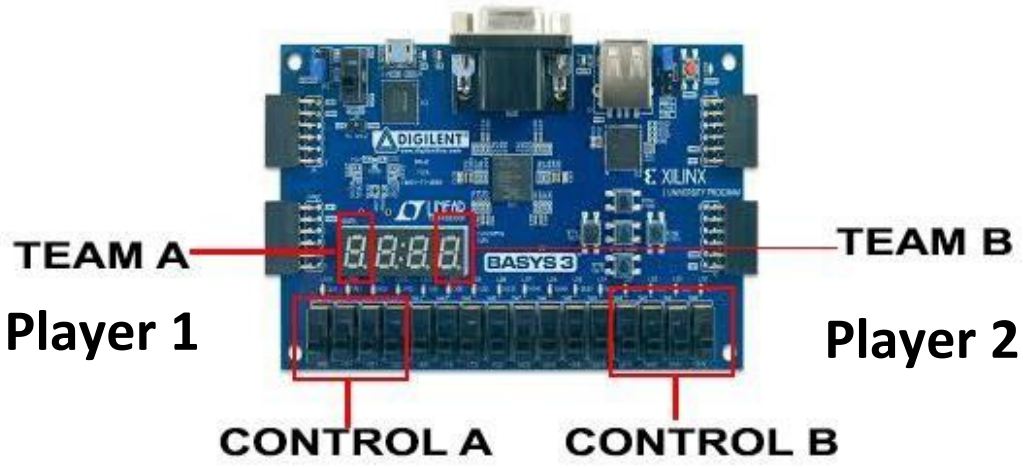
Lab 4
Seven Segment Display
Haider Raheem
3/4/2020

BINARY TO DECIMAL GAME



The Design Methodology:

My lab focused on making a game in which two sets of controls 4 switches each will be able to control 2 different seven segment displays. The first one to count consecutively to 9 wins.

The purpose of the lab was to operate the seven-segment display present on the Basys 3 and most importantly more than one displays on one time. So key concept was using a multiplexer at the end whose select line was a 16ms clock, which changed from 0 to 1 and vice versa. So with this changing select line the output is given to two different seven-segment displays from two different inputs as the clock is fast and human eye cannot detect these rapid changing outputs on LEDs, we only see two displays switched on, at the same time.

In order give desired output we used a decoder which decoded the binary numbers to desired 7 bit value that was to be given to negative logic cathodes of the seven segment display.

In order to operate two seven segment displays at extreme sides we gave desired output to anodes via multiplexer. The desired outputs given to anodes and the outputs decided for the decoder were based on the truth table used in the preliminary report.

Lab 4
Seven Segment Display
Haider Raheem
3/4/2020

Truth Table:

Inputs				Outputs							Decimal Number
A	**B**	**C**	**D**	**a**	**b**	**c**	**d**	**e**	**f**	**g**	
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**
0	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**2**
0	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**3**
0	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**4**
0	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**5**
0	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**6**
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**7**
1	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**8**

Logic Equations:

a = A + C + BD +Bnot.Dnot

b = Bnot + Cnot.Dnot + CD

c = B + Cnot + D

d = Bnot.Dnot + C.Dnot + B.Cnot.D + Bnot.C + A

e=Bnot.Dnot + C.Dnot

f=A +Cnot.Dnot + B.Cnot + BDnot

g=A + B.Cnot + Bnot.C + C.Dnot

Software implementation:

First of all I wrote the VHDL code for clock divider to get 16ms clock for the mux, then I wrote the code for decoder and multiplexer. Lastly I wrote the code for Top module that connected the modules accordingly. As two scores were to be displayed a decoder was initialized twice in the top module. The VHDL code for the project is present at the end. The software test bench code successful supported the proposed methodology plan and finally the software implementation finished.

For Decoder when case was used and for Multiplexer if statements were used in VHDL coding. For clock divider an integer counter signal and rising edge function was used.

Modules in the code:

2 Decoders : 4 x 7

Clock Divider: 100MHz to 62.5 Hz

Multiplexer : 3 to 2

Hardware implementation:

Inputs: Basys 3 switches

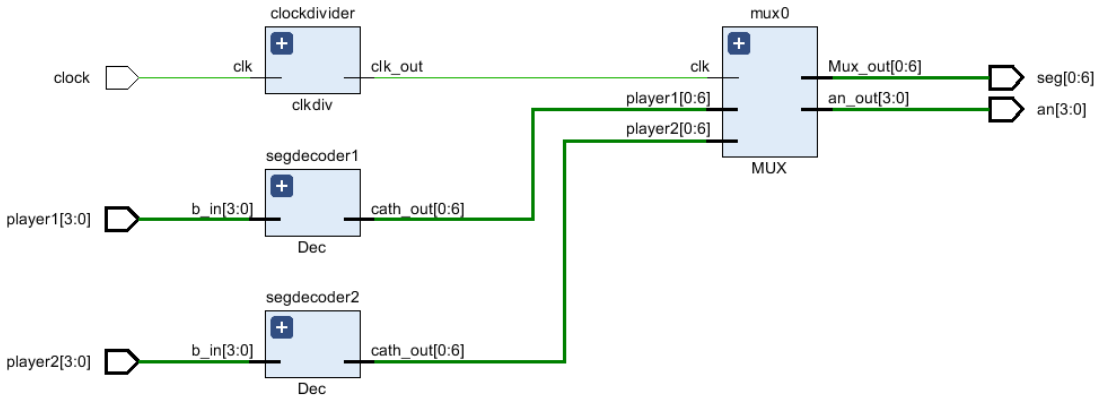
Output: seven Segment display

A bitstream was generated then Basys 3 board was connected to the computer and using the hardware manager in Xilinx Vivado the Basys 3 board was programmed and Hardware implementation worked according to prelim report.

RESULTS:

The results of the lab verified the success of the preliminary work. The Hardware and software worked as planned in the prelim report.

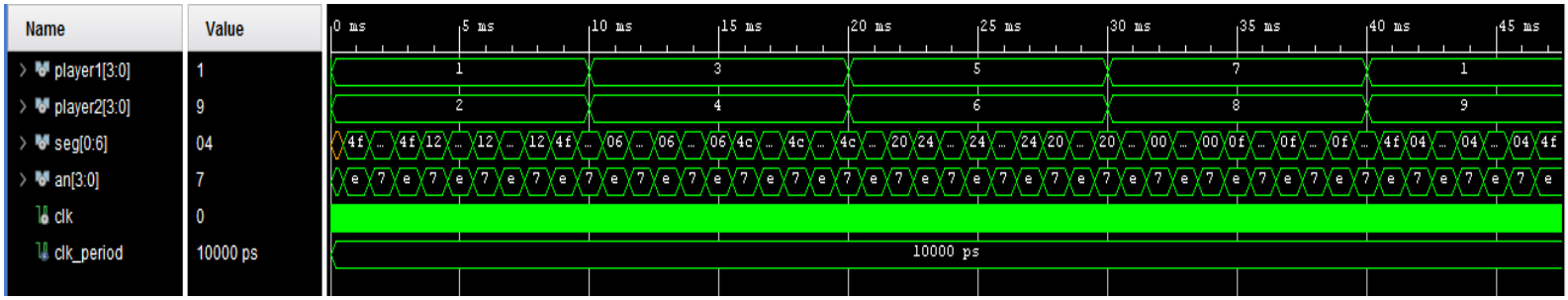
Following is the RTL schematic for the game:

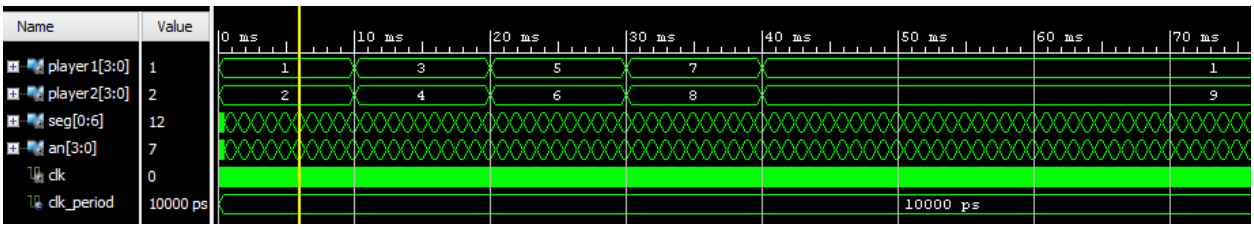


The RTL schematic displays the connections and the use of each module explained above. The three inputs are built in clock, Player 1 score from switches and player 2 switches where as these as displayed using two outputs seg and an for the built in seven segment display

Lab 4
Seven Segment Display
Haider Raheem
3/4/2020

Test Bench simulation results:



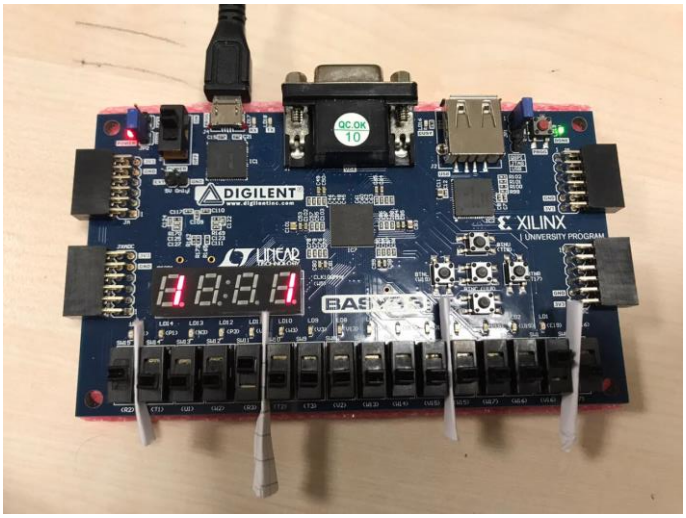


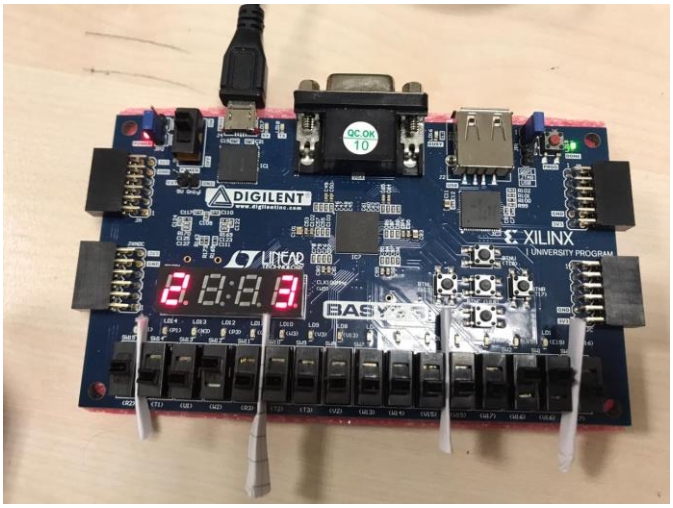
These results show how each set of four bits displays the scores of player 1 and player 2. Similarly, the waves from “an” display the interchanging displays due to clock from clock divider and Multiplexer. The “seg” is the signal given at the cathodes it changes with mux and clock too between the two scores changing due to clock.

Lab 4
Seven Segment Display
Haider Raheem
3/4/2020

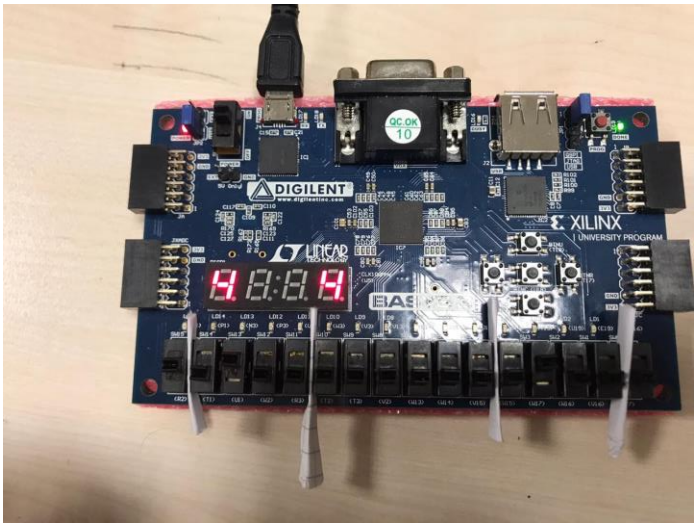
Hardware Results:

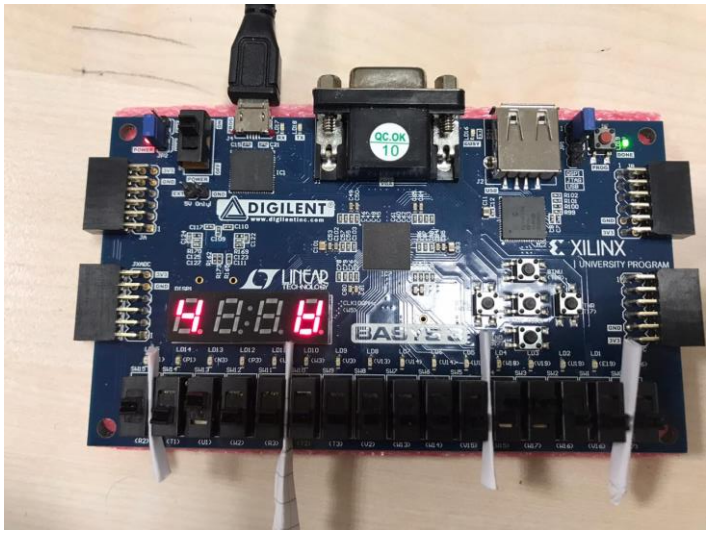
After programming the basys board the hardware implementation is displayed in the following pictures



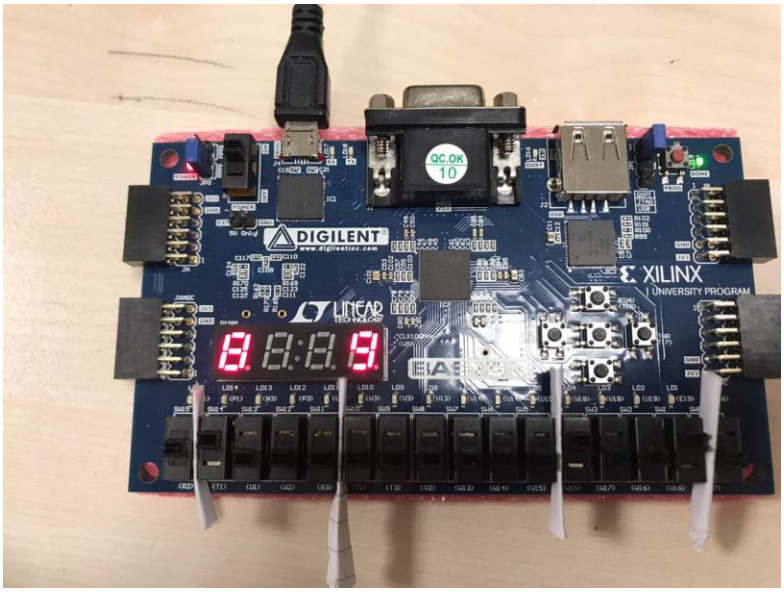


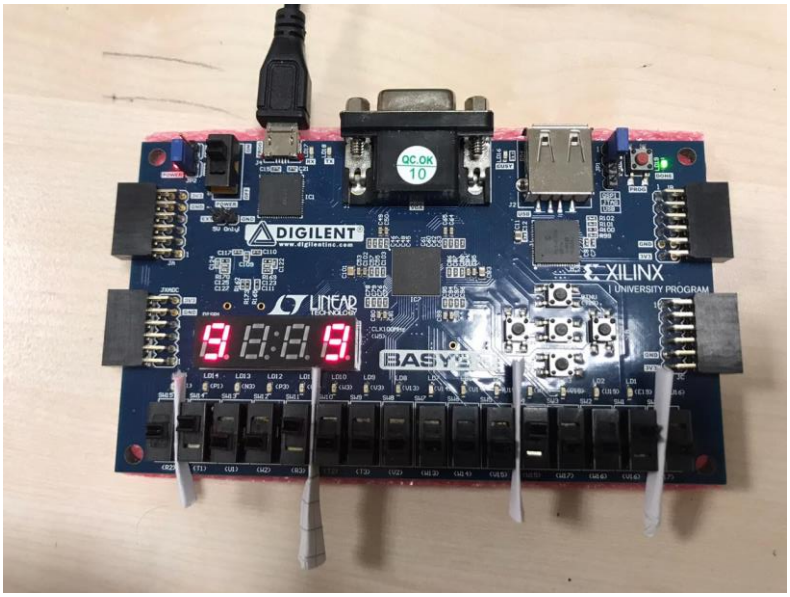
Lab 4
Seven Segment Display
Haider Raheem
3/4/2020





Lab 4
Seven Segment Display
Haider Raheem
3/4/2020





The pictures display the scores on two extreme sides of the display. One for Player1 and one for player two similarly the two set of controls is labelled using the papers in the photos. These results are similar to the proposed hardware implementation. The only thing different is not using the two extreme switch that is because my Basys 3 board's switches v17 and R2 are faulty.

Conclusion:

This Lab taught me how to code for the seven segment displays. I learnt to design Multiplexer, Decoder, clock divider and top module on Vivado using VHDL. Writing code for its test bench was a bit tricky as a clock was to be generated otherwise it was easy. I learnt depths of simulating results. One of the most important concept learnt in this lab was using modules and top modules to design and code via VHDL. The only hard part was understanding the working of the cathodes and anodes of the seven segment displays after it everything became easy making truth table and coding all got easy.

Appendices:

TOP MODULE VHDL CODE:

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

entity TopModule is
   Port ( clock : in STD_LOGIC;
        player1 : in STD_LOGIC_VECTOR (3 downto 0);
        player2 : in std_logic_vector(3 downto 0);
        seg : out STD_LOGIC_VECTOR (0 to 6);
        an : out STD_LOGIC_VECTOR (3 downto 0));
end TopModule;

architecture Behavioral of TopModule is

signal clk_tmp: STD_LOGIC;
signal player_1 : STD_LOGIC_VECTOR (0 to 6);
signal player_2 : STD_LOGIC_VECTOR (0 to 6);


component clkdiv is
   Port ( clk : in STD_LOGIC;
        clk_out: out STD_LOGIC);
end component;

component Dec is
   Port ( b_in : in STD_LOGIC_VECTOR (3 downto 0);
        cath_out : out STD_LOGIC_VECTOR (0 to 6));
end component;
```

```
component MUX is
   Port ( clk : in STD_LOGIC;
          player1 : in STD_LOGIC_VECTOR (0 to 6);
          player2 : in STD_LOGIC_VECTOR (0 to 6);
          Mux_out : out STD_LOGIC_VECTOR (0 to 6);
          an_out: out STD_LOGIC_VECTOR (3 downto 0));
end component;

begin

clockdivider : clkdiv PORT MAP(
clk => clock,
clk_out => clk_tmp);

segdecoder1 : Dec PORT MAP(
b_in => player1,
cath_out => player_1);

segdecoder2 : Dec PORT MAP(
b_in => player2,
cath_out => player_2);

mux0 : MUX PORT MAP(
clk => clk_tmp,
player1 => player_1,
player2 => player_2,
mux_out => seg,
an_out => an);

end Behavioral;
```

CLOCK DIVIDER VHDL CODE:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity clkdiv is
   Port ( clk : in STD_LOGIC;
          clk_out : out STD_LOGIC);
end clkdiv;

architecture clockdivider of clkdiv is
signal temp : std_logic := '0';
```

```
signal counter: integer:=0;


begin
process(clk)
begin
   if rising_edge(clk) then
      counter <= counter + 1;
      if (counter = 49999) then
      temp <= not temp;
      counter <= 0;
      end if;
   end if;
end process;

clk_out <= temp;
end clockdivider;
```

DECODER VHDL CODE:

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

entity Dec is
   Port ( b_in : in STD_LOGIC_VECTOR (3 downto 0);
          cath_out : out STD_LOGIC_VECTOR (0 to 6));
end Dec;

architecture Behavioral of Dec is
signal cath_tmp : STD_LOGIC_VECTOR (0 to 6);

begin
process(b_in, cath_tmp)
begin
   case b_in is
      when "0000" => cath_tmp <= "0000001"; -- "0"
      when "0001" => cath_tmp <= "1001111"; -- "1"
      when "0010" => cath_tmp <= "0010010"; -- "2"
      when "0011" => cath_tmp <= "0000110"; -- "3"
      when "0100" => cath_tmp <= "1001100"; -- "4"
      when "0101" => cath_tmp <= "0100100"; -- "5"
      when "0110" => cath_tmp <= "0100000"; -- "6"
      when "0111" => cath_tmp <= "0001111"; -- "7"
      when "1000" => cath_tmp<= "0000000"; -- "8"
      when "1001" => cath_tmp <= "0000100"; -- "9"
      when others => cath_tmp <= "1000000";
```

```
   end case;
end process;

cath_out <= cath_tmp;
end Behavioral;
```

MULTIPLEXER VHDL CODE:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity MUX is
   Port ( clk : in STD_LOGIC;
          player1 : in STD_LOGIC_VECTOR (0 to 6);
          player2 : in STD_LOGIC_VECTOR (0 to 6);
          Mux_out : out STD_LOGIC_VECTOR (0 to 6);
          an_out : out STD_LOGIC_VECTOR (3 downto 0));
end MUX;

architecture Behavioral of MUX is
signal mux_tmp : STD_LOGIC := '0';

begin

process(clk)
begin
   if rising_edge(clk) then
      mux_tmp <= not mux_tmp;
   end if;
end process;

process(mux_tmp)
begin
   if mux_tmp = '0' then
      Mux_out <= player2;
      an_out <= "0111";
   elsif mux_tmp = '1' then
      Mux_out <= player1;
      an_out <= "1110";
   end if;
end process;

end Behavioral;
```

TEST BENCH CODE:

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity TopModule_tb is
end;

architecture bench of TopModule_tb is

  component TopModule
     Port ( clock : in STD_LOGIC;
           player1 : in STD_LOGIC_VECTOR (3 downto 0);
           player2 : in std_logic_vector(3 downto 0);
           seg : out STD_LOGIC_VECTOR (0 to 6);
           an : out STD_LOGIC_VECTOR (3 downto 0));
  end component;

  signal player1: STD_LOGIC_VECTOR (3 downto 0);
  signal player2: std_logic_vector(3 downto 0);
  signal seg: STD_LOGIC_VECTOR (0 to 6);
  signal an: STD_LOGIC_VECTOR (3 downto 0);


  signal clk : STD_LOGIC;
  constant clk_period : time := 10 ns;


begin

  uut: TopModule port map ( clock   => clk,
                            player1 => player1,
                            player2 => player2,
                            seg     => seg,
                            an      => an );
   clocking : process
  begin
    clk <= '0';
    wait for clk_period/2;
```

```
   clk <= '1';
   wait for clk_period/2;
  end process;

  stimulus: process
  begin


    -- Put initialisation code here
player1<= "0000";
player2<= "0000";

    -- Put test bench stimulus code here
wait for 10ns;
player1<= "0001";
player2<= "0010";
wait for 10ms;
player1<= "0011";
player2<= "0100";
wait for 10ms;
player1<= "0101";
player2<= "0110";
wait for 10ms;
player1<= "0111";
player2<= "1000";
wait for 10ms;
player1<= "0001";
player2<= "1001";
    wait;
  end process;


end;
```

CONSTRAINTS:

```
set_property PACKAGE_PIN W5 [get_ports clock]
        set_property IOSTANDARD LVCMOS33 [get_ports clock]

set_property PACKAGE_PIN V16 [get_ports {player1[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {player1[0]}]
set_property PACKAGE_PIN W16 [get_ports {player1[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {player1[1]}]
set_property PACKAGE_PIN W17 [get_ports {player1[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {player1[2]}]
```

```
set_property PACKAGE_PIN W15 [get_ports {player1[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {player1[3]}]

set_property PACKAGE_PIN R3 [get_ports {player2[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {player2[0]}]
set_property PACKAGE_PIN W2 [get_ports {player2[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {player2[1]}]
set_property PACKAGE_PIN U1 [get_ports {player2[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {player2[2]}]
set_property PACKAGE_PIN T1 [get_ports {player2[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {player2[3]}]
```

##7 segment display

```
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]


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