AnaSampaio13 ... 11 seconds ago

..

Pictures 8 minutes ago

README.md 11 seconds ago

README.md

#Digital-Electronics-1

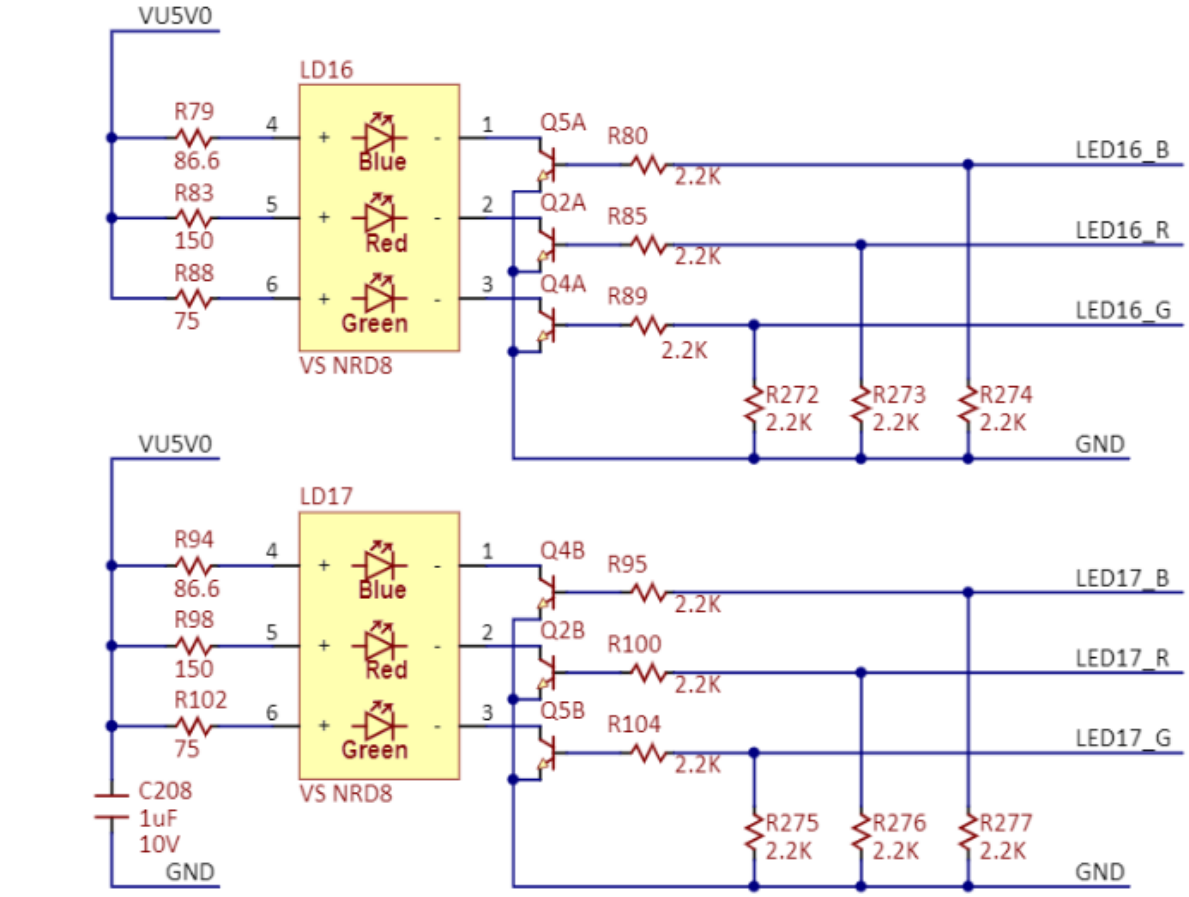
https://github.com/AnaSampaio13/Digital-Electronics-1

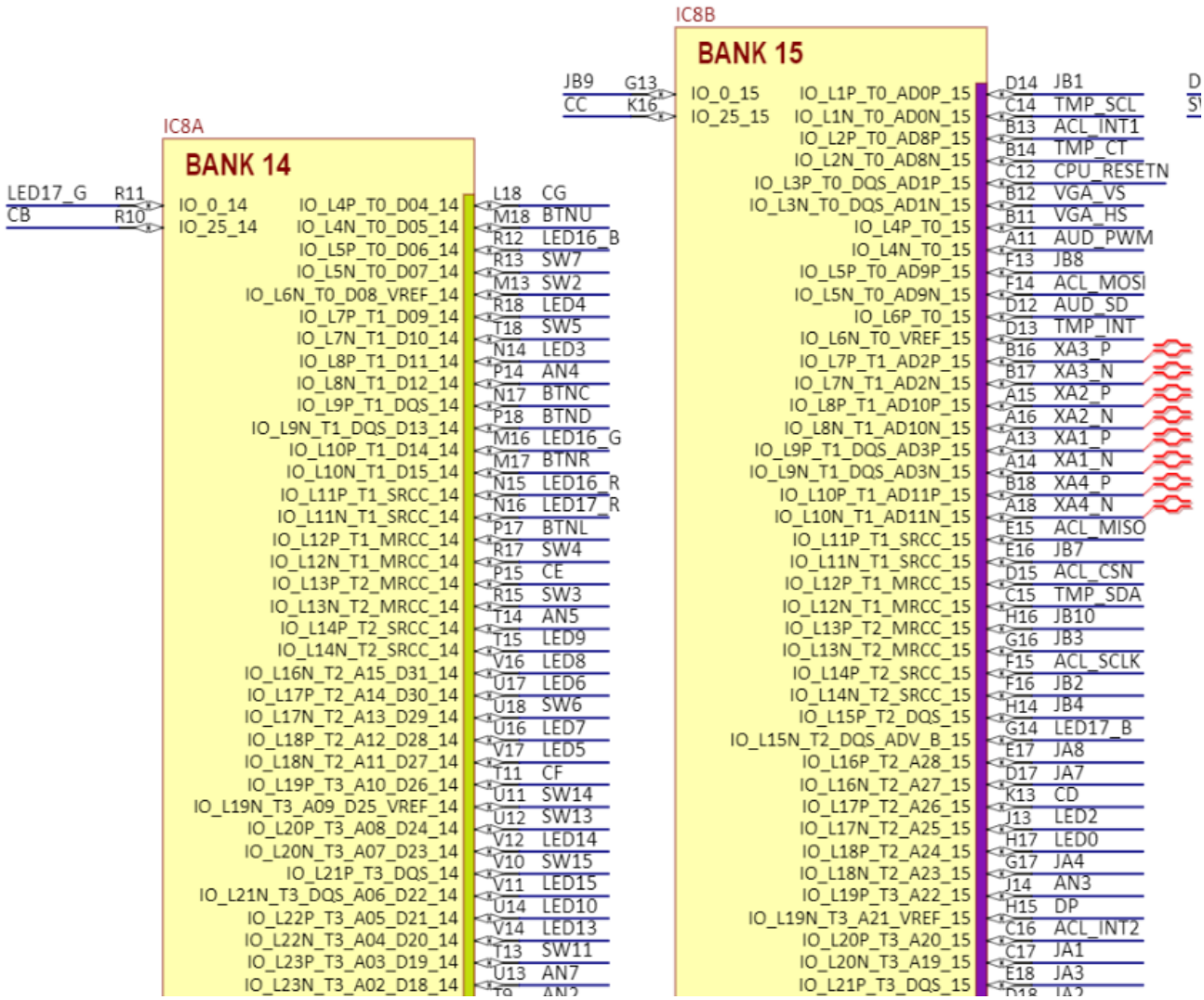
##Exercise 1

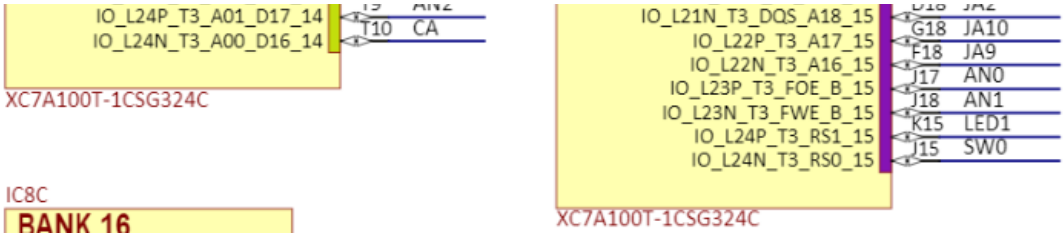
###State table

Input P	0	0	1	1	0	1	0	1	1	1	1	0
Clock	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
State	A	A	B	C	C	D	A	B	C	D	B	B
Output R	0	0	0	0	0	1	0	0	0	1	0	0

###Image


VU5V0
LD16
R79
86.6
R83
150
R88
75
Blue
Red
Green
VS NRD8
Q5A
Q2A
Q4A
R80
R85
R89
2.2K
LED16_B
LED16_R
LED16_G
R272
R273
R274
GND
VU5V0
LD17
R94
86.6
R98
150
R102
75
Blue
Red
Green
VS NRD8
C208
1uF
10V
GND
Q4B
Q2B
Q5B
R95
R100
R104
LED17_B
LED17_R
LED17_G
R275
R276
R277
GND

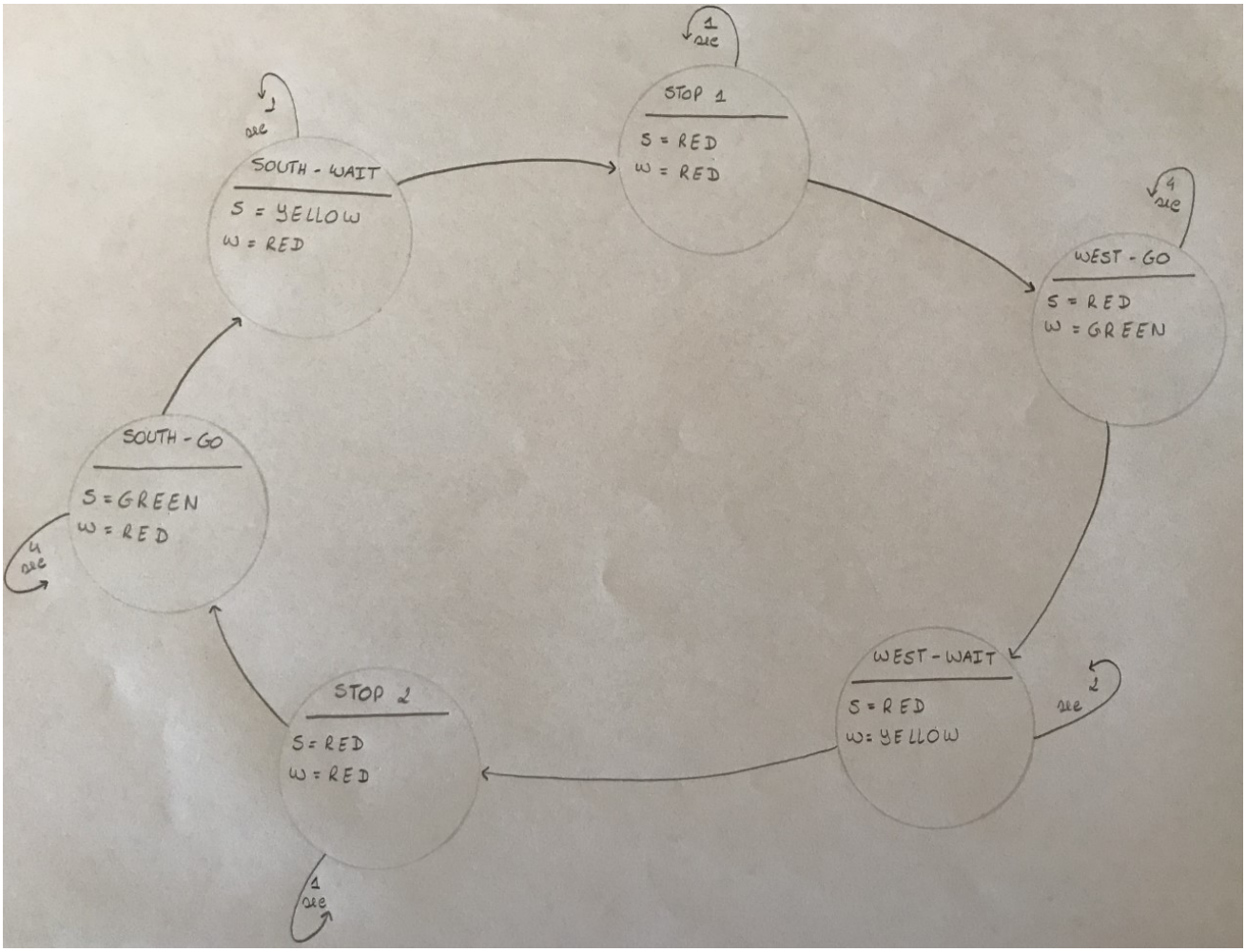

IC8A
BANK 14
LED17_G R11
CB R10
IO_0_14
IO_25_14
IO_L4P_T0_D04_14 L18 CG
IO_L4N_T0_D05_14 M18 BTNU
IO_L5P_T0_D06_14 R12 LED16_B
IO_L5N_T0_D07_14 R13 SW7
IO_L6N_T0_D08_VREF_14 M13 SW2
IO_L7P_T1_D09_14 R18 LED4
IO_L7N_T1_D10_14 T18 SW5
IO_L8P_T1_D11_14 N14 LED3
IO_L8N_T1_D12_14 P14 AN4
IO_L9P_T1_DQS_14 N17 BTNC
IO_L9N_T1_DQS_D13_14 P18 BTND
IO_L10P_T1_D14_14 M16 LED16_G
IO_L10N_T1_D15_14 M17 BTNR
IO_L11P_T1_SRCC_14 N15 LED16_R
IO_L11N_T1_SRCC_14 N16 LED17_R
IO_L12P_T1_MRCC_14 P17 BTNL
IO_L12N_T1_MRCC_14 R17 SW4
IO_L13P_T2_MRCC_14 P15 CE
IO_L13N_T2_MRCC_14 R15 SW3
IO_L14P_T2_SRCC_14 T14 AN5
IO_L14N_T2_SRCC_14 T15 LED9
IO_L16N_T2_A15_D31_14 V16 LED8
IO_L17P_T2_A14_D30_14 U17 LED6
IO_L17N_T2_A13_D29_14 U18 SW6
IO_L18P_T2_A12_D28_14 U16 LED7
IO_L18N_T2_A11_D27_14 V17 LED5
IO_L19P_T3_A10_D26_14 T11 CF
IO_L19N_T3_A09_D25_VREF_14 U11 SW14
IO_L20P_T3_A08_D24_14 U12 SW13
IO_L20N_T3_A07_D23_14 V12 LED14
IO_L21P_T3_DQS_14 V10 SW15
IO_L21N_T3_DQS_A06_D22_14 V11 LED15
IO_L22P_T3_A05_D21_14 U14 LED10
IO_L22N_T3_A04_D20_14 V14 LED13
IO_L23P_T3_A03_D19_14 T13 SW11
IO_L23N_T3_A02_D18_14 U13 AN7
IC8B
BANK 15
JB9 G13
CC K16
IO_0_15
IO_25_15
IO_L1P_T0_AD0P_15 D14 JB1
IO_L1N_T0_AD0N_15 C14 TMP_SCL
IO_L2P_T0_AD8P_15 B13 ACL_INT1
IO_L2N_T0_AD8N_15 B14 TMP_CT
IO_L3P_T0_DQS_AD1P_15 C12 CPU_RESETN
IO_L3N_T0_DQS_AD1N_15 B12 VGA_VS
IO_L4P_T0_15 B11 VGA_HS
IO_L4N_T0_15 A11 AUD_PWM
IO_L5P_T0_AD9P_15 F13 JB8
IO_L5N_T0_AD9N_15 F14 ACL_MOSI
IO_L6P_T0_15 D12 AUD_SD
IO_L6N_T0_VREF_15 D13 TMP_INT
IO_L7P_T1_AD2P_15 B16 XA3_P
IO_L7N_T1_AD2N_15 B17 XA3_N
IO_L8P_T1_AD10P_15 A15 XA2_P
IO_L8N_T1_AD10N_15 A16 XA2_N
IO_L9P_T1_DQS_AD3P_15 A13 XA1_P
IO_L9N_T1_DQS_AD3N_15 A14 XA1_N
IO_L10P_T1_AD11P_15 B18 XA4_P
IO_L10N_T1_AD11N_15 A18 XA4_N
IO_L11P_T1_SRCC_15 E15 ACL_MISO
IO_L11N_T1_SRCC_15 E16 JB7
IO_L12P_T1_MRCC_15 D15 ACL_CSN
IO_L12N_T1_MRCC_15 C15 TMP_SDA
IO_L13P_T2_MRCC_15 H16 JB10
IO_L13N_T2_MRCC_15 G16 JB3
IO_L14P_T2_SRCC_15 F15 ACL_SCLK
IO_L14N_T2_SRCC_15 F16 JB2
IO_L15P_T2_DQS_15 H14 JB4
IO_L15N_T2_DQS_ADV_B_15 G14 LED17_B
IO_L16P_T2_A28_15 E17 JA8
IO_L16N_T2_A27_15 D17 JA7
IO_L17P_T2_A26_15 K13 CD
IO_L17N_T2_A25_15 J13 LED2
IO_L18P_T2_A24_15 H17 LED0
IO_L18N_T2_A23_15 G17 JA4
IO_L19P_T3_A22_15 J14 AN3
IO_L19N_T3_A21_VREF_15 H15 DP
IO_L20P_T3_A20_15 C16 ACL_INT2
IO_L20N_T3_A19_15 C17 JA1
IO_L21P_T3_DQS_15 E18 JA3



RGB LED	Artix-7 pin names	Red	Yellow	Green
LD16	N15, M16, R12	`1,0,0`	`1,1,0`	`0,1,0`
LD17	N16, R11, G14	`1,0,0`	`1,1,0`	`0,1,0`

##Exercise 2

###State diagram



###Listing of VHDL code of sequential process

```
p_traffic_fsm : process(clk)
    begin
        if rising_edge(clk) then
```

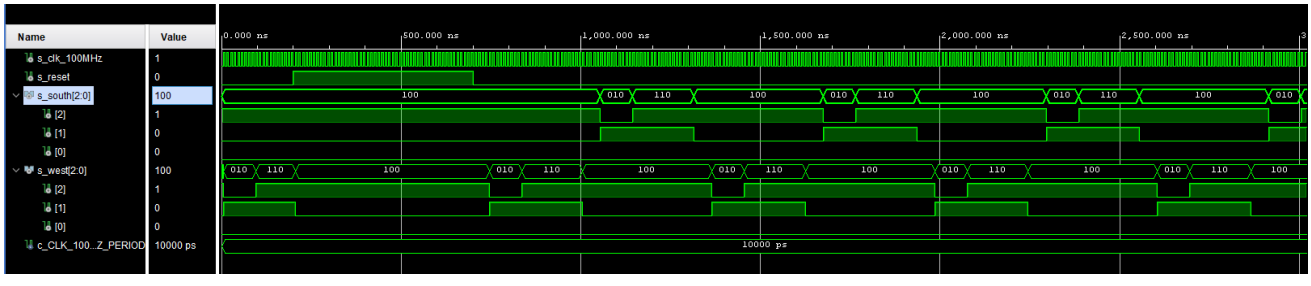
```
if (reset = '1') then       -- Synchronous reset
    s_state <= STOP1 ;      -- Set initial state
    s_cnt   <= c_ZERO;      -- Clear all bits
elsif (s_en = '1') then
    -- Every 250 ms, CASE checks the value of the s_state
    -- variable and changes to the next state according
    -- to the delay value.
    case s_state is
        -- If the current state is STOP1, then wait 1 sec
        -- and move to the next GO_WAIT state.
        when STOP1 =>
            -- Count up to c_DELAY_1SEC
            if (s_cnt < c_DELAY_1SEC) then
                s_cnt <= s_cnt + 1;
            else
                -- Move to the next state
                s_state <= WEST_GO;
                -- Reset local counter value
                s_cnt   <= c_ZERO;
            end if;
        when WEST_GO =>
            if (s_cnt < c_DELAY_2SEC) then
                s_cnt <= s_cnt + 1;
            else
                s_state <= WEST_WAIT;
                s_cnt   <= c_ZERO;
            end if;
        when WEST_WAIT =>
            if (s_cnt < c_DELAY_4SEC) then
                s_cnt <= s_cnt + 1;
            else
                s_state <= STOP2;
                s_cnt   <= c_ZERO;
            end if;
        when STOP2 =>
            if (s_cnt < c_DELAY_1SEC) then
                s_cnt <= s_cnt + 1;
            else
                s_state <= SOUTH_GO;
                s_cnt   <= c_ZERO;
            end if;
        when SOUTH_GO =>
            if (s_cnt < c_DELAY_2SEC) then
                s_cnt <= s_cnt + 1;
            else
                s_state <= SOUTH_WAIT;
                s_cnt   <= c_ZERO;
            end if;
        when SOUTH_WAIT =>
            if (s_cnt < c_DELAY_4SEC) then
                s_cnt <= s_cnt + 1;
            else
                s_state <= STOP1;
                s_cnt   <= c_ZERO;
            end if;
```

```
                    -- It is a good programming practice to use the
                    -- OTHERS clause, even if all CASE choices have
                    -- been made.
                    when others =>
                        s_state <= STOP1;
                end case;
            end if; -- Synchronous reset
        end if; -- Rising edge
    end process p_traffic_fsm;
```

Listing of VHDL code of combinatorial process

```
  p_output_fsm : process(s_state)
      begin
          case s_state is
              when STOP1 =>
                  south_o <= "100";   -- Red
                  west_o  <= "100";   -- Red
              when WEST_GO =>
                  south_o <= "100";   -- Red
                  west_o  <= "010";   -- Green
              when WEST_WAIT =>
                  south_o <= "100";   -- Red
                  west_o  <= "110";   -- Yellow
              when STOP2 =>
                  south_o <= "100";   -- Red
                  west_o  <= "100";   -- Red
              when SOUTH_GO =>
                  south_o <= "010";   -- Green
                  west_o  <= "100";   -- Red
              when SOUTH_WAIT =>
                  south_o <= "110";   -- Yellow
                  west_o  <= "100";   -- Red
              when others =>
                  south_o <= "100";   -- Red
                  west_o  <= "100";   -- Red
          end case;
      end process p_output_fsm;
```

Screenshot

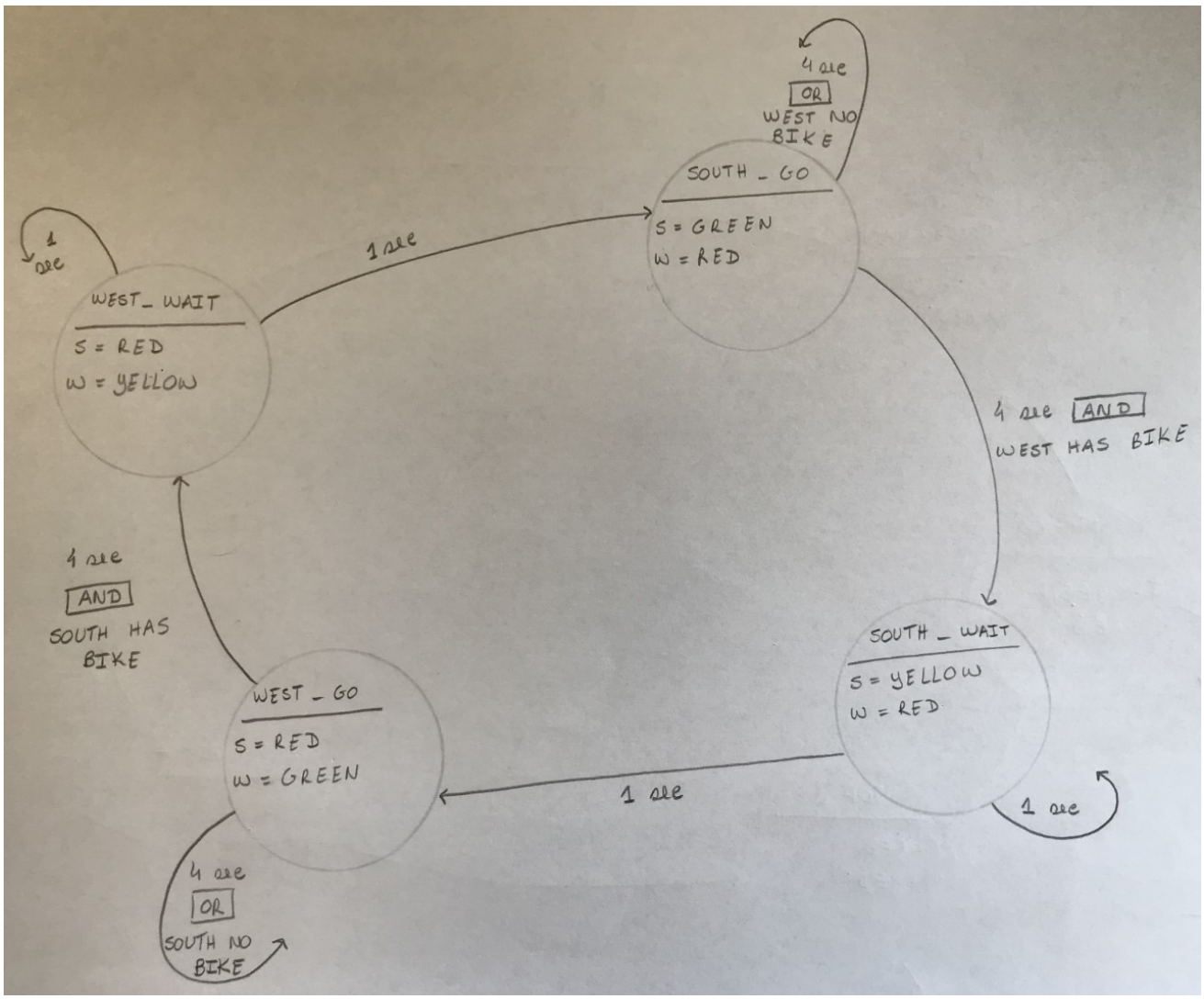


##Exercise 3

###State table

Input	No bikes	`West_i`	`West_i`	No bikes	`South_i`	`South_i`
Delay	4 seconds	1 second	4 seconds	4 seconds	4 seconds	1 second
State	`South_go`	`South_wait`	`West_go`	`West_go`	`West_go`	`West_wait`
Output	S: Green; W: Red	S: Yellow; W: Red	S: Red; W: Green	S: Red; W: Green	S: Red; W: Green	S: Red; W: Yellow

###State diagram



###Listing of VHDL code of sequential process

```
p_smart_traffic_fsm : process(clk)
    begin
        if rising_edge(clk) then
            if (reset = '1') then       -- Synchronous reset
                s_state <= STOP1 ;      -- Set initial state
                s_cnt   <= c_ZERO;      -- Clear all bits
            elsif (s_en = '1') then
                -- Every 250 ms, CASE checks the value of the s_state
                -- variable and changes to the next state according
                -- to the delay value.
                case s_state is
                    when South_go =>
                        -- Count up to c_DELAY_4SEC
                        if (s_cnt < c_DELAY_4SEC) then
                                s_cnt <= s_cnt + 1;
                        elsif (west_i = '1') then
                            -- Move to the next state
                            s_state <= South_wait;
                            -- Reset local counter value
                            s_cnt   <= c_ZERO;
                        end if;
                    when South_wait =>
                        -- Count up to c_DELAY_1SEC
                        if (s_cnt < c_DELAY_1SEC) then
                                s_cnt <= s_cnt + 1;
                        else
                            -- Move to the next state
                            s_state <= West_go;
                            -- Reset local counter value
                            s_cnt   <= c_ZERO;
                        end if;
                    when West_go =>
                        -- Count up to c_DELAY_4SEC
                        if (s_cnt < c_DELAY_4SEC) then
                                s_cnt <= s_cnt + 1;
                        elsif (South_i = '1') then
                            -- Move to the next state
                            s_state <= West_wait;
                            -- Reset local counter value
                            s_cnt   <= c_ZERO;
                        end if;
                    when West_wait =>
                        -- Count up to c_DELAY_1SEC
                        if (s_cnt < c_DELAY_1SEC) then
                                s_cnt <= s_cnt + 1;
                        else
                            -- Move to the next state
                            s_state <= South_go;
                            -- Reset local counter value
                            s_cnt   <= c_ZERO;
                        end if;
                    -- It is a good programming practice to use the
```

```
                -- OTHERS clause, even if all CASE choices have
                -- been made.
                when others =>
                    s_state <= South_go;
            end case;
        end if; -- Synchronous reset
    end if; -- Rising edge
end process p_smart_traffic_fsm;
```