Alp Tekin

22403597

EE102-02

12.11.2025

Bilkent University

Department of Electrical and Electronics Engineering

Lab 5 Report: Seven-Segment Display

Purpose

The purpose of the experiment is to use seven-segment display on Basys3 FPGA. The seven-segment display should also show multiple digits simultaneously using the persistence of vision effect.

Questions

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

The internal clock frequency of the Basys 3 board is 100 MHz, which corresponds to a clock period of 10 nanoseconds. This 100 MHz signal is used as the input clock for my design.

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

We can use a clock divider to use slower clock signals. A slower clock can be generated by dividing the main 100MHz clock. For instance, in my design I divide the 100 MHz signal down to 1 Hz to create a one second clock for the counter. By changing the output state after a set number of input clock cycles, a slower clock signal can be achieved.

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

Since the clock frequency must be obtained by dividing the 100MHz base clock by an integer value, it is not possible to generate any arbitrary frequency. Therefore, we can only create frequencies that are integer divisions of the 100MHz clock.

Design Specifications

In this logic design there are two inputs and two outputs. The inputs are clock (clk) and reset. The clock represents the 100 MHz constant clock signal of Basys3 and reset represents the reset switch which is used to restart the seven-segment display. The anode and cathode outputs work together to drive the seven-segment display, activating specific segments to show digits. The anode (an[0:3]) is a 4-bit output signal, while the cathode (cat[0:6]) is a 7-bit output signal which corresponds to individual LED segments of each digits.

I designed the system in a modular fashion to make it more organized. The top-module and three sub-modules are:

- top_seven_segment.vhd
    - hex_sec_counter.vhd
    - segment_driver.vhd
    - hex_decoder.vhd

The top-module "top_seven_segment.vhd" includes the clock and driver modules and connect them together. It integrates all submodules to show the complete seven-segment display. The "clock_divider.vhd" module is a clock counter for display. It takes the 100 MHz internal clock signal of the Basys3 board and the reset input. The module divides the main clock to generate a slower clock signal. This signal is used to refresh the digits on seven-segment display at a visible state. There are two outputs: hex_out, which represents the 16-bit hexadecimal second counter, and phase_count, which is used to control the on–off sequence of the four seven-segment digits. The hex_out output holds the current count value in hexadecimal form, while the phase_count output enables the clock division.

The second module "segment_driver.vhd" takes the outputs of the clock divider as inputs (phase_count and the 16-bit value num_in). It performs as digit multiplexer for the four-digit common anode display. Based on the phase_count, it activates exactly one anode line (an_out) and selects the corresponding 4-bits from num_in. In other words, for instance when the phase_count is "00" AN3 is activated and the four most significant bit from num_in (num_in[12:15]) is assigned to that particular digit. After that this 4-bit output (hex_out_to_dec) is sent to decoder.

The third module is "hex_decoder.vhd" takes the hex_in 4-bit and converts it into the corresponding seven-segment representation to display hexadecimal values. This module produces the output cat_out, which represents the cathode combination for the segments of the seven-segment display.

Also, the reset switch is used to restart the system. When it is turned on, the display shows 0, and when it is turned off, the system continues counting.

Methodology

I created a logic design which counts up every second in hexadecimal number system. The value of the counter is displayed on the four digit seven-segment display of Basys3 board. The display is shown using clock divider, a segment divider and a hexadecimal decoder. The anode and cathode outputs are used together to control which digits and segments need to be shown in any given time. Although only one digit can be active at a moment, in my design the digits are switched fast enough to appear as they are all on simultaneously. The system is on and off quickly so that we can easily understand each digit. This effect is called the persistence of vision.

After designing the clock divider and counter modules, I implemented the segment driver to control the active digit by designing a multiplexer. Then I created a decoder to translate the 4-bit hexadecimal value into seven-segment format using a 4x16 decoder. A constraint file was also created to correctly map the FPGA pins to the Basys3 board connections such as the clock input, reset switch, anode, and cathode outputs. I used a testbench to simulate the design and I ensure that the system is working properly. Finally, I implemented the design on the Basys3 board to observe the output on the seven-segment display.

Results

First, the RTL schematic of the design is shown in Figure 1 and Figure 2

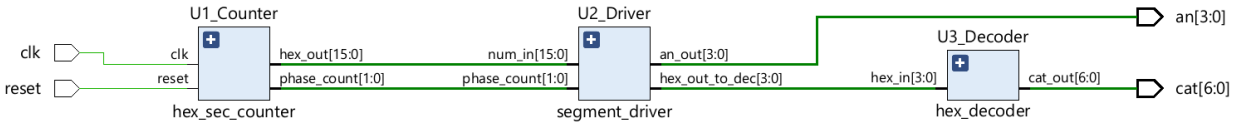

Figure 1: RTL schematic of the top_seven_segment.vhd

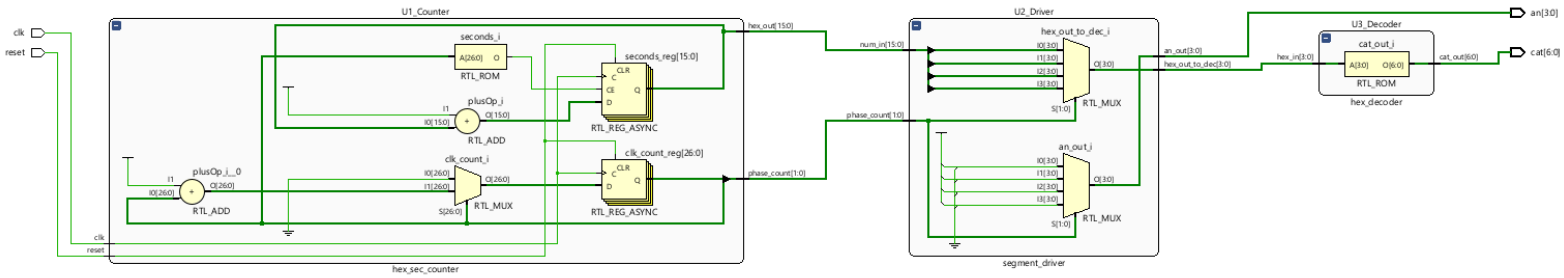

Figure 2: RTL schematic when the sub-modules are displayed

Before implementing the code on Basys3 board, I created a testbench code for the main module. The testbench for the main module shows that the design behaves properly when the simulation was made. Since the clock signal is one of the most essential parts of the design, I also created an extra testbench for clock signal. In Figure 3 and Figure 4, both simulation results for main module and clock are shown.

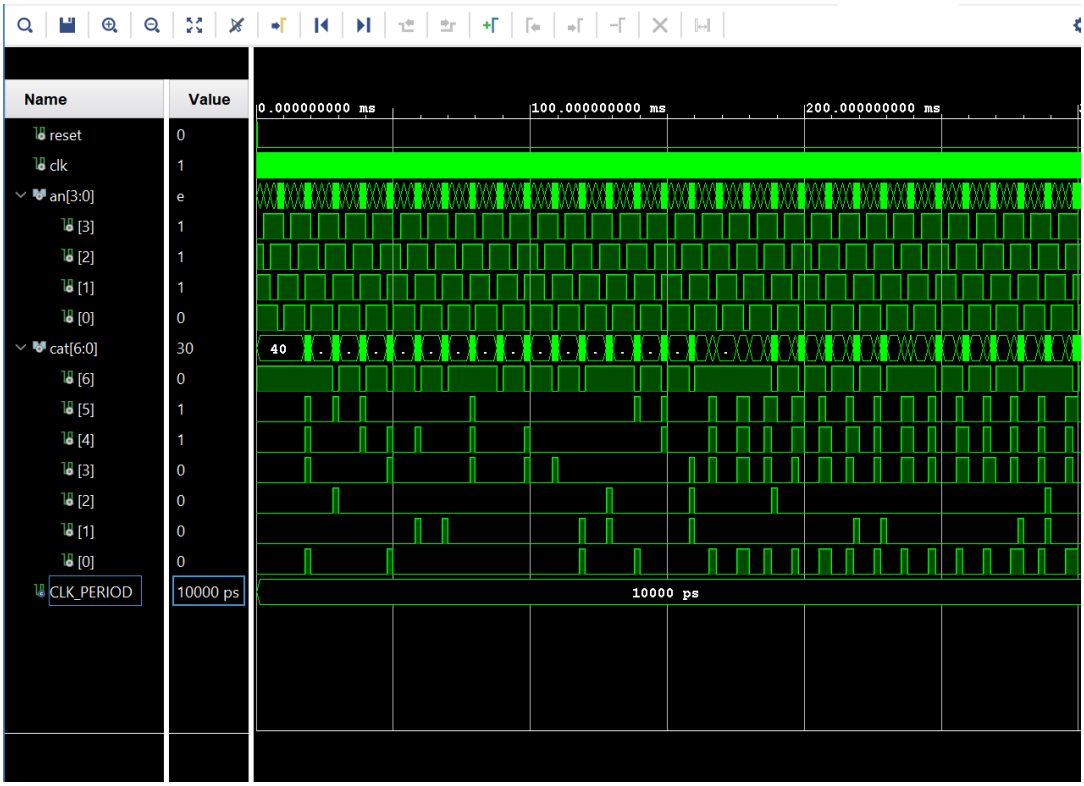

Figure 3: Simulation Results of the Main Module

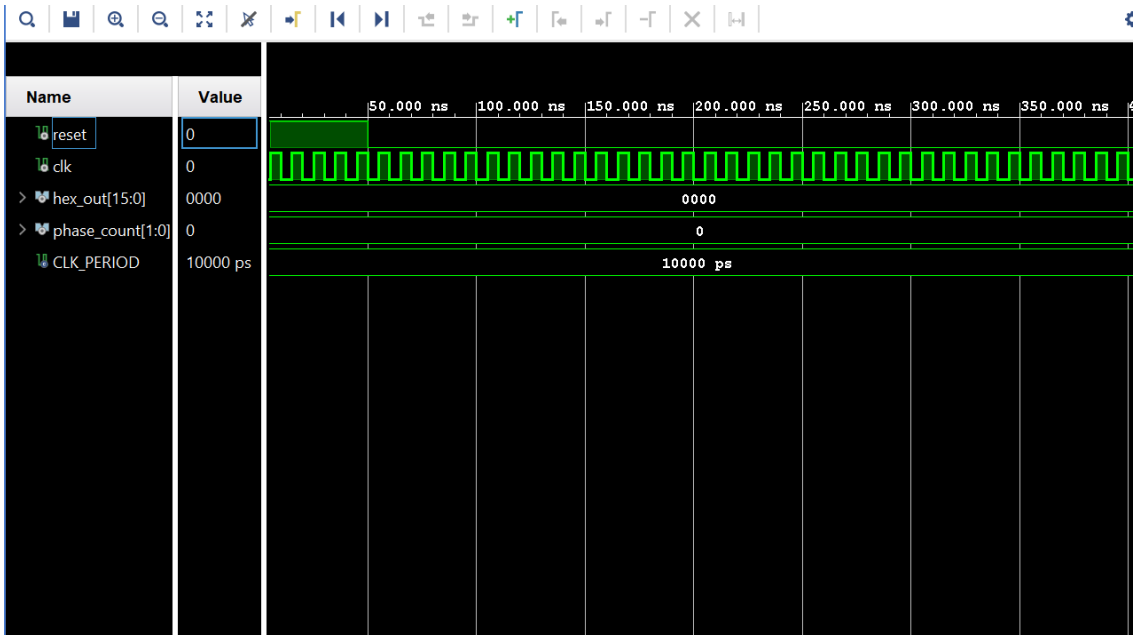

Figure 4: Simulation Results of the Clock Signal

After creating the constraint file ("constraint.xdc"), I generated the bitstream and then implemented it on Basys3 board. The results were corresponding with the simulation results and the seven-segment display behaves as expected. Figures 5-9 shows the outputs of the seven-segment display on Basys3 board. The rightmost switch resets the system and when it is turned on the left digit displays 0. After it is turned off, the system starts counting again from 0 in hexadecimal format.

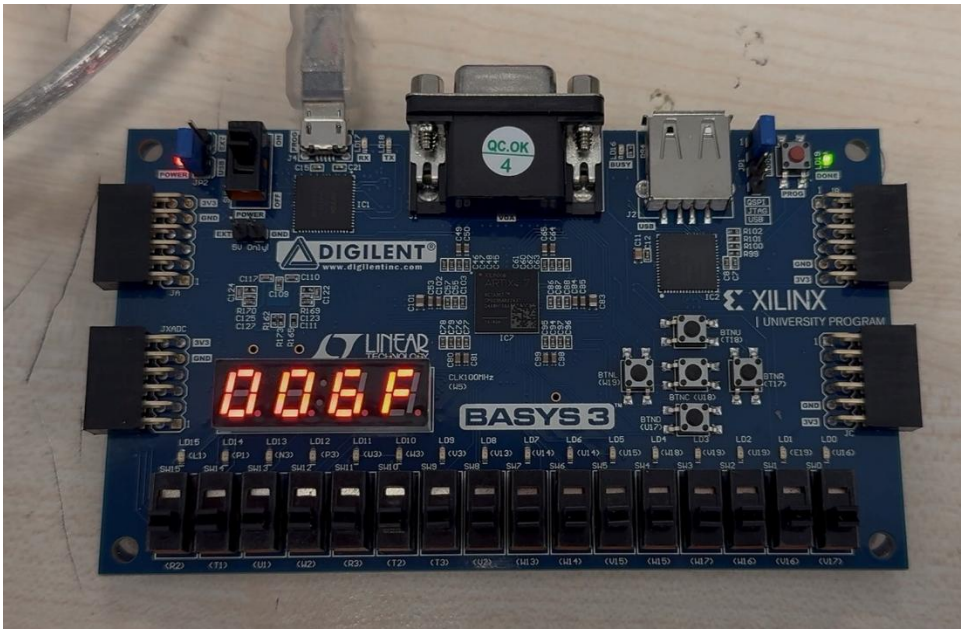

Figure 5: Output is "006F" after 111 seconds

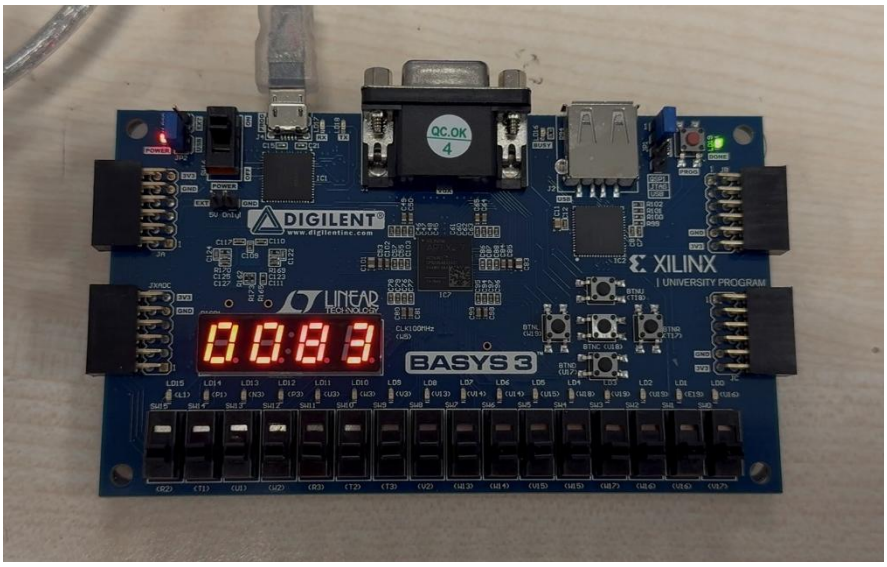

Figure 6: Output is “0083” after 131 seconds

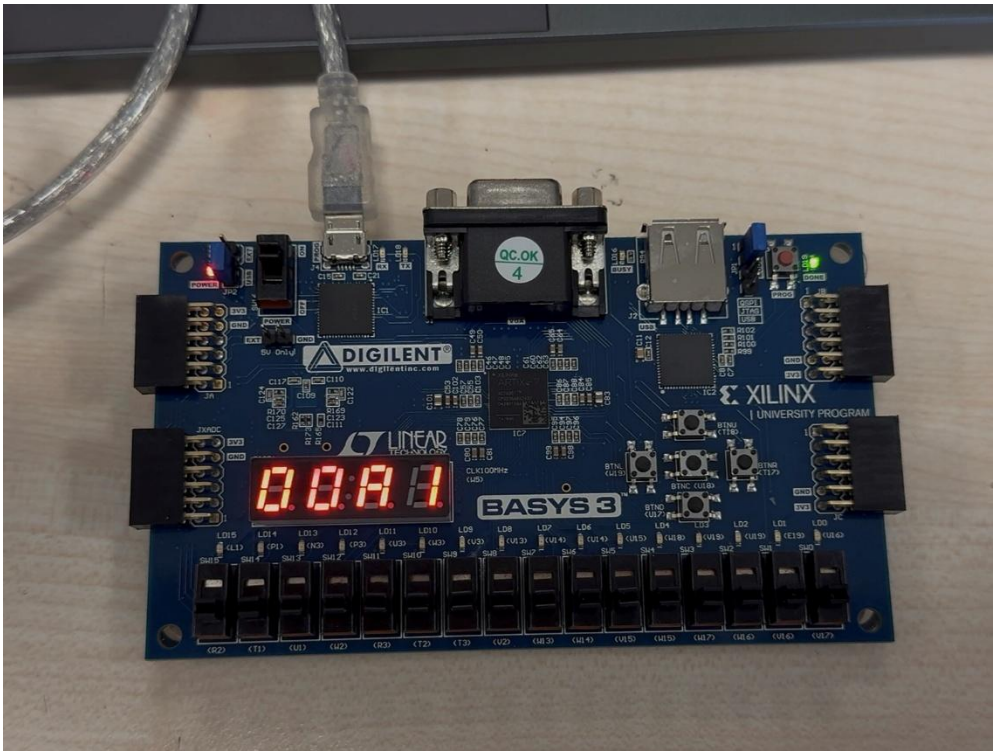

Figure 7: Output is “00A1” after 161 seconds

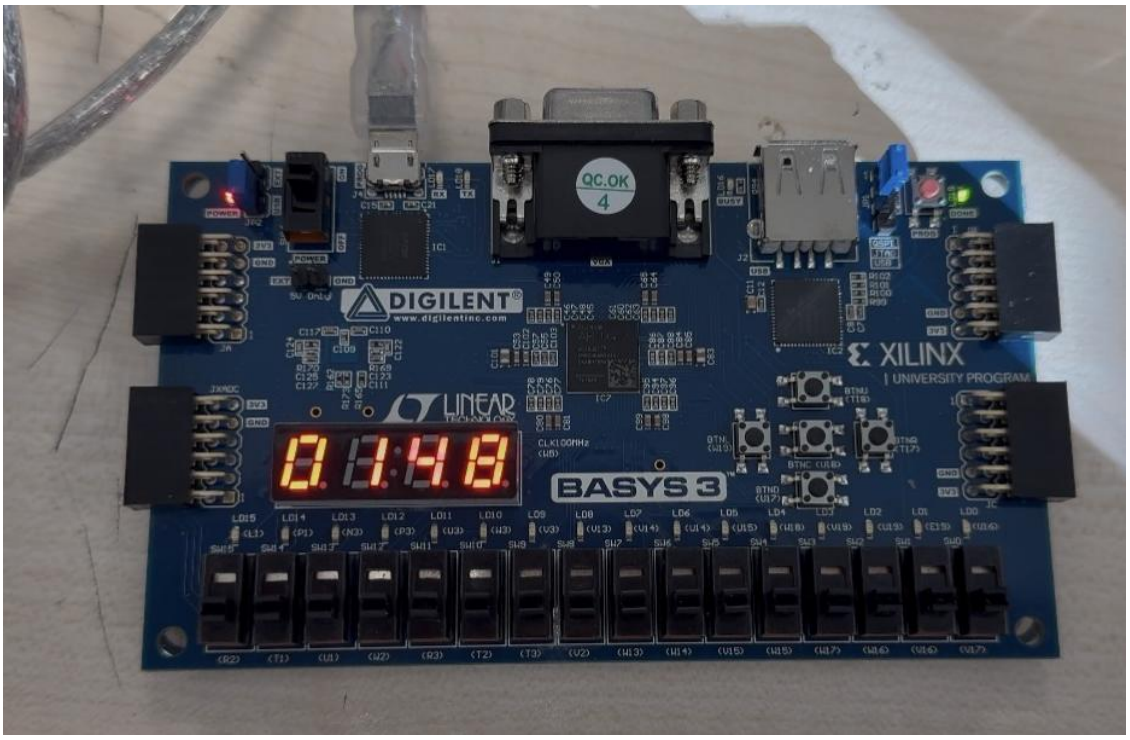

Figure 8: Output is “0148” after 328 seconds

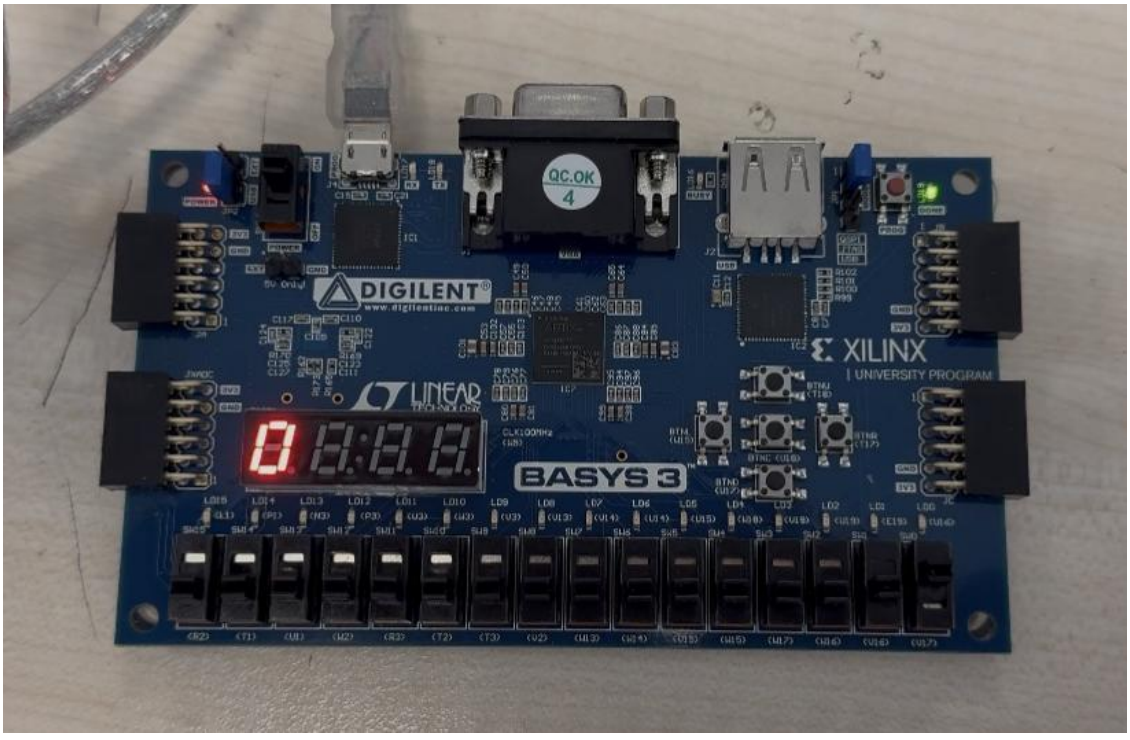

Figure 9: Output is “0” when reset switch is on

Conclusion

The purpose of the experiment was to learn how to use seven-segment display using VHDL. In this lab, I learned how the internal 100 MHz clock of the Basys3 board works and how it can be divided to create slower clock signals. After writing the codes and implementing it the results were as expected and the display was correct. However, while writing the codes, I encountered many bugs and errors that took a lot of time to fix. Also, I made a lot of research to learn how seven-segment display and clock signal works. I also understood how the persistence of vision effect allows the display to appear as all digits are shown at the same time.

Appendices

top_seven_segment.vhd

```
library ieee;
use ieee.std_logic_1164.all;
entity main is
   port (
      clk     : in  std_logic;
      reset   : in  std_logic;

      an      : out std_logic_vector(3 downto 0);
      cat     : out std_logic_vector(6 downto 0)
   );
end entity main;

architecture structural of main is

   signal hex_val      : std_logic_vector(15 downto 0);
   signal phase_sig    : std_logic_vector(1 downto 0);
   signal hex_to_dec   : std_logic_vector(3 downto 0);

   component hex_sec_counter
      port (
         clk            : in  std_logic;
```

```
        reset          : in  std_logic;
        hex_out        : out std_logic_vector(15 downto 0);
        phase_count    : out std_logic_vector(1 downto 0)
    );
  end component;

  component segment_driver
    port (
        phase_count    : in  std_logic_vector(1 downto 0);
        num_in         : in  std_logic_vector(15 downto 0);
        an_out         : out std_logic_vector(3 downto 0);
        hex_out_to_dec : out std_logic_vector(3 downto 0)
    );
  end component;

  component hex_decoder
    port (
        hex_in     : in  std_logic_vector(3 downto 0);
        cat_out    : out std_logic_vector(6 downto 0)
    );
  end component;

begin

  U1_Counter : hex_sec_counter
  port map (
    clk        => clk,
    reset      => reset,
    hex_out    => hex_val,
    phase_count => phase_sig
```

```
    );

    U2_Driver : segment_driver
    port map (
        phase_count     => phase_sig,
        num_in          => hex_val,
        an_out          => an,
        hex_out_to_dec  => hex_to_dec
    );

    U3_Decoder : hex_decoder
    port map (
        hex_in  => hex_to_dec,
        cat_out => cat
    );

end architecture structural;
```

hex_sec_counter.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity hex_sec_counter is
    port (
        clk             : in  std_logic;
        reset           : in  std_logic;

        hex_out         : out std_logic_vector(15 downto 0);
```

```
        phase_count     : out std_logic_vector(1 downto 0)
    );
end entity hex_sec_counter;

architecture behavioral of hex_sec_counter is

    constant SEC_LIMIT : natural := 99999999;

    signal clk_count    : unsigned(26 downto 0) := (others => '0');
    signal seconds      : unsigned(15 downto 0) := (others => '0');

begin

    process(clk, reset)
    begin
        if reset = '1' then
            clk_count <= (others => '0');
            seconds   <= (others => '0');

        elsif rising_edge(clk) then

            if clk_count = to_unsigned(SEC_LIMIT, clk_count'length) then

                clk_count <= (others => '0');
                seconds <= seconds + 1;

            else
                clk_count <= clk_count + 1;
            end if;
        end if;
```

```
    end process;

    hex_out <= std_logic_vector(seconds);

    phase_count <= std_logic_vector(clk_count(19 downto 18));

end architecture behavioral;
```

segment_driver.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity segment_driver is
    port (

        phase_count     : in  std_logic_vector(1 downto 0);
        num_in          : in  std_logic_vector(15 downto 0);

        an_out          : out std_logic_vector(3 downto 0);
        hex_out_to_dec  : out std_logic_vector(3 downto 0)
    );
end entity segment_driver;

architecture behavioral of segment_driver is
begin

    process(phase_count, num_in)
    begin
        case phase_count is
```

```
            when "00" =>
                an_out <= "0111";
                hex_out_to_dec <= num_in(15 downto 12);

            when "01" =>
                an_out <= "1011";
                hex_out_to_dec <= num_in(11 downto 8);

            when "10" =>
                an_out <= "1101";
                hex_out_to_dec <= num_in(7 downto 4);

            when "11" =>
                an_out <= "1110";
                hex_out_to_dec <= num_in(3 downto 0);

            when others =>
                an_out <= "1111";
                hex_out_to_dec <= (others => '0');
        end case;
    end process;

end architecture behavioral;
```

hex_decoder.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity hex_decoder is
```

```
    port (

        hex_in      : in  std_logic_vector(3 downto 0);
        cat_out     : out std_logic_vector(6 downto 0)
    );
end entity hex_decoder;

architecture behavioral of hex_decoder is

begin

    process(hex_in)
    begin
        case hex_in is
            when "0000" => cat_out <= "1000000"; -- 0
            when "0001" => cat_out <= "1111001"; -- 1
            when "0010" => cat_out <= "0100100"; -- 2
            when "0011" => cat_out <= "0110000"; -- 3
            when "0100" => cat_out <= "0011001"; -- 4
            when "0101" => cat_out <= "0010010"; -- 5
            when "0110" => cat_out <= "0000010"; -- 6
            when "0111" => cat_out <= "1111000"; -- 7
            when "1000" => cat_out <= "0000000"; -- 8
            when "1001" => cat_out <= "0010000"; -- 9


            when "1010" => cat_out <= "0001000"; -- A
            when "1011" => cat_out <= "0000011"; -- b
            when "1100" => cat_out <= "1000110"; -- C
            when "1101" => cat_out <= "0100001"; -- d
```

```
            when "1110" => cat_out <= "0000110"; -- E
            when "1111" => cat_out <= "0001110"; -- F
            when others => cat_out <= "1111111"; -- All segments are closed

        end case;
    end process;

end architecture behavioral;
```

tb_main.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity tb_main is
end tb_main;

architecture Behavioral of tb_main is

    constant CLK_PERIOD : time := 10 ns;

    signal reset : std_logic;
    signal clk   : std_logic;



    signal an    : std_logic_vector(3 downto 0);
    signal cat   : std_logic_vector(6 downto 0);

begin
```

```
dut: entity work.main(structural)
port map (
    clk   => clk,
    reset => reset,
    an    => an,
    cat   => cat
);



clock_process : process
begin
    clk <= '0';
    wait for CLK_PERIOD / 2;
    clk <= '1';
    wait for CLK_PERIOD / 2;
end process;

sim_process : process
begin

    reset <= '1';
    wait for 30 ns;

    reset <= '0';
    wait for 100 ns;
```

```
        wait for 100 ms;


        wait;
    end process;
end Behavioral;
```

tb_counter.vhd

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

entity tb_counter is
end tb_counter;

architecture Behavioral of tb_counter is

    constant CLK_PERIOD : time := 10 ns;

    signal reset       : std_logic;
    signal clk         : std_logic;
    signal hex_out     : std_logic_vector(15 downto 0);
    signal phase_count : std_logic_vector(1 downto 0);

begin

    dut: entity work.hex_sec_counter(behavioral)
    port map (
```

```
        clk         => clk,
        reset       => reset,
        hex_out     => hex_out,
        phase_count => phase_count
    );

    clock_process : process
    begin
        clk <= '0';
        wait for CLK_PERIOD / 2;
        clk <= '1';
        wait for CLK_PERIOD / 2;
    end process;

    sim_process : process
    begin
        reset <= '1';
        wait for 50 ns;
        reset <= '0';
        wait for 50 ns;
        wait for 150 ms;
        wait;
    end process;

end Behavioral;
```

References

https://github.com/dmadison/LED-Segment-ASCII?tab=readme-ov-file

basys3xdc.pdf

https://www.instructables.com/Seven-Segment-Display-Tutorial/

https://digilent.com/reference/programmable-logic/basys-3/reference-manual?srsltid=AfmBOooRAEIoAILwo0jWlWdzkXOxGtACzubry4eQmLKAEApm7FaXUY5B