Laboratórna úloha číslo 7

Daniel Haluška

GitHub:

Link repozitára: https://github.com/DaNNym99/Digital-electronics-1

1. Pravdivostne tabulky

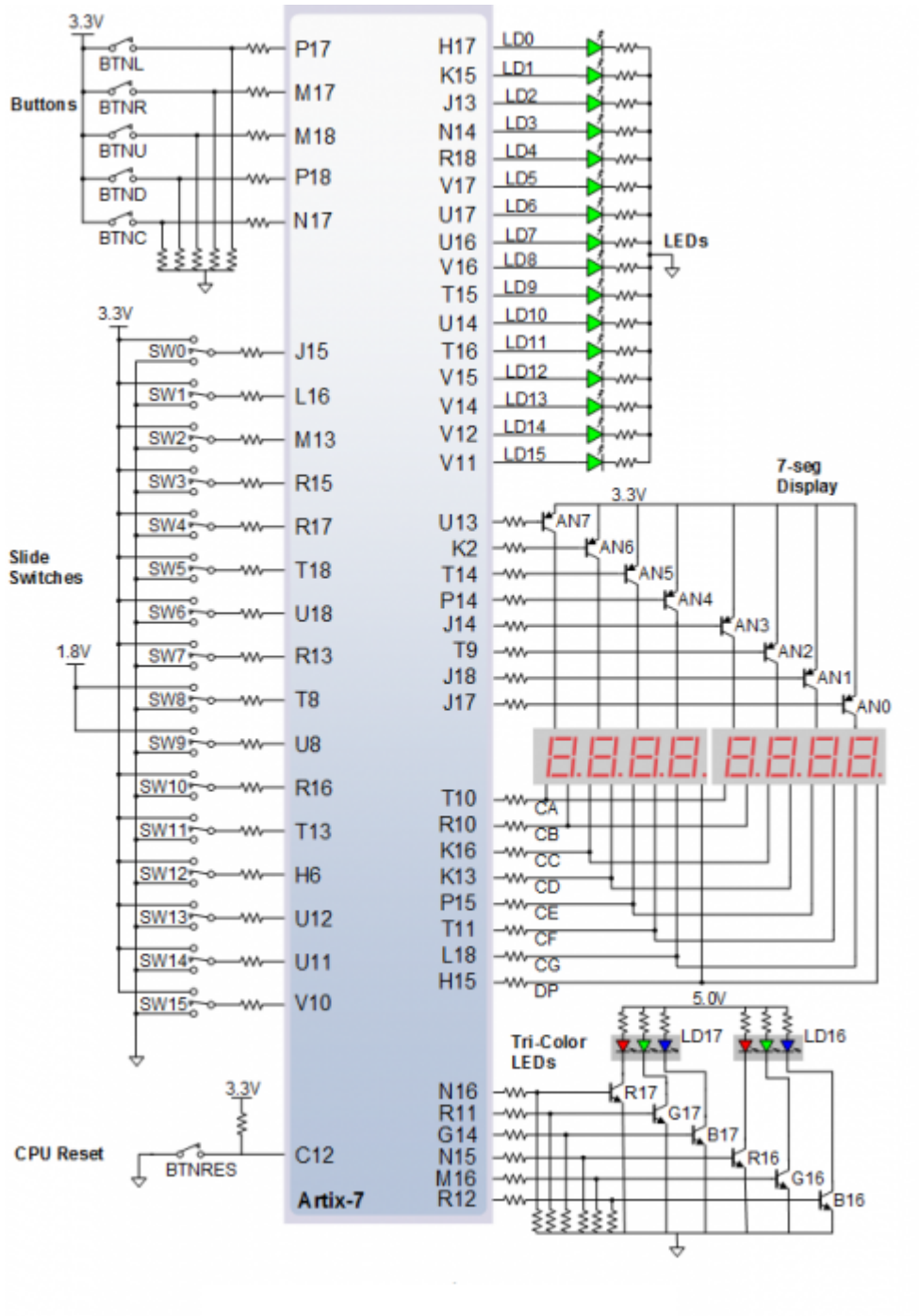
1.1. Tabulka stavov

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

1.2. Tabulka pripojeni RGB LED

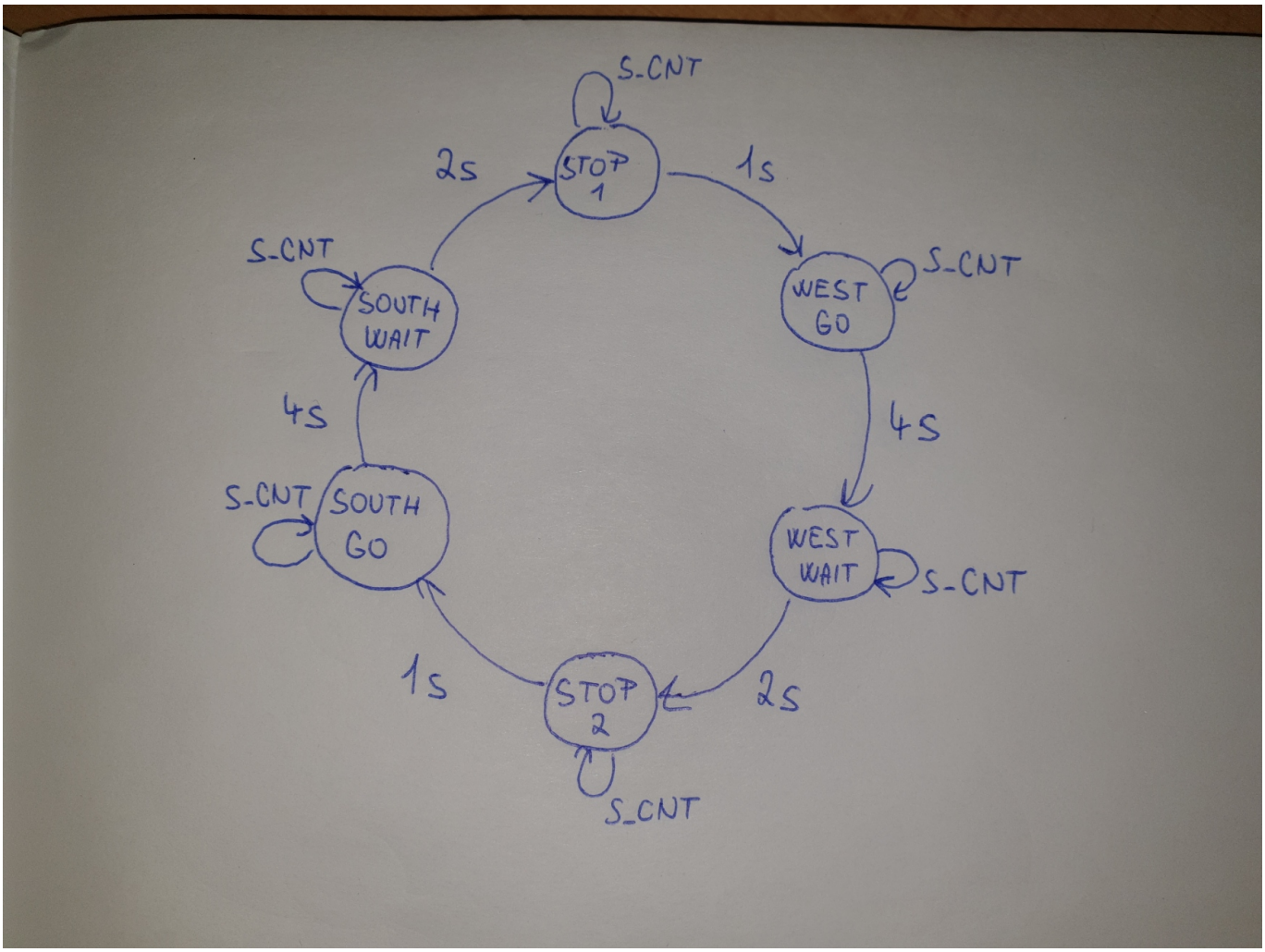
RGB LED	**Artix-7 pin names**	**Red**	**Yellow**	**Green**
LD16	N15, M16, R12	1,0,0	1,0,0	0,1,0
LD17	N16, R11, G14	1,0,0	1,0,0	0,1,0

1.3. Obrazok zapojenia Nexis A7 RGB LED



2. Funkcia semafóru

2.1. Stavový diagram



2.2. Proces 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;
```

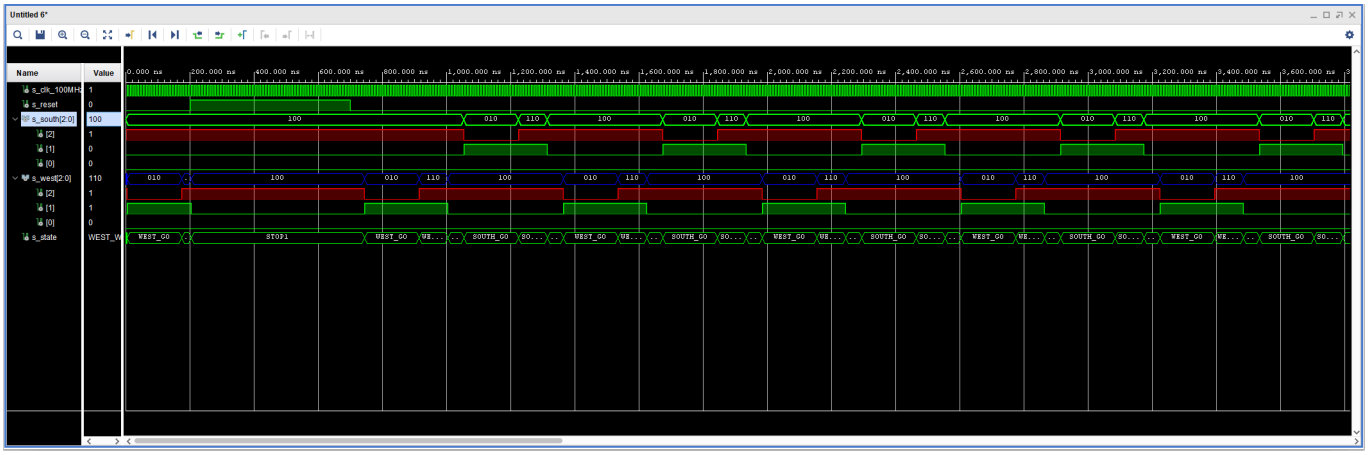
```
                    -- Reset local counter value
                    s_cnt   <= c_ZERO;
                end if;

            when WEST_GO =>

             -- Count up to c_DELAY_1SEC
                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_1SEC
                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_1SEC
                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_1SEC
                       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;
```

2.3. Proces p_output_fsm

```
p_output_fsm : process(s_state)
    begin
        case s_state is
            when STOP1 =>
                south_o <= "100";
                west_o  <= "100";
            when WEST_GO =>
                south_o <= "100";
                west_o  <= "010";
            when WEST_WAIT =>
                south_o <= "100";
                west_o  <= "110";
            when STOP2 =>
                south_o <= "100";
                west_o  <= "100";
            when SOUTH_GO =>
                south_o <= "010";
                west_o  <= "100";
            when SOUTH_WAIT =>
                south_o <= "110";
                west_o  <= "100";

            when others =>
                south_o <= "100";   -- Red
                west_o  <= "100";   -- Red
        end case;
    end process p_output_fsm;
```

2.4. Výstup simulácie

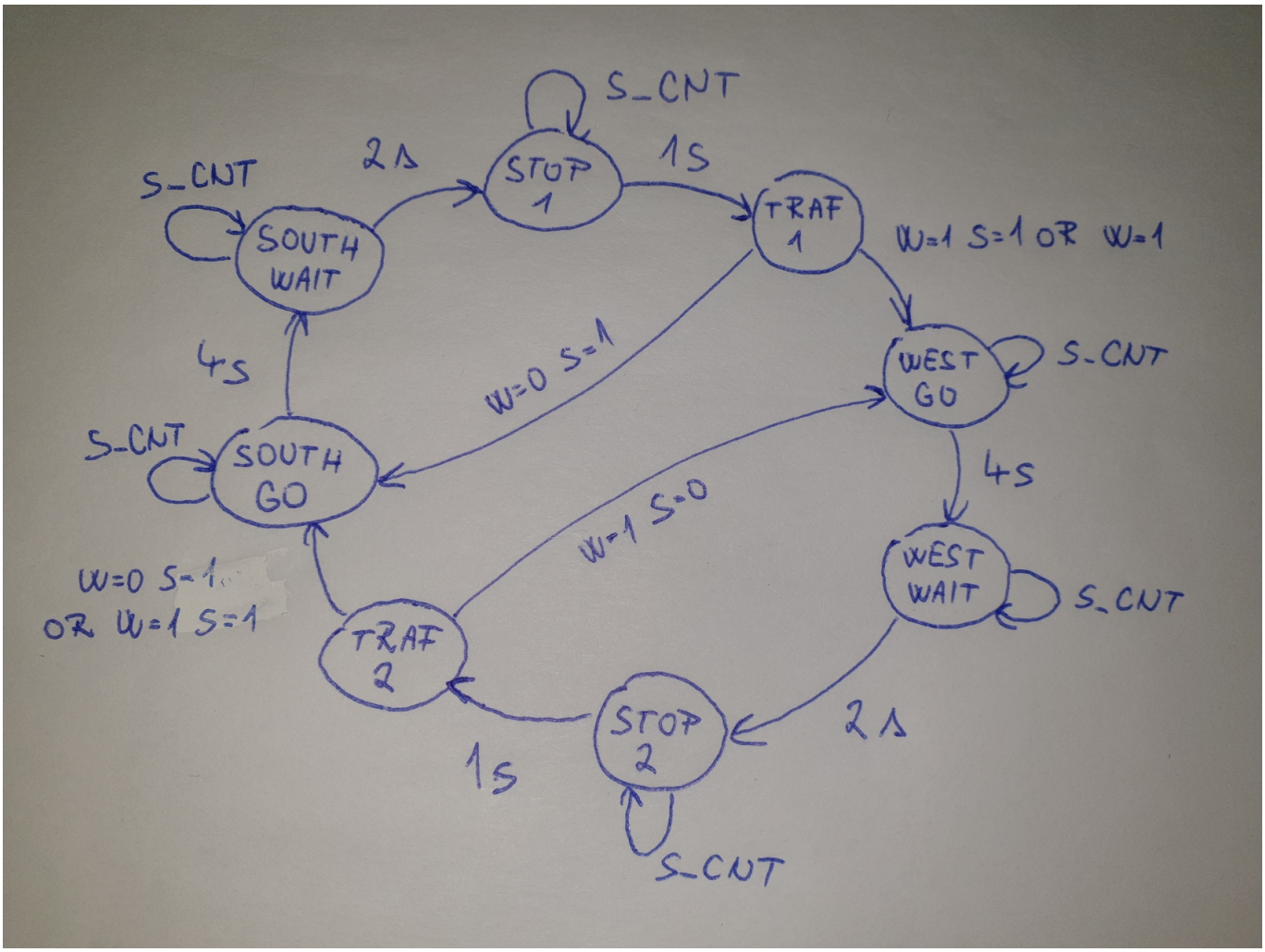


3. Controler

3.1. Tabuľka stavov

Current state	Direction South	Direction West	Delay
STOP1	red	red	1 sec
WEST_GO	red	green	4 sec
WEST_WAIT	red	yellow	2 sec
STOP2	red	red	1 sec
SOUTH_GO	green	red	4 sec
SOUTH_WAIT	yellow	red	2 sec
TRAF1	S = 1 W = 0	S = 1 W = 1 or S = 0 W = 1	0 sec
TRAF2	S = 1 W = 1 or S = 1 W = 0	S = 0 W = 1	0 sec

3.2. Stavový diagram



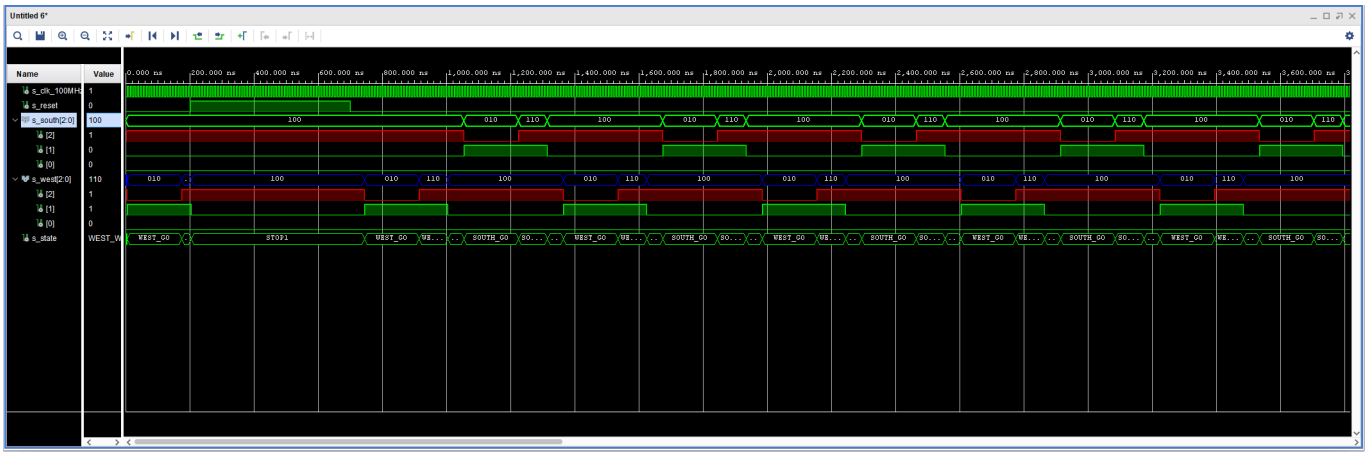
3.3. Proces p_smart_traffic_fsm

```
p_smart_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;

                        elsif(trafic_west = '0' and trafic_south = '0')then
                                s_state <= STOP1;
```

```
                    else

                        s_state   <= TRAF1;
                        s_cnt   <= c_ZERO;
                    end if;

                when TRAF1 =>

                        if(trafic_west = '1' and trafic_south = '0')then
                            s_state <= WEST_GO;
                        elsif(trafic_west = '0' and trafic_south = '1')then
                            s_state <= SOUTH_GO;
                        elsif(trafic_west = '1' and trafic_south = '1')then
                            s_state <= WEST_GO;
                        end if;

                when WEST_GO =>

                 -- Count up to c_DELAY_1SEC
                    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_1SEC
                    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;

                    elsif(trafic_west = '0' and trafic_south = '0')then
                            s_state <= STOP2;

                    else
                        -- Move to the next state
                        s_state <= TRAF2;
                        -- Reset local counter value
```

```
                        s_cnt   <= c_ZERO;
                    end if;

                    when TRAF2 =>

                        if(trafic_west = '1' and trafic_south = '0')then
                            s_state <= WEST_GO;
                        elsif(trafic_west = '0' and trafic_south = '1')then
                            s_state <= SOUTH_GO;
                        elsif(trafic_west = '1' and trafic_south = '1')then
                            s_state <= SOUTH_GO;
                        end if;

                    when SOUTH_GO =>

                   -- Count up to c_DELAY_1SEC
                    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_1SEC
                    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;

                when others =>
                    s_state <= STOP1;

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

3.4. Výstup simulácie


Untitled 6*
Name
Value
s_clk_100MHz
s_reset
s_south[2:0]
[2]
[1]
[0]
s_west[2:0]
[2]
[1]
[0]
s_state
WEST_W
100
010
110
STOP1
WEST_GO
SOUTH_GO