KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY
CAMPUS GROEP T LEUVEN

THE PIANO MASTER

Final Report for EE4: LED game

Team A-01
Doris Helina Fjuk
Dorien Jochmans
Kobe Vanlaere
Shumeng Wang

17 May 2019

TABLE OF CONTENTS

1. Introduction

The purpose of Engineering Experience 4 was to make an LED game based on electronic design. Our team decided to create a game called “The Piano Master”, which is intended to improve the skills of piano players. Starting to play the piano might be difficult, there are many

things that one needs to learn and comprehend. For instance, it is necessary to learn how the piano works and how the musical notes need to be read. Also, it is needed to have a piano to practice. All this combined ensures that many people drop out even before they have started. This is why we made a game to learn the piano. The game is analogous to the online learning tool for piano players where the tiles are falling down the screen and when it reaches the bottom, the user needs to play that note. Our game makes this virtual game to become live. Another difference between our game and the online version of it is that the user does not need an extra piano to practice and the instrument gives feedback by showing the score of how correctly the user is playing. What is more, the user can choose a level which is indicated by LEDs.

This report is divided into two parts - engineering part and enterprising part. In the engineering part, technical aspects behind the game are described such as the schematics and the code. In the enterprising part, the target group and the consequences of the product are discussed.

2. Engineering Part

2.1 Description

The product is divided into the following parts. The first part is a control system that keeps track of the inputs and drives the outputs. The second part is a sound system which makes the sound by itself and drives the loudspeaker. When the key is pressed the switch will make the sound signal go to the loudspeaker. Finally, the third part is a LED system. This is a matrix of LEDs that are controlled in rows and columns.

2.1.1 Control system

The control system is based on the microprocessor PIC18F25K50. Since there are insufficient pins on the microprocessor, a multiplexer, demultiplexer and IO expander became necessary to drive all the outputs and read all the inputs individually.

The demultiplexer was used for the outputs of the microprocessor and it transfers a series input signal to parallel output signal. In our project, it is applied in the LEDs' matrix. However, with the demultiplexer, the number of pins needed was still too large. Therefore, eight pins of demultiplexer were connected to the serial IO expander.

The multiplexer expands the inputs of the microprocessor by transferring eight bits parallel input signals to one series input signal. In this project, it transfers the signals from the keys to the microprocessor.

2.1.2 Sound system

The signals for the notes are made by eight triangle wave self-generating circuits that produce eigth different frequencies which correspond to one musical octave. Signals generated by the oscillation circuits are added by an adding circuit made by resistors and an operational amplifier. The switching of each signal is controlled by NPN transistors.

When a single key is pressed, the base of the transistor connects to VCC, forward operation mode, and makes the signal pass through the transistor to the adding part. When multiple keys are pressed, multiple signals are added by the adder.
Finally, the signal from the output of adder will be connected to a coupling capacitor, which is intended to filter out the DC component. The final signal will be played by the loudspeaker.

2.1.3 LED system

The LED system is composed of 6*8 matrix of LEDs, controlled by transistors for each row and each column. The output of the microprocessor is connected to the base resistors and determine if the row/column is on or not. This method allow us to have a lot of LED's that we can address with a limited output pins. Only if row and column are selected the LED will be on. This technique is also used in memories from all different kind of computers.

2.1.4 Interaction between sound system and LED system

In the idle state, the ground connects to the input of the microprocessor and also connecting a resistor in series. When the keys are pressed, the VCC supply, instead of the ground, connects to the input of the microprocessor.

This gives a high voltage to the microprocessor when the keys are played. When the microprocessor gets the binary signals, they will determine the state of the LED system.

2.2 Schematics

2.2.1 Sound system

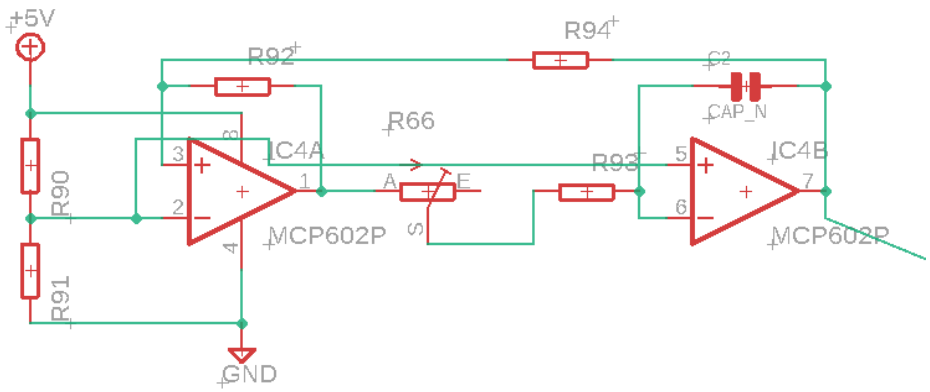


Figure 1. Triangle wave generating circuit.

Triangle wave generating circuit (Fig. 1) consist of a comparator and an integrator. The output of the comparator is a square wave like can be seen in Figure 2. The only difference is that we used a reference voltage of 2.5 volts instead of zero volt in the figure below. Hence, if we start with a high output of the comparator, there will be a current flowing toward the capacitor and the input of the second operational amplifier, thus it results in a negative ramp at the output. If the output crosses the reference voltage, the comparator will toggle and the output will go from high to low.

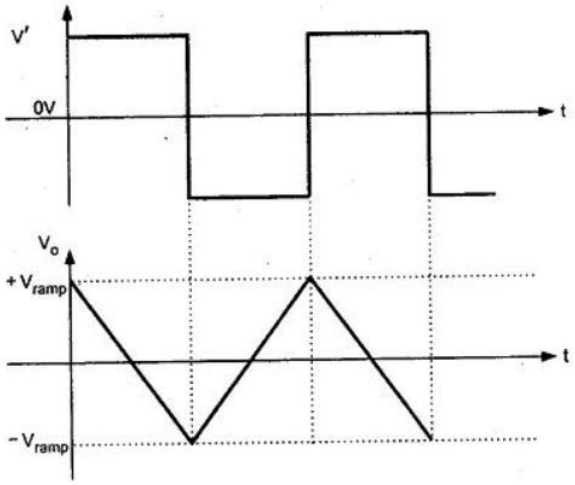


Figure 2. Outputs of the comparator and the integrator (without reference voltage).

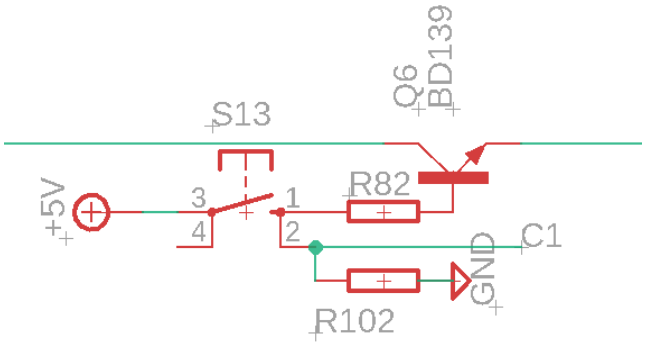


Figure 3. The piano keys.

The keys of the piano are essentially a switching circuit (Fig. 3). In addition to the switch that connects the signal from the sound to the adding circuit when the key is pressed, it is also necessary to let the microprocessor know that the key is pressed adding more complexity to the circuit. When the switch is closed, the output (C1, Figure 3) is connected to the ground, hence the microprocessor gets a logic of '0'. Also, the base of the transistor is connected to the ground ensuring that the sound signal will not pass the transistor. When the switch is closed, the input of the microprocessor is connected to VCC so it gets a logic of '1' and the base of the transistor will be connected to the VCC and the signal will pass the transistor.

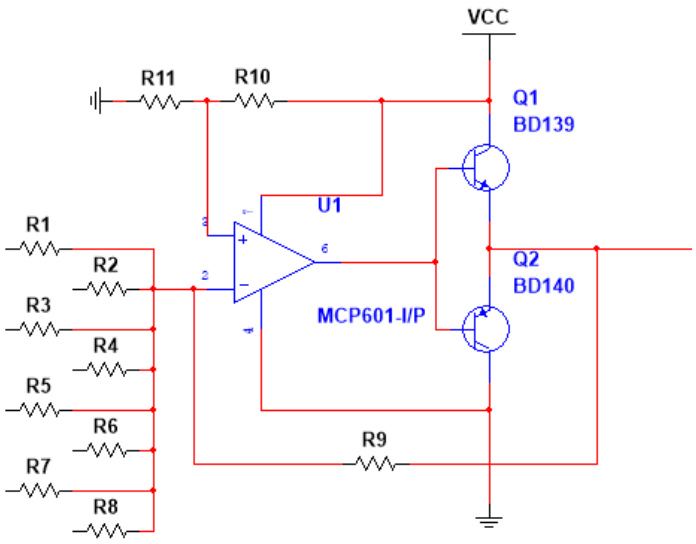


Figure 4. Adder and power amplifier.

Figure 4 depicts the adding circuit and a power amplifier circuit. The resistors from R1 to R9 and operational amplifier form a basic adding circuit. Since the signal generated from the oscillating circuit has an offset of 2.5V, it also needs a virtual ground of 2.5V, made by R10 and R11.

However, the operational amplifier is not able to deliver sufficient current to the loudspeaker. For that reason, a power amplifier was used. The base and emitter of transistor BD139 and BD140 are connected, the collector of BD139 is connected to VCC and the collector of BD140 is connected to the ground. The base of both transistors is the input signal, and the emitter of both transistors is the output of this power amplifier. In this situation, the NPN transistor BD139 conducts when the signal is higher than 2.5V, and PNP transistor BD140 conducts when the signal is lower than 2.5V. Thus, the signal always conducts

.

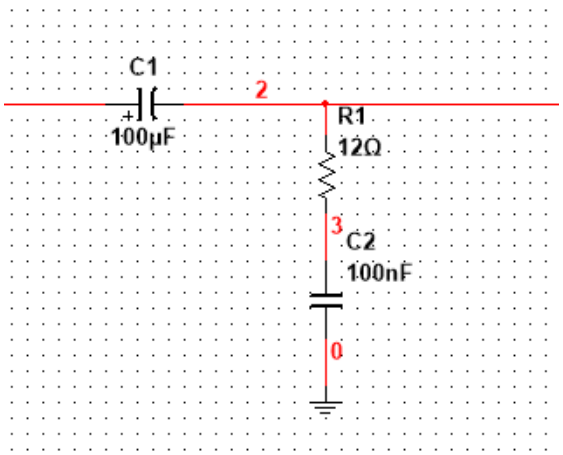


Figure 5. Coupling capacitor.

The circuit in Figure 5 connects the sound system to the loudspeaker. The combination of C1 and R1 is for the stability of the loudspeaker. C1 is the coupling capacitor, this is used for connecting, coupling the circuit together and filter out the DC component. The sound system gives an AC signal that the loudspeaker needs to play the sound but it can also contain a DC component that the loudspeaker does not want. This DC component is filtered out by the coupling capacitor (C1).

2.2.2 LED system

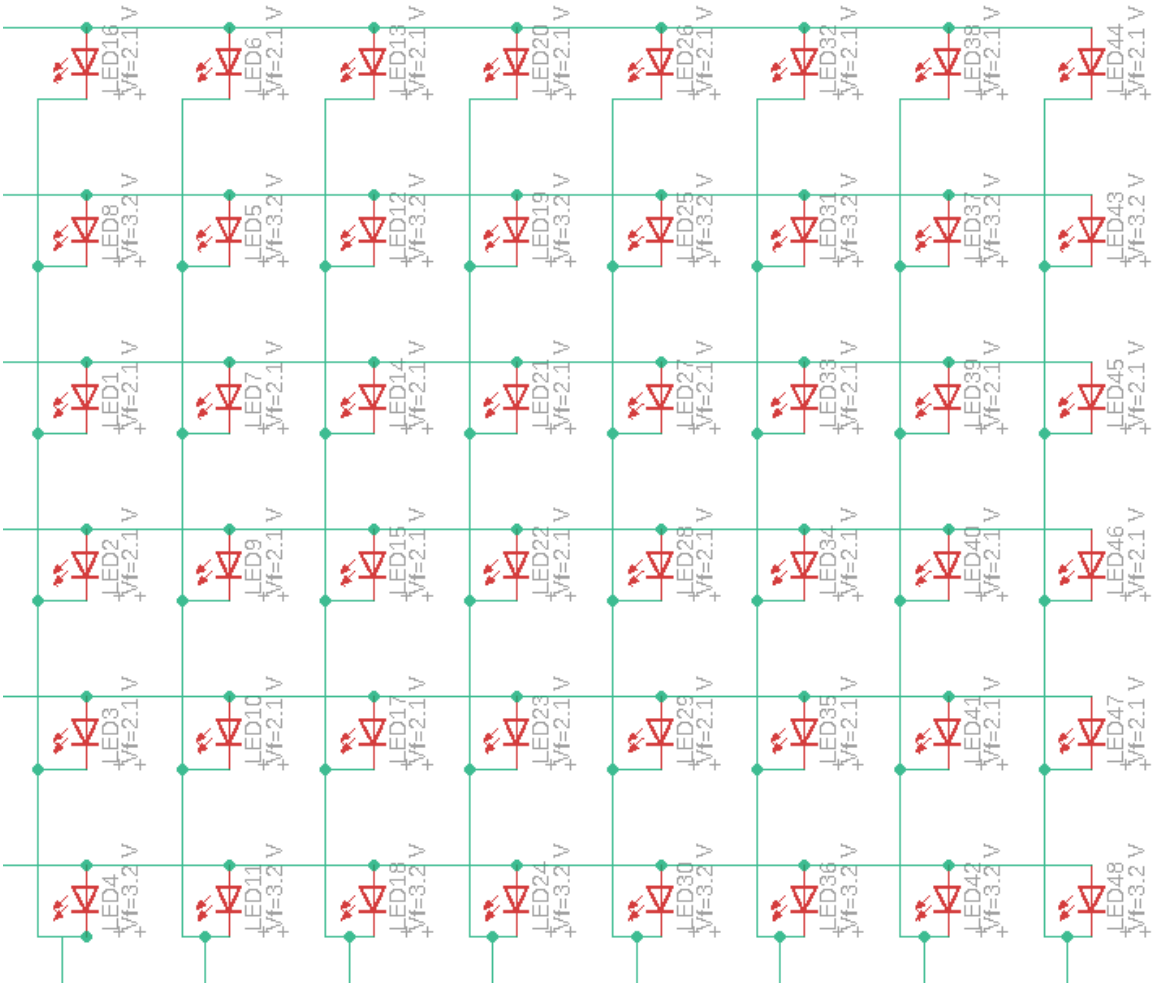


Figure 6. LEDs matrix.

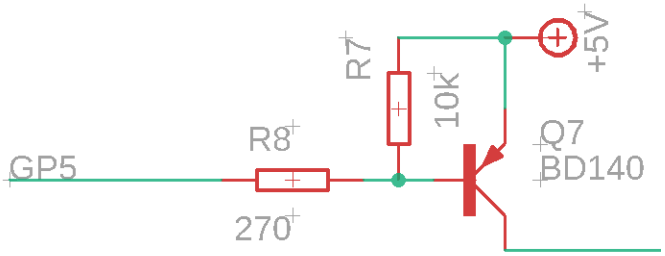


Figure 7. Row select.

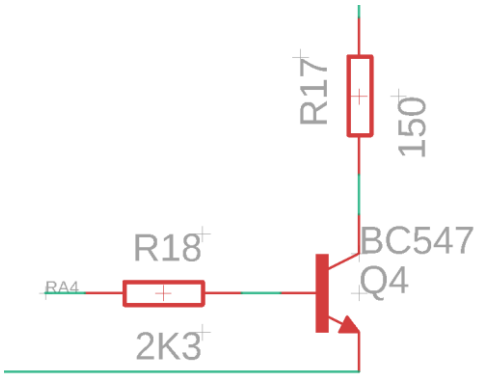


Figure 8. Column select.

In the LED circuit, the row is selected only if the input to the circuit is low and the column is selected when the input is high. The row selector transistors are PNP transistors for higher currents and the column transistors are NPN for the lower currents. Every row is selected ⅙ of the time as every column ⅛ of the time with such frequency that it is not visible.

2.3 Components

2.3.1 Multiplexer

CD74HCT4051E multiplexer was used. This is a digitally controlled analog switch. It uses silicon gate CMOS technology thus it can achieve good operating speeds with low power consumption. The switches have low ON resistance and low OFF leakages. It also has an enable control that can disable all the switches.

2.3.2 IO expander

For the IO expander we chose the MCP23S08 that provides 8 bit parallel I/O expansion. The MCP23S08 consists of multiple 8-bit configuration registers for input, output and polarity selection.

The interrupt capture register captures the port values at the time of the interrupt and saves that condition. The CS (chip select) pin decides the chip's working situation by capturing external chip select signal. The SCK pin receives external clock signal. The SI pin receives the series input signal. And the Power on Reset sets all registers to the default values and initializes the device.

2.3.3 Transistors

We used three different types of transistors all with different characteristics.

2.3.3.1 BC547 NPN low current transistor

This is an NPN type transistor which means that the collector and emitter will be left open (reverse biased) when the base pin is held at ground and will be closed (forward biased) when a signal is provided to base pin. BC547 has a gain value of 110 to 800, this value determines the amplification capacity of the transistor. The maximum amount of current that could flow through the collector pin is 100mA, hence we cannot connect loads that consume more than 100mA using this transistor.

2.3.3.2 BC 557 PNP low current transistor

This transistor has almost the same characteristics as the BC547 but this is a PNP transistor thus collector and emitter will be closed when the base pin is held at ground.

2.3.3.3 BD139 NPN high current transistor

These transistors are designed for audio amplifiers because they can drive a high current load. In this purpose the transistor is used in our circuit.

2.3.3.4 BD140 PNP high current transistor

This transistor has again the same purpose as the NPN version (BD139) but with the difference that it is a PNP so it is forward biased when the base pin is held at ground.

2.3.4 LEDs

For the LED's we used C503B for the red ones, these provide stable light for a long period of time. They have a maximum forward current of 25mA and a maximum power dissipation of 130mW. If the current is 20mA, the forward voltage is typically 2.1 V. For the green LEDs, we used LC503FPG1-30P-A3. They have the same maximum forward current of 25mA but they have a lower power dissipation of 100mW because of the difference in the forward voltage which is 3.2 V at 20mA current.

2.4 Calculations

2.4.1 Triangle wave generating circuit

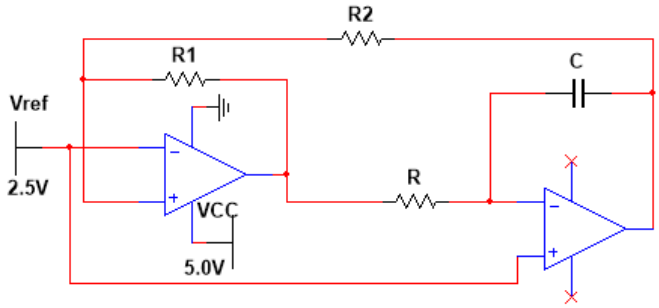


Figure 9. Triangle wave generating circuit.

The first opamp of generating circuit plays a role of hysteresis comparator and the second opamp of this generating circuit is an integration circuit, which transfers square signals to triangle signals for the following parts of sound system. In the integrating process, the integrating time constant is determined by the following formula:

$$f_{triangle} = \frac{R_1}{4RCR_2}$$

In this project, eight notes with specific frequencies were chosen as can be seen from Table 1.

Name of the note	Frequency (Hz)

C (low)	261.63
D	293.66
E	329.63
F	349.23
G	392
A	440
B	493.88
C (high)	523.25

Table 1. The frequencies of the chosen notes.

Therefore, the taken values were $R_1 = 100K\Omega$, $R_2 = 47K\Omega$, $C = 100nF$, and the value of resistor R was calculated and results are in Table 2.

Name of the note	**Resistance(Ω)**
C(low)	203.31K
D	181.13K
E	161.37K
F	152.31K
G	135.69K
A	120.89K
B	107.70K
C(high)	101.66K

Table 2. Calculation results of values of resistors R in triangle wave generating circuit.

Considering the tolerance of the resistors to which the frequency is sensitive, we implemented the circuit by replacing R with R and a potentiometer. The improved circuit is shown below.

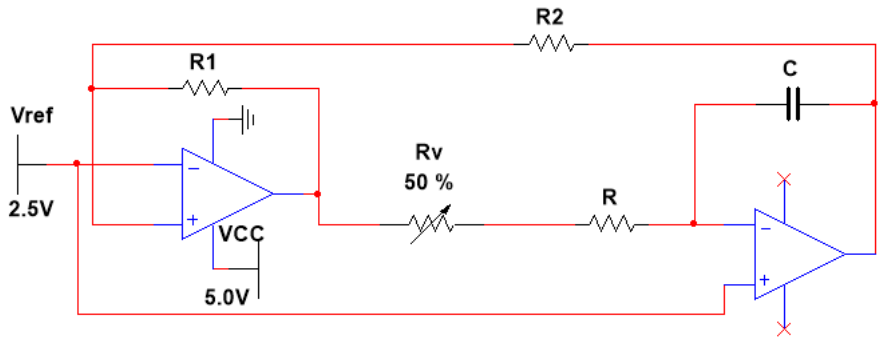


Figure 10. Implemented generating circuit.

The final obtained values are listed in Table 3.

Name of the note	Resistance of R (Ω)	Resistance of Rv (Ω)
C(low)	180K	47K
D	180K	47K
E	150K	47K
F	150K	47K
G	120K	47K
A	120K	47K
B	100K	47K
C(high)	100K	47K

Table 3. Values of resistors in implemented circuit

2.4.2 Loudspeaker driver circuit

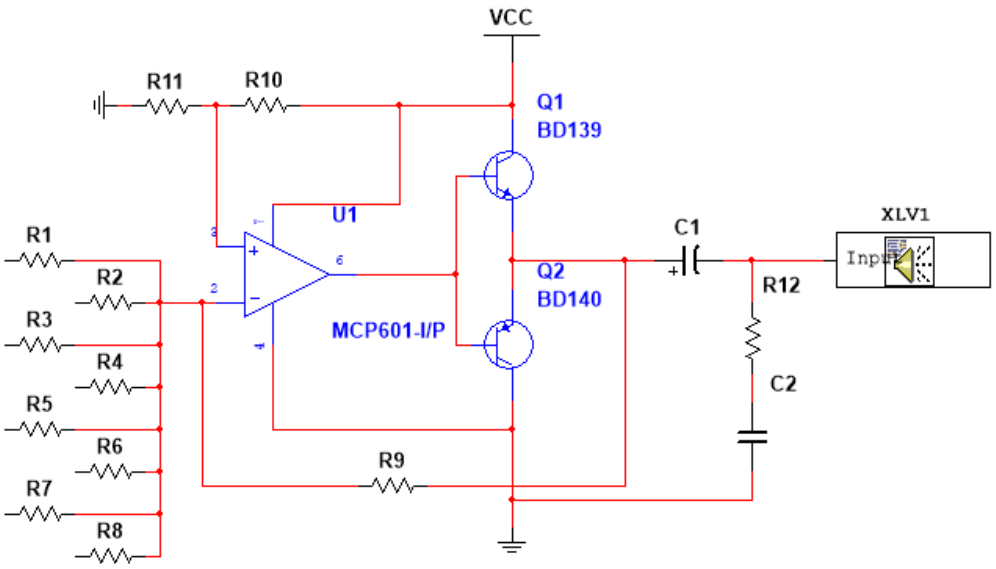


Fig 11. Loudspeaker driver circuit.

Driver circuit of the loudspeaker is composed of an operational amplifier, a power amplifier and a few necessary resistors and capacitors. The circuit plays three major roles, one is adding all signals together, another is providing stable and sufficient current supply, the last is amplifying each signal with certain gain, which is determined by

$$A_v = \frac{R_{output}}{R_{input}} = \frac{R_i}{R_9}$$

From the calculation, the amplitude of the wave is approximately 1V. Thus, we only need to set the R_{output} and R_{input} the same $10K\Omega$. The resistors R11 and R10 are voltage divider resistors to supply 2.5V virtual ground to the positive side of the opamp. Thus, they should have same value. Here we chose

$$R_{10} = R_{11} = 10K\Omega$$

R12 and C2 are to balance the the resistance and inductance inside the loudspeaker.

Therefore, we use the values from the slides.

$$R_{12} = 12\Omega$$

$$C_2 = 100nF$$

The C1 is the coupling capacitor. The value of it needs to be big enough to filter the DC component in the signal so it does not attenuate the AC signal at the same time. Thus we determined C1 to be

$$C_1 = 100\mu F$$

2.4.3 LED circuit

For BD140 (see Fig. 7):

$$\beta \approx 100$$

$$I_B = \frac{I_c}{\beta} = \frac{0.16\ A}{100} = 0.0016\ A$$

To make sure to have the transistor is in saturation mode, the value is divided by factor of 10.

$$R_8 = \frac{V_{cc} - V_{BE}}{I_B} \div 10 = \frac{5-0.7}{0.0016} \div 10 = 268.75\ \Omega \rightarrow \text{took } 270\ \Omega$$

For BC547 (see Fig.8):

β is between 200 and 450.

$$R_{17,green} = \frac{V_{cc} - V_{green} - V_{sat,1} - V_{sat,2}}{I_C} = \frac{5-3.2-0.2-0.2}{0.02} = 70\ \Omega$$

$$R_{17,red} = \frac{V_{cc} - V_{red} - V_{sat,1} - V_{sat,2}}{I_C} = \frac{5-2.1-0.2-0.2}{0.02} = 125\ \Omega$$

To attain the desired brightness of the LEDs, the resistance value was chosen to be 150 Ω.

$$I_B = \frac{I_c}{\beta} = \frac{0.02\ A}{200} = 0.1\ mA$$

$$R_{18} = \frac{V_{RA4} - V_{BE}}{I_B} \div 10 = \frac{5-0.7}{0.0001} \div 10 = 4300\ \Omega \rightarrow \text{took 2K3 } \Omega^*$$

*Due to calculation error smaller resistor was chosen. This causes some energy loss in the system but will not be dangerous for the components.

2.5 FSM program

2.5.1 FSM game program

There are in total 6 states in the project. These are IDLE, CHOOSE LEVEL, READY-SET-GO, PLAY, SHOW SCORE and EMERGENCY state. FSM game diagram can be found in Appendix 1. In the following, the general and the programming (if necessary) logic of states is explained.

2.5.1.1 IDLE state

This state is the default state of the game. When the programme begins, it will always start from this state and it can be transferred to IDLE state from all states when the RESET button is pushed. In the IDLE state, the uppest and lowest row of LEDs are ON and green.

2.5.1.2 CHOOSE LEVEL state

In CHOOSE LEVEL state, the player can choose the level of the piano game they prefer. There are three levels in total and are shown with roman numbers on the LED grid. The level can be changed only from the IDLE state or at the CHOOSE LEVEL state. If there is no operation in CHOOSE LEVEL state for more than 20 seconds, it will return to IDLE state.
In this state, the level counter will store the level data, which increases one unit for each push of LEVEL button.

2.5.1.3 READY SET GO state

When the player is ready to begin the piano game, this state serves as a count down for players to be ready.
This state can only be started from IDLE or CHOOSE LEVEL state when the START button is pushed. When counter is greater than 3, it will proceed PLAY state.

2.5.1.4 PLAY state

This is the main state which also holds the largest amount of complexities in it. This state can only be started from READY SET GO state. In PLAY state, the LEDs show the song the user has to play and at the same time shows the score change and run-time feedback if the user played correctly (Figure 12). In the beginning, the score is maximum (100%) and it is reduced with every mistake.

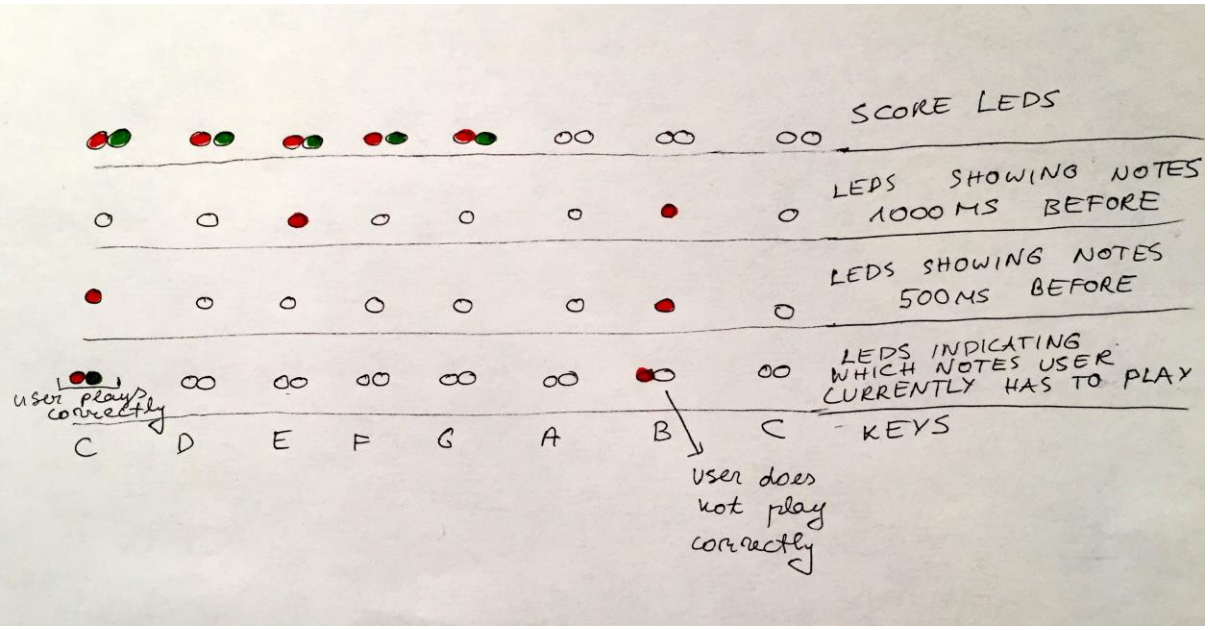


Figure 12. LEDs' arrangement and functionality.

In the beginning of the program, the songs are defined as two-dimensional arrays (Appendix 4). Each note of the song has a beginning time and an ending time which is why each song has two times of two-dimensional arrays. Also, to reduce the work of the microcontroller, the 2D song arrays are not iterated through with every loop but the row of the 2D array will be incremented when the maximum time of one subarray is reached. Therefore, each song has also a 1D array of maximum times. It cannot be in the ending time 2D array as in songs, as there can be pauses in song.

There are 3 rows of LEDs showing what to play and they have a time difference of 500 ms. Therefore, there are three counters as well, where every next counter is 500 ms behind. The LED will be ON if the particular counter is between the beginning time and ending time of the note. If the last counter is between the beginning time and ending time, the user has to play the note. If he plays it, the green LEDs in that row also turns on. If he misses the note, the score will be reduced. This state ends when the counter reaches the length of the song and it will proceed to SHOW SCORE state. Code examples of PLAY state are in Appendix 5 and 6.

2.5.1.5 SHOW SCORE state

SHOW SCORE state shows the score user obtained with the game. Essentially, it just holds the score (which was already visible in the PLAY state) for 5 seconds and then goes back to IDLE state. In this state, the number of leds glowing in the row of leds illustrates the score.

2.5.1.6 EMERGENCY state

This state plays a role as a pause button for safety reasons. It can be transferred from every states when the STOP button is pushed and it can go back to the previous state to the exact same spot when the STOP button is pushed again. When the programme is in this state, all LEDs are off.

2.5.2 The multiplexer program

We have eight piano keys that we want to check constantly during the game. For that, the multiplexer is used, thus a program that tells which selection lines have to be ON for which key is necessary. FSM is used for this program in which each key defines a state. The states look like depicted in Figure 13.

```
case READ_S1 :
    key_C1 = READ_MUX;
    MUX_S1 = 1;
    MUX_S2 = 0;
    MUX_S3 = 0;
    current_switch = READ_S2;
    break;

case READ_S2 :
    key_D = READ_MUX;
    MUX_S1 = 0;
    MUX_S2 = 1;
    MUX_S3 = 0;
    current_switch = READ_S3;
    break;
```

Figure 13. Multiplexer program.

As can be seen from Figure 13, first they read the input from the multiplexer and only after that the selection lines get their value. These values are already defining the selection lines for the next state, in this way the system has enough time to put the selection lines right before it has to read the input from the multiplexer.

2.5.3 The demultiplexer program

As can be seen from Figure 6, there are 48 leds that have to be controlled. To do this, a serial IO expander which controls 8 lines, all the rows and two columns (from the demultiplexer), and 6 outputs from the microprocessor for the remaining liners. A part of the demultiplexer program is shown in Figure 14.

```
case ROW_SCORER:
    IOEXP_send(252, 0);
    for(int i= 0; i<6; i++)
    {
        if(i==0)
        {
            LED1_DEMUX_OUT= score_ledsR[0];
        }
        if(i==1)
        {
            LED2_DEMUX_OUT= score_ledsR[1];
        }
        if(i==2)
        {
            LED3_DEMUX_OUT= score_ledsR[2];
        }
        if(i==3)
        {
            LED4_DEMUX_OUT= score_ledsR[3];
        }
        if(i==4)
        {
            LED5_DEMUX_OUT= score_ledsR[4];
        }
        if(i==5)
        {
            LED6_DEMUX_OUT= score_ledsR[5];
        }
    }
    for(int i= 1; i<3; i++)
    {
        if(score_ledsR[i+5]== 1)
        {
            io+= j[i-1];
        }
    }

    io-=128;
    IOEXP_send(io, 0);
    io= 252;
    current_row= ROW_SCOREG;
    break;
```

Figure 14. Demultiplexer program.

Each state represents a row in the demultiplexer. In the first line it sends data to the IO expander to puts everything off. Then it gives the columns that are directly controlled by the microprocessor the right value (0 if the led has to be off and 1 if it has to be on). Then it checks

the remaining two columns which are controlled by the IO expander. “j” is a small array {2, 1} so it counts the right number with the IO expander. In the last step, the right amount gets subtracted from the “io” value, this depends the row that goes on. The IO expander sends its data to the circuit, the program goes to the next state/row and everything starts again.

2.6 Measurements

2.6.1 The red LED

The voltage drop was measured by the oscilloscope

$$U_{forward} = 2.0V$$

After checking the datasheet, when the forward current of red LED is 2.0V, the current was approximately

$$I_{forward} = 10mA$$

Thus, the total power of each red LED in our project is

$$P = U_{forward} I_{forward} = 20.0mW$$

2.6.2 The green LED

The voltage drop was measured by the oscilloscope

$$U_{forward} = 3.1V$$

After checking the datasheet, when the forward current of green LED LC503FPG1-30P-A3 is 3.1V, the current is approximately

$$I_{forward} = 12mA$$

Thus, the total power of each green LED in our project is

$$P = U_{forward} I_{forward} = 37.2mW$$

2.6.3 The loudspeaker

The power of loudspeaker is determined by input signal. Therefore, the rms voltage was measured when different notes were played. Then the active power dissipation was calculated by

$$P = \frac{U_{rms}^2}{R}$$

The results of rms voltages and (active) powers are shown below.

Note	RMS voltage (mV)	Active power(mV)
C(low)	743	69.0
D	744	69.2
E	748	69.9
F	760	72.2
G	732	67.0
A	710	63.0
B	712	63.4
C(high)	710	63.0

3. Enterprising Part

3.1 Target group

3.1.1 Description

The target group of this product is mainly young people, from the age of 10 to 25 years old who want to learn the piano but who are feeling too old for music school or are not taking it to seriously. Hence, this group of people are searching for an easy way to start to play the piano, they want a low threshold. They have no knowledge of music but there is a will to play the piano. It needs to be affordable and easy to handle. The quality is less important because they

just start to play the piano and are using this product to learn how to play it. We have a fairly large target group, there are a lot of students that start to play a new musical instrument on their own. The target group will not grow over the years because their will not be more people who get the intention to play the piano.

3.1.1.Consequences on product

Based on the age of our target group, we know that the biggest part of our group do not work yet and do not have a big budget to start to play the piano. Our product has not as much keys as a real piano and the sound and the touch of the keys are not as good as a real piano so the price needs to be as low as possible. They do not want it to be more than that because they want an affordable object, so it is a conscious choice to make a limited piano. Our target group wants an easy way to start to play the piano as they do not want to learn musical notes or theories regarding the rhythm. This is why the LEDs are showing which keys they need to play at what time. Because of this, they do not need any foreknowledge. In other words, the threshold to start playing is low because of that. The LEDs and the score bar give it a play factor so this also lowers the threshold even more. The young age of our target group might mean that they like to play games. The piano is easy to handle, it is small enough to fit in the hand. The keys and buttons are named on the keys, thus one knows immediately what every keys means. They do not want the best quality so we have chosen for a small loudspeaker, this costs less than a sophisticated one and it does its job. It also reduces the price and that is also something they are interested in. The materials that we use are not too expensive either, we have chosen them on functionality and price instead of quality.

4. Conclusion

This semester we created an electronic game for the course EE4. The product is a piano game built for people at the age of 10 to 25 years old who want to learn to play the piano. The LEDs on the piano indicate which note the user has to play at what time. There is a bar that indicates the user’s score.

Technically, the game is divided into three parts. The control system controls the inputs and outputs of the whole game. There are more outputs and inputs needed than there are pins on our microprocessor, thus the multiplexer and the IO expander were used. The second part is the sound system. That, in turn, is divided into three parts. First, there is a triangular wave

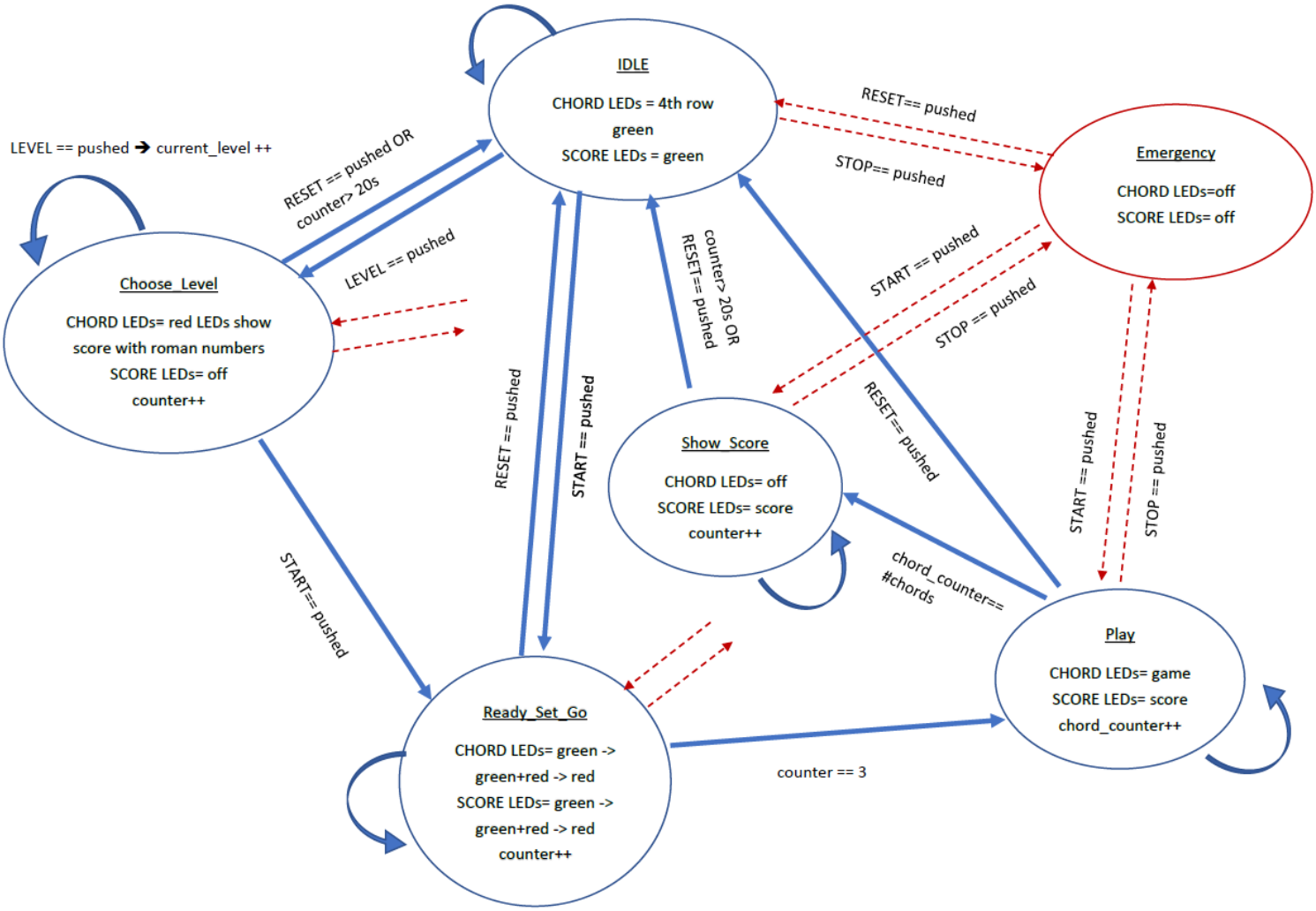
oscillator that makes a signal with a specific frequency. After that, the keys connect the signal to the second part, the adding circuit. When the keys are pressed there is also a signal going to the control system. The adding circuits add all the signals together when there are more keys pressed at the same time. Finally, there is a power amplifier that drives the loudspeaker. The last part is the LED system. There are far more LEDs than output pins so there is a IO expander that convert one serial input into an eight bit output but this is not enough. There is also a demultiplexing layout with rows and columns that can be selected. Next time we are doing this project we would make clear that the PCB is perfect and without errors. The communication between the team members went well but the language difference stays difficult. We are happy with the outcome of the project and we have learned a lot.

5. References

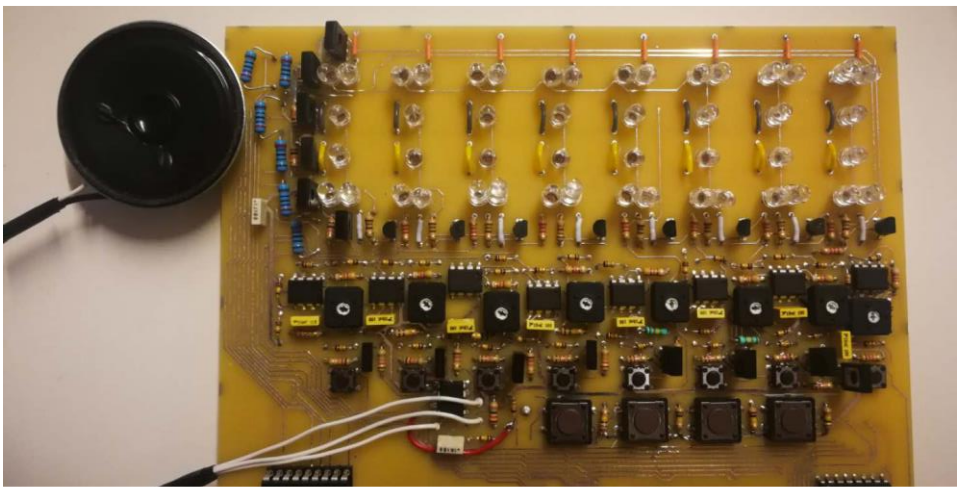
EEEGUIDE. (2019). *Triangular Wave Generator Using Op amp | EEEGUIDE*. [online] Available at: http://www.eeeguide.com/triangular-wave-generator-using-op-amp/ [Accessed 14 May 2019].

6. Appendixes

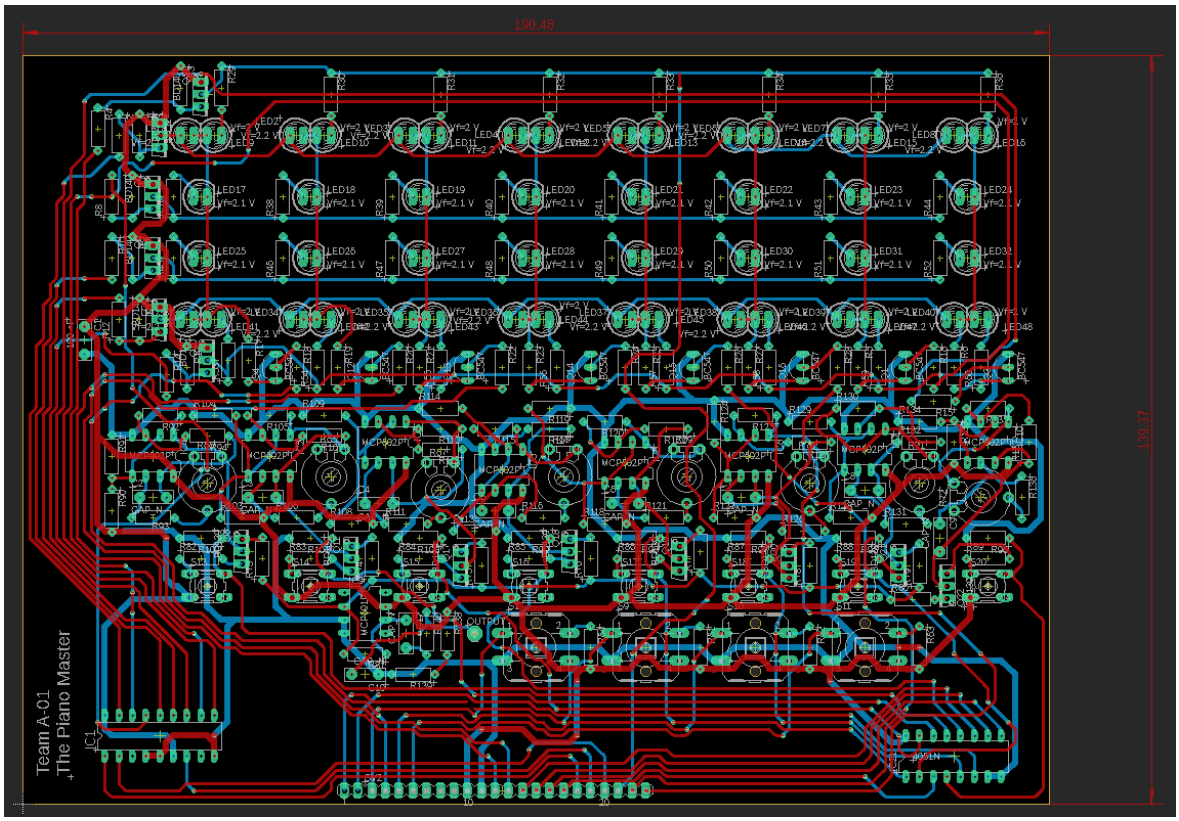
Appendix 1. FSM diagram.



Appendix 2. Picture of final PCB.



Appendix 3. Screenshot of board layout.



Appendix 4. FSM game program song definition.

```
//Level 1 song
static int max_times1[3] = {4500, 6000, 8000};
static int b1[3][8]={{0, 1500, 2000, 4000, 3000, 0, 0, 0}, {0, 5000, 4500, 0, 0, 0, 0, 0}, {6000, 0, 0, 0, 0, 0, 0, 0}};
static int e1[3][8]={{1500, 2000, 3000, 4500, 4000, 0, 0, 0}, {0, 6000, 5000, 0, 0, 0, 0, 0}, {8000, 0, 0, 0, 0, 0, 0, 0}};


//Level 2 song
static int max_times2[13] = {1500, 2000, 3500, 6000, 8000, 8500, 10000, 10500, 12000, 13000, 14000, 15000, 16000};
static int b2[13][8]={{0, 500, 1000, 0, 0, 0, 0, 0}, {1500, 0, 0, 0, 0, 0, 0, 0}, {2000, 2500, 3000, 0, 0, 0, 0, 0}, {3500, 0, 4000, 4500, 5000, 0, 0, 0},
                      {0, 0, 6000, 6500, 7000, 0, 0, 0}, {0, 0, 0, 0, 8000, 8250, 0, 0}, {9500, 0, 9000, 8750, 8500, 0, 0, 0},
                      {0, 0, 0, 0, 10000, 10250, 0, 0}, {11500, 0, 11000, 10750, 10500, 0, 0, 0}, {12000, 0, 0, 0, 12500, 0, 0, 0}, {13000, 0, 0, 0, 0, 0, 0, 0},
                      {14000, 0, 0, 0, 14500, 0, 0, 0}, {15000, 0, 0, 0, 0, 0, 0, 0}};
static int e2[13][8]={{500, 1000, 15000, 0, 0, 0, 0, 0}, {1800, 0, 0, 0, 0, 0, 0, 0}, {2500, 3000, 3500, 0, 0, 0, 0, 0}, {4000, 0, 4500, 5000, 6000, 0, 0, 0},
                      {0, 0, 6500, 7000, 7800, 0, 0, 0}, {0, 0, 0, 0, 8250, 8500, 0, 0}, {10000, 0, 9500, 9000, 8750, 0, 0, 0},
                      {0, 0, 0, 0, 10250, 10500, 0, 0}, {11800, 0, 11500, 11000, 10750, 0, 0, 0}, {12500, 0, 0, 0, 13000, 0, 0, 0}, {13800, 0, 0, 0, 0, 0, 0, 0},
                      {14500, 0, 0, 0, 15000, 0, 0, 0}, {16000, 0, 0, 0, 0, 0, 0, 0}};
```

Appendix 5. FSM game program PLAY state example 1.

```
    case FSM_PLAY :
// *** outputs ***

        play_counter1++;
        play_counter2++;
        play_counter3++;
        show_score(score_leds_on);
        int a = play_counter1;
        int c = play_counter2;
        int d = play_counter3;

        if (current_level == 1) {

            song_length = 8000;
            counter_level_1++;

            //If the maximum time of one song row is reached, change to the next
            if (max_times1[sr1] < play_counter1) {
                sr1 = (sr1 + 1u) % 3;}
            if (max_times1[sr2] < play_counter2) {
                sr2 = (sr2 + 1u) % 3;}
            if (max_times1[sr3] < play_counter3) {
                sr3 = (sr3 + 1u) % 3;}

            //If the play_counter1 is between the beginning time and end time of the note, LED will be ON
            set_play_array(leds_1, (a>=b1[sr1][0] && a<e1[sr1][0]), (a>=b1[sr1][1] && a<e1[sr1][1]),
                    (a>=b1[sr1][2] && a<e1[sr1][2]), (a>=b1[sr1][3] && a<e1[sr1][3]),
                    (a>=b1[sr1][4] && a<e1[sr1][4]),
            (a>=b1[sr1][5] && a<e1[sr1][5]), (a>=b1[sr1][6] && a<e1[sr1][6]),
                    (a>=b1[sr1][7] && a<e1[sr1][7]));
            //Similar to previous one, only here play_counter2 is used which is 500ms behind
            set_play_array(leds_2, (c>=b1[sr2][0] && c<e1[sr2][0]), (c>=b1[sr2][1] && c<e1[sr2][1]),
                    (c>=b1[sr2][2] && c<e1[sr2][2]), (c>=b1[sr2][3] && c<e1[sr2][3]), (c>=b1[sr2][4] && c<e1[sr2][4]),
            (c>=b1[sr2][5] && c<e1[sr2][5]), (c>=b1[sr2][6] && c<e1[sr2][6]), (c>=b1[sr2][7] && c<e1[sr2][7]));
```

Appendix 6. FSM game program PLAY state example 2.

```
    //If play_counter3 is between beginning and ending time and the correct piano key is pushed, this led will go green
     set_play_array(leds_3G, (d>=bl[sr3][0] && d<el[sr3][0] && key_Cl==1), (d>=bl[sr3][1] && d<el[sr3][1] && key_D==1),
            (d>=bl[sr3][2] && d<el[sr3][2] && key_E==1),
    (d>=bl[sr3][3] && d<el[sr3][3] && key_F==1), (d>=bl[sr3][4] && d<el[sr3][4] && key_G==1),
            (d>=bl[sr3][5] && d<el[sr3][5] && key_A==1),
    (d>=bl[sr3][6] && d<el[sr3][6] && key_B==1), (d>=bl[sr3][7] && d<el[sr3][7] && key_Ch==1));

    //Conditions where score is reduced
    int h =(d>=bl[sr3][0] && d<el[sr3][0] && key_Cl==0);
    int i = (d>=bl[sr3][1] && d<el[sr3][1] && key_D==0);
    int j = (d>=bl[sr3][2] && d<el[sr3][2] && key_E==0);
    int k = (d>=bl[sr3][3] && d<el[sr3][3] && key_F==0);
    int l = (d>=bl[sr3][4] && d<el[sr3][4] && key_G==0);
    int m = (d>=bl[sr3][5] && d<el[sr3][5] && key_A==0);
    int n = (d>=bl[sr3][6] && d<el[sr3][6] && key_B==0);
    int p = (d>=bl[sr3][7] && d<el[sr3][7] && key_Ch==0);

    reduce_score(h, i, j, k, l, m, n, p);

}

else if (current_level == 2) {

    counter_level_2++;
    song_length = 16000;
```