Mustafa Bahaeddin Bozan

22102838

EEE-102-001

Term Project: Digital Piano

A) YouTube Link

https://youtu.be/F09-xOU-xzs?si=JVjc40oCd_S82UOP

B) Introduction

As a term project I chose to build a digital piano. The purpose of this project is to combine digital design and music. The square waves changing periodically from 0 to 1 with different frequencies will be used to create sound signals. For this project one needs to use BASYS3 FPGA board, an audio amplifier and a speaker.

C) Methodology

Before starting to code the Digital Piano, one needs to determine which octaves and piano keys will be imitated. Then, one needs to find their frequencies and their relationships between different octaves. I chose to use B, A, G, F, E, D and C notes and all octaves in this project. Also, one needs to determine some constants by dividing 10^8 to the frequency of the piano key that is imitated to create approximate frequencies of that note from the internal clock of the BASYS3 FPGA board since its internal clock has 100 MHz frequency. The result of my search for 0th octave can be seen from the Table 1.

Piano key	Frequency (Hz)	The constant
B	30.86771	3240441
A	27.5	3636364
G	24.49971	4081634
F	21.82676	4580852
E	20.60172	4854369
D	18.35405	5449591
C	16.35160	6116208

Table 1. Frequency of piano keys and their constants. (Wikipedia, Piano key frequencies)

Then, when I looked for the frequency of piano keys in higher octaves, I realized that there is a relationship between higher frequencies and the 0th octave. Simply, their frequencies are changing according to the powers of 2. Therefore, I decided to use a multiplexer to change increment of the clock dividers and use only the 0th octave's constants in order to simplify my design.

Now, with this information one can move on to start to code the Digital Piano. First of all, one needs to code the entity part. In this part, there should be a generic which contains the constants in the Table 1. In the port part, one needs to define clock, note inputs and octave input. Also, since the octave and the output note will be displayed through 7 segment display, one needs to define anode and segment outputs.

The next part is to code the architecture. In the architecture part, firstly one needs to create some signals, namely a counter for dividing the internal clock of BASYS3 for different notes, an increment integer for incrementing the counter for different note frequencies with different octaves, and a 16-bit vector for 7 segment display. For 7 segment display we need to understand the concept of persistence of vision as well.

The core of the project is coding a clock process for incrementing the counter for notes and making the output 1 and 0 according to used note’s frequency since we are using square waves. In this process, one also needs to increment the 16-bit vector for 7 segment display.

Then, the rest of the code mainly depends on the use of multiplexers. One needs to create multiplexers for choosing increment constant, displaying the octaves on the 7 segment display, displaying the notes on the 7 segment display and selecting the output note according to the given inputs.

Finally, to create persistence of vision, one needs to code another process for choosing anodes according to the 16-bit vector which is created and incremented, and for displaying the octave and note that is played on the selected anodes.

The coding part of the Digital Piano is completed with these steps. Now, one can move on to simulate the code by coding a test bench. Then, one can finalize the project by completing the synthesis implementations and generating the bitstream steps.

There is one important point about the project. The output signal for the notes will be very weak due to BASYS3. Therefore, one needs to use an audio amplifier to amplify this weak signals. Otherwise, the sounds of the square wave probably will not be heard.

D) Results

After completing the coding of the Digital Piano according to the methodology section, I have checked the elaborated design of the project and coded the test bench to see whether the project is working as expected from the simulation.

Since my personal computer is broken, I cannot add the simulation results. However, the results were true as expected. The elaborated design can be seen from the Figure 1.

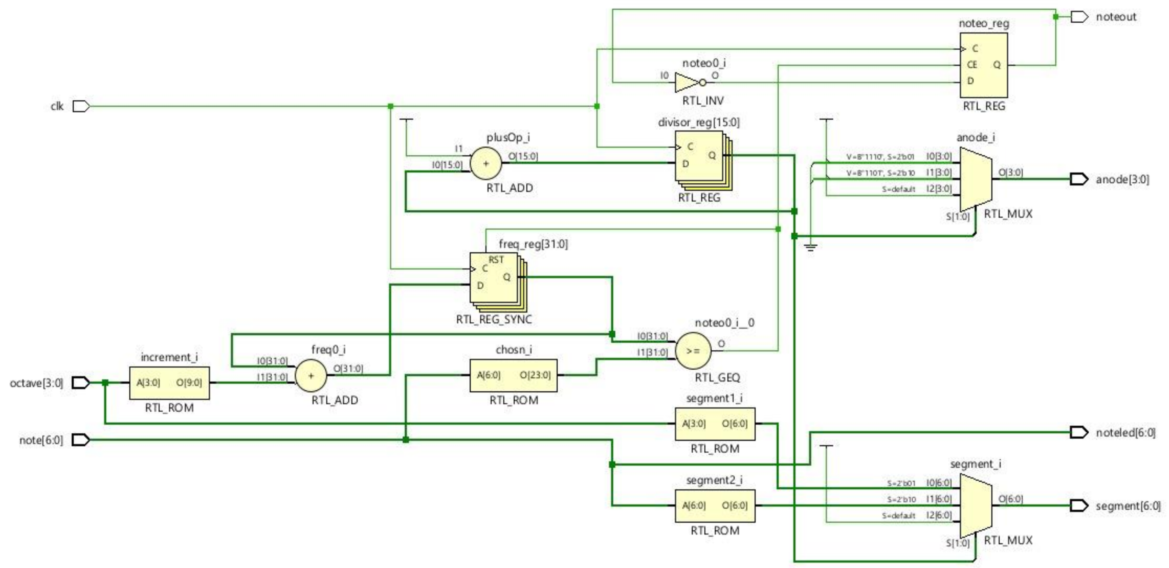

Figure 1. The elaborated design of the Digital Piano.

After completing the synthesis, implementation and generation of the bitstream parts, BASYS3 FPGA board was connected through auto connect and programmed to check the project and play some musics. To do this, I connected my output signal to the audio amplifier port of my TRC-11 which is my EEE-211 project and used a speaker to listen the notes. Some of the pictures taken from the video recorded while using the Digital Piano can be seen from the Figure 2-4.

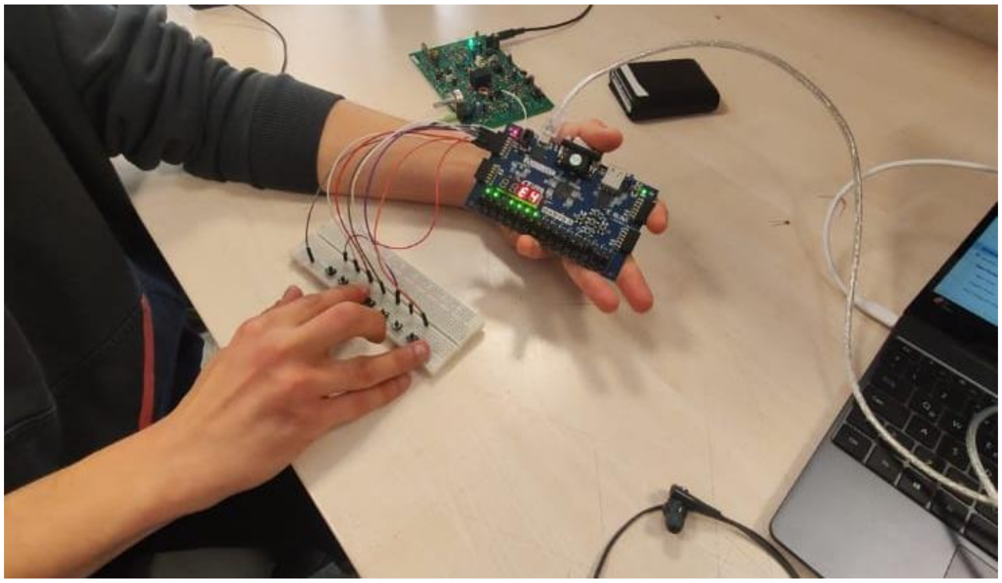

Figure 2. The octave is 4 and the played key is E.

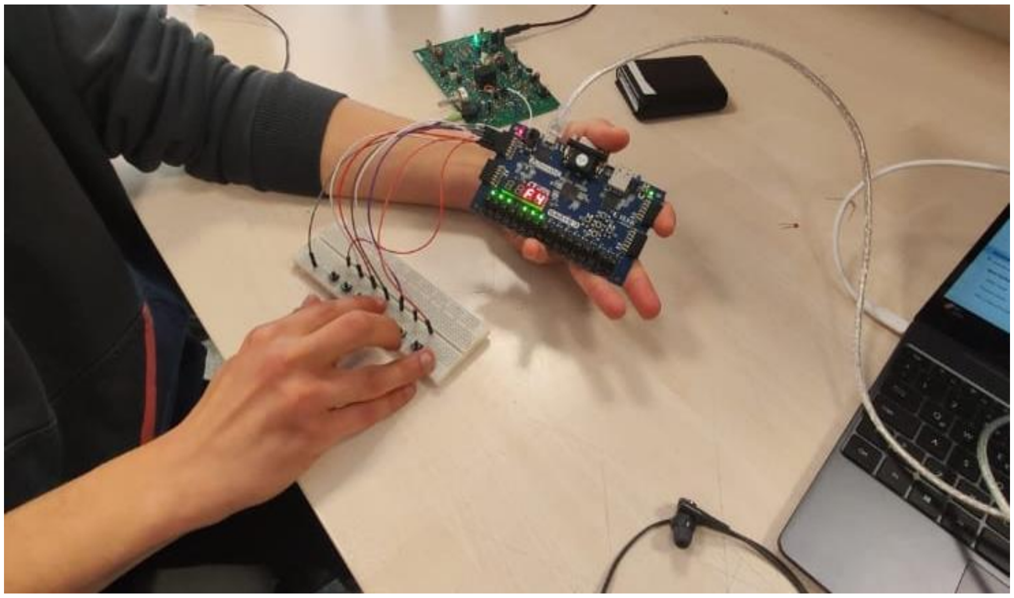

Figure 3. The octave is 4 and the played key is F.

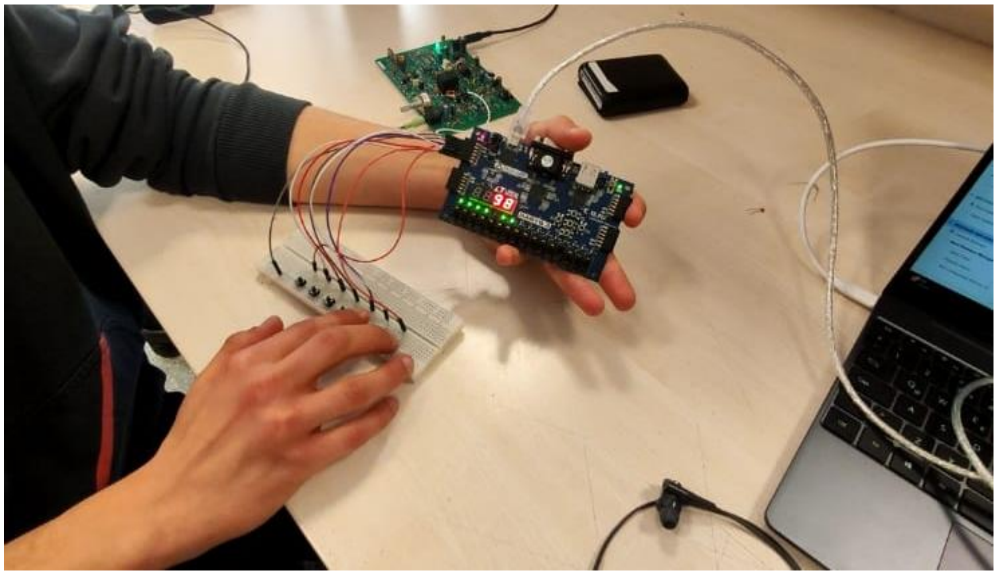

Figure 4. The octave is 8 and the played key is G.

E) Conclusion

For my term project I decided to build a Digital Piano in order to combine my digital design knowledge with my interest on music. I used most of the things I learnt during the lab sessions. Therefore, this project is very useful is terms of combining my understanding from the labs.

Besides, with this project, I learned how to use square waves to generate sounds. Also, some additional information is learnt such as use of generic in the entity part and use of integer signals.

All in all, this term project is very interesting and entertaining since I worked on one of my interests and combine most of my digital design knowledge. Thanks for everything.

F) References

Wikipedia contributors. (2021, December 26). *Piano key frequencies.* Wikipedia. https://en.wikipedia.org/wiki/Piano_key_frequencies

G) Appendix

Under this section, I am sharing the code of the Digital Piano project. However, since my personal computer is broken, I cannot add my test bench codes.

<u>Design Source</u>

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

entity digitalPiano is
   generic (b0 : integer := 3240441;
            a0 : integer := 3636364;
            g0 : integer := 4081634;
            f0 : integer := 4580852;
            e0 : integer := 4854369;
            d0 : integer := 5449591;
            c0 : integer := 6116208);

    Port ( clk : in STD_LOGIC;
           noteout : out STD_LOGIC;
           note : in std_logic_vector (6 downto 0);
           noteled : out std_logic_vector (6 downto 0);
           octave: in std_logic_vector (3 downto 0);
           anode: out std_logic_vector (3 downto 0);
           segment: out std_logic_vector (6 downto 0));
end digitalPiano;

architecture dp of digitalPiano is

signal freq: integer := 0;
signal noteo: std_logic := '0';
signal chosn: integer := 0;
signal increment: integer := 0;

signal segment1: std_logic_vector (6 downto 0);

signal segment2: std_logic_vector (6 downto 0);

signal divisor: std_logic_vector (15 downto 0) := "0000000000000000" ;


begin


noteled <= note;


with note select

chosn <= c0 when "0111111",

         d0 when "1011111",

         e0 when "1101111",

         f0 when "1110111",

         g0 when "1111011",

         a0 when "1111101",

         b0 when "1111110",

         0 when others;


process(clk)

begin

if rising_edge(clk) then

   divisor <= divisor + 1;

   if freq >= chosn then

      noteo <= not noteo;

      freq <= 0;

   else

      freq <= freq + increment;
```

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

with octave select
increment <= 1 when "0000",
             2 when "0001",
             4 when "0010",
             8 when "0011",
             16 when "0100",
             32 when "0101",
             64 when "0110",
             128 when "0111",
             256 when "1000",
             0 when others;

with octave select
segment1 <= "0000001" when "0000",
            "1001111" when "0001",
            "0010010" when "0010",
            "0000110" when "0011",
            "1001100" when "0100",
            "0100100" when "0101",
            "0100000" when "0110",
            "0001101" when "0111",
            "0000000" when "1000",
            "1111111" when others;
```

```
with note select
segment2 <= "0110001" when "0111111",
            "1000010" when "1011111",
            "0110000" when "1101111",
            "0111000" when "1110111",
            "0000100" when "1111011",
            "0001000" when "1111101",
            "1100000" when "1111110",
            "1111111" when others;

process(divisor (15 downto 14))
begin

case divisor (15 downto 14) is
when "01" => anode <= "1110";
when "10" => anode <= "1101";
when others => anode <= "1111";
end case;

case divisor (15 downto 14) is
when "01" => segment <= segment1;
when "10" => segment <= segment2;
when others => segment <= "1111111";
end case;

end process;

noteout <= noteo;
```

```
end dp;
```

Constrain

```
#clock
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]

##Pmod Header JA
##Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports {note[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {note[0]}]
##Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {note[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {note[1]}]
##Sch name = JA3
set_property PACKAGE_PIN J2 [get_ports {note[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {note[2]}]
##Sch name = JA4
set_property PACKAGE_PIN G2 [get_ports {note[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {note[3]}]
##Sch name = JA7
set_property PACKAGE_PIN H1 [get_ports {note[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {note[4]}]
##Sch name = JA8
set_property PACKAGE_PIN K2 [get_ports {note[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {note[5]}]
##Sch name = JA9
set_property PACKAGE_PIN H2 [get_ports {note[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {note[6]}]
##Sch name = JA10
set_property PACKAGE_PIN G3 [get_ports {noteout}]
set_property IOSTANDARD LVCMOS33 [get_ports {noteout}]

#leds
set_property PACKAGE_PIN V3 [get_ports {noteled[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {noteled[0]}]
set_property PACKAGE_PIN W3 [get_ports {noteled[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {noteled[1]}]
set_property PACKAGE_PIN U3 [get_ports {noteled[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {noteled[2]}]
set_property PACKAGE_PIN P3 [get_ports {noteled[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {noteled[3]}]
set_property PACKAGE_PIN N3 [get_ports {noteled[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {noteled[4]}]
set_property PACKAGE_PIN P1 [get_ports {noteled[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {noteled[5]}]
set_property PACKAGE_PIN L1 [get_ports {noteled[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {noteled[6]}]

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

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

# Switches
set_property PACKAGE_PIN V17 [get_ports {octave[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {octave[0]}]
set_property PACKAGE_PIN V16 [get_ports {octave[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {octave[1]}]
set_property PACKAGE_PIN W16 [get_ports {octave[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {octave[2]}]
set_property PACKAGE_PIN W17 [get_ports {octave[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {octave[3]}]
```