SHAPE AND DISTANCE DETECTION

Bilkent University Electrical and Electronics Engineering Department

EE102 Project Report

Yiğit Narter

22102718

Section 01

Video Presentation

https://www.youtube.com/watch?v=zwSNeKLz7V0&ab_channel=Yi%C4%9FitNarter

Purpose

In the term project, the purpose is to implement everything we have learned in the course using VHDL, such as combinational logic, registers or finite state machines (FSM). In my project, I wanted to design a system that displays the shapes of objects via the LEDs of the BASYS3 board, as well as the distance of the object through the seven-segment display. There is also an alarm system that rings with a higher frequency as the object comes closer.

Design Methodology

For the shape detection part, a Python code is used. In the code, I used a module called OpenCV. This module detects the shape using a method called “curve approximation”. In the VHDL part, the result is received from Python using UART communication. Lastly, the shape is indicated through the LEDs of the BASYS3 board.

For the distance detection and buzzer part, an HC-SR04 ultrasonic sensor is used. This component detects the distance of objects by emitting sound waves and calculating the traveled distance of the wave using the speed of sound and the time passed. The component needs a 10-microsecond trigger input, which will be generated with VHDL. After the distance is found in binary, it is converted to binary coded decimal (BCD) using the “Double Dabble” method. This is done so that the decimal digits can be displayed separately. Lastly, the result is displayed through the seven-segment display of the board. The buzzer is also controlled with the found distance, and the frequency of the buzzer increases as the object gets closer to the sensor.

Design Specifications

For detecting the shape and displaying it on the LEDs, UART communication is needed. The top module for this part, uart_main, takes three inputs:

- clk: 10MHz clock of the BASYS3
- reset: for resetting the data communication
- rx: one-bit data received by BASYS3 through the USB port of the computer.

There are three one-bit outputs: tr (triangle), rec (rectangle), and circle (cir) which are assigned to different LEDs of the board. Every time these shapes are detected, the corresponding LED turns on.

The top module uses a sub-module, uart_receiver, which functions for receiving the data. It has the same three inputs: clk, reset, rx. It gives as output the 8-bit received data “rx_data_out”. The module includes a finite state machine (FSM) to receive the data. The states in this FSM are EMPTY, START, LOAD and STOP. The receiver is in the “EMPTY” state until the serial start input ‘0’ is found. Then it goes to the “START” state. If the signal being received is valid, the “LOAD” state is reached and the serial 8-bit input is stored through a register, else the next state is “EMPTY”. After the stop input ‘1’ is received the “STOP” state is reached and the code moves back to the “EMPTY” state. Figures 1 and 2 show the RTL schematics for the modules “uart_main” and “uart_receiver”.

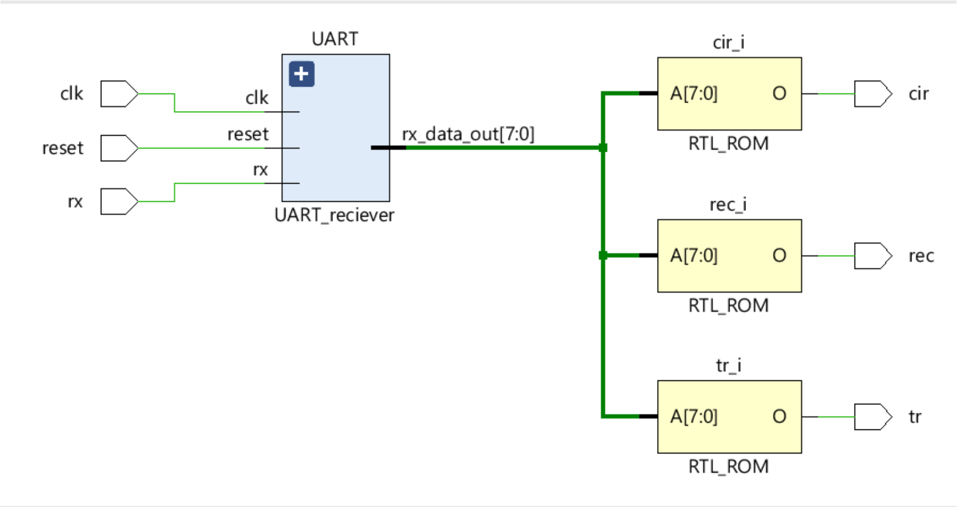

Figure 1: RTL schematic for "uart_main"

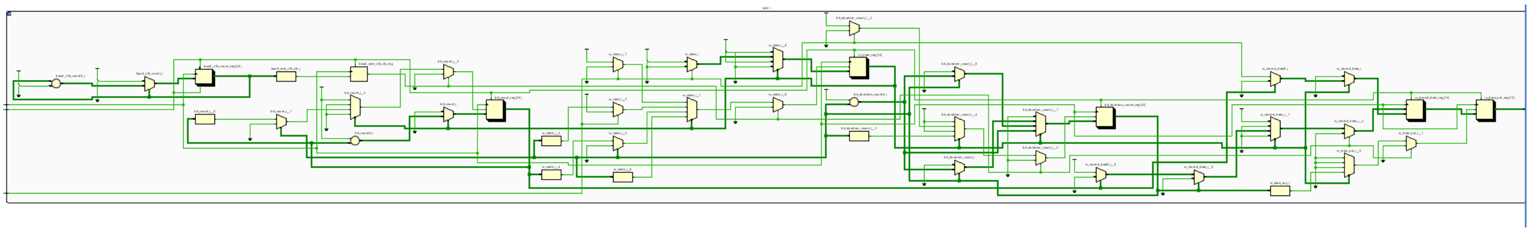

Figure 2: RTL schematic for "uart_receiver"

The top module for distance detection, distance_detect, has the following inputs:

- clk : 100MHz clock of the BASYS3 board.
- reset : to reset the detection.
- echo : echo input given by the ultrasonic sensor, used to determine the distance.
- measure : the distance is measured when this input is '1'. Taken from the user by a button on the BASYS3 board.
- enable : when this input is '1', the distance is displayed continuously on the seven-segment display. Taken from the user by a switch on the BASYS3 board.

The outputs are:

- buzzer : controls the buzzer, the buzzer goes off when this output is '1'.
- trigger : the necessary 10 us trigger for the ultrasonic sensor.
- data_valid : indicates when the measurement is done.
- an : controls the anodes of the seven-segment display.
- cat0, cat1, cat2, cat3, cat4, cat5, cat6: the bits control the cathodes of the seven-segment display.

The top modules has two states: "meas" and "hold". In the "meas" state, which is reached when the input "enable" is '1', the measurement is done and the result is displayed continuously. When the "enable" input is '0', the result is found only when the input "measure" is 1. Figure 3 shows the RTL schematic for distance_detect.

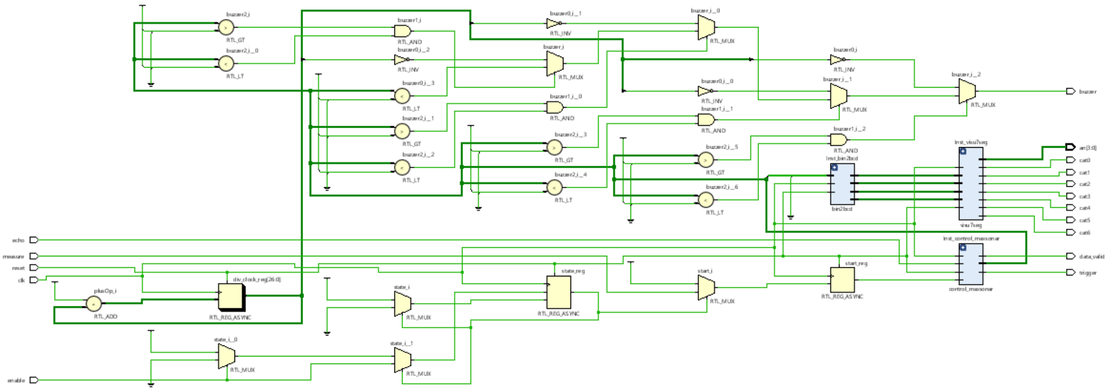

Figure 3: RTL schematic of "distance_detect"

The top module includes three sub-modules. The first one is "sensor_cont", with 4 inputs (clk, reset, start, echo) and 3 outputs (trigger, data_valid, distance). This module functions for generating the trigger for the sensor and calculating the distance when "start" input is '1'. After the distance is found as a binary number, the module "bin_to_dec" converts this number to binary-coded decimal (BCD). The inputs are clk, reset and b_in (number to be converted). The outputs dec_1, dec_10, dec_100 and dec_1000 are the digits of the BCD number. This module uses the "Double Dabble" method to convert the number. Lastly, the digits received from this module is displayed on the seven-segment display, which is done by the module "seven_segment". It takes clk, reset and the four digits as inputs. The outputs are an, cat0, cat1, cat2, cat3, cat4, cat5, cat6. They control the seven-segment display so that the number is displayed. Figures 4-6 show the RTL schematics for these modules.

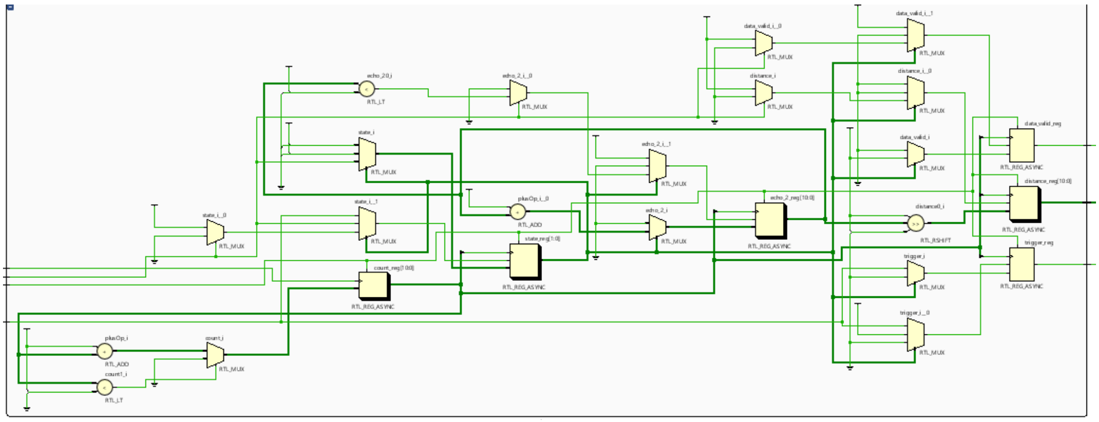

Figure 4: RTL schematic for "sensor_cont"

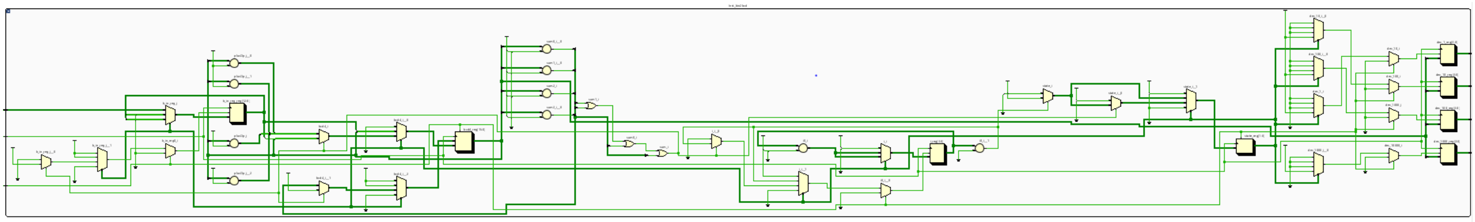

Figure 5: RTL schematic for "bin_to_dec"

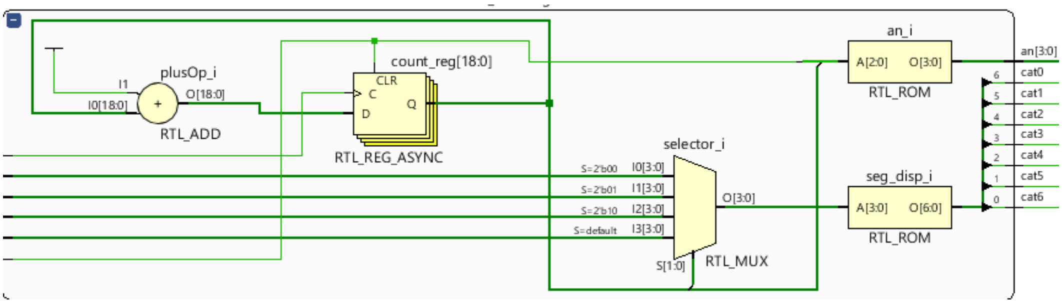

Figure 6: RTL schematic for "seven_segment"

The codes are then implemented on the BASYS3 board. The switches, buttons and PMOD connectors are used for the inputs. The outputs are displayed using the LEDs and the seven-segment display. Figure 7 shows the BASYS3 configuration.

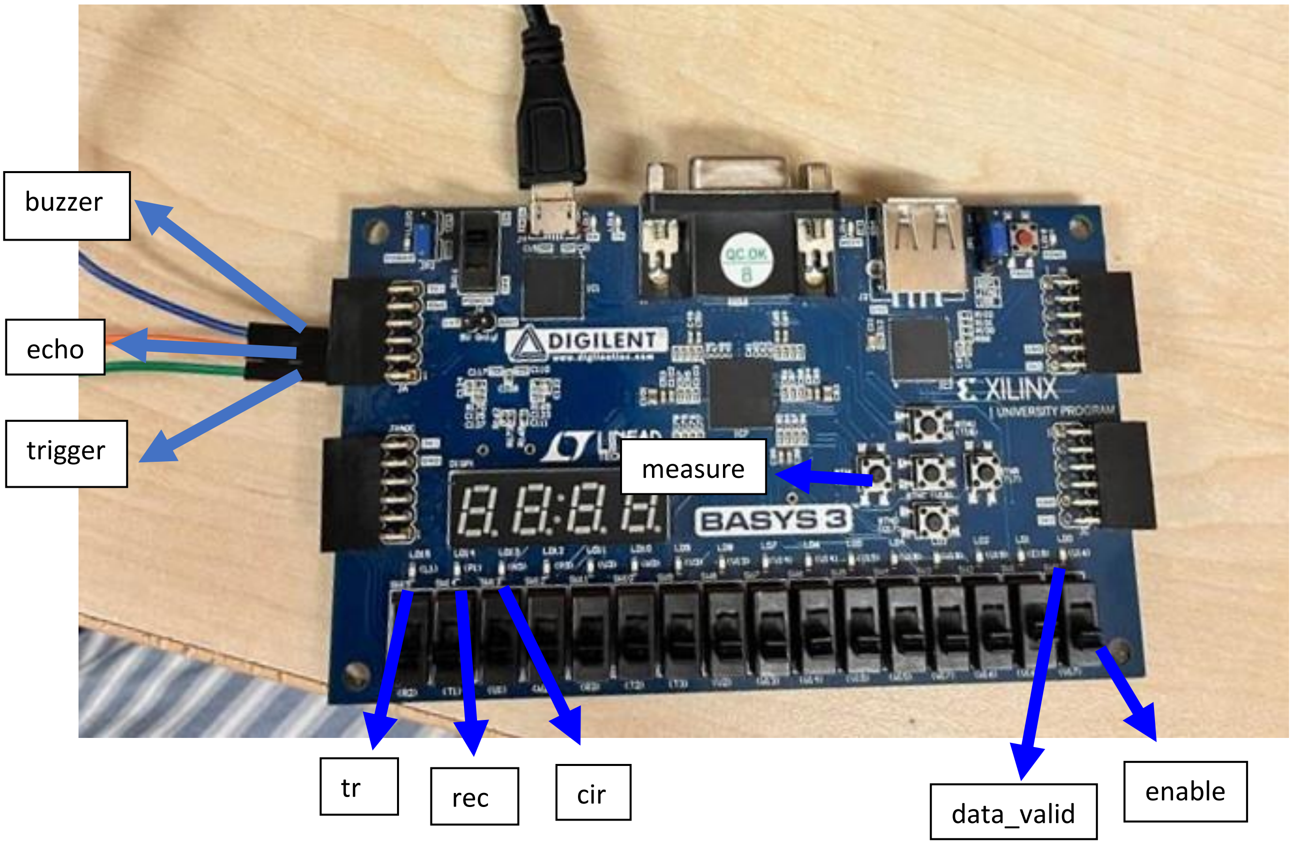

Figure 7: BASYS3 configuration

Results

For both shape and distance detections, the results were as expected. The shape detection is tested with images of several shapes. The LEDs of the corresponding shapes are turned on when the shapes are on the screen. Figures 8-10 show the results with the detected shapes.

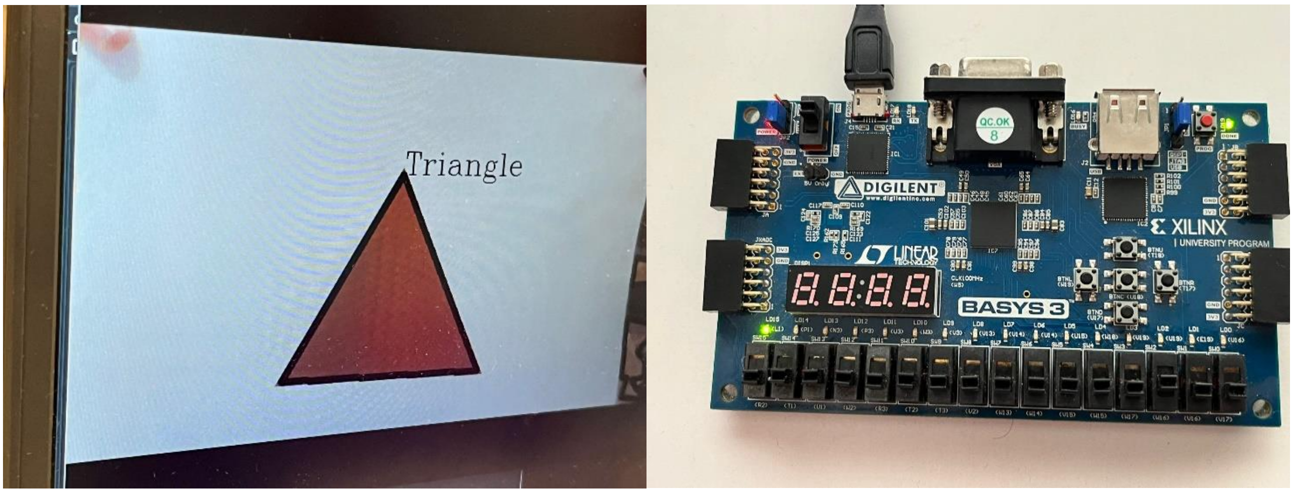

Figure 8: The shape is a triangle and the corresponding LED is on

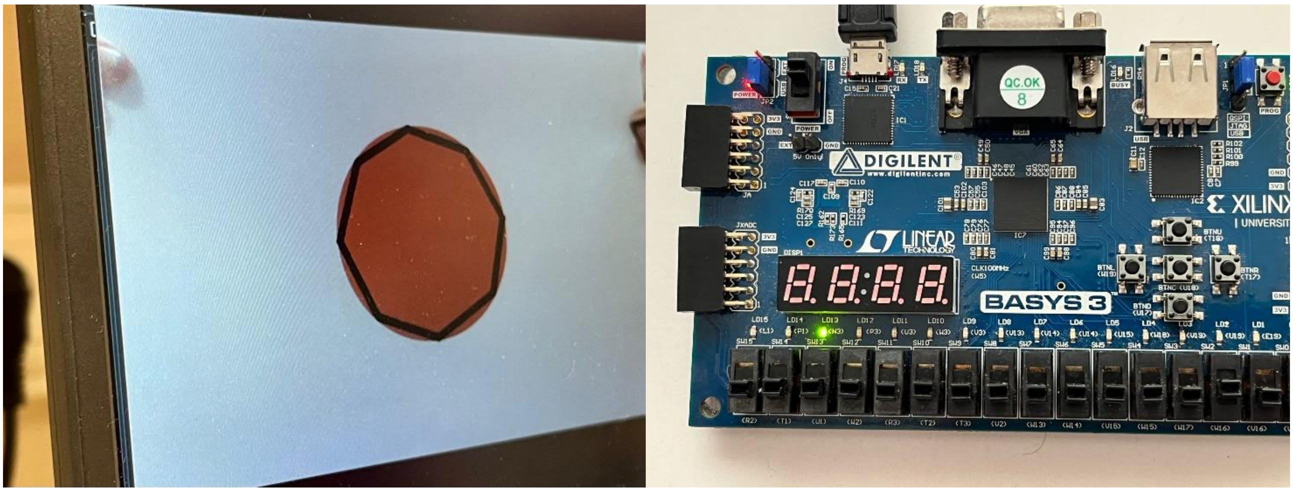

Figure 9: The shape is a circle and the corresponding LED is on

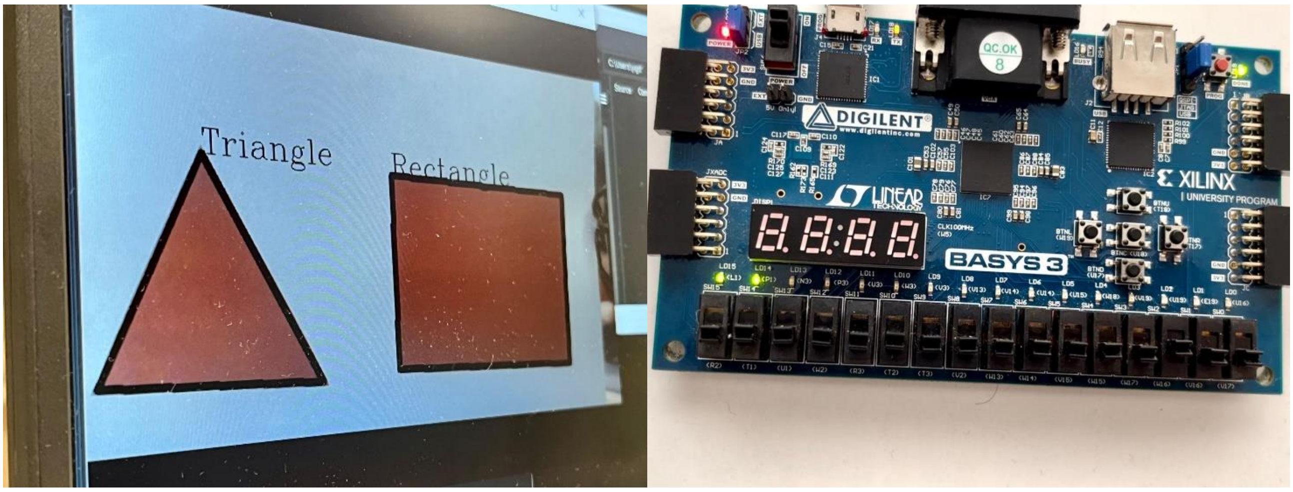

Figure 10: The shapes are a triangle and a rectangle and the corresponding LEDs are on

For the distance detection part, the distance is correctly displayed on the seven-segment display. The buzzer also rings with its frequency decreasing in every 10 cm range. Figures 11 and 12 show the set up circuit with the sensor and buzzer, along with the distance on the seven-segment display.

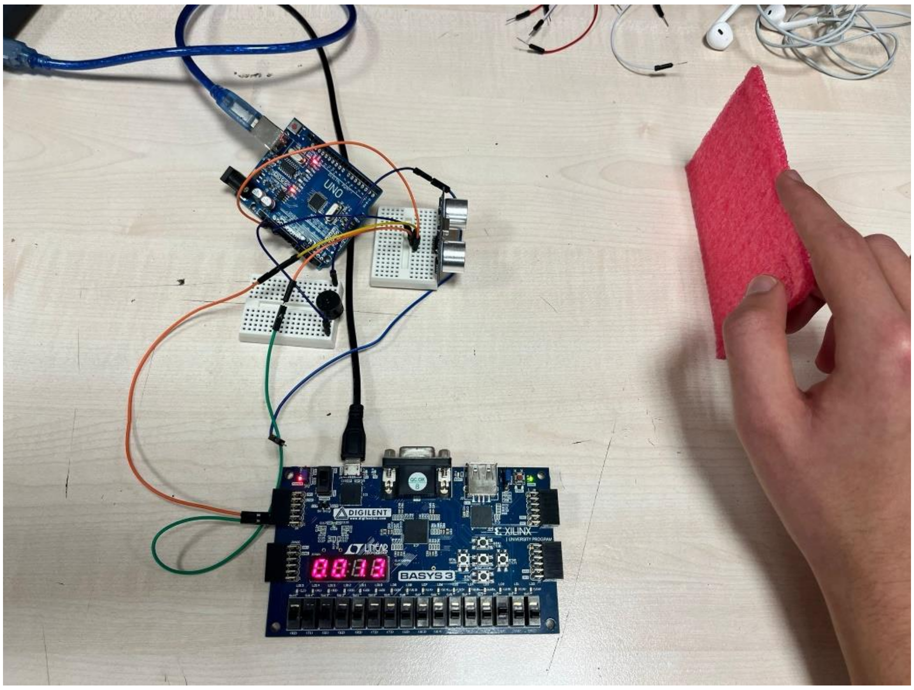

Figure 11: The distance is 13 cm and it is displayed

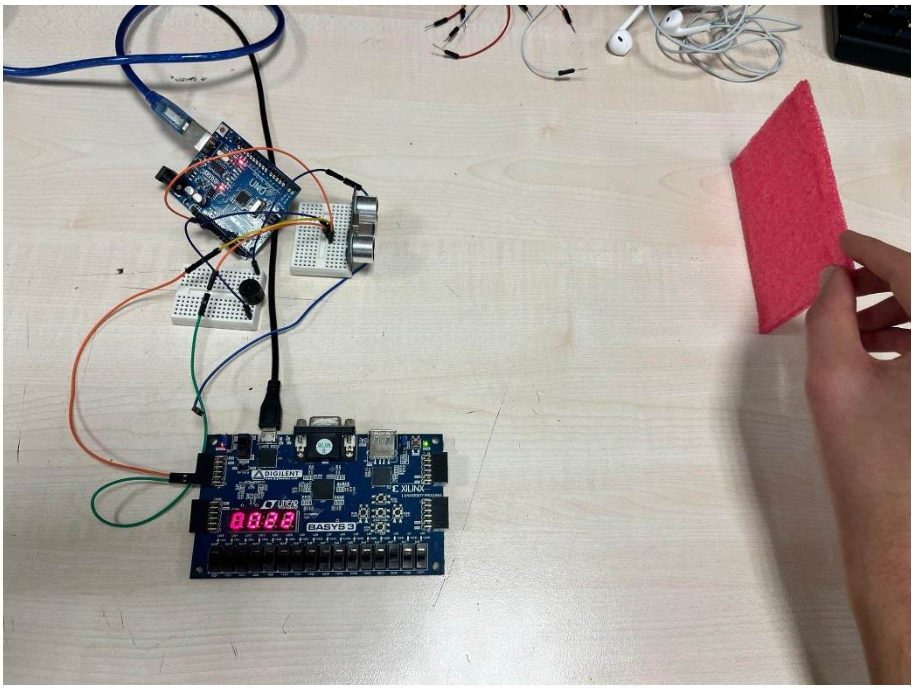

Figure 12: The distance is 22 cm and it is displayed

Conclusion

In my project, I wanted to implement a shape and distance detector using VHDL and the BASYS3 board. To do that, I used external components like an HCSR04 ultrasonic sensor and a buzzer. I also used an Arduino Uno board to supply 5V voltage to the sensor, as well as mini breadboards and jumpers for an organized circuit. For shape detection, I used a Python code and transmitted the result to BASYS3 using UART communication. Since the shape is already displayed with the LEDs as well as on the computer screen, I didn't use a VGA monitor. The project was successful, and the results turned out as expected. Still, in the future, this project can be improved by adding more types of shapes, such as hexagons and octagons, or by adding another external component like a camera module.

For the project, I had to do a lot of research. I studied the datasheets of the components and learned how to make these components work using VHDL. I also learned about the UART communication protocol and implemented it on VHDL. I had to use outside sources, which I have cited, in parts of my code. Additionally, the concepts we learned in this course, such as finite state machines and registers, are frequently used in the code I've written. Even though it was really hard to complete it, this project helped me implement the ideas we learned in this course. I got an idea about how these concepts relate to real-life problems and implementations. Overall, I can say that this project was a tiresome but also a very beneficial experience for me.

References

"Binary to BCD and BCD to Binary." *RealDigital*, https://www.realdigital.org/doc/6dae6583570fd816d1d675b93578203d

Canu, Sergio. "Real Time Shape Detection - Opencv with Python 3." *Pysource*, 9 Sept. 2019, https://pysource.com/2018/12/29/real-time-shape-detection-opencv-with-python-3/

Data Sheet - SparkFun Electronics. https://cdn.sparkfun.com/datasheets/Sensors/Proximity/apds9960.pdf

Özgür, Uğur. "HC-SR04 WITH FPGA." *Uğur ÖZGÜR*, 10 May 2020, https://ugur-ozgur.gen.tr/blog/index.php/2020/05/10/hc-sr04-with-fpga/

Russell. "UART in VHDL and Verilog for an FPGA." *Nandland*, 16 Sept. 2022, https://nandland.com/uart-serial-port-module/

Sudbin, Alexey. "UART Interface in VHDL for BASYS3 Board." *Hackster.io*, 1 Nov. 2020, https://www.hackster.io/alexey-sudbin/uart-interface-in-vhdl-for-basys3-board-eef170

Appendices

Design Codes

uart_main.vhd

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

entity uart_main is
 Port (clk : in STD_LOGIC;
 reset : in STD_LOGIC;
 rx : in std_logic;
 tr: out STD_lOGIC;
 rec: out STD_lOGIC;
 cir: out STD_lOGIC
 );
end main;

architecture Behavioral of uart_main is
  signal shape: std_logic_vector(7 downto 0);

begin

proc UART: entity work.uart_reciever(Behavioral)
 port map(clk => clk,
 reset => reset,
 rx => rx,
 rx_data_out => shape);

tr <= '1' when shape="01100001" else '0';
rec <= '1' when shape="01100010" else '0';
cir <= '1' when shape="01100011" else '0';
```

```
end Behavioral;
```

uart_receiver.vhd

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

entity uart_reciever is
 generic(
 BAUD_X16_CLK_TICKS: integer := 54);
 Port(clk : in std_logic;
 reset : in std_logic;
 rx : in std_logic;
 rx_data_out : out std_logic_vector (7 downto 0)
 );
end uart_reciever;

architecture Behavioral of uart_reciever is
 type reciever_state is (EMPTY, START, LOAD, STOP);
 signal pr_state: reciever_state := EMPTY;
 signal baud_rate_x16_clk : std_logic := '0';
 signal rec_data : std_logic_vector(7 downto 0) := (others => '0');

begin

 baud_x16_clk: process(clk)
 variable baud_x16_count: integer range 0 to (BAUD_X16_CLK_TICKS - 1) :=
(BAUD_X16_CLK_TICKS - 1);
 begin
     if rising_edge(clk) then
         if (reset = '1') then
             baud_rate_x16_clk <= '0';
             baud_x16_count := (BAUD_X16_CLK_TICKS - 1);
         else
             if (baud_x16_count = 0) then
                 baud_rate_x16_clk <= '1';
                 baud_x16_count := (BAUD_X16_CLK_TICKS - 1);
             else
                 baud_rate_x16_clk <= '0';
                 baud_x16_count := baud_x16_count - 1;
             end if;
         end if;
     end if;
 end process baud_x16_clk;

uart_fsm: process(clk)
    variable time_bit : integer range 0 to 15 := 0;
    variable count_bit : integer range 0 to 7 := 0;

 begin
     if rising_edge(clk) then
         if (reset = '1') then
             pr_state <= EMPTY;
             rec_data <= (others => '0');
             rx_data_out <= (others => '0');
             time_bit := 0;
             count_bit := 0;
```

```
else
    if (baud_rate_x16_clk = '1') then
       case pr_state is

            when EMPTY =>
                rec_data <= (others => '0');
                time_bit := 0;
                count_bit := 0;
                if (rx = '0') then
                   pr_state <= START;
                end if;

            when START =>
            if (rx = '0') then
               if (time_bit = 7) then
                   pr_state <= DATA;
                   time_bit := 0;
               else
                   time_bit := time_bit + 1;
               end if;
            else
               pr_state <= EMPTY;
            end if;

            when LOAD =>
                if (time_bit = 15) then
                    rec_data(count_bit) <= rx;
                    time_bit := 0;
                    if (count_bit = 7) then
                       pr_state <= STOP;
```

```
                                 time_bit := 0;
                              else
                                 count_bit := count_bit + 1;
                              end if;
                          else
                             time_bit := time_bit + 1;
                          end if;

                     when STOP =>
                          if (time_bit = 15) then
                             rx_data_out <= rec_data;
                             pr_state <= EMPTY;
                          else
                             time_bit := time_bit + 1;
                          end if;

                     when others =>
                        pr_state <= EMPTY;

                end case;
             end if;
         end if;
     end if;
 end process uart_fsm;
```

distance_detect.vhd

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

entity distance_detect is
```

```
    Port ( clk : in  std_logic;
           reset : in  std_logic;
           echo : in  std_logic;
           measure : in  std_logic;
           enable : in  std_logic;
           buzzer : out std_logic;
           trigger : out std_logic;
           data_valid : out std_logic;
           an : out std_logic_vector(3 downto 0);
           cat0 : out std_logic;
           cat1 : out std_logic;
           cat2 : out std_logic;
            cat3 : out std_logic;
            cat4 : out std_logic;
            cat5 : out std_logic;
             cat6 : out std_logic
           );
end distance_detect;


architecture Behavioral of distance_detect is


   component sensor_cont
            Port(
                       clk : in std_logic;
                       reset : in std_logic;
                       start : in std_logic;
                       echo : in std_logic;
                       trigger : out std_logic;
                       data_valid : out std_logic;
```

```
                distance : out std_logic_vector(10 downto 0)
                );
        end component;

component bin_to_dec
        Port(
                clk : in std_logic;
                reset : in std_logic;
                b_in : in std_logic_vector(12 downto 0);
                dec_1 : out std_logic_vector(3 downto 0);
                dec_10 : out std_logic_vector(3 downto 0);
                dec_100 : out std_logic_vector(3 downto 0);
                dec_1000 : out std_logic_vector(3 downto 0)
                );
        end component;

component seven_segment
        Port(
                clk : in std_logic;
                reset : in std_logic;
                dec_1 : in std_logic_vector(3 downto 0);
                dec_10 : in std_logic_vector(3 downto 0);
                dec_100 : in std_logic_vector(3 downto 0);
                dec_1000 : in std_logic_vector(3 downto 0);
                an : out std_logic_vector(3 downto 0);
                cat0 : out std_logic;
                cat1 : out std_logic;
                cat2 : out std_logic;
                cat3 : out std_logic;
                cat4 : out std_logic;
```

```
			cat5 : out std_logic;
			cat6 : out std_logic
			);
		end component;

	signal dec_1, dec_10, dec_100, dec_1000 : std_logic_vector(3 downto 0);
	signal b_in : std_logic_vector(12 downto 0);
	signal start : std_logic;
	signal distance : std_logic_vector(10 downto 0);
	signal distance_2 : std_logic_vector(12 downto 0);
	type state_type is (meas, hold);
	signal state: state_type;
	signal div_clock : unsigned(26 downto 0);

begin

	comp_sensor_cont: sensor_cont port map(
			clk => clk,
			reset => reset,
			start => start,
			echo => echo,
			trigger => trigger,
			data_valid => data_valid,
			distance => distance
		);

	comp_bin_to_dec: bin_to_dec port map(
			clk => clk,
			reset => reset,
			b_in => distance_2,
              dec_1 => dec_1,
              dec_10 => dec_10,
              dec_100 => dec_100,
              dec_1000 => dec_1000
      );

comp_seven_segment: seven_segment port map(
              clk => clk,
              reset => reset,
              dec_1 => dec_1,
              dec_10 => dec_10,
              dec_100 => dec_100,
              dec_1000 => dec_1000,
              an => an,
              cat0 => cat0,
              cat1 => cat1,
              cat2 => cat2,
              cat3 => cat3,
              cat4 => cat4,
              cat5 => cat5,
              cat6 => cat6
      );

fsm_sensor :process(clk, reset)
begin
  if (reset = '1') then
    state <= meas;
    start <= '0';
  elsif rising_edge(clk) then
    case state is
```

```
        when meas =>
          start <= '1';
          if enable = '0' then
            state <= hold;
          end if;
        when hold =>
          start <= measure;
          if enable = '1' then
            state <= meas;
          end if;
      end case;
    end if;
  end process;

  clk_div:process(clk, reset)
  begin
    if (reset = '1') then
      div_clock <= (others => '0');
    elsif rising_edge(clk) then
      div_clock <= div_clock + 1;
    end if;
  end process;

  distance_2 <= "00" & distance;

  buzzer <= not div_clock(26) when (distance > "00001001011" and distance <
"00001100100") else
         not div_clock(25) when (distance > "00000110010" and distance <
"00001001011") else
         not div_clock(24) when (distance > "00000011001" and distance <
"00000110010") else
```

```
            not div_clock(23) when (distance > "00000001010" and distance < "00000011001") else
            '1' when (distance < "00000001010") else
            '0';

end Behavioral;
```

sensor_cont.vhd

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

entity sensor_cont is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           start : in  STD_LOGIC;
           echo : in  STD_LOGIC;
           trigger : out  STD_LOGIC;
           data_valid : out  STD_LOGIC;
           distance : out  STD_LOGIC_VECTOR (10 downto 0));
end sensor_cont;

architecture Behavioral of sensor_cont is

   type state_type is (s0,s1,s2);
   signal state: state_type;
   signal count: unsigned(10 downto 0);
   signal clk_2: std_logic;
   signal echo_2: unsigned(10 downto 0);

begin
```

```
p_clk:process(clk, reset)
begin
  if (reset = '1') then
    count <= (others => '0');
  elsif rising_edge(clk) then
    if count < 1450 then
      count <= count + 1;
    else
      count <= (others => '0');
    end if;
  end if;
end process;
clk_2 <= count(10);

fsm_sensor_2 :process(clk_2, reset)
begin
  if (reset = '1') then
    state <= s0;
    echo_2 <= (others => '0');
    data_valid <= '0';
    distance <= (others => '0');
    trigger <= '0';
  elsif rising_edge(clk_2) then
    case state is
      when s0 =>          -- waits for start to activate trigger
        data_valid <= '0';
        echo_2 <= (others => '0'); -- restarts the count echo
        if start = '1' then
          trigger <= '1';
          state <= s1;
```

```
               end if;

            when s1 =>          -- trigger goes down after 10 microseconds
               trigger <= '0';
               if echo = '1' then
                  state <= s2;
               end if;

            when s2 =>
               if echo = '0' then
                  distance <= std_logic_vector(echo_2/4);
                  data_valid <= '1';
                  state <= s0;
               else
                  if echo_2 < 1600 then
                     echo_2 <= echo_2 + 1;
                  end if;
               end if;
         end case;
      end if;
   end process;

end Behavioral;
```

bin_to_dec.vhd

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

entity bin_to_dec is
    Port ( clk : in  std_logic;
        reset : in  std_logic;
        b_in : in  std_logic_vector (12 downto 0);
        dec_1 : out  std_logic_vector (3 downto 0);
        dec_10 : out  std_logic_vector (3 downto 0);
        dec_100 : out  std_logic_vector (3 downto 0);
        dec_1000 : out  std_logic_vector (3 downto 0));
end bin_to_dec;

architecture Behavioral of bin_to_dec is

  signal dec_1_i, dec_10_i, dec_100_i, dec_1000_i: unsigned(3 downto 0);
  signal b_in_reg: std_logic_vector(12 downto 0);
  signal bcd4: unsigned(15 downto 0);
  type state_type is (load, replace, compr, ready);
  signal comp_u, comp_d, comp_c, comp_m: std_logic;
  signal sum: std_logic;

begin

  conversion :process(clk, reset)

    variable i: integer range 0 to b_in'high;
    variable state: state_type;

  begin
    if reset = '1' then
      bcd4 <= (others => '0');
      state := load;
    elsif rising_edge(clk) then
      case state is
```

```
when load =>
   b_in_reg <= b_in;
   bcd4 <= (others => '0');
   i := b_in'high;
   state := replace;
 when replace =>
   bcd4(15 downto 1) <= bcd4(14 downto 0);
   bcd4(0) <= std_logic(b_in_reg(12));
   b_in_reg <= b_in_reg(11 downto 0) & '0';
   if i > 0 then
      i := i - 1;
      state := compr;
   else
      state := ready;
   end if;
 when compr =>
   if (sum = '1') then
     if (comp_u = '1') then
        bcd4( 3 downto  0) <= bcd4( 3 downto  0) + 3;
     end if;
     if (comp_d = '1') then
        bcd4( 7 downto  4) <= bcd4( 7 downto  4) + 3;
     end if;
     if (comp_c = '1') then
        bcd4(11 downto  8) <= bcd4(11 downto  8) + 3;
     end if;
     if (comp_m = '1') then
        bcd4 (15 downto 12) <= bcd4 (15 downto 12) + 3;
     end if;
     state := replace;
```

```
          else
            bcd4(15 downto 1) <= bcd4(14 downto 0);
            bcd4(0) <= std_logic(b_in_reg(12));
            b_in_reg <= b_in_reg(11 downto 0) & '0';
            if i > 0 then
              i := i - 1;
              state := compr;
            else
              state := ready;
            end if;
          end if;
        when ready =>
          dec_1  <= std_logic_vector(bcd4( 3 downto  0));
          dec_10 <= std_logic_vector(bcd4( 7 downto  4));
          dec_100 <= std_logic_vector(bcd4(11 downto  8));
          dec_1000 <= std_logic_vector(bcd4(15 downto 12));
          state := load;
      end case;
    end if;
  end process;

  comp_u <= '1' when bcd4( 3 downto  0) > 4 else '0';
  comp_d <= '1' when bcd4( 7 downto  4) > 4 else '0';
  comp_c <= '1' when bcd4(11 downto  8) > 4 else '0';
  comp_m <= '1' when bcd4(15 downto 12) > 4 else '0';

  sum <= comp_u OR comp_d OR comp_c OR comp_m;

end Behavioral;
```

seven_segment.vhd

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

entity seven_segment is
    Port ( clk : in  std_logic;
           reset : in  std_logic;
           dec_1 : in  std_logic_vector (3 downto 0);
           dec_10 : in  std_logic_vector (3 downto 0);
           dec_100 : in  std_logic_vector (3 downto 0);
           dec_1000 : in  std_logic_vector (3 downto 0);
           an : out  std_logic_vector (3 downto 0);
           cat0 : out  std_logic;
           cat1 : out  std_logic;
           cat2 : out  std_logic;
           cat3 : out  std_logic;
           cat4 : out  std_logic;
           cat5 : out  std_logic;
           cat6 : out  std_logic);
end seven_segment;

architecture Behavioral of seven_segment is
   signal count : unsigned (18 downto 0);
   signal seg_disp : std_logic_vector (6 downto 0);
   signal selector : std_logic_vector (3 downto 0);

begin


   count_seg :process(clk, reset)
   begin
```

```
  if (reset = '1') then
    count <= (others => '0');
  elsif rising_edge(clk) then
    count <= count + 1;
  end if;
end process;

with count(18 downto 17) & reset select
 an <= "1110" when "000",  -- 0
     "1101" when "010",  -- 1
     "1011" when "100",  -- 2
     "0111" when "110",  -- 3
     "1111" when others;  -- turned off

with count(18 downto 17) select
 selector <=  dec_1 when "00",
      dec_10 when "01",
      dec_100 when "10",
      dec_1000 when others;--

with selector select
 seg_disp <= "0000001" when "0000",  -- 0
       "1001111" when "0001",  -- 1
       "0010010" when "0010",  -- 2
       "0000110" when "0011",  -- 3
       "1001100" when "0100",  -- 4
       "0100100" when "0101",  -- 5
       "0100000" when "0110",  -- 6
       "0001111" when "0111",  -- 7
       "0000000" when "1000",  -- 8
```

```
            "0000100" when "1001",  -- 9
            "1111111" when others;  -- turned off

        cat0 <= seg_disp(6);
        cat1 <= seg_disp(5);
        cat2 <= seg_disp(4);
        cat3 <= seg_disp(3);
        cat4 <= seg_disp(2);
        cat5 <= seg_disp(1);
        cat6 <= seg_disp(0);

end Behavioral;
```