Bilkent University

Electrical and Electronics Department

EE102-01 Lab 5 Report:

“Seven Segment Display”

30/10/2023

Fatih Mehmet Çetin - 22201689

Purpose:

The main purpose of this lab was to understand how the seven-segment system in the BASYS3 FPGA Board works and create a design that used multiple seven-segment displays simultaneously. We got familiar with how the clock system works in FPGA programming. Decoders and encoders were used throughout the lab. The concept of the perception of human eye was examined as well as the clock divider concept.

Methodology:

In the seven-segment display system of BASYS3, there are anodes and cathodes that control the LED system **(Figure 1.1)**. The anodes represent a full digit whereas the cathodes represent individual LED parts in a single digit. Therefore, with 11 inputs, one can light up to 28 different LEDs. In this experiment the digits on the seven-segment display will be used to represent hexadecimal numbers.

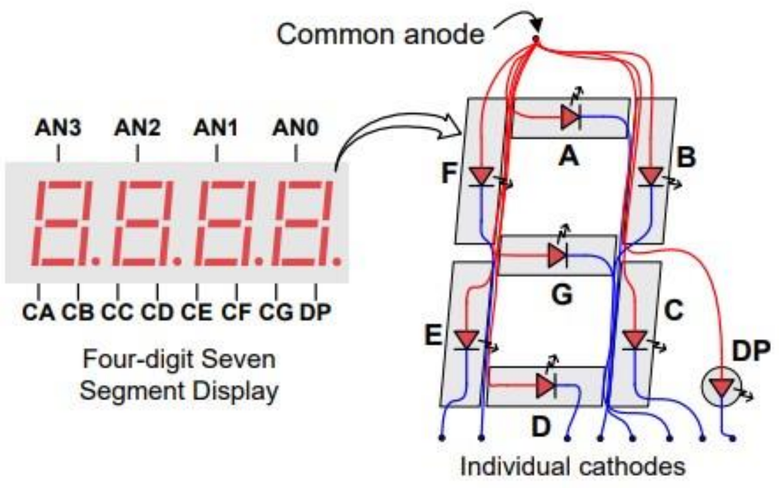

Figure 1.1: The Anode&Cathode Representation of the Seven-Segment Display

An important note is that no LEDs are on at the same time. Different LEDs are turned on and off by the system very quickly such that human eye cannot understand the difference. Human eye can notice frequencies up to 40-50Hz. If we use a high enough frequency, the human eye will see the seven-segment display as constant. The initial task we were given was to determine a frequency value and in this specific experiment, 100MHz/2^15 ~ 3000Hz will be the common frequency for each digit in the seven-segment display. This value will be more than enough to achieve the consistent picture.

After deciding on the frequency and the general design of the project, a main design source was created in Vivado with 2 inputs and 2 outputs. The inputs are the standard logic clock input and the number input which is a vector with a length of 16. The 2 outputs are simply the anode and cathode vectors which are respectively with length of 4 and 7 bits. Then, a testbench file was created and finally after the successful simulation, the constraints file was created, and the program was ready to be implemented on the BASYS3 FPGA Board.

Question 1: What is the internal clock frequency of Basys 3?

In the BASYS3 default clock frequency is 100MHz which is connected to pin W5 but its internal clock frequency exceeds 450MHz.

Question 2: How can you create a slower clock signal from this one?

Different algorithms can be used for this task. For instance, a counter can be used. When the counter hits a certain value, the divided clock signal could be inversed and as a result signals even lower than 100MHz.

Another easier way of doing the task is this; a standard logic vector with length of "n" whose value is zero at the start, can be incremented by one with the rising_edge command each time the internal clock value rises. Therefore, we can get a frequency of 100MHz/2^n Hz. The second way was used in this specific experiment.

Question 3: Can you create a clock with any arbitrary frequency lower than that of the internal clock? If not, which frequencies can you create?

It is important to state that only clocks with frequencies that has values 100Mhz/n where n is an integer. Because you can count only with integers, you can't increment the value of the divided_counter with decimal numbers. Therefore, you can only create certain values of frequency.

Design Specifications:

The initial decision that was given was to create a design where we can take some binary number inputs from the user and return them the hexadecimal equivalent and display the results on the seven-segment display. was not to use many sub-modules since they were going to make the code more complicated overall. Therefore, in the project there were only three sources (“seven_segment.vhd”; “ss_tb.vhd”; “ss_cst.xdc”) that was used.

In the seven-segment module, there were 2 inputs and 2 outputs. Inputs were the single-bit clock input (“clk”) and the 16-bit number input (“inn”). The number input was the 16-bit binary representation of the hexadecimal numbers that were going to be displayed on the 4 different LEDs. The outputs of the system were going to be the LEDs on the seven-segment display which was represented with 4 anode outputs and 7 cathode outputs.

Multiplexor, decoder and a clock divider module were used in the main design source. Here you can see the RTL schematic (**Figure 1.2)** and the internal structure of the “seven_segment.vhd”:

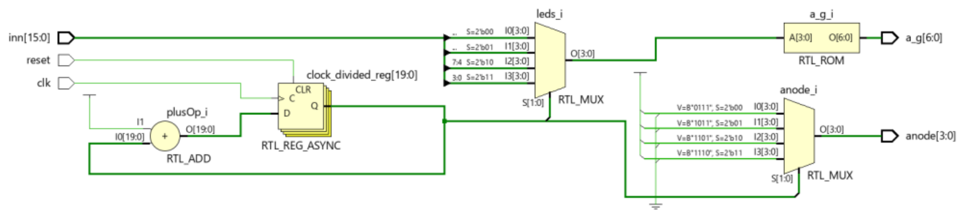

Figure 1.2: The RTL Schematic of “seven_segment.vhd”

Results:

In the “ss_tb.vhd” file, a testbench was simulated whose number inputs are “1111111100000000”,”1111000011110000”,”11001100110011001100” and “1010101010101010”. The anode and cathode outputs were consistent with the expected results. Here you can see the same simulations’ result from two different time scales in order to see the clock input better **(Figures 1.3&1.4)**.

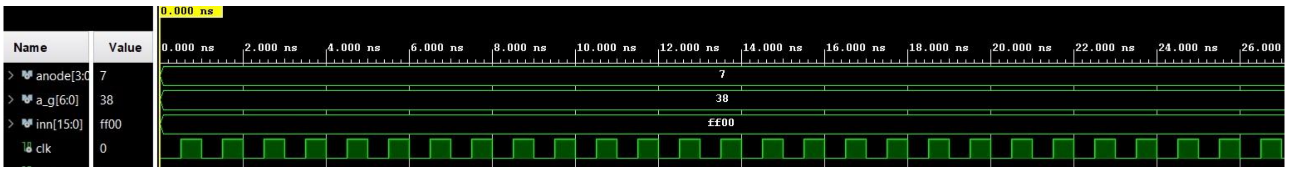

Figure 1.3: The Simulation Result of the Testbench graphed with a smaller time scale

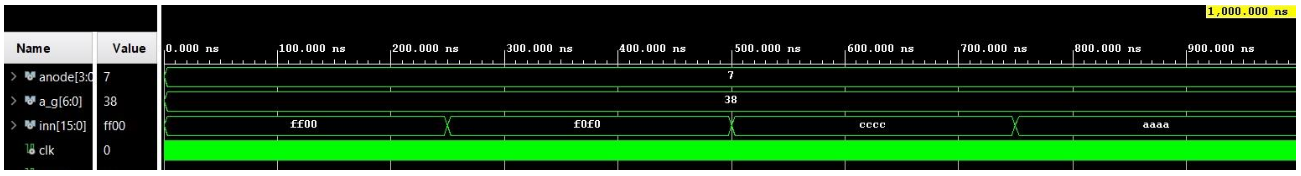

Figure 1.4: The Simulation Result of the Testbench graphed with a higher time scale

With these results it was proved that the design was working properly, and it was time we implement the design onto the BASYS3. 16 switches on the BASYS3 were assigned to 16 bits of the input vector “inn”. Here you can see some example input&output combinations from the working device **(Figures 1.5 up to 1.16)**:

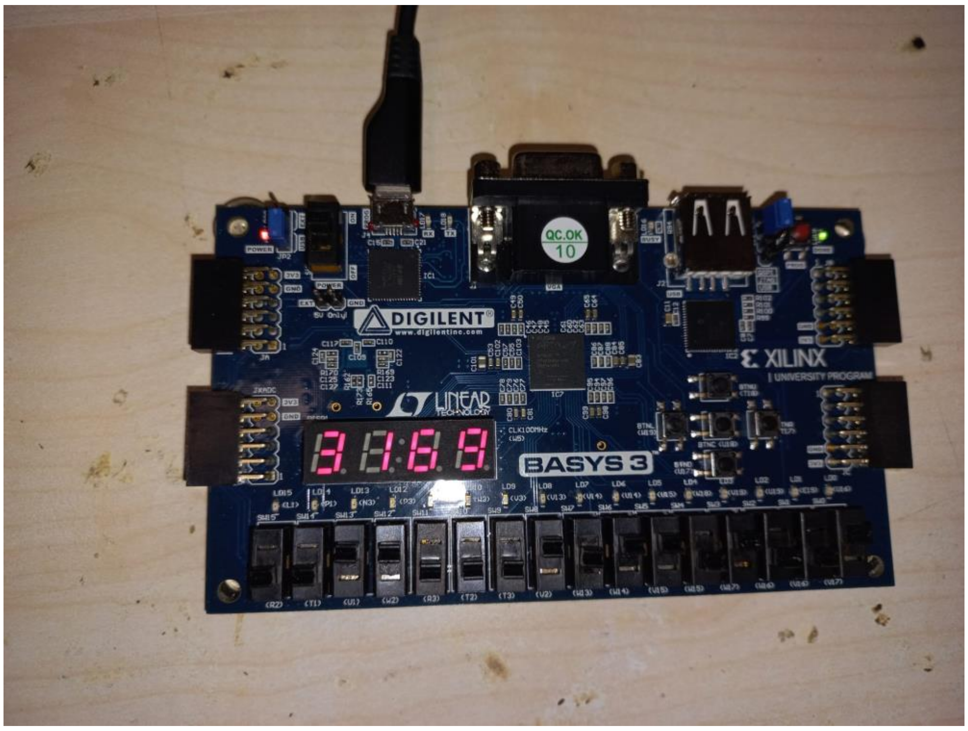

Figure 1.5: Seven-Segment Displaying “3169” when the input is “0011000101101001”

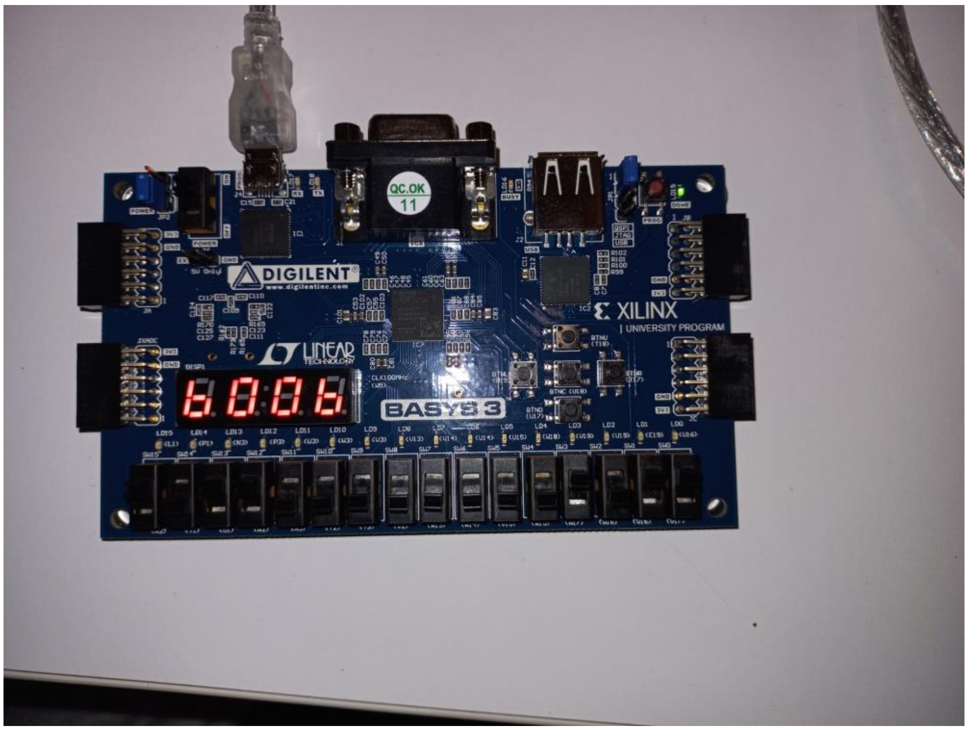

Figure 1.6: Seven-Segment Displaying “b00b” when the input is “1011000000001011”

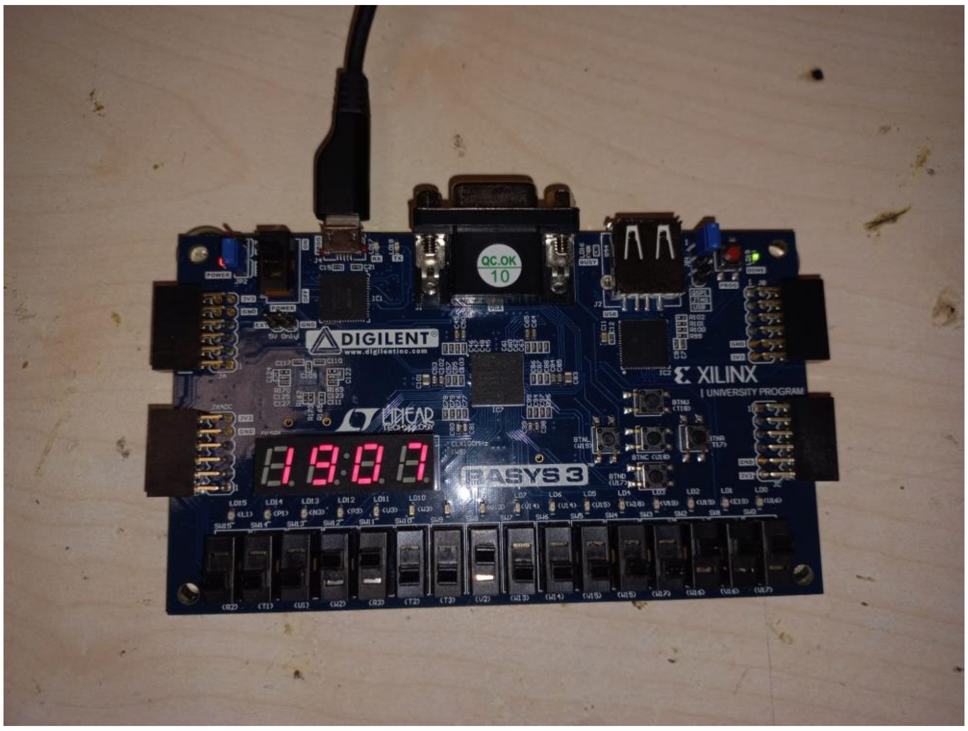

Figure 1.7: Seven-Segment Displaying "1907" when the input is "0001100100000111"

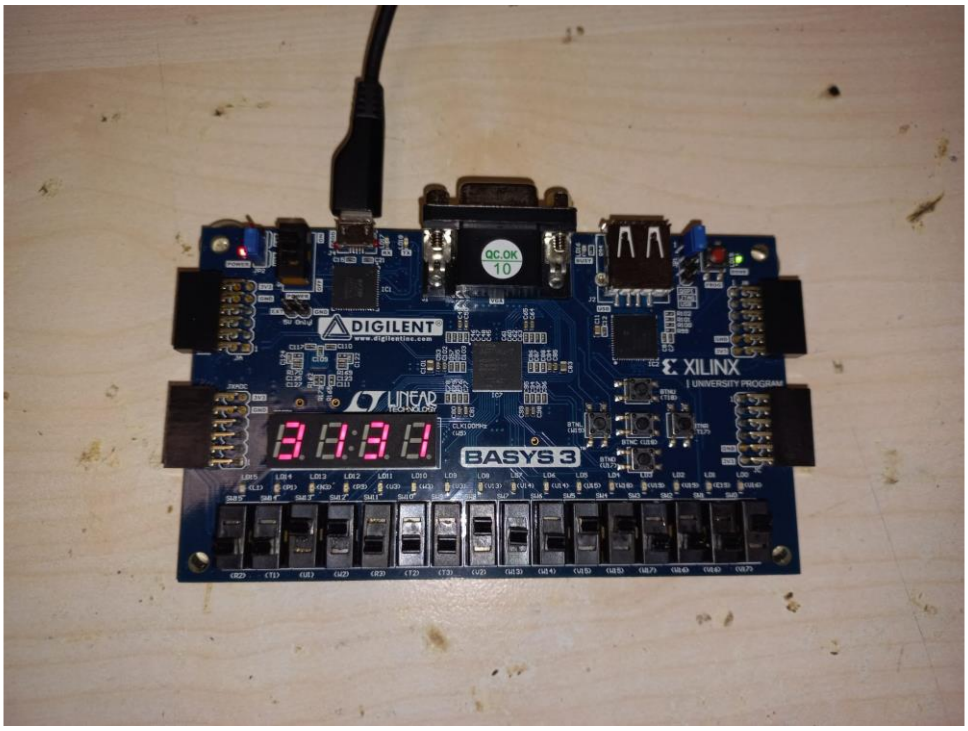

Figure 1.8: Seven-Segment Displaying "3131" when the input is "0011000100110001"

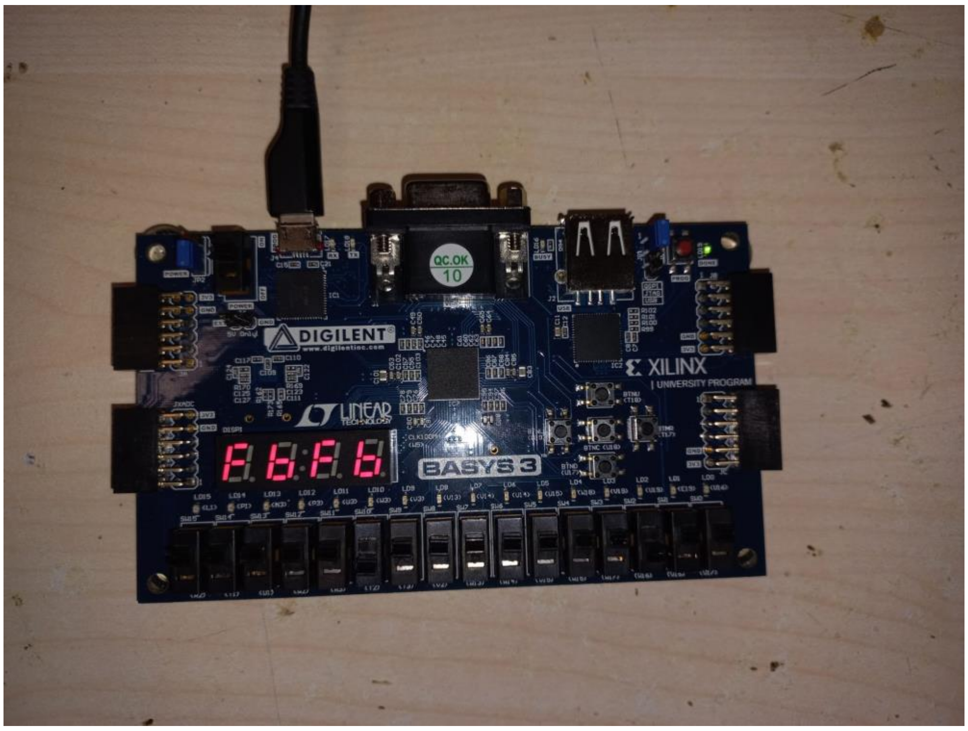

Figure 1.9: Seven-Segment Displaying “FbFb” when the input is “1111101111111011”

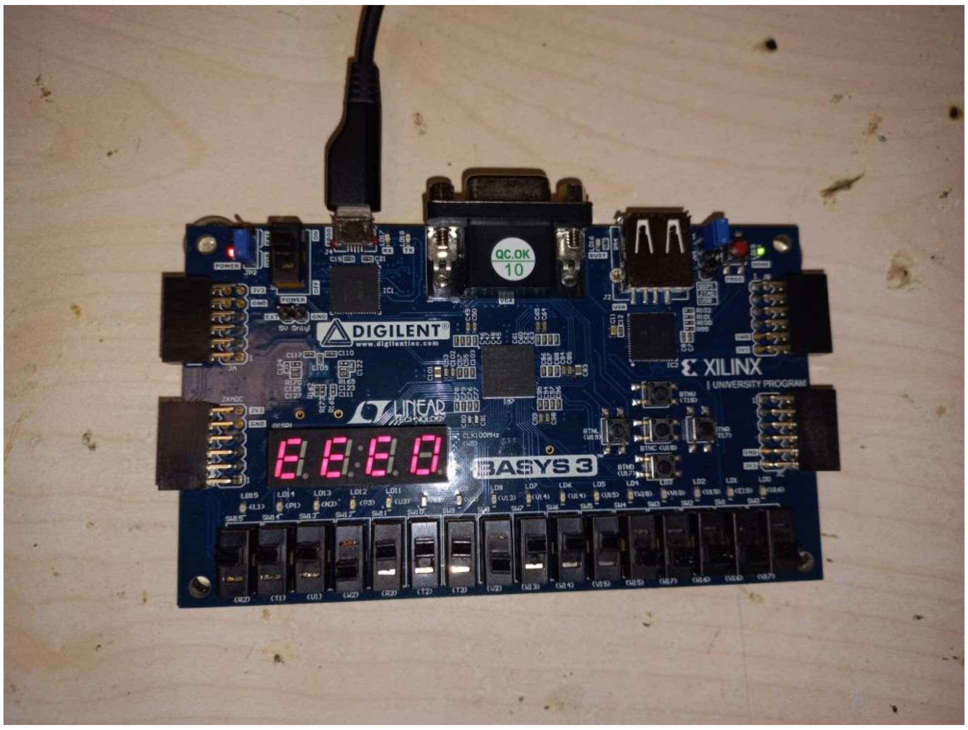

Figure 1.10: Seven-Segment Displaying “EEE0” when the input is “1110111011100000”

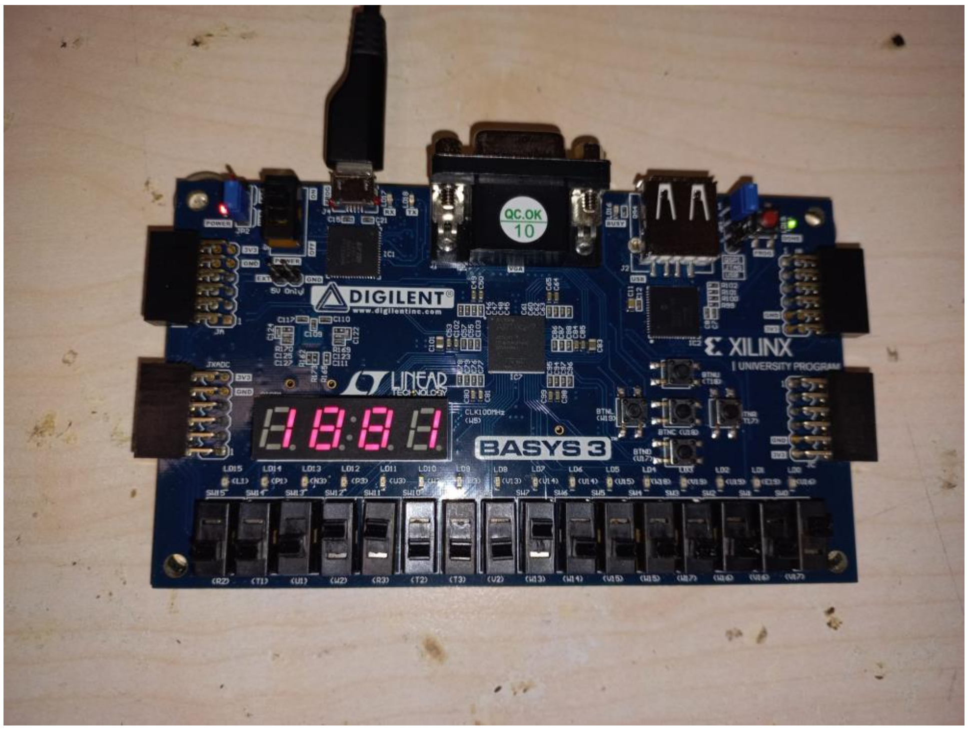

Figure 1.11: Seven-Segment Displaying “1881” when the input is "0001100010000001”

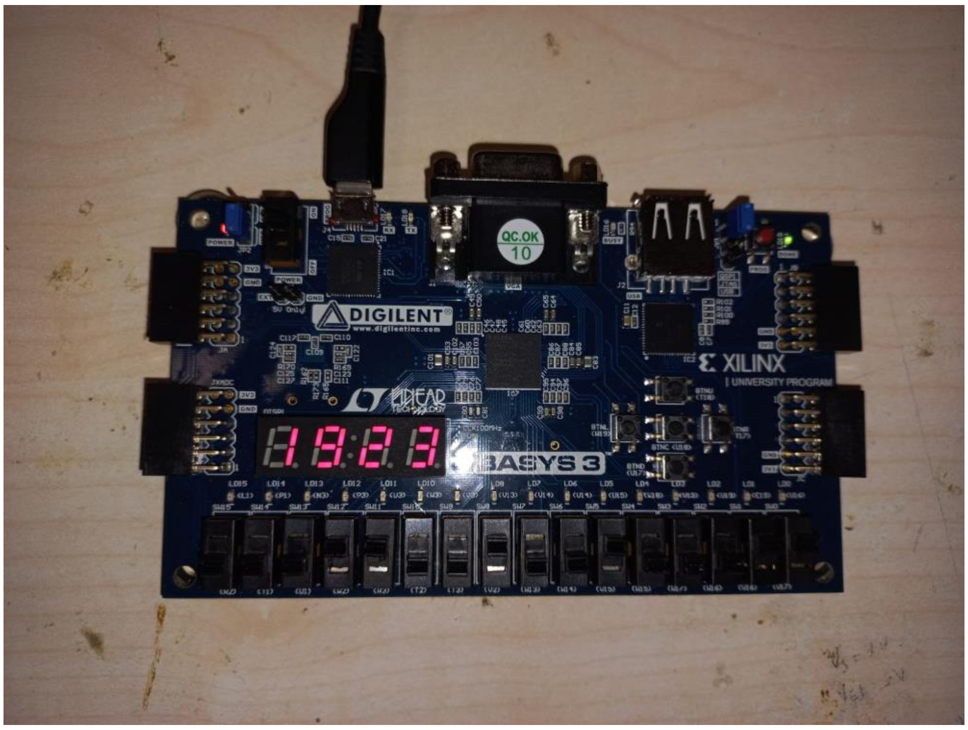

Figure 1.12: Seven-Segment Displaying “1923” when the input is "0001100100100011”

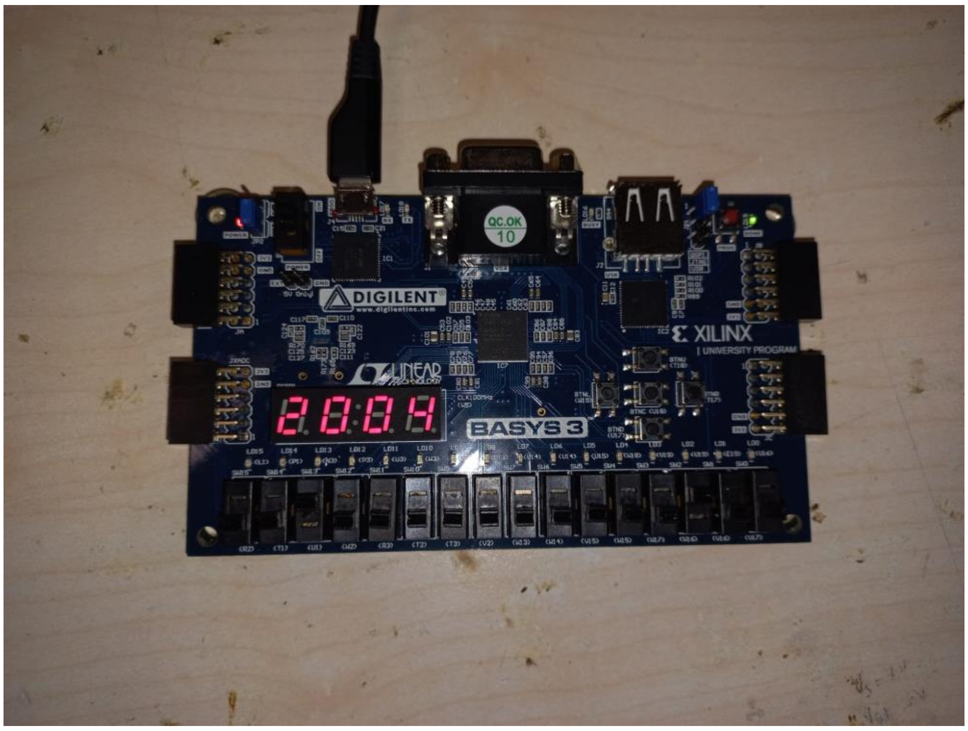

Figure 1.13: Seven-Segment Displaying “2004” when the input is "0010000000000100”

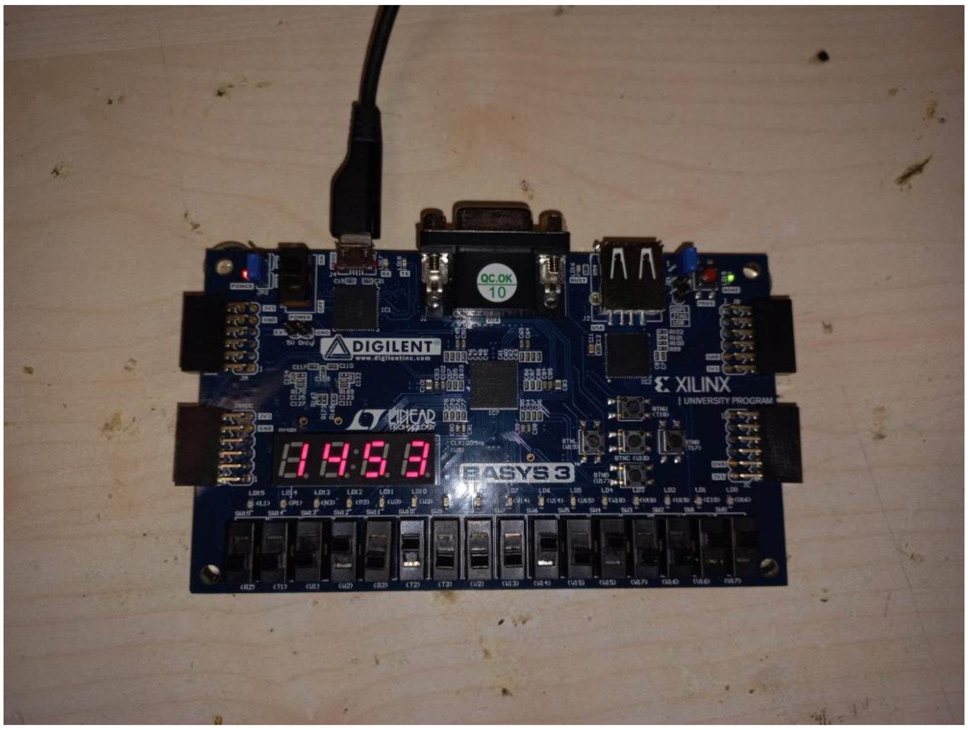

Figure 1.14: Seven-Segment Displaying “1453” when the input is "0001001001010011”

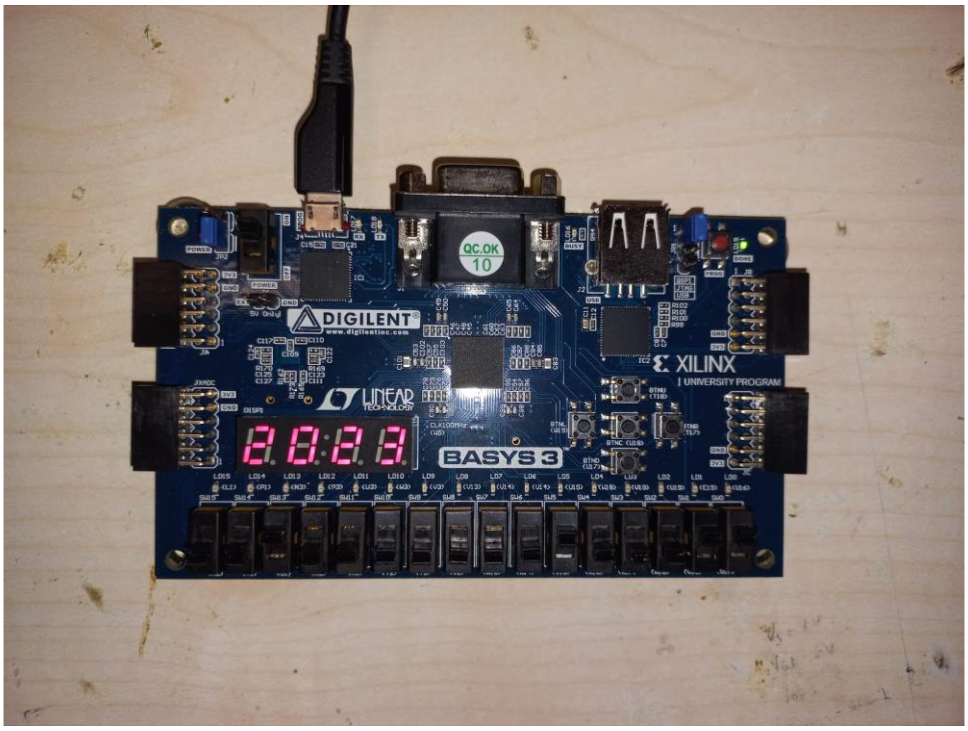

Figure 1.15: Seven-Segment Displaying “2023” when the input is "0010000000100011”

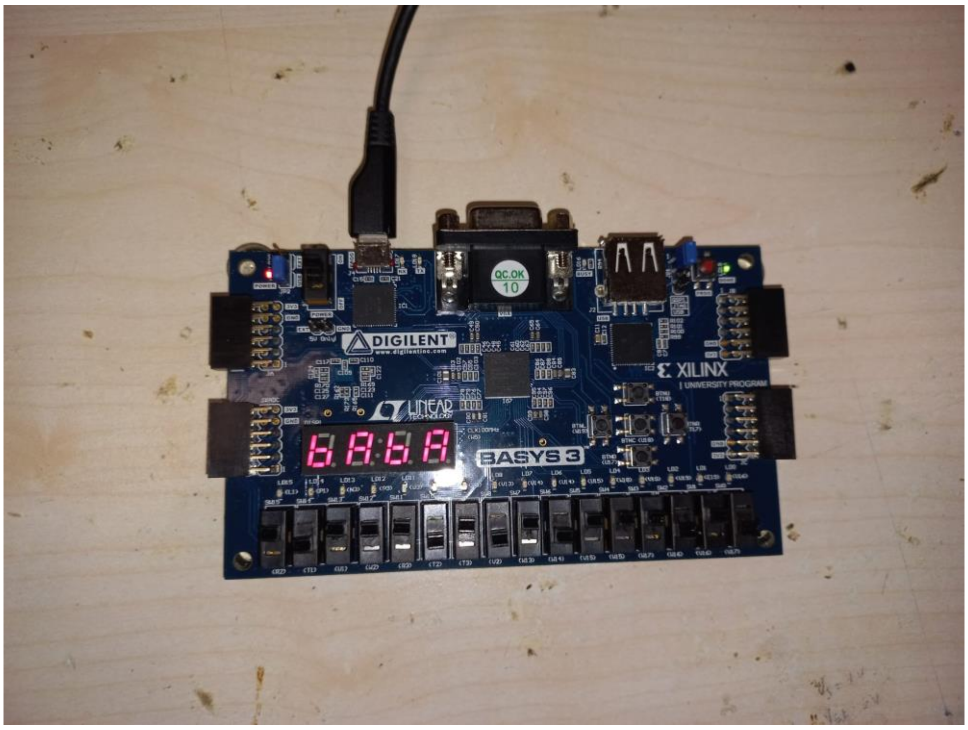

Figure 1.16: Seven-Segment Displaying “bAbA” when the input is "1011101010111010”

Conclusion:

The main purpose of the lab was to design a project that included the usage of the seven-segment display driver. I examined decoders, multiplexors and clock dividers and got used to them in this lab. Also, the human eye perception concept was examined, and it was purely the only thing that helped this lab to achieve its aim. It was really both a funny and a teaching lab, we learned many things. I myself was not very familiar with the clock divider concept and the clock concept in general in FPGA, but this lab really improved my skills on those topics. I believe the skills I learned in this lab will be very useful for my term project.

Appendices:

VHDL CODE:

seven_segment.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity seven_segment is
   Port ( anode: out std_logic_vector(3 downto 0);
      a_g : out std_logic_vector(6 downto 0);
      inn : in std_logic_vector(15 downto 0);
      clk: in std_logic );
```

```
end seven_segment;

architecture Behavioral of seven_segment is

signal clock_divided : std_logic_vector(19 downto 0) := "00000000000000000000" ;
signal act : std_logic_vector(1 downto 0);
signal leds : std_logic_vector(3 downto 0);



begin
   process(clk)
   begin
         if (rising_edge(clk)) then
            clock_divided <= clock_divided + 1;
         end if;
   act <= clock_divided(16 downto 15);
   end process;



   process(act)
   begin
   case act is
      when "00" => anode <= "0111";
         leds <= inn (15 downto 12);
      when "01" => anode <= "1011";
         leds <= inn (11 downto 8);
      when "10" => anode <= "1101";
         leds <= inn(7 downto 4);
      when "11" => anode <= "1110";
         leds <= inn(3 downto 0);
      when others => anode <= "1111";
```

```
    end case;
    end process;



    process(leds)
    begin
    case leds is
        when "0000" => a_g <= "0000001"; --0
        when "0001" => a_g <= "1001111"; --1
        when "0010" => a_g <= "0010010"; --2
        when "0011" => a_g <= "0000110"; --3
        when "0100" => a_g <= "1001100"; --4
        when "0101" => a_g <= "0100100"; --5
        when "0110" => a_g <= "0100000"; --6
        when "0111" => a_g <= "0001111"; --7
        when "1000" => a_g <= "0000000"; --8
        when "1001" => a_g <= "0000100"; --9
        when "1010" => a_g <= "0001000"; --A
        when "1011" => a_g <= "1100000"; --B
        when "1100" => a_g <= "0110001"; --C
        when "1101" => a_g <= "1000010"; --D
        when "1110" => a_g <= "0110000"; --E
        when "1111" => a_g <= "0111000"; --F
        when others => a_g <= "1111111"; --Empty
    end case;
    end process;
end Behavioral;
```

ss_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ss_tb is
end ss_tb;

architecture Behavioral of ss_tb is

component seven_segment port
(anode: out std_logic_vector(3 downto 0);
a_g : out std_logic_vector(6 downto 0);
inn : in std_logic_vector(15 downto 0);
clk: in std_logic);
end component;

signal anode: std_logic_vector (3 downto 0);
signal a_g : std_logic_vector(6 downto 0);
signal inn: std_logic_vector(15 downto 0);
signal clk: std_logic;
constant clk_period: time:= 1 ns;


begin
uut: seven_segment port map( anode => anode, a_g => a_g, inn => inn, clk => clk);


   start: process
   begin
   inn <= "1111111100000000";
   wait for 250 ns;
   inn <= "1111000011110000";
   wait for 250 ns;
```

```
    inn <= "1100110011001100";
    wait for 250 ns;
    inn <= "1010101010101010";
    wait for 250 ns;
    end process;

    clock: process
    begin
    clk <= '0';
    wait for clk_period /2;
    clk <= '1';
    wait for clk_period/2;
    clk <= '0';
    end process;

end Behavioral;
```

ss_cst.xdc

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

#clock signal:
set_property PACKAGE_PIN W5 [get_ports {clk}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk}]



#7 segment display:
#Cathodes:
set_property PACKAGE_PIN W7 [get_ports {a_g[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_g[6]}]
set_property PACKAGE_PIN W6 [get_ports {a_g[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_g[5]}]
set_property PACKAGE_PIN U8 [get_ports {a_g[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_g[4]}]
set_property PACKAGE_PIN V8 [get_ports {a_g[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_g[3]}]
set_property PACKAGE_PIN U5 [get_ports {a_g[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_g[2]}]
set_property PACKAGE_PIN V5 [get_ports {a_g[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_g[1]}]
set_property PACKAGE_PIN U7 [get_ports {a_g[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_g[0]}]
#Anodes:
set_property PACKAGE_PIN U2 [get_ports {anode[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[0]}]
set_property PACKAGE_PIN U4 [get_ports {anode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[1]}]
set_property PACKAGE_PIN V4 [get_ports {anode[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[2]}]
set_property PACKAGE_PIN W4 [get_ports {anode[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[3]}]
```