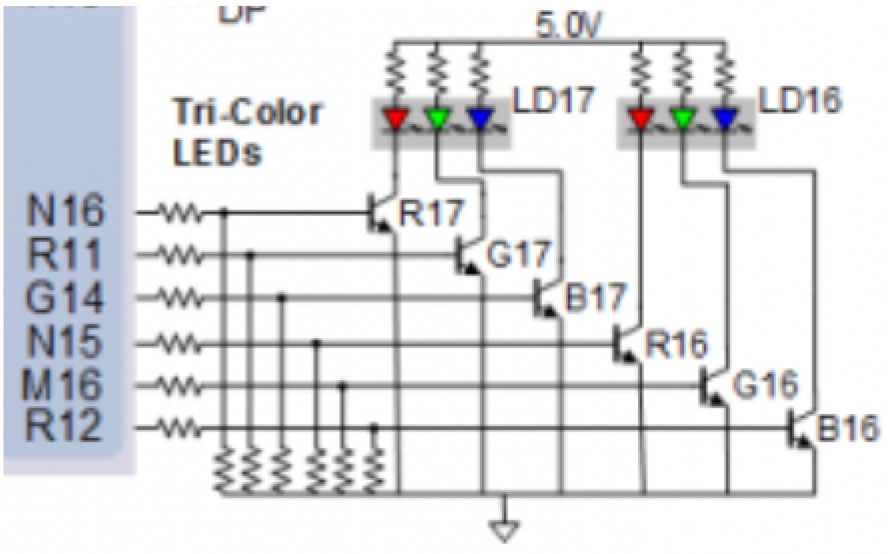
Labs 08-traffic_lights

1) Přípravné úkoly

a) Vyplněná tabulka stavů

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`

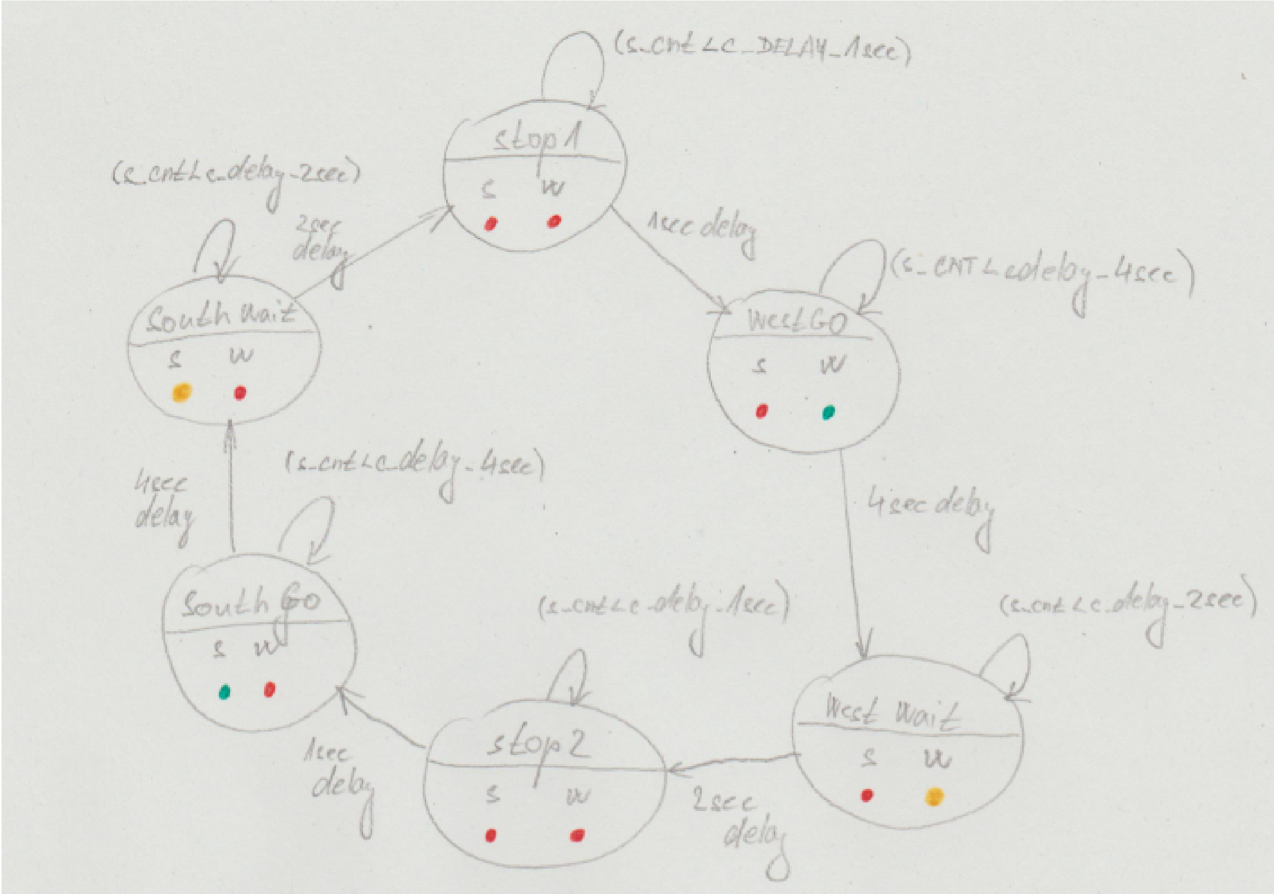
b) Obrázek s připojením RGB LED na desce Nexys A7 a vyplněnou tabulkou s nastavením barev



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`

1) Ovladač semaforu

a) Stavový diagram



b) Výpis VHDL kódu sekvenčního procesu p_traffic_fsm

```
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 =>
                -- Count up to c_DELAY_4SEC
                if (s_cnt < c_DELAY_4SEC) then
                    s_cnt <= s_cnt + 1;
                else
                    -- 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_2SEC
                if (s_cnt < c_DELAY_2SEC) then
                    s_cnt <= s_cnt + 1;
                else
                    -- Move to the next state
                    s_state <= STOP2;
                    -- Reset local counter value
                    s_cnt   <= c_ZERO;
                end if;

           when STOP2 =>
                -- 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;

          when SOUTH_GO =>
                -- Count up to c_DELAY_4SEC
                if (s_cnt < c_DELAY_4SEC) then
                    s_cnt <= s_cnt + 1;
                else
                    -- 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_2SEC
                if (s_cnt < c_DELAY_2SEC) then
                    s_cnt <= s_cnt + 1;
```

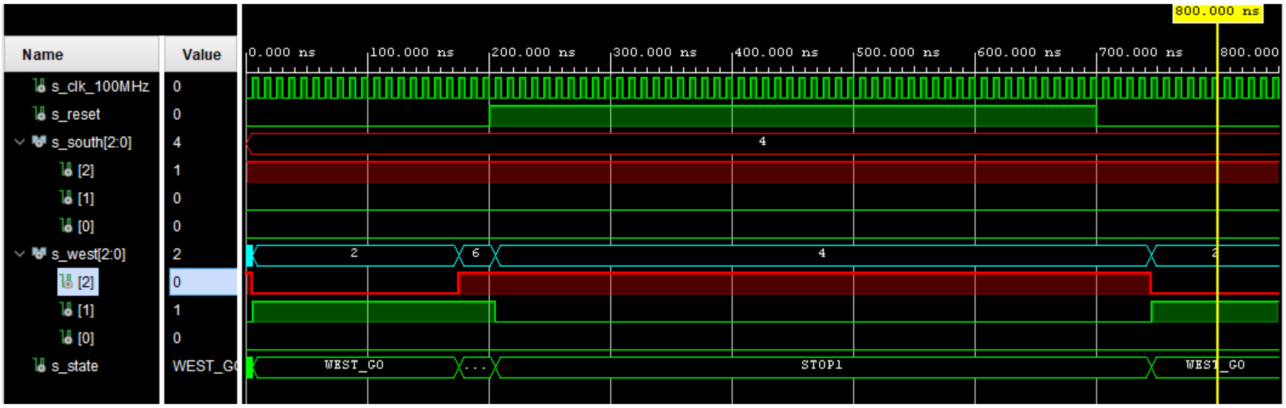
```vhdl
                    else
                        -- Move to the next state
                        s_state <= STOP1;
                        -- Reset local counter value
                        s_cnt   <= c_ZERO;
                    end if;
                -- It is a good programming practice to use the
                -- OTHERS clause, even if all CASE choices have
                -- been made.
                when others =>
                    s_state <= STOP1;

            end case;
        end if; -- Synchronous reset
    end if; -- Rising edge
end process p_traffic_fsm;
```

c) Výpis VHDL kódu kombinatorického procesu p_output_fsm

```vhdl
p_output_fsm : process(s_state)
    begin
        case s_state is
            when STOP1 =>
                south_o <= c_RED;   -- RED (RGB = 100)
                west_o  <= c_RED;   -- RED (RGB = 100)
             when WEST_GO =>
                south_o <= c_RED;   -- RED (RGB = 100)
                west_o  <= c_GREEN; -- GREEN (RGB = 010)

            when WEST_WAIT =>
                south_o <= c_RED;    -- RED (RGB = 100)
                west_o  <= c_YELLOW; -- YELLOW (RGB = 110)

            when STOP2 =>
                south_o <= c_RED;    -- RED (RGB = 100)
                west_o  <= c_RED;    -- RED (RGB = 100)

            when SOUTH_GO =>
                south_o <= c_GREEN;  -- GREEN (RGB = 010)
                west_o  <= c_RED;    -- RED (RGB = 100)

            when SOUTH_WAIT =>
                south_o <= c_YELLOW; -- YELLOW (RGB = 110)
                west_o  <= c_RED;    -- RED (RGB = 100)

            when others =>
                south_o <= c_RED;    -- RED (RGB = 100)
                west_o  <= c_RED;    -- RED (RGB = 100)
```

```
        end case;
    end process p_output_fsm;
```

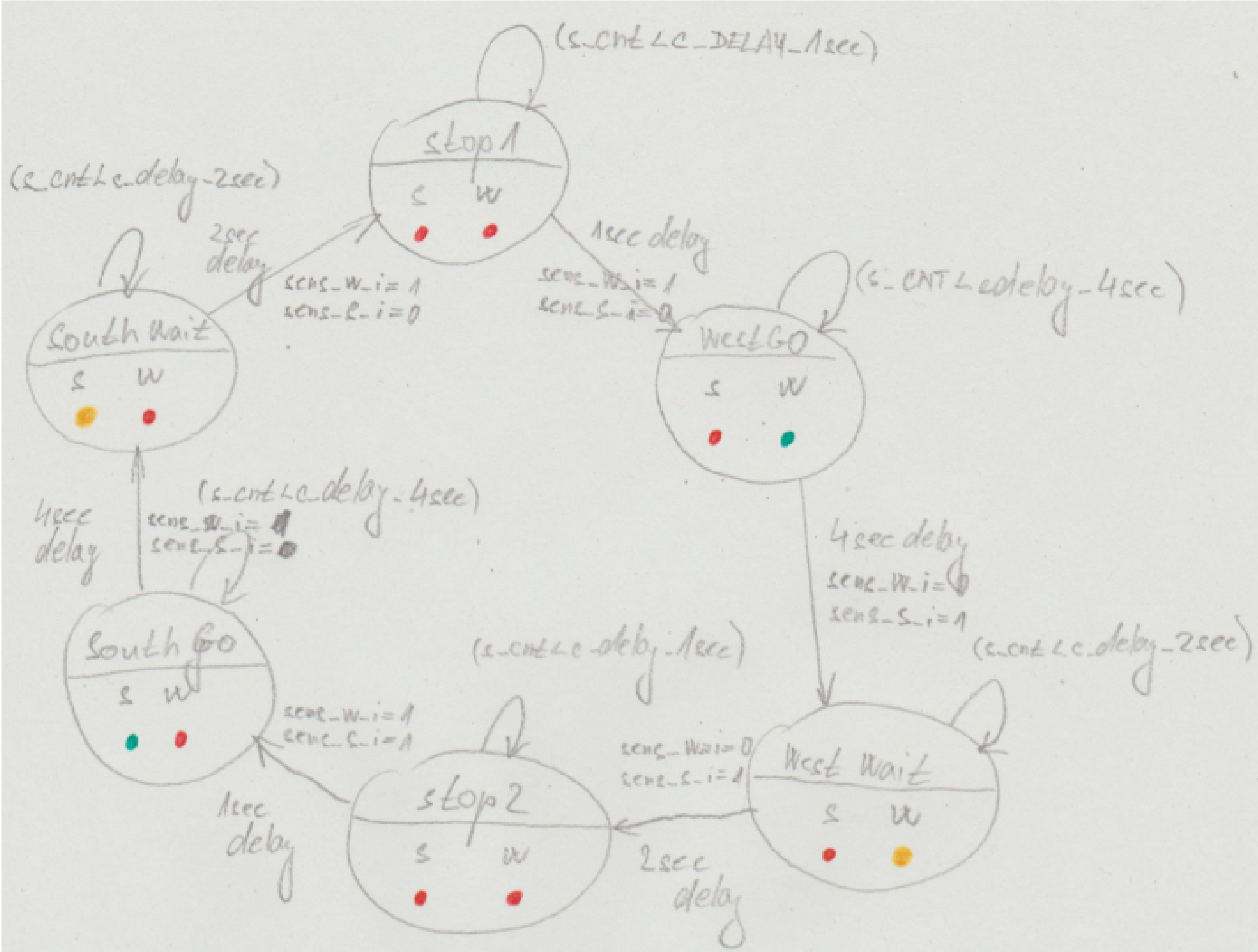
d) Screenshoty simulace



3) Inteligentní ovladač

a)

b) Stavový diagram



c) Výpis VHDL kódu sekvenčního procesu p_smart_traffic_fsm

```
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

                if ((sens_S_i = '1') and (sens_W_i = '0')) then

                      case s_state is

                          when WEST_GO =>
                                if (s_cnt < c_DELAY_4SEC) 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_2SEC) 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 others =>
                    s_state <= SOUTH_GO;

            end case;




        elsif ((sens_S_i = '0') and (sens_W_i = '1')) then

            case s_state is

                  when SOUTH_GO =>
                    if (s_cnt < c_DELAY_4SEC) 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_2SEC) then
                        s_cnt <= s_cnt + 1;
                    else
                        s_state <= STOP1;
                        s_cnt   <= c_ZERO;
                    end if;

                  when STOP1 =>
                    if (s_cnt < c_DELAY_1SEC) then
                        s_cnt <= s_cnt + 1;
                    else
                        s_state <= WEST_GO;
                        s_cnt   <= c_ZERO;
                    end if;
```

```
                   when others =>
                       s_state <= WEST_GO;

             end case;





        elsif ((sens_S_i = '0') and (sens_W_i = '0')) then
              s_state <= s_state;

        elsif ((sens_S_i = '1') and (sens_W_i = '1')) then

           case s_state is

               when STOP1 =>
                   if (s_cnt < c_DELAY_1SEC) then
                       s_cnt <= s_cnt + 1;
                   else
                       s_state <= WEST_GO;
                       s_cnt   <= c_ZERO;
                   end if;

               when WEST_GO =>
                   if (s_cnt < c_DELAY_4SEC) 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_2SEC) 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_4SEC) 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_2SEC) 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;
        end if; -- Synchronous reset
    end if; -- Rising edge
end process p_smart_traffic_fsm;
```