Lab 4

Pragaash Mohan

October 21, 2020

Part 1

```
--Implementing a FSM
library ieee ;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity fsm_1 is
port(                SW                : in std_logic_vector(1 downto 0);
                     KEY          : in std_logic_vector(1 downto 0);
                     LEDR          : out std_logic_vector(9 downto 0)
                     -- setting up ports for inputs and clock and data out
                     --R, W, clk, Z : in std_logic;
                     --data_out      : out std_logic
                     );

end fsm_1;

-------------------------------------------------------------------

architecture behav of fsm_1 is
                        signal W, R, Z, clk : std_logic;

                        --defining states of FSM

                        type t_states is (y8, y7, y6, y5, y4, y3, y2, y1, y0);
                        signal S : std_logic_vector(8 downto 0) := "000000000"; --S
                        shared variable ps_x, ps_y, counter : integer range 0 to 7

                        --setting up one-hot codes for states
                        attribute enum_encoding : string;
              attribute enum_encoding of t_states : TYPE IS "000100001 000000010 000
```

```
                        --Setting current and next state
                        signal c_state, n_state : t_states;

begin


                        W <= SW(1);
                        R <= SW(0);
                        clk <= KEY(0);
                        LEDR(8 downto 0) <= S;
                        LEDR(9) <= Z;


                        --setting up D Flip-flop
                        state_reg: process(clk, R)
                        begin

                                if rising_edge(clk) then


                                        if (R='1') then
                                                c_state <= y8;
                                                ps_y := 0;
                                                ps_x := 0; -- Resets and counter se
                                        elsif (clk'event and clk='1') then
                                                c_state <= n_state;

                                        end if;

                                end if;


                        end  process state_reg;




                        comb: process(c_state, W)
                        begin

                        if rising_edge(clk) then
```

```
if (ps_y > 3) then
                counter := ps_y;
                else
                counter := ps_x;
        end if;



        case c_state is

                when y8 =>

                if(W = '1') then
                n_state <= y3;
                S <= "000100001";
                else
                n_state <= y7;
                --Increment counter while resetting
                ps_x := 0;
                ps_y := ps_y +1;
                S <= "000000011";
                end if;

                when y7 =>
                if(W = '1') then
                n_state <= y3;
                ps_x := ps_x +1;
                ps_y := 0;
                S <= "000100001";
                else
                n_state <= y6;
                ps_x := 0;
                ps_y := ps_y +1;
                S <= "000000101";
                end if;

                when y6 =>
                if(W = '1') then
                n_state <= y3;
                ps_x := ps_x +1;
                ps_y := 0;
                S <= "000100001";
                else
```

```
n_state <= y5;
ps_x := 0;
ps_y := ps_y +1;
S <= "000001001";
end if;

when y5 =>
if(W = '1') then
n_state <= y3;
ps_x := ps_x +1;
ps_y := 0;
S <= "000100001";
else
n_state <= y4;
ps_x := 0;
ps_y := ps_y +1;
S <= "000010001";
end if;

when y4 =>
if(W = '1') then
n_state <= y3;
ps_x := ps_x +1;
ps_y := 0;
S <= "000100001";
else
n_state <= y4;
ps_x := 0;
ps_y := ps_y +1;
S <= "000010001";
end if;

when y3 =>
if(W = '1') then
n_state <= y2;
ps_x := ps_x +1;
ps_y := 0;
S <= "001000001";
else
n_state <= y7;
ps_x := 0;
ps_y := ps_y +1;
S <= "000000101";
end if;
```

```
                when y2 =>
                if(W = '1') then
                n_state <= y1;
                ps_x := ps_x +1;
                ps_y := 0;
                S <= "010000001";
                else
                n_state <= y7;
                ps_x := 0;
                ps_y := ps_y +1;
                S <= "000000101";
                end if;

                when y1 =>
                if(W = '1') then
                n_state <= y0;
                ps_x := ps_x +1;
                ps_y := 0;
                S <= "100000001";
                else
                n_state <= y7;
                ps_x := 0;
                ps_y := ps_y +1;
                S <= "000000101";
                end if;

                when y0 =>
                if(W = '1')         then
                n_state <= y0;
                ps_x := ps_x +1;
                ps_y := 0;
                S <= "100000001";
                else
                n_state <= y7;
                ps_x := 0;
                ps_y := ps_y +1;
                S <= "000000101";
                end if;


        end case;
```

```
                                    end if;

                                    end process comb;

end architecture behav;
```

Part 2

Main VHDL

```
library ieee ;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity part_2 is
port(                SW                : in std_logic_vector(1 downto 0);
                     KEY             : in std_logic_vector(1 downto 0);
                     LEDR             : out std_logic_vector(9 downto 0)

                     );

end part_2;

-------------------------------------------------------------------

architecture behav of part_2 is
                     signal W, R, clk, counter : std_logic;

                     --defining states of FSM

                     type t_states is (A, B, C, D, E, F, G, H, I);
                     signal S : std_logic_vector(3 downto 0) := "0000"; --Start
                     shared variable ps_x, ps_y : integer range 0 to 7 := 0; --i

                     --setting up one-hot codes for states
                     attribute enum_encoding : string;
            attribute enum_encoding of t_states : TYPE IS "0000 0001 0010 0011 010

                     --Setting current and next state
                     signal c_state, n_state : t_states;

begin


                     --setting up D Flip-flop
                     state_reg: process(clk, R)
                     begin

                          if rising_edge(clk) then
```

```
                    if (R='1') then
                            c_state <= A;
                            ps_y := 0;
                            ps_x := 0; -- Resets and counter se
                    elsif (clk'event and clk='1') then
                            c_state <= n_state;

                    end if;

            end if;


end  process state_reg;




comb: process(c_state, W)
begin

if rising_edge(clk) then

            if ((ps_y > 3) or (ps_x > 3)) then
                  counter <= '1';
                            else
                            counter <= '0';

                    end if;


   case c_state is

                            when A =>
                            if (W = '0') then
                            n_state <= B;
                            S <= "0001";
                            --Increment counter while resetting

                            ps_x := ps_x +1;
                            ps_y := 0;
```

```
else
n_state <= F;
ps_x := 0;
ps_y := ps_y +1;
S <= "0101";
end if;

when B =>
if(W = '0') then
n_state <= C;
ps_x := ps_x +1;
ps_y := 0;
S <= "0010";
else
n_state <= F;
ps_x := 0;
ps_y := ps_y +1;
S <= "0101";
end if;

when C =>
if(W = '0') then
n_state <= D;
ps_x := ps_x +1;
ps_y := 0;
S <= "0011";
else
n_state <= F;
ps_x := 0;
ps_y := ps_y +1;
S <= "0101";
end if;

when D =>
if(W = '0') then
n_state <= E;
ps_x := ps_x +1;
ps_y := 0;
S <= "0100";
else
n_state <= F;
ps_x := 0;
ps_y := ps_y +1;
S <= "0101";
```

```
end if;

when E =>
if(W = '0') then
n_state <= E;
ps_x := ps_x +1;
ps_y := 0;
S <= "0100";
else
n_state <= F;
ps_x := 0;
ps_y := ps_y +1;
S <= "0101";
end if;

when F =>
if(W = '1') then
n_state <= G;
ps_x := ps_x +1;
ps_y := 0;
S <= "0110";
else
n_state <= B;
ps_x := 0;
ps_y := ps_y +1;
S <= "0001";
end if;

when G =>
if(W = '1') then
n_state <= H;
ps_x := ps_x +1;
ps_y := 0;
S <= "0111";
else
n_state <= B;
ps_x := 0;
ps_y := ps_y +1;
S <= "0001";
end if;

when H =>
if(W = '1') then
n_state <= I;
```

```
                    ps_x := ps_x +1;
                    ps_y := 0;
                    S <= "1000";
                    else
                    n_state <= B;
                    ps_x := 0;
                    ps_y := ps_y +1;
                    S <= "0001";
                    end if;

                    when I =>
                    if(W = '1')         then
                    n_state <= I;
                    ps_x := ps_x +1;
                    ps_y := 0;
                    S <= "1000";
                    else
                    n_state <= B;
                    ps_x := 0;
                    ps_y := ps_y +1;
                    S <= "0001";
                    end if;



                end case;


            end if;

            end process comb;

    W <= SW(1);
    R <= SW(0);
    clk <= KEY(0);
    LEDR(3 downto 0) <= S;
    --LEDR(9) <= counter; --OBS! Error.
```

```
end architecture behav;
```