8. Traffic light controller

Lab assignment

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

3. Smart controller. Submit:

   - [x] State table,
   - [x] State diagram,
   - [x] Listing of VHDL code of sequential process `p_smart_traffic_fsm` with syntax highlighting.

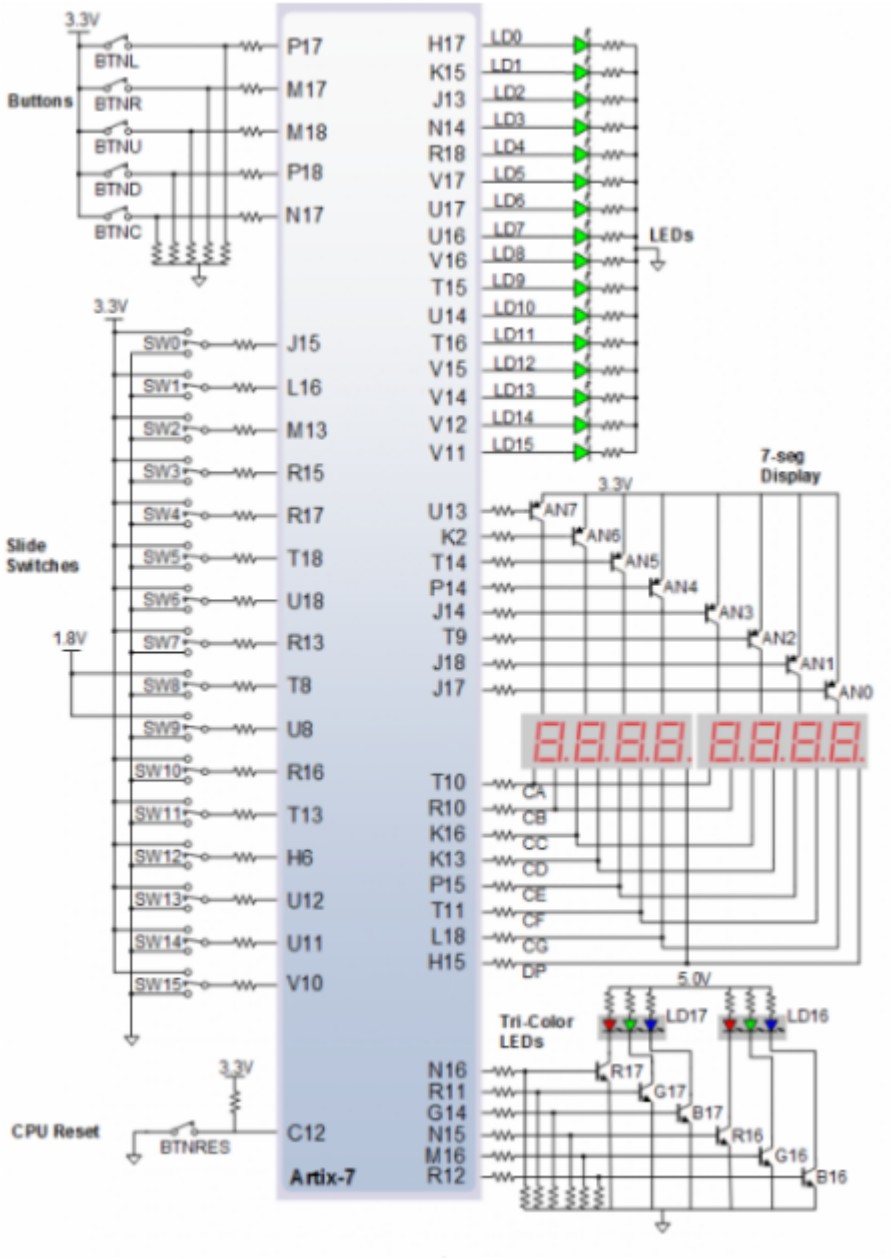
1. Preparation tasks (done before the lab at home)

Read the article Implementing a Finite State Machine in VHDL (parts **A Bit of Background** and **The Finite State Machine**) and understand what an FSM is.

Table with the state names and output values accoding to the given inputs. Let the reset has just been applied.

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

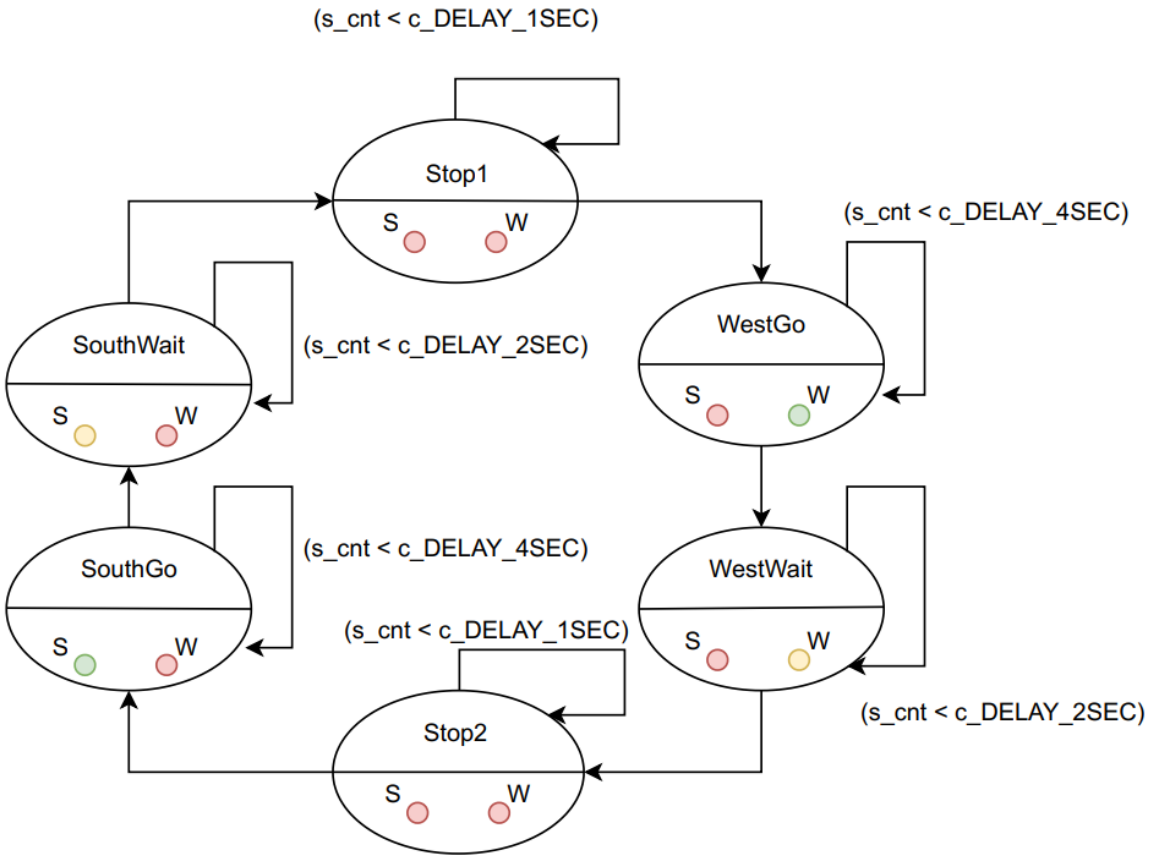
See schematic or reference manual of the Nexys board and see table with the connection of two RGB LEDs.



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

State diagram



Listing of VHDL code of sequential process `p_traffic_fsm`

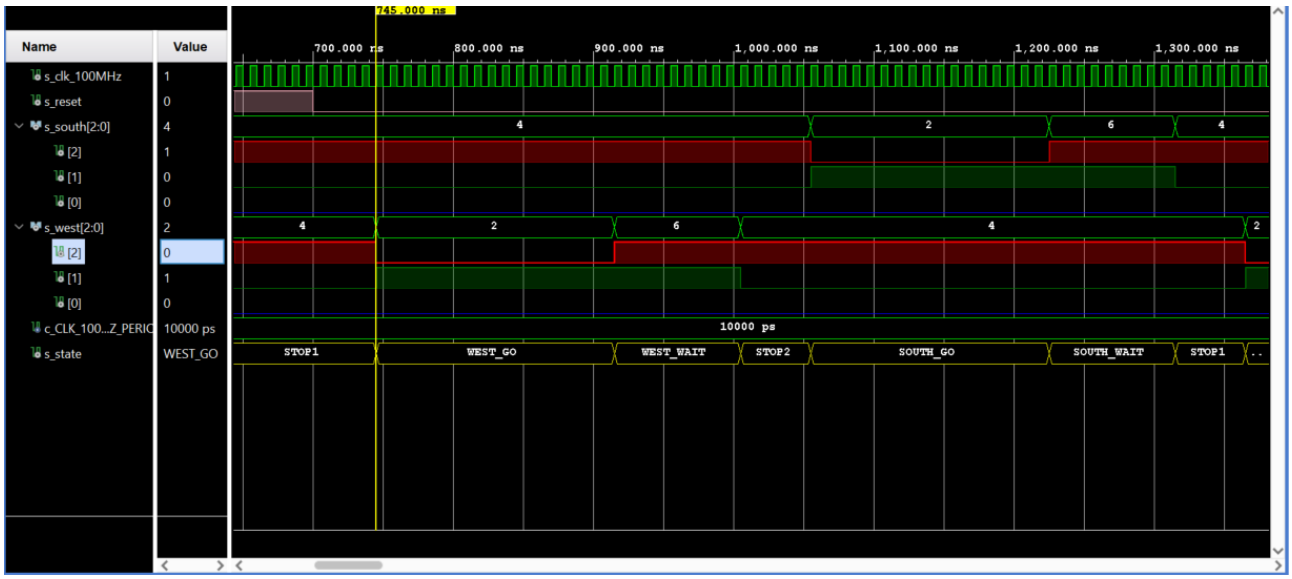
```
p_traffic_fsm : process(clk)
    begin
        if rising_edge(clk) then
            if (reset = '1') then       -- Synchronous reset
                s_state <= STOP1 ;      -- Set initial state
                s_cnt   <= c_ZERO;      -- Clear all bits

            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 =>
                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;
            when others =>
                s_state <= STOP1;

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

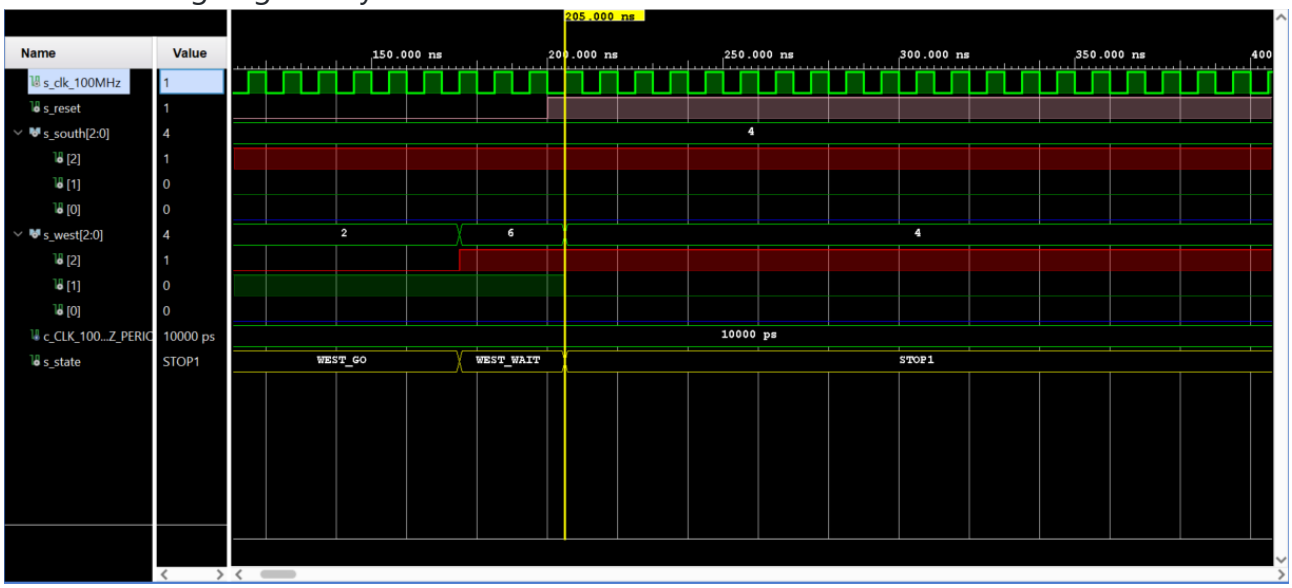
Listing of VHDL code of combinatorial process `p_output_fsm`

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

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



Detail of rising edge for synchronous reset



3. Smart controller

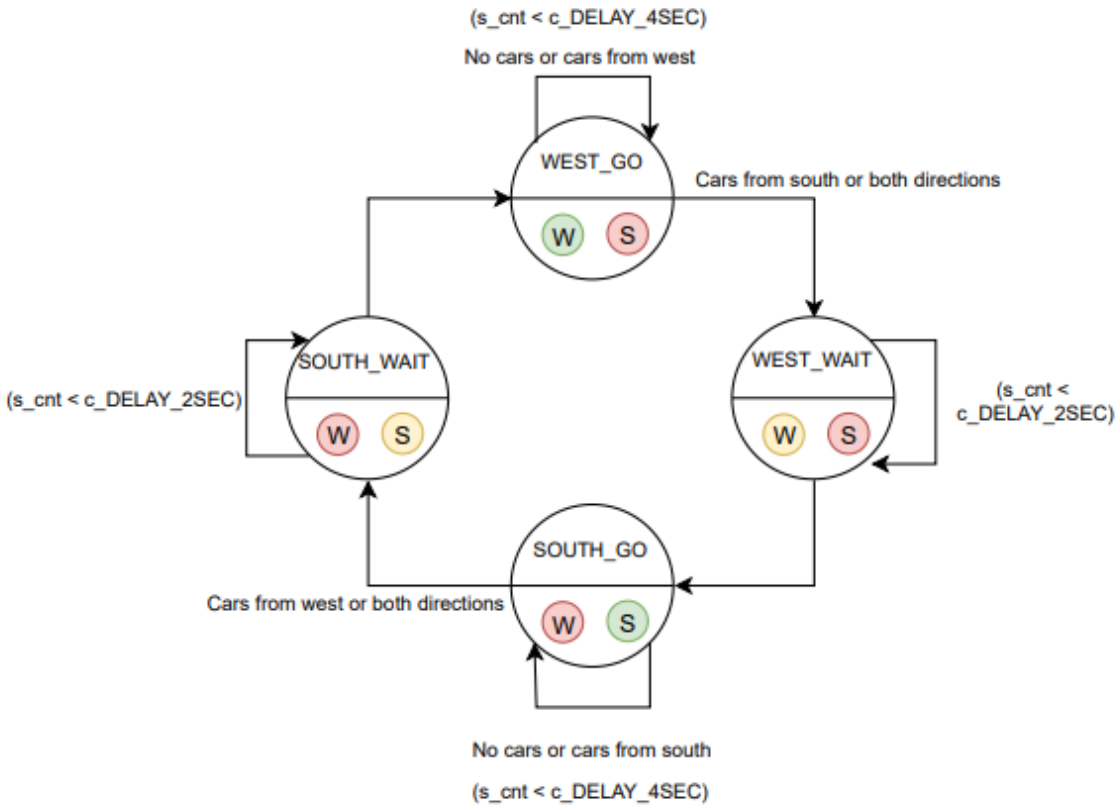
States we will need

Current state	Direction South	Direction West	Delay
WEST_GO	red	green	4 sec
WEST_WAIT	red	yellow	2 sec
SOUTH_GO	green	red	4 sec
SOUTH_WAIT	yellow	red	2 sec

State table with conditions for sensors

Current state\Input	No cars	Cars to West	Carst to South	Cars both directions
WEST_GO	WEST_GO	WEST_GO	WEST_WAIT	WEST_WAIT
WEST_WAIT	SOUTH_GO	SOUTH_GO	SOUTH_GO	SOUTH_GO
SOUTH_GO	SOUTH_GO	SOUTH_WAIT	SOUTH_GO	SOUTH_WAIT
SOUTH_WAIT	WEST_GO	WEST_GO	WEST_GO	WEST_GO

State diagram

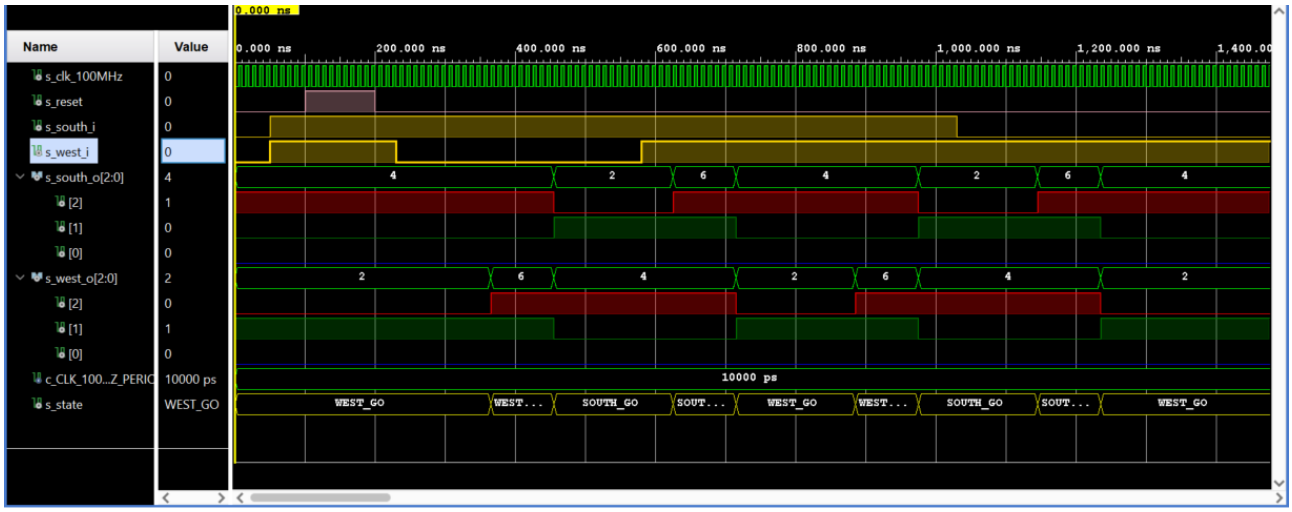


Listing of VHDL code of sequential process `p_smart_traffic_fsm` (longer version)

```
p_smart_traffic_fsm : process(clk)
    begin
        if rising_edge(clk) then
            if (reset = '1') then       -- Synchronous reset
                s_state <= WEST_GO ;    -- 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 WEST_GO =>
                        if (s_cnt < c_DELAY_4SEC) then
                            s_cnt <= s_cnt + 1;
                        else
                            if (south_i = '0' and west_i = '0') then
                                s_state <= WEST_GO;
                            elsif (south_i = '0' and west_i = '1') then
                                s_state <= WEST_GO;
                            elsif (south_i = '1' and west_i = '0') then
                                s_state <= WEST_WAIT;
                            elsif(south_i = '1' and west_i = '1') then
```

```
                        s_state <= WEST_WAIT;
                    else
                        s_state <= WEST_GO;
                    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 <= 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 (south_i = '0' and west_i = '0') then
                        s_state <= SOUTH_GO;
                    elsif (south_i = '0' and west_i = '1') then
                        s_state <= SOUTH_WAIT;
                    elsif (south_i = '1' and west_i = '0') then
                        s_state <= SOUTH_GO;
                    elsif(south_i = '1' and west_i = '1') then
                        s_state <= SOUTH_WAIT;
                    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 <= WEST_GO;
                    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 <= WEST_GO;

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

Simulation


Name
Value
s_clk_100MHz
s_reset
s_south_i
s_west_i
s_south_o[2:0]
[2]
[1]
[0]
s_west_o[2:0]
[2]
[1]
[0]
c_CLK_100...Z_PERIO
s_state
0
0
0
0
4
1
0
0
2
0
1
0
10000 ps
WEST_GO
0.000 ns
200.000 ns
400.000 ns
600.000 ns
800.000 ns
1,000.000 ns
1,200.000 ns
10000 ps
WEST_GO
WEST...
SOUTH_GO
SOUT...
WEST_GO
WEST...
SOUTH_GO
SOUT...
WEST_GO