Lab 5: Seven-Segment Display

Date: 30.10.2023

Name: Şevval ERBAY

Section: 2

Purpose:

The purpose of this lab is to teach us how to use seven-segment display on BASYS3. In order to do that, I chose to do a second counter in hexadecimal base.

Methodology:

The internal clock frequency of the BASYS3 is 100 MHz. thus, we cannot light up more than 1 digit at a time. To slow down the internal clock frequency, one can use a clock divider, which we implemented in this lab. Also, it should be noted that one cannot create a clock with any arbitrary frequency lower than the internal clock frequency of the BASYS3. Moreover, we can only use integers in the clock creation process. Thus, we can only create new clocks with frequencies between 1 Hz to 100 Mhz.

In order to display multiple digits simultaneously, we make use of “persistence of vision”, which is basically turning the LEDs on and off very quickly so that naked eye cannot differentiate the delay. To get the optimal results, the period must be between 1 ms to 16 ms. In this specific application, 10.5 ms is chosen. A counter with sufficiently high frequency is implemented to make us of “persistence of vision”. Then, a decoder is used to decide which display will light up and a multiplexer is used to decide which digits will be displayed. RTL schematic (Figure.1) is synthetized. A test bench code is written to simulate the results. Lastly, the constraint code is written and the bit stream is generated. The process is simulated by using BASYS3’s seven-segment display and one of the buttons.

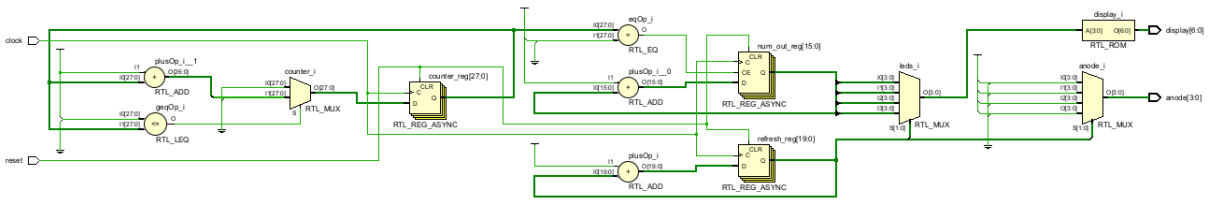

Figure.1 RTL Schematic of seven_segment_display.vhd

There are 2 inputs, **clock** and **reset**, and 2 outputs, **anode** and **display**. **anode** is used to choose which digit will light up and **display** is used to show different values a digit can obtain. **reset** is used to reset both the counter and the displayed value to 0.

clock is used to obtain the new clock frequency of the BASYS3 after lowering the internal clock frequency from 100 MHz. This process is made by using *if-elsif-else* statements and *rising_edge* function.

one_second_enable signal and **counter** signal are used to count the passing seconds.

A decoder is written by using *case-when* statements to choose which cathode pattern will be used.

refresh signal is used to generate the 10.5 ms period.

In the 4-to-1 multiplexer, we generate signals based on the 2 most significant digits of the **refresh** signal, which will in turn decide which digits will be displayed. **num_out** signal counts in hexadecimal and is shown in the seven-segment display according to this 4-to-1 multiplexer. **leds** signal attain its value by the 4-bit parts of **num_out** signal.

Results:

A waveform is observed by a test bench (testbench.vhd). also, it should be noted that since our counter changes its value once for every second, we cannot observe the shift in the **display** output in the test bench shown below. This is due to the test bench having a much shorter time span (1000 ns) compared to 1 s. To check, I made the simulation run for 5 s and it worked the way it should but the graph was compressed and looked like a solid green bar. For this purpose, the 5 s test bench result is not included.

The middle button (U18) is used to reset the counter and restart the counting process.

The waveform and the BASYS3 representations are in parallel.

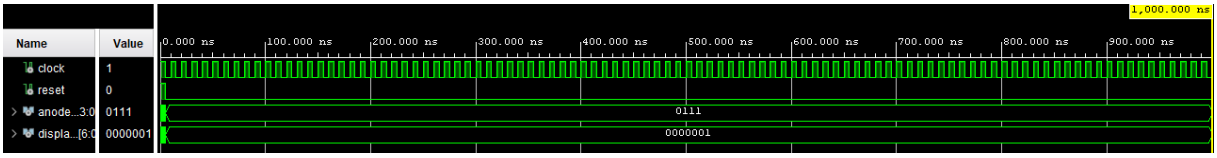

Figure.2 Waveform, generated by the test bench

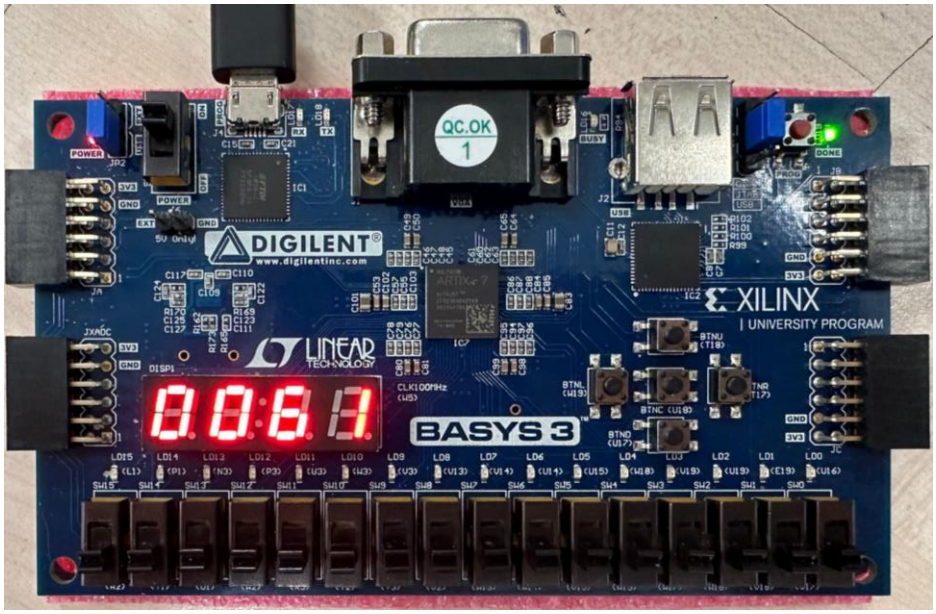

Figure.3 Display after 97 seconds (61 in hexadecimal)

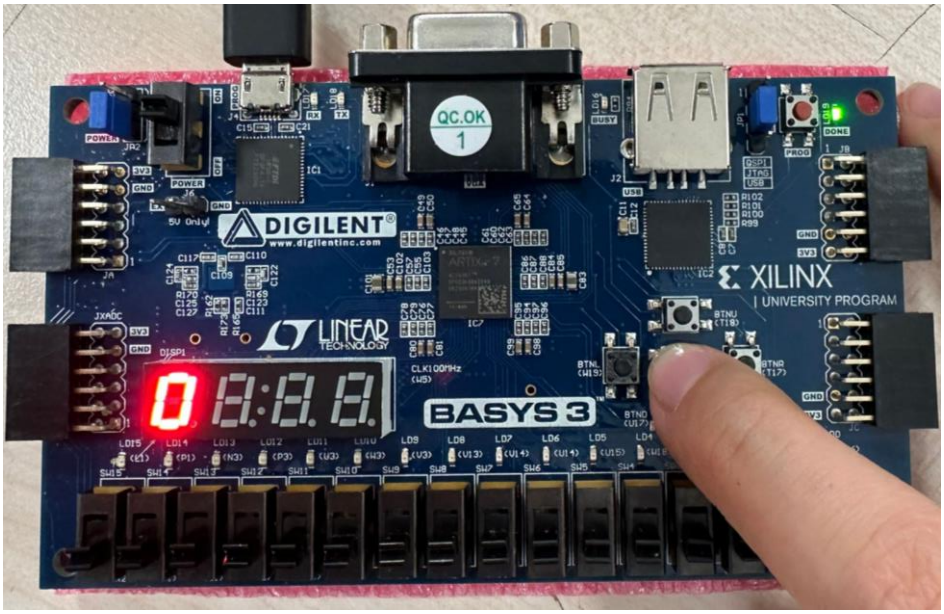

Figure.4 “Reset” button U18

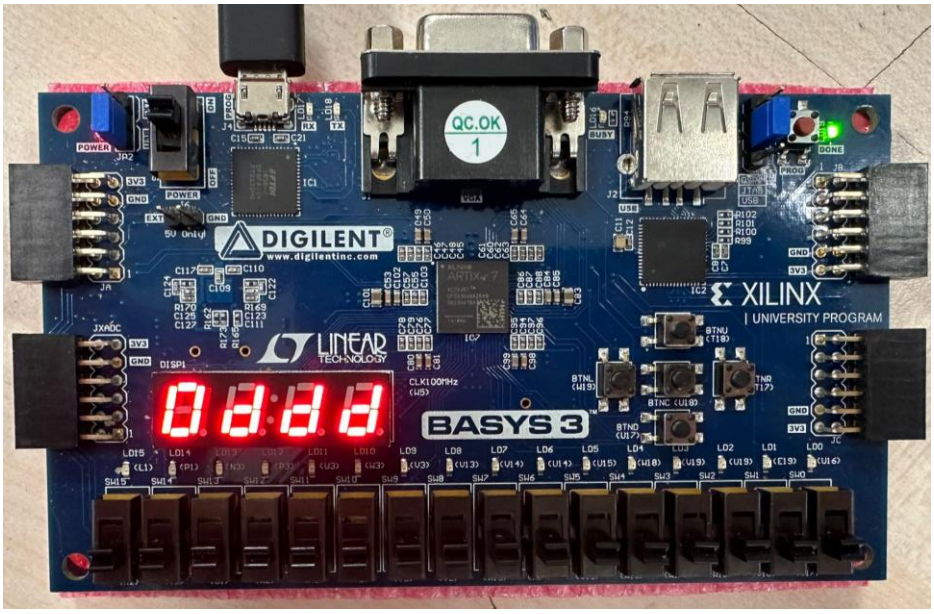

Figure.5 Display after 3549 seconds (DDD in hexadecimal)

Conclusion:

In this lab, we learned how to use the seven-segment display on our BASYS3s by using VHDL. To do that, a second counter in hexadecimal base is implemented. I learned how to properly use the seven-segment display. Every seven-segment display has 4 anodes which controls the digits. Those anodes are chosen by a multiplexer. All in all, the experiment was successful at the end.

APPENDIX

testbench.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std;
entity testbench is
end entity testbench;
architecture Behavioral of testbench is
    signal clock, reset: std_logic :='0';
    signal anode_tb: std_logic_vector(3 downto 0) := (others => '0');
    signal display_tb: std_logic_vector(6 downto 0) := (others => '0');
    component seven_segment_display
        Port (clock, reset: in std_logic :='0';
              anode: out std_logic_vector(3 downto 0);
              display: out std_logic_vector(6 downto 0));
    end component;
begin
    uut: seven_segment_display
    port map ( clock => clock, reset => reset, anode => anode_tb, display => display_tb);
    process
    begin
        clock <= '1';
        wait for 5 ns;
        clock <= '0';
        wait for 5 ns;
    end process;
    process
    begin
```

```
        reset <= '1';
        wait for 5 ns;
        reset <= '0';
        wait;
        wait for 5 ns;
        reset <= '1';
        wait;
    end process;
end Behavioral;
```

seven_segment_display.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
entity seven_segment_display is
    Port ( clock , reset : in STD_LOGIC;
           anode : out STD_LOGIC_VECTOR (3 downto 0);
           display : out STD_LOGIC_VECTOR (6 downto 0));
end seven_segment_display;
architecture Behavioral of seven_segment_display is
signal counter: STD_LOGIC_VECTOR (27 downto 0);
signal one_second_enable: std_logic;
signal num_out: STD_LOGIC_VECTOR (15 downto 0);
signal leds: STD_LOGIC_VECTOR (3 downto 0);
signal refresh: STD_LOGIC_VECTOR (19 downto 0);
begin
process(leds)
begin
    case leds is
        when "0000" => display <= "0000001"; -- "0"
        when "0001" => display <= "1001111"; -- "1"
        when "0010" => display <= "0010010"; -- "2"
        when "0011" => display <= "0000110"; -- "3"
        when "0100" => display <= "1001100"; -- "4"
        when "0101" => display <= "0100100"; -- "5"
        when "0110" => display <= "0100000"; -- "6"
        when "0111" => display <= "0001111"; -- "7"
        when "1000" => display <= "0000000"; -- "8"
        when "1001" => display <= "0000100"; -- "9"
```

```
        when "1010" => display <= "0000010"; -- a
        when "1011" => display <= "1100000"; -- b
        when "1100" => display <= "0110001"; -- C
        when "1101" => display <= "1000010"; -- d
        when "1110" => display <= "0110000"; -- E
        when others => display <= "0111000"; -- F
    end case;
end process;
process(clock,reset)
begin
    if reset='1' then
        refresh <= (others => '0');
    elsif rising_edge(clock) then
        refresh <= refresh + 1;
    end if;
    case refresh(19 downto 18) is
        when "00" => anode <= "0111";
            leds <= num_out(15 downto 12);
        when "01" => anode <= "1011";
            leds <= num_out(11 downto 8);
        when "10" => anode <= "1101";
            leds <= num_out(7 downto 4);
        when others => anode <= "1110";
            leds <= num_out(3 downto 0);
    end case;
end process;
process(clock, reset)
begin
        if reset='1' then
            counter <= (others => '0');
```

```
        elsif rising_edge(clock) then
            if counter>=x"5F5E0FF" then
                counter <= (others => '0');
            else
                counter <= counter + "0000001";
            end if;
        end if;
end process;
one_second_enable <= '1' when counter=x"5F5E0FF" else '0';
process(clock, reset)
begin
        if reset='1' then
            num_out <= (others => '0');
        elsif rising_edge(clock) then
             if one_second_enable='1' then
                num_out <= num_out + x"0001";
             end if;
        end if;
end process;
end Behavioral;
```

const.xdc

```
set_property PACKAGE_PIN W5 [get_ports clock]
   set_property IOSTANDARD LVCMOS33 [get_ports clock]
set_property PACKAGE_PIN U18 [get_ports reset]
   set_property IOSTANDARD LVCMOS33 [get_ports reset]

set_property PACKAGE_PIN W7 [get_ports {display[6]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {display[6]}]
set_property PACKAGE_PIN W6 [get_ports {display[5]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {display[5]}]
set_property PACKAGE_PIN U8 [get_ports {display[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {display[4]}]
set_property PACKAGE_PIN V8 [get_ports {display[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {display[3]}]
set_property PACKAGE_PIN U5 [get_ports {display[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {display[2]}]
set_property PACKAGE_PIN V5 [get_ports {display[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {display[1]}]
set_property PACKAGE_PIN U7 [get_ports {display[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {display[0]}]

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]}]
```