# Bilkent University Electrical and Electronics Department

# EE102-03 Lab 5 Report:

# Seven-Segment Display

Cankut Bora Tuncer – 22001770

## Purpose:

This lab aims to design a seven-segment display driver to light up the LEDs on the segment sequentially. There is an up-counting operation on the hexadecimal base.

## Methodology:

The working principle of the seven-segment display is straightforward. In each segment, there are 7 LEDs. They are driven by two inputs, the anode and the cathode (Figure 1). The anodes drive the segments; the cathodes drive the LEDs. It is not possible to light multiple segments. Thus the segments should turn on-off very quickly so that the "persistence of vision" is achieved.

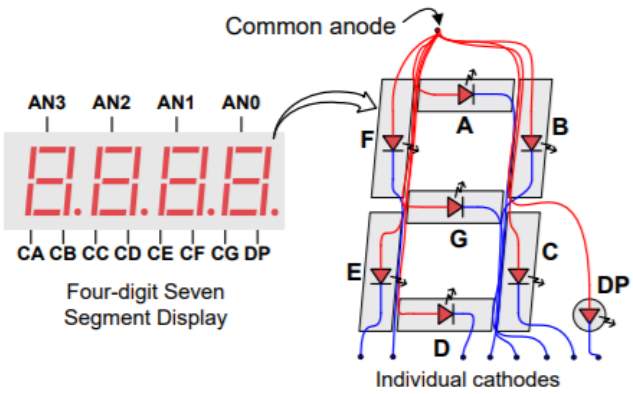


*Figure 1 The Seven Segment Display*

The initial task was to decide which clock frequency to use. The Basys3 has an integrated clock of 100MHz. The base clock is of the design is chosen as this value. To turn on and off the anodes, a phase clock must be selected between 1 and 16ms. For the last clock operation, a second indicator is initialized to count the seconds. The next step is to design the segment driver. After the design process, the testbench and constraint file is generated. As for the final step, tests are conducted on the waveform graph and also Basys3.

## Design Specifications:

There are 2 inputs and 2 outputs. The first input is the clock value and it is fixed(100MHz). The second input is the reset selection. When reset is asserted, both the clock and the display are restarted. Anode (4 bit) and cathode(7 bit) outputs are selected to light up the LEDs.

The 7 segment display driver is designed in a modular fashion. The "main_7segment.vhd" is the top module(), which initializes the clock and driver module. The "segment_clock.vhd" module has 2 inputs and 4 outputs. It can be summarized as:

- clk: input, the 100Mhz clock from Basys3
- reset: input, the switch input to restart the counting sequence
- clk_counter: output, it counts the clock ticks(not crucial for the design, is there for testing)
- phase_counter: output, the phase clock divided from the main clock to create the on-off sequence of the segments.
- sec: output, counts the seconds

- sec_en: output, in each second it is asserted 1 else 0

The phase_counter output is a clock divider. It is obtained from indexing the clk_counter between 19 and 18th bits which corresponds to 10.5 ms each. When the clk_counter reaches to 1 second, the sec output is incremented by 1. Also the sec_en output is asserted 1. The outputs from the clock module are inputted to the "segment7_driver" module. The driver module has 2 inputs and 4 outputs. It can be summarized as:

- clk: input, the 100Mhz clock from Basys3
- reset: input, the switch input to restart the counting sequence
- phase_counter: input, the phase clock divided from the main clock to create the on-off sequence of the segments.
- number: input, the sec output from clock module
- sec_en: input, in each second it is asserted 1 else 0
- anode: output, drives the segments
- cathode: output, drives the LEDs

The anode selection is implemented with a "case-when" statement; i.e., when the phase_counter is "00", the first segment will light up; the segment displays the first digit of the hexadecimal number according to the cathode combination. The cathode combination is under the "LED_register.vhd" module. The LED module has 1 input and 1 output. It can be summarized as:

- led_comb: input, the number to display
- cathode: output, the cathode combination

The cathode pins are asserted to 0 to light up the desired number to enable current flow over LEDs.

There are 2 seperate testbenches: “mainTB.vhd” and “clockTB.vhd”. The main testbench simulates the main module; the clock testbench simulates the clock module. The DUT’s(Design Under Test) are initialized via. “entity work.<module_name>”. There are 2 processes, the clock process, and the stimulated process. The clock process is to toggle the clock between 1 and 0 every 10ns. The stimulated process contains the actual simulation of the design.

## Results:

The most crucial part of the design was to generate accurate clock and counter values for the phase and second counter. Thus a separate testbench is created for only simulating the clock module. As can be seen from Figures 2 and 3, the clock module functions as intended. The clock toggles every 10ns and resets asserted at the start. The phasor starts from “00” and increments until “11” then restarts, simulating the anode selection. When the clk_counter reaches 1, the sec is incremented by 1, and sec_en is 1 at the point where clk_counter totals at 1 second.

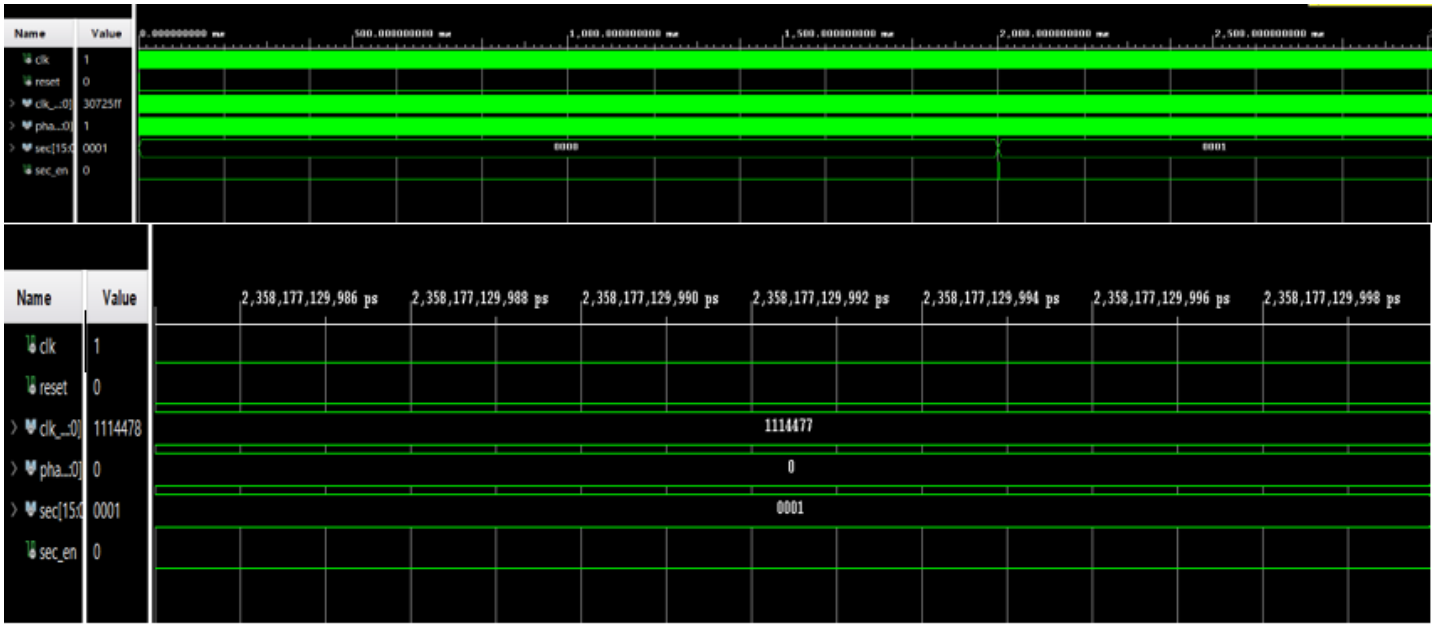


*Figure 2 & 3 The waveforms – clockTB.vhd*

As for the main testbench, the outputs, anode, and cathode should work in parallel to capture the “persistence of vision” effect. The rightmost digit counts the seconds, and the second rightmost digit counts every 15 seconds (since the counting sequence is on a hexadecimal base). The second leftmost and the leftmost digit counts according to the previous digits. Until the time reaches 1 second, the cathode combination must be all the same and 0(“1111110”). After that, the rightmost digit is incremented by 1. The reset input is asserted at the start.


*Figure 4 The waveforms – maimTB.vhd*

The schematic form of the design can be seen in Figure 5. For convenience, a modular hierarchy is preferred.

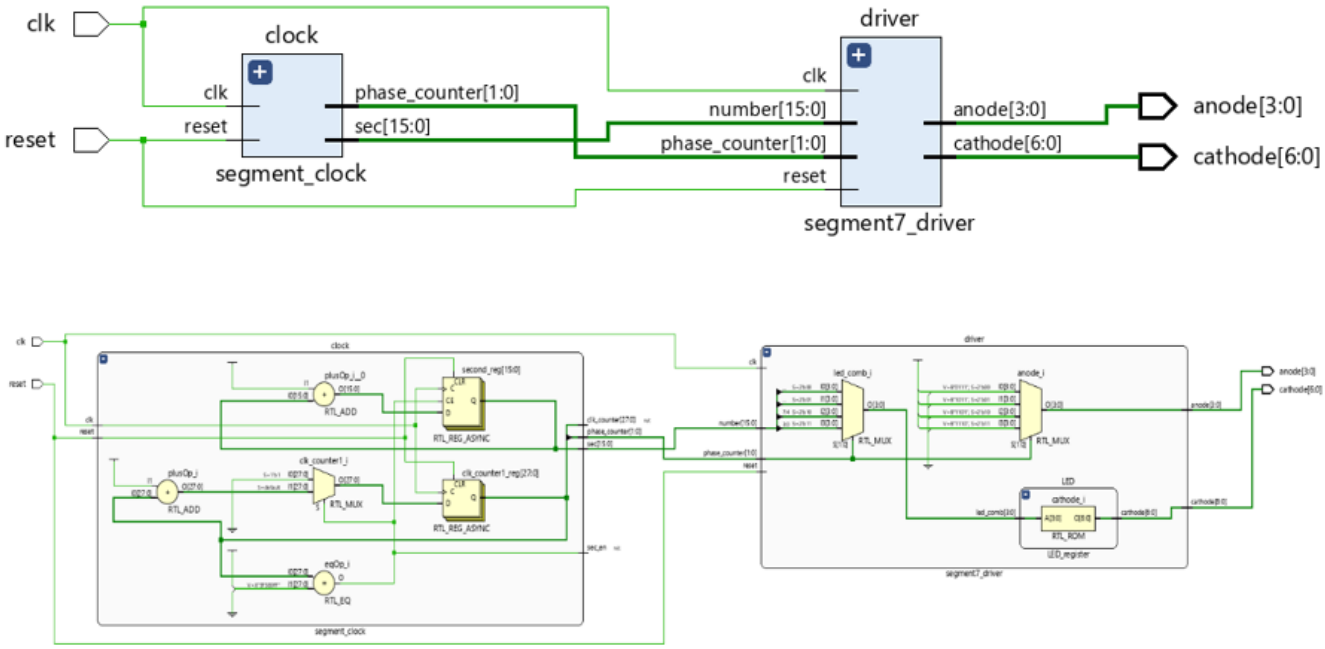


*Figure 5 The RTL Schematic*

Finally, the code is implemented on the Basys3 board. The configuration can be seen in Figure 6.

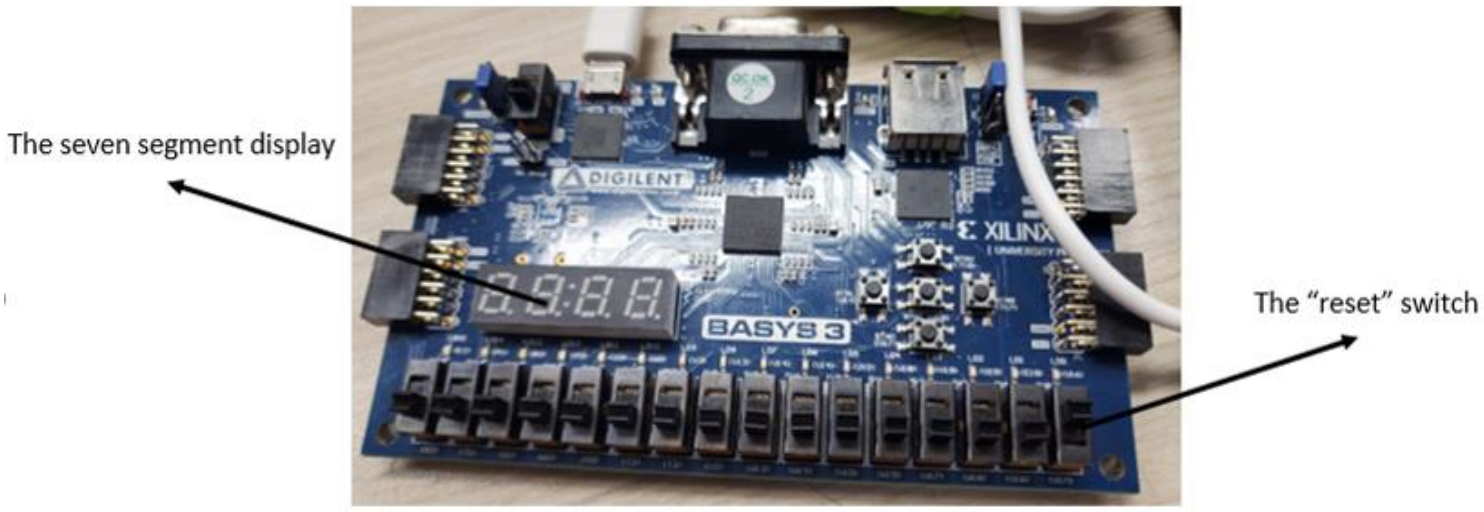


*Figure 6 The Basys3 Configuration*

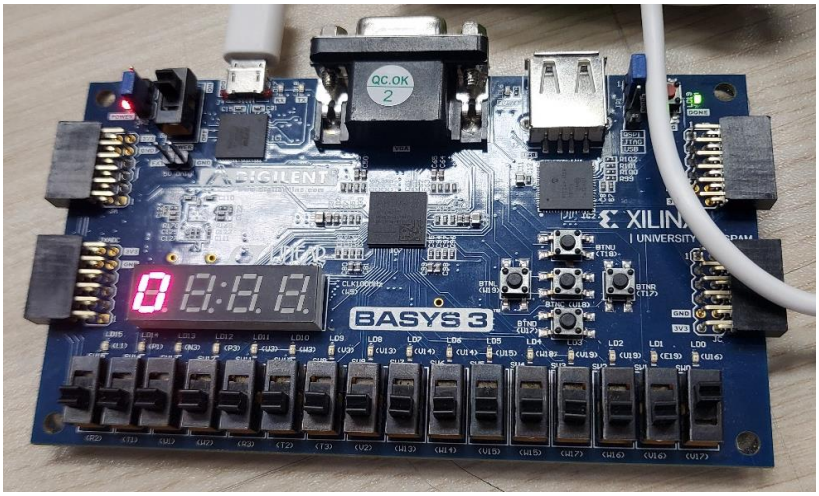


*Figure 7 When the reset is asserted*

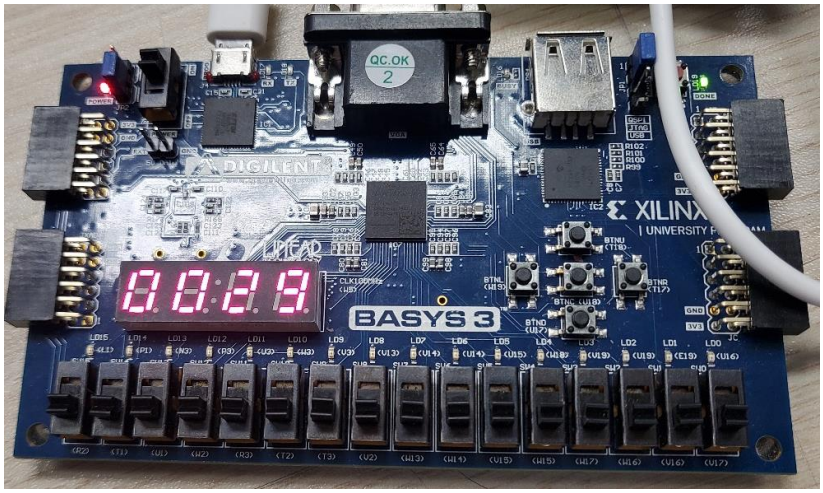


*Figure 8 After 41 seconds, 29 in hexadecimal base*

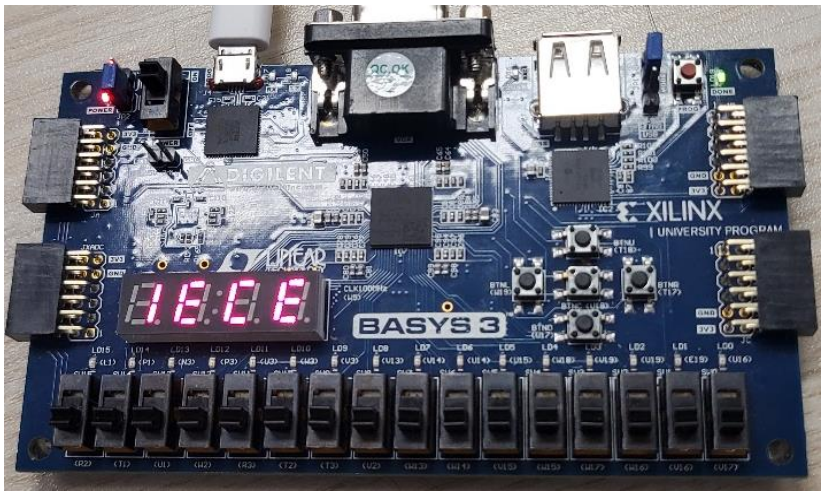


*Figure 9 After 7886 seconds, 1ece in hexadecimal base*

# Conclusion:

The purpose of this lab was to design a 7 segment display driver on VHDL. The simulation and implementation results were consistent with the expected results. The internal clock of the Basys 3 is 100MHz, which is the base clock utilized in the design. However, a slower clock could be used. To do so, there could be used a clock divider. For instance, a 50MHz clock can be achieved from 100MHz if we toggle the 50Mhz clock for every 2 ticks of the 100MHz clock. However, it is impossible to create a clock with an arbitrary value lower than the base clock of the Basys3. The clock divider works by toggling the divided clock according to the preselected tick count of the base clock. For instance, if we choose to toggle 2 ticks per toggle, we choose to create 100/2 = 50Mhz clock. Since we cannot select fractional ticks per toggle number, the divisor must be an integer value. Thus we can divide the 100Mhz clock by between 1-100M per toggle(100Mhz-1Hz).

# Appendix:

## main_7segment.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
----------------------------------------------------------------------------
entity main_7segment is
    Port ( clk: in std_logic; -- clock input
           reset: in std_logic; -- reset input
           anode: out std_logic_vector(3 downto 0);
           cathode: out std_logic_vector(6 downto 0)
     );
end main_7segment;
----------------------------------------------------------------------------
architecture rtl of main_7segment is
    signal phase_counter: std_logic_vector(1 downto 0);
    signal sec_enable: std_logic;
    signal number: std_logic_vector(15 downto 0);
begin
----------------------------------------------------------------------------
clock: entity work.segment_clock(rtl_clock)
    port map(clk => clk, reset => reset,
             phase_counter => phase_counter,
             sec_en => sec_enable,
```

```
        sec => number);
-------------------------------------------------------------------------
driver: entity work.segment7_driver(rtl_driver)
   port map(clk => clk, reset => reset, phase_counter => phase_counter, number => number,
        sec_en => sec_enable, anode => anode, cathode => cathode);
end rtl;
```

## segment7_driver.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
-------------------------------------------------------------------------
entity segment7_driver is
   Port (clk: in std_logic;
        reset: in std_logic;
        phase_counter: in std_logic_vector(1 downto 0);
        number: in std_logic_vector(15 downto 0);
        sec_en: in std_logic;
        anode : out std_logic_vector(3 downto 0);
        cathode: out std_logic_vector(6 downto 0)
        );
end segment7_driver;
-------------------------------------------------------------------------
architecture rtl_driver of segment7_driver is
   signal led_comb: std_logic_vector(3 downto 0);
   begin
-------------------------------------------------------------------------
   process(phase_counter) begin
```

```
        case phase_counter is
            when "00" => anode <= "0111"; -- 1st digit
                    led_comb <= number(15 downto 12);
            when "01" => anode <= "1011"; -- 2nd digit
                    led_comb <= number(11 downto 8);
            when "10" => anode <= "1101"; -- 3rd digit
                    led_comb <= number(7 downto 4);
            when "11" => anode <= "1110"; -- 4th digit
                    led_comb <= number(3 downto 0);
            when others => anode <= "1111";
        end case;
    end process;
----------------------------------------------------------------------------
LED: entity work.LED_register(rtl_LED)
    port map(led_comb=>led_comb, cathode => cathode);
----------------------------------------------------------------------------
end rtl_driver;
```

## segment_clock.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD;
----------------------------------------------------------------------------
entity segment_clock is
    Port ( clk: in std_logic;
           reset: in std_logic;
           clk_counter: out std_logic_vector(27 downto 0);
           phase_counter: out std_logic_vector(1 downto 0);
           sec: out std_logic_vector(15 downto 0);
```

```
        sec_en: out std_logic);
end segment_clock;
--------------------------------------------------------------------------------
architecture rtl_clock of segment_clock is
    signal clk_counter1: std_logic_vector(27 downto 0);
    signal second: std_logic_vector(15 downto 0):= (others => '0');
begin
--------------------------------------------------------------------------------
    process(CLK) begin
        if(reset = '1') then
            clk_counter1 <= (others => '0');
            second <= (others => '0');
        else
            if rising_edge(CLK) then
                clk_counter1 <= clk_counter1 + '1';
                if clk_counter1 =x"5F5E0FF" then
                    second <= second + '1';
                    clk_counter1 <= (others => '0');
                end if;
            end if;
        end if;
    end process;
--------------------------------------------------------------------------------
sec_en <= '1' when clk_counter1 =x"5F5E0FF" else '0';
phase_counter <= clk_counter1(19 downto 18);
clk_counter <= clk_counter1;
sec <= second;
--------------------------------------------------------------------------------
end rtl_clock;
```

## LED_register.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------------------------------------
entity LED_register is
    Port ( led_comb: in std_logic_vector(3 downto 0);
           cathode: out std_logic_vector(6 downto 0));
end LED_register;
----------------------------------------------------------------------------
architecture rtl_LED of LED_register is
begin
----------------------------------------------------------------------------
    process(led_comb) begin
        case led_comb is
            when "0000" => cathode <= "0000001";
            when "0001" => cathode <= "1001111";
            when "0010" => cathode <= "0010010";
            when "0011" => cathode <= "0000110";
            when "0100" => cathode <= "1001100";
            when "0101" => cathode <= "0100100";
            when "0110" => cathode <= "0100000";
            when "0111" => cathode <= "0001111";
            when "1000" => cathode <= "0000000";
            when "1001" => cathode <= "0000100";
            when "1010" => cathode <= "0000010";
            when "1011" => cathode <= "1100000";
            when "1100" => cathode <= "0110001";
            when "1101" => cathode <= "1000010";
            when "1110" => cathode <= "0110000";
            when "1111" => cathode <= "0111000";
```

```
            when others => cathode <= "1111111";
        end case;
    end process;
-------------------------------------------------------------------------------
end rtl_LED;
```

## mainTB.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-------------------------------------------------------------------------------
entity mainTB is
end mainTB;
-------------------------------------------------------------------------------
architecture tb of mainTB is
signal reset,clk: std_logic;
signal anode: std_logic_vector(3 downto 0);
signal cathode: std_logic_vector(6 downto 0);
begin
-------------------------------------------------------------------------------
   dut: entity work.main_7segment(rtl)
      port map (clk => clk, reset=>reset,
                anode => anode,
                cathode=> cathode
                );
-------------------------------------------------------------------------------
   clock_process :process
   begin
       clk <= '0';
       wait for 10 ns;
       clk <= '1';
```

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

## clockTB.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-------------------------------------------------------------------------
entity clockTB is
end clockTB;
-------------------------------------------------------------------------
architecture tb_clock of clockTB is
signal clk,reset: std_logic;
signal clk_counter: std_logic_vector(27 downto 0);
signal phase_counter: std_logic_vector(1 downto 0);
signal sec: std_logic_vector(15 downto 0);
signal sec_en: std_logic;
begin
-------------------------------------------------------------------------
dut: entity work.segment_clock
    port map ( clk => clk, reset => reset, clk_counter => clk_counter,
```

```
            phase_counter => phase_counter, sec => sec, sec_en => sec_en
        );
-------------------------------------------------------------------------------
    clock_process: process begin
        clk <= '0';
        wait for 10ns;
            clk <= '1';
        wait for 10ns;
    end process;
-------------------------------------------------------------------------------
    stim_proc: process
    begin
         reset <= '1';
       wait for 20 ns;
        reset <= '0';
       wait;
    end process;
-------------------------------------------------------------------------------
end tb_clock;
```