Emre Akdemir

22302606

EE102-02

EE-102 FINAL PROJECT REPORT: LASER GUIDED TURRET

14/05/2025

Purpose

The aim of this project was to create a laser supported turret system that dynamically follows and targets a specific object using a laser module, two servo motors, and a camera for face detection.

Methodology

One of the objectives of this project is adjusting servo motors according to the x and y coordinates of the centre of the face. The coordinates are calculated using OpenCV libraries of Python. Thus, dynamically, the coordinates of the face on 640x480 resolution camera are sent in byte(8-bit) form to BASYS3 via Micro USB data cable using UART Protocol. In VHDL, a UART module has been used to interpret the coming location data. A clock division module is used to get a 180 kHz clock to make servo sensitive at level of 1 degree. Two servo_pwm modules are used to send the proper pwm signals to the servo motors. First pwm_signal module is used for pan angle and second pwm_signal is used for tilt angle.

Design Specifications

Clk180kHz.vhd

The control signal for the servo motor can be listed as two frequencies: Refresh frequency and pulse width range. The desired sensitivity for turret is up to 180 positions, which also means sensitive at level of 1 degree. Therefore, needed minimum frequency is range of the pulse width range over resolution, which is 180kHZ. Since servo motor's refresh frequency is 20ms, using a clock divider module, BASYS3's internal clock of 100MHz is divided into 180 kHz by adding 1 to the counter, which is limited at 278 ((100Mhz / 180kHz)/2, divided into 2 due to 50% duty cycle) and when clock is reached up to 278, output is set to 1.

Uart_rx.vhd

For the UART Protocol part, a module and submodule are used. Uart_rx_inst module is used to receive 8-bit data. Baud_rate is set to 115200, which is the number of bits received in one second. Clock_freq is the FPGA clock frequency, which is default 100Mhz. Bit_ticks is the number of FPGA clock cycles corresponding to the duration of one UART bit. For 115200 baud rate on 100Mhz, it is approximately 868 cycles per bit. Rx is the serial input line from UART transmitter, rx_data holds the last received byte and rx_valid pulses to "1" for exactly one clock cycle whenever new data is received. Uart Rx is implemented as finite state machine with the states of idle, start, data and stop. Idle is when waiting for start bit. Start is when verifying the start bit validity. Data is when receiving each data bit and stop is when checking the stop bit and signaling data reception. In IDLE state, line is normally idle (rx = "1"). A falling edge (rx= "0") indicates the start bit. In START state, waits half a bit duration to confirm that it is really a start bit and not noise. If still low, confirms valid start bit; else, returns to IDLE (false start). In DATA state, samples the RX line at each bit's midpoint. Stores each bit into shift_reg and moves through all 8 bits. UART sends least significant bit first, thus first received bit goes into shift_reg(0). Finally, at STOP state, waits one full bit duration for stop bit. After confirming stop bit, moves data from shift register to rx_data_reg and pulses rx_valid_reg signal high to indicate new data is ready.

Servo_uart.vhd

Servo_uart_dual is the controller module for two servo motors. It receives two consecutive UART bytes, each representing a servo angle (0-180), and assigns them to servo_angle1 and servo_angle2. There are two states in this module. WAIT_BYTE1 is the first byte state and angle for servo 1(Pan Angle). WAIT_BYTE2 is the second byte state and angle for servo 2(Tilt Angle). Uart_rx submodule is instantiated in this module. Angle1_reg and Angle2_reg stores current servo angles and initialized to 90 degrees. Rx_data_i carries the received UART byte. Rx_valid_i becomes "1" for one clock cycle when a byte is received. In case 1, which is WAIT_BYTE1, if the received value is lower than 180, it

assigns to angle1_reg, otherwise assigns it to 180(servo angle limit). Then moves to the next state, which is WAIT_BYTE2. Same as before, but now updates angle2_reg, then returns to WAIT_BYTE1 for next servo update. Finally, angle1_reg and angle2_reg are drived to the servo_angle1 and servo_angle2 continuosly as outputs.

Servo_pwm.vhd and Servo_pwm22.vhd

There are two servo_pwm modules in the project. Each of them creates a pwm signal for servos. Module takes a desired servo angle (pos = 0 to 180 degrees) and generates a pwm signal with a pulse width between 0.5 ms to 2.5 ms, the standard servo control range. It runs on a clock of 180 kHz, which is used to count time ticks. It outputs a servo signal: “1” when pulse is active and “0” otherwise. Each cycle is 20 ms, therefore pulse width is mapped linearly from 0.5 ms to 2.5 ms as the input pos goes from 0 to 180. For 20ms frame, each full PWM cycle is (180kHz * 20ms) 3600 ticks. Cnt counts from 0 to 3599, pwmi holds the calculated pulse width in ticks and pos_clamped ensures that the angle doesn’t go above 180. Pwmi is adjusted by pos_clamped * 2 + 90, which maps 0-180 as 90-450. 90 ticks is equivalent of 0.5 ms and 450 ticks is equivalent of 2.5ms. Depending on pos value, servo is assigned to “1” as cnt < pwmi, else “0”. So the width of the “1” time in clock ticks is directly proportional to the input angle.

Servo_top.vhd

The servo_top_dual module serves as the top-level controller for operating two servo motors using UART input. It receives two consecutive angle values (0–180) over UART via the onboard USB-UART interface, decodes them using the servo_uart_dual module, and outputs each angle as an 8-bit value. A clk180kHz divider generates a 180 kHz clock from the 100 MHz system clock, suitable for servo PWM timing. These angle values are then fed into two independent PWM generators (servo_pwm and servo_pwm2), which produce corresponding pulse-width signals (servo1 and servo2) to drive each servo with precise control over their positions.

Results

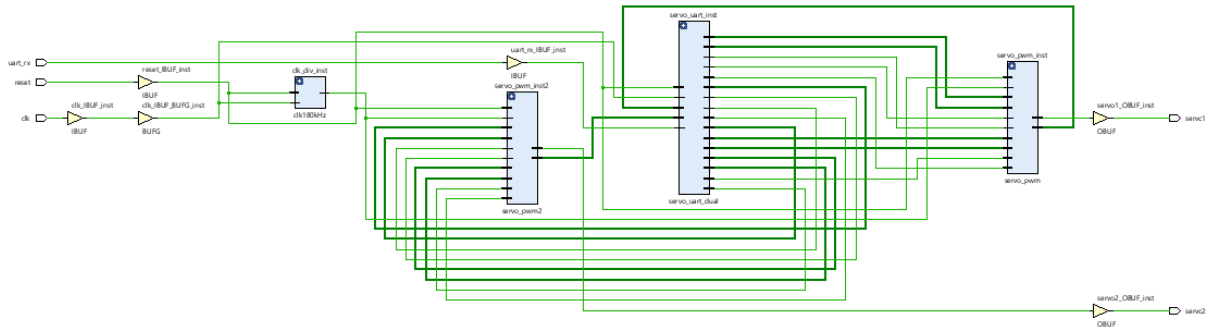

Figure 1: RTL Schematics of the Top Module

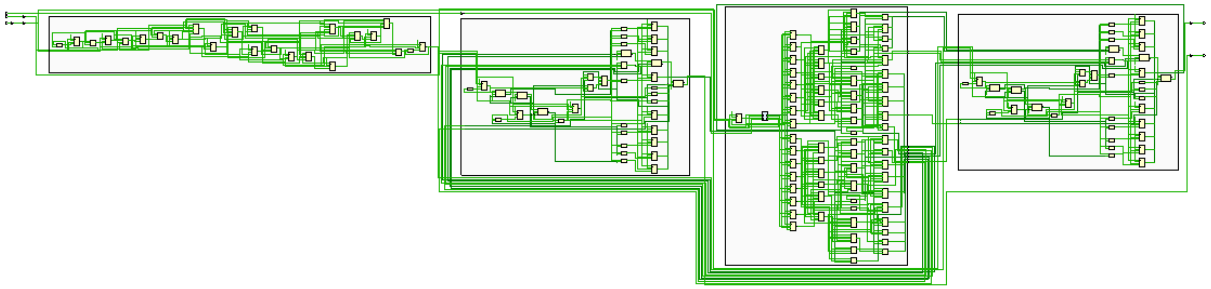

Figure 2: RTL Schematics of the Top Module (Expanded)

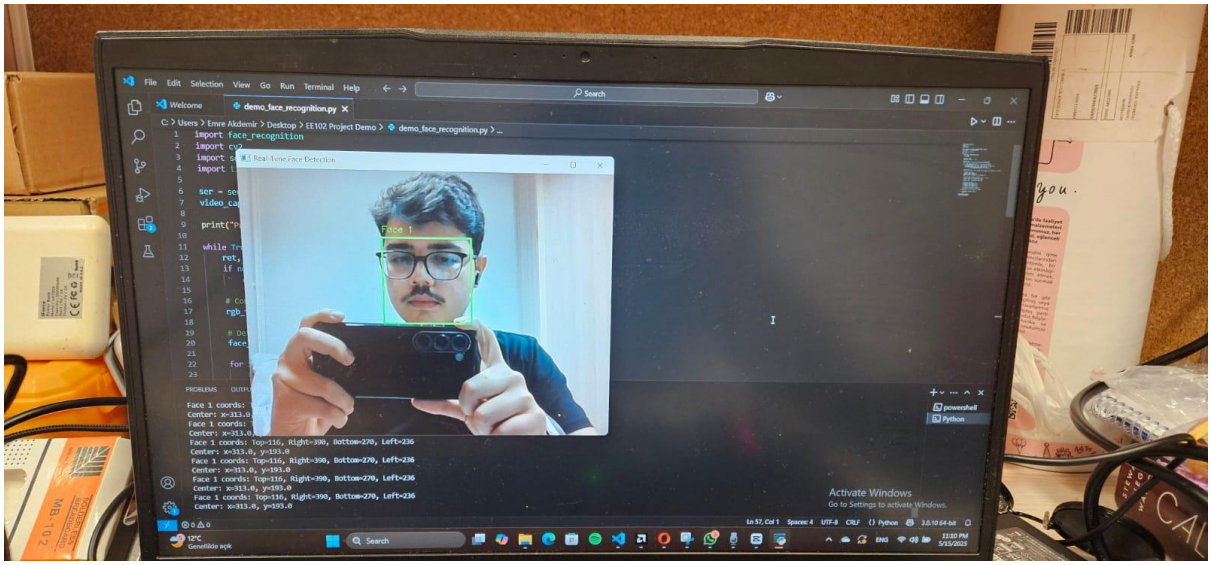

Figure 3: Python Code and Dynamically Tracked Face Frame

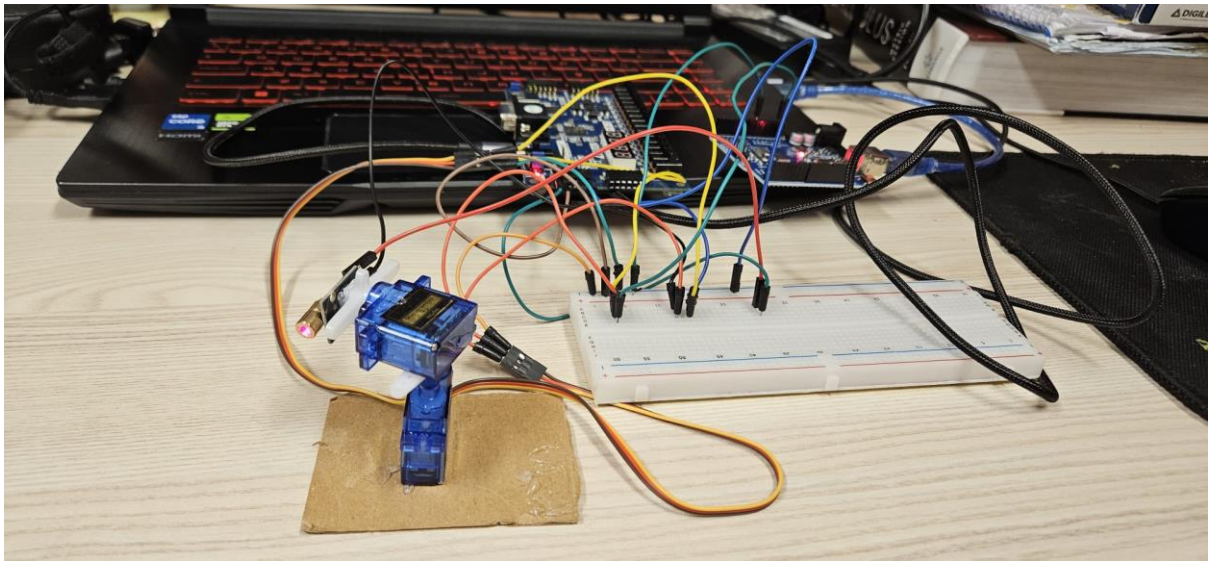

Figure 6: The Turret System

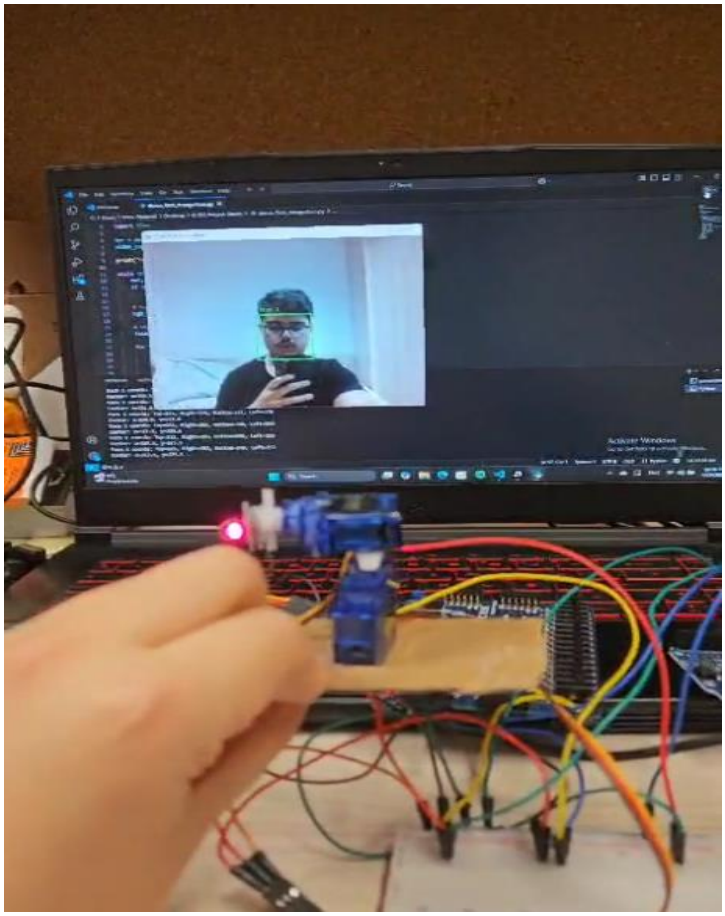

Figure 7: Dynamic Tracking

Conclusion:

This project successfully demonstrated a laser-guided turret system capable of tracking a human face in real time using OpenCV-based Python face detection, UART communication, and precise servo control implemented on a BASYS3 FPGA board. The integration of software and hardware components allowed for dynamic adjustment of the turret's orientation based on the received face coordinates. The system achieved high positional sensitivity thanks to a custom 180 kHz clock signal for PWM generation, enabling 1-degree resolution for both pan and tilt movements. However, one of the most significant challenges encountered during testing was the physical misalignment between the camera and the servo motor axes. Since the camera was mounted on a laptop and not directly aligned with the center of rotation of the turret, the perceived coordinates of the face did not always correspond accurately to the direction in which the turret should point. This discrepancy led to inconsistent tracking behavior, especially during rapid head movements or off-center positioning. While the software and electronic design functioned correctly, this hardware alignment issue highlighted the importance of mechanical calibration. Future improvements could include designing a custom mount that aligns the camera with the turret's center of rotation to enhance tracking precision. Despite this limitation, the project effectively demonstrated fundamental principles of embedded systems design, real-time control, and human-object interaction through autonomous tracking.

Reference:

https://www.codeproject.com/Articles/513169/Servomotor-Control-with-PWM-and-VHDL

https://www.codeproject.com/Articles/443644/Frequency-Divider-with-VHDL

http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf

Code:

Face_recognition.py

```
import face_recognition import cv2 import serial import time

ser = serial.Serial('COM5', baudrate=115200, timeout=1) video_capture =
cv2.VideoCapture(0)

print("Press 'q' to quit.")

while True: ret, frame = video_capture.read() if not ret: continue
```

```
rgb_frame = frame[:, :, ::-1]


face_locations = face_recognition.face_locations(rgb_frame)

for i, (top, right, bottom, left) in enumerate(face_locations):

    cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0),
2)
    cv2.putText(frame, f"Face {i+1}", (left, top - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)


    print(f"Face {i+1} coords: Top={top}, Right={right},
Bottom={bottom}, Left={left}")


    x_coord = (left + right) / 2
    y_coord = (top + bottom) / 2
    print(f"Center: x={x_coord:.1f}, y={y_coord:.1f}")


    scaled_x = 255 - int((x_coord / 640) * 255)

    scaled_y = 255 - int((y_coord / 480) * 255)

    # Ensure 0..255 range
    if scaled_x < 0: scaled_x = 0
    if scaled_x > 255: scaled_x = 255
    if scaled_y < 0: scaled_y = 0
    if scaled_y > 255: scaled_y = 255

    ser.write(bytes([scaled_y, scaled_x]))

cv2.imshow('Real-Time Face Detection', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

video_capture.release() cv2.destroyAllWindows()

Servo_top.vhd

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

entity servo_top_dual is Port (clk : in std_logic; -- 100 MHz reset : in std_logic; uart_rx : in std_logic; -- from onboard USB-UART (pin B18 on Basys3) servo1 : out std_logic; -- to servo #1 servo2 : out std_logic -- to servo #2); end servo_top_dual;

architecture Behavioral of servo_top_dual is

```
component clk180kHz is
    Port (
        clk     : in  std_logic;
        reset   : in  std_logic;
        clk_out : out std_logic
    );
end component;

component servo_uart_dual is
    Port (
        clk          : in  std_logic;
        reset        : in  std_logic;
        rx           : in  std_logic;
        servo_angle1 : out std_logic_vector(7 downto 0);
        servo_angle2 : out std_logic_vector(7 downto 0)
    );
end component;

component servo_pwm is
    Port (
        clk   : IN  STD_LOGIC;
        reset : IN  STD_LOGIC;
        pos   : IN  STD_LOGIC_VECTOR(7 downto 0);
        servo : OUT STD_LOGIC
    );
end component;

component servo_pwm2 is
    Port (
        clk   : IN  STD_LOGIC;
        reset : IN  STD_LOGIC;
        pos2  : IN  STD_LOGIC_VECTOR(7 downto 0);
```

```
        servo2: OUT STD_LOGIC
    );
end component;

signal clk_180khz : std_logic;
signal angle1_sig : std_logic_vector(7 downto 0);
signal angle2_sig : std_logic_vector(7 downto 0);
```

begin

```
clk_div_inst : clk180kHz
    port map(
        clk     => clk,
        reset   => reset,
        clk_out => clk_180khz
    );


servo_uart_inst : servo_uart_dual
    port map(
        clk          => clk,
        reset        => reset,
        rx           => uart_rx,
        servo_angle1 => angle1_sig,
        servo_angle2 => angle2_sig
    );


servo_pwm_inst : servo_pwm
    port map(
        clk   => clk_180khz,
        reset => reset,
        pos   => angle1_sig,
        servo => servo1
    );


servo_pwm_inst2 : servo_pwm2
    port map(
        clk   => clk_180khz,
        reset => reset,
```

```
        pos2  => angle2_sig,
        servo2 => servo2
    );
```

end Behavioral;

Clk180kHz.vhd

library IEEE; use IEEE.STD_LOGIC_1164.ALL;

entity clk180kHz is Port (clk : in STD_LOGIC; -- 100 MHz input reset : in STD_LOGIC; clk_out : out STD_LOGIC -- ~180 kHz output); end clk180kHz;

architecture Behavioral of clk180kHz is signal counter : integer range 0 to 278 := 0; signal clk_reg : std_logic := '0'; begin process(clk, reset) begin if reset = '1' then counter <= 0; clk_reg <= '0'; elsif rising_edge(clk) then if counter = 278 then counter <= 0; clk_reg <= not clk_reg; else counter <= counter + 1; end if; end if; end process;

```
clk_out <= clk_reg;
```

end Behavioral;

Servo_uart.vhd

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

entity servo_uart_dual is Port (clk : in std_logic; -- 100 MHz reset : in std_logic; rx : in std_logic; -- from USB-UART servo_angle1 : out std_logic_vector(7 downto 0); servo_angle2 : out std_logic_vector(7 downto 0)); end servo_uart_dual;

architecture Behavioral of servo_uart_dual is

```
component uart_rx is
    Port (
        clk      : in  std_logic;
        reset    : in  std_logic;
        rx       : in  std_logic;
        rx_data  : out std_logic_vector(7 downto 0);
        rx_valid : out std_logic
```

```
    );
end component;


signal angle1_reg : unsigned(7 downto 0) := to_unsigned(90, 8);
signal angle2_reg : unsigned(7 downto 0) := to_unsigned(90, 8);


signal rx_data_i  : std_logic_vector(7 downto 0);
signal rx_valid_i : std_logic;


type rx_dual_state is (WAIT_BYTE1, WAIT_BYTE2);
signal current_state : rx_dual_state := WAIT_BYTE1;
```

begin

```
uart_rx_inst : uart_rx
    port map(
        clk      => clk,
        reset    => reset,
        rx       => rx,
        rx_data  => rx_data_i,
        rx_valid => rx_valid_i
    );


process(clk, reset)
begin
    if reset = '1' then
        angle1_reg    <= to_unsigned(90, 8);
        angle2_reg    <= to_unsigned(90, 8);
        current_state <= WAIT_BYTE1;

    elsif rising_edge(clk) then
        if rx_valid_i = '1' then
            case current_state is
                when WAIT_BYTE1 =>
                    if unsigned(rx_data_i) <= 180 then
                        angle1_reg <= unsigned(rx_data_i);
                    else
```

```
                        angle1_reg <= to_unsigned(180, 8);
                    end if;
                    current_state <= WAIT_BYTE2;

                when WAIT_BYTE2 =>
                    if unsigned(rx_data_i) <= 180 then
                        angle2_reg <= unsigned(rx_data_i);
                    else
                        angle2_reg <= to_unsigned(180, 8);
                    end if;
                    current_state <= WAIT_BYTE1;
            end case;
        end if;
    end if;
end process;


servo_angle1 <= std_logic_vector(angle1_reg);
servo_angle2 <= std_logic_vector(angle2_reg);
```

end Behavioral;

Uart_rx.vhd

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

entity uart_rx is Port (clk : in std_logic; reset : in std_logic; rx : in std_logic; rx_data : out std_logic_vector(7 downto 0); rx_valid : out std_logic); end uart_rx;

architecture Behavioral of uart_rx is

```
constant BAUD_RATE  : integer := 115200;
constant CLOCK_FREQ : integer := 100_000_000;
constant BIT_TICKS  : integer := CLOCK_FREQ / BAUD_RATE;

type rx_state_type is (IDLE, START, DATA, STOP);
signal rx_state  : rx_state_type := IDLE;
signal bit_cnt   : integer range 0 to 7 := 0;
signal baud_cnt  : integer range 0 to BIT_TICKS-1 := 0;

signal shift_reg : std_logic_vector(7 downto 0) := (others => '0');
```

```
signal rx_data_reg  : std_logic_vector(7 downto 0) := (others =>
'0');
signal rx_valid_reg : std_logic := '0';
```

begin

```
rx_data  <= rx_data_reg;
rx_valid <= rx_valid_reg;

process(clk, reset)
begin
    if reset = '1' then
        rx_state     <= IDLE;
        bit_cnt      <= 0;
        baud_cnt     <= 0;
        shift_reg    <= (others => '0');
        rx_data_reg  <= (others => '0');
        rx_valid_reg <= '0';

    elsif rising_edge(clk) then


        rx_valid_reg <= '0';

        case rx_state is

            when IDLE =>

                if rx = '0' then
                    rx_state <= START;
                    baud_cnt <= 0;
                end if;

            when START =>

                if baud_cnt = BIT_TICKS/2 then
                    if rx = '0' then
                        rx_state <= DATA;
                        bit_cnt  <= 0;
                    else
```

```
                        rx_state <= IDLE;
                    end if;
                    baud_cnt <= 0;
                else
                    baud_cnt <= baud_cnt + 1;
                end if;

            when DATA =>
                if baud_cnt = BIT_TICKS-1 then

                    shift_reg(bit_cnt) <= rx;
                    baud_cnt <= 0;
                    if bit_cnt = 7 then
                        rx_state <= STOP;
                    else
                        bit_cnt <= bit_cnt + 1;
                    end if;
                else
                    baud_cnt <= baud_cnt + 1;
                end if;

            when STOP =>

                if baud_cnt = BIT_TICKS-1 then
                    baud_cnt     <= 0;
                    rx_state     <= IDLE;
                    rx_data_reg  <= shift_reg;
                    rx_valid_reg <= '1';   -- signal that new data
is ready
                else
                    baud_cnt <= baud_cnt + 1;
                end if;

        end case;
    end if;
end process;


end Behavioral;
```

Servo_pwm.vhd

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

entity servo_pwm is Port (clk : IN STD_LOGIC; reset : IN STD_LOGIC; pos : IN STD_LOGIC_VECTOR(7 downto 0); : OUT STD_LOGIC); end servo_pwm;

architecture Behavioral of servo_pwm is

```
signal cnt        : unsigned(11 downto 0) := (others => '0');
signal pwmi       : unsigned(11 downto 0);
signal pos_clamped: unsigned(7 downto 0);
```

begin

```
process(pos)
begin
    if unsigned(pos) > 180 then
        pos_clamped <= to_unsigned(180, 8);
    else
        pos_clamped <= unsigned(pos);
    end if;
end process;

pwmi <= resize(pos_clamped * 2 + to_unsigned(90, 7), 12);

process(clk, reset)
begin
    if reset = '1' then
        cnt <= (others => '0');
    elsif rising_edge(clk) then
        if cnt = to_unsigned(3599, 12) then
            cnt <= (others => '0');
        else
            cnt <= cnt + 1;
        end if;
    end if;
end process;

servo <= '1' when cnt < pwmi else '0';
```

end Behavioral;

Servo_pwm22.vhd

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

entity servo_pwm2 is Port (clk : IN STD_LOGIC; reset : IN STD_LOGIC; pos2 : IN STD_LOGIC_VECTOR(7 downto 0); servo2: OUT STD_LOGIC); end servo_pwm2;

architecture Behavioral of servo_pwm2 is

```
signal cnt         : unsigned(11 downto 0) := (others => '0');  --
0..3599
signal pwmi        : unsigned(11 downto 0);
signal pos2_clamped: unsigned(7 downto 0) := (others => '0');
```

begin

```
process(pos2)
begin
    if unsigned(pos2) > 180 then
        pos2_clamped <= to_unsigned(180, 8);
    else
        pos2_clamped <= unsigned(pos2);
    end if;
end process;

pwmi <= resize(pos2_clamped * 2 + to_unsigned(90, 7), 12);

process(clk, reset)
begin
    if reset = '1' then
        cnt <= (others => '0');
    elsif rising_edge(clk) then
        if cnt = to_unsigned(3599, 12) then
            cnt <= (others => '0');
        else
            cnt <= cnt + 1;
        end if;
    end if;
end process;
```

```
servo2 <= '1' when cnt < pwmi else '0';
```

end Behavioral;

Basys3.xdc

set_property PACKAGE_PIN W5 [get_ports {clk}]

set_property IOSTANDARD LVCMOS33 [get_ports {clk}]

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

set_property PACKAGE_PIN B18 [get_ports {uart_rx}] set_property IOSTANDARD LVCMOS33 [get_ports {uart_rx}]

set_property PACKAGE_PIN J1 [get_ports {servo1}] set_property IOSTANDARD LVCMOS33 [get_ports {servo1}]

set_property PACKAGE_PIN L2 [get_ports {servo2}] set_property IOSTANDARD LVCMOS33 [get_ports {servo2}]