Report to Lab 5:
Seven Segment Display

Arda Özkut
22101727
Section 2
Physics Department, Ihsan Doğramacı Bilkent University
(Dated: November 21, 2022)

I. PURPOSE OF THE LAB ASSIGNMENT

Designing a seven segment display driver that lights up LEDs and familiarising oneself with the concept of persistence of vision and how it applies to the refresh rate of the seven segment display

II. METHODOLOGY

Each seven segment display are driven by an anode and a cathode. The shared anode enable the segments while the cathodes enable individual leds. Due to this fact it is not possible to drive more than one seven segment display. That's why one has to light up the displays one at a time in a circular manner that has the frequency called the refresh rate of the display. For specific refresh rate, the sampling rate of the eye is surpassed which creates what is called "persistence of vision".

First the driver which selects the anodes must be designed. Then a LED driver that selects the LEDs according to the number that is incremented by the clock pulse should be designed. At the end, should be written a test bench to test the signals produced by the signals should be designed.

When designing a clock of desired period one needs to use a clock divider unit that inputs the internal clock and outputs frequency f such that 100 is divisible by f without remainder. This is because a clock divider unit uses the internal clock of 100Mhz to toggle between pulses of the clock with desired frequency.

III. DESIGN SPECIFICATIONS

As can be seen in [FIG. 1] the design has two inputs and two outputs. Clock is the internal 100Mhz oscillator and the second input 'res', is the reset signal that resets the counting operation. The outputs, anode and cathode outputs which are 4 and 7 bits respectively are signals that are created according to the clock.

The implementation is designed in a hierarchical manner. The modules are described bellow.

1. clk_seg.vhd

clock (in): The internal clock of 100Mhz frequency.
res (in): reset signal
clocount (out, 28 bits): counts the clock tics
digit (out, 2 bits): determines the on-off sequence of digits (enables anodes)
sec (out, 16 bits): counts seconds, (nnnn) or 0xn for each digit
sec_is (out): each second it is 1, between it is 0

2. dri7er.vhd

clock (in): The internal clock of 100Mhz frequency.
res (in): switch input for resetting the signal

digit (in): digit enable signal (enables anodes)
num_hex (in, 16 bits): second input from the clock module
sec_is (in): each second it is 1, between it is 0
cathode (out, 7 bits): drives the LEDs of each segment
anode (out, 4 bits): drives the anodes or digits

3. led_reg.vhd

combof_LED (in, 4 bits): combinations of numbers to display in each segment
cathode (out, 7 bits): cathode combinations (0 for LED to turn on 1 to turn the LED off)

The int_clock block which is created inside the main module is an entity of the clk_seg.vhd RTL design of which can be seen in [FIG. 2]. The drv block [FIG. 3], also created in the top module, drives the anode and cathode signals according to the input of int_clock and the internal 100Mhz clock of the BASYS 3 board. The LED block [FIG. 3] which is basically a Read-only-Memory that contains the values of cathode signals determines which LEDs to light up according to the corresponding combof_LED signal.
I also wrote a test bench of the main module to see if the module is working properly before loading the bitstream to the FPGA.

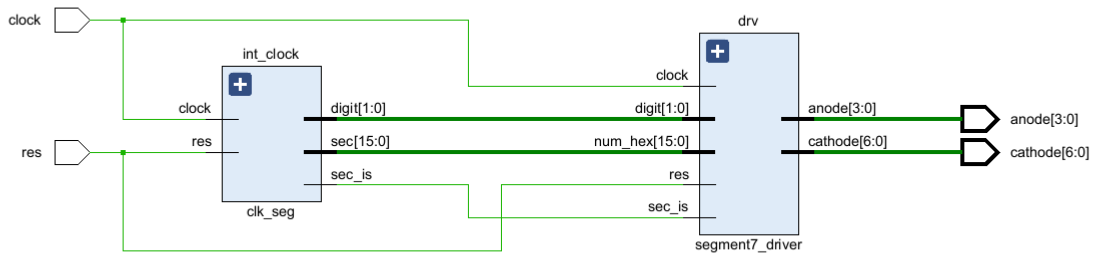


FIG. 1: RTL Schematic

IV. RESULTS

As can be seen from figures 1, 2 and 3 the implementation was successful. From figures 6, 7 and 8 the implementation onto the FPGA board was successful as well. Simulation results can be observed in figure 4 however for the simulation to run faster I chose the time between the pulses to be 1 ns, this can be seen in Appendix A under tb_top.vhd. The correct time for the pulse duration was calculated but never used. In figure 4 since the waveform spans over 2.5 seconds individual pulses are not seen. If we zoom in, in figure 5, we can see individual anode and cathode vector signals in an expanded way.

V. CONCLUSION

The purpose of this lab was to design a driver for the 7-segment display on the BASYS 3 board. The expected results are achieved both with the implementation and the simulation. I have tried other values for the refresh rate which gave a counter that "flickers" because the persistence of vision is not achieved. This way I learned what persistence of vision is and how monitors and refresh rates has to do with that concept. I have learned how to drive a 7-segment display as well as how ROM are created using hardware description languages.

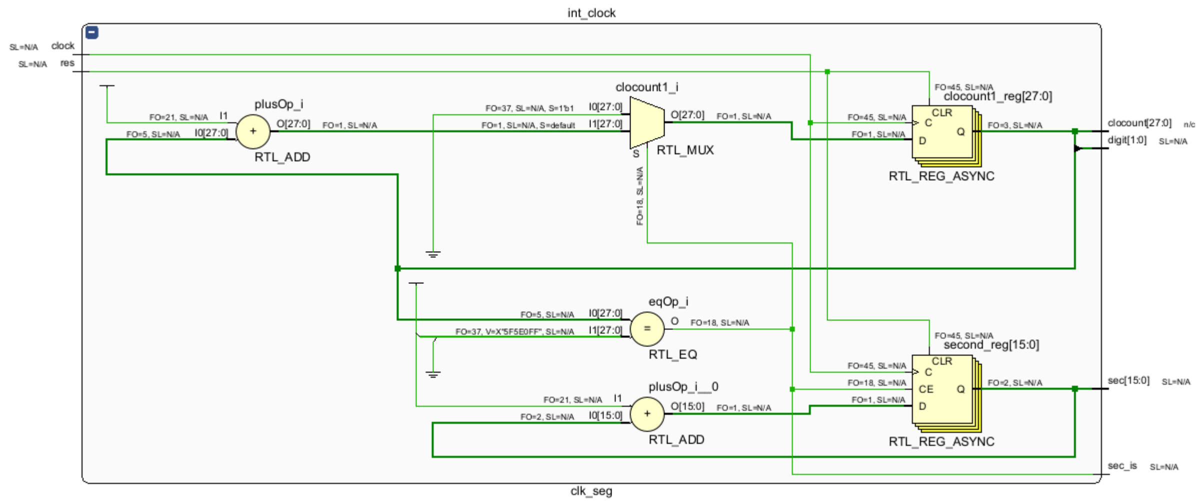


FIG. 2: expanded RTL Schematic of int_clock

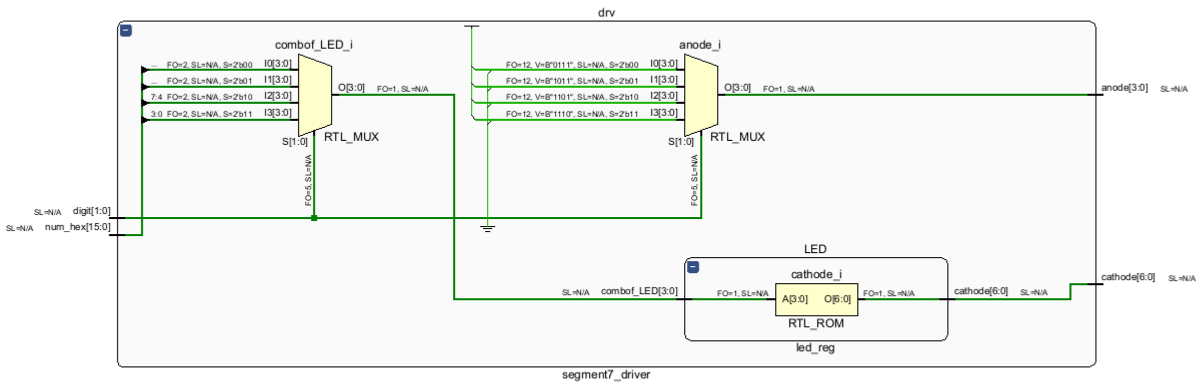
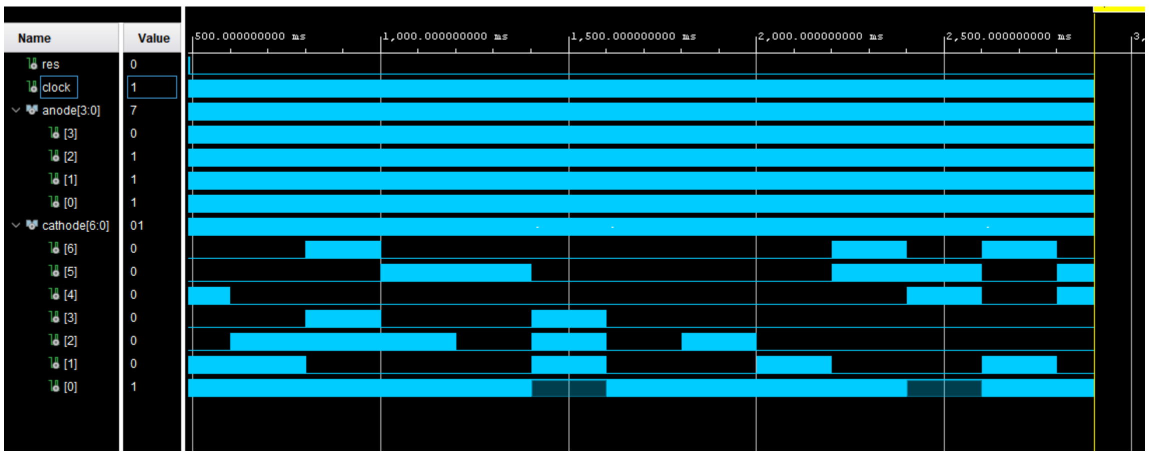


FIG. 3: expanded RTL Schematic of drv and LED



Name	Value
res	0
clock	1
anode[3:0]	7
[3]	0
[2]	1
[1]	1
[0]	1
cathode[6:0]	01
[6]	0
[5]	0
[4]	0
[3]	0
[2]	0
[1]	0
[0]	1

FIG. 4: Test Bench Results

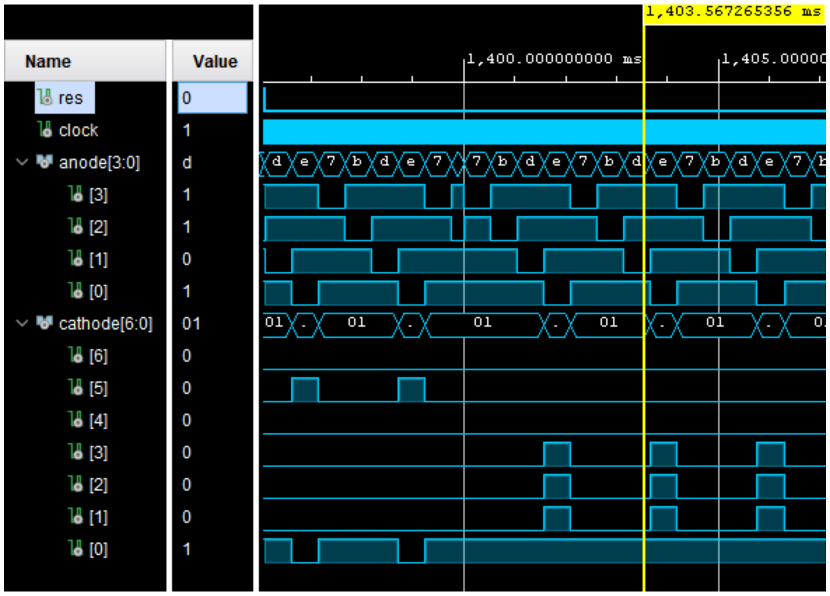


FIG. 5: Zoomed Test Bench Results

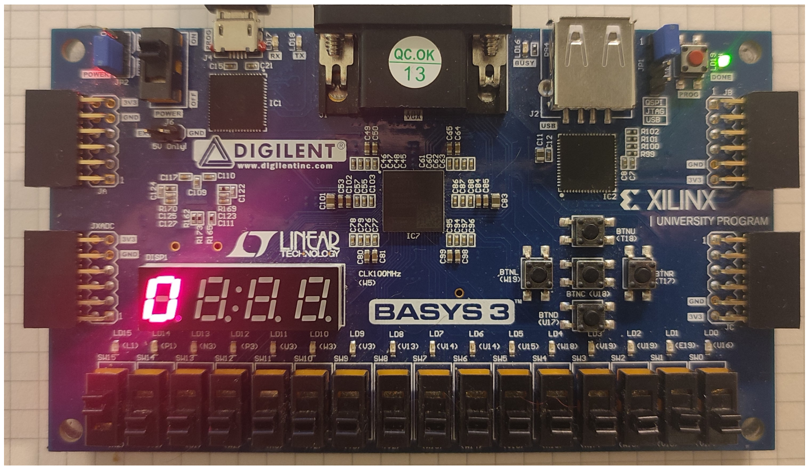


FIG. 6: Reset : On

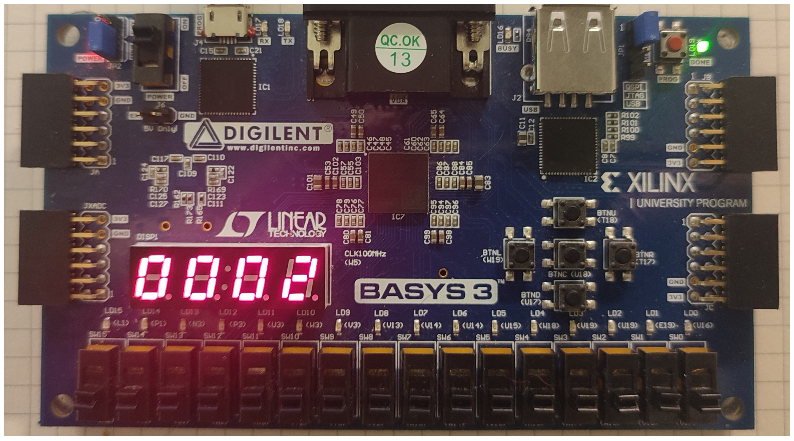


FIG. 7: Reset : Off
Time: 0x2 seconds

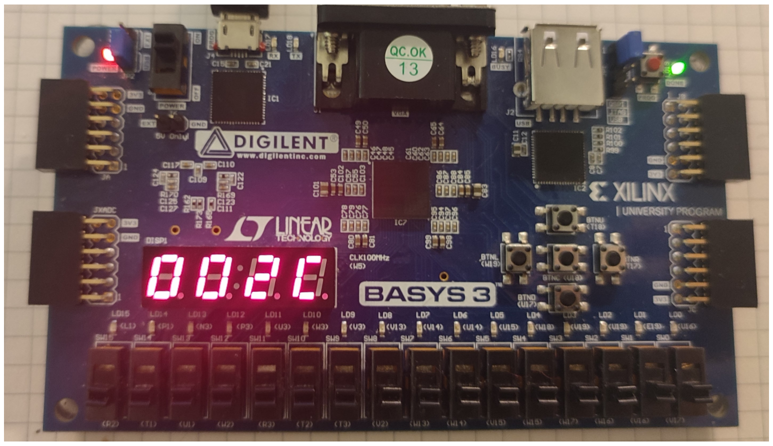


FIG. 8: Reset : Off
Time: 0x2C (44) seconds

VI. APPENDIX

A. Code

1. *Top Level (main.vhd)*

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

entity main is
    Port (
        clock: in std_logic;
        res: in std_logic;
        anode: out std_logic_vector(3 downto 0);
        cathode: out std_logic_vector(6 downto 0)
    );
end main;

architecture rtl of main is
    signal digit: std_logic_vector(1 downto 0);
    signal second_is: std_logic;
    signal num_hex: std_logic_vector(15 downto 0);

begin

    int_clock: entity work.clk_seg(rtl_clock)
        port map(
            clock => clock,
            res => res,
            digit => digit,
            sec_is => second_is,
            sec => num_hex);

    drv: entity work.segment7_driver(rtl_driver)
        port map(
            clock => clock,
            res => res,
            digit => digit,
            num_hex => num_hex,
            sec_is => second_is,
            anode => anode,
            cathode => cathode);
end rtl;
```

2. *clk_seg.vhd*

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

entity clk_seg is
    Port (
        clock: in std_logic;
        res: in std_logic;
        clocount: out std_logic_vector(27 downto 0);
        digit: out std_logic_vector(1 downto 0);
        sec: out std_logic_vector(15 downto 0);
        sec_is: out std_logic);
end clk_seg;

architecture rtl_clock of clk_seg is
    signal clocount1: std_logic_vector(27 downto 0);
    signal second: std_logic_vector(15 downto 0):= (others => '0');

begin

    process(clock) begin
        if(res = '1') then
            clocount1 <= (others => '0');
            second <= (others => '0');
        else
            if rising_edge(clock) then
                clocount1 <= clocount1 + '1';
                if clocount1 =x"5F5E0FF" then
                    second <= second + '1';
                    clocount1 <= (others => '0');
                end if;
            end if;
        end if;
    end process;

    sec_is <= '1' when clocount1 =x"5F5E0FF" else '0';
    digit <= clocount1(19 downto 18);
    clocount <= clocount1;
    sec <= second;

end rtl_clock;
```

3. led_reg.vhd

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

entity clk_seg is
    Port (
        clock: in std_logic;
        res: in std_logic;
        clocount: out std_logic_vector(27 downto 0);
        digit: out std_logic_vector(1 downto 0);
        sec: out std_logic_vector(15 downto 0);
        sec_is: out std_logic);
end clk_seg;

architecture rtl_clock of clk_seg is
    signal clocount1: std_logic_vector(27 downto 0);
    signal second: std_logic_vector(15 downto 0):= (others => '0');
```

```
begin

    process(clock) begin
        if(res = '1') then
            clocount1 <= (others => '0');
            second <= (others => '0');
        else
            if rising_edge(clock) then
                clocount1 <= clocount1 + '1';
                if clocount1 =x"5F5E0FF" then
                    second <= second + '1';
                    clocount1 <= (others => '0');
                end if;
            end if;
        end if;
    end process;

    sec_is <= '1' when clocount1 =x"5F5E0FF" else '0';
    digit <= clocount1(19 downto 18);
    clocount <= clocount1;
    sec <= second;

end rtl_clock;
```

4. dri7er.vhd

```
--se7en segment driver
--WHATS IN THE BOX????
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD;

entity clk_seg is
    Port (
        clock: in std_logic;
        res: in std_logic;
        clocount: out std_logic_vector(27 downto 0);
        digit: out std_logic_vector(1 downto 0);
        sec: out std_logic_vector(15 downto 0);
        sec_is: out std_logic
        );
end clk_seg;

architecture rtl_clock of clk_seg is
    signal clocount1: std_logic_vector(27 downto 0);
    signal second: std_logic_vector(15 downto 0):= (others => '0');

begin

    process(clock) begin
        if(res = '1') then
            clocount1 <= (others => '0');
            second <= (others => '0');
        else
            if rising_edge(clock) then
                clocount1 <= clocount1 + '1';
                if clocount1 =x"5F5E0FF" then
                    second <= second + '1';
                    clocount1 <= (others => '0');
                end if;
            end if;
        end if;
    end process;

    sec_is <= '1' when clocount1 =x"5F5E0FF" else '0';
    digit <= clocount1(19 downto 18);
    clocount <= clocount1;
    sec <= second;

end rtl_clock;
```

5. tb_ top.vhd

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


entity tb_top is
end tb_top;

architecture top_test of tb_top is
    signal res, clock : std_logic;
    signal anode : std_logic_vector(3 downto 0);
    signal cathode : std_logic_vector(6 downto 0);

begin

    DUT: entity work.main(rtl)   --Device Under Test
        port map(
            clock => clock,
            res => res,
            anode => anode,
            cathode => cathode
        );
    clock_process : process
    begin
        clock <= '0';
        wait for 1 ns;
        clock <= '1';
        wait for 1 ns;
    end process;

    stimulus : process
    begin
        res <= '1';
        wait for 2 ns;
        res <= '0';

        wait;
    end process;
end top_test;
```