جامعة نيويورك أبوظبي

NYU ABU DHABI

Advanced Digital Logic

ENGR – UH 2310

Lab 3 Part 2

Group 1

Spring, 2025

This report is entirely our own work, and we have kept a copy for our own records. We are aware of the University's policies on cheating, plagiarism, and the resulting consequences of their breach.

Submitted by:

Name	Participation	Net ID
Damiane Kapanadze	Test Bench, Lab Report	dk4770
Seoyoon Jung	Code, Lab Report	sj4260
Sipan Hovsepian	Code, Lab Report	sh7437

Contents

1 Timing sequences

For detailed timing data, refer to the spreadsheet: Google Sheets Link.

1.1 Normal mode

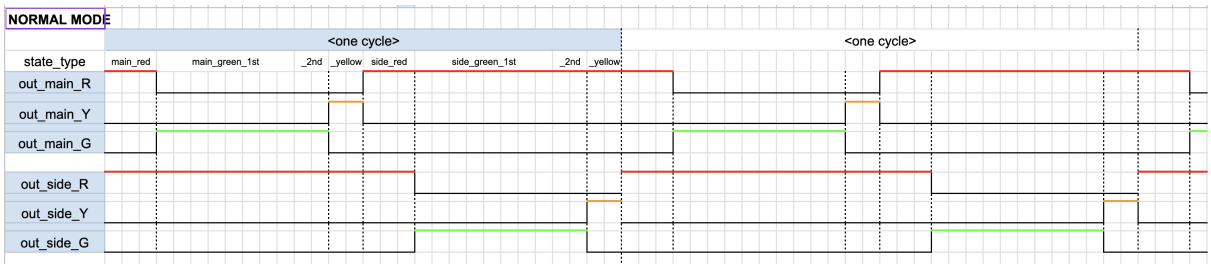


Figure 1: Normal mode

1.2 Normal mode with walk lights

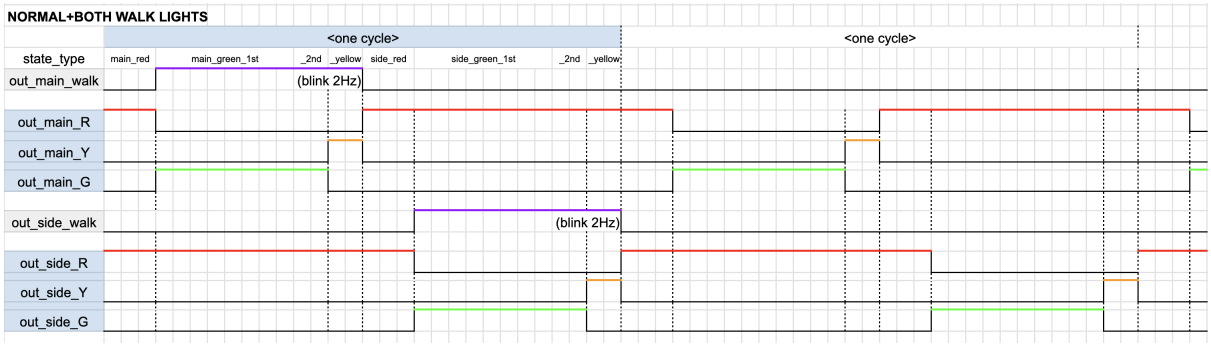


Figure 2: Normal mode with both walk lights triggered

1.3 Extended green phase

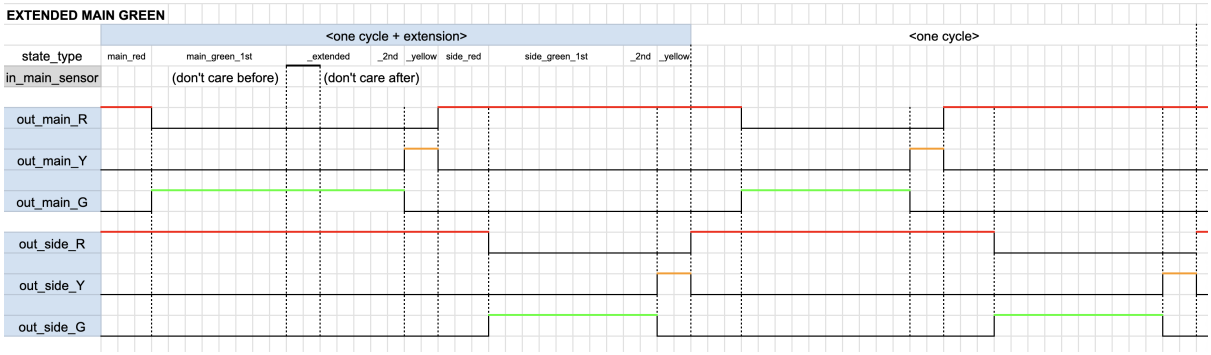


Figure 3: Extended green phase for main street

1.4 Extended green phase with walk lights

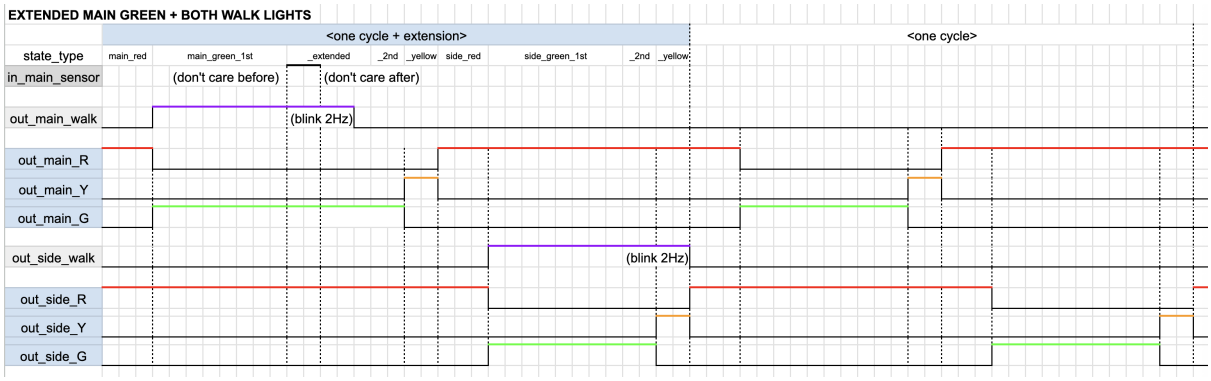
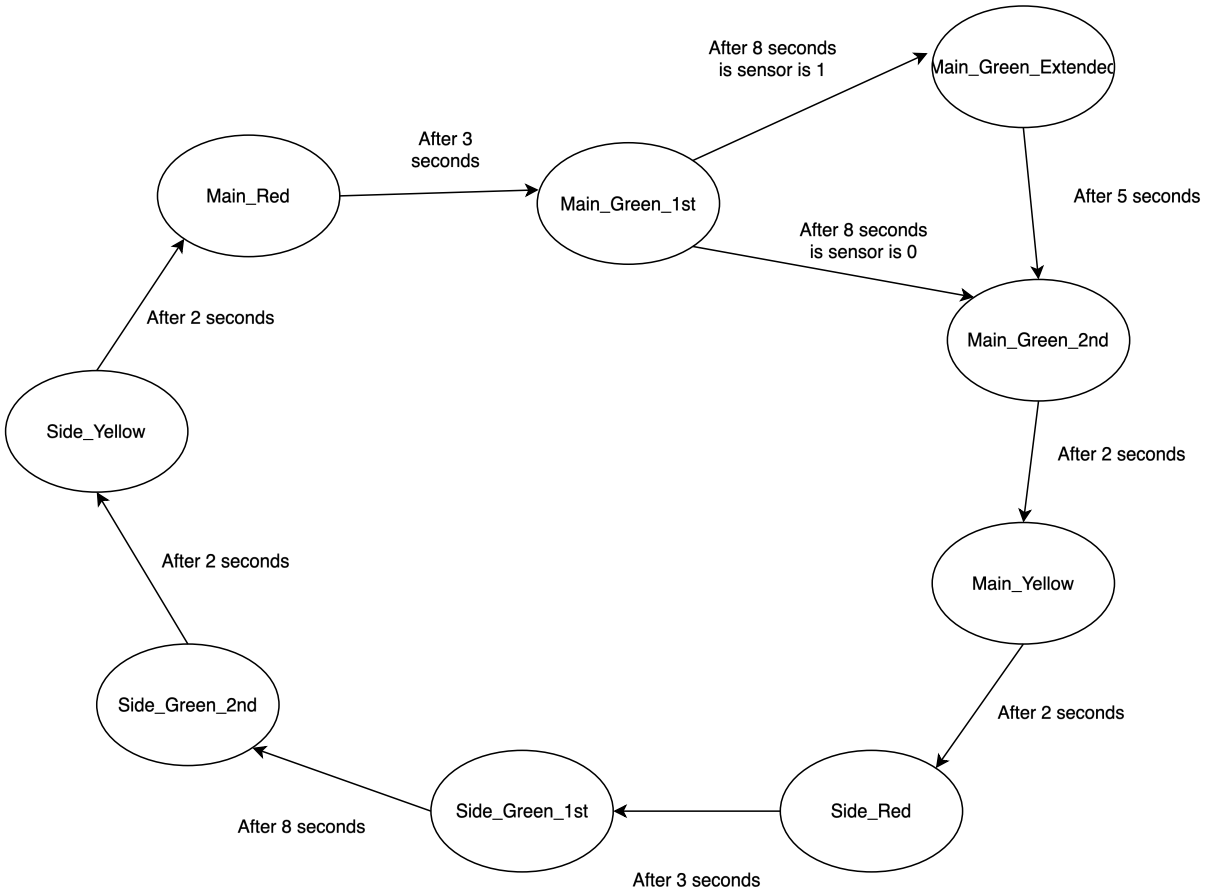


Figure 4: Extended green phase with for main street with both walk lights triggered

2 Devising the FSM

2.1 State diagram



2.2 Board setup

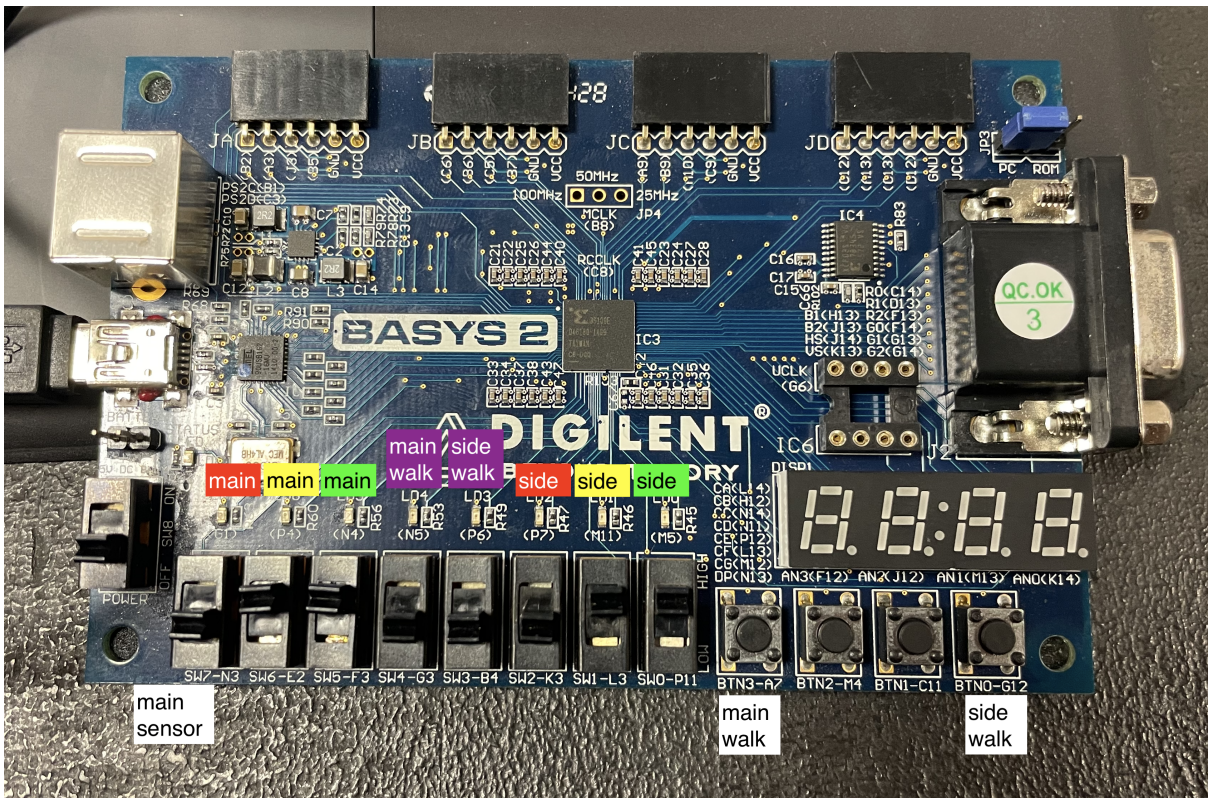


Figure 5: Board setup based on UCF

3 VHDL code of all modules

3.1 FSM module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity FSM is PORT (
    in_clk : IN std_logic;
    in_sensor_main : IN std_logic;
    in_main_walk : IN std_logic;
    in_side_walk : IN std_logic;
    out_main_red : OUT std_logic;
    out_main_yellow : OUT std_logic;
```

```
20      out_main_green : OUT std_logic;
21      out_side_red : OUT std_logic;
22      out_side_yellow : OUT std_logic;
23      out_side_green : OUT std_logic;
24      out_main_walk : OUT std_logic;
25      out_side_walk : OUT std_logic
26  );
27  end FSM;
28
29  architecture Behavioral of FSM is
30
31  --** FSM states
32  TYPE state_type IS (
33      main_red,
34      main_green_1st, -- first 8s (available for sensor input)
35      main_green_2nd, -- 2s
36      main_green_1st_extended,
37      main_yellow,
38      side_red,
39      side_green_1st,
40      side_green_2nd,
41      side_yellow
42  );
43  SIGNAL state, next_state : state_type := main_red;
44
45  --** constants, signals for clock dividers
46  constant sec_05 : integer := 25e6; -- counts 0.5sec, since board
        operates on 50MHz
47  constant sec_1 : integer := 2 * sec_05;
48  constant sec_2 : integer := 2 * sec_1;
49  constant sec_3 : integer := 3 * sec_1;
50  constant sec_5 : integer := 5 * sec_1;
51  constant sec_8 : integer := 8 * sec_1;
52  signal clk_count_state : integer range 0 to sec_8;
53  signal clk_count_blink : integer range 0 to sec_05;
54
55  --** signals for registers
56      -- store whether pedestrian has pressed walk button
57      -- a signal becomes a register when assigned inside clocked
           process
58  signal reg_main_walk, reg_side_walk, reg_blink : std_logic :=
        '0';
59
60
61  begin
62
63  --** clock divider process for state transitions; sequential
64  -- WHEN to move to next
65  PROCESS (in_clk)
66  BEGIN
67      if (rising_edge(in_clk)) then
```

```
68
69          CASE state IS
70
71              -- states with 3 s period: both lights red
72              WHEN main_red | side_red =>
73                  if (clk_count_state = sec_3) then
74                      state <= next_state;
75                      clk_count_state <= 0;
76                  else
77                      clk_count_state <= clk_count_state + 1;
78                  end if;
79
80              -- state with 8 s period: first green phase
81              WHEN main_green_1st =>
82                  if (clk_count_state = sec_8) then
83                      if (in_sensor_main = '1') then -- check
                            sensor
84                          state <= main_green_1st_extended;
85                      else
86                          state <= main_green_2nd;
87                      end if;
88
89                      clk_count_state <= 0;
90                  else
91                      clk_count_state <= clk_count_state + 1;
92                  end if;
93
94              -- state with 5 s period: extended green phase (main)
95              WHEN main_green_1st_extended =>
96                  if (clk_count_state = sec_5) then
97                      state <= main_green_2nd;
98                      clk_count_state <= 0;
99                  else
100                     clk_count_state <= clk_count_state + 1;
101                 end if;
102
103             -- states with 2 s period
104             WHEN main_green_2nd | main_yellow | side_green_2nd |
                    side_yellow =>
105                 if (clk_count_state = sec_2) then
106                     state <= next_state;
107                     clk_count_state <= 0;
108                 else
109                     clk_count_state <= clk_count_state + 1;
110                 end if;
111
112             -- state with 8 s period: second green phase (side)
113             WHEN side_green_1st =>
114                 if (clk_count_state = sec_8) then
115                     state <= side_green_2nd;
116                     clk_count_state <= 0;
```

```
117                 else
118                     clk_count_state <= clk_count_state + 1;
119                 end if;
120 
121             WHEN OTHERS =>
122                 clk_count_state <= 0;
123 
124         END CASE;
125     end if;
126 END PROCESS;
127 
128 --** clock divider process for blinking operation; sequential
129 -- BONUS revise this process (and possibly other parts) such that
        the blinking starts properly, with light off and counter at
       0, right after relevant state transitions
130 PROCESS (in_clk)
131 BEGIN
132     if (rising_edge(in_clk)) then
133         if (clk_count_blink = sec_05) then
134             reg_blink <= not reg_blink;
135             clk_count_blink <= 0;
136         else
137             clk_count_blink <= clk_count_blink + 1;
138         end if;
139     end if;
140 END PROCESS;
141 
142 --** register; reg_side_walk
143 -- BONUS revise such that any further request that comes in just
       after the cycle has finished is not ignored
144 PROCESS (in_clk)
145 BEGIN
146 
147     if (rising_edge(in_clk)) then
148 
149         -- condition of clearing the request
150         if (state = main_red  and in_side_walk = '0') then
151             reg_side_walk <= '0';
152         -- for all other states, set the register in case the
               button is pushed
153         elsif (in_side_walk = '1') then
154             reg_side_walk <= '1';
155         -- note: no final else, to hold request once set (and
               until reset)
156         end if;
157     end if;
158 
159 END PROCESS;
160 
161 --** register; reg_main_walk (remembers walk request)
```

```
162 -- BONUS revise such that any further request that comes in just
    after the cycle has finished is not ignored
163 PROCESS (in_clk)
164 BEGIN
165
166     if (rising_edge(in_clk)) then
167
168         -- condition of clearing the request
169         if (state = side_red  and in_main_walk = '0') then
170             reg_main_walk <= '0';
171         -- for all other scenarios, set the register in case the
            button is pushed
172         elsif (in_main_walk = '1') then -- if button is pushed
173             reg_main_walk <= '1'; -- hold a pending request
174         -- note: no final else, to hold request once set (and
            until reset)
175         end if;
176     end if;
177
178 END PROCESS;
179
180 --** next state; sequential
181 -- WHERE to move next
182 PROCESS (state, in_sensor_main)
183 BEGIN
184
185 CASE state IS
186
187     WHEN main_red =>
188         next_state <= main_green_1st;
189
190     WHEN main_green_1st =>
191         if (in_sensor_main = '1') then
192             next_state <= main_green_1st_extended;
193         else
194             next_state <= main_green_2nd;
195         end if;
196
197     WHEN main_green_1st_extended =>
198         next_state <= main_green_2nd;
199
200     WHEN main_green_2nd =>
201         next_state <= main_yellow;
202
203     WHEN main_yellow =>
204         next_state <= side_red;
205
206     WHEN side_red =>
207         next_state <= side_green_1st;
208
209     WHEN side_green_1st =>
```

```
210         next_state <= side_green_2nd;
211
212     WHEN side_green_2nd =>
213         next_state <= side_yellow;
214
215     WHEN side_yellow =>
216         next_state <= main_red;
217
218     WHEN OTHERS => -- in case of invalid state,
219         next_state <= state; -- restart the cycle
220
221 END CASE;
222
223 END PROCESS;
224
225 --** outputs; combinational
226 PROCESS (state, reg_side_walk, reg_main_walk, reg_blink)
227 BEGIN
228
229 -- default assignments
230 out_main_red <= '0';
231 out_main_yellow <= '0';
232 out_main_green <= '0';
233 out_side_red <= '0';
234 out_side_yellow <= '0';
235 out_side_green <= '0';
236 out_main_walk <= '0';
237 out_side_walk <= '0';
238
239 CASE state IS
240
241     WHEN main_red =>
242         out_main_red <= '1';
243         out_side_red <= '1';
244
245     WHEN main_green_1st =>
246         out_main_green <= '1';
247         out_side_red <= '1';
248         out_main_walk <= reg_main_walk;
249
250     WHEN main_green_1st_extended =>
251         out_main_green <= '1';
252         out_side_red <= '1';
253         out_main_walk <= reg_main_walk and reg_blink; -- can't
              only be reg_blink since it toggles
254
255     WHEN main_green_2nd =>
256         out_main_yellow <= '1';
257         out_side_red <= '1';
258         out_main_walk <= reg_main_walk and reg_blink;
259
```

```
260     WHEN main_yellow =>
261         out_main_yellow <= '1';
262         out_side_red <= '1';
263         out_main_walk <= reg_main_walk and reg_blink;
264 
265     WHEN side_red =>
266         out_main_red <= '1';
267         out_side_red <= '1';
268 
269     WHEN side_green_1st =>
270         out_main_red <= '1';
271         out_side_green <= '1';
272         out_side_walk <= reg_side_walk;
273 
274 
275     WHEN side_green_2nd =>
276         out_main_red <= '1';
277         out_side_green <= '1';
278         out_side_walk <= reg_side_walk and reg_blink;
279 
280 
281     WHEN side_yellow =>
282         out_main_red <= '1';
283         out_side_yellow <= '1';
284         out_side_walk <= reg_side_walk and reg_blink;
285 
286     WHEN OTHERS =>
287         NULL;
288 
289 
290 
291 
292 END CASE;
293 
294 END PROCESS;
295 
296 end Behavioral;
```

Listing 1: VHDL code for "FSM" module

3.2 TestBench Module

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3 
4 -- Uncomment the following library declaration if using
5 -- arithmetic functions with Signed or Unsigned values
6 --USE ieee.numeric_std.ALL;
7 
8 ENTITY FSM_tb_c IS
9 END FSM_tb_c;
```

```
10
11 ARCHITECTURE behavior OF FSM_tb_c IS
12
13     -- Component Declaration for the Unit Under Test (UUT)
14
15     COMPONENT FSM
16     PORT(
17          in_clk : IN  std_logic;
18          in_sensor_main : IN  std_logic;
19          in_main_walk : IN  std_logic;
20          in_side_walk : IN  std_logic;
21          out_main_red : OUT  std_logic;
22          out_main_yellow : OUT  std_logic;
23          out_main_green : OUT  std_logic;
24          out_side_red : OUT  std_logic;
25          out_side_yellow : OUT  std_logic;
26          out_side_green : OUT  std_logic;
27          out_main_walk : OUT  std_logic;
28          out_side_walk : OUT  std_logic
29         );
30     END COMPONENT;
31
32
33    --Inputs
34    signal in_clk : std_logic := '0';
35    signal in_sensor_main_tb : std_logic := '0';
36    signal in_main_walk_tb : std_logic := '0';
37    signal in_side_walk_tb : std_logic := '0';
38
39     --Outputs
40    signal out_main_red_tb : std_logic;
41    signal out_main_yellow_tb : std_logic;
42    signal out_main_green_tb : std_logic;
43    signal out_side_red_tb : std_logic;
44    signal out_side_yellow_tb : std_logic;
45    signal out_side_green_tb : std_logic;
46    signal out_main_walk_tb : std_logic;
47    signal out_side_walk_tb : std_logic;
48
49    -- Clock period definitions
50    constant in_clk_period : time := 5 ns;
51
52 BEGIN
53
54     -- Instantiate the Unit Under Test (UUT)
55    uut: FSM PORT MAP (
56            in_clk => in_clk,
57            in_sensor_main => in_sensor_main_tb,
58            in_main_walk => in_main_walk_tb,
59            in_side_walk => in_side_walk_tb,
60            out_main_red => out_main_red_tb,
```

```
61            out_main_yellow => out_main_yellow_tb,
62            out_main_green => out_main_green_tb,
63            out_side_red => out_side_red_tb,
64            out_side_yellow => out_side_yellow_tb,
65            out_side_green => out_side_green_tb,
66            out_main_walk => out_main_walk_tb,
67            out_side_walk => out_side_walk_tb
68          );
69
70    -- Clock process definitions
71    in_clk_process :process
72    begin
73         in_clk <= '0';
74         wait for in_clk_period/2;
75         in_clk <= '1';
76         wait for in_clk_period/2;
77    end process;
78
79
80    -- Stimulus process
81    stim_proc: process
82    begin
83 -- Before this, change: constant sec_05 : integer := 25e6; to 25
      e1 in FSM
84 --       TB1 Everything is Zero
85
86 --       TB 2  - sensor on
87 --       in_sensor_main_tb <= '1';
88
89
90
91 --  TB_3    - press both walk buttons with sensor off
92 --       wait for 10 us;
93          -- TB_4 - change sensor to 1
94 --       in_sensor_main_tb <= '1';
95 --       in_side_walk_tb <= '1';
96 --       in_main_walk_tb <= '1';
97 --
98 --       wait for 5 us;
99 --       in_side_walk_tb <= '0';
100 --      in_main_walk_tb <= '0';
101
102
103 --  TB_4 - An x error is assigned for the sensor
104 --       in_sensor_main_tb <= 'X';
105
106 --       TB_5 - An x error is assigned to both walk buttons
107         in_side_walk_tb <= 'X';
108         in_main_walk_tb <= 'X';
109
110       wait;
```

```
111     end process;
112
113 END;
```

Listing 2: VHDL code for "FSM" module

4 Testbench simulation results

4.1 All inputs off

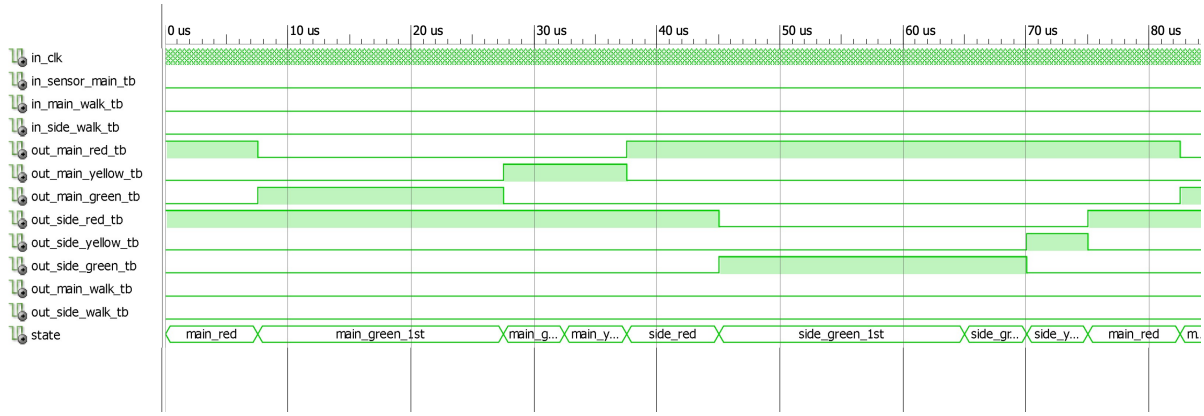


Figure 6: Board setup based on UCF

4.2 Main sensor on (Main green extended)

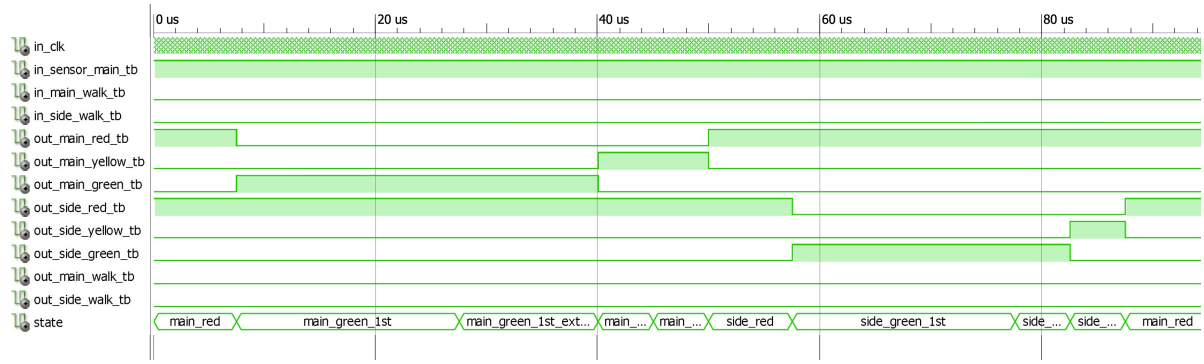


Figure 7: Board setup based on UCF

4.3 Both walk buttons on

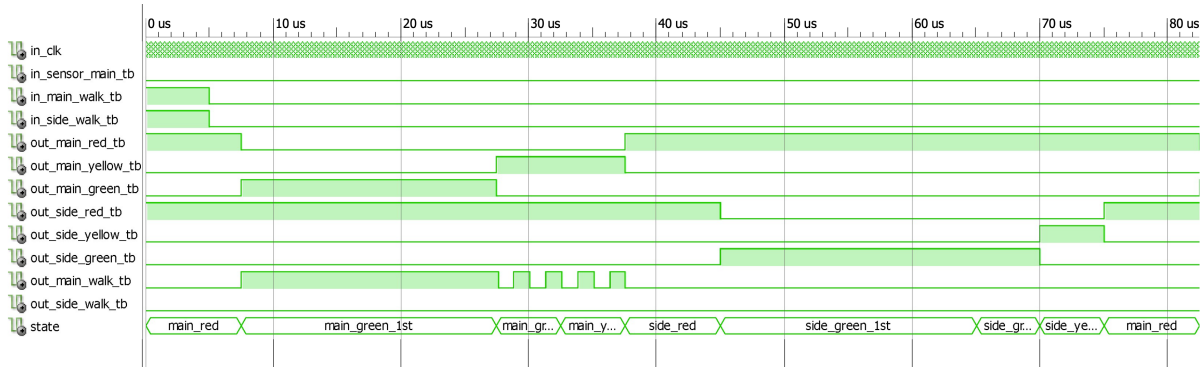


Figure 8: Board setup based on UCF

4.4 Both walk buttons and main sensor on

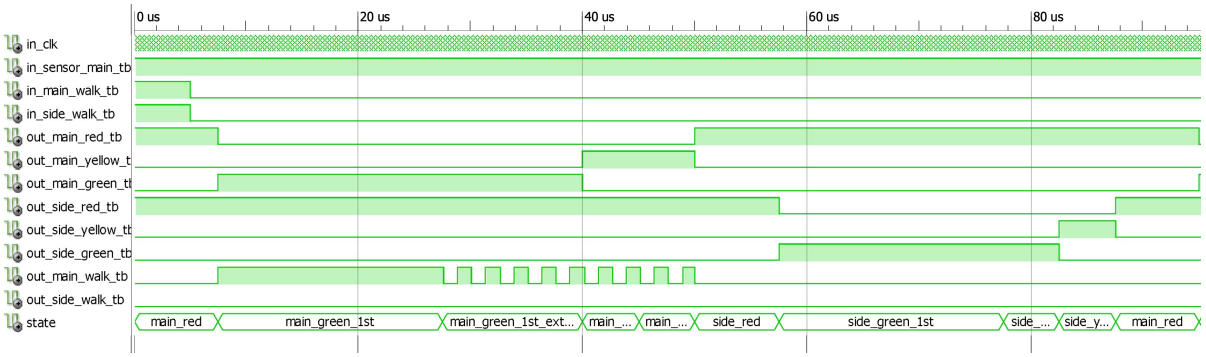


Figure 9: Board setup based on UCF

4.5 ”X” error for main sensor

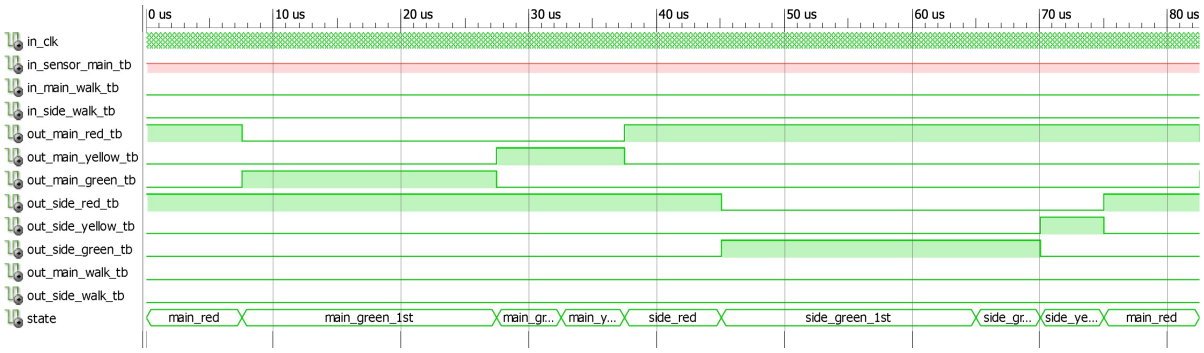


Figure 10: Board setup based on UCF

4.6 ”X” error for both walk buttons

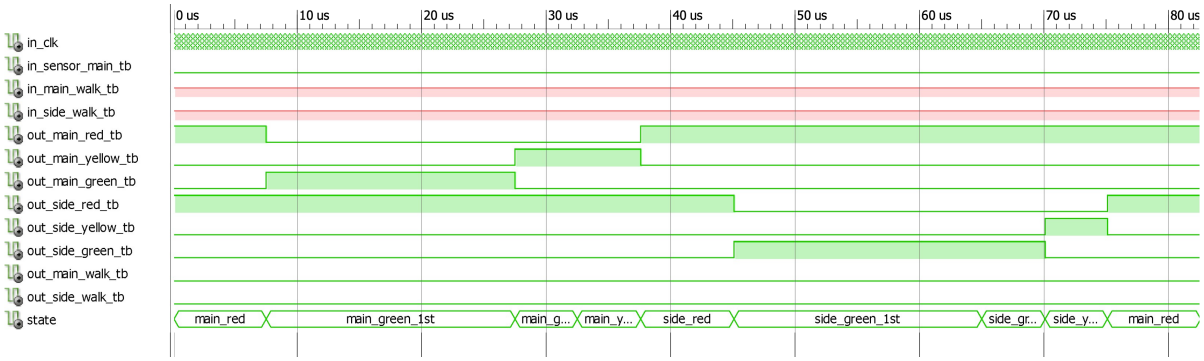


Figure 11: Board setup based on UCF

5 Constraints file

```
# clock pin for Basys2 Board
NET "in_clk" LOC = "B8"; # Bank = 0, Signal name = MCLK

# Pin assignment for LEDs
NET "out_main_red" LOC = "G1" ; # Bank = 3, Signal name = LD7
NET "out_main_yellow" LOC = "P4" ; # Bank = 2, Signal name = LD6
NET "out_main_green" LOC = "N4" ;  # Bank = 2, Signal name = LD5

NET "out_main_walk" LOC = "N5" ;
NET "out_side_walk" LOC = "P6" ;

NET "out_side_red" LOC = "P7" ;
NET "out_side_yellow" LOC = "M11" ;
NET "out_side_green" LOC = "M5" ;

# Pin assignment for SWs
NET "in_sensor_main" LOC = "N3";  # Bank = 2, Signal name = SW7

# Pin assignment for BTNs
NET "in_main_walk" LOC = "A7";
NET "in_side_walk" LOC = "G12";
```

Listing 3: Constraints file for FSM module

6 FPGA implementation

Click to watch video demonstration