Lab 5: 7 segment display

Sait Sarper Özaslan

22002861

Lab4

Section 102-2

Aim:

The aim of this lab was to implement a design on BASYS3 that uses 7 segment display while the segments are working simultaneously. This lab was also aimed at the topic of persistence of vision which is the core idea behind 7 segment display working properly to the naked eye.

Questions:

- 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?

We can create a clock divider whose number of ticks will total 100 MHZ in the end.

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

We cannot create a clock with arbitrary frequency lower than that of the internal clock as the number of ticks cannot be anything other than an integer value. For instance, we can choose to create a clock that toggles 4 ticks every toggle we would be creating 100/4 = 25 Mhz clock.

Since the number of ticks has to be integer number, we cannot create arbitrary frequency lower than that of the internal clock. We can only create between 1 tick every toggle to 100 Mega (times 10^6) ticks that is to way we would be making clocks with the frequency of 1 Hz to 100 Mhz.

Methodology:

To begin with, the first task was to decide on what kind of implementation I could use. Through the suggestions of the lab file, I decided to make a decimal counter first, but I realized there were confusing mismatches in the logic vector ranges I had to use while writing it's VHDL code, so I decided to make a hexadecimal display that has more compact design and less ambiguity. After deciding on the design I needed to understand how a 7-segment display works.

Essentially a 7-segment display consists of 4 anodes, each of which corresponds to a digit, and 7 cathodes each of which correspond to segments of the digits. Normally to light up a segment anode is driven high while the cathode is driven low However due to usage of transistor to drive current in BASYS3, the anode is inverted. This situation impacts the VHDL code that will be written.

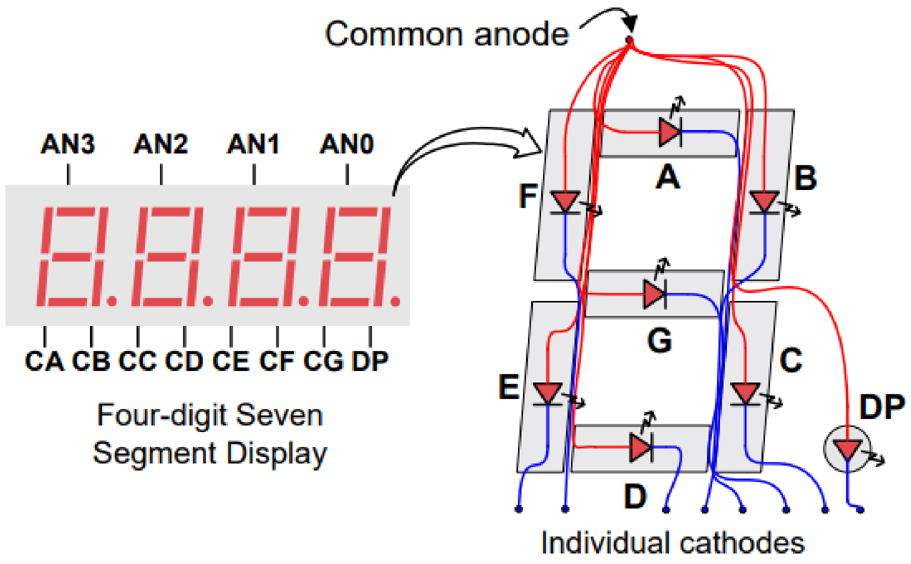

Figure 1: Anodes and Cathodes of 7-segment display

To display all digits appropriately all 4 digits should be driven 1 to 16 ms which translates refresh frequency of 1kHZ to 60 Hz. Since BASYS3 has an internal clock of 100 MHz, I needed to use a phase(clock) divider to implement the design successfully. After deciding on what I needed write a testbench to check whether the design is correct or not.

Design Specifications:

The VHDL code was written in a modular fashion, under the main module I wrote the driver 7-segment module. There is also another sub module under driver 7 segment called cathodes and there is also the sclock module which is the clock divider. To begin with:

Main module consists of 2 inputs and 2 outputs all of which are:

Clk(in): Base 100 MHz frequency of BASYS3,

Res(in): the reset function for the 7-segment display, resets the counter to 0

Dig(out): The output for 4 of which digits is going to be displayed.

Seg(out): The output for 7 of which digits needs to be displayed.

The anode and cathode outputs get their values from the driver 7-segment which consists of 4 inputs and 2 outputs. The selection of anode is written through a multiplexer. To understand driver 7 segment, I need to explain the sclock module first. sclock module is essentially a clock divider that is initialized under driver 7-segment and consists of 2 inputs and 4 outputs:

- Clk(in): Base 100MHz frequency of BASYS3
- Res(in): the reset function
- T_counter(out): Counts the clock ticks according to base counter
- Phase(out): Created from main clock which created the on and off situation for the segments.
- Second(out): counts the second
- Sec_en(out): in each second it is asserted 1.

There is also cathodes module under driver 7 segment module which connects cathodes of 7 segment to the driver, consists of 1 input LED which is for the LED combination of segments and 1 output cathode, which is the segments of the 7-segment display. Finally there is the driver 7 segment which consists of 5 inputs and 2 outputs:

- Clk(in): Base 100 MHz frequency clock of BASYS3.
- Res(in): The reset function.
- Phase(in): The divided phase from the main clock.
- Num(in): The "second" output from the clock module.
- Sec_en(in): Asserted 1 in each second, else 0
- Dig(out): Anodes of 7-segment display.
- seg(out): Cathodes of the 7-segment display.

The anodes are selected through a multiplexer and cathodes are derived from the "cathode" module which consists of 1 input ledc, which is the led combination and 1 output cathode, which shows which segments are going to be lighted.

Finally, I needed to use 2 different testbenches to check whether the design was working properly. One for the sclock module and one for the main module.

Results:

After writing the vhdl code for the clock divider I wrote the test bench code for it to see whether it was working properly. The phasor starts from 00 and goes up to 11. After this it restarts and the phase was picked between 18th and 19th digits of the t_counter which corresponds to 10.5 miliseconds of refresh rate. This value was low enough to achieve persistence of vision. The res is 1 at initial point and then it is 0, finally the sec_en output is asserted 1 in each second.

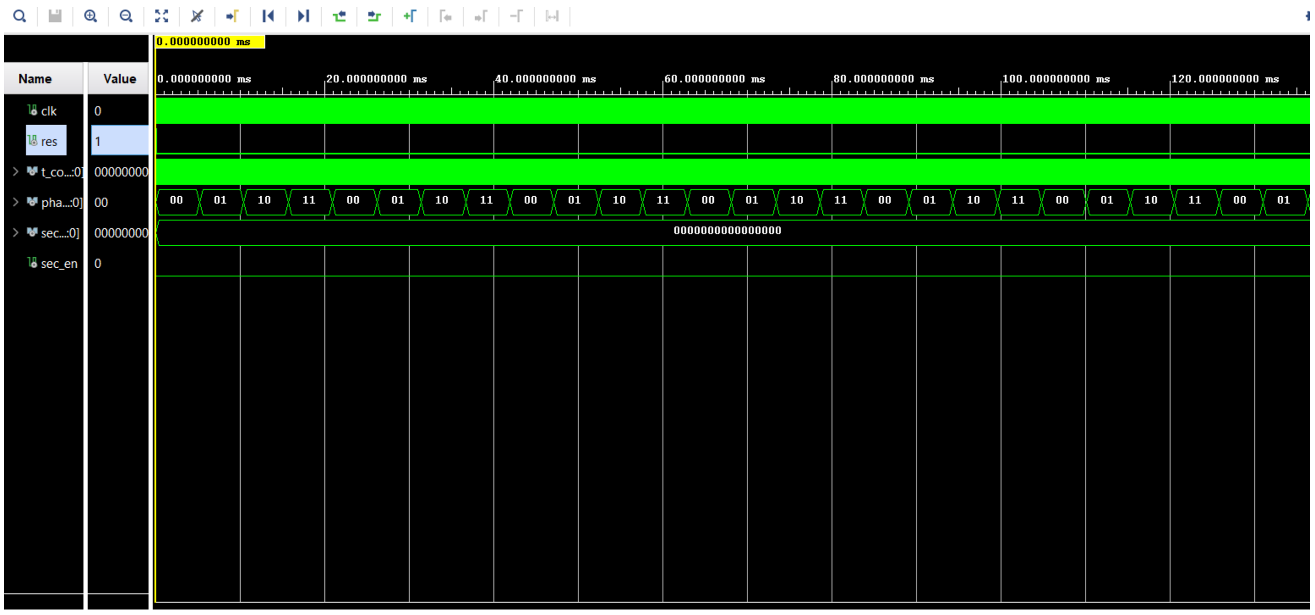

Figure 2: Test Bench, the res value is 1 in the beginning

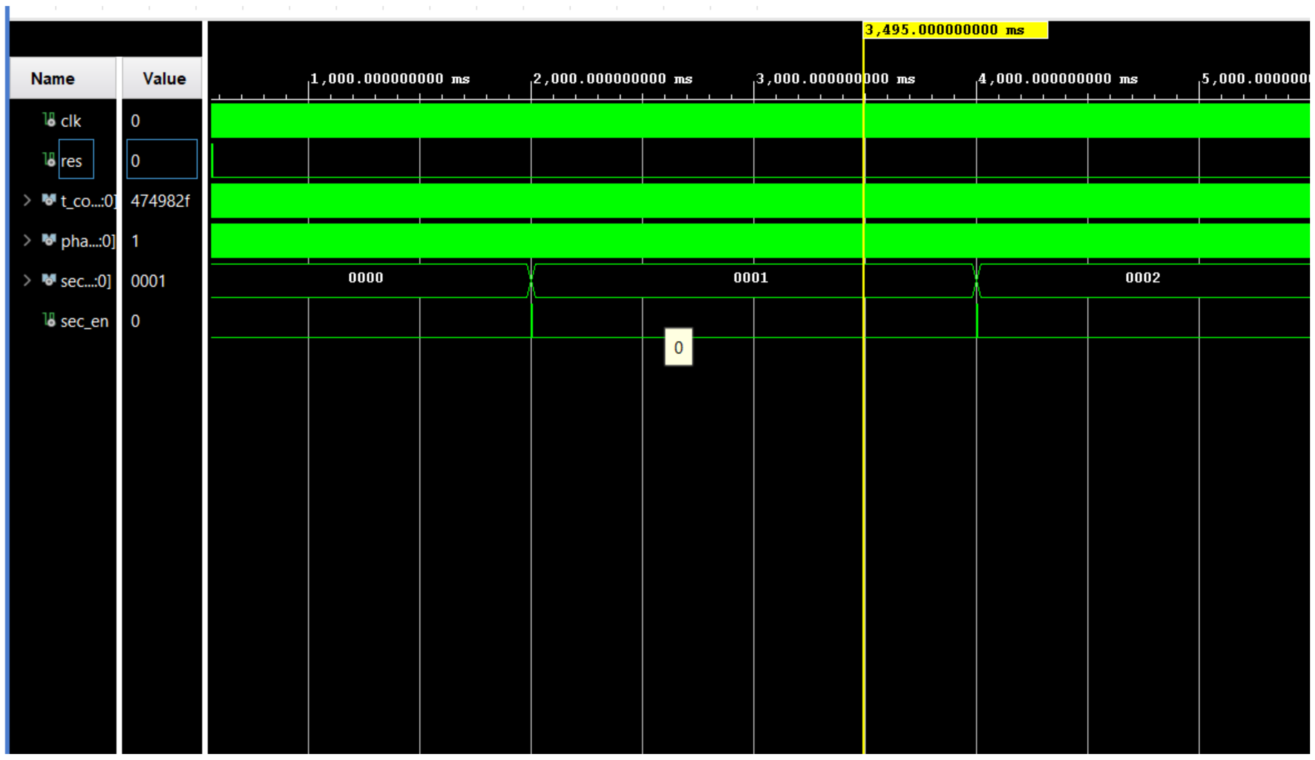

Figure 3: Test Bench the display of increments.

After seeing that the clock testbench seemed to work without any issues I wrote the remaining code and wrote a testbench for the main module. According to the testbench and idea of this counter, which is to increment the value displayed in the 7-segment by 1 each second. As this is an hexadecimal display I expected an increment in the 3rd digit after the 15 seconds.

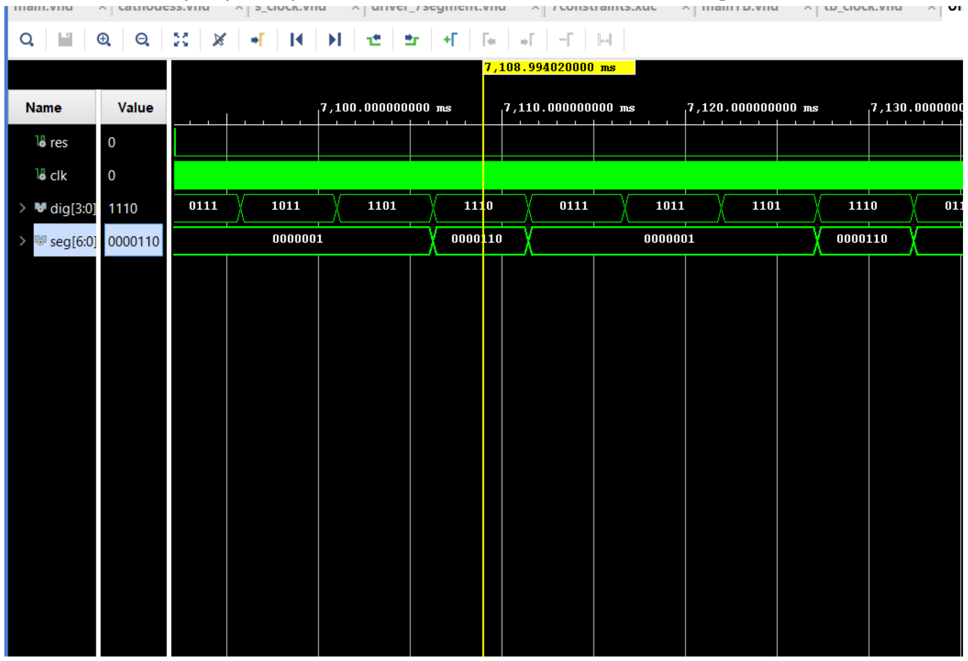

Figure 4: Test Bench for main module, value 3 in decimal

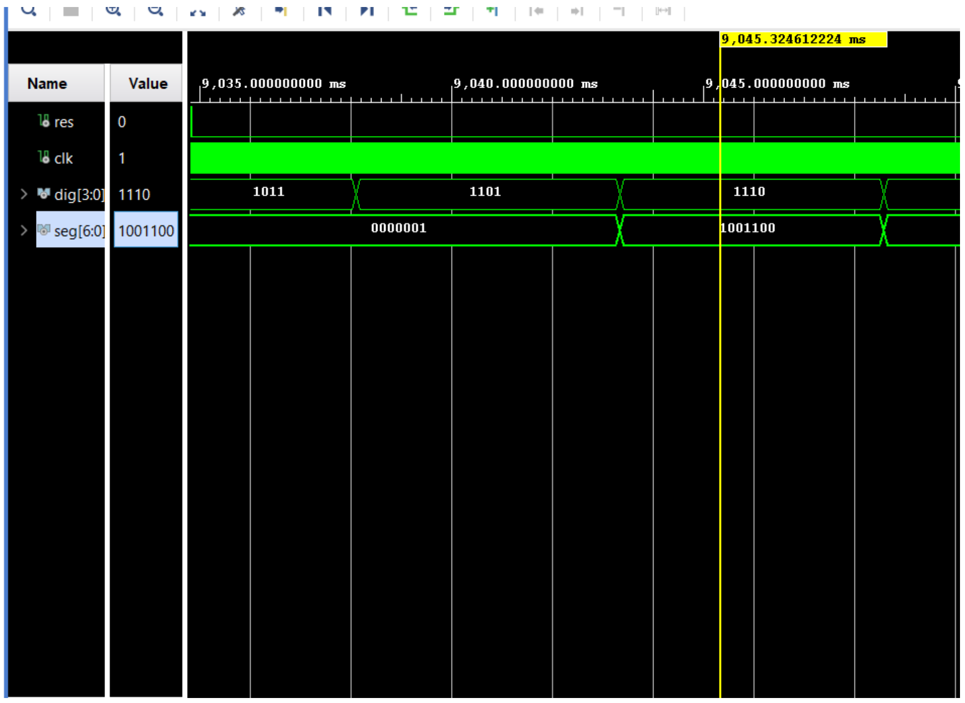

Figure 5: Value 5 in decimal of the test bench main module

After seeing the test bench was working appropriately, I implemented this design to BASYS3. Here are two examples from BASYS3:

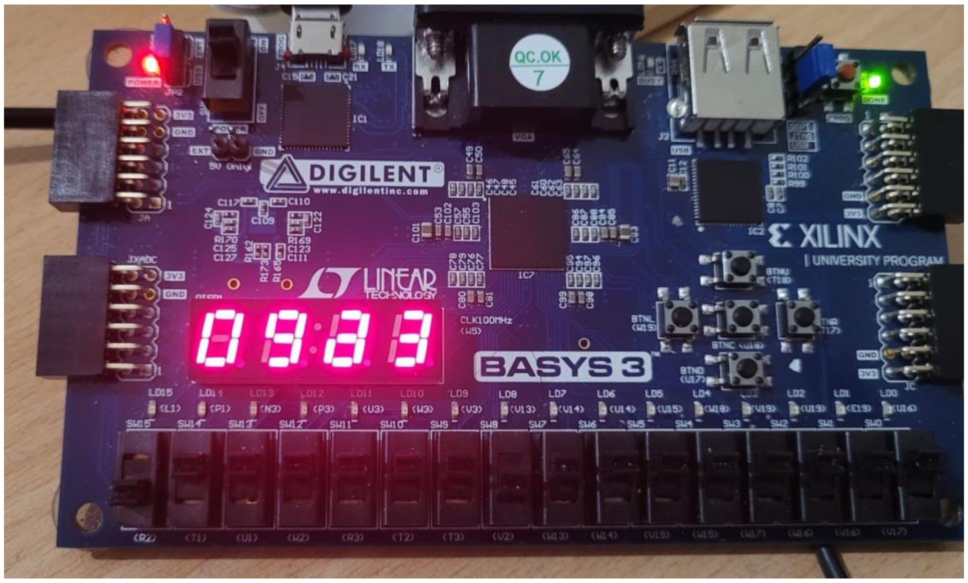
Figure 6: Case 9a3 in hexadecimal(2467 seconds)

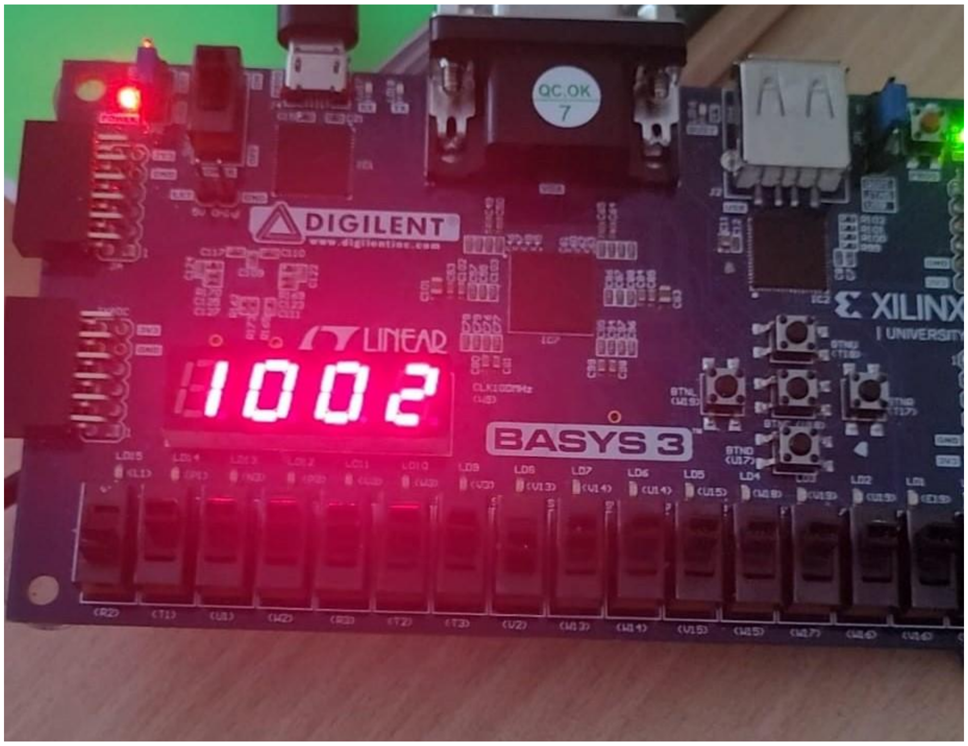
Figure 7: Case 1002 in hexadecimal(4092 seconds)

Conclusion:

In this lab I figured out how to use a clock and how to divide the clock frequency to achieve persistence of vision effect. I made hexadecimal counter which was quite a challenge as I was confused in how I was going to make the led combinations at first and how I was going to divide the clock . There were some other complications as some initial values I wrote for the sclock module didn’t work due to failing at how to implement the ğhase.. However, the result seemed to align with the test bench so overall it was successful.

Appendix:

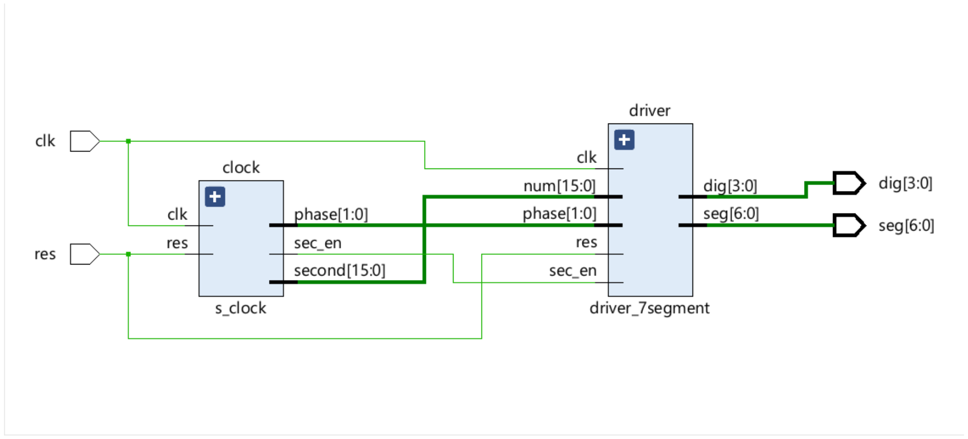

Figure 8:RTL schematic of the main module

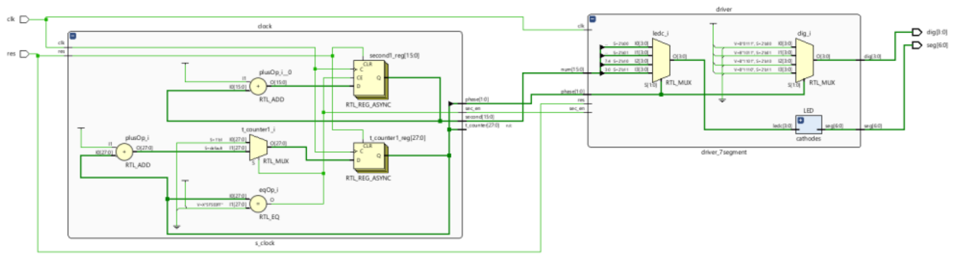

Figure 9: RTL schematic of driver_7segment and s_clock module

Main.vhdl

entity main is

Port (clk: in std_logic; -- clock input

res: in std_logic; -- reset input

dig: out std_logic_vector(3 downto 0);

seg: out std_logic_vector(6 downto 0)

);

```
end main;
--------------------------------------------------------------------------
architecture smain of main is
            signal phase: std_logic_vector(1 downto 0);
            signal enable: std_logic;
            signal num: std_logic_vector(15 downto 0);
begin
--------------------------------------------------------------------------
clock: entity work.s_clock(sclock)
            port map(clk => clk, res => res,
            phase => phase,
            sec_en => enable,
            second => num);
--------------------------------------------------------------------------
driver: entity work.driver_7segment (sdriver)
            port map(clk => clk, res => res, phase => phase, num => num,
            sec_en => enable, dig => dig, seg => seg);
end smain;
```

Driver_7segment.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
--------------------------------------------------------------------------
entity driver_7segment is
Port (clk: in std_logic;
res: in std_logic;
phase: in std_logic_vector(1 downto 0);
num: in std_logic_vector(15 downto 0);

sec_en: in std_logic;

dig : out std_logic_vector(3 downto 0);

seg: out std_logic_vector(6 downto 0)

);

end driver_7segment;

--------------------------------------------------------------------------------

architecture sdriver of driver_7segment is

            signal ledc: std_logic_vector(3 downto 0);

begin

--------------------------------------------------------------------------------

process(phase) begin

case phase is

            when "00" => dig <= "0111"; -- 1st digit

            ledc <= num(15 downto 12);

            when "01" => dig <= "1011"; -- 2nd digit

            ledc <= num(11 downto 8);

            when "10" => dig <= "1101"; -- 3rd digit

            ledc <= num(7 downto 4);

            when "11" => dig <= "1110"; -- 4th digit

            ledc <= num(3 downto 0);

            when others => dig <= "1111";

    end case;

end process;

--------------------------------------------------------------------------------

LED: entity work.cathodes (sLED)

port map(LEDc=> ledc, seg => seg);

--------------------------------------------------------------------------------

end sdriver;
```

Sclock.vhdl

```
entity s_clock is

Port ( clk: in std_logic;

res: in std_logic;

t_counter: out std_logic_vector(27 downto 0);

phase: out std_logic_vector(1 downto 0);

second: out std_logic_vector(15 downto 0);

sec_en: out std_logic);

end s_clock;

-------------------------------------------------------------------------

architecture sclock of s_clock is

        signal t_counter1: std_logic_vector(27 downto 0);

        signal second1: std_logic_vector(15 downto 0):= (others => '0');

begin

-------------------------------------------------------------------------

process(CLK) begin

if(res = '1') then

        t_counter1 <= (others => '0');

        second1 <= (others => '0');

else

        if rising_edge(CLK) then

                t_counter1 <= t_counter1 + '1';

        if t_counter1 =x"5F5E0FF" then

                second1 <= second1 + '1';

                t_counter1 <= (others => '0');

                end if;

        end if;

end if;

end process;
```

```
----------------------------------------------------------------------------
sec_en <= '1' when t_counter1 =x"5F5E0FF" else '0';
phase <= t_counter1(19 downto 18);
t_counter <= t_counter1;
second <= second1;
----------------------------------------------------------------------------
end sclock;
```

Cathodes.vhdl

```
entity cathodes is
Port ( ledc: in std_logic_vector(3 downto 0);
seg: out std_logic_vector(6 downto 0));
end cathodes;
----------------------------------------------------------------------------
architecture sLED of cathodes is
begin
----------------------------------------------------------------------------
process(ledc) begin
case ledc is
   when "0000" => seg <= "0000001"; --0
   when "0001" => seg <= "1001111"; --1
   when "0010" => seg <= "0010010"; --2
   when "0011" => seg <= "0000110"; --3
   when "0100" => seg <= "1001100"; --4
   when "0101" => seg <= "0100100"; --5
   when "0110" => seg <= "0100000"; --6
   when "0111" => seg <= "0001111"; --7
   when "1000" => seg <= "0000000"; --8
   when "1001" => seg <= "0000100"; --9
   when "1010" => seg <= "0000010"; --a
   when "1011" => seg <= "1100000"; --b
   when "1100" => seg <= "0110001"; --c
   when "1101" => seg <= "1000010"; --d
   when "1110" => seg <= "0110000"; --e
   when "1111" => seg <= "0111000"; --f
   when others => seg <= "1111111"; --closed
end case;
end process;
--------------------------------------------------------------------------
end sLED;
```

MainTB

```
entity mainTB is
end mainTB;
--------------------------------------------------------------------------
architecture tb of mainTB is
   signal res,clk: std_logic;
   signal dig: std_logic_vector(3 downto 0);
   signal seg: std_logic_vector(6 downto 0);
begin
--------------------------------------------------------------------------
dut: entity work.main(smain)
            port map (clk => clk, res=>res,
            dig=> dig,
            seg=> seg
            );
--------------------------------------------------------------------------
clock_process :process
```

```
begin
        clk <= '0';
    wait for 10 ns;
        clk <= '1';
    wait for 10 ns;
end process;
--------------------------------------------------------------------------
stim_proc: process
begin
        res <= '1';
    wait for 20 ns;
        res <= '0';
   wait;
end process;
--------------------------------------------------------------------------
end tb;
```

ClockTB:

```
entity clockTB is
end clockTB;
--------------------------------------------------------------------------
architecture tb_clock of clockTB is
    signal clk,res: std_logic;
    signal t_counter: std_logic_vector(27 downto 0);
    signal phase: std_logic_vector(1 downto 0);
    signal second: std_logic_vector(15 downto 0);
    signal sec_en: std_logic;
begin
--------------------------------------------------------------------------
```

```
dut: entity work.s_clock
            port map ( clk => clk, res => res, t_counter => t_counter,
            phase => phase, second => second, sec_en => sec_en
            );
--------------------------------------------------------------------------------
clock_process: process
begin
       clk <= '0';
   wait for 10ns;
        clk <= '1';
   wait for 10ns;
end process;
--------------------------------------------------------------------------------
stim_proc: process
begin
       res <= '1';
   wait for 20 ns;
       res <= '0';
   wait;
end process;
--------------------------------------------------------------------------------
end tb_clock;
```