May 16th 2025

Alp Efe Kılıçarslan, 22402390

EEE 102-2

Video Link: https://www.youtube.com/watch?v=hih20TZsonk

Term Project: Analog-to-Digital Voltage Analyzer

Purpose:

The purpose of this term project was to measure and display voltage in a multitude of ways using a BASYS 3 FPGA, a breadboard, a potentiometer, buzzers and jumper cables.

Design Specifications:

In order to design the project, firstly the parameters set in the revised project proposal were looked at. Using the document; a seven-segment display which shows the read voltage, a LED bar system and a buzzer control for significant changes in voltage were decided upon to be created. The software was written using Vivado in VHDL as follows:

- xadc_wiz_0.xci: This was used as the main IP core element to measure voltage. The xADC, in this specific configuration, works by comparing the voltage between two ports and outputting the data as a 16-bit logic vector whose first twelve bits are the significant ones. The module was configured to use the VAUXP14 VAUXN14 terminals. This specific config was picked by testing the project.

- LED_Controller.vhd: This module controlled the onboard BASYS 3 LEDs. The module takes in the data outputted by the xADC wizard and converts it to an integer. The integer is multiplied by 1000 and divided by 4095 to get the millivolts as an easily workable format. Once the data reaches a thresholded value – which was set as 125mV per LED – an LED on the BASYS 3 FPGA lights up to indicate a rough increase in voltage.

- Display_Controller.vhd: This module was used to display the read voltage on the seven-segment display and update live. The module took in the 12-bit ADC value from the xADC wizard and converted into an integer like in the LED_Controller module. After getting the integer, each digit is taken separately by using a modulus for each respective digit, converted back to standard logic vectors and assigned to four bits per digit for a 16-bit signal. When these values are generated, the

module assigns the required cathode bits and cycles through the required anode to show the read digit. This process happens at 4kHz, allowing each digit to show a thousand times per second. This allows for persistence of vision for each digit. Since this would normally cause ghosting and make it hard for the human eye to read the proper voltage, the input was fed through the following data slowing module to make the inputs change less frequently.

- XADC_Interface.vhd: This module takes in a 12-bit value at 100MHz and outputs the same 12-bit value at 80Hz, which prevents ghosting to make the seven-segment display function properly.

- Buzzer_Controller.vhd: This module takes in the 12-bit ADC and converts it to an integer in the same way as Display_Controller.vhd and LED_Controller.vhd. Then the value is divided by a threshold, set at 125 for this specific case, and compares the final value to the previous value. If the values are equal, it outputs a “00” vector. If the current value is less than the previous value, it outputs a “01” vector and if the opposite is true, it outputs a “10” vector. These outputs are used to toggle a buzzer for a significant change in voltage. If the output indicates that the input voltage is rising, a tone is applied to a buzzer and if it is falling, a tone is applied to another buzzer which is in series with a resistor. This allows for the system to output a high pitch noise when the voltage is increasing and a low pitch noise when the voltage is decreasing.

- main.vhd: This modue combines all previously described submodules into one top module. The main has the following inputs and outputs:

    - clk : in std_logic;
    - reset : in std_logic;
    - ADC : in std_logic_vector(1 downto 0);
    - hold_switch : in std_logic;
    - leds : out std_logic_vector(7 downto 0);
    - anodes : out std_logic_vector(3 downto 0);
    - segments : out std_logic_vector(7 downto 0);
    - buzzer : out std_logic_vector(1 downto 0)

  These are used as the final system constraints for the project. The clock was used as the default clock of the BASYS 3, which was 100MHZ, and fed to every submodule. The reset was used to test the functionality of each submodule when their respective inputs were set as zero. The ADC vector was set as the pins whose voltages were to be compared by the xADC module. The hold switch was used to freeze the process of all submodules and to hold their current values no matter the input. The outputs were used as the constraints for the physical components of the design. The anodes and segments were for the display controller, the leds were for the LED controller and the buzzer was for the buzzer controller.

The software part was completed after each module was written and implemented with the right constraint file. For the hardware part, a potentiometer was connected to a breadboard and given a constant voltage of 1 Volt at 0.5 Amps through a variable power supply and it's output was connected to the port that corresponds to the xADC's positive terminal. Additionally, the ground of the potentiometer was connected to the negative terminal of the xADC's port. After these were connected, two jumper cables were connected to the positive input of two buzzers who were also grounded. It should be noted that one of the buzzers were connected in series to a resistor with a resistance of 10 Ohms. By doing all these steps, the design was finalized.

Methodology:

Phase One) For the first part of the project, the modules that were specified in the design were written. A testbench was used to test the behavior of the submodules (main_tb.vhd). The testbench was run the results were observed. After the correct behavior was observed, the RTL schematic for the mail module was generated and recorded.

Phase Two) After the functionality of the modules were confirmed, the real-life circuit was built using the specifications in the design. Different input states were tested for the project and compared with the ideal results. Additionally, the input of the circuit was directly connected to the power supply to see is the circuit was measuring voltage correctly. The outputs of the project were recorded and analyzed.

Results:

Phase One) The RTL Schematics were looked at in both shrunken and expanded formats (Figure 1.1-1.2).

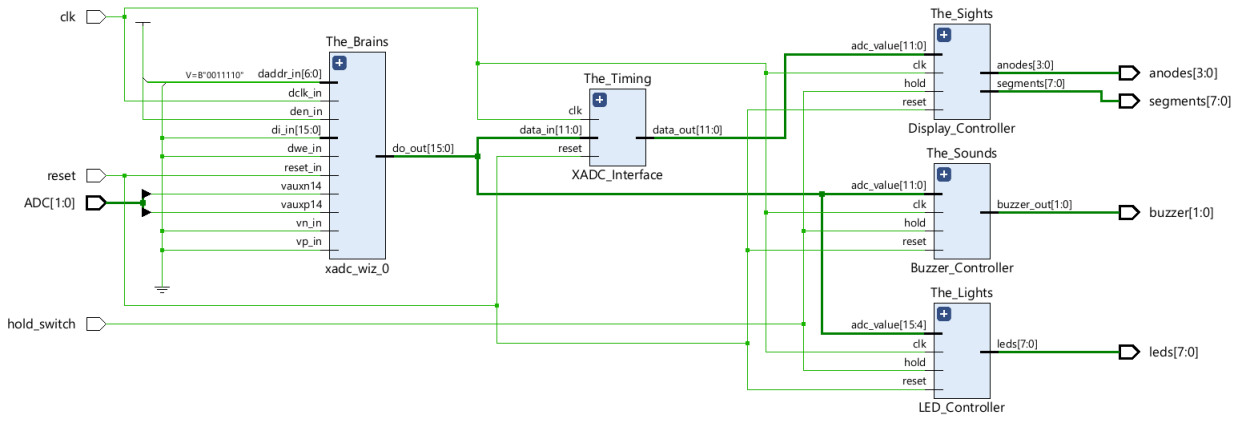

Figure 1.1: Compressed RTL Schematic of main.vhd

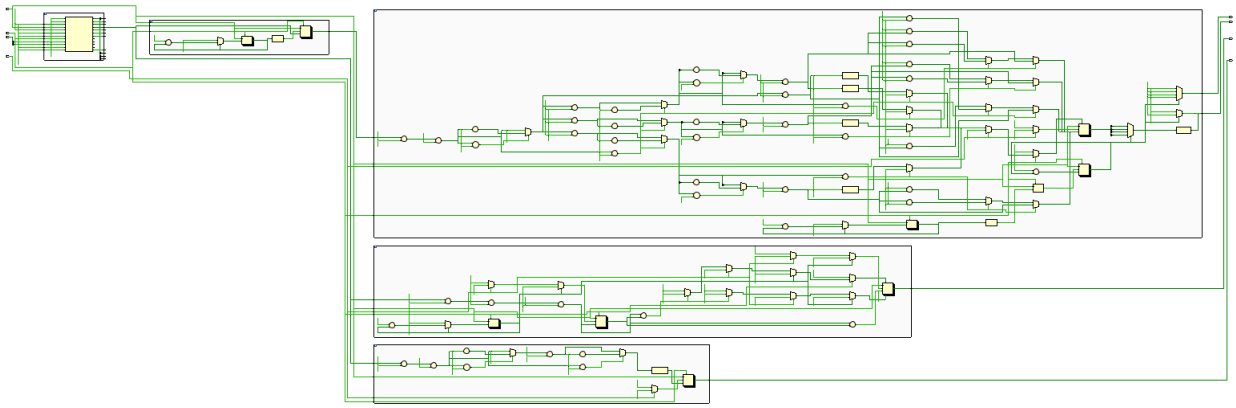

Figure 1.2: Full view of the RTL Schematic of main.vhd

After that, the testbench was ran for 100 microseconds to get all possible values of the ADC (Figure 1.3).

Figure 1.3: Full view of the simulation of main_tb.vhd

The behavior of the testbench was found to match the expected results of all modules.

Phase Two) The bitstream was generated and the board was connected to the computer who held the code. The circuit was built according to the design specifications and the program was loaded onto the BASYS 3 FPGA (Figure 2.1).

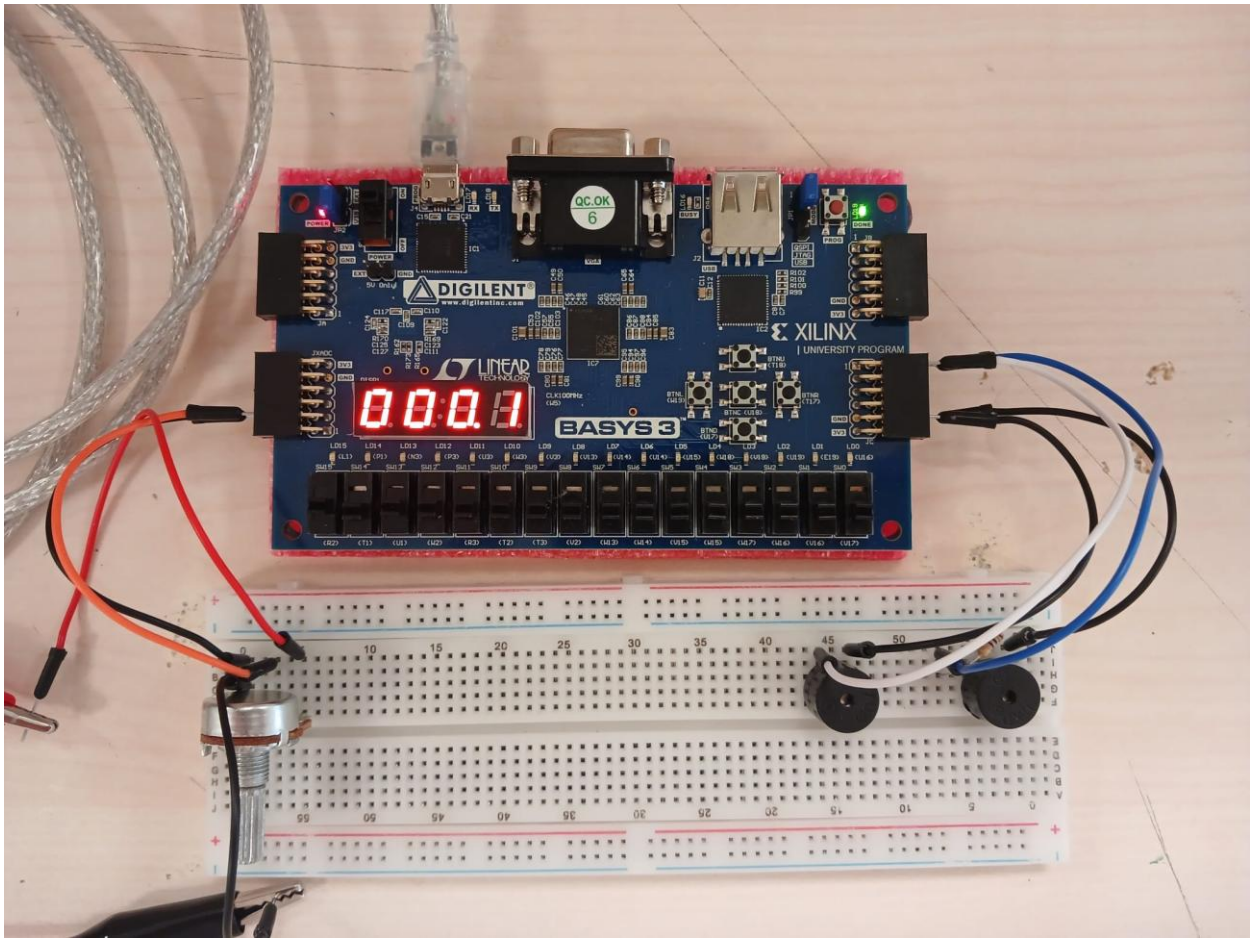

Figure 2.1: Analog-to-Digital Voltage Analyzer on startup

The power supply was turned on and the potentiometer was modulated as desired to test the mechanism (See Video). Lastly, the power supply was directly connected to the BASYS 3's xADC input ports and different voltages were given to see if the circuit was displaying a correct voltage (Figure 2.2-2.6).

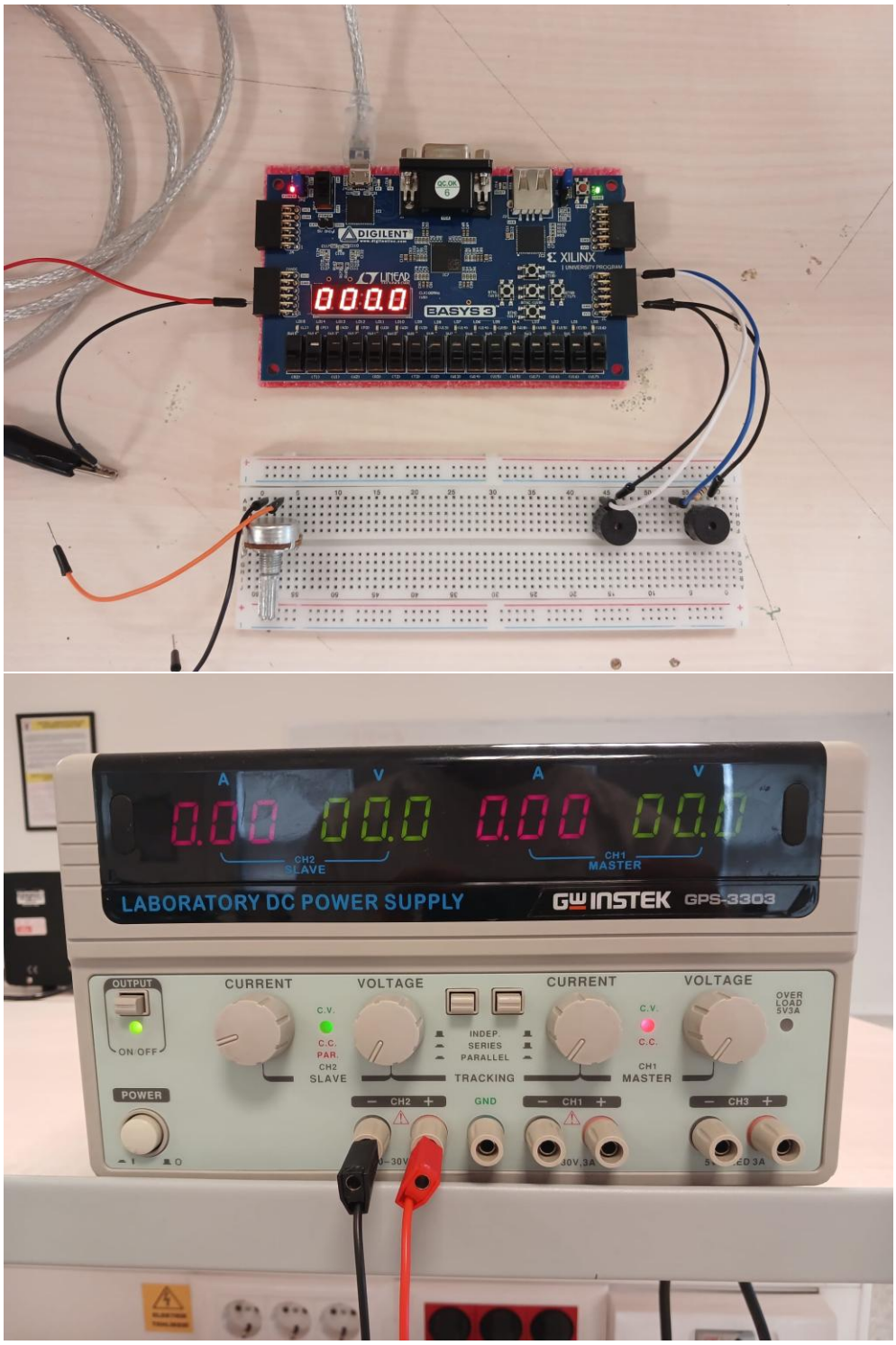

Figure 2.2: State of the design when it's receiving 0 Volts

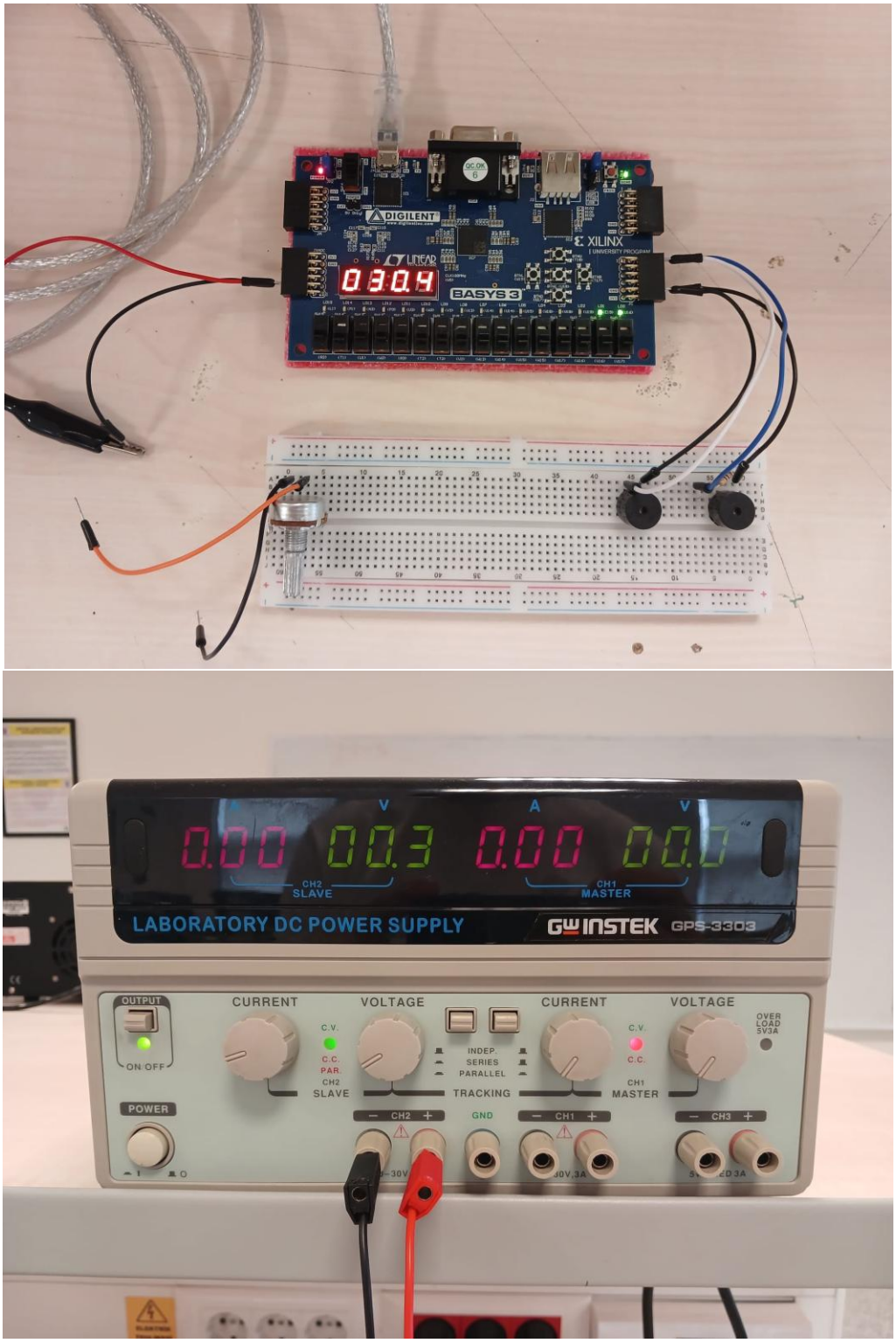

Figure 2.3: State of the design when it’s receiving 0.3 Volts

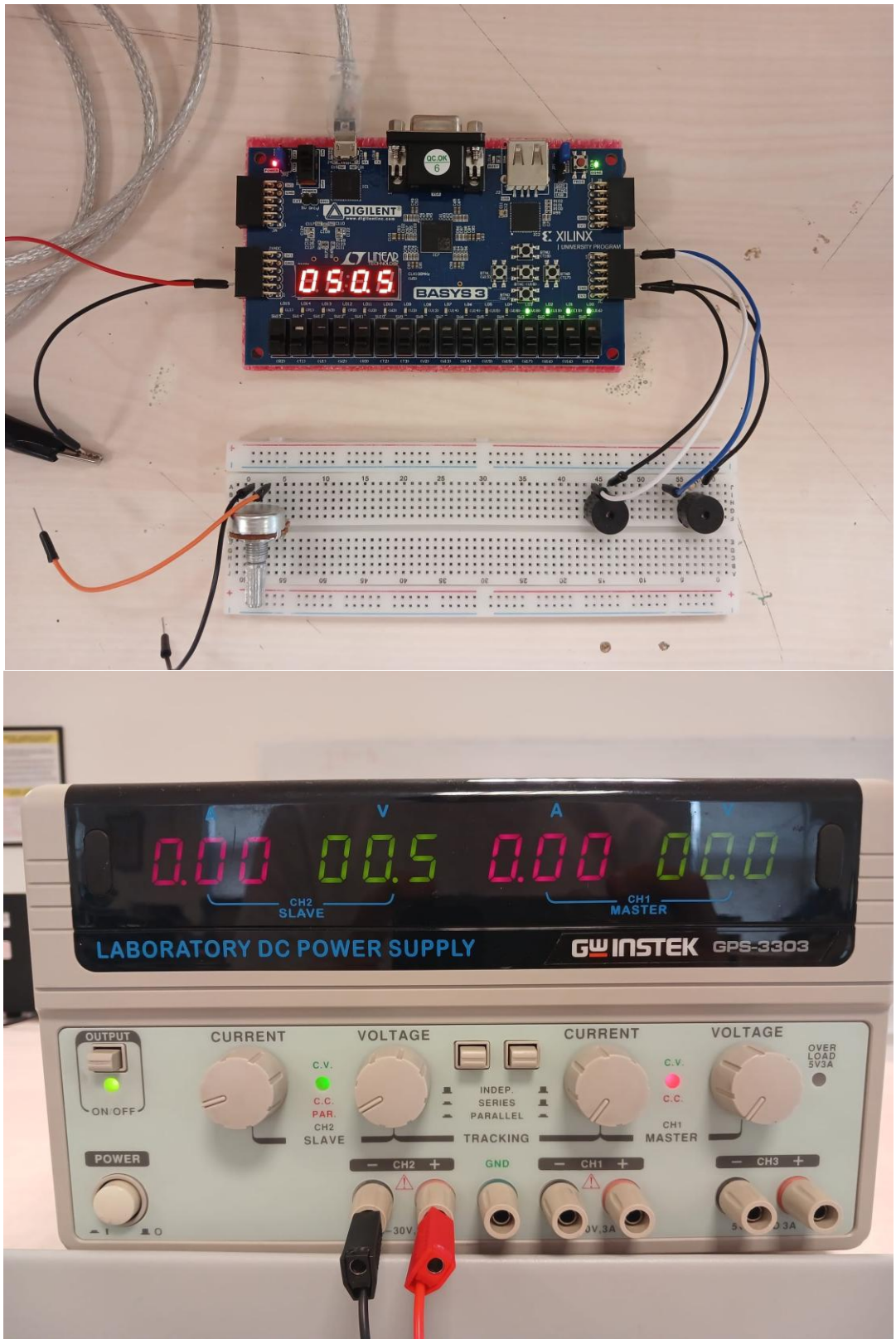

Figure 2.4: State of the design when it’s receiving 0.5 Volts

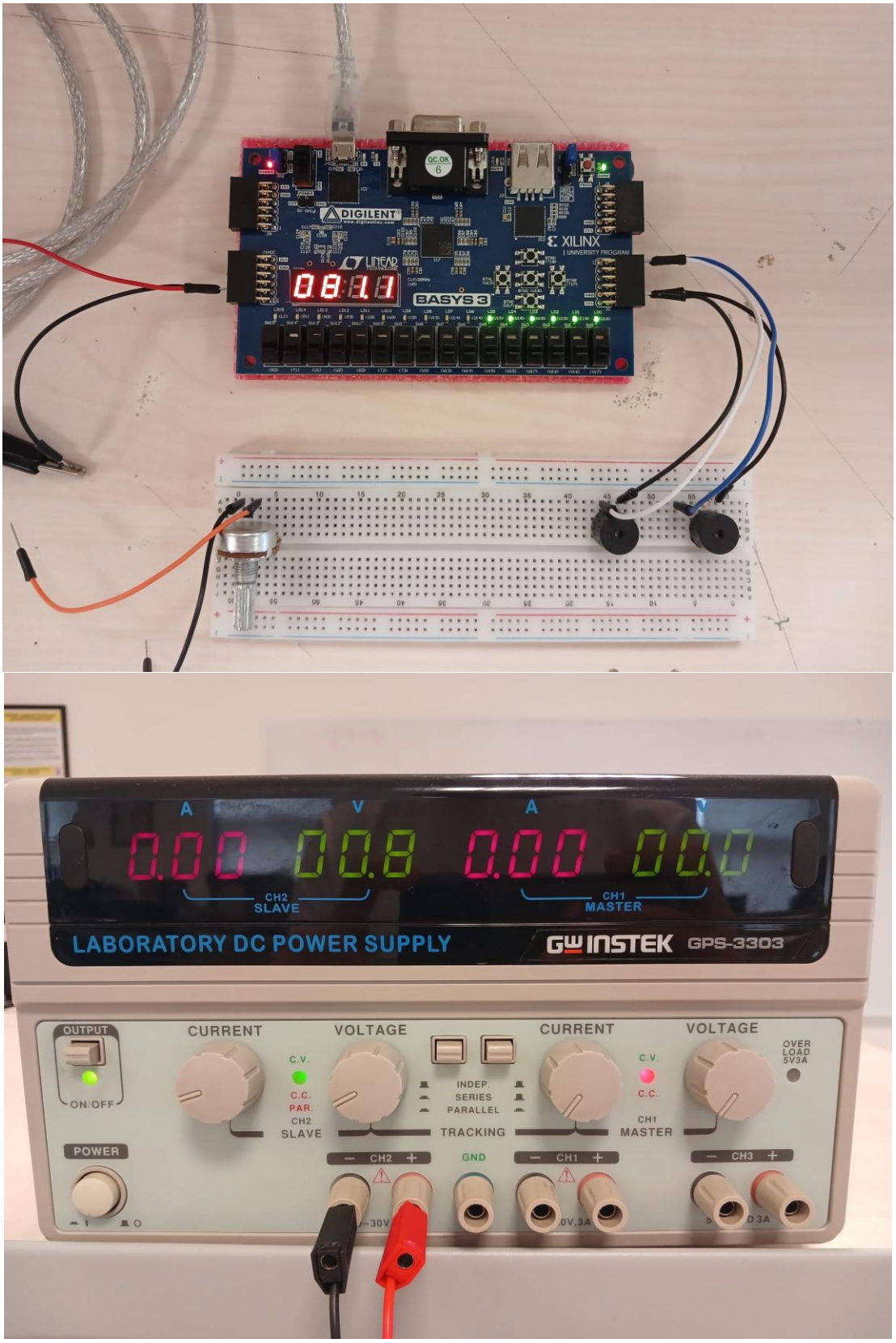

Figure 2.5: State of the design when it's receiving 0.8 Volts

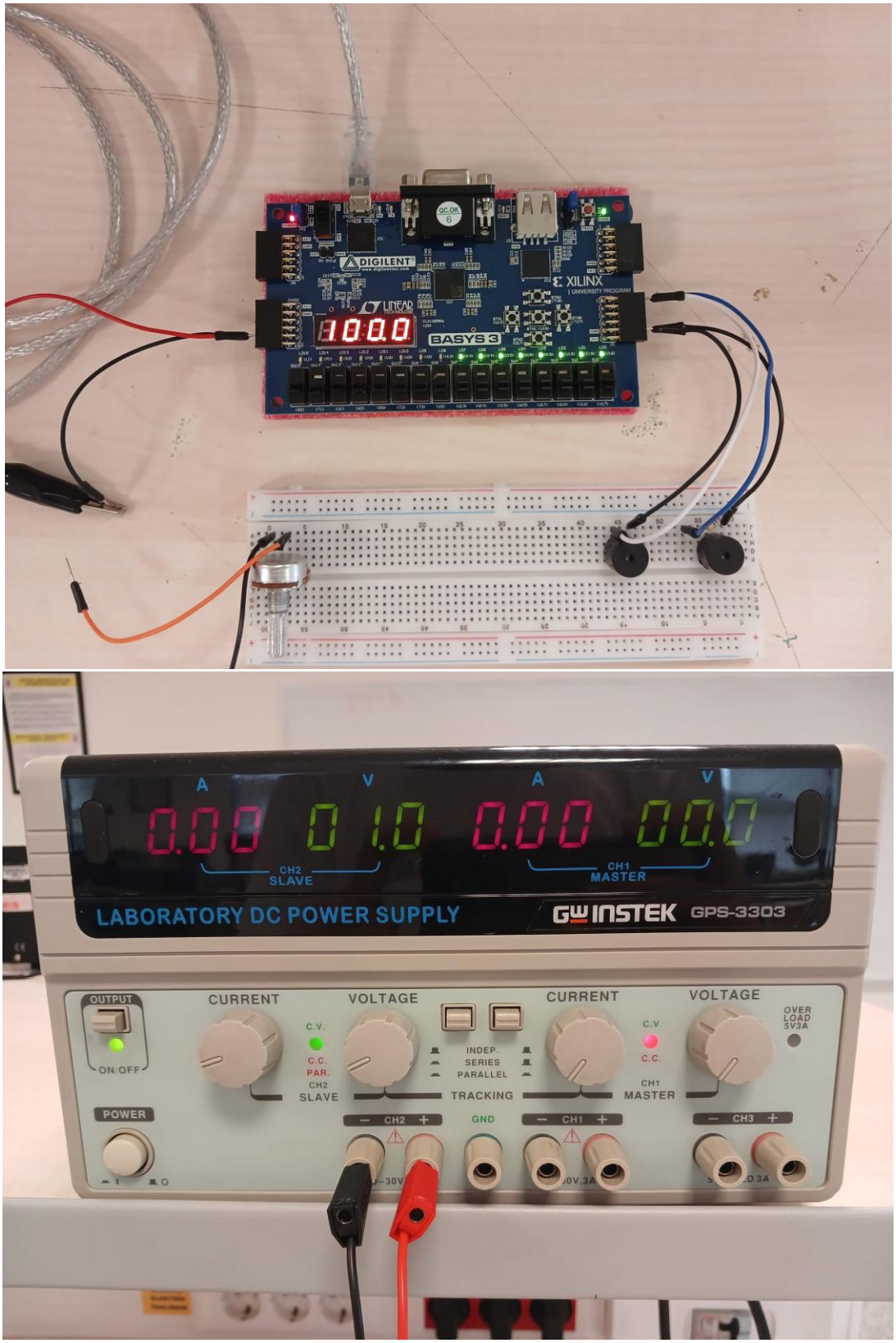

Figure 2.6: State of the design when it's receiving 1 Volts

After these images were analyzed, the project was found to function as desired.

Conclusion:

This term project successfully demonstrated an end-to-end implementation of an analog-to-digital voltage analyzer using the BASYS 3 FPGA and supporting hardware. Through a modular design approach, each component, from voltage acquisition with the XADC to output signaling via LEDs, seven-segment displays, and buzzers, was developed, tested, and integrated effectively. The system accurately captured and displayed voltage readings in real time, with visual and auditory indicators for significant changes. The inclusion of a hold function added an extra layer of usability, allowing the system to pause and retain readings when needed. Both the simulation results and real-world testing confirmed the correctness and reliability of the design. Overall, the project met all it's design goals and through making the project, the user was able to further understand VHDL programming and hardware integration.

References:

AMD. (2023). *7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide (UG480)*. AMD. https://docs.amd.com/r/en-US/ug480_7Series_XADC/7-Series-FPGAs-and-Zynq-7000-SoC-XADC-Dual-12-Bit-1-MSPS-Analog-to-Digital-Converter-User-Guide-UG480

Digilent. (n.d.). *Basys 3 Reference Manual.* Digilent Inc. https://digilent.com/reference/programmable-logic/basys-3/reference-manual

Digilent. (n.d.). *Basys 3 Reference Manual (PDF).* Digilent Inc. https://digilent.com/reference/_media/basys3:basys3_rm.pdf

Appendix:

main.vhd

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

entity main is
  Port (
```

```
        clk         : in  std_logic;
        reset       : in  std_logic;
        ADC         : in  std_logic_vector(1 downto 0);
        hold_switch : in  std_logic;
        leds        : out std_logic_vector(7 downto 0);
        anodes      : out std_logic_vector(3 downto 0);
        segments    : out std_logic_vector(7 downto 0);
        buzzer      : out std_logic_vector(1 downto 0)
    );
end main;

architecture Behavioral of main is

  component xadc_wiz_0 is
    Port (
      dclk_in       : in  std_logic;
      reset_in      : in  std_logic;
      den_in        : in  std_logic;
      dwe_in        : in  std_logic;
      daddr_in      : in  std_logic_vector(6 downto 0);
      di_in         : in  std_logic_vector(15 downto 0);
      do_out        : out std_logic_vector(15 downto 0);
      drdy_out      : out std_logic;
      vauxp14       : in std_logic;
      vauxn14       : in std_logic;
      busy_out      : out std_logic;
      channel_out   : out std_logic_vector(4 downto 0);
      eoc_out       : out std_logic;
      eos_out       : out std_logic;
      alarm_out     : out std_logic;
```

```
    vp_in         : in  std_logic;
    vn_in         : in  std_logic
  );
end component;


component XADC_Interface is
 Port  (
      clk      : in  std_logic;
      reset      : in  std_logic;
      data_in  : in  std_logic_vector(11 downto 0);
      data_out : out std_logic_vector(11 downto 0)
  );
end component;

component LED_Controller is
  Port (
    clk       : in  std_logic;
    reset     : in  std_logic;
    hold      : in  std_logic;
    adc_value : in  std_logic_vector(11 downto 0);
    leds      : out std_logic_vector(7 downto 0)
  );
end component;

component Display_Controller is
  Port (
    clk         : in  std_logic;
    reset       : in  std_logic;
    hold        : in  std_logic;
```

```
    adc_value   : in  std_logic_vector(11 downto 0);
    anodes      : out std_logic_vector(3 downto 0);
    segments    : out std_logic_vector(7 downto 0)
  );
end component;

component Buzzer_Controller is
  Port (
    clk        : in  std_logic;
    reset      : in  std_logic;
    hold       : in  std_logic;
    adc_value  : in  std_logic_vector(11 downto 0);
    buzzer_out : out std_logic_vector(1 downto 0)
  );
end component;

signal adc_data_internal : std_logic_vector(15 downto 0);
signal adc_value_12bit   : std_logic_vector(11 downto 0);
signal slow_adc_12bit    : std_logic_vector(11 downto 0);
signal EnableInt         : std_logic:='1';

begin

The_Brains: xadc_wiz_0
  port map (
    dclk_in     => clk,
    reset_in    => reset,
    den_in      => '1',
    dwe_in      => '0',
    daddr_in    => "0011110",
```

```
    di_in       => (others => '0'),
    do_out      => adc_data_internal,
    drdy_out    => open,
    vauxp14     => ADC(0),
    vauxn14     => ADC(1),
    busy_out    => open,
    channel_out => open,
    eoc_out     => EnableInt,
    eos_out     => open,
    alarm_out   => open,
    vp_in       => '0',
    vn_in       => '0'
  );

adc_value_12bit <= adc_data_internal(15 downto 4); -- Truncating to 12-bit

The_Timing: XADC_Interface
  port map (
    clk       => clk,
    reset     => reset,
    data_in   => adc_value_12bit,
    data_out  => slow_adc_12bit
  );

The_Lights: LED_Controller
  port map (
    clk       => clk,
    reset     => reset,
    hold      => hold_switch,
    adc_value => adc_value_12bit,
```

```
      leds      => leds
    );

  The_Sights: Display_Controller
    port map (
      clk       => clk,
      reset     => reset,
      hold      => hold_switch,
      adc_value => slow_adc_12bit,
      anodes    => anodes,
      segments  => segments
    );

  The_Sounds: Buzzer_Controller
    port map (
      clk        => clk,
      reset      => reset,
      hold       => hold_switch,
      adc_value  => adc_value_12bit,
      buzzer_out => buzzer
    );

end Behavioral;
```

XADC_Interface.vhd

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

entity XADC_Interface is
```

```
    Port (
        clk      : in  std_logic;
        reset    : in  std_logic;
        data_in  : in  std_logic_vector(11 downto 0);
        data_out : out std_logic_vector(11 downto 0)
    );
end XADC_Interface;

architecture Behavioral of XADC_Interface is
    signal counter             : integer range 0 to 12500000 := 0; -- for 80Hz
    signal data_reg            : std_logic_vector(11 downto 0);
begin

    process(clk, reset)
    begin
        if reset = '1' then
            counter   <= 0;
            data_reg  <= (others => '0');
        elsif rising_edge(clk) then
            if counter = 12499999 then
                counter   <= 0;
                data_reg  <= data_in;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

    data_out <= data_reg;
```

```
end Behavioral;
```

Display_Controller.vhd

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

entity Display_Controller is
    Port (
        clk         : in  std_logic;
        reset       : in  std_logic;
        hold        : in  std_logic;
        adc_value   : in  std_logic_vector(11 downto 0);
        anodes      : out std_logic_vector(3 downto 0);
        segments    : out std_logic_vector(7 downto 0)
    );
end Display_Controller;

architecture Behavioral of Display_Controller is
    -- 100MHz / 25000 = 4kHz refresh frequency.
    signal refresh_divider : unsigned(14 downto 0) := (others => '0');
    signal refresh_clk     : std_logic := '0';
    signal digit_counter   : unsigned(1 downto 0) := (others => '0');
    signal digit_values    : std_logic_vector(15 downto 0);
    signal latched_digits  : std_logic_vector(15 downto 0) := (others => '0');
    signal current_seg     : std_logic_vector(7 downto 0);

begin
```

```
process(clk)
begin
    if rising_edge(clk) then
        if reset = '1' then
            refresh_divider <= (others => '0');
            refresh_clk <= '0';
        else
            if refresh_divider = 24999 then
                refresh_divider <= (others => '0');
                refresh_clk <= '1';
            else
                refresh_divider <= refresh_divider + 1;
                refresh_clk <= '0';
            end if;
        end if;
    end if;
end process;

process(clk)
begin
    if rising_edge(clk) then
        if reset = '1' then
            digit_counter <= (others => '0');
        else
            if refresh_clk = '1' then
                digit_counter <= digit_counter + 1;
            end if;
        end if;
    end if;
end process;
```

```
    -- Combinational: Converting ADC value to BCD digits (millivolts 0-1000)
    process(adc_value)
        variable mv_temp : integer;
    begin
        mv_temp := to_integer(unsigned(adc_value)) * 1000 / 4095;

        digit_values(15 downto 12) <= std_logic_vector(to_unsigned((mv_temp / 1000) mod 10,
4));
        digit_values(11 downto 8)  <= std_logic_vector(to_unsigned((mv_temp / 100) mod 10, 4));
        digit_values(7 downto 4)   <= std_logic_vector(to_unsigned((mv_temp / 10) mod 10, 4));
        digit_values(3 downto 0)   <= std_logic_vector(to_unsigned(mv_temp mod 10, 4));
    end process;



    process(clk)
    begin
        if rising_edge(clk) then
            if reset = '1' then
                latched_digits <= (others => '0');
            else
                if hold = '0' then
                    latched_digits <= digit_values;
                end if;
            end if;
        end if;
    end process;

    process(digit_counter, latched_digits)
        variable bcd_digit : std_logic_vector(3 downto 0);
```

```
begin
    case digit_counter is
        when "00" =>
            anodes <= "1110";
            bcd_digit := latched_digits(3 downto 0);
        when "01" =>
            anodes <= "1101";
            bcd_digit := latched_digits(7 downto 4);
        when "10" =>
            anodes <= "1011";
            bcd_digit := latched_digits(11 downto 8);
        when others =>
            anodes <= "0111";
            bcd_digit := latched_digits(15 downto 12);
    end case;

    -- 7-segment decoder (active-low)
    case bcd_digit is
        when "0000" => current_seg(6 downto 0) <= "1000000"; -- 0
        when "0001" => current_seg(6 downto 0) <= "1111001"; -- 1
        when "0010" => current_seg(6 downto 0) <= "0100100"; -- 2
        when "0011" => current_seg(6 downto 0) <= "0110000"; -- 3
        when "0100" => current_seg(6 downto 0) <= "0011001"; -- 4
        when "0101" => current_seg(6 downto 0) <= "0010010"; -- 5
        when "0110" => current_seg(6 downto 0) <= "0000010"; -- 6
        when "0111" => current_seg(6 downto 0) <= "1111000"; -- 7
        when "1000" => current_seg(6 downto 0) <= "0000000"; -- 8
        when "1001" => current_seg(6 downto 0) <= "0010000"; -- 9
        when others => current_seg(6 downto 0) <= "1111111"; -- blank
    end case;
```

```
            if digit_counter = "01" then
                current_seg(7) <= '0';
            else
                current_seg(7) <= '1';
            end if;
        end process;

        segments <= current_seg;

    end Behavioral;
```

Buzzer_Controller.vhd

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


    entity Buzzer_Controller is
        Port (
            clk        : in  std_logic;
            reset      : in  std_logic;
            hold       : in  std_logic;
            adc_value  : in  std_logic_vector(11 downto 0);
            buzzer_out : out std_logic_vector(1 downto 0)       -- "10"=rise, "01"=fall, "00"=no change
        );
    end Buzzer_Controller;

    architecture Behavioral of Buzzer_Controller is
```

```
    signal div_counter    : integer range 0 to 49999999 := 0;
    signal prev_threshold : integer range 0 to 8 := 0;
    signal buzzer_reg     : std_logic_vector(1 downto 0) := "00";
begin
    process(clk)
        variable curr_threshold : integer range 0 to 8;
    begin
        if rising_edge(clk) then
            if reset = '1' then
                div_counter    <= 0;
                prev_threshold <= 0;
                buzzer_reg     <= "00";
            else
                if div_counter = 49999999 then

                    if hold = '0' then
                        curr_threshold := (to_integer(unsigned(adc_value)) * 1000 / 4095) / 125;

                        if curr_threshold > prev_threshold then
                            buzzer_reg <= "10";  -- rising
                        elsif curr_threshold < prev_threshold then
                            buzzer_reg <= "01";  -- falling
                        else
                            buzzer_reg <= "00";  -- unchanged
                        end if;
                        prev_threshold <= curr_threshold;
                    end if;
                    div_counter <= 0;
                else
                    div_counter <= div_counter + 1;
```

```
                end if;
            end if;
        end if;
    end process;

    buzzer_out <= buzzer_reg;
end Behavioral;
```

main_tb.vhd

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

    entity main_tb is
    end main_tb;

    architecture Behavioral of main_tb is

        signal clk         : std_logic := '0';
        signal reset       : std_logic := '1';
        signal hold        : std_logic := '0';

        signal adc_value   : std_logic_vector(11 downto 0) := (others => '0');

        signal leds        : std_logic_vector(7 downto 0);
        signal anodes      : std_logic_vector(3 downto 0);
        signal segments    : std_logic_vector(7 downto 0);
        signal buzzer_out  : std_logic_vector(1 downto 0);

    begin
```

```
clk_process : process
begin
    while now < 100 us loop
        clk <= '0';
        wait for 5 ns;
        clk <= '1';
        wait for 5 ns;
    end loop;
    wait;
end process;

stimulus : process
begin
    wait for 20 ns;
    reset <= '0':
    wait for 100 us;
    wait;
end process;

adc_driver : process(clk)
    variable adc_val : unsigned(11 downto 0) := (others => '0');
begin
    if rising_edge(clk) then
        adc_value <= std_logic_vector(adc_val);
        adc_val := adc_val + 1;
    end if;
end process;

LED_INST : entity work.LED_Controller
    port map (
```

```
            clk       => clk,
            reset     => reset,
            hold      => hold,
            adc_value => adc_value,
            leds      => leds
        );

    DISP_INST : entity work.Display_Controller
        port map (
            clk       => clk,
            reset     => reset,
            hold      => hold,
            adc_value => adc_value,
            anodes    => anodes,
            segments  => segments
        );

    BUZZ_INST : entity work.Buzzer_Controller
        port map (
            clk       => clk,
            reset     => reset,
            hold      => hold,
            adc_value  => adc_value,
            buzzer_out => buzzer_out
        );

end Behavioral;
```

please_work.xdc

```
## Clock signal
```

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

set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports reset]


## XADC Analog Inputs (using the JXADC header pins)
#set_property PACKAGE_PIN J3 [get_ports {ADC(0)}]
#set_property IOSTANDARD LVCMOS33 [get_ports {ADC(0)}]
#set_property PACKAGE_PIN K3 [get_ports {ADC(1)}]
#set_property IOSTANDARD LVCMOS33 [get_ports {ADC(1)}]

#set_property -dict { PACKAGE_PIN L3   IOSTANDARD LVCMOS33 } [get_ports {ADC(0)}]
#set_property -dict { PACKAGE_PIN M3   IOSTANDARD LVCMOS33 } [get_ports
{ADC(1)}]

set_property PACKAGE_PIN L3 [get_ports ADC[0]]
set_property PACKAGE_PIN M3 [get_ports ADC[1]]



set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {leds[0]}]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports {leds[1]}]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports {leds[2]}]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports {leds[3]}]
set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports {leds[4]}]
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports {leds[5]}]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {leds[6]}]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {leds[7]}]
```

```
set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports {segments[0]}]
set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports {segments[1]}]
set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports {segments[2]}]
set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports {segments[3]}]
set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports {segments[4]}]
set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports {segments[5]}]
set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports {segments[6]}]
set_property -dict { PACKAGE_PIN V7   IOSTANDARD LVCMOS33 } [get_ports {segments[7]}]

set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports {anodes[0]}]
set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports {anodes[1]}]
set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports {anodes[2]}]
set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports {anodes[3]}]

set_property -dict { PACKAGE_PIN L17   IOSTANDARD LVCMOS33 } [get_ports {buzzer[0]}]
set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 } [get_ports {buzzer[1]}]



set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports hold_switch]
```