TE2002B : Actividad 3.1 – MOORE

Equipo 3:

David Israel Vázquez Leal

Juan Carlos Hernández Ríos

José Emiliano Flores Martínez

Programa MOORE_3.vhd (entidad principal):

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity MOORE_3 is
  port (clk_3, rst_3 : in std_logic;
        left_3, right_3, interm_3 : in std_logic;
        lLights_3, rLights_3 : out std_logic_vector(2 downto 0));
end MOORE_3;

architecture Behavior of MOORE_3 is

-- State types: s0: both off | s1_x: left directional | s2_x: right directional | s3_x: intermittent
type state_type is (s0, s1_0, s1_1, s1_2, s1_3, s2_0, s2_1, s2_2, s2_3, s3_0, s3_1);
signal present_state_3, next_state_3 : state_type;

-- Define constants
   constant clk_3_FREQ: integer := 50000000;  -- 50 MHz clock frequency
   constant DELAY_100ns_3: integer := 5;  -- A delay of 100ns for intermittents (5 cycles of 20ns)
            constant DELAY_20ns_3: integer := 1;  -- A delay of 20ns for directional lights (the 4 state cycle lasts 80ns)

   -- Define signals
   signal counter_3: integer range 0 to 50000001 := 0;
   signal delay_done_3: std_logic := '1';
            signal skip_delay_3: std_logic := '0';
            signal delay_sel_3: std_logic := '0'; -- 0: delay of 100ns | 1: delay of 20ns

begin

  process(clk_3, rst_3)
  begin
    if (rising_edge(clk_3)) then
                                    -- State transitions
                                    present_state_3 <= next_state_3;
                        end if;
            end process;

            -- Delay for each sequence to take some time
            delay: process (clk_3)
            begin
                        if (rising_edge(clk_3)) then
                                    -- Checks if delay is skipped, or if delay is done (either 100ns or 20ns according to delay_sel_3)
                                    if ((counter_3>=DELAY_100ns_3 and delay_sel_3 = '0') or (counter_3>=DELAY_20ns_3 and delay_sel_3 = '1')
or skip_delay_3='1') then
                                                counter_3 <= 0;
                                                delay_done_3 <= '1';
                                    else
                                                counter_3 <= counter_3 + 1;
                                                delay_done_3 <= '0';
                                    end if;
                        end if;
            end process;

   -- Selecting next state
            C1: process(delay_done_3, clk_3, present_state_3, left_3, right_3, interm_3)

                        begin

                        -- most actions will not require to skip delay
                        skip_delay_3 <= '0';

                        case present_state_3 is

                                    --Only activated when lights are off (sequence of lights has completed its cycle)
                                    --Allows for skipping delay if lights are off, in state s0 (to activate a sequence instantly)
                                    when s0 =>
```

```
		if (interm_3 = '1') then
			next_state_3 <= s3_1;
			delay_sel_3 <= '0';
			skip_delay_3 <= '1';
		elsif (right_3 = '1') then
			next_state_3 <= s2_1;
			delay_sel_3 <= '1';
			skip_delay_3 <= '1';
		elsif (left_3 = '1') then
			next_state_3 <= s1_1;
			delay_sel_3 <= '1';
			skip_delay_3 <= '1';
		else
			next_state_3 <= s0;
			delay_sel_3 <= '0';
		end if;

	when s1_1 =>
		if (delay_done_3 = '1') then
			next_state_3 <= s1_2;
			delay_sel_3 <= '1';
		end if;

	when s1_2 =>
		if (delay_done_3 = '1') then
			next_state_3 <= s1_3;
			delay_sel_3 <= '1';
		end if;
	when s1_3 =>
		if (delay_done_3 = '1') then
			next_state_3 <= s1_0;
			delay_sel_3 <= '1';
		end if;

	-- Only s1_0, s2_0, s3_0 and s0 allow different state changes, as their cycles have ended
	when s1_0 =>
		if (delay_done_3 = '1') then
			if (interm_3 = '1') then
				next_state_3 <= s3_1;
				delay_sel_3 <= '0';
			elsif (right_3 = '1') then
				next_state_3 <= s2_1;
				delay_sel_3 <= '1';
			elsif (left_3 = '1') then
				next_state_3 <= s1_1;
				delay_sel_3 <= '1';
			else
				next_state_3 <= s0;
				delay_sel_3 <= '0';
			end if;
		end if;

	when s2_1 =>
		if (delay_done_3 = '1') then
			next_state_3 <= s2_2;
			delay_sel_3 <= '1';
		end if;

	when s2_2 =>
		if (delay_done_3 = '1') then
			next_state_3 <= s2_3;
			delay_sel_3 <= '1';
		end if;
	when s2_3 =>
		if (delay_done_3 = '1') then
			next_state_3 <= s2_0;
			delay_sel_3 <= '1';
		end if;

	when s2_0 =>
		if (delay_done_3 = '1') then
			if (interm_3 = '1') then
				next_state_3 <= s3_1;
				delay_sel_3 <= '0';
			elsif (right_3 = '1') then
				next_state_3 <= s2_1;
				delay_sel_3 <= '1';
			elsif (left_3 = '1') then
				next_state_3 <= s1_1;
				delay_sel_3 <= '1';
			else
				next_state_3 <= s0;
				delay_sel_3 <= '0';
			end if;
		end if;
```

```
                        when s3_1 =>
                                if (delay_done_3 = '1') then
                                        next_state_3 <= s3_0;
                                        delay_sel_3 <= '0';
                                end if;

                        when s3_0 =>
                                if (delay_done_3 = '1') then
                                        if (interm_3 = '1') then
                                                next_state_3 <= s3_1;
                                                delay_sel_3 <= '0';
                                        elsif (right_3 = '1') then
                                                next_state_3 <= s2_1;
                                                delay_sel_3 <= '1';
                                        elsif (left_3 = '1') then
                                                next_state_3 <= s1_1;
                                                delay_sel_3 <= '1';
                                        else
                                                next_state_3 <= s0;
                                                delay_sel_3 <= '0';
                                        end if;
                                end if;
                end case;
        end process;

        -- State actions
        C2 : process(present_state_3)
        begin
                case present_state_3 is
                        when s0 =>
                                rLights_3 <= "000";
                                lLights_3 <= "000";

                        when s1_1 =>
                                rLights_3 <= "000";
                                lLights_3 <= "001";

                        when s1_2 =>
                                rLights_3 <= "000";
                                lLights_3 <= "011";

                        when s1_3 =>
                                rLights_3 <= "000";
                                lLights_3 <= "111";

                        when s1_0 =>
                                rLights_3 <= "000";
                                lLights_3 <= "000";

                        when s2_1 =>
                                lLights_3 <= "000";
                                rLights_3 <= "001";

                        when s2_2 =>
                                lLights_3 <= "000";
                                rLights_3 <= "011";

                        when s2_3 =>
                                lLights_3 <= "000";
                                rLights_3 <= "111";

                        when s2_0 =>
                                lLights_3 <= "000";
                                rLights_3 <= "000";

                        when s3_1 =>
                                lLights_3 <= "111";
                                rLights_3 <= "111";

                        when s3_0 =>
                                lLights_3 <= "000";
                                rLights_3 <= "000";

                        when others =>
                                rLights_3 <= "000";
                                lLights_3 <= "000";

                end case;
        end process;


end Behavior;
```

Programa tb_MOORE_3.vhd (Testbench):

```
LIBRARY ieee;
USE         ieee.std_logic_1164.ALL;
USE         ieee.std_logic_signed.ALL;
USE         ieee.std_logic_textio.ALL;
USE         std.textio.ALL;

--Entity: no port list!
ENTITY  tb_MOORE_3 IS
END         tb_MOORE_3;

--Architecture
ARCHITECTURE test_architecture OF tb_MOORE_3 IS
            COMPONENT MOORE_3
                        PORT (clk_3, rst_3 : in std_logic;
                                    left_3, right_3, interm_3 : in std_logic;
                                    lLights_3, rLights_3 : out std_logic_vector(2 downto 0)
                                    );
            END COMPONENT;

SIGNAL      clk_3_tb : STD_LOGIC := '0';
SIGNAL rst_3_tb, left_3_tb, right_3_tb, interm_3_tb         :           STD_LOGIC := '0';                                   --INPUT
SIGNAL      lLights_3_tb, rLights_3_tb                                                  :           STD_LOGIC_VECTOR( 2 DOWNTO 0 ) := "000";
            --INPUT
SIGNAL expectL_3, expectR_3 : STD_LOGIC_VECTOR (2 DOWNTO 0);

constant clk_3_period : time := 20 ns; --clock period of fpga de10

BEGIN
            --DUT Instantiation
            DUT                     :           MOORE_3 PORT MAP( clk_3_tb, rst_3_tb, left_3_tb, right_3_tb, interm_3_tb, lLights_3_tb,
rLights_3_tb);

            -- Process for generating the clock
  clock_process: PROCESS
            BEGIN
                        clk_3_tb <= '0';
                        wait for clk_3_period/2;
                        clk_3_tb <= '1';
                        wait for clk_3_period/2;
            END PROCESS;


            --Stimulus by hand drawn waves, poor coverage
            stim_proc   :           PROCESS
                                                BEGIN
                                                WAIT FOR 0 ns;
                                                            rst_3_tb <= '0'; left_3_tb <= '0'; right_3_tb <= '0'; interm_3_tb <= '0';
expectL_3 <= "000"; expectR_3 <= "000";
                                                WAIT FOR clk_3_period * 20;
                                                            rst_3_tb <= '0'; left_3_tb <= '1'; right_3_tb <= '0'; interm_3_tb <= '0';
expectL_3 <= "000"; expectR_3 <= "000";
                                                WAIT FOR  clk_3_period * 20;
                                                            rst_3_tb <= '0'; left_3_tb <= '0'; right_3_tb <= '0'; interm_3_tb <= '1';
expectL_3 <= "000"; expectR_3 <= "000";
                                                WAIT FOR  clk_3_period * 20;
                                                            rst_3_tb <= '0'; left_3_tb <= '0'; right_3_tb <= '1'; interm_3_tb <= '0';
expectL_3 <= "000"; expectR_3 <= "000";
                                                WAIT FOR  clk_3_period * 20;
                                                            rst_3_tb <= '0'; left_3_tb <= '0'; right_3_tb <= '0'; interm_3_tb <= '0';
expectL_3 <= "000"; expectR_3 <= "000";
                                                WAIT;
                                    END PROCESS;

            --Monitor
            txt_out                 :           PROCESS( expectL_3, expectR_3 )
                                    VARIABLE str_o      :           LINE;
                                                BEGIN
                                                WRITE( str_o, STRING'( " expectL_3= " ));           WRITE( str_o,
expectL_3 );
                                                WRITE( str_o, STRING'( " expectR_3= " ));           WRITE( str_o,
expectR_3           );
                                                WRITE( str_o, STRING'( " lLights_3_tb= " ));        WRITE( str_o,
lLights_3_tb        );
                                                WRITE( str_o, STRING'( " rLights_3_tb= " ));        WRITE( str_o,
rLights_3_tb        );
                                                ASSERT false REPORT TIME'IMAGE( NOW ) & str_o.ALL
                                                            SEVERITY note;
                                                DEALLOCATE( str_o );
                                    END PROCESS;

END ARCHITECTURE;
```

Resultados de la Simulación: