Preparation tasks (done before the lab at home)

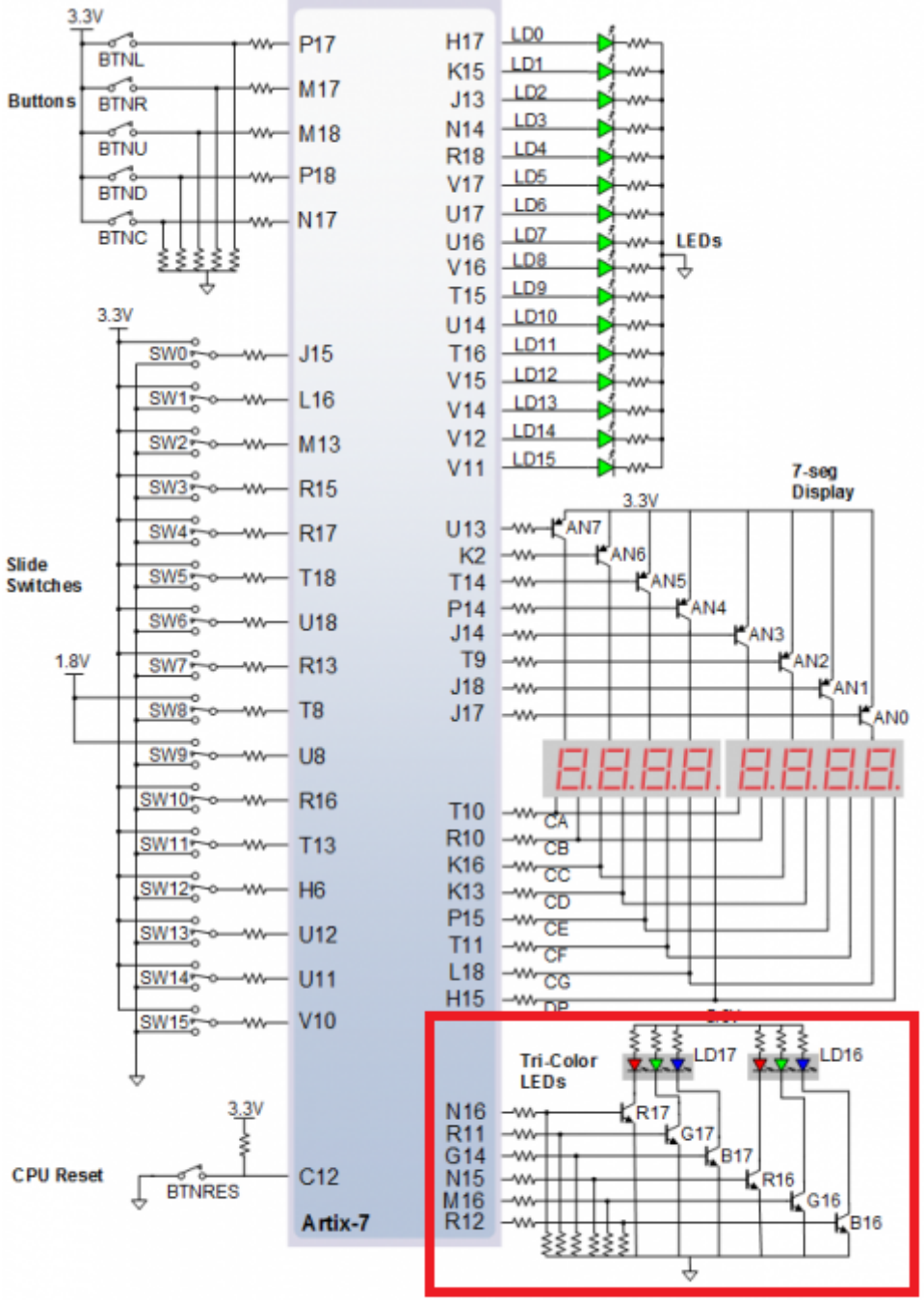
Fill in the table with the state names and output values accoding to the given inputs. Let the reset has just been applied.

Input P	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**
Clock	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
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 RGB

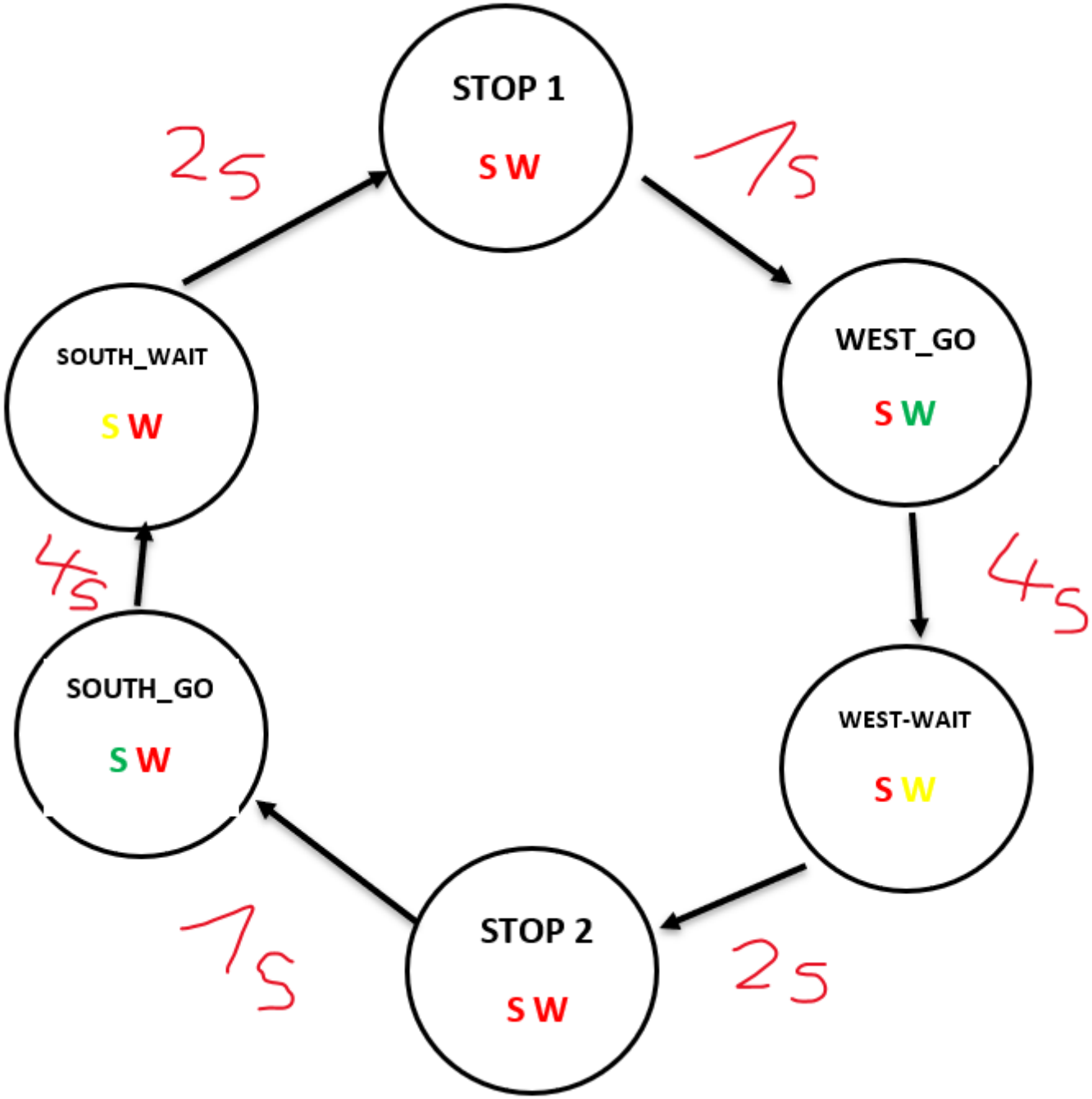
RGB LED	Artix-7 pin names	Red	Yellow	Green

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`



Part 2: Traffic light controller

State diagram



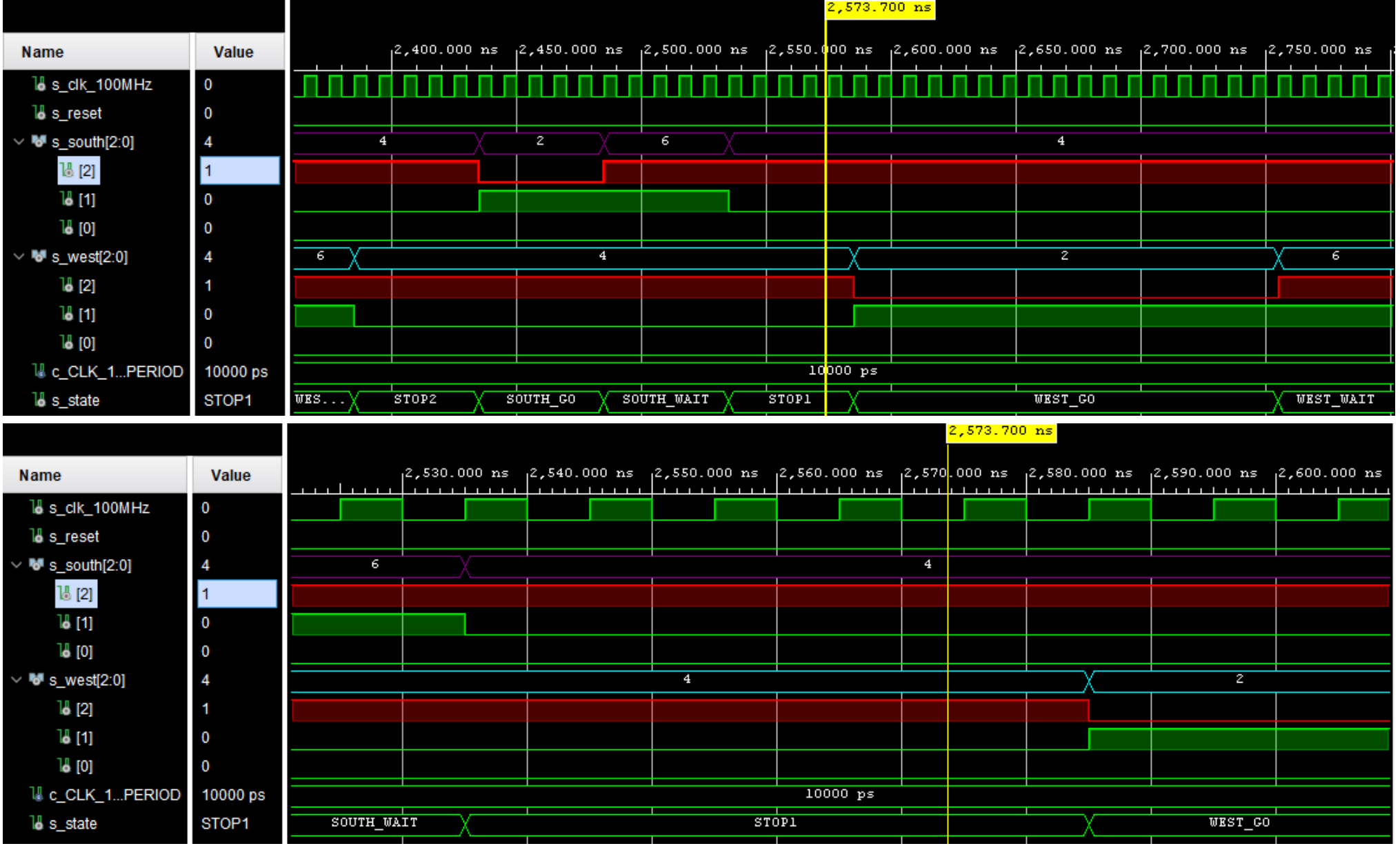
Listing of VHDL code of sequential process p_traffic_fsm

```
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_1SEC) 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_1SEC) 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 <= c_RED;
                west_o  <= c_RED;
            when WEST_GO =>

                south_o <= c_RED;
                west_o  <= c_GREEN;
            when WEST_WAIT =>
                south_o <= c_RED;
                west_o  <= c_YELLOW;
            when STOP2 =>
                south_o <= c_RED;
                west_o  <= c_RED;
            when SOUTH_GO =>
                south_o <= c_YELLOW;
                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;
```



Part 3: Smart controller