Labs - 08 - Traffic lights

Lab assignment

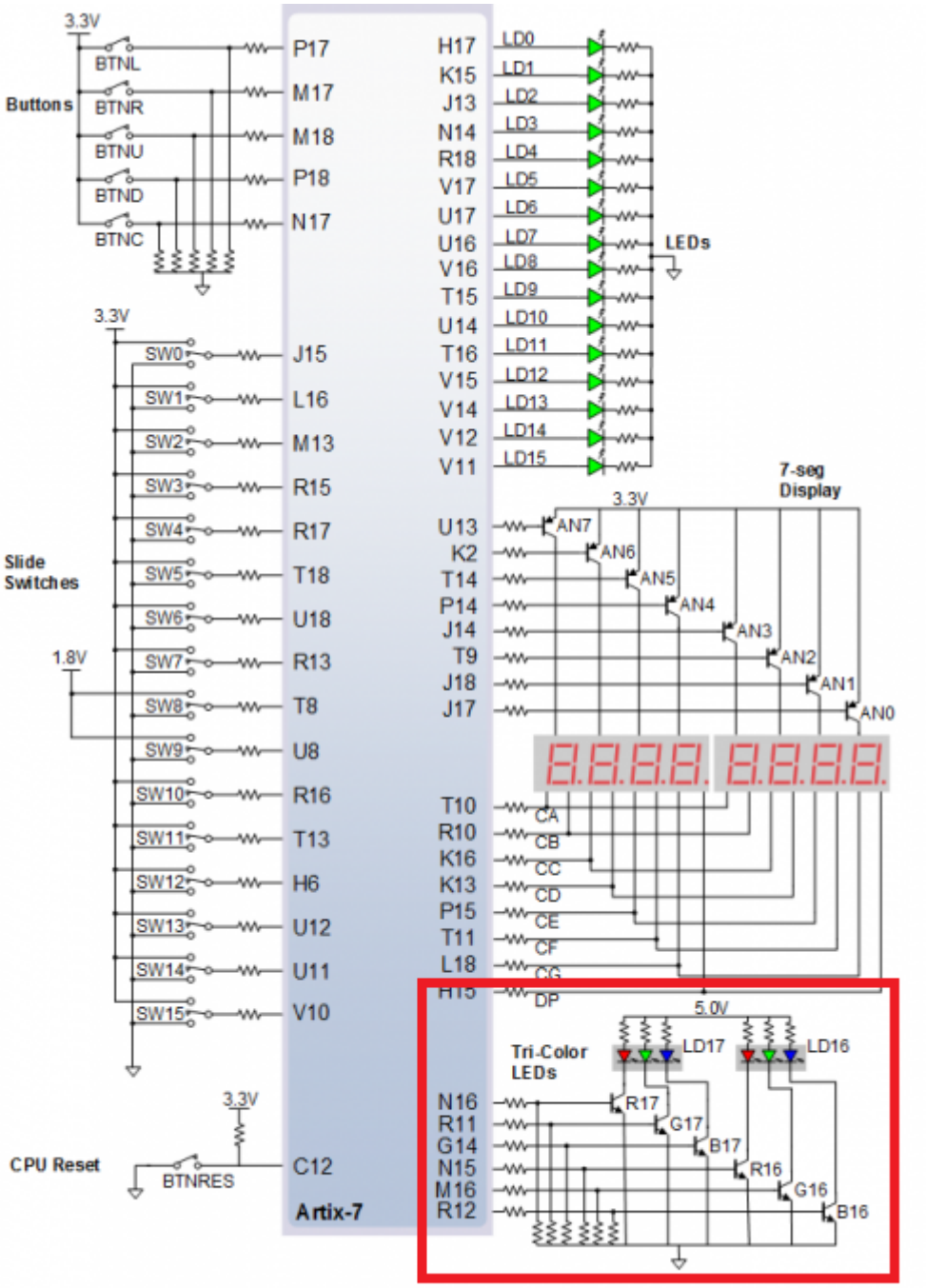
1. Preparation tasks (done before the lab at home). Submit:
    - Completed state table,
    - Figure with connection of RGB LEDs on Nexys A7 board and completed table with color settings.
2. Traffic light controller. Submit:
    - State diagram,
    - Listing of VHDL code of sequential process `p_traffic_fsm` with syntax highlighting,
    - Listing of VHDL code of combinatorial process `p_output_fsm` with syntax highlighting,
    - Screenshot(s) of the simulation, from which it is clear that controller works correctly.
3. Smart controller. Submit:
    - State table,
    - State diagram,
    - Listing of VHDL code of sequential process `p_smart_traffic_fsm` with syntax highlighting.

1. Preparation tasks

1.1. Completed state table

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

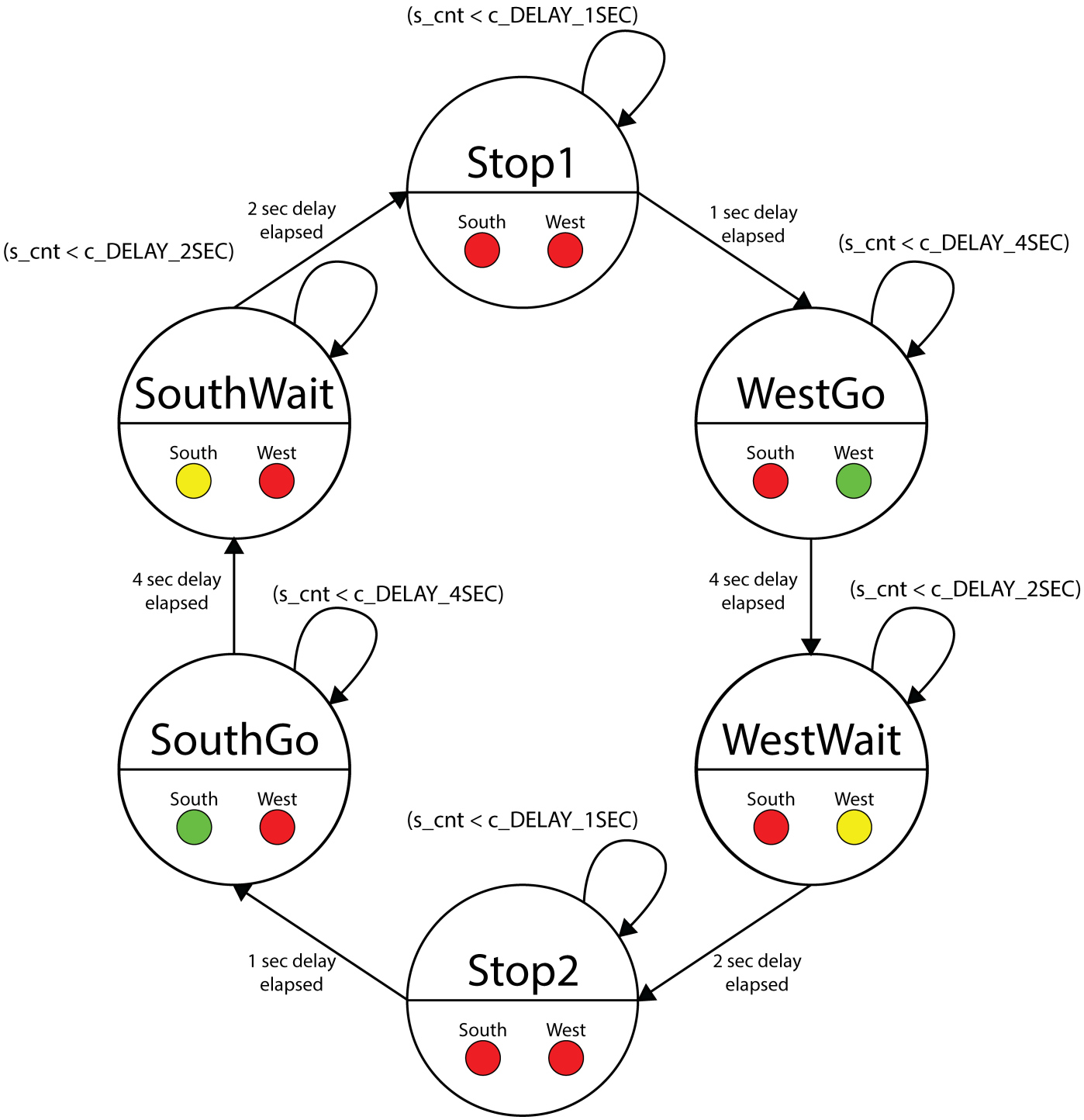
1.2. Figure with connection of RGB LEDs on Nexys A7 board and completed table with color settings



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`

2. Traffic light controller

2.1. State diagram



2.2. Listing of VHDL code of sequential process `p_traffic_fsm` with syntax highlighting

```
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
```

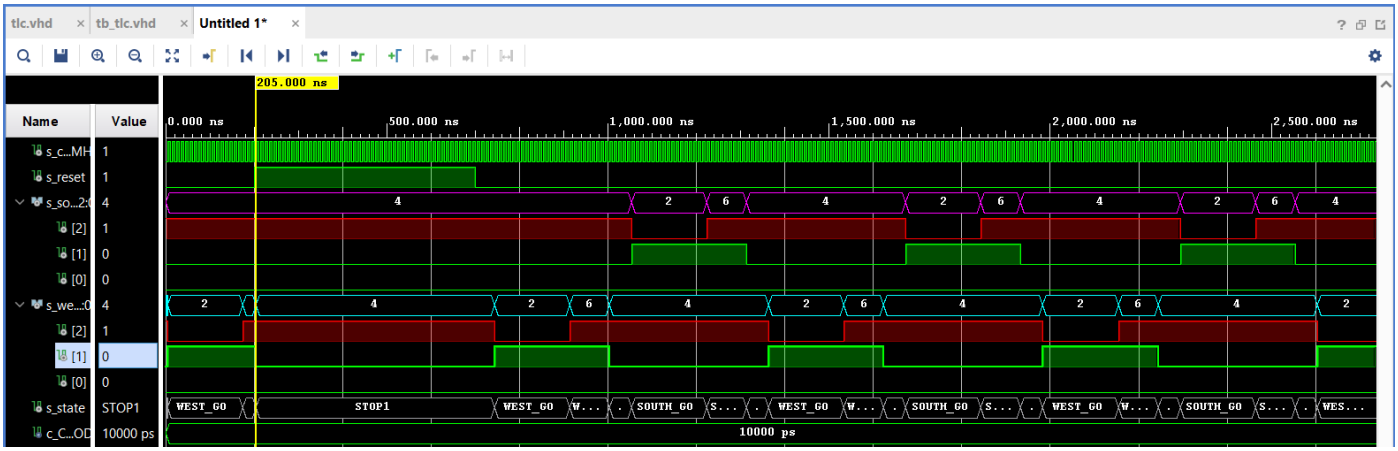
```
        -- If the current state is STOP1, then wait 1 sec
        -- and move to the next GO_WAIT state.
        when STOP1 =>
            -- Count up to c_DELAY_1SEC
            if (s_cnt < c_DELAY_1SEC) then
                s_cnt <= s_cnt + 1;
            else
                -- Move to the next state
                s_state <= WEST_GO;
                -- Reset local counter value
                s_cnt   <= c_ZERO;
            end if;

        when WEST_GO =>
            if (s_cnt < c_DELAY_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 =>
```

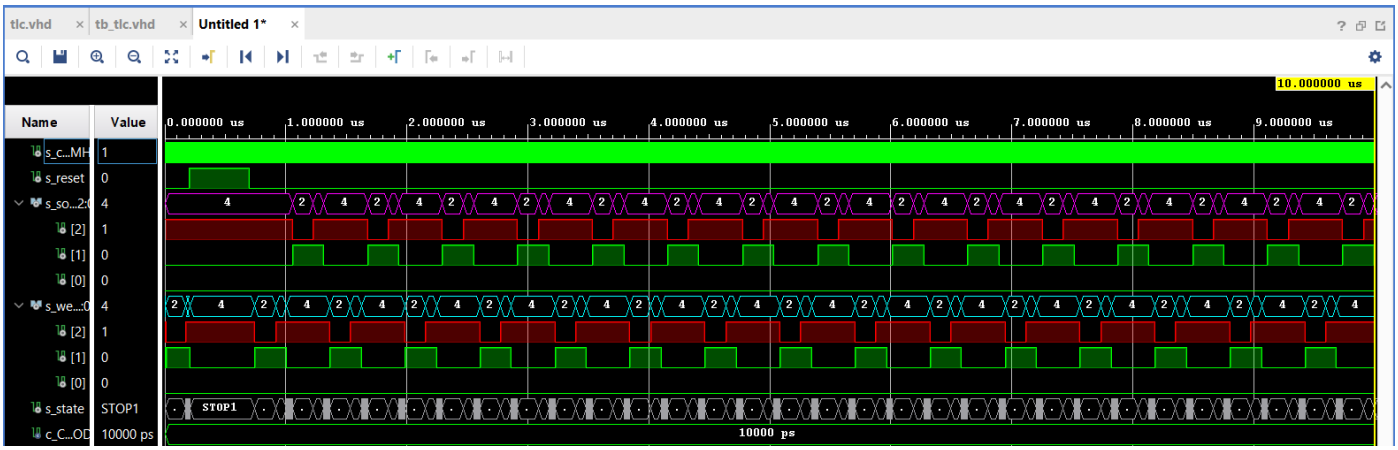
```
                    s_state <= STOP1;

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

2.3. Listing of VHDL code of combinatorial process `p_output_fsm` with syntax highlighting

```
p_output_fsm : process(s_state)
begin
    case s_state is
        when STOP1 =>
            south_o <= "100";   -- Red (RGB = 100)
            west_o  <= "100";   -- Red (RGB = 100)

        when WEST_GO =>
            south_o <= "100";   -- Red (RGB = 100)
            west_o  <= "010";   -- Green (RGB = 010)

        when WEST_WAIT =>
            south_o <= "100";   -- Red (RGB = 100)
            west_o  <= "110";   -- Yellow (RGB = 110)

        when STOP2 =>
            south_o <= "100";   -- Red (RGB = 100)
            west_o  <= "100";   -- Red (RGB = 100)

        when SOUTH_GO =>
            south_o <= "010";   -- Green (RGB = 010)
            west_o  <= "100";   -- Red (RGB = 100)

        when SOUTH_WAIT =>
            south_o <= "110";   -- Yellow (RGB = 110)
            west_o  <= "100";   -- Red (RGB = 100)

        when others =>
            south_o <= "100";   -- Red
            west_o  <= "100";   -- Red
    end case;
end process p_output_fsm;
```

2.4. Screenshot(s) of the simulation, from which it is clear that controller works correctly



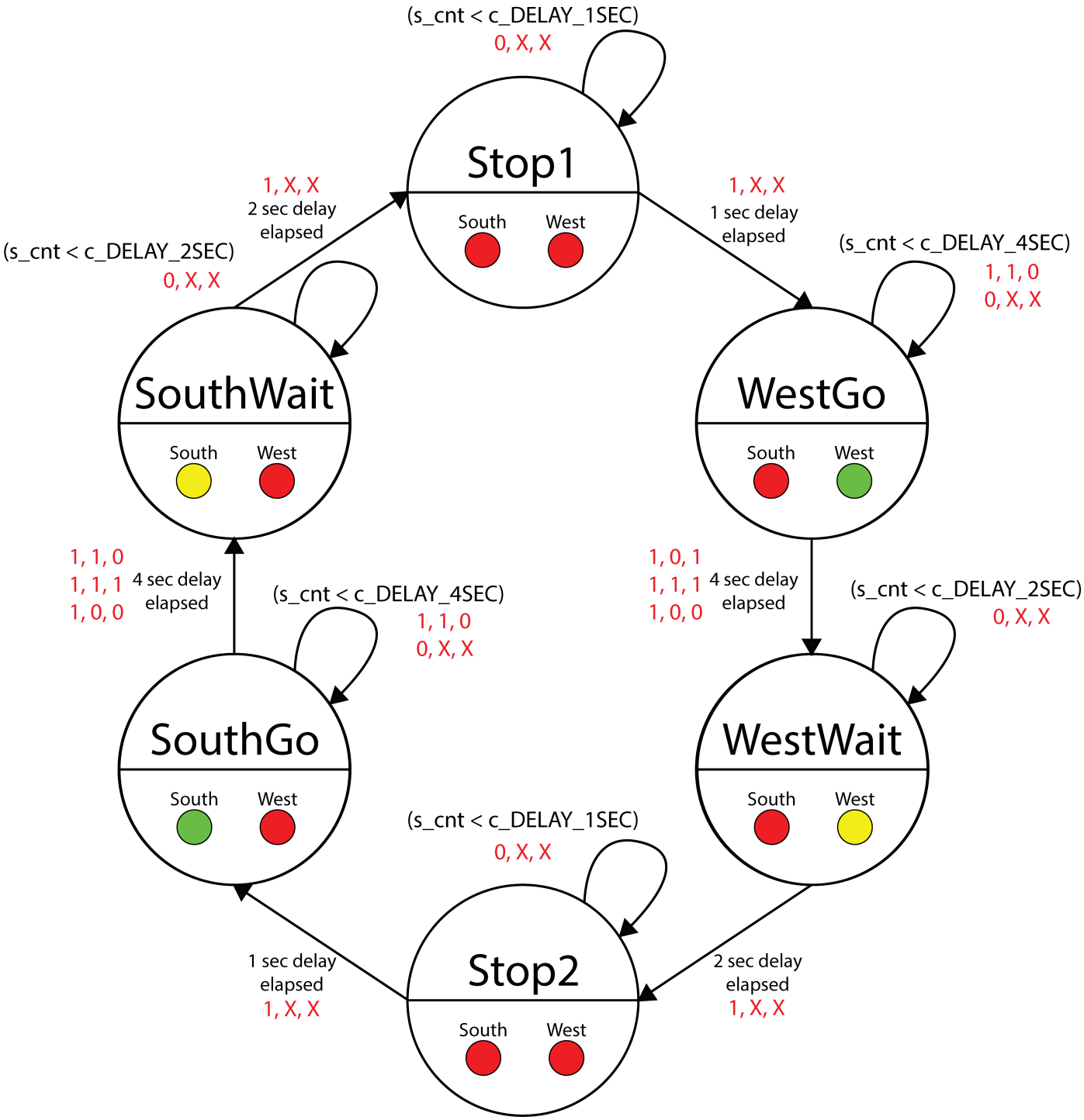


3. Smart controller

3.1. State table

Current state	Direction South	Direction West	Delay	Input
STOP1	red	red	1 sec	Unchanged
WEST_GO	red	green	4 sec	0, X, X or 1, 1, 0 go to WEST_GO else go to WEST_WAIT
WEST_WAIT	red	yellow	2 sec	Unchanged
STOP2	red	red	1 sec	Unchanged
SOUTH_GO	green	red	4 sec	0, X, X or 1, 0, 1 go to SOUTH_GO else go to SOUTH_WAIT
SOUTH_WAIT	yellow	red	2 sec	Unchanged

3.2. State diagram



3.3. Listing of VHDL code of sequential process `p_smart_traffic_fsm` with syntax highlighting

Zjednodušení kódu by se dalo provést odstrněním STOP1 a STOP2 a následnou úpravou kódu

```
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
                    -- 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_4SEC) then
                    s_cnt <= s_cnt + 1;
                else
                    if (sens_west = '1' and sens_south = '0') then
                        s_state <= WEST_GO;
                    else
                        s_state <= WEST_WAIT;
                    end if;
                    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
                    if (sens_west = '0' and sens_south = '1') then
                        s_state <= SOUTH_GO;
                    else
                        s_state <= SOUTH_WAIT;
                    end if;
                    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 others =>
                    s_state <= STOP1;

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