Satılmış Furkan Kahraman 22002075 EE 102-01 21.11.2022

Lab 5 : Seven Segment Display

Objective

The purpose of this lab was to use the clock feature of the BASYS 3 card to prepare a 7-segment display whose repetition rate is much faster than the human eye can perceive. So it seems like a single number is printed on the screen at any given moment. Also in this lab I designed a timer that works depending on the clock. I calculated this as one second at a certain frequency of the clock and changed the value displayed on the screen.

Methodology

Before I started writing the VHDL code, I started to gather information about how the system works. To understand the working logic of seven segment display, I looked at how the bcd to binary decoder works. In order for the anode and the matching cathode to light up at the same time, it is sufficient for the specific anode to receive the value 0 in the decoder, while the bits corresponding to the required number value for the cathode must be zero. This decoder, which determines the light emitting algorithm of the cathode, is shown in figure 1.

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

Figure 1 – BCD to 7 Segment Display

In the schematic I found in BASYS 3 manual booklet, I noticed that the anode and cathode outputs are separate and there is 1 anode and 8 cathodes for each 7 segment display (Figure 1). Anodes and cathodes are working as active low. Active low means that the led works when there is no signal to that bit. Since only one of the anode signals will be on at a time, I used the repeat rate here. Since there are 4 anodes, I divided the existing iteration rate into 4 and made the anodes and the cathodes connected to those anodes light up from the rightmost bit to the leftmost bit in order in each iteration. Since BASYS has its own clock frequency of 100 MHz, I had to divide it before using it. So I wrote a counter that counts each clock beat one by one and resets itself when it reaches 100 million. I had this counter increment the second variable by one during the self reset phase.

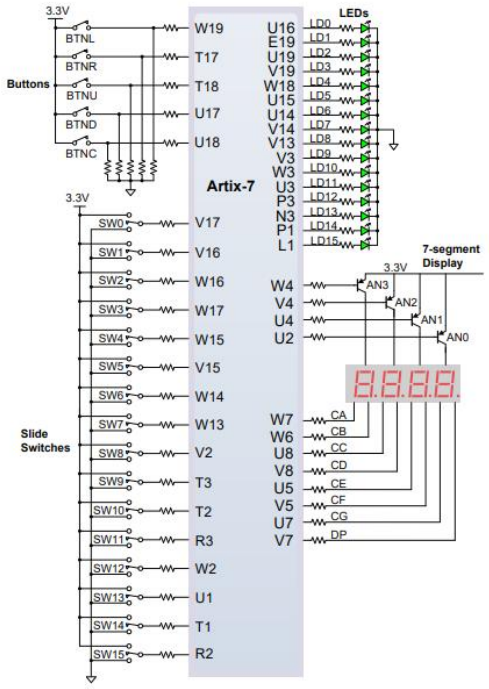

Figure 2 – 7 Segment Display Schematic

Since 100 Mhz is very fast, I created a counter using bits 19 and 18 of the counter that counts the clock pulse.Since the rate of change of these bits is approximately equal to 1.3 ms, it equals a refresh rate of approximately 760 Hz in 1 second. Also, since I have to divide this 760 hertz by 4, each anode actually blinks 190 times per second.

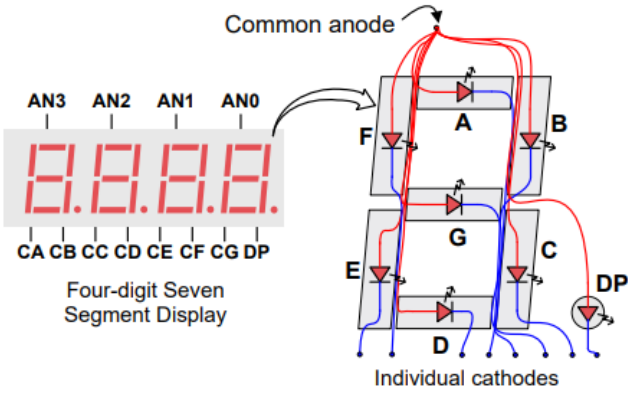

Figure 3 – Anode and cathode schematic

Finally, I assigned all these select and values to the multiplexers and made the cathodes and anodes blink simultaneously.I also added a reset button to reset the counter. this way there is no need to wait for the circuit to complete to get all the digits of the counter "0000".

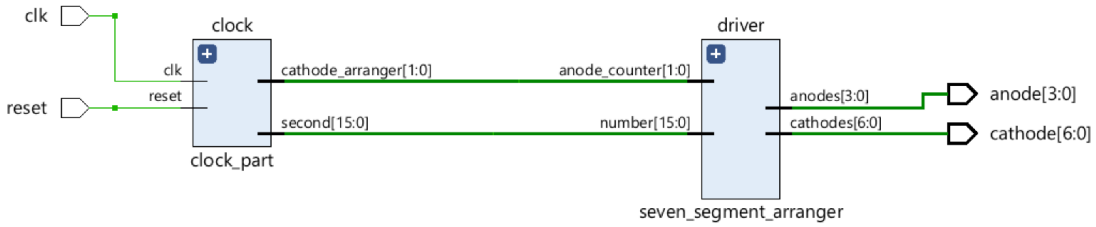

Figure 4 – Elaborated Design

Result

As a result, despite the 10 MHz clock signal in BASYS, it can be observed that the screen repetition frequency is 760 hertz in the simulation results. On the screen, it is understood that the cathode and the anode are working simultaneously. Since the numbers are in hexadecimal number system, each 7 segment display is respectively "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" reflects the values "a", "b", "c", "d", "E", "F".

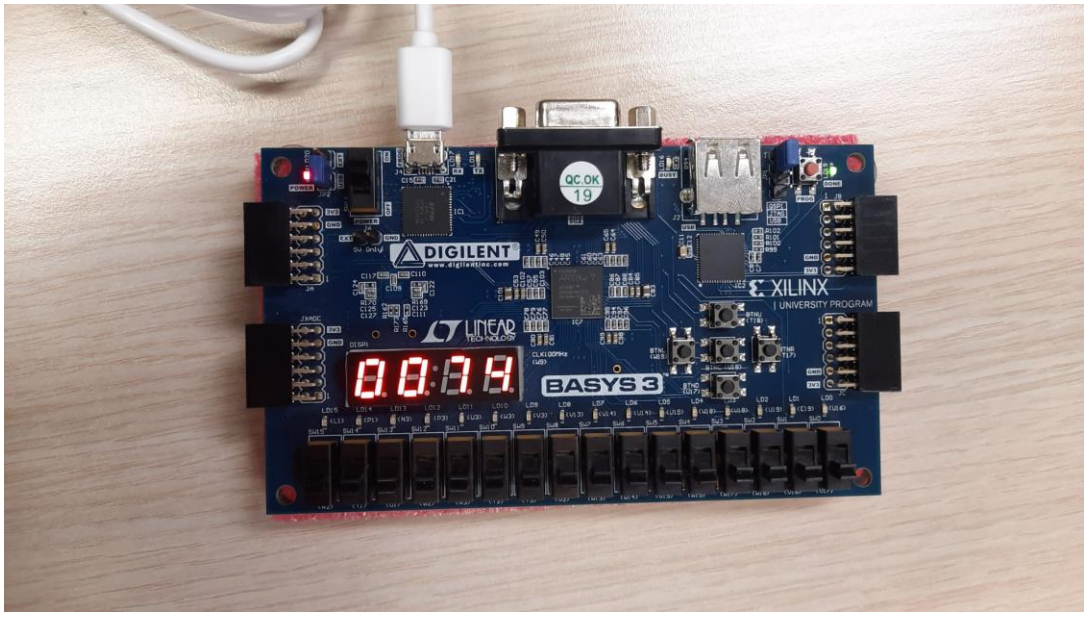

Figure 5 – "0074" is shown on the screen which means 116. seconds

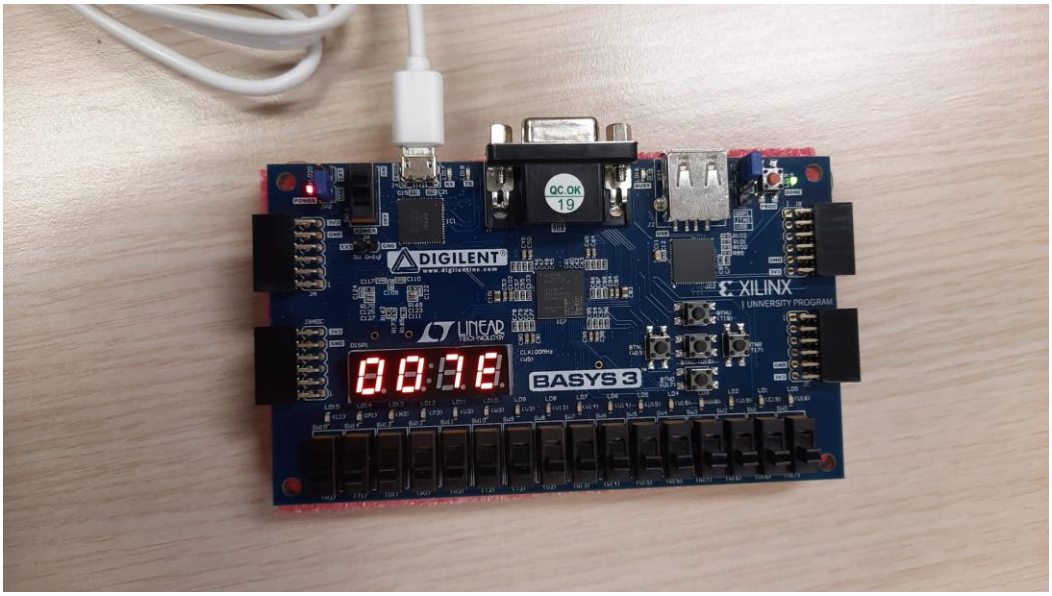

Figure 6 – "007E" is shown on the screen which means 126. seconds

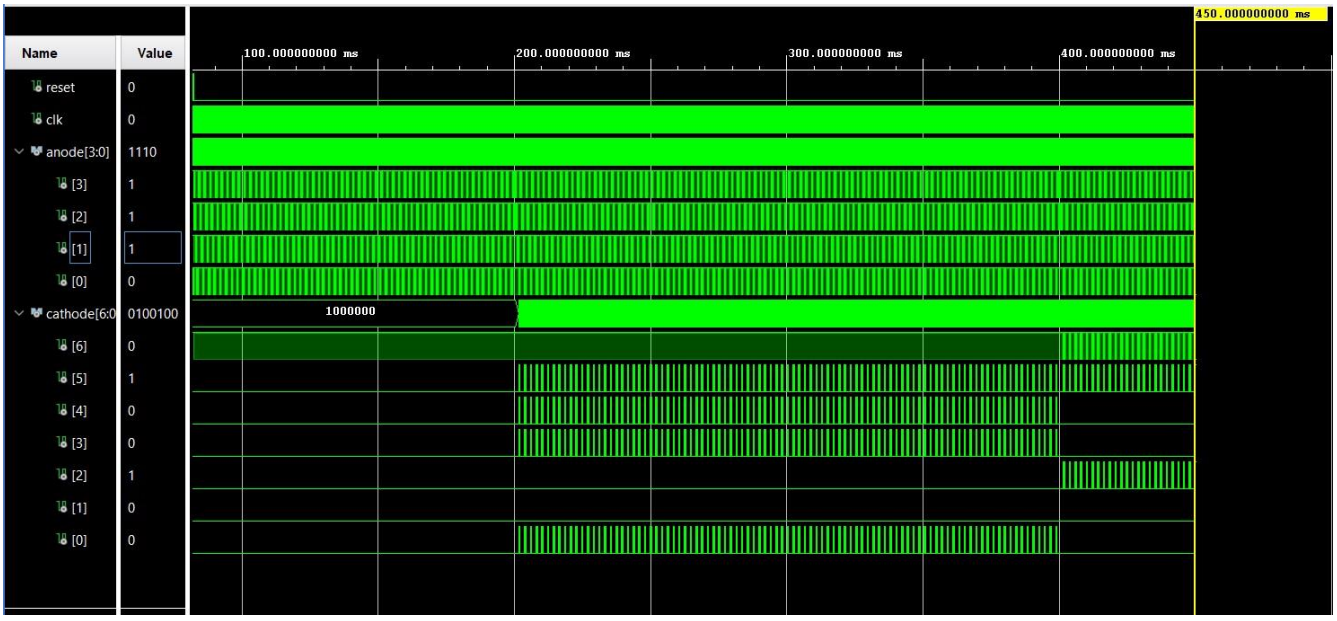

Figure 7 – Simulation Results for 450 ms

Conclusion

The purpose of this lab was to understand the use of the clock signal. Besides, it was to understand how the 7-segment display is used on the FPGA and how a function can be created with it. Unlike other labs, we performed sequential operations in this lab. In the circuit we designed, the previous output affected the next input.

Appendices

Top Module Code

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

entity top_module is
    Port(clk : in std_logic;
         reset_counter : in std_logic;
         anode_select : out std_logic_vector(3 downto 0);
         cathode_select : out std_logic_vector(6 downto 0));

-- entity the inputs and outputs of circuit --

end top_module;

architecture top_module_arch of top_module is
        signal phase_count: std_logic_vector(1 downto 0);
        signal sec_en: std_logic;
        signal number: std_logic_vector(15 downto 0);

component clock_part
            Port( clk : in std_logic;
            reset : in std_logic;
            cathode_arranger : out std_logic_vector(1 downto 0);
             second : out std_logic_vector(15 downto 0);
             count1enable : out std_logic);
end component;

component seven_segment_arranger
            Port(number : in std_logic_vector(15 downto 0);
             anode_counter : in std_logic_vector(1 downto 0);
             clk : in std_logic;
             reset : in std_logic;
             anodes: out std_logic_vector(3 downto 0);
             cathodes : out std_logic_vector(6 downto 0);
             sec_enable: in std_logic);

begin

driver1 : seven_segment_arranger port map(number => number, anode_counter =>anode_select , cathodes => cathode_select , sec_enable => sec_enable, reset => reset_counter  )
```

```
clock1 : clock_part port map(clk => clock, reset => reset_counter , cathode_arranger => phase_count,
second => number)

end top_module_arch;
```

Clock Divider Code

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

entity clock_part is
    Port( clk : in std_logic;
          reset : in std_logic;
          cathode_arranger : out std_logic_vector(1 downto 0);
          second : out std_logic_vector(15 downto 0);
          count_enable : out std_logic);

end clock_part;

architecture Behavioral of clock_part is

signal clock_counter: std_logic_vector(26 downto 0);
signal second_counter: std_logic_vector(15 downto 0):= (others => '0');
-- second_counter counts according to frequency of BASYS clock--

begin
process(clk) begin
if(reset = '1') then
    clock_counter <= (others => '0');
    second_counter <= (others => '0');
else
    if rising_edge(clk) then
        clock_counter <= clock_counter + '1';
    if clock_counter =x"5F5E0FF" then;
-- this is hexadecimal equivalent of "99999999"--
        second_counter <= second_counter + '1';
-- this part makes 0 clock counter when it gets reached "99999999"--
        clock_counter <= (others => '0');
    end if;
    end if;
end if;
end process;

count_enable <= '1'
           when clock_counter =x"5F5E0FF" else '0';
cathode_arranger <= clock_counter(19 downto 18);
-- this part arrange clock divider --
```

```
second <= second_counter;


end Behavioral;
```

Seven Segment Arrenger Code

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

entity seven_segment_arranger is
    Port(number : in std_logic_vector(15 downto 0);
         anode_counter : in std_logic_vector(1 downto 0);
         clk : in std_logic;
         reset : in std_logic;
         anodes: out std_logic_vector(3 downto 0);
         cathodes : out std_logic_vector(6 downto 0);
         sec_enable: in std_logic);

end seven_segment_arranger;

architecture Behavioral of seven_segment_arranger is

signal led_select: std_logic_vector(3 downto 0);
signal num : std_logic_vector(15 downto 0);
signal second_enable : std_logic_vector(1 downto 0);

begin
process(anode_counter)
begin
case anode_counter is

-- anode decoder's code is here ( it only choose one of anode from 4 anode ) --

    when "00" => anodes <= "0111";
    led_select <= number(15 downto 12);
    when "01" => anodes <= "1011";
    led_select <= number(11 downto 8);
    when "10" => anodes <= "1101";
    led_select <= number(7 downto 4);
    when "11" => anodes <= "1110";
    led_select <= number(3 downto 0);
    when others => anodes <= "1111";
end case;
end process;

component seven_segment_decoder
          Port( bit4decoder : in std_logic_vector(3 downto 0);
```

```
            bit7decoder : out std_logic_vector(6 downto 0));

end Behavioral;
```

Seven Segment Decoder Code

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

entity seven_segment_decoder is
    Port( bit4decoder : in std_logic_vector(3 downto 0);
          bit7decoder : out std_logic_vector(6 downto 0));

end seven_segment_decoder;

architecture Behavioral of seven_segment_decoder is

begin

with bit4 select

-- bcd to 7-segment display decoders code is here --

bit7 <= "1000000" when "0000",
        "1111001" when "0001",
        "0100100" when "0010",
        "0110000" when "0011",
        "0011001" when "0100",
        "0010010" when "0101",
        "0000010" when "0110",
        "1111000" when "0111",
        "0000000" when "1000",
        "0010000" when "1001",
        "0100000" when "1010",
        "0000011" when "1011",
        "1000110" when "1100",
        "0100001" when "1101",
        "0000110" when "1110",
        "0001110" when others;

end Behavioral;
```

Test Bench Code

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

entity tb is
end tb;

architecture rtl of tb is
signal reset
signal clock_signal: std_logic;
signal anode: std_logic_vector(3 downto 0);
signal cathode: std_logic_vector(6 downto 0);

component top_module
Port(clk : in std_logic;
        reset : in std_logic;
        anode  : out std_logic_vector(3 downto 0);
        cathode : out std_logic_vector(6 downto 0));
end component;

begin

 UUT: top_module
 port map (clk => clock_signal,
                reset=>reset,
               anode => anode,
               cathode=> cathode);

 clock_process :process

begin

 clock_signal <= '0';
 wait for 1 ns;
 clock_signal <= '1';
 wait for 1 ns;
 end process;
 stim_proc: process

begin
 reset <= '1';
 wait for 1 ns;
 reset <= '0';
```

```
 wait;
 end process;

end rtl;
```