EEE 102 Final Project

M. Tuna Alatan

May 2024

Video

The demonstration video of the experiment can be accessed through the following link: `https://youtu.be/-ABMD4VGhRw`.

1 Purpose

The purpose of this project is to design and implement a color tracking camera using the BASYS3 FPGA board and VHDL as our hardware description language.

2 Design Specifications

The program's objective is to take the angle input from the PC and use the data to rotate the servo motor and display it on the seven segment displays.

The program has the following inputs and outputs:

- `clk_in (std_logic)` : the main input clock of the board.
- `uart_rx_in (std_logic)` : the input data received from the COM port of the PC.
- `rst_in (std_logic)` : the input reset switch (active-high).
- `set_in (std_logic)` : the input set switch, when high sets the angle to 90° (active-high).
- `set_out (std_logic)` : the output signal that turns on a LED if `set_in` is high.
- `rx_done_tick_out (std_logic)` : the output signal that turns on a LED if the data is received properly.
- `servo_out (std_logic)` : the output signal that controls the rotation of the servo

- `seven_seg_out (std_logic_vector(7 downto 0))` : the output digit data for the seven segment display.
- `anodes_out (std_logic_vector(3 downto 0))` : the output index data for which seven segment display to be active.

There is a counter and a multiplexer used to control the timing of the `anode_out` signal. Three multiplexers control which digit of the angle data is to be displayed; these multiplexers are driven by the `anode_out` signal as the select signal. Additionally, there is a multiplexer that selects between the angle data gathered from the UART and the set angle (90°), the multiplexer uses the `set_in` signal as the select signal.

The `seven_seg_out` and the `anodes_out` data are stored in registers and the `servo_out` signal is stored in a D-flip flop. The other outputs are outputs of combinational circuits so they are connected directly.

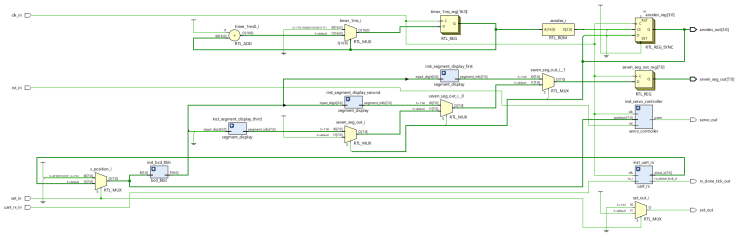


Figure 1: The RLT schematic of the whole program.

There are four submodules that are used in this design.

- `servo_controller`
- `uart_rx`
- `bcd_8bit`
- `segment_display`

2.1 servo_controller

This submodule creates the PWM (pulse width modulation) signal that controls the rotation of the servo. It takes a position data as input and outputs a PWM signal that will rotate the servo according to the position input. We send the PWM signal to the one of the Pmod pins of the BASYS3 board.

```
component servo_controller is
        generic (
                clk_freq                : real := 100
                    _000_000.0;
                pulse_freq              : real := 50.0;
                min_pulse_us    : real:= 500.0;
                max_pulse_us    : real:= 2500.0;
                step_count              : positive := 128
        );
        Port (
                clk                     : in std_logic;
                rst                     : in std_logic;
                position        : in integer range 0 to
        step_count - 1;

                pwm : out std_logic
        );
end component;
```

PWM signal is a pulse width modified signal which means that the duration of the signal being high is timed according to the users need.

For servo motors the signal should be sent every 20ms and for the servo that we use in this project the minimum pulse width must be 0.5ms and the maximum width must be 2.5ms. When the signal has the minimum pulse width it will rotate 0°, and when it has the maximum width it will rotate 180°.

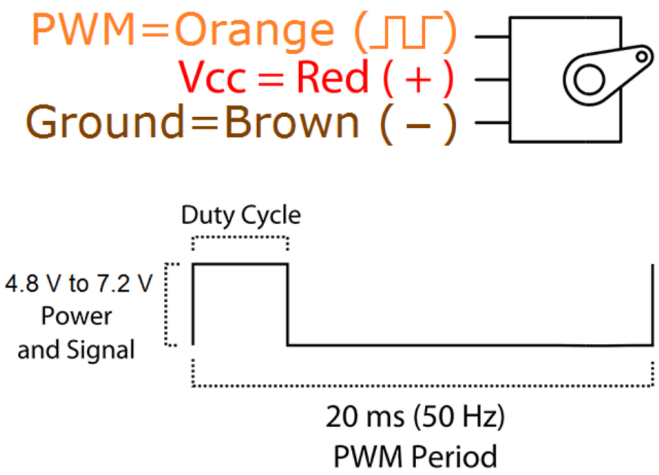


Figure 2: The servo inputs and the pwm signal diagram.

The `position` signal is determines how much should the servo rotate. It is a value ranging from 0 to `STEP_COUNT`(determines how many division should 180° have). This value is received from the UART device. More information about servo motors and using VHDL to control it can be found in [1].

2.2 `uart_rx`

This submodule receives data from the UART device, in this case from the PC.

```
component uart_rx is
        generic (
                c_clkfreq                       : integer := 100
                    _000_000;
                c_baudrate                      : integer := 115
                    _200
        );
        port (
                clk                             : in std_logic;
                rx_i                            : in std_logic;
                dout_o                          : out
                    std_logic_vector (7 downto 0);
                rx_done_tick_o    : out std_logic
);
end component;
```

UART communication protocol is an asynchronous serial protocol which means that the operations are independent of the clock and the data is sent bit by bit. The `c_baudrate` is the constant that hold the value for the rate in which

the date is going to be received. In this case it is 115200 so each for each bit to be received the machine will wait for `c_clkfreq/c_buadrate` clock cycles.

The submodule works a FSM where there are four states: `S_IDLE, S_START, S_DATA, S_STOP`. When there is no communication the machine is in the `S_IDLE` state until the transmitter changes the signal from high to low.

When the transmitter starts the data transmission the machine is in the `S_DATA` state and the receiver will start to read each bit one by one and store it in a shift register. In this code, the data sent by the PC is `rx_i` and the shift register that hold the data is `dout_o`. The machine will return to the state `S_IDLE` when the transmission is over. Figure 3 can be used to better understand the timing of the communication.

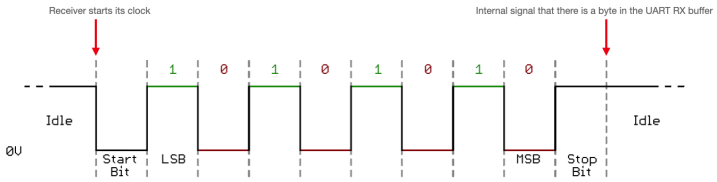


Figure 3: Timing diagram of an UART receiver receiving 8-bit data.[2]

The UART receiver code is adopted from [3].

2.3 bcd_8bit

This submodules converts each digit of the 8-bit angle data into to binary. The binary representation of the digits then send to the seven segment display.

```
component bcd_8bit is
        Port (
                B: in std_logic_vector (7 downto 0);
                P: out std_logic_vector (9 downto 0)
        );
end component;
```

`B` is the 8-bit integer to be converted, and `P` is converted 10-bit integer. The bits 9-8 are the hundreds, the bits 7-4 are tens and the bits 3-0 are ones, Figure 4. The submodule uses multiple adders to calculate this value.

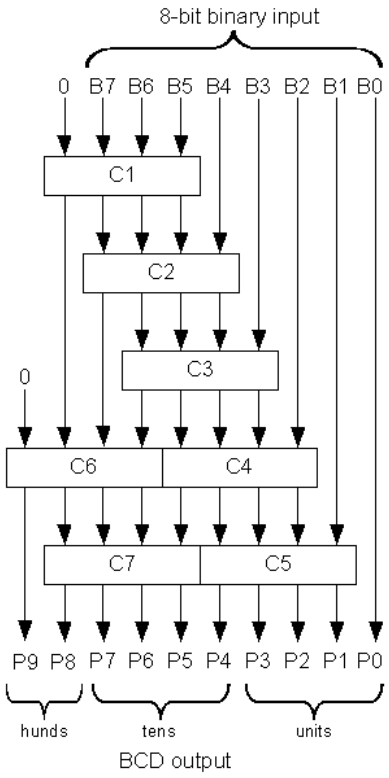


Figure 4: 8-bit binary to BCD converter schematic.

2.4 segment_display

This submodule works as decoder for the seven-segment display. It takes the binary representation of the digit and outputs the corresponding seven-segment display output.

```
component segment_display is
        Port (
            input_digit : in std_logic_vector (3 downto
                0);
            segment_info: out std_logic_vector (7 downto
                0)
        );
end component;
```

The `segment_info` is 8-bit integer, which holds the information for which cathode outputs of the seven-segment should be high. Each ten digit has a specific cathode information and this module converts the each digit to its corresponding cathode information. The BASYS3 cathode pin layout can be seen in Figure 5.

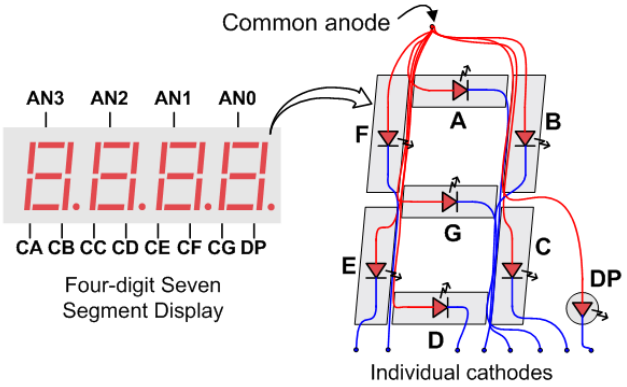


Figure 5: The BASYS3 seven-segment display cathode layout.

When using the seven-segment displays we can only display one at a time, it is not possible to turn all of the four displays at the same time. In order to overcome this we can create an illusion by changing the active display so frequently that for the human eye, it looks like they are turned on at the same time. In this project we changed the display every 1ms and the following code snippet demonstrates how it is implemented in the top module.

```
anode_process : process (clk_in) begin
        if (rising_edge(clk_in)) then

                anodes(3) <= '1';

                if (timer_1ms = timer_1ms_lim - 1) then
                        timer_1ms <= 0;
                        anodes(2 downto 1) <= anodes(1
                            downto 0);
                        anodes(0) <= anodes(2);
                else
                        timer_1ms <= timer_1ms + 1;
                end if;

        end if;
end process;
```

2.5 Python Script

The frame data from the webcam is connected to the PC and the computer is making the required computations. We wrote a python program that takes the frame data and outputs the corresponding angle data to the serial ports of the PC. The python code and all of the VHDL code will be in the **Appendices** section.

3 Methodology

In this project we used the following components:

- BASYS3 FPGA board
- USB webcam
- MG996R servo motor
- Level converter (3.3V to 5V)
- 5V power supply

First, we start off by mounting the webcam onto the servo. As can be seen in Figure 6 we achieve this by wrapping a rubber band around the webcam base and one of the provided servo headers.

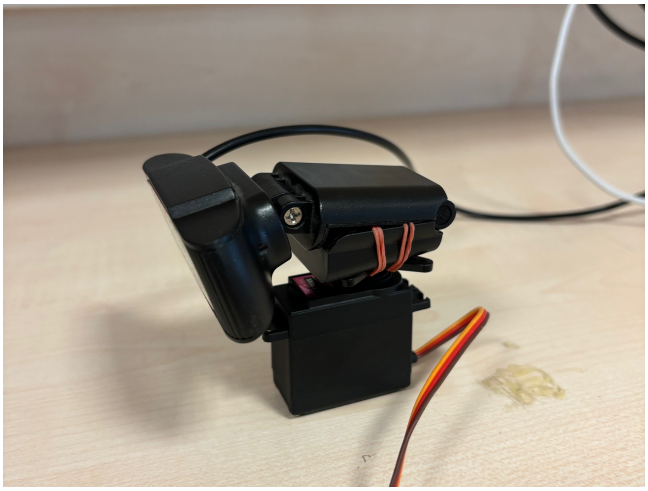


Figure 6: The webcam and servo setup.

Then we continue by constructing our circuit. We connect the `servo_out` output signal of the program to the low-level input of the level converter and the servo's **PWM** pin to the high-level output of the level converter. For the low-level voltage we use the BASYS3 board's 3.3V output and its ground; for the high-level voltage we use the 5V power supply. When everything is connected properly our circuit should look like in Figure 7.

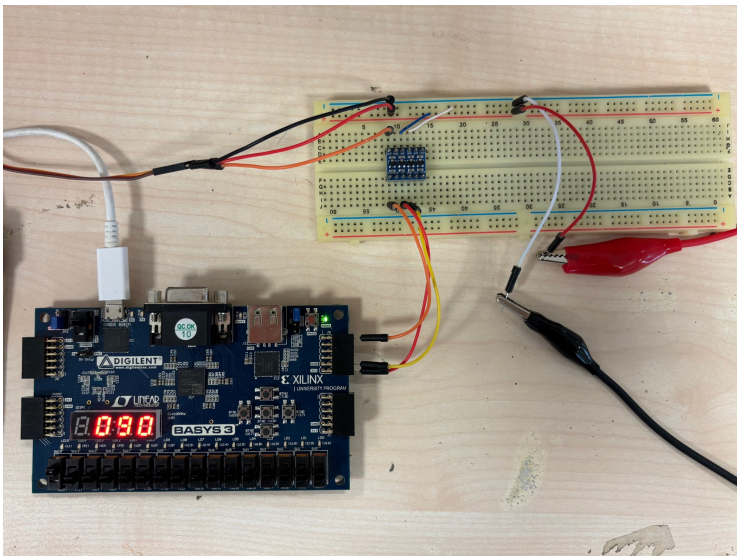
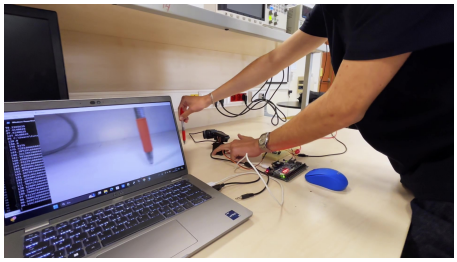


Figure 7: The circuit implemented on a breadboard.

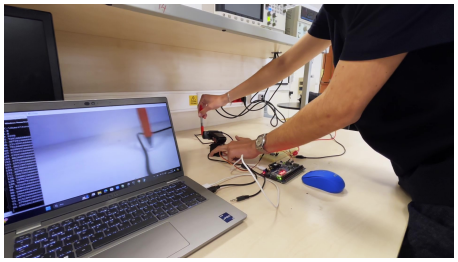
Finally, we should run the python script because right now even if we send our program to the board it will wait for an 8-bit integer to be sent.

4 Results

When we run our program we can observe that the camera is tracking the red pen. We can use Figure 8 to check whether the program is working properly.



(a) The webcam video at and its position at time t.



(b) The webcam video at and its position at time $t + 2s$.

Figure 8: Comparison of two frames that are 2 seconds apart.

As can be observed from Figure 8a the pen is at the edge of the frame and the pen is to the left of the black cable. Moreover, in Figure 8b the pen is now at the right side of the black cable and now we can see the cable in the video

even though we could not in Figure 8a. This shows that our camera is tracking the pen properly.

Also, in Figure 7 we can see that the set signal is high (the outer left switch is on) and the seven-segment display is displaying 90° as well which also proves that the seven-segment display is working properly.

5 Conclusion

The aim of the project was to create a color tracking camera using BASYS3 board and VHDL as our hardware description language and we achieved our aim. Additionally, we learned how to control servo motors, use a seven-segment display, and receive data from a PC using UART communication protocol.

When looking at the results we can see that the camera can track the color red but when its whole motion is observed we can see that the motion is a bit choppy. The camera can sometimes overshoot or cannot keep up with the pen. First, to overcome this issue we added sampling for the angle data. Now, the data is sent only after a certain amount of samples are gathered and the average of the certain number of samples is the new angle data that is sent. This addition solved the choppiness to some extent and smoothed out the motion but there was still a considerable amount of overshooting or lagging To overcome these issues a PID controller can be used but it has to be implemented on the python script and with that, the project can start to diverge from a VHDL project.

Also, this project could have been done using a CMOS camera and directly connecting to the FPGA board's Pmod pins but implementing the same program in the same time interval with the same VHDL knowledge that we have would have been really challenging.

References

[1] J. J. Jensen, "Rc servo controller using pwm from an fpga pin," Aug 2023. [Online]. Available: https://vhdlwhiz.com/rc-servo-controller-using-pwm/

[2] V. H. Adams, "Universal asynchronous receiver transmitter (uart)," Mar 2021. [Online]. Available: https://vanhunteradams.com/Protocols/UART/UART.html

[3] M. B. Aykenar, "Vhdl ile fpga programlama." [Online]. Available: https://github.com/mbaykenar/apis_anatolia/tree/main/VHDL_ile_FPGA_PROGRAMLAMA

5.1 Appendices

top_module_final.vhd

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
    instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity top_module_final is
        Port (
                clk_in                          : in
                    std_logic;
                uart_rx_in                      : in
                    std_logic;
                rst_in                          : in
                    std_logic;
                set_in                          : in
                    std_logic;


                set_out                         : out
                    std_logic;
                rx_done_tick_out        : out std_logic;
                servo_out                       : out
                    std_logic;

                seven_seg_out           : out
                    std_logic_vector(7 downto 0);
                anodes_out                      : out
                    std_logic_vector(3 downto 0)
        );
end top_module_final;

architecture Behavioral of top_module_final is

component servo_controller is
        generic (
```

```
                clk_freq                : real := 100
                    _000_000.0;
                pulse_freq              : real := 50.0;
                min_pulse_us    : real:= 500.0;
                max_pulse_us    : real:= 2500.0;
                step_count              : positive := 128
        );
        Port (
                clk                     : in std_logic;
                rst                     : in std_logic;
                position        : in integer range 0 to
                    step_count - 1;

                pwm : out std_logic
        );
end component;

component uart_rx is
        generic (
                c_clkfreq               : integer := 100
                    _000_000;
                c_baudrate              : integer := 115
                    _200
        );
        port (
                clk                             : in
                    std_logic;
                rx_i                    : in std_logic;
                dout_o                  : out
                    std_logic_vector (7 downto 0);
                rx_done_tick_o  : out std_logic
);
end component;

component bcd_8bit is
        Port (
                B: in std_logic_vector (7 downto 0);
                P: out std_logic_vector (9 downto 0)
        );
end component;

component segment_display is
        Port (
                input_digit : in std_logic_vector (3
                    downto 0);
```

```
                segment_info : out std_logic_vector (7
                    downto 0)
        );
end component;


constant STEP_COUNT : integer := 180;
signal s_position       : integer range 0 to STEP_COUNT -
    1;
signal s_data_out       : std_logic_vector(7 downto 0) :=
    (others => '0');
signal s_rst_uart       : std_logic := '0';

signal anodes : std_logic_vector (3 downto 0) := "1110" ;

constant clk_freq               : integer := 100_000_000;
constant timer_1ms_lim  : integer := clk_freq / 1000;
signal timer_1ms                : integer range 0 to
    timer_1ms_lim := 0;

signal s_bdc_conv : std_logic_vector(9 downto 0) := (
    others => '0');

signal first_dig_info           : std_logic_vector(3
    downto 0) := (others => '0');
signal second_dig_info          : std_logic_vector(3
    downto 0) := (others => '0');
signal third_dig_info           : std_logic_vector(3
    downto 0) := (others => '0');

signal first_seg_signal         : std_logic_vector(7
    downto 0) := (others => '0');
signal second_seg_signal        : std_logic_vector(7
    downto 0) := (others => '0');
signal third_seg_signal         : std_logic_vector(7
    downto 0) := (others => '0');


begin

inst_servo_controller : servo_controller
        generic map(
                clk_freq                => 100_000_000.0,
                pulse_freq              => 50.0,
                min_pulse_us    => 500.0,
                max_pulse_us    => 2500.0,
```

```
                step_count              => STEP_COUNT
        )
        Port map(
                clk                     => clk_in,
                rst                     => rst_in,
                position        => s_position,

                pwm => servo_out
        );

inst_uart_rx : uart_rx
        generic map(
                        c_clkfreq               => 100
                            _000_000,
                        c_baudrate              => 115
                            _200
                )

        port map(
                        clk
                           => clk_in,
                        rx_i                    =>
                            uart_rx_in,
                        dout_o                  =>
                            s_data_out,
                        rx_done_tick_o  =>
                            rx_done_tick_out
        );

inst_bcd_8bit : bcd_8bit
        port map(
                B => std_logic_vector(to_unsigned(
                    s_position, 8)),
                P => s_bdc_conv
        );

inst_segment_display_first : segment_display
        port map(
                input_digit             => first_dig_info
                    ,
                segment_info    => first_seg_signal
        );

inst_segment_display_second : segment_display
        port map(
```

```
                input_digit             =>
                    second_dig_info,
                segment_info    => second_seg_signal
        );

inst_segment_display_third : segment_display
        port map(
                input_digit             => third_dig_info
                    ,
                segment_info    => third_seg_signal
        );

p_main : process begin
        first_dig_info  <= s_bdc_conv(3 downto 0);
        second_dig_info <= s_bdc_conv(7 downto 4);
        third_dig_info  <= "00" & s_bdc_conv(9 downto 8);

        if (set_in = '1') then
                s_position <= 90;
                set_out <= '0';
        else
                s_position <= to_integer(unsigned(
                    s_data_out));
                set_out <= '1';
        end if;
end process p_main;

anode_process : process (clk_in) begin
        if (rising_edge(clk_in)) then

                anodes(3) <= '1';

                if (timer_1ms = timer_1ms_lim - 1) then
                        timer_1ms <= 0;
                        anodes(2 downto 1) <= anodes(1
                            downto 0);
                        anodes(0) <= anodes(2);
                else
                        timer_1ms <= timer_1ms + 1;
                end if;

        end if;
end process;

cathode_process : process (clk_in) begin
        if (rising_edge(clk_in)) then
```

```
                if (anodes(0) = '0') then
                        seven_seg_out <= first_seg_signal
                            ;
                elsif (anodes(1) = '0') then
                        seven_seg_out <=
                            second_seg_signal;
                elsif (anodes(2) = '0') then
                        seven_seg_out <= third_seg_signal
                            ;
                else
                        seven_seg_out <= (others => '1');
                end if;
        end if;
end process;

anodes_out <= anodes;

end Behavioral;
```

servo_controller.vhd

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
    instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
use ieee.math_real.round;

entity servo_controller is
        generic (
                clk_freq : real := 100_000_000.0;
                pulse_freq : real := 50.0;
                min_pulse_us : real:= 500.0;
                max_pulse_us : real:= 2500.0;
                step_count : positive := 128
        );
        Port (
                clk : in std_logic;
                rst : in std_logic;
                position : in integer range 0 to
                    step_count - 1;

                pwm : out std_logic
        );
end servo_controller;

architecture Behavioral of servo_controller is
        function cycles_per_us (us_count : real) return
            integer is
        begin
                return integer(round((clk_freq / 1
                    _000_000.0) * us_count));
        end function;

        constant min_count : integer := cycles_per_us(
            min_pulse_us);
        constant max_count : integer := cycles_per_us(
            max_pulse_us);
```

```
        constant min_max_range_us : real := max_pulse_us
            - min_pulse_us;
        constant step_us : real := min_max_range_us /
            real(step_count - 1);
        constant cycles_per_step : positive :=
            cycles_per_us(step_us);

        constant counter_max : integer := integer(round(
            clk_freq / pulse_freq)) - 1;
        signal counter : integer range 0 to counter_max
            := 0;

        signal duty_cycle : integer range 0 to max_count
            := 0;

begin
        PWM_PROC : process(clk)
        begin
                if rising_edge(clk) then
                        if rst = '1' then
                                pwm <= '0';
                                counter <= 0;
                        end if;

                        if (counter < counter_max) then
                                if (counter < duty_cycle)
                                    then
                                        pwm <= '1';
                                else
                                        pwm <= '0';
                                end if;
                                counter <= counter + 1;
                        else
                                counter <= 0;
                        end if;
                end if;
        end process;

        DUTY_CYCLE_PROC : process(clk)
        begin
                if rising_edge(clk) then
                        if rst = '1' then
                                duty_cycle <= min_count;
                        else
```

```
                                duty_cycle <= position *
                                    cycles_per_step +
                                    min_count;
                            end if;
                        end if;
                end process;
end Behavioral;
```

segment_display.vhd

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
    instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity segment_display is
        Port (
                input_digit : in std_logic_vector (3
                    downto 0);
                segment_info : out std_logic_vector (7
                    downto 0)
        );
end segment_display;

architecture Behavioral of segment_display is

begin

process (input_digit) begin
        case input_digit is
                when "0000" =>
                        segment_info <= "00000011";

                when "0001" =>
                        segment_info <= "10011111";

                when "0010" =>
                        segment_info <= "00100101";

                when "0011" =>
                        segment_info <= "00001101";

                when "0100" =>
                        segment_info <= "10011001";
```

```
                when "0101" =>
                        segment_info <= "01001001";

                when "0110" =>
                        segment_info <= "01000001";

                when "0111" =>
                        segment_info <= "00011111";

                when "1000" =>
                        segment_info <= "00000001";

                when "1001" =>
                        segment_info <= "00001001";

                when others =>
                        segment_info <= "11111111";

        end case;
end process;

end Behavioral;
```

bcd_8bit.vhd

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
    instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity bcd_8bit is
        Port (
                B: in std_logic_vector (7 downto 0);
                P: out std_logic_vector (9 downto 0)
        );
end bcd_8bit;

architecture Behavioral of bcd_8bit is
begin

bcd1: process(B)
        variable z: std_logic_vector (17 downto 0);
        begin
                for i in 0 to 17 loop
                        z(i) := '0';
                end loop;
                z(10 downto 3) := B;

                for i in 0 to 4 loop
                        if z(11 downto 8) > 4 then
                                z(11 downto 8) := z(11
                                    downto 8) + 3;
                        end if;
                        if z(15 downto 12) > 4 then
                                z(15 downto 12) := z(15
                                    downto 12) + 3;
                        end if;
                        z(17 downto 1) := z(16 downto 0);
```

```
                end loop;

                P <= z(17 downto 8);

        end process bcd1;
end Behavioral;
```

uart_rx.vhd

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


entity uart_rx is
generic (
c_clkfreq               : integer := 100_000_000;
c_baudrate              : integer := 115_200
);
port (
clk                             : in std_logic;
rx_i                    : in std_logic;
dout_o                  : out std_logic_vector (7 downto
    0);
rx_done_tick_o   : out std_logic
);
end uart_rx;

architecture Behavioral of uart_rx is

constant c_bittimerlim   : integer := c_clkfreq/c_baudrate
    ;

type states is (S_IDLE, S_START, S_DATA, S_STOP);
signal state : states := S_IDLE;

signal bittimer : integer range 0 to c_bittimerlim := 0;
signal bitcntr   : integer range 0 to 7 := 0;
signal shreg     : std_logic_vector (7 downto 0) := (
    others => '0');

begin

P_MAIN : process (clk) begin
if (rising_edge(clk)) then

        case state is

                when S_IDLE =>

                        rx_done_tick_o  <= '0';
                        bittimer                <= 0;

                        if (rx_i = '0') then
```

```
                state   <= S_START;
        end if;

when S_START =>

        if (bittimer = c_bittimerlim/2-1)
            then
                state           <= S_DATA
                    ;
                bittimer        <= 0;
        else
                bittimer        <=
                    bittimer + 1;
        end if;

when S_DATA =>

        if (bittimer = c_bittimerlim-1)
            then
                if (bitcntr = 7) then
                        state   <= S_STOP
                            ;
                        bitcntr <= 0;
                else
                        bitcntr <=
                            bitcntr + 1;
                end if;
                shreg           <= rx_i &
                    (shreg(7 downto 1));
                bittimer        <= 0;
        else
                bittimer        <=
                    bittimer + 1;
        end if;

when S_STOP =>

        if (bittimer = c_bittimerlim-1)
            then
                state
                    <= S_IDLE;
                bittimer
                    <= 0;
                rx_done_tick_o  <= '1';
        else
                bittimer        <=
```

```
                                    bittimer + 1;
                            end if;

        end case;

end if;
end process P_MAIN;

dout_o  <= shreg;

end Behavioral;
```

color_tracker.py

```
import cv2
import numpy as np
import serial

def send_integer_to_serial(integer_value, port_name='COM5
    ', baud_rate=115200):
    try:
        # Open serial port
        ser = serial.Serial(port_name, baud_rate)

        # Convert integer to bytes
        byte_value = integer_value.to_bytes(1, byteorder=
            'big')

        # Send byte over serial
        ser.write(byte_value)

        # Close serial port
        ser.close()

    except serial.SerialException as e:
        print()

def track_red():
    cap = cv2.VideoCapture(1)
    n_samples = 0
    total_angle = 0
    fov = 60
    prev_ang = 90
    total_samples = 6

    while True:
        ret, frame = cap.read()
        #frame = cv2.flip(frame, 1)

        if not ret:
            break

        # Convert BGR to HSV
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Define range of red color in HSV
        lower_red = np.array([0, 100, 100])
        upper_red = np.array([10, 255, 255])
```

```
            # Threshold the HSV image to get only red colors
            mask = cv2.inRange(hsv, lower_red, upper_red)

            vector = np.sum(mask, axis=0)
            #print('Prev angle:', prev_ang)
            angle = min((((len(vector) - (np.argmax(vector)))
                / len(vector)) * fov) + (prev_ang - fov / 2),
                180)



            if n_samples < total_samples:
                n_samples += 1
                total_angle += angle
            else:
                av_angle = max(total_angle / total_samples,
                    0)
                prev_ang = av_angle
                print(av_angle)
                send_integer_to_serial(int(av_angle))
                n_samples = 0
                total_angle = 0


            # Display the resulting frame
            cv2.imshow('Frame', frame)

            # Exit if 'q' is pressed
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        # Release the capture
        cap.release()
        cv2.destroyAllWindows()

if __name__ == "__main__":
    track_red()
```