Berfin Lara Bilgen

22202945

EEE102-03

TERM PROJECT: MINI DIGITAL PIANO

YouTube video link: https://www.youtube.com/watch?v=VHD2iQPSBws&t=4s

A) OBJECTVE

Purpose of this project is to create a digital piano which can play all the notes on a regular piano by using BASYS3 and VHDL as our main language. I chose this project because it is the combination of the things we have learned on this course and my passion in music.

B) INTRODUCTION

This Project is called a "Mini Digital Piano" because it has 8 buttons, however it can still play all the notes on a regular piano. By changing the internal clock frequency of BASYS3 for all the notes on 0th octave, all notes on 0th octave can be heard. With the usage of switches on BASYS3 in order to change the octaves, and a "Flat(♭) Maker" button on the breadboard, all of the frequencies, therefore notes can be obtained from BASYS3. Additionally, TRC-11 has been used from EEE-211 course to amplify the sound which displayed from BASYS3.

C) METHODOLOGY

Output displayed from BASYS3 is a square wave with a specific period. This is because each note has its own unique frequency, therefore period. To hear the desired note from BASYS3, the displayed square wave must have the same period with that note. In order to achieve that, there are some procedures, which is the following:

First of all, the desired note has to be selected by pressing the corresponding button on breadboard.

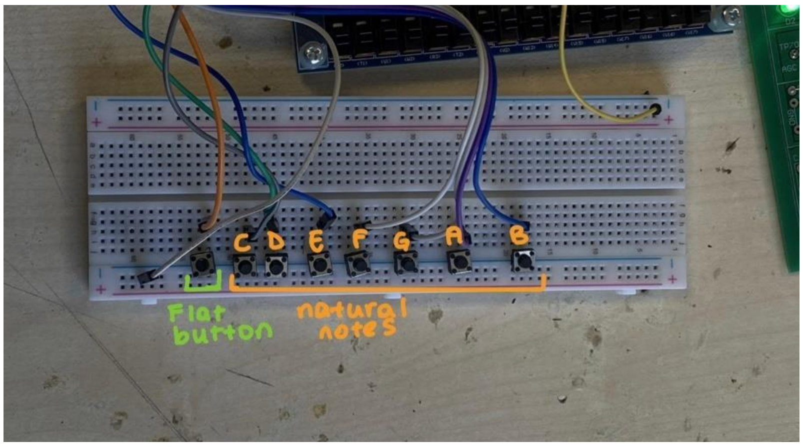


Figure 1.1 Buttons on breadboard, corresponds to keys on piano for one octav

Selected note has a corresponding constant as an input, which determined by the following formula:

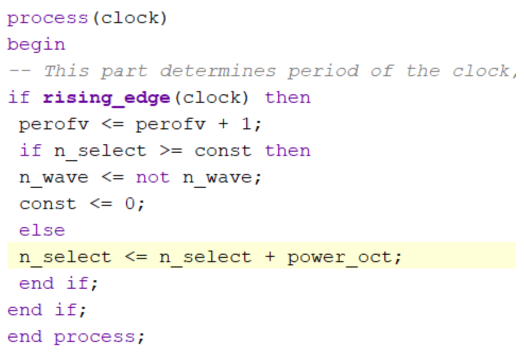
$$100MHz \times [period\ of\ chosen\ note\ on\ 0th\ octave]Hz^{-1} = constant$$

The selected octave has no effect on determining this constant, and reason of this will be explained later. For now, how to hear a note from 0th octave will be explained, assume selected octave is 0 for now.

Piano keys on 0th octave	Periods (Hz^{-1})	Corresponding constant
A	0.03636363636	3636364
A♭	0.03852593605	3852593
B	0.03239631317	3239631
B♭	0.0343226965	3432269
C	0.06115609482	6115609
D	0.05448388775	5448388
D♭	0.0577236894	5772369
E	0.0485396365	4853963
E♭	0.05142593842	5142594
F	0.04581532028	4581532
G	0.04081680967	4081681
G♭	0.04324389775	4324389

Table 1 Periods of piano keys on 0th octave and their corresponding constants [1]

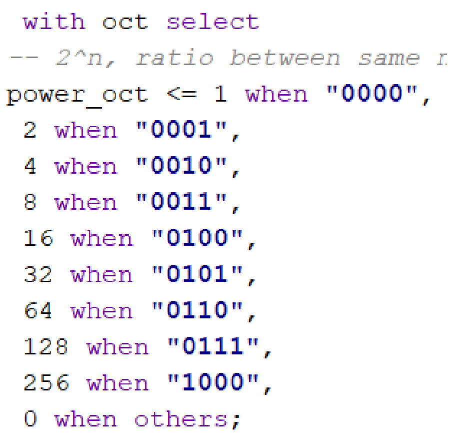
After the constant is determined, one should change the period of this square wave for the desired note.



```
process(clock)
begin
-- This part determines period of the clock,
if rising_edge(clock) then
 perofv <= perofv + 1;
 if n_select >= const then
 n_wave <= not n_wave;
 const <= 0;
 else
 n_select <= n_select + power_oct;
 end if;
end if;
end process;
```

Figure 1.2 Part of the code which creates a square wave with desired period

Note that **“n_wave”** is the generated square wave in each clock cycle which has 10ns period, and initially defined as 0. **“perofv”** is for 7-segment display, therefore we are ignoring it for this topic. **“const”** is a constant which is determined by selected note, calculated previously on *Table1*. **“n_select”** is a variable and it is 0 initially. **“power_oct”** is a variable which depending on the selected octave, assuming octave is 0 now, it is 2^0 (1).

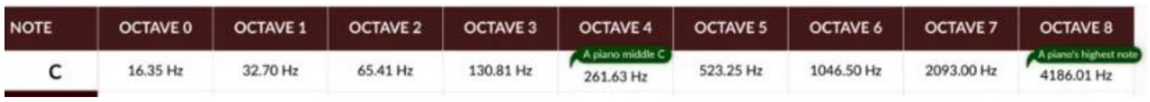


```
 with oct select
-- 2^n, ratio between same r
power_oct <= 1 when "0000",
 2 when "0001",
 4 when "0010",
 8 when "0011",
 16 when "0100",
 32 when "0101",
 64 when "0110",
 128 when "0111",
 256 when "1000",
 0 when others;
```

*Figure 1.3 How **"power_oct"** is depending on **"oct"**, where **"oct"** is the selected octave*

This part determines how octave changes, and it will be explained later. For now, assuming **"power_oct"** is 2^0 (1) will be adequate to acknowledge how the constant affects the period of square wave. Continuing from *Figure1.2*, assume the note is selected as C_0 (C on 0th octave) by pressing the button on breadboard After the button is pressed, the clock cycle (which has a period of 10ns) on *Figure 1.2* will start. Firstly, **"const"** (corresponding constant for note C_0) becomes 6115609. For the first 10ns, **"n_select"** is (initially defined as 0) not greater than or equal to **"const"**. Therefore, it passes on the second case where **"n_select"** is now equal to [**"n_select"** + **"power_oct"**] where **"power_oct"** accepted as 2^0 (1) for 0th octave. Accordingly, [**"n_select"** + 1] loop will continue until **"n_select"** is greater than or equal to **"const"** which is 6115609. Meanwhile, **"n_wave"** will keep outputting "0" from the first cycle to 6115608th cycle. Note that period of clock cycle is 10ns, hence "0" wave will display for 61156080ns. Eventually, **"n_select"** will become equal to **"const"** and **"n_wave"** will display "1" for 10ns. After that, **"const"** becomes 0 again (check *Figure1*.2) and cycle will complete. Finally, by observing the generated wave and the total time spent on finishing this wave, it can be observed that this generated wave has a period of 61156090ns ($61156090 \times 10^{-9} = 0.06115609$) which is the same with the period of C_0. This is how constants determine the period of the generated square wave.

Additionally, there is another constant which determines the period of the generated square wave, which is **"power_oct"**. Previously, it was accepted as 2^0 (1) for 0th octave. Also, it can observed from *Figure1.3* that there are other constants for other octaves. The reason behind that is because there is a correlation between the same note on different octaves. It can be seen from *Figure1.4* that, as octave increases by 1, frequency doubles therefore period halves.



NOTE	OCTAVE 0	OCTAVE 1	OCTAVE 2	OCTAVE 3	OCTAVE 4	OCTAVE 5	OCTAVE 6	OCTAVE 7	OCTAVE 8
C	16.35 Hz	32.70 Hz	65.41 Hz	130.81 Hz	A piano middle C 261.63 Hz	523.25 Hz	1046.50 Hz	2093.00 Hz	A piano's highest note 4186.01 Hz

Figure1.3 Frequency of note C on different octaves [2]

Assuming selected note is C_1 (period of C_1 is 0.03057805, half of C_0). The loop on *Figure1.2* started. Note that constant does not depend on the octave as previously mentioned, therefore **“const”** is 6115609. At first clock cycle, since **“n_select”** is (initially defined as 0) not greater than or equal to **“const”**, therefore cycle passed to the second case where **“n_select”** is now equal to [**“n_select”** + **“power_oct”**]. However, this time **“power_oct”** is 2^1 (2) from *Figure1.3*. Note that **“n_select”** is initially 0. Adding 2 to **“n_select”** until it is greater than or equal to the **“const”**, 6115609. For 3057804 clock cycles of 10ns, **“n_select” < “const”** therefore **“n_wave”** outputs “0” for 30578040ns. Finally, at 3057805th cycle **“n_select”> “const”** hence, **“n_wave”** outputs “1” for 10ns.

It took 30578050ns to create the output wave, which is same with the period of C_1. ($30578050 \times 10^{-9} = 0.03057805\ Hz^{-1}$). Therefore, the wave corresponding to C_1 has displayed from BASYS3.

Correlation between period of a note on octave number (n) and octave 0th of a note can be formulaized as the following:

$$Period_{note_0} \times 2^{-n} = Period_{note_n}$$

Period of a note on 0th octave is 2^n times greater than octave n^{th}. This explains why **“power_oct”** is 2^n when selected octave is n on *Figure1.3*.

With the system above, a square wave which has the same period of the selected note, can be generated and displayed from BASYS3. By creating an output port **“n_out”** which equals to **“n_wave”**, and connecting this port to amplifier section on TRC-11, displayed sound can be amplified, which provides higher audio.

3 Anodes of 7-Segment Display, have been used to display the selected octave, note. 3 anodes have used because some notes are flat(♭) notes and these notes should be indicated a flat(♭) symbol after them such as when A indicates note A, A♭ indicates A flat. If selected note is a flat(♭) note, 3rd anode displays a “b” letter, if not it is inactivated.

From *Figure1.1*, it can be seen that there are total of 8 buttons, it has arranged as a 8-bit input vector, and it determines the **“note”**. MSB of that vector determines whether this note is flat or not and rest of the bits determines type of that note (A,B,C…). Therefore 3rd anode activates, when MSB is 0 (active low). 1st anode displays octaves, which controlled by switches on BASYS3. Finally, 2nd anode displays type of that notes, which depends on last 7 bits of the input **“note”** vector.

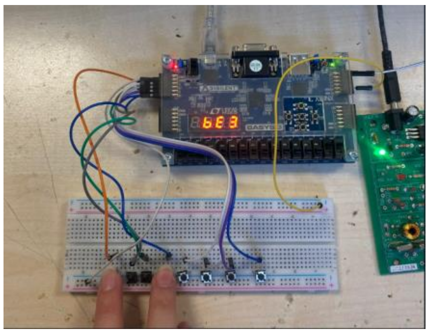


Figure1.4 MSB is “0”, anode 3 is activated and displays “b”, indicating that note is a flat note

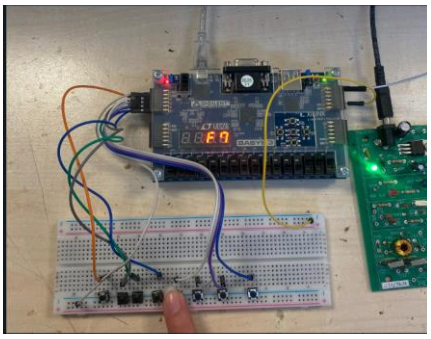
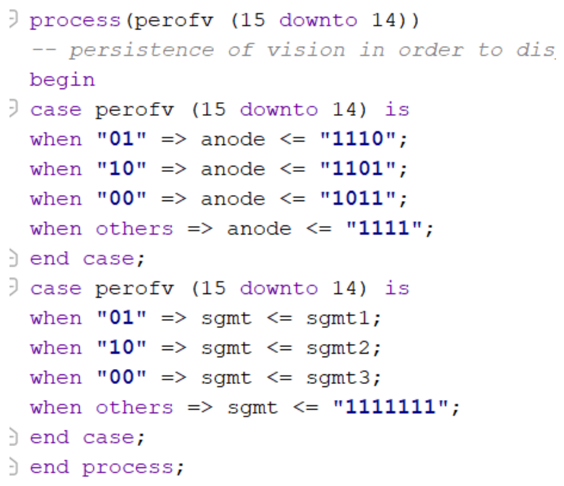


Figure1.5 MSB is “1”, anode 3 is deactivated indicating that note is naturel note.

Normally, 3 Anodes of the 7-Segment Display cannot be displayed at the same time. However, from the previous labs, it has been learned that use of persistence of vision can provide this. Anodes turning on and off so quickly, so fast that human eye cannot catch will create a sense that they are displaying at the same time. Going back on *Figure1.2*, there exists a 16-bit vector **"perofv"** which increases each clock cycle (10ns). Most significant two bits of this vector is used to activate the anodes separately.



```
process(perofv (15 downto 14))
-- persistence of vision in order to dis
begin
case perofv (15 downto 14) is
when "01" => anode <= "1110";
when "10" => anode <= "1101";
when "00" => anode <= "1011";
when others => anode <= "1111";
end case;
case perofv (15 downto 14) is
when "01" => sgmt <= sgmt1;
when "10" => sgmt <= sgmt2;
when "00" => sgmt <= sgmt3;
when others => sgmt <= "1111111";
end case;
end process;
```

Figure1.6   Persistence of Vision has been created to activate 3 anodes at different times.

The vector **"perofv"** changes so quickly, causing anodes to get activated and deactivated also so fast that human eye cannot catch therefore sees anodes as they are displaying at the same time.

D) RESULTS

After code is done, a mini digital piano has obtained which can play each note on a regular piano and displays currently playing note.

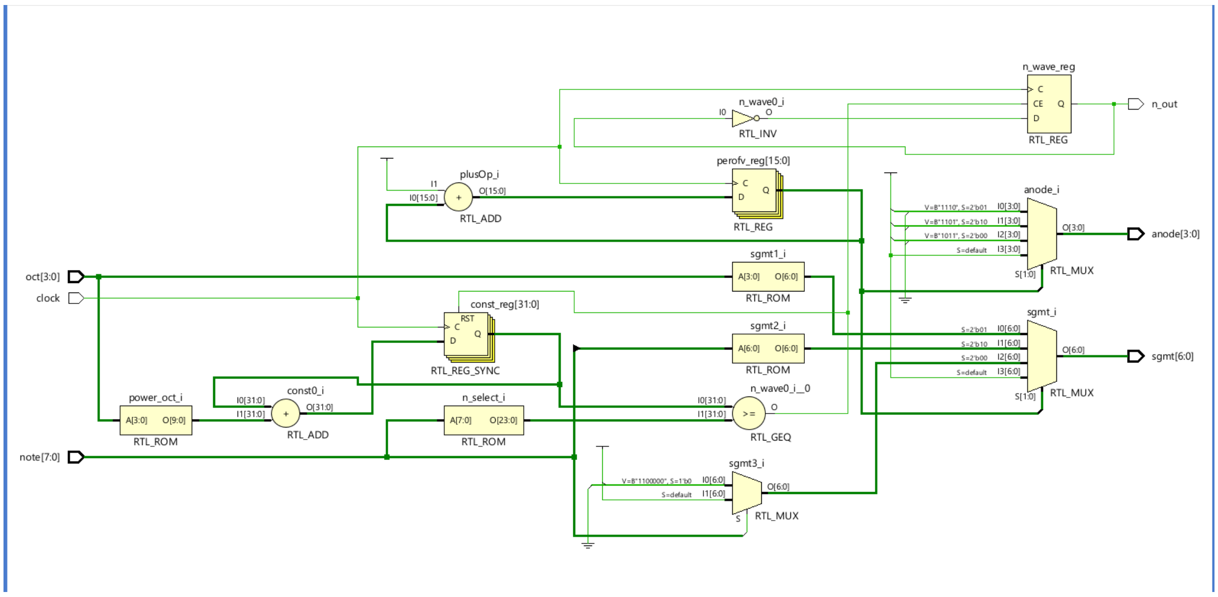
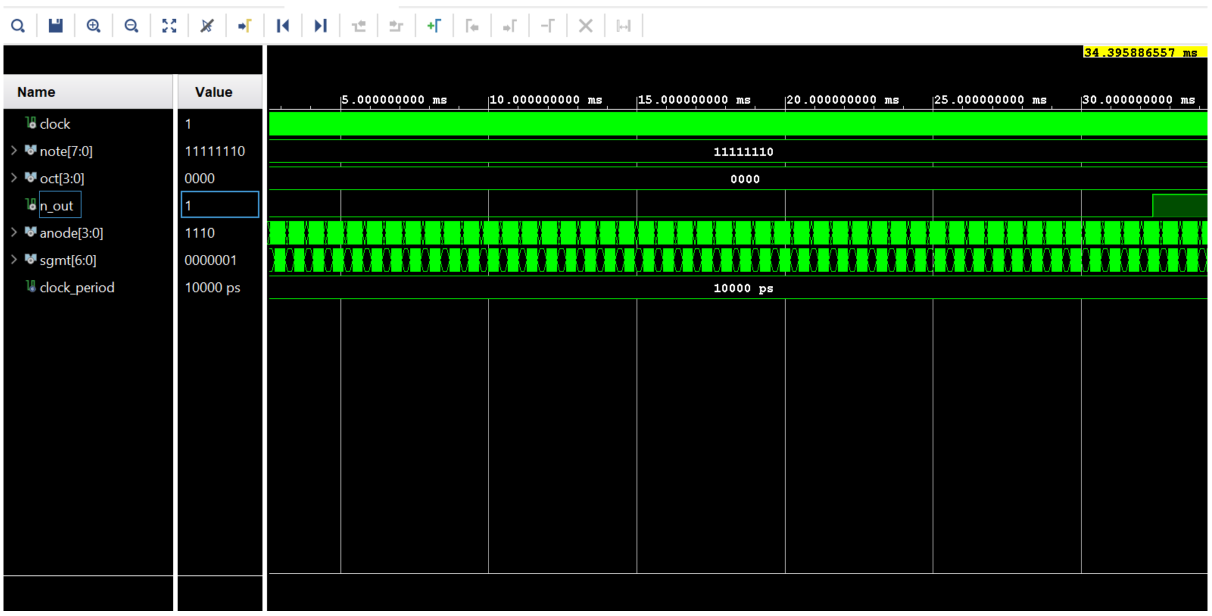


Figure2.1 *Schematic of the final code*

Since the period of each generated wave for specific note is different and generated period is depending on manually selected input, creating a testbench for each note separately seemed trivial. Also, even a small delay on testbench can be a mislead, since the wrong period on testbench may indicate wrong note, consequently indicating a mistake on the code. Therefore, instead of writing the testbench, trying the code in real life seemed more time consuming and has less error risk. Despite these risks, I still wrote a testbench for B0, simulation is the following:



Name	Value
clock	1
note[7:0]	11111110
oct[3:0]	0000
n_out	1
anode[3:0]	1110
sgmt[6:0]	0000001
clock_period	10000 ps

Figure 2.2 *Simulation of the code for the note B0*

As mentioned, simulation has delays which contradict the real output of the code. Normally 10ns delay on period may be trivial. However in this project, the period determines the sound generated, therefore testbenches are not accurate for this project.

Some results of actual mini digital piano are the following:

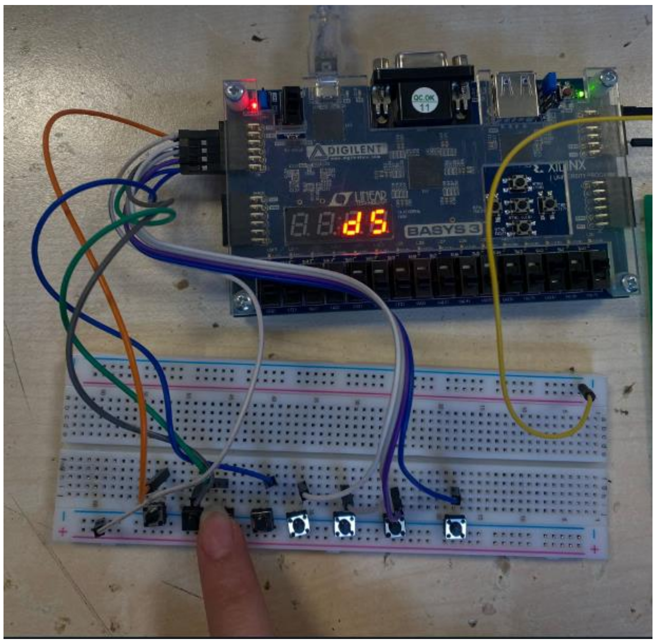


Figure 2.3 *Note D on 5th octave*

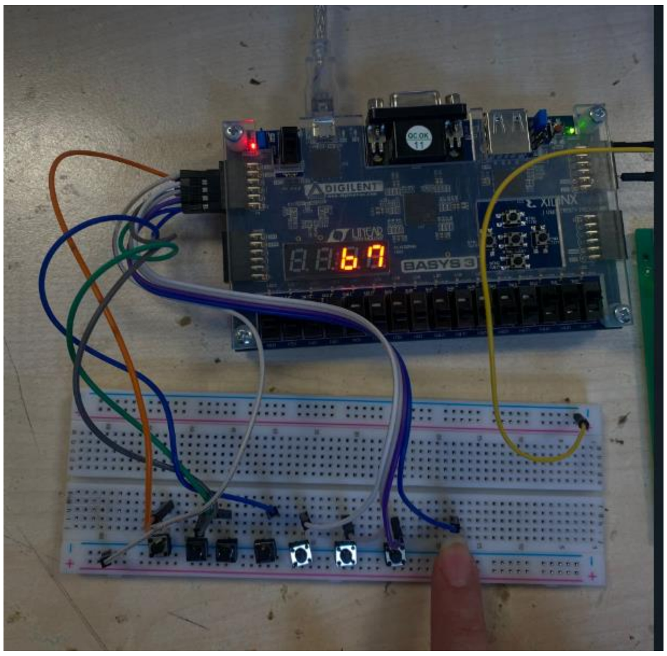


Figure2.4 *Note B on 7th octave*

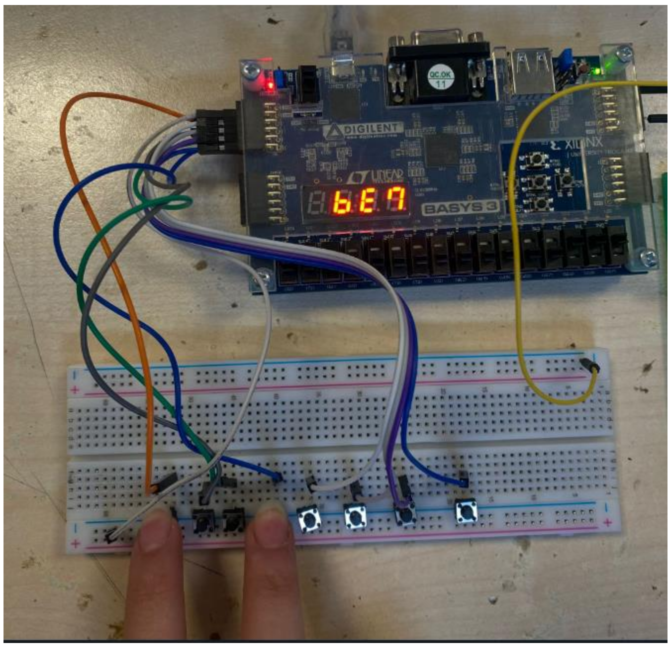


Figure2.5 *Note E*♭ (E flat) on 7th octave

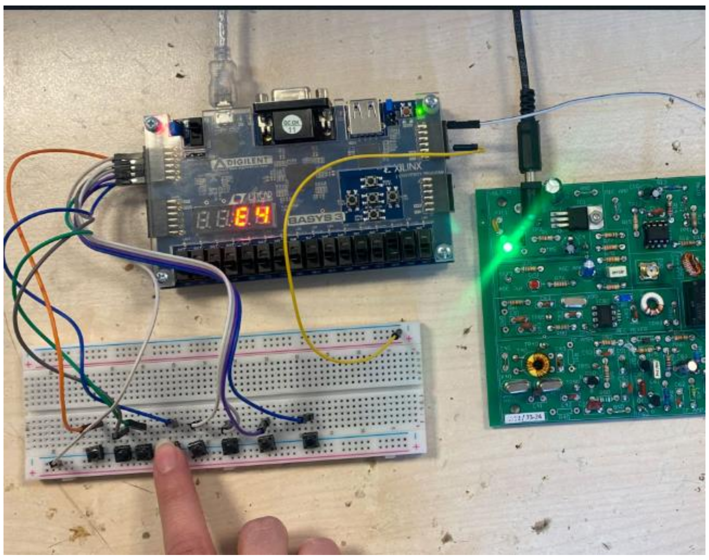


Figure2.6 *Note E on 4th octave*

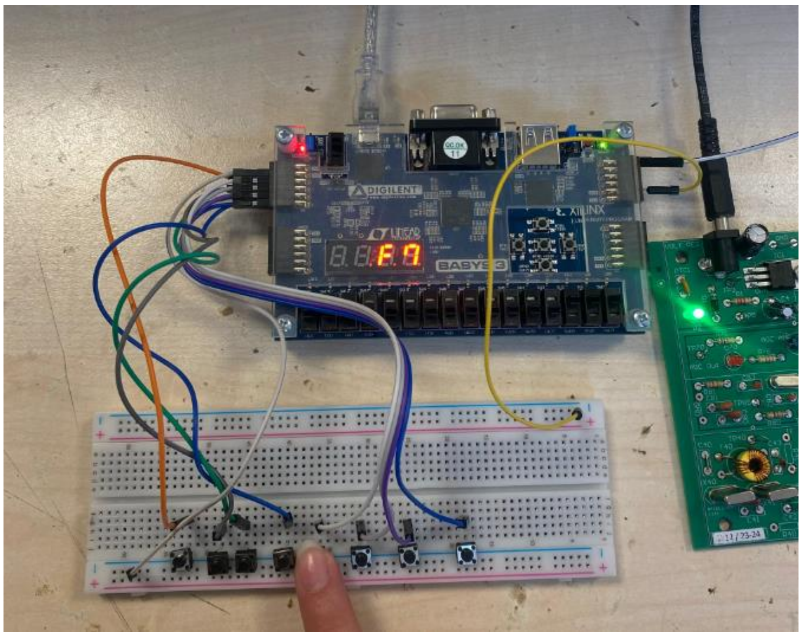


Figure2.7 *Note F on 7th octave*

E) CONCLUSION

In conclusion, this project aimed to combine my digital design knowledge and my music knowledge. Things that we have learned on the labs have created great amount of the project. Using 7-Segment Display, persistence of vision and changing clock frequency of BASYS3 are some examples of the things that I have learned on this course and used on my project. I also learned how to generate square waves with BASYS3.

Besides digital design, I also had a chance to use the TRC-11 on this project, therefore I experienced how to use amplifier of the TRC-11.

Overall, this project was an interesting experience for me since I created something by using combination of my knowledge on different areas.

F) REFERENCES

[1] https://en.wikipedia.org/wiki/Piano_key_frequencies
[2] https://mixbutton.com/mixing-articles/music-note-to-frequency-chart/

G) APPENDIX

main.vhd:

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



entity bethoweenDigital is
 generic (
 -- Note's period at th octave*internal frequency of BASYS3
 a0 : integer := 3636364;
 Ba0: integer := 3852593; -- Ba0 means A0 Flat
 b0 : integer := 3240441;
 Bb0: integer:= 3432269;
 c0 : integer := 6116208;
 d0 : integer := 5449591;
 Bd0: integer := 5770640;
 e0 : integer := 4854369;
 Be0: integer := 5142594;
 f0 : integer := 4580852;
 g0 : integer := 4081634;
 Bg0: integer := 432439);

 Port ( clock : in STD_LOGIC;
 n_out : out STD_LOGIC;
 note : in std_logic_vector (7 downto 0);
 oct: in std_logic_vector (3 downto 0);
 anode: out std_logic_vector (3 downto 0);
 sgmt: out std_logic_vector (6 downto 0));
```

```
end bethoweenDigital;

architecture dp of bethoweenDigital is
-- For 7 Segment Display
signal perofv: std_logic_vector (15 downto 0) := "0000000000000000" ;
signal sgmt1: std_logic_vector (6 downto 0);
signal sgmt2: std_logic_vector (6 downto 0);
signal sgmt3: std_logic_vector (6 downto 0);
-- For output frequency
signal const: integer := 0;
signal n_wave: std_logic := '0';
signal n_select: integer := 0;
signal power_oct: integer := 0;

begin



with note select
n_select <= c0 when "10111111",
 d0 when "11011111",
 e0 when "11101111",
 f0 when "11110111",
 g0 when "11111011",
 a0 when "11111101",
 b0 when "11111110",
 Bb0 when "01111110",
 Ba0 when "01111101",
 Bd0 when "01011111",
 Be0 when "01101111",
 Bg0 when "01111011",
 0 when others;

 with oct select
-- 2^n, ratio between same notes different octaves, determines icreasing ratio of frequency,
therefore period.
power_oct <= 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;

process(clock)
begin
-- This part determines period of the clock, therefore determines heard frequency which
corresponds desired note. (pvm)
if rising_edge(clock) then
 perofv <= perofv + 1;
 if const >= n_select then
 n_wave <= not n_wave;
 const <= 0;
 else
 const <= const + power_oct;
 end if;
end if;
end process;

with oct select
```

```
sgmt1 <= "0000001" when "0000", --0th octave ,segment displays 0
 "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(6 downto 0) select
sgmt2 <= "0110001" when "0111111" ,
 "1000010" when "1011111",
 "0110000" when "1101111",
 "0111000" when "1110111",
 "0000100" when "1111011",
 "0001000" when "1111101",
 "1100000" when "1111110",
 "1111111" when others;

with note(7 downto 7)select
sgmt3 <= "1100000" when "0",
 "1111111" when others;


process(perofv (15 downto 14))
-- persistence of vision in order to display several segments at the same time(!)
begin
case perofv (15 downto 14) is
when "01" => anode <= "1110";
when "10" => anode <= "1101";
when "00" => anode <= "1011";
when others => anode <= "1111";
end case;
case perofv (15 downto 14) is
when "01" => sgmt <= sgmt1;
when "10" => sgmt <= sgmt2;
when "00" => sgmt <= sgmt3;
when others => sgmt <= "1111111";
end case;
end process;
n_out <= n_wave;
end dp;
```

xdc.xdc :

```
#clock
set_property PACKAGE_PIN W5 [get_ports clock]
set_property IOSTANDARD LVCMOS33 [get_ports clock]
create_clock -add -name sys_clock_pin -period 10.00 -waveform {0 5} [get_ports clock]
##Pmod Header JA
##Sch name = JA1
## MSB YI 1 E MI 7 YE MI TAKICAM???
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 {note[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {note[7]}]

##Sch name = JB1
 #For Flat
set_property PACKAGE_PIN A14 [get_ports {n_out}]
set_property IOSTANDARD LVCMOS33 [get_ports {n_out}]

#7 segment display
set_property PACKAGE_PIN W7 [get_ports {sgmt[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sgmt[6]}]
set_property PACKAGE_PIN W6 [get_ports {sgmt[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sgmt[5]}]
set_property PACKAGE_PIN U8 [get_ports {sgmt[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sgmt[4]}]
set_property PACKAGE_PIN V8 [get_ports {sgmt[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sgmt[3]}]
set_property PACKAGE_PIN U5 [get_ports {sgmt[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sgmt[2]}]
set_property PACKAGE_PIN V5 [get_ports {sgmt[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sgmt[1]}]
set_property PACKAGE_PIN U7 [get_ports {sgmt[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sgmt[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 {oct[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {oct[0]}]
set_property PACKAGE_PIN V16 [get_ports {oct[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {oct[1]}]
set_property PACKAGE_PIN W16 [get_ports {oct[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {oct[2]}]
set_property PACKAGE_PIN W17 [get_ports {oct[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {oct[3]}]
```

tb.vhd :

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

entity tb_bethoweenDigital is
end tb_bethoweenDigital;

architecture sim of tb_bethoweenDigital is
   component bethoweenDigital
```

```
        port (
            clock : in std_logic;
            n_out : out std_logic;
            note : in std_logic_vector(7 downto 0);
            oct : in std_logic_vector(3 downto 0);
            anode : out std_logic_vector(3 downto 0);
            sgmt : out std_logic_vector(6 downto 0)
        );
    end component;

    signal clock : std_logic := '0';
    signal note : std_logic_vector(7 downto 0) := (others => '0');
    signal oct : std_logic_vector(3 downto 0) := (others => '0');

    signal n_out : std_logic;
    signal anode : std_logic_vector(3 downto 0);
    signal sgmt : std_logic_vector(6 downto 0);

    constant clock_period : time := 10 ns;

begin
    comp: bethoweenDigital port map (
        clock => clock,
        n_out => n_out,
        note => note,
        oct => oct,
        anode => anode,
        sgmt => sgmt
    );

    clock_process : process
    begin
        clock <= '0';
        wait for clock_period/2;
        clock <= '1';
        wait for clock_period/2;
    end process;

    tb: process
    begin -- we will simulate when we press B0 note on octave 0
        note <= "11111110"; -- B0 note binary code from main code
        oct <= "0000"; -- 0th octave
        wait for 100 ns;

        wait;
    end process;

end sim;
```