EEE102 Lab 5 Report

Nizam Ercan 22302317 EEE102-2

1- Purpose:

The aim of this lab is to understand how to use the 7-segment-display of Basys3 by designing a stopwatch that uses multiple digits of the 7-segment-display.

2- Methodology:

Each segment of the 7-segment-display is connected to a common anode which means that Basys3 has 4 common anodes since it only has 4 digits. However, each individual LEDs in a digit has their own cathodes which helps us to control which LED to turn on in a digit as you can see in Figure 1.

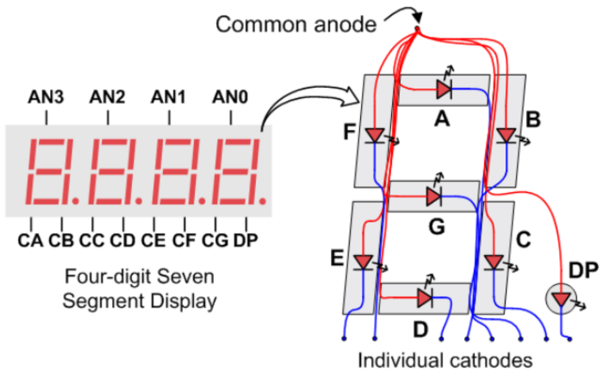


Figure 1: LED configuration of Basys3

To activate an individual segment in a digit, the common anode for that digit should be high and the cathode for the segment should be low. However, as you can see in Figure 2, Basys3 uses transistors to obtain enough current to turn on the LEDs which means that the common anode inputs are reversed and a low signal is needed for it to be active.

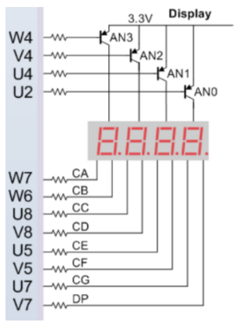


Figure 2: Anode and Cathode pin locations of Basys3

Unfortunately, we can’t turn on all 4 digits of the 7-segment-display at the same time. However, we can turn on and off a single digit for a small time sequentially so that the phenomenon called “persistence of vision" occurs which makes us see that they are turned on at the same time. In Figure 3, you can see a time diagram for persistence of vision to occur.

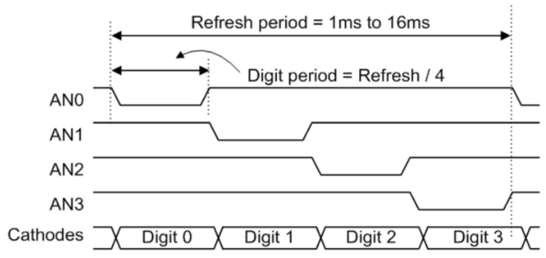


Figure 3: Timin diagram of the 7-segment-display

I am going to use the internal clock of Basys 3 which has a frequency of 100Mhz. I will use this clock to generate an anode refresh counter which needs a refresh period of 1ms to 16ms according to Figure 3. I will also create a second counter that I will use in counting the seconds. Then, I will use these counters to determine which common anodes and segments connected to that common anode to turn on. Finally, I will write the constraint file and the testbench to simulate my design.

Here are the answers for the questions asked in the lab manual:

1- What is the internal clock frequency of Basys 3?

The internal clock of Basys3 has a frequency of 100MHz and it is connected to W5 port

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

By writing a separate counter that increases with the clock and resetting this counter at a specific number k. When the resetting occurs, we should increase another signal. This signal acts as a clock with frequency f/k where f is the frequency of the initial clock.

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

No, we can't. In the previous question, I showed that we can make the frequency of the clock f/k where k is a natural number and f is the frequency of our clock. Since k is a natural number, we can't obtain certain values for the frequency.

3- Design Specifications:

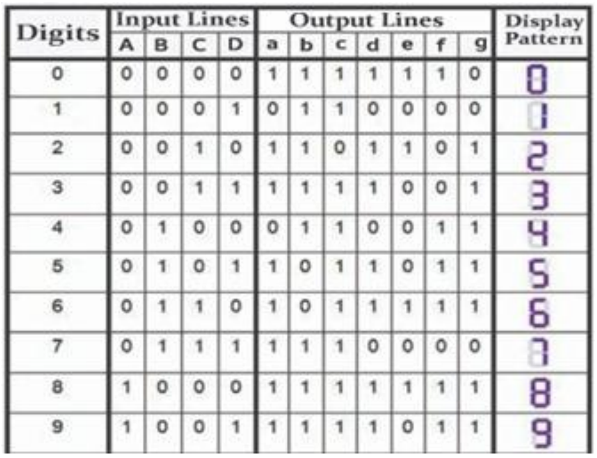
There will be 1 input and 2 outputs for the design which are the clock input "clk", the output that shows which common anodes to turn on "anode_active" which is a 4-bit std_logic_vector type and the output that shows which segment to turn on "display" which is a 7-bit std_logic_vector type. The design has 4 essential parts which are the "anode_mode" counter, the "second" counter, 4 decoders that assign specific values for the digits of the minutes and the seconds to the appropriate display signal and one final multiplexer that assigns the correct anode_active and display signal according to the anode_mode. Below you can see the internal signals and their purposes':

- anode_counter(integer): counter for the "anode_mode"
- second_counter(integer): counter for "second"
- anode_mode(integer): this signal shows which anode should be turned on(0 means forth digit, 1 means third digit, 2 means second digit and 3 means the first digit)
- second(integer): shows the number of seconds we want to display.
- minute(integer): shows the number of minutes we want to display.
- hour(integer): shows the number of hours we want to display(This signal isn't related to this lab session; I am planning to use this code as a part of my term project so this signal is for that.)
- display1(std_logic_vector(6 downto 0)) : shows the cathode configuration of the first digit.
- display2(std_logic_vector(6 downto 0)) : shows the cathode configuration of the second digit.
- display3(std_logic_vector(6 downto 0)) : shows the cathode configuration of the third digit.
- display4(std_logic_vector(6 downto 0)) : shows the cathode configuration of the fourth digit.
- seconds_first_digit(integer): first digit of the "second" signal.
- seconds_second_digit(integer): second digit of the "second" signal.
- minute_first_digit(integer): first digit of the "minute" signal.
- minute_second_digit(integer): second digit of the "minute" signal.

My code increases the "second_counter" with the clock of 100Mhz. I want "second" signal to increase with a frequency of 1Hz. Therefore, I made the "second" signal increase everytime the "second_counter" gets the value 100000000. After this value "second_counter" becomes 0 which repeats a procces where the "second" signal increases every second. By making a counter that can count real seconds, I easily created a counter for minutes and hours too.

I also created a counter for the "anode_mode". The "anode_counter" increases with the clock of 100Mhz. I want the "anode_mode" to change every 1ms so that all of the digits will be turned and off at a total of 4 ms which is enough for persistence of vision to occur. I am making the "anode_mode" to increase when "anode_counter" becomes 100000 which makes 1ms. After that the "anode_counter" becomes 0 which makes the whole process repeat. Also "anode_mode" becomes 0 after 3.

After assigning the proper values for the signals "second" and "minutes", I divided their digits to "seconds_first_digit", "seconds_second_digit", "minute_first_digit" and "minute_second_digit". Then, I wrote 2 processes for "second" and "minute" and wrote 2 case-when statements to assign each digit in the processes to different display signals. You can see the truth table for assigning integers to 7-bit binary numbers in Figure 4.



Digits	Input Lines				Output Lines							Display Pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	1	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7
8	1	0	0	0	1	1	1	1	1	1	1	8
9	1	0	0	1	1	1	1	1	0	1	1	9

Figure 4: Truth table for transferring digits to display signals

Finally, I wrote a process for "anode_mode" where I wrote a case-when statement for assigning the "anode_active " and "display" outputs according to the "anode_mode ".

4- Result:

Before implementing my design to the FPGA, I wrote a testbench as you can see in "sim.vhd" to see any potential error in my code. In Figure 4, you can see a general outline of how my design works in the first 3 seconds. You can see that after every second, the "display" signal changes from "0000001" to "1001111" and from "1001111" to "0010010" which shows that the digit increases from 0 to 1 and from 1 to 2. (The timing of the simulations is incorrect as I increased the speed of my counters in the main code so that the simulation runs faster. In the final version of my codes in the appendix, I wrote the true version with correct timing.)

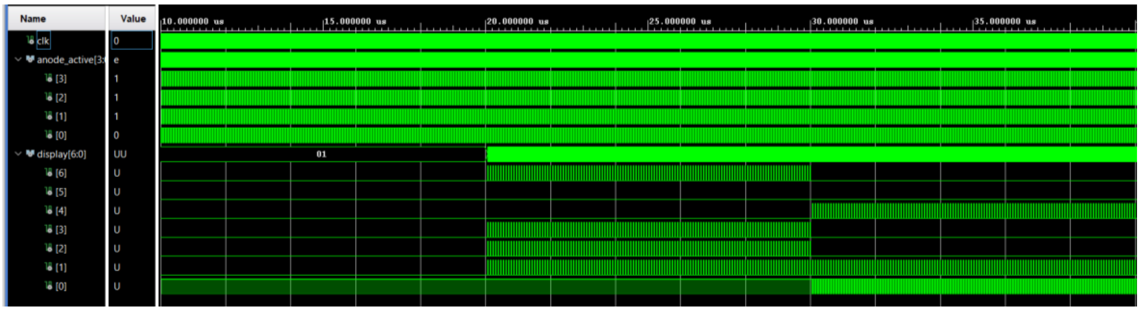


Figure 5: My simulation in the first 3 seconds

In Figure 5, you can see a zoomed part of my simulation during the third second. You can see that the active anodes change to '0' respectively which shows that the persistence of vision is achieved. Also, when "anode_active" is "1110", which is fourth digit, "display" becomes "0010010" which corresponds to the integer 2 according to the decoder in my main module "clock.vhd". In other 3 values of "anode_active", "display" is "0000001" which corresponds to the integer 0. This means that my code is working properly since I want the first three digits to be 0 and the last digit to be 2 in order to show the third second (00 02).

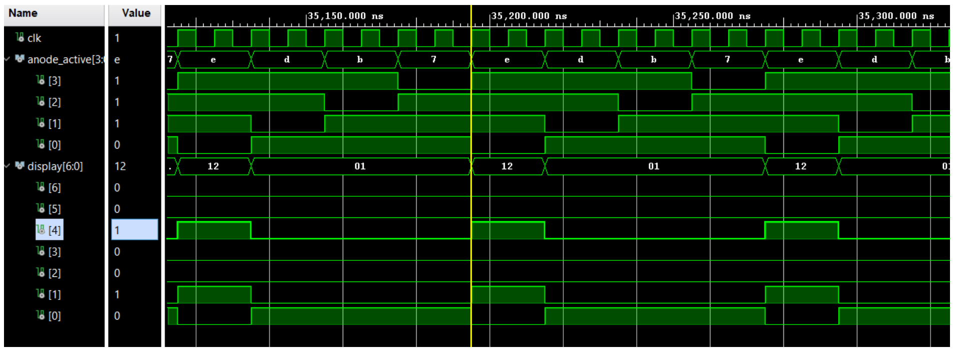


Figure 6: zoomed third second

The RTL design for “clock.vhd” can be seen in Figure 6. I didn’t use a modular design so the decoders and the multiplexers used in the design can’t be seen properly. However, by looking at “clock.vhd” you can see the case statements used for the “display” signals and “anode_active” signals which indicates the usage of decoders and multiplexers.

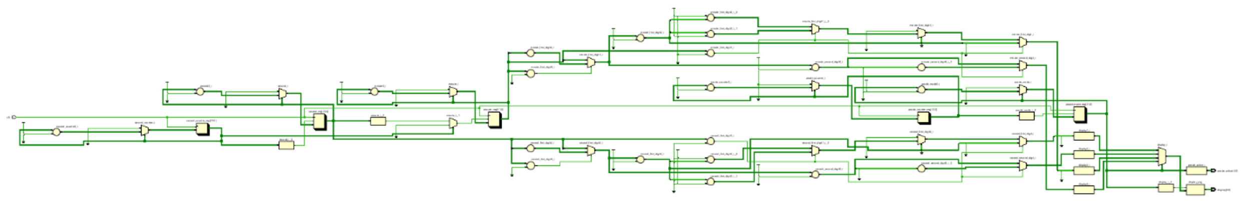


Figure 7: RTL schematics

Finally, I am implementing my code on my Basys3. After synthesizing and implementing my design, I generated the bitstream and connected my Basys3. In Figure 7 and 8, you can see two examples from my working Basys3.

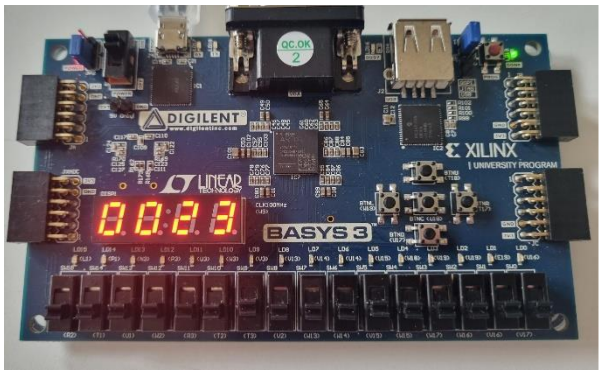


Figure 7: Example from 23th second

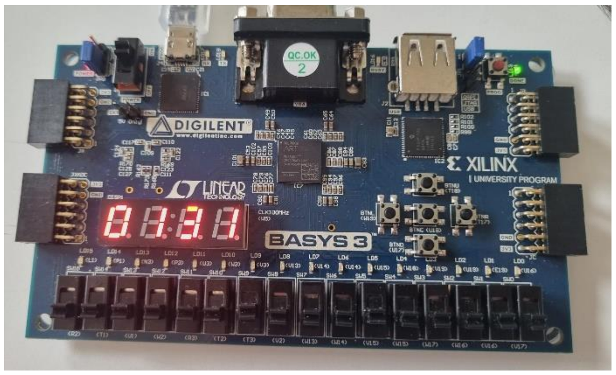


Figure 8: Example from the 91th second

5- Conclusion:

The purpose of this lab was to design a circuit that can use multiple digits of the 7-segment-display of the Basys3. I designed a stopwatch that can count seconds and minutes to achieve that. I designed two counters that have different frequencies to count seconds and achieve persistence of vision. In the end, I was able to achieve all of the lab's goals.

6- References

https://digilent.com/reference/basys3/refmanual?_ga=2.185577961.1516997771.1708019 872-1867094360.1708019872

7- Appendices

clock.vhd:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity clock is

    Port ( clk : in STD_LOGIC;

           anode_active : out std_logic_vector(3 downto 0);

           display : out std_logic_vector(6 downto 0)

           );

end clock;


architecture Behavioral of clock is


    signal anode_counter : integer := 0;

    signal second_counter : integer := 0;


    signal anode_mode : integer := 0;


    signal second : integer := 0;

    signal minute : integer := 00;

    signal hour : integer := 12;


    signal display1 : std_logic_vector(6 downto 0);

    signal display2 : std_logic_vector(6 downto 0);

    signal display3 : std_logic_vector(6 downto 0);
    signal display4 : std_logic_vector(6 downto 0);

    signal second_first_digit : integer := 0;
    signal second_second_digit : integer := 0;
    signal minute_first_digit : integer := 0;
    signal minute_second_digit : integer := 0;
begin

process(anode_mode)
begin
case anode_mode is
    when 0 => anode_active <= "1110";
              display <= display1;
    when 1 => anode_active <= "1101";
              display <= display2;
    when 2 => anode_active <= "1011";
              display <= display3;
    when 3 => anode_active <= "0111";
               display <= display4;
    when others => anode_active <= "1111";
end case;
end process;


process(clk)
begin

    if rising_edge(clk) then

        --anodes 100000
        if anode_counter = 100000 then
            anode_counter <= 0;
            if anode_mode = 3 then
                anode_mode <= 0;
            else
                anode_mode <= anode_mode + 1;
            end if;
        else
            anode_counter <= anode_counter + 1;
        end if;



        --seconds 100000000
        if second_counter = 100000000 then
            second_counter <= 0;
            if second = 59 then
                second <= 0;
                if minute = 59 then
                    minute <= 0;
                    if hour = 23 then
                        hour <= 0;
                    else
                        hour <= hour + 1;
                    end if;
                else
                    minute <= minute + 1;
                end if;

            else
                second <= second + 1;
            end if;
        else
            second_counter <= second_counter + 1;
        end if;

    end if;

end process;
```

```
second_first_digit <= second mod 10;
second_second_digit <= second / 10;

process(second)
begin
case second_first_digit is

    when 0 => display1 <= "0000001";
    when 1 => display1 <= "1001111";
    when 2 => display1 <= "0010010";
    when 3 => display1 <= "0000110";
    when 4 => display1 <= "1001100";
    when 5 => display1 <= "0100100";
    when 6 => display1 <= "0100000";
    when 7 => display1 <= "0001111";
    when 8 => display1 <= "0000000";
    when 9 => display1 <= "0000100";
    when others => display1 <= "0011000";

end case;

case second_second_digit is
    when 0 => display2 <= "0000001";
    when 1 => display2 <= "1001111";
    when 2 => display2 <= "0010010";
    when 3 => display2 <= "0000110";
    when 4 => display2 <= "1001100";
    when 5 => display2 <= "0100100";
    when 6 => display2 <= "0100000";
    when 7 => display2 <= "0001111";
    when 8 => display2 <= "0000000";
    when 9 => display2 <= "0000100";
    when others => display2 <= "0011000";
```

```
end case;

end process;


minute_first_digit <= minute mod 10;

minute_second_digit <= minute / 10;


process(minute)

begin


    case minute_first_digit is


        when 0 => display3 <= "0000001";

        when 1 => display3 <= "1001111";

        when 2 => display3 <= "0010010";

        when 3 => display3 <= "0000110";

        when 4 => display3 <= "1001100";

        when 5 => display3 <= "0100100";

        when 6 => display3 <= "0100000";

        when 7 => display3 <= "0001111";

        when 8 => display3 <= "0000000";

        when 9 => display3 <= "0000100";

        when others => display3 <= "0011000";


    end case;


    case minute_second_digit is

        when 0 => display4 <= "0000001";

        when 1 => display4 <= "1001111";

        when 2 => display4 <= "0010010";

        when 3 => display4 <= "0000110";

        when 4 => display4 <= "1001100";

        when 5 => display4 <= "0100100";

        when 6 => display4 <= "0100000";

        when 7 => display4 <= "0001111";
        when 8 => display4 <= "0000000";
        when 9 => display4 <= "0000100";
        when others => display4 <= "0011000";

    end case;
end process;

end Behavioral;
```

sim.vhd:

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

entity sim is
end sim;

architecture Behavioral of sim is

    signal clk: std_logic;
    signal anode_active: std_logic_vector(3 downto 0);
    signal display: std_logic_vector(6 downto 0);

    component clock is
    Port ( clk : in STD_LOGIC;
           anode_active : out std_logic_vector(3 downto 0);
           display : out std_logic_vector(6 downto 0)
           );
    end component;

begin

dut: clock port map(clk, anode_active, display);

tb: process
```

```
begin

    clk <= '0';
    wait for 5 ns;
    clk <= '1';
    wait for 5 ns;

end process;

end Behavioral;
```