Assignment 8

Link to this Assignment Link to top of repository

1. Preparation tasks

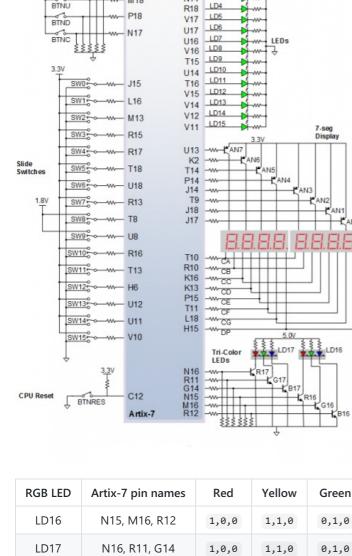
1.1 State table

Input P	0	0	1	1	0	1	0	1	1	1	1	0	0	1	1	1
Clock	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
State	Α	Α	В	С	С	D	Α	В	С	D	В	В	В	С	D	В
Output R	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0

LD1 K15 LD2 BTNR J13 N14 M18 BTNU

1.2 Connection of RGB LEDs on Nexys A7

LD0 H17



2.1 State diagram		
	STOP1 (RED,RED) 1s	

WEST_GO

(RED,GREEN) 4s

SOUTH_WAIT

(YELLOW,RED) 2s

2. Traffic light controller

```
SOUTH_GO
(GREEN,RED)
                                                        WEST_WAIT
(RED,YELLOW)
                                                             2s
                                 STOP2
                               (RED,RED)
2.2 Listing of VHDL code of sequential process p_traffic_fsm
      -- p_traffic_fsm:
      -- The sequential process with synchronous reset and clock_enable
      -- entirely controls the s_state signal by CASE statement.
      p_traffic_fsm : process(clk)
      begin
          if rising_edge(clk) then
                                         -- Synchronous reset-- Set initial state
              if (reset = '1') then
                   s_state <= STOP1 ;</pre>
                   s_cnt <= c_ZERO;
                                           -- Clear all bits
```

name of state (south, west) traffic light duration of state

```
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 \ensuremath{\text{sec}}
                      -- and move to the next GO_WAIT state.
                          -- Count up to c_DELAY_1SEC
                          if (s_cnt < c_DELAY_1SEC) then</pre>
                             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</pre>
                              s_cnt <= s_cnt + 1;
                              s_state <= WEST_WAIT;</pre>
                              s_cnt <= c_ZERO;</pre>
                          end if;
                      when WEST_WAIT =>
                          if (s_cnt < c_DELAY_2SEC) then</pre>
                              s_cnt <= s_cnt + 1;
                              s_state <= STOP2;
                              s_cnt <= c_ZERO;</pre>
                          end if;
                      when STOP2 =>
                          if (s_cnt < c_DELAY_1SEC) then</pre>
                              s_cnt <= s_cnt + 1;
                              s_state <= SOUTH_GO;
                              s_cnt <= c_ZERO;</pre>
                          end if;
                      when SOUTH_GO =>
                          if (s_cnt < c_DELAY_4SEC) then</pre>
                              s_cnt <= s_cnt + 1;
                              s_state <= SOUTH_WAIT;</pre>
                              s_cnt <= c_ZERO;</pre>
                          end if;
                      when SOUTH_WAIT =>
                          if (s_cnt < c_DELAY_2SEC) then</pre>
                              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;
2.3 Listing of VHDL code of combinatorial process p_output_fsm
      -- p_output_fsm:
      \mbox{--} 
 The combinatorial process is sensitive to state changes, and sets
      -- the output signals accordingly. This is an example of a Moore
      -- state machine because the output is set based on the active state.
      ______
     p_output_fsm : process(s_state)
         case s_state is
              when STOP1 =>
                 south_o <= c_RED;</pre>
                 west_o <= c_RED;
              when WEST_GO =>
```

when others => south_o <= c_RED;</pre> west_o <= c_RED;

south_o <= c_RED;</pre> west_o <= c_GREEN;</pre>

south_o <= c_RED;</pre> west_o <= c_RED;</pre>

south_o <= c_GREEN;</pre> west_o <= c_RED; when SOUTH_WAIT =>

> south_o <= c_YELLOW;</pre> west_o <= c_RED;</pre>

when WEST_WAIT => south_o <= c_RED;</pre> west_o <= c_YELLOW;</pre>

when STOP2 =>

when SOUTH_GO =>

end ca	ise:						
	ss p_output_fsm;						
Value	ot of simulation	s 1,000.000 ns 1,500	2,60, 4 2 4 2 6 4			2 6 4 2 6	989 ns 5,590.908
				1222			
. Smart co				10000 ps			İ
. Smart co		Output south	No cars	Cars on west	Cars on south	Booth direction	
. Smart co	9	Output south	No cars WEST_GO		Cars on south WEST_GO	Booth direction WEST_GO	
. Smart co	Output west			Cars on west			
Smart co 1 State table Actual State STOP1 WEST_GO	Output west	RED	WEST_GO	Cars on west WEST_GO	WEST_GO	WEST_GO	
Smart co 1 State table Actual State STOP1 WEST_GO	Output west RED GREEN	RED	WEST_GO WEST_GO	Cars on west WEST_GO WEST_GO	WEST_GO WEST_WAIT	WEST_GO WEST_WAIT	
. Smart co .1 State table Actual State STOP1 WEST_GO WEST_WAIT	Output west RED GREEN YELLOW	RED RED	WEST_GO WEST_GO STOP2	Cars on west WEST_GO WEST_GO STOP2	WEST_GO WEST_WAIT STOP2	WEST_GO WEST_WAIT STOP2	

WEST_GO (RED,GREEN) 4s

(1,X)

WEST_WAIT

(RED,YELLOW)

(X,0) SOUTH_GO (GREEN,RED)

(X,1)

SOUTH_WAIT (YELLOW,RED) 2s

3.2 State diagram

STOP2 (RED,RED) 1s 3.3 Listing of VHDL code of sequential process 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 ;</pre> -- Set initial state -- Clear all bits s_cnt <= c_ZERO; 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 ${\sf GO_WAIT}$ state. when STOP1 => -- Count up to c_DELAY_1SEC if (s_cnt < c_DELAY_1SEC) then</pre> s_cnt <= s_cnt + 1; -- Move to the next state s_state <= WEST_GO; -- Reset local counter value s_cnt <= c_ZERO; end if;

when others =>

end if; -- Synchronous reset

end case;

end if; -- Rising edge end process p_smart_traffic_fsm;

s_state <= STOP1;

STOP1 (RED,RED) 1s

LEGEND: name of state (south, west) traffic light duration of state

(south, west) traffic sensor

```
when WEST_GO =>
   if (s_cnt > c_DELAY_4SEC) and (south_sense_i = '1') then
        s_state <= WEST_WAIT;</pre>
        s_cnt <= c_ZERO;</pre>
    elsif (s_cnt > c_DELAY_4SEC) then
        s_cnt <= c_DELAY_4SEC + 1;</pre>
        s_cnt <= s_cnt + 1;
    end if;
when WEST_WAIT =>
    if (s_cnt < c_DELAY_2SEC) then</pre>
        s_cnt <= s_cnt + 1;
        s_state <= STOP2;
        s_cnt <= c_ZERO;</pre>
    end if;
when STOP2 =>
   if (s_cnt < c_DELAY_1SEC) then</pre>
        s_cnt <= s_cnt + 1;
       s_state <= SOUTH_GO;
        s_cnt <= c_ZERO;</pre>
    end if;
when SOUTH_GO =>
    if (s_cnt > c_DELAY_4SEC) and (west_sense_i = '1') then
       s_state <= SOUTH_WAIT;</pre>
        s_cnt <= c_ZERO;
    elsif (s_cnt > c_DELAY_4SEC) then
       s_cnt <= c_DELAY_4SEC + 1;</pre>
       s_cnt <= s_cnt + 1;
    end if;
when SOUTH_WAIT =>
   if (s_cnt < c_DELAY_2SEC) then</pre>
       s_cnt <= s_cnt + 1;
       s_state <= STOP1;
       s_cnt <= c_ZERO;
   end if;
```