DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE

POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE

MASTER DEGREE IN

AUTOMATION ENGINEERING

VHDL Pong Game

Professor:

Martino De Carlo

Team:

Angiolillo Alessandro (590317)

Basciani Andrea (590251)

Bux Luca (590260)

de Giosa Fabio (589928)

Accademic Year 2022-2023

SUMMARY

1. INTRODUCTION

This paper concerns about the development of a Pong game, using VHDL language and a FPGA board. **Pong** is a two-dimensional arcade table-tennis game developed by Atari in 70's and very popular until the 00's. In this game two players can control two paddles by moving them vertically across the borders of the screen; to score a point the player must make the opposite player not able to return the ball on the other side. To win a game, a player needs to score 3 or 10 points depending on the rules.

VHDL (VHSIC Hardware Description Language) is a hardware description language that can model the behavior and structure of digital systems at multiple levels of abstraction. It is commonly used to write text models that describe logic circuits, which are then processed by synthesis programs and tested using simulation models in testbench. Some features that make VHDL useful to this project are the concurrency, which means that multiple operations can occur simultaneously and the simulation and synthesis processes: VHDL can be used for simulation, allowing the possibility to verify the functionality of a design before it is implemented in hardware. It can be also used for synthesis, which is the process of translating a high-level description into a physical implementation using programmable logic devices or application-specific integrated circuits.

2. OBJECTIVES

The main goals of this project are:

- **Establish a VGA connection**: it is required a successful connection between the FPGA board and the monitor so that it's possible to display all the components of the game.
- **Implement a pitch design**: a good design that allows to represent all the components (paddles, ball, borders) in the right positions.
- **Manage collisions**: a key objective is to rule the collisions between all the components: the ball and the paddles, the ball and the borders and the paddles and the borders.
- **Use the ADC**: in order to make the players able to move the paddles it is necessary an analog to digital conversion of the values provided by the two potentiometers (joysticks).

3. REQUIREMENTS

3.1 HARDWARE

3.1.1 FPGA BOARD

The Intel DE10-Lite MAX 10 FPGA board is a development platform designed for learning, prototyping, and creating FPGA-based projects. It is built around the Intel MAX 10 FPGA, which is a low-cost, low-power programmable device that has around 50K logic elements (LEs) and on-die analog-to-digital converter (ADC), on-board high-speed USB-Blaster, SDRAM, accelerometer, VGA output, and a 2x20 GPIO expansion connector, that provides a means to connect external devices and sensors for expanded functionalities.

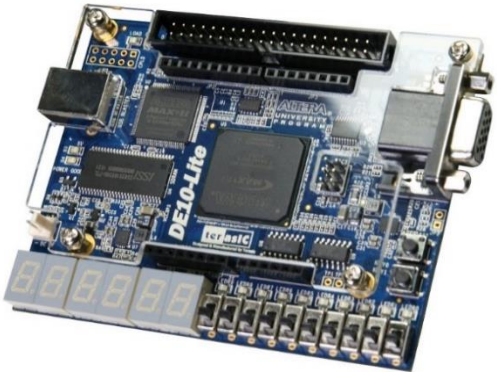


Figure 1 Intel DE10-Lite MAX

To facilitate data storage and transfer, the board incorporates 2,080 kilobits of embedded memory, used to store data within the FPGA itself. For display capabilities, the Intel DE10-Lite MAX 10 FPGA board includes a VGA output port that enables users to connect the board to standard monitors and display graphical content. The Intel DE10-Lite MAX 10 FPGA board is fully compatible with the Intel Quartus Prime design software, used to develop this project. This software provides a user-friendly environment for designing, simulating, and programming the FPGA; it also offers a wide range of design tools and features that facilitate the development process.

3.1.2 MONITOR

For this project, it has been used an ACER LCD monitor, which has 1440x900 pixels and a refresh rate of 60Hz. It has also the following specifications:

VESA Signal 1440 x 900 @ 60 Hz timing

General timing

Screen refresh rate	60 Hz
Vertical refresh	55.919117647059 kHz
Pixel freq.	106.47 MHz

Horizontal timing (line)

Polarity of horizontal sync pulse is negative.

Scanline part	Pixels	Time [µs]
Visible area	1440	13.52493660186
Front porch	80	0.75138536676998
Sync pulse	152	1.427632196863
Back porch	232	2.1790175636329
Whole line	1904	17.882971729126

Vertical timing (frame)

Polarity of vertical sync pulse is positive.

Frame part	Lines	Time [ms]
Visible area	900	16.094674556213
Front porch	1	0.017882971729126
Sync pulse	3	0.053648915187377
Back porch	28	0.50072320841552
Whole frame	932	16.666929651545

Figure 2 Monitor's parameters

3.1.2 JOYSTICKS

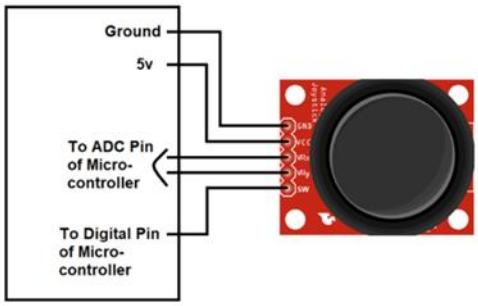


Figure 3 joystick's wirings

Two joysticks were used to control the paddles. These joysticks have an operating voltage of 5V with an internal potentiometer value of 10k. The devices features 2.54mm pin interface leads, allowing for easy connectivity, and their dimensions are of 1.57 inches x 1.02 inches x 1.26 inches (4.0 cm x 2.6 cm x 3.2 cm). Designed to operate within a temperature range of 0 to 70 °C, it ensures reliable performance even in diverse environmental conditions.

3.1.4 JUMPERS

A total of six jumpers were utilized. Two jumpers were employed to establish a connection between the +5V input pins and the VCC5 port of the FPGA, ensuring a stable power supply. Another two jumpers were used to link the GND pins to the GND port of the FPGA, enabling proper grounding.

Lastly, an additional two jumpers were utilized to connect the VRx pins, responsible for carrying resistance values, to two specific channels of the ADC, namely ADC_IN0 and ADC_IN1.

3.1.5 SEVEN-SEGMENTS DISPLAY

The DE10-Lite board incorporates six 7-segment displays, which serve the purpose of numerically representing information. In the present scenario, these displays are utilized to exhibit the game score. As per the game's rules, once a player accumulates a total of three points, the score resets to its initial state.

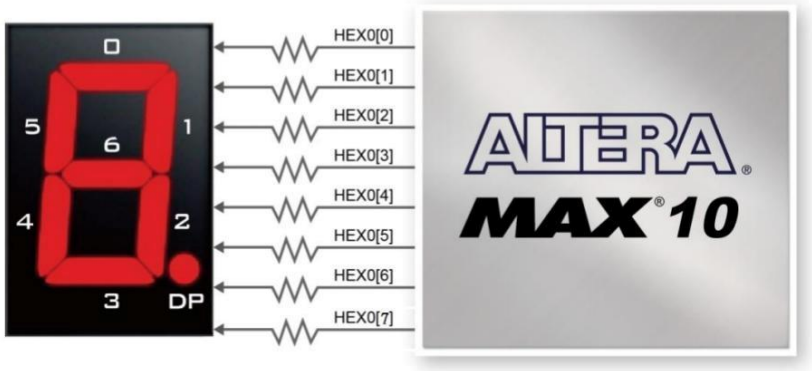


Figure 4 Seven Segments' Display Wirings

3.2 SOFTWARE

To implement the VHDL code, the Quartus Prime Lite Edition was used. It is a software tool developed by Intel (formerly Altera) that is used for designing and programming FPGA devices and CPLDs. It is part of the Quartus Prime design software suite, which provides a comprehensive environment for digital logic design, verification, synthesis, and implementation. Quartus Prime Lite Edition supports multiple design entry methods, including schematic capture, Verilog, VHDL, and SystemVerilog. The tool includes a timing analyzer that verifies the performance of the designed circuit by calculating and analyzing the critical paths and ensuring that timing requirements are met. It helps in identifying and fixing timing violations.

4. IMPLEMENTATION

4.1 VGA

To establish the VGA connection on the Intel DE10-Lite MAX 10 FPGA board, the next steps must be followed: First, an IP file is created in VHDL to define the required functionality. Then, it's necessary to set the parameters of the monitor: the frequency of the "inclk0" input, the pixel frequency, and the output clock parameters. Next, symbol files are generated for the VGA controller, "pll0", and hardware image generator. These symbols encapsulate the respective modules and their own functionalities. In the final stage is constructed a block diagram, which represent the overall system architecture. Each symbol is interconnected to establish the necessary communication and data flow among the modules and so, through this systematic approach, the VGA monitor can be effectively displayed.

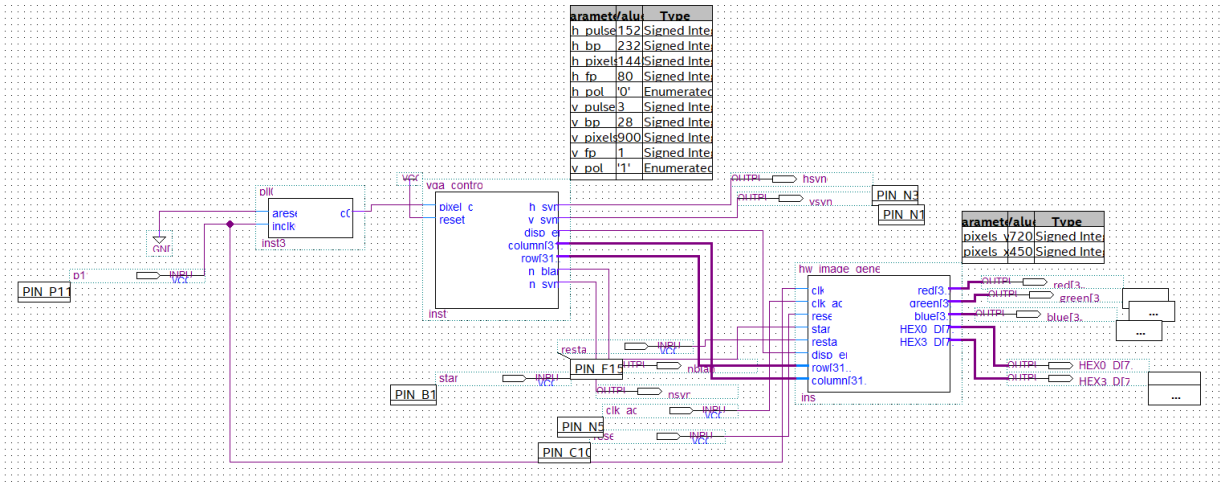


Figure 5 Block diagram

4.2 ADC

To convert the resistance value into a vector that can be utilized in the code, the implementation of the ADC intellectual property is required. Initially, it is selected the "ADC Controller for DE-series boards" IP file from the IP catalog and configured it with the appropriate parameters for this project, including the De 10-Lite board, 2 channels (one for each joystick), and VHDL language for synthesis. Therefore, the connections are activated, and the code is generated. Once the IP code has been generated, the "unnamed.qip" fil, which contain the ADC component within its hierarchy, is included in the project.

```
entity unnamed is
    port (
        CLOCK : in  std_logic                      := '0';
        CH0   : out std_logic_vector(11 downto 0);
        CH1   : out std_logic_vector(11 downto 0);
        CH2   : out std_logic_vector(11 downto 0);
        CH3   : out std_logic_vector(11 downto 0);
        CH4   : out std_logic_vector(11 downto 0);
        CH5   : out std_logic_vector(11 downto 0);
        CH6   : out std_logic_vector(11 downto 0);
        CH7   : out std_logic_vector(11 downto 0);
        RESET : in  std_logic                      := '0'
    );
end entity unnamed;
```

Figure 6 ADC's entity

The component provided by the intellectual property is used into the “hw_image_generator” file to read the joystick’s horizontal position to move the paddles:

```
adc1: unnamed port map (clk_adc,joy1, joy2, ch2, ch3, ch4, ch5, ch6, ch7, reset);
```

For this function the channels 0 and 1 are mapped to two signals, and the clock is set to 10Mhz using one of the internal clocks of the board.

4.3 CODE

4.3.1 ENTITY

```
ENTITY hw_image_generator IS
  GENERIC(
    pixels_y :  INTEGER := 720;
    pixels_x :  INTEGER := 450);
  PORT(
    clk, clk_adc, reset, start, restart: in std_logic;
    disp_ena :  IN   STD_LOGIC;
    row      :  IN   INTEGER;
    column   :  IN   INTEGER;
    red      :  OUT  STD_LOGIC_VECTOR(3 DOWNTO 0) := (OTHERS => '0');
    green    :  OUT  STD_LOGIC_VECTOR(3 DOWNTO 0) := (OTHERS => '0');
    blue     :  OUT  STD_LOGIC_VECTOR(3 DOWNTO 0) := (OTHERS => '0');
    HEX0_D : out std_logic_vector(7 downto 0);
    HEX3_D : out std_logic_vector(7 downto 0));
END hw_image_generator;
```

Figure 7 hw_image_generator's entity

In the first part of the code, an entity called "hw_image_generator" is created, within which 2 parameters “pixels_x” and “pixels_y” (initialized to 450 and 720 respectively i.e. the center of the

screen) are declared as generics. Within the entity, the input and output ports used are also declared, and in this case 3 types can be distinguished:

- **STD_LOGIC**: represent 1 single bit which can be worth either 0 or 1;
- **INTEGER**: indicates an integer value;
- **STD_LOGIC_VECTOR**: are binary vectors; in this case binary vectors of 4 bits (3 DOWNTO 0) and vectors of 8 (7 DOWNTO 0) are declared.

“Clk”, “clk_adc” are clock signals and are declared as std_logic; they are the clock signals of the FPGA board and correspond to 50Mhz (PIN_P11) and 10Mhz (PIN_N5) respectively.

“Reset”, “disp_ena”, “start”, and “restart” are switches declared as std_logic. The first two refer to the ADC component and to the display respectively, while “start” and “restart” are switches mapped to the board (PIN_B14, PIN_F15).

Row and column are integers that represent the coordinates in pixels within the screen. Red, green and blue are output std_logic_vectors whose, depending on their values, print the desired color shade on the screen.

red[3]	Output	PIN_Y1	3	B3_N0	PIN_Y1	2.5 V		12mA ...ault)	2 (default)
red[2]	Output	PIN_Y2	3	B3_N0	PIN_Y2	2.5 V		12mA ...ault)	2 (default)
red[1]	Output	PIN_V1	2	B2_N0	PIN_V1	2.5 V		12mA ...ault)	2 (default)
red[0]	Output	PIN_AA1	3	B3_N0	PIN_AA1	2.5 V		12mA ...ault)	2 (default)
green[3]	Output	PIN_R1	2	B2_N0	PIN_R1	2.5 V		12mA ...ault)	2 (default)
green[2]	Output	PIN_R2	2	B2_N0	PIN_R2	2.5 V		12mA ...ault)	2 (default)
green[1]	Output	PIN_T2	2	B2_N0	PIN_T2	2.5 V		12mA ...ault)	2 (default)
green[0]	Output	PIN_W1	2	B2_N0	PIN_W1	2.5 V		12mA ...ault)	2 (default)
blue[3]	Output	PIN_N2	2	B2_N0	PIN_N2	2.5 V		12mA ...ault)	2 (default)
blue[2]	Output	PIN_P4	2	B2_N0	PIN_P4	2.5 V		12mA ...ault)	2 (default)
blue[1]	Output	PIN_T1	2	B2_N0	PIN_T1	2.5 V		12mA ...ault)	2 (default)
blue[0]	Output	PIN_P1	2	B2_N0	PIN_P1	2.5 V		12mA ...ault)	2 (default)
HEX0_D[7]	Output	PIN_D15	7	B7_N0	PIN_D15	2.5 V		12mA ...ault)	2 (default)
HEX0_D[6]	Output	PIN_C17	7	B7_N0	PIN_C17	2.5 V		12mA ...ault)	2 (default)
HEX0_D[5]	Output	PIN_D17	7	B7_N0	PIN_D17	2.5 V		12mA ...ault)	2 (default)
HEX0_D[4]	Output	PIN_E16	7	B7_N0	PIN_E16	2.5 V		12mA ...ault)	2 (default)
HEX0_D[3]	Output	PIN_C16	7	B7_N0	PIN_C16	2.5 V		12mA ...ault)	2 (default)
HEX0_D[2]	Output	PIN_C15	7	B7_N0	PIN_C15	2.5 V		12mA ...ault)	2 (default)
HEX0_D[1]	Output	PIN_E15	7	B7_N0	PIN_E15	2.5 V		12mA ...ault)	2 (default)
HEX0_D[0]	Output	PIN_C14	7	B7_N0	PIN_C14	2.5 V		12mA ...ault)	2 (default)
HEX3_D[7]	Output	PIN_A19	7	B7_N0	PIN_A19	2.5 V		12mA ...ault)	2 (default)
HEX3_D[6]	Output	PIN_B22	6	B6_N0	PIN_B22	2.5 V		12mA ...ault)	2 (default)
HEX3_D[5]	Output	PIN_C22	6	B6_N0	PIN_C22	2.5 V		12mA ...ault)	2 (default)
HEX3_D[4]	Output	PIN_B21	6	B6_N0	PIN_B21	2.5 V		12mA ...ault)	2 (default)
HEX3_D[3]	Output	PIN_A21	6	B6_N0	PIN_A21	2.5 V		12mA ...ault)	2 (default)
HEX3_D[2]	Output	PIN_B19	7	B7_N0	PIN_B19	2.5 V		12mA ...ault)	2 (default)
HEX3_D[1]	Output	PIN_A20	7	B7_N0	PIN_A20	2.5 V		12mA ...ault)	2 (default)
HEX3_D[0]	Output	PIN_B20	6	B6_N0	PIN_B20	2.5 V		12mA ...ault)	2 (default)

Figure 8 Pin planner

The last declared ports, “HEX0_D” and “HEX3_D”, are 8-bit std_logic_vector; they are used for displaying the actual score on the seven-segment display.

4.3.2 ARCHITECTURE

Within the architecture section there is the core part of the code. At the beginning of this section are declared and initialized all the signals necessary to implement the entire code.

```
ARCHITECTURE behavior OF hw_image_generator IS
signal x1: integer:=pixels_x;
signal y1:integer:= 20;
signal l: integer:= 15;
signal h: integer:=70;
signal x2: integer:=pixels_x;
signal y2:integer:= 1420;
signal xp: integer:=pixels_x;
signal yp:integer:= pixels_y;
signal r: integer:=10;
signal clk_i, rst: std_logic:='0';
signal counter: integer:=1;
signal vx: integer:=5;
signal vy: integer:=5;
signal vr1: integer:=7;
signal vr2: integer:=7;
signal pt1, pt2: integer range 0 to 9:=0;
signal joy1, joy2, ch2, ch3, ch4, ch5, ch6, ch7: std_logic_vector(11 downto 0);
```

Figure 9 hw_image_generator's architecture

The “x1”, “y1”, “x2”, “y2”, “xp” and “yp” signals represent the position of the left paddle (“x1”, “y1”), the right paddle (“x2”, “y2”) and the ball (“xp”, “yp”), respectively. Then, there are two std_logic signals: “clk_i” which is used as an internal clock (slower than the clock signal coming from the board) and “rst” which is used as a support signal related to the end-of-game reset. In addition, “l” and “h” are half the length and half the height of the paddles, “r” is the radius of the ball and “counter” is a signal used in the processs used to implement the 100Hz clock divider, which produces a clock signal suitable for the use case.

```
process  (clk)
        begin
            if rising_edge(clk) then
                counter<=counter+1;
                if counter= 500000 then
                    clk_i<= not clk_i;
                    counter<=1;
                end if;
            end if;
    end process;
```

Figure 10 Clock divider

So in this process “vx” and “vy” are the velocities of the ball, “vr1” is the velocity of the left paddle, “vr2” is the velocity of the right paddle, and lastly “pt1” and “pt2” are signals used to mark the current score. The last signals declared are: “joy1”, “joy2”, “ch2”, “ch3”, “ch4”, “ch5”, “ch6”, “ch7”; they are std_logic_vectors composed of 12 bits which refer to the component called unnamed that allows the use of an ADC controller for moving the paddles. In this case only the first 2 channels of the ADC are used.

4.3.3 PADDLES MOVEMENT

```
process(joy1, clk_i)
begin
if rising_edge(clk_i) then
if (x1-h>20 and x1+h<880) then
   if (joy1 <="110111111111" and joy1>="000111111111") then
   x1<=x1;
   vr1<=0;
   elsif (joy1 >"110111111111")
   then x1<=x1-7;
   vr1<=-7;
   elsif (joy1<"000111111111")
   then x1<=x1+7;
   vr1<=+7;
   end if;
   elsif (x1-h<=20) then x1<=x1+1;
   elsif (x1+h>=880) then x1<=x1-1;
   end if;
   end if;
end process;
```

Figure 11 Left paddle's movement

This process describes the behavior of the left paddle in relation to the input provided by the joystick. To achieve a proper usage of the joystick, some tolerances are defined: as the joystick can produce a value between 0 and 4095 to map his horizontal position, it has been established that for values less than 511 (000111111111 in binary) the position to read is right, while for values upper than 3583 (110111111111 in binary) the position to read is left. In the range between these two values, the position of the joystick to read must be the central one. The process is defined with sensitivity to the signals "joy1" and "clk_i"; inside the process, the code is enclosed within an if statement, ensuring that the logic is only executed on the rising edge of the clock signal. The next condition checks if the difference between the central position of the paddle and half the height of the paddle is greater than 20 and in the meantime the sum between them is less than 880. This condition verifies that the paddle is within the game boundaries to prevent it from moving off the screen. If the paddle is within the game boundaries, the next condition checks the value of the "joy1" signal against the range from "000111111111" to "110111111111". This range represents joystick values that indicate the paddle should not move; if the joystick value falls in this range, the paddle's position ("x1") remains unchanged and its velocity ("vr1") is set to zero. Instead, if the joystick value is greater than "110111111111", that indicates the movement in the left direction, the paddle's position is decremented by 7 and its velocity is set to -7, and again if the joystick value is less than "000111111111", that indicates the movement in the right direction, the paddle's position is incremented by 7 and its velocity is set to +7. The end if statement closes the condition related to the joystick values. Now, two if statements are used as a security measure in case the paddle hits the borders: when this happens, the central position is decremented by 1 if the paddle hits the upper board and it is incremented by 1 if the paddle hits the lower board, so that paddle and boards can never touch themselves.

```
process(joy2, clk_i)
begin
if rising_edge(clk_i) then
if (x2-h>20 and x2+h<880) then
   if (joy2 <="110111111111" and joy2>="000111111111") then
   x2<=x2;
   vr2<=0;
   elsif (joy2 >"110111111111")
   then x2<=x2-7;
   vr2<=-7;
   elsif (joy2<"000111111111")
   then x2<=x2+7;
   vr2<=+7;
   end if;
   elsif (x2-h<=20) then x2<=x2+1;
   elsif (x2+h>=880) then x2<=x2-1;
   end if;
   end if;
end process;
```

Figure 12 Right paddle's movement

The logic used to control the behavior of the right paddle is the same. Obviously in this case it is required to use the "joy2" signal so that the other player can use another joystick to move his paddle. For the same reason, in this process are also used the "x2" and "v2" signals.

4.3.4 BALL MOVEMENT AND COLLISIONS

```
process(clk_i,rst, start, restart)
begin

if(rising_edge(clk_i)and(start='1')) then
   if ((rst='1')and(restart='1')) then
      vx<=5;
      vy<=5;
      pt1<=0;
      pt2<=0;
      rst<='0';
    else
   xp<=xp+vy;
   yp<=yp+vx;
```

Figure 13 Ball's position

This process is employed to calculate the position of the ball and manage its collisions with the field's boundaries and the paddles at each clock cycle, with the same clock used to determine the paddles' position. The sensitivity list of this process is composed by the internal clock defined in its own process, the reset, start and restart signals. Initially, a check is done on the start signal: the player decides, using a switch, when to start the game, and the same signal is also used for pausing and resuming the game. Once the conditions are met (and the game is started), a sequence of nested if clauses is implemented within the initial condition. The following if statement verifies whether the game has ended or not (the "rst" signal, pulled up if one of the two players has reached a score of 3 points and scores another point) and if the players desire to play a new match. If this does happen, the ball's position and speed's components as well as the players' scores are reset to their initial values, otherwise the ball proceeds with its movement, increasing the position along the x and y axes.

```
if(xp+r<=880 and xp-r>=20) then
   if(yp-r=35)and(xp<x1+h and xp>x1-h) then
   vy<=vy+vr1;
   vx<=-vx;
   yp<=yp+3*abs(vx);
   elsif(yp+r=1405)and(xp<x2+h and xp>x2-h) then
   vy<=vy+vr2;
   vx<=-vx;
   yp<=yp-3*abs(vx);
```

Figure 14 Ball's collision with paddles

If neither player has won yet, the ball moves within the field and a series of check on its position determine its interaction with the various elements present on the screen. In the first place the vertical playing space is fixed, the ball can move freely, without colliding with the borders if its rows number (in this case indicated by the signal “xp”) is between 20 and 880 pixels (the field vertical borders). Then, if the ball is in this space, it is checked if it does touch one of the two paddles. If the ball is located inside the area assigned to one of the paddles, its horizontal speed is inverted (due to collision), the paddle's velocity is added to the ball's vertical velocity (to better simulate the collision on the vertical axe) and the horizontal position is summed to three times its speed. This adjustment is necessary due to the update frequency (determined by "clk_i"); otherwise, the collision would still be detected by the logic, but the ball would move beyond the paddle, which means that a point is being scored.

Inside the same if clause that delimits the playing vertical space, it’s checked if one of the player has scored a point evaluating if the ball is not within the vertical area of the paddles and if it overcomes the horizontal position of one of the two paddles. In this case the other player gets a point, and if its score is less than 3, the game continues. Therefore, the ball is set at the center of the field, and it is directed toward the player who just scored. The verification is done using the “elsif” clause since it’s required that the ball remains in the playing ground, but it is not in the paddle’s space. Then the check on the vertical borders is closed and with an else statement it is taken care of the collision with borders.

```
elsif(yp-r<20) then
if(pt2<3) then
pt2<=pt2+1;
xp<=pixels_x;
yp<=pixels_y;
vx<=5;
vy<=5;
else rst<='1';
xp<=pixels_x;
yp<=pixels_y;
      vx<=0;
      vy<=0;
end if;
elsif(yp+r>1420)then
if(pt1<3) then
pt1<=pt1+1;
xp<=pixels_x;
yp<=pixels_y;
vx<=-5;
vy<=5;
else rst<='1';
      xp<=pixels_x;
      yp<=pixels_y;
      vx<=0;
      vy<=0;
end if;
end if;
```

Figure 15 Point assignment

```
else
    vy<=-vy;
    if(xp-r<=20) then xp<=xp+abs(vy);
    elsif(xp+r>=880) then xp<=xp-abs(vy);
    end if;
    end if;
    end if;
end if;
    end process;
```

Figure 16 Ball's collision with borders

The control is like the one done with the paddles, then the ball's vertical speed is inverted, and the vertical position is increased (or decreased if the ball touches the lower border) by the absolute value of its speed.

4.3.5 SEVEN SEGMENTS DISPLAY

The scoreboard is visualized to the players with two of the six seven segments displays provided by the De10-Lite board. It is implemented the display's logic in its dedicated file, and it is mapped into the hw_image_generator file to represent the score, using the "pt1" and "pt2" integer signals as inputs and the "HEX3_D" and "HEX0_D" vectors as outputs.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity seven_segment is
   port( input: in integer range 0 to 9;
         seg_out: out std_logic_vector (7 downto 0));
end seven_segment;

architecture sev_seg of seven_segment is
signal yt: std_logic_vector(7 downto 0);
   begin
         process (input)
            begin
                case input is
                when 0 => yt <= "00111111";
                when 1 => yt <= "00000110";
                when 2 => yt <= "01011011";
                when 3 => yt <= "01001111";
                when 4 => yt <= "01100110";
                when 5 => yt <= "01101101";
                when 6 => yt <= "01111101";
                when 7 => yt <= "00000111";
                when 8 => yt <= "01111111";
                when others =>yt <= "01110111";
                end case;
             end process;
             seg_out<= not yt;
          end sev_seg;
```

Figure 17 Seven segments display architecture

```
component seven_segment is port(
 input: in integer range 0 to 9;
 seg_out: out std_logic_vector (7 downto 0)
 );
 end component seven_segment;
```

Figure 18 Seven segments display component in hw_img_generator

```
begin
    left_player_score_display  : seven_segment port map (pt1, HEX3_D);
    right_player_score_display : seven_segment port map (pt2, HEX0_D);
```

Figure 19 Seven segments display port maps

4.3.6 PITCH DESIGN

The following part of the code involves the drawing of the field on a display screen. The code defines a process named “campo” which takes inputs such as “disp_ena” (display enable), “row” and “column” (coordinates), and “start” (start signal).

```
campo: process (disp_ena, row, column, start)
begin
if(disp_ena = '1') then

if (row<=20 or row>=880) then
      red <= (OTHERS => '1');
     green  <= (OTHERS => '1');
     blue <= (OTHERS => '1');

      elsif (abs(row-xp)<r) and (abs(column-yp)<r) then
         red <= (OTHERS => '1');
         green  <= (OTHERS => '0');
         blue <= (OTHERS => '0');

      elsif ((column<=715 and column>=705)and(row>=430 and row<=470))then
      if (start='0')then
            red <= (OTHERS => '1');
            green  <= (OTHERS => '1');
            blue <= (OTHERS => '1');
      end if;
      elsif ((column<=735 and column>=725)and(row>=430 and row<=470))then
      if (start='0')then
            red <= (OTHERS => '1');
            green  <= (OTHERS => '1');
            blue <= (OTHERS => '1');
      end if;
```

Figure 20 Pitch design code

The process “campo” is sensitive to changes in the signals “disp_ena”, “row”, “column” and “start”. If “disp_ena” is enabled, the following conditions are evaluated. If the “row” value is less than or equal to 20 or greater than or equal to 880, the border of the field is drawn. The RGB values (red, green and blue) are set to ‘1’ for all pixels in the border, resulting in a white color. If the absolute difference between the “row” and “xp” coordinates is less than “r” and the absolute difference between the “column” and “yp” coordinates is less than “r”, a red ball is drawn. The RGB values are set to (‘1’,’0’,’0’) for all pixels within the ball’s radius, resulting in a red color. If the “column” value is between 705 and 715 and the “row” value is between 430 and 470, a pause area is defined. If the “start” signal is ‘0’, the RGB values are set to ‘1’ for all pixels within the pause area, resulting in a white color. Similarly, if the “column” value is between 725 and 735 and “row” value is between 430 and 470, another pause area is defined with the same behavior.

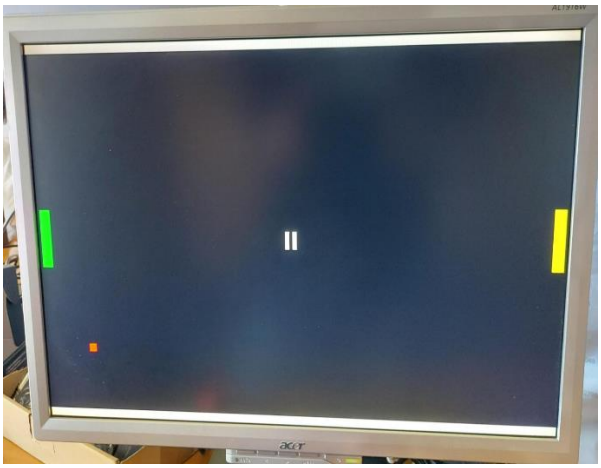
```
        elsif(abs(row-x1)<h AND abs(column-y1)<l) THEN
           red <= (OTHERS => '0');
           green  <= (OTHERS => '1');
           blue <= (OTHERS => '0');
        elsif(abs(row-x2)<h AND abs(column-y2)<l) THEN
           red <= (OTHERS => '1');
           green  <= (OTHERS => '1');
           blue <= (OTHERS => '0');
        else
           red <= (OTHERS => '0');
           green  <= (OTHERS => '0');
           blue <= (OTHERS => '0');
        end if;
end if;
end process campo;
```

Figure 21 Paddles' drawing

This part of the code is responsible for drawing two paddles or rackets on the display screen. The first "elsif" statement checks if the absolute difference between the "row" and "x1" coordinates is less than "h" (half height) and the absolute difference between the "column" and "y1" coordinates is less than "l" (half length). If this condition is true, it means the current pixel belongs to paddle 1. In this case, the red component of the RGB color is set to '0, indicating no red color. The green component is set to '1', indicating a full green color. The blue component is set to '0', indicating no blue color. The second 'elsif' statement checks if the absolute difference between the "row" and "x2" coordinates il less than "h" and the absolute difference between the "column" and "y2" coordinates is less than "l". if this condition is true, it means the current pixel belong to paddle 2. In this case, the red component of the RGB color is set to '1' (on), indicating a full red color. The green component is set to '1', indicating a full green color. The blue component is set to '0' (off), indicating no blue color. If none of the above conditions are true, the 'else' block is executed, indicating that the current pixel does not belong to any paddle. In this case, all RGB components are set to '0' (off), resulting in a black color. The overall effect of this code is that it determines the color of each pixel on the display screen based on its position relative to the coordinates of the two paddles. Pixels within the boundaries of paddle 1 are displayed as green, paddle 2 as yellow (a combination of red and green), and all other pixels are black.

5. CONCLUSIONS

The implementation of the Pong game using the VHDL language, and the FPGA board has reached successful results. The game functioned as intended, providing an engaging game experience. The ball movement, paddle controls, collision detection, the pause and restart, and scoring mechanisms were all accurately implemented and responsive. The game exhibited smooth gameplay with minimal latency, thanks to the efficient hardware design and FPGA optimizations; it also responded properly to user input, allowing players to control the paddles with precision and the scoring system accurately tracked points for both players, updating the seven segments display in real time. Therefore, the game met all the main initial goals, including a functional use of external components like ADC and VGA.



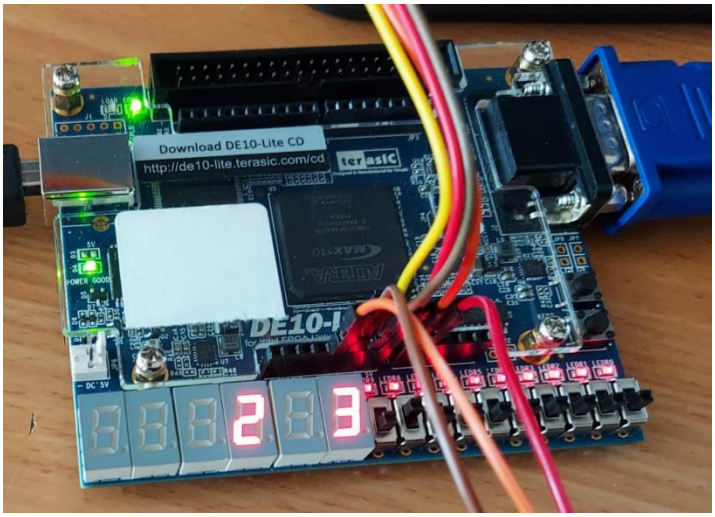


Figure 22 Final results

5.1 DIFFICULTIES IN THE DEVELOPMENT

The difficulties encountered during the development of the game were several. One problem concerned about the connections of the two joysticks for the movement of the paddles arose, in fact using the above-described procedure to use the ADC had not been configured to read two channels but initially only one; so, at the beginning, one of the two joysticks was not being read.

Another difficulty encountered was in the collision-related portion of the code. In particular, it was noted that due to the frequency of the display not being exactly the same as that defined in the code, there was a need to balance well the paddle speeds with the ball speeds: with a very fast ball movement there were problems in collisions with both paddles and the borders of the field; this problem was solved not only by balancing the velocities, but also by adding to the position of the ball a precise number of pixels depending on whether the collision occurs on either paddle or on the banks of the court.

5.2 FUTURE DEVELOPMENTS

The project exhibits promising potential for further developments and expansions in various ways:

- The players' score can be displayed at the center of the screen, similarly to the original game, utilizing a design resembling a seven-segment display or employing a distinct font style.
- Sounds can be incorporated to further enrich the user experience, for instance the ball colliding on the borders and paddles or sound feedback indicating a point being scored.
- A function can be implemented to generate two random values, which can be used to set the ball's initial velocities, introducing an element of unpredictability and variability to gameplay.
- Add visual traces following the ball's movement when it reaches a certain value of velocity to enhance the perception of speed and dynamism.
- On-screen visual indicators for "Pause" and "Start" can be integrated, denoting the corresponding game states, and providing intuitive user guidance.
- The paddles can be designed with curved edges, allowing collisions with the ball in these areas to create unpredictable effects for the opponent player.
- An additional enhancement could involve the addition of a central dashed line representing the midfield, providing a more accurate field design.

By exploring these potential routes for further development, the project can be elevated to a higher level of sophistication and user engagement, offering an enhanced and immersive gaming experience.