Part 1: Preparation tasks:

Complete 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

Color settings table:

RGB LED	Artix-7 pin names	Red	Yellow	Green
LD16	N15, M16, R12	1,0,0	1,1,0	0,1,0
LD17	N16, G14, R11	1,0,0	1,1,0	0,1,0

Color settings table:

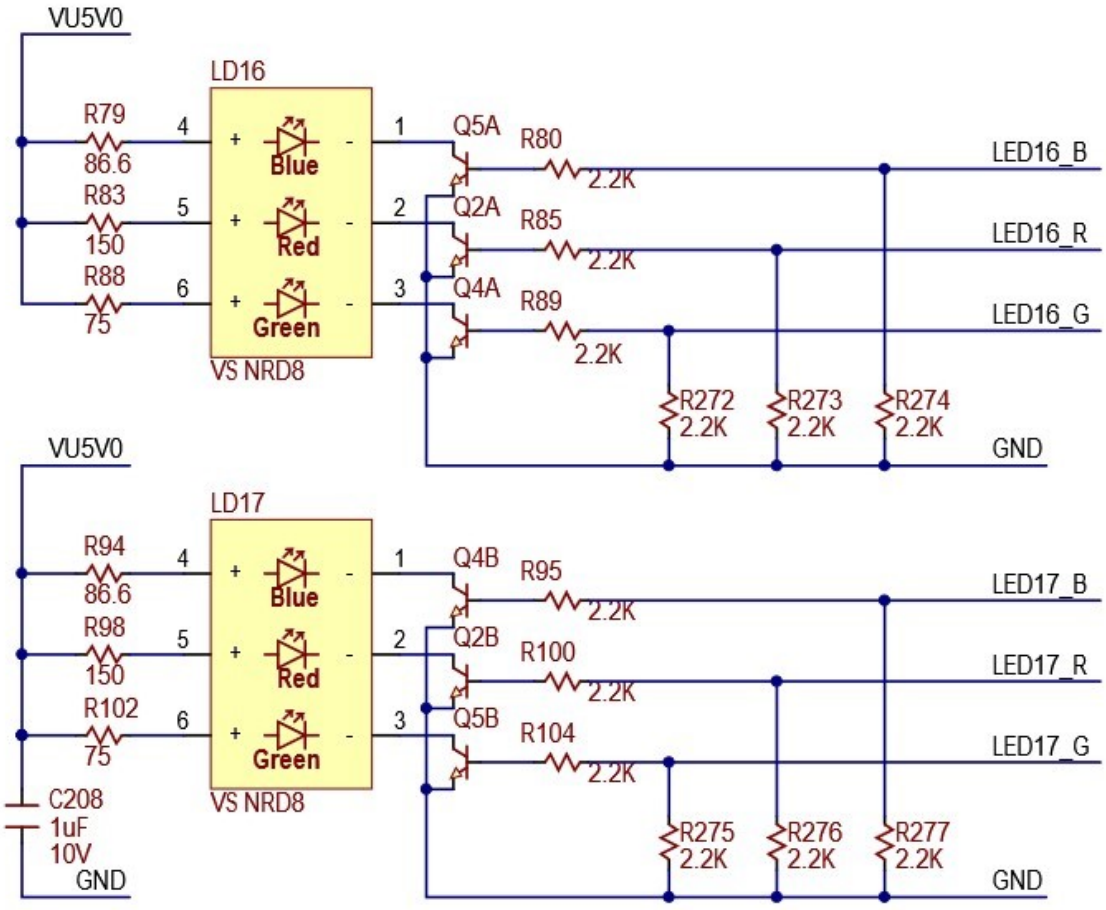
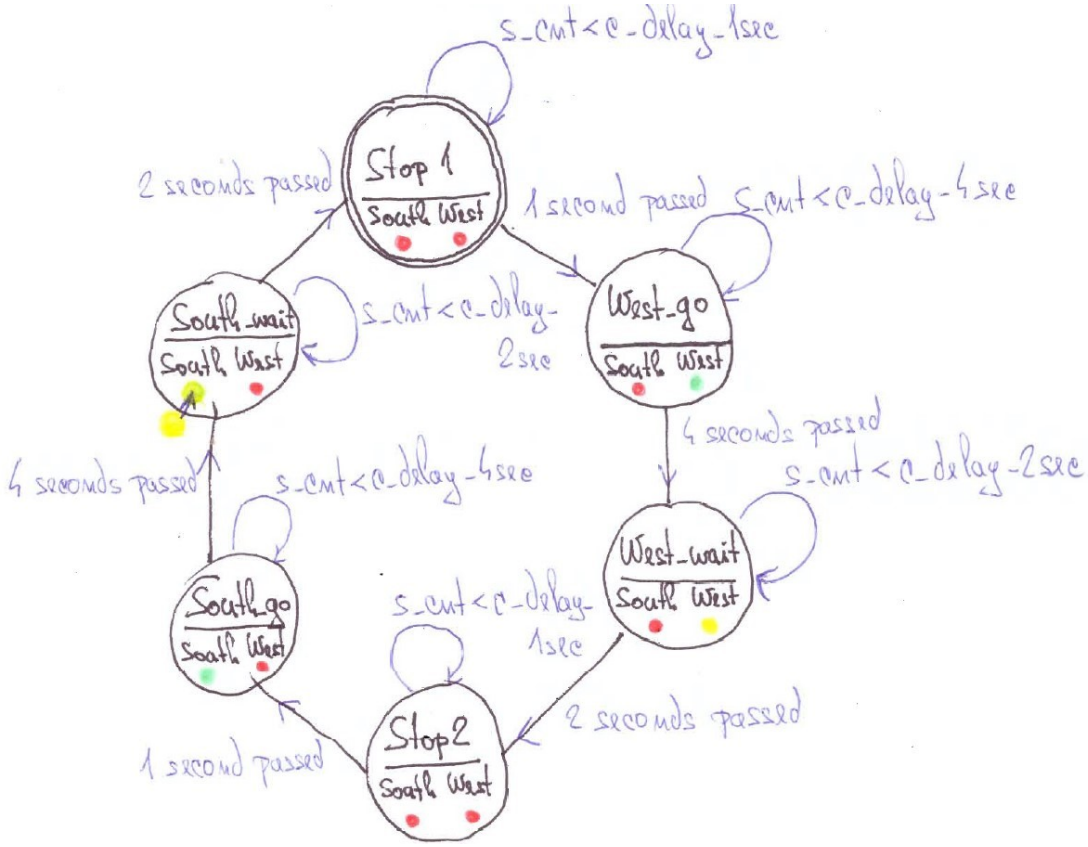


Image taken from Nexys A7 datasheet

Part 2: Traffic light controller:

State diagram:



Listing of VHDL code of sequential process p_traffic_fsm:

```
p_traffic_fsm : process(clk) -- ovládání stavů
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;
```

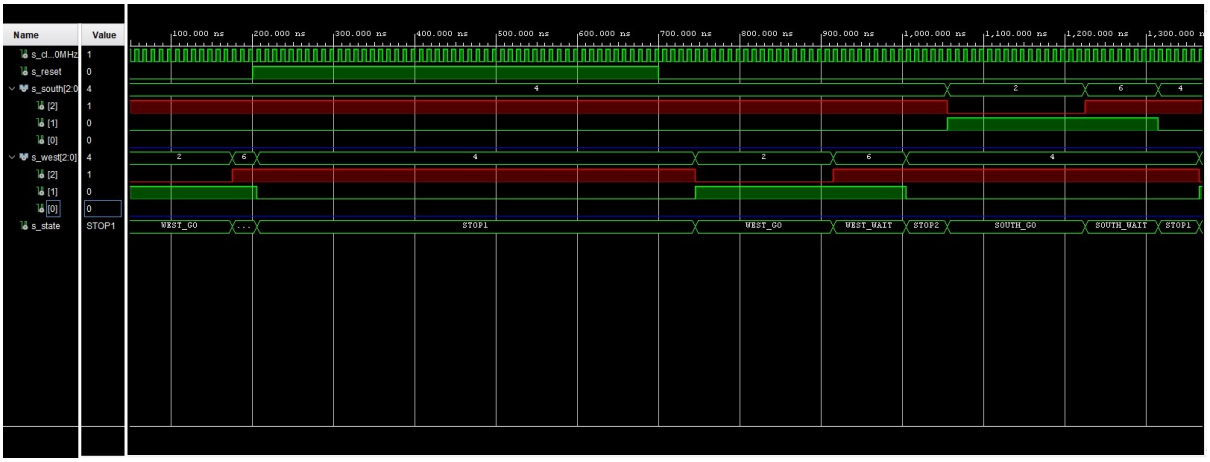
```
        when WEST_GO =>

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

        when WEST_WAIT =>

            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 =>

            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 =>

            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 =>

            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;
```

Listing of VHDL code of combinatorial process p_output_fsm:

```
p_output_fsm : process(s_state) -- ovládání výstupů
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 (RGB = 100)
            west_o  <= "100";   -- Red (RGB = 100)
    end case;
end process p_output_fsm;
```

Simulation screenshot:



Part 3: Smart controller:

State table, part 1:

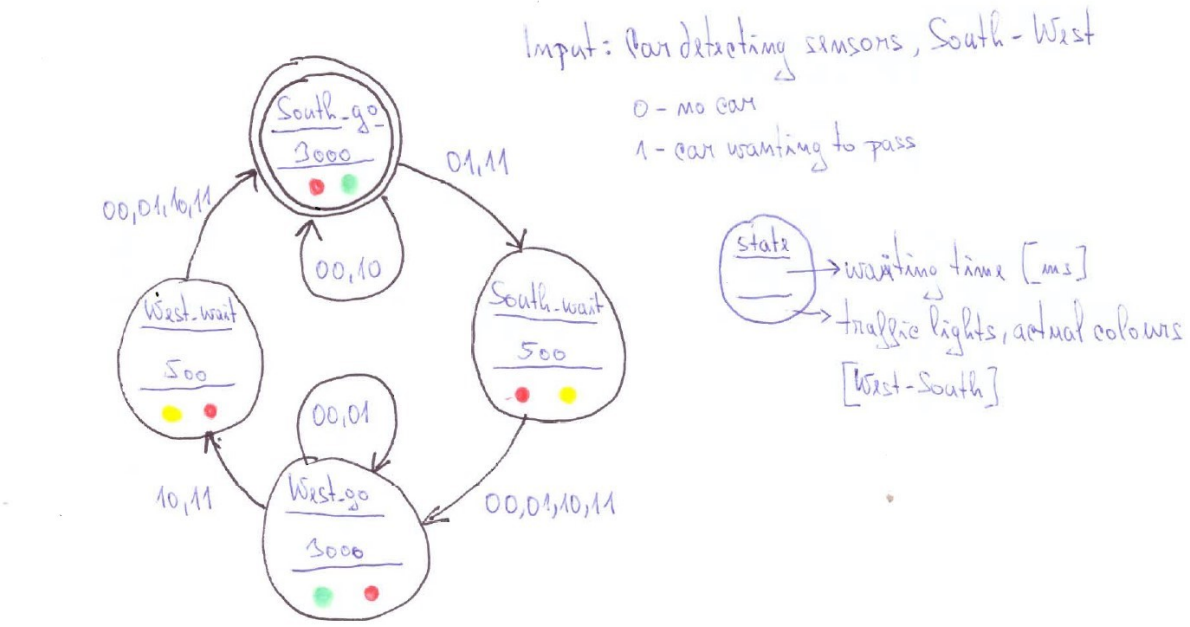
Input Sensor [South-West]	00	01	10	11	00
Clock	↑	↑	↑	↑	↑
Actual State	South_go	South_go	South_go	South_go	Sout_wait
Output Colours [West-South]	red-green	red-green	red-green	red-green	red-yellow
Next State	South_go	Sout_wait	Sout_go	Sout_wait	West_go

State table, part 2:

Input Sensor [South-West]	00	01	10	11	00
Clock	↑	↑	↑	↑	↑
Actual State	South_go	South_go	South_go	South_go	Sout_wait

Input Sensor [South-West]	**00**	**01**	**10**	**11**	**00**
Output Colours [West-South]	green-red	green-red	green-red	green-red	yellow-red
Next State	South_go	Sout_wait	Sout_go	Sout_wait	West_go

State diagram:



Listing of VHDL code of sequential process p_smart_traffic_fsm:

```
p_smart_traffic_fsm : process(clk) -- ovládání stavů
begin
    s_sensors_i <= sensors_i;
    if rising_edge(clk) then
        if (reset = '1') then       -- Synchronous reset
            s_smart_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_smart_state
            -- variable and changes to the next state according
            -- to the delay value.
```

```
    case s_smart_state is

        -- If the current state is SOUTH_GO, then wait 3 seconds
        -- and move to the next GO_WAIT state.
        when SOUTH_GO =>
            -- Count up to c_DELAY_1SEC
            if (s_cnt < c_DELAY_3SEC) then
                s_cnt <= s_cnt + 1;
            else
                if (s_sensors_i = "01" or s_sensors_i = "11") then
                    -- Move to the next state
                    s_smart_state <= SOUTH_WAIT;
                    -- Reset local counter value
                    s_cnt   <= c_ZERO;
                else
                    s_cnt <= c_ZERO;
                end if;
            end if;

        when SOUTH_WAIT =>

            if (s_cnt < c_DELAY_500mSEC) then
                s_cnt <= s_cnt + 1;
            else
                -- Move to the next state
                s_smart_state <= WEST_GO;
                -- Reset local counter value
                s_cnt   <= c_ZERO;
            end if;

        when WEST_GO =>

            if (s_cnt < c_DELAY_3SEC) then
                s_cnt <= s_cnt + 1;
            else
                if (s_sensors_i = "10" or s_sensors_i = "11") then
                    -- Move to the next state
                    s_smart_state <= WEST_WAIT;
                    -- Reset local counter value
                    s_cnt   <= c_ZERO;
                else
                    s_cnt <= c_ZERO;
                end if;
            end if;
        when WEST_WAIT =>

            if (s_cnt < c_DELAY_500mSEC) then
                s_cnt <= s_cnt + 1;
            else
                -- Move to the next state
                s_smart_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_smart_state <= SOUTH_GO;

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