The Design Methodology

The aim of this project was to design and implement an interactive entertainment system using two ultrasonic distance sensors, the responses to the sensor input would be both audio and visual.

There was an “eye” in a black and white colour scheme on a VGA display. In an initial state, it will look straight ahead, and there will be a sound playing in the background on a connected Speaker, this is when there is no object in front of either Ultrasonic Sensor, which is the “00” case. If there is an object to the right or left of the ‘eye’, which are the “01” and “10” cases respectively, then the ‘eyes’ will ‘look’ towards the object and the sound playing will change to another sound depending on the direction. If there are objects on both sides, the “11” case, then the eyes will stay as they are, but no sound will be played. The following Figure 1 is the RTL Schematic diagram of the system as synthesised by Vivado, it shows the general connectivity of the system and the workings of it.

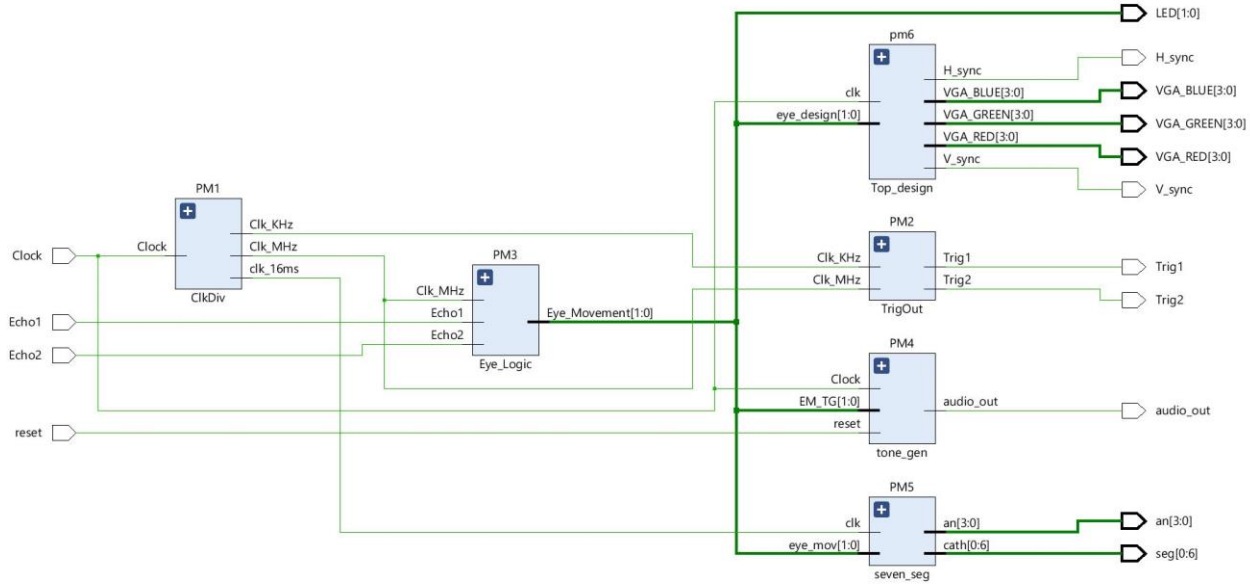

Figure 1. RTL Schematic of the System

Ultrasonic Distance Sensor (HC-SR04)

In my project I have made use of the Ultrasonic sensor, HC-SR04 for detecting objects through the distance measurement function. This device transmits a short burst of ultrasonic sound to the target when a trigger pulse is applied to it. The returned echo is detected by the receiver of the sensor to generate a pulse. The distance of the target can be determined by measuring the width of the received echo pulse and applying basic distance speed time formula.

In order to properly operate, Ultrasonic sensor HC-SR04 requires a trigger (pulse) input of 10µs. In response to this, 8 cycle sonic burst (8 pulses of 40 KHz) is sent out towards the target. The trigger pulses from left/right sensors are automatically generated at an update rate of 100ms. Thus continuous detection process in active to search for any object in our selected range. The information of distance (travel forward and bounce back) is received in the Echo signal (width of the pulse). Figure 1 shows a timing waveform to clarify the principle of operation.

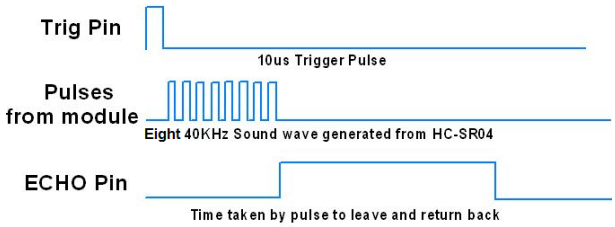

Figure 2. Timing Diagram for HC-SR04 [1]

The Basys3 board works on 100 MHz clock frequency, so by down converting the frequency we can generate a trigger pulse. The returning Echo signal is within the measurement capability of the Basys3 board. Thus distance measurement function can be performed conveniently.

In my project a reference range of 2 to 20 cm is programmed to detect objects. Any object that is outside this range will not be acknowledged. The selected sensor can very conveniently perform the required task.

Distance is measured by first measuring the width of the echo pulse. Following procedure is followed to determine distance in cm:

$t_{pulse} = Width\ of\ echo\ pulse\ in\ \mu\ sec,$

Speed of sound ~ 343 m/s = 0.0343 cm/ µsec,

We divide this time by 2 since sound wave goes out of the sensor and is reflected back, so we have 0.01715 cm / µsec or

$distance\ (cm) = \frac{t_{pulse}}{58},$

Audio Synthesis

In the proposed project, audio tones were also to be played on detection of object in the range of ultrasonic sensor. To achieve this objective a PWM (Pulse Width Modulation) signal of specified frequency was generated so that it may be sent out of Pmod connector pin to a speaker for realizing the audio tone. In practice, musical notes have been standardized and are represented with Note names that are further classified into octaves for which frequency values have been defined. Figure 2 shows a set of notes for Octave 4 along with the corresponding frequencies.

Note	Octave	Frequency
C	4	261.636
D	4	293.665
E	4	329.228
F	4	349.228
G	4	391.995
A	4	440.0
B	4	493.883

Figure 3. Table of Note frequencies

A note is just a PWM signal with specified frequency. Notes as given in Figure 2 were generated in the code. In order to do this, basic clock frequency of 100 MHz was divided by using a counter. The value of counter at which it should be reset was found out by using the following formula:

$$Counter_{value} = \frac{100\ MHz}{(f_{red}\ X\ 2),}, where\ f_{req}\ is\ the\ required\ tone\ frequency$$

Depending on which ultrasonic sensor detected an object in its range, a pre-determined counter value is loaded for comparison with an up counter. As soon as the value of up counter exceeds the reference value, the up counter is reset to count again from zero. On each reset operation the output being sent to audio pin is also inverted. Thus we get a sequence of square wave pulses with 50 % duty cycle. This process continues in an infinite loop. The reference value that is loaded for comparison with counter is selected based on detection of object by either one of the ultrasonic sensor or both.

VGA Controller

In my project the visual interactive part was implemented through display of eye movement on the monitor through Video Graphics Array (VGA) port. VGA is a standard analog interface that can be used to control monitors for display purposes. It receives vertical and horizontal sync inputs along with RGB (Red, Green, Blue) inputs. In order to fulfil our aim, the Basys3 built-in VGA port was made use of. VHDL code was written that defines two counters. One counter controls the horizontal sync, the counter value equals the pixel's column coordinate and the other controls the vertical sync, the counter value equals the pixel's rows. Video signal is displayed when display active condition is high. The display is defined in the code in terms of pixels along with colour information in RGB. For the black background, all of the 4 bit RGB are set to '0's, and the white eyes are with RGB set to '1's. Three images have been made, and based on information received from ultrasonic distance sensors, logic to display corresponding eye is also handled in the code.

Results

The results were as planned overall, the Ultrasonic Sensors were working properly and sensed objects within 20cm (any objects beyond this were ignored). This is satisfactory as that was the limit we defined in our code. Further, as we can see in Figures 4, 5, 6 and 7, the "eye" is programmed to be in black and white and is looking in the directions we want it to for each case. It looks straight ahead in the "00" and "11" case, and also right and left (from the perspective of the system) in the "01" and "10" cases. The speaker also played a musical note for each case, as is labelled in the figures.

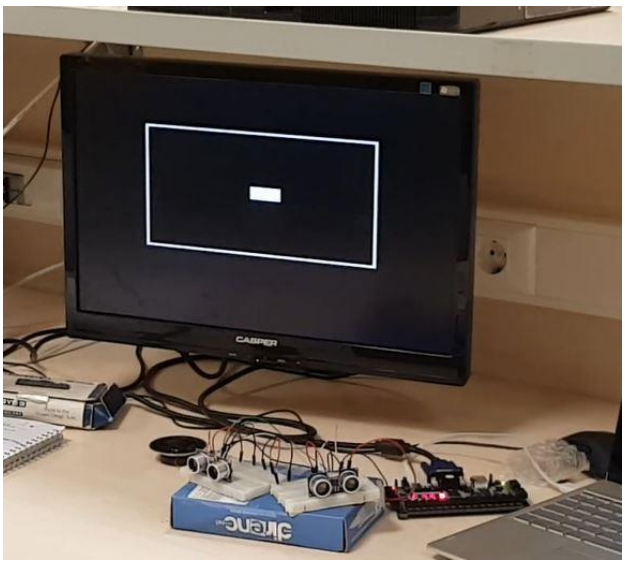

Figure 4. "00" (F4 is played)

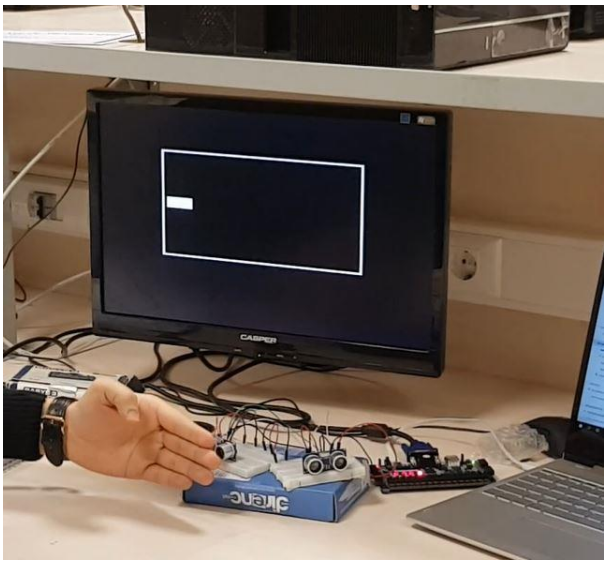

Figure 5. "01" (A4 is played)

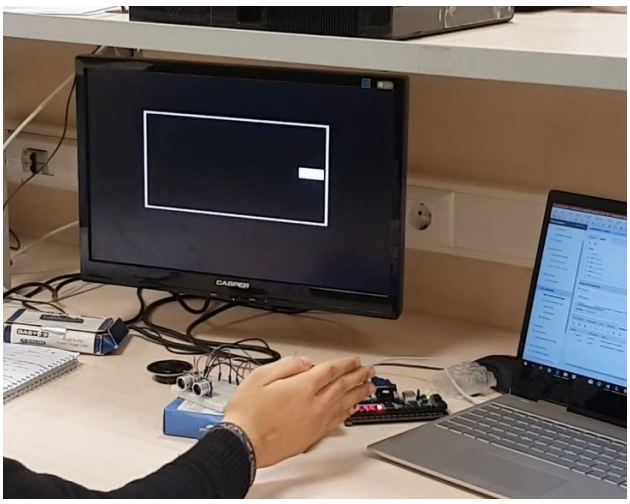

Figure 6. "10" (D4 is played)

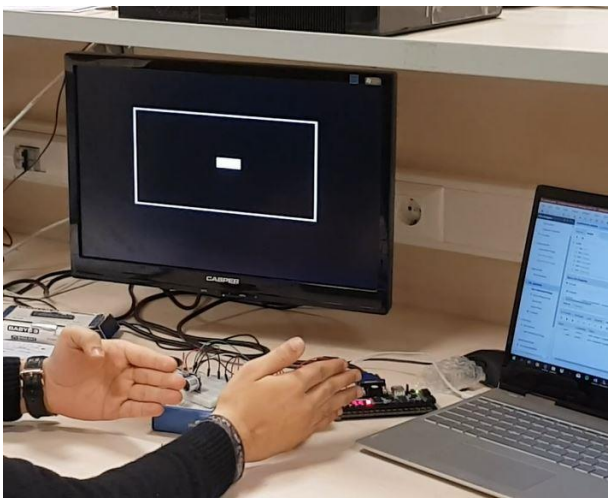

Figure 7. "11" (no sound is played)

Conclusion

The project had complete functionality generally. My phase 1 (Ultrasonic Distance Sensor working) and phase 2 (Audio Synthesis playback on the speaker), worked completely. In my final demonstration, the plan was to make an eye with the use of 8x8 Dot Matrix LEDs, but the device I received was faulty as not all the LEDs turned on, so I could not make the eyes be shown properly. Instead, as there was not enough time to acquire a new 8x8 Dot Matrix LED module, I used a VGA display to show the eye. This was a suitable alternative, as the VGA actually helped show the output on a better and clearly visible display, and should I have chosen so, I can make the eye in any colour. The Ultrasonic Sensors were working properly and detecting the directions properly. The speaker made a unique sound for each case of inputs. The VGA display showed the eye moving in response each case. So overall, I achieved my goal, and the project was working properly.

Materials

- Basys3 FPGA Board
- Vivado on a Computer
- Jumper Wires
- Speaker
- Resistor
- Solderless Breadboards
- Ultrasonic Sensors - HC-SR04
- LCD Display
- VGA Cable

References

[1] https://www.electronicwings.com/sensors-modules/ultrasonic-module-hc-sr04

Appendices

Data Sheets used:

https://elecfreaks.com/estore/download/EF03085-HC-SR04_Ultrasonic_Module_User_Guide.pdf

Project Video Link:

https://drive.google.com/open?id=1MjWRYYbbYtsME42MXrKtwkZXfSH7Ljmn

MAIN.vhd

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

entity MAIN is
  Port ( Clock, reset : in STD_LOGIC; -- Primary Clock of 100 MHz
        Trig1  : out STD_LOGIC; -- Trigger for the sensor1
                        Trig2  : out STD_LOGIC; -- Trigger for the sensor2
                        Echo1  : in STD_LOGIC; -- Return pulse received from Ultrasonic distance
sensor
                        Echo2  : in STD_LOGIC; -- Return pulse received from Ultrasonic distance
sensor
                        audio_out : out std_logic;
                        an : out STD_LOGIC_VECTOR(3 downto 0);
                        seg : out STD_LOGIC_VECTOR(0 to 6);
                        H_sync : out STD_LOGIC; -- VGA Ports
          V_sync : out STD_LOGIC; -- VGA Ports
          VGA_RED: out STD_LOGIC_VECTOR (3 downto 0); -- VGA Ports
          VGA_BLUE : out STD_LOGIC_VECTOR (3 downto 0); -- VGA Ports
          VGA_GREEN : out STD_LOGIC_VECTOR (3 downto 0);
        LED  : out STD_LOGIC_VECTOR ( 1 downto 0)
        );
end MAIN;

architecture Behavioral of MAIN is

COMPONENT ClkDiv
    Port ( Clock : in STD_LOGIC; -- 100 MHz
          Clk_MHz, Clk_KHz, clk_16ms : out STD_LOGIC); -- Primary clock down converted to
generate, 1 MHz and 1 KHz
end COMPONENT;

COMPONENT TrigOut
    Port (          Clk_MHz  : in STD_LOGIC; -- 1 MHz Clock
```

```
                        Clk_KHz  : in STD_LOGIC; -- 1 KHz Clock
                        Trig1  : out STD_LOGIC;
                        Trig2  : out STD_LOGIC); -- 10 micro second pulse to Trigger
Ultrasonic distance sensor
end COMPONENT;

COMPONENT Eye_Logic
   Port (       Echo1 : in STD_LOGIC; -- Echo received in response to auto trigger
                        Echo2 : in STD_LOGIC; -- Echo received in response to auto trigger
                        Clk_MHz : in STD_LOGIC; -- 1 MHz Clock
                        Eye_Movement : out STD_LOGIC_VECTOR ( 1 downto 0)
                        );
end COMPONENT;

COMPONENT tone_gen
Port (    Clock, reset: in std_logic;
          EM_TG : in std_logic_vector(1 downto 0);
          audio_out : out std_logic
     );
end COMPONENT;

COMPONENT seven_seg is
   Port ( clk : in STD_LOGIC;
          eye_mov : in STD_LOGIC_VECTOR(1 downto 0);
          an : out STD_LOGIC_VECTOR(3 downto 0);
          cath : out STD_LOGIC_VECTOR(0 to 6));
end COMPONENT;

COMPONENT Top_design is
   Port ( eye_design: std_logic_vector (1 downto 0);
          clk: in STD_LOGIC;
          H_sync : out STD_LOGIC; -- VGA Ports
          V_sync : out STD_LOGIC; -- VGA Ports
          VGA_RED: out STD_LOGIC_VECTOR (3 downto 0); -- VGA Ports
```

```
        VGA_BLUE : out STD_LOGIC_VECTOR (3 downto 0); -- VGA Ports
        VGA_GREEN : out STD_LOGIC_VECTOR (3 downto 0)); -- VGA Port
end COMPONENT;

   SIGNAL Clk_MHz, Clk_KHz, clk_16ms : STD_LOGIC; -- 1 MHz Clock
   SIGNAL EM : STD_LOGIC_VECTOR ( 1 downto 0); --, Temporary Eye movement
begin

   PM1 : ClkDiv PORT MAP ( Clock => Clock, -- 100 MHz
                           Clk_MHz => Clk_MHz, -- 1 MHz Clock
                           Clk_KHz => Clk_KHz,
                           clk_16ms => clk_16ms); -- 1 KHz Clock

   PM2 : TrigOut PORT MAP ( Clk_MHz => Clk_MHz, -- 1 MHz Clock
                                                Clk_KHz => Clk_KHz, -- 1
KHz Clock
                                                Trig1 => Trig1,
                            Trig2 => Trig2); -- Trigger for the sensor


   PM3 : Eye_Logic PORT MAP (    Echo1 => Echo1, -- Return pulse received from Ultrasonic
distance  sensor
                                                Echo2 => Echo2, -
- Return pulse received from Ultrasonic distance  sensor
                                                Clk_MHz =>
Clk_MHz, -- 1 MHz Clock
                                                Eye_Movement
=> EM); -- distance in binary (9-bit)

        PM4 : tone_gen PORT MAP(          Clock => Clock,
                                                reset => reset,
                                                EM_TG => EM,
                                                audio_out=>audio_out);
        PM5 : SEVEN_SEG PORT MAP(
```

```
        clk => clk_16ms,
        eye_mov => EM,
        an => an,
        cath => seg);

        PM6 : Top_design PORT MAP(
        clk => Clock,
        eye_design => EM,
        H_sync => H_sync,
        V_sync => V_sync,
        VGA_RED => VGA_RED,
        VGA_BLUE => VGA_BLUE,
        VGA_GREEN => VGA_GREEN);

    LED <= EM;

end Behavioral;
```

ClkDiv.vhd

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

entity ClkDiv is
    Port ( Clock : in STD_LOGIC; -- Primary Clock of 100 MHz
           Clk_MHz, Clk_KHz, clk_16ms  : out STD_LOGIC -- 1 MHz Clock, 1KHz
           );
end ClkDiv;

architecture Behavioral of ClkDiv is

    SIGNAL temp_Clk_MHz, temp_Clk_KHz, temp_clk_16ms : STD_LOGIC := '1';
    SIGNAL temp3, temp4, temp5 : INTEGER := 0;

begin

    process ( Clock)
    begin
        -- 1 MHz Clock
        if ( rising_edge( Clock)) then
            if ( temp3 < 49) then
                temp3 <= temp3 + 1;
            else
                temp3 <= 0;
                temp_Clk_MHz <= NOT temp_Clk_MHz;
            end if;
        end if;
        -- 1 KHz Clock
        if ( rising_edge( Clock)) then
            if ( temp4 < 49999) then
                temp4 <= temp4 + 1;
            else
                temp4 <= 0;
```

```
            temp_Clk_KHz <= NOT temp_Clk_KHz;
        end if;
    end if;
    -- 16ms Clock
    if ( rising_edge( Clock)) then
        if ( temp5 < 19999) then
            temp5 <= temp5 + 1;
        else
            temp5 <= 0;
            temp_clk_16ms <= NOT temp_clk_16ms;
        end if;
    end if;
  end process;
  Clk_MHz <= temp_Clk_MHz;
  Clk_KHz <= temp_Clk_KHz;
  clk_16ms <= temp_clk_16ms;
end Behavioral;
```

TrigOut.vhd

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

entity TrigOut is
    Port ( Clk_MHz  : in STD_LOGIC; -- 1 MHz
           Clk_KHz  : in STD_LOGIC; -- 1 KHz
           Trig1  : out STD_LOGIC;
          Trig2  : out STD_LOGIC); -- 10 micro second pulse for Ultrasonic distance sensor
end TrigOut;

architecture Behavioral of TrigOut is

    SIGNAL clk_temp : STD_LOGIC := '1';

begin

    process (Clk_MHz)

    VARIABLE intk, intm : integer:= 0;

    begin
        if (rising_edge(Clk_MHz)) then
            if (intm < 10) then
                intm := intm + 1;
            end if;
            if (intm = 10) then
                clk_temp <= not clk_temp;
                intk := 0;
            end if;

        end if;

        if (rising_edge(Clk_KHz)) then
```

```
            if (intk < 100) then
                intk := intk + 1;
               end if;
            if (intk = 100) then
                    clk_temp <= not clk_temp;
                    intm := 0;
            end if;

        end if;

        Trig1 <= clk_temp;
            Trig2 <= clk_temp;
    end process;
end Behavioral;
```

Eye_Logic.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;

entity Eye_Logic is
    Port (  Echo1 : in STD_LOGIC; -- Echo received in response to auto trigger
            Echo2 : in STD_LOGIC; -- Echo received in response to auto trigger
          Clk_MHz : in STD_LOGIC; -- 1 MHz Clock
          Eye_Movement : out STD_LOGIC_VECTOR (1 downto 0)); -- Eye movement direction,
00=Out of Range, 01=Right, 10=Left, 11= Straight
end Eye_Logic;

architecture Behavioral of Eye_Logic is

    SIGNAL Dist1, Dist2 : STD_LOGIC_VECTOR (14 downto 0); -- Distance resolved in 15 bits for
more accuracy
    SIGNAL tempd1, tempd2 : STD_LOGIC_VECTOR (8 downto 0);
    SIGNAL Dist_cm, Dist2_cm : INTEGER := 0;

begin

    PROCESS (Clk_MHz)
    begin
       if ( Clk_MHz'event and Clk_MHz = '1') then
          if ( Echo1 = '1') then
             Dist1 <= Dist1 + 1;
          else
             Dist1 <= "000000000000000";
          end if;

          if ( Echo1 = '0') then
             if ( Dist1 = "000000000000000") then
```

```
            else
               tempd1 <= std_logic_vector(to_unsigned(to_integer(unsigned( Dist1 ) / 58), 9));
                    Dist_cm <= to_integer(unsigned( tempd1));
                    -- Divide echo time by 58 to obtain distance in cm
            end if;
         end if;
      end if;
   end process;

PROCESS (Clk_MHz)
   begin
      if ( Clk_MHz'event and Clk_MHz = '1') then
         if ( Echo2 = '1') then
            Dist2 <= Dist2 + 1;
         else
            Dist2 <= "000000000000000";
         end if;

         if ( Echo2 = '0') then
            if ( Dist2 = "000000000000000") then
            else
               tempd2 <= std_logic_vector(to_unsigned(to_integer(unsigned( Dist2 ) / 58), 9));
                    Dist2_cm <= to_integer(unsigned( tempd2));
                   -- Divide echo time by 58 to obtain distance in cm
            end if;
         end if;
      end if;
   end process;

   process (Dist_cm, Dist2_cm)
   begin
     if ( (Dist_cm > 2 and Dist_cm <= 20) and (Dist2_cm =0 or Dist2_cm>20)) then
            Eye_Movement <= "01";
            elsif ( (Dist_cm = 0 or Dist_cm>20) and (Dist2_cm > 2 and Dist2_cm <= 20)) then
```

```
                Eye_Movement <= "10";
                elsif ( (Dist_cm > 2 and Dist_cm <= 20) and (Dist2_cm > 2 and Dist2_cm <= 20))
then
                Eye_Movement <= "11";
                else
                Eye_Movement <= "00";
        end if;
    end process;

end Behavioral;
```

tone_gen.vhd

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


entity tone_gen is
    Port (  Clock, reset: in std_logic;
            EM_TG : in std_logic_vector(1 downto 0);
            audio_out : out std_logic);
end tone_gen;

architecture Behavioral of tone_gen is
    signal Y : std_logic_vector(17 downto 0) := (others => '0');
    signal audio_pulse : std_logic := '0';
    signal clear : std_logic;
    signal ffin, ffout : unsigned(17 downto 0);

-- Standard Notes
            constant C4  : std_logic_vector := "101110101010001001";  -- 191113 (261.626 Hz)
    constant D4  : std_logic_vector := "101001100100010110";  -- 170262 (293.665 Hz)
    constant E4  : std_logic_vector := "100101000010000110";  -- 151686 (329.228 Hz)
    constant F4  : std_logic_vector := "100010111101000101";  -- 143173 (349.228 Hz)
    constant G4  : std_logic_vector := "011111001001000001";  -- 127553 (391.995 Hz)
    constant A4  : std_logic_vector := "011011101111100100";  -- 113636 (440.0 Hz)
    constant B4  : std_logic_vector := "011000101101110111";  -- 101239 (493.883 Hz)
            constant FA  : std_logic_vector := "000011000011010100";  -- 12500 (4000 Hz)

 begin

process (Clock, reset)
begin
    if (reset='1') then
```

```
        ffout <= (others => '0');
    elsif rising_edge(Clock) then
        ffout <= ffin;
    end if;
end process;

ffin <= (others => '0') when clear = '1'
         else ffout+1;
-- output
Y <= std_logic_vector(ffout);

process(Y, EM_TG, audio_pulse)
    begin
        case (EM_TG) is
            when "01" =>
                if(Y >= A4) then
                    clear <= '1';
                    audio_pulse <= not audio_pulse;
                else
                    clear <= '0';
                end if;
            when "10" =>
                if(Y >= D4) then
                    clear <= '1';
                    audio_pulse <= not audio_pulse;
                else
                    clear <= '0';
                end if;
            when "00" =>
                if(Y >= F4) then
                    clear <= '1';
                    audio_pulse <= not audio_pulse;
                else
                    clear <= '0';
```

```
            end if;

        when others =>
            audio_pulse <= '0';
            clear <= '1';
    end case;
audio_out <= audio_pulse;
end process;
end Behavioral;
```

seven_seg.vhd

```
--Eyes For Testing Purposes
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity seven_seg is
    Port ( clk : in STD_LOGIC;
           eye_mov : in STD_LOGIC_VECTOR(1 downto 0);
           an : out STD_LOGIC_VECTOR(3 downto 0);
           cath : out STD_LOGIC_VECTOR(0 to 6));
end seven_seg;

architecture Behavioral of seven_seg is
signal mux : STD_LOGIC_VECTOR(3 downto 0);

begin

process(clk)
begin
    if rising_edge(clk) then
        mux <= mux + 1;
    end if;
end process;

process(mux)
begin
    if eye_mov = "00" then
        if mux = "00" then
            an <= "0111";
            cath <= "0110001";
        elsif mux = "01" then
            an <= "1011";
```

```
        cath <= "0000110";
    elsif mux = "10" then
        an <= "1101";
        cath <= "0110000";
    elsif mux = "11" then
        an <= "1110";
        cath <= "0000111";
    end if;
elsif eye_mov = "01" then
    if mux = "00" then
        an <= "0111";
        cath <= "0110000";
    elsif mux = "01" then
        an <= "1011";
        cath <= "0000111";
    elsif mux = "10" then
        an <= "1101";
        cath <= "0110000";
    elsif mux = "11" then
        an <= "1110";
        cath <= "0000111";
    end if;
 elsif eye_mov = "10" then
    if mux = "00" then
        an <= "0111";
        cath <= "0110001";
    elsif mux = "01" then
        an <= "1011";
        cath <= "0000110";
    elsif mux = "10" then
        an <= "1101";
        cath <= "0110001";
    elsif mux = "11" then
        an <= "1110";
```

```
            cath <= "0000110";
        end if;
    elsif eye_mov = "11" then
        if mux = "00" then
            an <= "0111";
            cath <= "0110001";
        elsif mux = "01" then
            an <= "1011";
            cath <= "0000110";
        elsif mux = "10" then
            an <= "1101";
            cath <= "0110000";
        elsif mux = "11" then
            an <= "1110";
            cath <= "0000111";
        end if;
end if;
end process;
end Behavioral;
```

Top_Design.vhd

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

entity Top_design is
    Port (
          clk: in STD_LOGIC;
          eye_design: in std_logic_vector (1 downto 0);
          H_sync : out STD_LOGIC; -- VGA Ports
          V_sync : out STD_LOGIC; -- VGA Ports
          VGA_RED: out STD_LOGIC_VECTOR (3 downto 0); -- VGA Ports
          VGA_BLUE : out STD_LOGIC_VECTOR (3 downto 0); -- VGA Ports
          VGA_GREEN : out STD_LOGIC_VECTOR (3 downto 0)); -- VGA Port
end Top_design;

architecture Behavioral of Top_design is

component VGAsync
    Generic (   Horizontal_FrameSize : natural;
                Vertical_FrameSize : natural
             );
    Port ( Clock : in STD_LOGIC;
          y_control : out STD_LOGIC_VECTOR (11 downto 0);
          x_control : out STD_LOGIC_VECTOR (11 downto 0);
          video_on : out STD_LOGIC;
          H_sync : out STD_LOGIC;
          V_sync : out STD_LOGIC
         );
end component;

COMPONENT VGA
   Generic (   Horizontal_FrameSize : natural;
               Vertical_FrameSize : natural
            );
```

```
  Port (  Clock : in STD_LOGIC;
          eye_design: in STD_logic_vector ( 1 downto 0);
          X_counter, Y_counter : in STD_LOGIC_VECTOR ( 11 downto 0);
          Video : in STD_LOGIC;
          VGA_RED, VGA_GREEN, VGA_BLUE : out STD_LOGIC_VECTOR ( 3 downto 0)
        );
end component;

signal clock50MHZ : STD_LOGIC := '0';
SIGNAL X_count : STD_LOGIC_VECTOR (11 downto 0);
SIGNAL Y_count : STD_LOGIC_VECTOR (11 downto 0);
SIGNAL Vodeo_on : STD_LOGIC;
SIGNAL Seconds, Minutes : INTEGER range 0 to 60;
SIGNAL Hours : INTEGER range 0 to 24;
SIGNAL Clock250MHz, reset : STD_LOGIC;
Signal dist,dist2 : STD_LOGIC_VECTOR(8 downto 0);
SIGNAL sync_tempK,sync_tempM,sync_temp1hz : STD_LOGIC := '1';
SIGNAL int1,int2, int3, int4, int5: INTEGER := 0;
SIGNAL button2 : std_logic := '0';

begin
clock_div50Mhz_and_50khz_and_clk1Hz: process(Clk)
begin
   if(Clk'event and Clk = '1') then -- 50 MHz Clk
      clock50MHZ <= not clock50MHZ;
   end if;
    if ( rising_edge( Clk)) then
       if ( int1 < 49) then  -- 1Mhz Clock
             int1 <= int1 + 1;
          else
             int1 <= 0;
             sync_tempM <= NOT sync_tempM;
          end if;
       end if;
```

```
    if ( rising_edge( Clk)) then
        if ( int2 < 49999) then -- 50Khz clock
            int2 <= int2 + 1;
        else
            int2 <= 0;
            sync_tempK <= NOT sync_tempK;
        end if;
     end if;

     if ( rising_edge( Clk)) then
            if ( int3 < 49999999) then -- 1hz clock
               int3 <= int3 + 1;
            else
               int3 <= 0;
               sync_temp1hz <= NOT sync_temp1hz;
            end if;
         end if;
end process;
    M1 : VGAsync  Generic map ( 1280, 1024)
                port map( Clock => Clk ,
                          y_control => Y_count ,
                          x_control => x_count ,
                          video_on => vodeo_on ,
                          H_sync => H_sync,
                          V_sync => V_sync
                         );

     M2 : VGA GENERIC MAP ( 1280, 1024)
                port map ( Clock => Clk ,
                          eye_design => eye_design,
                          x_counter => x_count ,
                          y_counter => Y_count ,
                          video => vodeo_on,
```

```
                        VGA_RED => VGA_RED ,
                        VGA_GREEN => VGA_GREEN ,
                        VGA_BLUE => VGA_BLUE
                       );
end Behavioral;
```

VGAsync.vhd

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

entity VGAsync is
    Generic (    Horizontal_FrameSize : natural;
                 Vertical_FrameSize : natural
             );
    Port ( Clock : in STD_LOGIC;
           Y_control : out STD_LOGIC_VECTOR ( 11 downto 0);
           X_control : out STD_LOGIC_VECTOR ( 11 downto 0);
           Video_on : out STD_LOGIC;
           H_sync : out STD_LOGIC;
           V_sync : out STD_LOGIC
          );
end VGAsync;

architecture Behavioral of VGAsync is
--1280x1024@60Hz--
    CONSTANT FRAME_WIDTH : natural := Horizontal_FrameSize;
    CONSTANT FRAME_HEIGHT : natural := Vertical_FrameSize;

    CONSTANT H_FP : natural := 48; --H front porch width (pixels)
    CONSTANT H_PW : natural := 112; --H sync pulse width (pixels)
    CONSTANT H_MAX : natural := 1688; --H total period (pixels)

    CONSTANT V_FP : natural := 1; --V front porch width (lines)
    CONSTANT V_PW : natural := 3; --V sync pulse width (lines)
    CONSTANT V_MAX : natural := 1066; --V total period (lines)

    CONSTANT H_POL : std_logic := '1';
    CONSTANT V_POL : std_logic := '1';
```

```
    SIGNAL Active  : std_logic;

    -- Horizontal and Vertical counters
    SIGNAL X_cntr_reg : std_logic_vector ( 11 downto 0) := ( others =>'0');
    SIGNAL Y_cntr_reg : std_logic_vector ( 11 downto 0) := ( others =>'0');

    -- Horizontal and Vertical Sync
    SIGNAL H_sync_reg : std_logic := NOT H_POL;
    SIGNAL V_sync_reg : std_logic := NOT V_POL;

begin

-- Generate Horizontal, Vertical counters and the Sync signals
    -- Horizontal counter
    process ( Clock)
    begin
        if ( rising_edge( Clock)) then
            if ( X_cntr_reg = (H_MAX - 1)) then
                X_cntr_reg <= (others =>'0');
            else
                X_cntr_reg <= X_cntr_reg + 1;
            end if;
        end if;
    end process;

    -- Vertical counter
    process ( Clock)
    begin
        if ( rising_edge( Clock)) then
            if ( (X_cntr_reg = (H_MAX - 1)) and (Y_cntr_reg = (V_MAX - 1))) then
                Y_cntr_reg <= (others =>'0');
            elsif (X_cntr_reg = (H_MAX - 1)) then
                Y_cntr_reg <= Y_cntr_reg + 1;
```

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

    --Horizontal sync
    process ( Clock)
    begin
        if ( rising_edge( Clock)) then
            if ( X_cntr_reg >= (H_FP + FRAME_WIDTH - 1)) and (X_cntr_reg < (H_FP +
FRAME_WIDTH + H_PW - 1)) then
                H_sync_reg <= H_POL;
            else
                H_sync_reg <= not(H_POL);
            end if;
        end if;
    end process;

    -- Vertical sync
    process ( Clock)
    begin
        if ( rising_edge( Clock)) then
            if ( Y_cntr_reg >= (V_FP + FRAME_HEIGHT - 1)) and (Y_cntr_reg < (V_FP +
FRAME_HEIGHT + V_PW - 1)) then
                V_sync_reg <= V_POL;
            else
                V_sync_reg <= not(V_POL);
            end if;
        end if;
    end process;

    -- active signal
    Active <= '1'   when ( X_cntr_reg < FRAME_WIDTH and Y_cntr_reg < (FRAME_HEIGHT))
                else '0';
```

```
    Y_control <= Y_cntr_reg ;
    X_control <= X_cntr_reg ;
    Video_on <= Active ;
    H_sync <= H_sync_reg ;
    V_sync <= V_sync_reg ;

end Behavioral;
```

VGA.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;

entity VGA is
    Generic (    Horizontal_FrameSize : natural;
                 Vertical_FrameSize : natural
             );
    Port (  Clock : in STD_LOGIC;
            eye_design: in std_logic_vector(1 downto 0);
            X_counter, Y_counter : in STD_LOGIC_VECTOR ( 11 downto 0);
            Video : in STD_LOGIC;
            VGA_RED, VGA_GREEN, VGA_BLUE : out STD_LOGIC_VECTOR ( 3 downto 0)
          );
end VGA;

architecture Behavioral of VGA is

    SIGNAL X, Y : std_logic_vector ( 11 downto 0);
    SIGNAL Pixel, Box,Box2,Box3,box4,box5,box6, box7, box8, Go, chance : std_logic := '0';
    SIGNAL no: std_logic:= '1';
    SIGNAL eye : std_logic_vector (1 downto 0);
    SIGNAL VGA_red_o : std_logic_vector ( 3 downto 0);
    SIGNAL VGA_green_o : std_logic_vector ( 3 downto 0);
    SIGNAL VGA_blue_o : std_logic_vector ( 3 downto 0);
    SIGNAL VGA_red_p : std_logic_vector ( 3 downto 0);
    SIGNAL VGA_green_p : std_logic_vector ( 3 downto 0);
    SIGNAL VGA_blue_p : std_logic_vector ( 3 downto 0);
    -- VGA R, G and B signals to connect output with the design
    SIGNAL VGA_red_comb : std_logic_vector ( 3 downto 0);
    SIGNAL VGA_green_comb : std_logic_vector ( 3 downto 0);
    SIGNAL VGA_blue_comb : std_logic_vector ( 3 downto 0);
```

```
begin
eye <= eye_design;

 process ( Clock, eye)
 begin

 if (rising_edge(clock)) then
 if ( eye = "00") then

        if ( x >= 610 and x <= 710 and  y >= 425 and y <= 475 ) then
           box <= '1';
           else
           box <= '0';
           end if;
    end if;
 if (  eye = "11") then

        if ( x >= 610 and x <= 710 and  y >= 425 and y <= 475 ) then
           box <= '1';
           else
           box <= '0';
           end if;
    end if;

if ( eye = "01") then
        if ( x >= 270 and x <= 370 and  y >= 425 and y <= 475 ) then
              box <= '1';
           else
           box <= '0';

           end if;
 end if;
```

```
 if ( eye = "10") then



        if ( x > 910 and x <= 1010 and  y >= 425 and y <= 475 ) then
              box <= '1';
           else
           box <= '0';
          end if;
    end if;
end if;
end process;

process(clock)
begin
        if (( x > 250 and x < 1030) and ( y > 200 and y < 210))   then
              box7 <= '1';
           else
              box7 <= '0';
          end if;
        if (x > 250 and x < 1030 ) and ( y > 700 and y < 710)   then
                box8 <= '1';
           else
                box8 <= '0';
           end if;
        if (( x > 250 and x < 260) and ( y > 200 and y < 710))   then
              box6 <= '1';
           else
              box6 <= '0';
          end if;
        if (x > 1020 and x < 1030 ) and ( y > 200 and y < 710)   then
                box2 <= '1';
           else
                box2 <= '0';
        end if;
```

```
 end process;

    X <= X_counter;
    Y <= Y_counter;

     VGA_red_o <= "1111";--red;
     VGA_green_o <= "1111";--green;
     VGA_blue_o <= "1111";--blue
     no <=  box or box2 or box6 or box7 OR box8 ;

process(clock)
begin
if rising_edge(clock) then

if (no = '1') then
    VGA_red_comb <= (no & no & no & no) and VGA_red_o;
    VGA_green_comb <= (no & no & no & no) and VGA_green_o;
    VGA_blue_comb <= (no & no & no & no) and VGA_blue_o;

   else
    VGA_blue_comb <= "0000";
    VGA_red_comb <= "0000" ;
    VGA_green_comb <= "0000";
 end if;
end if;
end process;

process ( Clock)
begin
       if ( rising_edge( Clock ) and Video = '1') then
          VGA_RED <= vga_red_comb;
          VGA_GREEN <= vga_green_comb;
          VGA_BLUE <= vga_blue_comb;
       end if;
```

```
    end process;
end Behavioral;
```

Constraints.xdc

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


## LEDs
set_property PACKAGE_PIN U16 [get_ports {LED[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]

set_property PACKAGE_PIN E19 [get_ports {LED[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]


##Buttons
set_property PACKAGE_PIN U18 [get_ports reset]
        set_property IOSTANDARD LVCMOS33 [get_ports reset]


##Pmod Header JA

##Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports {Echo1}]
set_property IOSTANDARD LVCMOS33 [get_ports {Echo1}]

##Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {Trig1}]
set_property IOSTANDARD LVCMOS33 [get_ports {Trig1}]

##Sch name = JA3
set_property PACKAGE_PIN J2 [get_ports {Echo2}]
set_property IOSTANDARD LVCMOS33 [get_ports {Echo2}]
```

```
##Sch name = JA4
set_property PACKAGE_PIN G2 [get_ports {Trig2}]
set_property IOSTANDARD LVCMOS33 [get_ports {Trig2}]

##Sch name = JA7
set_property PACKAGE_PIN H1 [get_ports {audio_out}]
        set_property IOSTANDARD LVCMOS33 [get_ports {audio_out}]

#7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]


set_property PACKAGE_PIN U2 [get_ports {an[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
```

```
#VGA Connector
#Bank = 14, Pin name = ,                                    Sch name = VGA_R0
set_property PACKAGE_PIN G19 [get_ports {VGA_RED[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_RED[0]}]
#Bank = 14, Pin name = ,                                    Sch name = VGA_R1
set_property PACKAGE_PIN H19 [get_ports {VGA_RED[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_RED[1]}]
#Bank = 14, Pin name = ,                                    Sch name = VGA_R2
set_property PACKAGE_PIN J19 [get_ports {VGA_RED[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_RED[2]}]
#Bank = 14, Pin name = ,                                    Sch name = VGA_R3
set_property PACKAGE_PIN N19 [get_ports {VGA_RED[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_RED[3]}]
#Bank = 14, Pin name = ,                                    Sch name = VGA_B0
set_property PACKAGE_PIN N18 [get_ports {VGA_BLUE[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_BLUE[0]}]
#Bank = 14, Pin name = ,                                          Sch name = VGA_B1
set_property PACKAGE_PIN L18 [get_ports {VGA_BLUE[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_BLUE[1]}]
#Bank = 14, Pin name = ,                                    Sch name = VGA_B2
set_property PACKAGE_PIN K18 [get_ports {VGA_BLUE[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_BLUE[2]}]
#Bank = 14, Pin name = ,                                          Sch name = VGA_B3
set_property PACKAGE_PIN J18 [get_ports {VGA_BLUE[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_BLUE[3]}]
#Bank = 14, Pin name = ,                                    Sch name = VGA_G0
set_property PACKAGE_PIN J17 [get_ports {VGA_GREEN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_GREEN[0]}]
#Bank = 14, Pin name = ,                                Sch name = VGA_G1
set_property PACKAGE_PIN H17 [get_ports {VGA_GREEN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_GREEN[1]}]
#Bank = 14, Pin name = ,                                    Sch name = VGA_G2
set_property PACKAGE_PIN G17 [get_ports {VGA_GREEN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_GREEN[2]}]
```

```
#Bank = 14, Pin name = ,                    Sch name = VGA_G3
set_property PACKAGE_PIN D17 [get_ports {VGA_GREEN[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGA_GREEN[3]}]
#Bank = 14, Pin name = ,                              Sch name = VGA_HS
set_property PACKAGE_PIN P19 [get_ports H_sync]
set_property IOSTANDARD LVCMOS33 [get_ports H_sync]
#Bank = 14, Pin name = ,                    Sch name = VGA_VS
set_property PACKAGE_PIN R19 [get_ports V_sync]
set_property IOSTANDARD LVCMOS33 [get_ports V_sync]

set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]

set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
```