EEE102 Introduction to Digital Circuit Design Project Report

YouTube Link

https://www.youtube.com/watch?v=5W7BYjQx-zw

Abstract / Objective

Purpose of this project is to generate various analog sound waves in different frequencies from digital signal of the BASYS3, basically a synthesizer. Thus, one should know the concepts of digital design such as Pulse With Modulation (PWM), Video Graphics Array (VGA), 7-Segment Display, Clock Division. Also, some physics knowledge should be used to define the frequencies of various sounds to be played on the BASYS3. Procedure for the project is to create sound waves by using PWM, then adding them to get output wave and finally altering the user interface to give information about the device's state.

Design Specification Plan

We must use PWM to generate analog sound waves from square waves. To get a variety of frequency we must define 12 switch to each note in an octave, and assign 3 switches to change the octave which our sound waves are in. With this we can generate multiple sound waves and add them together to generate a polyphonic sound wave. To indicate which notes are playing at the time, we have two user interface component. One is 7 segment display, which indicates the octave number and a note indicator. Note indicator on 7 segment display works only if one note is played at the time, because to indicate more than one note on 7 segment display, one should encode each chord pattern to the seven segment decoder which would not be efficient and would be impossible with 4 output 7 segment display. Other user interface component is a 1280 * 1024 resolutions 60Hz VGA display. On this display a piano is designed so that user can identify which notes are played at

the time by looking at the note indicator at the screen. On the screen we have a piano and whenever a note is played, the corresponding note on the piano is highlighted.

Design Methodology

Components: BASYS3 FPGA Board, Audio Amplifier, VGA Input 60 Hz screen, VGA Cable.

Modules: To create sound waves we must have a wave generator module. Another module is needed to sum these generated sound waves. Rest of the modules are needed to construct a user interface.

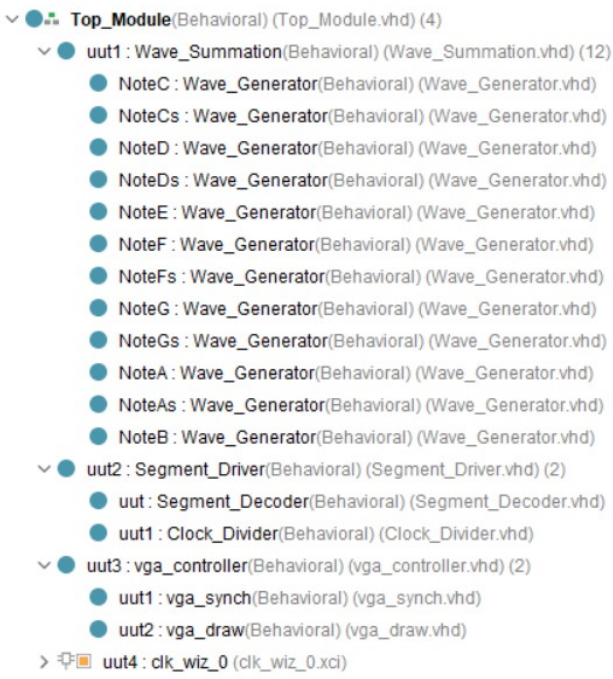

Figure 1: Modules and Submodules of Synthesizer Design

RTL Diagram: These diagrams the logic circuit equivalent of the VHDL code of the corresponding modules.

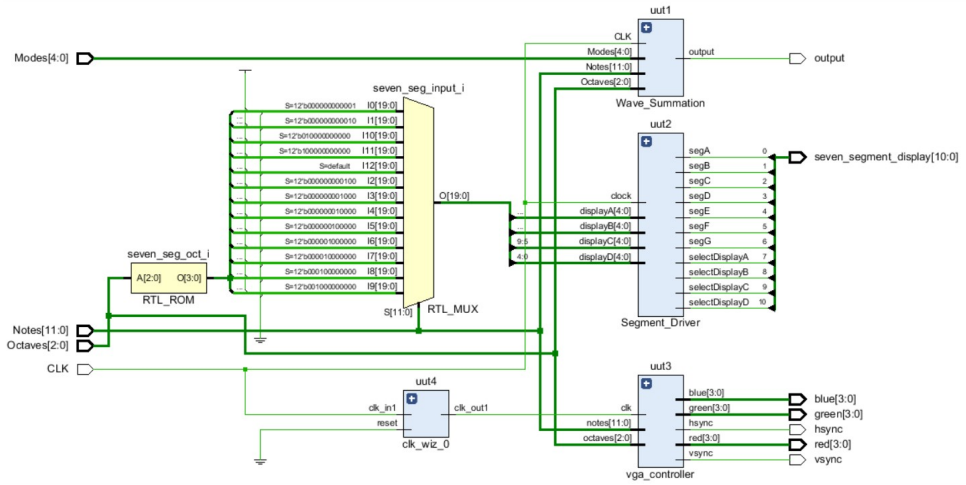

Figure 2: RTL Schmeatics of overall design

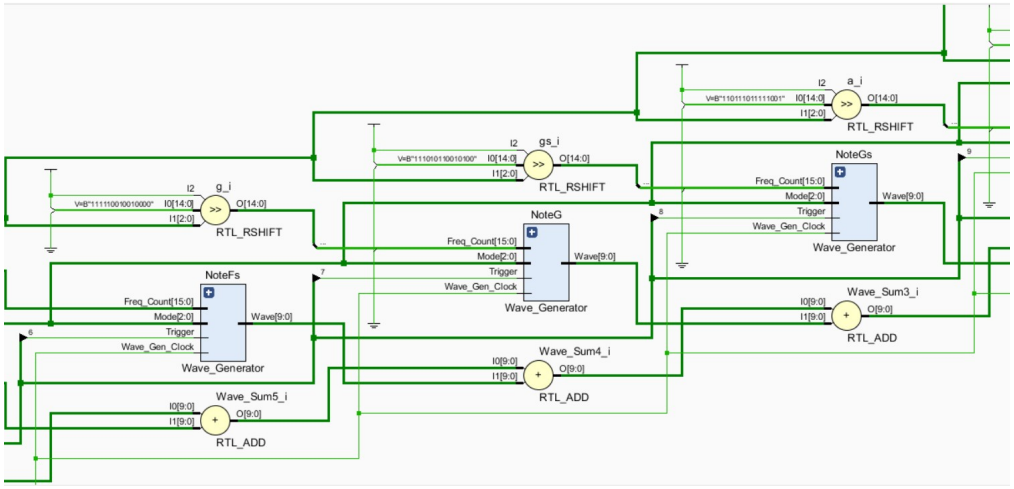

Figure 3: Wave Summation RTL Schematic

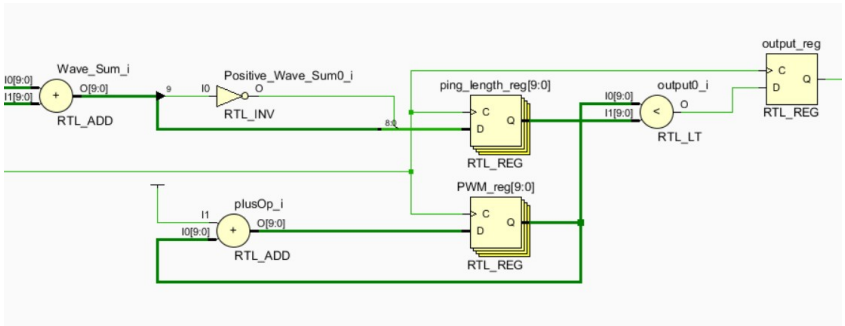

Figure 4: RTL Schematics of PWM part of Wave Summation

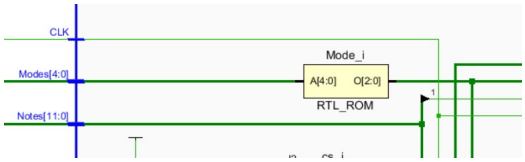

Figure 5: Mode Selector RTL Schematic

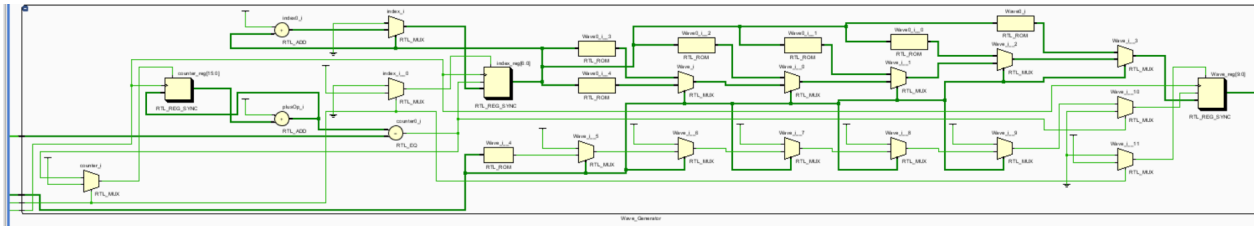

Figure 6: Wave Generator RTL Schematic

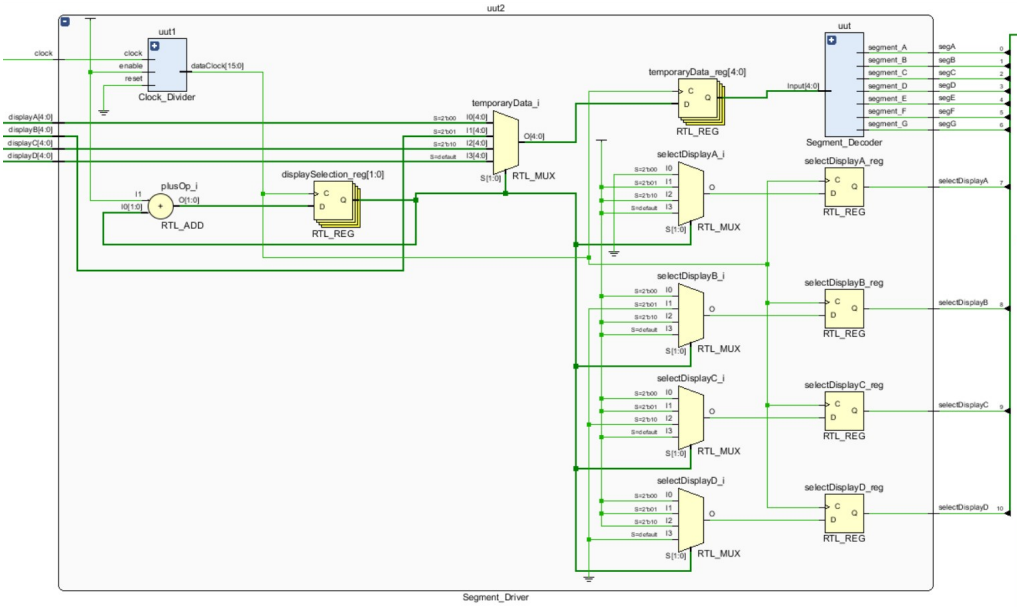

Figure 7: Segment Driver RTL Schematic

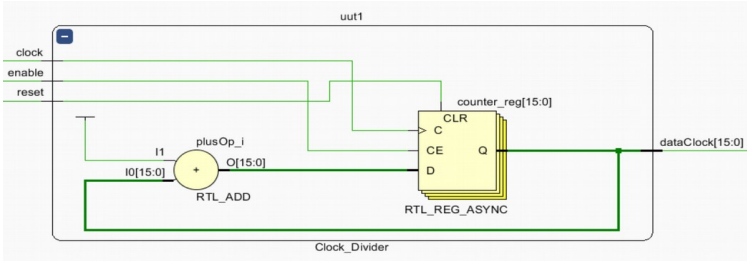

Figure 8: Clock Divider RTL Schematic

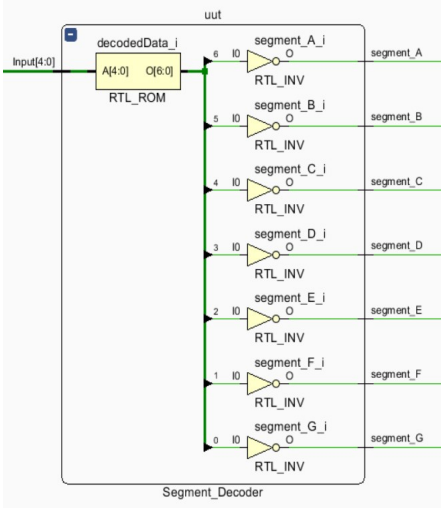

Figure 9: Segment Decoder RTL Schematic

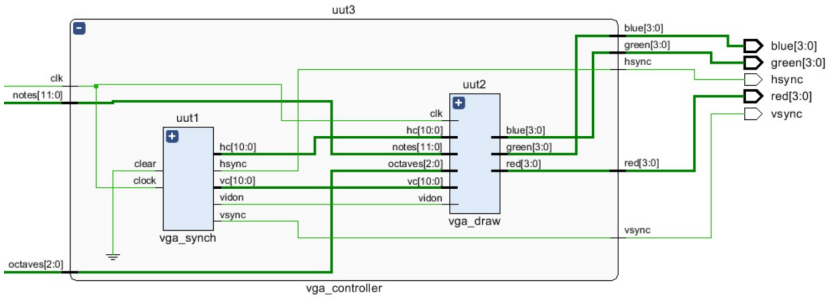

Figure 10: VGA Controller RTL Schematic

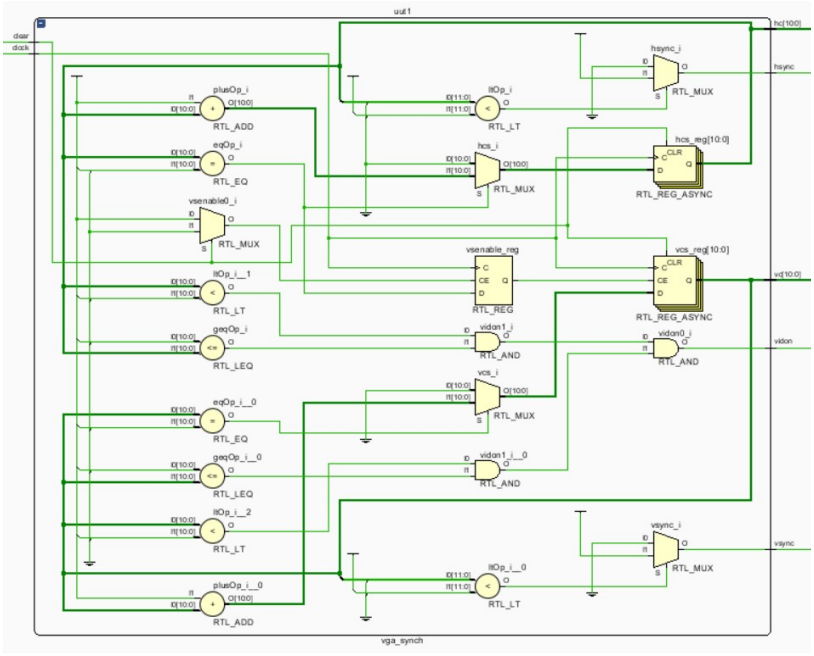

Figure 11: VGA Synchronizations RTL Schematic

Results

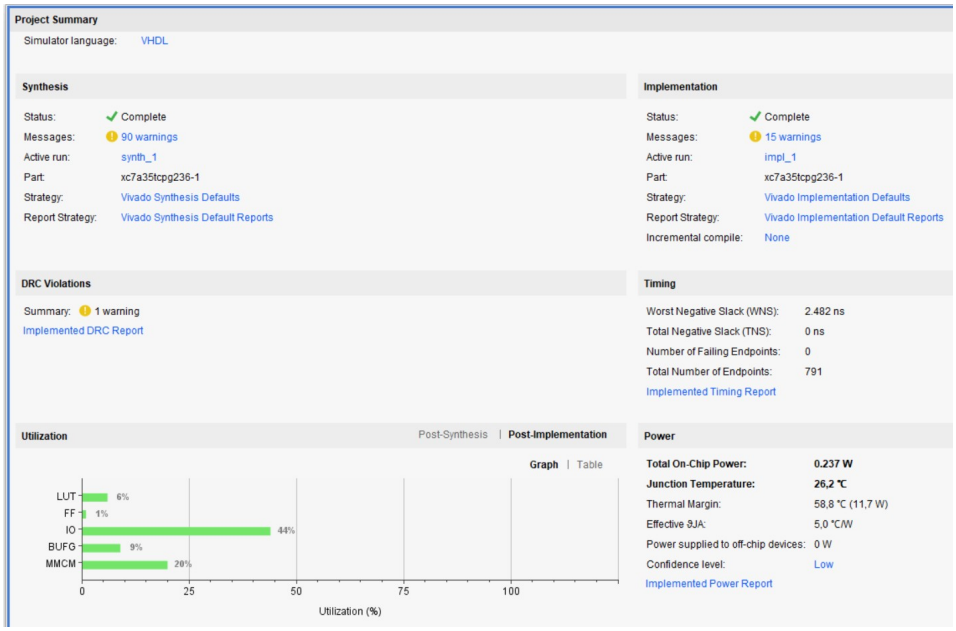

Figure 12: Project Summary

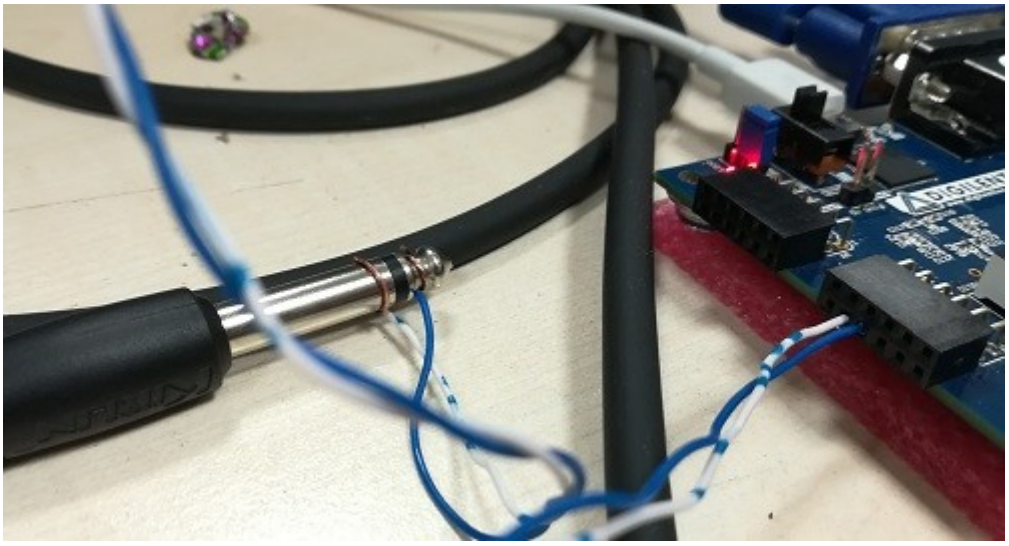

Figure 13: Pin output to amplifier

Figure 14: Amplifier input

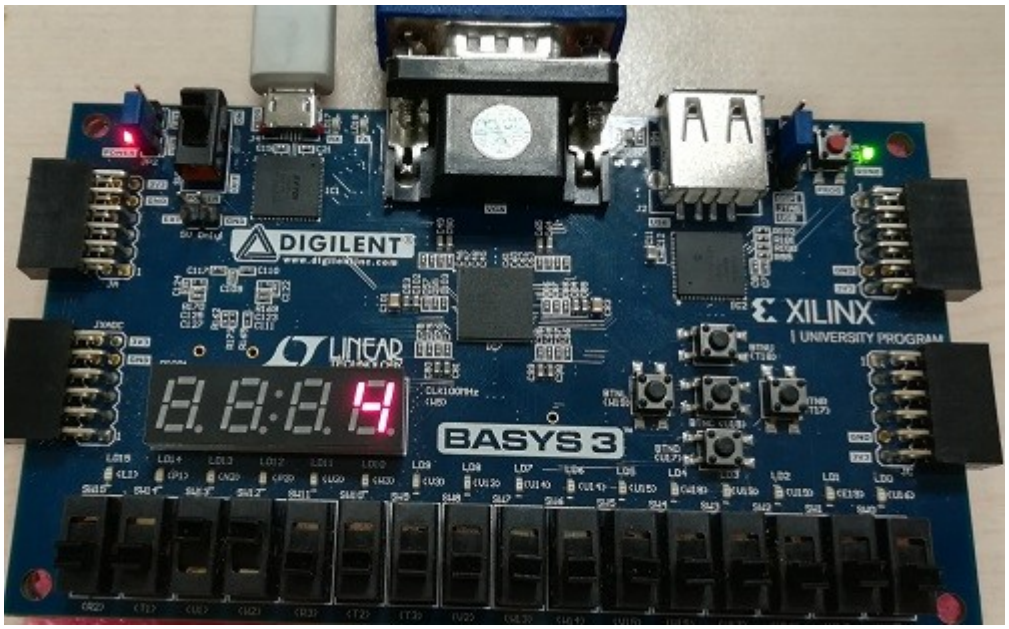

Figure 15: zero input at 4th octave

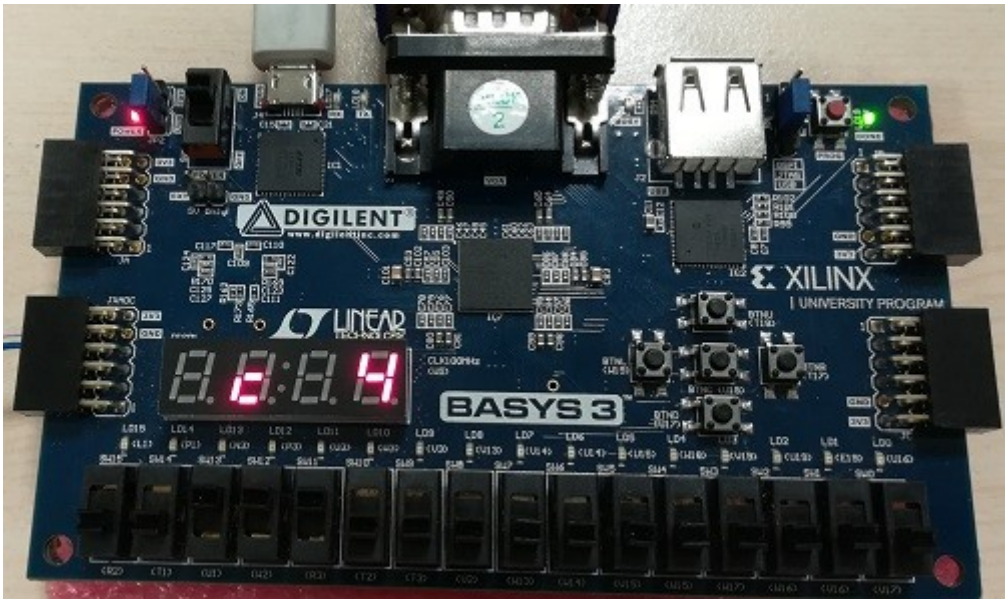

Figure 16: One input at 4th octave (C - Do Note)

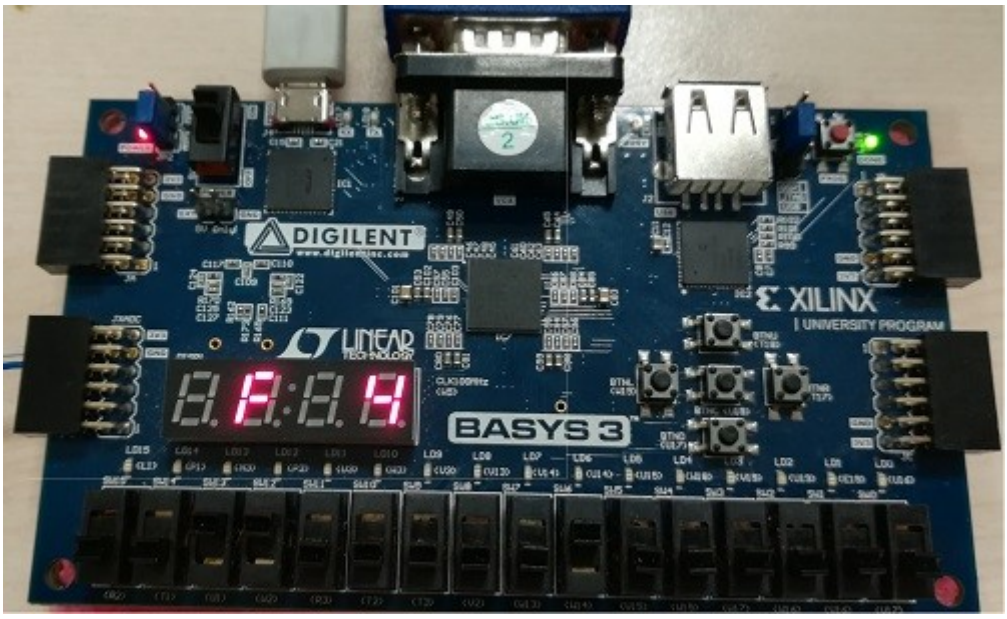

Figure 17: One input at 4th octave (F - Fa Note)

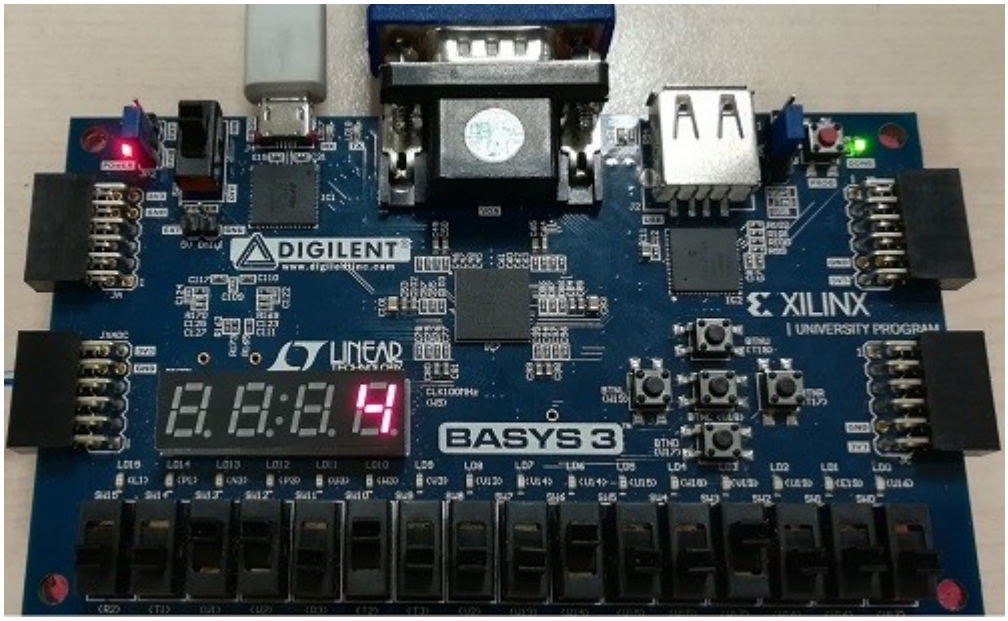

Figure 18: Multiple input at 4th octave (C# - F - G#)

Figure 18: Zero input VGA output

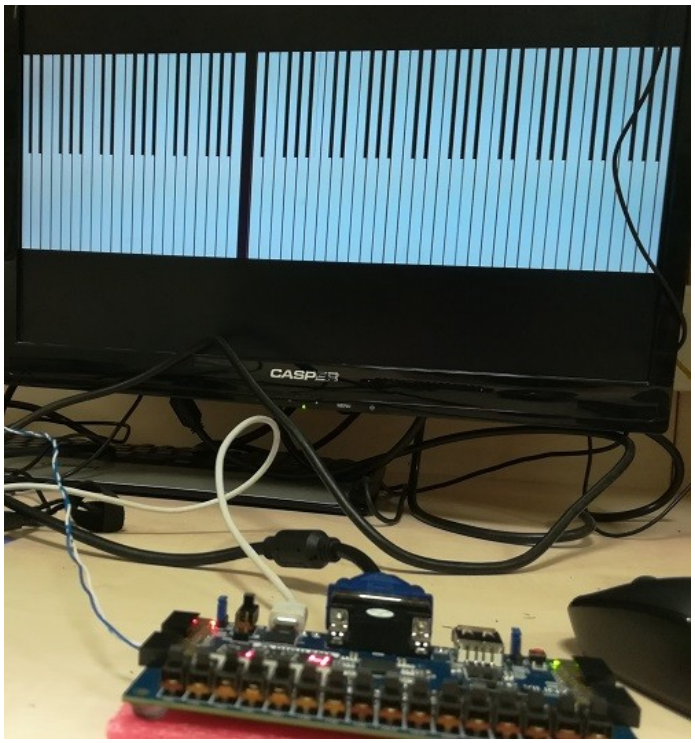

Figure 19: One input at 4th octave (C Note)

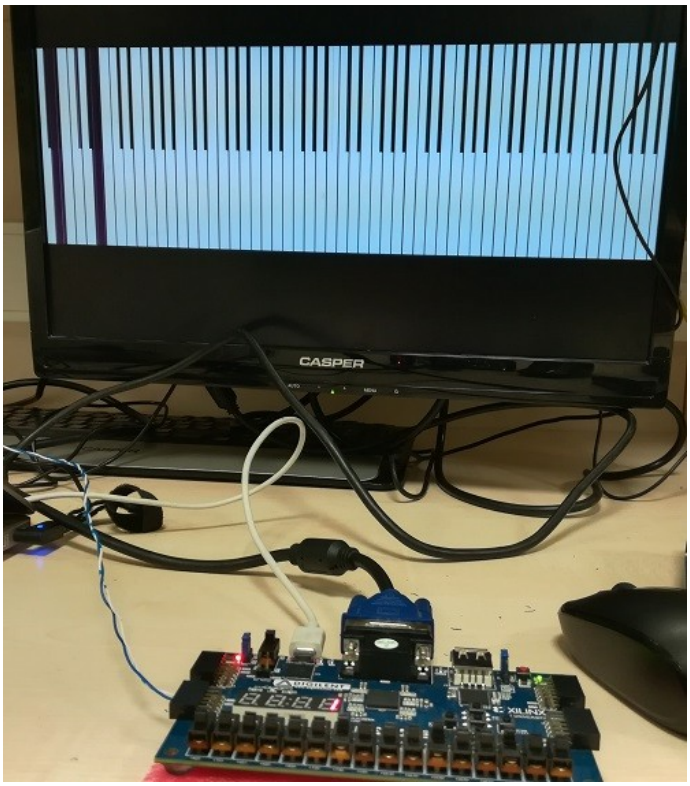

Figure 20: Multiple input at 1st octave (D - D# - A)

Conclusion

From digital waves we can observe analog outputs such as sinusoidal sound waves by using PWM. This PWM ping values can be stored in a ROM (sinusoidal, sawtooth, triangle waves) to be recalled by the code for PWM value at the current time. The generation of wave depends on the corresponding switch. This switches determines which notes to play at which octave. After generation we can add each wave to get output PWM ping length, in other words we can get a polyphonic synthesizer.

Appendices

Top_Module.vhd

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

entity Top_Module is
  Port ( CLK: in std_logic;
         Notes : in STD_LOGIC_VECTOR(11 downto 0);
         Octaves : in STD_LOGIC_VECTOR(2 downto 0);
         Modes : in STD_LOGIC_VECTOR(4 DOWNTO 0);
         seven_segment_display : out std_logic_vector(10 downto 0);
         output : out std_logic;
         hsync : out std_logic;
         vsync : out std_logic;
         red : out std_logic_vector(3 downto 0);
         green : out std_logic_vector(3 downto 0);
         blue : out std_logic_vector(3 downto 0));
end Top_Module;

architecture Behavioral of Top_Module is
    component Wave_Summation is
    Port (  CLK: in std_logic;
            Notes: in std_logic_vector(11 downto 0);
            Octaves: in std_logic_vector(2 downto 0);
            Modes : in std_logic_vector(4 downto 0);
            output: out std_logic);
    end component;

    component Segment_Driver is
    Port ( displayA : in STD_LOGIC_VECTOR (4 downto 0);
           displayB : in STD_LOGIC_VECTOR (4 downto 0);
           displayC : in STD_LOGIC_VECTOR (4 downto 0);
           displayD : in STD_LOGIC_VECTOR (4 downto 0);
           clock : in STD_LOGIC;
           segA : out STD_LOGIC;
           segB : out STD_LOGIC;
           segC : out STD_LOGIC;
           segD : out STD_LOGIC;
           segE : out STD_LOGIC;
           segF : out STD_LOGIC;
           segG : out STD_LOGIC;
           selectDisplayA : out STD_LOGIC;
           selectDisplayB : out STD_LOGIC;
           selectDisplayC : out STD_LOGIC;
           selectDisplayD : out STD_LOGIC);
    end component;

    component vga_controller is
        Port (  clk : in std_logic;
                notes: in std_logic_vector(11 downto 0);
                octaves : in std_logic_vector(2 downto 0);
                hsync : out std_logic;
                vsync : out std_logic;
                red : out std_logic_vector(3 downto 0);
                green : out std_logic_vector(3 downto 0);
```

```
             blue : out std_logic_vector(3 downto 0));
    end component;

    component clk_wiz_0 is
        Port ( clk_out1 : out std_logic;
               reset : in std_logic;
               locked : out std_logic;
               clk_in1 : in std_logic);
    end component;

    signal seven_seg_oct : std_logic_vector(4 downto 0);
    signal koutput : std_logic_vector(10 downto 0);
    signal seven_seg_input : std_logic_vector(19 downto 0);
    signal clk108Mhz : std_logic;
    signal locked : std_logic;

begin

    uut1 : wave_summation port map(
        CLK => CLK,
        Notes => Notes,
        Octaves => Octaves,
        Modes => Modes,
        output => output
    );

    uut2 : segment_driver port map(
        displayA => std_logic_vector(seven_seg_input(19 downto 15)),
        displayB => std_logic_vector(seven_seg_input(14 downto 10)),
        displayC => std_logic_vector(seven_seg_input(9 downto 5)),
        displayD => std_logic_vector(seven_seg_input(4 downto 0)),
        clock => CLK,
        segA =>  seven_segment_display(0),
        segB =>  seven_segment_display(1),
        segC =>  seven_segment_display(2),
        segD =>  seven_segment_display(3),
        segE =>  seven_segment_display(4),
        segF =>  seven_segment_display(5),
        segG =>  seven_segment_display(6),
        selectDisplayA =>  seven_segment_display(7),
        selectDisplayB =>  seven_segment_display(8),
        selectDisplayC =>  seven_segment_display(9),
        selectDisplayD =>  seven_segment_display(10)
    );

    uut3 : vga_controller Port Map(
        clk => CLK108Mhz,
        notes => Notes,
        octaves => Octaves,
        hsync => hsync,
        vsync => vsync,
        red => red,
        green => green,
        blue => blue
```

```
    );

    uut4 : clk_wiz_0 Port Map(
        clk_out1 => clk108Mhz,
        reset => '0',
        locked => locked,
        clk_in1 => clk
    );

    with Octaves  select seven_seg_oct <=
        "00111" when "000",
        "01000" when "001",
        "01001" when "010",
        "01010" when "011",
        "01011" when "100",
        "01100" when "101",
        "01101" when "110",
        "01110" when "111",
        "11111" when others;

    with Notes select seven_seg_input <=
        seven_seg_oct & "10000" & "00010" & "11111" when "000000000001",
        seven_seg_oct & "01111" & "00010" & "11111" when "000000000010",
        seven_seg_oct & "10000" & "00011" & "11111" when "000000000100",
        seven_seg_oct & "01111" & "00011" & "11111" when "000000001000",
        seven_seg_oct & "10000" & "00100" & "11111" when "000000010000",
        seven_seg_oct & "10000" & "00101" & "11111" when "000000100000",
        seven_seg_oct & "01111" & "00101" & "11111" when "000001000000",
        seven_seg_oct & "10000" & "00110" & "11111" when "000010000000",
        seven_seg_oct & "01111" & "00110" & "11111" when "000100000000",
        seven_seg_oct & "10000" & "00000" & "11111" when "001000000000",
        seven_seg_oct & "01111" & "00000" & "11111" when "010000000000",
        seven_seg_oct & "10000" & "00001" & "11111" when "100000000000",
        seven_seg_oct & "111111111111111" when others;

end Behavioral;
```

Wave_Summation.vhd

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

entity Wave_Summation is
Port (  CLK: in std_logic;
        Notes: in std_logic_vector(11 downto 0);
        Octaves: in std_logic_vector(2 downto 0);
        Modes: in std_logic_vector(4 downto 0);
        output: out std_logic
        );
end Wave_Summation;

architecture Behavioral of Wave_Summation is
    component Wave_Generator is
```

```
      Port (Trigger : in STD_LOGIC;
            Mode : in STD_LOGIC_VECTOR(2 downto 0);
            Freq_Count : in STD_LOGIC_VECTOR (15 downto 0);
            Wave_Gen_Clock : in STD_LOGIC;
            Wave : out STD_LOGIC_VECTOR (9 downto 0));
    end component;

    -- For notes
    signal  WaveC, WaveCs, WaveD, WaveDs, WaveE, WaveF, WaveFs, WaveG, WaveGs,
WaveA, WaveAs, WaveB : signed(9 downto 0);
    signal c, cs, d, ds, e, f, fs, g, gs, a, as, b, tmp : unsigned(15 downto 0);
    signal Mode : std_logic_vector(2 downto 0);

    -- For Wave Summation
    signal Wave_Sum : STD_LOGIC_VECTOR(9 downto 0); --signal for summed sine
waves (2's compliment -512 to 511)
    signal Positive_Wave_Sum : STD_LOGIC_VECTOR(9 downto 0); --unsigned 0 to
1023, for use in PWM generator

    -- For PWM
    signal ping_length : unsigned (9 downto 0);
    signal PWM : unsigned (9 downto 0) := to_unsigned(0, 10);

begin

    NoteC   : Wave_Generator port map (  Trigger => Notes(0), Mode => Mode,
Freq_Count => std_logic_vector(c), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveC);
    NoteCs  : Wave_Generator port map (  Trigger => Notes(1), Mode => Mode,
Freq_Count => std_logic_vector(cs), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveCs);
    NoteD   : Wave_Generator port map (  Trigger => Notes(2), Mode => Mode,
Freq_Count => std_logic_vector(d), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveD);
    NoteDs  : Wave_Generator port map (  Trigger => Notes(3), Mode => Mode,
Freq_Count => std_logic_vector(ds), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveDs);
    NoteE   : Wave_Generator port map (  Trigger => Notes(4), Mode => Mode,
Freq_Count => std_logic_vector(e), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveE);
    NoteF   : Wave_Generator port map (  Trigger => Notes(5), Mode => Mode,
Freq_Count => std_logic_vector(f), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveF);
    NoteFs  : Wave_Generator port map (  Trigger => Notes(6), Mode => Mode,
Freq_Count => std_logic_vector(fs), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveFs);
    NoteG   : Wave_Generator port map (  Trigger => Notes(7), Mode => Mode,
Freq_Count => std_logic_vector(g), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveG);
    NoteGs  : Wave_Generator port map (  Trigger => Notes(8), Mode => Mode,
Freq_Count => std_logic_vector(gs), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveGs);
```

```
    NoteA   : Wave_Generator port map (  Trigger => Notes(9), Mode => Mode,
Freq_Count => std_logic_vector(a), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveA);
    NoteAs  : Wave_Generator port map (  Trigger => Notes(10), Mode => Mode,
Freq_Count => std_logic_vector(as), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveAs);
    NoteB   : Wave_Generator port map (  Trigger => Notes(11), Mode => Mode,
Freq_Count => std_logic_vector(b), Wave_Gen_Clock => CLK, signed(Wave) =>
WaveB);

    c <= "1011101010100110" srl to_integer(unsigned(Octaves));
    cs <= "1011000000100101" srl to_integer(unsigned(Octaves));
    d <= "1010011001000011" srl to_integer(unsigned(Octaves));
    ds <= "1001110011110001" srl to_integer(unsigned(Octaves));
    e <= "1001010000100100" srl to_integer(unsigned(Octaves));
    f <= "1000101111010100" srl to_integer(unsigned(Octaves));
    fs <= "1000001111110111" srl to_integer(unsigned(Octaves));
    g <= "0111110010010000" srl to_integer(unsigned(Octaves));
    gs <= "0111010110010100" srl to_integer(unsigned(Octaves));
    a <= "0110111011111001" srl to_integer(unsigned(Octaves));
    as <= "0110100010111110" srl to_integer(unsigned(Octaves));
    b <= "0110001011011011" srl to_integer(unsigned(Octaves));

    with Modes select Mode <=
        "001" when "10000",
        "010" when "01000",
        "011" when "00100",
        "100" when "00010",
        "101" when "00001",
        "000" when others;

    Wave_Sum <= std_logic_vector((WaveC + WaveCs + WaveD + WaveDs + WaveE +
WaveF + WaveFs + WaveG + WaveGs + WaveA + WaveAs + WaveB));

    Positive_Wave_Sum <= not Wave_Sum(9) & Wave_Sum(8 downto 0);

    process(CLK)
        begin
        if (rising_edge(CLK)) then
            if (PWM < ping_length) then
                output <= '1';
            else
                output <= '0';
            end if;
            PWM <= PWM + 1;
            ping_length <= unsigned(Positive_Wave_Sum);
        end if;
    end process;
end Behavioral;
```

Wave Generator.vhd

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Wave_Generator is
    Port ( Trigger : in STD_LOGIC;
           Mode : in STD_LOGIC_VECTOR(2 downto 0);
           Freq_Count : in STD_LOGIC_VECTOR (15 downto 0);
           Wave_Gen_Clock : in STD_LOGIC;
           Wave : out STD_LOGIC_VECTOR (9 downto 0));
end Wave_Generator;

architecture Behavioral of Wave_Generator is

    signal index : integer range 0 to 64 := 0;
    type wave_memory is array (0 to 63) of integer range -63 to 64;
    signal amplitudeSin : wave_memory := (  0, 7, 13, 19, 25, 30, 35, 40, 45,
                                            49, 52, 55, 58, 60, 62, 63,
                                           63, 63, 62, 60, 58, 55, 52, 49, 45,
                                            40, 35, 30, 25, 19, 13, 7,
                                           0, -7,-13,-19,-25,-30,-35,-40,-45,-
                                            49,-52,-55,-58,-60,-62,-63,
                                           -63,-63,-62,-60,-58,-55,-52,-49,-
                                            45,-40,-35,-30,-25,-19,-13,-7);
    signal amplitudeSaw : wave_memory := (  64, 62, 60, 58, 56, 54, 52, 50, 48,
                                            46, 44, 42, 40, 38, 36, 34,
                                           32, 30, 28, 26, 24, 22, 20, 18, 16,
                                                   14, 12, 10, 8, 6, 4, 2,
                                           0, -2, -4, -6, -8, -10, -12, -14, -
                                         16, -18, -20, -22, -24, -26, -28, -30,
                                           -32, -34, -36, -38, -40, -42, -44, -
                                      46, -48, -50, -52, -54, -56, -58, -60, -62);
    signal amplitudeTri : wave_memory := (  0, 4, 8, 12, 16, 20, 24, 28, 32, 36,
                                                   40, 44, 48, 52, 56, 60,
                                           64, 60, 56, 52, 48, 44, 40, 36, 32,
                                            28, 24, 20, 16, 12, 8, 4,
                                           0, -4,-8,-12,-16,-20,-24,-28,-32,-
                                                   36,-40,-44,-48,-52,-56,-60,
                                           -63,-60,-56,-52,-48,-44,-40,-36,-
                                            32,-28,-24,-20,-16,-12,-8,-4);
    signal amplitudeViolin : wave_memory := (   -58, -55, -51,  -48, -45, -43, -
                                41, -41, -41, -42,-43, -44,  -44,  -44, -42, -39,
                                                -34, -27, -19, -10,0,  11, 22,
                                          32, 41, 49, 56, 60, 63, 64,63, 61,
                                                58, 55, 51, 48, 45, 43, 41,
                                          41,41, 42, 43, 44,  44, 44, 42,  39,
                                                34, 27,  19,  10,  0,  -11, -22,
                                -32, -41,  -49, -58, -62,-63, -63,-62,  -60);
    signal amplitudeSax : wave_memory := (60,  57,  52,  48,  42,  36,  28,  21,
                                          10,   2,  -7, -19, -27, -30, -32, -33,
                                             -33, -31, -27, -21, -15,  -6,   5,
                                       10,  16,  25,  28,  29,  29,  26,  23,  17,
                                              12,   5,  -2,  -9, -17, -26, -33, -
                                      42, -48, -53, -57, -61, -62, -63, -62, -61,
                                             -58, -50, -42, -34, -25, -17, -10,
                                      5,  14,  21,  31,  42,  46,  54,  58, 60);
```

```
    signal amplitudeFlute : wave_memory := (-62, -61, -56, -48, -40, -32, -23, -
                                    14,  -5,   4,  12,  20,  27,  32,  38,  42,
                                        44,  40,  34,  26,  17,  10,   6,
                                    5,   5,   6,   8,  12,  18,  22,  28,  33,
                                        39,  44,  49,  54,  58,  60,  62,
                                    62,  60,  55,  50,  42,  32,  21,  13,   8,
                                         4,   2,   0,  -1,  -1,  -1,  -1,
                                    0,   1,   2,   2,   1,  -2, -35, -51, -62);

begin

    process(Wave_Gen_Clock, Trigger)
        variable counter : unsigned(15 downto 0) := to_unsigned(0, 16);

    begin
        if rising_edge(Wave_Gen_Clock) then
            if Trigger = '1' then
                counter := counter + 1;
                if counter = unsigned(Freq_Count) then
                    counter := to_unsigned(0, 16);
                    if Mode = "000" then
                        Wave <= std_logic_vector(to_signed(amplitudeSin(index),
10));
                    elsif Mode = "001" then
                        Wave <= std_logic_vector(to_signed(amplitudeSaw(index),
10));
                    elsif Mode = "010" then
                        Wave <= std_logic_vector(to_signed(amplitudeTri(index),
10));
                    elsif Mode = "011" then
                        Wave <=
std_logic_vector(to_signed(amplitudeViolin(index), 10));
                    elsif Mode = "100" then
                        Wave <= std_logic_vector(to_signed(amplitudeSax(index),
10));
                    elsif Mode = "101" then
                        Wave <=
std_logic_vector(to_signed(amplitudeFlute(index), 10));
                    end if;
                    index <= index + 1;
                    if index = 63 then
                        index <= 0;
                    end if;
                end if;
            else
                Wave <= "0000000000";
                index <= 0;
                counter := to_unsigned(0, 16);
            end if;
        end if;
    end process;
end Behavioral;
```

Segment Driver.vhd

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

entity Segment_Driver is
    Port ( displayA : in STD_LOGIC_VECTOR (4 downto 0);
           displayB : in STD_LOGIC_VECTOR (4 downto 0);
           displayC : in STD_LOGIC_VECTOR (4 downto 0);
           displayD : in STD_LOGIC_VECTOR (4 downto 0);
           clock : in STD_LOGIC;
           segA : out STD_LOGIC;
           segB : out STD_LOGIC;
           segC : out STD_LOGIC;
           segD : out STD_LOGIC;
           segE : out STD_LOGIC;
           segF : out STD_LOGIC;
           segG : out STD_LOGIC;
           selectDisplayA : out STD_LOGIC;
           selectDisplayB : out STD_LOGIC;
           selectDisplayC : out STD_LOGIC;
           selectDisplayD : out STD_LOGIC);
end Segment_Driver;

architecture Behavioral of Segment_Driver is
    component Segment_Decoder
        Port ( Input : in STD_LOGIC_VECTOR (4 downto 0);
               segment_A : out STD_LOGIC;
               segment_B : out STD_LOGIC;
               segment_C : out STD_LOGIC;
               segment_D : out STD_LOGIC;
               segment_E : out STD_LOGIC;
               segment_F : out STD_LOGIC;
               segment_G : out STD_LOGIC);
    end component;

    component Clock_Divider
        Port ( clock : in STD_LOGIC;
               enable : in STD_LOGIC;
               reset : in STD_LOGIC;
               dataClock : out STD_LOGIC_VECTOR (15 downto 0));
    end component;

    Signal temporaryData : std_logic_vector (4 downto 0);
    Signal clockWord : std_logic_vector (15 downto 0);
    Signal slowClock : std_logic;
begin

    uut: Segment_Decoder PORT MAP(
        Input => temporaryData,
        segment_A => segA,
        segment_B => segB,
        segment_C => segC,
```

```
        segment_D => segD,
        segment_E => segE,
        segment_F => segF,
        segment_G => segG
    );

    uut1: Clock_Divider PORT MAP(
        clock => clock,
        enable => '1',
        reset => '0',
        dataClock => clockWord
    );

slowClock <= clockWord(15);

process(slowClock)
    variable displaySelection : std_logic_vector(1 downto 0);
    begin
    if slowClock'event and slowClock = '1' then
        case displaySelection is
            when "00" => temporaryData <= displayA;
                selectDisplayA <= '0';
                selectDisplayB <= '1';
                selectDisplayC <= '1';
                selectDisplayD <= '1';
                displaySelection := displaySelection + 1;
            when "01" => temporaryData <= displayB;
                selectDisplayA <= '1';
                selectDisplayB <= '0';
                selectDisplayC <= '1';
                selectDisplayD <= '1';
                displaySelection := displaySelection + 1;
            when "10" => temporaryData <= displayC;
                selectDisplayA <= '1';
                selectDisplayB <= '1';
                selectDisplayC <= '0';
                selectDisplayD <= '1';
                displaySelection := displaySelection + 1;
            when others => temporaryData <= displayD;
                selectDisplayA <= '1';
                selectDisplayB <= '1';
                selectDisplayC <= '1';
                selectDisplayD <= '0';
                displaySelection := displaySelection + 1;
        end case;
    end if;
end process;
end Behavioral;
```

Segment_Decoder.vhd

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

```
entity Segment_Decoder is
    Port ( Input : in std_logic_vector(4 downto 0);
           segment_A : out STD_LOGIC;
           segment_B : out STD_LOGIC;
           segment_C : out STD_LOGIC;
           segment_D : out STD_LOGIC;
           segment_E : out STD_LOGIC;
           segment_F : out STD_LOGIC;
           segment_G : out STD_LOGIC);
end Segment_Decoder;

architecture Behavioral of Segment_Decoder is
signal decodedData : std_logic_vector (6 downto 0);
begin

    with Input select decodedData <=
        "1111101" when "00000", -- a
        "0011111" when "00001", -- b
        "0001101" when "00010", -- c
        "0111101" when "00011", -- d
        "1001111" when "00100", -- e
        "1000111" when "00101", -- f
        "1111011" when "00110", -- g
        "0110000" when "00111", -- 1
        "1101101" when "01000", -- 2
        "1111001" when "01001", -- 3
        "0110011" when "01010", -- 4
        "1011011" when "01011", -- 5
        "1011111" when "01100", -- 6
        "1110000" when "01101", -- 7
        "1111111" when "01110", -- 8
        "0110111" when "01111",
        "0000000" when others;

    segment_A <= not decodedData(6);
    segment_B <= not decodedData(5);
    segment_C <= not decodedData(4);
    segment_D <= not decodedData(3);
    segment_E <= not decodedData(2);
    segment_F <= not decodedData(1);
    segment_G <= not decodedData(0);

end Behavioral;
```

Clock Divider.vhd

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

entity Clock_Divider is
    Port ( clock : in STD_LOGIC;
           enable : in STD_LOGIC;
```

```
           reset : in STD_LOGIC;
           dataClock : out STD_LOGIC_VECTOR (15 downto 0));
end Clock_Divider;

architecture Behavioral of Clock_Divider is

begin

    process(clock, reset, enable)
    variable counter : std_logic_vector (15 downto 0):=(others =>'0');
    begin
        if reset = '1' then
            counter := (others => '0');
        elsif enable = '1' and clock' event and clock = '1' then
            counter := counter + 1;
        end if;

    dataClock <= counter;

    end process;
end Behavioral;
```

VGA Controller.vhd

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

entity vga_controller is
    Port (  clk : in std_logic;
            notes: in std_logic_vector(11 downto 0);
            octaves : in std_logic_vector(2 downto 0);
            hsync : out std_logic;
            vsync : out std_logic;
            red : out std_logic_vector(3 downto 0);
            green : out std_logic_vector(3 downto 0);
            blue : out std_logic_vector(3 downto 0));
end vga_controller;

architecture Behavioral of vga_controller is
    component vga_synch is
        Port (clock : in std_logic;
              clear : in std_logic;
              hsync : out std_logic;
              vsync : out std_logic;
              vidon : out std_logic;
              hc :  out std_logic_vector(10 downto 0);
              vc :  out std_logic_vector(10 downto 0));
     end component;

     component vga_draw is
        Port (vidon : in std_logic;
              clk : in std_logic;
              notes : in std_logic_vector(11 downto 0);
              octaves : in std_logic_vector(2 downto 0);
```

```
            hc :  in std_logic_vector(10 downto 0);
            vc :  in std_logic_vector(10 downto 0);
            red :  out std_logic_vector(3 downto 0);
            blue :  out std_logic_vector(3 downto 0);
            green :  out std_logic_vector(3 downto 0));
    end component;

 signal vidon : std_logic;
 signal clr : std_logic;
 signal hc : std_logic_vector(10 downto 0);
 signal vc : std_logic_vector(10 downto 0);

begin

    uut1 : vga_synch Port Map(clk, clr, hsync, vsync, vidon, hc, vc);
    uut2 : vga_draw port map(vidon, clk, notes, octaves, hc, vc, red, green,
blue);


end Behavioral;
```

VGA_Sycnh.vhd

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


entity vga_synch is
  Port ( clock : in std_logic;
         clear : in std_logic;
         hsync : out std_logic;
         vsync : out std_logic;
         vidon : out std_logic;
         hc :  out std_logic_vector(10 downto 0);
         vc :  out std_logic_vector(10 downto 0));
end vga_synch;

architecture Behavioral of vga_synch is

constant hpixels : std_logic_vector (10 downto 0) := "11010011000"; --1688
constant vpixels : std_logic_vector (10 downto 0) := "10000101010"; --1066
constant hbp : std_logic_vector (10 downto 0) := "00101101000"; --360
constant vbp : std_logic_vector (10 downto 0) := "00000101001"; --41
constant hfp : std_logic_vector (10 downto 0) := "11001101000"; --1640
constant vfp : std_logic_vector (10 downto 0) := "10000101001"; --1065
signal hcs : std_logic_vector (10 downto 0);
signal vcs : std_logic_vector (10 downto 0);
signal vsenable : std_logic;

begin
process(clock, clear)
begin
    if clear = '1' then
```

```
        hcs <= "00000000000";
    elsif(clock'event and clock = '1') then
        if hcs = hpixels - 1 then
            hcs <= "00000000000";
            vsenable <= '1';
        else
            hcs <= hcs + 1;
            vsenable <= '0';
        end if;
     end if;
end process;

hsync <= '0' when hcs < 112 else '1';

process(clock, clear)
begin
    if clear = '1' then
        vcs <= "00000000000";
    elsif(clock'event and clock = '1' and vsenable = '1') then
        if vcs = vpixels - 1 then
            vcs <= "00000000000";
        else
            vcs <= vcs + 1;
        end if;
     end if;
end process;

vsync <= '0' when vcs < 3 else '1';

vidon <= '1' when (((hcs < hfp) and (hcs >= hbp)) and ((vcs < vfp) and (vcs >=
vbp))) else '0';

hc <= hcs;
vc <= vcs;

end Behavioral;
```

VGA_Draw.vhd

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

entity vga_draw is
  Port (vidon : in std_logic;
        clk : in std_logic;
        notes : in std_logic_vector(11 downto 0);
        octaves : in std_logic_vector(2 downto 0);
        hc :  in std_logic_vector(10 downto 0);
        vc :  in std_logic_vector(10 downto 0);
        red :  out std_logic_vector(3 downto 0);
        blue :  out std_logic_vector(3 downto 0);
        green :  out std_logic_vector(3 downto 0));
end vga_draw;
```

```
architecture Behavioral of vga_draw is
constant width: integer := 23;
constant height: integer := 256;
constant line_interval : integer := 19;
constant line_number : integer := 8;
constant line_thickness : integer := 1;
constant hbp : integer := 248;
constant hfp : integer := 48;
constant hsp : integer := 112;
constant vbp : integer := 38;
constant vfp : integer := 1;
constant vsp : integer := 3;

begin

process(vidon, hc, vc, notes, octaves)
begin
    red <= "0000";
    blue <= "0000";
    green <= "0000";

    if vidon = '1' then
        for i in 0 to 55 loop
            if (i mod 7 = 0 or i mod 7 = 1 or i mod 7 = 3 or i mod 7 = 4 or i
mod 7 = 5) and hc - hbp - hsp < (i + 1) * width + line_number and hc - hbp - hsp
> (i + 1) * width - line_number and (vc - vsp - vbp) < 512 and (vc - vsp - vbp)
> 256 then
                if ((notes((i - 7 * to_integer(unsigned(octaves))) * 2 + 1) =
'1' and i mod 7 < 2 ) or (notes((i - 7 * to_integer(unsigned(octaves))) * 2) =
'1' and i mod 7 > 2)) and  i / 7 = to_integer(unsigned(octaves)) then
                    red <= "0100";
                    blue <= "0000";
                    green <= "0100";
                else
                    red <= "0000";
                    blue <= "0000";
                    green <= "0000";
                end if;
            elsif (hc - hbp - hsp) < (i + 1) * width - line_thickness and (hc -
hbp - hsp) > i * width and (vc - vsp - vbp) < 768 and (vc - vsp - vbp)  > 256
then
                if ((notes((i - 7 * to_integer(unsigned(octaves))) * 2) = '1'
and i mod 7 < 3) or (notes((i - 7 * to_integer(unsigned(octaves))) * 2 - 1) =
'1' and i mod 7 > 2)) and i / 7 = to_integer(unsigned(octaves)) then
                    red <= "0100";
                    blue <= "0000";
                    green <= "0100";
                else
                    red <= "1111";
                    blue <= "1111";
                    green <= "1111";
                end if;
            end if;
        end loop;
```

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

Constraints.xdc

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

# Switches
set_property PACKAGE_PIN V17 [get_ports {Notes[11]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[11]}]
set_property PACKAGE_PIN V16 [get_ports {Notes[10]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[10]}]
set_property PACKAGE_PIN W16 [get_ports {Notes[9]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[9]}]
set_property PACKAGE_PIN W17 [get_ports {Notes[8]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[8]}]
set_property PACKAGE_PIN W15 [get_ports {Notes[7]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[7]}]
set_property PACKAGE_PIN V15 [get_ports {Notes[6]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[6]}]
set_property PACKAGE_PIN W14 [get_ports {Notes[5]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[5]}]
set_property PACKAGE_PIN W13 [get_ports {Notes[4]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[4]}]
set_property PACKAGE_PIN V2 [get_ports {Notes[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[3]}]
set_property PACKAGE_PIN T3 [get_ports {Notes[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[2]}]
set_property PACKAGE_PIN T2 [get_ports {Notes[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[1]}]
set_property PACKAGE_PIN R3 [get_ports {Notes[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Notes[0]}]
set_property PACKAGE_PIN W2 [get_ports {Octaves[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Octaves[0]}]
set_property PACKAGE_PIN U1 [get_ports {Octaves[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Octaves[1]}]
set_property PACKAGE_PIN T1 [get_ports {Octaves[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {Octaves[2]}]
#set_property PACKAGE_PIN R2 [get_ports {sw[15]}]
#     set_property IOSTANDARD LVCMOS33 [get_ports {sw[15]}]

#7 segment display
set_property PACKAGE_PIN W7 [get_ports {seven_segment_display[0]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {seven_segment_display[0]}]
set_property PACKAGE_PIN W6 [get_ports {seven_segment_display[1]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {seven_segment_display[1]}]
```

```
set_property PACKAGE_PIN U8 [get_ports {seven_segment_display[2]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {seven_segment_display[2]}]
set_property PACKAGE_PIN V8 [get_ports {seven_segment_display[3]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {seven_segment_display[3]}]
set_property PACKAGE_PIN U5 [get_ports {seven_segment_display[4]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {seven_segment_display[4]}]
set_property PACKAGE_PIN V5 [get_ports {seven_segment_display[5]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {seven_segment_display[5]}]
set_property PACKAGE_PIN U7 [get_ports {seven_segment_display[6]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {seven_segment_display[6]}]

#set_property PACKAGE_PIN V7 [get_ports dp]
#     set_property IOSTANDARD LVCMOS33 [get_ports dp]

set_property PACKAGE_PIN U2 [get_ports {seven_segment_display[7]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {seven_segment_display[7]}]
set_property PACKAGE_PIN U4 [get_ports {seven_segment_display[8]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {seven_segment_display[8]}]
set_property PACKAGE_PIN V4 [get_ports {seven_segment_display[9]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {seven_segment_display[9]}]
set_property PACKAGE_PIN W4 [get_ports {seven_segment_display[10]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {seven_segment_display[10]}]


###Buttons
set_property PACKAGE_PIN U18 [get_ports Modes[0]]
      set_property IOSTANDARD LVCMOS33 [get_ports Modes[0]]
set_property PACKAGE_PIN T18 [get_ports Modes[1]]
      set_property IOSTANDARD LVCMOS33 [get_ports Modes[1]]
set_property PACKAGE_PIN W19 [get_ports Modes[2]]
      set_property IOSTANDARD LVCMOS33 [get_ports Modes[2]]
set_property PACKAGE_PIN T17 [get_ports Modes[3]]
      set_property IOSTANDARD LVCMOS33 [get_ports Modes[3]]
set_property PACKAGE_PIN U17 [get_ports Modes[4]]
      set_property IOSTANDARD LVCMOS33 [get_ports Modes[4]]

##PWM Output
set_property PACKAGE_PIN N1 [get_ports {output}]
      set_property IOSTANDARD LVCMOS33 [get_ports {output}]

##VGA Connector
set_property PACKAGE_PIN G19 [get_ports {red[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {red[0]}]
set_property PACKAGE_PIN H19 [get_ports {red[1]}]
```

```
      set_property IOSTANDARD LVCMOS33 [get_ports {red[1]}]
set_property PACKAGE_PIN J19 [get_ports {red[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {red[2]}]
set_property PACKAGE_PIN N19 [get_ports {red[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {red[3]}]
set_property PACKAGE_PIN N18 [get_ports {blue[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {blue[0]}]
set_property PACKAGE_PIN L18 [get_ports {blue[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {blue[1]}]
set_property PACKAGE_PIN K18 [get_ports {blue[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {blue[2]}]
set_property PACKAGE_PIN J18 [get_ports {blue[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {blue[3]}]
set_property PACKAGE_PIN J17 [get_ports {green[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {green[0]}]
set_property PACKAGE_PIN H17 [get_ports {green[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {green[1]}]
set_property PACKAGE_PIN G17 [get_ports {green[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {green[2]}]
set_property PACKAGE_PIN D17 [get_ports {green[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {green[3]}]
set_property PACKAGE_PIN P19 [get_ports hsync]
      set_property IOSTANDARD LVCMOS33 [get_ports hsync]
set_property PACKAGE_PIN R19 [get_ports vsync]
      set_property IOSTANDARD LVCMOS33 [get_ports vsync]
```