Name: Ezgi

Surname: Demir

ID: 22103304

EEE102 TERM PROJECT

Purpose:

The objective of this project is to create a mini-basketball game using the BASYS3 development board. The game will utilize an IR (infrared) sensor to detect successful shots, and the points scored will be displayed on BASYS3's seven-segment display.

Design Specification:

Here are the inputs and the outputs of my top-module. Also, you can see my modules in Figure 2.

1. **Inputs:**
    - clk (std_logic): Clock signal used for synchronization.
    - reset (std_logic): Reset signal for initializing the system.
    - IR (std_logic): Input signal from the IR sensor indicating the detection of a successful shot.
2. **Outputs:**
    - sev_seg (std_logic_vector(6 downto 0)): Output signal for displaying the score on the seven-segment display.
    - anode_sel (std_logic_vector(3 downto 0)): Output signal for selecting the active digit on the seven-segment display.

Also, as a sub-module I used a clock divider, The clock division is achieved using a counter mechanism. When the reset signal is high, the match1 signal and the count_sig counter are both resets to their initial values. During the positive edge of the clk_in signal, the counter is incremented. If the counter reaches the specified value, the match1 signal is updated according to a case statement. The counter is then reset to 0. This creates a divided clock signal with a frequency that is a fraction of the input clock frequency. In the basketball game project, I utilized a clock divider to manage the timing and synchronization of game components. By dividing the input clock frequency, a lower frequency clock signal was generated. This enabled precise coordination of the scoring mechanism, display updates, and IR sensor input processing. The clock divider ensured seamless operations, accurate point counting, and timely display updates. Its role in generating a slower clock signal facilitated smooth and synchronized gameplay.

Furthermore, In the seven_segment sub-module, I worked with hexadecimal numbers to show them on my BASYS display.

Methodology:

1. First, I have written my top-module main.vhd , inputs and outputs.
2. Then, I worked on my clkdivider.vhd sub-module.
3. After that, I began writing a code for the ultrasonic sensor without specifying which specific sensor I would use in my project. However, I encountered difficulties when trying to convert my distance display code into a score counter. Additionally, I faced issues with the trig and echo input and output. Consequently, I decided to change the sensor to the IR sensor (Figure 6), which would be more suitable for writing the code and integrating it into my basketball score-counting algorithm.
4. Then, the seven_segment sub-module was written and all of these modules and other sub-modules were also brought together.
5. Finally, I ran my code, programmed my board, and made sensor connections with jumpers and breadboard. I used a sponge ball in order not to damage my sensor and also the hoop. Once I threw the ball through the basketball hoop the sensor which is connected to the breadboard under the hoop detects the balls. I adjusted my sensor the way that it detects objects under 5cm, I have adjusted this value considering the diameter of my hoop, therefore, it won't see any objects passing near the hoop or see me. I also control, if my reset button works properly (Figure 1 and 5). It resets the scoreboard on the seven-segment display hence, players can start again for example after the time runs out and compete with each other. That's why the reset button is crucial.
6. In my proposal I mentioned I could improve my project if I had adequate time however, I had a problem with the sensors and it took my time very much. Moreover, I couldn't move further since I have written I could maybe try the further steps in my proposal.

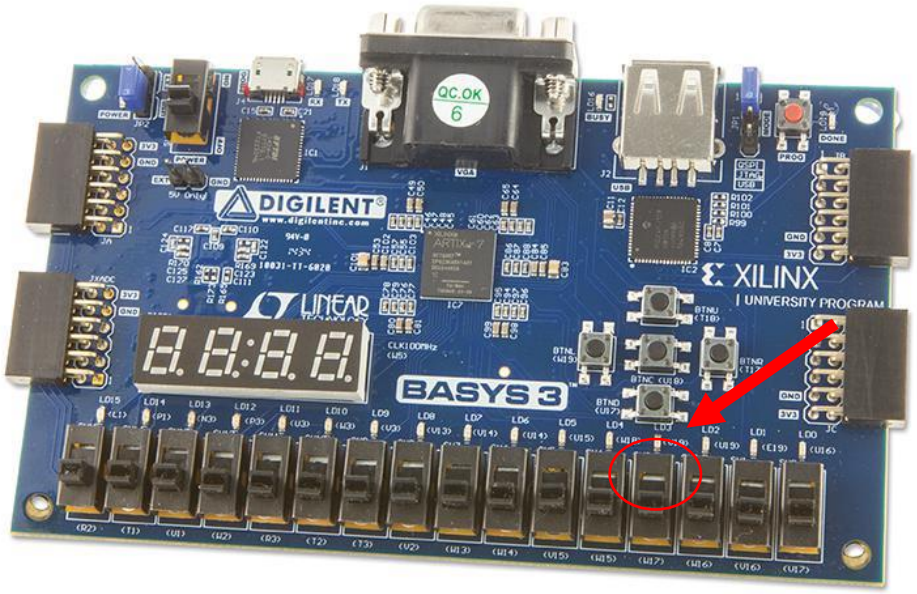

Figure 1: Reset button selected

Results:

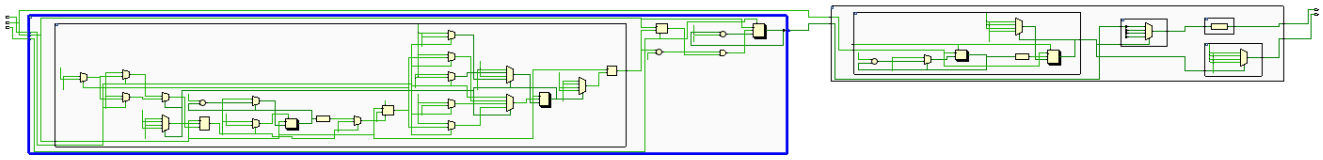

Figure 2: Elaborated Design of the code

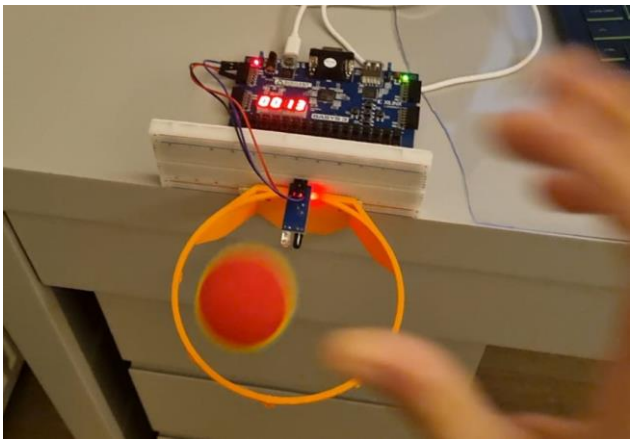

Figure 3: Ball, before passing the hoop

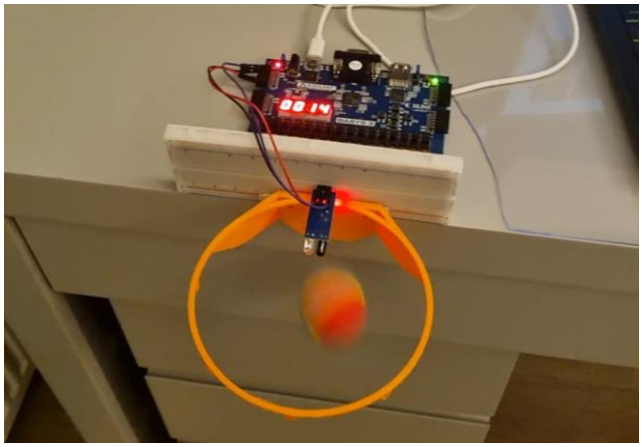

Figure 4: Ball, after passing the hoop

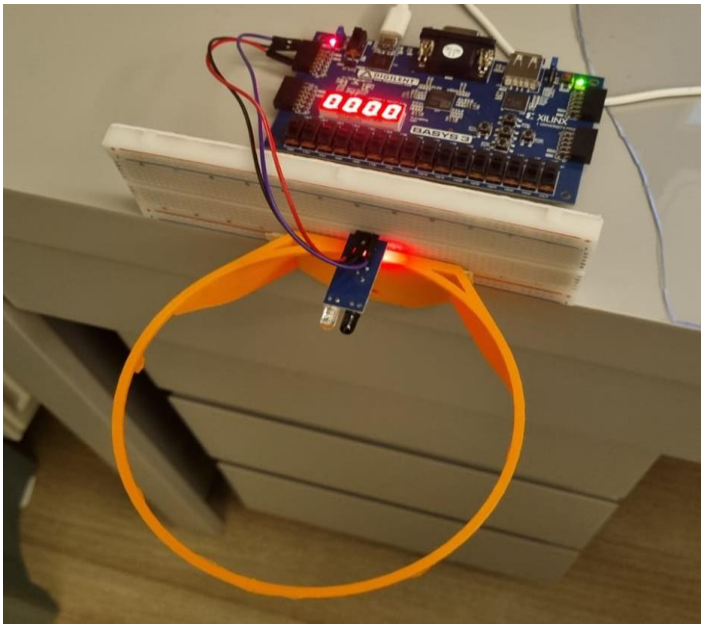

Figure 5: After pushing the "reset" button

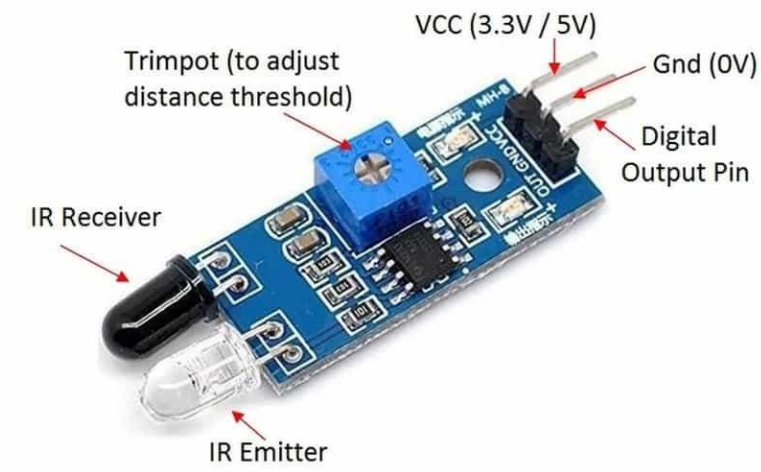

Figure 6: IR sensor

Conclusion:

In conclusion, this project aimed to develop a mini-basketball game using the BASYS3 development board and an IR sensor for shot detection. The integration of modules such as the clock divider ensured precise timing and synchronization for smooth gameplay. Despite initial challenges with the ultrasonic sensor, the switch to the IR sensor enabled the successful implementation of the score-counting algorithm. While time constraints limited further improvements, the project demonstrated the functionality of the game and showcased the potential for future enhancements. Overall, the project successfully showcased the integration of hardware components and coding to create an engaging mini-basketball game.

Youtube video link:

https://www.youtube.com/watch?v=HXj6WfAwCEI

References:

- https://how2electronics.com/bidirectional-visitor-counter-with-automatic-light-control-using-arduino/
- EEE102 2023 Spring Semester Lecture Notes by Volkan Kurşun
- https://www.realdigital.org/doc/9c21eab4a0f85c50486858a87380d1f6

Appendices:

main.vhd

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



entity Top is
Port (clk : in std_logic;
      reset : in std_logic;
      IR : in std_logic;
      sev_seg : out std_logic_vector (6 downto 0);
      anode_sel : out std_logic_vector (3 downto 0));

end Top;

architecture Behavioral of Top is

component IR_top
Port (clk : in std_logic;
      reset : in std_logic;
      IR : in std_logic;
      counterout : out std_logic_vector(15 downto 0));
end component;

component main_sev_seg
Port ( Counter_out : in STD_LOGIC_VECTOR (15 downto 0);
       Clk_In : in STD_LOGIC;
       Seven_segments : out STD_LOGIC_VECTOR (6 downto 0);
       anod_sel : out STD_LOGIC_VECTOR (3 downto 0));
```

```
end component;


signal count_out : STD_LOGIC_VECTOR(15 downto 0);

begin

UUT1 : IR_top
   port map (clk => clk,
             reset => reset,
             IR => IR,
             counterout => count_out);

UUT2 : main_sev_seg
   port map (Counter_out => count_out,
             Clk_In => clk,
             Seven_segments => sev_seg,
             anod_sel => anode_sel);



end Behavioral;
```

IR.vhd

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

entity IR_top is
Port (clk : in std_logic;
```

```
    reset : in std_logic;
    IR : in std_logic;
    counterout : out std_logic_vector(15 downto 0));
end IR_top;

architecture Behavioral of IR_top is

component sensor_debounce is
   Generic (
   clk_freq: integer := 100_000_000;
   debounce_freq: integer := 10000);
   Port (
   clk: in std_logic;
   sensor_in: in std_logic;
   sensor_out: out std_logic
   );
end component;


signal counter : integer := 0;
signal Debounce_IR: std_logic;
begin

Debounce: sensor_debounce generic map(debounce_freq => 100000)
port map( clk => clk, sensor_in => IR, sensor_out => Debounce_IR );

process(clk, reset)
   variable IR_prev : std_logic := '1';  -- Initialize previous IR value to '1'
begin
```

```
    if reset = '1' then
        counter <= 0;
        IR_prev := '1';  -- Reset the previous IR value

    elsif rising_edge(clk) then
        if Debounce_IR = '0' and IR_prev = '1' then
            counter <= counter + 1;
        end if;

        IR_prev := Debounce_IR;  -- Store the current IR value for the next clock cycle
    end if;
end process;

counterout <= std_logic_vector(to_unsigned(counter, 16));

end Behavioral;
```

clkdivider.vhd

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

entity point_clk_divider is
 Port ( clk_in : in STD_LOGIC;
 Reset : in STD_LOGIC;
 clk_out : out STD_LOGIC_VECTOR (1 downto 0));
end point_clk_divider;

architecture Behavioral of point_clk_divider is
```

```
SIGNAL match1 : STD_LOGIC_VECTOR (1 DOWNTO 0);
SIGNAL count_sig : INTEGER RANGE 0 TO 6249 :=0;

begin
PROCESS (Reset, clk_in)
BEGIN
IF( Reset = '1') THEN
 match1 <= "00";
 count_sig <= 0;
ELSIF RISING_EDGE(clk_in) THEN
 IF(count_sig = 6249) THEN
 CASE match1 IS
 WHEN "00" => match1 <= "01";
 WHEN "01" => match1 <= "10";
 WHEN "10" => match1 <= "11";
 WHEN others => match1 <= "00";
 END CASE;
 count_sig <= 0;
 ELSE
 count_sig <= count_sig + 1;
 END IF;
END IF;
END PROCESS;
clk_out <= match1;
end Behavioral;
```

seven.seg.vhd

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

entity main_sev_seg is
 Port ( Counter_out : in STD_LOGIC_VECTOR (15 downto 0);
       Clk_In : in STD_LOGIC;
       Seven_segments : out STD_LOGIC_VECTOR (6 downto 0);
       anod_sel : out STD_LOGIC_VECTOR (3 downto 0));
end main_sev_seg;

architecture Behavioral of main_sev_seg is

COMPONENT multiplexer
 PORT ( Counter_out : in STD_LOGIC_VECTOR (15 downto 0);
 sev_sel : in STD_LOGIC_VECTOR (1 downto 0);
 mux_out : out STD_LOGIC_VECTOR (3 downto 0));
 END COMPONENT;

COMPONENT seven_segment
 PORT ( out_of_mux : in STD_LOGIC_VECTOR (3 downto 0);
 seven_seg : out STD_LOGIC_VECTOR (6 downto 0));
END COMPONENT;

COMPONENT point_clk_divider
 PORT ( clk_in : in STD_LOGIC;
 Reset : in STD_LOGIC;
```

```
 clk_out : out STD_LOGIC_VECTOR (1 downto 0));
END COMPONENT;

COMPONENT inv_decoder
 PORT ( input : in STD_LOGIC_VECTOR (1 downto 0);
 output : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

SIGNAL temp_dig : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL temp_reset : STD_LOGIC;
SIGNAL temp_clk : STD_LOGIC_VECTOR (1 DOWNTO 0);

begin

OPP1 : multiplexer
 PORT MAP (Counter_out => Counter_out,
        sev_sel (1 DOWNTO 0) => temp_clk (1 DOWNTO 0),
        mux_out (3 DOWNTO 0) => temp_dig (3 DOWNTO 0));

OPP2 : seven_segment
 PORT MAP (out_of_mux (3 DOWNTO 0) => temp_dig (3 DOWNTO 0),
        seven_seg (6 DOWNTO 0) => Seven_segments (6 DOWNTO 0));

OPP3 : point_clk_divider
 PORT MAP (clk_in => Clk_In,
 Reset => temp_reset,
 clk_out (1 DOWNTO 0) => temp_clk (1 DOWNTO 0));

OPP4 : inv_decoder
```

```
 PORT MAP (input (1 DOWNTO 0) => temp_clk (1 DOWNTO 0),
 output (3 DOWNTO 0) => anod_sel (3 DOWNTO 0));

end Behavioral;
```

seven_segment.vhd

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

entity seven_segment is
 Port ( out_of_mux : in STD_LOGIC_VECTOR (3 downto 0);
       seven_seg : out STD_LOGIC_VECTOR (6 downto 0));
end seven_segment;

architecture Behavioral of seven_segment is

begin

PROCESS(out_of_mux)
BEGIN
CASE out_of_mux IS
 WHEN "0000" => seven_seg <= "1000000"; -- 0
 WHEN "0001" => seven_seg <= "1111001"; -- 1
 WHEN "0010" => seven_seg <= "0100100"; -- 2
 WHEN "0011" => seven_seg <= "0110000"; -- 3
 WHEN "0100" => seven_seg <= "0011001"; -- 4
 WHEN "0101" => seven_seg <= "0010010"; -- 5
```

```
 WHEN "0110" => seven_seg <= "0000010"; -- 6
 WHEN "0111" => seven_seg <= "1111000"; -- 7
 WHEN "1000" => seven_seg <= "0000000"; -- 8
 WHEN "1001" => seven_seg <= "0011000"; -- 9
 WHEN "1010" => seven_seg <= "0001000"; -- 10 -> A
 WHEN "1011" => seven_seg <= "0000011"; -- 11 -> b
 WHEN "1100" => seven_seg <= "1000110"; -- 12 -> C
 WHEN "1101" => seven_seg <= "0100001"; -- 13 -> d
 WHEN "1110" => seven_seg <= "0000110"; -- 14 -> E
 WHEN others => seven_seg <= "0001110"; -- 15 -> F
END CASE;
END PROCESS;

end Behavioral;
```

return.vhd

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

entity sensor_debounce is
    Generic (
    clk_freq: integer := 100_000_000;
    debounce_freq: integer := 10000);
    Port (
    clk: in std_logic;
    sensor_in: in std_logic;
    sensor_out: out std_logic
    );
end sensor_debounce;
```

```
architecture Behavioral of sensor_debounce is

constant timer_limit: integer := clk_freq/debounce_freq;

type debounce_state is (ZERO, RISE, ONE, FALL);
signal state: debounce_state := ZERO;
signal timer_enable: std_logic := '0';
signal timer_out: std_logic := '0';
signal timer: integer range 0 to timer_limit := 0;
begin

StateMachine: process(clk) begin
   if rising_edge(clk) then
      case state is
         when ZERO =>
            sensor_out <= '0';
            if sensor_in = '1' then
               state <= RISE;
            end if;
         when RISE =>
            sensor_out <= '0';
            timer_enable <= '1';
            if timer_out = '1' then
               state <= ONE;
               timer_enable <= '0';
            end if;
            if sensor_in = '0' then
               state <= ZERO;
```

```
            timer_enable <= '0';
        end if;
      when ONE =>
        sensor_out <= '1';
        if sensor_in = '0' then
          state <= FALL;
        end if;
      when FALL =>
        timer_enable <= '1';
        sensor_out <= '1';
        if timer_out = '1' then
          state <= ZERO;
          timer_enable <= '0';
        end if;
        if sensor_in = '1' then
          state <= ONE;
          timer_enable <= '0';
        end if;
    end case;
  end if;
end process;

TimerProcesss: process(clk) begin
  if rising_edge(clk) then
    if timer_enable = '1' then
      if (timer = timer_limit - 1) then
        timer_out <= '1';
        timer <= 0;
      else
```

```
            timer <= timer + 1;
            timer_out <= '0';
        end if;
    else
        timer <= 0;
        timer_out <= '0';
    end if;
  end if;
end process;

end Behavioral;
```

mux.vhd

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

entity multiplexer is
 Port ( Counter_out : in STD_LOGIC_VECTOR (15 downto 0);
-- grp1 : in STD_LOGIC_VECTOR (3 downto 0);
-- grp2 : in STD_LOGIC_VECTOR (3 downto 0);
-- grp3 : in STD_LOGIC_VECTOR (3 downto 0);
-- grp4 : in STD_LOGIC_VECTOR (3 downto 0);
 sev_sel : in STD_LOGIC_VECTOR (1 downto 0);
 mux_out : out STD_LOGIC_VECTOR (3 downto 0));
end multiplexer;

architecture Behavioral of multiplexer is
```

```
SIGNAL match : STD_LOGIC_VECTOR (3 DOWNTO 0);

begin

PROCESS (Counter_out, sev_sel)
 BEGIN
 CASE sev_sel IS
 WHEN "00" => match <= Counter_out(3 downto 0);
 WHEN "01" => match <= Counter_out(7 downto 4);
 WHEN "10" => match <= Counter_out(11 downto 8);
 WHEN others => match <= Counter_out(15 downto 12);
 END CASE;
 mux_out <= match;
END PROCESS;

end Behavioral;
```

invertdecoder.vhd

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

entity inv_decoder is
 Port ( input : in STD_LOGIC_VECTOR (1 downto 0);
 output : out STD_LOGIC_VECTOR (3 downto 0));
end inv_decoder;
architecture Behavioral of inv_decoder is

begin
```

```
PROCESS(input)
BEGIN
CASE input IS
 WHEN "00" => output <= "1110";
 WHEN "01" => output <= "1101";
 WHEN "10" => output <= "1011";
 WHEN "11" => output <= "0111";
 WHEN others => output <= "0000";
END CASE;
END PROCESS;
end Behavioral;
```