Department of Electronic and Telecommunication Engineering
University of Moratuwa

Simple Voice Recorder

Priyashan, B.W.S.	190476V
Punsara, K.K.S.	190481G
Rajapaksha, R.M.P.A.P.	190484T
Ranasinghe, K.K.H.	190494A

Supervisor: Mr. Sahan Liyanaarachchi

This report is submitted as partial fulfilment of module EN1093

August 2021

Abstract

We were given the task to design and implement a simple voice recorder capable of recording, saving, selectively playing back multiple voice recordings, and adding adjustments to the spectrum during playback. Our project was successful in recording, saving, and playing back up to three recordings. Furthermore, we were successful in adding three adjustments; down sampling, enhancing, and shifting to the spectrum during playback. Recording and playback of our voice recorder are based on the TMRpcm library which samples at 16 kHz and encodes 8 bits per sample. Voice recordings are saved on an SD card. The final breadboard design, based on the ATmega328-PU single-chip microcontroller consists of four push buttons that interact with the user interface through an LCD. Our project is complete with a validated PCB and Enclosure design.

Table of Contents

1. Introduction.

The main goal of this project was to design and implement a simple voice recorder that is capable to sample, encode and store the bit stream of the given sound source.

The Voice recorder that was developed by us is powered by an Atmega32 microcontroller, and it is capable of doing three main tasks. There are,

1. **Record and save multiple voice recordings**
    - The Voice recorder is able to record up to 3 recordings (16 kHz sampling rate and 8-bit PCM resolution) and save them on an SD card.
2. **Selectively playback the recordings**
    - Users can select recordings that are saved in SD card through the menu and playback them
3. **Do minor adjustments to its spectrum during playback**
    - Users can add three adjustments to recordings. There are,
        - Frequency shift
        - Frequency scaling
        - Selective enhancement of the spectrum.

In the designing stage of this project, we used many software tools such as Microchip studio for coding, Matlab for calculations, Proteus for simulations, Altium for PCB design, and Solidworks for enclosure design.

The following is the complete report of the challenges that we faced during the project and how we solved them.

2. Methodology

2.1 Audio Recording / Analog to Digital Conversion

When we capture a sound wave using a microphone, it creates an analog voltage signal proportional to the pressure variation of that sound. But, for storing, transferring, and modifying purposes, it would be more convenient if this analog information can be converted into a set of digital data.

The ATmega328P microcontroller includes an inbuilt ADC module that makes it appliable in designing a voice recorder. This module has a resolution of up to *10 bits per sample* and has a conversion time from *65 to 260 µs*. This module is connected to 8 multiplexed single-ended input channels coming from the pins AD0 to AD7 of port A.

2.1.1 Pulse Code Modulation (PCM)

Pulse Code Modulation (PCM) is a commonly used technique to convert analog information into digitized data. There are 3 main steps in this conversion technique.

1. **Sampling**

In this step, the analog signal is sampled at constant intervals in time known as the *Sampling Intervals*. The corresponding Sampling Frequency is determined by the *Nyquist Sampling Theorem*, which states that *a bandlimited continuous-time signal can be sampled and perfectly reconstructed from its samples if the waveform is sampled over twice as fast as its highest frequency component*. According to the above theorem, when implementing a Voice Recorder, the frequency of the sampling process must be at least 8 kHz, since the highest frequency of a voice signal is considered to be 4 kHz.

2. **Quantizing**

After the sampling process, the signal is *quantized*, i.e. each sample is approximated to one of several selected discrete levels. Depending on the way of choosing these quantization levels, this step can be achieved in two different techniques, known as *Uniform* and *Non-Uniform Quantization*. In the Uniform Quantization, all the quantizing levels are equally spaced in amplitude, in contrast to the Non-Uniform Quantization where the quantizing levels are specifically chosen to capture more common lower amplitude signals in a higher resolution. However, since the ADC module in the ATmega328P provides 1024 ($= 2^{10}$) uniform quantizing levels, in this project, we have used a quantization method consisting of 256 ($= 2^{8}$)

equally spaced quantizing levels. This much of a resolution is practically enough to capture common electrical signal variations in an amplified audio signal.

3. Encoding

In the Encoding process, each sample is represented by a digital code unique to the corresponding quantization level of the sample. Since a number of 2^8 levels have been used for quantization in this case, each sample would be replaced by an 8-bit binary code; a byte.

2.1.2 TMRpcm Library

In this Voice Recorder Design, both the *audio recording* and *audio playback* implementations are based on the TMRpcm library. TMRpcm is an Arduino library that enables *Asynchronous Playback of 8-bit, 8-32 kHz sampling rate, mono type WAV audio files* stored in an SD card. (The asynchronous playback specifies that the playing of an audio file does not block the execution of the other codes in the main loop.) This library can be used on many of the Arduino development boards such as *Uno*, *Nano*, *Mega*, and even on the corresponding Atmel microcontrollers including ATmega328P.

This library provides a broad range of functions regarding audio recording and playback and some of those main functions are listed down below.

- ***TMRpcm.play(filename);*** plays a WAV file
- ***TMRpcm.stopPlayback();*** stops the music, but leaves the timer running
- ***TMRpcm.isPlaying();*** returns 1 if music is playing, 0 if not
- ***TMRpcm.pause();*** pauses/unpauses playback
- ***TMRpcm.volume(0);*** 1 (up) or 0 (down) to control volume
- ***TMRpcm.setVolume(0);*** set volume level from 0 to 7
- ***TMRpcm.startRecording(filename, sample rate, pin);*** start recording audio
- ***TMRpcm.stopRecording(filename);*** stop recording audio

Even in the TMRpcm library, the same Pulse Code Modulation (PCM) technique is used to digitally represent the audio data. In order to understand how this PCM method is implemented in TMRpcm, we have to take a look at how it uses the inbuilt ADC module and the corresponding registers.

2.1.3 The Implementation of Analog to Digital Conversion in TMRpcm Library

Since the **PRADC** (*Power Reduction ADC* bit) of the **PRR** (*Power Reduction Register*) has an initial value of logical 0 (i.e. power reduction is *disabled*), by default, the ADC module is automatically powered up when the clock signal is supplied to the microcontroller. (refer to Figure 1)

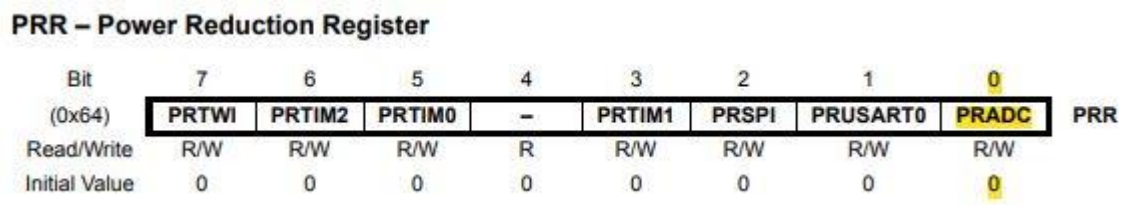


Figure 1 - PRR

Prior to performing any conversion, the ADC module must be enabled by setting the **ADEN** (*ADC Enable* bit) of **ADCSRA** (*ADC Control and Status Register - A*) to a logical one. This step is implemented in the TMRpcm library by the following line of code.

```
ADCSRA |= _BV(ADEN) | _BV(ADATE); //ADC Enable, Auto-trigger enable
```

Figure 2 - ADCSRA

Then the *sampling frequency* must be set for the digital conversion. The input clock frequency for the ADC module is determined by dividing the system clock frequency (16 MHz) by the *prescaling factor* given by the bit combination of the *ADC Prescaler Select* bits (**ADPS2:0**) in the **ADCSRA** register. (refer to Figure 2) And when calculating the sampling frequency, note that a normal conversion takes 13 ADC clock cycles. (refer to Table 1)

Table 1 - the sampling frequency calculation

ADPS2	**ADPS1**	**ADPS0**	**Prescaling Factor**	**ADC Clock Frequency** *(16 MHz / prescaling factor)*	**Sampling Frequency** *(ADC Clock frequency / 13 clock cycles)*
0	0	0	2	8 MHz *	
0	0	1	2	8 MHz *	
0	1	0	4	4 MHz *	
0	1	1	8	2 MHz *	

1	0	0	16	1 MHz	76.923 kHz
1	0	1	32	500 kHz	38.461 kHz
1	1	0	64	250 kHz	19.230 kHz
1	1	1	128	125 kHz #	

* not achievable due to the performance limitation of the ADC module.

not achievable in the TMRpcm library

When initializing a recording using the TMRpcm library, the sampling frequency must be given as an input to the *startRecording* function. Then the library will automatically set the prescaler bits to obtain the required sampling frequency. The corresponding coding from the library is shown below.

```
byte prescaleByte = 0;

if (SAMPLE_RATE < 18000) {prescaleByte = B00000110;} //ADC division factor
64 (16MHz / 64 / 13clock cycles = 19230 Hz Max Sample Rate)
else if (SAMPLE_RATE < 27000) {prescaleByte = B00000101;} //32 (38461Hz
Max)
else {prescaleByte = B00000100;} //16 (76923Hz Max)

ADCSRA = prescaleByte; //Adjust sampling rate of ADC depending on sample
rate
```

Due to the performance limitation, the maximum possible sampling frequency is limited to 76 kHz.

By setting up the **REFS1** and **REFS0** (*Reference Selection* bits) in the **ADMUX** (*ADC Multiplexer Selection Register*), the voltage reference for the ADC can be selected. (refer to Figure 3 and Table 2)

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 3 - ADMUX

Table 2 - Reference Selection bit configuration

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, internal VREF turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V voltage reference with external capacitor at AREF pin

The TMRpcm library is set for the second voltage reference shown above, where the voltage reference is taken through the AVCC pin by connecting it directly to the VCC and bypassing it to the ground through a capacitor; the same configuration in an Arduino Uno board. Rifer the following code from the TMRpcm library and Figure 3.

```
ADMUX |= _BV(REFS0) | _BV(ADLAR); //Analog 5v reference, left-shift result so only high byte needs to be read
```

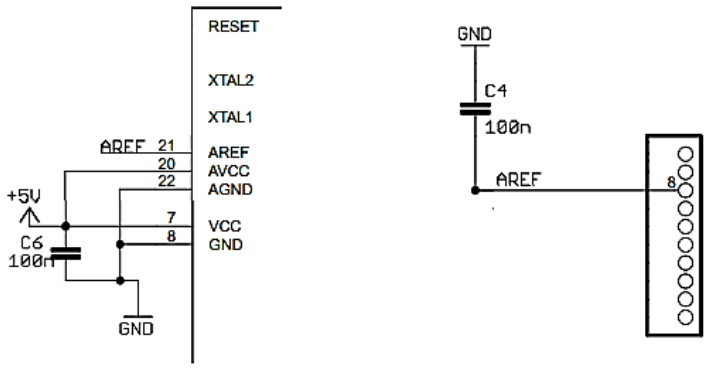


Figure 4 - Voltage Reference in an Arduino Uno

Then the analog input channel can be selected by changing the bits in the *Analog Channel Selection* bits; **MUX3:0** in the **ADMUX** register. Refer to Figure 3 and Table 3.

Table 3 - Analog Channel Selection bit configuration

MUX3	MUX2	MUX1	MUX0	Single-Ended Input
0	0	0	0	ADC0
0	0	0	1	ADC1
0	0	1	0	ADC2
0	0	1	1	ADC3
0	1	0	0	ADC4
0	1	0	1	ADC5
0	1	1	0	ADC6
0	1	1	1	ADC7

Same as with the sampling frequency, the pin that is used to give the analog input must be given as an input to the *startRecording* function by using the corresponding Arduino analog pin notation (from A0 to A7). This parameter ("*pin*") will be then taken as a byte and the corresponding input pin will be connected to the ADC.

```
#if defined(ADMUX)
       ADMUX = (pin & 0x07); // same as (pin & 0b00000111)
#endif
```

After setting the above bit configurations, the *sampling process* can be begun. To start an analog to digital conversion *manually*, the **ADSC** (*ADC Start Conversion* bit) in the **ADCSRA** register must be written to a logical one. (refer to Figure 2) During the conversion period, this bit will remain as a one, and when the conversion is completed, the value will return to zero. Thus, to implement a recurring sampling process using this method, the ADSC bit has to be written up again and again manually.

As a more convenient alternative, the *Auto Triggered Mode* can be used to achieve a repeating sample-taking process. In this method, the conversion will be continuously repeated periodically based on a trigger source selected, i.e. a conversion will be automatically started on each positive edge of the trigger signal. To enable this auto triggered mode, a logical one must be written to the *ADC Auto Trigger Enable* bit (**ADATE**) in **ADCSRA** (refer to Figure 2), and for selecting the preferred trigger source, the bit selection of the *ADC Auto Trigger Source* bits (**ADTS2:0**) in the *ADC Control and Status Register - B* (**ADCSRB**) must be set.

ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 5 - ADCSRB

In the TMRpcm library, a previously set timer that is created using the given sampling rate is used as the auto trigger source. Refer to the following lines of code to see the selected bit configuration in the ADTS bits used in the library.

```
ADCSRA |= _BV(ADEN) | _BV(ADATE); //ADC Enable, Auto-trigger enable

ADCSRB |= _BV(ADTS0) | _BV(ADTS2); //Attach ADC start to TIMER1 Compare Match B
flag
```

After the conversion is completed, the encoded value is stored in the *ADC Data Registers* which consist of two 8-bit registers; **ADCL** and **ADCH**. When the result is *left-adjusted* by writing the **ADLAR** (*ADC Left Adjust Result* bit) in the **ADMUX** register (refer to Figure 3), the 8 most significant bits of the 10-bit value will be written to the ADCH register left-adjusted, and the remaining 2 bits will be written on the ADCL. (refer to Figure 6)

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Figure 6 - ADLAR and ADCH

When only an 8-bit precision is required (as the case in the TMRpcm library), only reading the ADCH register is enough. And then, this read encoded sample value will be transferred into the SD card module connected to the microcontroller to be stored in the corresponding audio recording WAV file.

2.2 User Interface

We have four push buttons to control the display. Those four buttons are up button, down button, select button and back button.

In our user interface we have 4 main parts. Those are,

- Main menu
- Recording menu
- Play list
- Playing menu

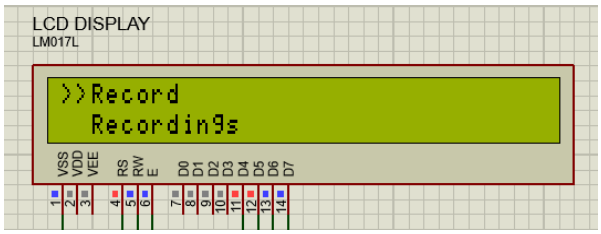


Figure 8 - Main Menu

This is the menu that we will see when our voice recorder is power on.

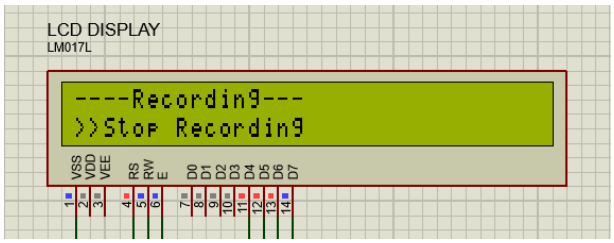
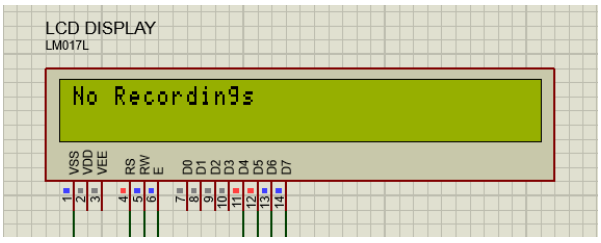


Figure 7 - Recording Menu

If we select record action in the main menu, we will go to recording menu and if we select stop recording, we will go back to main menu and our recording will be saved in SD card.



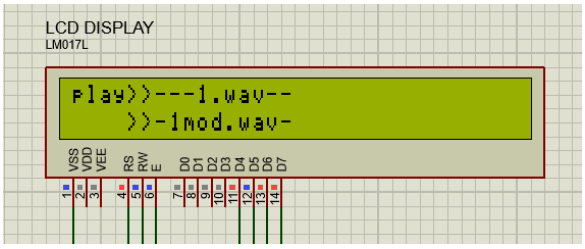


Figure 9 - Play List

If we go to the recordings without recording any voice, we will be indicated that there are no recordings.

If we go to recordings after record, we will see the newly recorded file in the play list. Also, we can go up and down and select previously recorded recordings too.

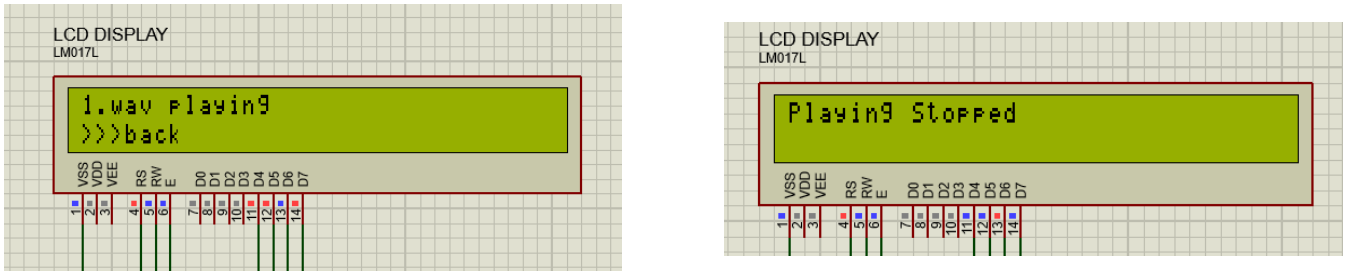


Figure 10 - Playing Menu

If we select a recording to play we can play it and if we select the given back option voice recorder will stop playing the recording.

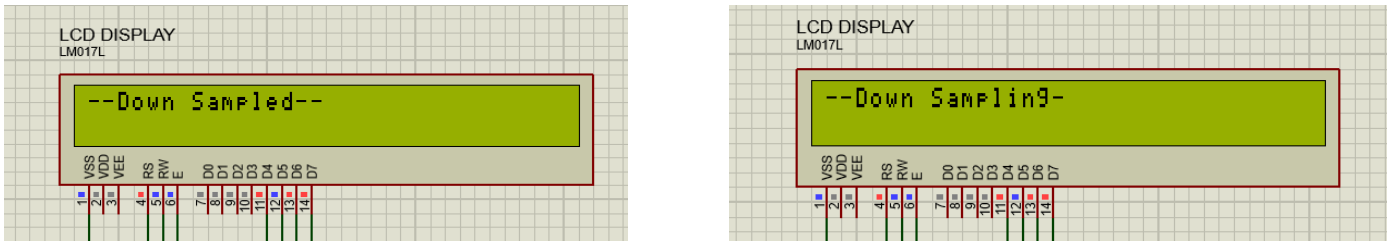


Figure 11

If we play the modified recording first voice recorder will process the audio file and create a new modified audio file with selected effect.

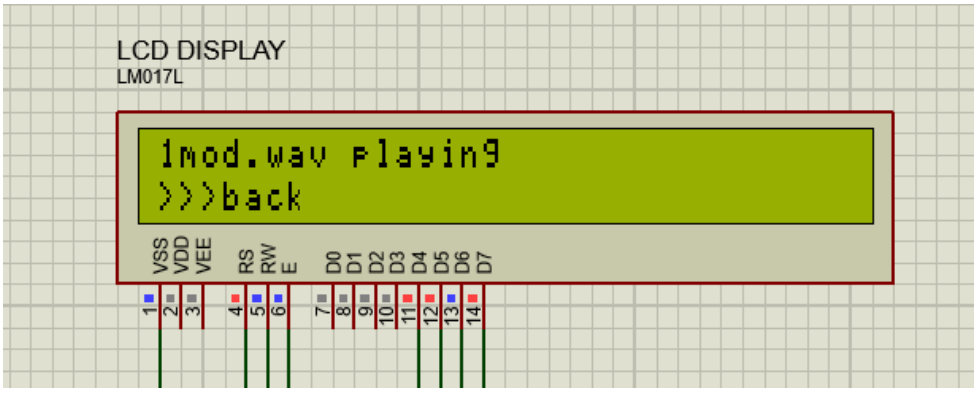


Figure 12

After creating the new audio file with the effect, we can play the recording with added effect.

2.3 Prototype Design

The main requirement of the project was to successfully record and save multiple recordings. And then, selectively playback the recordings.

2.3.1 Breadboard design using Arduino microcontroller

Initial design of the project was done on a breadboard using the Arduino micro-controller to implement and validate our ideas. We were successful in recording and saving up to 3 recordings. Due to the instability of the code when program memory usage is maximum, we had to limit the maximum number of recordings to 3.

Push buttons were successfully integrated with the LCD display for a smooth interaction with the user interface. Voice recorded using the microphone module was sampled at 16 kHz and encoded at 8 bits per sample using the TMRpcm library in Arduino. Recordings were saved in .wav format. We were successful in selectively playing the 3 recordings using the speaker. Initial design on the breadboard using the Arduino micro-controller proved successful. Next, we shifted our prototype on the breadboard using an ATMEGA 328-PU single-chip microcontroller.

2.3.2 Breadboard design using ATMEGA 328-PU

When shifting our initial design to the ATMEGA 328-PU chip microcontroller, several codes were converted to AVR. As we implemented our main idea around the TMRpcm library, most of the code was converted to AVR from the Arduino sketch. Since the sound source was sampled at 16 kHz, a clock speed of 16 MHz is required. To fulfil this requirement an external 16 MHz crystal oscillator was used with the chip.

When using a 16 MHz crystal oscillator with ATMEGA 328-PU we had to first change the “Fuse Low Bytes”. There are a total of 19 fuse bytes used in the ATMEGA 328-PU, they can be categorized under “Extended Fuse Bye”, “Fuse High Byte” and “Fuse Low Byte”. Setting up an external crystal oscillator does not require changing of Extended and Fuse High Bytes.

When selecting the clock source for the ATMEGA we had to consider the following process according to Table 4 given below. First of all, it should be noted that, a fuse byte is unprogrammed if it is set to 1.

Table 4 - Fuse Low Byte

Low Fuse Byte	Bit No.	Description	Default Value
CKDIV8[4]	7	Divide clock by 8	0 (programmed)
CKOUT[3]	6	Clock output	1 (unprogrammed)
SUT1	5	Select start-up time	1 (unprogrammed)[1]
SUT0	4	Select start-up time	0 (programmed)[1]
CKSEL3	3	Select Clock source	0 (programmed)[2]
CKSEL2	2	Select Clock source	0 (programmed)[2]
CKSEL1	1	Select Clock source	1 (unprogrammed)[2]
CKSEL0	0	Select Clock source	0 (programmed)[2]

CKDIV8, controls whether the clock rate should be divided by 8 or not. Since, we use an external 16 MHz crystal oscillator, bit 7 was unprogrammed (1).

CKOUT, sets whether clock output is present in PORTB0 of the microcontroller. There was no need for this setting in our project, therefore bit 6 was unprogrammed (1).

SUT1 and SUT0 sets the startup time of the micro controller, which prevents the chip from attempting to start before the supply voltage has had time to reach an acceptable minimum level. Default setting of 10 is for a startup delay of six clock cycles from power-down and power-save and an addition startup delay of 14 clock cycles plus 65ms for RESET. According to Table 5, on the safe side for a low power crystal oscillator, a maximum delay of 16,000 clock cycles from power-down and power-save is needed. Therefore, both SUT1 and SUT0, i.e. bits 5 and 4 is unprogrammed (1). In addition, CKSEL0, i.e. bit 0 is also unprogrammed (1).

Table 5 - Start up times for the Low Power crystal oscillator clock selection

Oscillator Source / Power Conditions	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	CKSEL0	SUT[1:0]
Ceramic resonator, fast rising power	258 CK	14CK + 4.1ms(1)	0	00
Ceramic resonator, slowly rising power	258 CK	14CK + 65ms(1)	0	01
Ceramic resonator, BOD enabled	1K CK	14CK(2)	0	10
Ceramic resonator, fast rising power	1K CK	14CK + 4.1ms(2)	0	11
Ceramic resonator, slowly rising power	1K CK	14CK + 65ms(2)	1	00
Crystal Oscillator, BOD enabled	16K CK	14CK	1	01
Crystal Oscillator, fast rising power	16K CK	14CK + 4.1ms	1	10
Crystal Oscillator, slowly rising power	16K CK	14CK + 65ms	1	11

Bits 3 through 0 are used to select the clock source. According to Table 6, we need a low power crystal oscillator, which means bits 3 to 0 can takes values in the range 1111 - 1000. According to Table 7, it is stated that for a frequency range of 8 - 16 MHz CKSEL[3:1] must be set to 111. Range of capacitors that should be connected to the oscillator is recommended to be between 12 and 22pF. We have used two 22 pF capacitors in our design.

Table 6 - Device Clocking Options Select

Device Clocking Option	CKSEL[3:0]
Low Power Crystal Oscillator	1111 - 1000
Full Swing Crystal Oscillator	0111 - 0110
Low Frequency Crystal Oscillator	0101 - 0100
Internal 128kHz RC Oscillator	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved	0001

Table 7 - Low power crystal oscillator operating modes

Frequency Range [MHz]	CKSEL[3:1][2]	Range for Capacitors C1 and C2 [pF]
0.4 - 0.9	100[3]	–
0.9 - 3.0	101	12 - 22
3.0 - 8.0	110	12 - 22
8.0 - 16.0	111	12 - 22

In conclusion,

Table 8 - Fuse Low Byte for 16 MHz crystal oscillator

CKDIV8	CKOUT	SUT1	SUT0	CKSEL3	CKSEL2	CKSEL1	CKSEL0
1	1	1	1	1	1	1	1

Therefore, the new Fuse Low Byte should be 11111111, which is 0xFF in hexadecimal notation.

After figuring out the correct Fuse Low Byte, fuse bytes were burnt to the microcontroller using avrdude and Arduino as ISP. AVRDUDE is command line driven utility required to program changes to the ATMEGA 328-PU or any other Atmel microcontroller.

First, the Arduino microcontroller was connected to the laptop and "Arduino as ISP" sketch was uploaded to the Arduino. At this stage, there should be no connections between external components and the Arduino.

Then connections must be made between the Arduino and the microcontroller as follows;

Arduino Pin 13 - PB5 / SCK

Arduino Pin 12 - PB4 / MISO

Arduino Pin 11 - PB3 / MOSI

Arduino Pin 10 - PC6 / RESET

After the following connections were set up, fuse low bytes were burnt to the microcontroller. This procedure is repeated to burn hex codes to the micro controller.

We were also successful in recording, saving and playing back up to 3 recordings using the ATMEGA 328-PU microcontroller. For the ease of shifting from Arduino prototype to the microcontroller prototype on the breadboard, the initial connections were kept the same.

Unfortunately, due to a fault in the microphone module, there was a significant noise in all the recordings. Because of the noise, the clarity of the recording was quite low. But, due to the noise the recording when played through the speaker lacked clarity. But, when the recordings were listened from the laptop, the voice recordings could be heard even with the noise.

PWM output from the micro-controller cannot be directly played through a speaker as PWM is a digital signal. Therefore, a RC filter design was used as a low-pass filter and driver for the speaker.

When calculating values for the resistor and capacitor, we wanted the cut-off frequency to be 4kHz to allow only voice to be output to the speaker. A 0.22μF capacitor required a resistor of 150Ω to apply the low pass filter at the required frequency. But, since a RC filter is a passive filter design, the total output voltage had a significant decrease. This led to a low volume output from the speaker.

Another requirement of the project was to apply adjustments to the spectrum during playback. Although, this was not implemented in the main code due to instability, we were successful in applying 3 effects to a recorded .wav file using the micro-controller on the breadboard. The 3 effects namely; Down sampling, Enhance and Shifting could be demonstrated individually in the prototype. The above effects were added to a saved recording in the SD card, and played using the laptop. Further information on effects applied is discussed.

2.4 Components

2.4.1. Microphone module

The microphone module used for the project was GY-MAX 4466. MAX 4466 with its in-built adjustable gain and sound amplification was ideal for our requirement. Unfortunately, the module used in the project had a significant noise added to the recordings.

2.4.2. Push buttons

Four 12mm tactile push buttons were used for UP, DOWN, SELECT and BACK functions.

2.4.3. LCD Display

1602A (16x2) LCD display was used in the project. A potentiometer was used to control the brightness of the display.

2.4.4. SD card module

Micro-SD Memory Card adapter was used to store and playback voice recordings. The module consists of 6 pins which include Chip Select (CS), MISO, MOSI, SCK, VCC and GND. A 7 GB SAMSUNG SD card was used with the module.

2.4.5. Speaker

8 Ohm 3W speaker was used to play the stored recordings.

2.4.6. Capacitors

One 0.22μF capacitors was used for the RC filter design. Two 22pF capacitors were used to connect the 16 MHz crystal oscillator to ground.

2.4.7. Crystal oscillator

16 MHz crystal oscillator was used with the micro-controller.

2.4.8. Micro-controller

ATMEGA 328PU micro controller was used for the project. This micro-controller was used as it was readily available and ease of burning hex files using Arduino as ISP.

2.4.9. Resistors

One 10 kΩ resistor was used to connect RESET pin of the micro-controller to 5V. One 150Ω resistor was used in the RC filter design.

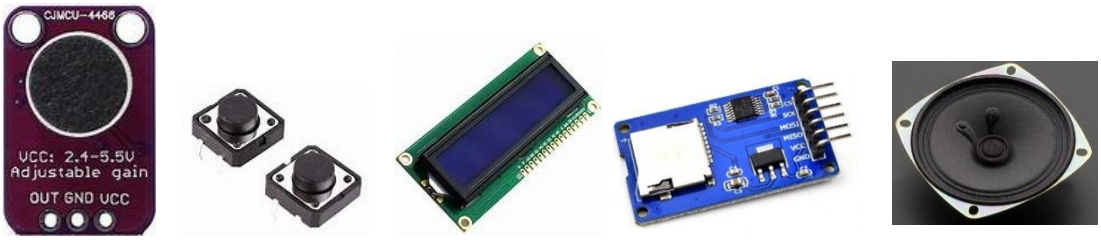


Figure 13 - (from left to right, in order) GY MAX 4466, 12 mm tactile push buttons, 1602A LCD display, Micro-SD Memory Card Adapter, 8 Ohm 3W Speaker

2.5 Audio Enhancement Techniques

Our voice recorder has the ability to add three effects to the recorded voices. Those are Frequency scaling, Selective Enhancing & Frequency shifting. To achieve these effects some adjustments must be done to the spectrum of recorded voice samples. However, the processing power of the Atmega32 microcontroller is not enough to add those changes directly to the spectrum. So, we build algorithms to change time-domain sample values according to some mathematical methods to achieve expected spectrum adjustments.

2.5.1 Frequency Scaling

The bandwidth of the recorded voice recorder is 4 kHz. The target of the frequency scaling is, scale all frequency components in the range 0 – 8khz. To achieve this, we have to downsample the voice file with the rate of 2. The spectrum of voice before and after downsampling with the ratio of 2 is shown in the figure.

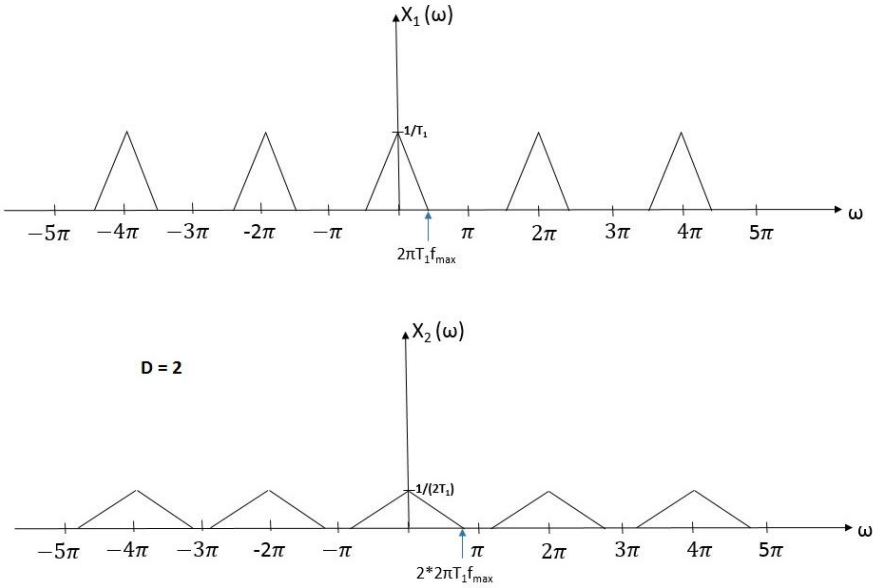


Figure 14 - spectrum before and after downsampling

According to the figure, *sampling frequency* >= 4 × *bandwidth* to avoid distortion in the spectrum.

Since the bandwidth of the voice recorder is 4 kHz, the Minimum sample rate of the voice recorder is 16 kHz.

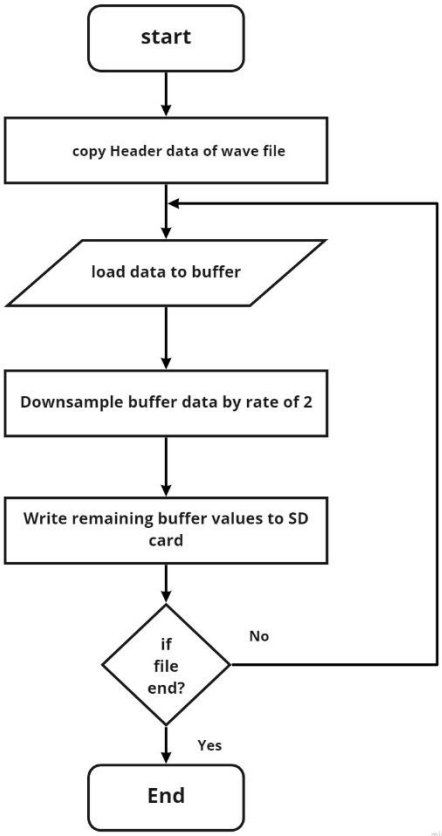


Figure 15 - simplified flowchart of frequency scaling algorithm

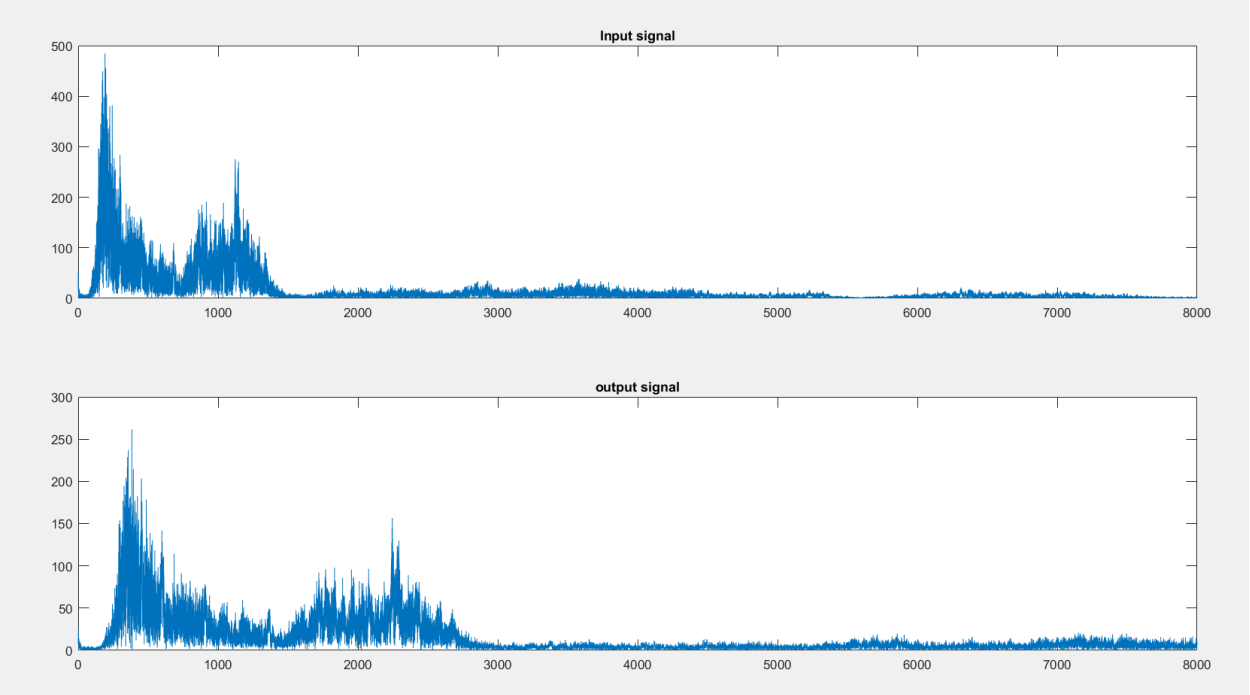


Figure 16 - frequency domain representation of sample voice file before and after frequency scaling

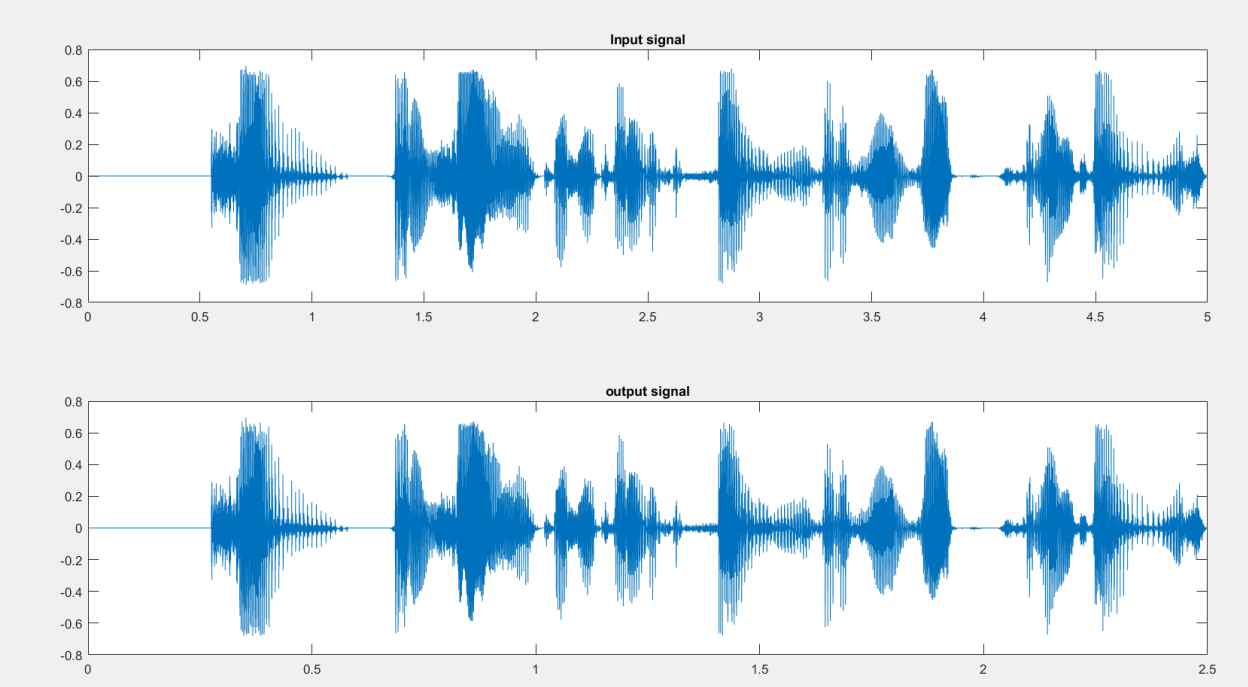


Figure 17 - time-domain representation of sample voice files before and after frequency scaling

2.5.2 Selective Enhancing

In selective enhancing, we divide the spectrum into two parts as low frequency range (0 Hz < f < 2500 Hz) and High frequency range (2500 Hz < f < 4000 Hz). Then we enhance the high-frequency range by increasing its amplitude by a factor of 8 and reduce the amplitude of the low-frequency range by a factor of 1/8. However, since we are doing all calculations in the time domain we have to use the convolution property for Fourier transform to enhance the spectrum.

$$h(t) * x(t) \xleftrightarrow{\mathcal{F}} H(\omega)\, X(\omega)$$

Equation 1 - convolution property

We can get filter coefficients of h(t) which have a gain of -9.03 dB for low-frequency range and 9.03 dB gain for high-frequency range using Matlab.

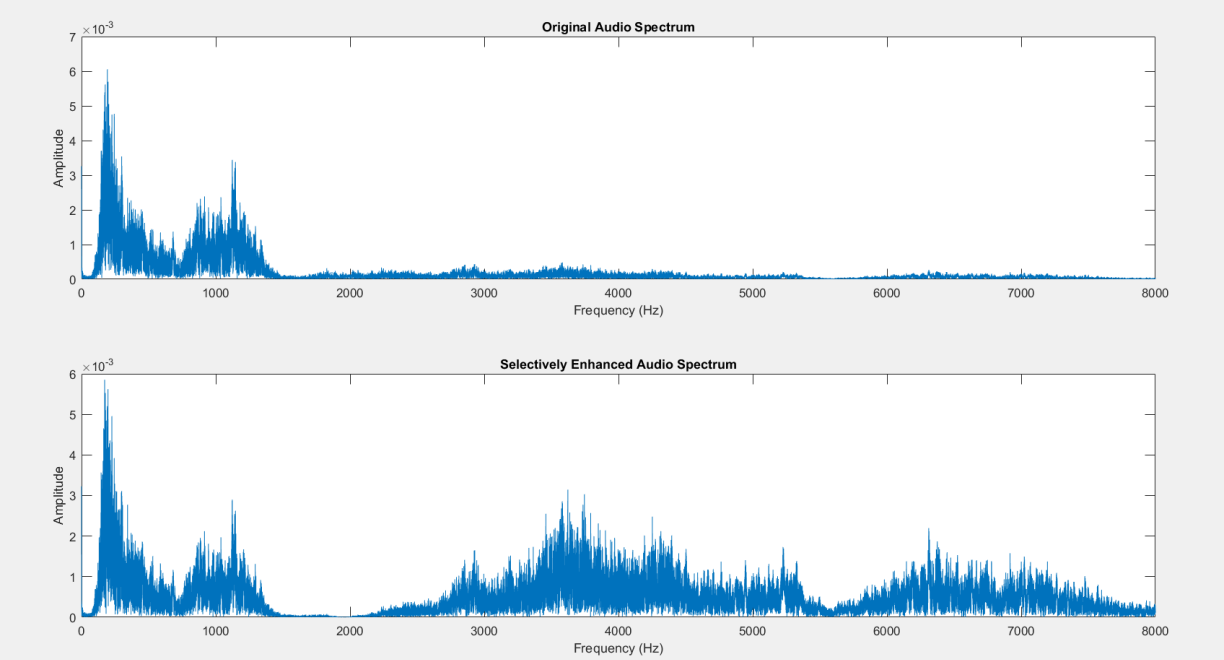


Figure 18 - the spectrum of sample voice file before and after selective enhancing

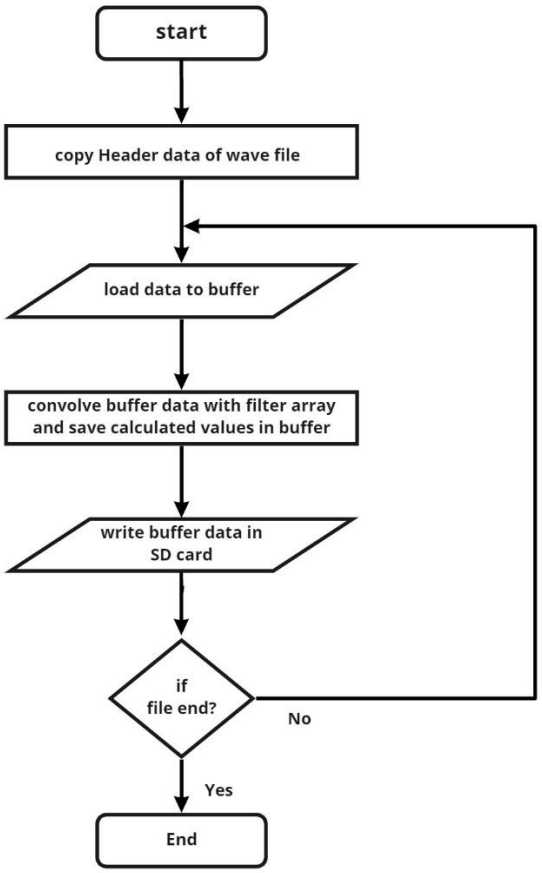


Figure 19 - simplified flowchart of frequency enhancing algorithm

2.5.3 Frequency shifting

In the frequency shifting effect, we shift all frequency components in the spectrum to higher frequencies by 2000 Hz. To do this process we used both convolution and multiplication properties of Fourier transform.

$$s(t)\,p(t) \overset{\mathcal{F}}{\longleftrightarrow} \frac{1}{2\pi}[S(\omega) * P(\omega)]$$

Equation 2 - multiplication property

$x(t)$ = voice record

$y(t) = cos(2\pi f t)$

$h(t)$ = high pass filter (cutoff frequency - f)

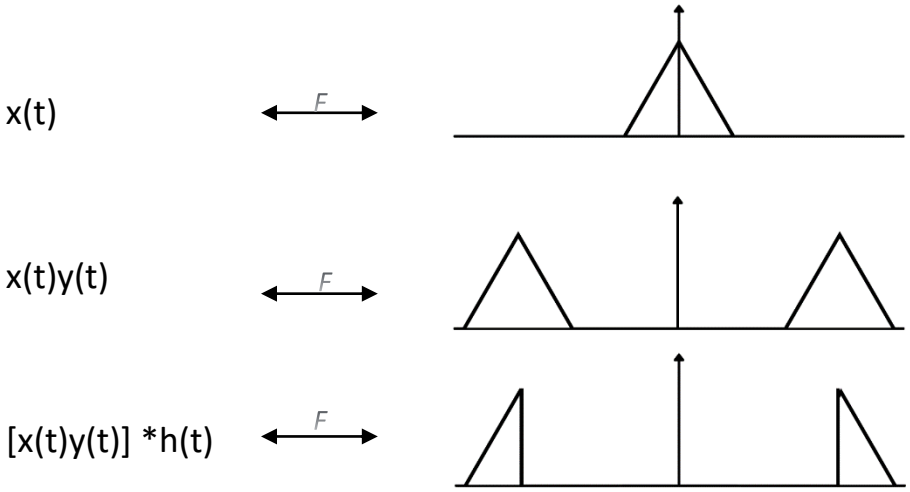


Figure 20 - spectrum

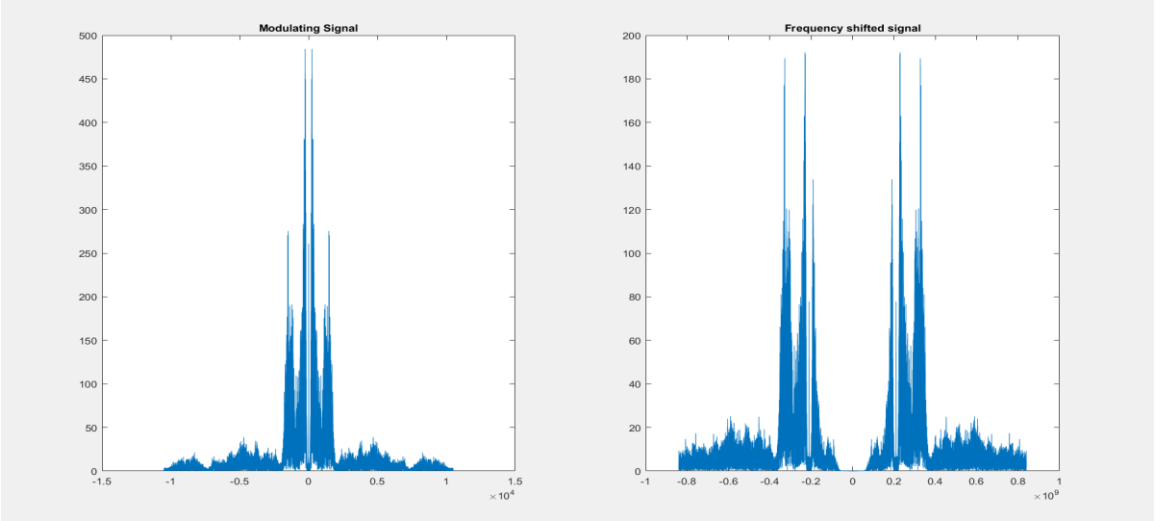


Figure 21 - original spectrum and shifted spectrum of sample voice record

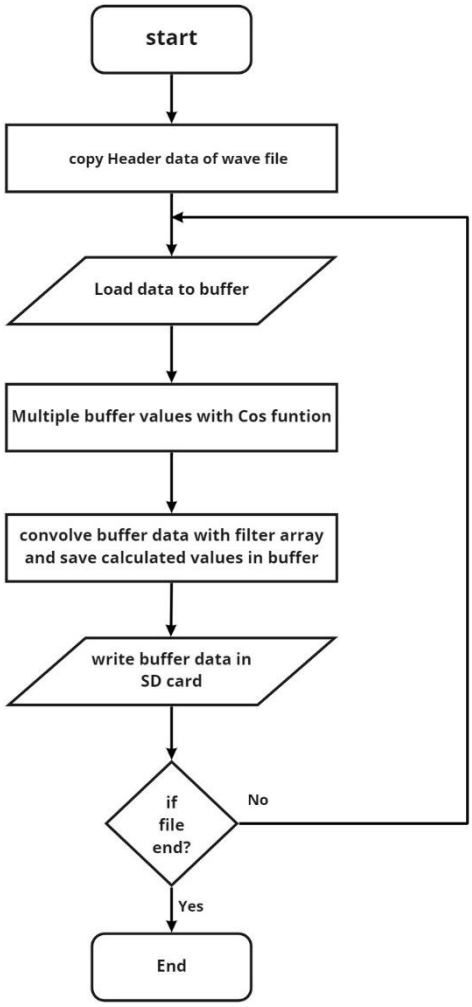
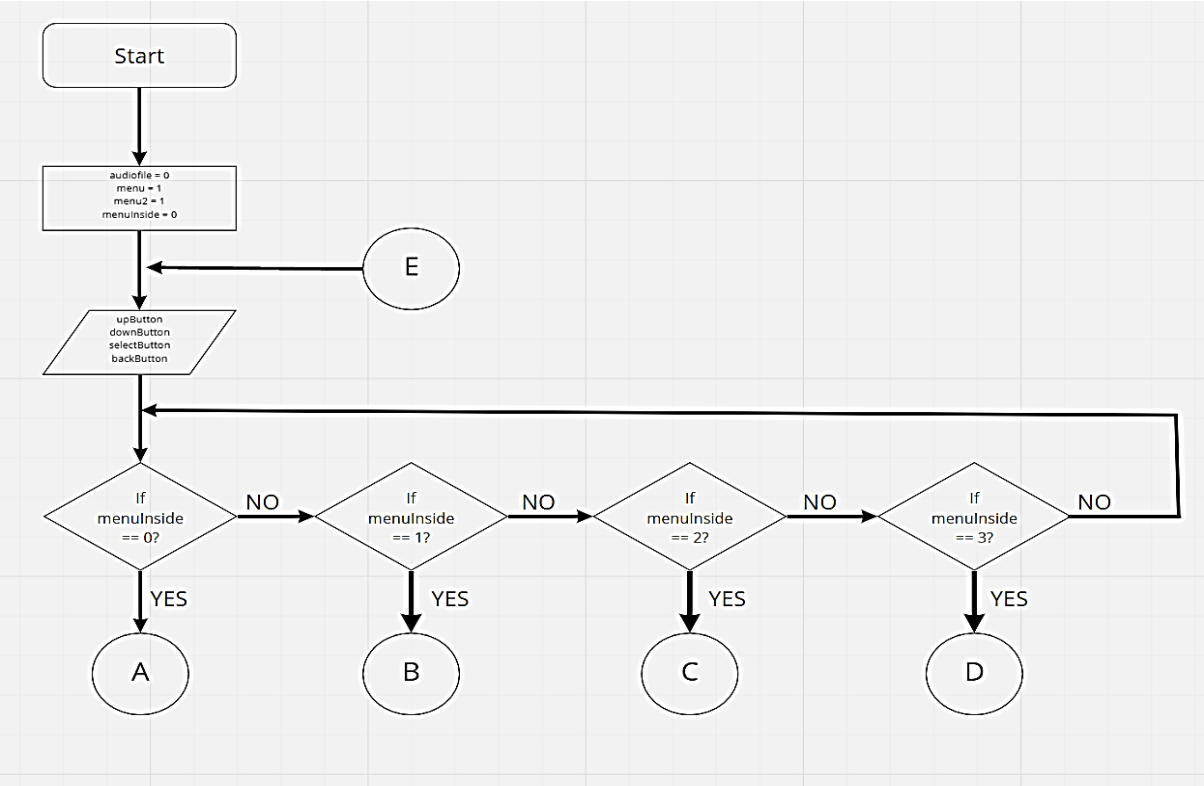
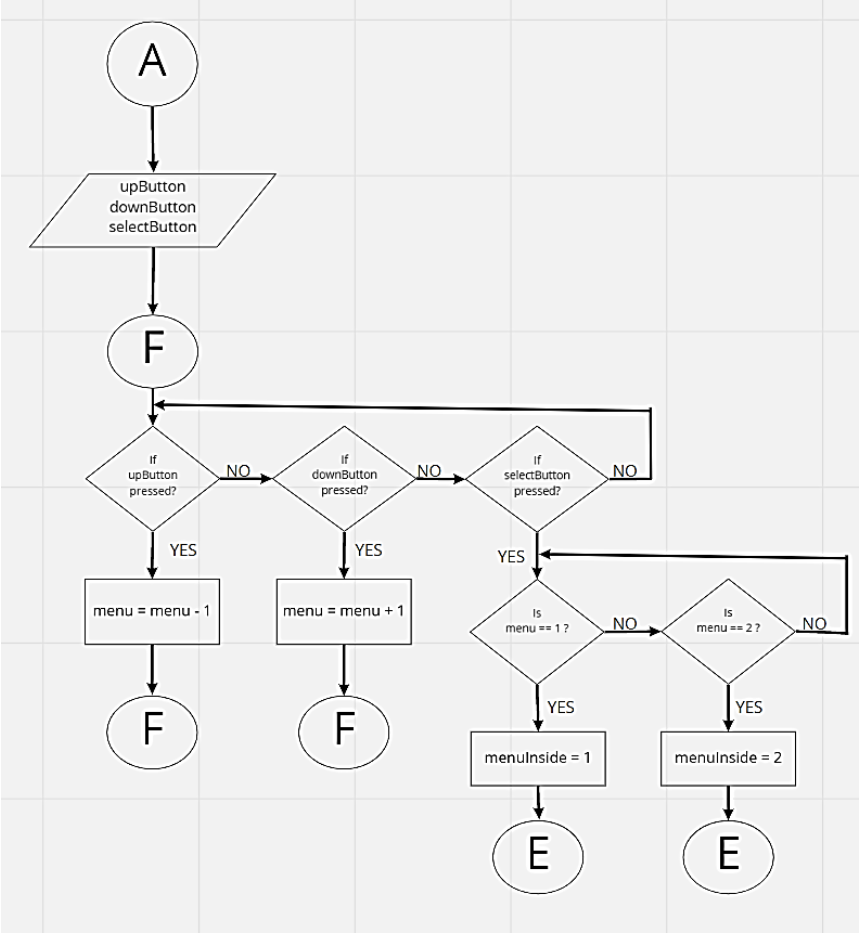
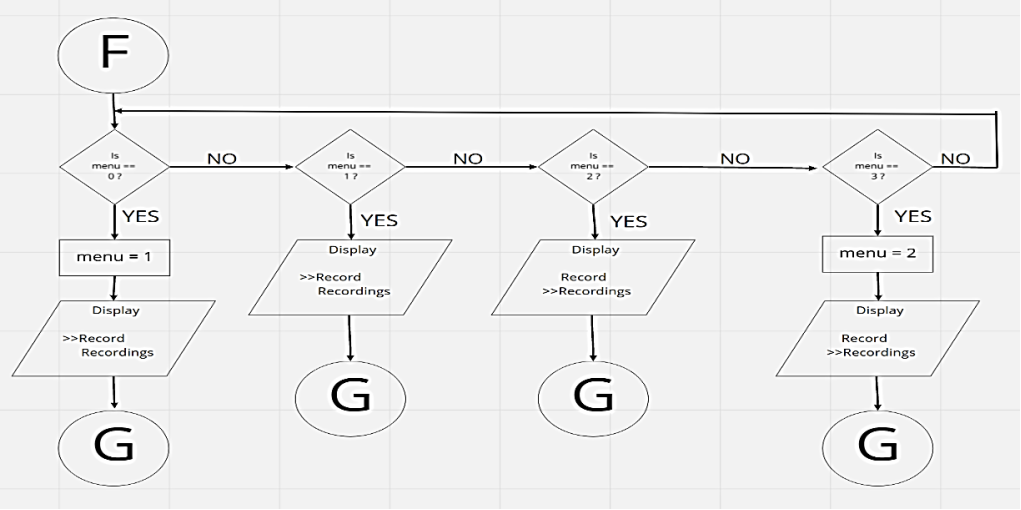


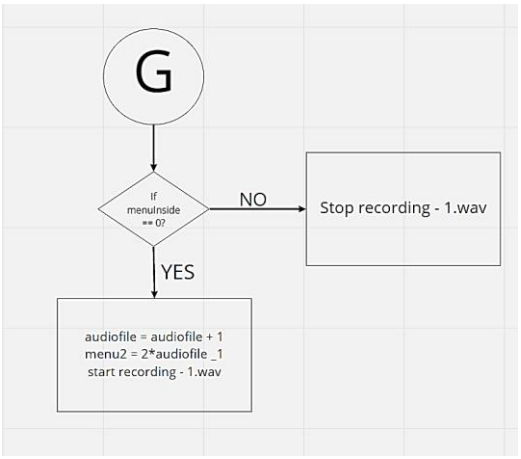
Figure 22 - simplified flowchart of the algorithm

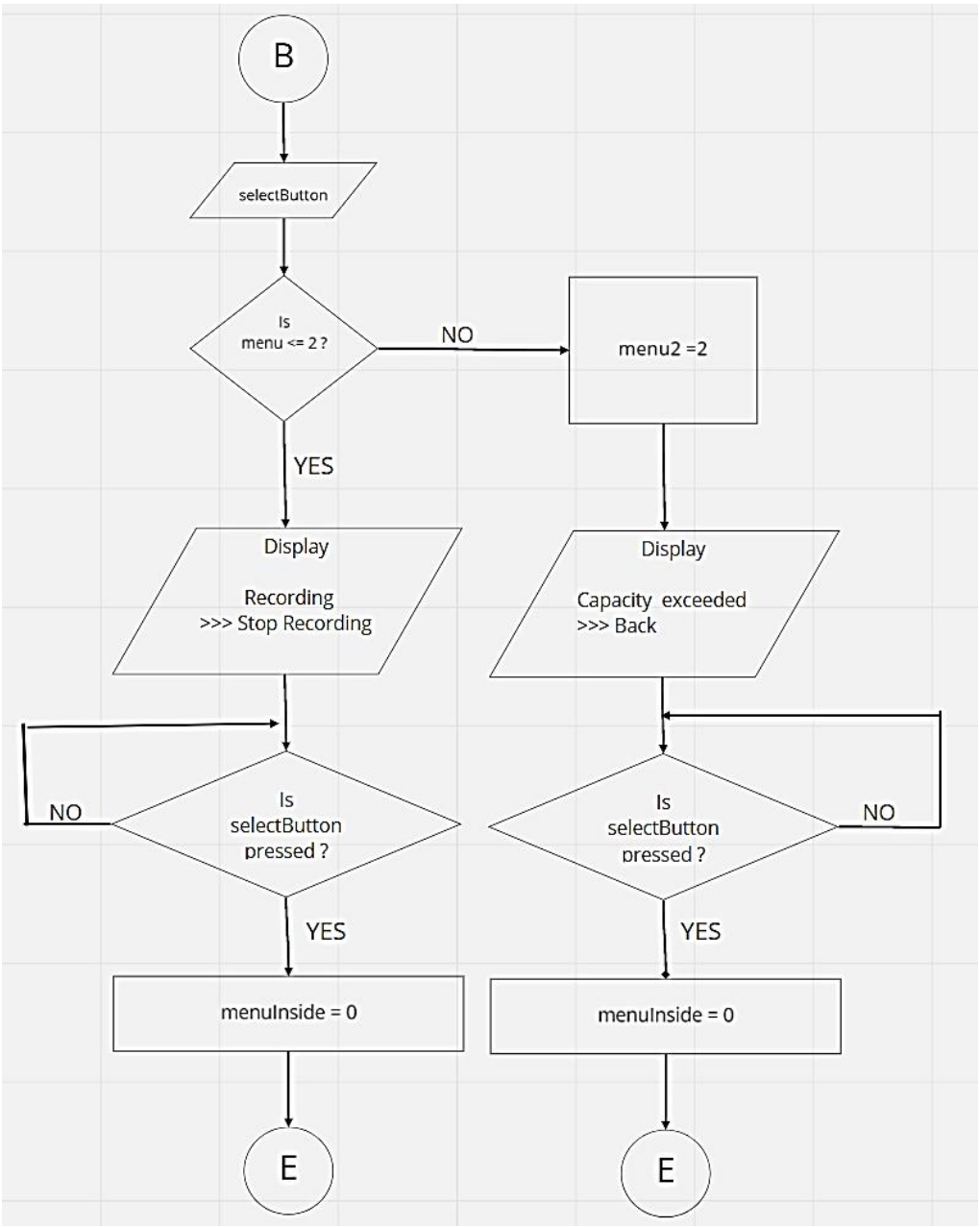
2.6 Algorithm of the Main Program

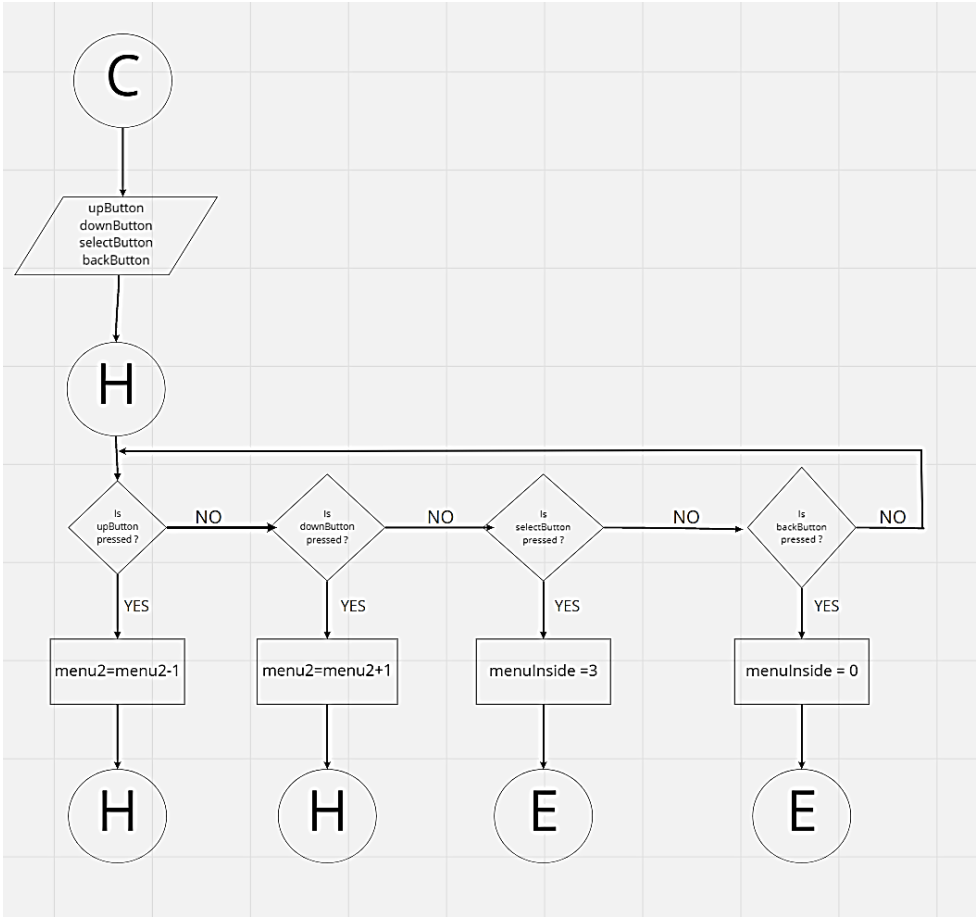


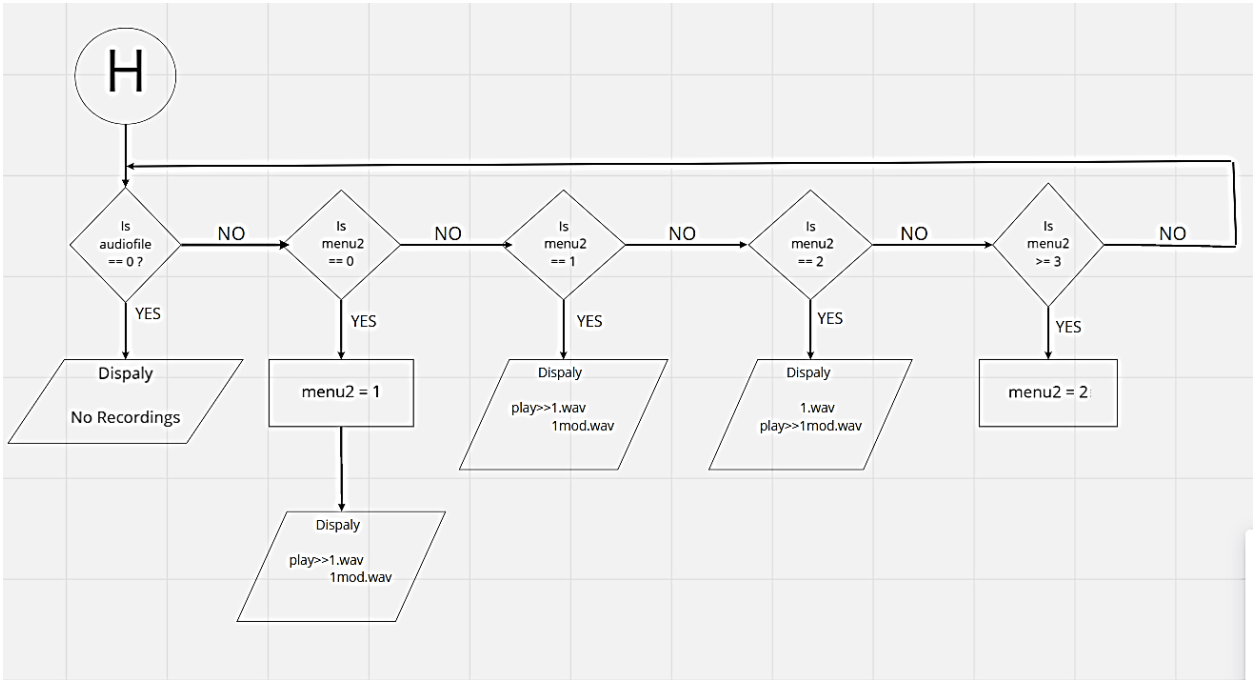


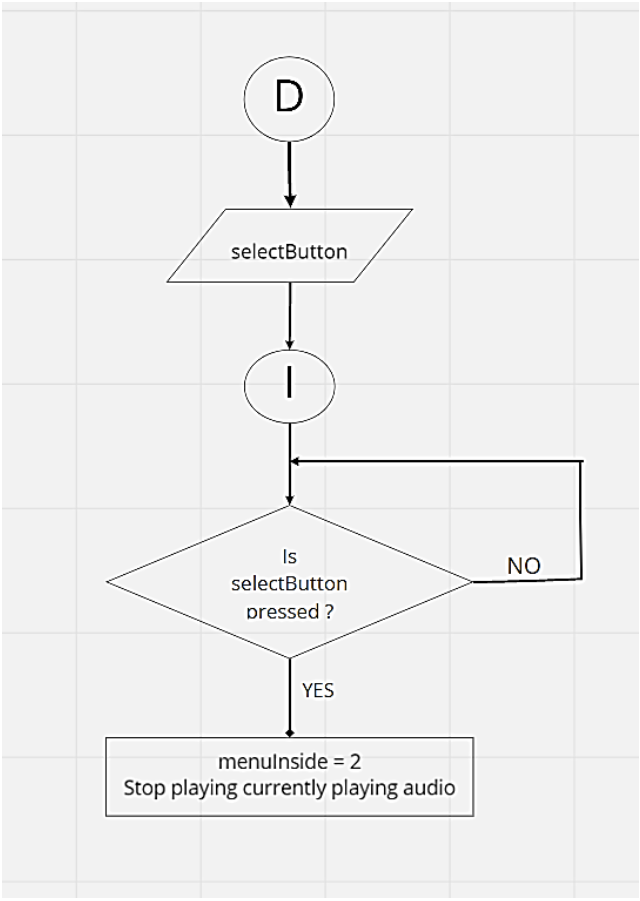

F
Is menu == 0 ?
NO
Is menu == 1 ?
NO
Is menu == 2 ?
NO
Is menu == 3 ?
NO
YES
menu = 1
Display
>>Record Recordings
YES
Display
>>Record Recordings
YES
Display
Record >>Recordings
YES
menu = 2
Display
Record >>Recordings
G
G
G
G

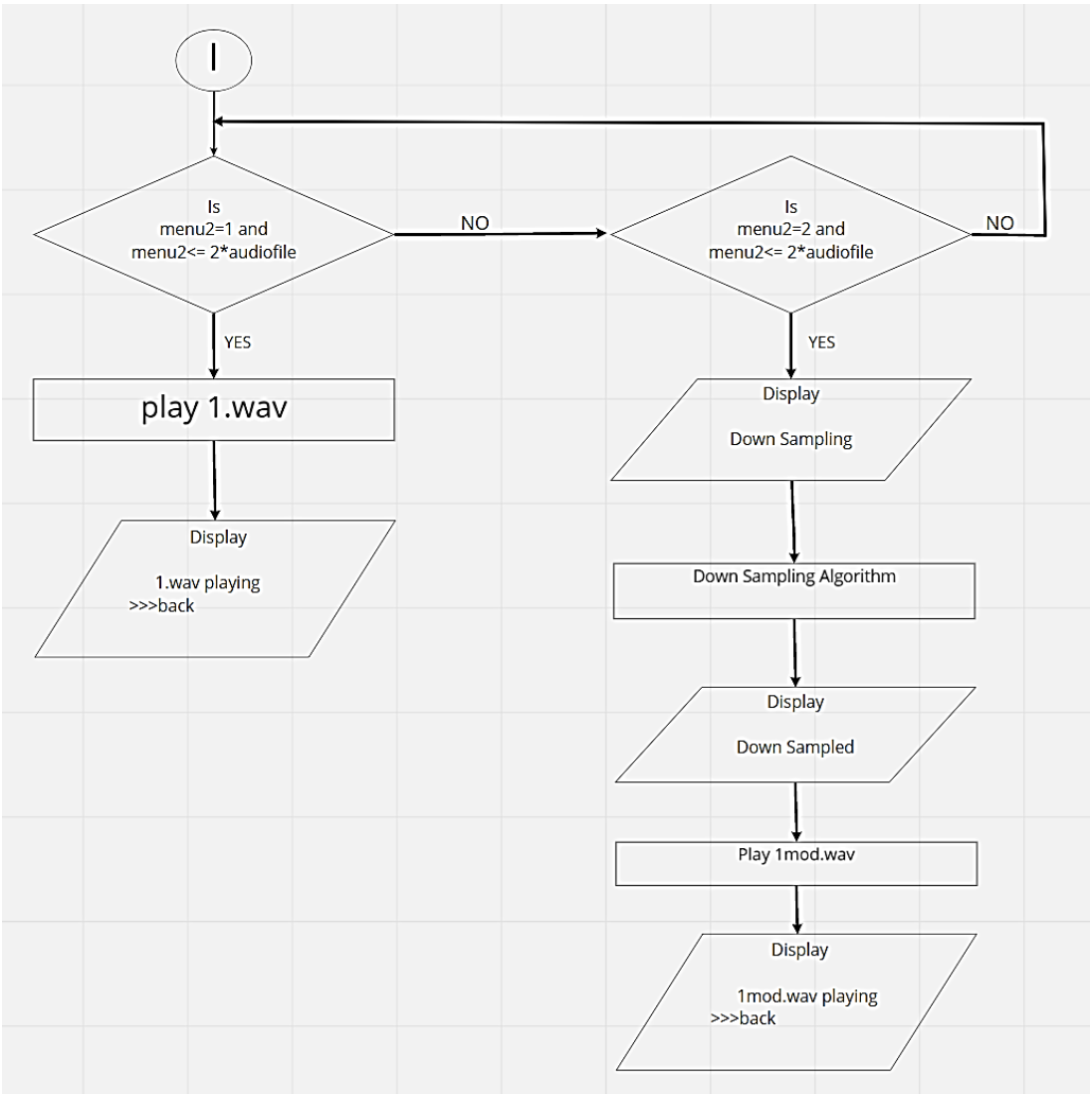

G
If menuInside == 0?
NO
Stop recording - 1.wav
YES
audiofile = audiofile + 1
menu2 = 2*audiofile _1
start recording - 1.wav


B
selectButton
Is menu <= 2 ?
NO
menu2 =2
YES
Display
Recording
>>> Stop Recording
Display
Capacity exceeded
>>> Back
NO
Is selectButton pressed ?
NO
Is selectButton pressed ?
YES
YES
menuInside = 0
menuInside = 0
E
E


C
upButton
downButton
selectButton
backButton
H
Is upButton pressed ?
NO
Is downButton pressed ?
NO
Is selectButton pressed ?
NO
Is backButton pressed ?
NO
YES
YES
YES
YES
menu2=menu2-1
menu2=menu2+1
menuInside =3
menuInside = 0
H
H
E
E


H
Is audiofile == 0 ?
NO
Is menu2 == 0
NO
Is menu2 == 1
NO
Is menu2 == 2
NO
Is menu2 >= 3
NO
YES
YES
YES
YES
YES
Dispaly
No Recordings
menu2 = 1
Dispaly
play>>1.wav
1mod.wav
Dispaly
play>>1.wav
1mod.wav
Dispaly
1.wav
play>>1mod.wav
menu2 = 2


D
selectButton
I
Is
selectButton
pressed ?
NO
YES
menuInside = 2
Stop playing currently playing audio


I
Is
menu2=1 and
menu2<= 2*audiofile
NO
Is
menu2=2 and
menu2<= 2*audiofile
NO
YES
YES
play 1.wav
Display
1.wav playing
>>>back
Display
Down Sampling
Down Sampling Algorithm
Display
Down Sampled
Play 1mod.wav
Display
1mod.wav playing
>>>back

3. Discussion

First of all, at the beginning of the project, we divided work with our four members. Akila took the recording part and PCB designing part. Shivanka took the playback part and breadboard implementation part. Kithdara took the Matlab coding part and adding the part of the effects. Samal took the user interface part and enclosure designing part. The Hard-working of our team members and correct guidance from our group instructors paves the way to a successful project.

During this project, our team members faced some common problems. The main problem was we had to work from home due to covid 19 pandemic situation. It was a new experience and a challenge to us but doing regular zoom meetings with our instructors and among us, we manage the pandemic situation. Apart from that lack of coding knowledge is also a common problem we faced during this project. Regular group meetings and discussions and doing research on the internet we managed to overcome that problem too.

When it comes to recording and play part, initially we tried to implement those in Arduino simulation using RC ladder but later we used TMRpcm library. First, we tried to use only play and record prat in library but it was unsuccessful, so we used whole library later. Also, we managed to implement the required three effects (frequency scaling, selective enhancing, frequency shifting) in Matlab. When we convert those 3 Matlab codes to Arduino and AVR we faced some problems. Since the Arduino board has limited memory and ram, we have to add additional loops and other necessary changes to implement those effects in Arduino.

Also, we faced some problems when we combine codes to make our full code. Since atmega328p microcontroller has limited memory and global variable memory we had to reduce some features in the voice recorder. Finally, we were able to make a voice recorder that can record 3 voice recordings and selectively playback the with frequency scaling effect.

When it comes to the breadboard implementation part we faced some problems too. The main problem was due to the pandemic situation we cannot find the necessary parts needed to project. We have to online order them and wait until they arrive. Also, we had a problem with our microphone module and speaker module. We bought a microphone module with adjustable gain but when we start recording from it there is a loud noise in the recorded audio

file. We tried to manage that noise by adding a RC filter. We think an error in the microphone module is the reason for this loud noise.

Also, we had a speaker module with an amplifier circuit but unfortunately, it burned. So, we had to use an 8ohm speaker in our breadboard implementation without using amplification. Also, in enclosure and PCB design, we faced some practical problems, but we were able to solve them. Finally, with correct guidance and effort from our team members, our group was able to make a working voice recorder.

4. References

Atmel Corporation. (n.d.). *ATmega328P, 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash, DATASHEET*, ATmega328P (microchip.com)

docr. (2019, February 27). *Arduino ADC Programming*. Retrieved from YouTube: https://youtu.be/YYE9jq5sXo4

Hienzsch, D. (n.d.). *Arduino from scratch part 10 - ATmega328P subsystem*. Retrieved from Rheingold Heavy: https://rheingoldheavy.com/arduino-from-scratch-part-10-atmega328p-subsystem/

Smyth, T. (2019, October 15). *Nyquist Sampling Theorem.* Retrieved from http://musicweb.ucsd.edu/~trsmyth/digitalAudio171/Nyquist_Sampling_Theorem.html

TMRpcm/Home, https://github.com/TMRh20/TMRpcm/wiki, 2020

Learn About ATmega328P Fuse Bits and How to Use Them with an External Crystal Oscillator, Learn About ATmega328P Fuse Bits and How to Use Them with an External Crystal Oscillator - Projects (allaboutcircuits.com)

LCD interfacing with ATMEGA32 AVR MICROCONTROLLER, https://microcontrollerslab.com/lcd-interfacing-atmega32-avr-microcontroller/

5. Appendices

5.1 Appendix 1 – Proteus Simulation

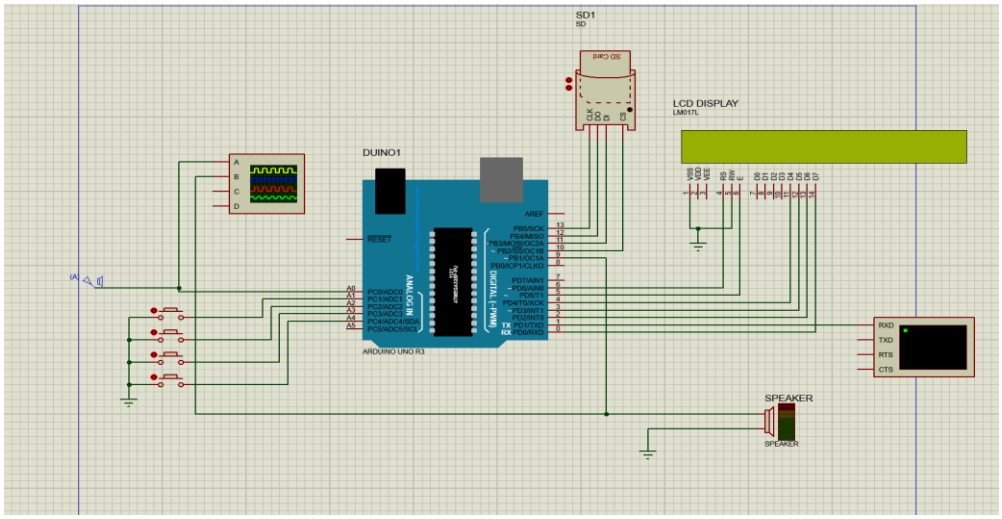


Figure 23 - Proteus simulation of project using Arduino

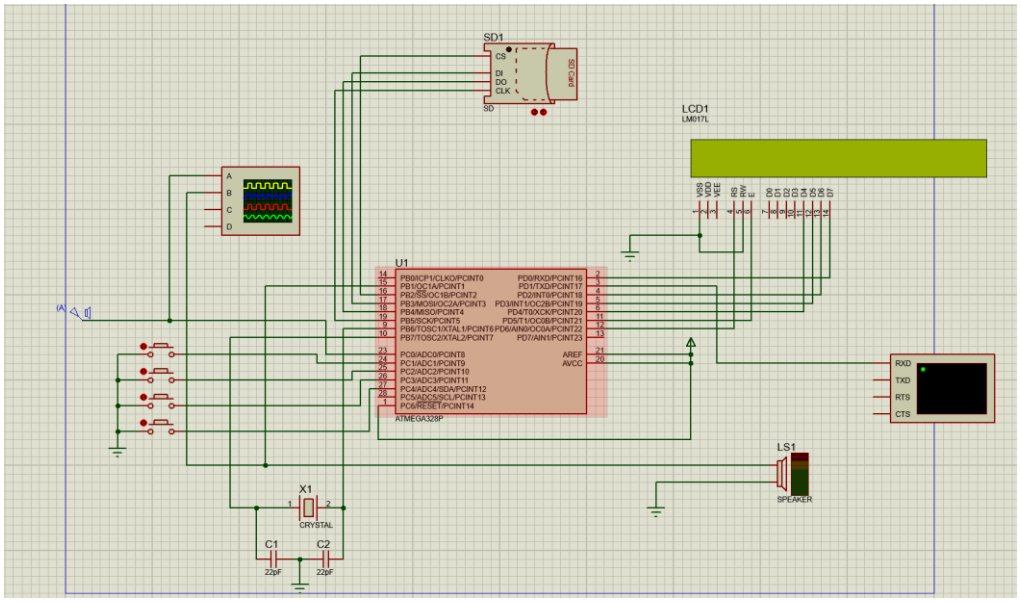


Figure 24 - Proteus simulation of project using ATMEGA 328-PU microcontroller

5.2 Appendix 2 – Breadboard Implementation

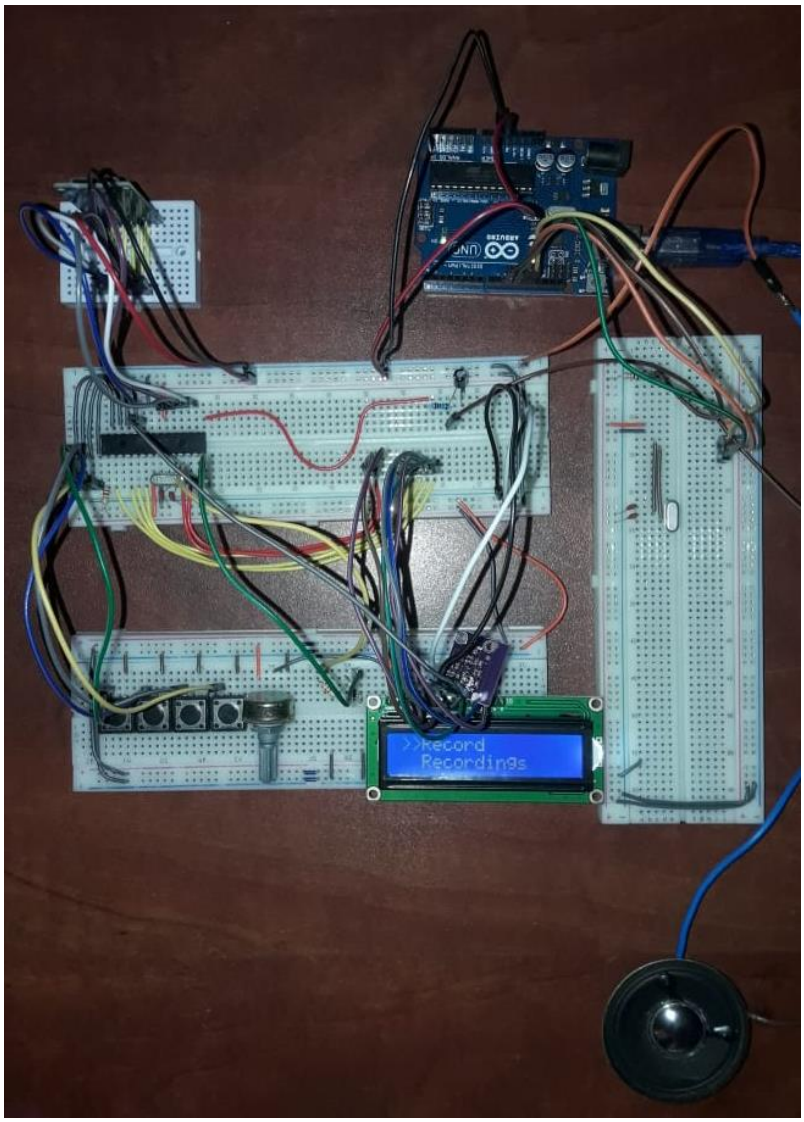
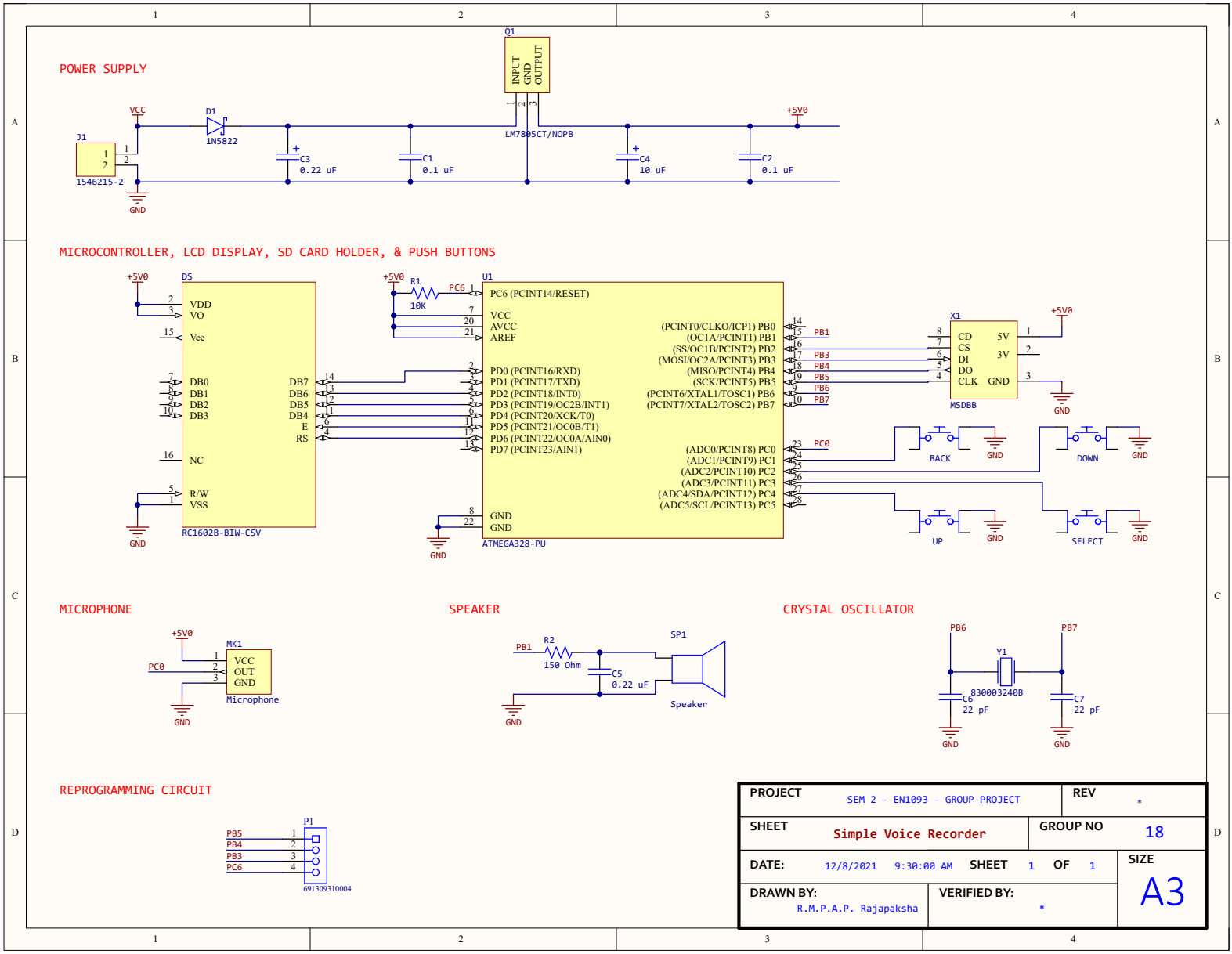
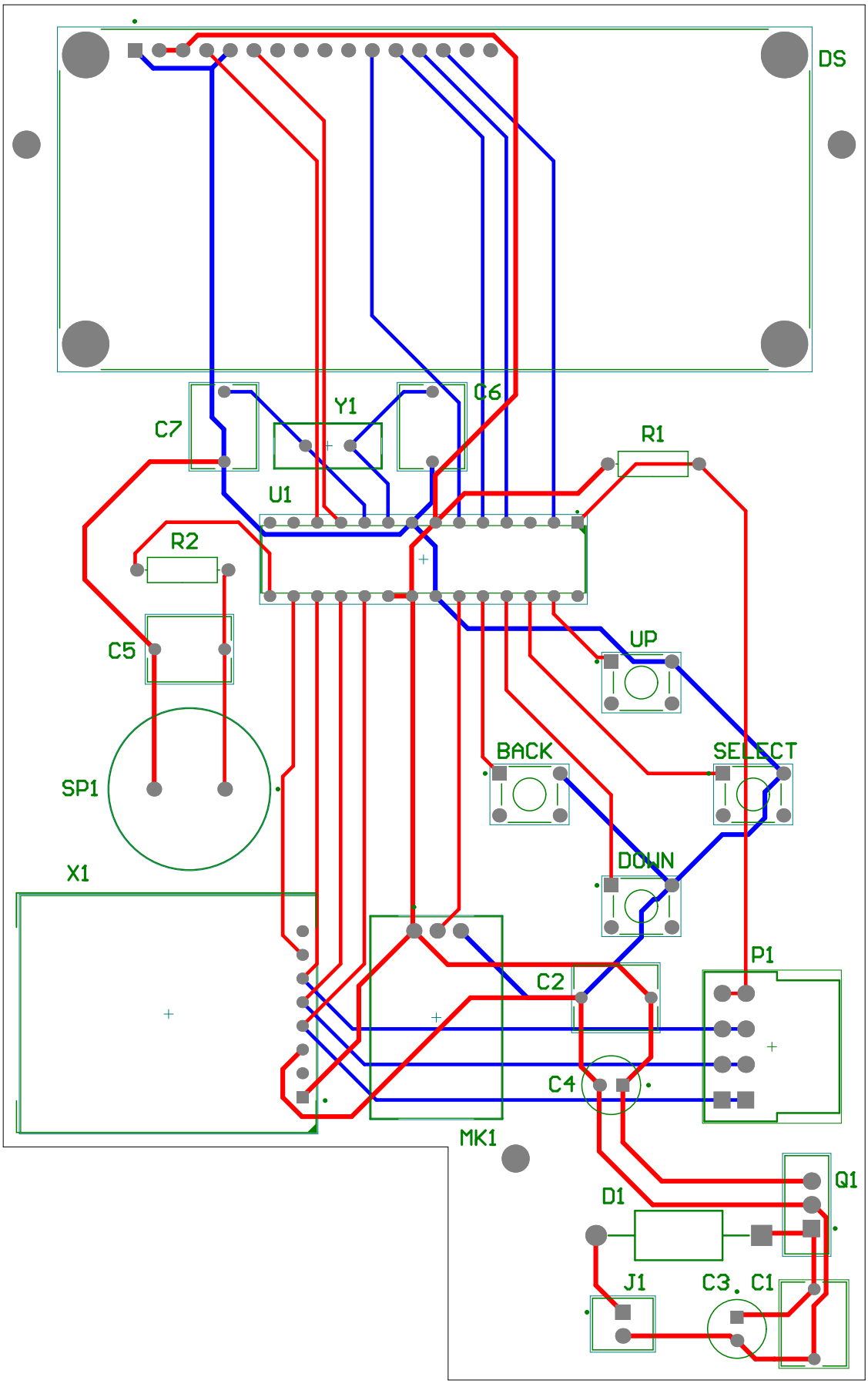


Figure 25 - Prototype on breadboard using ATMEGA 328-PU microcontroller

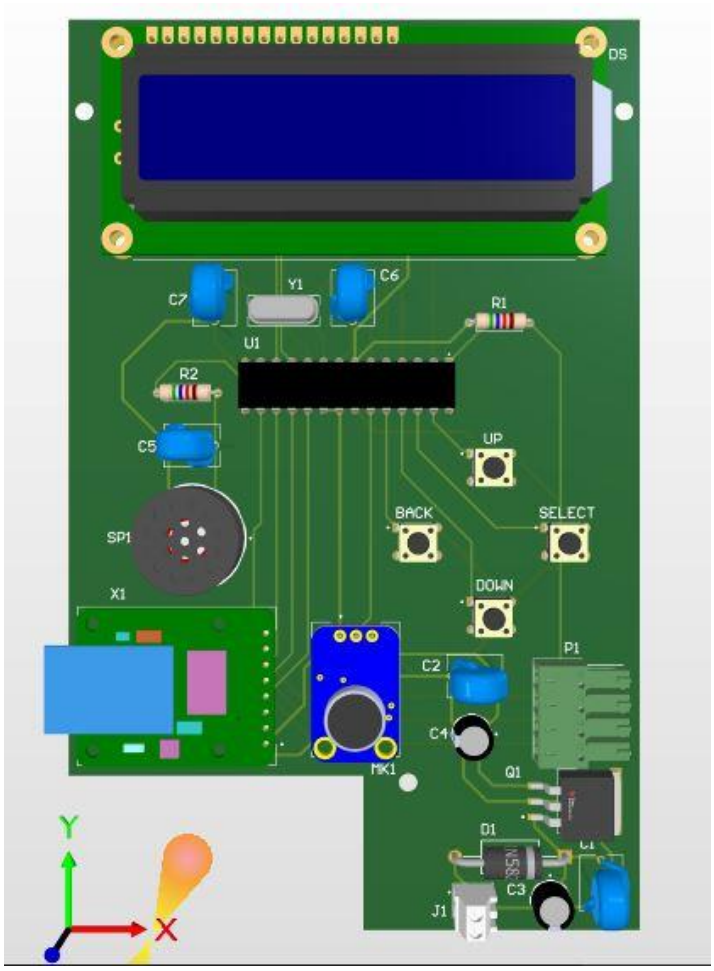
5.3 Appendix 3 - PCB Schematic

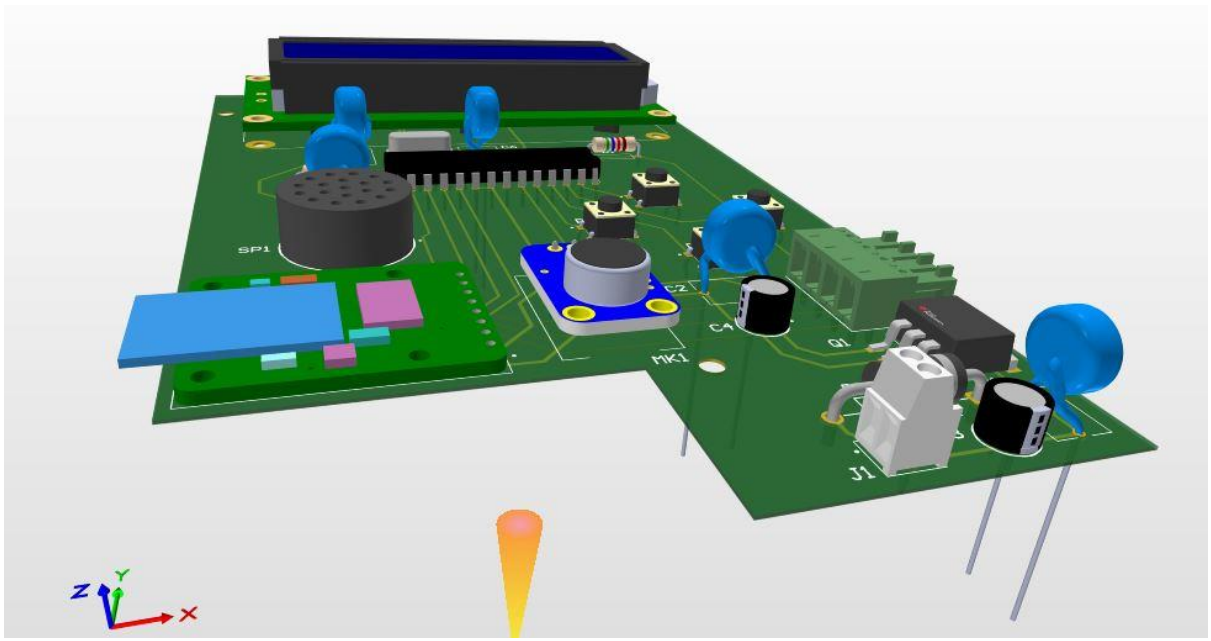


5.4 Appendix 4 - PCB Layout



5.5 Appendix 5 – PCB 3D View





5.6 Appendix 6 – Enclosure Design

Enclosure is included with two sub parts.

- Upper Case
- Lower Case
- Battery Cover

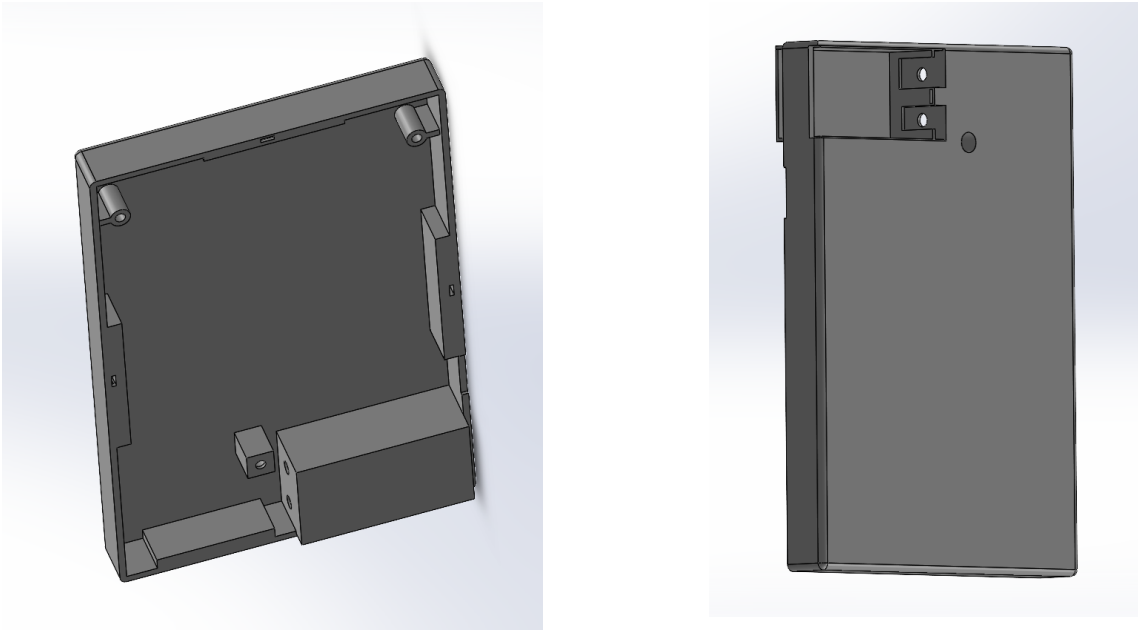


Figure 26 - Upper Case

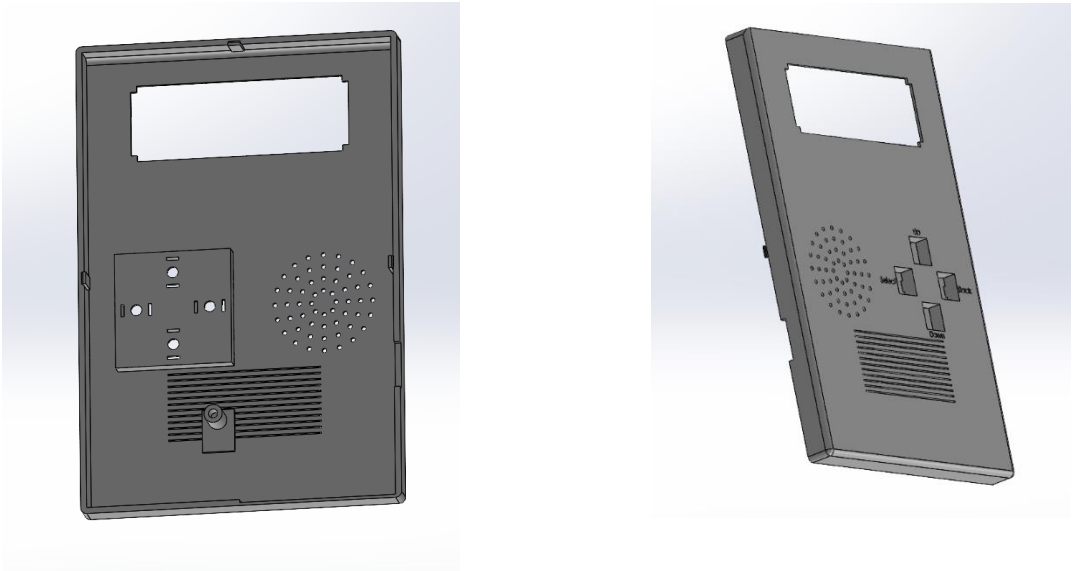


Figure 27 - Lower Case

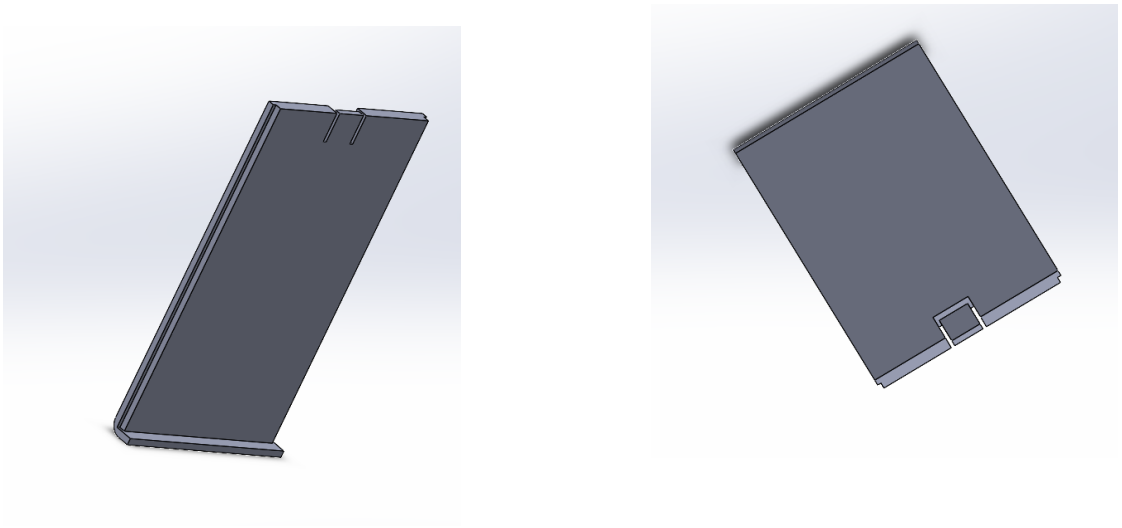
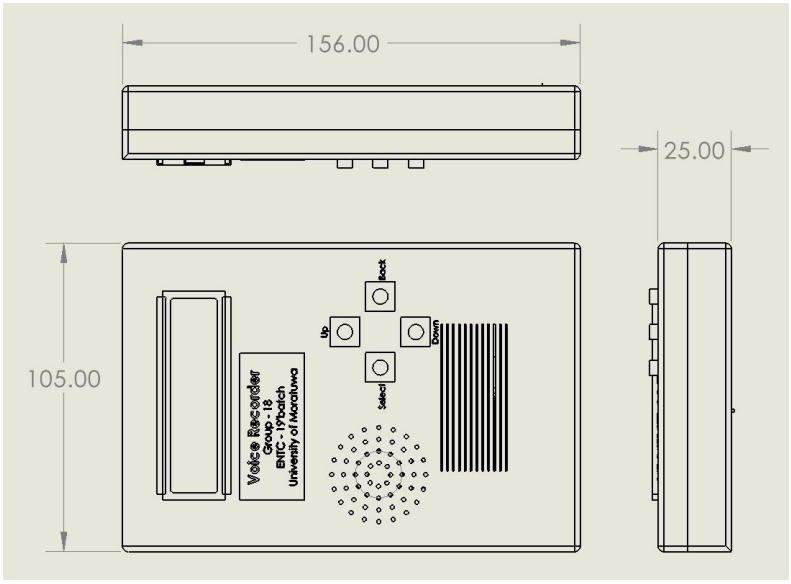
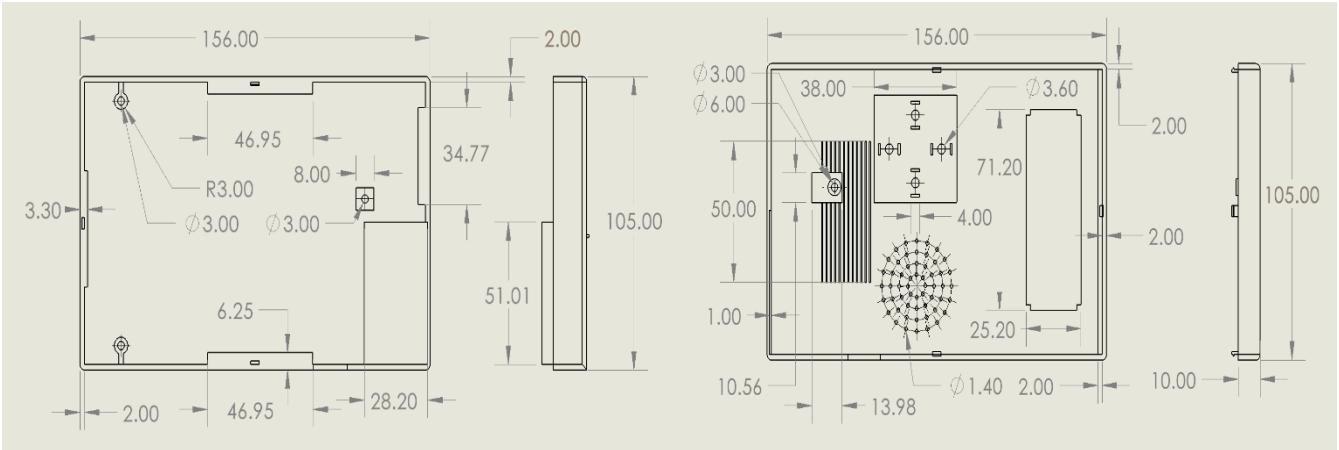


Figure 28 - Battery Cover

Measurements of Enclosure





Color

- *Case* – Grey
- *Push Buttons* – Black
- *Name Plate* – Blue

Mass Properties

- *Material used* - Acrylonitrile Butadiene Styrene (ABS) Plastic

Part Name	Mass (g)	Total (g)
Upper Case	29.03	29.03
Lower Case	53.91	53.91
Push Buttons	0.59×4	2.36
Battery Cover	2.78	2.78
Battery	37	37
PCB	140	140
Total		**265.08**