EE 102 PROJECT

https://youtu.be/QqmCzYrXEKU

Melih Kutay Yağdereli

22002795 Section 2

Purpose:

The main purpose of this design is to implement a game on FGPA Basys-3 with VHDL and displaying on a screen. This game will be controlled by single player and displays the score on the seven-segment display.

Design Specifications:

This game will use a small square to indicate the player and the main goal to stay on the screen by passing through holes on the vertical lines. The game will have. In the first mode, there will be moving vertical lines with different gape sizes. The player will control the ball's position with a button. The score is calculated according to the line that had passed. The score will be shown on the Basys-3 seven-segment screen.

This project uses VGA in order to display the game on a 60hz 680 x 480 pixel resolution screen. The timings for front porch back porch and the syncs pulse are adjusted from internet. The VGA timing module counts according to these timings with using 25mhz clock and determines the visible area.

There is game logic module. The first part of the module utilizes a small finite state machine in order to switch between start mode and game over mode. It checks whether reset button is activated or not and if reset is activated it resets the vertical bars and the score. after that button pulse and the fall and jump values defined and by creating a simple shift register the jump value subtracted from the Y value of the player and fall value added so that a jump motion created for the player. Each time player passes through a bar the current score is updated by one and after crashed state is reaches it checks whether it is a high score or not. The movement of the pipes defined as in each frame the pipes move two pixels left and when they reach the one end of the screen they move back to other end of screen with different positioned gate and the new location of the gape determined by the pseudo number generator.

Colors module defines the color of the all the objects defined in the project namely the player, the bars and the background. And it also determines the shape of the player In the last there part of the code I used code from a GitHub user nurizmn. The original code can be found here : FlappybirdVHDL/FlappyBird at master · nurizmn/FlappybirdVHDL · GitHub

.

Clock Divider module creates a 25mHz clock for VGA signals by dividing the 100mHz clock.

Random number generator module creates a random number using the pseudo algorithm and outputs it. IN this part of the project I used code from : Pseudo Random Number Generator using LFSR in VHDL - Stack Overflow

Seven segment Display module gets the high score and the current score data and selects the anodes and pins accordingly to these data. In this part of code I also used code from a GitHub user nurizmn.

TOP MODULE:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity topModule is
	port(
		clk: in std_logic;
		btn: in std_logic_vector(3 downto 0);
		btnud: in std_logic_vector(1 downto 0);
		Hsync: out std_logic;
		Vsync: out std_logic;
		red: out std_logic_vector(2 downto 0);
		green: out std_logic_vector(2 downto 0);
		blue: out std_logic_vector(2 downto 0);
		anot: out std_logic_vector(3 downto 0);
		seg: out std_logic_vector(6 downto 0);
		dp: out std_logic
	);
end topModule;

architecture Behavioral of topModule is
   signal VGA_VS,VGA_HS,xend,yend,rst,blank: std_logic;
	signal pixelX, pixelY: std_logic_vector(9 downto 0);
	signal pixX, pixY: unsigned(9 downto 0);
	signal rgb: std_logic_vector(8 downto 0);
```

```
    signal PlayerY: unsigned(9 downto 0):= to_unsigned(200,10);
    signal playerX: unsigned(9 downto 0):= to_unsigned(100,10);
    signal PlayerOn,  BarOn: std_logic;
    signal PlayerRGB,  BarRGB: std_logic_vector(7 downto 0);
    signal Crash: boolean;

        signal clock25mhz: std_logic;


        signal random: unsigned (pixelX'range) := (others => '0');

    signal dataIn: std_logic_vector(15 downto 0);
    signal dpIn: std_logic_vector(3 downto 0);
    signal blankSegments: std_logic_vector(3 downto 0);


    signal CurrentscoreOut: std_logic_vector(7 downto 0);
    signal highscoreOut: std_logic_vector(7 downto 0);


    signal button0, button0Pulse: std_logic;

begin

vgaController: entity work.vgaController
        port map(clk=>clock25mhz, reset=>rst, VGA_HS=>VGA_HS, VGA_VS=>VGA_VS,
pixelX=>pixelX, pixelY=>pixelY, blank=>blank,
                                        Yend=>Yend, Xend=>Xend);
buttonHandler: entity work.buttonHandler
```

```
                port map (btn=>button0, clk=>clock25mhz, reset=>rst,
buttonPulse=>button0Pulse);

clockDivider: entity work.clockDivider

        port map (clock=>clk, clockDividedByFour=>clock25mhz, reset=>rst);

gamelogic: entity work.gamelogic

             port map( clk => clock25mhz, btn=>btn, button0Out=>button0,
button0pulse=>button0pulse,

                    random=>random,pixelX=>pixelX,pixelY=>pixelY,

                    PlayerYOut=>PlayerY,PlayerOn=>PlayerOn,crash=>crash,

                    BarOnOut=>BarOn, highscoreOut=>highscoreOut,

                    CurrentscoreOut=>CurrentscoreOut, rgb=>rgb, red=>Red,

                    green=>green, blue=>Blue, VGA_HS=>VGA_HS, VGA_VS=>VGA_VS,
Hsync=>Hsync, VGA_VSync=>VGA_VSync,

                    rst=>rst);


Colors: entity work.Colors

        port map( blank=>blank, PlayerY=>PlayerY, PlayerX=>PlayerX,
PlayerOnOut=>PlayerOn,

                BarOn=>BarOn, PlayerRGBOut=>PlayerRGB,

                BarRGBOut=>BarRGB,

                pixelX=>pixelX, pixelY => pixelY, rgb=>rgb);


randomnumberGenerator: entity work.randomnumberGenerator

             port map( pixelX=>pixelX, randomOut=>random, clk=>clock25mhz,
pixX=>pixX, pixY=>pixY);


sevensegmentDisplay: entity work.sevensegmentDisplay

        port map(clk=>clock25mhz, dataIn=>dataIn, dpIn=>dpIn, Blank=>blankSegments,
seg=>seg, dp=>dp, an=>anot);
```

```
    pixX <= unsigned(pixelX);
    pixY <= unsigned(pixelY);

    dataIn <= highscoreOut & CurrentscoreOut;
    blankSegments <= "0000";
    dpIn <= "0100";
 end Behavioral;
```

gamelogic:

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

entity gamelogic is
    Port ( clk: in std_logic;
           random: in unsigned(9 downto 0);
           pixelX, pixelY: in std_logic_vector(9 downto 0);
           PlayerYOut: out unsigned(9 downto 0);
           Crash: out boolean;
           PlayerOn: in std_logic;
           HighScoreOut, CurrentScoreOut: out std_logic_vector(7 downto 0);
           rgb: in std_logic_vector(8 downto 0);
           RED,green,blue: out std_logic_vector(2 downto 0);
           VGA_HS, VGA_VS: in std_logic;
           Hsync, Vsync: out std_logic;
           btn: in std_logic_vector(3 downto 0);
           button0Out: out std_logic;
           button0pulse: in std_logic;
           rst,BarONOut: out std_logic
    );
end gamelogic;
```

```
architecture Behavioral of gamelogic is

constant BITS_OF_DEBOUNCE_TIMER: natural := 18;
signal debounceCounterValue, debounceCounterValueNext: unsigned(
BITS_OF_DEBOUNCE_TIMER - 1 downto 0) := (others=>'0');

type state is (start, play);
signal stateRegister, stateNext: state;

signal button0, button0Next: std_logic;
signal jumpRegister, jumpRegisterNext: unsigned(4 downto 0) := "00000";
signal fallRegister, fallRegisterNext: unsigned(3 downto 0) := "0000";
signal birdDelay, birdDelayNext: unsigned(20 downto 0):= to_unsigned(0,21);

signal firstDigit, firstDigitNext, SecondDigit, SecondDigitNext: unsigned (3 downto 0);
signal HighestScore, HighestScoreNext: unsigned (7 downto 0);

signal highscores: std_logic_vector(7 downto 0) := (others=>'0');
signal currentscores: std_logic_vector(7 downto 0) := (others=>'0');

signal Xpos, Ypos: unsigned(9 downto 0);

signal col1XNext, col1YNext: unsigned (9 downto 0) := (others => '1');
signal col2XNext, col2YNext: unsigned (9 downto 0):= (others => '1');
signal col3XNext, col3YNext: unsigned (9 downto 0):= (others => '1');
signal PlayerYNext, PlayerYTemp, PlayerYTemp2: unsigned(9 downto 0):=
to_unsigned(200,10);
```

```
signal col1X, col1Y: unsigned (pixelX'range) := (others => '1');
signal col2X, col2Y: unsigned (pixelX'range) := (others => '1');
signal col3X, col3Y: unsigned (pixelX'range) := (others => '1');
signal PlayState, col1On, col2On, col3On: std_logic := '0';
signal delay, delayNext: Natural := 0;


signal PlayerY: unsigned(9 downto 0);
signal BarON: std_logic;
signal crashed: std_logic;

begin

Xpos <= unsigned(pixelX);
Ypos <= unsigned(pixelY);

process(clk, btn)
begin
   rst <='0';
   if btn(3)='1' then
      rst <= '1';
      PlayerYTemp2 <= to_unsigned(200,10);
      birdDelay <= (others=>'0');
      col1X <= (others => '0');
      col2X <= (others => '0');
      col3X <= (others => '0');
      delay <= 0;
      stateRegister <= start;
      HighestScore <= (others => '0');
      firstDigit <= (others => '0');
```

```
    SecondDigit <= (others => '0');
    Crash <= false;
elsif (clk'event and clk='1') then
    Hsync <= VGA_HS;
    Vsync <= VGA_VS;
    RED <= rgb(8 downto 6);
    GREEN <= rgb(5 downto 3);
    BLUE <= rgb(2 downto 0);
    PlayerYTemp2 <= PlayerYNext;
    birdDelay <= birdDelayNext;

    HighScoreOut <= highscores;-- next states also act as buffer
    CurrentScoreOut <= currentscores;
    stateRegister <= stateNext;

    debounceCounterValue <= debounceCounterValueNext;
    button0 <= button0Next;

    if button0Pulse='1' then
        Crash <= false;
        jumpRegister <= "10111";
        fallRegister <= "0000";
    else
        jumpRegister <= jumpRegisterNext;
        fallRegister <= fallRegisterNext;
    end if;

    --signals for columns
    col1X <= col1XNext;
    col2X <= col2XNext;
    col3X <= col3XNext;
    col1Y <= col1YNext;
```

```
        col2Y <= col2YNext;
        col3Y <= col3YNext;
        delay <= delayNext;

        --Signals for Score
        HighestScore <= HighestScoreNext;
        firstDigit <= firstDigitNext;
        SecondDigit <= SecondDigitNext;

        if (crashed = '1') then
            Crash <= true;
        end if;
    end if;
end process;


PlayerYOut <= PlayerYTemp2;

--Debounce Logic
debounceCounterValueNext <= debounceCounterValue + 1;
button0Next <= '1' when (btn(0)='1' and debounceCounterValue = 0) else
                    '0' when (btn(0)='0' and debounceCounterValue = 0) else
                   button0;
button0Out<=button0;


PlayState <= '1' when delay = 200000 else '0';
delayNext <= 0 when delay > 200000 else delay + 1;

process(birdDelay, button0Pulse, jumpRegister, fallRegister)
        begin
                if birdDelay=0 then
```

```
                if jumpRegister/="00000" then
                        jumpRegisterNext <= '0' & jumpRegister(4 downto 1);
                        fallRegisterNext <= fallRegister;
                else
                        jumpRegisterNext <= jumpRegister;
                        fallRegisterNext <= fallRegister(2 downto 0) & '1';
                end if;
        else
                jumpRegisterNext <= jumpRegister;
                fallRegisterNext <= fallRegister;
        end if;
    end process;


    birdDelayNext <= birdDelay+2;
    PlayerYTemp <= to_unsigned(200,10) when (stateRegister=start and
button0Pulse='1') else
                                        (PlayerYTemp2 - jumpRegister + fallRegister)
when birdDelay=0 and stateRegister/=start
                                        else PlayerYTemp2;
    PlayerYNext <= PlayerYTemp when (PlayerYTemp>=0 and PlayerYTemp<425) else
                                                (others=>'0');
    PlayerY <= to_unsigned(100,10);
    PlayerYOut<= PlayerY;



    firstDigitNext <= firstDigit + 1 when (PlayerY = col1X or PlayerY = col2X or
PlayerY = col3X) and PlayState = '1' else
        (others => '0') when (stateRegister=start and button0Pulse='1') or firstDigit = 9
else
        firstDigit;
SecondDigitNext <= SecondDigit + 1 when firstDigit = 9 else
        (others => '0') when (stateRegister=start and button0Pulse='1') else
        SecondDigit;
```

```
HighestScoreNext <= (SecondDigit & firstDigit) when ((SecondDigit&firstDigit) >
HighestScore and crashed='1') else HighestScore;


highscores <= std_logic_vector(HighestScore);
currentscores <= std_logic_vector(SecondDigit & firstDigit);



    process (stateRegister, button0Pulse, crashed, col1X, col2X, col3X, PlayState,
            col1On, col2On, col3On, random, col1Y, col2Y, col3Y)
    begin
        --Defaults
        stateNext <= stateRegister;
        col1XNext <= col1X;
        col2XNext <= col2X;
        col3XNext <= col3X;
        col1YNext <= col1Y;
        col2YNext <= col2Y;
        col3YNext <= col3Y;

        case stateRegister is
            when start =>
                if (btn(0)='1') then
                    stateNext <= play;
                    col1XNext <= to_unsigned((638+80), 10);
                    col2XNext <= (others => '1');
                    col3XNext <= (others => '1');
                    col1YNext <= random;
                else
                    stateNext <= start;
                end if;
            when play =>
```

```
            if PlayState = '1' then
               if col1On = '1' then
                  col1XNext <= col1X - 2;
               end if;
               if col2On = '1' then
                  col2XNext <= col2X - 2;
               end if;
               if col3On = '1' then
                  col3XNext <= col3X - 2;
               end if;
            end if;
            if col3X = 450 then
               col1XNext <= to_unsigned((638+80), 10);
               col1YNext <= random;
            end if;
            if col2X = 450 then
               col3XNext <= to_unsigned((638+80), 10);
               col3YNext <= random;
            end if;
            if col1X = 450 then
               col2XNext <= to_unsigned((638+80), 10);
               col2YNext <= random;
            end if;
            if crashed = '1'  then
               stateNext <= start;
            else
               stateNext <= play;
            end if;
      end case;
   end process;

col1On <= '1' when col1X <= (639+80) and col1X > 0 else '0';
```

```
col2On <= '1' when col2X <= (639+80) and col2X > 0 else '0';
col3On <= '1' when col3X <= (639+80) and col3X > 0 else '0';

BarON <= '1' when     (col1On='1' and ((Xpos >= col1X - 30 and Xpos < col1X) or (col1X <= 30 and Xpos < col1X)) and (Ypos <= col1Y or Ypos > col1Y+100)) or
                (col2On='1' and ((Xpos >= col2X - 30 and Xpos < col2X) or (col2X <= 30 and Xpos < col2X)) and (Ypos <= col2Y or Ypos > col2Y+150)) or
                (col3On='1' and ((Xpos >= col3X - 30 and Xpos < col3X) or (col3X <= 30 and Xpos < col3X)) and (Ypos <= col3Y or Ypos > col3Y+80))
            else '0';
BarONOut<=BarON;

crashed <= '1' when PlayerOn='1' and  BarON='1' else '0';

end Behavioral;
```

Colors:

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

entity Colors is
    Port ( blank: in std_logic;
        PlayerY, PlayerX: in unsigned(9 downto 0);
        PlayerOnOut: out std_logic;
        BarOn: in std_logic;
        PlayerRgbOut: out std_logic_vector(8 downto 0);
        BarRGBOut: out std_logic_vector(8 downto 0);
        pixelX: in std_logic_vector(9 downto 0);
        pixelY: in std_logic_vector(9 downto 0);
        rgb: out std_logic_vector(8 downto 0));
end Colors;
```

```
architecture Behavioral of Colors is

signal pixX, pixY: unsigned(9 downto 0);
signal Background: std_logic_vector(8 downto 0);



signal PlayerOn: std_logic;


signal PlayerRGB: std_logic_vector(8 downto 0);
signal BarRGB: std_logic_vector(8 downto 0);


begin

pixX <= unsigned(pixelX);
pixY <= unsigned(pixelY);

PlayerOnOut <= PlayerOn;

PlayerRgbOut <= PlayerRGB;
BarRGBOut <= BarRGB;
      PlayerOn <= '1' when
                (pixY > PlayerY and pixY < PlayerY + 25 and pixX > PlayerX and pixX <
PlayerX + 33)
                else '0';
     PlayerRGB <= "111111111";


          BarRGB <= "111111111";
   Background <= "000000000";
   rgb <= (others=>'0')     when blank='1' else
```

```
        BarRGB          when BarOn='1' else
        PlayerRGB when PlayerOn='1' else
        Background;

end Behavioral;
```

Clock Divider:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use IEEE.STD_LOGIC_unsigned.all;

entity clockDivider is
port ( clock,reset: in std_logic;
    clockDividedByFour: out std_logic);
end clockDivider;

architecture behavioral of clockDivider is

signal divider: std_logic_vector(1 downto 0);

begin

process(clock,reset)
begin
if(reset = '1') then
divider <= "00";
elsif rising_edge(clock) then
divider <= divider + '1';
end if;
end process;
clockDividedByFour <= divider(1);
```

```
end behavioral;

pseudorandomNumberGenerator:
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity pseudorandomNumberGenerator is
  Port ( pixelX: in std_logic_vector(9 downto 0);
         randomOut: out unsigned (9 downto 0);
         clk: in std_logic;
         pixX,pixY: in unsigned(9 downto 0)
         );
end pseudorandomNumberGenerator;

architecture Behavioral of pseudorandomNumberGenerator is
    signal Q, QTemp: unsigned (pixelX'range) := (others => '0');
begin

           process(clk)
           begin
                    if rising_edge(clk) then
           QTemp <=  Q(6 downto 5) & (pixX(3 downto 1) xor pixY(8 downto 6)) & Q(7) &
(pixX(5 downto 2) xor pixY(9 downto 6));
           Q <= "00" & QTemp(9 downto 6) & "1111";
                    end if;
           end process

    RandomOut <= Q;
end Behavioral;
Seven Segment Display:
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sevenSegmentDisplayModule is
generic(DIVIDING_BITS: natural := 15);
	port(
	clk: in std_logic;
	dataIn: in std_logic_vector(15 downto 0);
	dpIn: in std_logic_vector(3 downto 0);
	blank: in std_logic_vector(3 downto 0);
	seg: out std_logic_vector(6 downto 0);
	dp: out std_logic;
	an: out std_logic_vector(3 downto 0)
	);
end sevenSegmentDisplayModule;

architecture Behavioral of sevenSegmentDisplayModule is
	signal count: std_logic_vector(DIVIDING_BITS-1 downto 0) := (others=>'0');
	signal digit: std_logic_vector(1 downto 0);
	signal pinselect: std_logic_vector(3 downto 0);
begin
	process(clk)
	begin
		if (clk'event and clk='1') then
			count <= std_logic_vector(unsigned(count) + 1);
		end if;
	end process;

	digit <= count(DIVIDING_BITS-1 downto DIVIDING_BITS-2);

	with digit select
		an <=
```

```
                "111" & blank(0) when "00",
                "11" & blank(1) & '1' when "01",
                '1' & blank(2) & "11" when "10",
                blank(3) & "111" when others;

with digit select
        pinselect <=
                dataIn(3 downto 0) when "00",
                dataIn(7 downto 4) when "01",
                dataIn(11 downto 8) when "10",
                dataIn(15 downto 12) when others;

with digit select
        dp <=
                not dpIn(0) when "00",
                not dpIn(1) when "01",
                not dpIn(2) when "10",
                not dpIn(3) when others;

with pinselect select
seg <= "1000000" when "0000",
                        "1111001" when "0001",
                        "0100100" when "0010",
                        "0011001" when "0100",
                        "0010010" when "0101",
                        "0000010" when "0110",
                        "1111000" when "0111",
                        "0000000" when "1000",
                        "0010000" when "1001",
                        "0001000" when "1010",
                        "0000011" when "1011",
                        "1000110" when "1100",
```

```
                    "0100001" when "1101",
                    "0000110" when "1110",
                    "0001110" when others;

end Behavioral;
```