March 26th 2025

Alp Efe Kılıçarslan, 22402390

EEE 102-2

LAB-05: Seven-Segment Display

Purpose:

The purpose of this experiment is to design and implement a number displayer dependent on an internal clock which outputs it's display onto the integrated seven-segment display on the BASYS 3. The final design should show different digits simultaneously.

Design Specifications:

The circuit will cycle through numbers at a rate of one per second and display them on the designated seven-segment display and will be reset according to a predetermined switch state. The design will use the default internal clock of the BASYS 3 as it's clock input. Hence, in total there will be two inputs; the clock and the reset switch. Depending on these inputs, the seven-segment display will be dependent on the anode and the cathode states, which are designated four bits and seven bits respectively. To implement the required circuit, a modular design will be used. A main top module will control the submodules who will control smaller parts of the circuit. A clock and a driver module will be created to control said components. The clock module will be controlled by the BASYS 3's default frequency and the reset switch and will output the number of seconds, clock ticks for testing and an enable. The driver module will take these outputs as its own inputs and output the required states for the anode and the cathode which will control the seven-segment display.

Using these modules, a simulation will be run on a testbench to gather data for the ideal cases. The gathered information will be compared to real-world states of the designed circuit.

Methodology:

Three questions that were asked in the laboratory assignment were researched before formulating any modules. These questions and their answers were:

• What is the internal clock frequency of Basys 3?

- The BASYS 3 has a default clock frequency of a 100MHz. This means that there are one hundred million edge rises and falls per second.

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

- A clock divisor can be used to sample a higher number of ticks per second to formulate a clock signal that actuates in a lesser frequency. This operation can be done with integer division. For instance, to get a 5MHz frequency, we can create a divisor which puts out a rising edge after twenty toggles are measured from the default clock signal.

• Can you create a clock with any arbitrary frequency lower than that of the internal clock? If

not, which frequencies can you create?

- The slower signal cannot have an arbitrary frequency, since the toggle requires a constant number of voltage changes per instance to function properly. Therefore, the slower signal must toggle once per after a set constant integer of edges are received from the default clock signal.

After these answers were found, the following steps were taken in order.

Task-1) The described design was taken as a base and was coded. First of all, the clock module was created. This module took in the default clock signal of the BASYS 3 and converted into a form that updated every one real-life second. After the clock module was formed, the cathode editing submodule was created. This submodule took in a four- bit value that represented a hexadecimal number in binary and outputted a seven-bit value, which were the states of each segment of each digit. Using the cathode setter, a segment driver module was created. The segment driver was able to select the digit to be represented for each possible digit. This was done by assigning the cathode states to the anode positions of the seven-segment display. After all submodules were coded and implemented, a main program was used to combine all modules in a functional manner. To make the code meaningful for the BASYS 3, a constraint file was created and each cathode state was mapped to the related port and the reset switch was mapped to the rightmost switch.

Using the given parameters, a testbench to simulate the design in Vivado was made and simulated. The resulting data was recorded for later use. Additionally, the RTL Schematic for the main circuit was gathered for further analysis.

The BASYS 3 was programmed with the code and the changes were photographed.

Results:

The RTL schematic was gathered for the main module of the code (Figure 1.1). The testbench simulation was ran and analyzed for each state (Figure 1.2-1.3).

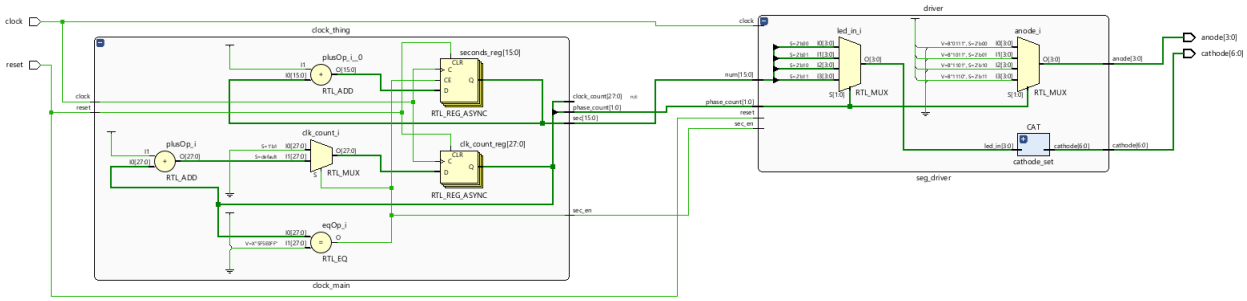

Figure 1.1: RTL Schematic for the Main Module

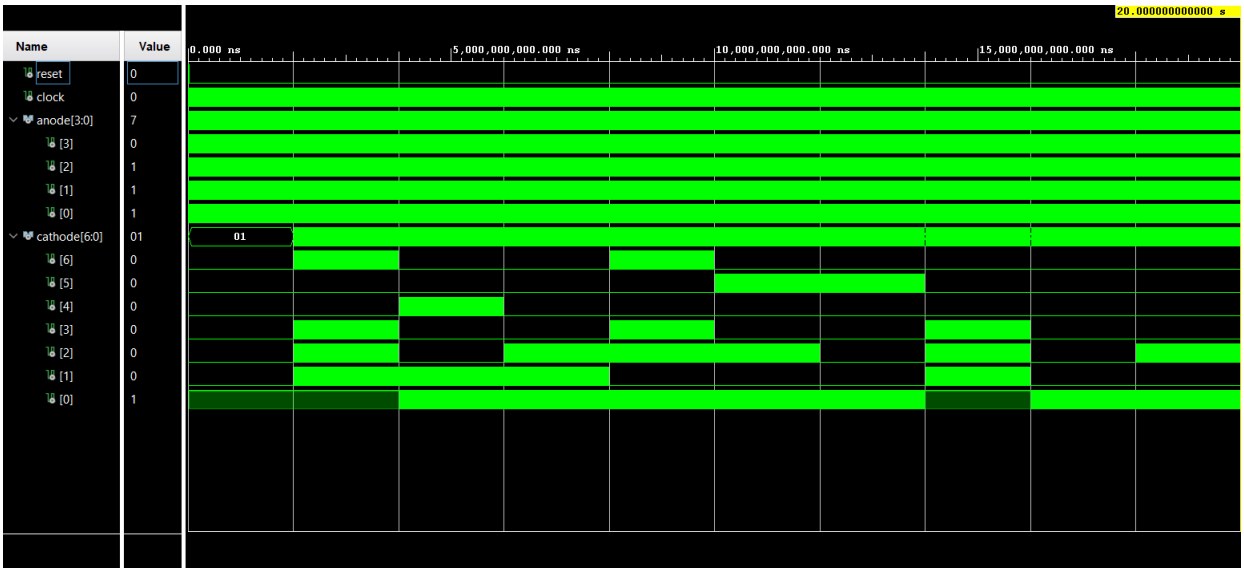

Figure 1.2: Complete Testbench Simulation with Persistence of Vision

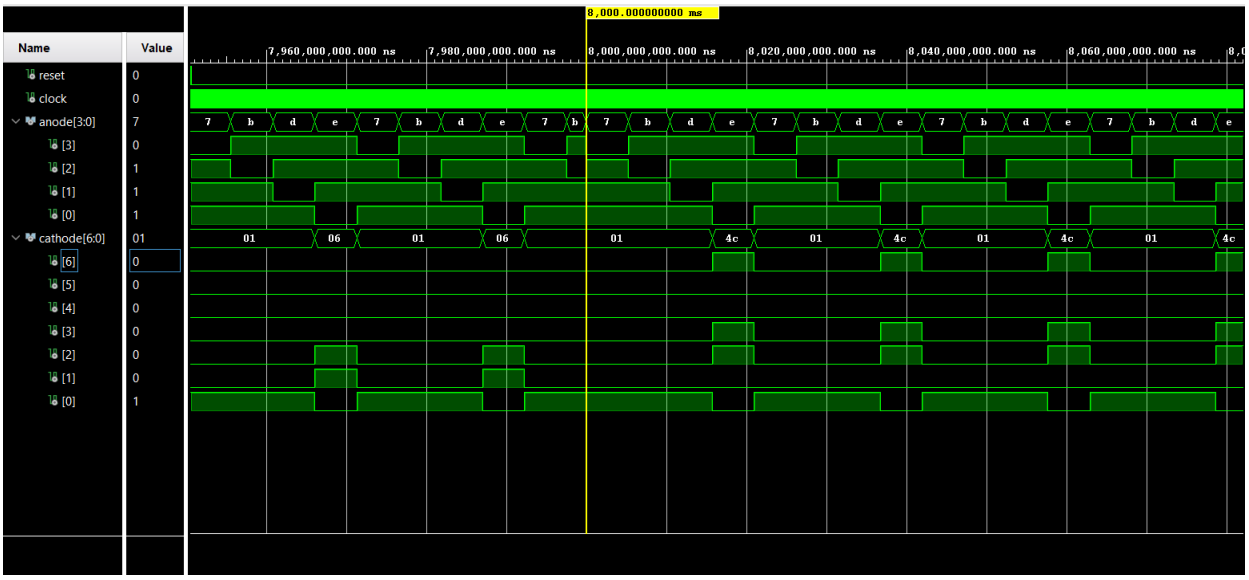

Figure 1.3: Small Section of the Testbench showing periodic triggers

No discrepancies were found in the resulting simulations and a bitstream was generated to implement the code onto the FPGA. After connecting the BASYS 3 and running the code, the digital screen incremented once per second on startup. An increment of one per second was observed on the digital screen from zero in hexadecimal. The output for different states of the screen were observed and correlated with the timeframe (Figure 1.4-1.8). The reset switch was flipped and an immediate drop to zero on the screen was observed (Figure 1.9).

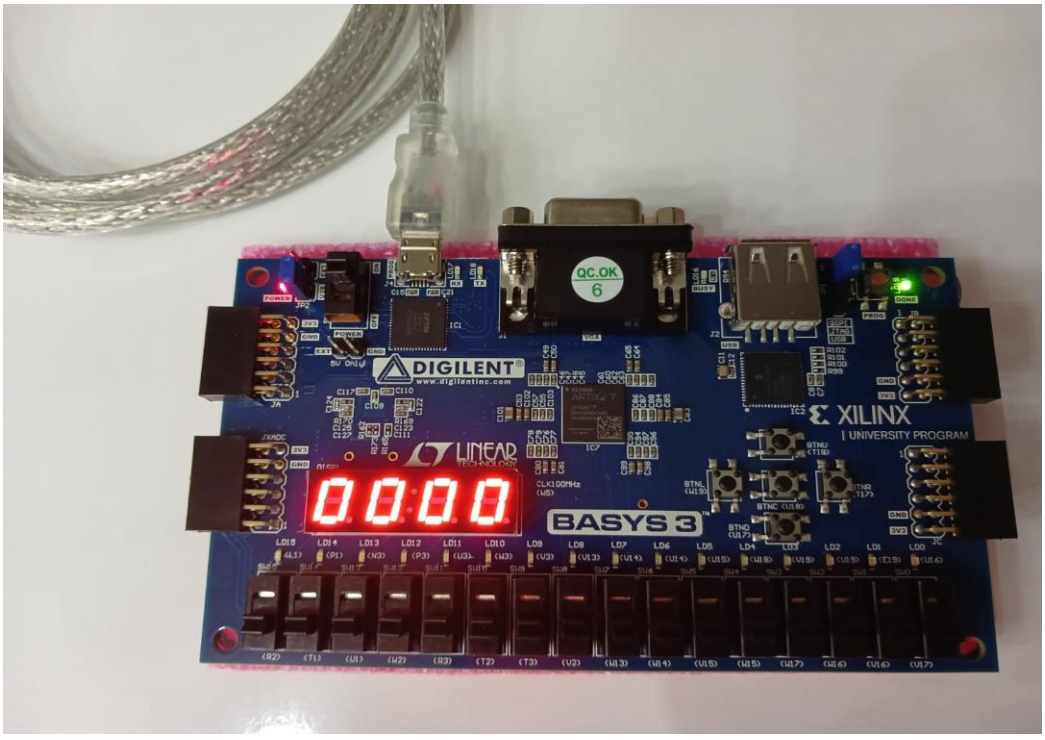

Figure 1.4: “0000” in base 16 on the Seven-Segment Display after booting (zero seconds)

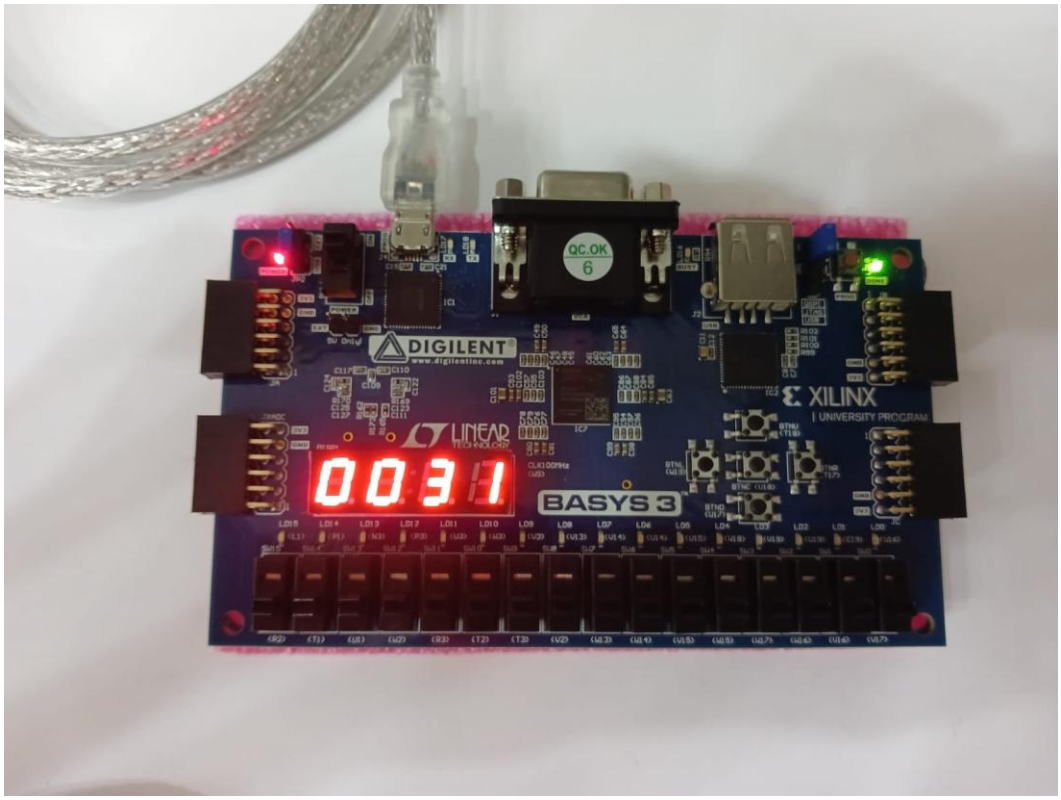

Figure 1.5: “0031” in base 16 on the Seven-Segment Display after 49 seconds

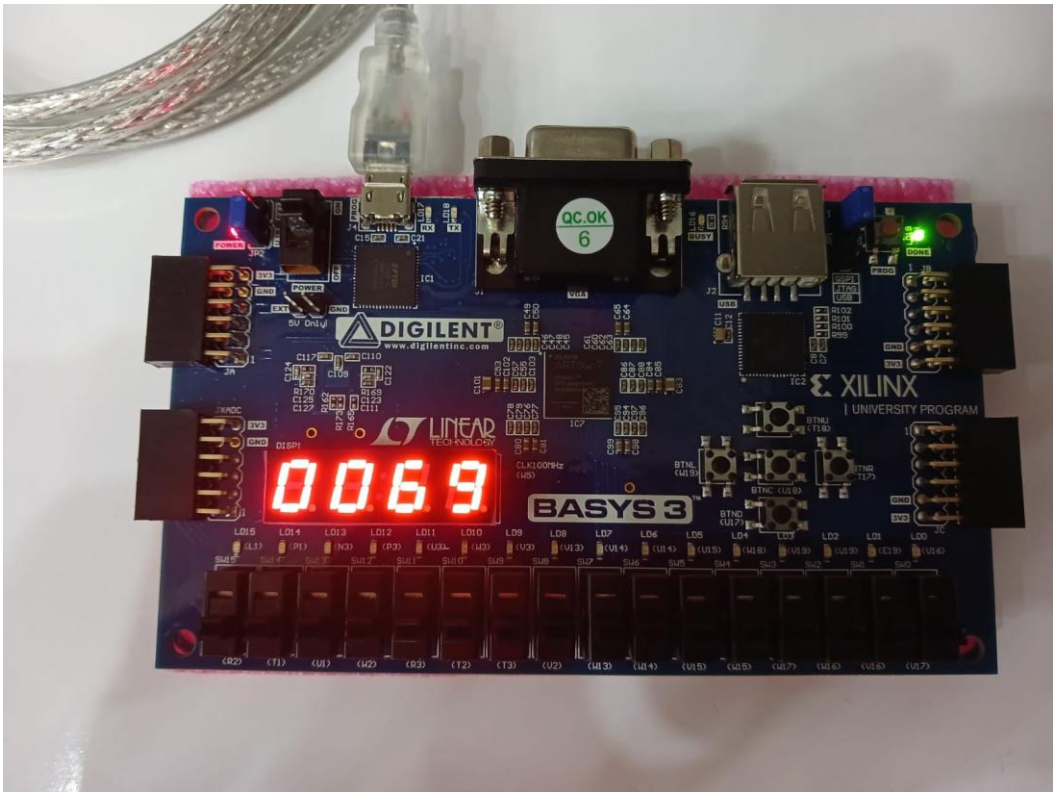

Figure 1.6: “0069” in base 16 on the Seven-Segment Display after 105 seconds

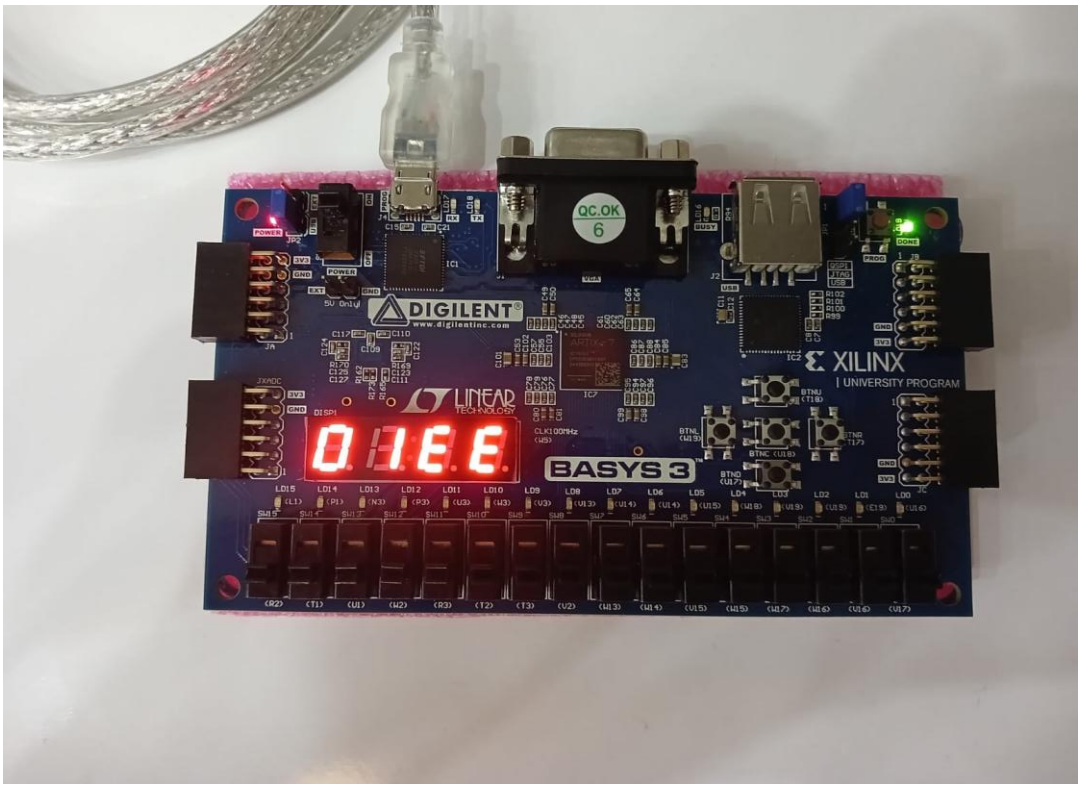

Figure 1.7: “01EE” in base 16 on the Seven-Segment Display after 494 seconds

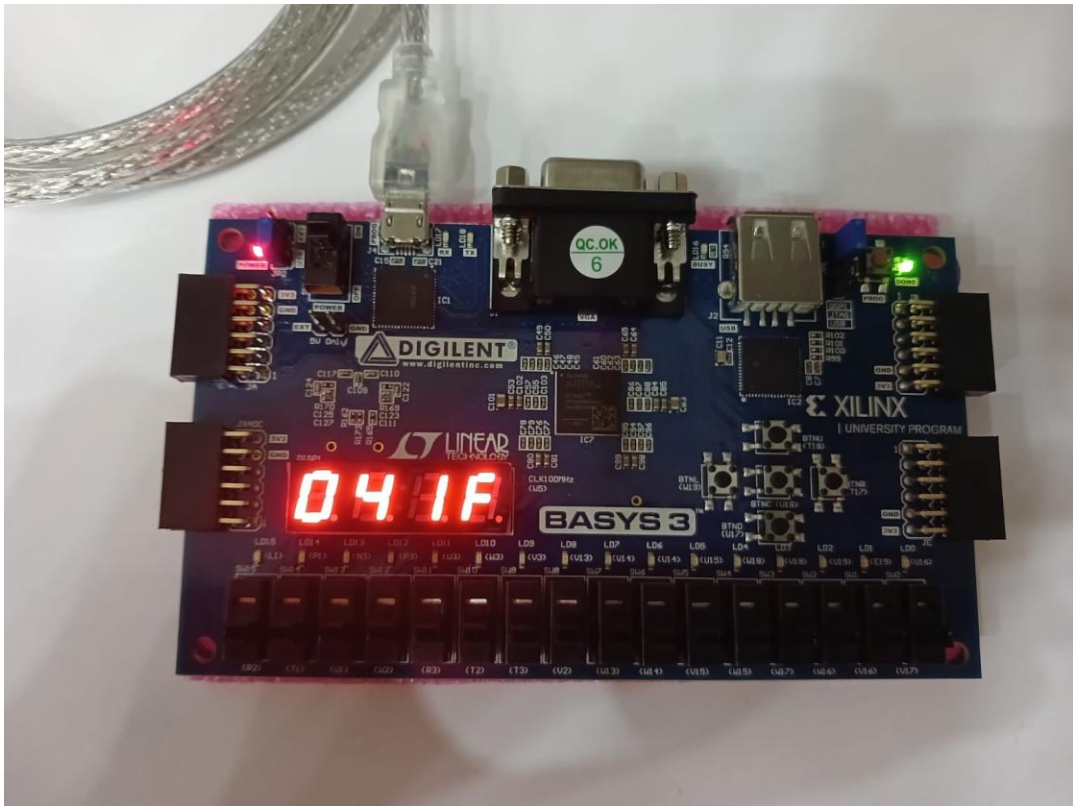

Figure 1.8: “041F” in base 16 on the Seven-Segment Display after 1055 seconds

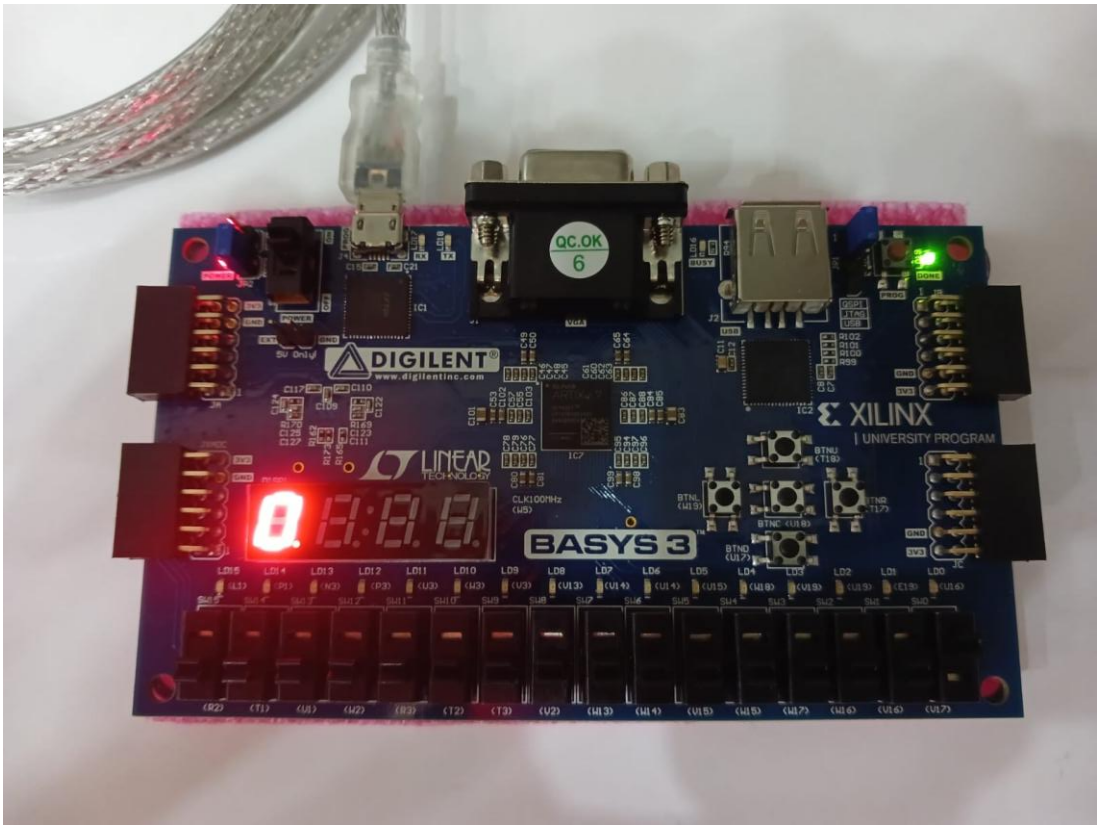

Figure 1.9: “0” on the Seven-Segment Display when the reset switch is flipped

By comparing the real-world results with the ideal cases of the display, no errors were found. Incrementation was implemented and outputted in a functional manner.

Conclusion:

The goal of this lab assignment was to achieve persistence of vision for a seven-segment display and display individual digits using a BASYS 3. To do this, VHDL was used to implement the code for the design. The real-world results and the simulations were found to be equivalent for the display states. Through this experiment, the manipulation of the seven-segment display was learned and clock division was better understood.

References:

All About FPGA. (n.d.). *VHDL code for clock divider*. All About FPGA. Retrieved March 25, 2025, from https://allaboutfpga.com/vhdl-code-for-clock-divider/?srsltid=AfmBOorwiAxYdd6wLCfCQzb2ZSNFojFm0iNlScBjuF6jmGy3rFwnqCLQ

Wikipedia contributors. (n.d.). *Persistence of vision*. Wikipedia, The Free Encyclopedia. Retrieved March 26, 2025, from https://en.wikipedia.org/wiki/Persistence_of_vision

Appendices:

main.vhd

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



entity main is
 Port ( clock : in STD_LOGIC;
 reset : in STD_LOGIC;
 anode : out STD_LOGIC_VECTOR (3 downto 0);
 cathode : 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_thing: entity work.clock_main(clk)
    port map(clock => clock,
    reset => reset,
    phase_count => phase_count,
    sec_en => sec_en,
    sec => num);
```

```
driver: entity work.seg_driver(driver)
    port map(clock => clock,
    reset => reset,
    phase_count => phase_count,
    num => num,
    sec_en => sec_en,
    anode => anode,
    cathode => cathode);

end Behavioral;
```

clock_main.vhd

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


entity clock_main is
 Port ( clock : 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_main;

architecture clk of clock_main is
```

```
    signal clk_count: std_logic_vector(27 downto 0);
    signal seconds: std_logic_vector(15 downto 0):= (others => '0');
begin
    process(clock)
    begin
    if(reset = '1') then
        clk_count <= (others => '0');
        seconds <= (others => '0');
    else
        if rising_edge(clock) 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;
```

seg_driver.vhd

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

```
entity seg_driver is
 Port ( clock : 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;
        anode : out STD_LOGIC_VECTOR (3 downto 0);
        cathode : out STD_LOGIC_VECTOR (6 downto 0));
end seg_driver;

architecture driver of seg_driver is

signal led_in: std_logic_vector(3 downto 0);
begin
process(phase_count) begin

    case phase_count is

        when "00" => anode <= "0111";
    led_in <= num(15 downto 12);
        when "01" => anode <= "1011";
    led_in <= num(11 downto 8);
        when "10" => anode <= "1101";
    led_in <= num(7 downto 4);
        when "11" => anode <= "1110";
    led_in <= num(3 downto 0);
        when others => anode <= "1111";

    end case;
end process;
```

```
CAT: entity work.cathode_set(reg)
port map(led_in=>led_in, cathode => cathode);
end driver;
```

cathode_set.vhd

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


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


architecture reg of cathode_set is
    begin
    process(led_in) begin
    case led_in is
     when "0000" => cathode <= "0000001";
     when "0001" => cathode <= "1001111";
     when "0010" => cathode <= "0010010";
     when "0011" => cathode <= "0000110";
     when "0100" => cathode <= "1001100";
     when "0101" => cathode <= "0100100";
     when "0110" => cathode <= "0100000";
     when "0111" => cathode <= "0001111";
     when "1000" => cathode <= "0000000";
     when "1001" => cathode <= "0000100";
     when "1010" => cathode <= "0000010";
```

```
     when "1011" => cathode <= "1100000";
     when "1100" => cathode <= "0110001";
     when "1101" => cathode <= "1000010";
     when "1110" => cathode <= "0110000";
     when "1111" => cathode <= "0111000";
     when others => cathode <= "1111111";
    end case;
end process;
end reg;
```

testbench_main.vhd

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



entity testbench_main is
end testbench_main;

architecture Behavioral of testbench_main is

    signal reset,clock: std_logic;
    signal anode: std_logic_vector(3 downto 0);
    signal cathode: std_logic_vector(6 downto 0);

    begin
        DUT: entity work.main(Behavioral)
        port map (clock => clock, reset=>reset,
        anode => anode,
        cathode => cathode
        );
        clock_thinger :process
```

```
        begin

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

        simulate: process
        begin
            reset <= '1';
            wait for 10 ns;
            reset <= '0';
        wait;
    end process;
end Behavioral;
```

testbench_clk.vhd

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



entity testbench_clk is
end testbench_clk;



architecture Behavioral of testbench_clk is

    signal clock,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_main
        port map ( clock => clock, reset => reset, clock_count => clock_count,
        phase_count => phase_count, sec => sec, sec_en => sec_en
        );
        clock_thingerooo: process
        begin
        clock <= '0';
        wait for 10 ns;
        clock <= '1';
        wait for 10 ns;
        end process;
        sim: process
        begin
        reset <= '1';
        wait for 10 ns;
        reset <= '0';
        wait;
    end process;
end Behavioral;
```

7seg_pinout.xdc

```
set_property PACKAGE_PIN W5 [get_ports clock]
 set_property IOSTANDARD LVCMOS33 [get_ports clock]
 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clock]
set_property PACKAGE_PIN V17 [get_ports {reset}]
 set_property IOSTANDARD LVCMOS33 [get_ports {reset}]
```

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