Name: Emre Şaş

Section:2

ID:22102764

Bilkent University Electrical and Electronics Department

EEE-102-02 Lab 5 Report

Purpose:

Purpose of this lab is to understand how seven-segment display screen works and how to use clock properties to create simulation of time.

Questions:

- **What is the internal clock frequency of BASYS3?**

The Basys 3 board comes with a 100 MHz oscillator on board. However this can be adjusted in constraints depending on the need of our project. In this lab default value is used.

- **How can you create a slower clock signal from this one?**
    - To create a slower clock signal from a faster one, use a clock divider that divides the original clock frequency by a specified ratio. Ensure that the slower clock signal meets your design's timing requirements. By using clock dividers and counter algorthms we can create stopwatch.

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

    - In BASYS3, you cannot create a clock with an arbitrary frequency lower than the internal clock. The frequencies you can generate are typically limited to specific values that are integer divisors, fractions, or multiples of the internal clock frequency. These specific frequencies are determined by the FPGA's clock resources and clock management capabilities. For example default frequency of BASYS3 is 100MHz ,meaning, we can create clocks at 50 MHZ, 10MHz ,1MHz etc.

Methodology:

In order to use the seven-segment display properly one must understand the concept of persistence of vision. Persistance of vision is actually deceiving our eyes by creating the illusion of LED's on display screean are continuously lighten up. BASYS3 does not have ability to light all of the LED segments at once so we use the trick depending on the data given by data sheet of our FPGA (Figure 1).

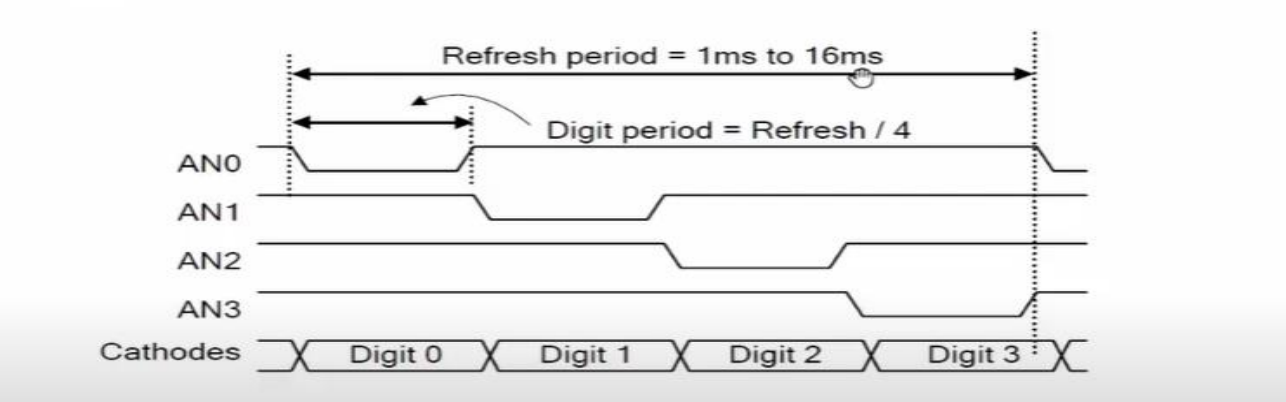

Figure 1

Therefore, we need to drive the power so fast that our eyes cannot obtain that these LED's are opening and closing. After completing the anodes we need to take care of cathodes which will decide which LED strips will be lightned which not. To be able to get right result Figure 2 should be studied.

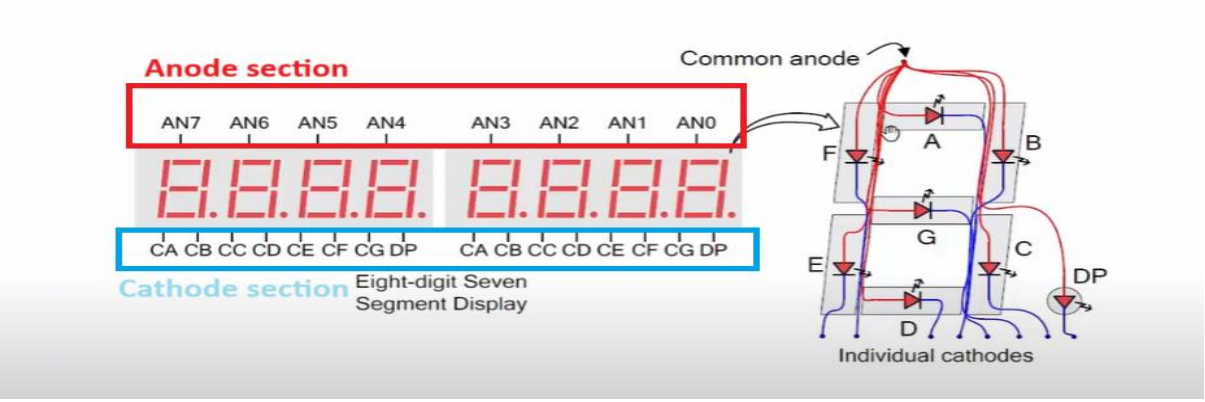

Figure 2

Finally, we should create clock divider and counter to create and count cycles to get centiseconds and seconds. We need a clock divider since the BASYS has an embedded clock of 100MHz Which value is too big to count seconds.

Design specifications:

The Stopwatch entity is designed for controlling a stopwatch display with four digits and a 7-segment display. The stopwatch can be controlled using the start_signal and reset_signal inputs. When start_signal is asserted ('1'), the stopwatch starts, and when deasserted ('0'), it stops. The reset_signal input is used to reset the stopwatch to zero.

The entity operates based on an internal clock signal, Imh, and generates two derived clock signals, centi (for centiseconds) and CLK_240Hz (for anode rotation).

The stopwatch keeps track of time in centiseconds and also rotates through the four display digits at a rate of 240Hz. When started, the stopwatch increments the time display, and the multiplexed 7-segment display cycles through the four digits, showing the elapsed time. When stopped, the stopwatch retains the time displayed.

Processes:

Control Process: This process handles the control input debouncing and toggles the ACTIVATE signal when the start/stop button is pressed. The stopwatch can be stopped by setting stop_signal to '1'.

Clock Generation Process: This process generates two clock signals, centi (for centiseconds) and CLK_240Hz (for anode rotation). It uses an internal count to divide the input clock into these two frequencies.

Cathode Logic Process: This process handles the logic for the 7-segment display. It divides the count into four digits (count1 to count4) and displays the corresponding 7-segment patterns on ct_1 to ct_4 based on the count values.

Anode Rotation Process: This process controls the selection of the active digit in the multiplexed display based on the CLK_240Hz signal. It selects one of the four cathode outputs (ct_1 to ct_4) to be displayed.

Note: The specifics of the 7-segment patterns for each digit are provided in the code using the **case** statements in the "Cathode Logic Process."

Timing: The centiseconds clock (centi) has a period of 10 ms, and the anode rotation clock (CLK_240Hz) has a period of approximately 4.1667 ms, achieving a 240Hz rotation frequency.

Reset Behavior: When reset_signal is asserted ('1'), the stopwatch is reset to zero. When deasserted ('0'), it resumes counting from zero.

Simulation and Verification: The Stopwatch design can be simulated to verify its functionality. Inputs such as start_signal and reset_signal can be controlled to start, stop, and reset the stopwatch, and the outputs can be monitored to check the accuracy of timekeeping and the display.

Clock Requirements: The design assumes the presence of an external clock input,Imh. The clock frequency should be appropriate for generating centiseconds and anode rotation frequencies.

Multiplexing: The design employs multiplexing to display four digits with a single 7-segment display. The anode control selects the active digit, and the cathode control provides the segment patterns for the selected digit.

Segment Patterns: The 7-segment patterns for each digit are defined in the "Cathode Logic Process."

Debouncing: The design includes debouncing logic for the start/stop and reset control inputs to prevent noise and glitches.

After completing all of these steps we can look for schematic (Figure 3), testbench (Figure 4) and implement the code to our FPGA board.

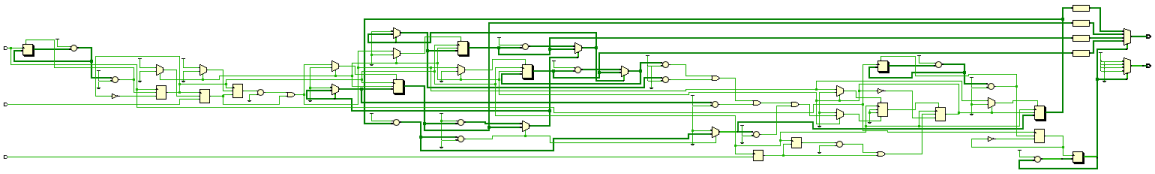

Figure 3

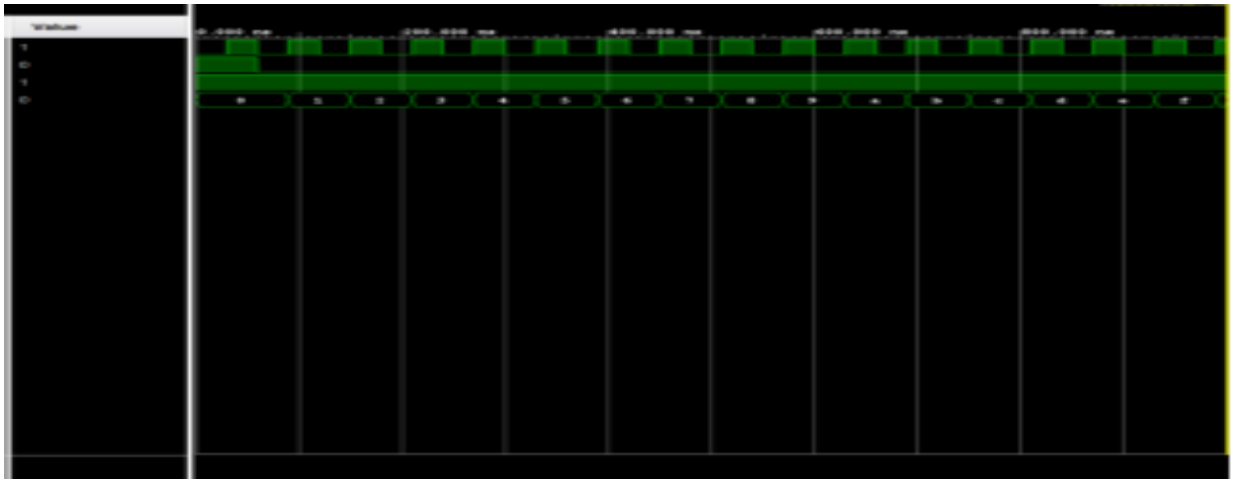

Figure 4

Persistance of vision can be observed by using slowmotion. As it can be seen in video 1 (https://drive.google.com/file/d/17aI2ghnynrAovYCLSj0Is013Yq54HcGh/view?usp=drive_link) segments are turned off and on in an order.

Real time clock can be seen in video 2

(https://drive.google.com/file/d/1lNJMcMMs6ToGs5inqSA76x3uj8bF3lVx/view?usp=drive_link)

Consclusion:

The driver and clock modules, as well as the seven-segment display, were implemented using VHDL. During this lab work, a number of concepts have been reaffirmed, such as how a seven-segment display functions, how to build a clock and driver, and the notion of persistence of vision. There is a small error in my clock since I have used 240 HZ cycles and count them 41.667 times which leads us to 10.000.080 and When the desired number (41.667 cycles) is arrived I approximated the result as 1/10 of a second.

Appendices:

Design code

--

-- Company:

-- Engineer: emre şaş

--

-- Create Date: 29.10.2023 14:43:44

-- Design Name:

-- Module Name: agaaaa - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

--

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

library IEEE;
use IEEE.NUMERIC_STD.All;
use IEEE.STD_LOGIC_1164.ALL;
entity Stopwatch is
port ( start_signal: in STD_LOGIC;
  reset_signal : in STD_LOGIC;
Imh: in STD_LOGIC;
cathode_out : out STD_LOGIC_VECTOR(7 downto 0);
AN : out STD_LOGIC_VECTOR(3 downto 0));
end Stopwatch;
architecture behavioral of Stopwatch is
signal centi : STD_LOGIC := '0';
signal CLK_240Hz : STD_LOGIC := '0';
signal activate : STD_LOGIC := '0';
signal start_1: STD_LOGIC :=
'0';
signal reset_1: STD_LOGIC :=
```

```
'0';
signal PB_sig_1: STD_LOGIC :=
'0';
signal PB_sig_2: STD_LOGIC :=
'0';
signal stop_signal : STD_LOGIC :=
'0';
signal ct_1 : STD_LOGIC_VECTOR(7 downto 0) := "00000010";
signal ct_2 : STD_LOGIC_VECTOR(7 downto 0) := "00000010";
signal ct_3 : STD_LOGIC_VECTOR(7 downto 0) := "00000011";
signal ct_4 : STD_LOGIC_VECTOR(7 downto 0) := "00000010";
begin

process(centi) begin
if(rising_edge(centi)) then
if(start_signal = '1') then -- START/STOP button toggle
start_1 <= '1';
elsif(start_signal = '0') then
start_1 <= '0';
end if;

PB_sig_1 <= start_1;
if(PB_sig_1 = '0' and start_1 = '1') then
activate <= not activate;
end if;

if(stop_signal = '1') then
activate <= '0';
end if;
end if;
```

```
end process;

process(Imh)
variable count_centi : unsigned(18 downto 0) := to_unsigned(0,19);
variable count_240Hz : unsigned(15 downto 0) := to_unsigned(0,16);
begin
if(rising_edge(Imh)) then

count_centi := count_centi + 1;
if(count_centi = to_unsigned(500000,19)) then
centi <= not centi;
count_centi := to_unsigned(0,19);
end if;
count_240Hz := count_240Hz + 1;
if(count_240Hz = to_unsigned(41667,16)) then
CLK_240Hz <= not CLK_240Hz;
count_240Hz := to_unsigned(0,16);
end if;

end if;
end process;

process(Imh, centi, reset_signal, activate)
variable count1 : unsigned := "0000";
variable count2 : unsigned := "0000";
variable count3 : unsigned := "0000";
variable count4 : unsigned := "0000";
begin

if(rising_edge(centi)) then
```

```
if(activate = '0') then
if(reset_signal = '1') then
reset_1 <= '1';
elsif(reset_signal = '0') then
reset_1 <= '0';
end if;

PB_sig_2 <=reset_1;
if(PB_sig_2 = '0' and reset_1 = '1') then
count1 := "0000";
count2 := "0000";
count3 := "0000";
count4 := "0000";
stop_signal <= '0';
end if;


elsif(activate = '1') then
count1 := count1 + 1;
if(count1 = "1010") then
count2 := count2 + 1;
count1 := "0000";
end if;

if(count2 = "1010") then
count3 := count3 + 1;
count2 := "0000";
end if;

if(count3 = "1010") then
```

```
count4 := count4 + 1;
count3 := "0000";
end if;

if(count4 = "0101" and count3 = "1001" and count2 = "1001"
and count1 = "1001") then
stop_signal <= '1';
end if;
end if;
end if;
case count1 is
when "0000" => ct_1 <= "00000011"; --0
when "0001" => ct_1 <= "10011111"; --1
when "0010" => ct_1 <= "00100101"; --2
when "0011" => ct_1 <= "00001101"; --3
when "0100" => ct_1 <= "10011001"; --4
when "0101" => ct_1 <= "01001001"; --5
when "0110" => ct_1 <= "01000001"; --6
when "0111" => ct_1 <= "00011111"; --7
when "1000" => ct_1 <= "00000001"; --8
when "1001" => ct_1 <= "00011001"; --9
when others => ct_1 <= "11111111"; --no
end case;
case count2 is
when "0000" => ct_2 <= "00000011"; --0
when "0001" => ct_2 <= "10011111"; --1
when "0010" => ct_2 <= "00100101"; --2
when "0011" => ct_2 <= "00001101"; --3
when "0100" => ct_2 <= "10011001"; --4
when "0101" => ct_2 <= "01001001"; --5
```

```
when "0110" => ct_2 <= "01000001"; --6
when "0111" => ct_2 <= "00011111"; --7
when "1000" => ct_2 <= "00000001"; --8
when "1001" => ct_2 <= "00011001"; --9
when others => ct_2 <= "11111111"; --no
end case;
case count3 is
when "0000" => ct_3 <= "00000010"; --0
when "0001" => ct_3 <= "10011110"; --1
when "0010" => ct_3 <= "00100100"; --2
when "0011" => ct_3 <= "00001100"; --3
when "0100" => ct_3 <= "10011000"; --4
when "0101" => ct_3 <= "01001000"; --5
when "0110" => ct_3 <= "01000000"; --6
when "0111" => ct_3 <= "00011110"; --7
when "1000" => ct_3 <= "00000000"; --8
when "1001" => ct_3 <= "00011000"; --9
when others => ct_3 <= "11111110"; --no
end case;
case count4 is
when "0000" => ct_4 <= "11111111"; --0
when "0001" => ct_4 <= "10011111"; --1
when "0010" => ct_4 <= "00100101"; --2
when "0011" => ct_4 <= "00001101"; --3
when "0100" => ct_4 <= "10011001"; --4
when "0101" => ct_4 <= "01001001"; --5
when "0110" => ct_4 <= "01000001"; --6
when "0111" => ct_4 <= "00011111"; --7
when "1000" => ct_4 <= "00000001"; --8
when "1001" => ct_4 <= "00011001"; --9
```

```
when others => ct_4 <= "11111111"; --no
end case;
end process;

process(CLK_240Hz)
variable AN_count : unsigned(1 downto 0) := "00";
begin
if(rising_edge(CLK_240Hz)) then
AN_count := AN_count + 1;
end if;
case AN_count is
when "00" => AN <= "1110"; cathode_out <= ct_1;
when "01" => AN <= "1101"; cathode_out <= ct_2;
when "10" => AN <= "1011"; cathode_out <= ct_3;
when "11" => AN <= "0111"; cathode_out <= ct_4;
when others =>
end case;
end process;
end Behavioral;
----------------------------------------------------------------------------------
-- Company:
-- Engineer: emre şaş
--
-- Create Date: 29.10.2023 14:43:44
-- Design Name:
-- Module Name: agaaaa - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
```

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

--

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

TEST BENCH CODE

--

-- Company:

-- Engineer: emre şaş

--

-- Create Date: 29.10.2023 14:43:44

-- Design Name:

-- Module Name: test

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

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

entity Chronometer_Testbench is
end Chronometer_Testbench;

architecture testbench of Chronometer_Testbench is
    signal start_signal, reset_signal, centi: std_logic := '0';
    signal cathode_o: std_logic_vector(7 downto 0);
    signal AN: std_logic_vector(3 downto 0);
begin
    UUT: entity work.Chronometer
        port map (
            start_signal => start_signal,
            reset_signal => reset_signal,
            centi => centi,
            cathode_o => cathode_o,
            AN => AN
```

```
        );

    process
    begin
        while now < 2000 ns loop -- Simulate for 2 us
            centi <= '0';
            wait for 5 ns;
            centi <= '1';
            wait for 5 ns;
        end loop;
        wait;
    end process;

    process
    begin
        wait for 10 ns; -- Wait for initial signals stabilization

        start_signal <= '1';
        wait for 1000 ns; -- Wait for 1 us

        reset_signal <= '1';
        wait for 10 ns;
        reset_signal <= '0';

        wait;
    end process;
end architecture testbench;
```

Constraints

set_property PACKAGE_PIN W5 [get_ports {centi}]

set_property IOSTANDARD LVCMOS33 [get_ports {centi}]

#create_clock -add -name sys_clk_pin -period 10.00 -waveform {01} [get_ports{centi}]

set_property PACKAGE_PIN U18 [get_ports {start_signal}]

set_property IOSTANDARD LVCMOS33 [get_ports {start_signal}]

set_property PACKAGE_PIN T17 [get_ports {reset_signal}]

set_property IOSTANDARD LVCMOS33 [get_ports {reset_signal}]

set_property PACKAGE_PIN U2 [get_ports {AN[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]

set_property PACKAGE_PIN U4 [get_ports {AN[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]

set_property PACKAGE_PIN V4 [get_ports {AN[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]

set_property PACKAGE_PIN W4 [get_ports {AN[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]

set_property PACKAGE_PIN W7 [get_ports {cathode_out[7]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cathode_out[7]}]

set_property PACKAGE_PIN W6 [get_ports {cathode_out[6]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cathode_out[6]}]

set_property PACKAGE_PIN U8 [get_ports {cathode_out[5]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cathode_out[5]}]

set_property PACKAGE_PIN V8 [get_ports {cathode_out[4]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cathode_out[4]}]

set_property PACKAGE_PIN U5 [get_ports {cathode_out[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cathode_out[3]}]

set_property PACKAGE_PIN V5 [get_ports {cathode_out[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cathode_out[2]}]

set_property PACKAGE_PIN U7 [get_ports {cathode_out[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cathode_out[1]}]

set_property PACKAGE_PIN V7 [get_ports {cathode_out[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {cathode_out[0]}]