Ege Ozan Özyedek
Project Report
EEE102 – 02

- YouTube Video Link

  You can find the video presentation at: https://youtu.be/eYcAsJIgDBs

- Abstract

  The goal of this project was to create a game using the VHDL coding language and to utilize the BASYS 3 board and its VGA connection. This goal was achieved by creating a game in which the player can move, interact with the enemies by using its ability to shoot a projectile and can pass to the next level by obtaining a hexadecimal code on the 7-segment display and then entering it via the switches on the BASYS 3 board. This report will explain the idea and technicalities of the project in detail and will give examples of the results.

- The Design Specification Plan

  The design of the project starts with the explanation of the game. The game has a simple idea: the player tries to pass to the next level by defeating the enemy and therefore obtaining the treasure from it. This treasure holds the code that will open the gate at the end of the level. After obtaining the treasure, the player can observe the secret code on the 7-segment display, in hexadecimal form. To proceed to the next level, the player has to enter this 4-bit hex code in 16-bit binary form via the 16 switches. The player then proceeds to the next level upon entering the correct code.

  Onto some non-technical mechanics. The player movement is possible with the four directional buttons which exist on the BASYS 3. The action button, which enables the player to pick up the treasure or to throw a projectile, is possible by pressing the middle button. The enemies move in relation to the players position, it comes closer to the player as the player enters the specified range of the enemy. The player has to escape or defeat the enemy since if the enemy comes close enough, the players position is reset and the game restarts. The projectile moves faster than both the enemy and the player, and is smaller. When the enemy is in the specified hostility range of the projectile, the enemy is defeated and drops the treasure behind. Upon the player picking up the treasure, the code is shown on the 7-segment display. This code is random (as random as it gets, this will be explained further on the report) and enables the game to have an unpredictability to it.

- The Design Methodology

  - BASYS Implementation

    The project uses most elements of the board, both as inputs and outputs. The 5 buttons are used, 4 of the directional ones are inputs for the movement of the character while the fifth one is used as the action button and determines whether something is being picked up or a projectile needs to be thrown. The 7-segment display is used to display the random hexadecimal code and

displays 0000 on default. The switches under the display is used as the input code that the player will enter and the corresponding LED's light up when the switches are inputted as 1. The 100MHz clock is used as the pixel clock for the display synchronization and also in many other modules as a mechanic

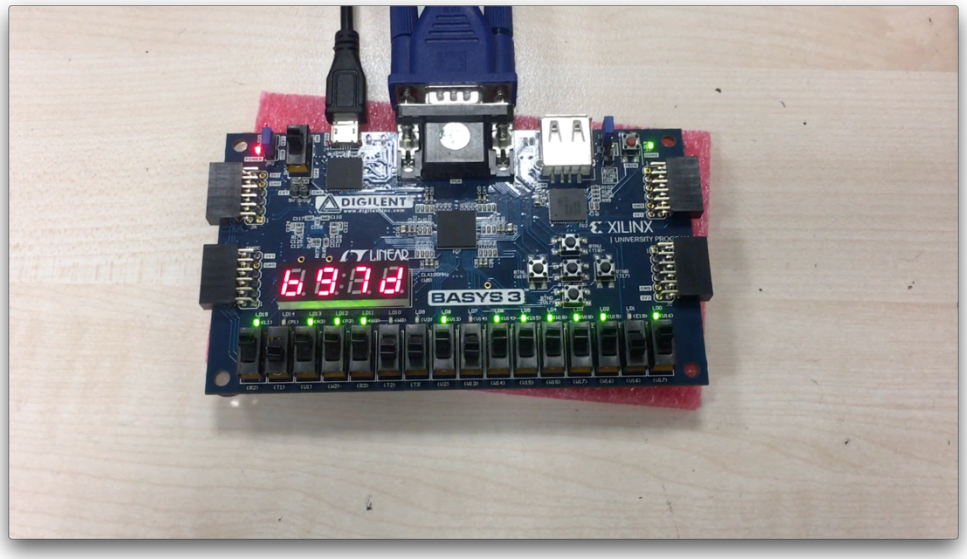

Figure 1: BASYS 3 board, showing the randomly generated hex code on the display

refresher. For example, a 100Hz clock is used for the refreshment of the characters movement

- VHDL Modules

  This section will go further into the details and the mechanics of the game by explaining the VHDL code and the modules.

  Top Module (main): The top module contains many components inside it, but there exist no operations. These components are mostly for the connection

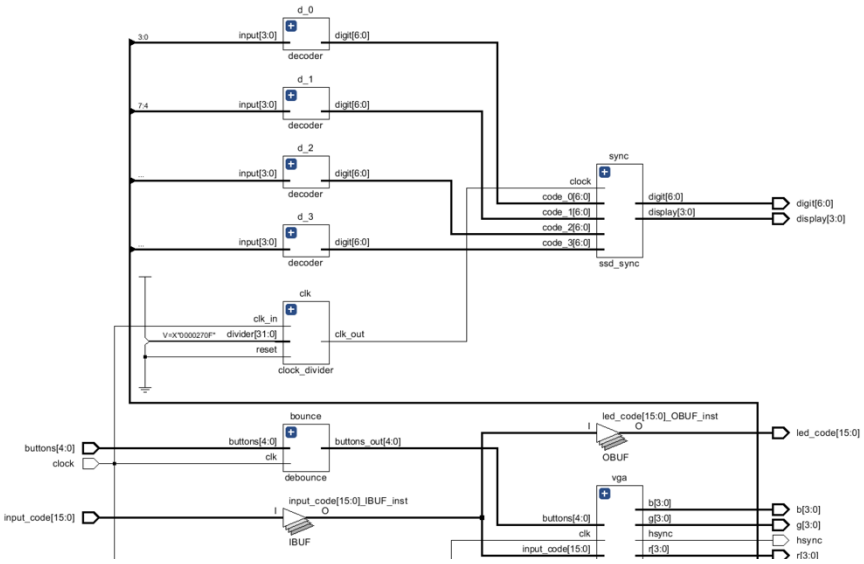

Figure 2: The RTL Schematic of the top module

between the FPGA and the code, and these components being on the top module makes it clearer which outputs are related with the board.

debouncer: The debouncer, just as the name suggests, makes the button inputs more reliable by passing the inputs to two other variables and then putting them all into an AND gate. This is needed since the buttons on BASYS 3 can oscillate and create unwanted inputs at times, which has to be prevented. Outputs the correctly inputted buttons. I was able to write this

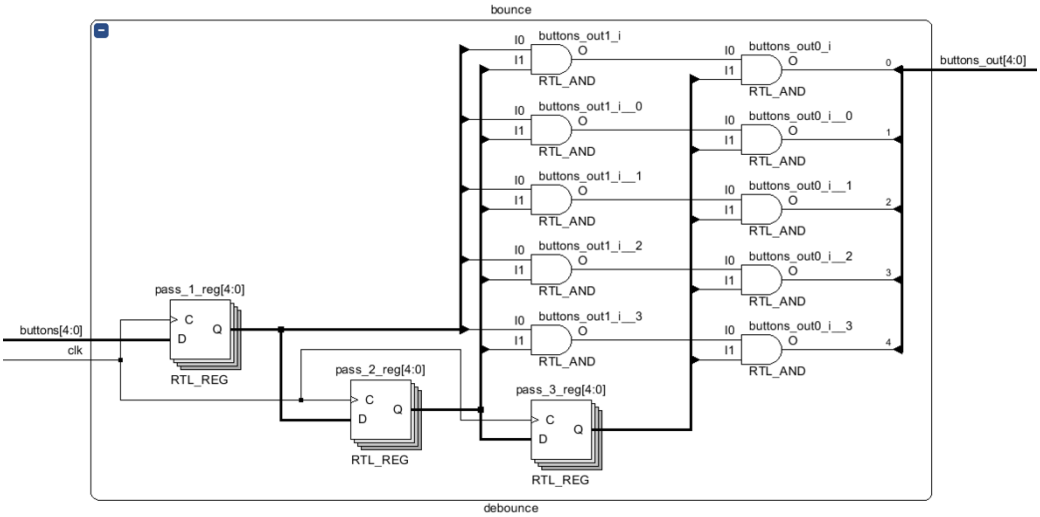

Figure 3: The RTL schematic of the debouncer

with the help of another code I found on the web which implemented the debouncer for a single button, I was able to turn it into a module which outputted all 4 buttons debounced.[1]

vga_main: This can be viewed as the top module for the vga operations. It contains two components, level_drawer and vga_sync which will be explained in the two sections.

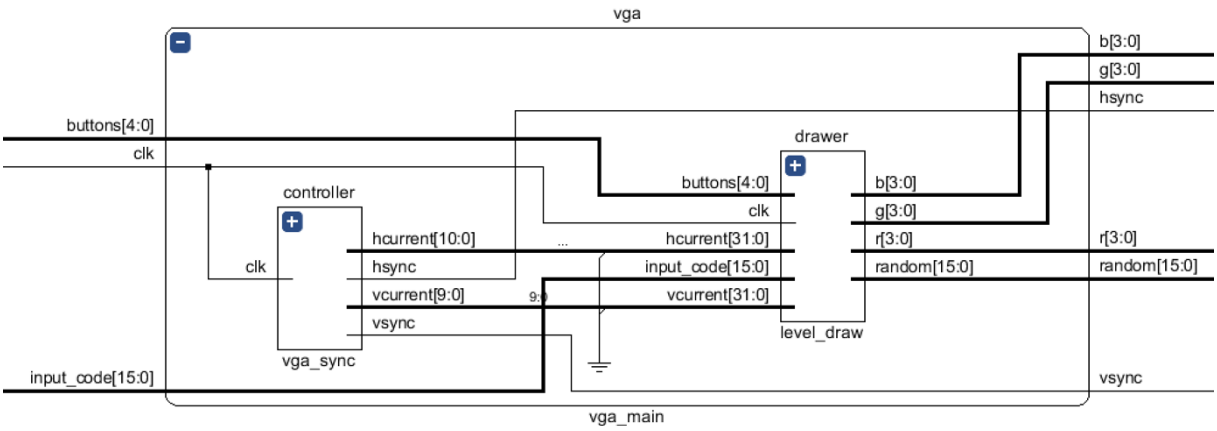

Figure 4: The RTL schematic of vga_main

[1] "VHDL Debouncer Circuit," Electrical Engineering Stack Exchange, , accessed December 29, 2018, https://electronics.stackexchange.com/questions/32260/vhdl-debouncer-circuit.

vga_sync: This module exists so that the timings of the vga connections are done correctly. The VGA connection requires specific timings to make the visualization possible. These timings are the front and back porch, sync pulse and the display time. The only visible timing from these is the display time. This module makes it possible so that the code only outputs colors at the display time and not on the other three. The other three exist for several reasons. The sync pulse exists so that the screen refreshes at the specified time. The front and back porches are mostly signal clearers, in other words they are there so that the signals from one refresh cycle to the next don't intertwine.

rng (random number generator): This module is intended to create random 16-bit binary numbers. At every rising edge of the 100 MHz clock, the code gets the last 3 bits of the 16-bit signal and inputs them to XOR's. The output of these XOR are then put into the start of the 16-bit number and it continues to create a number every rising edge. It is not specifically random, since the numbers are always the same at their corresponding clock cycle. But they become random when an unpredictable input comes into play. In our case, this is the time when the player picks up the treasure from the enemy. Since it is impossible to know when the player will pick up the code, and since the code does this operation 100 million times at every second it becomes pretty random. I was able to write a different rendition (16 bits) of a random number generator that I found while researching.[2]

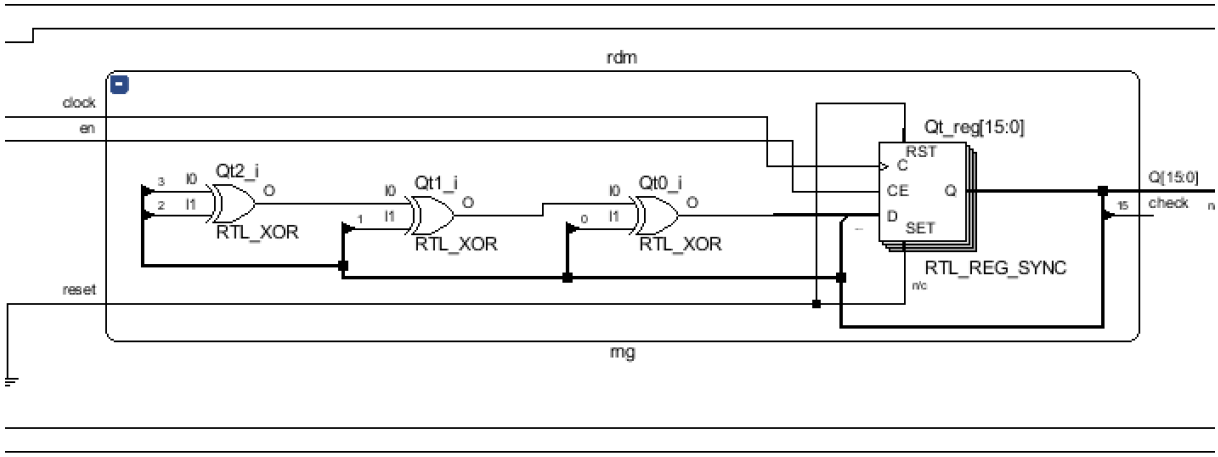

Figure 5: The RTL Schematic of the random number generator

ssd_sync: This module creates the synchronization between the digit that will be displayed, and which display will light up with that specific digit.

[2] "Pseudo Random Number Generator Using LFSR in VHDL," Stack Overflow, , accessed December 29, 2018, https://stackoverflow.com/questions/43081067/pseudo-random-number-generator-using-lfsr-in-vhdl.

decoder: The decoder outputs the hexadecimal version of a 4-bit binary number. This enables to code to output the digit and show it on the 7-segment display.

clock_divider: The clock divider makes it so that the 100MHz clock can be used in different frequencies for different purposes. This module enables me to separate the pixel clock and the movement clock, for example, which is 100Hz.

packages: This module contains a variety of packages, which contain different procedures and array types. I use a procedure called rectangle to draw a specific rectangle at a specified x and y position and height and width. This enables me to draw the level however I like.

player: This is the module where the x and y position of the player gets changed. These positions are changed via the button inputs and gets incremented 4 pixels every time the clock is on rising edge and a button input exists. Another condition which is very important is something called reachable, which determines whether the pixels around the player are reachable. If they are reachable, meaning they have the same rgb as the play area, then the condition is 1 and the character can move to that point. If it is not reachable, meaning it's not the same rgb as the play area, then the player does not move. This is implemented since we do not want the player to go out of bounds and stay on the play area. The player also outputs something called direction, which is a 4-bit vector. This determines the direction in which the player is currently moving and enables the projectile fired to move to that direction.

enemy: The enemies behave similar to the player; however, are very hostile. They try to get close to the player so that they can defeat the player. The code does this by looking at the difference between the x and y positions of the player and the enemy. If, let's say, the difference between the x positions of the player and enemy are negative (which means the enemy is to the right of the player) the enemy moves to the left and tries to equalize its x position with the players. The enemies have a range in which they are hostile, this range is determined by the length between their top left points, which is calculated with the Pythagorean Theorem. The enemies also have a reachability condition, which stops them from moving out of bounds.

projectile: The projectile is, again, similar to the player or the enemy. It changes its xpos and ypos according to its reachability condition. It gets the direction output from the player and uses it to determine its own position. It has a condition, called expired, which determines whether the projectile can still move or not. If it collides to the wall, then it is not usable, therefore expired.

level_draw: This module is where the game conditions exist. Conditions such as the level transitions, the enemy death, the treasure spawn all happen here. The projectile conditions, such as when the projectile gets drawn and how expired condition affects the projectile are all determined here. The levels also get drawn here using the draw package that is in the packages module. It has many if conditions and many determine what gets drawn when.

I would also like to mention the digital design part of the project. There exists many finite-state machines inside the code. One example to this would be the level transition. There exists two inputs in this case and one output. If the player is within the doors range the output is 0 but the state changes. If the player is in the range and also entered the correct code, then the state changes and the output is 1. If the code is wrong, the state changes but the output is still 0. This is not the only example though, the enemy and character movement, the projectile behavior and some general mechanics can all be observed as finite state machines.

- Results

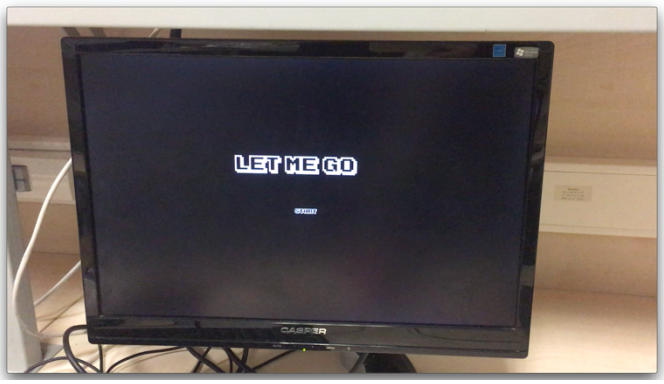

Figure 5: Start Screen

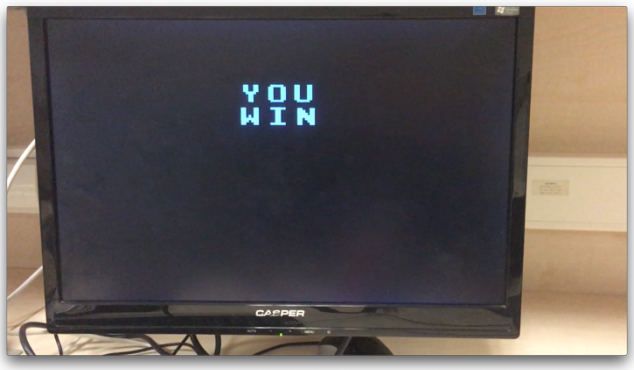

Figure 6: End Screen

The results were as expected for the most part. The drawing of the level, the behavior of the player and the enemy are all working correctly. I did have some trouble finding good colors for the map, since the VGA connection on BASYS 3 takes

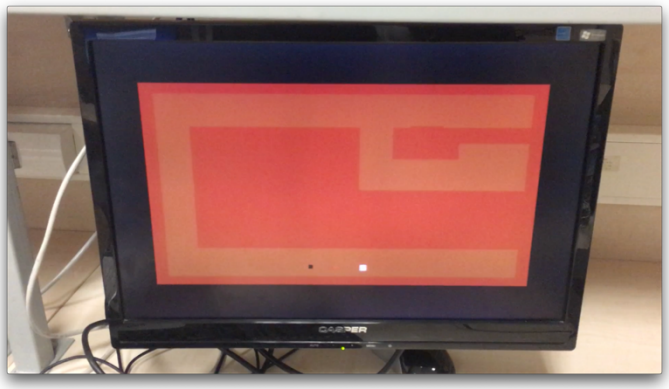

Figure 7: The enemy and the player, the player throws a projectile

Figure 8: The treasure drops after the enemy is defeated.

4-bit rgb values instead of 8-bit values. The projectile was also troublesome, since it is an entity that exists without any input other than the start condition, but did work alright at the end. Some stills from the game can be seen above.

- Conclusion

I will start the end of the report by first talking about the problems that I encountered along the way, and how I resolved them. The first hardship that I came across was the lack of a random number generator on VHDL. This came across as a surprise to me, since every coding language that I have coded in till this day had a library which contained a random number generated. This problem allowed me to research more about random number generators and how they really generate these "random" numbers. After my research I was able to code a random number generator; however, there existed another problem. This problem was the non-randomness of the output. I resolved this by making it so that the players input determines the time when the randomly generated number is obtained, which makes it pretty random. Another hardship was the movement of the player, at first it was extremely fast. So fast that the player would disappear out of the map right at the start. This was resolved by creating a slower clock (100Hz) to check the button inputs. The movement wasn't the only problem though, the player was able to reach out of the bounds of the map. There had to be constraints to keep the player inside the play area. I first thought about using an array to resolve this issue, but then found a simpler way which was to check the color of the pixels around the player. The projectile was another issue, it was pretty unpredictable and hard to implement correctly. This was resolved by initializing its position at the start of the game to near the player.

The project was a great learning experience, in many different ways. First off, it familiarized me with different aspects of digital design, such as finite-state machines. It further improved my coding ability and understanding of VHDL and I also believe improved my ability to code in general. It was pretty fun for me to code modules such as enemy, which had no user input and was its own entity. It also showed me how software and hardware can work in harmony to create something beautiful. This is something we see every day, but it was great to create this harmony myself. All in all, this was a great learning experience that goes beyond VHDL and digital design in general and has thought me a lot. I look forward to making a similar project in the future.

- References

"Pseudo Random Number Generator Using LFSR in VHDL." Stack Overflow. Accessed December 29, 2018. https://stackoverflow.com/questions/43081067/pseudo-random-number-generator-using-lfsr-in-vhdl.

"VHDL Debouncer Circuit." Electrical Engineering Stack Exchange. Accessed December 29, 2018. https://electronics.stackexchange.com/questions/32260/vhdl-debouncer-circuit.

- Appendices

  - Top Module (main)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity main is
   Port ( input_code : in STD_LOGIC_VECTOR (15 downto 0);
        buttons : in STD_LOGIC_VECTOR (4 downto 0);
        led_code : out STD_LOGIC_VECTOR (15 downto 0);
        digit : out STD_LOGIC_VECTOR (6 downto 0);
        display : out STD_LOGIC_vector (3 downto 0);
        hsync, vsync : out std_logic;
        r, g, b : out std_logic_vector (3 downto 0);
        clock : in STD_LOGIC);
end main;

architecture Behavioral of main is

component vga_main is
  Port ( input_code: in std_logic_vector (15 downto 0);
  clk : in std_logic;
  buttons : in std_logic_vector (4 downto 0);
  hsync, vsync : out std_logic;
  r, g, b : out std_logic_vector (3 downto 0);
  random: out std_logic_vector (15 downto 0));
end component;

component clock_divider is
Port ( clk_in : in STD_LOGIC;
        reset : in STD_LOGIC;
        divider : in integer;
        clk_out : out STD_LOGIC);
        end component;

component decoder is
Port ( input : in std_logic_vector (3 downto 0);
     digit : out std_logic_vector (6 downto 0));
end component;

component ssd_sync is
Port ( clock : in STD_LOGIC;
        code_0 : in STD_LOGIC_VECTOR (6 downto 0); --decoder output
        code_1 : in STD_LOGIC_VECTOR (6 downto 0); --decoder output
        code_2 : in STD_LOGIC_VECTOR (6 downto 0); --decoder output
        code_3 : in STD_LOGIC_VECTOR (6 downto 0); --decoder output
        digit : out STD_LOGIC_VECTOR (6 downto 0);
```

```
        display : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component debounce is
Port (buttons: in std_logic_vector (4 downto 0);
    clk : in std_logic;
    buttons_out : out std_logic_vector (4 downto 0));
end component;

signal buttons_out: std_logic_vector(4 downto 0);
signal code_0 : STD_LOGIC_VECTOR (6 downto 0);
signal code_1 : STD_LOGIC_VECTOR (6 downto 0);
signal code_2 : STD_LOGIC_VECTOR (6 downto 0);
signal code_3 : STD_LOGIC_VECTOR (6 downto 0);
signal clk_out : std_logic;
signal random : std_logic_vector (15 downto 0);

begin

bounce : debounce port map ( buttons, clock, buttons_out);
vga : vga_main port map (input_code, clock, buttons_out, hsync, vsync, r, g, b, random);
clk : clock_divider port map (clock, '0', 9999 , clk_out);
d_0 : decoder port map (random (3 downto 0), code_0);
d_1 : decoder port map (random (7 downto 4), code_1);
d_2 : decoder port map (random (11 downto 8), code_2);
d_3 : decoder port map (random (15 downto 12), code_3);
sync : ssd_sync port map (clk_out, code_0, code_1, code_2, code_3, digit, display);

led_code  <= input_code;

end Behavioral;
```

- debouncer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity debounce is
Port (buttons: in std_logic_vector (4 downto 0);
    clk : in std_logic;
    buttons_out : out std_logic_vector (4 downto 0));
end debounce;
architecture Behavioral of debounce is
signal pass_1, pass_2, pass_3: std_logic_vector( 4 downto 0);
begin
  process(clk) begin
    if rising_edge(clk) then
      for i in 0 to 4 loop
        pass_1(i) <= buttons(i);
        pass_2(i) <= pass_1(i);
        pass_3(i) <= pass_2(i) ;
        end loop;
    end if;
```

```
    end process;
    process(pass_1, pass_2, pass_3) begin
      for i in 0 to 4 loop
        buttons_out(i) <= pass_1(i)  and pass_2(i)  and pass_3(i) ;
      end loop;
    end process;
end Behavioral;
```

- clock_divider

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

entity clock_divider is
    Port ( clk_in : in STD_LOGIC;
           reset : in STD_LOGIC;
           divider : in integer;
           clk_out : out STD_LOGIC);
end clock_divider;

architecture Behavioral of clock_divider is

signal out_signal: STD_LOGIC;
signal counter : integer range 0 to 10**7 := 0;

begin

process( clk_in, reset) begin
    if (reset = '1') then
        out_signal <= '0';
        counter <= 0;

    elsif rising_edge(clk_in) then
        if (counter = divider) then
            out_signal <= not out_signal;
            counter <= 0;
        else
            counter <= counter + 1;
        end if;
     end if;
end process;

clk_out <= out_signal;

end Behavioral;
```

- rng (random number generator)

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

```
entity rng is
Port ( clock : in STD_LOGIC;
      reset : in STD_LOGIC;
      en : in STD_LOGIC;
      Q : out STD_LOGIC_VECTOR (15 downto 0);
      check: out STD_LOGIC);
end rng;

architecture Behavioral of rng is

signal Qt: STD_LOGIC_VECTOR(15 downto 0) := x"0001";

begin

PROCESS(clock)
variable tmp : STD_LOGIC := '0';
BEGIN

IF rising_edge(clock) THEN
  IF (reset='1') THEN
    Qt <= x"0001";
  ELSIF en = '1' THEN
    tmp := Qt(3) XOR Qt(2) XOR Qt(1) XOR Qt(0);
    Qt <= tmp & Qt(15 downto 1);
  END IF;

END IF;
END PROCESS;
check <= Qt(15);
Q <= Qt;

end Behavioral;
```

- decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity decoder is
   Port ( input : in std_logic_vector (3 downto 0);
        digit : out std_logic_vector (6 downto 0));
end decoder;

architecture Behavioral of decoder is

begin
process (input) begin
case input is
   when "0000" => digit <= "0000001"; -- "0"
   when "0001" => digit <= "1001111"; -- "1"
   when "0010" => digit <= "0010010"; -- "2"
   when "0011" => digit <= "0000110"; -- "3"
   when "0100" => digit <= "1001100"; -- "4"
```

```
   when "0101" => digit <= "0100100"; -- "5"
   when "0110" => digit <= "0100000"; -- "6"
   when "0111" => digit <= "0001111"; -- "7"
   when "1000" => digit <= "0000000"; -- "8"
   when "1001" => digit <= "0000100"; -- "9"
   when "1010" => digit <= "0000010"; -- a
   when "1011" => digit <= "1100000"; -- b
   when "1100" => digit <= "0110001"; -- C
   when "1101" => digit <= "1000010"; -- d
   when "1110" => digit <= "0110000"; -- E
   when "1111" => digit <= "0111000"; -- F
   end case;
end process;
end Behavioral;
```

- ssd_sync

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

entity ssd_sync is
   Port ( clock : in STD_LOGIC;
        code_0 : in STD_LOGIC_VECTOR (6 downto 0); --decoder output
        code_1 : in STD_LOGIC_VECTOR (6 downto 0); --decoder output
        code_2 : in STD_LOGIC_VECTOR (6 downto 0); --decoder output
        code_3 : in STD_LOGIC_VECTOR (6 downto 0); --decoder output
        digit : out STD_LOGIC_VECTOR (6 downto 0);
        display : out STD_LOGIC_VECTOR (3 downto 0));
end ssd_sync;

architecture Behavioral of ssd_sync is

signal count : integer range 0 to 4;

begin

process( clock)
begin

if (rising_edge(clock)) then

   case count is
      when 0 => display <= "1110";
            digit <= code_0;
      when 1 => display <= "1101";
            digit <= code_1;
      when 2 => display <= "1011";
            digit <= code_2;
      when 3 => display <= "0111";
            digit <= code_3;
      when others => display <= "1111";
               digit <= "1111111";
```

```
    end case;

    count <= count + 1;
    if (count = 4) then

       count <= 0;

     end if;

  end if;
  end process;


  end Behavioral;
```

- vga_main

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

entity vga_main is
  Port ( input_code: in std_logic_vector (15 downto 0);
  clk : in std_logic;
  buttons : in std_logic_vector (4 downto 0);
  hsync, vsync : out std_logic;
  r, g, b : out std_logic_vector (3 downto 0);
  random: out std_logic_vector (15 downto 0));
end vga_main;

architecture Behavioral of vga_main is

component vga_sync is
    Port ( clk : in STD_LOGIC;
    hsync, vsync : out STD_LOGIC;
    hcurrent, vcurrent : out integer);
end component;

component level_draw is
    Port ( input_code: in std_logic_vector (15 downto 0);
          clk : in STD_LOGIC;
          hcurrent, vcurrent : in integer;
          r,g,b : out STD_LOGIC_VECTOR (3 downto 0);
          buttons : in std_logic_vector (4 downto 0);
           random: out std_logic_vector (15 downto 0));
end component;
signal hcurrent: integer range 0 to 1904:= 0;
signal vcurrent: integer range 0 to 932:= 0;

begin

controller : vga_sync port map (clk, hsync, vsync, hcurrent, vcurrent);
```

```
drawer : level_draw port map (input_code, clk, hcurrent, vcurrent, r, g, b, buttons,
random);

end Behavioral;
```

- vga_sync

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

entity vga_sync is
   Port ( clk : in STD_LOGIC;
          hsync, vsync : out STD_LOGIC;
          hcurrent : out integer range 0 to 1904:= 0;
          vcurrent : out integer range 0 to 932:= 0);
end vga_sync;

architecture Behavioral of vga_sync is

signal hpos: integer range 0 to 1904:= 0;
signal vpos: integer range 0 to 932:= 0;

begin

process(clk) begin

   if rising_edge(clk) then
      if (hpos < 1904) then
         hpos <= hpos + 1;
      else
         hpos <= 0;
         if (vpos < 932) then
            vpos <= vpos + 1;
            else
            vpos <= 0;
         end if;
      end if;
      if (hpos > 80 and hpos < 232) then
         hsync <= '0';
      else
         hsync <= '1';
      end if;
      if (vpos > 1 and vpos < 4) then
         vsync <= '0';
      else
         vsync <= '1';
      end if;

      hcurrent <= hpos;
      vcurrent <= vpos;

   end if;
```

```
end process;


end Behavioral;
```

- level_drawer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use work.draw.all;
use work.arrays.all;
use work.IsSomething.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity level_draw is
   Port ( input_code: in std_logic_vector (15 downto 0);
        clk : in STD_LOGIC;
        hcurrent, vcurrent : in integer;
        r,g,b : out STD_LOGIC_VECTOR (3 downto 0);
        buttons: in std_logic_vector(4 downto 0);
        random: out std_logic_vector (15 downto 0));
end level_draw;

architecture Behavioral of level_draw is

component player is
Port ( buttons : in STD_LOGIC_VECTOR (4 downto 0);
   clock, reset, difficulty : in std_logic;
   inx, iny, hcurrent, vcurrent: in integer;
   rgb_sig, play_rgb: in std_logic_vector (11 downto 0);
   cxpos, cypos : out integer;
   direction: out std_logic_vector (3 downto 0);
   action: out std_logic);
end component;

component rng is
Port ( clock : in STD_LOGIC;
     reset : in STD_LOGIC;
     en : in STD_LOGIC;
     Q : out STD_LOGIC_VECTOR (15 downto 0);
     check: out STD_LOGIC);
     end component;

component enemy is
Port ( clock, reset, enemy_type, difficulty : in std_logic;
        inx, iny, hcurrent, vcurrent, cposx, cposy, hostility_range : in integer;
        rgb_sig, play_rgb: in std_logic_vector (11 downto 0);
        eposx, eposy : out integer);
end component;
```

```
component projectile is
  Port ( clock: in std_logic;
  cposx, cposy, hcurrent, vcurrent: in integer;
  rgb_sig, play_rgb: in std_logic_vector (11 downto 0);
  direction: in std_logic_vector( 3 downto 0);
  pposx, pposy: out integer;
  enable: in std_logic;
  expired: out std_logic;
  reset: in std_logic );
end component;

component clock_divider is
    Port ( clk_in : in STD_LOGIC;
           reset : in STD_LOGIC;
           divider : in integer;
           clk_out : out STD_LOGIC);
end component;


--signals
signal level, prev_level: std_logic_vector(1 downto 0):= "00";
signal restart_level: std_logic_vector (1 downto 0);
signal draw : std_logic_vector (20 downto 0);
signal incx, cposx, doorx, tposx, pposx, points, eposx, inex : integer range 0 to 1904;
signal incy, cposy, doory, tposy, pposy, eposy, iney : integer range 0 to 932;
signal rgb_o : RGB;
signal rgb_sig, rgb_change : std_logic_vector (11 downto 0);
signal reset_p, indicator, reset_e, action, enable_prj, expired, reset_prj, enemy_type,
difficulty, clk_out, draw_enemy, draw_treasure: std_logic;
signal rng_enable: std_logic:= '1';
signal enemy_hp: std_logic_vector (2 downto 0);
signal direction: std_logic_vector(3 downto 0);
signal dir_of_prj: std_logic_vector (3 downto 0);
signal rng_random: std_logic_vector (15 downto 0);

--constants
constant outside_rgb: std_logic_vector(11 downto 0):= "100000100000";
constant inside_rgb: std_logic_vector(11 downto 0):= "100000110000";
constant character_rgb: std_logic_vector(11 downto 0):= "111111111111";
constant door_rgb: std_logic_vector(11 downto 0):= "100000100000";
constant enemy_rgb: std_logic_vector(11 downto 0):= "000000000000";
constant treasure_rgb: std_logic_vector(11 downto 0):= "111111110000";

begin


divider: clock_divider port map (clk, '0', 100000000, clk_out);
rdm : rng port map (clk, '0', rng_enable, rng_random);
ply: player port map (buttons, clk, reset_p, difficulty, incx, incy, hcurrent, vcurrent,
rgb_sig, inside_rgb, cposx, cposy, direction, action);
enm1: enemy port map (clk, reset_e, enemy_type, difficulty ,inex, iney, hcurrent,
vcurrent, cposx, cposy, 200, rgb_sig, inside_rgb, eposx, eposy);
```

```
prj: projectile port map (clk, cposx, cposy, hcurrent, vcurrent, rgb_sig, inside_rgb,
dir_of_prj, pposx, pposy, enable_prj, expired, reset_prj);
--enemy_prj: projectile port map (clk, cposx, cposy, hcurrent, vcurrent, rgb_sig,
inside_rgb, dir_of_prj, pposx, pposy, enable_prj, expired, reset_prj);

r <= rgb_sig(11 downto 8);
g <= rgb_sig(7 downto 4);
b <= rgb_sig(3 downto 0);

level_process : process (hcurrent, vcurrent, clk, clk_out) begin

reset_prj <= '0';
reset_e <= '0';
reset_p <= '0';

   if rising_edge(clk) then

      if level = "00" then
      inex <= 664;
      iney <= 732;
      name( LETMEGO, hcurrent, vcurrent, "111111111111", rgb_o(0), draw(0));
      start (ST, hcurrent, vcurrent, "111111111111", rgb_o(1), draw(1));
         if action = '1' then
            level <= "01";
         end if;
      end if;

      if level = "01" then
         doorx <= 1384;
         doory <= 277;
         incx <= 1694;
         incy <= 732;
         inex <= 664;
         iney <= 732;

         --draw level

         rectangle( 1200, 660, hcurrent, vcurrent, 584, 152, outside_rgb, rgb_o(0),
draw(0)); -- outside constraints
         rectangle( 1120, 600, hcurrent, vcurrent, 624, 182, inside_rgb, rgb_o(1), draw(1));
-- inside
         rectangle( 20, 20, hcurrent, vcurrent, cposx, cposy, character_rgb, rgb_o(20),
draw(20));
         rectangle( 511, 400, hcurrent, vcurrent, 724, 282, outside_rgb, rgb_o(2), draw(2));
         rectangle( 510, 200, hcurrent, vcurrent, 1234, 482, outside_rgb, rgb_o(3),
draw(3));
         rectangle( 201, 100, hcurrent, vcurrent, 1334, 282, outside_rgb, rgb_o(4),
draw(4));
         rectangle( 210, 50, hcurrent, vcurrent, 1534, 282, outside_rgb, rgb_o(5), draw(5));
         rectangle( 100, 10, hcurrent, vcurrent, doorx, doory, door_rgb , rgb_o(6),
draw(6));
```

```
        --numbers(digits, points, hcurrent, vcurrent, "111111111111", rgb_o(7), draw(7));

      -- reset condition when player dies
      if (((cposx - eposx + 3) ** 2 + (cposy - eposx + 3) ** 2 <= 20 ** 2) and enemy_hp(0) =
'0') then
         reset_p <= '1';
         reset_e <= '1';
      end if;

      -- enemy draw condition
      if (enemy_hp(0) = '0') then
         rectangle( 14, 14, hcurrent, vcurrent, eposx, eposy, enemy_rgb, rgb_o(8),
draw(8));
         tposx <= eposx + 5;
         tposy <= eposy + 5;
      end if;
       --projectile conditions
       if (action = '1' and indicator = '0') then
         reset_prj <= '1';
         dir_of_prj <= direction;
         enable_prj <= '1';
         indicator <= '1';
      end if;

      if (expired = '1')  then
         enable_prj <= '0';
         reset_prj <= '1';
         indicator <= '0';
      end if;

         -- enemy death/ treasure draw condition
         if ( isInRange(pposx + 5, pposy + 2, eposx + 10, eposy + 10, 30) = '1' and draw(19) =
'1') then
            enemy_hp(0) <= '1';
            draw_treasure <= '1';
            points <= points + 1;
         end if;

         -- Level transition
          if ( isInRange(cposx + 10, cposy + 10, doorx , doory, 30) = '1'  and rng_random =
input_code ) then
            random <= x"0000";
            rng_enable <= '1';
            enemy_hp <= "000";
            level <= "11";
         end if;



         --draws treasure upon enemy death
         if draw_treasure = '1' then
            rectangle( 10, 10, hcurrent, vcurrent, tposx, tposy, treasure_rgb, rgb_o(8),
draw(8));
```

```
        end if;

        --obtains the randomly generated number
        if ( isInRange(cposx + 10, cposy + 10, tposx +5 , tposy + 5, 15) = '1' and action = '1')
then
           rng_enable <= '0';
           random <= rng_random;
           draw_treasure <= '0';
        end if;

        --enables projectile draw
        if (enable_prj = '1') then
          rectangle( 8, 8, hcurrent, vcurrent, pposx, pposy, "111100000000" , rgb_o(19),
draw(19)); -- projectile
        end if;
      end if;

        if level = "11" then
        won (wonA, hcurrent, vcurrent, "111111111111", rgb_o(15), draw(15));
           --rectangle( 1200, 660, hcurrent, vcurrent, 584, 152, "111100000000", rgb_o(0),
draw(0));
           if action = '1' then

              level <= "01";
           end if;
        end if;

        if (draw(20) = '1') then
           rgb_sig <= rgb_o(20);
        elsif (draw(19) = '1') then
           rgb_sig <= rgb_o(19);
        elsif (draw(18) = '1') then
          rgb_sig <= rgb_o(18);
        elsif (draw(17) = '1') then
          rgb_sig <= rgb_o(17);
        elsif (draw(16) = '1') then
           rgb_sig <= rgb_o(16);
        elsif (draw(15) = '1') then
           rgb_sig <= rgb_o(15);
        elsif (draw(14) = '1') then
           rgb_sig <= rgb_o(14);
        elsif (draw(13) = '1') then
           rgb_sig <= rgb_o(13);
        elsif (draw(12) = '1') then
           rgb_sig <= rgb_o(12);
        elsif (draw(11) = '1') then
           rgb_sig <= rgb_o(11);
        elsif (draw(10) = '1') then
           rgb_sig <= rgb_o(10);
        elsif (draw(9) = '1') then
           rgb_sig <= rgb_o(9);
        elsif draw(8) = '1' then
```

```
            rgb_sig <= rgb_o(8);
        elsif (draw(7) = '1') then
            rgb_sig <= rgb_o(7);
        elsif (draw(6) = '1') then
            rgb_sig <= rgb_o(6);
        elsif (draw(5) = '1') then
            rgb_sig <= rgb_o(5);
        elsif (draw(4) = '1') then
            rgb_sig <= rgb_o(4);
        elsif (draw(3) = '1') then
            rgb_sig <= rgb_o(3);
        elsif (draw(2) = '1') then
            rgb_sig <= rgb_o(2);
        elsif (draw(1) = '1') then
            rgb_sig <= rgb_o(1);
        elsif (draw(0) = '1') then
            rgb_sig <= rgb_o(0);
        else
            rgb_sig <= (others => '0');
        end if;

    end if;
end process;
end Behavioral;
```

- packages (for the sake of the reader, I will not put the bitmaps in their entirety, just the names of the constants)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

package arrays is
    type DIR_ARRAY is array (4 downto 0) of std_logic_vector (3 downto 0);
    type RGB is array (20 downto 0) of std_logic_vector (11 downto 0);
    type INT_ARRAY is array (5 downto 0) of integer;
    type SAVE_RGB is array (1903 downto 0, 932 downto 0) of std_logic_vector(11 downto
0);
    type BITMAP_LETMEGO is array (89 downto 0) of std_logic_vector(499 downto 0) ;
    type BITMAP_START is array (16 downto 0) of std_logic_vector (299 downto 0);
    type BITMAP_WON is array (125 downto 0) of std_logic_vector (299 downto 0);
    type BITMAP_NUMBERS is array (23 downto 0) of std_logic_vector (282 downto 0);
--    type BITMAP_DIFFICULTY
--    type BITMAP_NORMAL
--    type BITMAP_HARD

    constant LETMEGO: BITMAP_LETMEGO:=


    constant ST: BITMAP_START:= (bitmap);
```

```
   constant wonA : BITMAP_WON := (bitmap);


   constant digits: BITMAP_NUMBERS:= (bitmap);


end arrays;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;


package isSomething is

   function isInRange( cxpos, cypos, xpos, ypos, rangeamount: in integer) return
std_logic;

   procedure isReachable( signal clock: in std_logic;
   xpos, ypos, hcurrent, vcurrent: in integer;
   rgb_sig, play_rgb: in std_logic_vector(11 downto 0);
   signal reachable: out std_logic_vector (3 downto 0));

end isSomething;

package body isSomething is

   function isInRange(cxpos, cypos, xpos, ypos, rangeamount: in integer) return std_logic
is

   begin
      if ( (cxpos - xpos) ** 2 + (cypos - ypos) ** 2 <= rangeamount ** 2 ) then
      return '1';
      else
      return '0';
      end if;
   end function;

   procedure isReachable( signal clock: in std_logic;
      xpos, ypos, hcurrent, vcurrent: in integer;
      rgb_sig, play_rgb: in std_logic_vector(11 downto 0);
      signal reachable: out std_logic_vector (3 downto 0)) is
   begin

      if rising_edge(clock) then
         if vcurrent = ypos - 1 and (hcurrent <= xpos + 21 and hcurrent >= xpos - 1) then
               if (rgb_sig = play_rgb) then
                  reachable(0) <= '1';
               else
                  reachable(0) <= '0';
```

```
            end if;
         end if;
         if hcurrent = xpos + 21 and (vcurrent <= ypos + 21 and vcurrent >= ypos - 1) then
            if (rgb_sig = play_rgb) then
               reachable(1) <= '1';
            else
               reachable(1) <= '0';
            end if;
         end if;
         if vcurrent = ypos + 21 and (hcurrent <= xpos + 21 and hcurrent >= xpos - 1) then
            if (rgb_sig = play_rgb) then
               reachable(2) <= '1';
            else
               reachable(2) <= '0';
            end if;
         end if;
         if hcurrent = xpos - 1 and (vcurrent <= ypos + 21 and vcurrent >= ypos - 1) then
            if (rgb_sig = play_rgb) then
               reachable(3) <= '1';
            else
               reachable(3) <= '0';
            end if;
         end if;
      end if;
   end procedure;
end isSomething;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
library work;
use work.isSomething.all;
use work.arrays.all;

package draw is
   procedure rectangle( xside, yside: in integer;
    hcurrent, vcurrent, xpos, ypos: in integer;
    rgb: in std_logic_vector (11 downto 0);
   signal rgb_o: out std_logic_vector (11 downto 0);
   signal draw: out std_logic);

   procedure circle( radius: in integer;
       hcurrent, vcurrent, xpos, ypos: in integer;
       rgb: in std_logic_vector (11 downto 0);
      signal rgb_o: out std_logic_vector (11 downto 0);
      signal draw: out std_logic;
      signal enable: in std_logic);

   procedure name( bitmap: in BITMAP_LETMEGO;
   hcurrent, vcurrent: in integer;
       rgb: in std_logic_vector (11 downto 0);
```

```
      signal rgb_o: out std_logic_vector (11 downto 0);
      signal draw: out std_logic);

   procedure start(bitmap: in BITMAP_START;
         hcurrent, vcurrent: in integer;
          rgb: in std_logic_vector (11 downto 0);
         signal rgb_o: out std_logic_vector (11 downto 0);
         signal draw: out std_logic);

   procedure won (bitmap: in BITMAP_WON;
   hcurrent, vcurrent: in integer;
   rgb: in std_logic_vector (11 downto 0);
   signal rgb_o: out std_logic_vector (11 downto 0);
   signal draw: out std_logic );

   procedure numbers(
   bitmap: in BITMAP_NUMBERS;
   number, hcurrent, vcurrent: in integer;
   rgb: in std_logic_vector (11 downto 0);
   signal rgb_o: out std_logic_vector (11 downto 0);
   signal draw: out std_logic);

end draw;

package body draw is
   procedure rectangle( xside, yside: in integer;
    hcurrent, vcurrent, xpos, ypos: in integer;
   rgb: in std_logic_vector (11 downto 0);
   signal rgb_o: out std_logic_vector (11 downto 0);
   signal draw: out std_logic) is
   begin
      if ((hcurrent > xpos and hcurrent < (xpos + xside)) and (vcurrent > ypos and vcurrent
< (ypos + yside))) then
         rgb_o <= rgb;
         draw <= '1';
      else
         draw <= '0';
      end if;
   end rectangle;

   procedure circle( radius: in integer;
          hcurrent, vcurrent, xpos, ypos: in integer;
          rgb: in std_logic_vector (11 downto 0);
         signal rgb_o: out std_logic_vector (11 downto 0);
         signal draw: out std_logic;
         signal enable: in std_logic) is
         begin

         if (enable = '1' and (hcurrent - xpos) ** 2 + (vcurrent - ypos) ** 2 < radius ** 2)
then
            rgb_o <= rgb;
            draw <= '1';
```

```
        else
           draw <= '0';
        end if;
   end circle;

   procedure name( bitmap: in BITMAP_LETMEGO;
   hcurrent, vcurrent: in integer;
         rgb: in std_logic_vector (11 downto 0);
        signal rgb_o: out std_logic_vector (11 downto 0);
        signal draw: out std_logic) is
        begin

           if ( hcurrent > 954 and hcurrent < 1454) and (vcurrent > 332 and vcurrent < 422)
then
           draw <= not bitmap(421 - vcurrent)(1453 - hcurrent);
           rgb_o <= rgb;
           end if;
        end name;

   procedure start(bitmap: in BITMAP_START;
           hcurrent, vcurrent: in integer;
               rgb: in std_logic_vector (11 downto 0);
              signal rgb_o: out std_logic_vector (11 downto 0);
              signal draw: out std_logic) is begin
        if ( hcurrent > 1145 and hcurrent < 1345) and (vcurrent > 531 and vcurrent < 549)
then
                        draw <= bitmap(548 - vcurrent)(1344 - hcurrent);
                        rgb_o <= rgb;
                        end if;
                     end start;



     procedure won (bitmap: in BITMAP_WON;
     hcurrent, vcurrent: in integer;
     rgb: in std_logic_vector (11 downto 0);
     signal rgb_o: out std_logic_vector (11 downto 0);
     signal draw: out std_logic ) is begin
     if (hcurrent > 1034 and hcurrent < 1334) and (vcurrent > 232 and vcurrent < 358)
then
           draw <= bitmap(357 - vcurrent)(1333 - hcurrent);
           rgb_o <= rgb;
           end if;
     end won;


     procedure numbers(
        bitmap: in BITMAP_NUMBERS;
        number, hcurrent, vcurrent: in integer;
        rgb: in std_logic_vector (11 downto 0);
        signal rgb_o: out std_logic_vector (11 downto 0);
        signal draw: out std_logic) is begin
```

```
        for i in 0 to 9 loop
          if (i = number) then
            if (hcurrent > 1172 and hcurrent < 1196) and (vcurrent > 79 and vcurrent <
105) then
                draw <= bitmap(104 - vcurrent)(1195 - hcurrent + (29 * (9 - number)));
                rgb_o <= rgb;
                end if;
              end if;
          end loop;
              end numbers;


end draw;
```

- Constraints

```
##leds
set_property PACKAGE_PIN L1 [get_ports { led_code[15]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[15]}]
set_property PACKAGE_PIN P1 [get_ports { led_code[14]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[14]}]
set_property PACKAGE_PIN N3 [get_ports { led_code[13]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[13]}]
set_property PACKAGE_PIN P3 [get_ports { led_code[12]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[12]}]
set_property PACKAGE_PIN U3 [get_ports { led_code[11]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[11]}]
set_property PACKAGE_PIN W3 [get_ports { led_code[10]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[10]}]
set_property PACKAGE_PIN V3 [get_ports { led_code[9]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[9]}]
set_property PACKAGE_PIN V13 [get_ports { led_code[8]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[8]}]
set_property PACKAGE_PIN V14 [get_ports { led_code[7]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[7]}]
set_property PACKAGE_PIN U14 [get_ports { led_code[6]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[6]}]
set_property PACKAGE_PIN U15 [get_ports { led_code[5]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[5]}]
set_property PACKAGE_PIN W18 [get_ports { led_code[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[4]}]
 set_property PACKAGE_PIN V19 [get_ports { led_code[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[3]}]
set_property PACKAGE_PIN U19 [get_ports { led_code[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[2]}]
set_property PACKAGE_PIN E19 [get_ports { led_code[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[1]}]
set_property PACKAGE_PIN U16 [get_ports { led_code[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports { led_code[0]}]

##switches
```

```
set_property PACKAGE_PIN R2 [get_ports {input_code[15]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[15]}]
set_property PACKAGE_PIN T1 [get_ports {input_code[14]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[14]}]
set_property PACKAGE_PIN U1 [get_ports {input_code[13]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[13]}]
set_property PACKAGE_PIN W2 [get_ports {input_code[12]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[12]}]
set_property PACKAGE_PIN R3 [get_ports {input_code[11]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[11]}]
set_property PACKAGE_PIN T2 [get_ports {input_code[10]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[10]}]
set_property PACKAGE_PIN T3 [get_ports {input_code[9]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[9]}]
set_property PACKAGE_PIN V2 [get_ports {input_code[8]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[8]}]
set_property PACKAGE_PIN W13 [get_ports {input_code[7]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[7]}]
set_property PACKAGE_PIN W14 [get_ports {input_code[6]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[6]}]
set_property PACKAGE_PIN V15 [get_ports {input_code[5]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[5]}]
set_property PACKAGE_PIN W15 [get_ports {input_code[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[4]}]
set_property PACKAGE_PIN W17 [get_ports {input_code[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[3]}]
set_property PACKAGE_PIN W16 [get_ports {input_code[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[2]}]
set_property PACKAGE_PIN V16 [get_ports {input_code[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[1]}]
set_property PACKAGE_PIN V17 [get_ports {input_code[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {input_code[0]}]

##clock
set_property PACKAGE_PIN W5 [get_ports {clock}]
   set_property IOSTANDARD LVCMOS33 [get_ports {clock}]
          create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
clock]

##7 segment display - digits
set_property PACKAGE_PIN W7 [get_ports {digit[6]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {digit[6]}]
set_property PACKAGE_PIN W6 [get_ports {digit[5]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {digit[5]}]
set_property PACKAGE_PIN U8 [get_ports {digit[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {digit[4]}]
set_property PACKAGE_PIN V8 [get_ports {digit[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {digit[3]}]
set_property PACKAGE_PIN U5 [get_ports {digit[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {digit[2]}]
set_property PACKAGE_PIN V5 [get_ports {digit[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {digit[1]}]
```

```
set_property PACKAGE_PIN U7 [get_ports {digit[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {digit[0]}]

##7 segment display - displays
set_property PACKAGE_PIN U2 [get_ports {display[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {display[0]}]
set_property PACKAGE_PIN U4 [get_ports {display[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {display[1]}]
set_property PACKAGE_PIN V4 [get_ports {display[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {display[2]}]
set_property PACKAGE_PIN W4 [get_ports {display[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {display[3]}]

##Buttons
set_property PACKAGE_PIN T18 [get_ports {buttons[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {buttons[0]}]
set_property PACKAGE_PIN T17 [get_ports {buttons[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {buttons[1]}]
set_property PACKAGE_PIN U17 [get_ports {buttons[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {buttons[2]}]
set_property PACKAGE_PIN W19 [get_ports {buttons[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {buttons[3]}]
set_property PACKAGE_PIN U18 [get_ports {buttons[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {buttons[4]}]

## VGA
set_property PACKAGE_PIN G19 [get_ports {r[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {r[0]}]
set_property PACKAGE_PIN H19 [get_ports {r[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {r[1]}]
set_property PACKAGE_PIN J19 [get_ports {r[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {r[2]}]
set_property PACKAGE_PIN N19 [get_ports {r[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {r[3]}]
set_property PACKAGE_PIN N18 [get_ports {b[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {b[0]}]
set_property PACKAGE_PIN L18 [get_ports {b[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {b[1]}]
set_property PACKAGE_PIN K18 [get_ports {b[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {b[2]}]
set_property PACKAGE_PIN J18 [get_ports {b[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {b[3]}]
set_property PACKAGE_PIN J17 [get_ports {g[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {g[0]}]
set_property PACKAGE_PIN H17 [get_ports {g[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {g[1]}]
set_property PACKAGE_PIN G17 [get_ports {g[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {g[2]}]
set_property PACKAGE_PIN D17 [get_ports {g[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {g[3]}]

set_property PACKAGE_PIN P19 [get_ports hsync]
```

```
    set_property IOSTANDARD LVCMOS33 [get_ports hsync]
set_property PACKAGE_PIN R19 [get_ports vsync]
    set_property IOSTANDARD LVCMOS33 [get_ports vsync]
```