Yiğit Narter
22102718
Section 01
Lab Report 5
21.11.2022

LAB 5: SEVEN-SEGMENT DISPLAY

Purpose

The purpose of this lab is to design a seven-segment display on BASYS3 board using the LEDs of the segment. The display should show multiple different digits simultaneously.

Methodology

The design will essentially be a hexadecimal counter that will count up with each second. The seven-segment is expected to show the number of seconds that has passed in hexadecimal base. The seven-segment display in the design is controlled by two inputs called anode and cathode. The anodes control the segments that are displayed and the cathodes control the LEDs. Even though it is not possible to light multiple segments at the same time, we can create an illusion that they are on at the same time since the displays can be turned on and off very quickly. This is called the persistence of vision.

After creating a code for the clock counter, I wrote the codes to control the displays that will light up and a segment driver to control the digits of the number that is displayed. I simulated the design using test bench codes and implemented the design on the BASYS3 board.

Questions

- What is the internal clock frequency of BASYS3?

BASYS3 has an internal clock frequency of 100MHz. I also chose this value for the frequency of the input clock of the design.

- How can you create a slower clock signal from this one?

We can use a clock divider. For example, to generate a 25MHz clock out of the internal clock, we can toggle the 25 MHz clock for every 4 ticks of the internal 100MHz clock. To divide the clock, we need to toggle it according to the selected tick count of the base clock.

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

The number of ticks per toggle number should be integer values, so it isn’t possible to choose arbitrary frequencies. So we can create frequencies that result in an integer value when we divide 100MHz with it.

Yiğit Narter
22102718
Section 01
Lab Report 5
21.11.2022

Design Specifications

There are 2 inputs and 2 outputs in this design. The inputs are clock, which has a fixed value of 100MHz and reset, which is used to restart the display. The 4-bit output anode and the 7-bit output cathode control the LEDs to light up.

Like the previous lab, I designed the code in a modular fashion. The top module “main.vhd” includes the clock and driver modules. The “clock.vhd” module functions as a clock counter for the display. It has 2 inputs clk, which is the 100Mhz internal clock of Basys3 and reset, which restarts the counting. There are 4 outputs: clock_count which counts the clock ticks to test the clock, phase_count which is used to control on-off sequence for the segments, sec which is the number of seconds and sec_en, which is asserted to 1 in each second, else it is 0. The clock division is enabled with the phase_count output.

The second module “segment_driver.vhd” takes the outputs of the clock module as inputs. The driver module has 5 inputs (clk, reset, phase_count,num which is the sec output of the previous module and sec_en) and 2 outputs which are “an” (anode which controls the segments) and “cat” (cathode which controls the LEDs).

The last module “cathode_reg.vhd” drives the cathode according to the number input. It is used under the “segment_driver.vhd” module. It takes “led_in” (the number we want to display) as input and the output is the corresponding cathode combination.

Results

Figures 1 and 2 show the RTL schematic of the design.

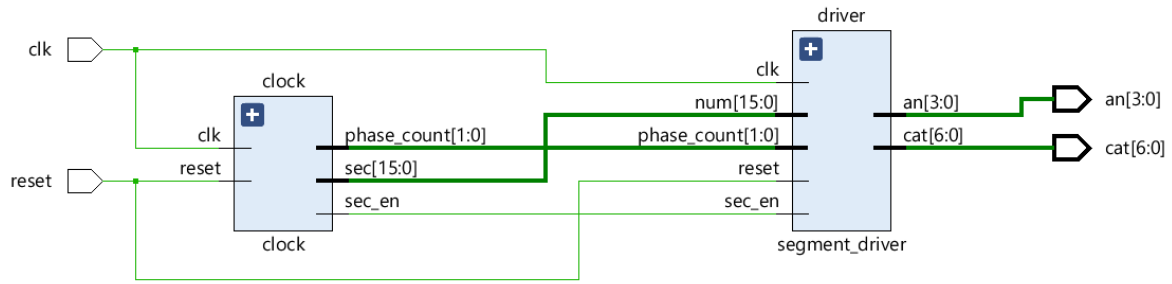

Figure 1: RTL schematic of the main module

Yiğit Narter
22102718
Section 01
Lab Report 5
21.11.2022

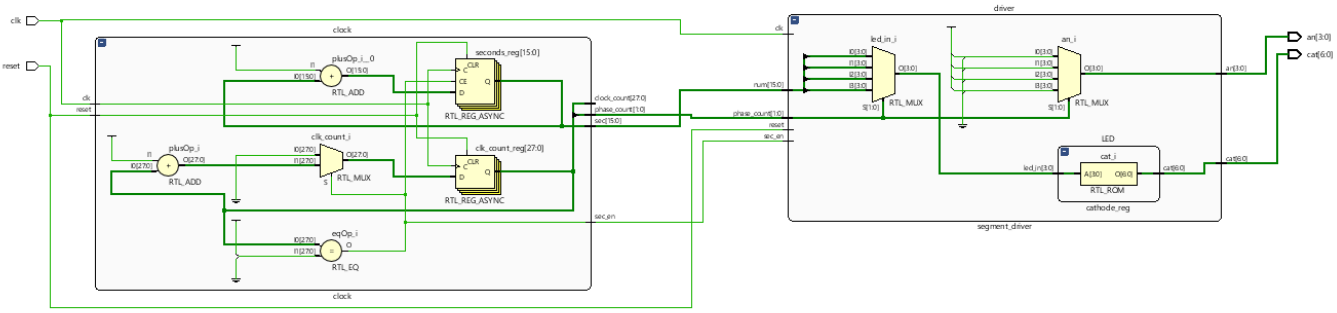

Figure 2: RTL schematic with the clock and segment modules displayed

To simulate the codes, I created a test bench code for the main module. Also, to make sure the clock module worked properly I created a separate test bench code for it. Figures 3-4 show the simulation results of the test bench codes.

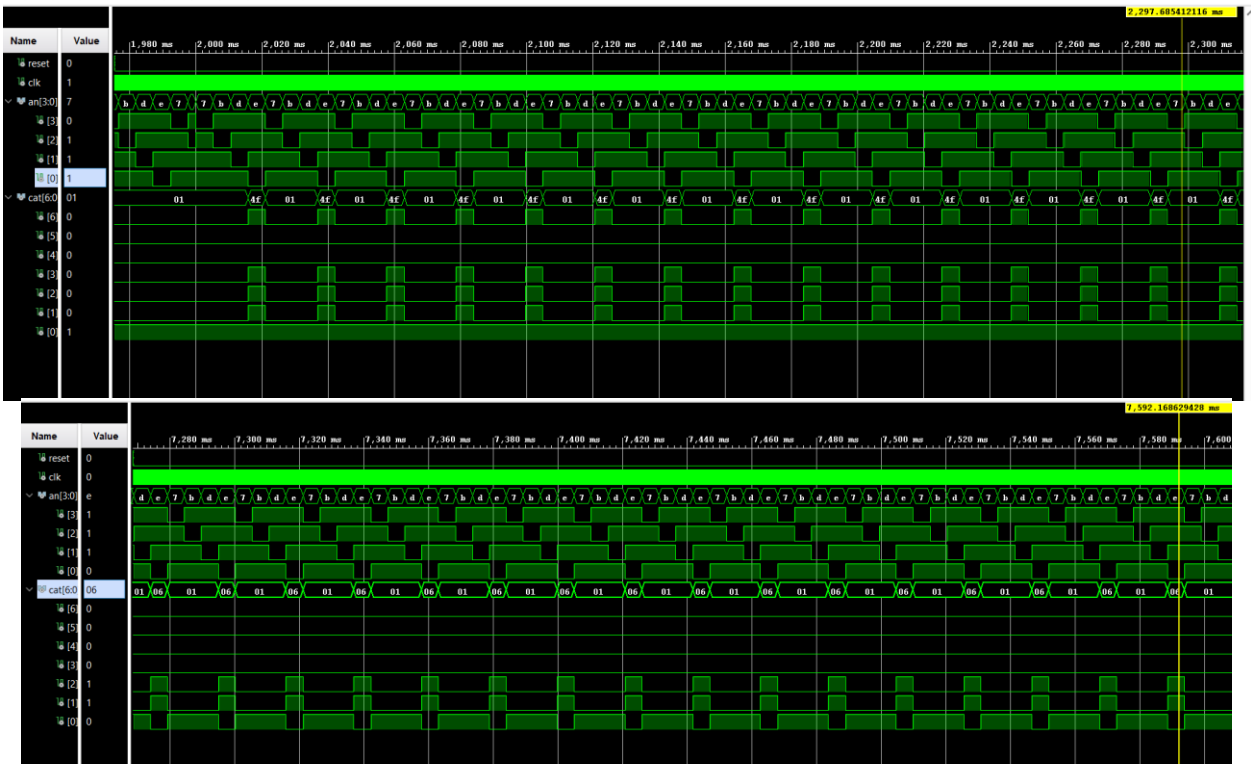

Figure 3: Simulation results of the main test bench code

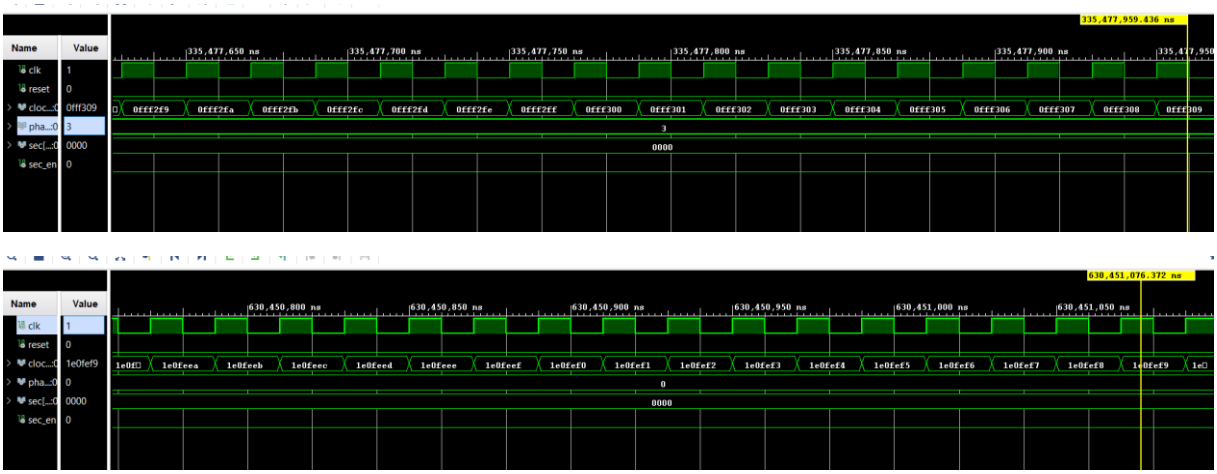

Figure 4: Simulation results of the clock test bench code

After creating the constraint file “pin_7seg.xdc”, the bitstream is generated and the design is implemented on the BASYS3 board. The results were as expected. Figures 5-10 show the outputs on the BASYS3 board corresponding to the seconds that has passed. The rightmost switch corresponds to the input “reset”.

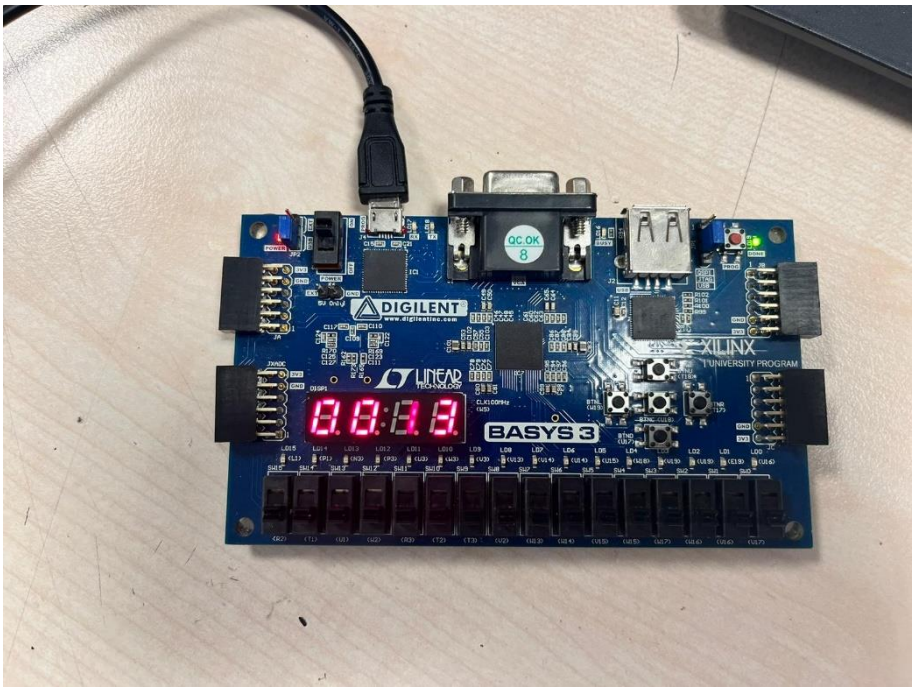

Figure 5: Output is "0013" in hexadecimal base after 19 seconds

Yiğit Narter
22102718
Section 01
Lab Report 5
21.11.2022

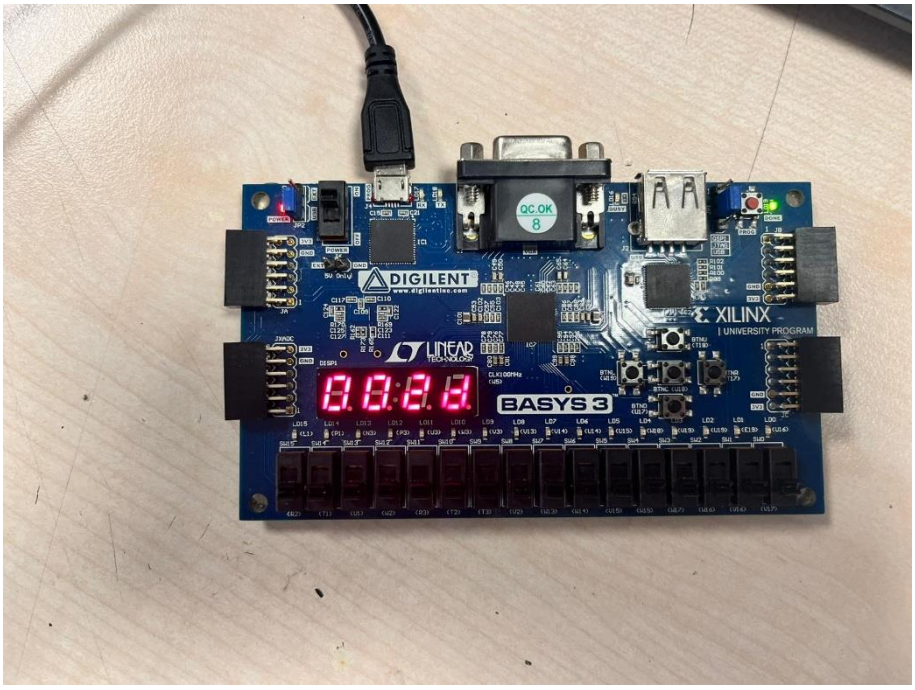

Figure 6: Output is "002d" in hexadecimal base after 45 seconds

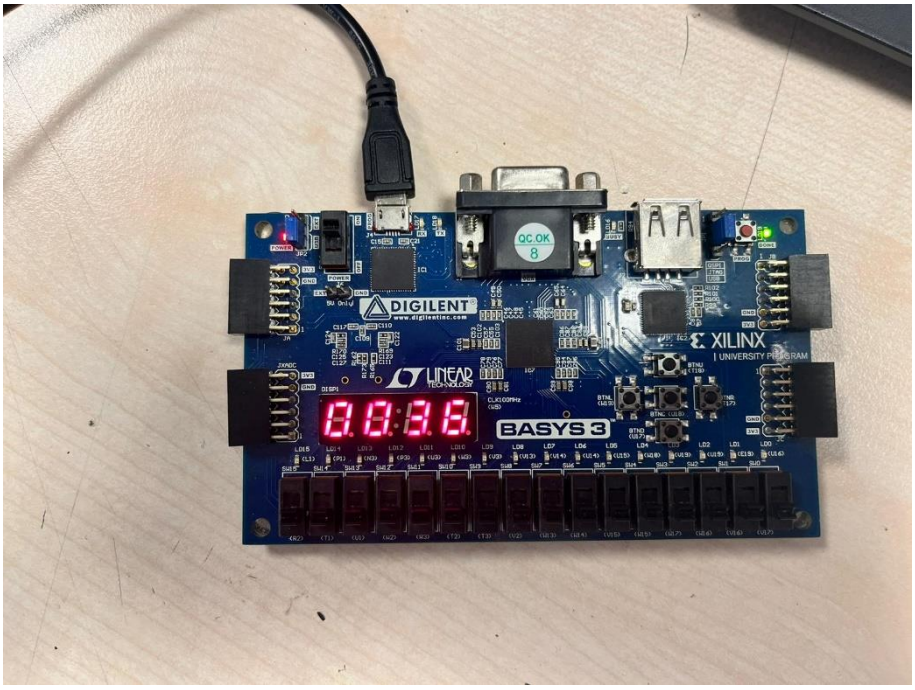

Figure 7: Output is "0036" in hexadecimal base after 54 seconds

Yiğit Narter
22102718
Section 01
Lab Report 5
21.11.2022

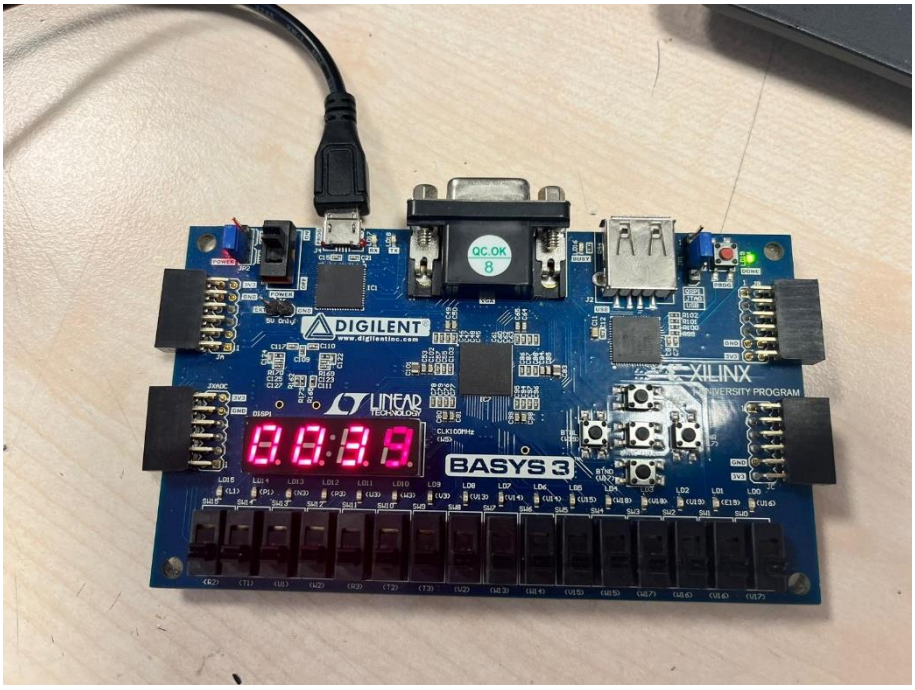

Figure 8: Output is "0039" in hexadecimal base after 57 seconds

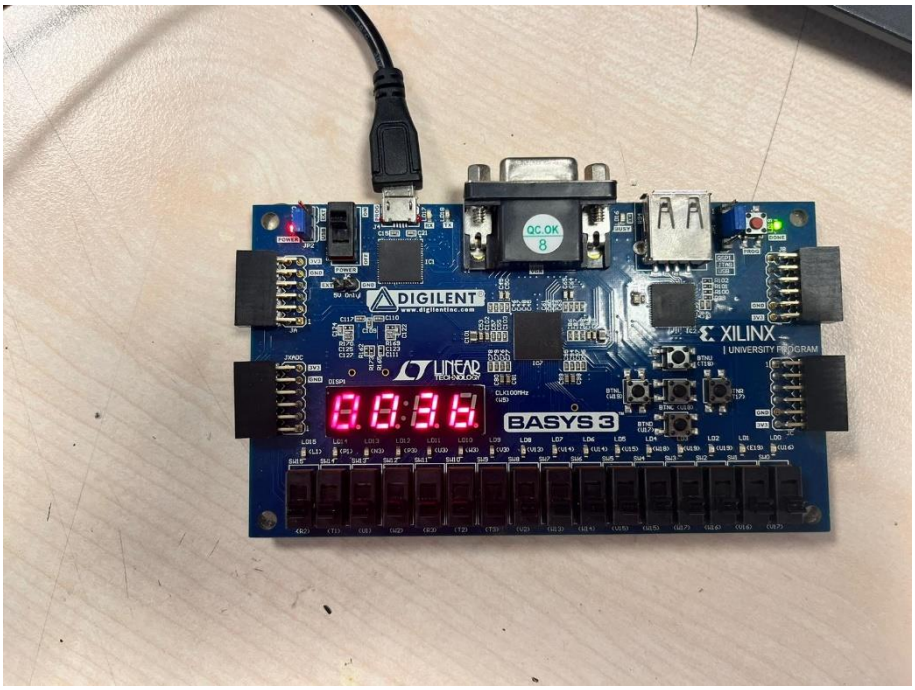

Figure 9: Output is "003b" in hexadecimal base after 59 seconds

Yiğit Narter
22102718
Section 01
Lab Report 5
21.11.2022

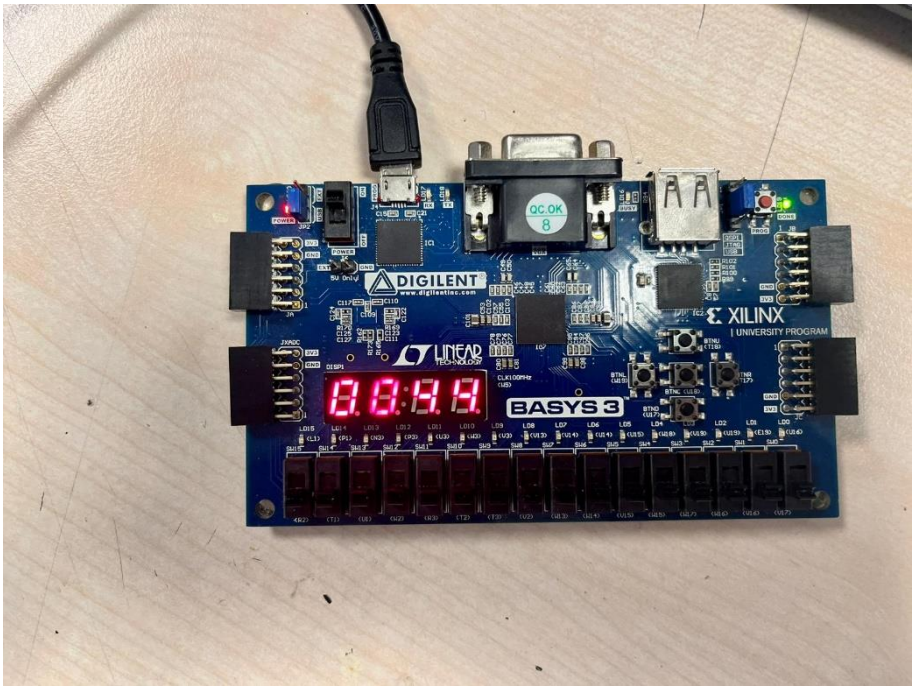

Figure 10: Output is "0044" in hexadecimal base after 68 seconds

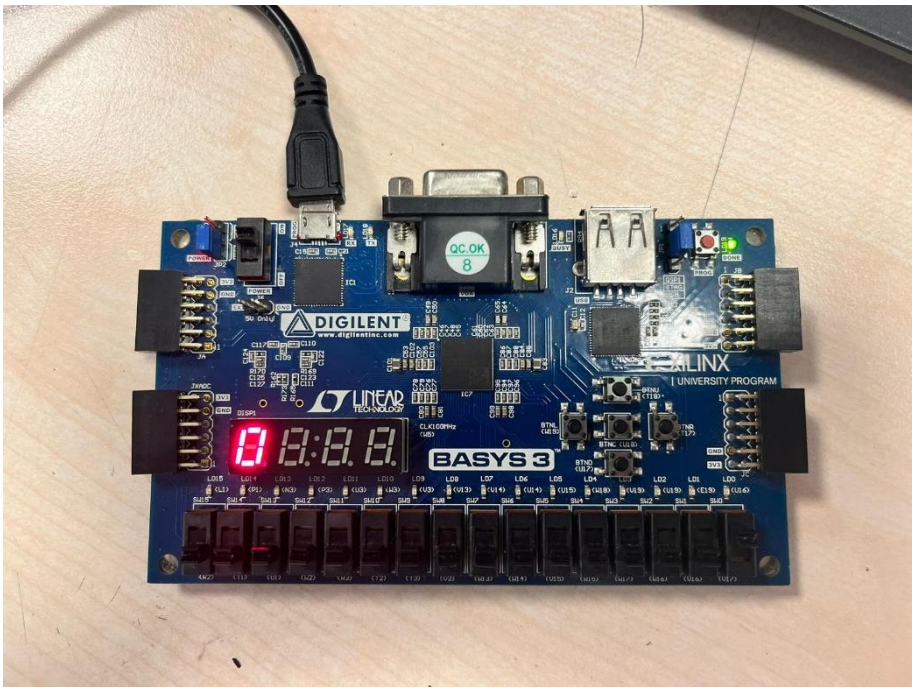

Figure 11: Output is "0" when the reset switch is on

Yiğit Narter
22102718
Section 01
Lab Report 5
21.11.2022

Conclusion

The aim of this experiment was to implement a seven-segment display using VHDL. I first learned how to use the display of the BASYS3 board. Then after writing the code and implementing it, the result turned out as expected and the display was successful. Even though I had to do a lot of research, I can say I learned a lot about how the internal clock of BASYS3 works and how the LEDs of the segments are controlled.

Appendices

Design Codes

main.vhd

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

entity main is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           an : out STD_LOGIC_VECTOR (3 downto 0);
           cat : out STD_LOGIC_VECTOR (6 downto 0));
end main;

architecture Behavioral of main is
 signal phase_count: std_logic_vector(1 downto 0);
 signal sec_en: std_logic;
 signal num: std_logic_vector(15 downto 0);
begin

clock: entity work.clock(clk)
 port map(clk => clk, reset => reset,
 phase_count => phase_count,
```

```
 sec_en => sec_en,
 sec => num);
----------------------------------------------------------------------------
driver: entity work.segment_driver(driver)
 port map(clk => clk, reset => reset, phase_count => phase_count, num => num,
 sec_en => sec_en, an => an, cat => cat);

end Behavioral;
```

cathode_reg.vhd

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

entity cathode_reg is
    Port ( led_in : in STD_LOGIC_VECTOR (3 downto 0);
           cat : out STD_LOGIC_VECTOR (6 downto 0));
end cathode_reg;

architecture reg of cathode_reg is
begin
 process(led_in) begin
 case led_in is
     when "0000" => cat <= "0000001";
     when "0001" => cat <= "1001111";
     when "0010" => cat <= "0010010";
     when "0011" => cat <= "0000110";
     when "0100" => cat <= "1001100";
     when "0101" => cat <= "0100100";
     when "0110" => cat <= "0100000";
```

```
    when "0111" => cat <= "0001111";
    when "1000" => cat <= "0000000";
    when "1001" => cat <= "0000100";
    when "1010" => cat <= "0000010";
    when "1011" => cat <= "1100000";
    when "1100" => cat <= "0110001";
    when "1101" => cat <= "1000010";
    when "1110" => cat <= "0110000";
    when "1111" => cat <= "0111000";
    when others => cat <= "1111111";
 end case;
 end process;
```

clock.vhd

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



entity clock is
   Port ( clk : in STD_LOGIC;
          reset : in STD_LOGIC;
          clock_count : out STD_LOGIC_VECTOR (27 downto 0);
          phase_count : out STD_LOGIC_VECTOR (1 downto 0);
          sec : out STD_LOGIC_VECTOR (15 downto 0);
          sec_en : out STD_LOGIC);
end clock;
```

```
architecture clk of clock is
 signal clk_count: std_logic_vector(27 downto 0);
 signal seconds: std_logic_vector(15 downto 0):= (others => '0');
begin

 process(clk) begin
 if(reset = '1') then
 clk_count <= (others => '0');
 seconds <= (others => '0');
 else
 if rising_edge(CLK) then
 clk_count <= clk_count + '1';
 if clk_count =x"5F5E0FF" then
 seconds <= seconds + '1';
 clk_count <= (others => '0');
 end if;
 end if;
 end if;
 end process;

sec_en <= '1' when clk_count =x"5F5E0FF" else '0';
phase_count <= clk_count(19 downto 18);
clock_count <= clk_count;
sec <= seconds;

end clk;
```

segment_driver.vhd

```
library IEEE;
```

Yiğit Narter
22102718
Section 01
Lab Report 5
21.11.2022

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

entity segment_driver is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           phase_count : in STD_LOGIC_VECTOR (1 downto 0);
           num : in STD_LOGIC_VECTOR (15 downto 0);
           sec_en : in STD_LOGIC;
           an : out STD_LOGIC_VECTOR (3 downto 0);
           cat : out STD_LOGIC_VECTOR (6 downto 0));
end segment_driver;

architecture driver of segment_driver is
 signal led_in: std_logic_vector(3 downto 0);
 begin

process(phase_count) begin
 case phase_count is
 when "00" => an <= "0111";
 led_in <= num(15 downto 12);
 when "01" => an <= "1011";
 led_in <= num(11 downto 8);
 when "10" => an <= "1101";
 led_in <= num(7 downto 4);
 when "11" => an <= "1110";
 led_in <= num(3 downto 0);
 when others => an <= "1111";
```

```
 end case;
 end process;

CAT: entity work.cathode_reg(reg)
 port map(led_in=>led_in, cat => cat);

end driver;
```

tb_main.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_main is
--  Port ( );
end tb_main;

architecture Behavioral of tb_main is
signal reset,clk: std_logic;
signal an: std_logic_vector(3 downto 0);
signal cat: std_logic_vector(6 downto 0);
begin

dut: entity work.main(Behavioral)
 port map (clk => clk, reset=>reset,
 an => an,
 cat=> cat
 );

clock :process
 begin
```

```
clk <= '0';
wait for 10 ns;
clk <= '1';
wait for 10 ns;
end process;

sim: process
begin
reset <= '1';
wait for 20 ns;
reset <= '0';
wait;
end process;

end Behavioral;
```

tb_clk.vhd

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

entity tb_clk is
--  Port ( );
end tb_clk;

architecture Behavioral of tb_clk is
signal clk,reset: std_logic;
signal clock_count: std_logic_vector(27 downto 0);
signal phase_count: std_logic_vector(1 downto 0);
signal sec: std_logic_vector(15 downto 0);
```

```
signal sec_en: std_logic;
begin

dut: entity work.clock
 port map ( clk => clk, reset => reset, clock_count => clock_count,
 phase_count => phase_count, sec => sec, sec_en => sec_en
 );

 clock: process begin
 clk <= '0';
 wait for 10ns;
 clk <= '1';
 wait for 10ns;
 end process;

 sim: process
 begin
 reset <= '1';
 wait for 20 ns;
 reset <= '0';
 wait;
 end process;

end Behavioral;
```

pin_7seg.xdc

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

Yiğit Narter
22102718
Section 01
Lab Report 5
21.11.2022

```
set_property PACKAGE_PIN V17 [get_ports {reset}]
   set_property IOSTANDARD LVCMOS33 [get_ports {reset}]


set_property PACKAGE_PIN W7 [get_ports {cat[6]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cat[6]}]
set_property PACKAGE_PIN W6 [get_ports {cat[5]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cat[5]}]
set_property PACKAGE_PIN U8 [get_ports {cat[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cat[4]}]
set_property PACKAGE_PIN V8 [get_ports {cat[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cat[3]}]
set_property PACKAGE_PIN U5 [get_ports {cat[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cat[2]}]
set_property PACKAGE_PIN V5 [get_ports {cat[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cat[1]}]
set_property PACKAGE_PIN U7 [get_ports {cat[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {cat[0]}]
set_property PACKAGE_PIN U2 [get_ports {an[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
```