My DE1 repisitory link

DE1 - Jiří Navrátil 222721

Traffic light controller

Table with the state names and output values

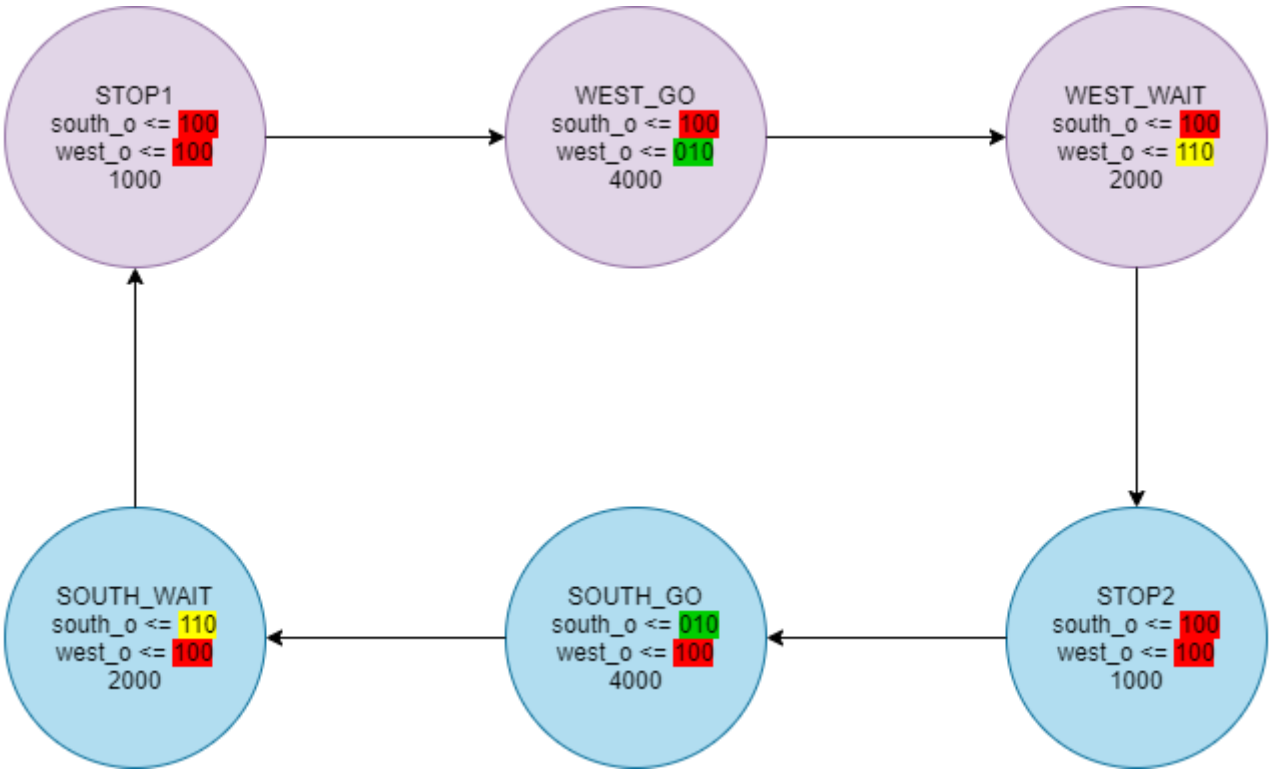
Input P	0	0	1	1	0	1	0	1	1	1	1	0	0	1	1	1
State	A	A	B	C	C	D	A	B	C	D	B	B	B	C	D	B
Output R	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0

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

Traffic light controller

State diagram



Traffic FSM process

```
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_GO
                    if (s_cnt < c_DELAY_GO) 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_WAIT
                    if (s_cnt < c_DELAY_WAIT) 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;
```

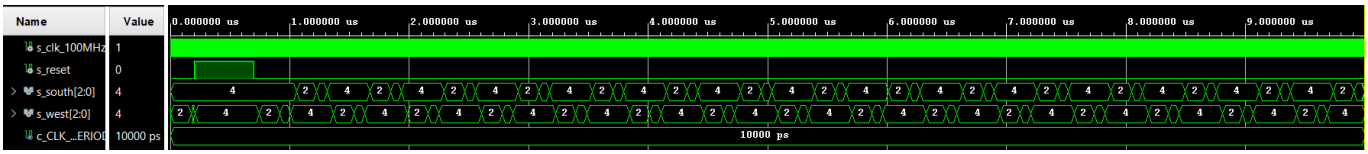
```
                    else
                        -- Move to the next state
                        s_state <= SOUTH_GO;
                        -- Reset local counter value
                        s_cnt   <= c_ZERO;
                    end if;

                when SOUTH_GO =>
                    -- Count up to c_DELAY_GO
                    if (s_cnt < c_DELAY_GO) 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_WAIT
                    if (s_cnt < c_DELAY_WAIT) 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;
```

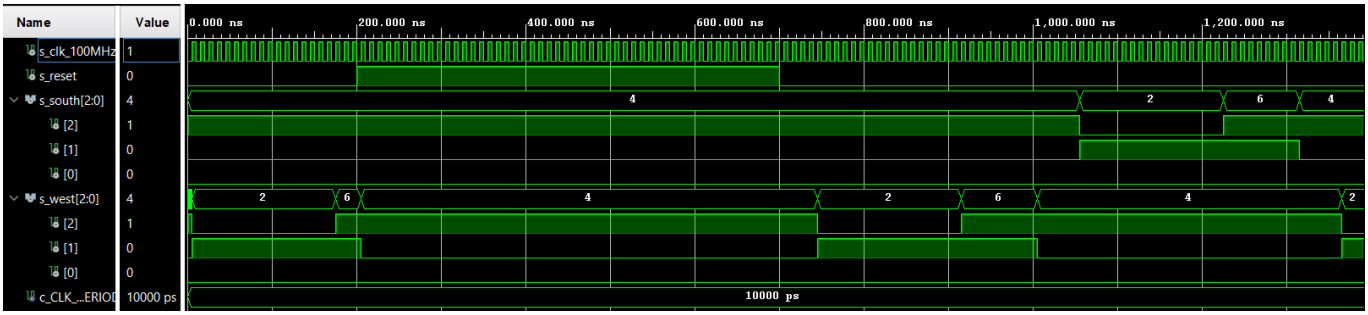
Output FSM process

```
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;
```

Result





Smart controller

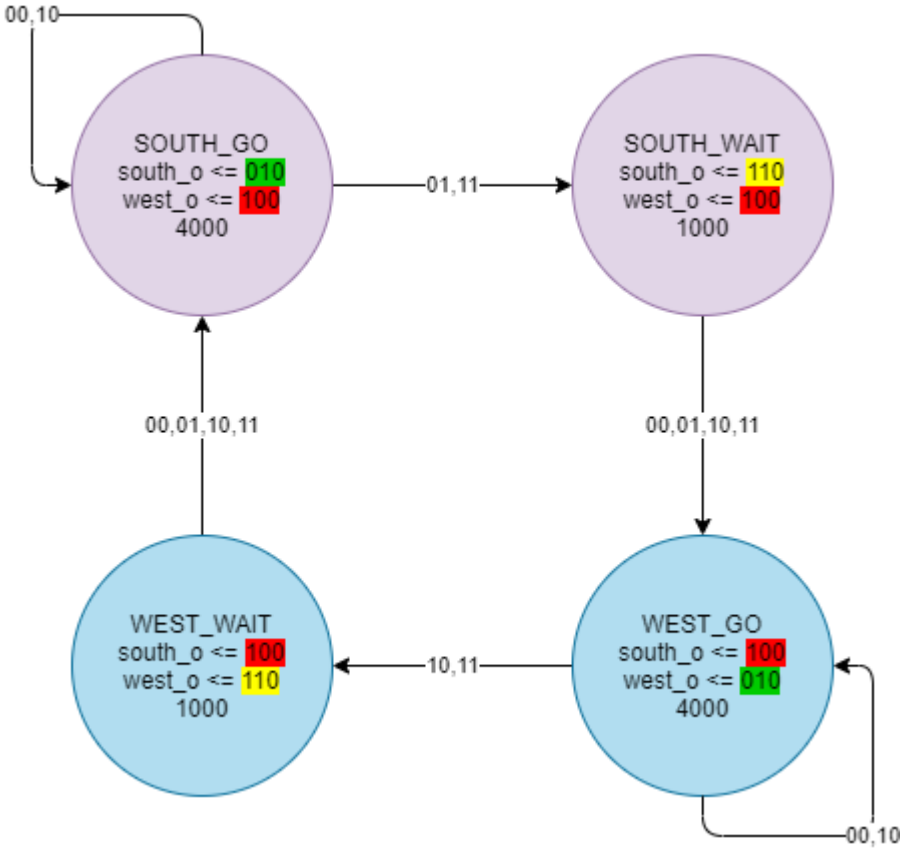
State table

Current state	Direction South	Direction West	Delay
SOUTH_GO	green	red	4 sec
SOUTH_WAIT	yellow	red	1 sec
WEST_GO	red	green	4 sec
WEST_WAIT	red	yellow	1 sec

Sensor '00' - no cars	Sensor '01' - cars on west	Sensor '10' - cars on south	Sensor '11' - cars on both roads
SOUTH_GO	SOUTH_WAIT	SOUTH_GO	SOUTH_WAIT

Sensor '00' - no cars	Sensor '01' - cars on west	Sensor '10' - cars on south	Sensor '11' - cars on both roads
WEST_GO	WEST_GO	WEST_GO	WEST_GO
WEST_GO	WEST_GO	WEST_WAIT	WEST_WAIT
SOUTH_GO	SOUTH_GO	SOUTH_GO	SOUTH_GO

State diagram



Smart traffic FSM process

```
p_smart_traffic_fsm : process(clk)
begin
    if rising_edge(clk) then
        if (reset = '1') then       -- Synchronous reset
            s_state <= SOUTH_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

                -- If the current state is STOP1, then wait 1 sec
                -- and move to the next GO_WAIT state.
```

```
                when SOUTH_GO =>
                    -- Count up to c_DELAY_GO
                    if (s_cnt < c_DELAY_GO and (sensor = "00" or sensor = "10"))
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_WAIT
                    if (s_cnt < c_DELAY_WAIT) 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_GO
                    if (s_cnt < c_DELAY_GO and (sensor = "00" or sensor = "01"))
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_WAIT
                    if (s_cnt < c_DELAY_WAIT) 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;

                -- 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;
```

Result