green light
306         outNSred <= '1';
307         outNSamber <= '0';
308         outNSgreen <= '0';
309         outEWred <= '0';
310         outEWamber <= '0';
311         outEWgreen <= blnk;
312         state_num <= "1001";
313         NS_clear_crossing <= '0';
314         EW_clear_crossing <= '0';
315         NS_cross_display <= '0';
316         EW_cross_display <= '0';
317
```

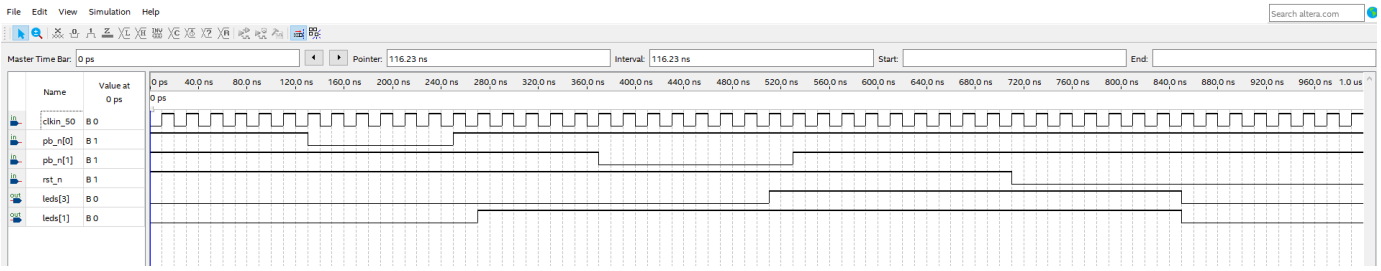
```
318         WHEN S10 =>          -- NS red light; EW green light
319         outNSred <= '1';
320         outNSamber <= '0';
321         outNSgreen <= '0';
322         outEWred <= '0';
323         outEWamber <= '0';
324         outEWgreen <= '1';
325         state_num <= "1010";
326         NS_clear_crossing <= '0';
327         EW_clear_crossing <= '0';
328         NS_cross_display <= '0';
329         EW_cross_display <= '1';   -- EW crossing display: leds(2) turns on
330
331         WHEN S11 =>          -- NS red light; EW green light
332         outNSred <= '1';
333         outNSamber <= '0';
334         outNSgreen <= '0';
335         outEWred <= '0';
336         outEWamber <= '0';
337         outEWgreen <= '1';
338         state_num <= "1011";
339         NS_clear_crossing <= '0';
340         EW_clear_crossing <= '0';
341         NS_cross_display <= '0';
342         EW_cross_display <= '1';   -- EW crossing display: leds(2) turns on
343
344         WHEN S12 =>          -- NS red light; EW green light
345         outNSred <= '1';
346         outNSamber <= '0';
347         outNSgreen <= '0';
348         outEWred <= '0';
349         outEWamber <= '0';
350         outEWgreen <= '1';
351         state_num <= "1100";
352         NS_clear_crossing <= '0';
353         EW_clear_crossing <= '0';
354         NS_cross_display <= '0';
355         EW_cross_display <= '1';   -- EW crossing display: leds(2) turns on
356
357         WHEN S13 =>          -- NS red light; EW green light
358         outNSred <= '1';
359         outNSamber <= '0';
360         outNSgreen <= '0';
361         outEWred <= '0';
362         outEWamber <= '0';
363         outEWgreen <= '1';
364         state_num <= "1101";
365         NS_clear_crossing <= '0';
366         EW_clear_crossing <= '0';
367         NS_cross_display <= '0';
368         EW_cross_display <= '1';   -- EW crossing display: leds(2) turns on
369
370         WHEN S14 =>          -- NS red light; EW amber light
371         outNSred <= '1';
372         outNSamber <= '0';
373         outNSgreen <= '0';
374         outEWred <= '0';
375         outEWamber <= '1';
376         outEWgreen <= '0';
377         state_num <= "1110";
378         NS_clear_crossing <= '0';
379         EW_clear_crossing <= '1';  -- EW crossing display: led(2) turns off (cleared)
380         NS_cross_display <= '0';
381         EW_cross_display <= '0';
382
383         WHEN S15 =>          -- NS red light; EW amber light
384         outNSred <= '1';
385         outNSamber <= '0';
386         outNSgreen <= '0';
387         outEWred <= '0';
388         outEWamber <= '1';
389         outEWgreen <= '0';
390         state_num <= "1111";
391         NS_clear_crossing <= '0';
392         EW_clear_crossing <= '0';
393         NS_cross_display <= '0';
394         EW_cross_display <= '0';
395
396         WHEN others =>       -- all segment lights off
397         outNSred <= '0';
398         outNSamber <= '0';
399         outNSgreen <= '0';
400         outEWred <= '0';
401         outEWamber <= '0';
402         outEWgreen <= '0';
403         state_num <= "0000";
404         NS_clear_crossing <= '0';
405         EW_clear_crossing <= '0';
406         NS_cross_display <= '0';
407         EW_cross_display <= '0';
408
409     END CASE;
410
411 END PROCESS;
412
413 END ARCHITECTURE SM;
414
```

Simulations:

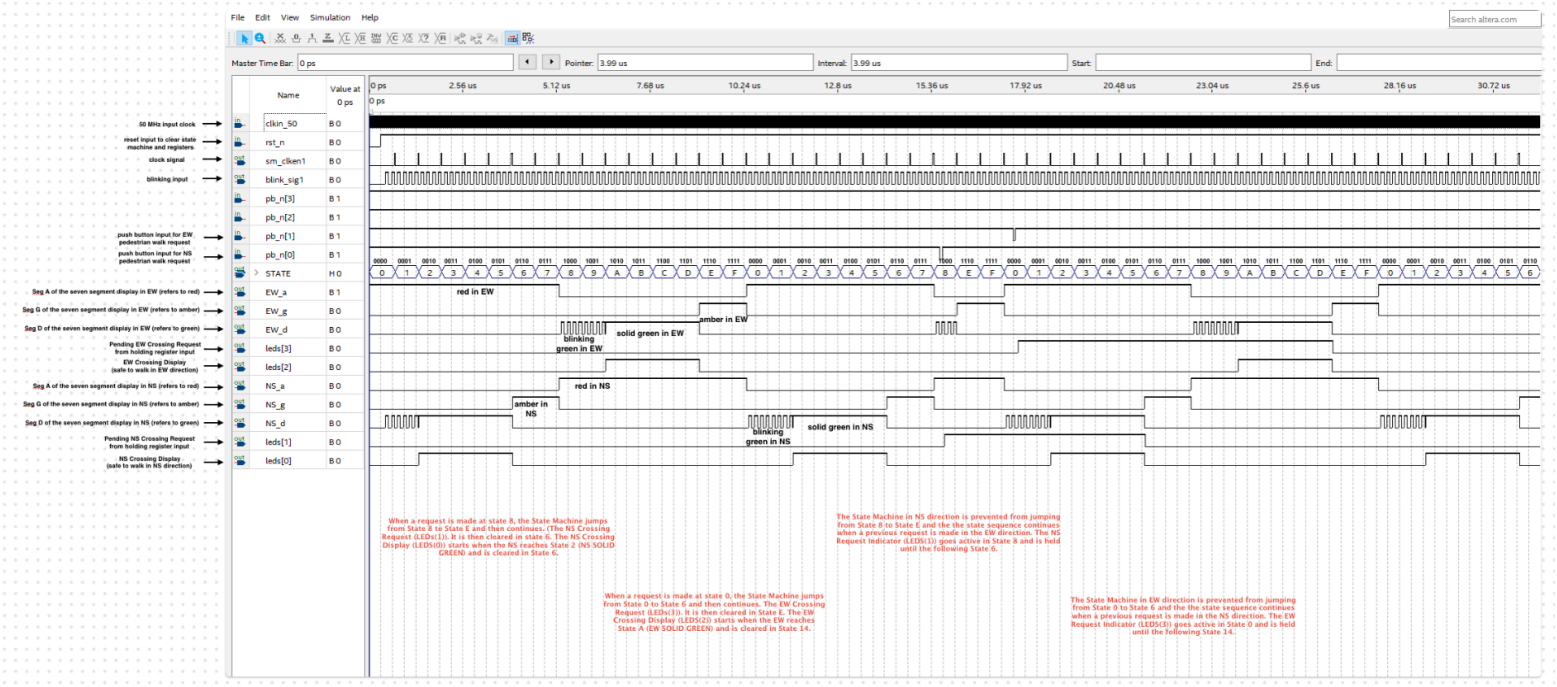
- Synchronizer and Holding Register Simulation (from Part B) (Annotated)



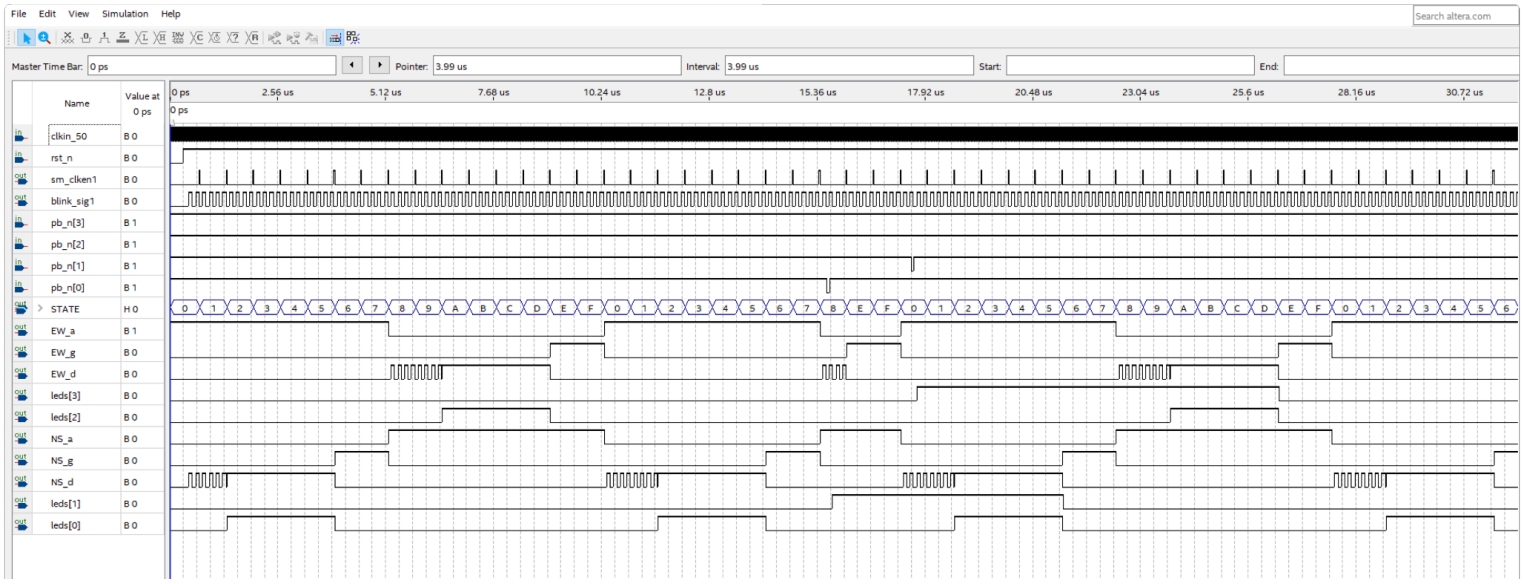
(Original)



- Student 1 Simulation (from Part H) (Annotated)

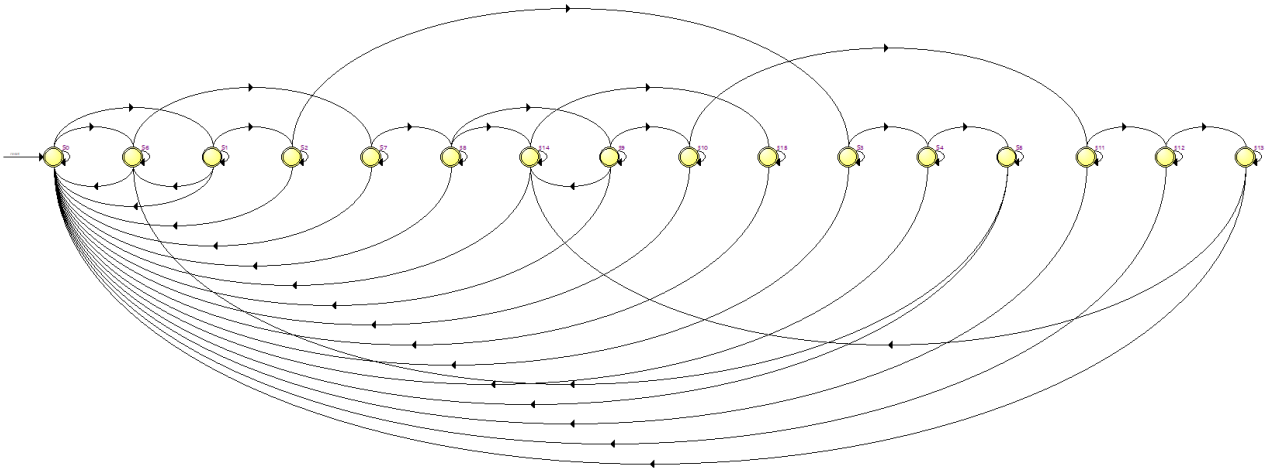


(Original)



State Machine:

- State_Machine Diagram



- State_Machine Table

	Source State	Destination State	Condition
1	S0	S6	(!NS_cross_req).(EW_cross_req).(Register_Section).(!reset)
2	S0	S1	(!NS_cross_req).(!EW_cross_req).(Register_Section).(!reset) + (NS_cross_req).(Register_Section).(!reset)
3	S0	S0	(!Register_Section) + (Register_Section).(reset)
4	S1	S6	(!NS_cross_req).(EW_cross_req).(Register_Section).(!reset)
5	S1	S2	(!NS_cross_req).(!EW_cross_req).(Register_Section).(!reset) + (NS_cross_req).(Register_Section).(!reset)
6	S1	S1	(!Register_Section).(!reset)
7	S1	S0	(reset)
8	S2	S2	(!Register_Section).(!reset)
9	S2	S3	(Register_Section).(!reset)
10	S2	S0	(reset)
11	S3	S3	(!Register_Section).(!reset)
12	S3	S0	(reset)
13	S3	S4	(Register_Section).(!reset)
14	S4	S0	(reset)
15	S4	S5	(Register_Section).(!reset)
16	S4	S4	(!Register_Section).(!reset)
17	S5	S6	(Register_Section).(!reset)
18	S5	S0	(reset)
19	S5	S5	(!Register_Section).(!reset)
20	S6	S6	(!Register_Section).(!reset)
21	S6	S7	(Register_Section).(!reset)
22	S6	S0	(reset)
23	S7	S7	(!Register_Section).(!reset)
24	S7	s8	(Register_Section).(!reset)
25	S7	S0	(reset)
26	s8	s9	(!NS_cross_req).(Register_Section).(!reset) + (NS_cross_req).(EW_cross_req).(Register_Section).(!reset)
27	s8	s8	(!Register_Section).(!reset)
28	s8	S0	(reset)
29	s8	s14	(NS_cross_req).(!EW_cross_req).(Register_Section).(!reset)
30	s9	s9	(!Register_Section).(!reset)

31	s9	S0	(reset)
32	s9	s10	(!NS_cross_req).(Register_Section).(!reset) + (NS_cross_req).(EW_cross_req).(Register_Section).(!reset)
33	s9	s14	(NS_cross_req).(!EW_cross_req).(Register_Section).(!reset)
34	s10	S0	(reset)
35	s10	s10	(!Register_Section).(!reset)
36	s10	s11	(Register_Section).(!reset)
37	s11	S0	(reset)
38	s11	s11	(!Register_Section).(!reset)
39	s11	s12	(Register_Section).(!reset)
40	s12	S0	(reset)
41	s12	s12	(!Register_Section).(!reset)
42	s12	s13	(Register_Section).(!reset)
43	s13	S0	(reset)
44	s13	s13	(!Register_Section).(!reset)
45	s13	s14	(Register_Section).(!reset)
46	s14	S0	(reset)
47	s14	s14	(!Register_Section).(!reset)
48	s14	s15	(Register_Section).(!reset)
49	s15	S0	(!Register_Section).(reset) + (Register_Section)
50	s15	s15	(!Register_Section).(!reset)