## Lab 5: Seven-Segment Display

**Date:** 21.11.2022

**Name:** Orkun İbrahim Kök

**Section:** EE102-01

## Purpose

The purpose of this lab is to understand the working principle behind the seven-segment display and also learning the anode and cathode terms which are related to seven-segment display on Basys3 FPGA.

## Design Specifications

My design is to convert a decimal number to a hexadecimal number and display the output on seven-segment display. Overall, my circuit has 2 input values and 2 outputs values. Input values are clock and reset. The seven-segment display resets to “0” value whenever reset switch is on. Otherwise, clock will be the input value which will correspond to seconds in my circuit design. I used a multiplexer to select which digit to display. I also used a clock divider, which acts like a clock counter, which increments the digit by one on the seven-segment display.

## Methodology

The circuit that I designed converts decimal to hexadecimal numbers. It did this with a clock input. Whenever the clock input is 1, it will be incremented by one every second, thus the second (decimal base) will be converted to hexadecimal numbers and will be displayed on the seven-segment display. I used a multiplexer to select which digit to display. I also used a clock divider, which acts like a clock counter, to increment the digit by one on the seven-segment display. The logic behind the seven-segment display is about lighting up the corresponding LEDs. In order to reach “persistence of vision” concept, the cathode inputs (segments) of the seven-segment display must be turned on and off rapidly so that continuum will be achieved.

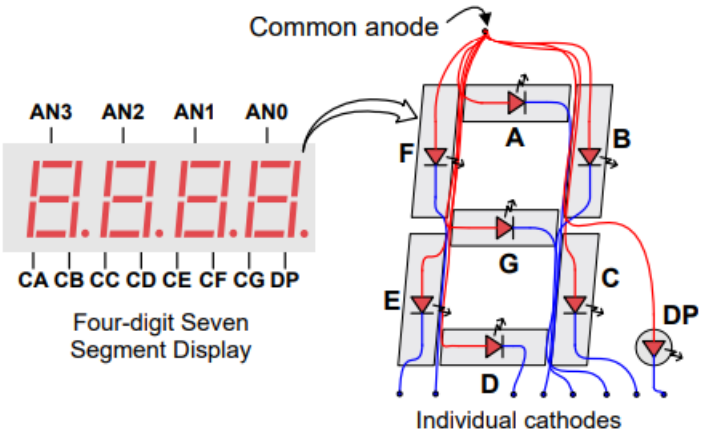


Figure 1: Seven-segment display with anode and cathode nodes

Q1) What is the internal clock frequency of Basys 3?

The internal clock frequency of Basys3 is 100MHz (Figure 2)

Q2) How can you create a slower clock signal from this one?

By using a clock divider, it is possible to make slower clock signal that Basys3 internally has (100MHz). If we choose 25MHz and 4 ticks, these values will give us 100MHz. Therefore a slower clock can be created via clock divider

Q3) Can you create a clock with any arbitrary frequency lower than that of the internal clock? If not, which frequencies can you create?

We cannot create a clock with any arbitrary frequency lower than the internaş clock because the selected tick value will play an important role on clock frequencies. Clock ticks can be named as divisors, therefore, we can only calculate the clock frequencies starting from 0MHz and ending at 100MHz.

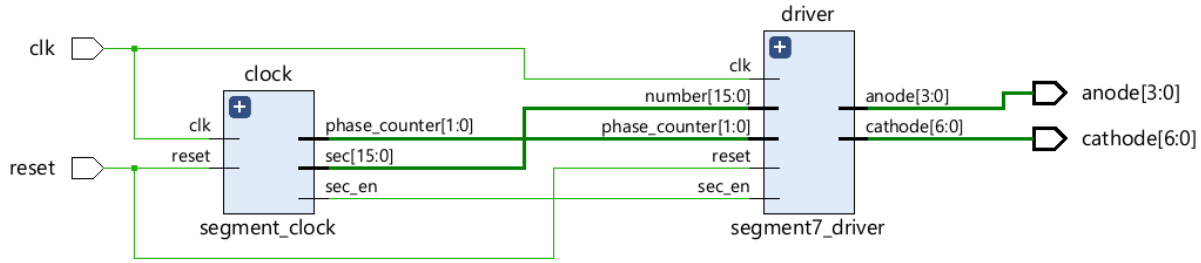


Figure 2.a: RTL Schematic

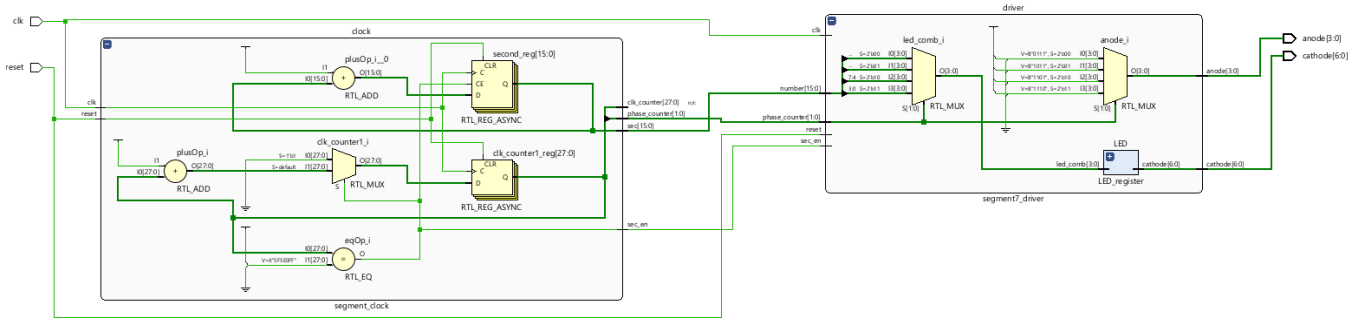

Figure 2.b: RTL Schematic

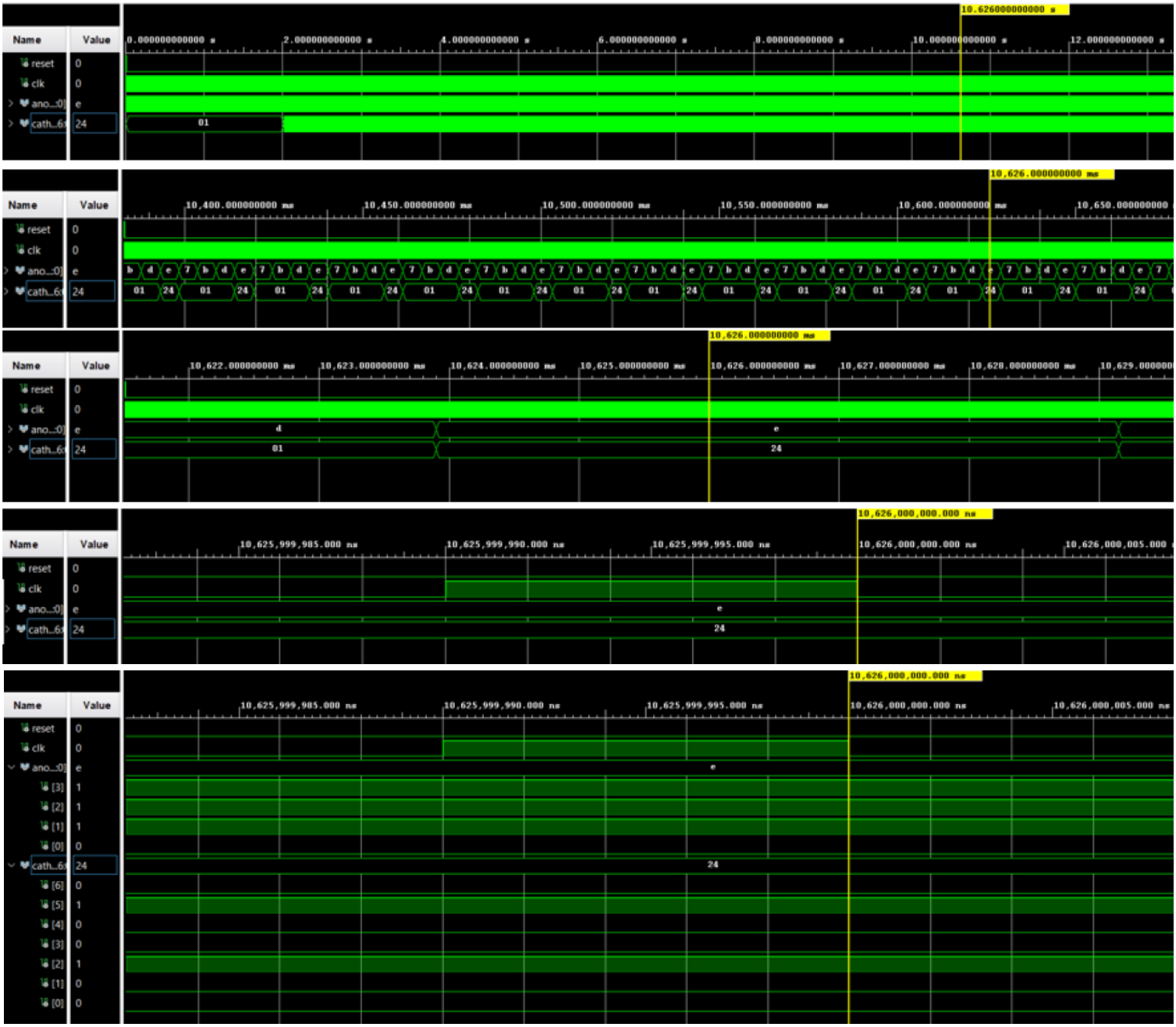


Figure 3: Waveforms

## Results

First I wrote the VHDL codes for the circuit that I designed (Figure 2.a & Figure 2.b), and after that I also wrote a testbench code in order to observe the output values of the circuit (Figure 3). As mentioned, there are only two inputs in my circuit, reset and clock. Reset input is well-working, when it becames on, the seven-segment display resets its value to 0 (Figure 4). The clock input is toggling at every 10ns, thus the concept “persistence of vision” is achieved. After that, I took photos of my working FPGA and recorded the values (Figure 5.a & 5.b). Overall, I made a working decimal to hexadecimal converter and displayed the result via seven-segment display on Basys3 FPGA.

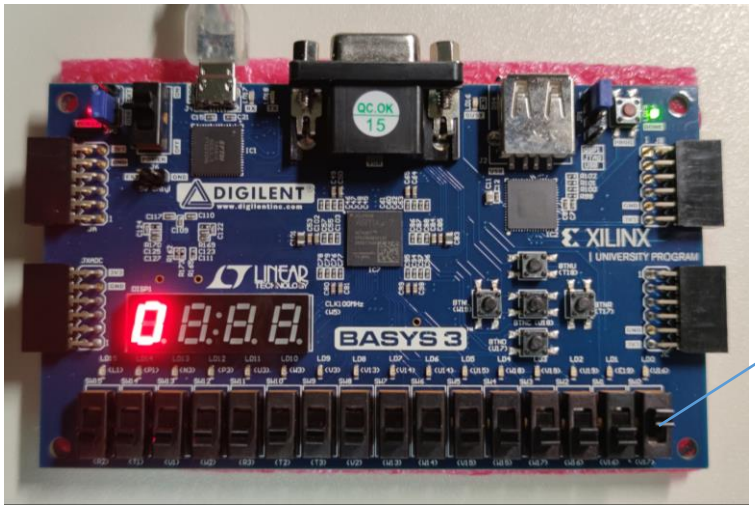


Figure 4: Reset value is set to 1

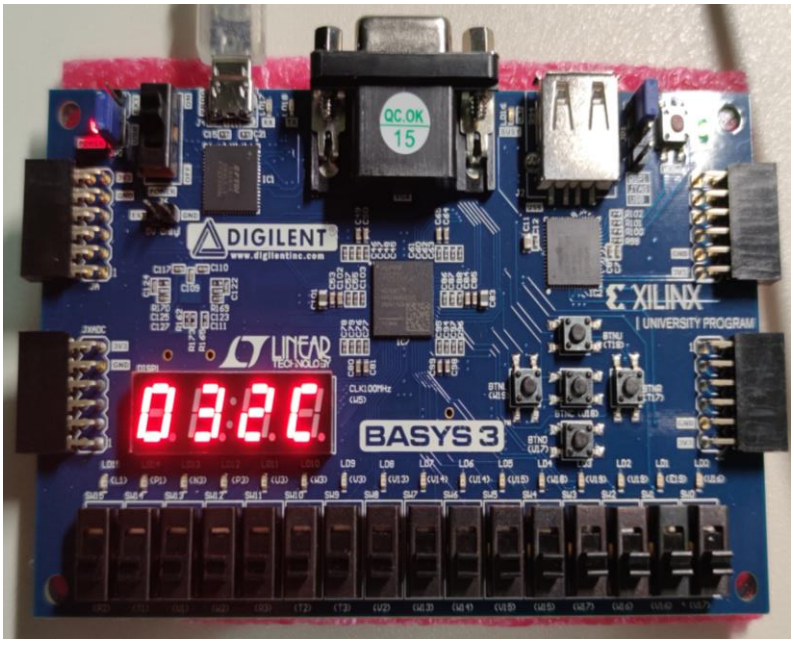


Figure 5.a: Clock input is 1 and after 812 second

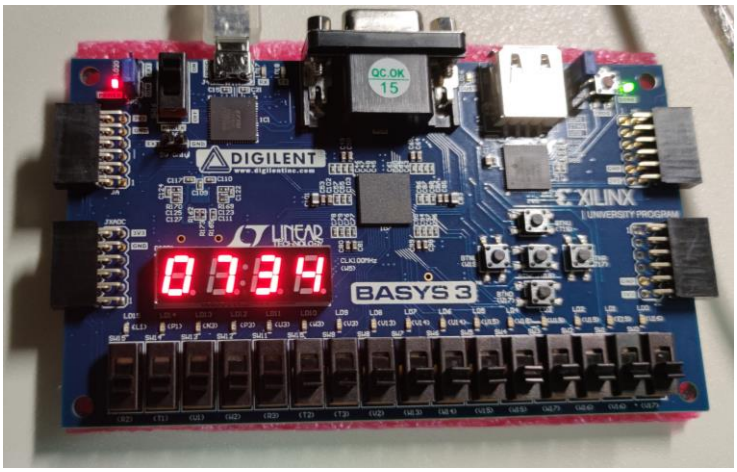


Figure 5.b: Clock input is 1 and after
1844 second

## Conclusion

The purpose of the experiment is to understand the logic behind the seven-segment display and learn how to implement it in VHDL. The circuit that I designed work well and it was successful at the end. While observing the anode and cathode nodes of seven-segment display, first it is hard for me to understand which input will g oto which node, but then I figured it out. Now, I learnt how to use a seven-segment display in a digital circuit and combine it with other circuit elements, in my case, which are multiplexer and clock divider.

# Appendices

**seven_seg.vhd**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
----------------------------------------------------------------------------
entity seven_seg is
Port ( clock: in std_logic;
reset: in std_logic;
and: out std_logic_vector(3 downto 0);
cat: out std_logic_vector(6 downto 0)
);
end seven_seg;
architecture rtl of seven_seg is
signal phasecount: std_logic_vector(1 downto 0);
signal sec_en: std_logic;
signal sec: std_logic_vector(15 downto 0);
begin
clock: entity work.clkseg(rtl_clock)
port map(clock => clock, reset => reset,
phasecount => phasecount,
sec_en => sec_en,
sec => sec);
driver: entity work.segment7_driver(rtl_driver)
port map(clock => clk, reset => reset, phasecount => phasecount, sec => sec,
sec_en => sec_en, and => and, cat => cat);
end rtl;
```

**7segdriver.vhd**

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

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
entity 7segdriver is
Port (clock: in std_logic;
reset: in std_logic;
phasecount: in std_logic_vector(1 downto 0);
sec in std_logic_vector(15 downto 0);
sec_en: in std_logic;
and : out std_logic_vector(3 downto 0);
cat: out std_logic_vector(6 downto 0)
);
end 7segdriver;
architecture rtl_driver of 7segdriver is
signal ledcom: std_logic_vector(3 downto 0);
begin
----------------------------------------------------------------------------
process(phasecount) begin
case phasecount is
when "00" => and <= "0111";
ledcom <= sec(15 downto 12);
when "01" => and <= "1011";
ledcom <= sec(11 downto 8);
when "10" => and <= "1101";
ledcom <= sec(7 downto 4);
when "11" => and <= "1110";
ledcom <= sec(3 downto 0);
when others => and <= "1111";
end case;
end process;
LED: entity work.LED_register(rtl_LED)
port map(ledcom=>ledcom, cat => cat);
```

```
end rtl_driver;
```

**clkseg.vhd**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD;
entity clkseg is
Port ( clock: in std_logic;
reset: in std_logic;
clockcount: out std_logic_vector(27 downto 0);
phasecount: out std_logic_vector(1 downto 0);
sec: out std_logic_vector(15 downto 0);
sec_en: out std_logic);
end clkseg;
architecture rtl_clock of clkseg is
signal clockcounter0: std_logic_vector(27 downto 0);
signal secd: std_logic_vector(15 downto 0):= (others => '0');
begin
process(CLK) begin
if(reset = '1') then
clockcounter0 <= (others =>   '0');
secd <= (others => '0');
elsewqr
if rising_edge(CLK) then
clockcounter0 <= clockcounter0 + '1';
clockcounter0 =x"5F5E0FF" then
secd <= secd + '1';
clockcounter0 <= (others => '0');
end if;
end if;
end if;
```

```
end process;
sec_en <= '1' when clockcounter0 =x"5F5E0FF" else '0';
phasecount <= clockcounter0 (19 downto 18);
clockcount <= clockcounter0;
sec <= secd;
end rtl_clock;
```

**LEDreg.vhd**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity LEDreg is
Port ( ledcom: in std_logic_vector(3 downto 0);
cat: out std_logic_vector(6 downto 0));
end LEDreg;
architecture rtl_LED of LEDreg is
begin
process(ledcom) begin
case ledcom 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;
end rtl_LED;
```

**test_bench1.vhd**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity test_bench1 is
end test_bench1;
architecture testb of test_bench1 is
signal reset: std_logic;
signal clock: std_logic;
signal and: std_logic_vector(3 downto 0);
signal cat: std_logic_vector(6 downto 0);
begin
dut: entity work.seven_seg(rtl)
port map (clock => clock, reset=>reset,
and => and,
cat=> cat
);
clock_process :process
begin
clock <= '0';
wait for 10 ns;
clock <= '1';
wait for 10 ns;
end process;
stim_proc: process
begin
```

```
reset <= '1';
wait for 20 ns;
reset <= '0';
wait;
end process;
end testb;
```