xmanas07 / **Digital-electronics-1**

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main ▾ ···

Digital-electronics-1 / Labs / 08-traffic_lights /

xmanas07 … 36 seconds ago

..

images	2 hours ago
traffic	36 seconds ago
README.md	4 minutes ago

README.md

Digital-electronics-1

úkol 1: Preparation tasks

Filled out 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

Connection of RGB LEDs on NExys A7 board

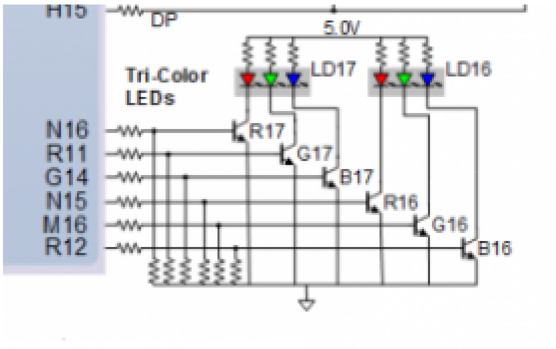
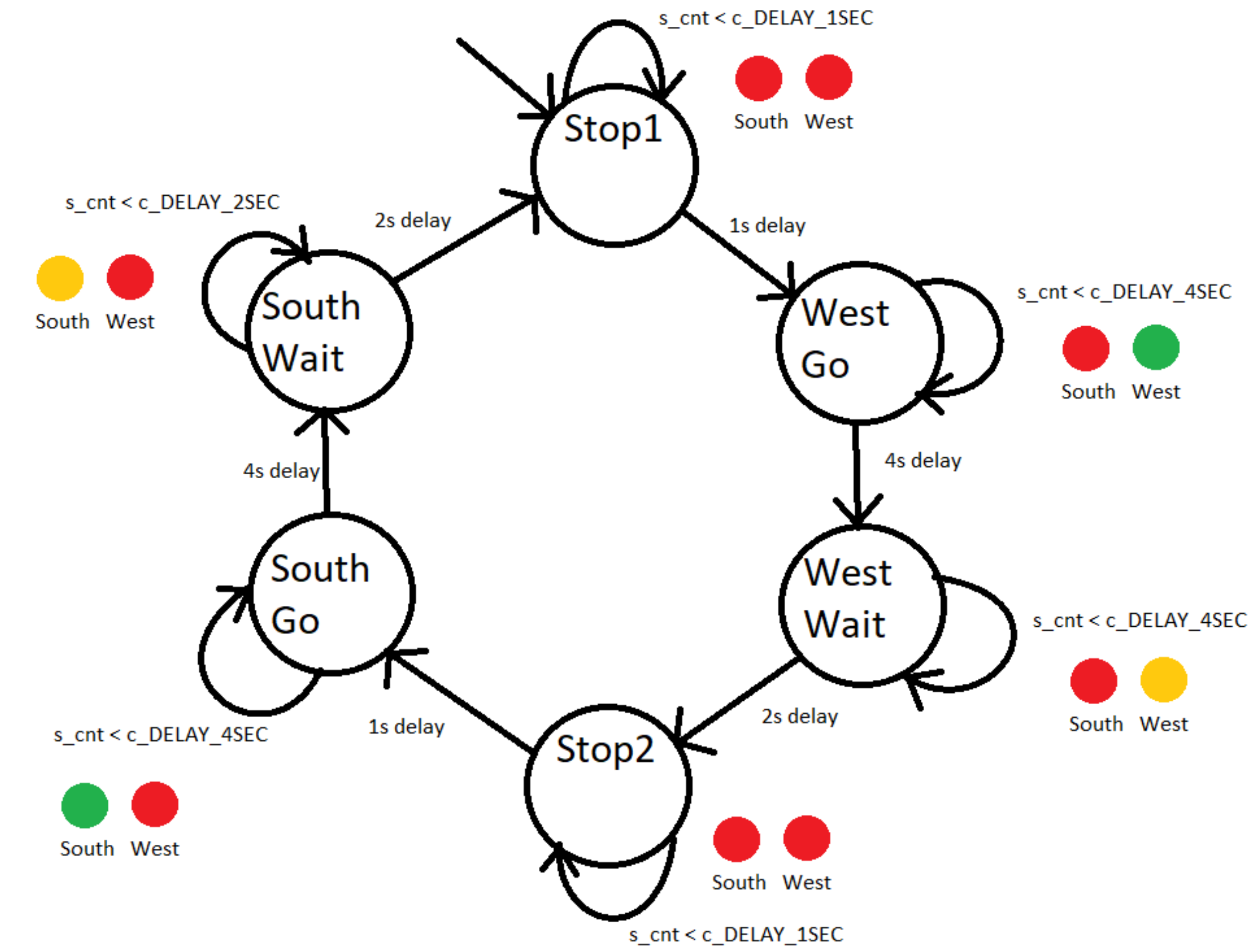


Figure with connection of RGB LEDs on Nexys A7 board

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`

úkol 2: Traffic light controller

State diagram



p_traffic_fsm process (`tlc`)

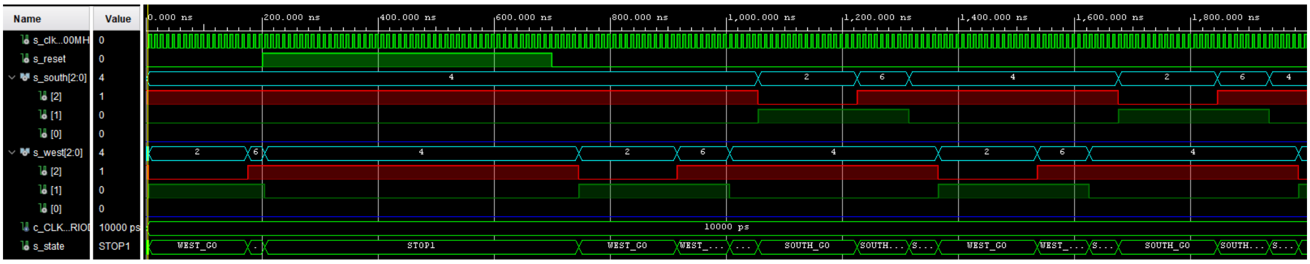
```
p_traffic_fsm : process(clk)
begin
    if rising_edge(clk) then
        if (reset = '1') then       -- Synchronous reset
            s_state <= STOP1 ;      -- Set initial state
            s_cnt   <= c_ZERO;      -- Clear all bits

        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_4SEC
                    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_2SEC
                    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_4SEC
                        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_2SEC
                        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;
```

p_output_fsm process (`tlc`)

```
p_output_fsm : process(s_state)
    begin
        case s_state is
            when STOP1 =>
                south_o <= c_RED;    -- RED (RGB = 100)
                west_o  <= c_RED;
            when WEST_GO =>
                south_o <= c_RED;
                west_o  <= c_GREEN;  -- GREEN (RGB = 010)

                -- WRITE YOUR CODE HERE
            when WEST_WAIT =>
                south_o <= c_RED;
                west_o  <= c_YELLOW; -- YELLOW (RGB = 110)

            when STOP2 =>
                south_o <= c_RED;
                west_o  <= c_RED;

            when SOUTH_GO =>
                south_o <= c_GREEN;
                west_o  <= c_RED;

            when SOUTH_WAIT =>
                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;
```

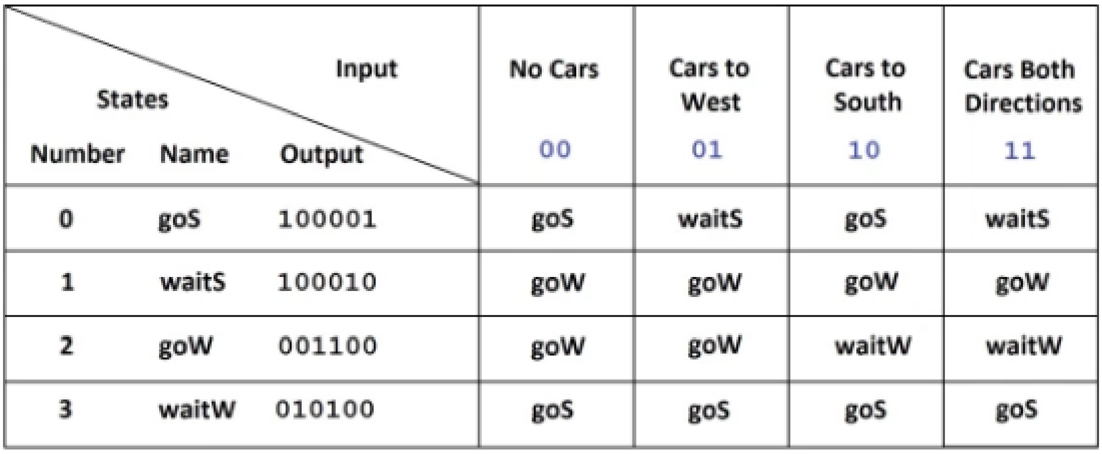
Screenshot with waveforms



Name	Value
s_clk...00MHz	0
s_reset	0
s_south[2:0]	4
[2]	1
[1]	0
[0]	0
s_west[2:0]	4
[2]	1
[1]	0
[0]	0
c_CLK...RIOD	10000 ps
s_state	STOP1

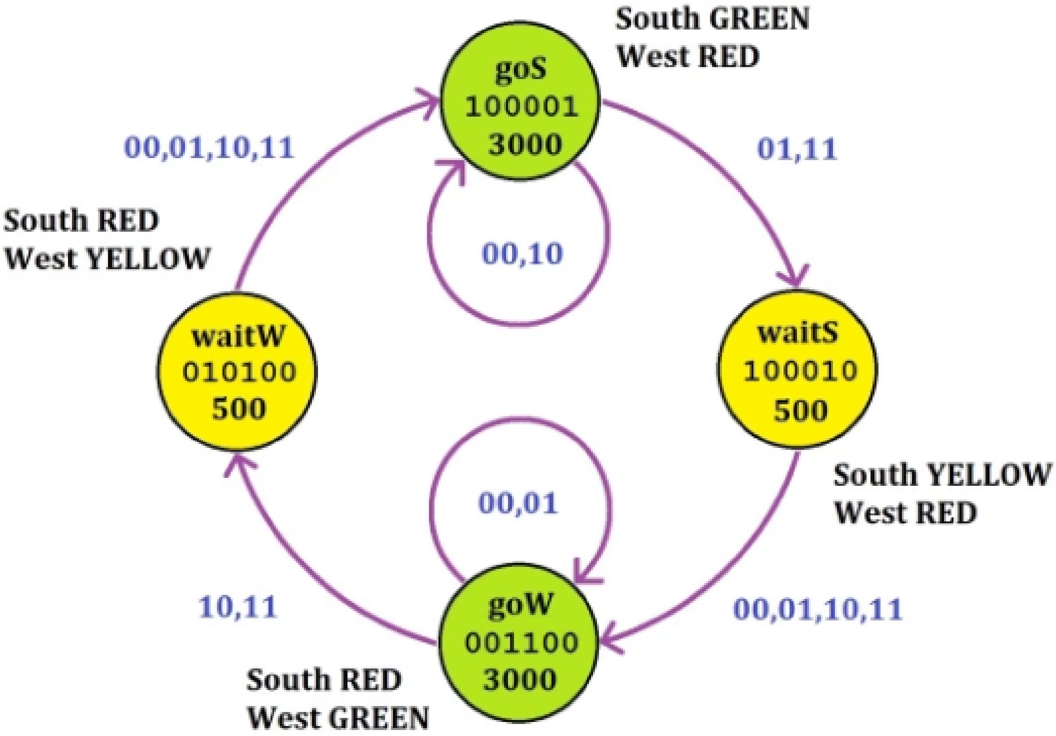
úkol 3: Smart Controller

State table



States		Input	No Cars	Cars to West	Cars to South	Cars Both Directions
Number	Name	Output	00	01	10	11
0	goS	100001	goS	waitS	goS	waitS
1	waitS	100010	goW	goW	goW	goW
2	goW	001100	goW	goW	waitW	waitW
3	waitW	010100	goS	goS	goS	goS

State diagram



p_smart_traffic_fsm process (stlc)

```
p_smart_traffic_fsm : process(clk)
    begin
        if rising_edge(clk) then
            if (reset = '1') then       -- Synchronous reset
                s_state <= goS ;      -- 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 goS =>
   if (cars = "00" or cars = "10") then
   s_state  <= goS;
    if (s_cnt < c_DELAY_4SEC) then
        s_cnt <= s_cnt + 1;
    end if;

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

    -- 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 <= goW;
        -- Reset local counter value
        s_cnt   <= c_ZERO;
    end if;

when goW =>
   if (cars = "00" or cars = "01") then
   s_state  <= goW;
    if (s_cnt < c_DELAY_4SEC) then
        s_cnt <= s_cnt + 1;
    end if;
   else
    -- Count up to c_DELAY_4SEC
    if (s_cnt < c_DELAY_4SEC) then
        s_cnt <= s_cnt + 1;
    else
        -- Move to the next state
        s_state <= waitW;
        -- Reset local counter value
        s_cnt   <= c_ZERO;
    end if;
   end if;
when waitW =>

    -- 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 <= goS;
            -- Reset local counter value
            s_cnt   <= c_ZERO;
          end if;


        when others =>
          s_state <= goS;

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