EEE 102

Term Project: Not Hitting The Obstacles

EEE

Section-006

Kadir Metehan Çalışkan

Content Table

Youtube Video:

https://www.youtube.com/watch?v=8TaSRi_iNZ0

Description:

The game includes 3 lanes. The obstacles come pseudo-randomly in these 3 lanes. The purpose of the game is to continue the game as long as you can without hitting the pseudo-random obstacles and, increase your score as much as possible. The score is determined by using a counter and it is displayed on the seven-segment display. The counter counts the second. There are 3 lives right and it is displayed on LEDs (L1, P1, N3). If all lives right are over, the game is over and generating of obstacles is stopped.

Background:

Linear Feedback Shift Register:

LFSR is formed from a simple shift register with feedback from two or more points in shift register by using XOR gate and this situation generates function to endlessly cycle through a sequence of patterns. (*Tutorial: Linear Feedback Shift Registers*, n.d.)

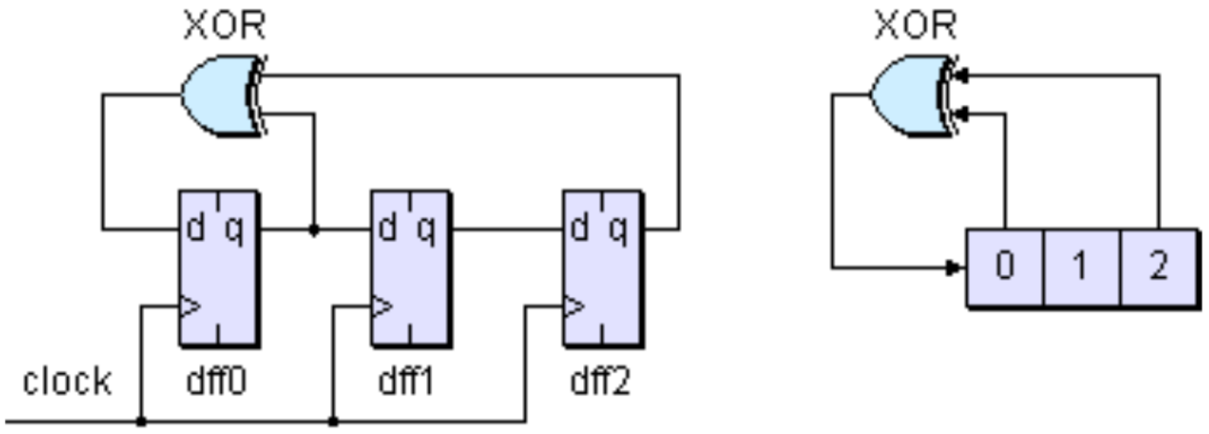

Figure 1: Linear Feedback Shift Register with XOR gate.

Thanks to LFSR, pseudo-random obstacles was generated.

Design Methodology:

1. Lane Generation:

Firstly, different pathway is created by using shift register. Given code takes 4 inputs (game_clock, obst_in, dead, reset) and it creates 32- bit binary number as output. The reason of the output is 32- bit binary number is to increase scalability of the movement of obstacles.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
```

```
entity shift_register is
   Port ( game_clock : in STD_LOGIC;
                   obst_in : in STD_LOGIC;
                                          dead : in STD_LOGIC;
                                   reset : in STD_LOGIC;
        path : out  STD_LOGIC_VECTOR (31 downto 0));
end shift_register;

architecture Behavioral of shift_register is
  signal lane : std_logic_vector (31 downto 0) := "00000000000000000000000000000000";
begin
  process (game_clock, obst_in, dead, reset)
        begin
          if ((reset = '1') or (dead = '1')) then
                  lane <= "00000000000000000000000000000000";
          elsif (rising_edge(game_clock)) then
                  lane <= obst_in & lane (31 downto 1);
                end if;
        end process;

        path <= lane;



end Behavioral;
```

2. Obstacles Generation:

In this part, Linear feedback shift register is used to create obstacles which come pseudo-randomly. The output of this module must be 32-bit binary number because our pathways consist of 32- bit binary number. Also, if “dead” goes to “1”, generation of obstacles is stopped and all obstacles in the screen disappear. However, if “reset” goes to “1”, the generation of obstacles starts again.

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;


entity Linear_Feedback_shift_reg is
   port ( game_clock : in std_logic;
                        dead : in std_logic;
                       reset : in std_logic;
                                      obst_out : out std_logic);
end Linear_Feedback_shift_reg;


architecture Behavioral of Linear_Feedback_shift_reg is
   signal linear_feedback_shift_register : std_logic_vector (31 downto 0) :=
"00000000000000000000000000100000";
            signal linear_feedback_shift_register_next : std_logic_vector (31 downto 0) :=
"00000000000000000000000000100000";
begin


   process(game_clock, reset)
            begin
              if (dead = '1') then
                        linear_feedback_shift_register <= "00000000000000000000000000000000";
              elsif (reset = '1') then
                        linear_feedback_shift_register <= "00000000000000000000000000100000";
              else
                                if (rising_edge(game_clock)) then
                                  linear_feedback_shift_register <= linear_feedback_shift_register_next;
                        end if;
                      end if;
            end process;


            process(linear_feedback_shift_register)
```

```
        begin

           linear_feedback_shift_register_next (31 downto 1) <= linear_feedback_shift_register (30
downto 0);

                linear_feedback_shift_register_next (0) <= linear_feedback_shift_register(31) xor
linear_feedback_shift_register(5);

        end process;


        obst_out <= linear_feedback_shift_register(0);



end Behavioral;
```

3. Selection of Lane:

This module determines on which lane the obstacles should be. This module includes 16-bit binary number consisting of three of random “1”. The reason of the converting 16-bit binary number to decimal number and deciding according to integer value of 24576 is to increase scalability. If we divide the lane according to different integer, the game still works properly. However, the optimum division value is 24576 because the decision can be made by looking at the first 2 bits of the 16-bit binary number to select 1 of the 3 lanes. Decimal conversion of “1100000000000000” is 49152, decimal conversion of “0110000000000000” is 24576. “0010000000000000” is always less than 24576 when first 2 most significant bits are fixed. Lanes can be selected easily by looking first 2 most significant bits. For instance, “11” goes to top lane.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use IEEE.NUMERIC_STD.ALL;


entity select_lane is

  Port (    game_clock : in  STD_LOGIC;

                 obst_in : in  STD_LOGIC;

      obstacle_1 : out  STD_LOGIC;

      obstacle_2 : out  STD_LOGIC;

      obstacle_3 : out  STD_LOGIC);
```

```
end select_lane;


architecture Behavioral of select_lane is
  signal linear_feedback_shift_register : std_logic_vector (15 downto 0) := "0010000010000001";
          signal linear_feedback_shift_register_next : std_logic_vector (15 downto 0) :=
"0010000010000001";
begin


  process(game_clock)
          begin
                    if (rising_edge(game_clock)) then
                              linear_feedback_shift_register <= linear_feedback_shift_register_next;
                    end if;
            if (obst_in = '1') then
                      if ((conv_integer(linear_feedback_shift_register)) > 49152) then
                        obstacle_1 <= '1';
                                obstacle_2 <= '0';
                              obstacle_3 <= '0';
                      elsif (((conv_integer(linear_feedback_shift_register)) > 24576 )
and((conv_integer(linear_feedback_shift_register)) < 49152)) then
                        obstacle_1 <= '0';
                              obstacle_2 <= '1';
                              obstacle_3 <= '0';
                      elsif ((conv_integer(linear_feedback_shift_register)) < 24576) then
                       obstacle_1 <= '0';
                              obstacle_2 <= '0';
                              obstacle_3 <= '1';
                    else
                      obstacle_1 <= '0';
                              obstacle_2 <= '0';
                              obstacle_3 <= '0';
                    end if;
                    else
                      obstacle_1 <= '0';
               obstacle_2 <= '0';
                    obstacle_3 <= '0';
            end if;
          end process;


          process(linear_feedback_shift_register)
          begin
            linear_feedback_shift_register_next (15 downto 1) <= linear_feedback_shift_register (14
downto 0);
                    linear_feedback_shift_register_next (0) <= linear_feedback_shift_register(15) xor
linear_feedback_shift_register(5);
          end process;


end Behavioral;
```

4. Score Display:

This module consists of 3 entities. First part is the decoder part. Inputs are determined in the "case" state and outputs are transferred to seven segment display. To display only decimal number, we can use binary to BCD converter. The score increases one point per second. Clock pulse can be easily arranged according to desired frequency. The reason of writing complex seven-segment display code is that the score is displayed on seven-segment display and as the score increases from less significant digit to most significant digit, digits show decimal numbers in order. For instance, when the score is "9", seven-segment display shows "9" instead of "0009".

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decoder is

  Port (     alu_value : in std_logic_vector(7 downto 0);
                                                    sign : in std_logic;
                                                           valid : in std_logic;
                 clk : in std_logic;
              anode : out std_logic_vector(3 downto 0);
             sev_seg : out std_logic_vector(6 downto 0));
end decoder;
architecture Behavioral of decoder is
  component binary_bcd_converter
     Port ( binary_input : in std_logic_vector(7 downto 0);
              lsb_out : out std_logic_vector(3 downto 0);
              msb_out : out std_logic_vector(3 downto 0);
             imsb_out : out std_logic_vector(3 downto 0));
  end component;

          component clk_div
                     Port (  clk : in std_logic;
                                         out_clk : out std_logic);
          end component;

  signal   count_digit : std_logic_vector(1 downto 0);
  signal   digit : std_logic_vector (3 downto 0);
  signal   lsb,msb,imsb : std_logic_vector(3 downto 0);
          signal   out_clk : std_logic;

begin
  converter: binary_bcd_converter
          port map ( binary_input => alu_value,
            lsb_out => lsb,
            msb_out => msb,
               imsb_out => imsb);


  my_clk: clk_div
          port map (clk => clk,
                   out_clk => out_clk );
  process (out_clk)
  begin
    if (rising_edge(out_clk)) then
       count_digit <= count_digit + 1;
    end if;
  end process;
  process(digit)
  begin
  case digit is
   when "0000" => sev_seg <= "0000001";
   when "0001" => sev_seg <= "1001111";
   when "0010" => sev_seg <= "0010010";
   when "0011" => sev_seg <= "0000110";
   when "0100" => sev_seg <= "1001100";
   when "0101" => sev_seg <= "0100100";
   when "0110" => sev_seg <= "0100000";
   when "0111" => sev_seg <= "0001111";
   when "1000" => sev_seg <= "0000000";
   when "1001" => sev_seg <= "0000100";
   when others => sev_seg <= "1111111";
   end case;
   end process;


  anode <= "0111" when count_digit = "00" else
```

```
"1011" when count_digit = "01" else
"1101" when count_digit = "10" else
"1110" when count_digit = "11" else
"1111";



process (count_digit, lsb, msb, imsb, sign, valid)
  variable imsb_v, msb_v : std_logic_vector(3 downto 0);
begin
  imsb_v := imsb;
          msb_v := msb;

          if (imsb_v = X"0") then
            if (msb_v = X"0") then
                    msb_v := X"F";
                  end if;
            imsb_v := X"F";
          end if;

          if (valid = '1') then
                  if (sign = '0') then
                          case count_digit is
                                  when "00" => digit <= "1111";
                                  when "01" => digit <= imsb_v;
                                  when "10" => digit <= msb_v;
                                  when "11" => digit <= lsb;
                                  when others => digit <= "0000";
                          end case;
                  else
```

```
                                    case count_digit is
                                                when "00" => digit <= "1110";
                                                when "01" => digit <= imsb_v;
                                                when "10" => digit <= msb_v;
                                                when "11" => digit <= lsb;
                                                when others => digit <= "0000";
                                    end case;
                                    end if;
                        else digit <= "1110";
                        end if;
            end process;


end Behavioral;
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity binary_bcd_converter is
   Port ( binary_input : in std_logic_vector(7 downto 0);
           lsb_out : out std_logic_vector(3 downto 0);
           msb_out : out std_logic_vector(3 downto 0);
          imsb_out : out std_logic_vector(3 downto 0));
end binary_bcd_converter;
architecture Behavioral of binary_bcd_converter is
begin
   process(binary_input)
      variable count_total : INTEGER range 0 to 255 := 0;
```

```
               variable lsb,msb,imsb : INTEGER range 0 to 9 := 0;
begin
            count_total := 0;
 if (binary_input(7) = '1') then count_total := count_total + 128;  end if;
 if (binary_input(6) = '1') then count_total := count_total +  64;  end if;
 if (binary_input(5) = '1') then count_total := count_total +  32;  end if;
 if (binary_input(4) = '1') then count_total := count_total +  16;  end if;
 if (binary_input(3) = '1') then count_total := count_total +   8;  end if;
 if (binary_input(2) = '1') then count_total := count_total +   4;  end if;
 if (binary_input(1) = '1') then count_total := count_total +   2;  end if;
 if (binary_input(0) = '1') then count_total := count_total +   1;  end if;

 msb  := 0;
            imsb := 0;
            lsb  := 0;

 for I in 1 to 2 loop
               exit when (count_total >= 0 and count_total < 100);
               imsb := imsb + 1;
               count_total := count_total - 100;
 end loop;

 for I in 1 to 9 loop
    exit when (count_total >= 0 and count_total < 10);
    msb := msb + 1;
               count_total := count_total - 10;
 end loop;

            lsb := count_total;
```

```
      case lsb is
        when 0 =>  lsb_out <= "0000";
        when 1 =>  lsb_out <= "0001";
        when 2 =>  lsb_out <= "0010";
        when 3 =>  lsb_out <= "0011";
        when 4 =>  lsb_out <= "0100";
        when 5 =>  lsb_out <= "0101";
        when 6 =>  lsb_out <= "0110";
        when 7 =>  lsb_out <= "0111";
        when 8 =>  lsb_out <= "1000";
        when 9 =>  lsb_out <= "1001";
        when others =>  lsb_out <= "0000";
end case;

      case msb is
        when 9 =>  msb_out <= "1001";
        when 8 =>  msb_out <= "1000";
        when 7 =>  msb_out <= "0111";
        when 6 =>  msb_out <= "0110";
        when 5 =>  msb_out <= "0101";
        when 4 =>  msb_out <= "0100";
        when 3 =>  msb_out <= "0011";
        when 2 =>  msb_out <= "0010";
        when 1 =>  msb_out <= "0001";
        when 0 =>  msb_out <= "0000";
        when others =>  msb_out <= "0000";
  end case;
```

```
            case imsb is
              when 2 =>  imsb_out <= "0010";
              when 1 =>  imsb_out <= "0001";
              when 0 =>  imsb_out <= "0000";
              when others =>  imsb_out <= "0000";
   end case;
  end process;
end Behavioral;
--------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity clk_div is
   Port (  clk : in std_logic;
          out_clk : out std_logic);
end clk_div;

architecture Behavioral of clk_div is
  constant max_count : integer := (1000);
  signal tmp_clk : std_logic := '0';
begin
  process (clk,tmp_clk)
    variable div_cnt : integer := 0;
  begin
    if (rising_edge(clk)) then
      if (div_cnt = MAX_COUNT) then
        tmp_clk <= not tmp_clk;
        div_cnt := 0;
      else
        div_cnt := div_cnt + 1;
      end if;
    end if;
    out_clk <= tmp_clk;
  end process;
end Behavioral;
```

5. Clock:

In this game, we use many different clock signals. The clock of Basys3 is not used in this game. Instead of it, our desired clock signal is created by using core generator included in the ISE Design Suite.

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

library unisim;
use unisim.vcomponents.all;

entity clk_wiz is
port
 (clk_in_1          : in     std_logic;
  clk_out_1          : out    std_logic);
end clk_wiz;
architecture Behavirol of clk_wiz is
  attribute CORE_GENERATION_INFO : string;
  attribute CORE_GENERATION_INFO of Behavirol : architecture is
"clk_wiz,clk_wiz,{component_name=clk_wiz,use_phase_alignment=true,use_min_o_jitter=false,use_ma
x_i_jitter=false,use_dyn_phase_shift=false,use_inclk_switchover=false,use_dyn_reconfig=false,feedback
_source=FDBK_AUTO,primtype_sel=PLL_BASE,num_out_clk=1,clkin1_period=10.000,clkin2_period=10.0
00,use_power_down=false,use_reset=false,use_locked=false,use_inclk_stopped=false,use_status=false,
use_freeze=false,use_clk_valid=false,feedback_type=SINGLE,clock_mgr_type=AUTO,manual_override=f
alse}";
  signal clkin1      : std_logic;
  signal clkfbout         : std_logic;
  signal clkfbout_buf     : std_logic;
  signal clkout0          : std_logic;
  signal clkout1_unused   : std_logic;
  signal clkout2_unused   : std_logic;
  signal clkout3_unused   : std_logic;
  signal clkout4_unused   : std_logic;
  signal clkout5_unused   : std_logic;
  signal locked_unused    : std_logic;

begin
  clkin1_buf : IBUFG
  port map
   (O => clkin1,
    I => clk_in_1);
  pll_base_inst : PLL_BASE
  generic map
   (BANDWIDTH            => "OPTIMIZED",
    CLK_FEEDBACK         => "CLKFBOUT",
    COMPENSATION         => "SYSTEM_SYNCHRONOUS",
    DIVCLK_DIVIDE        => 5,
    CLKFBOUT_MULT        => 34,
    CLKFBOUT_PHASE       => 0.000,
    CLKOUT0_DIVIDE       => 27,
    CLKOUT0_PHASE        => 0.000,
    CLKOUT0_DUTY_CYCLE   => 0.500,
    CLKIN_PERIOD        => 10.000,
    REF_JITTER          => 0.010)
  port map
   (CLKFBOUT           => clkfbout,
    CLKOUT0            => clkout0,
    CLKOUT1            => clkout1_unused,
    CLKOUT2            => clkout2_unused,
    CLKOUT3            => clkout3_unused,
    CLKOUT4            => clkout4_unused,
    CLKOUT5            => clkout5_unused,
    LOCKED             => locked_unused,
    RST                => '0',
    CLKFBIN            => clkfbout_buf,
    CLKIN              => clkin1);
  clkf_buf : BUFG
  port map
   (O => clkfbout_buf,
    I => clkfbout);


  clkout1_buf : BUFG
  port map
   (O   => clk_out_1,
    I   => clkout0);
end Behavirol;
```

6. VGA:

VGA Controller is the digital circuit designed to drive VGA displays. It reads from Frame Buffer which represents the frame to be displayed, and generates necessary data and sync signals for display purpose (*Design of a Simple VGA Controller in VHDL and Verilog*, n.d.). VGA controller consists of 5 signals (Horizonstal sync, Vertical sync, red, green, blue). By using main 3 color, we can generate other colors.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity vga is
port(
  rst        : in std_logic;
  pixel_clk  : in std_logic;

  HS         : out std_logic;
  VS         : out std_logic;
  hcount     : out std_logic_vector(10 downto 0);
  vcount     : out std_logic_vector(10 downto 0);
  blank      : out std_logic
);
end vga;

architecture Behavioral of vga is
constant HMAX  : std_logic_vector(10 downto 0) := "01100100000";
constant VMAX  : std_logic_vector(10 downto 0) := "01000001101";
constant HLINES: std_logic_vector(10 downto 0) := "01010000000";
constant HFP   : std_logic_vector(10 downto 0) := "01010001000";
constant HSP   : std_logic_vector(10 downto 0) := "01011101000";
constant VLINES: std_logic_vector(10 downto 0) := "00111100000";

constant VFP   : std_logic_vector(10 downto 0) := "00111100010";
constant VSP   : std_logic_vector(10 downto 0) := "00111100100";
constant SPP   : std_logic := '0';
signal hcounter : std_logic_vector(10 downto 0) := (others => '0');
signal vcounter : std_logic_vector(10 downto 0) := (others => '0');
signal video_enable: std_logic;

begin
  hcount <= hcounter;
  vcount <= vcounter;
  blank <= not video_enable when rising_edge(pixel_clk);
  h_count: process(pixel_clk)
  begin
    if(rising_edge(pixel_clk)) then
      if(rst = '1') then
        hcounter <= (others => '0');
      elsif(hcounter = HMAX) then
        hcounter <= (others => '0');
      else
        hcounter <= hcounter + 1;
      end if;
    end if;
  end process h_count;

  v_count: process(pixel_clk)
  begin
    if(rising_edge(pixel_clk)) then
      if(rst = '1') then
        vcounter <= (others => '0');
      elsif(hcounter = HMAX) then
```

```
            if(vcounter = VMAX) then
              vcounter <= (others => '0');
            else
              vcounter <= vcounter + 1;
            end if;
          end if;
        end if;
      end process v_count;
      do_hs: process(pixel_clk)
      begin
        if(rising_edge(pixel_clk)) then
          if(hcounter >= HFP and hcounter < HSP) then
            HS <= SPP;
          else
            HS <= not SPP;
          end if;
        end if;
      end process do_hs;
      do_vs: process(pixel_clk)
      begin
        if(rising_edge(pixel_clk)) then
          if(vcounter >= VFP and vcounter < VSP) then
            VS <= SPP;
          else
            VS <= not SPP;
          end if;
        end if;
      end process do_vs;
        video_enable <= '1' when (hcounter < HLINES and vcounter < VLINES) else '0';
```

```
end Behavioral;
```

7. Graphics:

In this module, we drew the player and the obstacles on the screen by converting 2 binary numbers to decimal numbers (hcount, vcount). According to decimal numbers, we can determine the coordinates and draw them.

hcount => x axis

vcount=> y axis

By using the redgreenblue signal, the obstacles and the player were assigned colors.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use ieee.std_logic_unsigned.all;

use ieee.std_logic_arith.all;

use ieee.numeric_std.all;

entity graphics is

  Port ( pixel_clk   : in STD_LOGIC;

             player_location  : in STD_LOGIC_VECTOR (2 downto 0);

      obst_locs_1,

      obst_locs_2,

      obst_locs_3 : in STD_LOGIC_VECTOR (31 downto 0);

                         dead       : in STD_LOGIC;

                         hcount      : in STD_LOGIC_VECTOR (10 downto 0);

                         vcount      : in STD_LOGIC_VECTOR (10 downto 0);

                         RedGreenBlue        : out  STD_LOGIC_VECTOR (7 downto 0));

end graphics;


architecture Behavioral of graphics is

constant distance : integer := 70;

signal x,y : integer range 0 to 800;
signal draw_obst, draw_plyr :std_logic;

signal color_selector : std_logic_vector (2 downto 0);

begin
x <= conv_integer(hcount);
y <= conv_integer(vcount);


draw_obst <= '1' when

 (((x > 690) and (x < 730) and (y > 110) and (y < 150) and (obst_locs_1(31) = '1')) or
        ((x > 670) and (x < 710) and (y > 110) and (y < 150) and (obst_locs_1(30) = '1')) or
  ((x > 650) and (x < 690) and (y > 110) and (y < 150) and (obst_locs_1(29) = '1')) or
        ((x > 630) and (x < 670) and (y > 110) and (y < 150) and (obst_locs_1(28) = '1')) or
        ((x > 610) and (x < 650) and (y > 110) and (y < 150) and (obst_locs_1(27) = '1')) or
        ((x > 590) and (x < 630) and (y > 110) and (y < 150) and (obst_locs_1(26) = '1')) or
        ((x > 570) and (x < 610) and (y > 110) and (y < 150) and (obst_locs_1(25) = '1')) or
  ((x > 550) and (x < 590) and (y > 110) and (y < 150) and (obst_locs_1(24) = '1')) or
  ((x > 530) and (x < 570) and (y > 110) and (y < 150) and (obst_locs_1(23) = '1')) or
        ((x > 510) and (x < 550) and (y > 110) and (y < 150) and (obst_locs_1(22) = '1')) or
        ((x > 490) and (x < 530) and (y > 110) and (y < 150) and (obst_locs_1(21) = '1')) or
        ((x > 470) and (x < 510) and (y > 110) and (y < 150) and (obst_locs_1(20) = '1')) or
        ((x > 450) and (x < 490) and (y > 110) and (y < 150) and (obst_locs_1(19) = '1')) or
        ((x > 430) and (x < 470) and (y > 110) and (y < 150) and (obst_locs_1(18) = '1')) or
        ((x > 410) and (x < 450) and (y > 110) and (y < 150) and (obst_locs_1(17) = '1')) or
        ((x > 390) and (x < 430) and (y > 110) and (y < 150) and (obst_locs_1(16) = '1')) or
        ((x > 370) and (x < 410) and (y > 110) and (y < 150) and (obst_locs_1(15) = '1')) or
        ((x > 350) and (x < 390) and (y > 110) and (y < 150) and (obst_locs_1(14) = '1')) or
        ((x > 330) and (x < 370) and (y > 110) and (y < 150) and (obst_locs_1(13) = '1')) or
        ((x > 310) and (x < 350) and (y > 110) and (y < 150) and (obst_locs_1(12) = '1')) or
        ((x > 290) and (x < 330) and (y > 110) and (y < 150) and (obst_locs_1(11) = '1')) or
        ((x > 270) and (x < 310) and (y > 110) and (y < 150) and (obst_locs_1(10) = '1')) or
        ((x > 250) and (x < 290) and (y > 110) and (y < 150) and (obst_locs_1(9) = '1')) or
        ((x > 230) and (x < 270) and (y > 110) and (y < 150) and (obst_locs_1(8) = '1')) or
        ((x > 210) and (x < 250) and (y > 110) and (y < 150) and (obst_locs_1(7) = '1')) or
  ((x > 190) and (x < 230) and (y > 110) and (y < 150) and (obst_locs_1(6) = '1')) or
        ((x > 170) and (x < 210) and (y > 110) and (y < 150) and (obst_locs_1(5) = '1')) or
        ((x > 150) and (x < 190) and (y > 110) and (y < 150) and (obst_locs_1(4) = '1')) or
        ((x > 130) and (x < 170) and (y > 110) and (y < 150) and (obst_locs_1(3) = '1')) or
        ((x > 110) and (x < 150) and (y > 110) and (y < 150) and (obst_locs_1(2) = '1')) or
        ((x > 90) and (x < 130) and (y > 110) and (y < 150) and (obst_locs_1(1) = '1')) or
        ((x > 70) and (x < 110) and (y > 110) and (y < 150) and (obst_locs_1(0) = '1')) or


 ((x > 690) and (x < 730) and (y > 220) and (y < 260) and (obst_locs_2(31) = '1')) or
        ((x > 670) and (x < 710) and (y > 220) and (y < 260) and (obst_locs_2(30) = '1')) or
        ((x > 650) and (x < 690) and (y > 220) and (y < 260) and (obst_locs_2(29) = '1')) or
        ((x > 630) and (x < 670) and (y > 220) and (y < 260) and (obst_locs_2(28) = '1')) or
        ((x > 610) and (x < 650) and (y > 220) and (y < 260) and (obst_locs_2(27) = '1')) or
        ((x > 590) and (x < 630) and (y > 220) and (y < 260) and (obst_locs_2(26) = '1')) or
        ((x > 570) and (x < 610) and (y > 220) and (y < 260) and (obst_locs_2(25) = '1')) or
        ((x > 550) and (x < 590) and (y > 220) and (y < 260) and (obst_locs_2(24) = '1')) or
        ((x > 530) and (x < 570) and (y > 220) and (y < 260) and (obst_locs_2(23) = '1')) or
        ((x > 510) and (x < 550) and (y > 220) and (y < 260) and (obst_locs_2(22) = '1')) or
        ((x > 490) and (x < 530) and (y > 220) and (y < 260) and (obst_locs_2(21) = '1')) or
        ((x > 470) and (x < 510) and (y > 220) and (y < 260) and (obst_locs_2(20) = '1')) or
        ((x > 450) and (x < 490) and (y > 220) and (y < 260) and (obst_locs_2(19) = '1')) or
        ((x > 430) and (x < 470) and (y > 220) and (y < 260) and (obst_locs_2(18) = '1')) or
        ((x > 410) and (x < 450) and (y > 220) and (y < 260) and (obst_locs_2(17) = '1')) or
        ((x > 390) and (x < 430) and (y > 220) and (y < 260) and (obst_locs_2(16) = '1')) or
        ((x > 370) and (x < 410) and (y > 220) and (y < 260) and (obst_locs_2(15) = '1')) or
        ((x > 350) and (x < 390) and (y > 220) and (y < 260) and (obst_locs_2(14) = '1')) or
        ((x > 330) and (x < 370) and (y > 220) and (y < 260) and (obst_locs_2(13) = '1')) or
        ((x > 310) and (x < 350) and (y > 220) and (y < 260) and (obst_locs_2(12) = '1')) or
        ((x > 290) and (x < 330) and (y > 220) and (y < 260) and (obst_locs_2(11) = '1')) or
        ((x > 270) and (x < 310) and (y > 220) and (y < 260) and (obst_locs_2(10) = '1')) or
        ((x > 250) and (x < 290) and (y > 220) and (y < 260) and (obst_locs_2(9) = '1')) or
        ((x > 230) and (x < 270) and (y > 220) and (y < 260) and (obst_locs_2(8) = '1')) or
        ((x > 210) and (x < 250) and (y > 220) and (y < 260) and (obst_locs_2(7) = '1')) or
        ((x > 190) and (x < 230) and (y > 220) and (y < 260) and (obst_locs_2(6) = '1')) or
        ((x > 170) and (x < 210) and (y > 220) and (y < 260) and (obst_locs_2(5) = '1')) or
        ((x > 150) and (x < 190) and (y > 220) and (y < 260) and (obst_locs_2(4) = '1')) or
        ((x > 130) and (x < 170) and (y > 220) and (y < 260) and (obst_locs_2(3) = '1')) or
        ((x > 110) and (x < 150) and (y > 220) and (y < 260) and (obst_locs_2(2) = '1')) or
        ((x > 90) and (x < 130) and (y > 220) and (y < 260) and (obst_locs_2(1) = '1')) or
        ((x > 70) and (x < 110) and (y > 220) and (y < 260) and (obst_locs_2(0) = '1')) or

((x > 690) and (x < 730) and (y > 330) and (y < 370) and (obst_locs_3(31) = '1')) or
        ((x > 670) and (x < 710) and (y > 330) and (y < 370) and (obst_locs_3(30) = '1')) or
        ((x > 650) and (x < 690) and (y > 330) and (y < 370) and (obst_locs_3(29) = '1')) or
        ((x > 630) and (x < 670) and (y > 330) and (y < 370) and (obst_locs_3(28) = '1')) or
        ((x > 610) and (x < 650) and (y > 330) and (y < 370) and (obst_locs_3(27) = '1')) or
        ((x > 590) and (x < 630) and (y > 330) and (y < 370) and (obst_locs_3(26) = '1')) or
        ((x > 570) and (x < 610) and (y > 330) and (y < 370) and (obst_locs_3(25) = '1')) or
        ((x > 550) and (x < 590) and (y > 330) and (y < 370) and (obst_locs_3(24) = '1')) or
        ((x > 530) and (x < 570) and (y > 330) and (y < 370) and (obst_locs_3(23) = '1')) or
```

```
            ((x > 510) and (x < 550) and (y > 330) and (y < 370) and (obst_locs_3(22) = '1')) or
            ((x > 490) and (x < 530) and (y > 330) and (y < 370) and (obst_locs_3(21) = '1')) or
            ((x > 470) and (x < 510) and (y > 330) and (y < 370) and (obst_locs_3(20) = '1')) or
            ((x > 450) and (x < 490) and (y > 330) and (y < 370) and (obst_locs_3(19) = '1')) or
            ((x > 430) and (x < 470) and (y > 330) and (y < 370) and (obst_locs_3(18) = '1')) or
            ((x > 410) and (x < 450) and (y > 330) and (y < 370) and (obst_locs_3(17) = '1')) or
            ((x > 390) and (x < 430) and (y > 330) and (y < 370) and (obst_locs_3(16) = '1')) or
            ((x > 370) and (x < 410) and (y > 330) and (y < 370) and (obst_locs_3(15) = '1')) or
            ((x > 350) and (x < 390) and (y > 330) and (y < 370) and (obst_locs_3(14) = '1')) or
            ((x > 330) and (x < 370) and (y > 330) and (y < 370) and (obst_locs_3(13) = '1')) or
            ((x > 310) and (x < 350) and (y > 330) and (y < 370) and (obst_locs_3(12) = '1')) or
            ((x > 290) and (x < 330) and (y > 330) and (y < 370) and (obst_locs_3(11) = '1')) or
            ((x > 270) and (x < 310) and (y > 330) and (y < 370) and (obst_locs_3(10) = '1')) or
            ((x > 250) and (x < 290) and (y > 330) and (y < 370) and (obst_locs_3(9) = '1')) or
            ((x > 230) and (x < 270) and (y > 330) and (y < 370) and (obst_locs_3(8) = '1')) or
            ((x > 210) and (x < 250) and (y > 330) and (y < 370) and (obst_locs_3(7) = '1')) or
            ((x > 190) and (x < 230) and (y > 330) and (y < 370) and (obst_locs_3(6) = '1')) or
            ((x > 170) and (x < 210) and (y > 330) and (y < 370) and (obst_locs_3(5) = '1')) or
            ((x > 150) and (x < 190) and (y > 330) and (y < 370) and (obst_locs_3(4) = '1')) or
            ((x > 130) and (x < 170) and (y > 330) and (y < 370) and (obst_locs_3(3) = '1')) or
            ((x > 110) and (x < 150) and (y > 330) and (y < 370) and (obst_locs_3(2) = '1')) or
            ((x > 90) and (x < 130) and (y > 330) and (y < 370) and (obst_locs_3(1) = '1')) or
            ((x > 70) and (x < 110) and (y > 330) and (y < 370) and (obst_locs_3(0) = '1')))
else '0';


draw_plyr <= '1' when

  (((x > 70) and (x < 110) and (y > 110) and (y < 150) and (player_location = "100")) or
            ((x > 70) and (x < 110) and (y > 220) and (y < 260) and (player_location = "010")) or
            ((x > 70) and (x < 110) and (y > 330) and (y < 370) and (player_location = "001"))) else '0';


color_selector <= draw_obst & draw_plyr & dead;
with color_selector select
  RedGreenBlue <= "11100000" when "100",
  "00000011" when "110",
            "00000011" when "010",
            "11111111" when "101",
            "11111111" when "111",
            "11111111" when "011",
            "00000000" when others;
end Behavioral;
```

8. Game Logic:

Game is created inside of this module. Game_logic module determines the obstacles and player locations, score, number of right of lives by getting inputs from player. The clock can be arrange almost 8Hz because of the optimal sensitivity. The most challenging part of this module is hit detection. The algorithm of the hit detection is that player remains stationary and if the less significant bit of the lane is 1, player loses one of the his/her right of lives.

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


entity game_logic is
  port ( Clk : in std_logic;
                         player_input : in std_logic_vector (2 downto 0);
                         player_location   : out std_logic_vector (2 downto 0);
                         obst_locs_1  : out std_logic_vector (31 downto 0);
```

```
                                obst_locs_2  : out std_logic_vector (31 downto 0);
                                obst_locs_3  : out std_logic_vector (31 downto 0);
           score  : out std_logic_vector (7 downto 0);
                         remaining_life : out std_logic_vector (1 downto 0);
                         dead_state : out std_logic);

end game_logic;

architecture Behavioral of game_logic is
constant clk_frequency : integer := 25270000;
constant game_frequency : integer := 8;
component shift_register
   port ( game_clock : in std_logic;
                   obst_in : in std_logic;
                                        dead : in std_logic;
                                       reset : in std_logic;
                          path : out std_logic_vector (31 downto 0));
end component;
component Linear_Feedback_shift_reg
   port (game_clock : in std_logic;
                    dead : in std_logic;
                   reset : in std_logic;
                 obst_out : out std_logic);
end component;

component select_lane
   port (game_clock : in  STD_LOGIC;
                  obst_in : in  STD_LOGIC;
```

```
    obstacle_1 : out  STD_LOGIC;
    obstacle_2 : out  STD_LOGIC;
    obstacle_3 : out  STD_LOGIC);
end component;
signal enable_game_update : std_logic := '0';
signal lose_state : std_logic := '0';
signal current_player_location : std_logic_vector (2 downto 0) := "010";
signal path1, path2, path3 :  std_logic_vector (31 downto 0);
signal current_obst : std_logic;
signal current_obstacle_1,current_obstacle_2,current_obstacle_3: std_logic;
signal lives_left : std_logic_vector (1 downto 0) := "11";
signal global_reset : std_logic;
signal current_score : std_logic_vector (7 downto 0) := "00000000";
begin

global_reset <= player_input(0);

lane1_sr : shift_register
  port map ( game_clock => enable_game_update,
                   obst_in => current_obstacle_1,
                                        dead => lose_state,
                              reset => global_reset,
                                      path => path1);

lane2_sr : shift_register
  port map ( game_clock => enable_game_update,
                   obst_in => current_obstacle_2,
                                        dead => lose_state,
                                        reset => global_reset,
                                           path => path2);

lane3_sr : shift_register
   port map ( game_clock => enable_game_update,
                        obst_in => current_obstacle_3,
                                             dead => lose_state,
                                        reset => global_reset,
                                           path => path3);

obst_locs_1 <= path1;
obst_locs_2 <= path2;
obst_locs_3 <= path3;

obst_generator : Linear_Feedback_shift_reg
   port map ( game_clock => enable_game_update,
                          dead => lose_state,
                         reset => global_reset,
                                           obst_out => current_obst);

lane_selector : select_lane
   port map ( game_clock => enable_game_update,
                        obst_in => current_obst,
        obstacle_1 => current_obstacle_1,
        obstacle_2 => current_obstacle_2,
        obstacle_3 => current_obstacle_3);

   process(Clk)
```

```
            variable count : integer := 0;
          begin
            if (rising_edge(Clk)) then
                    enable_game_update <= '0';
                          count := (count + 1);
              if (count = (clk_frequency / game_frequency)) then
                            enable_game_update <= '1';
                                  count := 0;
                          end if;
                  end if;
          end process;

    process(enable_game_update, player_input)
    begin
           if (rising_edge(enable_game_update)) then
       if ((current_player_location = "100") and (player_input = "010")) then
                      current_player_location <= "010";
                  elsif (current_player_location = "010") then
                      if (player_input = "100") then
                              current_player_location <= "100";
                      elsif (player_input = "010") then
                              current_player_location <= "001";
                           end if;
                  elsif (current_player_location = "001") then
                      if (player_input = "100") then
                              current_player_location <= "010";
                            end if;
                    end if;
```

```
                player_location <= current_player_location;
                end if;
        end process player_movement;

        process(enable_game_update)
        begin
          if (rising_edge(enable_game_update)) then
            if (global_reset = '1') then
                    lives_left <= "11";
            elsif ((current_player_location = "100" and (path1(0) = '1')) or
                     (current_player_location = "010" and (path2(0) = '1')) or
                           (current_player_location = "001" and (path3(0) = '1'))) then
                    lives_left <= (lives_left - 1);
                  end if;
          end if;
        end process;

  process(enable_game_update)
        variable count : integer := 0;
        begin
          if (rising_edge(enable_game_update)) then
                  if (global_reset = '1') then
                          current_score <= "00000000";
                  elsif (lives_left /= "00") then
                          count := count + 1;
                                if (count = 8) then
                                  current_score <= current_score + 1;
                                        count := 0;
                    end if;
                end if;
            end if;
        end process;



        score <= current_score;

   process (enable_game_update)
        begin
          if (rising_edge(enable_game_update)) then
            if (global_reset = '1') then
                  lose_state <= '0';
                elsif (lives_left = "00") then
                  lose_state <= '1';
                elsif (current_score <= "01100100") then
                  lose_state <= '0';
       end if;
            end if;
        end process;

remaining_life <= lives_left;
dead_state <= lose_state;


end Behavioral;
```

9. Main Module:

All modules are combined by main module. Main module takes 2 inputs (player input, clock). Main is not that different from the game_logic module. The most explicit difference is that remaining life rights are shown on the LEDs by using multiplexer.

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

entity main_module is
port ( Clk  : in std_logic;
       player_input : in std_logic_vector (2 downto 0);
             RedGreenBlue  : out std_logic_vector (7 downto 0);
             HS : out std_logic;
             VS   : out std_logic;
             LD7, LD6, LD5  : out std_logic;
       Segs : out  std_logic_vector (6 downto 0);
             AN   : out  std_logic_vector (3 downto 0));
end main_module;

architecture Behavioral of main_module is

component game_logic is
  port ( Clk         : in std_logic;
                player_input : in std_logic_vector (2 downto 0);
                                player_location   : out std_logic_vector (2 downto 0);
                                obst_locs_1  : out std_logic_vector (31 downto 0);
                                obst_locs_2  : out std_logic_vector (31 downto 0);
                                obst_locs_3  : out std_logic_vector (31 downto 0);
                                 score      : out std_logic_vector (7 downto 0);
                         remaining_life : out std_logic_vector (1 downto 0);
                         dead_state : out std_logic);

end component;
```

```
component vga is
  port ( rst        : in std_logic;
        pixel_clk   : in std_logic;
        HS          : out std_logic;
        VS          : out std_logic;
        hcount      : out std_logic_vector(10 downto 0);
        vcount      : out std_logic_vector(10 downto 0);
        blank       : out std_logic);
end component;

component clk_wiz is
  port (clk_in_1          : in     std_logic;
       clk_out_1          : out    std_logic);
end component;

component graphics is
  port (pixel_clk    : in std_logic;
              player_location   : in STD_LOGIC_VECTOR (2 downto 0);
       obst_locs_1,
       obst_locs_2,
       obst_locs_3 : in STD_LOGIC_VECTOR (31 downto 0);
                            dead        : in std_logic;
               hcount    : in std_logic_vector (10 downto 0);
                             vcount    : in std_logic_vector (10 downto 0);
          RedGreenBlue    : out std_logic_vector (7 downto 0));
end component;

component decoder is
  port (   alu_value : in std_logic_vector(7 downto 0);
                                                      sign : in std_logic;
                                               valid : in std_logic;
               clk : in std_logic;
           anode : out std_logic_vector(3 downto 0);
          sev_seg : out std_logic_vector(6 downto 0));
end component;


signal pixel_clk : std_logic;
signal reset : std_logic := '0';
signal score : std_logic_vector (7 downto 0);
signal remaining_life : std_logic_vector (1 downto 0);
signal dead_state : std_logic := '0';
signal player_location : std_logic_vector (2 downto 0) := "010";
signal obstacles_to_draw_1, obstacles_to_draw_2, obstacles_to_draw_3 : std_logic_vector (31 downto
0);
signal horizontal_count, vertical_count : std_logic_vector (10 downto 0);
signal blank : std_logic := '0';
signal sign : std_logic := '0';
signal valid : std_logic := '1';
begin
 pixel_clock : clk_wiz
 port map ( clk_in_1  => Clk,
           clk_out_1 => pixel_clk);



 game_mechanics : game_logic
 port map ( Clk => pixel_clk,
           player_input => player_input,
           player_location   => player_location,
                                        obst_locs_1  => obstacles_to_draw_1,
                                        obst_locs_2  => obstacles_to_draw_2,
                                        obst_locs_3  => obstacles_to_draw_3,
                                            score   => score,
                                remaining_life  => remaining_life,
                                dead_state  => dead_state);

sync_module : vga
port map ( rst => reset,
        pixel_clk => pixel_clk,
                                        HS  => HS,
                                        VS  => VS,
                                        hcount => horizontal_count,
                                        vcount => vertical_count,
                                        blank => blank);

draw_the_boxes : graphics
port map ( pixel_clk => pixel_clk,
        player_location  => player_location,
        obst_locs_1 => obstacles_to_draw_1,
                                        obst_locs_2 => obstacles_to_draw_2,
                                        obst_locs_3 => obstacles_to_draw_3,
                                        dead        => dead_state,
                                        hcount      => horizontal_count,
                                        vcount      => vertical_count,
                                        RedGreenBlue        => RedGreenBlue);

score_disp : decoder
port map (     alu_value => score,
                                                    sign => sign,
                                                         valid => valid,
                        clk => pixel_clk,
                      anode => AN,
                     sev_seg => Segs);
process (remaining_life)
begin
  if (remaining_life = "11") then
    LD7 <= '1';
    LD6 <= '1';
    LD5 <= '1';
        elsif (remaining_life = "10") then
          LD7 <= '1';
    LD6 <= '1';
    LD5 <= '0';
        elsif (remaining_life = "01") then
          LD7 <= '1';
    LD6 <= '0';
    LD5 <= '0';
        else
          LD7 <= '0';
    LD6 <= '0';
    LD5 <= '0';
        end if;
end process;
end Behavioral;
```

Outputs:

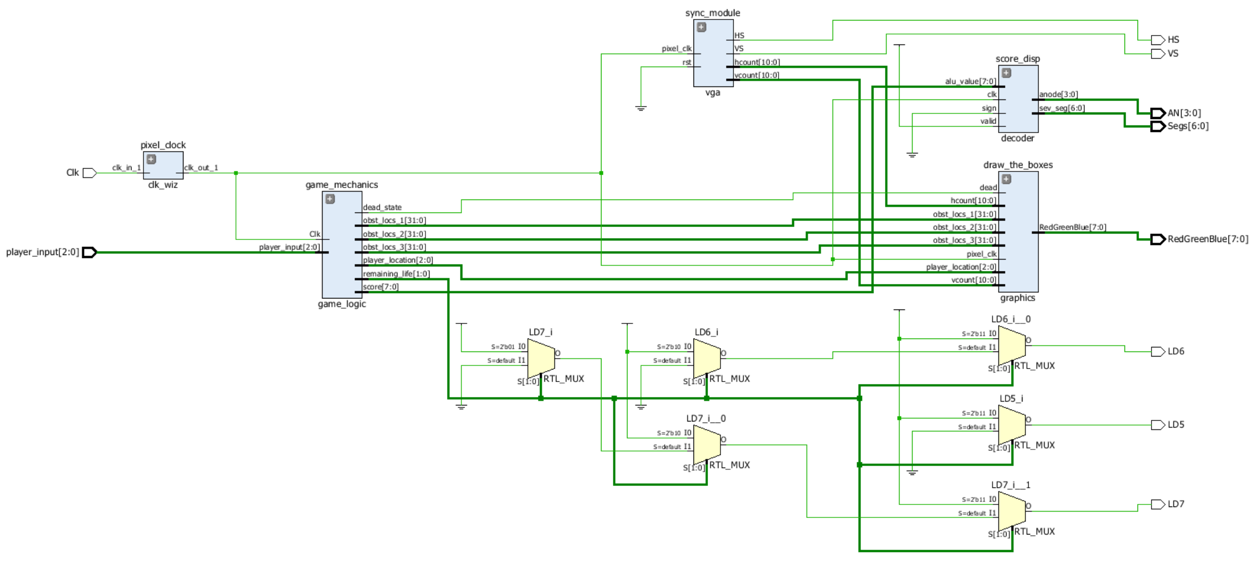

Figure 2: Schematic of the game

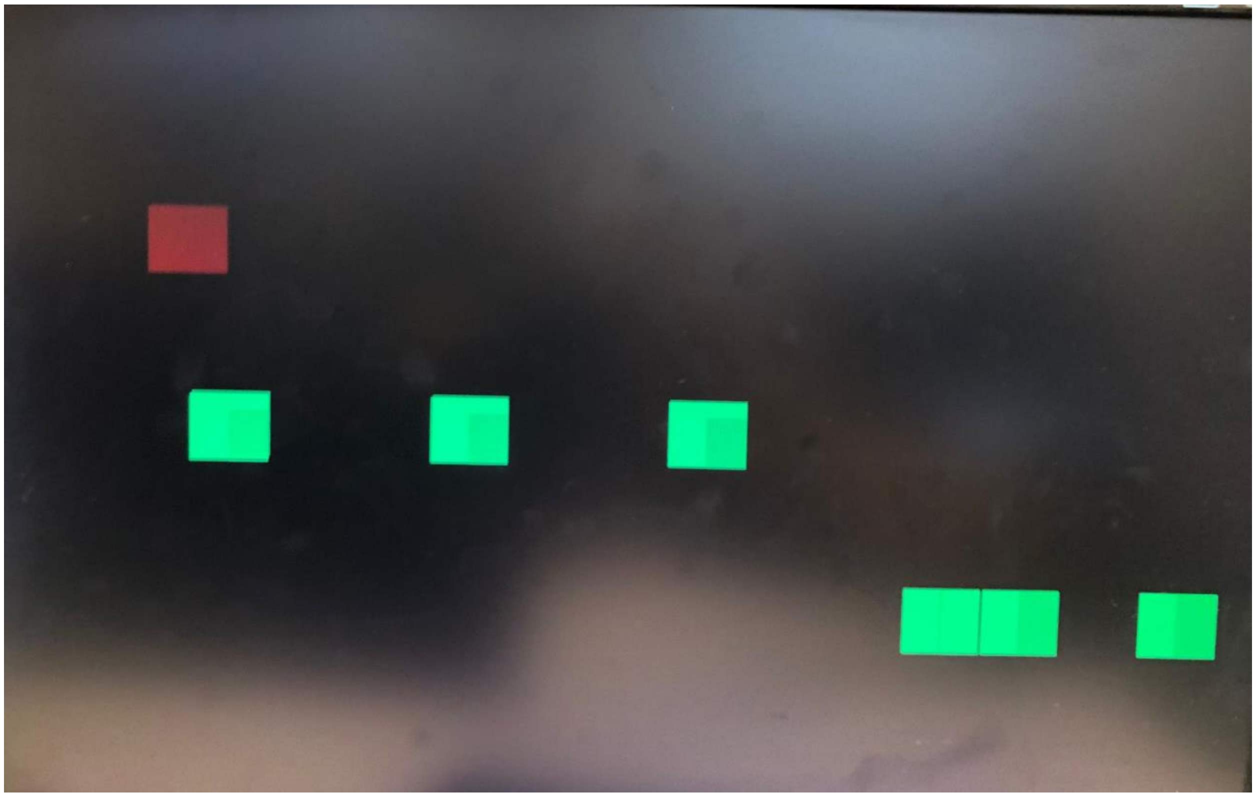

Figure 3: an example of the beginning of the game

Conclusion:

I have completed my term project that I mentioned in the proposal. The project has successfully achieved its purpose of entertaining the player. I think the most challenging part of this project is to generate pseudo-random obstacles. I faced many problems in this part. I used linear feedback shift register to generate pseudo-random obstacles and solved my problems.

References:

- *Tutorial: Linear Feedback Shift Registers*. (n.d.). Eetimes.

  https://www.eetimes.com/tutorial-linear-feedback-shift-registers-lfsrs-part-1/

- *Design of a Simple VGA Controller in VHDL and Verilog*. (n.d.). Instructables.

  https://www.instructables.com/Design-of-a-Simple-VGA-Controller-in-VHDL/

Appendix:

```
# Clock signal

set_property PACKAGE_PIN W5 [get_ports Clk]

set_property IOSTANDARD LVCMOS33 [get_ports Clk]

# LEDs

set_property PACKAGE_PIN N3 [get_ports {LD7}]

set_property IOSTANDARD LVCMOS33 [get_ports {LD7}]

set_property PACKAGE_PIN P1 [get_ports {LD6}]

set_property IOSTANDARD LVCMOS33 [get_ports {LD6}]

set_property PACKAGE_PIN L1 [get_ports {LD5}]

set_property IOSTANDARD LVCMOS33 [get_ports {LD5}]

#7 segment display

set_property PACKAGE_PIN W7 [get_ports {Segs[6]}]

set_property IOSTANDARD LVCMOS33 [get_ports {Segs[6]}]

set_property PACKAGE_PIN W6 [get_ports {Segs[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Segs[5]}]

set_property PACKAGE_PIN U8 [get_ports {Segs[4]}]

set_property IOSTANDARD LVCMOS33 [get_ports {Segs[4]}]

set_property PACKAGE_PIN V8 [get_ports {Segs[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {Segs[3]}]

set_property PACKAGE_PIN U5 [get_ports {Segs[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {Segs[2]}]

set_property PACKAGE_PIN V5 [get_ports {Segs[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {Segs[1]}]

set_property PACKAGE_PIN U7 [get_ports {Segs[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {Segs[0]}]

set_property PACKAGE_PIN U2 [get_ports {AN[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]

set_property PACKAGE_PIN U4 [get_ports {AN[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]

set_property PACKAGE_PIN V4 [get_ports {AN[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]

set_property PACKAGE_PIN W4 [get_ports {AN[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]

#Buttons

set_property PACKAGE_PIN W19 [get_ports player_input[0]]

set_property IOSTANDARD LVCMOS33 [get_ports player_input[0]]

set_property PACKAGE_PIN U17 [get_ports player_input[1]]

set_property IOSTANDARD LVCMOS33 [get_ports player_input[1]]

set_property PACKAGE_PIN T18 [get_ports player_input[2]]

set_property IOSTANDARD LVCMOS33 [get_ports player_input[2]]

#VGA Connector

set_property PACKAGE_PIN G19 [get_ports {RedGreenBlue[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {RedGreenBlue[0]}]
set_property PACKAGE_PIN H19 [get_ports {RedGreenBlue[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {RedGreenBlue[1]}]

set_property PACKAGE_PIN J19 [get_ports {RedGreenBlue[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {RedGreenBlue[2]}]

set_property PACKAGE_PIN N18 [get_ports {RedGreenBlue[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {RedGreenBlue[3]}]

set_property PACKAGE_PIN L18 [get_ports {RedGreenBlue[4]}]

set_property IOSTANDARD LVCMOS33 [get_ports {RedGreenBlue[4]}]

set_property PACKAGE_PIN J17 [get_ports {RedGreenBlue[5]}]

set_property IOSTANDARD LVCMOS33 [get_ports {RedGreenBlue[5]}]

set_property PACKAGE_PIN H17 [get_ports {RedGreenBlue[6]}]

set_property IOSTANDARD LVCMOS33 [get_ports {RedGreenBlue[6]}]

set_property PACKAGE_PIN G17 [get_ports {RedGreenBlue[7]}]

set_property IOSTANDARD LVCMOS33 [get_ports {RedGreenBlue[7]}]

set_property PACKAGE_PIN P19 [get_ports HS]

set_property IOSTANDARD LVCMOS33 [get_ports HS]

set_property PACKAGE_PIN R19 [get_ports VS]

set_property IOSTANDARD LVCMOS33 [get_ports VS]
```