Lab 8 - Traffic light controller (Ondřej Smola - 217628)

1st part - Preparation task

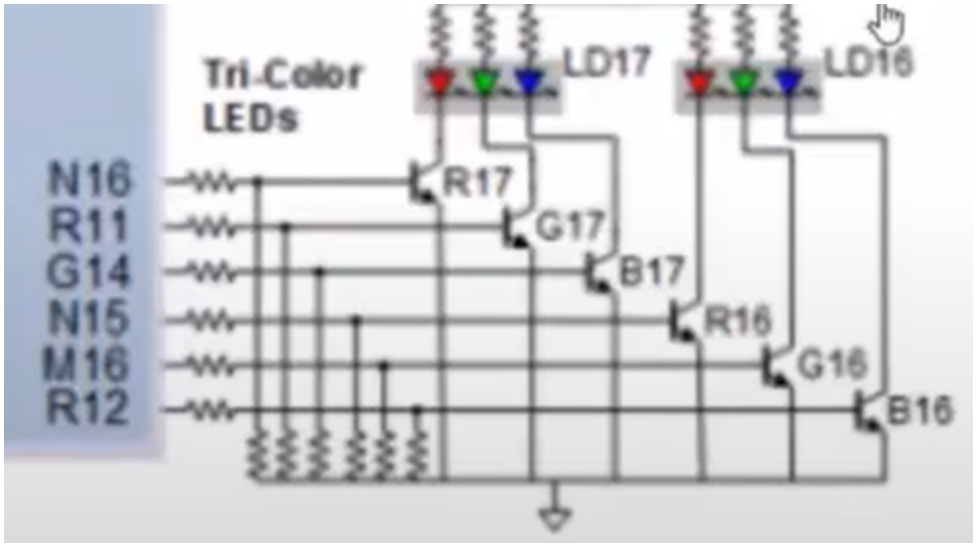
State table

Input P	0	0	1	1	0	1	0	1	1	1	1	0	0	1	1

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

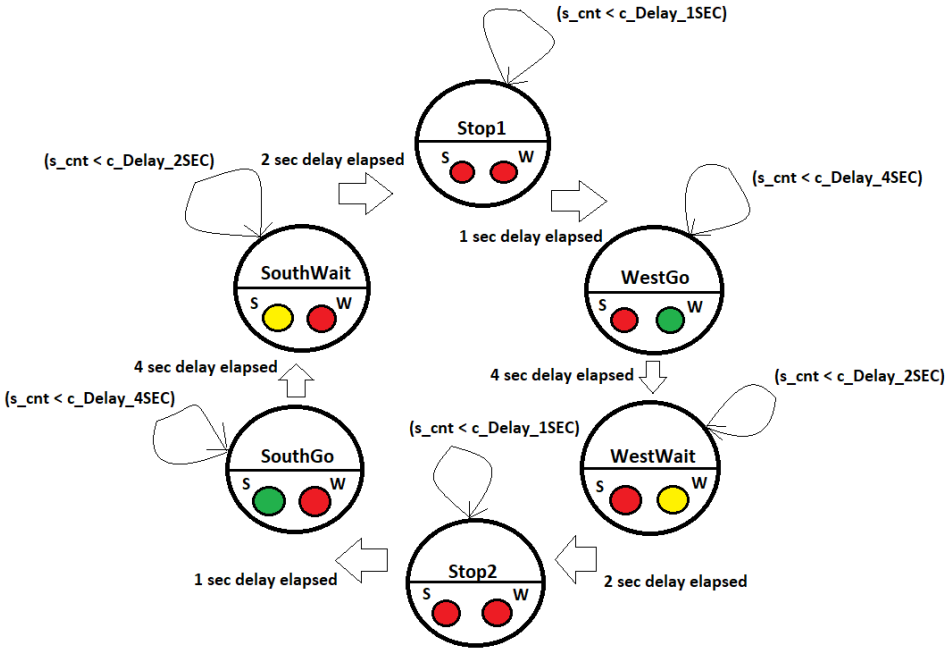
Figure with connection RGB LEDs on Nexys A7 board and completed table with color settings

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



2nd part - Traffic light controller

State diagram



Listing of VHDL code of sequential process 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 =>
```

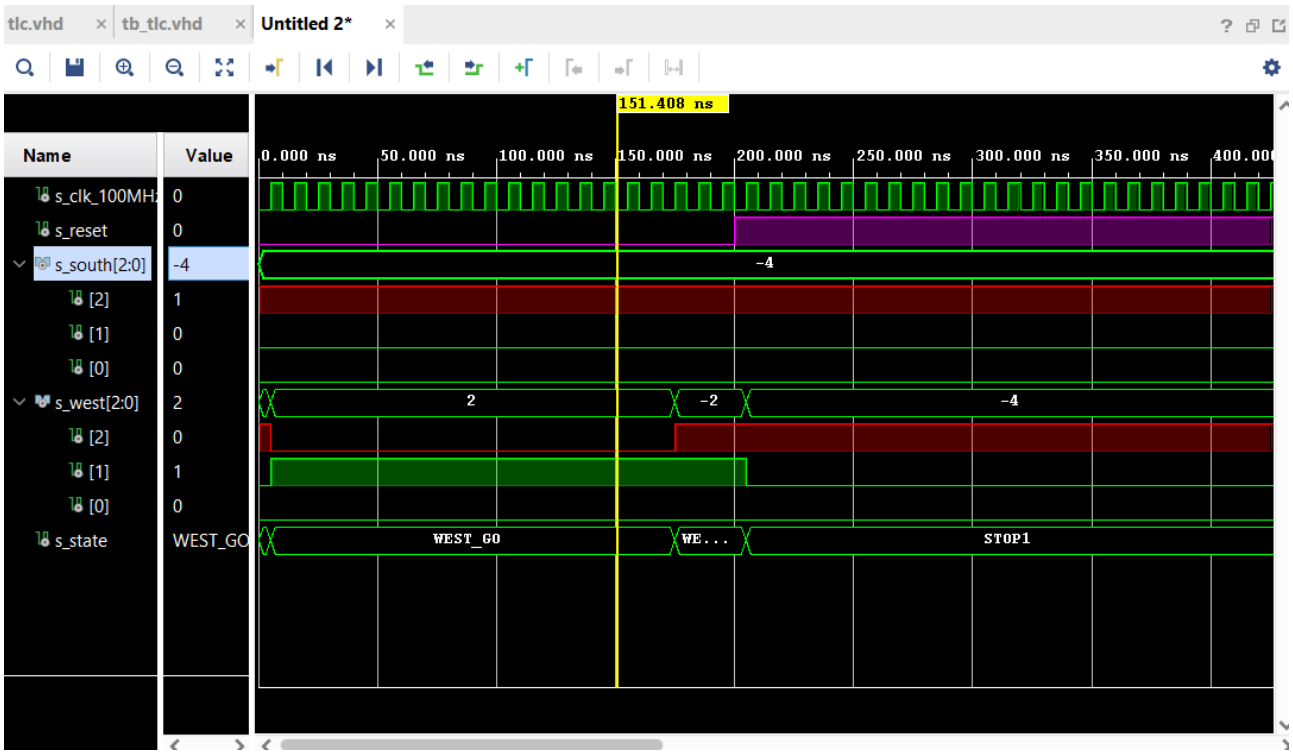
```
            -- Count up to c_DELAY_1SEC
            if (s_cnt < c_DELAY_4SEC) then
                s_cnt <= s_cnt + 1;
            else
                -- Move to the next state
                s_state <= WEST_WAIT;
                -- Reset local counter value
                s_cnt   <= c_ZERO;
            end if;

        when WEST_WAIT =>
            -- Count up to c_DELAY_1SEC
            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_1SEC
            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_1SEC
            if (s_cnt < c_DELAY_2SEC) then
                s_cnt <= s_cnt + 1;
            else
                -- Move to the next state
                s_state <= STOP1;
                -- Reset local counter value
```

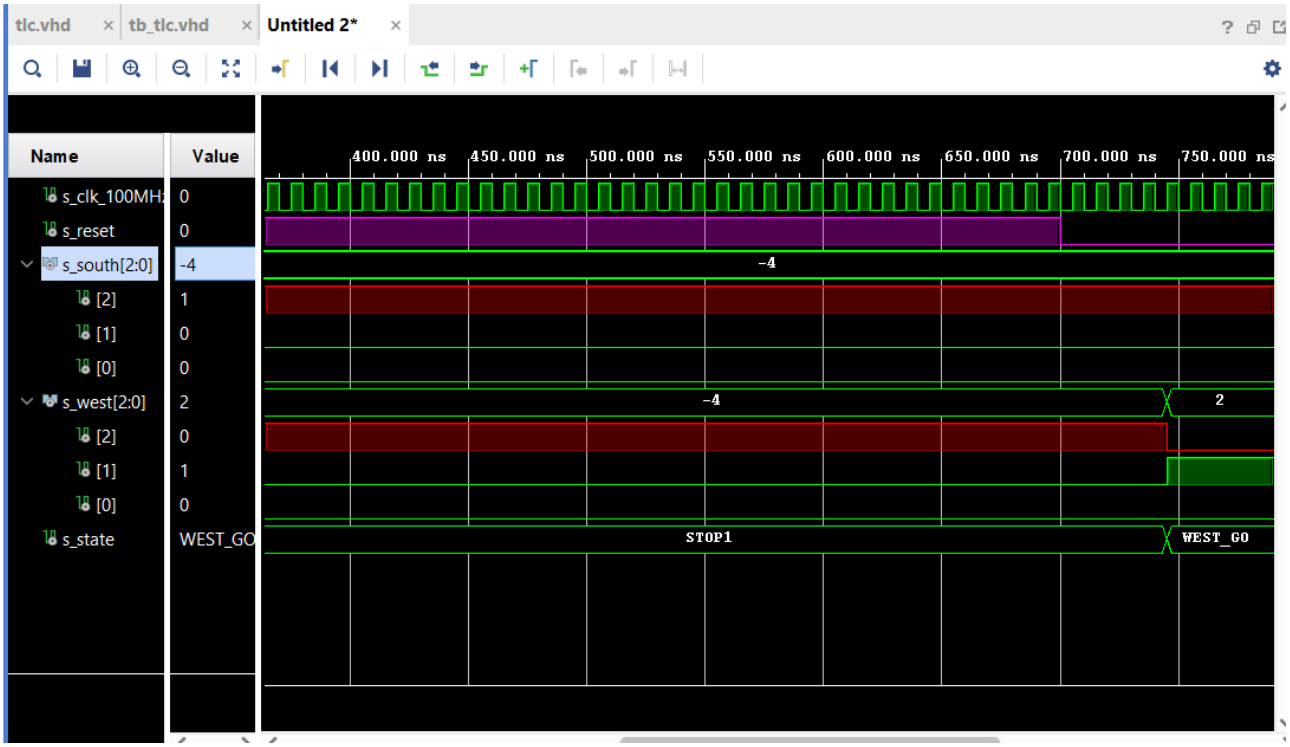
```
                        s_cnt   <= c_ZERO;
                    end if;

                    -- 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;
```

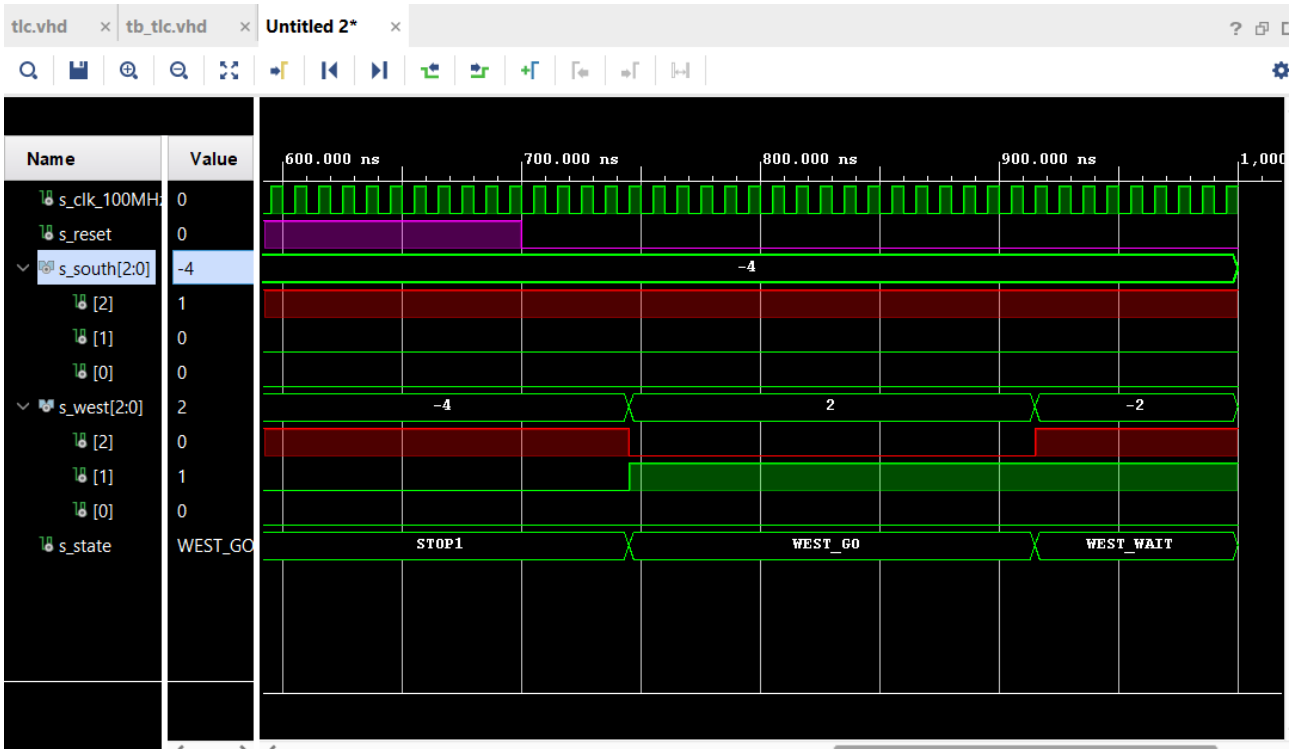
Listing of VHDL code of combinatorial process p_output_fsm

```
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;
```

Screenshots of simulation





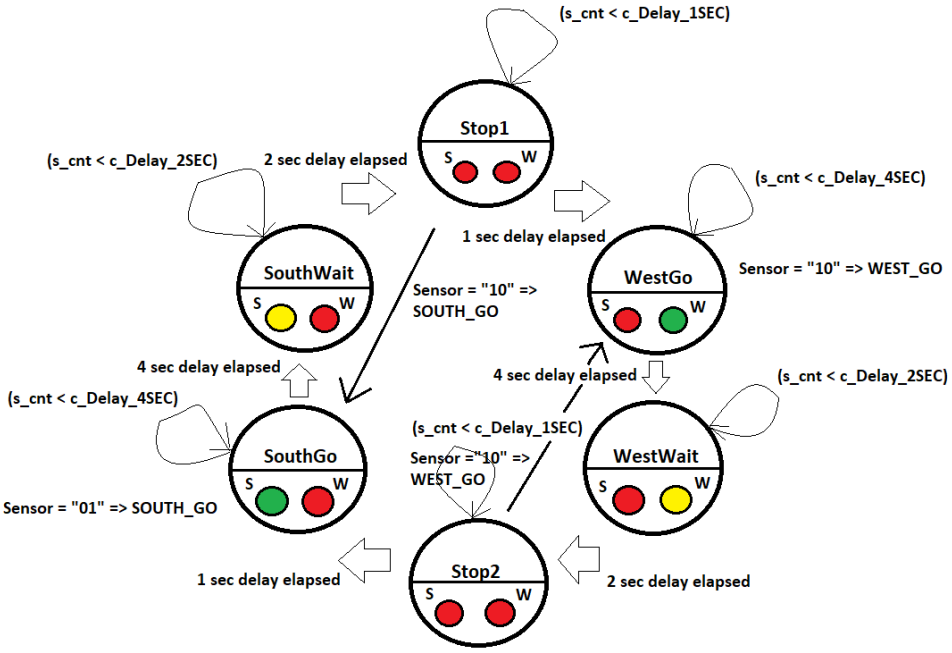


3rd part - Smart controller

State table

Current state	*Direction South*	*Direction West*	*Delay*	*Sensor state*
STOP1	red	red	1 sec	s_sensor = "01" => SOUTH_GO ; else => `WEST_GO`
WEST_GO	red	green	4 sec	s_sensor = "10" => WEST_GO ; else WEST_WAIT
WEST_WAIT	red	yellow	2 sec	STOP2
STOP2	red	red	1 sec	sensor_state = "10" => WEST_GO ; else => `SOUTH_GO`
SOUTH_GO	green	red	4 sec	s_sensor = "01" => SOUTH_GO ; SOUTH_WAIT
SOUTH_WAIT	yellow	red	2 sec	STOP1

State diagram



Listing of VHDL code of sequential process 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
                -- 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
                            if(s_sensor = "01") then
                                s_state <= SOUTH_GO;
                                s_cnt <= c_ZERO;
                            else
                                -- Move to the next state
                                s_state <= WEST_GO;
                                -- Reset local counter value
```

```
            s_cnt   <= c_ZERO;
        end if;
    end if;

when WEST_GO =>
    if (s_cnt < c_DELAY_4SEC) then
        s_cnt <= s_cnt + 1;
    else
        if(s_sensor = "10") then
            s_state <= WEST_GO;
            s_cnt <= c_ZERO;
        else
            s_state <= WEST_WAIT;
            s_cnt   <= c_ZERO;
        end if;
    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
        if(s_sensor = "10") then
            s_state <= WEST_GO;
            s_cnt <= c_ZERO;
        else
            s_state <= SOUTH_GO;
            s_cnt   <= c_ZERO;
        end if;
    end if;

when SOUTH_GO =>
    if (s_cnt < c_DELAY_4SEC) then
        s_cnt <= s_cnt + 1;
    else
        if(s_sensor = "01") then
            s_state <= SOUTH_GO;
            s_cnt <= c_ZERO;
        else
            s_state <= SOUTH_WAIT;
            s_cnt   <= c_ZERO;
        end if;
    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; -- Synchronous reset
        end if; -- Rising edge
    end process p_smart_traffic_fsm;
```