EMOTION CLASSIFIER

Bilkent University Electrical and Electronics Engineering Department
EEE102 Term Project

CANKUT BORA TUNCER
22001770
Section 3

Youtube:

https://www.youtube.com/watch?v=AvGhKGVoAjg

Objective:

This project aimed to classify human emotions and display the detected emotion to the outside world via Basys3.

Methodology:

The first step was to design a custom-made classification algorithm. To do so, a base model (RESNET50_V2) is processed with feature extraction and fine-tuning After the training; it is chosen the most accurate model (82% acc, %80 val_acc). As for the VHDL part, the design is supposed to execute the following tasks:

1) Communicating with the computer using serial communication (UART).
2) Reading the received data and adjusting the servos' configuration synchronously
3) Displaying the emotion results to the seven-segment display.

Since the Deep Learning model looks for the eyebrows and mouth in the given human face, the servos represent the dummy's eyebrows and mouth by following the same principle. Face.

After the design process, the project was simulated on the FPGA.

Design Specifications:

There are five inputs and nine outputs as follows:

clk	: in std_logic
reset	: in std_logic
servo1, servo2	: out std_logic
servo3, servo4, servo5, servo6	: out std_logic
dummy	: in std_logic_vector (7 downto 0)
sel	: in std_logic
rx	: in std_logic
data_out	: out std_logic_vector (7 downto 0)
anode	: out std_logic_vector (3 downto 0)
cathode	: out std_logic_vector (6 downto 0)

The design includes six servo drivers, 1 seven segment driver, 1 UART communicator, and 11 different submodules(Figure 1).

Figure 1 The Hyerarchy

The “UART_reciever.vhd” module receives the 8-bit data from the as serial. It consists of three inputs(clk, reset, rx) and 1 output(rx_data_out)(Figure 2). The UART module has an FSM with four states, IDLE, START, DATA, STOP. The receiver looks for an input ‘0’ in the serial during the IDLE state, starting the transmission and going into the START state. In the START state, the code checks if the start signal is valid or not; if the signal is valid, move to DATA else go back to IDLE. The code reads the serial in the DATA state and stores the 8-bit input one by 1 to a register. Finally, the code waits for stop signal ‘1’ and moves back to the IDLE state.

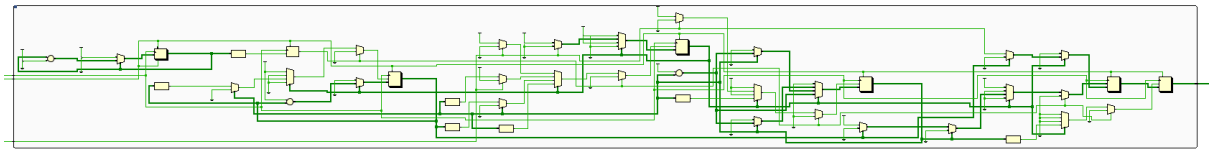

Figure 2 The RTL of the UART module

The output data is then forwarded to the “emotion_projection.vhd” module(Figure 3). It has three inputs(clk, reset, data_in) and six outputs(servo1, servo2, servo3, servo4, servo5, servo6). This module interprets the emotion input and adjusts the servo positions accordingly. As seen in Figure 4, if an “11000000” arrives, it is interpreted as the “sad” emotion and the new positions are assigned to the servo positions.

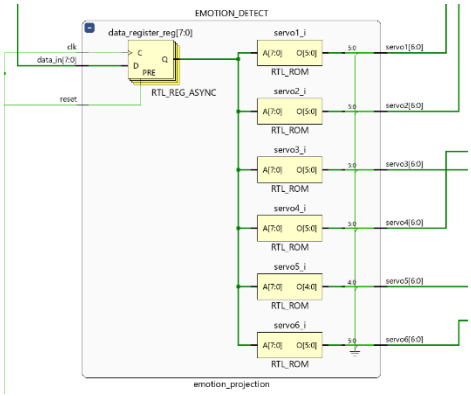

Figure 3 The RTL of the Emotion Detect

```
when "11000000" => -- SAD
    servo1 <= DEGREE_134; -- MBL Mouth Bottom Left
    servo2 <= DEGREE_0; -- MBR Mouth Bottom Right
    servo3 <= DEGREE_45; -- MUL Mouth Upper Left
    servo4 <= DEGREE_90; -- MUR Mouth Upper Right
    servo5 <= DEGREE_66; -- EL Eye Left
    servo6 <= DEGREE_22; -- ER Eye Right;
```

Figure 4 The “Sad” case

The servo positions are sent to the “servo_main.vhd” module(Figure 5). It has two inputs(clk, reset) and two outputs(pos, servo). This module consists of 2 other submodules: “servo_clkdiv.vhd” and “servo_pwm.vhd”. The servo needs 0.5-2.5ms pulses in 20ms periods. The servo can turn by 128 steps. Since the servo works with the Pulse Width Modulation Principle, the clock required for the servo is 2ms/128 = 64kHz. Thus the clock divider module generates the 64kHz clock for the “servo_pwm”. The PWM module then modulates the output signal for the given position.

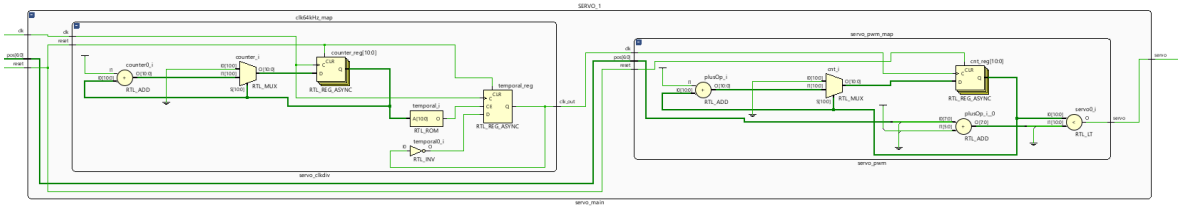

Figure 5 The RTL of the Servo Module

The output is also projected to the seven-segment display. The “seven-segment.vhd” receives the emotion data and projects it into the display(Figure 6). It has 3 inputs(clk, reset, emotion) and 2 outputs(cathode, anode).

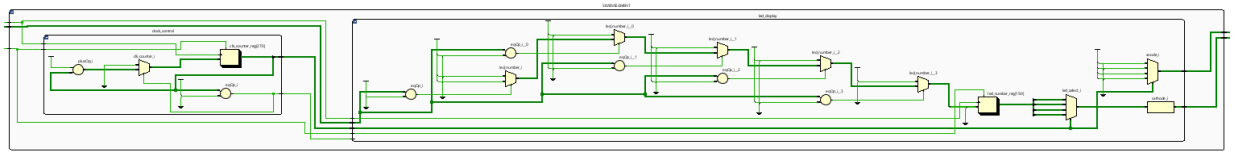

Figure 6 The RTL of the Seven Segment Module

Besides receiving the emotion input via. UART, the user can also give the data with the switches. The user can provide custom inputs by switching the “sel” switch. The RTL of the

main can be seen in Figure 7. Also, the Basys3 and dummy configuration is depicted in Figure 8-9.

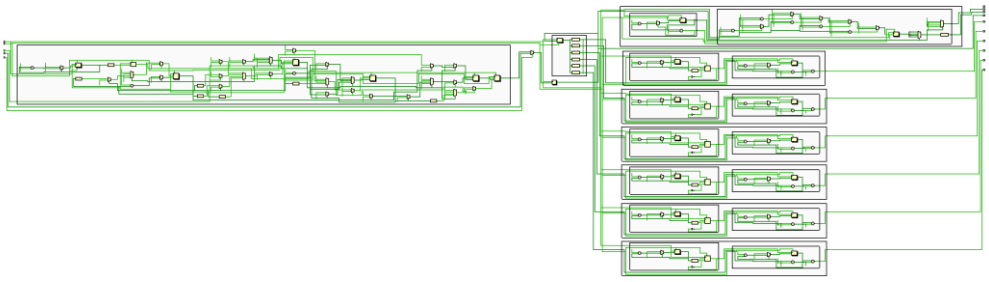

Figure 7 The RTL of the Main

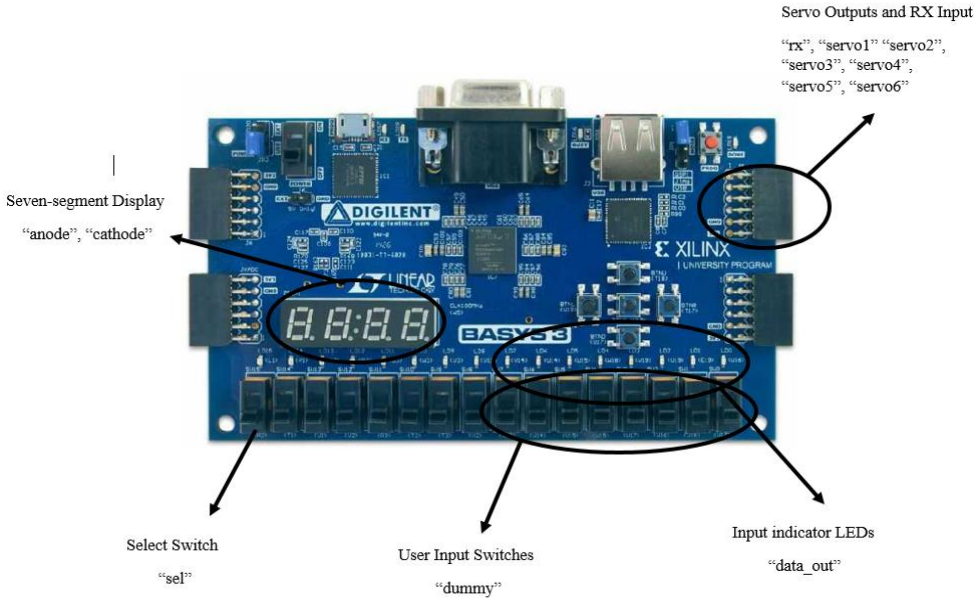

Figure 8 The Basys3 Configuration

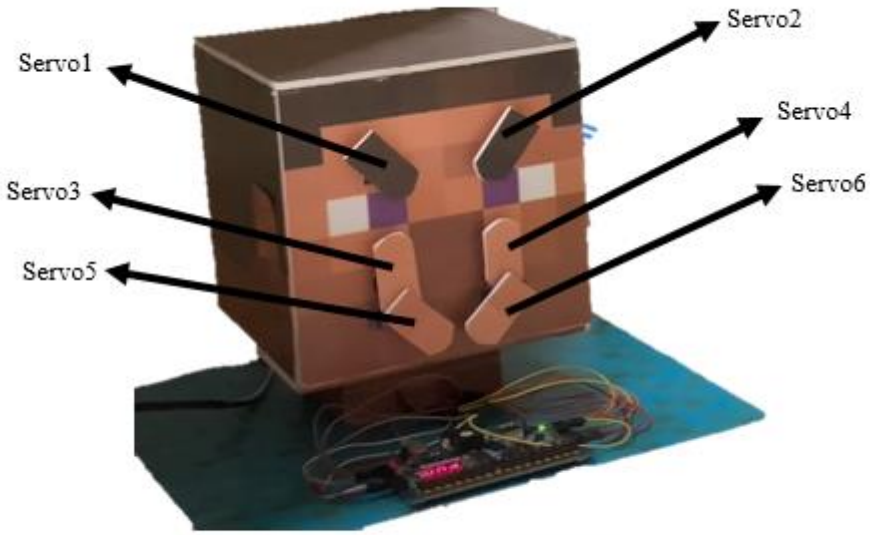

Figure 9 The Dummy Configuration

Results:

There are six possible configurations of the servos' and the seven-segment display. The configurations are as follows:

When:

UART input: "10000000" and Sel = '1' or

User input: "10000000" and Sel = '0'

Then:

Emotion: "Angry"

servo1: DEGREE_134

servo2: DEGREE_0

servo3: DEGREE_45

servo4: DEGREE_90

servo5: DEGREE_0

servo6: DEGREE_90

seven-segment: ANG

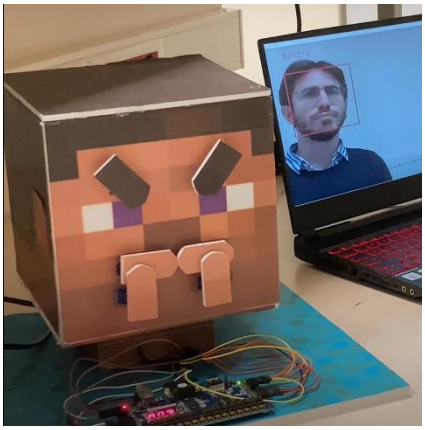

Figure 10 The "Angry" Configuration

When:

UART input: "11000000" and Sel = '1' or

User input: "11000000" and Sel = '0'

Then:

Emotion: "Sad"

servo1: DEGREE_134

servo2: DEGREE_0

servo3: DEGREE_45

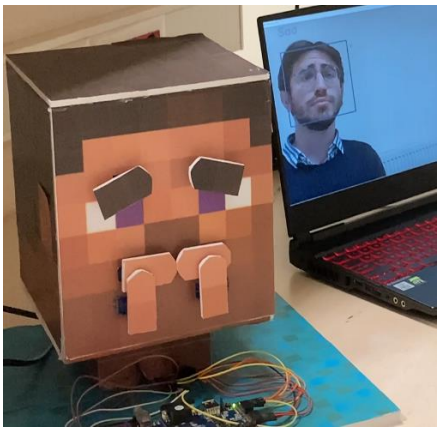

Figure 11 The "Sad" Configuration

servo4: DEGREE_90

servo5: DEGREE_66

servo6: DEGREE_22

seven-segment: SAD

When:

UART input: “11100000” and Sel = ‘1’ or

User input: “11100000” and Sel = ‘0’

Then:

Emotion: “Happy”

servo1: DEGREE_45

servo2: DEGREE_90

servo3: DEGREE_134

servo4: DEGREE_0

servo5: DEGREE_45

servo6: DEGREE_45

seven-segment: HAP

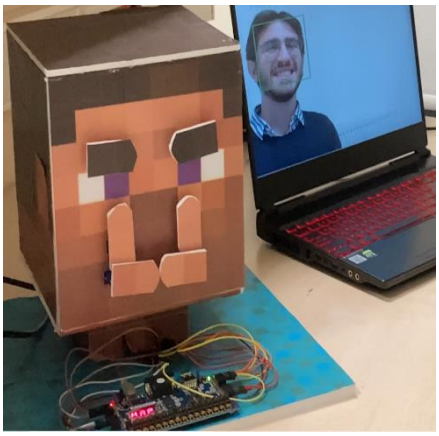

Figure 12 The “Happy” Configuration

When:

UART input: “11110000” and Sel = ‘1’ or

User input: “11110000” and Sel = ‘0’

Then:

Emotion: “Suprised”

servo1: DEGREE_0

servo2: DEGREE_134

servo3: DEGREE_90

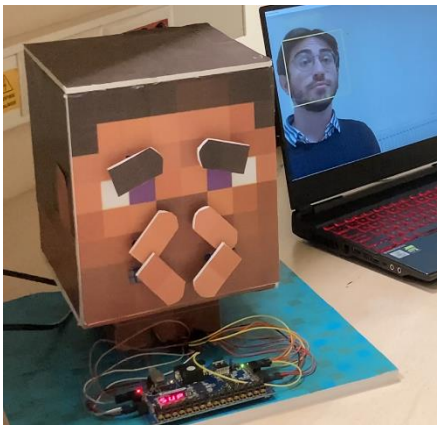

Figure 13 The “Suprised” Configuration

servo4: DEGREE_45

servo5: DEGREE_66

servo6: DEGREE_22

seven-segment: SUP

When:

UART input: “11111000” and Sel = ‘1’ or

User input: “11111000” and Sel = ‘0’

Then:

Emotion: “Neutral”

servo1: DEGREE_45

servo2: DEGREE_90

servo3: DEGREE_45

servo4: DEGREE_90

servo5: DEGREE_45

servo6: DEGREE_45

seven-segment: NEU

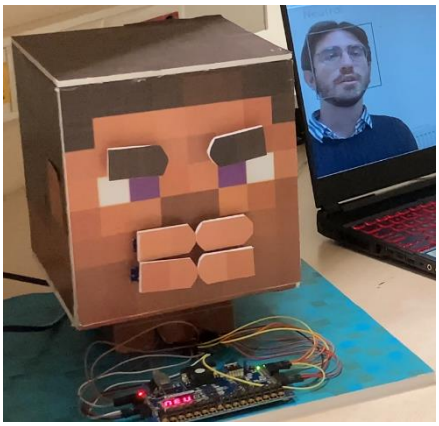

Figure 14 The “Neutral” Configuration

When:

UART input: others and Sel = ‘1’ or

User input: others and Sel = ‘0.’

Then:

Emotion: “Undefined”

servo1: DEGREE_0;

servo2: DEGREE_134

servo3: DEGREE_134

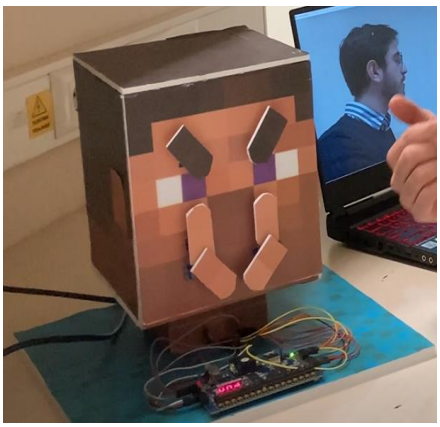

Figure 15 The “Undefined” Configuration

servo4: DEGREE_0;

servo5: DEGREE_0;

servo6: DEGREE_90;

seven-segment: UND

Conclusion:

In this project, I utilized Deep Learning with a VHDL project. The training of the emotion classifier took 5 months prior to the project proposal. Different from what is available on the web, I have improved the accuracy of the model from 65% to 80%. There is a bias towards the ‘sad’ emotion class, however, I compensate it with minor calibrations. Since this is a VHDL project the main logic is located in the FPGA. The computer and python are only used for face and emotion detection. The data processing occurs at Basys3. When the emotion data arrives over the UART, the VHDL design recognizes the data and drives the servos and seven-segment display accordingly. Furthermore, for testing purposes, users may give custom inputs rather than UART.

Although, I had to pivot from what I did propose at the start, this way the project become more of a VHDL project rather than a Python project. Now, the communication direction is turned. Rather than sending data to PC, the FPGA receives the data. The servos’ are moving in synchrony thus precision was important. Since there is always an error possibility in Deep Learning applications, the data is transmitted every 5 consequent recognitions. I have learned many techniques from outside sources and I have included them in my code after citing them. For instance, the UART was a new concept for me thus I had to cite from another source and modify the code according to my project’s needs.

Still, there is some room for improvement. The emotion classification is accurate but it can be more precise. Although I have used the AffectNet dataset, I did not train my model with RGB but grayscales images. Traning with RGB images could have improved the accuracy. Furthermore, the camera used in this project was the built-in camera in the laptop. It had a low resolution which again influenced the results. With a high-resolution camera, this problem can be overcome. Utilizing a USB camera with the Basys3, and sending the data over not UART but UDP over ethernet or Wifi is possible and could be done in another project to develop this one.

References:

Ramos, Carlos A. "Servomotor Control with PWM and VHDL." CodeProject. CodeProject, December 20, 2012. https://www.codeproject.com/Articles/513169/Servomotor-Control-with-PWM-and-VHDL.

Alex. "UART_Controller." GitHub, 2020. https://github.com/AlexHDL/UART_controller.

"VHDL Code for Seven-Segment Display on Basys 3 FPGA." FPGA Projects, Verilog Projects, VHDL Projects - FPGA4student.com. Accessed December 30, 2021. https://www.fpga4student.com/2017/09/vhdl-code-for-seven-segment-display.html.

Appendix:

Main.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------------------------------
entity main is
    Port ( clk                              : in STD_LOGIC;
           reset                            : in STD_LOGIC;
           servo1, servo2, servo3, servo4, servo5, servo6 : out std_logic;
           dummy                            : in std_logic_vector (7 downto 0);
           sel                              : in std_logic;
           rx                               : in std_logic;
           data_out                         : out std_logic_vector (7 downto 0);
           anode                            : out STD_LOGIC_VECTOR (3 downto 0);
           cathode                          : out STD_LOGIC_VECTOR (6 downto 0)
           );
end main;
----------------------------------------------------------------------
architecture Behavioral of main is

signal emotion: std_logic_vector(7 downto 0);
signal servo_loc1, servo_loc2, servo_loc3, servo_loc4, servo_loc5, servo_loc6: std_logic_vector(6
downto 0);
signal data_emot : std_logic_vector (7 downto 0);
signal data_in: std_logic_vector(7 downto 0):="00000000";
begin


----------------------------------------------------------------------
UART: entity work.UART_reciever(Behavioral)
    port map(clk      => clk,
             reset    => reset,
             rx       => rx,
```

```
            rx_data_out => emotion);
-------------------------------------------------------------------
sel_proc: process begin
   if rising_edge(clk) then
      if sel = '1' then
         data_out <= emotion;
         data_emot <= emotion;
      else
         data_out <= dummy;
         data_emot <= dummy;
      end if;
   end if;
end process;
-------------------------------------------------------------------
EMOTION_DETECT: entity work.emotion_projection(Behavioral)
   port map(clk => clk,
            reset => reset,
            data_in => data_emot,
            servo1 => servo_loc1,
            servo2 => servo_loc2,
            servo3 => servo_loc3,
            servo4 => servo_loc4,
            servo5 => servo_loc5,
            servo6 => servo_loc6);
-------------------------------------------------------------------
SEVENSEGMENT: entity work.sevensegment(rtl)
   port map(clk => clk,
            reset => reset,
            emotion=> data_emot,
            anode => anode,
            cathode => cathode);
-------------------------------------------------------------------
SERVO_1: entity work.servo_main(Behavioral)
   port map(clk      => clk,
            reset    => reset,
            pos      => servo_loc1,
```

```
            servo    => servo1);
    ----------------------------------------------------------------------
    SERVO_2: entity work.servo_main(Behavioral)
       port map(clk      => clk,
                reset    => reset,
                pos      => servo_loc2,
                servo    => servo2);
    ----------------------------------------------------------------------
    SERVO_3: entity work.servo_main(Behavioral)
       port map(clk      => clk,
                reset    => reset,
                pos      => servo_loc3,
                servo    => servo3);
    ----------------------------------------------------------------------
    SERVO_4: entity work.servo_main(Behavioral)
       port map(clk      => clk,
                reset    => reset,
                pos      => servo_loc4,
                servo    => servo4);
    ----------------------------------------------------------------------
    SERVO_5: entity work.servo_main(Behavioral)
       port map(clk      => clk,
                reset    => reset,
                pos      => servo_loc5,
                servo    => servo5);
    ----------------------------------------------------------------------
    SERVO_6: entity work.servo_main(Behavioral)
       port map(clk      => clk,
                reset    => reset,
                pos      => servo_loc6,
                servo    => servo6);
    ----------------------------------------------------------------------
    end Behavioral;
```

uart_reciever.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 rx_states_t is (IDLE, START, DATA, STOP);
   signal rx_state: rx_states_t := IDLE;

   signal baud_rate_x16_clk  : std_logic := '0';
   signal rx_stored_data     : std_logic_vector(7 downto 0) := (others => '0');

begin
----------------------------------------------------------------------
   baud_rate_x16_clk_generator: 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_rate_x16_clk_generator;
--------------------------------------------------------------------------------
    UART_rx_FSM: process(clk)
        variable bit_duration_count : integer range 0 to 15 := 0;
        variable bit_count          : integer range 0 to 7  := 0;
    begin
        if rising_edge(clk) then
            if (reset = '1') then
                rx_state <= IDLE;
                rx_stored_data <= (others => '0');
                rx_data_out <= (others => '0');
                bit_duration_count := 0;
                bit_count := 0;
            else
                if (baud_rate_x16_clk = '1') then
                    case rx_state is

                        when IDLE =>

                            rx_stored_data <= (others => '0');
                            bit_duration_count := 0;
                            bit_count := 0;

                            if (rx = '0') then
                                rx_state <= START;
                            end if;

                        when START =>

                            if (rx = '0') then
                                if (bit_duration_count = 7) then
```

```
                rx_state <= DATA;
                bit_duration_count := 0;
            else
                bit_duration_count := bit_duration_count + 1;
            end if;
        else
            rx_state <= IDLE;
        end if;

    when DATA =>

        if (bit_duration_count = 15) then
            rx_stored_data(bit_count) <= rx;
            bit_duration_count := 0;
            if (bit_count = 7) then
                rx_state <= STOP;
                bit_duration_count := 0;
            else
                bit_count := bit_count + 1;
            end if;
        else
            bit_duration_count := bit_duration_count + 1;
        end if;

    when STOP =>

        if (bit_duration_count = 15) then
            rx_data_out <= rx_stored_data;
            rx_state <= IDLE;
        else
            bit_duration_count := bit_duration_count + 1;
        end if;

    when others =>
        rx_state <= IDLE;
end case;
```

```
                end if;
            end if;
        end if;
    end process UART_rx_FSM;
-------------------------------------------------------------------------
end Behavioral;
```

emotion_projection.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.all;
-------------------------------------------------------------------------
entity emotion_projection is

    Port ( clk: in std_logic;
           reset: in std_logic;
           data_in : in std_logic_vector(7 downto 0);
           servo1, servo2, servo3, servo4, servo5, servo6: out std_logic_vector(6 downto 0));

end emotion_projection;
-------------------------------------------------------------------------
architecture Behavioral of emotion_projection is

    signal data_register: std_logic_vector(7 downto 0):=(others => '0');
    signal servo_case: std_logic_vector(2 downto 0);
    signal counter: std_logic_vector(10 downto 0);

    constant DEGREE_134: std_logic_vector(6 downto 0):= "0110000";
    constant DEGREE_90: std_logic_vector(6 downto 0):= "0100000";
    constant DEGREE_45: std_logic_vector(6 downto 0):= "0010000";
    constant DEGREE_66: std_logic_vector(6 downto 0):= "0011000";
    constant DEGREE_22: std_logic_vector(6 downto 0):= "0001000";
    constant DEGREE_0:  std_logic_vector(6 downto 0):= "0000000";
    constant RES:       std_logic_vector(7 downto 0):= "11111111";
begin
```

```
-------------------------------------------------------------------
servo_proc: process begin
    case data_register is
        when "10000000" => -- ANGRY
            servo1 <= DEGREE_134; -- MBL Mouth Bottom Left
            servo2 <= DEGREE_0; -- MBR Mouth Bottom Right
            servo3 <= DEGREE_45; -- MUL Mouth Upper Left
            servo4 <= DEGREE_90; -- MUR Mouth Upper Right
            servo5 <= DEGREE_0; -- EL Eye Left
            servo6 <= DEGREE_90; -- ER Eye Right;
        when "11000000" => -- SAD
            servo1 <= DEGREE_134; -- MBL Mouth Bottom Left
            servo2 <= DEGREE_0; -- MBR Mouth Bottom Right
            servo3 <= DEGREE_45; -- MUL Mouth Upper Left
            servo4 <= DEGREE_90; -- MUR Mouth Upper Right
            servo5 <= DEGREE_66; -- EL Eye Left
            servo6 <= DEGREE_22; -- ER Eye Right;
        when "11100000" => -- HAPPY
            servo1 <= DEGREE_45; -- MBL Mouth Bottom Left
            servo2 <= DEGREE_90; -- MBR Mouth Bottom Right
            servo3 <= DEGREE_134; -- MUL Mouth Upper Left
            servo4 <= DEGREE_0; -- MUR Mouth Upper Right
            servo5 <= DEGREE_45; -- EL Eye Left
            servo6 <= DEGREE_45; -- ER Eye Right;
        when "11110000" => -- SUPRISED
            servo1 <= DEGREE_0;  -- MBL Mouth Bottom Left
            servo2 <= DEGREE_134; -- MBR Mouth Bottom Right
            servo3 <= DEGREE_90; -- MUL Mouth Upper Left
            servo4 <= DEGREE_45; -- MUR Mouth Upper Right
            servo5 <= DEGREE_66; -- EL Eye Left
            servo6 <= DEGREE_22; -- ER Eye Right;
        when "11111000" => -- NEUTRAL
            servo1 <= DEGREE_45; -- MBL Mouth Bottom Left
            servo2 <= DEGREE_90; -- MBR Mouth Bottom Right
            servo3 <= DEGREE_45; -- MUL Mouth Upper Left
            servo4 <= DEGREE_90; -- MUR Mouth Upper Right
```

```
            servo5 <= DEGREE_45; -- EL Eye Left
            servo6 <= DEGREE_45; -- ER Eye Right;
        when others => -- DUMP
            servo1 <= DEGREE_0;  -- MBL Mouth Bottom Left
            servo2 <= DEGREE_134; -- MBR Mouth Bottom Right
            servo3 <= DEGREE_134; -- MUL Mouth Upper Left
            servo4 <= DEGREE_0;  -- MUR Mouth Upper Right
            servo5 <= DEGREE_0; -- EL Eye Left;
            servo6 <= DEGREE_90; -- ER Eye Right;
    end case;
end process;
----------------------------------------------------------------------
    emotion_proc: process begin
        if reset = '1' then
            data_register <= RES;
        else
            if rising_edge(clk) then
                data_register <= data_in;
            end if;
        end if;
    end process;
----------------------------------------------------------------------
end Behavioral;
```

sevensegment.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
----------------------------------------------------------------------
entity sevensegment is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           emotion: in STD_LOGIC_VECTOR(7 downto 0);
           anode : out STD_LOGIC_VECTOR (3 downto 0);
           cathode : out STD_LOGIC_VECTOR (6 downto 0));
end sevensegment;
```

```
----------------------------------------------------------------------
architecture rtl of sevensegment is

signal phase_control: STD_LOGIC_VECTOR (1 downto 0);
signal sec_control: STD_LOGIC;
begin
----------------------------------------------------------------------
clock_control: entity work.clock_control(rtlclock)
   port map(clk => clk,
            reset => reset,
            phase => phase_control,
            sec => sec_control);
----------------------------------------------------------------------
led_display: entity work.led_display(rtl_display)
   port map(clk => clk,
            reset => reset,
            phase => phase_control,
            sec => sec_control,
            emotion => emotion,
            anode => anode,
            cathode => cathode);
----------------------------------------------------------------------
end rtl;
```

clock_control.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
----------------------------------------------------------------------
entity clock_control is
Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           phase : out  STD_LOGIC_VECTOR (1 downto 0);
           sec : out STD_LOGIC );
end clock_control;
----------------------------------------------------------------------
```

```
architecture rtlclock of clock_control is
        signal clk_counter: STD_LOGIC_VECTOR (27 downto 0);
      signal phase_counter: STD_LOGIC_VECTOR (1 downto 0);
begin
process(clk,reset) begin
      if reset = '1' then
        clk_counter <= (others => '0');
      else
        if rising_edge(clk) then
        clk_counter <= clk_counter + '1';
        if clk_counter =x"5F5E0FF" then
              clk_counter <= (others => '0');
        end if;
      end if;
  end if;
end process;
--------------------------------------------------------------------
sec <= '1' when clk_counter =x"5F5E0FF" else '0';
phase <= clk_counter(19 downto 18);
--------------------------------------------------------------------
end rtlclock;
```

led_display.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
--------------------------------------------------------------------
entity led_display is
   Port (clk:in STD_LOGIC;
      reset: in STD_LOGIC;
      phase: in  STD_LOGIC_VECTOR (1 downto 0);
      sec: in STD_LOGIC;
      emotion: in std_logic_vector(7 downto 0);
      anode : out  STD_LOGIC_VECTOR (3 downto 0);
      cathode: out  STD_LOGIC_VECTOR (6 downto 0));
end led_display;
```

```
--------------------------------------------------------------------------
architecture rtl_display of led_display is

signal led_number:  STD_LOGIC_VECTOR (15 downto 0);
signal led_select:  STD_LOGIC_VECTOR (3 downto 0);

begin
--------------------------------------------------------------------------
   process(phase, led_number) begin
      case phase is
         when "00" => anode <= "0111";
                   led_select <= led_number(15 downto 12);
         when "01" => anode <= "1011";
                   led_select <= led_number(11 downto 8);
         when "10" => anode <= "1101";
                   led_select <= led_number(7 downto 4);
         when "11" => anode <= "1110";
                   led_select <= led_number(3 downto 0);
         when others => anode <= "0000";

       end case;
    end process;
--------------------------------------------------------------------------
   process(clk,reset) begin
      if reset = '1' then
         led_number <= (others => '0');
      elsif rising_edge(clk) then
         if emotion = "10000000" then -- angry
            led_number <= "0000010101101111";
         elsif emotion = "11000000" then -- sad
            led_number <= "0011000001001111";
         elsif emotion = "11100000" then -- happy
            led_number <= "0001000000101111";
         elsif emotion = "11110000" then -- suprised
            led_number <= "0011011100101111";
```

```
            elsif emotion = "11111000" then -- neutral
               led_number <= "0101100001111111";
            else
               led_number <= "0111010101001111";

               end if;
            end if;
      end process;
     ----------------------------------------------------------------------
      process(led_select) begin
         case led_select is          --1234567
            when "0000" => cathode <= "0001000"; -- a
            when "0001" => cathode <= "1001000"; -- h
            when "0010" => cathode <= "0011000"; -- p
            when "0011" => cathode <= "0100100"; -- s
            when "0100" => cathode <= "1000010"; -- d
            when "0101" => cathode <= "0001001"; -- n
            when "0110" => cathode <= "0000100"; -- g
            when "0111" => cathode <= "1000001"; -- u
            when "1000" => cathode <= "0110000"; -- e
            when others => cathode <= "1111111";
         end case;
      end process;
   ----------------------------------------------------------------------
   end rtl_display;
```

servo_main.vhd

```
   library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
   ----------------------------------------------------------------------
   entity servo_main is
      PORT(
         clk  : IN  STD_LOGIC;
         reset: IN  STD_LOGIC;
         pos  : IN  STD_LOGIC_VECTOR(6 downto 0);
         servo: OUT STD_LOGIC
```

```
    );
end servo_main;
----------------------------------------------------------------------
architecture Behavioral of servo_main is
    signal clk_out : STD_LOGIC := '0';
begin
    clk64kHz_map: entity work.servo_clkdiv(behavioral) PORT MAP(
            clk, reset, clk_out
        );
----------------------------------------------------------------------
    servo_pwm_map: entity work.servo_pwm(Behavioral) PORT MAP(
            clk_out, reset, pos, servo
        );
----------------------------------------------------------------------
end Behavioral;
```

servo_clkdiv.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------------------------------
entity servo_clkdiv is
    Port (
        clk    : in  STD_LOGIC;
        reset  : in  STD_LOGIC;
        clk_out: out STD_LOGIC
    );
end servo_clkdiv;
----------------------------------------------------------------------
architecture Behavioral of servo_clkdiv is
    signal temporal: STD_LOGIC;
    signal counter : integer range 0 to 1560 := 0;
begin
----------------------------------------------------------------------
    freq_divider: process (reset, clk) begin
        if (reset = '1') then
```

```
            temporal <= '0';
            counter  <= 0;
        elsif rising_edge(clk) then
            if (counter = 1560) then
                temporal <= NOT(temporal);
                counter  <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;
    ----------------------------------------------------------------------
    clk_out <= temporal;
    ----------------------------------------------------------------------
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(6 downto 0);
        servo : OUT STD_LOGIC
    );
end servo_pwm;
----------------------------------------------------------------------
architecture Behavioral of servo_pwm is

    signal cnt : unsigned(10 downto 0);
    signal pwmi: unsigned(7 downto 0);
```

```
begin
-------------------------------------------------------------------------
    pwmi <= unsigned('0' & pos) + 32;
    contador: process (reset, clk) begin
        if (reset = '1') then
            cnt <= (others => '0');
        elsif rising_edge(clk) then
            if (cnt = 1279) then
                cnt <= (others => '0');
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end process;
    -------------------------------------------------------------------------
    servo <= '1' when (cnt < pwmi) else '0';
-------------------------------------------------------------------------
end Behavioral;
```