**ELEC 204 Digital Design Project Report**

Name: Hasan Güney 60333

Ekrem Yiğiter 59721

Date: 11/30/2019

FIND THE BOX GAME

In this game, we try to find the square which is selected randomly. Firstly, our code selects a random number which is between 0 and 15 when we push the button. Then this number provide us a square because in our code, we matched the random numbers with a square. Basically, we have 16 squares for 16 numbers. After that, we are expected to find the chosen square by using switches. We have only 10 switches to match them with squares. We solved this problem by using one switch to select up squares. When our switch is 0, we can select 8 down squares. When our switch is 1, we can select 8 up squares with the 8 switches. User is given 3 tries to find the square. Also, user can see remaining tries with the LEDs above the switches. In first step, first three LED is lighting. If we tried but we did not find the square, 1 LED will be closed. If we don’t find it in 3 tries, all LEDs will be closed and we see ‘YOU LOSE’ on the seven segment display. If we find it, we will see ‘YOU WIN’.

**Methods**

In the main module we have 5 inputs and 3 outputs. Inputs are Switch, UpDown, Show, Clk and chooser; outputs are life, SevenSegControl and SevenSegBus. Switch input has 8 bits. With this input user can choose the squares. Other input is UpDown and it has 2 bits. This input basically selects the upper squares. Other input is Show and it has 2 bits. This input shows the correct square if we did not find the selected square in 3 tries. Other input is chooser and it has 2 bits. This input chooses a random square when user push the button. Last input is clock and it has 2 bits. We use this input in different modules in the code and this is the master clock of the board. One of the outputs is life. It has 3 bits and it shows how many bits are remaining. Other output is SevenSegControl and it has 8 bits. This basically control what we want to write seven segment display. Last input is SevenSegBus and it has 8 bits. This output decides where to display the outputs we chose in SevenSegControl.

Our vhdl code must match a randomly chosen number and a square that we want to show on the seven segment display. The code does this choosing by pushing a button. We did this by using master clock of the FPGA. The codes must choose a number when the clock counts all 16 numbers fast. This counting will be so fast so we cannot know which number will be selected. Therefore, this choosing will be like random choosing. After choosing number and matching them with a square, our code asks to choose a square from the user. After choosing the square, the code must check whether the square is correct or not. If the square is correct the code says to user ‘YOU WIN’. If the square is not correct, the code close FPGA’s one LED and this means we have lost one try and two tries are remained. If user do not find the square in three tries, the code says to user ‘YOU LOSE’. After using all three tries and losing the game, we will be able to see where the correct square was.

**Problems encountered, errors and warnings resolved**

In our code, we choose a square in bottom of the seven segment display and if we choose right above of that square we encounter a problem because we have only 1 result to display that bit of the seven segment display but we divide the bit in two different square. In order to solve this problem, we created a signal named as checker and we put if statements in our code. These checker signals check the squares with the if statements. For instance, if we chose a square and if we chose another square which is below or above the first square, we basically close the first square and light the both squares by showing ‘8’ on seven segment display. Another important problem we encountered was about clock. When we look at the description of the error we saw that we could solve this problem by writing ‘NET "chooser" CLOCK\_DEDICATED\_ROUTE = FALSE;‘ in the ucf file. The main reason of this issue is that our clock is not at the optimal place in terms of speed. However, the description says that if this speed is acceptable for our code, this error can be demoted to a warning. With this way, we solved this problem. Another problem was that user was given 3 tries so we wanted to use while loop. However, we encounter some problems because we could not quit out of the loop and our code was not completed. In order to solve this problem, we removed while loop from our code and we add if statements. With the process function we were able to repeat the code so using if statement was a true choice for our code. In our code, we tried to use arrays, because we wanted to match squares with numbers so we tried to do that with arrays but we did not solve the problems with arrays. For this reason, we removed arrays and we add signal as std\_logic and we solved this problem. One of the problems we encountered was that we received too different and undesired outputs on the seven segment display. Then, we noticed that our FPGA board was too slow and it could not reach the speed of clock. For this reason, we add a module in our code and we aimed to make the clock slower. After that we solved most of our warnings.

**Conclusion**

All in all, we try to design a game. In this game we try to find a square which is selected randomly. This project was so instructive because we use many functions of vhdl language. We have seen some of these functions in tutorial lab videos. For this reason, using those functions was not difficult for us. However, for our game we had to use different functions and these were very difficult. Also, we saw many different errors and warnings that we have never seen before but we try to eliminate all of them and we made a project with no errors and some small warnings.

**References**

[Online]. Available: 1. https://github.com/arash-codes/Elec204-labs.M. M. Mano and M. D.

Ciletti, Digital design with an introduction to the Verilog HDL, VHDL, and SystemVerilog. New York, NY: Pearson, 2019.

**Appendix 1. Lab source code**

**MAIN MODULE**

**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**entity main is**

**Port ( Switch : in STD\_LOGIC\_VECTOR (7 downto 0);**

**UpDown : in STD\_LOGIC;**

**Show : in STD\_LOGIC;**

**clk : in STD\_LOGIC;**

**chooser : in std\_logic;**

**life : out std\_logic\_vector(2 downto 0):="111";**

**SevenSegControl : out std\_logic\_vector(7 downto 0):=x"11";**

**SevenSegBus : out std\_logic\_vector(7 downto 0)**

**);**

**end main;**

**architecture Behavioral of main is**

**COMPONENT sevenSegment**

**PORT(**

**A : IN std\_logic\_vector(3 downto 0);**

**B : IN std\_logic\_vector(3 downto 0);**

**C : IN std\_logic\_vector(3 downto 0);**

**D : IN std\_logic\_vector(3 downto 0);**

**E : IN std\_logic\_vector(3 downto 0);**

**F : IN std\_logic\_vector(3 downto 0);**

**G : IN std\_logic\_vector(3 downto 0);**

**H : IN std\_logic\_vector(3 downto 0);**

**clk : IN std\_logic;**

**SevenSegControl : OUT std\_logic\_vector(7 downto 0);**

**SevenSegBus : OUT std\_logic\_vector(7 downto 0)**

**);**

**END COMPONENT;**

**COMPONENT showWhere**

**Port (**

**Show : in STD\_LOGIC;**

**genVal : in integer;**

**resultA : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultB : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultC : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultD : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultE : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultF : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultG : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultH : out STD\_LOGIC\_VECTOR (3 downto 0);**

**clk : in STD\_LOGIC**

**);**

**END COMPONENT;**

**COMPONENT lifeCount**

**Port ( counter : in integer;**

**life : out STD\_LOGIC\_VECTOR (2 downto 0);**

**clk : in STD\_LOGIC**

**);**

**END COMPONENT;**

**COMPONENT random**

**Port ( clk : in STD\_LOGIC;**

**chooser : in std\_logic;**

**genVal : out integer);**

**END COMPONENT;**

**COMPONENT clk\_divider**

**PORT(**

**clk : in STD\_LOGIC;**

**DCT : out STD\_LOGIC);**

**END COMPONENT;**

**signal resultA, resultB, resultC, resultD, resultE, resultF, resultG, resultH : std\_logic\_vector (3 downto 0) := x"d";**

**signal checker0, checker1, checker2, checker3, checker4, checker5, checker6, checker7,**

**checker8, checker9, checker10, checker11, checker12, checker13, checker14, checker15: std\_logic := '0';**

**signal A, B, C, D, E, F, G, H, life1: STD\_LOGIC\_VECTOR (3 downto 0);**

**signal genVal1 : integer;**

**signal count2 : integer := 0;**

**signal showed : std\_logic ;**

**signal DCT : std\_logic;**

**begin**

**Inst\_random: random PORT MAP(**

**clk => clk,**

**chooser=>chooser,**

**genVal => genVal1**

**);**

**Inst\_clk\_divider: clk\_divider PORT MAP(**

**clk => clk,**

**DCT => DCT**

**);**

**Inst\_showWhere: showWhere PORT MAP(**

**clk => clk,**

**Show => Show,**

**genVal => genVal1,**

**resultA => A,**

**resultB => B,**

**resultC => C,**

**resultD => D,**

**resultE => E,**

**resultF => F,**

**resultG => G,**

**resultH => H**

**);**

**Inst\_lifeCount: lifeCount PORT MAP(**

**clk => clk,**

**counter => count2,**

**life(0) => life1(0),**

**life(1) => life1(1),**

**life(2) => life1(2)**

**);**

**process(clk) is**

**begin**

**if rising\_edge(DCT) then**

**if (count2 <= 4) then**

**showed <= '0';**

**if ((Show = '1')) then**

**resultA <= A;**

**resultB <= B;**

**resultC <= C;**

**resultD <= D;**

**resultE <= E;**

**resultF <= F;**

**resultG <= G;**

**resultH <= H;**

**showed <= '1';**

**elsif ((UpDown = '0') and (Switch(0) = '1') and (checker0 = '0') and (genVal1 /= 0)) then**

**checker0 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker8 = '1') then**

**resultA <= x"c";**

**elsif (checker8 = '0') then**

**resultA <= x"b";**

**end if;**

**elsif ((UpDown = '0') and (Switch(1) = '1') and (checker1 = '0') and (genVal1 /= 1)) then**

**checker1 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker9 = '1') then**

**resultB <= x"c";**

**elsif (checker9 = '0') then**

**resultB <= x"b";**

**end if;**

**elsif ((UpDown = '0') and (Switch(2) = '1') and (checker2 = '0') and (genVal1 /= 2)) then**

**checker2 <= '1';**

**count2 <= count2 + 1; --increase count**

**if (checker10 = '1') then**

**resultC <= x"c";**

**elsif (checker10 = '0') then**

**resultC <= x"b";**

**end if;**

**elsif ((UpDown = '0') and (Switch(3) = '1') and (checker3 = '0') and (genVal1 /= 3)) then**

**checker3 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker11 = '1') then**

**resultD <= x"c";**

**elsif (checker11 = '0') then**

**resultD <= x"b";**

**end if;**

**elsif ((UpDown = '0') and (Switch(4) = '1') and (checker4 = '0') and (genVal1 /= 4)) then**

**checker4 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker12 = '1') then**

**resultE <= x"c";**

**elsif (checker12 = '0') then**

**resultE <= x"b";**

**end if;**

**elsif ((UpDown = '0') and (Switch(5) = '1') and (checker5 = '0') and (genVal1 /= 5)) then**

**checker5 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker13 = '1') then**

**resultF <= x"c";**

**elsif (checker13 = '0') then**

**resultF <= x"b";**

**end if;**

**elsif ((UpDown = '0') and (Switch(6) = '1') and (checker6 = '0') and (genVal1 /= 6)) then**

**checker6 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker14 = '1') then**

**resultG <= x"c";**

**elsif (checker14 = '0') then**

**resultG <= x"b";**

**end if;**

**elsif ((UpDown = '0') and (Switch(7) = '1') and (checker7 = '0') and (genVal1 /= 7)) then**

**checker7 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker15 = '1') then**

**resultH <= x"c";**

**elsif (checker15 = '0') then**

**resultH <= x"b";**

**end if;**

**elsif ((UpDown = '1') and (Switch(0) = '1') and (checker8 = '0') and (genVal1 /= 8)) then**

**checker8 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker0 = '1') then**

**resultA <= x"c";**

**elsif (checker0 = '0') then**

**resultA <= x"a";**

**end if;**

**elsif ((UpDown = '1') and (Switch(1) = '1') and (checker9 = '0') and (genVal1 /= 9)) then**

**checker9 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker1 = '1') then**

**resultB <= x"c";**

**elsif (checker1 = '0') then**

**resultB <= x"a";**

**end if;**

**elsif ((UpDown = '1') and (Switch(2) = '1') and (checker10 = '0') and (genVal1 /= 10)) then**

**checker10 <= '1';**

**count2 <= count2 + 1; --increase count**

**if (checker2 = '1') then**

**resultC <= x"c";**

**elsif (checker2 = '0') then**

**resultC <= x"a";**

**end if;**

**elsif ((UpDown = '1') and (Switch(3) = '1') and (checker11 = '0') and (genVal1 /= 11)) then**

**checker11 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker3 = '1') then**

**resultD <= x"c";**

**elsif (checker3 = '0') then**

**resultD <= x"a";**

**end if;**

**elsif ((UpDown = '1') and (Switch(4) = '1') and (checker12 = '0') and (genVal1 /= 12)) then**

**checker12 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker4 = '1') then**

**resultE <= x"c";**

**elsif (checker4 = '0') then**

**resultE <= x"a";**

**end if;**

**elsif ((UpDown = '1') and (Switch(5) = '1') and (checker13 = '0') and (genVal1 /= 13)) then**

**checker13 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker5 = '1') then**

**resultF <= x"c";**

**elsif (checker5 = '0') then**

**resultF <= x"a";**

**end if;**

**elsif ((UpDown = '1') and (Switch(6) = '1') and (checker14 = '0') and (genVal1 /= 14)) then**

**checker14 <= '1';**

**count2 <= count2 + 1;--increase count**

**if (checker6 = '1') then**

**resultG <= x"c";**

**elsif (checker6 = '0') then**

**resultG <= x"a";**

**end if;**

**elsif ((UpDown = '1') and (Switch(7) = '1') and (checker15 = '0') and (genVal1 /= 15)) then**

**checker15 <= '1';**

**count2 <= count2 + 1; --increase count**

**if (checker7 = '1') then**

**resultH <= x"c";**

**elsif (checker7 = '0') then**

**resultH <= x"a";**

**end if;**

**elsif ((UpDown = '0' and Switch(0) = '1'and checker0 = '0' and genVal1 = 0) or (UpDown = '0' and Switch(1) = '1'and checker1 = '0' and genVal1 = 1) or**

**(UpDown = '0' and Switch(2) = '1'and checker2 = '0' and genVal1 = 2) or (UpDown = '0' and Switch(3) = '1'and checker3 = '0' and genVal1 = 3) or**

**(UpDown = '0' and Switch(4) = '1'and checker4 = '0' and genVal1 = 4) or (UpDown = '0' and Switch(5) = '1'and checker5 = '0' and genVal1 = 5) or**

**(UpDown = '0' and Switch(6) = '1'and checker6 = '0' and genVal1 = 6) or (UpDown = '0' and Switch(7) = '1'and checker7 = '0' and genVal1 = 7) or**

**(UpDown = '1' and Switch(0) = '1'and checker8 = '0' and genVal1 = 8) or (UpDown = '1' and Switch(1) = '1'and checker9 = '0' and genVal1 = 9) or**

**(UpDown = '1' and Switch(2) = '1'and checker10 = '0' and genVal1 = 10) or (UpDown = '1' and Switch(3) = '1'and checker11 = '0' and genVal1 = 11) or**

**(UpDown = '1' and Switch(4) = '1'and checker12 = '0' and genVal1 = 12) or (UpDown = '1' and Switch(5) = '1'and checker13 = '0' and genVal1 = 13) or**

**(UpDown = '1' and Switch(6) = '1'and checker14 = '0' and genVal1 = 14) or (UpDown = '1' and Switch(7) = '1'and checker15 = '0' and genVal1 = 15)) then**

**resultA <= x"7";**

**resultB <= x"6";**

**resultC <= x"5";**

**resultD <= x"4";**

**resultE <= x"d";**

**resultF <= x"3";**

**resultG <= x"2";**

**resultH <= x"1";**

**end if;**

**if ((showed = '1') and (Show = '0'))then**

**count2 <= 3;**

**life(2)<= '0';**

**life(1)<= '0';**

**life(0)<= '0';**

**resultA <= x"0";**

**resultB <= x"9";**

**resultC <= x"2";**

**resultD <= x"8";**

**resultE <= x"d";**

**resultF <= x"3";**

**resultG <= x"2";**

**resultH <= x"1";**

**else**

**life(2)<= life1(2);**

**life(1)<= life1(1);**

**life(0)<= life1(0);**

**if (count2 = 3) then**

**resultA <= x"0";**

**resultB <= x"9";**

**resultC <= x"2";**

**resultD <= x"8";**

**resultE <= x"d";**

**resultF <= x"3";**

**resultG <= x"2";**

**resultH <= x"1";**

**end if;**

**end if;**

**end if;**

**end if;**

**end process;**

**Origi\_sevenSegment: sevenSegment PORT MAP(**

**A => resultA,**

**B => resultB,**

**C => resultC,**

**D => resultD,**

**E => resultE,**

**F => resultF,**

**G => resultG,**

**H => resultH,**

**clk => clk,**

**SevenSegControl => SevenSegControl,**

**SevenSegBus => SevenSegBus**

**);**

**end Behavioral;**

**RANDOM MODULE**

**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**entity random is**

**Port ( clk : in STD\_LOGIC;**

**chooser : in std\_logic;**

**genVal : out integer);**

**end random;**

**architecture Behavioral of random is**

**signal count : integer := 0;**

**begin**

**process(clk) is**

**begin**

**if rising\_edge(clk) then**

**count <= count + 1;**

**if ( count = 15) then**

**count <= 0;**

**end if;**

**end if;**

**if rising\_edge(chooser) then**

**genVal <= count;**

**end if;**

**end process;**

**end Behavioral;**

**CLOCK DIVIDER MODULE**

**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**entity clk\_divider is**

**Port ( clk : in STD\_LOGIC;**

**DCT : out STD\_LOGIC);**

**end clk\_divider;**

**architecture Behavioral of clk\_divider is**

**signal counterT : integer := 0;**

**signal dividedClkTimer : std\_logic := '0';**

**begin**

**timer: process(clk)**

**begin**

**if rising\_edge(clk) then**

**counterT <= counterT + 1;**

**dividedClkTimer <= '0';**

**if counterT = 10000000 then**

**dividedClkTimer <= '1';**

**counterT <= 0;**

**end if;**

**end if;**

**end process;**

**DCT <= dividedClkTimer;**

**end Behavioral;**

**SHOWWHERE MODULE**

**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**entity showWhere is**

**Port ( Show : in STD\_LOGIC;**

**genVal : in integer;**

**resultA : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultB : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultC : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultD : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultE : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultF : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultG : out STD\_LOGIC\_VECTOR (3 downto 0);**

**resultH : out STD\_LOGIC\_VECTOR (3 downto 0);**

**clk : in STD\_LOGIC**

**);**

**end showWhere;**

**architecture Behavioral of showWhere is**

**COMPONENT clk\_divider**

**PORT(**

**clk : in STD\_LOGIC;**

**DCT : out STD\_LOGIC);**

**END COMPONENT;**

**signal DCT : std\_logic;**

**begin**

**Inst\_clk\_divider: clk\_divider PORT MAP(**

**clk => clk,**

**DCT => DCT**

**);**

**process(clk) is**

**begin**

**if rising\_edge(DCT) then**

**if (Show = '1') then**

**if ((genVal = 0) or (genVal = 8)) then**

**resultB <= x"d";**

**resultC <= x"d";**

**resultD <= x"d";**

**resultE <= x"d";**

**resultF <= x"d";**

**resultG <= x"d";**

**resultH <= x"d";**

**if (genVal = 0) then**

**resultA <= x"b";**

**else**

**resultA <= x"a";**

**end if;**

**elsif ((genVal = 1) or (genVal = 9)) then**

**resultA <= x"d";**

**resultC <= x"d";**

**resultD <= x"d";**

**resultE <= x"d";**

**resultF <= x"d";**

**resultG <= x"d";**

**resultH <= x"d";**

**if (genVal = 0) then**

**resultB <= x"b";**

**else**

**resultB <= x"a";**

**end if;**

**elsif ((genVal = 2) or (genVal = 10)) then**

**resultA <= x"d";**

**resultB <= x"d";**

**resultD <= x"d";**

**resultE <= x"d";**

**resultF <= x"d";**

**resultG <= x"d";**

**resultH <= x"d";**

**if (genVal = 2) then**

**resultC <= x"b";**

**else**

**resultC <= x"a";**

**end if;**

**elsif ((genVal = 3) or (genVal = 11)) then**

**resultA <= x"d";**

**resultB <= x"d";**

**resultC <= x"d";**

**resultE <= x"d";**

**resultF <= x"d";**

**resultG <= x"d";**

**resultH <= x"d";**

**if (genVal = 3) then**

**resultD <= x"b";**

**else**

**resultD <= x"a";**

**end if;**

**elsif ((genVal = 4) or (genVal = 12)) then**

**resultA <= x"d";**

**resultB <= x"d";**

**resultC <= x"d";**

**resultD <= x"d";**

**resultF <= x"d";**

**resultG <= x"d";**

**resultH <= x"d";**

**if (genVal = 4) then**

**resultE <= x"b";**

**else**

**resultE <= x"a";**

**end if;**

**elsif ((genVal = 5) or (genVal = 13)) then**

**resultA <= x"d";**

**resultB <= x"d";**

**resultc <= x"d";**

**resultD <= x"d";**

**resultE <= x"d";**

**resultG <= x"d";**

**resultH <= x"d";**

**if (genVal = 5) then**

**resultF <= x"b";**

**else**

**resultF <= x"a";**

**end if;**

**elsif ((genVal = 6) or (genVal = 14)) then**

**resultA <= x"d";**

**resultB <= x"d";**

**resultC <= x"d";**

**resultD <= x"d";**

**resultE <= x"d";**

**resultF <= x"d";**

**resultH <= x"d";**

**if (genVal = 6) then**

**resultG <= x"b";**

**else**

**resultG <= x"a";**

**end if;**

**elsif ((genVal = 7) or (genVal = 15)) then**

**resultA <= x"d";**

**resultB <= x"d";**

**resultC <= x"d";**

**resultD <= x"d";**

**resultE <= x"d";**

**resultF <= x"d";**

**resultG <= x"d";**

**if (genVal = 7) then**

**resultH <= x"b";**

**else**

**resultH <= x"a";**

**end if;**

**end if;**

**end if;**

**end if;**

**end process;**

**end Behavioral;**

**LIFECOUNT MODULE**

**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**entity lifeCount is**

**Port ( counter : in integer;**

**life : out STD\_LOGIC\_VECTOR (2 downto 0);**

**clk : in STD\_LOGIC**

**);**

**end lifeCount;**

**architecture Behavioral of lifeCount is**

**COMPONENT clk\_divider**

**PORT(**

**clk : in STD\_LOGIC;**

**DCT : out STD\_LOGIC);**

**END COMPONENT;**

**signal DCT : std\_logic;**

**begin**

**Inst\_clk\_divider: clk\_divider PORT MAP(**

**clk => clk,**

**DCT => DCT**

**);**

**process(clk) is**

**begin**

**if rising\_edge(DCT) then**

**if (counter = 0) then**

**life(2)<= '1';**

**life(1)<= '1';**

**life(0)<= '1';**

**elsif (counter = 1) then**

**life(2)<= '0';**

**life(1)<= '1';**

**life(0)<= '1';**

**elsif (counter = 2) then**

**life(2)<= '0';**

**life(1)<= '0';**

**life(0)<= '1';**

**elsif (counter = 3) then**

**life(2)<= '0';**

**life(1)<= '0';**

**life(0)<= '0';**

**end if;**

**end if;**

**end process;**

**end Behavioral;**

**SEVENSEGMENT MODULE**

**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**use ieee.std\_logic\_unsigned.all;**

**entity sevenSegment is**

**port( A: in std\_logic\_vector(3 downto 0);**

**B: in std\_logic\_vector(3 downto 0);**

**C: in std\_logic\_vector(3 downto 0);**

**D: in std\_logic\_vector(3 downto 0);**

**E: in std\_logic\_vector(3 downto 0);**

**F: in std\_logic\_vector(3 downto 0);**

**G: in std\_logic\_vector(3 downto 0);**

**H: in std\_logic\_vector(3 downto 0);**

**clk:in std\_logic;**

**SevenSegControl: out std\_logic\_vector (7 downto 0):=x"ff";**

**SevenSegBus: out std\_logic\_vector (7 downto 0));**

**end sevenSegment;**

**architecture Behavioral of sevenSegment is**

**signal clkTrigger: std\_logic;**

**signal sevenSegValue: std\_logic\_vector(3 downto 0);**

**begin**

**SlowClock: entity work.slowerClock port map(**

**clk=>clk,**

**slowClock => clkTrigger);**

**Driver: entity work.driver port map (**

**clk=>clkTrigger,**

**A=>A,**

**B=>B,**

**C=>C,**

**D=>D,**

**E=>E,**

**F=>F,**

**G=>G,**

**H=>H,**

**sevenSegValue => sevenSegValue,**

**sevenSegNumber => sevenSegControl);**

**Decoder: entity work.decoder port map (**

**inValue =>sevenSegValue,**

**outValue => sevenSegBus);**

**end Behavioral;**

**UCF FILE**

**NET "SevenSegControl[0]" LOC = P50;**

**NET "SevenSegControl[1]" LOC = P49;**

**NET "SevenSegControl[2]" LOC = P52;**

**NET "SevenSegControl[3]" LOC = P56;**

**NET "SevenSegControl[4]" LOC = P59;**

**NET "SevenSegControl[5]" LOC = P57;**

**NET "SevenSegControl[6]" LOC = P60;**

**NET "SevenSegControl[7]" LOC = P61;**

**NET "SevenSegBus[7]" LOC = P71;**

**NET "SevenSegBus[6]" LOC = P62;**

**NET "SevenSegBus[5]" LOC = P65;**

**NET "SevenSegBus[4]" LOC = P72;**

**NET "SevenSegBus[3]" LOC = P73;**

**NET "SevenSegBus[2]" LOC = P98;**

**NET "SevenSegBus[1]" LOC = P64;**

**NET "SevenSegBus[0]" LOC = P70;**

**NET "clk" LOC = P40;**

**net "UpDown" loc = P78;**

**net "chooser" loc = P34;**

**net "Switch[0]" loc = P15;**

**net "Switch[1]" loc = P12;**

**net "Switch[2]" loc = P5;**

**net "Switch[3]" loc = P4;**

**net "Switch[4]" loc = P94;**

**net "Switch[5]" loc = P90;**

**net "Switch[6]" loc = P88;**

**net "Switch[7]" loc = P85;**

**net "Show" loc = P35;**

**net "life[0]" loc = P16;**

**net "life[1]" loc = P13;**

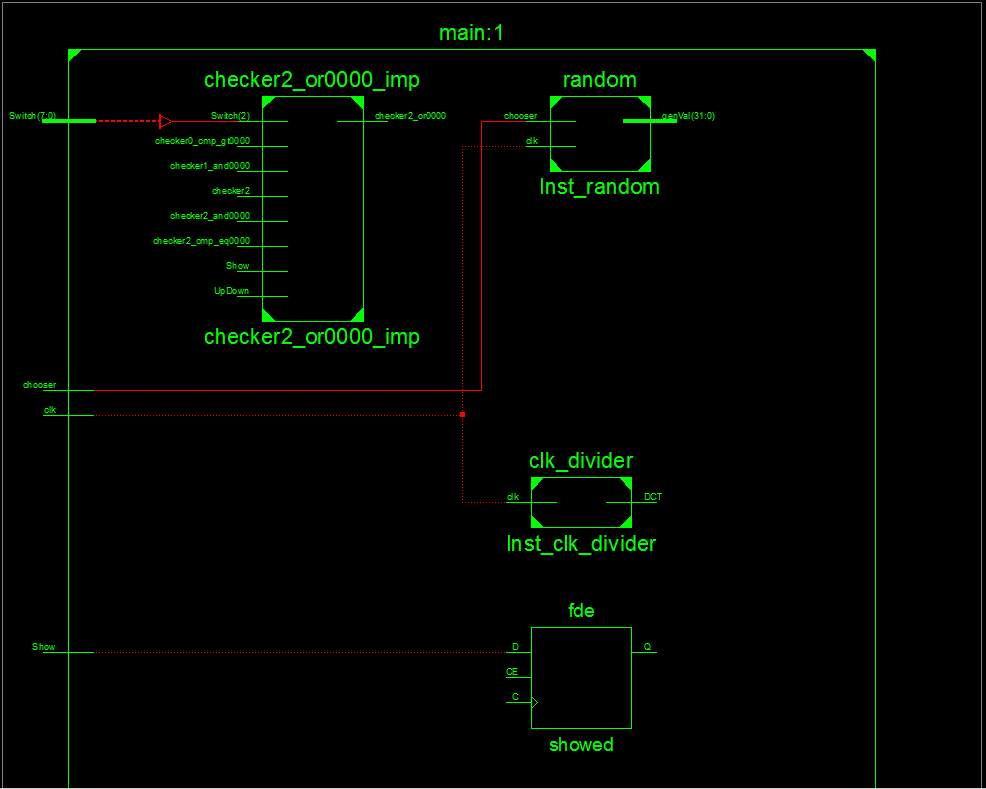
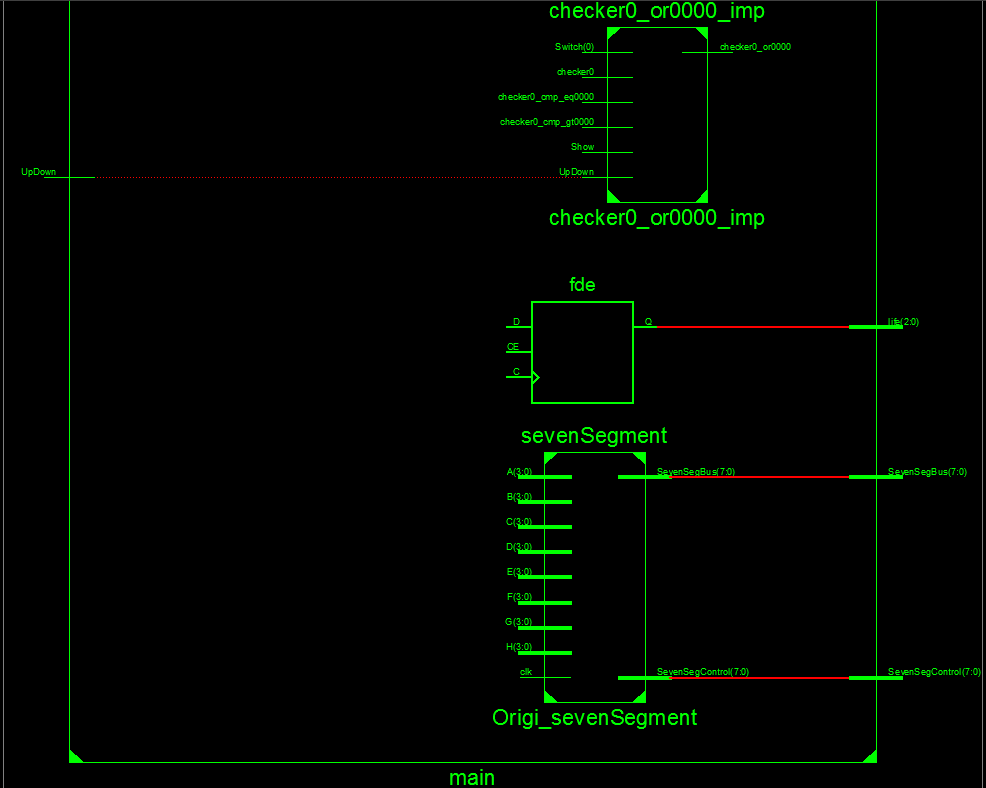
**net "life[2]" loc = P6;**

**NET "Show" CLOCK\_DEDICATED\_ROUTE = FALSE;**

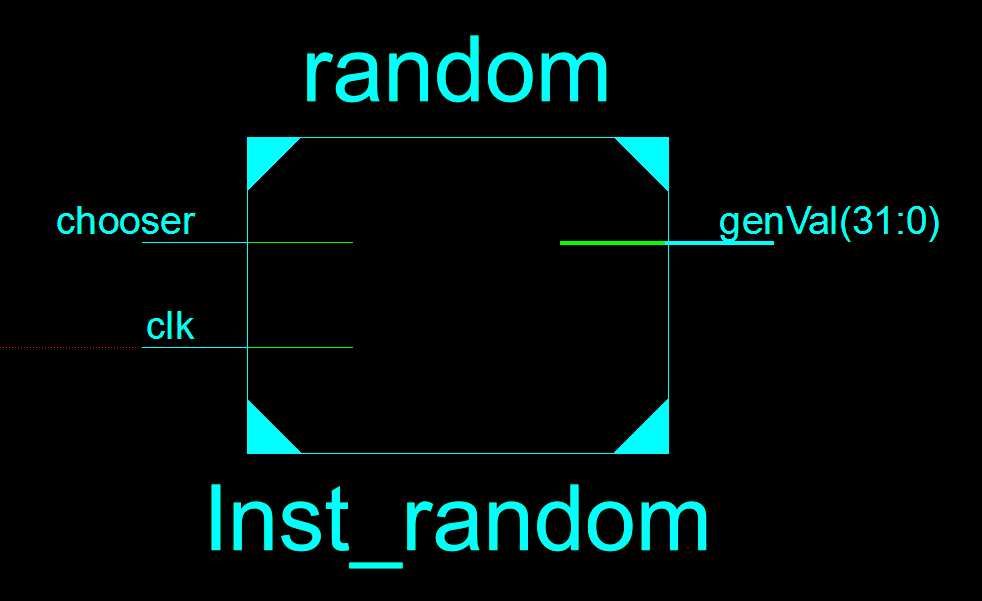
**NET "chooser" CLOCK\_DEDICATED\_ROUTE = FALSE;**

**Appendix 2. RTL schematics**

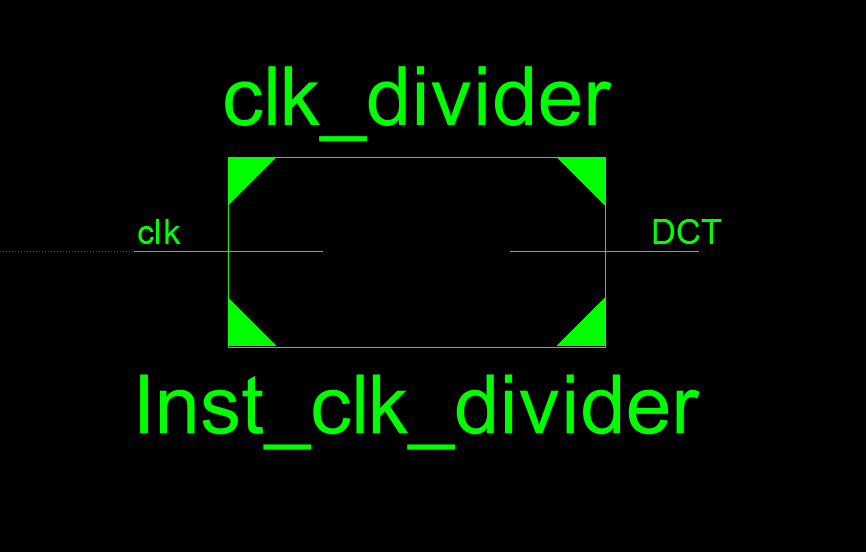
**Main Module**



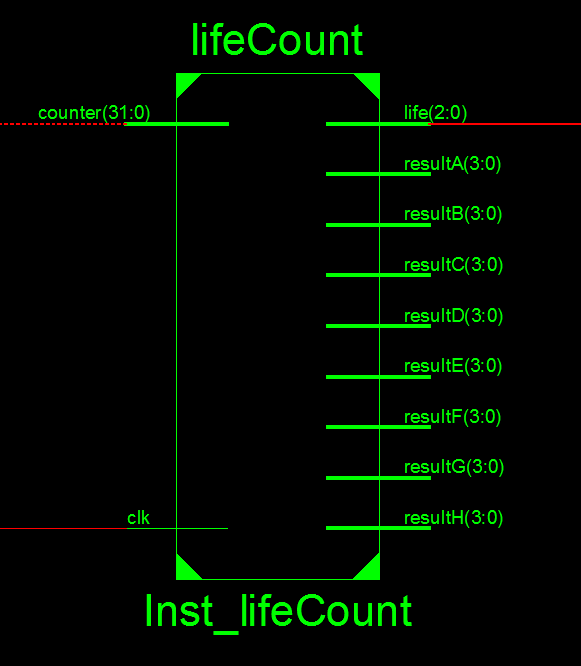
**Random Module**



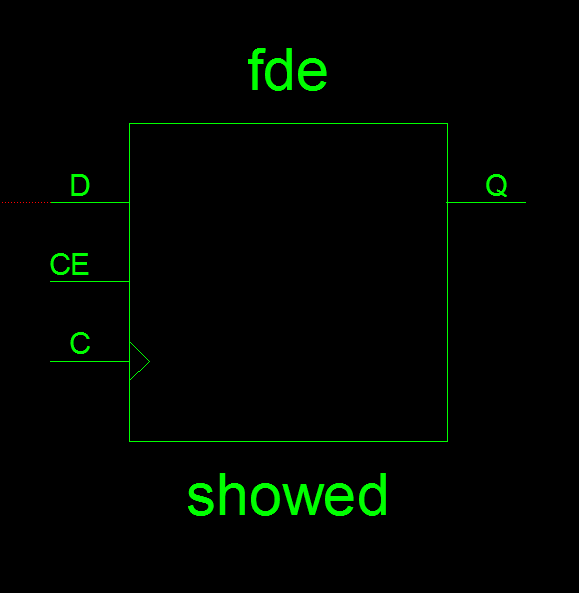
**Clock Divider Module**



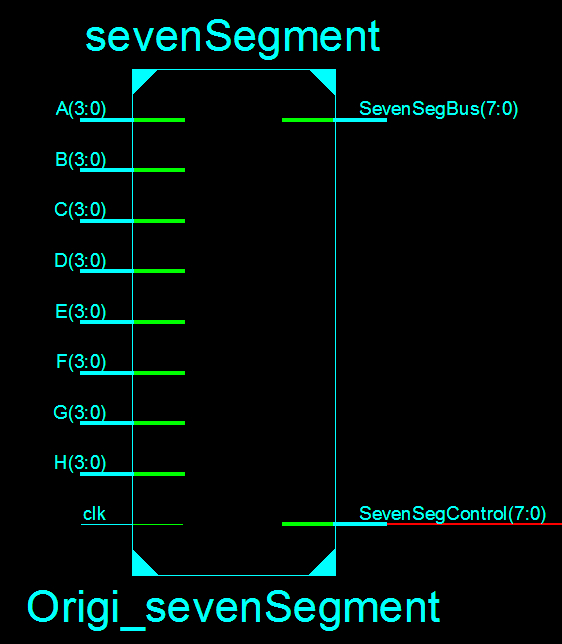
**Life Count Module**



**Show Module**



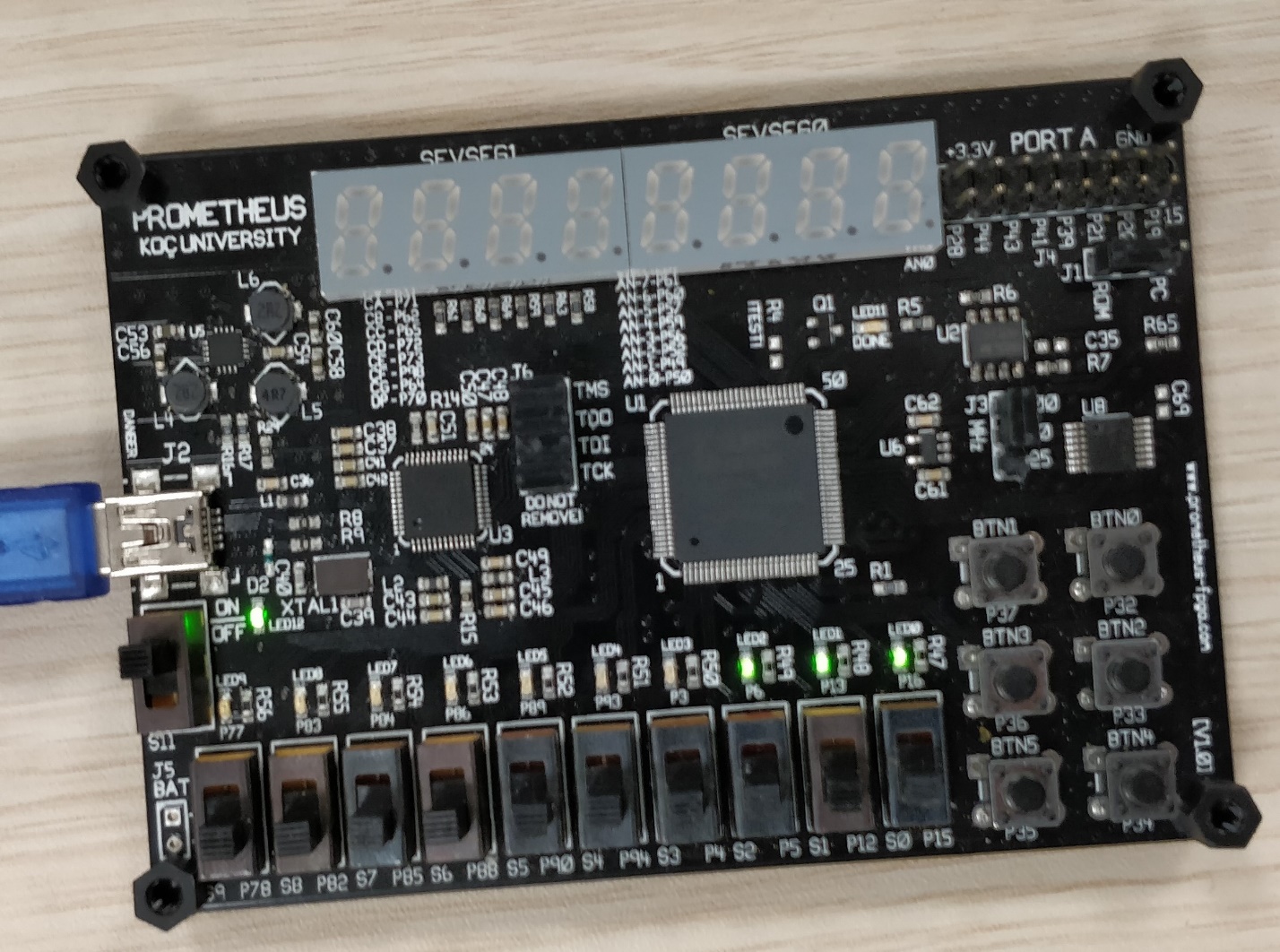
**Seven Segment Module**



**Appendix 3. FPGA Board photos showing working code**

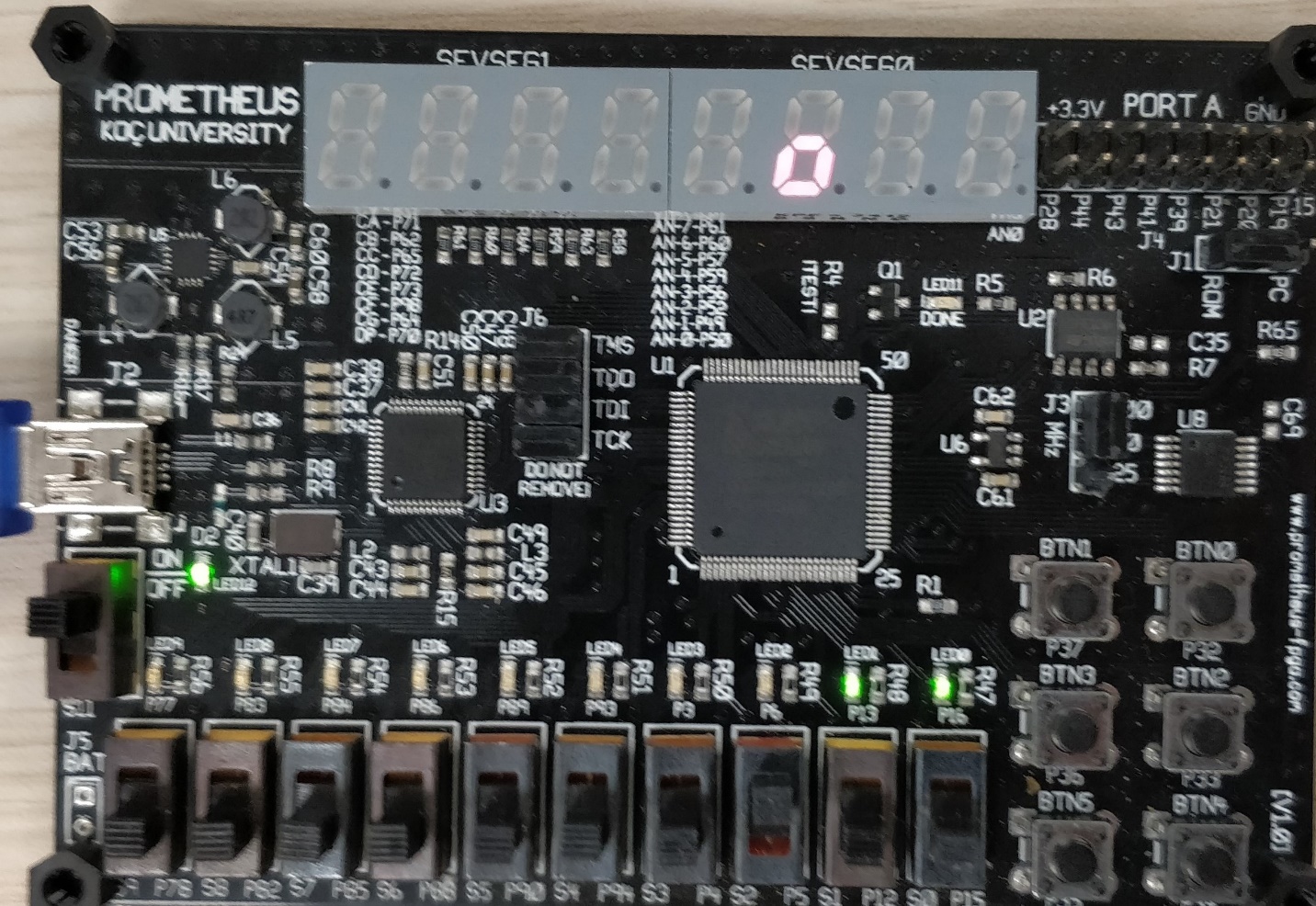
**TESTING GAME**

1. Opening



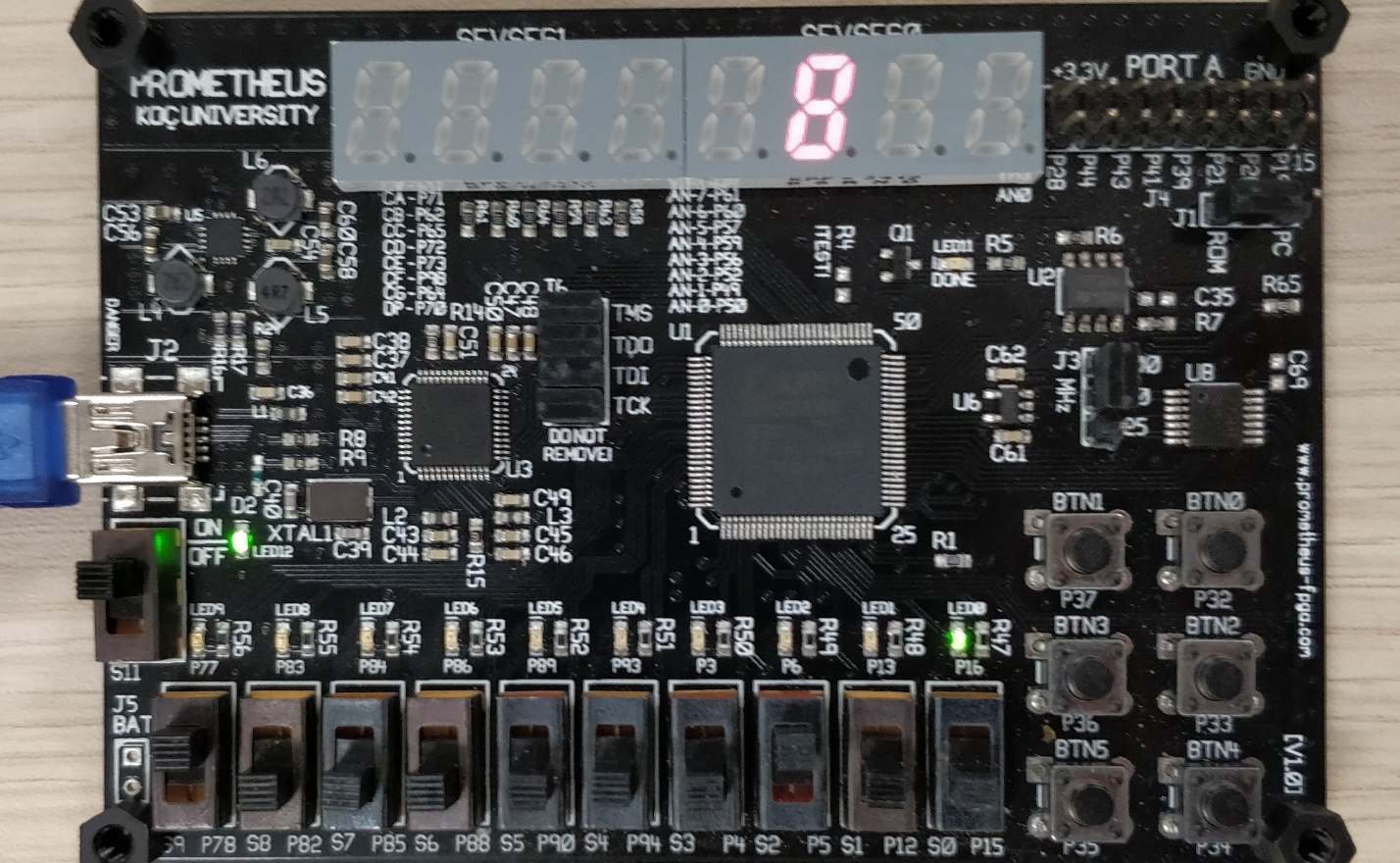
This is the opening scene of the game. As seen right most three LED is lighting and this means we have three tries (life).

1. First Try



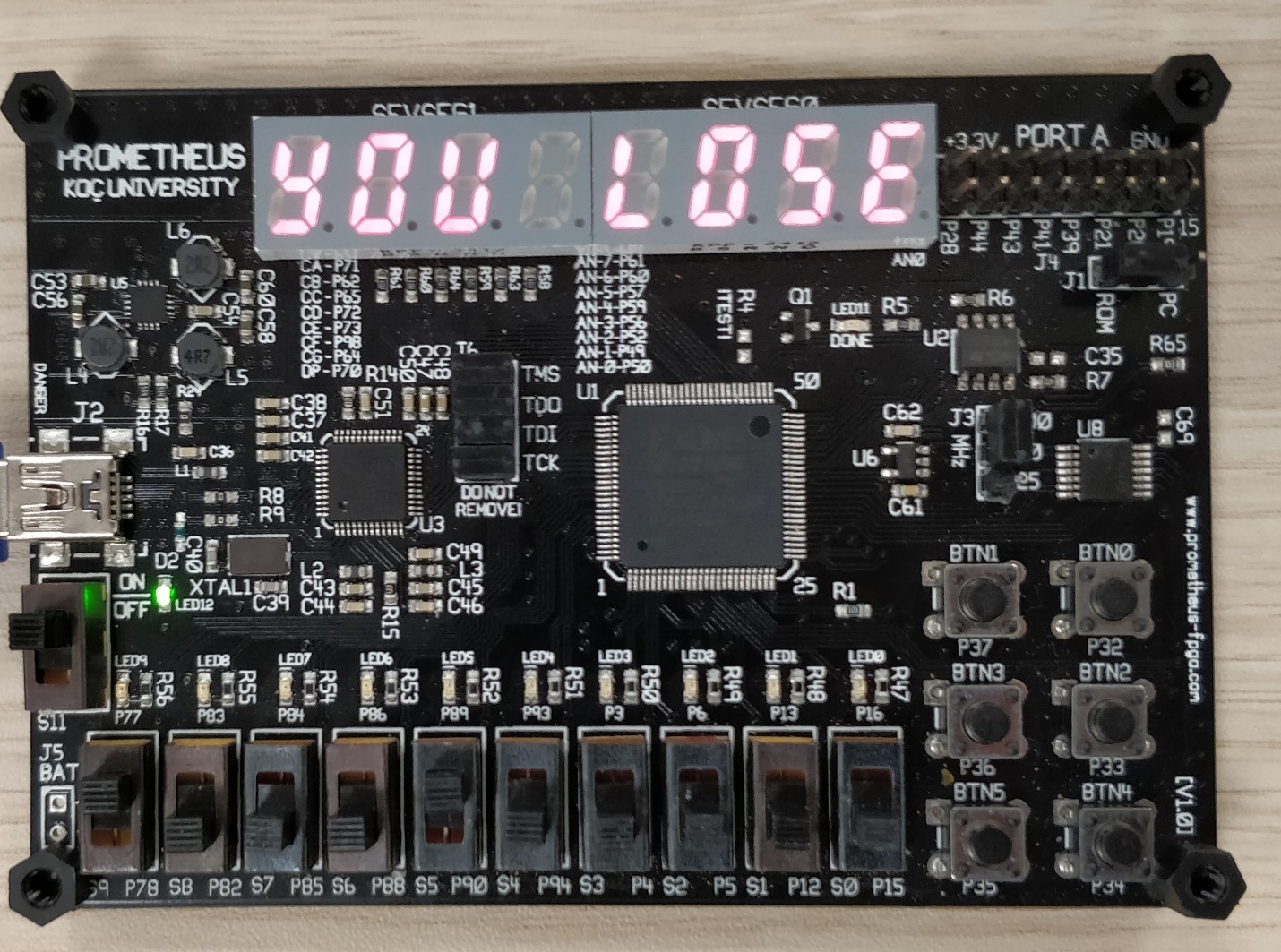
This is the first try. Our try was not correct so we have 2 tries remaining and we can see that there are 2 LEDs lighting. And we can see our false try.

1. Second Try



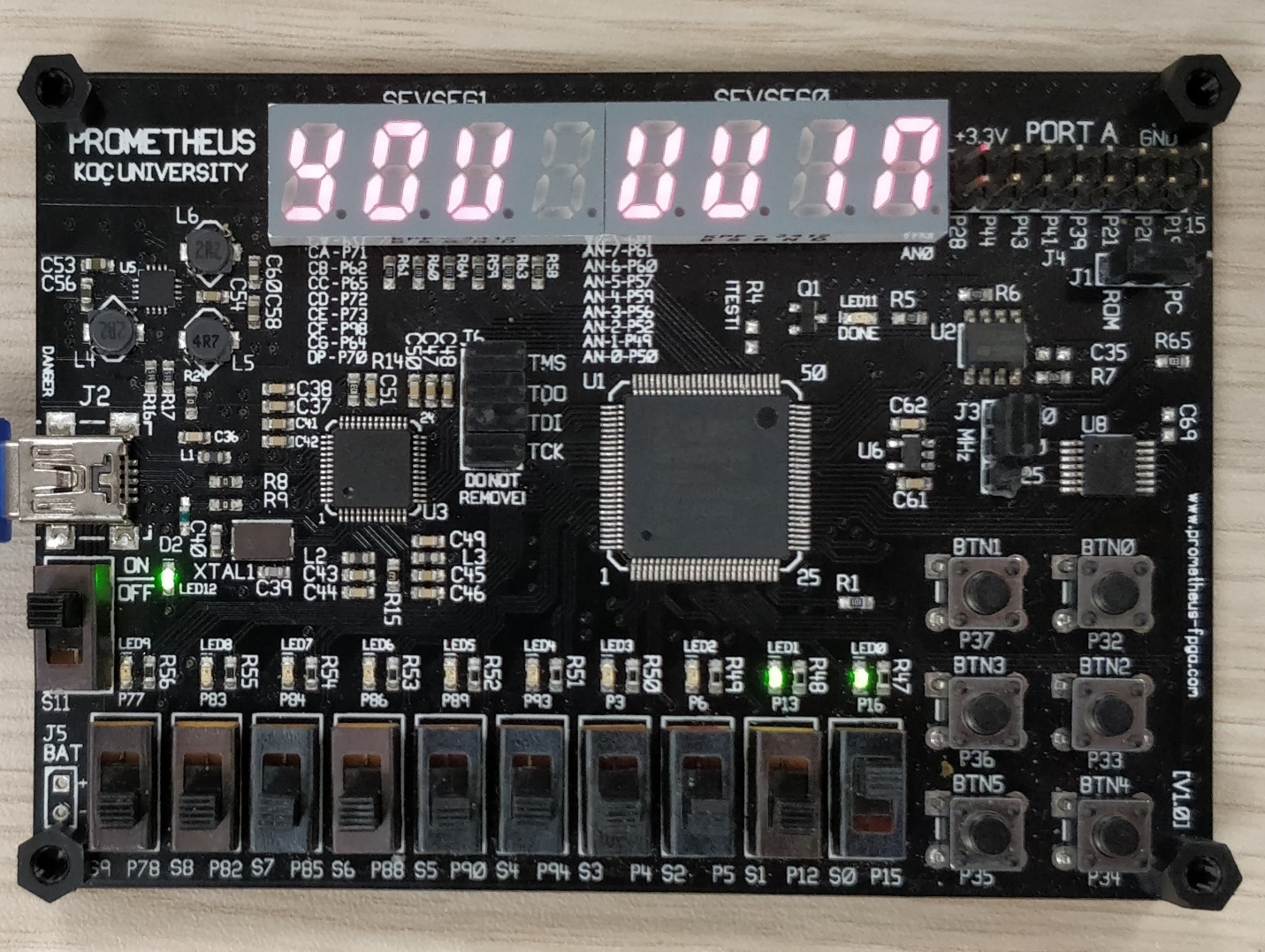
Our second try was above the first try. Therefore, we lighted two of them.

1. Last Try



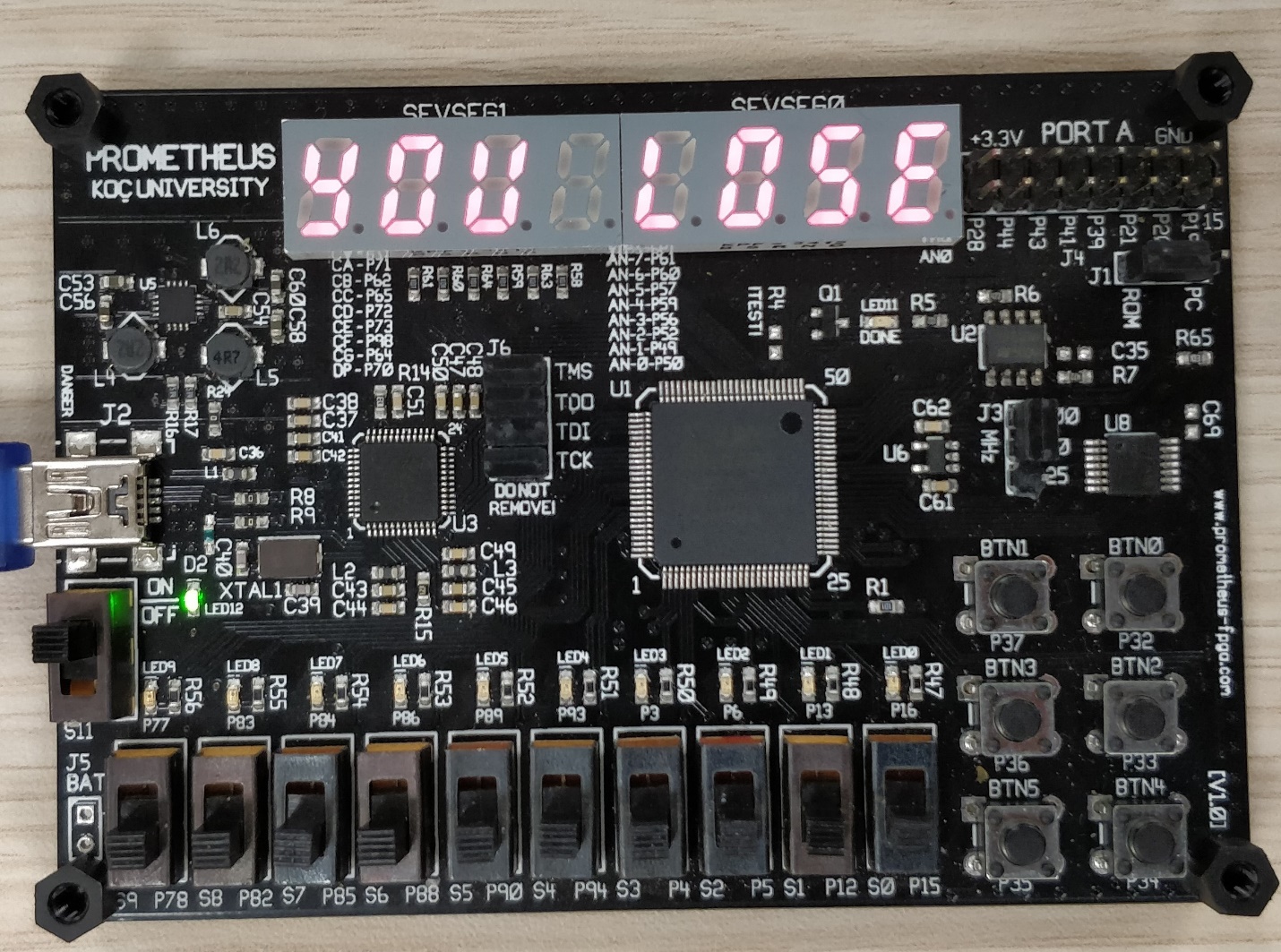
We used our last chance and did not find so we get ‘YOU LOSE’

1. Another Try



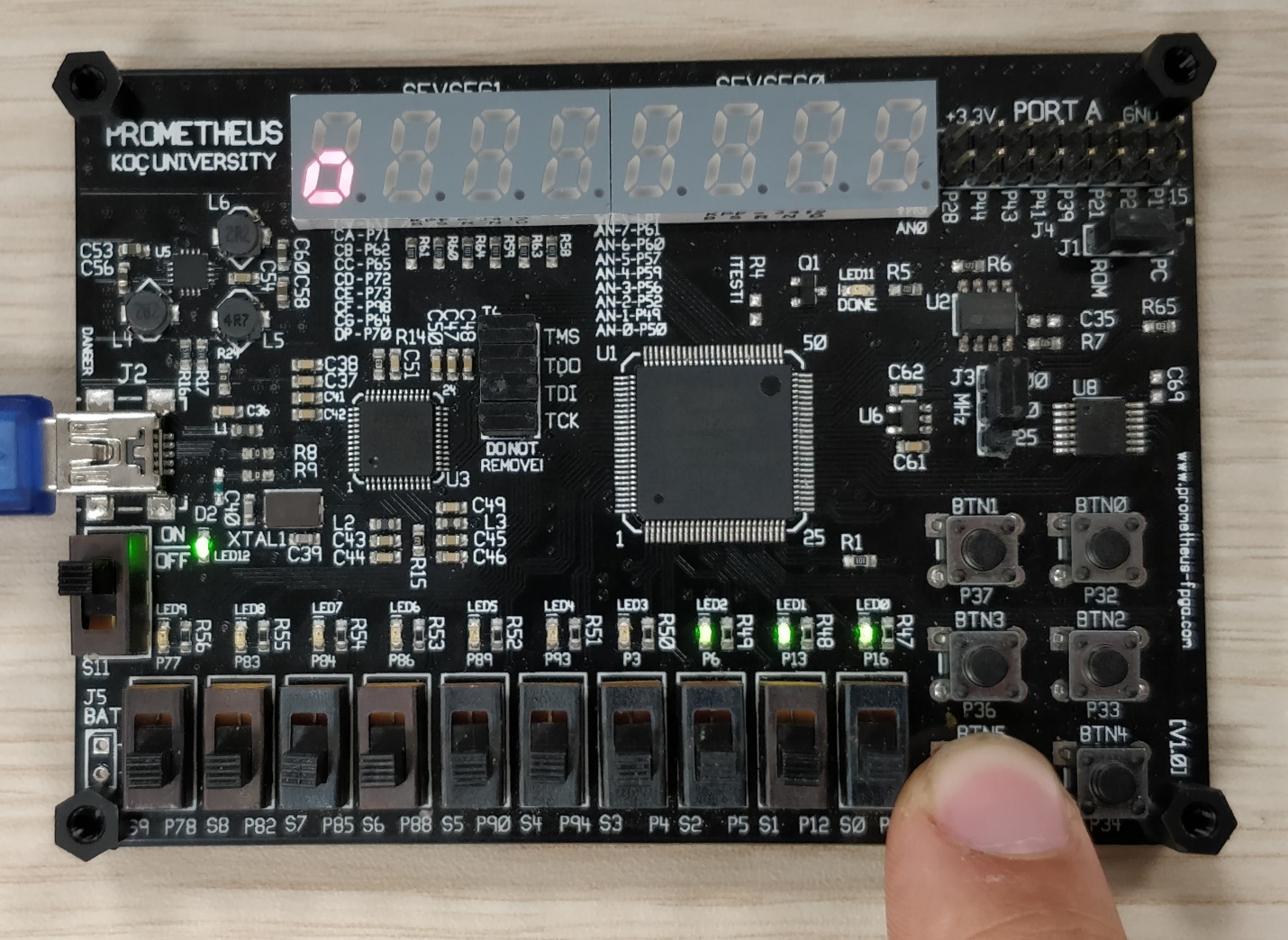
We tried again and we found in 1 try so we get ‘YOU WIN’

1. Another Try



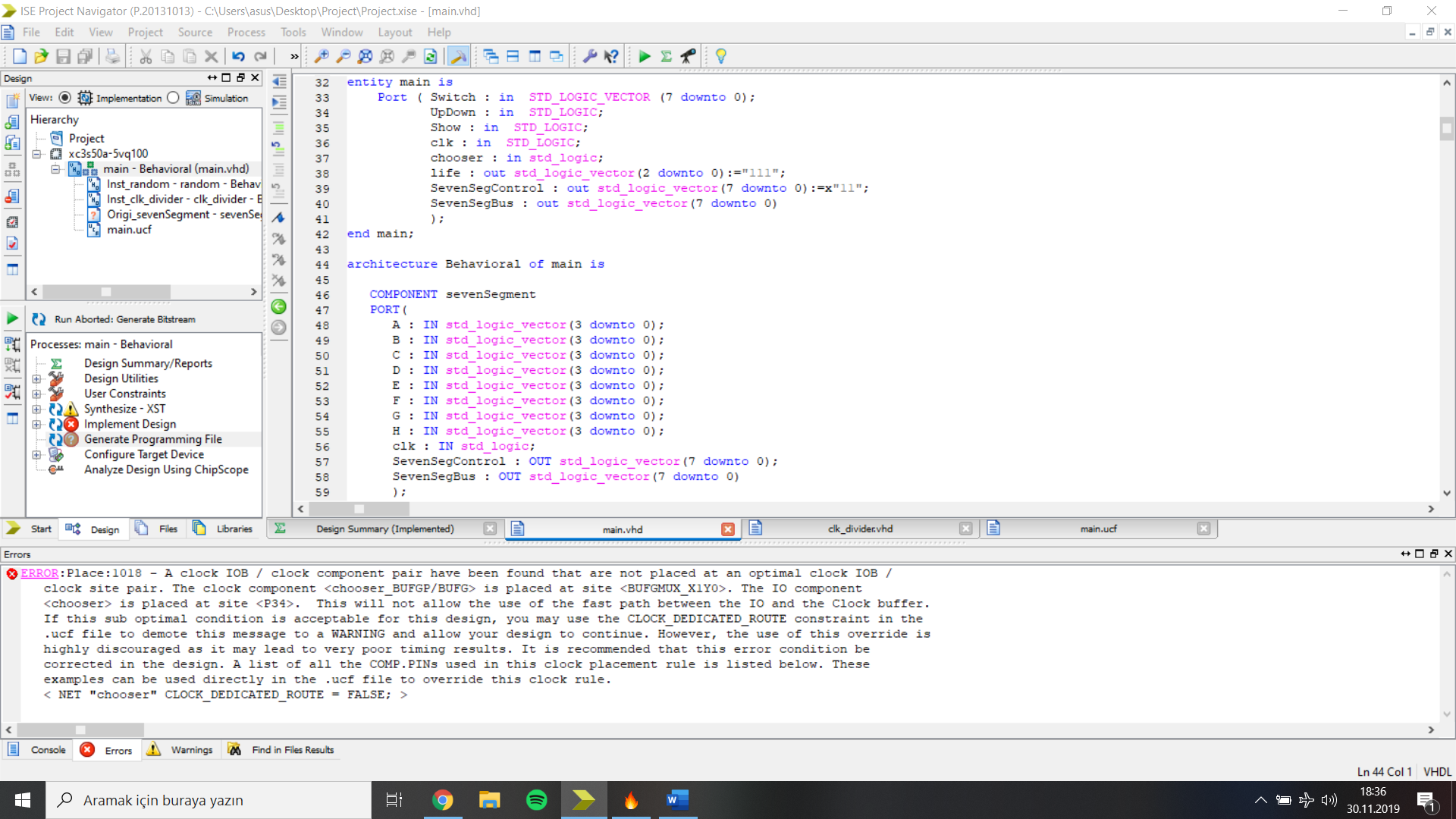
We tried again and we did not find. Let’s see where that was…

1. Where It Was



If we keep pushing the show button we can see where the correct square was. After pushing we get ‘YOU LOSE’ again.

**Appendix 4. Screenshots from Xilinx for the errors and other board issues**



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| clk | Chse | gVl(3) | gVl(2) | gVl(1) | gVl(0) | DgVl(3) | DgVl(2) | DgVl(1) | DgVl(0) |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

**State Table For Random Number Generator**

gVal represents genVal in the code. genVal is an integer in the code but to show in the state diagram it is shown in binary. genVal takes the values from 0 to 15 so it can be written in 4 bits.

**Truth Table:**

X is not in the code, it is used to represent if the chosen switch was right or not. UD represents UpDown in the code. Up represents if the upper box is light up or not, Down represents if the down box is light up or not both of them are not in the code as shown but as a condition. The truth table represents for one switch.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Switch(3) | UD | Shw | Bomb(3) | Checker(3) | Life(2) | Life(1) | Life(0) | Up | Down | X |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |