DEPARTMENT OF ELECTRONIC AND TELECOMMUNICATION ENGINEERING

UNIVERSITY

OF

MORATUWA

EN1093 - Laboratory Practice - I

SIMPLE VOICE RECORDER

GROUP 09

DATE OF SUBMISSION:

16/08/2021

GROUP MEMBERS	INDEX NUMBER
1. GAJAANAN S.	190185D
2. GAJOASHAN V.	190186G
3. GUNASEKARA M.S.K.	190202F
4. HETTIHEWA D.P.G.	190231R

Supervisor **:** Mr. HIRAN PERERA

TABLE OF CONTENTS

ABSTRACT……………………………………………………………………2

1. INTRODUCTION…………………………………………………………2

2. METHODS………………………………………………………………....2

    2.1 COMPONENTS ANALYSIS…………………………………………....2

    2.2 PCM THEORIES……………………………………………………....3

    2.3 PLAYBACK EFFECTS…………………………………………………4

    2.4 ALGORITHM DEVELOPMENT……………………………………....5

    2.5 SIMULATIONS AND INPUT-OUTPUT WAVEFORMS…………….....5

    2.6 SCHEMATIC AND PCB DESIGN…………………………………...6

    2.7 ENCLOSURE DESIGN………………………………………………..7

3. RESULTS…………………………………………………………………8

    3.1 USER INTERFACE…………………………………………………..8

    3.2 PHYSICAL PROTOTYPE…………………………………………...8

4. DISCUSSION……………………………………………………………...9

5. ACKNOWLEDGEMENTS……………………………………………....10

6. REFERENCES…………………………………………………………10

7. APPENDICES…………………………………………………………11

SIMPLE VOICE RECORDER

ABSTRACT

The main objective of this project was to design and implement a voice/sound recorder capable of simple voice modulation. Hence, we have developed a simple voice recorder that can playback audio which is recorded by the user through a microphone. The user is also able to implement a low-pass filter effect to recorded audio and playback the modified sound as well. Up to 3 recordings without any effects can be saved in the device and can be selectively played back, and we have made this using the ATmega328p microcontroller and various other inexpensive electronic components. Currently we have developed the physical prototype as a breadboard implementation. Furthermore, we have been involved with the PCB and enclosure designing which will influence the production of the finished product as a commercially successful device. Problems, improvements and alternatives regarding this project were also discussed and evaluated.

1. INTRODUCTION

This report explains in length, the design and manufacture of the of a simple voice recorder, through which the user can record multiple audio tracks, and play them back selectively, as recorded or by adding certain audio enhancements. This paper further details the process of the PCB design and also elaborates on the appropriate enclosure design process.

When designing the functionality of the prototype, many critical factors have to be considered. Firstly, this project is implemented using appropriate ADC and DAC with 8 bit PCM. Hence, aspects such as sampling rate, quantization and compression needed for the PCM should be evaluated when doing so.

Furthermore, the storage of recorded audio tracks in an SD card is another point of importance, with the number of samples stored, the file format used for storage etc. needed to be considered as well. With regards to audio enhancement, the type of effects to be implemented are essential to be identified when developing the algorithm. Next, concerning the perspective of the user of the prototype, elements such as input device, UI and output

device design should be executed. The use of an appropriate microphone module and any other required input devices, an LCD screen to act as a clean and simple UI and a speaker with an appropriate amplification setup are included in this.

Thus, the aforementioned features were experimented with using various parameters and variables and weighing out the pros and cons before they were finalized for this project. Hence, this paper details the attempt to develop a working prototype (breadboard implementation) of a simple voice recorder using the ATmega 328p microcontroller alongside other affordable electronic components, in a suitable way that can be expanded for commercial use later on. To complement this, various appropriate software were used to design appropriate PCB and enclosure designs, which will be elaborated in greater detail in the latter part of this report.

2. METHODS

2.1 COMPONENTS ANALYSIS

2.1.1 List of components used

I. Modules and External components

- 1 x KY-037 Microphone module
- 1 x 16x2 Alphanumeric LCD Display
- 1 x Micro SD Card Module
- 1 x PCF8574 LCD I2C
- 1 x 4x4 Keypad
- 1 x TPA3118 Amplifier module
- 1 x 8Ωspeaker
- 1 x 10 KΩ potentiometer
- 1 x 9V Battery

II.Main PCB

- 1 x ATmega 328p microcontroller
- 1 x DAC0808 IC
- 1 x LM7805CV Power regulator
- 1 x 16 kHz crystal oscillator
- 3 x100 nF capacitors
- 2 x 22 pF capacitors
- 1 x Switch
- 3 x10 KΩ resistors
- 1 x 330 nF capacitor
- 1 x 330Ω resistor

2.1.2 Descriptions of Important Components

A) ATmega 328p microcontroller- It is a single-chip microcontroller with a modified Harvard architecture 8-bit RISC processor core. It acts as the main processing unit of the voice recorder connecting all operations and components to achieve the required purpose.

B) KY-037 Microphone module- This module is used to detect analogue audio inputs given by the user to be recorded. It is connected to the microcontroller through Pin C0 to detect analogue inputs.

C) LCD Display and I2C- The LCD display is used to provide a simple and clean user interface. It has 16 columns, and 2 rows and alphanumeric characters can be displayed as needed. The PCF8574 is a silicon CMOS circuit. It provides general-purpose remote I/O expansion for most microcontroller families via the two-line bidirectional bus (I2C). It has been used in this project to reduce the number of connections from the LCD display to the microcontroller. The connection of the LCD to the microcontroller through the I2C is done through Pin C4 and C5.

D) Micro SD Card module- This module (MicroSD Card Adapter) is used for reading and writing through the file system and the SPI interface driver. The required files can be stored in a MicroSD card which can be attached to this module. For the voice recorder, we use this setup to store recorded audio inputs which can later be retrieved selectively for playback. This module is connected to the microcontroller through Pins B2-B5.

E) Keypad- This 16-button keypad provides a useful human interface component for microcontroller projects. By pressing the required button for desired input, it provides the user with an aspect of control with regards to what operation they need to obtain from the voice recorder. In this project, by inputting certain button combinations, the user can choose to record, selectively playback recordings, add effects and playback and delete recordings. The keypad is only connected for its first three columns and first two rows and is connected to the microcontroller using Pins C1-C3 and B1-B2 respectively.

F) Amplifier module and speaker- This setup is used to emit the playback audio to the listener. A simple 8-ohm speaker is used for this purpose. An amplification module is used to amplify the sound to make it more audible. The TPA3118 Amplifier user here is a Class-D stereo efficient, digital amplifier that can drive speakers up to 60W/8Ω. It is connected with the output of DAC IC.

G) DAC IC- The purpose of the DAC is to take the necessary digital samples that make up a stored recording and turn it back into an analogue signal. The DAC0808 used in this project is an 8-bit monolithic digital-to-analogue converter that interfaces directly with popular TTL, DTL or CMOS logic levels. It has been connected to the microcontroller using the 8 pins in Port D

2.2 PCM THEORIES

2.2.1 What is PCM

PCM or Pulse Code Modulation is a digital modulation technique that is used to convert analog input signals into digital data. In PCM, first we do sampling to convert the continous analog signal into a discrete signal. After that, quantization is done to convert discrete signals into digital signals. Finally, we encode the digital signal.

2.2.2 How PCM is used in the project

For any type of sound signal to be heard clearly, it should be sampled at a suitable sampling frequency. In our case, our signal is a simple voice signal which has a frequency spectrum of 0 to 4 kHz. As the sampling theory suggests a signal should be sampled with at least twice the frequency of the original signal to be able to decode the signal properly. So ideally a voice signal should be sampled at least of a sampling frequency of 8 kHz, which was our target for the project. To achieve this, what we used was timer interrupts. The basic theory behind interrupts is we are telling the timers of our microcontroller to do a specific task after a specific time period continuously. To do this first we have to configure the timer interrupt. The steps to do this using the 8-bit timer 2 is as follows.

- Set the Prescaler value to 8
- Load the TCNT2 register with value 0x05 (See Appendix II for calculation)
- Enable timer 2 overflow interrupt (TOIE2)
- Enable global interrupts
- Set the suitable operation in the ISR function (In this case, resetting TCNT2 to 0x05 and setting flag, a Boolean variable, as true)

After the configuration interrupts will occur at 8khz frequency and sampling will be done at 8khz frequency.

2.2.3 ADC operation

After sampling is done the next steps are quantization and encoding and for this, we can use the built-in 10-bit ADC in our ATmega microcontroller. After we initialized our ADC each sample will be in 10-bit resolution and will be stored in two registers ADCH and ADCL. But for our operation, we need the bit resolution in 8 bits. Because of that, we left align the values of the two registers by setting the ADLAR bit in the ADMUX register. Then, we only take the most significant 8bits which are stored in the ADCH register.

In our code to do this whole operation, we are using two while loops. The first one is to loop the sampling until the stop recording button is pressed and the second one is to wait until an interrupt occur and set the flag as true. When that happens we are reading the ADCH register and storing that value in a CSV file as an 8 bit binary sample (represented by a single ASCII character). Since a certain time is taken for the ADC convertion process, and the convertion can not be interrupted, it is required to wait until the conversion is finished. Instead of keeping the processor idle, we use this time to write the previous sample to the CSV file. With this pipelining technique, and the sample storing format we were able to reach a maximum sampling rate of 19.2 kHz. However, we have limited the sampling rate to 8 kHz for stable operation throughout the operation. This concludes the recording process of the Simple Voice Recorder.

2.2.4 DAC operation

To play a recorded file we should be able to reconstruct the analog signal using the digitalized data stored in our file. For that we need to take each sample from that file in the same sampling frequency we used in the ADC operation. For that, we can use the same timer interrupt we used before. To recreate the same signal, we used an external digital to analog converter (DAC). The DAC we chose for our project is DAC0808. The DAC was connected to the PORTD of our microcontroller and we are directly writing the 8-bit sample value to the PORTD. Writing to the PORTD directly is much faster than any other method including PWM techniques. Because of that there won't be any delays in this operation and therefore, we can recreate a signal which is very close to the voice signal we originally recorded.

2.3 PLAYBACK EFFECTS

2.3.1 Matlab

In this project for the effect to be induced in playback, lowpass filtering is used. For this effect, each sample is read one by one from the pre-recorded voice sample stored in the SD card,and the values corresponding to the filtered output are stored in a new file(effects file). In the playback, the effects file is opened and played. For computing the values for the filtered output, the corresponding Matlab function is found using the first-order lowpass filter transfer function. the Matlab code and the calculation for the transfer function for lowpass first-order filtering done for an audio sample in wave file are provided in appendix I Figure(a) and Figure (b). And the spectrum obtained from the Matlab implementation is given in Figure 2.3.1 (a). Here the audio sample is filtered using a lowpass filter with a cutoff frequency of 1000Hz.

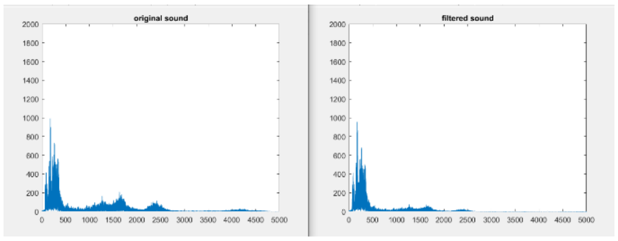

Figure 2.3.1 (a)

Similarly, in Matlab we had tested adding echo effects for the recorded wave file. In this, each sample of the input signal is read and an empty array is created to store the set of past sample data and then combined to give the resultant output. The Matlab code done for the echo effects is provided in Appendix I Figure (c) and Figure 2.3.1 (b) shows the original and resultant spectrum after the echo effects. However, the echo effects part of our testing was not implemented in the final product due to various limitations.

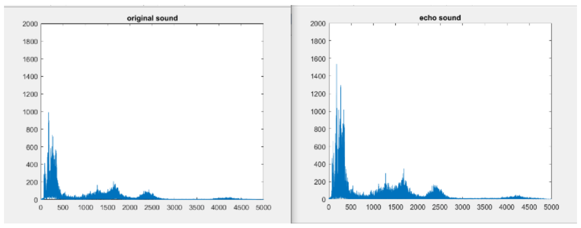

Figure 2.3.1 (b)

2.3.2 Arduino

With the physical implementation, the waveform obtained for the low pass filtering for a random signal is shown in Figure 2.3.2 (c).

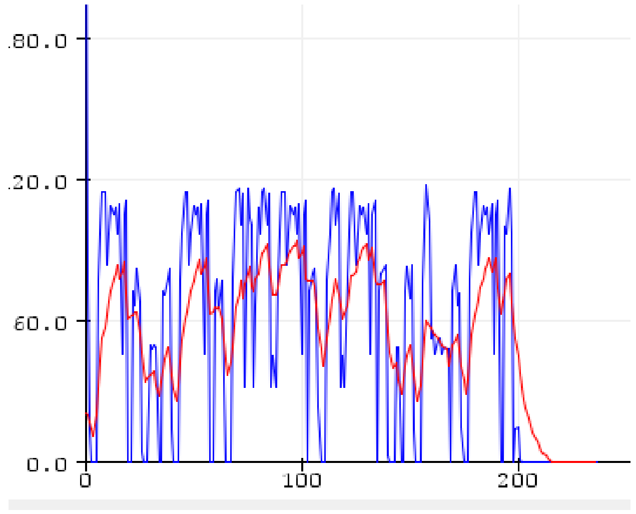

Figure 2.3.2 (c)

2.4 ALGORITHM DEVELOPMENT

The following block diagram is shown in figure 2.4.1 represents the basic functionality of a simple voice recorder. By following these basic steps, we develop the related algorithms and appropriate functions in the Arduino platform and design and simulates in proteus software.

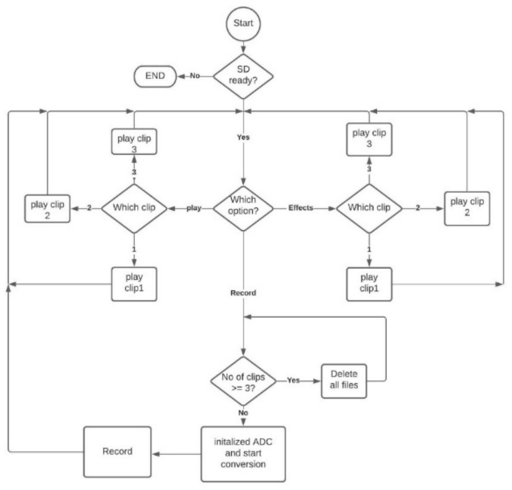

Figure 2.4.1

2.5 SIMULATIONS AND INPUT-OUTPUT WAVEFORMS

The Arduino implementation is simulated through proteus professional software. The circuit design for the proteus simulation is provided in figure 2.5.1.

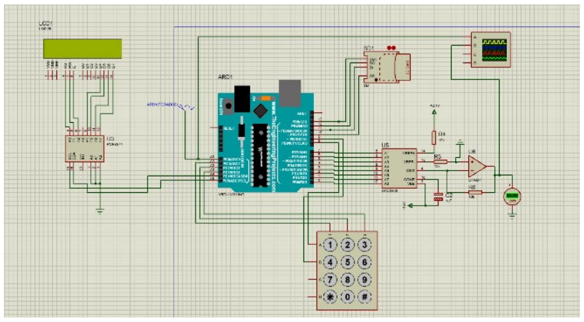

Figure 2.5.1

As an audio sample here sinusoidal signal of a single frequency of 500Hz is used. Samples are taken at a rate of nearly 4000 samples per second and stored in a comma-separated value file format in the SD card. The Arduino code for this implementation is attached in the Appendix III. Figure 2.5.2 shows the input and output waveform obtained in the simulation process.

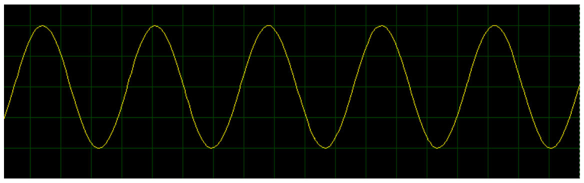

Figure 2.5.2 (a) Input sinusoidal

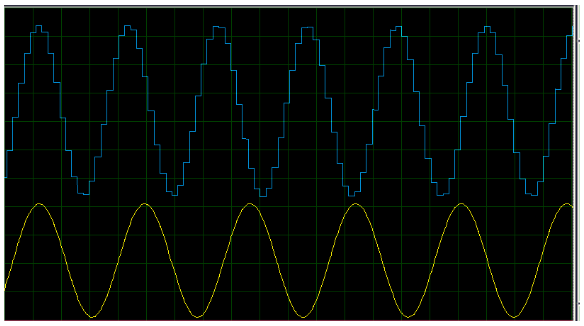

Figure 2.5.2 (b) Input and Output original waveform.

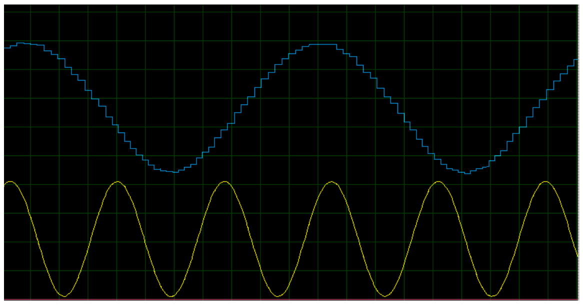

Figure 2.5.2 (c) Input and Output filtered waveform.

2.6 SCHEMATIC AND PCB DESIGN

The PCB design for the simple voice recorder circuit was made in Altium designer software.

PCB design contains three main units they are power supply unit, main microcontroller ATmega 328p, DAC IC.figure 2.6.1 shows each main unit.

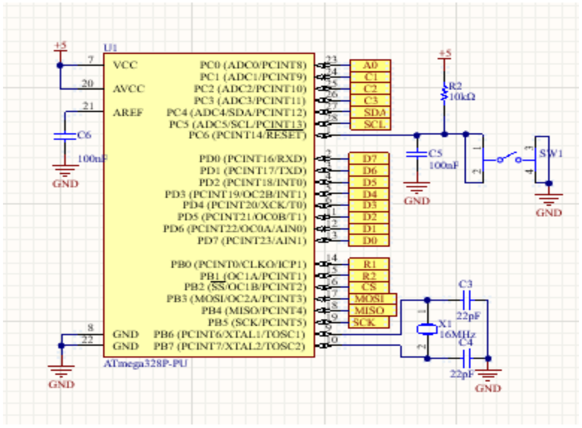

Figure 2.6.1 (a) ATmega 328 PU Schematic

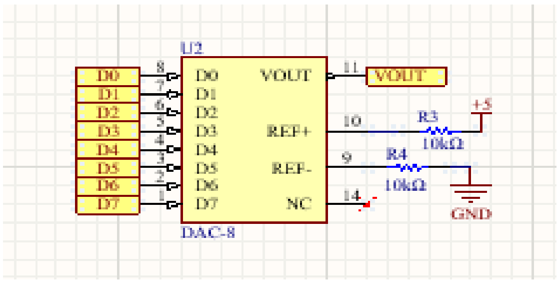

Figure 2.6.1 (b) DAC IC Schematic

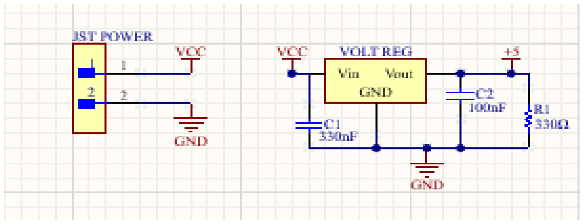

Figure 2.6.1 (c) Power supply Schematic

Other than the aforementioned aspects, JST connectors are used that for the module MIC, Amplifier, Keypad, LCD, SD card module. Figure 2.6.2 shows the final schematic diagram for the PCB design.

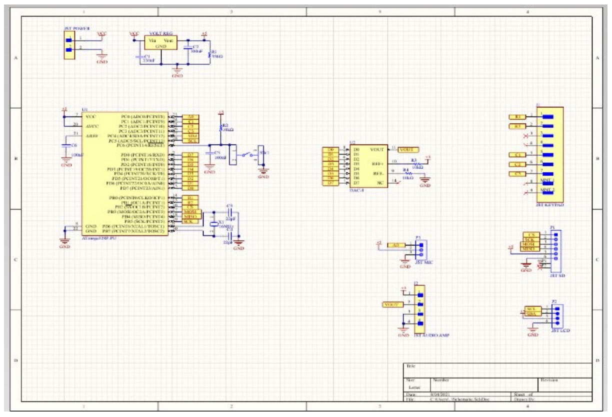

Figure 2.6.2

The final PCB design is made in two layers, which are the top layer and bottom layer. Two-layer routing is done to minimize the routing crossing the ICs to avoid overheating. Always used 45-degree routing bends, because in 90-degree routing bends electron flow is not smooth. Copper pouring is used to balance the heat and dilatation on the top layer of the PCB. All routing is done with the default width of 0.254mm. And there are also two holes presented to mount the PCB design with an enclosure with a hole diameter of 1mm. final PCB design has the dimension of 85mm X 82mm. Figures 2.6.3 shows the 2D and 3D views of the final PCB design.

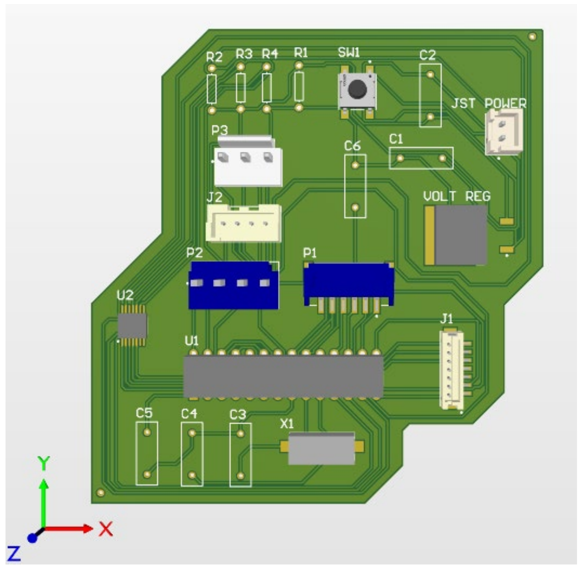

Figure 2.6.3 (a) PCD design 3D view

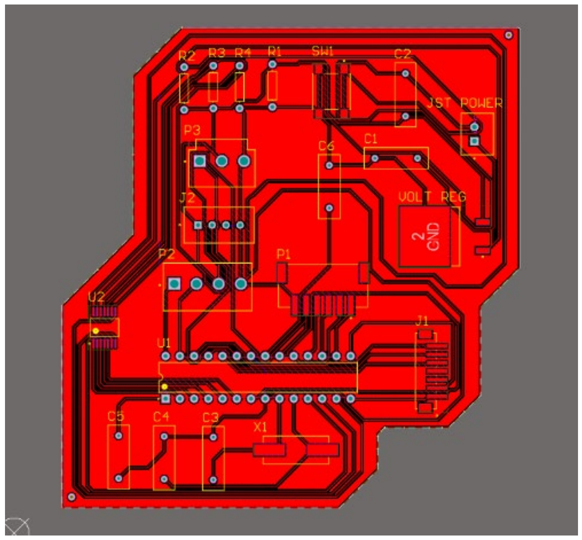

Figure 2.6.3 (b) PCB design 2D view

2.7 ENCLOSURE DESIGN

In designing the enclosure as seen in figure 2.7.1 we considered several factors. Convenient usability was considered when deciding the placement of buttons, display and the microphone. Unscrewing the two plates would reveal all the connections and the battery hostler for easy replacement of parts. Maintaining aesthetic look and feel complementing the functionality, was also a concern. Figure 2.7.2

shows the exploded view of the enclosure with the components.

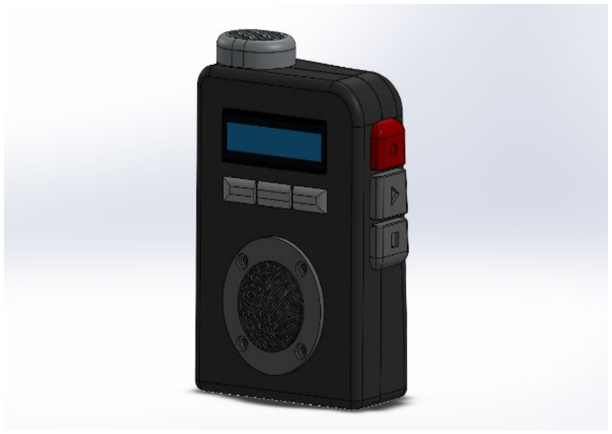

Figure 2.7.1

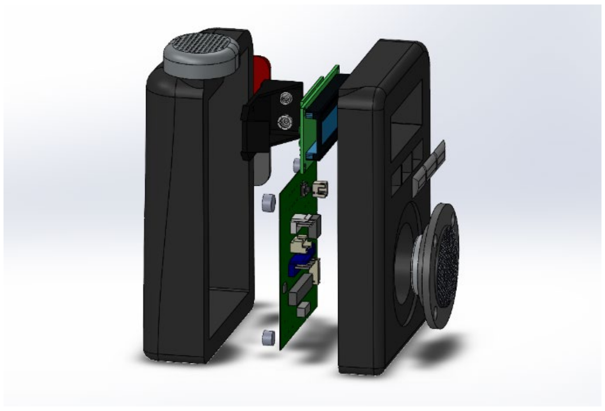

Figure 2.7.2

3. RESULTS

3.1 USER INTERFACE

For the user to interact with our product, we have included a keypad which can be used for selection of the recording option and selective playback of the different recorded files with or without effects. From the simulation stage we are developing the final product which will work as follows.

When the user turns on the device first a greeting message will be shown, then the selective option will be displayed. Six selective buttons are present for recording, playing the original file, playing the filter file, and selecting the appropriate audio clip. In the simulation, we designed for our device to contain upto three recordings. If we try to go beyond three files, there will be a prompt to delete the files first. Figure 3.1.1 shows how we implemented the user interface in the simulation.

To start recording, the user should press the record button (number 6) and after the subsequent press of the record button, the recording will be completed. After that, to play the original file, the user should press the play button (number 4). Then they have to select the file number, whether it is one, two or three. Similarly, for the play effects, the user should press the effect button (number 5) and choose the respective audio clip number.

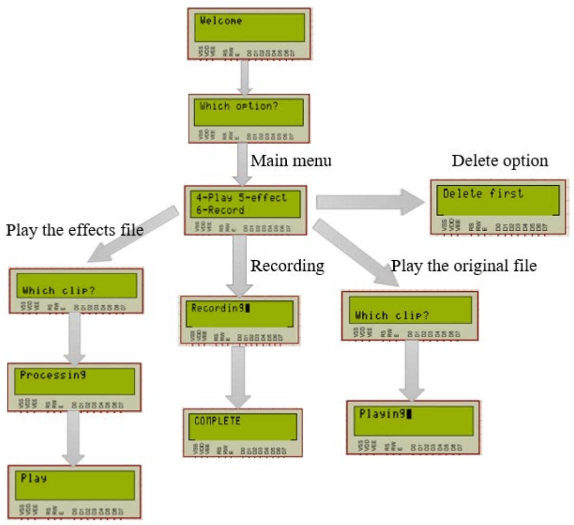

Figure 3.1.1

3.2 PHYSICAL PROTOTYPE

The following image in the figure 3.2.1 shows the physical circuit implementation using the breadboard. Using the aforementioned theories, simulations and components mentioned in the METHOD section, we have designed this circuit in a user-friendly way to achieve effective operation.

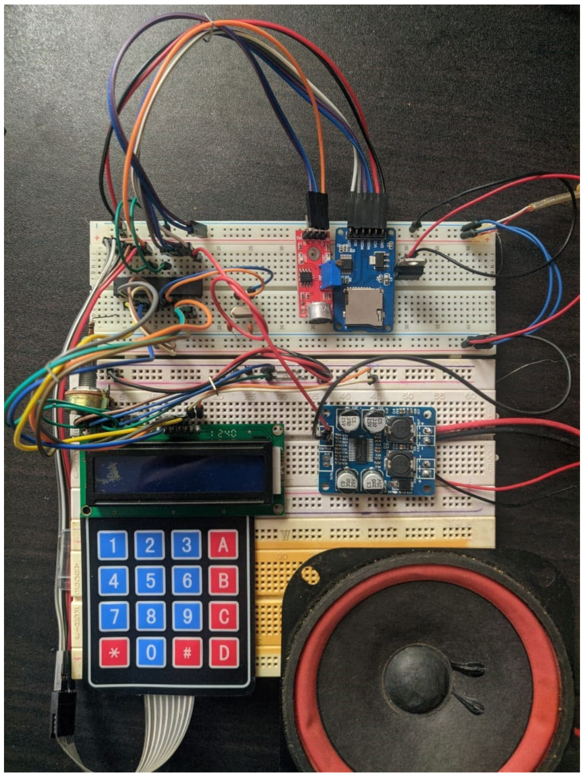
Figure 3.2.1

4. DISCUSSION

The current voice recorder we develop had some issues. The problems in the current designs are

1. Designed to record only three voice clips audio files in text format with a short duration of the period.
2. The recorded audio contains a lot of internal and external noise.
3. For the playback effects, only lowpass filtering is done and since the filtering process takes place during the playback it consumes a long duration for short audio effects to play.
4. Here we collect samples at a rate of 4000 samples per second but to regenerate the voice track back clearly, we need to implement the recorder with the minimum sampling rate of 8000 samples per second.
5. We used to save each recorded sample in comma-separated value format. This format might be insufficient for the audio recording and need to include some basic information about the recording.

As an improvement from the current design, we can add some additional features as follows try to include many recorded files to be saved in the SD card, use wave file format for recording each voice clip, adds more play effects feature like different filtering techniques and echo effects, implementing with the reduction of noise in recording and playback of the audio file and as for the commercial use finally developed a product with PCB and enclosure.

We can also use some alternative modules for the current design we are developed. We can use different input/output modules for the user interface. Instead of using a 16x2 LCD display and a 4x4 digital keypad, we can use an OLED display and push-button/other digital keypads. Use of different DAC modules and Amplifier modules.

In addition to the aforementioned problems, improvements and alternatives with regards to the circuit design, we as a team faced many external challenges during this project. Firstly, since this entire project was fully done through an online workspace as we were unable to meet in person, there were many tests which arose due to this. The main challenge was having to get familiarized with many different software (simulation software etc.) to facilitate working online. There were several bugs/ other issues we encountered when using these software, which were able to be solved after some research. Another important challenge we faced was reaching certain milestones at the required time, which we definitely would have achieved faster if we had the opportunity to meet in person. Furthermore, we also faced issues procuring certain components on time through online order due to various delays, lack of stock etc. which hindered our design process at certain stages. However, for the most part, we were able to overcome a majority of these challenges and deliver a satisfactory end product.

5. ACKNOWLEDGEMENTS

We, as a team, sincerely thank our direct project supervisor, Mr Hiran Perera for his continuous support, advice and invaluable feedback during our various weekly project assessments. Without his mentorship, we would have never have made the progress we accomplished during this project, and we extend our humble gratitude to him. We would also like to thank all other supervisors involved in the voice recorder project for their various advice and feedback during the common meetings held for all the groups and on the relevant forums as well.

6. REFERENCES

1. Scott Campbell, Arduino LCD Set Up and Programming Guide, https://www.circuitbasics.com/how-to-set-up-an-lcd-display-on-an-arduino/

2. Kashif Mirza, Interfacing SD Card module with Arduino in Proteus, April 27, 2021, https://projectiot123.com/2021/04/27/interfacing-sd-card-module-with-arduino-in-proteus/

3. 4x4 Keypad Interfacing With Arduino UNO, https://www.electronicwings.com/arduino/4x4-keypad-interfacing-with-arduino-uno

4. Interfacing LCD16x2 with AVR ATmega16/ATmega32 in 4-bit mode, https://www.electronicwings.com/avr-ATmega/interfacing-lcd-16x2-in-4-bit-mode-with-ATmega-16-32-

5. Interfacing a (micro)SD card with an ATmega328 microcontroller, September 26, 2015, https://shepherdingelectrons.blogspot.com/2015/09/interfacing-with-microsd-card-with.html

6. Make Your Own Spy Bug (Arduino Voice Recorder). https://www.instructables.com/Make-Your-Own-Spy-Bug-Arduino-Voice-Recorder/

7. Anshulpareek, Neetu.Auto voice record and playback. Published on June 9 2018. Retrived fromhttps://create.arduino.cc/projecthub/gadgetprogrammers/diy-auto-voice-record-and-playback-7a47d7

8. Sourav Gupta. Simple Arduino Voice Recorder for Spy Bug Voice Recording. Feb 02 2021. https://circuitdigest.com/microcontroller-projects/simple-arduino-voice-recorder-for-spy-bug-voice-recording

9. 16x2 Alphanumeric LCD Display Datasheet, https://components101.com/displays/16x2-lcd-pinout-datasheet

10. 4x4 Keypad Datasheet, https://components101.com/misc/4x4-keypad-module-pinout-configuration-features-datasheet

11. MicroSD Card Module Datasheet, https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwic5dXstbXyAhXIZCsKHTZRAqEQFnoECAIQAQ&url=https%3A%2F%2Fcurtocircuito.com.br%2Fdatasheet%2Fmodulo%2Fcartao_micro_SD.pdf&usg=AOvVaw0AXDdGO9AxGl8zzkNqNkuO

12. Arduino Timer Interrupts By Amanda Ghassaei https://www.google.com/amp/s/www.instructables.com/Arduino-Timer-Interrupts/%3famp_page=true

7. APPENDIX

Appendix I

```
LOWPASS FILTER EFFECT
AudioFile = 'sound.wav';
[st,fs] = audioread(AudioFile);
N = fs*10;
t = (1:fs*10)/fs;
Xt = st(1:N,1);
y=zeros(N+1,1);
for i=1:1:N

y(i+1)=0.8669*y(i)+0.1331*Xt(i);
    %y(n+1)=0.5335(n)+0.4665(n)
end

Xf = abs(fft(Xt));
f = 1/16:1/16:fs;
figure;
plot(f(1:8*fs),Xf(1:8*fs));
title('original sound');
xlim([0 5000])
ylim([0 2000])

Yf = abs(fft(y));
f = 1/16:1/16:fs;
figure;
plot(f(1:8*fs),Yf(1:8*fs));
title('filtered sound');
xlim([0 5000])
ylim([0 2000])
```

Figure (a)

```
IMPLEMENTATION OF TRANSFER FUNCTION

>> FC=1000;

>> WC=2*pi*FC;

>> num=WC;

>> den=[1,num];

>> H=tf(num,den)

H =

   6283

  --------

  s + 6283

Continuous-time transfer function.

>> Hd=c2d(H,1/44100)

Hd =

   0.1328

  ----------

  z - 0.8672

Sample time: 2.2676e-05 seconds

Discrete-time transfer function.
```

Figure (b)

```
ECHO EFFECT
AudioFile='sound.wav';
[st,fs] = audioread(AudioFile);
N = fs*10;
t = (1:fs*10)/fs;
Xt = st(1:N,1);
a=0.8;
d=2000;
y=zeros(N+d,1);
Xn=padarray(Xt,d,0,'pre');
for i=(d+1):1:N
    y(i-d)=Xt(i)+a*Xn(i-d);
end
```

Figure (c)

Appendix II

Calculation for the value of Timer Counter 2 register

$$Timer\ speed = \frac{Crystal\ speed}{prescaler} = \frac{16\ MHz}{8} = 2\ MHz$$

$$Duration\ of\ one\ timer\ count = \frac{1}{2 \times 10^6}\ s$$

$$Required\ Interrupt\ speed = \frac{1}{Required\ sampling\ rate} = \frac{1}{8000}\ s$$

$$if\ Number\ of\ timer\ counts\ required = x$$

$$then \quad \frac{1}{2 \times 10^6} \times x = \frac{1}{8000}$$

$$we\ get \quad x = 250$$

Therefore we must set the TCNT2 register to 5 or 0x05 in order for the interupt to occur after 250 timer counts

Appendix III

ARDUINO CODE

```
#include <Wire.h>

#include <Keypad.h>

#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x20, 16, 2);

#include <SPI.h>

#include <SD.h>

bool flag = false;

const byte rows = 2;

const byte cols = 3;

char keys[rows][cols] = {

  {'1', '2', '3'},

  {'4', '5', '6'}

};

byte rowPins[rows] = {8, 9};

byte colPins[cols] = {A1, A2, A3};


Keypad gajo = Keypad(makeKeymap(keys), rowPins, colPins, 2, 3);


const byte dacPin = 9;

long t, t0;

uint8_t sample;

int no_of_samples = 0;

String line;

float fs;

int CSPin = 10;

int welcome = 0;
int dt = 1000;
int playstate = 0;
int clips = 1;
char chosenKey;
char playKey;
File recFile;
char c[4][10] = {"rec1.csv", "rec2.csv", "rec3.csv"};
int i = 0;

File lfilter;
float y_pre=0;
int linep=0;
void setup()
{
  lcd.begin(16, 2);
  if (!SD.begin(CSPin)) {
    while (1);
  }

  while (welcome == 0) {
    lcd.setCursor(0, 0);
    lcd.print("Welcome");
    delay(dt);
    welcome = 1;
  }
  File root = SD.open("/");
  deleteFiles(root);
  root.close();
  DDRD = 0B11111111;
}
```

```
ISR(TIMER2_OVF_vect) {
  flag = true;
  //TCNT2 = 0x05;
}

void loop() {
  playstate = 0;
  lcd.noBlink();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Which option?");
  delay(dt);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("4-Play 5-effect");
  lcd.setCursor(0, 1);
  lcd.print("6-Record");
  do {
    chosenKey = gajo.getKey();
    delay(100);
  } while (chosenKey != '6' && chosenKey != '4' && chosenKey != '5');
  if (chosenKey == '6') {
    maincode_record();
  } else if (chosenKey == '4') {
    maincode_playfile(0);
  }
  else if (chosenKey == '5') {
    maincode_playfile(1);
  }
  playstate = 0;
  delay(100);
}

void maincode_record() {
  if (clips == i + 1 && clips != 4) {
    record();
  }
  else {
    lcd.clear();
    lcd.print("Delete first");
    delay(1000);
    lcd.clear();
  }
}
void record() {
  recFile = SD.open(c[i], FILE_WRITE);
  if (recFile) {
  } else {
    while (1) {
    }
  }
  delay(100);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Recording");
  lcd.setCursor(9, 0);
  lcd.blink();
  delay(200);
  ADC_init();
  while (1) {
    ADC_getSample(); //sample = ADCH;
    no_of_samples++;
/*     if (gajo.getKey() == '6'){
        break;
      }
*/   if (no_of_samples == 3000) {
      break;
    }
  }
  recFile.close();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("COMPLETE");
  delay(dt);
  no_of_samples = 0;
  i += 1;
  clips += 1;
}

void maincode_playfile(int m) {
  lcd.clear();
  lcd.setCursor(0, 1);
  lcd.print("Which clip?");
  while (1) {
    playKey = gajo.getKey();
    if (playKey == '1' || playKey == '2' || playKey == '3') {
      delay(100);
      lcd.clear();
      break;
    }
  }
  if (playKey == '1') {
    if (m==0){
    playfile(0);
    }else{
    play_effects(0);
    }
  }
  else if (playKey == '2') {
    if (m==0){
    playfile(1);
    }else{
    play_effects(1);
    }
  }
  else if (playKey == '3') {
    if (m==0){
    playfile(2);
    }else{
    play_effects(2);
    }
  }
}

int playfile(int d) {
  while (playstate == 0) {
    uint8_t line;
    recFile = SD.open(c[d], FILE_READ);
```

```
    if (recFile) {
      lcd.noBlink();
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Playing");
      lcd.setCursor(7, 0);
      lcd.blink();
      PORTD = B00000000;
      while (recFile.available()) {
        line = recFile.read();
        while (1) {
          if (flag) {
            PORTD = line;
            break;
          }
        }
        playstate = 1;
      }
    } else {
      //Serial.println("error opening REC.txt");
      lcd.clear();
      lcd.setCursor(0, 1);
      lcd.print("File n/a");
      delay(2000);
      lcd.clear();
      playstate = 1;
    }
    recFile.close();
  }
}
```

```
int play_effects(int R){
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Processing");
  lowfilter_main(R);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Play");
  lfilter = SD.open("lowpass.csv",FILE_READ);
   PORTD = B00000000;
   while (lfilter.available()) {
       line = lfilter.read();
       while (1) {
         if (flag) {
           PORTD = line.toInt();
           break;
         }
       }
     }
 lfilter.close();
 SD.remove("lowpass.csv");
}

int lowfilter_main(int l){
  recFile = SD.open(c[l], FILE_READ);
  lfilter = SD.open("lowpass.csv",FILE_WRITE);
  while(1){
          line = recFile.read();
          int line_=line.toInt();
         //int y_current=(0.5335*y_pre)+(0.4665*line_);//2000
         //int y_current=(0.7304*y_pre)+(0.2696*line_);//200-250
         int y_current=(0.8546*y_pre)+(0.1454*linep);//200-125
         lfilter.write((y_current));
         y_pre=y_current;
         linep=line_;
         if (!recFile.available()){break;}
       }
  lfilter.close();
  recFile.close();
}

void deleteFiles(File r) {
 while (true) {
   File dir = r.openNextFile();
   if (!dir) {
     dir.close();
     clips = 1;
     lcd.clear();

     break;
   }
   SD.remove(dir.name());
   dir.close();
 }
}

void ADC_init() {
 sei();
 TCCR2B = 0b000000001; // Timer Frequency = 16 MHz / 8 = 2 MHz
 TIMSK2 |= (1 << TOIE2); //Enable the timer overflow interrupt
 Serial.println("Timer interrupts configured");
 ADCSRA |= (1 << ADPS2) |  (1 << ADPS1) |  (1 << ADPS0);
 ADMUX |= 0x00;
 ADMUX |= (1 << REFS0);
 ADMUX |= (1 << ADLAR);
 ADCSRA |= (1 << ADEN);
 Serial.println("ADC Initialized");
}

void ADC_getSample() {
 ADCSRA |= (1 << ADSC);
 //Serial.println("ABC");
 // Starting conversion
 recFile.write(sample);
 while (1) {
   if (flag) {
     // writing previous reading to file while converting
     sample = ADCH;
     break;
   }
 }
 //read upper 8bits
}

void ADC_stop() {
 //ADCSRA |= (1<<ADEN);
 //Serial.println("DEF");
 //TIMSK2 |= (0 << TOIE2);
 cli();
```

```
}
```