PROIECT SCID

Reflex time measurement

Echipa:

Grupa 2124

Comsa Enoh

Ulican Alexandru

Zudor Crisztina

TEMA PROIECTULUI

9. Reflex time measurement

Un LED se aprinde la 1…5 secunde. Utilizatorul trebuie sa apese un buton cât mai repede posibil. Circuitul măsoară și afișează timpul in milisecunde dintre aprinderea LED-ului si apăsarea butonului. După 2 secunde , afișajul se stinge. După un timp aleator intre 1 și 5 secunde LED-ul se aprinde din nou si timpul de reacție este măsurat din nou.

Procedura este repetata de 5 ori .

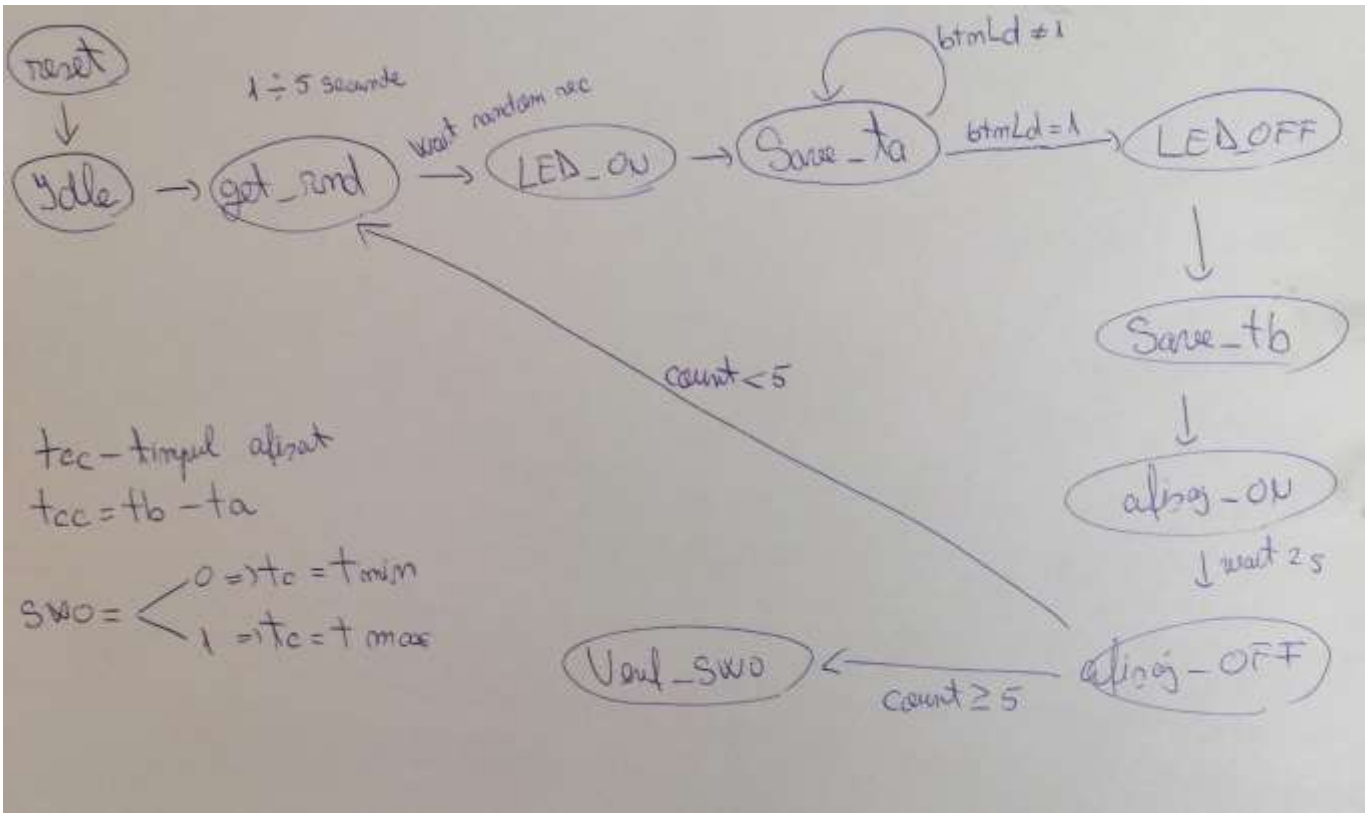
La finalul celor 5 repetiții, afișajul va indica :

• Timpul de reflex/reacție minim daca SW0=’0’

• Timpul de reflex/reacție maxim daca SW0=’1’ Butonul din mijloc resetează si repornește circuitul.

Nota : Puteți folosi orice operație de măsurare.

DIAGRAMA DE TRANZITII



IMPLEMENTARE

- ## fsm

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

entity fsm is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           sw : in STD_LOGIC_VECTOR (15 downto 0);
           led : out STD_LOGIC_VECTOR (15 downto 0);
           an : out std_logic_vector (3 downto 0);
           seg : out std_logic_vector (0 to 6);
           btn : in std_logic_vector (1 downto 0);
           dp : out std_logic
           );
end fsm;

architecture Behavioral of fsm is

type states is (idle, get_rnd, led_on, save_ta, led_off, save_tb, afisaj_on, afisaj_off, verif_sw0);

signal current_state, next_state : states;
signal count : integer range 0 to 6;
signal i : integer range 1 to 5;
signal rnd : STD_LOGIC_VECTOR (3 downto 0);
signal tc : STD_LOGIC_VECTOR (15 downto 0);
signal en_cnt : std_logic;
signal inc1sec : std_logic;
signal dp_iin : STD_LOGIC_VECTOR (3 downto 0):="1111";
```

```
component driver7seg is
    Port ( clk : in STD_LOGIC; --100MHz board clock input
           Din : in STD_LOGIC_VECTOR (15 downto 0); --16 bit binary data for 4 displays
           an : out STD_LOGIC_VECTOR (3 downto 0); --anode outputs selecting individual displays 3 to 0
           seg : out STD_LOGIC_VECTOR (0 to 6); -- cathode outputs for selecting LED-s in each display
           dp_in : in STD_LOGIC_VECTOR (3 downto 0); --decimal point input values
           dp_out : out STD_LOGIC; --selected decimal point sent to cathodes
           rst : in STD_LOGIC); --global reset
end component driver7seg;
component DeBounce is
    port(   Clock : in std_logic;
            Reset : in std_logic;
            button_in : in std_logic;
            pulse_out : out std_logic
        );
end component DeBounce;
component ceas is
    Port ( CLK100MHZ : in STD_LOGIC;
           btnL : in STD_LOGIC;
           btnR : in STD_LOGIC;
           btnC : in STD_LOGIC;
           seg : out STD_LOGIC_VECTOR (0 to 6);
           an : out STD_LOGIC_VECTOR (3 downto 0);
           dp : out STD_LOGIC);
end component ceas;

    type ms is record
        dig1 : integer range 0 to 9;
        dig2 : integer range 0 to 9;
    end record;
    type ds is record
```

```
        dig1 : integer range 0 to 9;
        dig2 : integer range 0 to 9;
    end record;
    type timp is record
        ms:ms;
        ds:ds;
    end record;
    signal t, ta, tb,tmax,tcc : timp := ((0,0),(0,0)) ;
    signal tmin:timp:=((9,9),(9,9));
    signal inc_ms : std_logic :=  '0';
    signal btnLd, btnRd : std_logic;

begin
deb1 : debounce port map (clock => clk, Reset => '0', button_in => btn(0), pulse_out => btnLd);
deb2 : debounce port map (clock => clk, Reset => '0', button_in => btn(1), pulse_out => btnRd);
u : driver7seg port map (clk => clk,
                         Din => tc,
                         an => an,
                         seg => seg,
                         dp_in => dp_iin,--afisseg
                         dp_out => dp,
                         rst => rst);

--ceas
msck : process (clk)
variable j : integer :=0;
begin
    if rising_edge(clk) then
       if j = 100000 then
          j:=0;
          inc_ms <= '1';
       else
```

```
          j := j+ 1;
          inc_ms <= '0';
       end if;
    end if;
end process;

secck : process(clk, rst)
variable j : integer := 0;
begin
  if rst = '1' then
    j := 0;
    inc1sec <= '0';
  elsif rising_edge(clk) then
    if en_cnt = '1' then
      if j = 10**8 - 1 then
        j := 1;
        inc1sec <= '1';
      else
        j := j + 1;
        inc1sec <= '0';
      end if;
    end if;
  end if;
end process;

ceasms : process (clk)
begin
    if rising_edge(clk) then
        if inc_ms = '1' then
            if t.ms.dig1 = 9 then
               t.ms.dig1 <= 0;
               if t.ms.dig2 = 9 then
```

```
                t.ms.dig2 <= 0;
                if t.ds.dig1 = 9 then
                  t.ds.dig1 <= 0;
                  if t.ds.dig2 = 9 then
                    t.ds.dig2 <= 0;
                    else
                      t.ds.dig2 <= t.ds.dig2 + 1;
                    end if;
                  else
                    t.ds.dig1 <= t.ds.dig1 + 1;
                  end if;
                else
                  t.ms.dig2 <= t.ms.dig2 + 1;
                end if;
              else
                t.ms.dig1 <= t.ms.dig1 + 1;
          end if;
          end if;
          end if;
end process;

rstprocc : process (clk, rst)
begin
  if rst = '1' then
    current_state <= idle;
  elsif rising_edge(clk) then
    current_state <= next_state;
  end if;
end process;

csns : process (current_state, sw, i,btnLd,count)
begin
```

```
  case current_state is
    when idle => next_state <= get_rnd;
    when get_rnd => next_state <= led_on;
    when led_on => next_state <=save_ta;
    when save_ta => if btnLd = '1' then
                      next_state <= led_off;
                    else
                      next_state <= save_ta;
                    end if;
    when led_off => next_state <=save_tb;
    when save_tb => next_state <= afisaj_on;
    when afisaj_on => next_state <= afisaj_off;
    when afisaj_off => if count=5 then
                      next_state <= verif_sw0;
                    else
                      next_state <=get_rnd;
                    end if;
    when others => next_state <= idle;
  end case;
end process;



--random number generator
lfsr: process (clk, rst)
variable shiftreg : std_logic_vector(15 downto 0) := x"ABCD";
variable firstbit : std_logic;
begin
  if rst = '1' then
    shiftreg := x"ABCD";
    rnd <= x"D";
  elsif rising_edge(clk) then
    firstbit := shiftreg(1) xnor  shiftreg(0);
```

```
    shiftreg := firstbit & shiftreg(15 downto 1);
    rnd <= shiftreg(3 downto 0);
  end if;
end process;

generate_i: process (clk, rst)
variable k : integer := 0;
begin
  if rst = '1' then
     i <= 1;
  elsif rising_edge(clk) then
     if current_state = get_rnd then
        if k=0 then
        i <= to_integer(unsigned(rnd));
        k:=1;
        elsif i>1 then
        en_cnt <='1';
        if inc1sec='1'then
        i<=i-1;
        end if;
     end if;
  end if;
  end if;
end process;

-- LED display
generate_led: process (clk, rst)
begin
  if rst = '1' then
  count<=0;
    led <= (others => '0');
  elsif rising_edge(clk) then
```

```
    if current_state = led_on then
      led(0) <= '1';
    elsif current_state = led_off then
      led(0) <= '0';
      count <= count+1;
    end if;
  end if;
end process;

-- citim timpul
generate_timp:process(clk, rst)
begin
    if rst = '1' then
    ta <= ((0,0),(0,0)) ;
    tb <= ((0,0),(0,0)) ;
  elsif rising_edge(clk) then
    if current_state = led_on then
      ta <= t;
    elsif current_state <= save_tb then
      tb <= t;

      ------- pentru tb.ms1---------
      if  tb.ms.dig1 < ta.ms.dig1 then
        if tb.ms.dig2=0 then
            if tb.ds.dig1=0 then
                tcc.ms.dig1 <= tb.ms.dig1 + 10 - ta.ms.dig1;
                tb.ds.dig1<=9;
                tb.ds.dig2<=9;
                tb.ds.dig2 <= tb.ds.dig2-1;
                else --if tb.ds.dig1>0 then
                tb.ds.dig1 <= tb.ds.dig1-1;
                tb.ms.dig2 <= 9;
```

```
            tcc.ms.dig1 <= tb.ms.dig1 +10 -ta.ms.dig1;
          end if;
            else
            tb.ms.dig2 <= tb.ms.dig2-1;
            tcc.ms.dig1 <= tb.ms.dig1 + 10 -ta.ms.dig1;
      end if;
            else
            tcc.ms.dig1 <= tb.ms.dig1 - ta.ms.dig1;
end if;

 ------- pentru tb.ms2---------
if  tb.ms.dig2 < ta.ms.dig2 then
        if tb.ds.dig1=0 then
            tcc.ms.dig2 <= tb.ms.dig2 + 10 - ta.ms.dig2;
            tb.ds.dig1<=9;
            tb.ds.dig2 <= tb.ds.dig2-1;
            else
            tb.ds.dig1 <= tb.ds.dig1-1;
            tcc.ms.dig2 <= tb.ms.dig2 +10 -ta.ms.dig2;
          end if;
            else
            tcc.ms.dig2 <= tb.ms.dig2 + 10 -ta.ms.dig2;
end if;

 ------- pentru tb.ds1---------
 if  tb.ds.dig1 < ta.ds.dig1 then
            tb.ds.dig2 <= tb.ds.dig2-1;
            tcc.ds.dig1 <= tb.ds.dig1 +10 -ta.ds.dig1;
          end if;
            else
             tcc.ds.dig1 <= tb.ds.dig1 +10 -ta.ds.dig1;
end if;
```

------- pentru tb.d2---------

tcc.ds.dig2 <= tb.ds.dig2 +10 -ta.ds.dig2;

tc <= std_logic_vector(to_unsigned(tcc.ds.dig2,4)) &
std_logic_vector(to_unsigned(tcc.ds.dig1,4)) &
std_logic_vector(to_unsigned(tcc.ms.dig2,4)) &
std_logic_vector(to_unsigned(tcc.ms.dig1,4));

--tmmin----

if tcc.ds.dig2 < tmin.ds.dig2 then
tmin.ds.dig2<=tcc.ds.dig2;
tmin.ds.dig1<=tcc.ds.dig1;
tmin.ms.dig2<=tcc.ms.dig2;
tmin.ms.dig1<=tcc.ms.dig1;
elsif tcc.ds.dig2=tmin.ds.dig2 and tcc.ds.dig1 < tmin.ds.dig1 then
tmin.ds.dig2<=tcc.ds.dig2;
tmin.ds.dig1<=tcc.ds.dig1;
tmin.ms.dig2<=tcc.ms.dig2;
tmin.ms.dig1<=tcc.ms.dig1;
elsif tcc.ds.dig1=tmin.ds.dig1 and tcc.ms.dig2 < tmin.ms.dig2 then
tmin.ds.dig2<=tcc.ds.dig2;
tmin.ds.dig1<=tcc.ds.dig1;
tmin.ms.dig2<=tcc.ms.dig2;
tmin.ms.dig1<=tcc.ms.dig1;
elsif tcc.ms.dig2=tmin.ms.dig2 and tcc.ms.dig1 < tmin.ms.dig1 then
tmin.ds.dig2<=tcc.ds.dig2;
tmin.ds.dig1<=tcc.ds.dig1;

```
tmin.ms.dig2<=tcc.ms.dig2;
tmin.ms.dig1<=tcc.ms.dig1;
end if;


---tmax-------
if tcc.ds.dig2 > tmax.ds.dig2 then
tmax.ds.dig2<=tcc.ds.dig2;
tmax.ds.dig1<=tcc.ds.dig1;
tmax.ms.dig2<=tcc.ms.dig2;
tmax.ms.dig1<=tcc.ms.dig1;
elsif tcc.ds.dig2=tmax.ds.dig2 and tcc.ds.dig1 > tmax.ds.dig1 then
tmax.ds.dig2<=tcc.ds.dig2;
tmax.ds.dig1<=tcc.ds.dig1;
tmax.ms.dig2<=tcc.ms.dig2;
tmax.ms.dig1<=tcc.ms.dig1;
elsif tcc.ds.dig1=tmax.ds.dig1 and tcc.ms.dig2 > tmax.ms.dig2 then
tmax.ds.dig2<=tcc.ds.dig2;
tmax.ds.dig1<=tcc.ds.dig1;
tmax.ms.dig2<=tcc.ms.dig2;
tmax.ms.dig1<=tcc.ms.dig1;
elsif tcc.ms.dig2=tmax.ms.dig2 and tcc.ms.dig1 > tmax.ms.dig1 then
tmax.ds.dig2<=tcc.ds.dig2;
tmax.ds.dig1<=tcc.ds.dig1;
tmax.ms.dig2<=tcc.ms.dig2;
tmax.ms.dig1<=tcc.ms.dig1;
elsif current_state = verif_sw0 then
  if sw(0)= '0' then
  --asistmin
   tc <= std_logic_vector(to_unsigned(tmin.ds.dig2,4)) &
        std_logic_vector(to_unsigned(tmin.ds.dig1,4)) &
        std_logic_vector(to_unsigned(tmin.ms.dig2,4)) &
```

```
            std_logic_vector(to_unsigned(tmin.ms.dig1,4));



        end if;
    elsif current_state = verif_sw0 then
        if sw(0)= '1' then
        --afis Tmax
        tc <= std_logic_vector(to_unsigned(tmax.ds.dig2,4)) &
            std_logic_vector(to_unsigned(tmax.ds.dig1,4)) &
            std_logic_vector(to_unsigned(tmax.ms.dig2,4)) &
            std_logic_vector(to_unsigned(tmax.ms.dig1,4));

    end if;
end if;
end if;
end process;



-- SSD display
--generate_timp: process (clk, rst)
--  variable mii1, sute1, zeci1, unitati1,mii2, sute2, zeci2,unitati2: integer range 0 to 9 := 0;
--begin
--  if rst = '1' then
--    tc <= (others => '0');
--    mii1 := 0;
--    sute1 := 0;
--    zeci1 := 0;
--    unitati1 := 0;
--    mii2 := 0;
--    sute2 := 0;
--    zeci2 := 0;
```

```
--    unitati2 := 0;
--  elsif rising_edge(clk) then
--    if current_state = led_on then
--     ta <= ((0,0),(0,0)) ;
--    end if;
--     if current_state = save_tb then
--    --luamtimp
--       mii2 := 0;
--    sute2 := 0;
--    zeci2 := 0;
--    unitati2 := 0;
--    end if;
--       mii1 := mii2-mii1;
--    sute1 := sute2-sute1;
--    zeci1 := zeci2-zeci1;
--    unitati1 := unitati2-unitati1;
--    tc <= std_logic_vector(to_unsigned(mii1,4)) &
--            std_logic_vector(to_unsigned(sute1,4)) &
--            std_logic_vector(to_unsigned(zeci1,4)) &
--            std_logic_vector(to_unsigned(unitati1,4));


--            end if;
--end process;

waitafison: process (clk, rst)
variable kk : integer := 0;
variable kkk : integer := 0;
begin
  if rst = '1' then
     kk := 0;
  elsif rising_edge(clk) then
     if current_state <= afisaj_on then
```

```
        if kk=0 then
        kkk:=2;
        kk:=1;
        elsif kkk>0 then
        en_cnt <='1';
        if inc1sec='1'then
        kkk:=kkk-1;
        end if;



     end if;
  end if;
  end if;
end process;

     dp_iin  <=  "0000"  when current_state = save_tb else  "1111";-- when   others;
   --  dp_iin<="1111"when current_state = save_tb



generate_final: process (clk, rst)
begin
  if rst = '1' then
  elsif rising_edge(clk) then

  end if;
end process;
end Behavioral;
```

- DeBounce

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

entity DeBounce is
    port(   Clock : in std_logic;
              Reset : in std_logic;
           button_in : in std_logic;
           pulse_out : out std_logic
        );
end DeBounce;

architecture behav of DeBounce is

--the below constants decide the working parameters.
--the higher this is, the more longer time the user has to press the button.
constant COUNT_MAX : integer := 10000000;
--set it '1' if the button creates a high pulse when its pressed, otherwise '0'.
constant BTN_ACTIVE : std_logic := '1';

signal count : integer := 0;
type state_type is (idle,wait_time); --state machine
signal state : state_type := idle;

begin

process(Reset,Clock)
begin
    if(Reset = '1') then
        state <= idle;
```

```
        pulse_out <= '0';
    elsif(rising_edge(Clock)) then
        case (state) is
            when idle =>
                if(button_in = BTN_ACTIVE) then
                    state <= wait_time;
                else
                    state <= idle; --wait until button is pressed.
                end if;
                pulse_out <= '0';
            when wait_time =>
                if(count = COUNT_MAX) then
                    count <= 0;
                    if(button_in = BTN_ACTIVE) then
                        pulse_out <= '1';
                    end if;
                    state <= idle;
                else
                    count <= count + 1;
                end if;
        end case;
    end if;
end process;

end architecture behav;
```

- driver7seg

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

```
entity driver7seg is
    Port ( clk : in STD_LOGIC; --100MHz board clock input
           Din : in STD_LOGIC_VECTOR (15 downto 0); --16 bit binary data for 4 displays
           an : out STD_LOGIC_VECTOR (3 downto 0); --anode outputs selecting individual displays 3 to 0
           seg : out STD_LOGIC_VECTOR (0 to 6); -- cathode outputs for selecting LED-s in each display
           dp_in : in STD_LOGIC_VECTOR (3 downto 0); --decimal point input values
           dp_out : out STD_LOGIC; --selected decimal point sent to cathodes
           rst : in STD_LOGIC); --global reset
end driver7seg;

architecture Behavioral of driver7seg is

signal clk1kHz : STD_LOGIC;
signal state : STD_LOGIC_VECTOR(16 downto 0);
signal addr : STD_LOGIC_VECTOR(1 downto 0);
signal cseg : STD_LOGIC_VECTOR(3 downto 0);

begin

div1kHz: process(clk, rst)
begin
  if rst = '1' then
       state <= '0' & X"0000";
  else
    if rising_edge(clk) then
       if state = '1' & X"869F" then --if counte reaches 99999
          state <= '0' & X"0000"; -- reset back to 0
       else
          state <= state+1;
       end if;
    end if;
```

```
   end if;
end process;

clk1Khz <= state(16); --assign MSB to frequency divider output


-- 2 bit counter generating 4 addresses for display multiplexing
counter_2bits: process(clk1kHz)
begin
  if rising_edge(clk1kHz) then
          addr <= addr+1;
  end if;
end process;

-- 2 to 4 decoder used to select one display of 4 at each sweeping address generated by the 2 bit counter
-- anodes are active low, decoder must provide '0' for activation
dcd3_8: process(addr)
begin
  case addr is
      when "00" =>  an <= "0111";
      when "01" =>  an <= "1011";
      when "10" =>  an <= "1101";
      when "11" =>  an <= "1110";
      when others => an <= "1111";
   end case;
end process;

--4 input multiplexer to select data to be sent to a single display
--synchronzied with addr and display activation with the anodes
data_mux4: process(addr,Din,dp_in)
begin
  case addr is
```

```
    when "00" =>  cseg <= Din(15 downto 12); --sending 4 upper bits targeted at display 3
                dp_out <= not dp_in(3); -- lighting up decimal point on display 3
    when "01" =>  cseg <= Din(11 downto 8); --sending next 4 bits targeted at display 2
                dp_out <= not dp_in(2); -- lighting up decimal point on display 2
    when "10" =>  cseg <= Din(7 downto 4);  -- ....
                dp_out <= not dp_in(1);
    when "11" =>  cseg <= Din(3 downto 0); -- ....
                dp_out <= not dp_in(0);
    when others => cseg <= "XXXX";
                 dp_out <= 'X';
  end case;
end process;

--binary to 7 segment decoder
--cathodes also active low, provide '0' for a lit up segment or decimal point
dcd7seg:process(cseg)
begin
  case cseg is
    when "0000" =>  seg <= "0000001";
    when "0001" =>  seg <= "1001111";
    when "0010" =>  seg <= "0010010";
    when "0011" =>  seg <= "0000110";
    when "0100" =>  seg <= "1001100";
    when "0101" =>  seg <= "0100100";
    when "0110" =>  seg <= "0100000";
    when "0111" =>  seg <= "0001111";
    when "1000" =>  seg <= "0000000";
    when "1001" =>  seg <= "0000100";
    when "1010" =>  seg <= "0000010";
    when "1011" =>  seg <= "1100000";
    when "1100" =>  seg <= "0110001";
    when "1101" =>  seg <= "1000010";
```

```
      when "1110" =>  seg <= "0110000";
      when "1111" =>  seg <= "0111000";
      when others => seg <= "XXXXXXX";
   end case;
end process;

end Behavioral;
```

- Basys3_master.xdc

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]



## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {sw[0]}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {sw[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {sw[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {sw[3]}]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {sw[4]}]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {sw[5]}]
set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports {sw[6]}]
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports {sw[7]}]
set_property -dict { PACKAGE_PIN V2    IOSTANDARD LVCMOS33 } [get_ports {sw[8]}]
set_property -dict { PACKAGE_PIN T3    IOSTANDARD LVCMOS33 } [get_ports {sw[9]}]
```

```
set_property -dict { PACKAGE_PIN T2    IOSTANDARD LVCMOS33 } [get_ports {sw[10]}]
set_property -dict { PACKAGE_PIN R3    IOSTANDARD LVCMOS33 } [get_ports {sw[11]}]
set_property -dict { PACKAGE_PIN W2    IOSTANDARD LVCMOS33 } [get_ports {sw[12]}]
set_property -dict { PACKAGE_PIN U1    IOSTANDARD LVCMOS33 } [get_ports {sw[13]}]
set_property -dict { PACKAGE_PIN T1    IOSTANDARD LVCMOS33 } [get_ports {sw[14]}]
set_property -dict { PACKAGE_PIN R2    IOSTANDARD LVCMOS33 } [get_ports {sw[15]}]


## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {led[0]}]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports {led[1]}]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports {led[2]}]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports {led[3]}]
set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports {led[4]}]
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports {led[5]}]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {led[6]}]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {led[7]}]
set_property -dict { PACKAGE_PIN V13   IOSTANDARD LVCMOS33 } [get_ports {led[8]}]
set_property -dict { PACKAGE_PIN V3    IOSTANDARD LVCMOS33 } [get_ports {led[9]}]
set_property -dict { PACKAGE_PIN W3    IOSTANDARD LVCMOS33 } [get_ports {led[10]}]
set_property -dict { PACKAGE_PIN U3    IOSTANDARD LVCMOS33 } [get_ports {led[11]}]
set_property -dict { PACKAGE_PIN P3    IOSTANDARD LVCMOS33 } [get_ports {led[12]}]
set_property -dict { PACKAGE_PIN N3    IOSTANDARD LVCMOS33 } [get_ports {led[13]}]
set_property -dict { PACKAGE_PIN P1    IOSTANDARD LVCMOS33 } [get_ports {led[14]}]
set_property -dict { PACKAGE_PIN L1    IOSTANDARD LVCMOS33 } [get_ports {led[15]}]


##7 Segment Display
set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports {seg[0]}]
set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports {seg[1]}]
set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports {seg[2]}]
set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports {seg[3]}]
```

```
set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports {seg[4]}]
set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports {seg[5]}]
set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports {seg[6]}]

set_property -dict { PACKAGE_PIN V7   IOSTANDARD LVCMOS33 } [get_ports dp]

set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports {an[0]}]
set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports {an[1]}]
set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports {an[2]}]
set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports {an[3]}]


##Buttons
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports rst]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports btnU]
set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports btn[0]]
set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports btn[1]]
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports btnD]


##Pmod Header JA
#set_property -dict { PACKAGE_PIN J1   IOSTANDARD LVCMOS33 } [get_ports {JA[0]}];#Sch name = JA1
#set_property -dict { PACKAGE_PIN L2   IOSTANDARD LVCMOS33 } [get_ports {JA[1]}];#Sch name = JA2
#set_property -dict { PACKAGE_PIN J2   IOSTANDARD LVCMOS33 } [get_ports {JA[2]}];#Sch name = JA3
#set_property -dict { PACKAGE_PIN G2   IOSTANDARD LVCMOS33 } [get_ports {JA[3]}];#Sch name = JA4
#set_property -dict { PACKAGE_PIN H1   IOSTANDARD LVCMOS33 } [get_ports {JA[4]}];#Sch name = JA7
#set_property -dict { PACKAGE_PIN K2   IOSTANDARD LVCMOS33 } [get_ports {JA[5]}];#Sch name = JA8
#set_property -dict { PACKAGE_PIN H2   IOSTANDARD LVCMOS33 } [get_ports {JA[6]}];#Sch name = JA9
#set_property -dict { PACKAGE_PIN G3   IOSTANDARD LVCMOS33 } [get_ports {JA[7]}];#Sch name = JA10

##Pmod Header JB
#set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports {JB[0]}];#Sch name = JB1
```

#set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports {JB[1]}];#Sch name = JB2

#set_property -dict { PACKAGE_PIN B15   IOSTANDARD LVCMOS33 } [get_ports {JB[2]}];#Sch name = JB3

#set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports {JB[3]}];#Sch name = JB4

#set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports {JB[4]}];#Sch name = JB7

#set_property -dict { PACKAGE_PIN A17   IOSTANDARD LVCMOS33 } [get_ports {JB[5]}];#Sch name = JB8

#set_property -dict { PACKAGE_PIN C15   IOSTANDARD LVCMOS33 } [get_ports {JB[6]}];#Sch name = JB9

#set_property -dict { PACKAGE_PIN C16   IOSTANDARD LVCMOS33 } [get_ports {JB[7]}];#Sch name = JB10

##Pmod Header JC

#set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 } [get_ports {JC[0]}];#Sch name = JC1

#set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports {JC[1]}];#Sch name = JC2

#set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports {JC[2]}];#Sch name = JC3

#set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports {JC[3]}];#Sch name = JC4

#set_property -dict { PACKAGE_PIN L17   IOSTANDARD LVCMOS33 } [get_ports {JC[4]}];#Sch name = JC7

#set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports {JC[5]}];#Sch name = JC8

#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports {JC[6]}];#Sch name = JC9

#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports {JC[7]}];#Sch name = JC10

##Pmod Header JXADC

#set_property -dict { PACKAGE_PIN J3   IOSTANDARD LVCMOS33 } [get_ports {JXADC[0]}];#Sch name = XA1_P

#set_property -dict { PACKAGE_PIN L3   IOSTANDARD LVCMOS33 } [get_ports {JXADC[1]}];#Sch name = XA2_P

#set_property -dict { PACKAGE_PIN M2   IOSTANDARD LVCMOS33 } [get_ports {JXADC[2]}];#Sch name = XA3_P

#set_property -dict { PACKAGE_PIN N2   IOSTANDARD LVCMOS33 } [get_ports {JXADC[3]}];#Sch name = XA4_P

#set_property -dict { PACKAGE_PIN K3   IOSTANDARD LVCMOS33 } [get_ports {JXADC[4]}];#Sch name = XA1_N

#set_property -dict { PACKAGE_PIN M3   IOSTANDARD LVCMOS33 } [get_ports {JXADC[5]}];#Sch name = XA2_N

#set_property -dict { PACKAGE_PIN M1   IOSTANDARD LVCMOS33 } [get_ports {JXADC[6]}];#Sch name = XA3_N

#set_property -dict { PACKAGE_PIN N1   IOSTANDARD LVCMOS33 } [get_ports {JXADC[7]}];#Sch name = XA4_N

##VGA Connector

#set_property -dict { PACKAGE_PIN G19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[0]}]

#set_property -dict { PACKAGE_PIN H19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[1]}]

#set_property -dict { PACKAGE_PIN J19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[2]}]

#set_property -dict { PACKAGE_PIN N19   IOSTANDARD LVCMOS33 } [get_ports {vgaRed[3]}]

#set_property -dict { PACKAGE_PIN N18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[0]}]

#set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[1]}]

#set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[2]}]

#set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[3]}]

#set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[0]}]

#set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[1]}]

#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[2]}]

#set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[3]}]

#set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS33 } [get_ports Hsync]

#set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports Vsync]

##USB-RS232 Interface

#set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports RsRx]

#set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports RsTx]

##USB HID (PS/2)

#set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33   PULLUP true } [get_ports PS2Clk]

#set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33   PULLUP true } [get_ports PS2Data]

##Quad SPI Flash

##Note that CCLK_0 cannot be placed in 7 series devices. You can access it using the

##STARTUPE2 primitive.

```
#set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports {QspiDB[0]}]
#set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports {QspiDB[1]}]
#set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports {QspiDB[2]}]
#set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 } [get_ports {QspiDB[3]}]
#set_property -dict { PACKAGE_PIN K19   IOSTANDARD LVCMOS33 } [get_ports QspiCSn]


## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```