Lab 4 Report

ECE 124

Group 3 - Session 203

Yashashwin Karthikeyan

LS203_T03_Lab4_REPORT_Yashashwin_Karthikeyan

Top level file: LogicalStep_Lab4_top.vhd

```
-- Section 203
-- Group 3: Yashashwin Karthikeyan and Roozbeh Ali
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY LogicalStep_Lab4_top IS PORT (
  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
  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

  ------------------------------------------------------------
  -- Temporary Signals for Waveform Analysis, Must be Commented Out for Board Flashing
  ------------------------------------------------------------

  sm_clken: out std_logic; -- Clock enable from the clock generator
  blink_sig: out std_logic; -- Blinking signal from the clock generator
  NS_green: out std_logic; -- Green color for NS
  NS_amber: out std_logic; -- Amber color for NS
  NS_red: out std_logic; -- Red color for NS
  EW_green: out std_logic; -- Green color for EW
  EW_amber: out std_logic; -- Amber color for EW
  EW_red: out std_logic  -- Red color for EW
); END LogicalStep_Lab4_top;

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

ARCHITECTURE SimpleCircuit OF LogicalStep_Lab4_top IS
  component segment7_mux port (
    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 (
    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 (
```

```
53      clkin: in std_logic;
54      rst_n: in std_logic;
55      rst_n_filtered: out std_logic;
56      pb_n: in  std_logic_vector (3 downto 0);
57      pb_n_filtered: out  std_logic_vector(3 downto 0)
58    ); end component;
59
60    component pb_inverters port (
61      rst_n: in  std_logic;
62      rst: out    std_logic;
63      pb_n_filtered: in  std_logic_vector (3 downto 0);
64      pb: out std_logic_vector(3 downto 0)
65    ); end component;
66
67    component synchronizer port(
68      input: in std_logic;
69      global_clock: in std_logic;
70      reset: in std_logic; -- sync reset
71      output: out std_logic
72    ); end component;
73
74    component holding_register port (
75      clk: in std_logic;
76      reset: in std_logic;
77      register_clr: in std_logic;
78      din: in std_logic;
79      dout: out std_logic
80    ); end component;
81
82    component State_Machine port (
83      clk_input, reset, clk_en, blink_seg: IN std_logic;
84      ns_button, ew_button: IN std_logic;
85      ns_green, ns_amber, ns_red: OUT std_logic;
86      ew_green, ew_amber, ew_red: OUT std_logic;
87      display_state: OUT std_logic_vector(3 downto 0);
88      ns_regclear, ew_regclear: OUT std_logic;
89      ns_crosslight, ew_crosslight: OUT std_logic
90    ); END component;
91
92  ----------------------------------------------------------------------------------------------------
93    -- set to FALSE for LogicalStep board downloads
94    CONSTANT  sim_mode: boolean := TRUE;
95
96    -- Active high reset button after filtering,
97    -- filtered reset button, and the synchronous reset after its input syncher
98    SIGNAL rst, rst_n_filtered, synch_rst: std_logic;
99
100   -- Temporary signal to map to the blink signal output from the clock generator
101   SIGNAL blink: std_logic;
102
103   -- vectors to represent the push buttons on the board after filtering and after inverting
104   SIGNAL pb_n_filtered, pb: std_logic_vector(3 downto 0);
105
106   -- vector to map the output of the synchronous inputs to the holding registers
```

```
107   SIGNAL sync_out: std_logic_vector(1 downto 0);
108
109   SIGNAL ns_green_l: std_logic;  -- The green signal for NS light
110   SIGNAL ns_amber_l: std_logic;  -- The amber signal for NS light
111   SIGNAL ns_red_l: std_logic;    -- The Red signal for NS light
112   SIGNAL ew_green_l: std_logic;  -- The Green signal for EW light
113   SIGNAL ew_amber_l: std_logic;  -- The Amber signal for EW light
114   SIGNAL ew_red_l: std_logic;    -- The RED signal for EW light
115   SIGNAL ew_register: std_logic     -- The register clear for EW holding input
116   SIGNAL ns_register: std_logic     -- The register clear for NS holding input
117   SIGNAL ew_button: std_logic;   -- The button indicator for the EW pedestrian button (LED)
118   SIGNAL ns_button: std_logic;   -- The button indicator for the NS pedestrian button (LED)
119   SIGNAL clk_enable: std_logic;  -- Clk_enable for the register to be taken from the clock generator
120
121 BEGIN
122 ----------------------------------------------------------------------------------------------------
123 INST0: pb_filters port map (
124   clkin_50,
125   rst_n,
126   rst_n_filtered,
127   pb_n,
128   pb_n_filtered);
129
130 INST1: pb_inverters port map (
131   rst_n_filtered,
132   rst,
133   pb_n_filtered,
134   pb);
135
136 INST2: synchronizer port map (
137   rst,
138   clkin_50,
139   synch_rst,
140   synch_rst);  -- Registers to sync inputs with each other
141
142 INST3: synchronizer port map (
143   pb(1),
144   clkin_50,
145   synch_rst,
146   sync_out(1));
147
148 INST4: synchronizer port map (
149   pb(0),
150   clkin_50,
151   synch_rst,
152   sync_out(0));
153
154 INST5: clock_generator  port map (
155   sim_mode,
156   synch_rst,
157   clkin_50,
158   clk_enable,
159   blink); -- Generates the enabling signal for the state machine and the blinking signal
160
```

```
161  INST6: holding_register port map (
162    clkin_50,
163    synch_rst,
164    ew_register,
165    sync_out(1),
166    ew_button); -- EW pedestrian button input holder
167
168  INST7: holding_register port map (
169    clkin_50,
170    synch_rst,
171    ns_register,
172    sync_out(0),
173    ns_button); -- NS pedestrian button input holder
174
175  INST8: STATE_MACHINE port map (
176    clkin_50,
177    synch_rst,
178    clk_enable,
179    blink,
180    ns_button,
181    ew_button,
182    ns_green_l,
183    ns_amber_l,
184    ns_red_l,
185    ew_green_l,
186    ew_amber_l,
187    ew_red_l,
188    leds(7 downto 4),
189    ns_register,
190    ew_register,
191    leds(0),
192    leds(2));
193
194  -- output signals for the traffic lights are concatenated
195  INST9: segment7_mux port map(
196    clkin_50,
197    ns_amber_l & "00" & ns_green_l & "00" & ns_red_l,
198    ew_amber_l & "00" & ew_green_l & "00" & ew_red_l,
199    seg7_data,
200    seg7_char2,
201    seg7_char1);
202
203  leds(3) <= ew_button;
204  leds(1) <= ns_button;
205
206  NS_green <= ns_green_l;
207  NS_amber <= ns_amber_l;
208  NS_red <= ns_red_l;
209  EW_green <= ew_green_l;
210  EW_amber <= ew_amber_l;
211  EW_red <= ew_red_l;
212  blink_sig <= blink;
213  sm_clken <= clk_enable;
214
```

```
215 END SimpleCircuit;
```

Subordinate file: PB_inverters.vhd

```
1  -- Section 203
2  -- Group 3: Yashashwin Karthikeyan and Roozbeh Ali
3  library ieee;
4  use ieee.std_logic_1164.all;
5
6  -- maps the pb switches from active low to active high
7  entity PB_inverters is port (
8    rst_n: in std_logic; -- Active low reset
9    rst: out std_logic; -- Active high
10   pb_n_filtered: in  std_logic_vector (3 downto 0); -- Button inputs after filtering, active low
11   pb: out   std_logic_vector(3 downto 0)    -- Button inputs, now active high
12 ); end PB_inverters;
13
14 architecture inv of PB_inverters is
15 begin
16   rst <= NOT(rst_n);
17   pb <= NOT(pb_n_filtered);
18 end inv;
```

Subordinate file: State_Machine.vhd

```
1  -- Section 203
2  -- Group 3: Yashashwin Karthikeyan and Roozbeh Ali
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  Entity State_Machine IS Port (
8    -- Global clk, synch reset to reset whole board, enable from clk generator,
9   clk_input, reset, clk_en, blink_seg: IN std_logic;
10  -- and blink_seg produced by clk generator to be used in the blinking green
11  ns_button, ew_button: IN std_logic; -- Input from the holding registers for the pedestrian buttons
12  ns_green, ns_amber, ns_red: OUT std_logic; -- The color outputs for NS
13  ew_green, ew_amber, ew_red: OUT std_logic; -- The color outputs for EW
14  display_state: OUT std_logic_vector(3 downto 0); -- State to be displayed on leds(7 downto 4)
15  ns_regclear, ew_regclear: OUT std_logic;   -- Pedestrian button clears
16  ns_crosslight, ew_crosslight: OUT std_logic    -- Output signal for pedetestrian crossing
17 ); END ENTITY;
18
19 architecture sm of State_Machine is
20 TYPE STATE_NAMES IS (S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15); -- 16 States
21
22 SIGNAL current_state, next_state: STATE_NAMES;
23
24 begin
25 -- Register Logic
```

```
Register_Section: PROCESS(clk_input)
begin
  if (rising_edge(clk_input)) then
    if (reset = '1') then       -- used to reset state to S0 if reset is pressed
        current_state <= S0;
    elsif (clk_en = '1') then   -- state progresses to next_state only if clk_en = 1
        current_state <= next_state;
    end if;
  end if;
end process;

-- Transition Logic: used to compute the next state of the state machine
Transition_Section: PROCESS(current_state, ew_button, ns_button)
begin
    case current_state is
        when S0 =>
            -- Skip red light duration (NS) if requested
            if (ew_button = '1' and ns_button = '0') then
                next_state <= S6;
            else
                next_state <= S1;
            end if;
        when S1 =>
            -- Skip red light duration (NS) if requested
            if (ew_button = '1' and ns_button = '0') then
                next_state <= S6;
            else
                next_state <= S2;
            end if;
        when S2 =>
            next_state <= S3;
        when 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 =>
            -- Skip red light duration (EW) if requested
            if (ns_button = '1' and ew_button = '0') then
                next_state <= S14;
            else
                next_state <= S9;
            end if;
        when S9 =>
            -- Skip red light duration (EW) if requested
            if (ns_button = '1' and ew_button = '0') 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 =>
            next_state <= S0;
    end case;
end process;

-- Decoder section: used to compute the output of the state machine.
Decoder_Section: PROCESS (current_state)
begin
        -- Ensure the registers do not clear on any case where not specified,
        ew_regclear <= '0';
        ns_regclear <= '0';
    case current_state is
        WHEN S0 =>
            display_state <= "0000";
        WHEN S1 =>
            display_state <= "0001";
        WHEN S2 =>
            display_state <= "0010";
        WHEN S3 =>
            display_state <= "0011";
        WHEN S4 =>
            display_state <= "0100";
        WHEN S5 =>
            display_state <= "0101";
        WHEN S6 =>
            -- NS button holding register is cleared at state 6 as per project specifications
            ns_regclear <= '1';
            display_state <= "0110";
        WHEN S7 =>
            display_state <= "0111";
        WHEN S8 =>
            display_state <= "1000";
        WHEN S9 =>
            display_state <= "1001";
        WHEN S10 =>
            display_state <= "1010";
        WHEN S11 =>
            display_state <= "1011";
        WHEN S12 =>
            display_state <= "1100";
        WHEN S13 =>
            display_state <= "1101";
        WHEN S14 =>
```

```
            -- EW button holding register is cleared at state 14 as per project specifications
            ew_regclear <= '1';
            display_state <= "1110";
        WHEN S15 =>
            display_state <= "1111";
    end case;

    case current_state is
        WHEN S0 | S1 =>
            -- Blinking Green NS, RED EW
            ns_green <= blink_seg;
            ns_amber <= '0';
            ns_red <= '0';
            ew_green <='0';
            ew_amber <='0';
            ew_red <='1';
            ns_crosslight <='0';
            ew_crosslight <='0';

        WHEN S2 | S3 | S4 | S5 =>
            -- GREEN NS, RED EW
            ns_green <= '1';
            ns_amber <= '0';
            ns_red <= '0';
            ew_green <= '0';
            ew_amber <= '0';
            ew_red <= '1';
            ns_crosslight <= '1';
            ew_crosslight <= '0';

        WHEN S6 | S7 =>
            -- AMBER NS, RED EW
            ns_green <= '0';
            ns_amber <= '1';
            ns_red <= '0';
            ew_green <= '0';
            ew_amber <= '0';
            ew_red <= '1';
            ns_crosslight <= '0';
            ew_crosslight <= '0';

        WHEN S8 | S9 =>
            -- RED NS, BLINKING GREEN EW
            ns_green <= '0';
            ns_amber <= '0';
            ns_red <= '1';
            ew_green <= blink_seg;
            ew_amber <= '0';
            ew_red <=   '0';
            ns_crosslight <= '0';
            ew_crosslight <= '0';

        WHEN S10 | S11 | S12 | S13 =>
            -- RED NS, GREEN EW
```

```
188             ns_green <= '0';
189             ns_amber <= '0';
190             ns_red <= '1';
191             ew_green <= '1';
192             ew_amber <= '0';
193             ew_red <='0';
194             ns_crosslight <= '0';
195             ew_crosslight <= '1';
196 
197         WHEN S14 | S15 =>
198             -- RED NS, AMBER EW
199             ns_green <= '0';
200             ns_amber <= '0';
201             ns_red <= '1';
202             ew_green <= '0';
203             ew_amber <= '1';
204             ew_red <= '0';
205             ns_crosslight <= '0';
206             ew_crosslight <= '0';
207     end case;
208 end process;
209 end architecture sm;
```

Subordinate file: holding_register.vhd

```
1  -- Section 203
2  -- Group 3: Yashashwin Karthikeyan and Roozbeh Ali
3  library ieee;
4  use ieee.std_logic_1164.all;
5 
6  -- Register to hold the input after the user presses one of the pedestrian buttons
7  entity holding_register is port (
8    clk: in std_logic;
9    reset: in std_logic;
10   register_clr: in std_logic;
11   din: in std_logic;
12   dout: out std_logic
13 ); end holding_register;
14 
15 architecture circuit of holding_register is
16     Signal sreg             : std_logic; -- value of register
17 BEGIN
18     process(clk) is
19     begin
20         if (rising_edge(clk)) then
21             -- Reset on register_clear or reset
22             sreg <= (not(register_clr or reset)) and (din or sreg);
23         end if;
24     -- no else block, hence latch is inferred
25 
26     end process;
27 
```

```
28    dout <= sreg; -- sreg outputs to output of the block
29  end circuit;
```

Subordinate file: synchronizer.vhd

```
1   -- Section 203, Group 3
2   -- Yashashwin Karthikeyan
3   -- Roozbeh Ali
4   library ieee;
5   use ieee.std_logic_1164.all;
6   use ieee.numeric_std.all;
7
8   -- Synchronizes an input based on the clock
9   Entity synchronizer IS Port
10  (
11      input : in std_logic;
12      global_clock : in std_logic;
13      reset   : in std_logic; -- Note this is a syncronized reset
14      output : out std_logic
15  );
16  end entity;
17
18  architecture main of synchronizer is
19  signal reg : std_logic; -- Register storage signal
20
21  begin
22
23      process(global_clock) is
24      begin
25          if (rising_edge(global_clock)) then
26              if (reset = '0') then -- If user doesn't want a reset, shift-register
27                  reg <= input;
28                  output <= reg;
29              else
30                  reg <= '0'; -- Reset register values
31                  output <= '0';
32              end if;
33          end if;
34      end process;
35  end main;
```

Images and Annotations

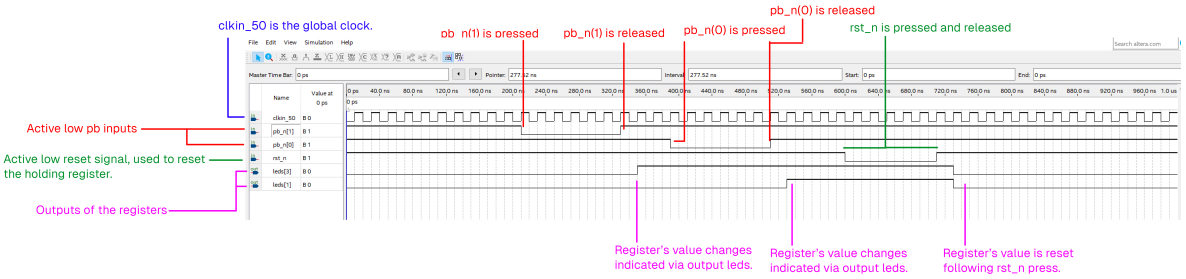


Figure 1: Register Waveform

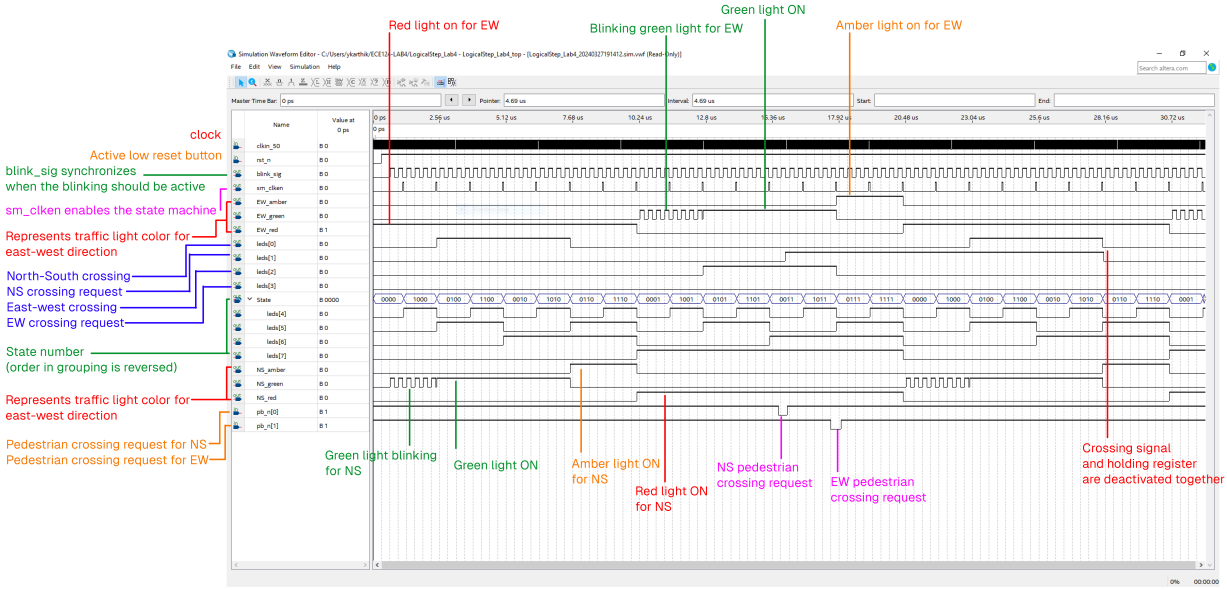


Figure 2: TLC Simulation Waveform

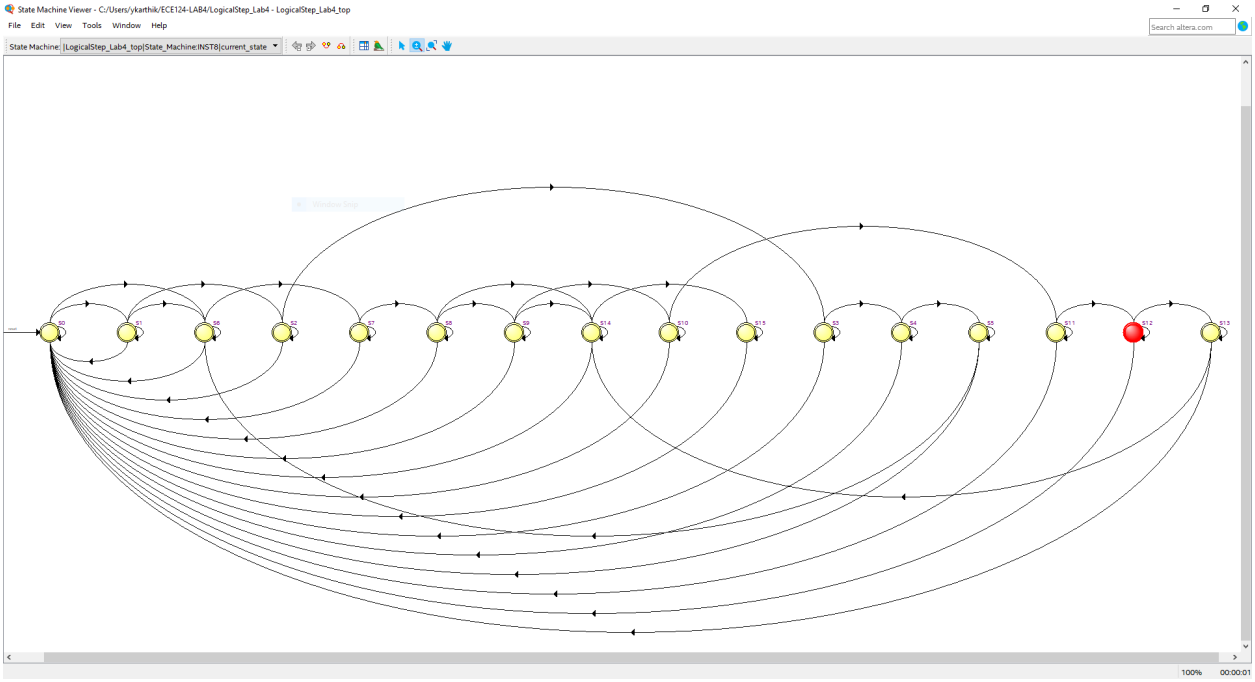
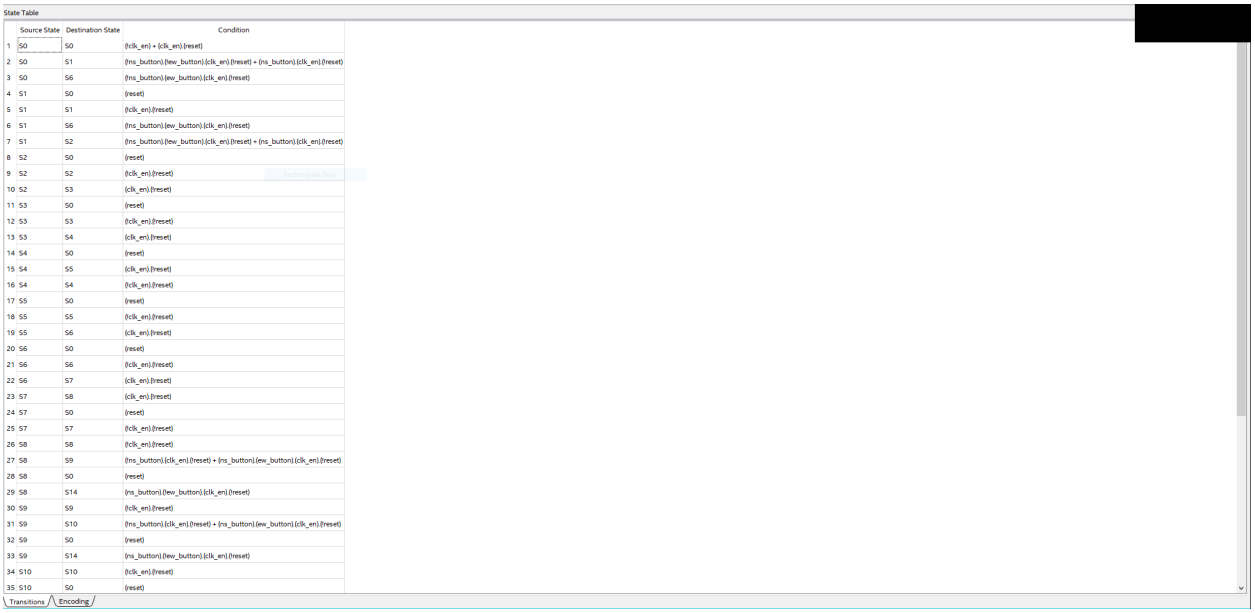


Figure 3: State Diagram

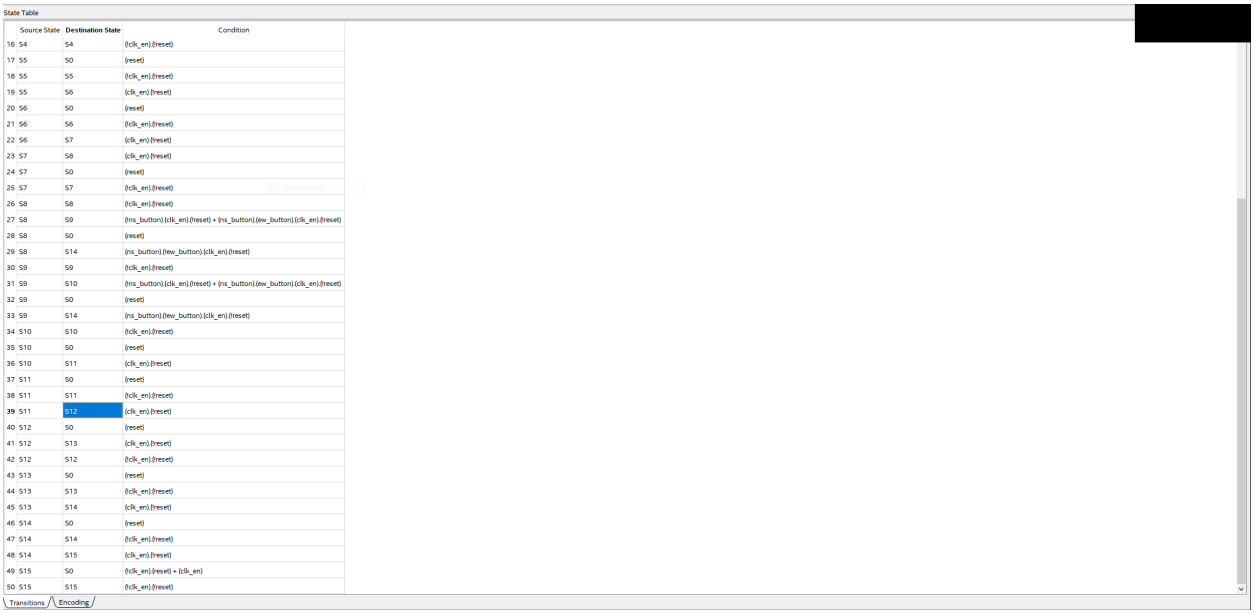


State Table

	Source State	Destination State	Condition
1	S0	S0	(!clk_en) + (clk_en).(reset)
2	S0	S1	(!ns_button).(!ew_button).(clk_en).(!reset) + (ns_button).(clk_en).(!reset)
3	S0	S6	(!ns_button).(ew_button).(clk_en).(!reset)
4	S1	S0	(reset)
5	S1	S1	(!clk_en).(!reset)
6	S1	S6	(!ns_button).(ew_button).(clk_en).(!reset)
7	S1	S2	(!ns_button).(!ew_button).(clk_en).(!reset) + (ns_button).(clk_en).(!reset)
8	S2	S0	(reset)
9	S2	S2	(!clk_en).(!reset)
10	S2	S3	(clk_en).(!reset)
11	S3	S0	(reset)
12	S3	S3	(!clk_en).(!reset)
13	S3	S4	(clk_en).(!reset)
14	S4	S0	(reset)
15	S4	S5	(clk_en).(!reset)
16	S4	S4	(!clk_en).(!reset)
17	S5	S0	(reset)
18	S5	S5	(!clk_en).(!reset)
19	S5	S6	(clk_en).(!reset)
20	S6	S0	(reset)
21	S6	S6	(!clk_en).(!reset)
22	S6	S7	(clk_en).(!reset)
23	S7	S8	(clk_en).(!reset)
24	S7	S0	(reset)
25	S7	S7	(!clk_en).(!reset)
26	S8	S8	(!clk_en).(!reset)
27	S8	S9	(!ns_button).(clk_en).(!reset) + (ns_button).(ew_button).(clk_en).(!reset)
28	S8	S0	(reset)
29	S8	S14	(ns_button).(!ew_button).(clk_en).(!reset)
30	S9	S9	(!clk_en).(!reset)
31	S9	S10	(!ns_button).(clk_en).(!reset) + (ns_button).(ew_button).(clk_en).(!reset)
32	S9	S0	(reset)
33	S9	S14	(ns_button).(!ew_button).(clk_en).(!reset)
34	S10	S10	(!clk_en).(!reset)
35	S10	S0	(reset)

Transitions | Encoding

Figure 4: State Transition Table - 1



State Table

	Source State	Destination State	Condition
16	S4	S4	(!clk_en).(!reset)
17	S5	S0	(reset)
18	S5	S5	(!clk_en).(!reset)
19	S5	S6	(clk_en).(!reset)
20	S6	S0	(reset)
21	S6	S6	(!clk_en).(!reset)
22	S6	S7	(clk_en).(!reset)
23	S7	S8	(clk_en).(!reset)
24	S7	S0	(reset)
25	S7	S7	(!clk_en).(!reset)
26	S8	S8	(!clk_en).(!reset)
27	S8	S9	(!ns_button).(clk_en).(!reset) + (ns_button).(ew_button).(clk_en).(!reset)
28	S8	S0	(reset)
29	S8	S14	(ns_button).(!ew_button).(clk_en).(!reset)
30	S9	S9	(!clk_en).(!reset)
31	S9	S10	(!ns_button).(clk_en).(!reset) + (ns_button).(ew_button).(clk_en).(!reset)
32	S9	S0	(reset)
33	S9	S14	(ns_button).(!ew_button).(clk_en).(!reset)
34	S10	S10	(!clk_en).(!reset)
35	S10	S0	(reset)
36	S10	S11	(clk_en).(!reset)
37	S11	S0	(reset)
38	S11	S11	(!clk_en).(!reset)
39	S11	S12	(clk_en).(!reset)
40	S12	S0	(reset)
41	S12	S13	(clk_en).(!reset)
42	S12	S12	(!clk_en).(!reset)
43	S13	S0	(reset)
44	S13	S13	(!clk_en).(!reset)
45	S13	S14	(clk_en).(!reset)
46	S14	S0	(reset)
47	S14	S14	(!clk_en).(!reset)
48	S14	S15	(clk_en).(!reset)
49	S15	S0	(!clk_en).(!reset) + (clk_en)
50	S15	S15	(!clk_en).(!reset)

Transitions / Encoding

Figure 5: State Transition Table - 2