Lab assigment 08 - Traffic lights

Prep. task -

Completed state table

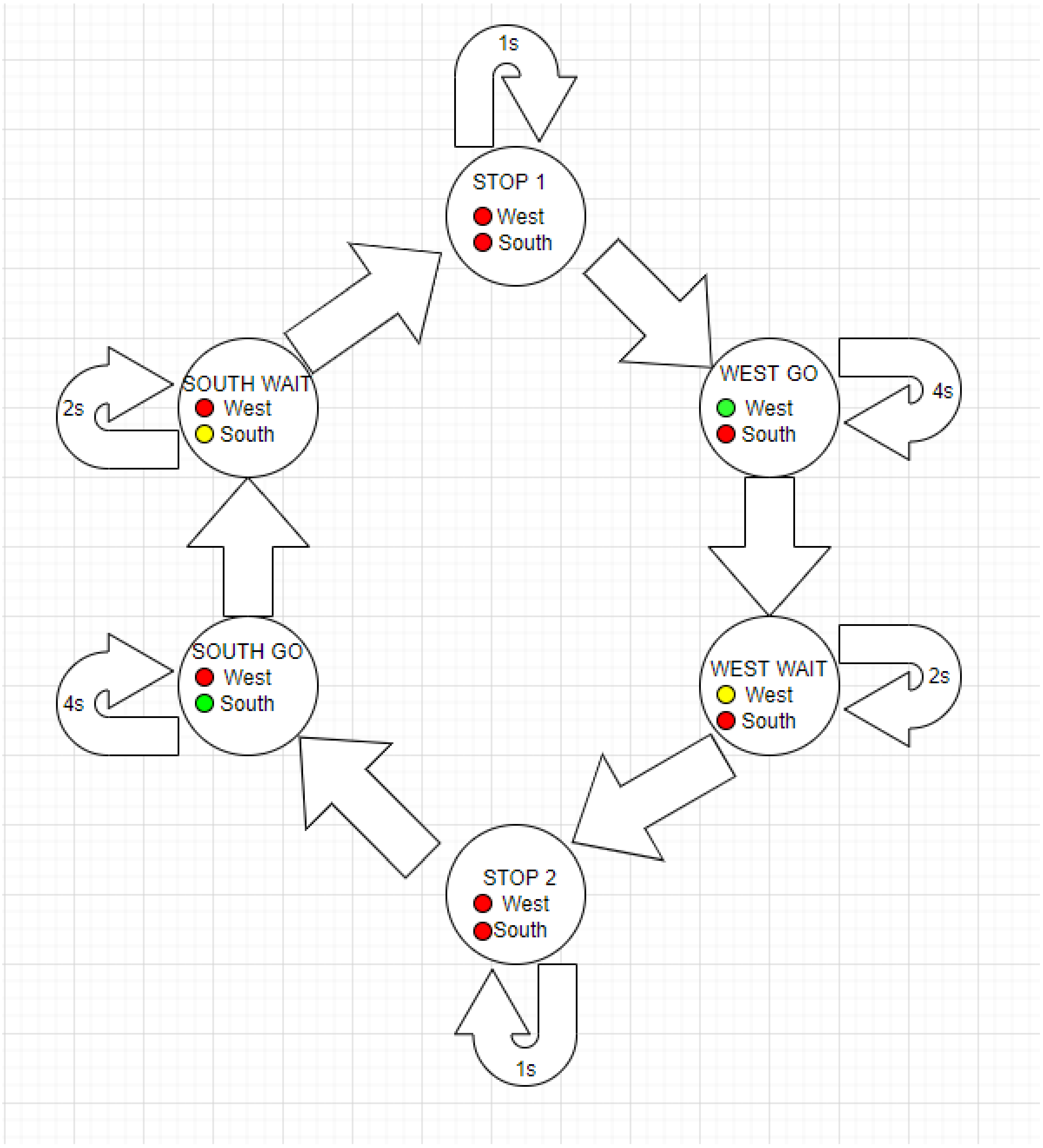
Input P	`0`	`0`	`1`	`1`	`0`	`1`	`0`	`1`	`1`	`1`	`1`	`0`
Clock	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
State	A	A	B	C	C	D	A	B	C	D	B	B
Output R	`0`	`0`	`0`	`0`	`0`	`1`	`0`	`0`	`0`	`1`	`0`	`0`

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`

Traffic light controller

State diagram



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

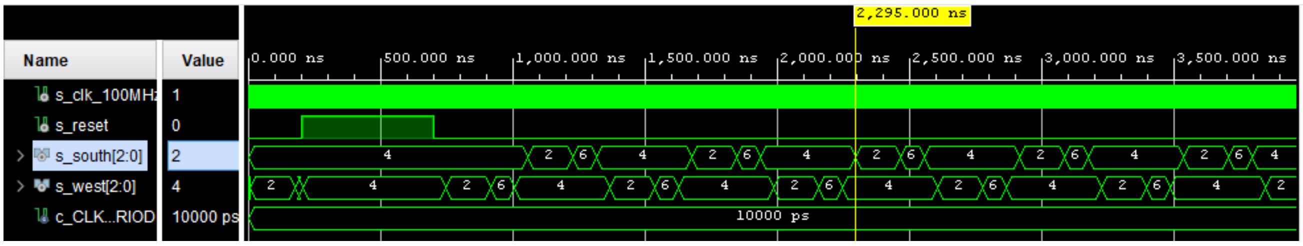
```
p_traffic_fsm : process(clk)
    begin
        if rising_edge(clk) then
            if (reset = '1') then       -- Synchronous reset
                s_state <= STOP1 ;      -- Set initial state
                s_cnt   <= c_ZERO;      -- Clear all bits

            elsif (s_en = '1') then
                -- Every 250 ms, CASE checks the value of the s_state
                -- variable and changes to the next state according
```

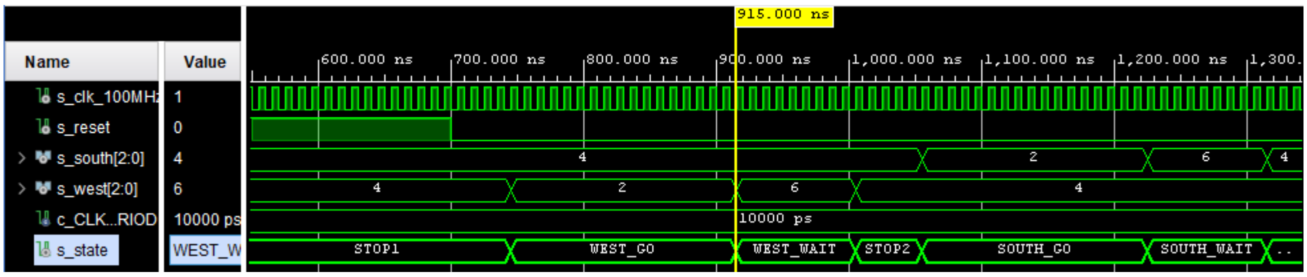
```vhdl
            -- to the delay value.
            case s_state is

                -- 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 =>
                    -- WRITE YOUR CODE HERE
                    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 =>
                    -- WRITE YOUR CODE HERE
                    if (s_cnt < c_DELAY_2SEC) then
                        s_cnt <= s_cnt + 1;
                    else
                        s_state <= STOP2;
                        s_cnt   <= c_ZERO;
                    end if;

                when STOP2 =>
                    -- WRITE YOUR CODE HERE
                    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 =>
                    -- WRITE YOUR CODE HERE
                    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 =>
                    -- WRITE YOUR CODE HERE
                    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;
```

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

```vhdl
p_output_fsm : process(s_state)
    begin
        case s_state is
            when STOP1 =>
                south_o <= c_RED;
                west_o  <= c_RED;

            when WEST_GO =>
                -- WRITE YOUR CODE HERE
                south_o <= c_RED;
                west_o  <= c_GREEN;

            when WEST_WAIT =>
                -- WRITE YOUR CODE HERE
                south_o <= c_RED;
                west_o  <= c_YELLOW;

            when STOP2 =>
                -- WRITE YOUR CODE HERE
                south_o <= c_RED;
                west_o  <= c_RED;

            when SOUTH_GO =>
                -- WRITE YOUR CODE HERE
                south_o <= c_GREEN;
                west_o  <= c_RED;

            when SOUTH_WAIT =>
                -- WRITE YOUR CODE HERE
                south_o <= c_YELLOW;
                west_o  <= c_RED;

            when others =>
                south_o <= c_RED;
                west_o  <= c_RED;
        end case;
    end process p_output_fsm;
```

Screenshot of the simulation, from which it is clear that controller works correctly





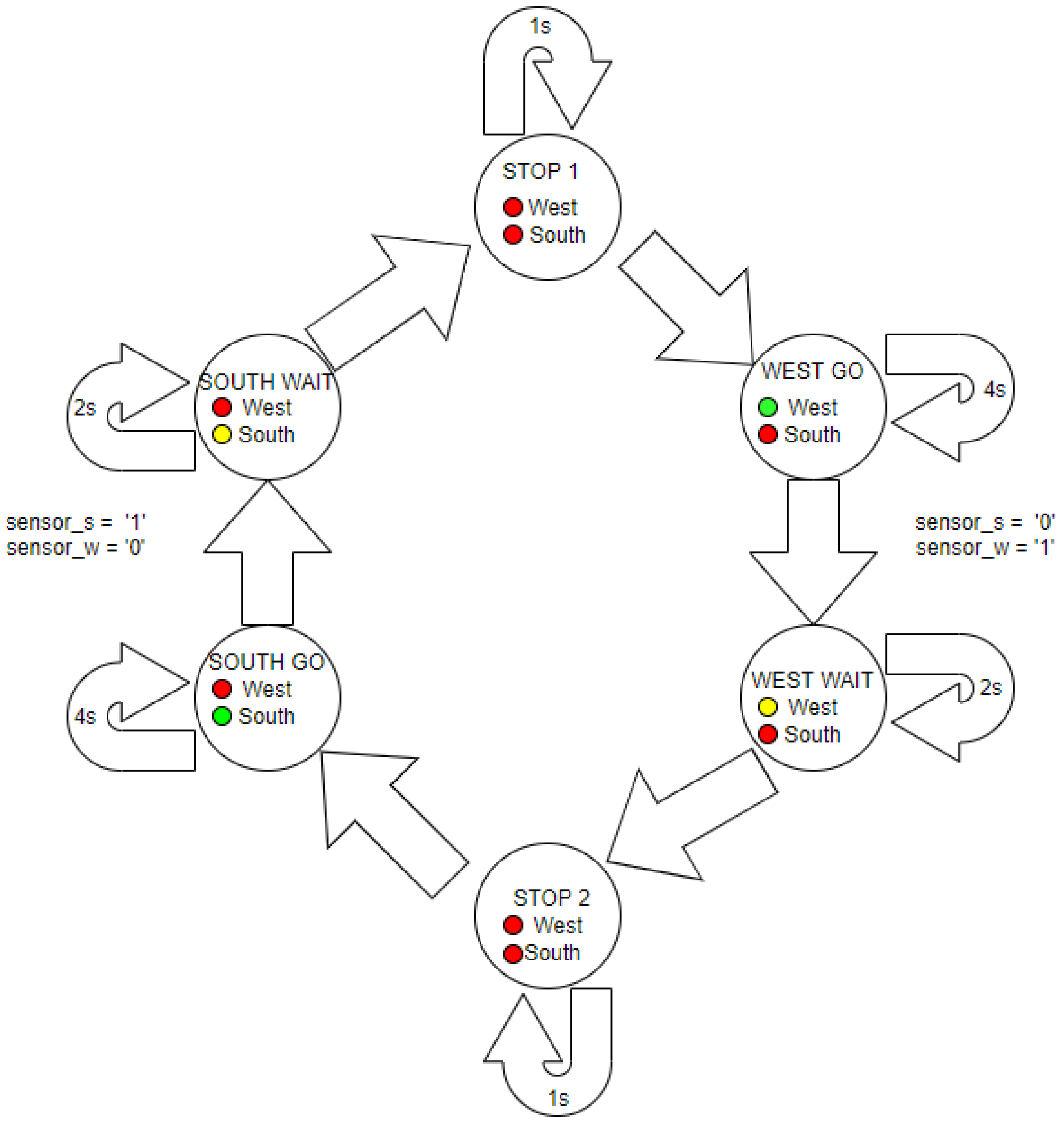
Smart controller

State table

Current state	Direction South	Direction West	Delay	Info
`STOP1`	red	red	1 sec	d/c

Current state	Direction South	Direction West	Delay	Info
`WEST_GO`	red	green	4 sec	sensor_w = '1' && sensor_o = '0'
`WEST_WAIT`	red	yellow	2 sec	d/c
`STOP2`	red	red	1 sec	d/c
`SOUTH_GO`	green	red	4 sec	sensor_w = '0' && sensor_o = '1'
`SOUTH_WAIT`	yellow	red	2 sec	d/c

State diagram



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

```
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 =>
    -- WRITE YOUR CODE HERE
    if(s_cnt < c_DELAY_4SEC) then
        s_cnt <= s_cnt + 1;
    else
        if(sensor_s = '0' and sensor_w = '1') then
        s_state <= WEST_GO;

        else
        s_state <= WEST_WAIT;
        s_cnt   <= c_ZERO;

        end if;
    end if;

when WEST_WAIT =>
    -- WRITE YOUR CODE HERE
    if (s_cnt < c_DELAY_2SEC) then
        s_cnt <= s_cnt + 1;
    else
        s_state <= STOP2;
        s_cnt   <= c_ZERO;
    end if;

when STOP2 =>
    -- WRITE YOUR CODE HERE
    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 =>
    -- WRITE YOUR CODE HERE
    if (s_cnt < c_DELAY_4SEC) then
        s_cnt <= s_cnt + 1;
```

```
                    else
                        if (sensor_s = '1' and sensor_w = '0') then
                        s_state <= SOUTH_GO;

                        else
                        s_state <= SOUTH_WAIT;
                        s_cnt   <= c_ZERO;

                        end if;
                    end if;

                when SOUTH_WAIT =>
                    -- WRITE YOUR CODE HERE
                    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;
```