Bilkent University

EE102-02 Lab 5 Report:

Seven-Segment Display

Mehmet Emre Uncu - 22003884

Purpose:

The purpose of this lab is to use multiple seven-segment display simultaneously in BASYS-3 FPGA.

Q&A:

- What is the internal clock frequency of Basys3?
  - BASYS3 has an internal clock frequency of 100MHZ
- How can you create a slower clock signal from this one?
  - The clock divider, whose ticks will total 100 M, can be used. For example, a 1MHZ clock can be made from 100MHZ if we toggle for every 100 ticks of the 100MHZ clock.
- 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 impossible to create a clock with any arbitrary frequency lower than that of the internal clock. Because the number of ticks (divisor) cannot be

anything other than an integer value. Thus 100MHZ-1HZ clock can be made by dividing 100MHZ by 1 to 100M.

Methodology:

The initial task was to decide which function would be implemented. After some research, it was decided to make a hexadecimal up-counter. Afterward, the working principle of the seven-segment display was investigated. The seven-segment display consists of seven LEDs in each segment and anodes and cathodes connected to their ends (Figure 1). The anodes drive the segments, and the cathodes drive the LEDs. The “persistence principle” is used to light multiple segments. Simply this principle means that segments are turned on and off very quickly. 100MHZ was selected as the clock frequency. According to all these, after generating the VHDL code, it will be simulated with testbench. Finally, the design will be observed via BASYS3.

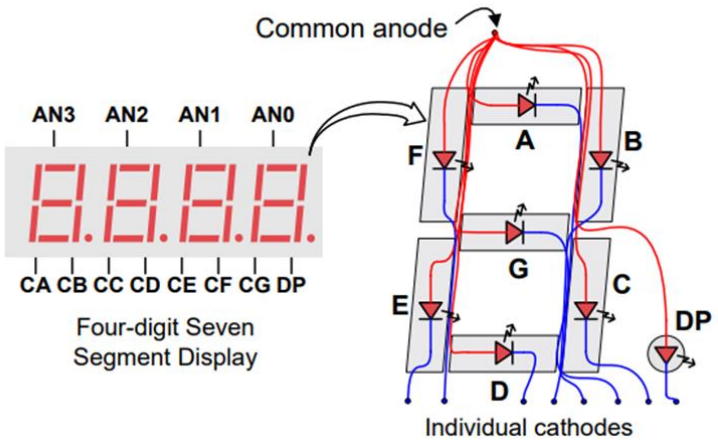


Figure 1 Seven-segment display

Design Specifications:

In the ‘segment’ top module (Figure 2), the clock is the first input, 100MHZ. The second input is reset. When it is active, the clock and display start again from 0. The outputs, anode and cathode are chosen to light up LEDs.

In the “clock” sub-module (Figure 3), there are two inputs and three outputs:

- Clk: input, 100MHZ
- Reset: input, to restart the counting
- Count_phase: output, the on-off sequence of the segments was produced by dividing the phase clock from the main clock
- Second: output, second counter
- Scnd: output, in each second it is 1

The ‘phase_counter’ output can be found by indexing the counter between the 19th and 18th bits, which is 10.5ms. The ‘second’ output is increased by one when the counter reaches the value of one second. The ‘scnd’ output is also asserted at 1. The outputs of this module are imputted to the ‘driver’ sub-module (Figure 4), which has five inputs and two outputs:

- Clk: input, 100MHZ
- Reset: input, to restart the counting
- Count_phase: input, the on-off sequence of the segments was produced by dividing the phase clock from the main clock
- Second: input, ‘second’ output from clock module
- Scnd: input, in each second it is 1
- Anode: output, 4-bit output that drives the segments
- Cathode: output, 7-bit output that drives the LEDs

The ‘driver’ module has ‘LED’ sub-module that has one input and one output:

- Combination: input, hexadecimal number
- Cathode: output, the cathode combination

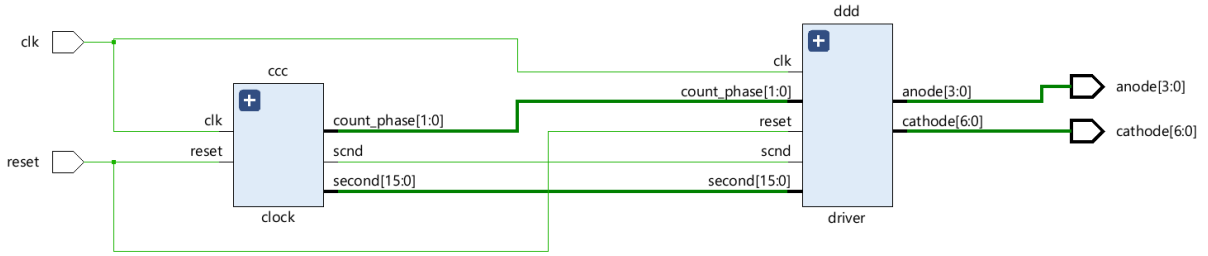


Figure 2 Schematic of 'segment' top module

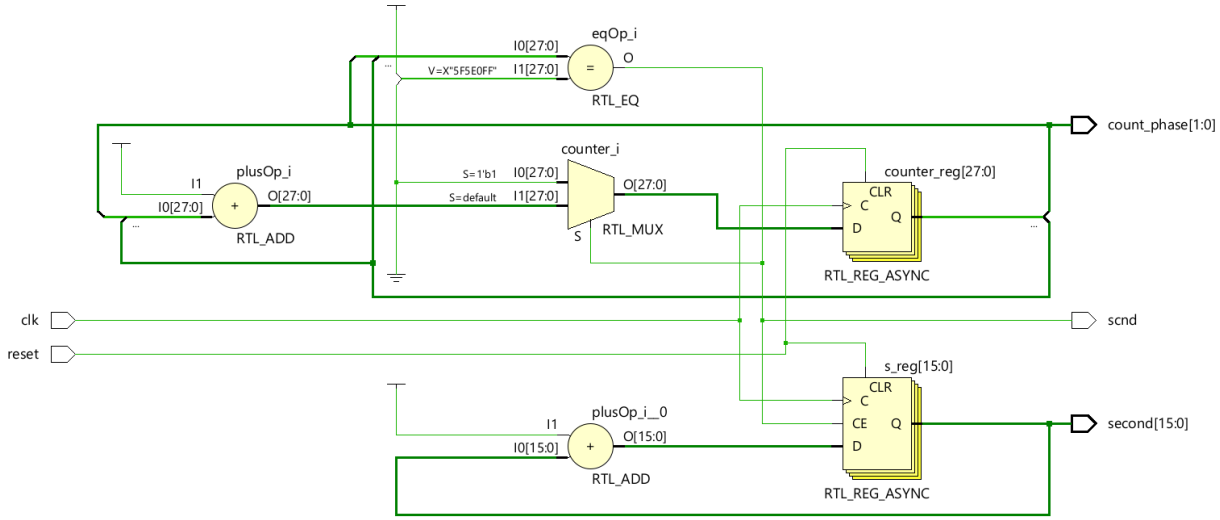


Figure 3 Schematic of 'clock' sub-module

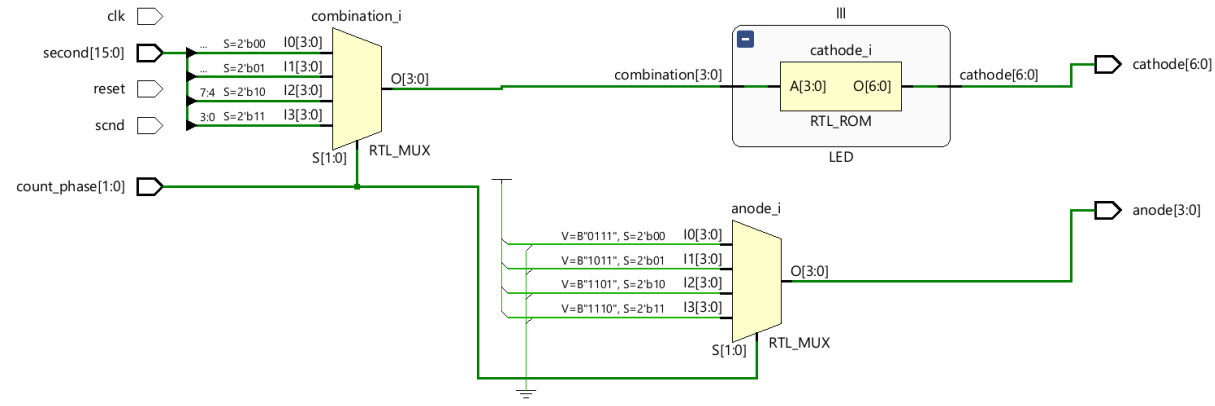
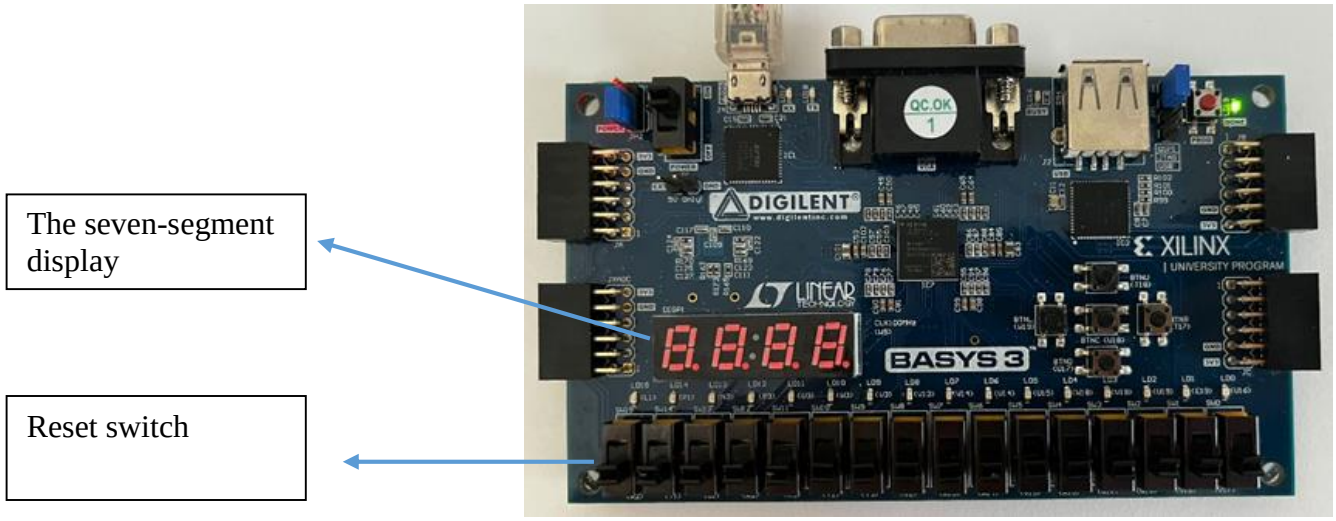


Figure 4 Schematic of 'driver' and 'LED' sub-modules

BASYS3 FPGA was programmed via a constraints file for implementation according to the following assignments:



Results:

After the design, a testbench was written, and a waveform was created for the top module to check if it works properly. The phasor starts from ‘00’ and goes until ‘11’. After this, it restarts, and the phase was picked between the 18th and 19th digits of the counter, which is 10.5ms. This refresh rate was suitable for obtaining the “persistence principle.” The reset is 1 until the time reached 1 second, so the cathode combination is ‘0000001’ and ‘0’ displayed. After that, the number shown in the seven-segment display increased each second, and accordingly, the cathode combination changed. The waveform (Figure 5) of the testbench is shown below:

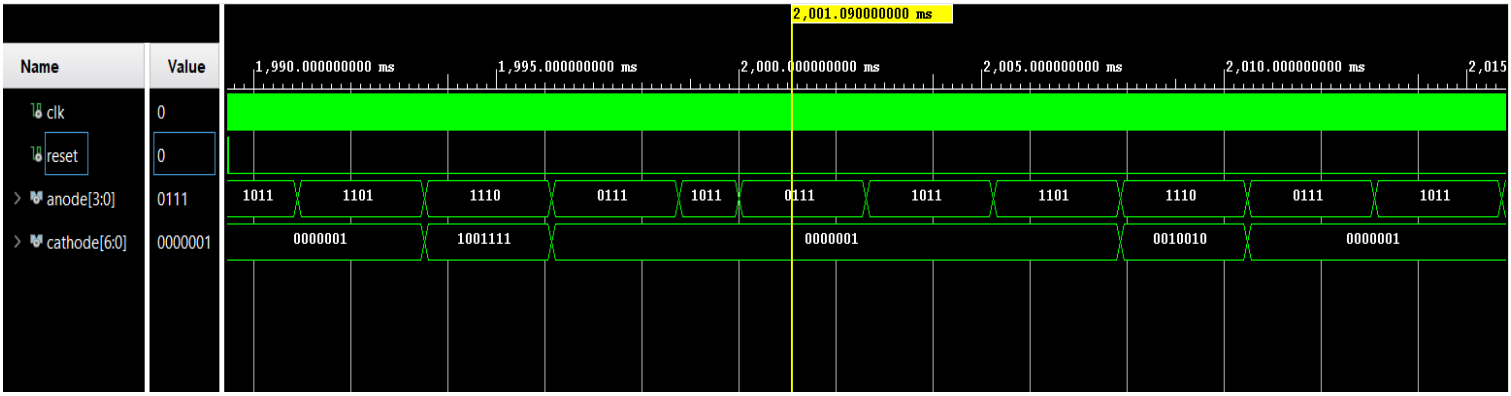
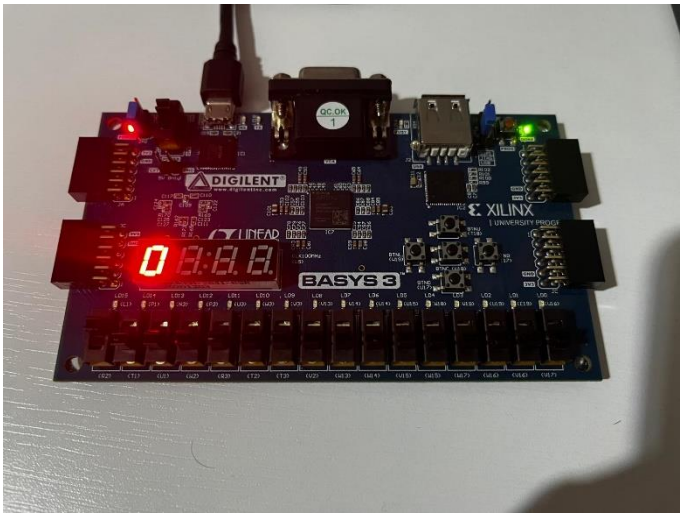
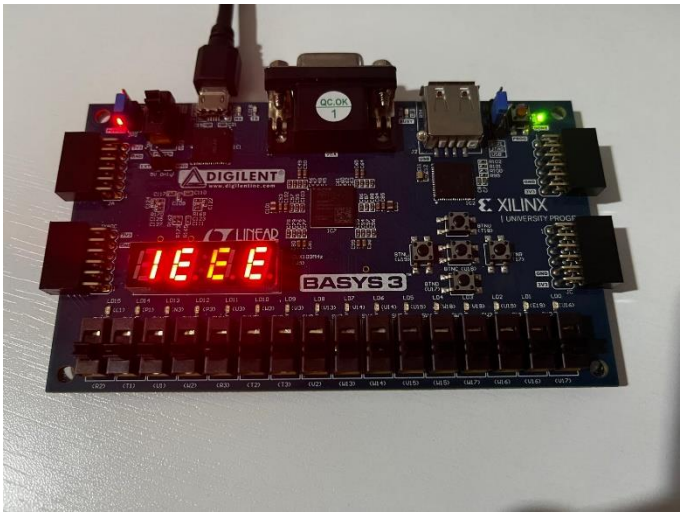


Figure 5 Cathode combinations '1001111' for '1' & '0010010' for '2'

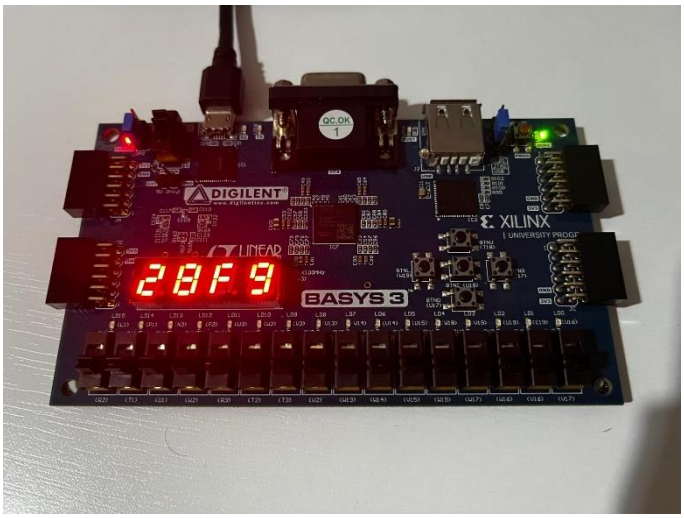
When reset switch is on the seven-segment display shows ‘0’



After 7918 seconds, ‘1EEE’ in hexadecimal base



After 10489 seconds, ‘28F9’ in hexadecimal base



Conclusion:

The purpose of this lab was to design a seven-segment display driver on BASYS3. With this lab, clock usage and how to divide the clock frequency to achieve the persistence of the vision effect were learned. The outputs are compared with the simulation's waveform and the expected results, so the design is verified. No errors were encountered in this lab, except for human errors that occurred during the design. These errors were fixed by schematizing and simulating with the Vivado.

Appendix:

SEGMENT.vhdl

```
entity segment is
    Port ( clk, reset : in std_logic;
           anode : out std_logic_vector (3 downto 0);
           cathode : out std_logic_vector (6 downto 0));
end segment;

architecture Behavioral of segment is

component clock
    port ( clk, reset : in std_logic;
           count_phase : out std_logic_vector (1 downto 0);
           second : out std_logic_vector (15 downto 0);
           scnd : out std_logic);
end component;

component driver
    port ( clk, reset : in std_logic;
           count_phase : in std_logic_vector (1 downto 0);
           second : in std_logic_vector (15 downto 0);
           scnd : in STD_LOGIC;
           anode : out std_logic_vector (3 downto 0);
           cathode : out std_logic_vector (6 downto 0));
end component;

signal count_phase: std_logic_vector(1 downto 0);
signal second: std_logic_vector(15 downto 0);
signal scnd: std_logic;
```

```
begin

ccc: clock
   port map (clk, reset, count_phase, second, scnd);

ddd: driver
   port map (clk, reset, count_phase, second, scnd, anode, cathode);

end Behavioral;
```

CLOCK.vhdl

```
entity clock is
   Port ( clk, reset : in std_logic;
          count_phase : out std_logic_vector (1 downto 0);
          second : out std_logic_vector (15 downto 0);
          scnd : out std_logic);
end clock;

architecture Behavioral of clock is

signal counter: std_logic_vector(27 downto 0);
signal s: std_logic_vector(15 downto 0):= (others => '0');

begin
process(clk) begin
   if(reset = '1') then
      counter <= (others => '0');
      s <= (others => '0');
   else
      if rising_edge(clk) then
         counter <= counter + '1';
```

```
            if counter = x"5F5E0FF" then
                s <= s + '1';
                counter <= (others => '0');
            end if;
        end if;
    end if;
end process;

scnd <= '1' when counter =x"5F5E0FF" else '0';
count_phase <= counter (19 downto 18);
second <= s;

end Behavioral;
```

DRIVER.vhdl

```
entity driver is
    Port ( clk, reset : in std_logic;
           count_phase : in std_logic_vector (1 downto 0);
           second : in std_logic_vector (15 downto 0);
           scnd : in STD_LOGIC;
           anode : out std_logic_vector (3 downto 0);
           cathode : out std_logic_vector (6 downto 0));
end driver;

architecture Behavioral of driver is

component LED
    port ( combination : in std_logic_vector (3 downto 0);
           cathode : out std_logic_vector (6 downto 0)  );
end component;
```

```
signal combination: std_logic_vector (3 downto 0);

begin
process(count_phase) begin
    case count_phase is
        when "00" => anode <= "0111";
            combination <= second (15 downto 12);
        when "01" => anode <= "1011";
            combination <= second (11 downto 8);
        when "10" => anode <= "1101";
            combination <= second (7 downto 4);
        when "11" => anode <= "1110";
            combination <= second (3 downto 0);
        when others => anode <= "1111";
    end case;
end process;

lll: LED
    port map(combination, cathode);

end Behavioral;
```

LED.vhdl

```
entity LED is
    Port ( combination : in std_logic_vector (3 downto 0);
           cathode : out std_logic_vector (6 downto 0)  );
end LED;

architecture Behavioral of LED is

begin
```

```
process(combination) begin
    case combination 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 Behavioral;
```

TESTBENCH

```
entity segment_test is
end segment_test;

architecture Behavioral of segment_test is

component segment
```

```
    port ( clk, reset : in std_logic;
           anode : out std_logic_vector (3 downto 0);
           cathode : out std_logic_vector (6 downto 0));
end component;

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

begin

karim: segment
    port map (clk, reset, anode, cathode);

pro1 :process
begin
    clk <= '0';
    wait for 10 ns;
    clk <= '1';
    wait for 10 ns;
end process;

pro2: process
begin
    reset <= '1';
    wait for 20 ns;
    reset <= '0';
    wait;
end process;
end Behavioral;
```