Ada Zağyapan

EEE102-02

26 March 2025

Lab-5: Seven-Segment Display

PURPOSE

The goal of this lab is to create and test a seven-segment display driver on the BASYS3 FPGA board using VHDL. Using a clock divider, the board's 100MHz internal clock is converted into a slower one and a hexadecimal counter that updates once every second is displayed on the seven-segment display.

METHODOLOGY

Initially, a clock divider (*clk_divider.vhd*) was created to convert the board's fast 100 MHz clock into a slower timing signal to make it suitable for human observation. A counter was implemented to increment with each rising edge of the clock. When a predefined value corresponding to one second was reached, the counter was reset to zero, and a separate seconds counter was incremented. Additionally, specific bits from this counter were used to produce a phase signal, which later controlled the multiplexing of the display.

Then, a display multiplex driver (*disp_mux_driver.vhd*) was developed. This module sequentially activated each of the four digits using the phase signal provided by the clock divider. For each digit, a 4-bit segment was extracted from the 16-bit seconds counter, and the

appropriate anode signal was assigned to ensure that the correct numeral appeared at each digit position.

Next, a seven-segment decoder (*seg_decoder.vhd*) was implemented. This component translated a 4-bit input into a 7-bit output pattern, mapping each hexadecimal digit to its corresponding pattern on the LED segments of the display.

Finally, all modules were integrated into a complete top-level design (*display_top.vhd*), they were tested using a testbench simulation, and an appropriate constraint file was created to map signals onto the pins of the BASYS3 board. The complete design was synthesized, programmed onto the board, and successfully demonstrated to function as intended.

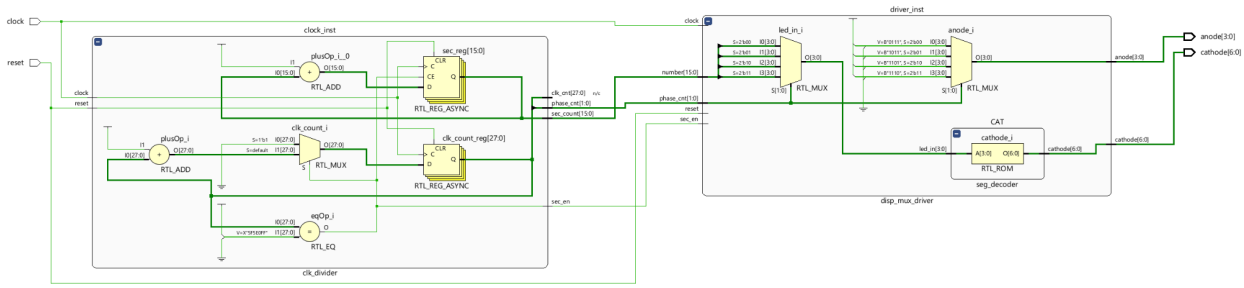

Figure 1: RTL Elaborated design schematic of *display_top.vhd*

DESIGN SPECIFICATIONS

clk_divider.vhd: In the clock divider module, the inputs are the high-frequency clock and a reset signal. Internally, there is a 28-bit signal called clk_count that increments on every rising edge of the clock, serving as the primary counter. When reset is asserted, clk_count is cleared to all zeros. Alongside clk_count, there is a 16-bit signal named sec which holds the seconds count which is also cleared to zero when reset pin is pushed. Every time clk_count reaches the

hexadecimal value x"5F5E0FF"—which corresponds to one second—the code increments sec by one and resets clk_count. Additionally, a single-bit output, sec_en, is driven high when clk_count reaches that same value, providing an enable signal that indicates the completion of a one-second period. A two-bit output called phase_cnt is derived from bits 19 and 18 of clk_count; these bits are used to create a phase signal that maintains the rapid cycling of the display digits for multiplexing purposes and to exploit persistence of vision, which gives the illusion to the observer that all digits are lit continuously. (Figure 2).

disp_mux_driver.vhd: In the display multiplexer driver, the design takes in the phase_cnt signal along with a 16-bit number (which is the seconds count from the clock divider) and a reset input. This module uses a local signal called led_in—a 4-bit vector—to temporarily hold the nibble corresponding to the digit that should currently be displayed. The process within this module uses a case statement on phase_cnt to select which digit of the 16-bit number to display: when phase_cnt is "00", it selects the most significant nibble (bits 15 downto 12) and drives the anode output with the pattern "0111"; when phase_cnt is "01", it selects the next nibble (bits 11 downto 8) and sets the anode to "1011"; when phase_cnt is "10", it takes bits 7 downto 4 and uses "1101" for the anode; and when phase_cnt is "11", it uses the least significant nibble (bits 3 downto 0) with an anode pattern "1110". The rapid switching among these selections creates the illusion that all digits are lit continuously. The module then instantiates the seven-segment decoder (referred to as CAT) by passing the led_in signal to it, so that the proper cathode pattern is generated for the active digit (Figure 3).

seg_decoder.vhd: The seven-segment decoder module accepts a 4-bit input called led_in and outputs a corresponding 7-bit pattern on the cathode output based on its value. This mapping

is implemented via a case statement where each possible 4-bit value (from "0000" to "1111") is associated with a specific 7-bit pattern that lights up the appropriate segments on the display to represent hexadecimal digits (0 to F). If an unexpected value is encountered, the default case ensures that all segments are turned off by setting the output to "1111111" (Figure 4).

display_top.vhd: The top-level module, named display_top, brings together the clock divider and the display multiplexer driver. It instantiates the clock divider to provide the seconds count, the phase signal for multiplexing, and the sec_en flag. This seconds count is then fed into the multiplexer driver, which uses the phase signal to select the correct 4-bit segment for display and the sec_en signal for synchronization. The final outputs of display_top are the anode and cathode signals that directly drive the seven-segment display.

simulation_top.vhd: In the testbench module, working at 450ms, *display_top* module is instantiated and a simple clock generation process toggles the clock, while a separate process initializes and subsequently deasserts the reset signal to start the system.

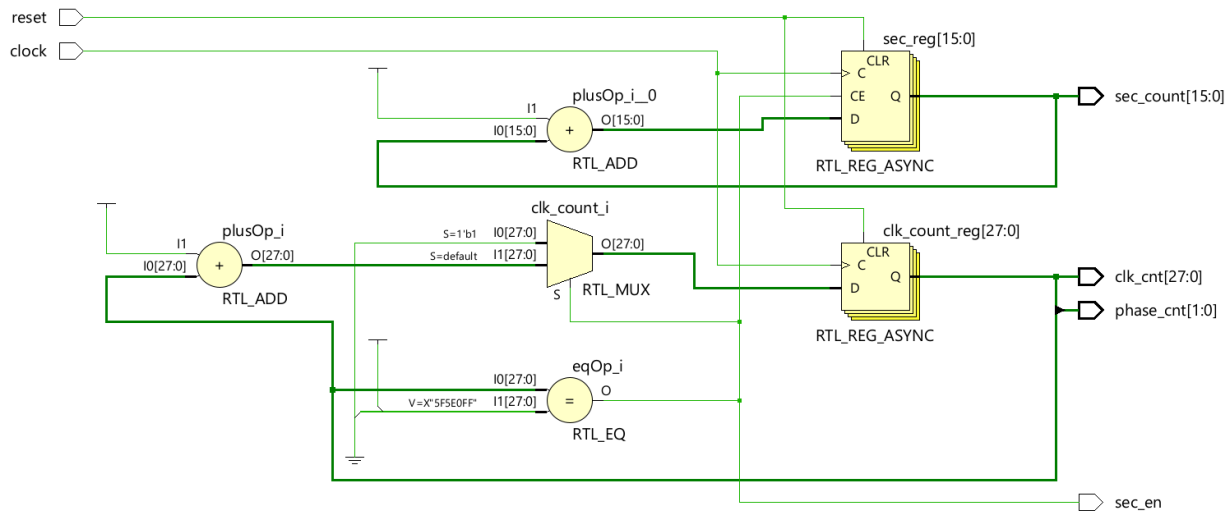

Figure 2: RTL schematic of *clk_divider.vhd*

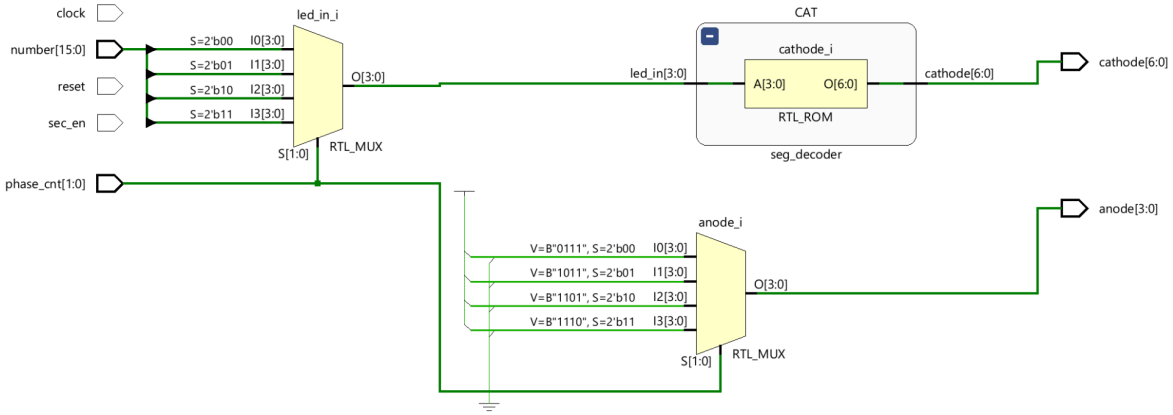

Figure 3: RTL schematic of *disp_mux_driver.vhd*

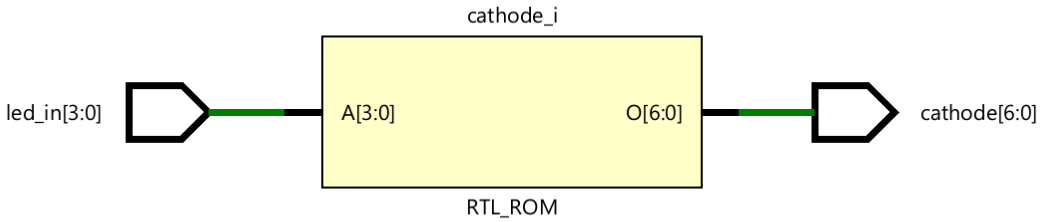

Figure 4: RTL schematic of *seg_decoder.vhd*

QUESTIONS

The internal clock frequency of the BASYS3 board is 100 MHz. To derive a slower clock signal from this high-frequency source, a clock divider is used. This clock divider means implementing a counter that increments with each 100 MHz clock cycle. When the counter reaches a predetermined value that represents the desired slower period—for example, 100 million counts for a 1-second period—the counter resets, and a slower clock signal is generated by toggling an output or asserting an enable signal.

However, any arbitrary frequency lower than 100 MHz cannot be generated simply by clock division. Because the divider relies on an integer count, the resulting slower clock frequency

must be an exact division of 100 MHz by an integer. Hence, only frequencies that are 100 MHz divided by an integer value can be created. For instance, 50 MHz (100 MHz/2), 25 MHz (100 MHz/4), or 1 Hz (100 MHz/100,000,000) can be obtained, but a frequency like 33 MHz cannot be generated because that would require division by a non-integer value.

RESULTS

The testbench simulation results confirmed that the clock divider and display multiplexer function as intended. In the simulation waveforms, the 28-bit counter (clk_count) steadily increments on each rising edge of the 100 MHz clock and resets to zero when it reaches the value x"5F5E0FF". At this reset point, the 16-bit seconds counter (sec) increments by one and the sec_en signal is briefly asserted, which verifies the one-second timing. Additionally, the phase_cnt signal—extracted from bits 19 and 18 of clk_count—cycles through its four states ("00", "01", "10", "11"), ensuring that the multiplexer sequentially selects each digit of the seconds count (Figures 5-6).

On the BASYS3 board, the outputs reflected the same behavior observed in simulation. The seven-segment display shows an incrementing hexadecimal number, with each digit being updated in rapid succession due to the persistence of vision. For example, after a few seconds, the display might show values like "0013" or "002D", indicating that the seconds counter is incrementing correctly. The rapid cycling of the anode signals (driven by the phase_cnt) in conjunction with the proper cathode patterns from the decoder creates the illusion that all digits are continuously lit, confirming that both the timing and display multiplexing are working as expected.

Figures 5-6: Simulation results for *top_simulation.vhd* at 20 seconds

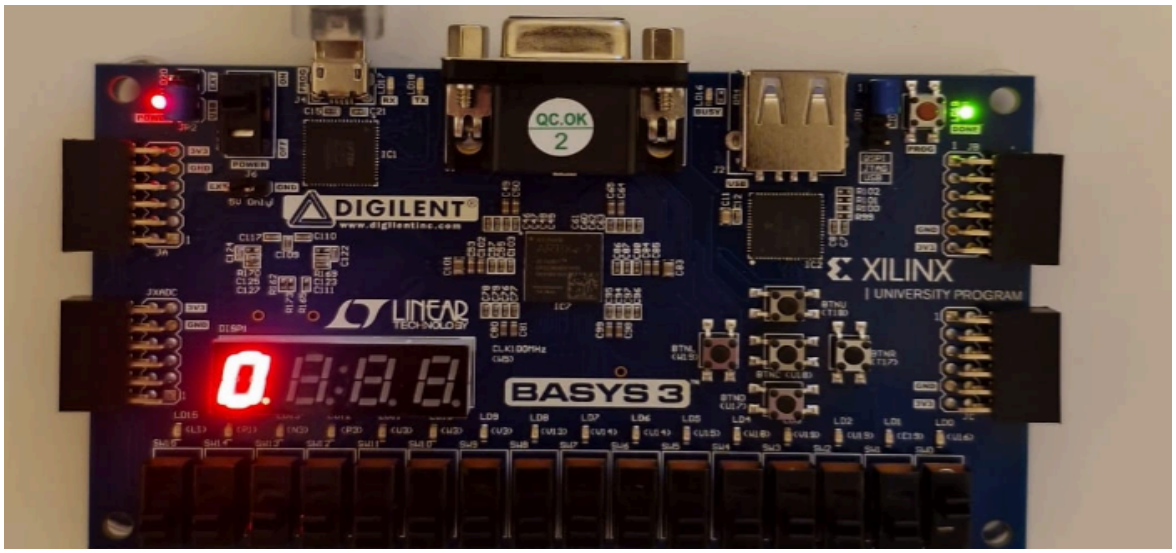

Figure 7: When reset switch is on

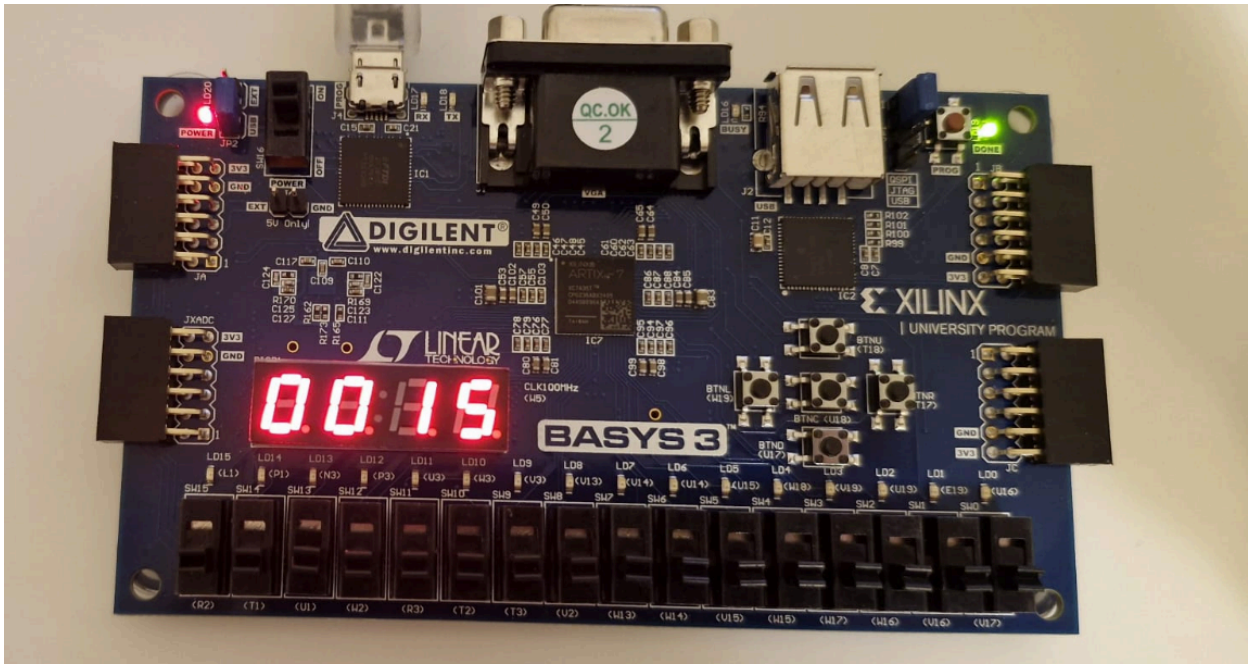

Figure 8: After 21 seconds, “0015” in hexadecimal

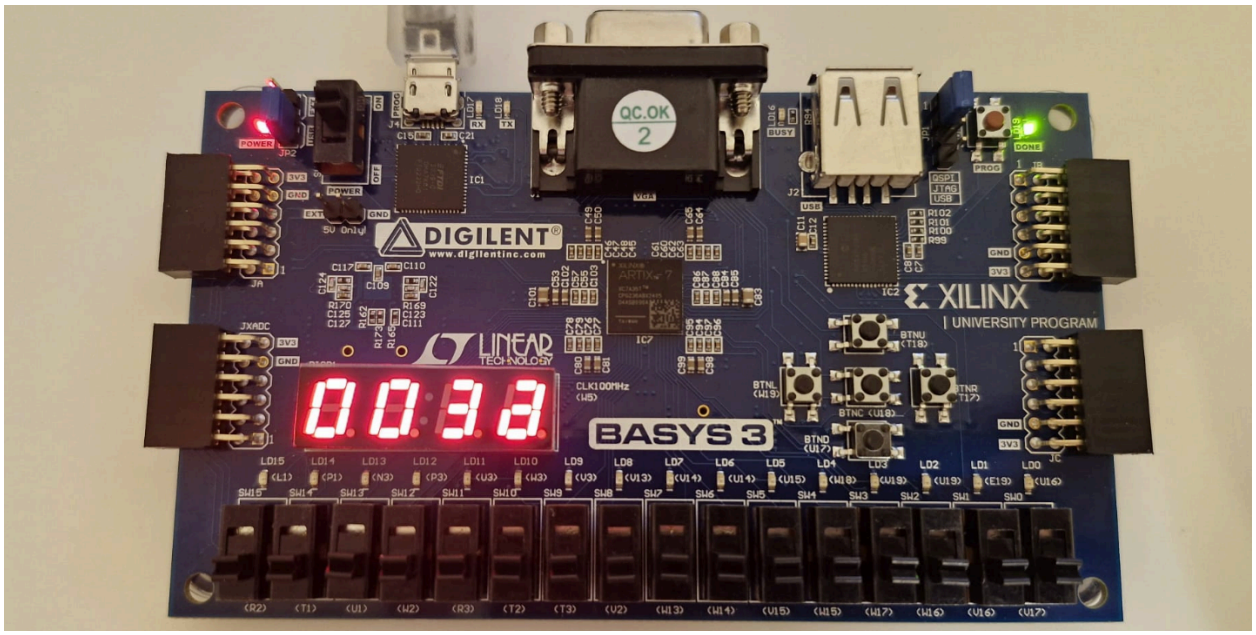

Figure 9: After 58 seconds, “003a” in hexadecimal

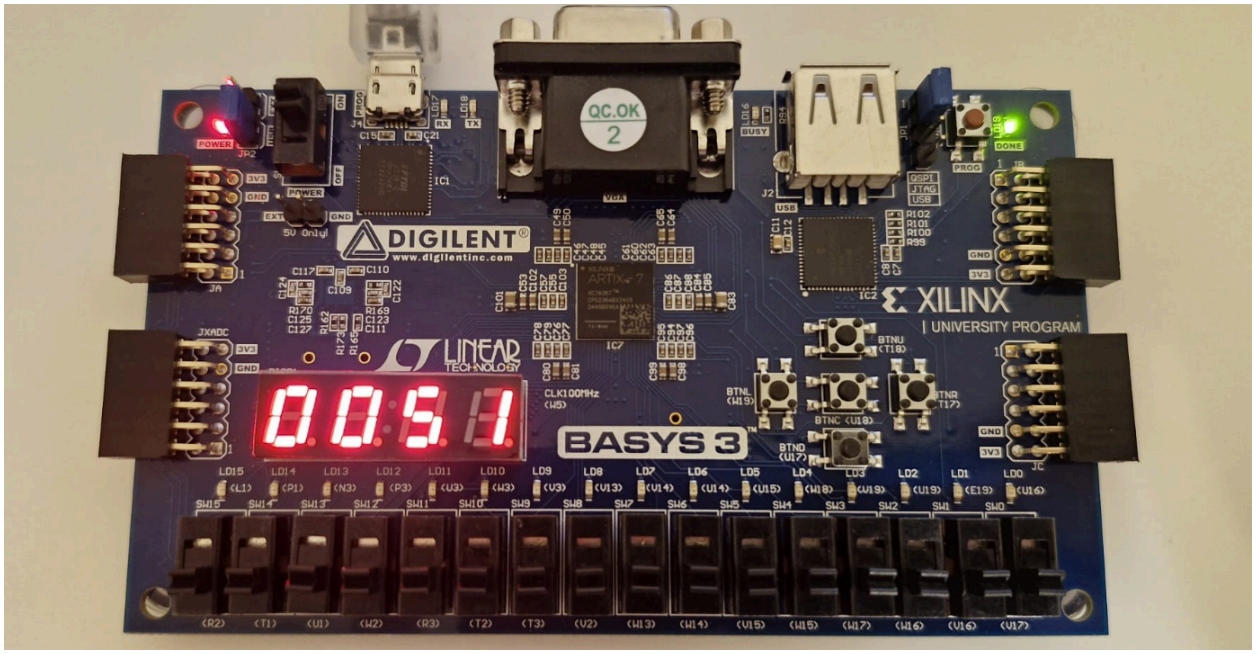

Figure 10: After 81 seconds, “0051” in hexadecimal

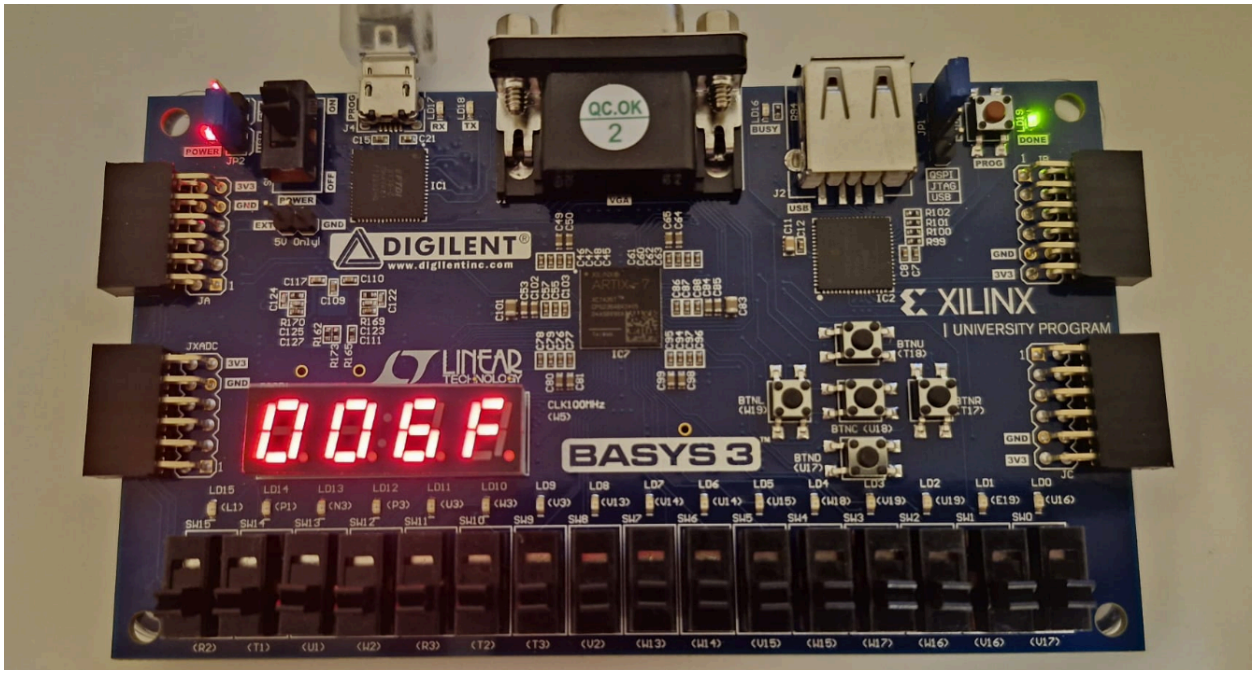

Figure 11: After 111 seconds, “006f” in hexadecimal

CONCLUSION

In conclusion, the primary goal of this experiment was to implement a seven-segment display using VHDL on the BASYS3 board. The process began with a thorough understanding of the board's seven-segment display and its associated hardware connections. After developing VHDL modules for the clock divider, display multiplexing, and segment decoding, these components were integrated to the implementation of the intended design. The resulting implementation functioned as expected: the display verified correct division of the internal 100 MHz clock and presented an incrementing hexadecimal counter.

APPENDIX

clk_divider.vhd:

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


entity clk_divider is
    Port (
        clock         : in  STD_LOGIC;
        reset       : in  STD_LOGIC;
        clk_cnt : out STD_LOGIC_VECTOR (27 downto 0);
        phase_cnt : out STD_LOGIC_VECTOR (1 downto 0);
        sec_count : out STD_LOGIC_VECTOR (15 downto 0);
        sec_en      : out STD_LOGIC
    );
end clk_divider;

architecture clk of clk_divider is
    signal clk_count : STD_LOGIC_VECTOR(27 downto 0);
    signal sec   : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
begin
    process(clock)
    begin
        if (reset = '1') then
            clk_count <= (others => '0');
            sec <= (others => '0');
        else
            if rising_edge(clock) then  -- changed from CLK to clk
                clk_count <= clk_count + '1';
                if clk_count = x"5F5E0FF" then
                    sec   <= sec + '1';
                    clk_count <= (others => '0');
                end if;
            end if;
```

```
        end if;
    end process;

    sec_en <= '1' when clk_count = x"5F5E0FF" else '0';
    phase_cnt <= clk_count(19 downto 18);
    clk_cnt <= clk_count;
    sec_count <= sec;
end clk;
```

disp_mux_driver.vhd:

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


entity disp_mux_driver is
    Port (
        clock         : in  STD_LOGIC;
        reset       : in  STD_LOGIC;
        phase_cnt : 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 disp_mux_driver;

architecture mux_arch of disp_mux_driver is
    signal led_in: STD_LOGIC_VECTOR(3 downto 0);
begin
    process(phase_cnt)
    begin
        case phase_cnt is
            when "00" =>
                anode <= "0111";
```

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

    CAT: entity work.seg_decoder(dec_arch)
    port map(
        led_in => led_in,
        cathode    => cathode
    );
end mux_arch;
```

seg_decoder.vhd:

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


entity seg_decoder is
    Port (
        led_in : in  STD_LOGIC_VECTOR (3 downto 0);
        cathode    : out STD_LOGIC_VECTOR (6 downto 0)
    );
end seg_decoder;

architecture dec_arch of seg_decoder is
begin
    process(led_in)
```

```
    begin
        case led_in 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 dec_arch;
```

display_top.vhd:

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

entity display_top is
    Port (
        clock   : in  STD_LOGIC;
        reset : in  STD_LOGIC;
        anode    : out STD_LOGIC_VECTOR (3 downto 0);
        cathode   : out STD_LOGIC_VECTOR (6 downto 0)
    );
end display_top;

architecture Behavioral of display_top is
    signal phase_cnt : STD_LOGIC_VECTOR (1 downto 0);
```

```
    signal sec_en      : STD_LOGIC;
    signal number        : STD_LOGIC_VECTOR (15 downto 0);
begin
    clock_inst: entity work.clk_divider
        port map (
            clock         => clock,
            reset       => reset,
            phase_cnt => phase_cnt,
            sec_en      => sec_en,
            sec_count   => number
        );

    driver_inst: entity work.disp_mux_driver
        port map (
            clock => clock,
            reset => reset,
            phase_cnt => phase_cnt,
            number => number,
            sec_en => sec_en,
            anode => anode,
            cathode => cathode
        );
end Behavioral;
```

top_simulation.vhd:

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

entity top_simulation is
end top_simulation;

architecture Behavioral of top_simulation is
    signal reset : std_logic;
    signal clock   : std_logic;
    signal anode    : std_logic_vector(3 downto 0);
    signal cathode   : std_logic_vector(6 downto 0);
begin
    dut: entity work.display_top(Behavioral)
```

```
        port map (
            clock   => clock,
            reset => reset,
            anode    => anode,
            cathode   => cathode
        );

    clk: process
    begin
        clock <= '0';
        wait for 10 ns;
        clock <= '1';
        wait for 10 ns;
    end process;

    sim: process
    begin
        reset <= '1';
        wait for 20 ns;
        reset <= '0';
        wait;
    end process;
end Behavioral;
```

cons.xdc:

```
set_property PACKAGE_PIN W5 [get_ports clock]
set_property IOSTANDARD LVCMOS33 [get_ports clock]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports clock]

set_property PACKAGE_PIN V17 [get_ports {reset}]
set_property IOSTANDARD LVCMOS33 [get_ports {reset}]

set_property PACKAGE_PIN W7 [get_ports {cathode[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathode[6]}]

set_property PACKAGE_PIN W6 [get_ports {cathode[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathode[5]}]
```

```
set_property PACKAGE_PIN U8 [get_ports {cathode[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathode[4]}]

set_property PACKAGE_PIN V8 [get_ports {cathode[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathode[3]}]

set_property PACKAGE_PIN U5 [get_ports {cathode[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathode[2]}]

set_property PACKAGE_PIN V5 [get_ports {cathode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathode[1]}]

set_property PACKAGE_PIN U7 [get_ports {cathode[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathode[0]}]

set_property PACKAGE_PIN U2 [get_ports {anode[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[0]}]

set_property PACKAGE_PIN U4 [get_ports {anode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[1]}]

set_property PACKAGE_PIN V4 [get_ports {anode[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[2]}]

set_property PACKAGE_PIN W4 [get_ports {anode[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[3]}]
```

Works Cited

Seven Segment Displays - GeeksforGeeks. (2014, August 16). https://www.geeksforgeeks.org/seven-segment-displays/

Eskişehir Technical University. (n.d.). LAB 2-INTRODUCTION TO VHDL AND FPGA HARDWARE IMPLEMENTATION DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING EEM 334-Digital Systems II. Retrieved March 26, 2025, from www.xilinx.com