COL 215 Lab Project Final Report

7up 7down

Made By: Sankalan Pal Chowdhury (2016CS10701) and Shreshth Tuli (2016CS10680)

UPDATED CODE

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


entity Random_Number_Generator is
  Port (
  clk : IN STD_LOGIC;
  random1 : INOUT STD_LOGIC_VECTOR(3 DOWNTO 0);
  init : IN STD_LOGIC:='0'
    );
end Random_Number_Generator;

architecture Behavioral of Random_Number_Generator is
    SIGNAL cnt : std_logic_vector(5 downto 0):="101001";
begin

process(clk)
begin
    if(clk='1' and clk'event) then
            if(init='1') then
            cnt(2 downto 0)<=std_logic_vector(unsigned(cnt(2 downto 0))+1);
                if(cnt(2 downto 0)="110") then
                    cnt(2 downto 0)<="001";
                    cnt(5 downto 3)<=std_logic_vector(unsigned(cnt(5 downto 3))+1);
                    if(cnt(5 downto 3)="110") then
                        cnt(5 downto 3)<="001";
                    end if;
                end if;
            else
                random1<=std_logic_vector(unsigned('0' & cnt(5 downto 3))+unsigned('0' & cnt(2 downto 0)));
            end if;

    end if;
END PROCESS;

end Behavioral;

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


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.CONV_STD_LOGIC_VECTOR;
use IEEE.NUMERIC_STD.ALL;


entity Control is
  Port (
  clk : IN STD_LOGIC;
  bid1 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  bid2 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  win1 : INOUT STD_LOGIC;
  win2 : INOUT STD_LOGIC;
  bid1_invalid : INOUT STD_LOGIC;
  bid2_invalid : INOUT STD_LOGIC;
  bid1_invalid_25 : INOUT STD_LOGIC;
  bid2_invalid_25 : INOUT STD_LOGIC;
  bid1_invalid_high : INOUT STD_LOGIC;
  bid2_invalid_high : INOUT STD_LOGIC;
  ssd : INOUT STD_LOGIC_VECTOR(15 DOWNTO 0):="0000101100011101";
  random1 : INOUT STD_LOGIC_VECTOR(3 DOWNTO 0);
  init : IN STD_LOGIC:='0';
  p1D : IN STD_LOGIC;
  p2D : IN STD_LOGIC;
  taxD : IN STD_LOGIC
    );
end Control;

architecture Behavioral of Control is
    SIGNAL p1_bid : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL p2_bid : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL p1_money : STD_LOGIC_VECTOR(10 DOWNTO 0):="01111110000";
    SIGNAL p2_money : STD_LOGIC_VECTOR(10 DOWNTO 0):="01111110000";
    SIGNAL tax : STD_LOGIC_VECTOR(10 DOWNTO 0):="00000000000";
    SIGNAL random : STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL cnt : std_logic_vector(5 downto 0):="101001";
    SIGNAL state : STD_LOGIC_VECTOR(2 DOWNTO 0):="000";
    SIGNAL counter, cycle : INTEGER Range 0 to 1100000000:=0;
    SIGNAL one_bid_done : STD_LOGIC:='0';
    SIGNAL p1p2 : STD_LOGIC_VECTOR(1 DOWNTO 0);
begin

process(clk)
begin
    if(clk='1' and clk'event) then
      bid1_invalid <= bid1_invalid_25 or bid1_invalid_high;
      bid2_invalid <= bid2_invalid_25 or bid2_invalid_high;

    --P1 bids
    if state = "000" then
        p1p2 <= "10";
        ssd <= "1011000111010001";
        p1_bid <= bid1;
        win1<='0';
        win2<='0';
        counter <= counter + 1;
        if(one_bid_done = '1') then
            if(unsigned(p1_bid(6 DOWNTO 0))/4>unsigned(p1_money)) then bid1_invalid_25<='1'; else
bid1_invalid_25<='0'; end if;
        else
            if(unsigned(p1_bid(6 DOWNTO 0))/4>unsigned(p1_money)-unsigned(p2_bid(6 DOWNTO 0))) then
bid1_invalid_25<='1'; else bid1_invalid_25<='0'; end if;
        end if;
        if(unsigned(p1_bid(6 DOWNTO 0))>unsigned(p2_money)) then bid1_invalid_high<='1'; else
bid1_invalid_high<='0'; end if;
        if(counter>100000000 and init='1' and bid1_invalid='0') then
            tax <= STD_LOGIC_VECTOR(unsigned(tax) + unsigned(p1_bid(6 DOWNTO 0))/4);
            p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money) - unsigned(p1_bid(6 DOWNTO 0))/4);
            if(one_bid_done='0') then state <= "001"; one_bid_done <= '1'; counter<=0; else state <= "010";
one_bid_done <= '0'; counter<=0; end if;
        end if;

   --P2 bids
   elsif state = "001" then
       p1p2 <= "01";
       ssd <= "1011000111010010";
       p2_bid <= bid2;
       win1<='0';
       win2<='0';
       counter <= counter + 1;
       if(one_bid_done = '0') then
        if(unsigned(p2_bid(6 DOWNTO 0))/4>unsigned(p2_money)) then bid2_invalid_25<='1'; else
bid2_invalid_25<='0'; end if;
       else
        if(unsigned(p2_bid(6 DOWNTO 0))/4>unsigned(p2_money)-unsigned(p1_bid(6 DOWNTO 0))) then
bid2_invalid_25<='1'; else bid2_invalid_25<='0'; end if;
       end if;
       if(unsigned(p2_bid(6 DOWNTO 0))>unsigned(p1_money)) then bid2_invalid_high<='1'; else
bid2_invalid_high<='0'; end if;
       if(counter>100000000 and init='1' and bid2_invalid='0') then
           tax <= STD_LOGIC_VECTOR(unsigned(tax) + unsigned(p2_bid(6 DOWNTO 0))/4);
           p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money) - unsigned(p2_bid(6 DOWNTO 0))/4);
           if(one_bid_done='0') then state <= "000"; one_bid_done <= '1'; counter<=0; else state <= "010";
one_bid_done <= '0'; counter<=0; end if;
        end if;

   --Take random number
   elsif state = "010" then
        if(init='0') then
            random <= random1;
            state <= "011";
        end if;

   --Calculations of money of each w.r.t random number
   elsif state = "011" then
        p1p2 <= "00";
        counter<=0;
        if(unsigned(random)>7 and p1_bid(7)='1' and p2_bid(7)='0') then
            p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)+unsigned(p1_bid(6 DOWNTO 0)));
            p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)-unsigned(p1_bid(6 DOWNTO 0)));
            win1<='1'; win2<='0';
        elsif(unsigned(random)>7 and p2_bid(7)='1' and p1_bid(7)='0') then
```

```
            p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)+unsigned(p2_bid(6 DOWNTO 0)));
            p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)-unsigned(p2_bid(6 DOWNTO 0)));
            win1<='0'; win2<='1';
        elsif(unsigned(random)>7 and p2_bid(7)='1' and p1_bid(7)='1') then
            p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)+unsigned(p2_bid(6 DOWNTO 0))-unsigned(p1_bid(6
DOWNTO 0)));
            p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)+unsigned(p1_bid(6 DOWNTO 0))-unsigned(p2_bid(6
DOWNTO 0)));
            win1<='1'; win2<='1';
        elsif(unsigned(random)<7 and p1_bid(7)='0' and p2_bid(7)='1') then
            p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)+unsigned(p1_bid(6 DOWNTO 0)));
            p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)-unsigned(p1_bid(6 DOWNTO 0)));
            win1<='1'; win2<='0';
        elsif(unsigned(random)<7 and p2_bid(7)='0' and p1_bid(7)='1') then
            p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)+unsigned(p2_bid(6 DOWNTO 0)));
            p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)-unsigned(p2_bid(6 DOWNTO 0)));
            win1<='0'; win2<='1';
        elsif(unsigned(random)<7 and p2_bid(7)='0' and p1_bid(7)='0') then
            p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)+unsigned(p2_bid(6 DOWNTO 0))-unsigned(p1_bid(6
DOWNTO 0)));
            p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)+unsigned(p1_bid(6 DOWNTO 0))-unsigned(p2_bid(6
DOWNTO 0)));
            win1<='1'; win2<='1';
        elsif(unsigned(random)=7 and unsigned(p1_bid(6 DOWNTO 0))>unsigned(p2_bid(6 DOWNTO 0))) then
            p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)+unsigned(p1_bid(6 DOWNTO 0)));
            p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)-unsigned(p1_bid(6 DOWNTO 0)));
            win1<='1'; win2<='0';
        elsif(unsigned(random)=7 and unsigned(p2_bid(6 DOWNTO 0))>unsigned(p1_bid(6 DOWNTO 0))) then
            p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)+unsigned(p2_bid(6 DOWNTO 0)));
            p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)-unsigned(p2_bid(6 DOWNTO 0)));
            win1<='0'; win2<='1';
        end if;
        state <= "100";

    --Display
    elsif state = "100" then
        counter <= counter+1;
        if(counter<20000000) then
            ssd <= "1111111111111111";
        elsif(counter<220000000) then --Display random number
            ssd(15 DOWNTO 8) <= "11111111";
            ssd(7 DOWNTO 4) <= STD_LOGIC_VECTOR(unsigned(random)/10);
            ssd(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(random)) -
10*to_integer(unsigned(random)/10),4);
        elsif(counter<240000000) then
                        ssd <= "1111111111111111";
        elsif(counter<440000000) then --Display who won
            if(win1='1' and win2='1') then ssd<="1111000111110010";
            elsif(win1='0' and win2='0') then ssd<="1111111111110000";
            elsif(win1='1' and win2='0') then ssd<="1111111111110001";
            elsif(win1='0' and win2='1') then ssd<="1111111111110010";
            end if;
        elsif(counter<460000000) then
            ssd <= "1111111111111111";
        elsif(counter<660000000) then --Display player 1 money
            ssd(15 DOWNTO 12) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)/1000),4);
            ssd(11 DOWNTO 8) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)/100) -
10*to_integer(unsigned(p1_money)/1000),4);
            ssd(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)/10) -
10*to_integer(unsigned(p1_money)/100),4);
            ssd(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)) -
10*to_integer(unsigned(p1_money)/10),4);
        elsif(counter<680000000) then
            ssd <= "1111111111111111";
        elsif(counter<880000000) then --Display player 2 money
            ssd(15 DOWNTO 12) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)/1000),4);
            ssd(11 DOWNTO 8) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)/100) -
10*to_integer(unsigned(p2_money)/1000),4);
            ssd(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)/10) -
10*to_integer(unsigned(p2_money)/100),4);
            ssd(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)) -
10*to_integer(unsigned(p2_money)/10),4);
        elsif(counter<900000000) then
            ssd <= "1111111111111111";
        elsif(counter<1100000000) then --Display tax
            ssd(15 DOWNTO 12) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(tax)/1000),4);
            ssd(11 DOWNTO 8) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(tax)/100) -
10*to_integer(unsigned(tax)/1000),4);
            ssd(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(tax)/10) -
10*to_integer(unsigned(tax)/100),4);
```

```
            ssd(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(tax)) -
10*to_integer(unsigned(tax)/10),4);
        elsif(counter=1100000000) then
            if(p2_money = "00000000000") then
                p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money) + unsigned(tax));
                state<="101"; counter<=0;
            elsif(p1_money = "00000000000") then
                p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money) + unsigned(tax));
                state<="101"; counter<=0;
            elsif(cycle mod 2 = 1) then
                state <= "000"; cycle <= cycle + 1;counter<=0; one_bid_done <= '0';
            else
                state <= "001"; cycle <= cycle + 1;counter<=0; one_bid_done <= '0';
            end if;
        end if;

    --Display Winner and reset for new game
    elsif state = "101" then
        counter <= counter+1;
        if(p1_money = "00000000000") then
            if counter < 50000000 then ssd <= "1111111111110001";
            elsif counter < 100000000 then ssd <= "1111111100011111";
            elsif counter < 150000000 then ssd <= "1111000111111111";
            elsif counter < 200000000 then ssd <= "0001111111111111";
            elsif(counter<250000000) then
                ssd <= "1111111111111111";
            elsif counter < 450000000 then
                ssd(15 DOWNTO 12) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)/1000),4);
                ssd(11 DOWNTO 8) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)/100) -
10*to_integer(unsigned(p2_money)/1000),4);
                ssd(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)/10) -
10*to_integer(unsigned(p2_money)/100),4);
                ssd(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)) -
10*to_integer(unsigned(p2_money)/10),4);
            elsif(counter<500000000) then
                ssd <= "1111111111111111";
            elsif counter < 700000000 then ssd <= "1111111111111111";
            elsif counter = 700000000 then
                ssd <= "1011000111010001";
                state <= "000"; counter<=0; cycle<=0;
                p1_money<="01111110000";
                p2_money<="01111110000";
                tax<="00000000000";
                one_bid_done <= '0';
            end if;
        elsif(p2_money = "00000000000") then
            if counter < 50000000 then ssd <= "0010111111111111";
            elsif counter < 100000000 then ssd <= "1111001011111111";
            elsif counter < 150000000 then ssd <= "1111111100101111";
            elsif counter < 200000000 then ssd <= "1111111111110010";
            elsif(counter<250000000) then
                ssd <= "1111111111111111";
            elsif counter < 450000000 then
                ssd(15 DOWNTO 12) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)/1000),4);
                ssd(11 DOWNTO 8) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)/100) -
10*to_integer(unsigned(p1_money)/1000),4);
                ssd(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)/10) -
10*to_integer(unsigned(p1_money)/100),4);
                ssd(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)) -
10*to_integer(unsigned(p1_money)/10),4);
            elsif(counter<500000000) then
                ssd <= "1111111111111111";
            elsif counter < 700000000 then ssd <= "1111111111111111";
            elsif counter = 700000000 then
                ssd <= "1011000111010001";
                state <= "000"; counter<=0; cycle<=0;
                p1_money<="01111110000";
                p2_money<="01111110000";
                tax<="00000000000";
                one_bid_done <= '0';
            end if;
        end if;
    end if;

    if p1D = '1' then
        ssd(15 DOWNTO 12) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)/1000),4);
        ssd(11 DOWNTO 8) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)/100) -
10*to_integer(unsigned(p1_money)/1000),4);
        ssd(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)/10) -
10*to_integer(unsigned(p1_money)/100),4);
```

```
        ssd(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p1_money)) -
10*to_integer(unsigned(p1_money)/10),4);
    elsif p2D = '1' then
        ssd(15 DOWNTO 12) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)/1000),4);
        ssd(11 DOWNTO 8) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)/100) -
10*to_integer(unsigned(p2_money)/1000),4);
        ssd(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)/10) -
10*to_integer(unsigned(p2_money)/100),4);
        ssd(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(p2_money)) -
10*to_integer(unsigned(p2_money)/10),4);
    elsif taxD = '1' then
        ssd(15 DOWNTO 12) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(tax)/1000),4);
        ssd(11 DOWNTO 8) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(tax)/100) -
10*to_integer(unsigned(tax)/1000),4);
        ssd(7 DOWNTO 4) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(tax)/10) -
10*to_integer(unsigned(tax)/100),4);
        ssd(3 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(to_integer(unsigned(tax)) -
10*to_integer(unsigned(tax)/10),4);
    end if;

    end if;


END PROCESS;

end Behavioral;


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


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.CONV_STD_LOGIC_VECTOR;
use IEEE.NUMERIC_STD.ALL;


entity Project_7u7d is
  Port (
  clk : IN STD_LOGIC;
  bid1 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  bid2 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  win1 : INOUT STD_LOGIC;
  win2 : INOUT STD_LOGIC;
  bid1_invalid : INOUT STD_LOGIC;
  bid2_invalid : INOUT STD_LOGIC;
  bid1_invalid_25 : INOUT STD_LOGIC;
  bid2_invalid_25 : INOUT STD_LOGIC;
  bid1_invalid_high : INOUT STD_LOGIC;
  bid2_invalid_high : INOUT STD_LOGIC;
  cathode : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
  anode : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
  test : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
  p1D : IN STD_LOGIC;
  p2D : IN STD_LOGIC;
  taxD : IN STD_LOGIC;
  init : IN STD_LOGIC:='0'
    );
end Project_7u7d;

architecture Behavioral of Project_7u7d is
    SIGNAL ssd : STD_LOGIC_VECTOR(15 DOWNTO 0):="0000101100011101";
    SIGNAL random1 : STD_LOGIC_VECTOR(3 DOWNTO 0);
begin

Display:
ENTITY work.lab4_seven_segment_display
port map(
    clk=>clk,
    cathode=>cathode,
    anode=>anode,
    b=>ssd
    );

Random_Number_Generator:
ENTITY work.Random_Number_Generator
port map(
    clk => clk,
    random1 => random1,
    init => init
    );
```

```
Control:
ENTITY work.Control
port map(
    clk => clk,
    bid1 => bid1,
    bid2 => bid2,
    win1 => win1,
    win2 => win2,
    bid1_invalid => bid1_invalid,
    bid2_invalid => bid2_invalid,
    bid1_invalid_25 => bid1_invalid_25,
    bid2_invalid_25 => bid2_invalid_25,
    bid1_invalid_high => bid1_invalid_high,
    bid2_invalid_high => bid2_invalid_high,
    random1 => random1,
    ssd => ssd,
    init=> init,
    p1D => p1D,
    p2D => p2D,
    taxD => taxD
    );


end Behavioral;
```

UPDATED 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
#Bank = 34, Pin name = ,                     Sch name = CLK100MHZ
#       set_property PACKAGE_PIN W5 [get_ports CLK]
#       set_property IOSTANDARD LVCMOS33 [get_ports CLK]
#       create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports
CLK]

create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]
# Switches
set_property PACKAGE_PIN R2 [get_ports bid1[7]]
set_property IOSTANDARD LVCMOS33 [get_ports bid1[7]]
set_property PACKAGE_PIN T1 [get_ports bid1[6]]
set_property IOSTANDARD LVCMOS33 [get_ports bid1[6]]
set_property PACKAGE_PIN U1 [get_ports bid1[5]]
set_property IOSTANDARD LVCMOS33 [get_ports bid1[5]]
set_property PACKAGE_PIN W2 [get_ports bid1[4]]
set_property IOSTANDARD LVCMOS33 [get_ports bid1[4]]
set_property PACKAGE_PIN R3 [get_ports bid1[3]]
set_property IOSTANDARD LVCMOS33 [get_ports bid1[3]]
set_property PACKAGE_PIN T2 [get_ports bid1[2]]
set_property IOSTANDARD LVCMOS33 [get_ports bid1[2]]
set_property PACKAGE_PIN T3 [get_ports bid1[1]]
set_property IOSTANDARD LVCMOS33 [get_ports bid1[1]]
set_property PACKAGE_PIN V2 [get_ports bid1[0]]
set_property IOSTANDARD LVCMOS33 [get_ports bid1[0]]
set_property PACKAGE_PIN W13 [get_ports bid2[7]]
set_property IOSTANDARD LVCMOS33 [get_ports bid2[7]]
set_property PACKAGE_PIN W14 [get_ports bid2[6]]
set_property IOSTANDARD LVCMOS33 [get_ports bid2[6]]
set_property PACKAGE_PIN V15 [get_ports bid2[5]]
set_property IOSTANDARD LVCMOS33 [get_ports bid2[5]]
set_property PACKAGE_PIN W15 [get_ports bid2[4]]
set_property IOSTANDARD LVCMOS33 [get_ports bid2[4]]
set_property PACKAGE_PIN W17 [get_ports bid2[3]]
set_property IOSTANDARD LVCMOS33 [get_ports bid2[3]]
set_property PACKAGE_PIN W16 [get_ports bid2[2]]
set_property IOSTANDARD LVCMOS33 [get_ports bid2[2]]
set_property PACKAGE_PIN V16 [get_ports bid2[1]]
set_property IOSTANDARD LVCMOS33 [get_ports bid2[1]]
set_property PACKAGE_PIN V17 [get_ports bid2[0]]
set_property IOSTANDARD LVCMOS33 [get_ports bid2[0]]

set_property PACKAGE_PIN U18 [get_ports init]
set_property IOSTANDARD LVCMOS33 [get_ports init]
set_property PACKAGE_PIN W19 [get_ports p1D]
set_property IOSTANDARD LVCMOS33 [get_ports p1D]
set_property PACKAGE_PIN T17 [get_ports p2D]
set_property IOSTANDARD LVCMOS33 [get_ports p2D]
set_property PACKAGE_PIN T18 [get_ports taxD]
set_property IOSTANDARD LVCMOS33 [get_ports taxD]


set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]


# LEDs

set_property PACKAGE_PIN W18 [get_ports test[1]]
set_property IOSTANDARD LVCMOS33 [get_ports test[1]]
set_property PACKAGE_PIN V19 [get_ports test[0]]
set_property IOSTANDARD LVCMOS33 [get_ports test[0]]

set_property PACKAGE_PIN U2 [get_ports anode[0]]
set_property IOSTANDARD LVCMOS33 [get_ports anode[0]]
set_property PACKAGE_PIN U4 [get_ports anode[1]]
set_property IOSTANDARD LVCMOS33 [get_ports anode[1]]
set_property PACKAGE_PIN V4 [get_ports anode[2]]
set_property IOSTANDARD LVCMOS33 [get_ports anode[2]]
set_property PACKAGE_PIN W4 [get_ports anode[3]]
set_property IOSTANDARD LVCMOS33 [get_ports anode[3]]

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

set_property PACKAGE_PIN L1 [get_ports bid1_invalid]
set_property IOSTANDARD LVCMOS33 [get_ports bid1_invalid]
set_property PACKAGE_PIN P1 [get_ports bid1_invalid_25]
set_property IOSTANDARD LVCMOS33 [get_ports bid1_invalid_25]
set_property PACKAGE_PIN N3 [get_ports bid1_invalid_high]
set_property IOSTANDARD LVCMOS33 [get_ports bid1_invalid_high]
set_property PACKAGE_PIN V13 [get_ports win1]
set_property IOSTANDARD LVCMOS33 [get_ports win1]

set_property PACKAGE_PIN U16 [get_ports bid2_invalid]
set_property IOSTANDARD LVCMOS33 [get_ports bid2_invalid]
set_property PACKAGE_PIN E19 [get_ports bid2_invalid_25]
set_property IOSTANDARD LVCMOS33 [get_ports bid2_invalid_25]
set_property PACKAGE_PIN U19 [get_ports bid2_invalid_high]
set_property IOSTANDARD LVCMOS33 [get_ports bid2_invalid_high]
set_property PACKAGE_PIN U14 [get_ports win2]
set_property IOSTANDARD LVCMOS33 [get_ports win2]



# Others (BITSTREAM, CONFIG)
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]

set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]

set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```

CODE DESCRIPTION

*Note: *Text* represents internal signals / port, ***Text*** represents design modules / entities.

The code is divided into 3 parts:

1. Random Number Generator
2. Control
3. Display

Display

The Seven Segment Display has been imported as a component from the code developed for Lab 4. The *ssd* signal used in the architecture of entity ***Control*** and ***Project_7u7d*** corresponds to the 16-bit input for the Lab 4 Module. Each 4-bit set of the *ssd* output represents the binary representation of the corresponding digit of the Seven Segment Display. The other ports including *cathode*, *anode* and *clk* (clock) have been mapped with this component.

Random Number Generator

The random number generator relies on the noise in the time for which the *init* button is pressed. Since this would be manually done, the resolution cannot be more than ≈ 100 Hz (refer to this). If measured to the accuracy of 10^{-8} seconds, we can assume the last few significant figures to be random and uniformly distributed. We use two counters both resetting at 6 to 1. One of these gets incremented at every clock cycle (when *init* is high) and the other gets incremented when the first goes from 6 to 1. These two independently simulate two dices. When *init* becomes low, the sum of the numbers on the two dice gets fed to the signal r*andom1*. In this way, *random1* gets a new (random) value whenever there is a pulse on *init*. When needed, it is stored into another signal *random*, which is then used.

Control

In the Control part of the code most of the ASM functionalities have been achieved. The overall Control block has been divided into 5 states (the buffer states in ASM have been combined for ease of code, but may be separated later; if required).

1. State 1:

   The *state* signal has been used to indicates the state of the system. The first state corresponding to *state* = “000”. The signal *p1p2* shows which one of the two players needs to bid. As per the ASM the player who bids first alternates between the two players. “10” corresponds to player 1 bidding first and “01” corresponds to player 2 bidding. In state 1, the signal *p1_bid* stores the bid amount of player 1. The 7 least significant bits of *bid1* represent value of bid from 0 to 127 and *bid(7)* = *p1_bid(7)* represents if the bid is for >7 or <7.

   Also, *win1* and *wins2* are two signals which store the winning player who won the current bid. They have to be ‘0’ till the bidding process is not complete*. Bid1_invalid*, *bid1_invalid_25* and *bid1_invalid_high* are three signals which show if the bid set by player 1 is valid or not. *Bid1_invalid_25* is ‘1’ if the player does not have 25% of the bid value in his purse, *bid1_invalid_high* is ‘1’ if the bid value is higher than the money in other player’s purse and *bid1_invalid* is ‘1’ in either of the two cases. The bid is not accepted if the *bid1_invalid* is high.

   When the player 1 has done bidding and bid is valid and he presses the *init* button then 25% of bid value is deducted from his purse *p1_money* (1008 initially) and added to *tax*. The signal *one_bid_done* is to store if player 1 had done first or player 2. If player 1 is first then *one_bid_done* is ‘0’ so goes to state 2 , otherwise if *one_bid_done* is ‘1’ then player 1 is the second bidder hence goes to state 3. The *counter* for timing is reset to 0.

2. State 2:
   This state is for player 2 bidding similar to that of player 1 as in state 1, but here the *p1p2* signal is “01” and not “10”.

3. State 3:
   Stores the random generated from the ***Random_Number_Generator*** as *random* signal for comparisons with *p1_bid* and *p2_bid* and calculation of outcome.

4. State 4:
   This state calculates the money in each of the two purses *p1_money* and *p2_money* as per the Game Description (in Design Document 1) and also shows the player winning the current bid. The values of *random*, *p1_bid* and *p2_bid* are compared to calculate and assign values to *p1_money* and *p2_money*. *Win1* and *win2* show if player 1, or player 2 has won or lost. *Win*1 is ‘1’ if player 1 has won his/her bid, and *win2* is ‘1’ if player 2 has won his/her bid. Note: it is possible that both win or lose their bids simultaneously.

5. State 5:
   This state displays the following sequentially each for 2 seconds:
   - Random Number 2 to 12
   - Which player win the current bid: 1 or 2 or both or none
   - Money left with Player 1
   - Money left with player 2
   - Tax

6. State 6:
   This state displays the winner for 2 seconds then his/her money (after adding the tax) then resets all signals and starts new game. At start the first bidding player is player 1.

There are certain state overrides for ease of play for the players.

1. Display of money of each player – Two ports *p1D* and *p2D* override the display on *ssd* with the money with the corresponding player
2. Display of tax – One port *taxD* overrides the display on *ssd* with the current tax

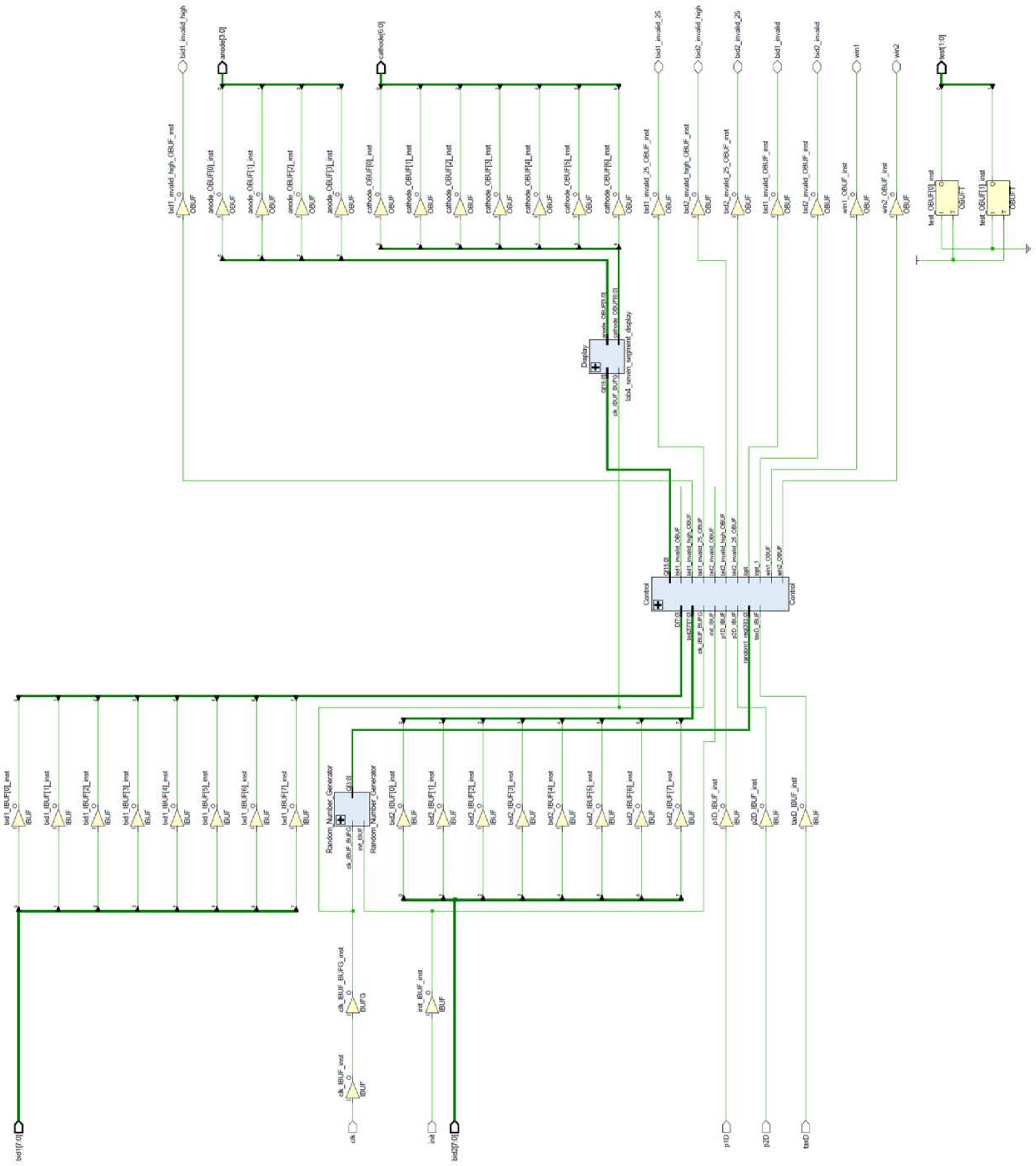
Also, for ease of distinguishing the money of each player, and state transitions small delays with no display are incorporated.

Synthesis Reports

```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
	   2 Input     31 Bit       Adders := 2
	   3 Input     11 Bit       Adders := 6
	   2 Input     11 Bit       Adders := 6
	   2 Input      4 Bit       Adders := 1
	   3 Input      4 Bit       Adders := 10
	   2 Input      3 Bit       Adders := 2
	   2 Input      2 Bit       Adders := 1
+---Registers :
	               31 Bit    Registers := 2
	               16 Bit    Registers := 1
	               11 Bit    Registers := 3
	                8 Bit    Registers := 2
	                6 Bit    Registers := 1
	                4 Bit    Registers := 2
	                2 Bit    Registers := 1
	                1 Bit    Registers := 25
+---Multipliers :
	                 4x4  Multipliers := 6
+---Muxes :
	   2 Input     31 Bit        Muxes := 23
	   4 Input     31 Bit        Muxes := 1
	   2 Input     16 Bit        Muxes := 24
	   5 Input     16 Bit        Muxes := 1
	   3 Input     16 Bit        Muxes := 1
	   4 Input     16 Bit        Muxes := 1
	   2 Input     11 Bit        Muxes := 17
	   4 Input     11 Bit        Muxes := 2
	   3 Input      6 Bit        Muxes := 1
	   2 Input      6 Bit        Muxes := 1
	   4 Input      4 Bit        Muxes := 1
	   2 Input      3 Bit        Muxes := 1
	   6 Input      3 Bit        Muxes := 1
	  11 Input      3 Bit        Muxes := 1
	   2 Input      1 Bit        Muxes := 66
	   4 Input      1 Bit        Muxes := 3
	   8 Input      1 Bit        Muxes := 1
	   9 Input      1 Bit        Muxes := 1
	   7 Input      1 Bit        Muxes := 2
	   6 Input      1 Bit        Muxes := 4
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------


Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |AND2   |    52|
|2     |AND3   |    14|
|3     |BUFG   |     1|
|4     |CARRY4 |    88|
|5     |INV    |     8|
|6     |LUT1   |    90|
|7     |LUT2   |   251|
|8     |LUT3   |    83|
|9     |LUT4   |   134|
|10    |LUT5   |    70|
|11    |LUT6   |   287|
|12    |NOR4   |     2|
|13    |NOR5   |     5|
|14    |OR2    |     1|
|15    |OR4    |     4|
|16    |FDRE   |   141|
|17    |IBUF   |    21|
|18    |OBUF   |    19|
|19    |OBUFT  |     2|
+------+-------+------+
```

Design Schematic



Implementation Report

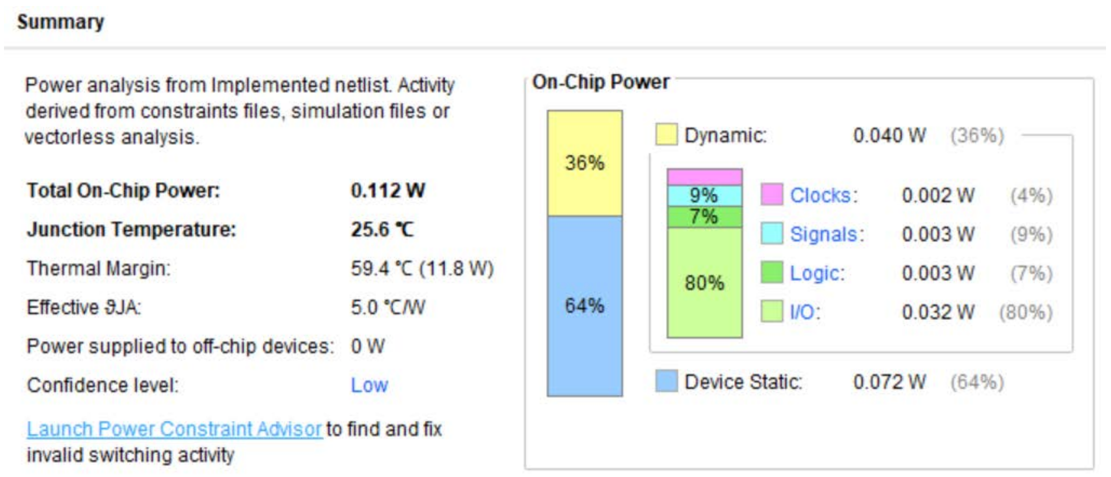
```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |  835 |     0 |     20800 |  4.01 |
|   LUT as Logic          |  835 |     0 |     20800 |  4.01 |
|   LUT as Memory         |    0 |     0 |      9600 |  0.00 |
| Slice Registers         |  141 |     0 |     41600 |  0.34 |
|   Register as Flip Flop |  141 |     0 |     41600 |  0.34 |
|   Register as Latch     |    0 |     0 |     41600 |  0.00 |
| F7 Muxes                |    0 |     0 |     16300 |  0.00 |
| F8 Muxes                |    0 |     0 |      8150 |  0.00 |
+-------------------------+------+-------+-----------+-------+

2. Memory
---------

+----------------+------+-------+-----------+-------+
|    Site Type   | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| Block RAM Tile |    0 |     0 |        50 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |        50 |  0.00 |
|   RAMB18       |    0 |     0 |       100 |  0.00 |
+----------------+------+-------+-----------+-------+
```

Power Utilization Report



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

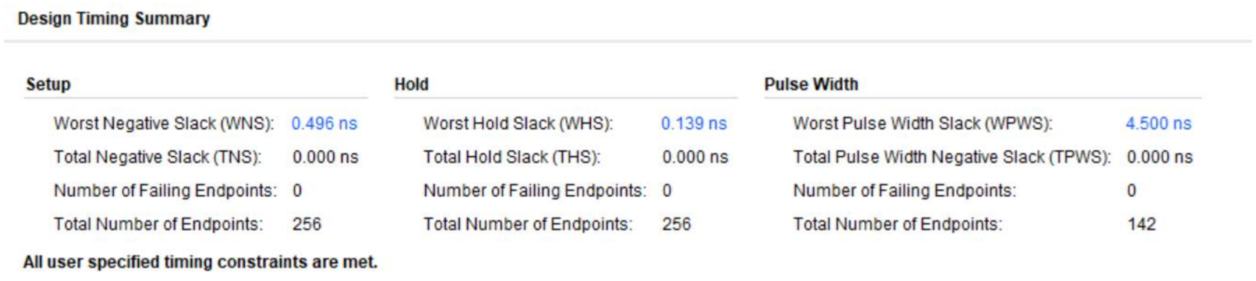
Total On-Chip Power:	**0.112 W**
Junction Temperature:	**25.6 °C**
Thermal Margin:	59.4 °C (11.8 W)
Effective ϑJA:	5.0 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

	Dynamic:	0.040 W	(36%)
	Clocks:	0.002 W	(4%)
	Signals:	0.003 W	(9%)
	Logic:	0.003 W	(7%)
	I/O:	0.032 W	(80%)
	Device Static:	0.072 W	(64%)

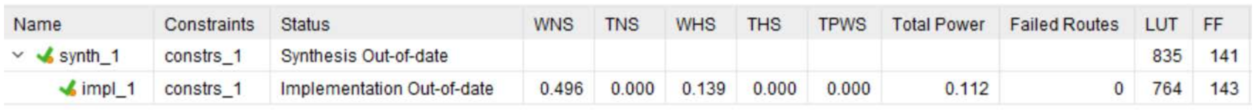
Timing Summary


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.496 ns	Worst Hold Slack (WHS):	0.139 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	256	Total Number of Endpoints:	256	Total Number of Endpoints:	142

All user specified timing constraints are met.

Design Run

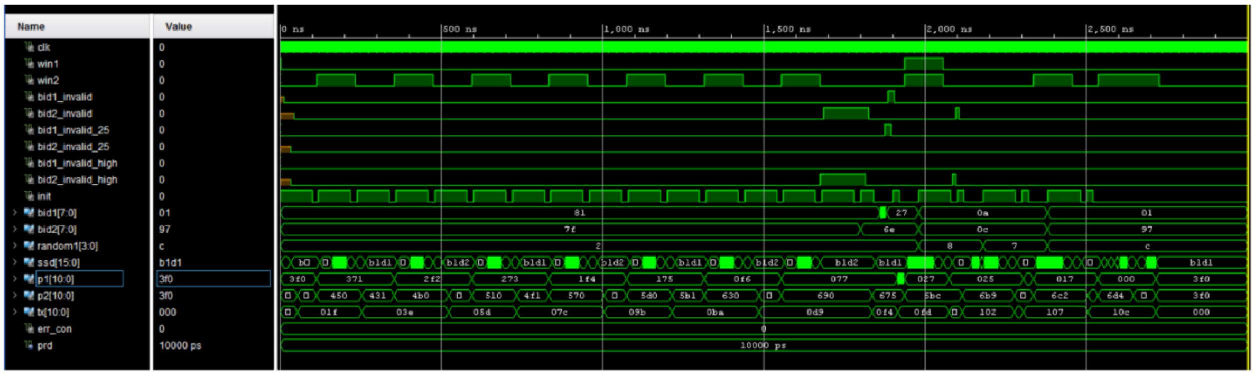


Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF
✓ synth_1	constrs_1	Synthesis Out-of-date								835	141
✓ impl_1	constrs_1	Implementation Out-of-date	0.496	0.000	0.139	0.000	0.000	0.112	0	764	143

SIMULATION USING TESTBENCH

The test bench we are currently using tests for the Control part only. For ease of testing, we created out ports corresponding to the *p1_money*, *p2_money* and *tax* signals (visible as p1,p2 and tx). The simulation consists of a single iteration of the game. We race past the first few rounds, but in the last rounds, we check for several possible conditions.

Other modules like ***lab4_seven_segment_display*** and ***Random_Number_Generator*** have not been tested on testbench as the prior has been tested and tried many times in previous labs, and the latter is dependent on randomness of human response time.



Timing Diagram for below testench file

Testbench file

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Control_tb is
end Control_tb;

architecture Behavioral of Control_tb is
    Component Control is
  Port (
  clk : IN STD_LOGIC;
  bid1 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  bid2 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  p1,p2,tx:out STD_LOGIC_VECTOR(10 DOWNTO 0);
  win1 : INOUT STD_LOGIC;
  win2 : INOUT STD_LOGIC;
```

```
    bid1_invalid : INOUT STD_LOGIC;
    bid2_invalid : INOUT STD_LOGIC;
    bid1_invalid_25 : INOUT STD_LOGIC;
    bid2_invalid_25 : INOUT STD_LOGIC;
    bid1_invalid_high : INOUT STD_LOGIC;
    bid2_invalid_high : INOUT STD_LOGIC;
    ssd : INOUT STD_LOGIC_VECTOR(15 DOWNTO 0):="0000101100011101";
    random1 : INOUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    init : IN STD_LOGIC:='0'
      );
  end Component;
  constant prd: time:=10 ns;
  signal
  clk,win1,win2,bid1_invalid,bid2_invalid,bid1_invalid_25,bid2_invalid_25,bid1_invalid_high,bid2_in
  valid_high,init:std_logic;
  signal bid1,bid2:std_logic_vector(7 downto 0);
  signal random1:std_logic_vector(3 downto 0);
  signal ssd:std_logic_vector(15 downto 0);
  signal p1,p2,tx:std_logic_vector(10 downto 0);
  signal err_con:integer:=0;
  begin
     clk_process :process
      begin
       clk <= '0';
       wait for prd/2;
       clk <= '1';
       wait for prd/2;
      end process;
      c1:control
          port map(
              clk => clk,
              bid1 => bid1,
              bid2 => bid2,
              p1=>p1,
              p2=>p2,
              tx=>tx,
              win1 => win1,
              win2 => win2,
              bid1_invalid => bid1_invalid,
              bid2_invalid => bid2_invalid,
              bid1_invalid_25 => bid1_invalid_25,
              bid2_invalid_25 => bid2_invalid_25,
              bid1_invalid_high => bid1_invalid_high,
              bid2_invalid_high => bid2_invalid_high,
              random1 => random1,
              ssd => ssd,
              init=> init
         );
          test_proc:process
            variable err_cnt:integer:=0;
            begin
            --basic test: p1 bids 1, p2 bids 127, random number geneator favors p2--
            bid1<="10000001";
            bid2<="01111111";
            random1<="0010";
            init<='1';
            for i in 0 to 14 loop
                  init<='1';
                  wait for 10*prd;
                  init<='0';
                  wait for 2*prd;
            end loop;
            assert (p1="00001110111") report "p1 incorrect";
            assert (p2="11010010000") report "p2 incorrect";
            assert (tx="00011011001") report "tax incorrect";
            assert (bid2_invalid_high='1') report "LED error";
            --this automatically tests for alternating between players
            --reduce p2's bid to 110
            bid2<="01101110";
            assert false report "testing for new bid";
            init<='1';
            wait for 4*prd;
            init<='0';
            wait for 2*prd;
            --p1 tries to bid more than he will have left if p2 wins
            bid1<="10101001";
            wait for 2*prd;
            assert (bid1_invalid_25='1') report "LED error";
            --now test for all 6 cases of p1,p2. One case is already done
            bid1<="00100111";
            wait for 2*prd;
            init<='1';
            wait for 2*prd;
            init<='0';
            wait for 6*prd;
            assert (p1="00000100111") report "p1 incorrect";
            assert (p2="11010111100") report "p2 incorrect";
            assert (tx="00011111101") report "tax incorrect";
            --both lose
            bid1<="00001010";
            bid2<="00001100";
            random1<="1000";
            init<='1';
            wait for 10*prd;
            init<='0';
            wait for 2*prd;
            init<='1';
            wait for 2*prd;
            init<='0';
            wait for 6*prd;
            assert (p1="00000100101") report "p1 incorrect";
            assert (p2="11010111001") report "p2 incorrect";
            assert (tx="00100000010") report "tax incorrect";
            --case of 7
            bid1<="00001010";
            bid2<="00001100";
            random1<="0111";
            init<='1';
            wait for 10*prd;
            init<='0';
            wait for 2*prd;
            init<='1';
            wait for 2*prd;
            init<='0';
            wait for 6*prd;
            assert (p1="00000010111") report "p1 incorrect";
            assert (p2="11011000010") report "p2 incorrect";
            assert (tx="00100000111") report "tax incorrect";
            --drive to end
            bid1<="00000001";
            bid2<="10010111";
            random1<="1100";
            init<='1';
            wait for 10*prd;
            init<='0';
            wait for 2*prd;
            init<='1';
            wait for 2*prd;
            init<='0';
            wait for 20*prd;--manually check ssd output in this part. We shal only assert the reset
  condition
            assert (p1="01111110000") report "p1 incorrect";
            assert (p2="01111110000") report "p2 incorrect";
            assert (tx="00000000000") report "tax incorrect";
            wait;
            end process;
  end Behavioral;
```

TESTING ON BOARD

The working of the ASM design and correct delays between different states was checked on board. The simulation results on vivado were validated as well. Some common strategies for testing on board included:

Loop Invariant

A preliminary test for the algorithm was to use loop invariant.

We know at the end of every bid in the sequence, the sum of *p1_money*, p2_mo*ney* and *tax* should be 2016. As we started with *p1_money* = *p2_money* = 1008, and are just transferring money from one person to another or adding to tax, thus no money is thrown out of the system. Hence, after every bid sum of the three displayed values was checked to be equal to 2016.

Also, at the end the money displayed of the winner, by same argument, should be displayed 2016. This preliminary test helped us to point out initial problems in the algorithm.

Alternating turns for player 1 and 2

Another preliminary test for our design was to check whether the *cycle* and *counter* signals were being updated properly after each bid iteration. This could be simply checked in the beginning if the turn to bid first alternates between the two players or not. If it doesn't, the signal assignments needed to be rectified.

Determination of correct state

The states 1 and 2 displayed "bid1" and "bid2" on the Seven Segment Display, as per the player who has to bid. Moreover, state 5 displays information of money of each player, tax, the player who won, random number after each bid in the sequence. The situations in which the code flow would stall helped us greatly to check state transitions and correct such errors.

Invalid LED's

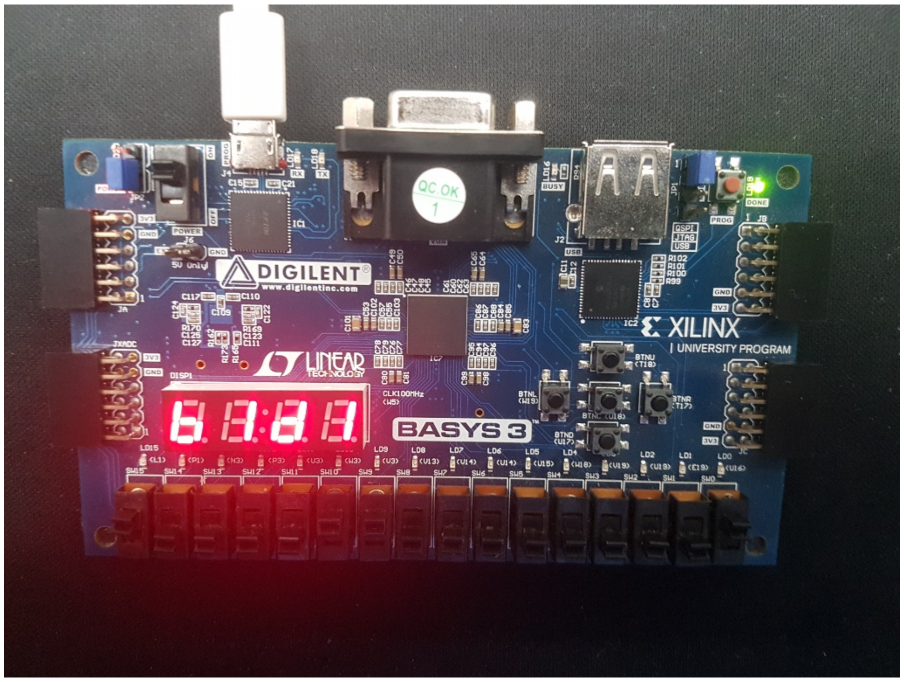
The *bid1_invalid* and *bid2_invalid* signals were '1' for the cases as explained in the Problem Description (Design Document 1). Any discrepancy in the signal assignment pointed out errors in the algorithm for different cases.

Calculation of correct values

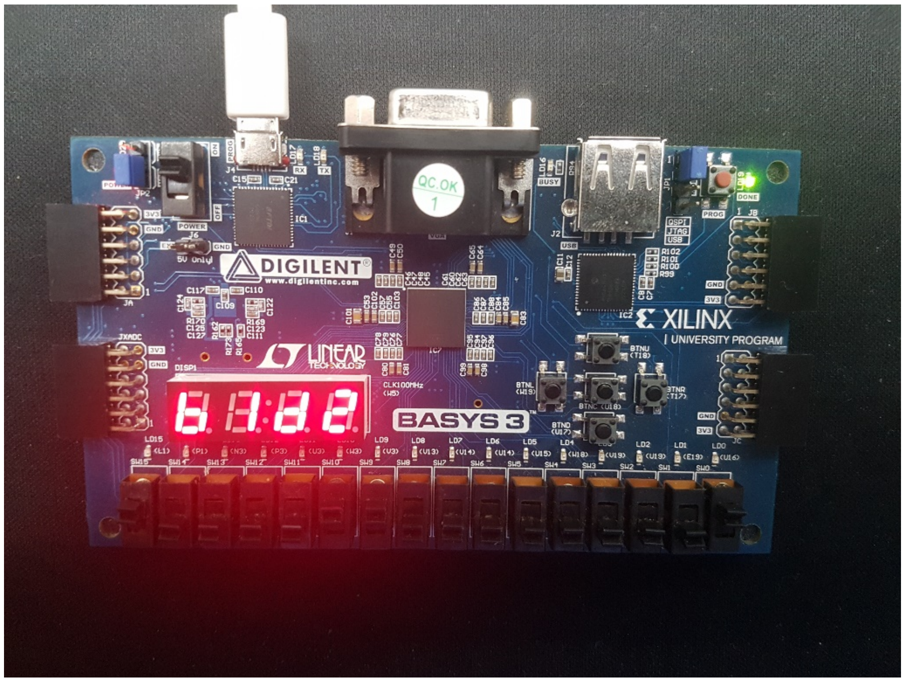
The values displayed for the six cases as in the 5th state of control module allowed to verify the correctness of the algorithm, proper distribution of money and correct decision of winner. We refined our algorithm greatly by testing these conditional assignments.

```
elsif state = "011" then
     p1p2 <= "00";
     counter<=0;
     if(unsigned(random)>7 and p1_bid(7)='1' and p2_bid(7)='0') then
         p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)+unsigned(p1_bid(6 DOWNTO 0)));
         p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)-unsigned(p1_bid(6 DOWNTO 0)));
         win1<='1'; win2<='0';
     elsif(unsigned(random)>7 and p2_bid(7)='1' and p1_bid(7)='0') then
         p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)+unsigned(p2_bid(6 DOWNTO 0)));
         p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)-unsigned(p2_bid(6 DOWNTO 0)));
         win1<='0'; win2<='1';
     elsif(unsigned(random)>7 and p2_bid(7)='1' and p1_bid(7)='1') then
         p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)
                                     +unsigned(p2_bid(6 DOWNTO 0))
                                     -unsigned(p1_bid(6 DOWNTO 0)));
         p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)
                                     +unsigned(p1_bid(6 DOWNTO 0))
                                     -unsigned(p2_bid(6 DOWNTO 0)));
         win1<='1'; win2<='1';
     elsif(unsigned(random)<7 and p1_bid(7)='0' and p2_bid(7)='1') then
         p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)+unsigned(p1_bid(6 DOWNTO 0)));
         p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)-unsigned(p1_bid(6 DOWNTO 0)));
         win1<='1'; win2<='0';
     elsif(unsigned(random)<7 and p2_bid(7)='0' and p1_bid(7)='1') then
         p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)+unsigned(p2_bid(6 DOWNTO 0)));
         p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)-unsigned(p2_bid(6 DOWNTO 0)));
         win1<='0'; win2<='1';
     elsif(unsigned(random)<7 and p2_bid(7)='0' and p1_bid(7)='0') then
         p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)
                                     +unsigned(p2_bid(6 DOWNTO 0))
                                     -unsigned(p1_bid(6 DOWNTO 0)));
         p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)
                                     +unsigned(p1_bid(6 DOWNTO 0))
                                     -unsigned(p2_bid(6 DOWNTO 0)));
         win1<='1'; win2<='1';
     elsif(unsigned(random)=7 and unsigned(p1_bid(6 DOWNTO 0))>unsigned(p2_bid(6 DOWNTO 0))) then
         p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)+unsigned(p1_bid(6 DOWNTO 0)));
         p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)-unsigned(p1_bid(6 DOWNTO 0)));
         win1<='1'; win2<='0';
     elsif(unsigned(random)=7 and unsigned(p2_bid(6 DOWNTO 0))>unsigned(p1_bid(6 DOWNTO 0))) then
         p2_money <= STD_LOGIC_VECTOR(unsigned(p2_money)+unsigned(p2_bid(6 DOWNTO 0)));
         p1_money <= STD_LOGIC_VECTOR(unsigned(p1_money)-unsigned(p2_bid(6 DOWNTO 0)));
         win1<='0'; win2<='1';
     end if;
```

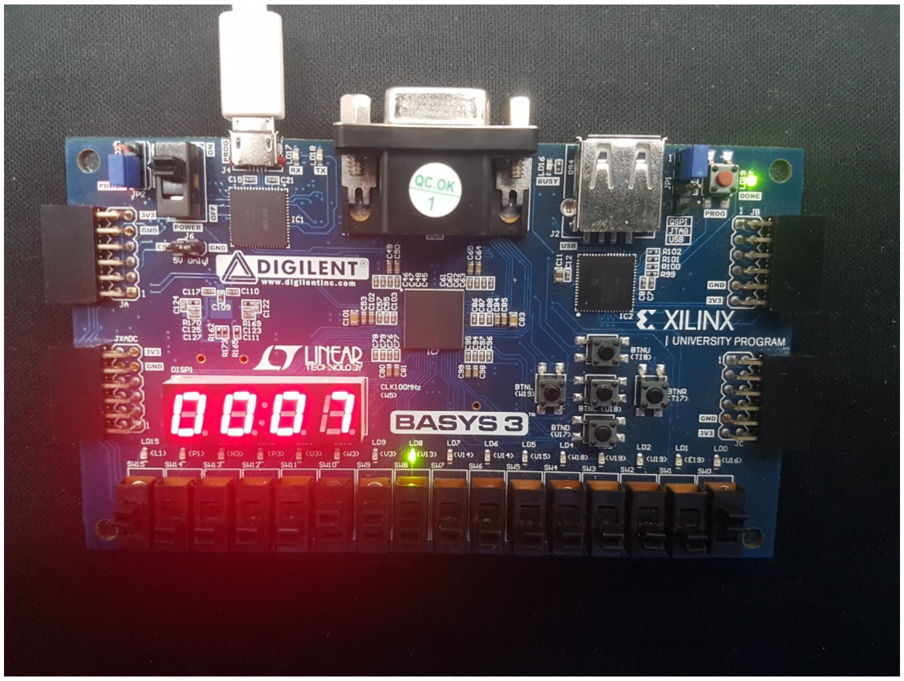
IMAGES OF CASES TESTED



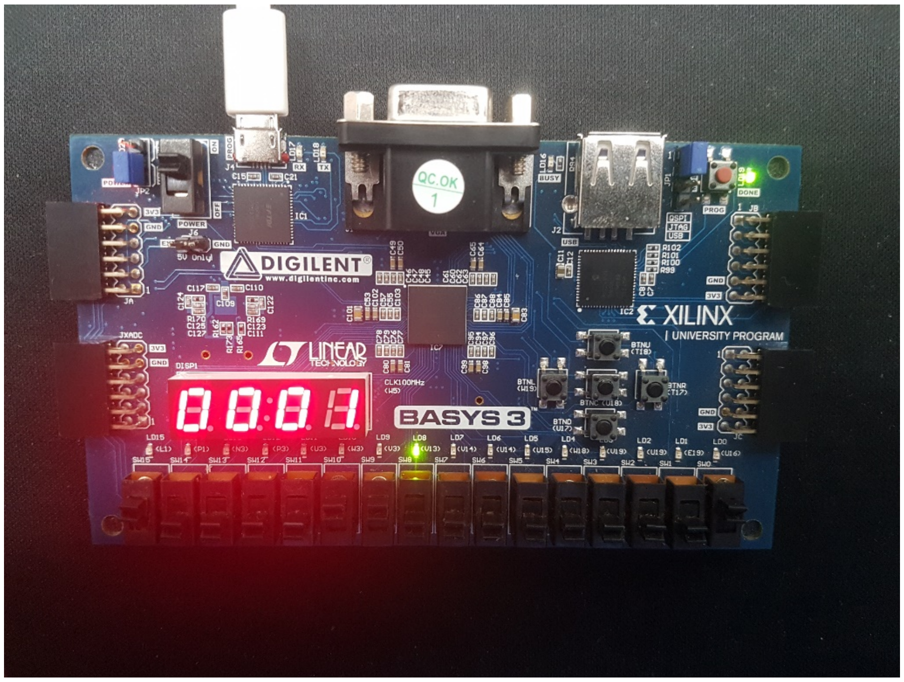
State 1 – Bid 1 = >7, 6



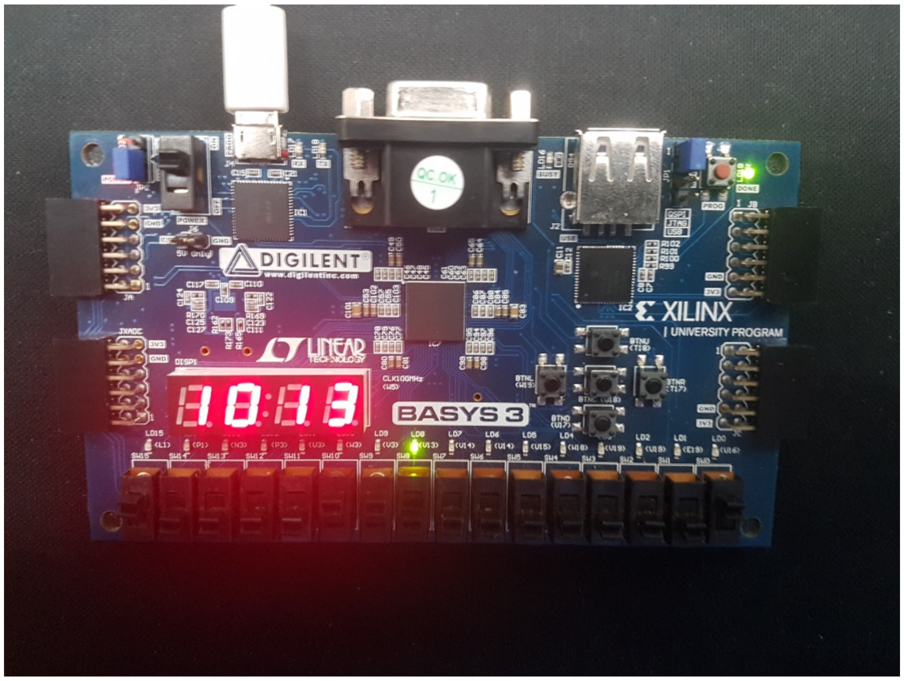
State 2 – Bid 2 = <7, 1



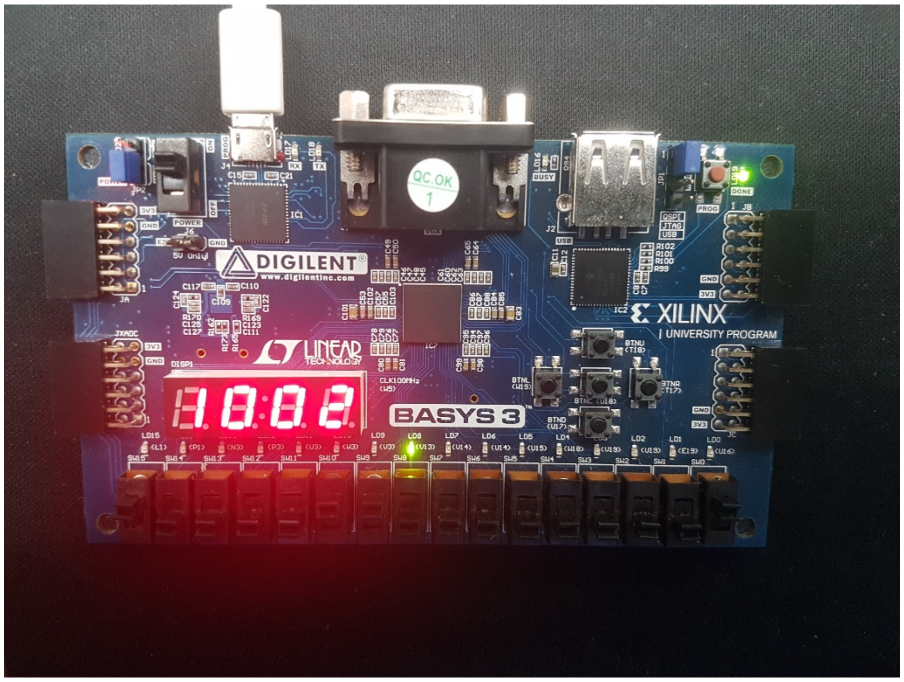
State 5 – Display of random number = 7



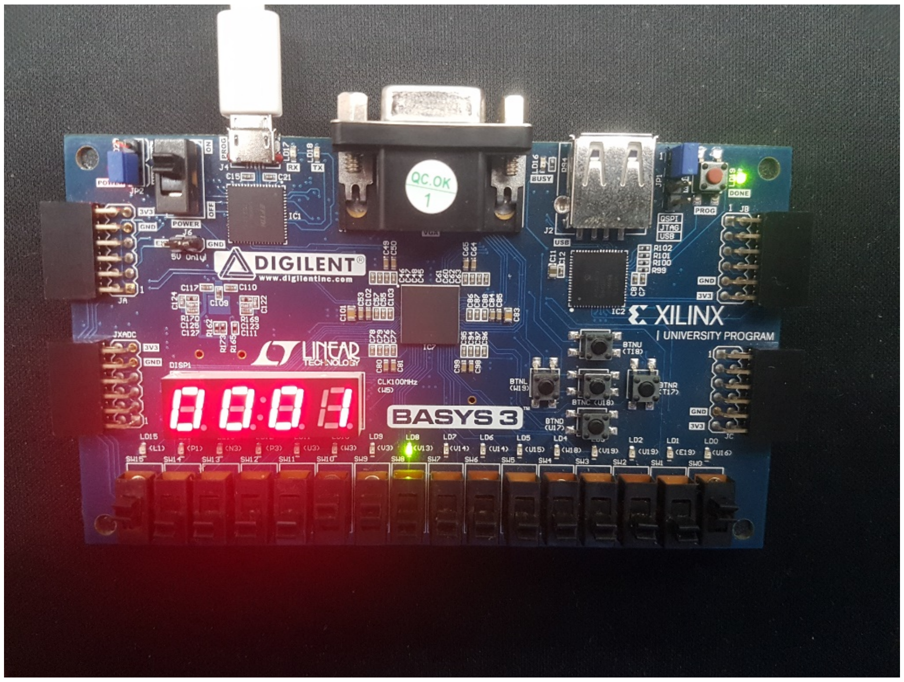
State 5 – Display of winner, here 1 (higher bid, random no. = 7)



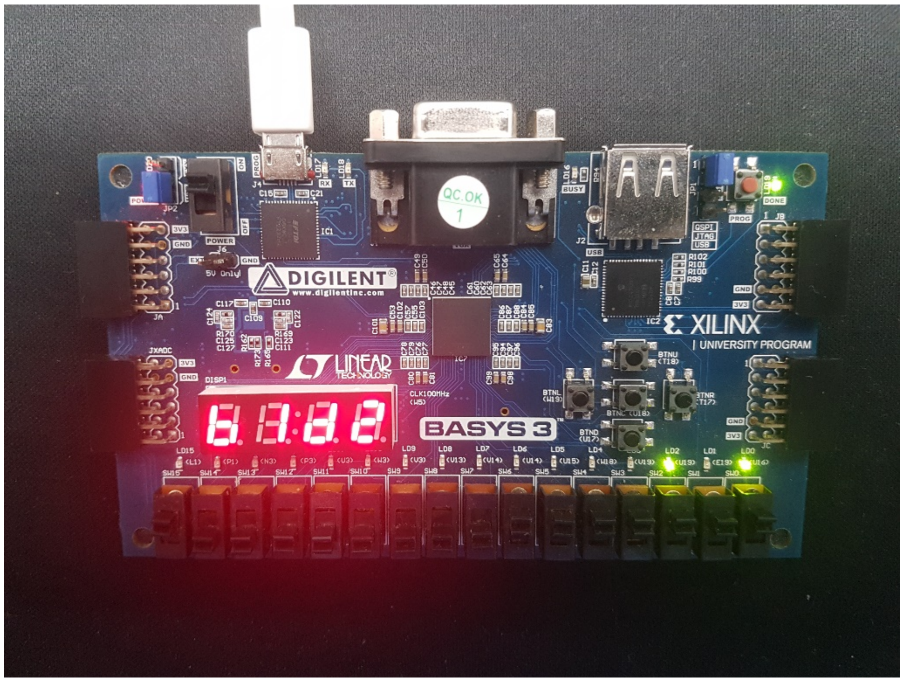
State 5 – Display of p1_money =
1008 – 1 (tax) + 6 (won bid) = 1013



State 5 – Display of p2_money =
1008 – 0 (tax) – 6 (lost bid) = 1002



State 5 – Display tax (here $\left\lfloor \frac{6}{4} \right\rfloor + \left\lfloor \frac{1}{4} \right\rfloor = 1$)



State 2 – Checking invalid bid

Experience and Learning

The main difference between this project and all other work we have done on FPGA’s before was the fact that the problem was incompletely specified, and we had the freedom to define the exact working of the circuit.

With this freedom came the responsibility to create something that could be justified mathematically and game theoretically. This required thinking and adjustment of rules (details provided in Design Document 1). The game, we now can prove, no longer has a trivial winning strategy, and thus, is worthwhile of playing.

At the core of the design lies the random number generator. We could always use the library function, or even create one using shift registers. Yet, what we are doing is not only extremely simple at both coding and Synthesis level, it also better than these other solutions. Basically, what the other designs try to do is to emulate a random sequence is a deterministic circuit, and thus can throw up only a pseudo-random sequence. The question we ask is: when there is so much randomness in the world, why try to create more?

In the design document, we had proposed an implementation of the I/O unit on a VGA screen. Unfortunately, we could not do that, but that is still a possibility for development…

Overall, it was a great experience in terms of both learning and fun for both of us :)