

Lab 4 Report

ECE 124

Group 8 Session 202

Meghana Reddy Vusirika

Main Files:

- LogicalStep_Lab4_top.vhd - The main file

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
-- 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;

-- Declaring design I/O and including parameters that customize the design of the LogicalStep_Lab4_top file
entity LogicalStep_Lab4_top is port (
    clk_in_50      : in std_logic;           -- 1 bit 50 MHz FPGA Clock input
    rst_n          : in std_logic;           -- 1 bit reset input (active low)
    pb_n           : in std_logic_vector(3 downto 0); -- 4 bit push-button inputs (active low)
    sw             : in std_logic_vector(7 downto 0); -- 8 bit switch inputs
    leds           : out std_logic_vector(7 downto 0); -- 8 bit output 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_clk_en1     : out std_logic;
    blink_sig1     : out std_logic;
    EW_a           : out std_logic;
    EW_g           : out std_logic;
    EW_d           : out std_logic;
    NS_a           : out std_logic;
    NS_g           : out std_logic;
    NS_d           : 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 digit selectors
    seg7_char2     : out std_logic;                   -- seg7 digit selectors
);
END LogicalStep_Lab4_top;

ARCHITECTURE simplecircuit OF LogicalStep_Lab4_top IS
    component segment7_mux port (
        clk      : in std_logic := '0';           -- digit 1 (left) = EW, digit 2 (right) = NS
        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
        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
        DOUT     : out std_logic_vector(6 downto 0);
        DIG2     : out std_logic;
        DIG1     : out std_logic
    );
    end component;

    component clock_generator port (
        sim_mode : in boolean;
        reset    : in std_logic;
        clk_in   : in std_logic;
        sm_clk_en : out std_logic;
        blink    : out std_logic
    );
    end component;

    component pb_filters port (
        clk_in : 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;
```

```

80
81 component pb_inverters port (
82
83     rst_n      : in std_logic;           -- 1 bit input from reset button before inverting (active low)
84     rst        : out std_logic;          -- 1 bit output from reset input after inverting (active high)
85     pb_n_filtered : in std_logic_vector(3 downto 0); -- 4 bit inputs from push buttons before inverting (active low)
86     pb         : out std_logic_vector(3 downto 0); -- 4 bit outputs from push buttons after inverting (active high)
87 );
88
89 end component;
90
91 component synchronizer port(
92
93     clk        : in std_logic;           -- 1 bit clock signal input
94     reset      : in std_logic;          -- 1 bit reset button input
95     din        : in std_logic;          -- 1 bit input from pb(0), pb(1), or reset button
96     dout       : out std_logic          -- 1 bit synchronizer output (to be used in holding register as an input)
97 );
98
99 end component;
100
101 component holding_register port (
102
103     clk        : in std_logic;           -- clock signal input
104     reset      : in std_logic;          -- reset button input
105     register_clr : in std_logic;         -- crossing signal clear input
106     din        : in std_logic;          -- output from synchronizer used as an input here
107     dout       : out std_logic          -- output that is held for the state machine
108 );
109
110 end component;
111
112 component State_Machine_Moore port (
113
114     -- Inputs --
115
116     clk_50      : in std_logic;         -- 1 bit clock input
117     clk_input    : in std_logic;         -- 1 bit clock input
118     reset       : in std_logic;          -- 1 bit reset input
119     NS_cross_req : in std_logic;         -- 1 bit NS pedestrian request (outputs from holding register) input
120     EW_cross_req : in std_logic;         -- 1 bit EW pedestrian request (outputs from holding register) input
121     blink       : in std_logic;         -- 1 bit blinking signal (clock generator's output) input
122
123     -- Outputs --
124
125     outNSred     : out std_logic;        -- 1 bit NS direction lights [red(seg a)] input
126     outNSamber   : out std_logic;        -- 1 bit NS direction lights [red(seg a)] input
127     outNSgreen   : out std_logic;        -- 1 bit NS direction lights [red(seg a)] input
128     outEWred     : out std_logic;        -- 1 bit NS direction lights [red(seg a)] input
129     outEWamber   : out std_logic;        -- 1 bit NS direction lights [red(seg a)] input
130     outEWgreen   : out std_logic;        -- 1 bit NS direction lights [red(seg a)] input
131     NS_clear_crossing : out std_logic;    -- 1 bit EW pedestrian request (outputs from holding register) input
132     EW_clear_crossing : out std_logic;    -- 1 bit EW pedestrian request (outputs from holding register) input
133     NS_cross_display : out std_logic;    -- 1 bit EW pedestrian request (outputs from holding register) input
134     EW_cross_display : out std_logic;    -- 1 bit EW pedestrian request (outputs from holding register) input
135
136     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
137 );
138
139 end component;
140
141
142 -----
143
144 CONSTANT sim_mode      : boolean := FALSE; -- set to FALSE for LogicalStep board downloads and set to TRUE for SIMULATIONS
145
146 SIGNAL rst, rst_n_filtered, synch_rst : std_logic;
147 SIGNAL sm_clklen, blink_sig          : std_logic;
148 SIGNAL pb_n_filtered, pb             : std_logic_vector(3 downto 0);
149
150 signal NSout : std_logic;           -- 1 bit NS direction synchronizer output & holding register input (signal)
151 signal Ewout : std_logic;           -- 1 bit EW direction synchronizer output & holding register input (signal)
152
153 -- signal NS_CROSSING : std_logic;
154 -- signal EW_CROSSING : std_logic;
155
156 signal NSred, NSamber, NSgreen, EWred, EWamber, EWgreen : std_logic; -- 1 bit NS and EW direction light signals
157
158 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)
159 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)
160 signal NS_HR_pbreq   : std_logic; -- 1 bit holding register output signal to state machine input
161 signal EW_HR_pbreq   : std_logic; -- 1 bit holding register output signal to state machine input
162
163 signal NSclear, Ewclear : std_logic; -- 1 bit holding register clear signals to clear crossing request
164
165
166 BEGIN
167
168     NS_out_concat <= NSamber & "00" & NSgreen & "00" & NSred; -- concatenation for NS direction display signal [segments G,F,E,D,C,B,A] (digit 2)
169     EW_out_concat <= EWamber & "00" & EWgreen & "00" & EWred; -- concatenation for EW direction display signal [segments G,F,E,D,C,B,A] (digit 1)
170
171 -- sm_clklen1 <= sm_clklen;
172 -- blink_sig1 <= blink_sig;
173 -- EW_a <= EWred;
174 -- EW_g <= EWamber;
175 -- EW_d <= EWgreen;
176 -- NS_a <= NSred;
177 -- NS_g <= NSamber;
178 -- NS_d <= NSgreen;
179
180 leds(1) <= NS_HR_pbreq; -- NS pedestrian crossing request displayed on leds(1)
181 leds(3) <= EW_HR_pbreq; -- EW pedestrian crossing request displayed on leds(3)
182
183 -----
184
185 INST0: pb_filters port map (clk_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
186 INST1: pb_inverters port map (rst_n_filtered, rst, pb_n_filtered, pb); -- converts push button inputs from active low to active high
187 INST2: synchronizer port map (clk_50, synch_rst, pb(0), NSout); -- synchronizer for NS direction also reset by synch_rst
188 INST3: synchronizer port map (clk_50, synch_rst, pb(1), Ewout); -- synchronizer for EW direction
189 INST4: synchronizer port map (clk_50, synch_rst, rst, synch_rst); -- synchronizer for reset button
190 INST5: clock_generator port map (sim_mode, synch_rst, clk_50, sm_clklen, blink_sig); -- state machine's clock and blink signals are created
191 INST6: holding_register port map (clk_50, synch_rst, NSclear, NSout, NS_HR_pbreq); -- holding register for NS direction
192 INST7: holding_register port map (clk_50, synch_rst, Ewclear, Ewout, EW_HR_pbreq); -- holding register for EW direction
193 INST8: State_Machine_Moore port map (clk_50, sm_clklen, synch_rst, NS_HR_pbreq, EW_HR_pbreq, blink_sig, NSred, NSamber, NSgreen, EWred, EWamber, EWgreen, NSclear, Ewclear, leds(0), leds(2), leds(7 downto 4)); -- Moore State Machine
194 INST9: segment7_mux port map (clk_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
195
196 END SimpleCircuit;
197
198
199

```

Sub Files:

- synchronizer.vhd

```
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
10 entity synchronizer is port (
11     clk      : in std_logic;          -- 1 bit clock signal input
12     reset    : in std_logic;          -- 1 bit reset button input
13     din      : in std_logic;          -- 1 bit input from pb(0), pb(1), or reset button
14     dout     : out std_logic          -- 1 bit synchronizer output (to be used in holding register as an input)
15 );
16
17 end synchronizer;
18
19
20
21 architecture circuit of synchronizer is
22     signal sreg : std_logic_vector(1 downto 0); -- 2 bit temporary signal (as there are 2 flip-flops)
23                                                     -- sreg(0) = signal that acts as the output of D-flip-flop 1 and input of D-flip-flop 2
24                                                     -- sreg(1) = signal that acts as the output D-flip-flop 2
25
26
27
28 BEGIN
29
30     PROCESS (clk) is
31         -- process updates with a clock
32     begin
33
34         if(rising_edge(clk)) then
35             -- when the clock signal is on its rising edge
36             sreg(0) <= (not reset and din); -- when the reset button is not pressed, sreg signal's 0th bit is set to din (input signal)
37             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
38         end if;
39     end process;
40
41     dout <= sreg(1); -- value of dout (output of the synchronizer) is set to the value of the sreg signal's 1st bit
42
43 end circuit;
```

- PB_inverters.vhd

```
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
10 entity PB_inverters is port (
11     -- converts push button inputs from active low to active high
12     rst_n      : in std_logic;          -- 1 bit input from reset button before inverting (active low)
13     rst        : out std_logic;          -- 1 bit input from reset input after inverting (active high)
14     pb_n_filtered : in std_logic_vector(3 downto 0); -- 4 bit inputs from push buttons before inverting (active low)
15     pb         : out std_logic_vector(3 downto 0); -- 4 bit inputs from push buttons after inverting (active high)
16 );
17
18 end PB_inverters;
19
20
21 architecture ckt of PB_inverters is
22     -- convert active low inputs to active high using the NOT operator
23
24 BEGIN
25
26     rst <= not(rst_n); -- assigning the inverted input of the rst_n to rst
27     pb <= not(pb_n_filtered); -- assigning the inverted input of the rst_n to rst
28
29 end ckt;
```

● holding_register.vhd

```

1  -- Lab Session 202
2  -- Team #8
3  -- Meghana Reddy Vusirika (21053635) & Okusanya Oluwatise (21054732)
4
5
6
7  library ieee;
8  use ieee.std_logic_1164.all;
9
10 entity holding_register is port (
11
12     clk          : in std_logic;          -- 1 bit clock signal input
13     reset        : in std_logic;          -- 1 bit reset button input
14     register_clr  : in std_logic;          -- 1 bit crossing signal clear input
15     din          : in std_logic;          -- 1 bit output from synchronizer used as an input here
16     dout         : out std_logic          -- 1 bit output that is held for the state machine
17 );
18
19 end holding_register;
20
21
22 architecture circuit_of_holding_register is
23     signal sreg : std_logic;
24
25
26
27 BEGIN
28
29 PROCESS (clk) is
30     -- process updates with a clock
31
32     begin
33
34         if(rising_edge(clk)) then
35             -- when the clock signal is on its rising edge
36
37             if (reset = '1') then
38                 -- when the reset button is pressed
39                 sreg <= '0';
40                 -- the value of the sreg signal is set to 0 (holding register is cleared)
41
42             else
43                 sreg <= ((sreg or din) and not (register_clr or reset));
44                 -- value of the combinational logic between sreg, din, and
45                 -- register_clr is updated to the sreg signal
46
47             end if;
48
49         end if;
50
51         dout <= sreg;
52         -- value of dout (output of holding register) is set to the value
53         -- of the sreg signal (this value is used as an input to the state machine)
54
55     end process;
56
57 end;

```

● State_Machine_Moore.vhd

```

1  -- Lab Session 202
2  -- Team #8
3  -- Meghana Reddy Vusirika (21053635) & Okusanya Oluwatise (21054732)
4
5
6
7  library ieee;
8  use ieee.std_logic_1164.all;
9  use ieee.numeric_std.all;
10
11 entity State_Machine_Moore is port (
12
13     -- Inputs --
14
15     clk_50       : in std_logic;          -- 1 bit clock input
16     clk_input    : in std_logic;          -- 1 bit clock input
17     reset        : in std_logic;          -- 1 bit reset input
18     NS_cross_req : in std_logic;          -- 1 bit NS pedestrian request (outputs from holding register) input
19     EW_cross_req : in std_logic;          -- 1 bit EW pedestrian request (outputs from holding register) input
20     blink        : in std_logic;          -- 1 bit blinking signal (clock generator's output) input
21
22     -- Outputs --
23
24     outNSred     : out std_logic;          -- 1 bit NS direction lights [red(seg a)] input
25     outNSamber   : out std_logic;          -- 1 bit NS direction lights [red(seg a)] input
26     outNSgreen   : out std_logic;          -- 1 bit NS direction lights [red(seg a)] input
27     outEWred     : out std_logic;          -- 1 bit NS direction lights [red(seg a)] input
28     outEWamber   : out std_logic;          -- 1 bit NS direction lights [red(seg a)] input
29     outEWgreen   : out std_logic;          -- 1 bit NS direction lights [red(seg a)] input
30     NS_clear_crossing : out std_logic;      -- 1 bit EW pedestrian request (outputs from holding register) input
31     EW_clear_crossing : out std_logic;      -- 1 bit EW pedestrian request (outputs from holding register) input
32     NS_cross_display : out std_logic;       -- 1 bit EW pedestrian request (outputs from holding register) input
33     EW_cross_display : out std_logic;       -- 1 bit EW pedestrian request (outputs from holding register) input
34
35     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
36 );
37
38 end entity;
39

```

```

40 |
41 |
42 | Architecture SM of State_Machine_Moore is
43 |
44 |     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)
45 |
46 |     SIGNAL current_state, next_state : STATE_NAMES; -- signals of type STATE_NAMES
47 |
48 | BEGIN
49 |
50 |     -----
51 |     --State Machine: Moore
52 |     -----
53 |
54 |     -- REGISTER LOGIC PROCESS
55 |
56 |     Register_Section: PROCESS (clk_input) -- this process updates with a clock
57 |     BEGIN
58 |
59 |         IF(rising_edge(clk_50)) THEN -- when the clock signal is on its rising edge
60 |
61 |             IF (reset = '1') THEN -- when the reset button is pressed
62 |                 current_state <= S0; -- the state jumps to state 0
63 |
64 |             ELSIF (reset = '0' and clk_input = '1') THEN -- when reset button is not pressed and the state machine clock enable signal is active
65 |                 current_state <= next_state; -- the state machine moves on to the next state (current state is updated to the next state value)
66 |             END IF;
67 |
68 |         END IF;
69 |
70 |     END PROCESS;
71 |
72 |
73 |
74 |
75 |
76 |
77 |     -- TRANSITION LOGIC PROCESS
78 |
79 |     Transition_Section: PROCESS (NS_cross_req, EW_cross_req, current_state) -- NS_cross_req = NS (pb(0) pushed, pb(1) not)
80 |                                     -- EW_cross_req = EW (pb(1) pushed, pb(0) not)
81 |     BEGIN
82 |
83 |         CASE current_state IS
84 |
85 |             WHEN S0 =>
86 |
87 |                 -- state jumps from 0 to 6 if crossing request is made in NS and not in EW (only pb(0) pushed)
88 |                 -- state goes to next state (1) otherwise
89 |
90 |                 IF(NS_cross_req='0' and EW_cross_req='1') THEN
91 |                     next_state <= S6;
92 |                 ELSE
93 |                     next_state <= S1;
94 |                 END IF;
95 |
96 |             WHEN S1 =>
97 |
98 |                 -- state jumps from 1 to 6 if crossing request is made in NS and not in EW (only pb(0) pushed)
99 |                 -- state goes to next state (2) otherwise
100 |
101 |                 IF(NS_cross_req='0' and EW_cross_req='1') THEN
102 |                     next_state <= S6;
103 |                 ELSE
104 |                     next_state <= S2;
105 |                 END IF;
106 |
107 |             WHEN S2 => -- no button is pushed, so go to next state
108 |                 next_state <= S3;
109 |
110 |             WHEN S3 => -- no button is pushed, so go to next state
111 |                 next_state <= S4;
112 |
113 |             WHEN S4 => -- no button is pushed, so go to next state
114 |                 next_state <= S5;
115 |
116 |             WHEN S5 => -- no button is pushed, so go to next state
117 |                 next_state <= S6;
118 |
119 |             WHEN S6 => -- no button is pushed, so go to next state
120 |                 next_state <= S7;
121 |
122 |             WHEN S7 => -- no button is pushed, so go to next state
123 |                 next_state <= S8;
124 |
125 |             WHEN S8 =>
126 |
127 |                 -- state jumps from 8 to 14 if crossing request is made in EW and not in NS (only pb(1) pushed)
128 |                 -- state goes to next state (9) otherwise
129 |
130 |                 IF(NS_cross_req='1' and EW_cross_req='0') THEN
131 |                     next_state <= S14;
132 |                 ELSE
133 |                     next_state <= S9;
134 |                 END IF;
135 |
136 |
137 |

```

```

138 WHEN S9 =>
139
140 -- state jumps from 9 to 14 if crossing request is made in EW and not in NS (only pb(1) pushed)
141 -- state goes to next state (10) otherwise
142
143 IF (NS_cross_req='1' and EW_cross_req='0') THEN
144     next_state <= S14;
145 ELSE
146     next_state <= S10;
147 END IF;
148
149 WHEN S10 => --no button is pushed, so go to next state
150     next_state <= S11;
151
152 WHEN S11 => --no button is pushed, so go to next state
153     next_state <= S12;
154
155 WHEN S12 => --no button is pushed, so go to next state
156     next_state <= S13;
157
158 WHEN S13 => --no button is pushed, so go to next state
159     next_state <= S14;
160
161 WHEN S14 => --no button is pushed, so go to next state
162     next_state <= S15;
163
164 WHEN S15 => --no button is pushed, so go to next state (goes back to state 0)
165     next_state <= S0;
166
167 END CASE;
168
169 END PROCESS;
170
171
172 -- DECODER SECTION PROCESS
173
174
175 Decoder_Section: PROCESS (current_state, blk)
176
177 -- outNSred    = NS direction red light (seg a)
178 -- outNSamber  = NS direction amber light (seg g)
179 -- outNSgreen  = NS direction green light (seg d)
180 -- outEWred    = EW direction red light (seg d)
181 -- outEWamber  = EW direction amber light (seg d)
182 -- outEWgreen  = EW direction green light (seg d)
183
184 BEGIN
185
186 CASE current_state IS
187
188 WHEN S0 => -- NS blinking green light; EW red light
189     outNSred <= '0';
190     outNSamber <= '0';
191     outNSgreen <= blk;
192     outEWred <= '1';
193     outEWamber <= '0';
194     outEWgreen <= '0';
195     state_num <= "0000";
196     NS_clear_crossing <= '0';
197     EW_clear_crossing <= '0';
198     NS_cross_display <= '0';
199     EW_cross_display <= '0';
200
201 WHEN S1 => -- NS blinking green light; EW red light
202     outNSred <= '0';
203     outNSamber <= '0';
204     outNSgreen <= blk;
205     outEWred <= '1';
206     outEWamber <= '0';
207     outEWgreen <= '0';
208     state_num <= "0001";
209     NS_clear_crossing <= '0';
210     EW_clear_crossing <= '0';
211     NS_cross_display <= '0';
212     EW_cross_display <= '0';
213
214 WHEN S2 => -- NS green light; EW red light
215     outNSred <= '0';
216     outNSamber <= '0';
217     outNSgreen <= '1';
218     outEWred <= '1';
219     outEWamber <= '0';
220     outEWgreen <= '0';
221     state_num <= "0010";
222     NS_clear_crossing <= '0';
223     EW_clear_crossing <= '0';
224     NS_cross_display <= '1'; -- NS crossing display: leds(0) turns on
225     EW_cross_display <= '0';
226

```



```

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 <= blk;
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 <= blk;
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

```



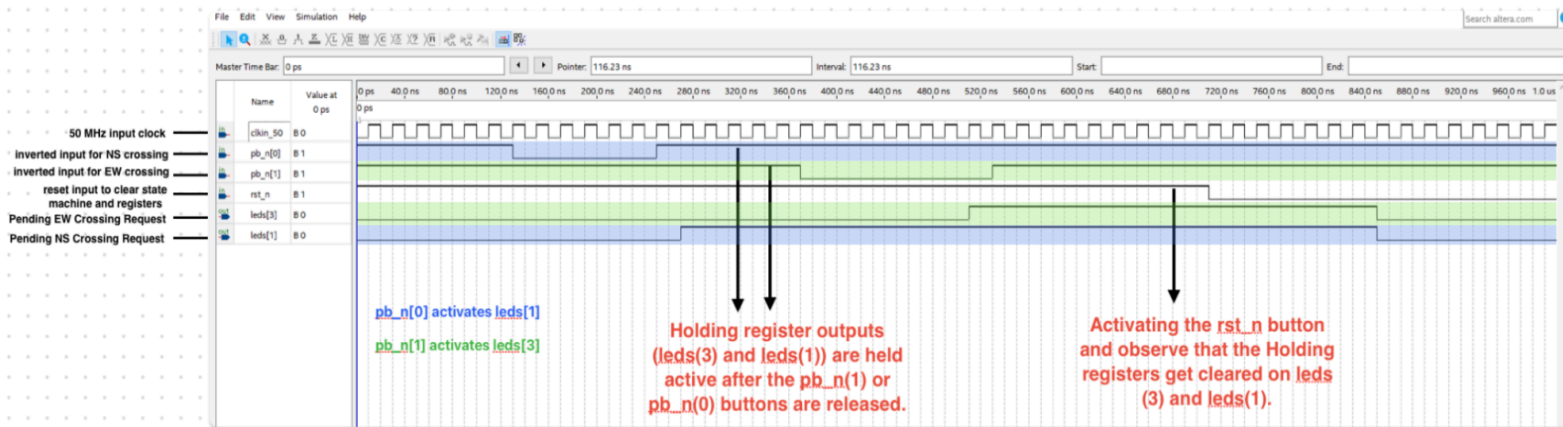
```

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

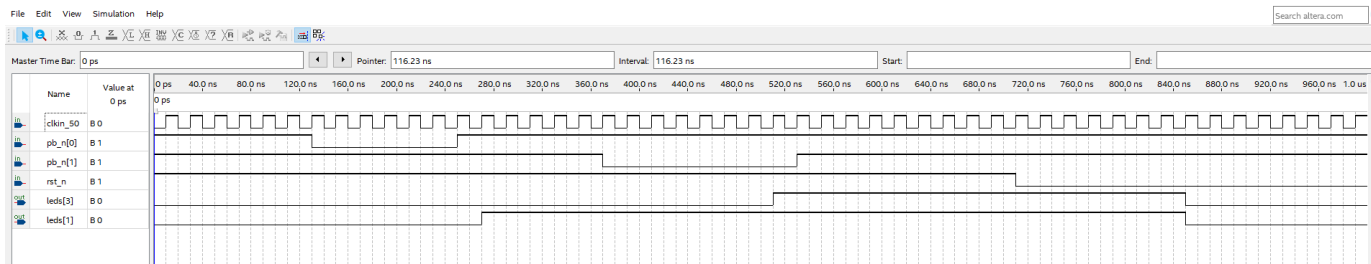
```

Simulations:

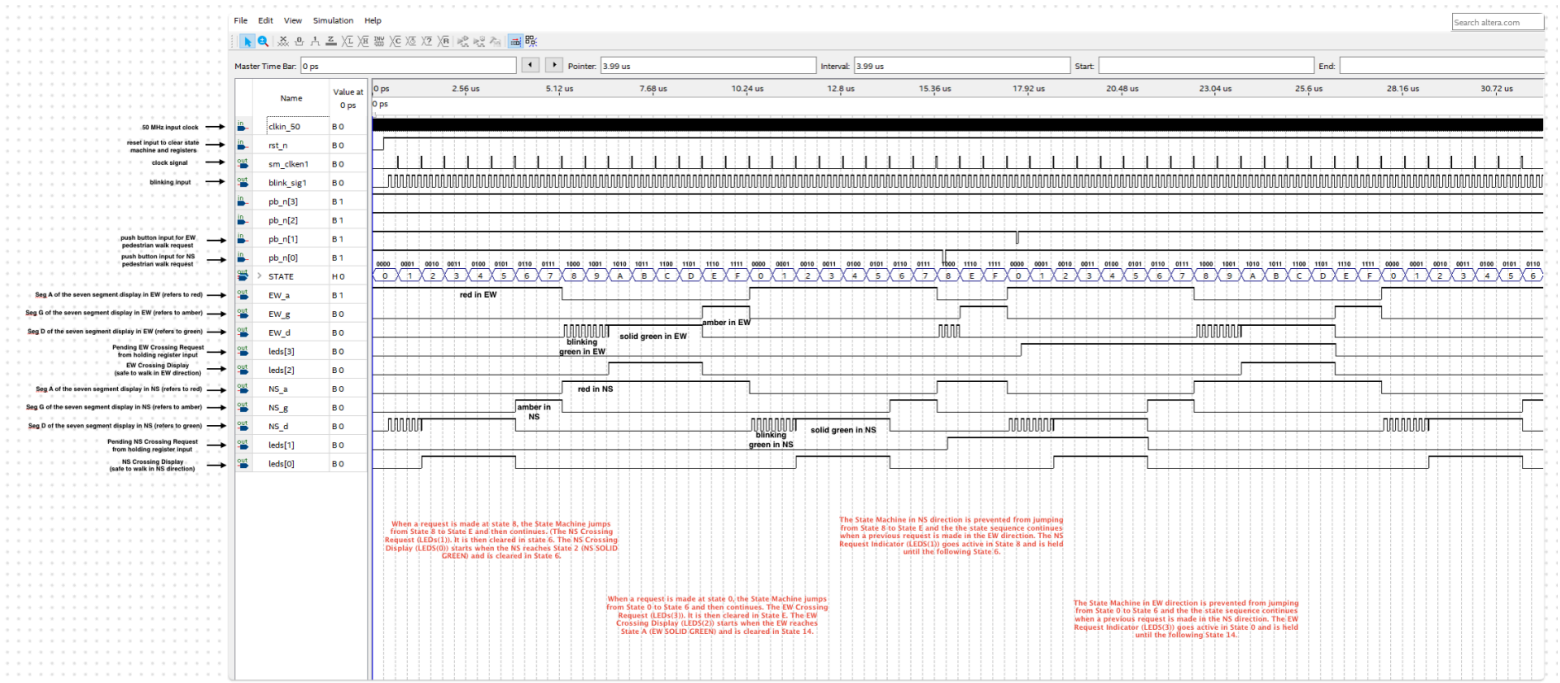
- Synchronizer and Holding Register Simulation (from Part B) (Annotated)



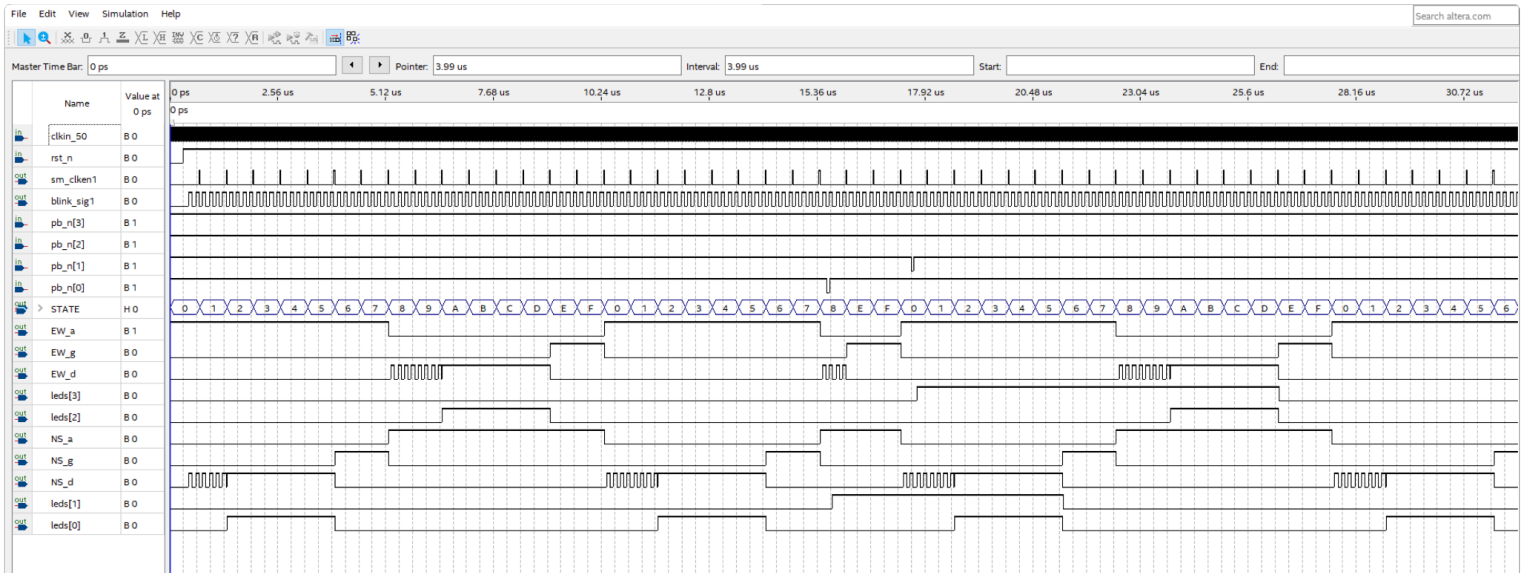
(Original)



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

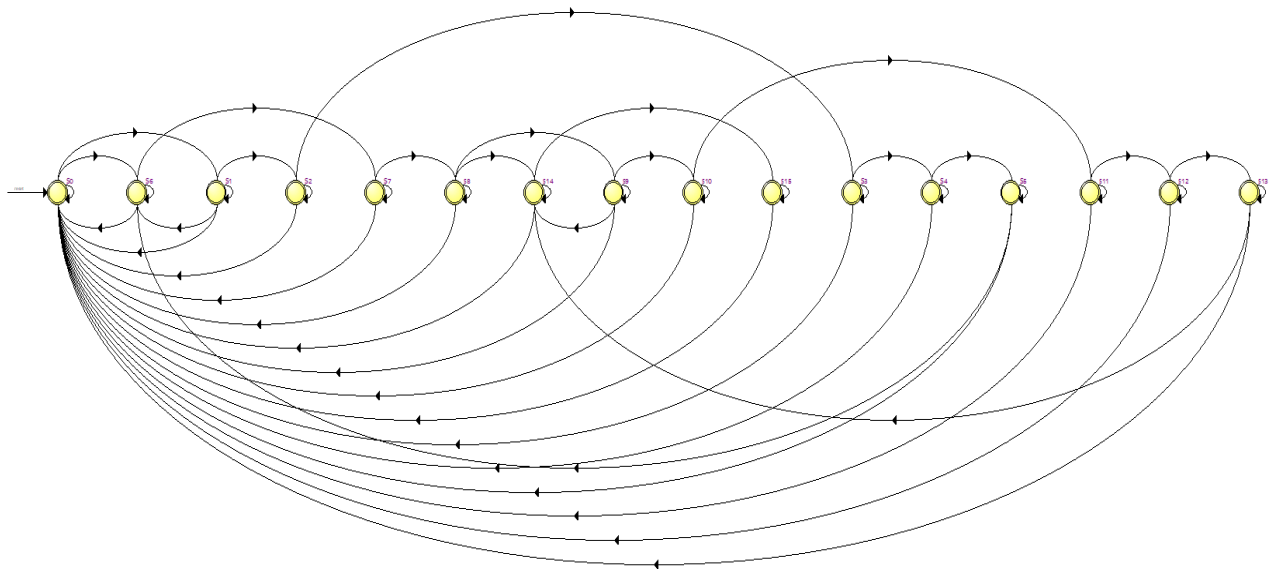


(Original)



State Machine:

- State_Machine Diagram



● State_Machine Table

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

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)