AUDIO EQUALISER AND AUDIO SPECTRUM ANALYSER

NOAMAAN MOHAMED

BSC

2018

ASTON UNIVERSITY

ASTON UNIVERSITY

AUDIO EQUALISER AND AUDIO SPECTRUM ANALYSER

NOAMAAN MOHAMED

BSC

2018

Summary

As people get older, their hearing range decreases which makes it harder for them to hear the treble and bass frequencies. The use of audio equalisers can help alleviate this issue as it allows for the gain control of specified frequency bands, audio equalisers are also useful in improving the frequency response on speakers. Therefore, the first purpose of this project is to create a 3- band audio equaliser.

The second objective of this project is to output the audio signal onto a display using only a micro controller. The reason for doing this is to see if eight frequency bands could be created using a digital filter. The purpose of using digital filters is to eliminate the cost of using a dedicated chip and give more flexibility in the bands themselves. This digital filter will be done using Goertzels algorithm, its used to create band pass filter.

The final objective of this project is to output the eight bands onto a neopixel display. A neopixel display is a circuit board with 64 LEDs on it that can be coded through its one data pin.

The equaliser is made using three different filter types and a summing amplifier. The results from this equaliser were that the bass, mid and treble bands all worked as expected and it resulted in an equaliser that had a +/-6dB gain.

The Goertzel filters were all coded using a microcontroller, its input were eight different frequencies. The result from this Goertzel was then inputted into the neopixel. The results from the filter was that it worked properly for frequencies above 500Hz but struggled to filters lower frequencies properly.

The neopixel shield was also coded using a microcontroller, the input from the Goertzel decided how many lights on the neopixel would light up. The results from the neopixel were also as expected.

The conclusion for this project was that all objectives were met but the Goertzel filter could be further refined to work with lower frequencies.

Acknowledgements

I would like to thanks those who helped me with this project, including my main supervisor Richard Nock and technicians Steve Dellow and Richard Reeves without whom this project would not be possible.

Contents

Introduction

The average hearing range of a young person is between 20Hz-20000Hz [1] but as a person ages their hearing range gradually decreases, and it becomes harder to hear the lower and higher frequencies. However, it is possible to use audio equalisers to adjust the frequency response of an audio signal. Equalisers can raise the amplitude of the lower and higher frequencies so that people who have difficulties hearing these frequencies are able to hear them, audio equaliser are also useful to improve the poor frequency response on speakers [2].

As well as hearing a change in the audio signal, showing the change visually will give greater clarity to the change happening to the audio signal, this can be done by including a display that can show the difference between the equalised and the original audio signal.

The purpose of this work was to create a three-band equaliser that could control the bass, mid and treble frequencies. This signal would then be passed through eight Goertzel filters [3]. The output of these would then be displayed on a NeoPixel 8x8 matrix [4].

The main objectives and their completion dates are given below.

- Design, build and characterise a 3-band tone control, as a self-contained module. Completed on 30th March.
- Use the Goertzel Algorithm to act as band pass filters to break down signal into different bands using an AVR. Completed on 13th April.
- Output Audio signals spectral content onto a neopixel board. Completes on by 16th April.

The procedure to create the equaliser was to first do the filter and gain calculations, once this was done the circuit was design using Proteus (software for designing circuits and PCBs). Thereafter the circuit was built using a breadboard to compare it to the simulation. Once this was working correctly a PCB was made and the equaliser was finalised. An oscilloscope was then connected to the equaliser to check whether it functioned properly.

The filters were designed by creating a Goertzel filter function and then applying that to the eight frequency bands. The purpose of using Goertzel filters is because they are digital filters, using them would remove the need of using a dedicated integrated chip to do filtering and this would save a lot on costs. These filters would output an integer result which reflected the current power of the frequency, the power was checked by using a serial monitor. The amount of lights that would light up on the neopixel depended on the power result. If the neopixel would react properly to the power number, then it means the neopixel was fine. The filter and the neopixel code was all done using an AVR microcontroller [5].

Background

Equaliser

The audio equaliser was supposed to be designed as a two-band equaliser that would control the bass and treble frequency, however a decision was made to add a mid band as well. The theory behind this circuit was to use to create a low pass filter that only let bass through, a band pass filter for mid frequencies and a high pass filter for treble frequencies. These outputs of these three filters will be connected to a potentiometer which controls the gain of the filter. Then these three outputs need to be combined using a summing amplifier [6]. A summing amplifier is used to combine the voltages present on the three outputs and converts them into one input. This output will then need to be lifted, as audio is usually centred around 0V which means the output is positive and negative, but the microcontroller cannot read negative voltages so one last op amp is used to centre the output around 2.5V. This equaliser design is like a graphic equaliser, although a graphic equaliser usually has more bands than three the way in which the gain is controlled is the same. The reason more bands were not added was to keep the product simple for its end user and to keeps costs lower as the more bands added meant the more expensive the product would become.

Goertzel Filter

Previously I was part of a project last year that used an MSGEQ7 microchip [7] to act as seven band pass filters and 7 peak detectors. This set up worked well but the display last year allowed for eight bands to fit but the project was limited to seven, also a circuit board had to be made for the MSGEQ7 which added to the costs. This project will develop a method to achieve better results than what the MSGEQ7 did and all the filtering by code will be done with code. Therefore, the Goertzel algorithm is used.

The Goertzel algorithm is used in digital signal processing for tone detection. The purpose of it is to create a narrow band pass filter that can detect certain frequencies in a signal or it can also be regarded as “the evaluation of a single bin of the FFT (Fast Fourier transformation)” [8]. This is useful for low power systems which do not have the processing power to calculate all bins simultaneously. The Goertzel algorithm instead runs multiple for different frequencies and outputs a result if a tone is detected. The maths behind this is given below.

Goertzels Maths

1. Set sampling rate(Fs) and block size(N) to control frequency resolution. The sampling rate is 10kHz and N=40 would give a frequency resolution of 250Hz. The reason for choosing this frequency is due to Nyquist’s theorem [9] that the sampling rate should at least be double the highest frequency used. The highest frequency being sampled should be no more than 4kHz therefore 10kHz was chosen as the sampling rate.
2. When setting N, the target frequency needs to be considered as the target frequencies should be integer multiples of sampling rate/N.
3. Calculate the precomputed constants.

$$Coeff = 2\cos(2\pi\left(\frac{f_{tone}}{10000Hz}\right))$$

4. Convert coeff into Q15 format, Q15 format needs to be used as the numbers need to have constant resolution to perform the Goertzel calculation [10].

5. There are three variables that need to be initialised at the beginning of the block chain.
    - Q0 = coeff * Q1 - Q2 + sample
    - Q2 = Q1
    - Q1 = Q0
6. This leads to the equation: power = $Q_1^2 + Q_2^2 - Q_1 * Q_2 * coeff$

The inspiration behind using Goertzel filters to analyse an audio signal comes from its use in DTMF (Dual-tone multi frequency) signalling [11]. DTMF signally was used in telecommunication systems to detect which number was pressed on the keypad (Figure 1). The Goertzel algorithm is used here to detect specific frequencies. It runs once for each frequency and if for example 697Hz and 1209Hz are detected then that means '1' was pressed.

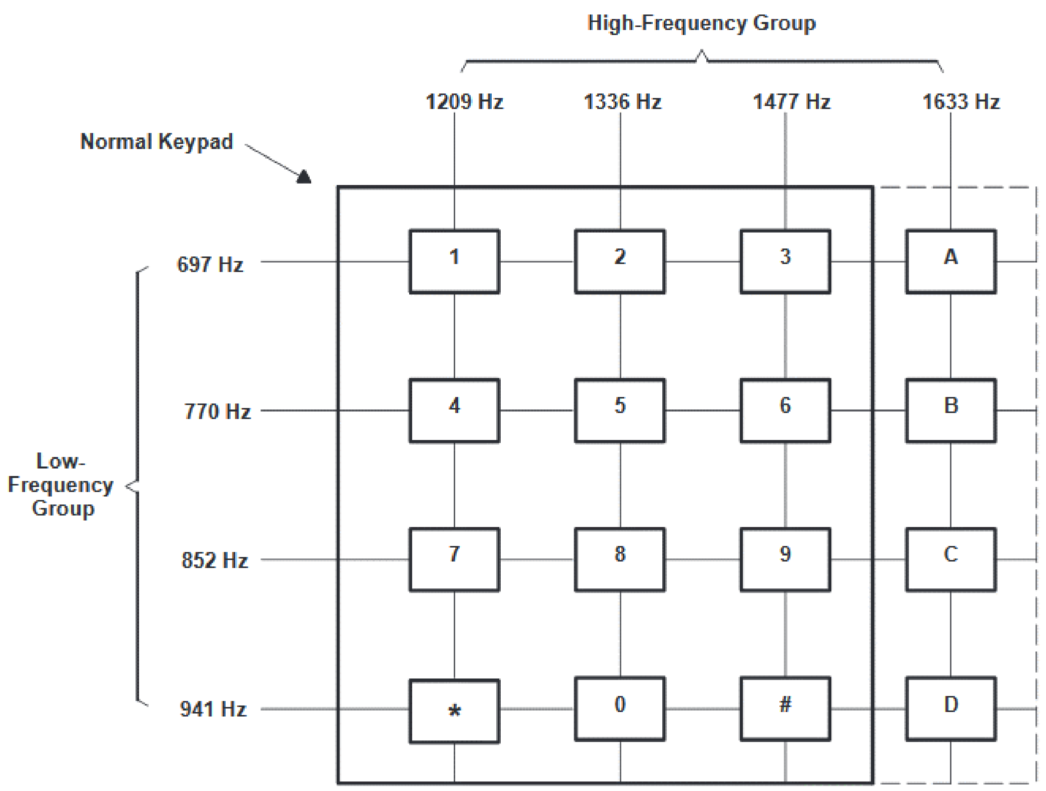


Figure 1. DTMF Keypad

However, DTMF signalling only covers a small frequency range and the Goertzel filter is used as a narrow band pass filter in this example. For this project the Goertzel filter is modified so that each filter has a larger band width and the spectrum analyser needs to cover a larger range of frequencies. This can be achieved my changing the inputs to the filter. The band width of the filter is determined by the number of samples used, the lower the number of samples, the larger the band width becomes. The purpose of increasing the band width is twofold. The first being that if a Goertzel filter was created to only have 20 samples and to detect the 1kHz frequency this would result in a 500Hz band width and even though the filter is set up to detect the 1kHz frequency it is detecting the frequencies of 750Hz to 1.25 kHz. The second advantage is the smaller the number of samples used means the less data that needs to be stored on the microcontroller. This is useful as it means less strain on the micro controller and it also means more efficiency.

The Goertzel filter is not without its disadvantages as in the figure below it shows how although the magnitude output from the Goertzel goes high if a tone is detected side lobes are also detected. These side lobes could cause a problem as they could be detected by the filter as false positives. In theory I should be able to create eight band pass filters using the Goertzel algorithm, and these filters should cover a range of frequencies which would then be outputted onto a display.

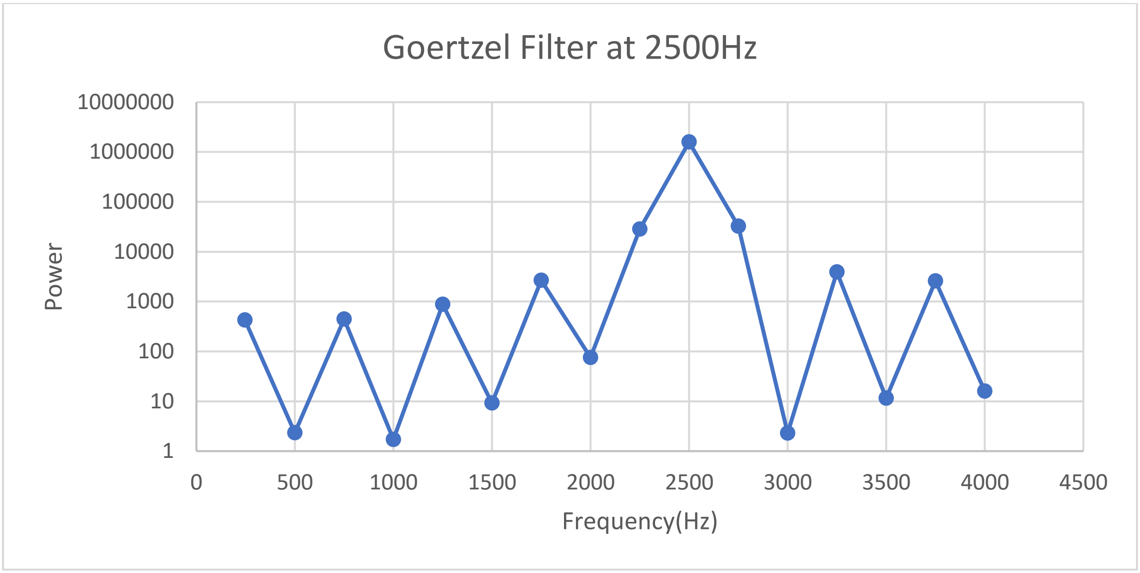


Figure 2. Example Goertzel Filter

AVR Microcontroller

The microcontroller used in this project is AVR specifically an ATMEGA164 (Figure 3), in appendix A the schematic for the microcontroller is shown there. THE AVR microcontroller is used to run all the code in this project. One of its functions in this project is analogue to digital conversion or ADC for short. The purpose of this function is to convert the analogue audio signal into a digital signal that can be inputted the Goertzel function, the reason for doing this is previously mentioned in the equaliser background.

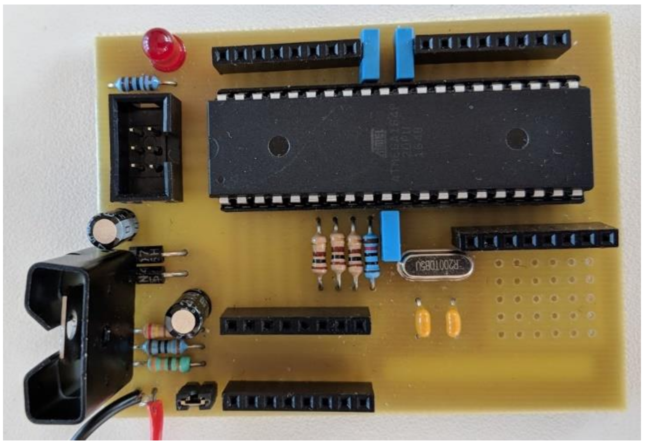


Figure 3. AVR Microcontroller Board

Neopixel

There were many options for displaying the results of the filters. Such as using individual LEDs, a LCD display but for this project a Neopixel shield was used. The reasoning behind this was that the shield has 64 LEDs in 8x8 shape, but they are all controlled by one data pin. The way the output works is that there are eight bands and the higher the power is the more lights that turn on and the maximum amount of lights that will turn on is eight for each band. This is shown in figure 4, the was the colours are set is also shown. The Neopixel shield also allows for chaining, this means multiple shields can chained together and display even more bands if needed. In this design two Neopixel shields are used one will display the left audio signal and the other will display the right audio signal.

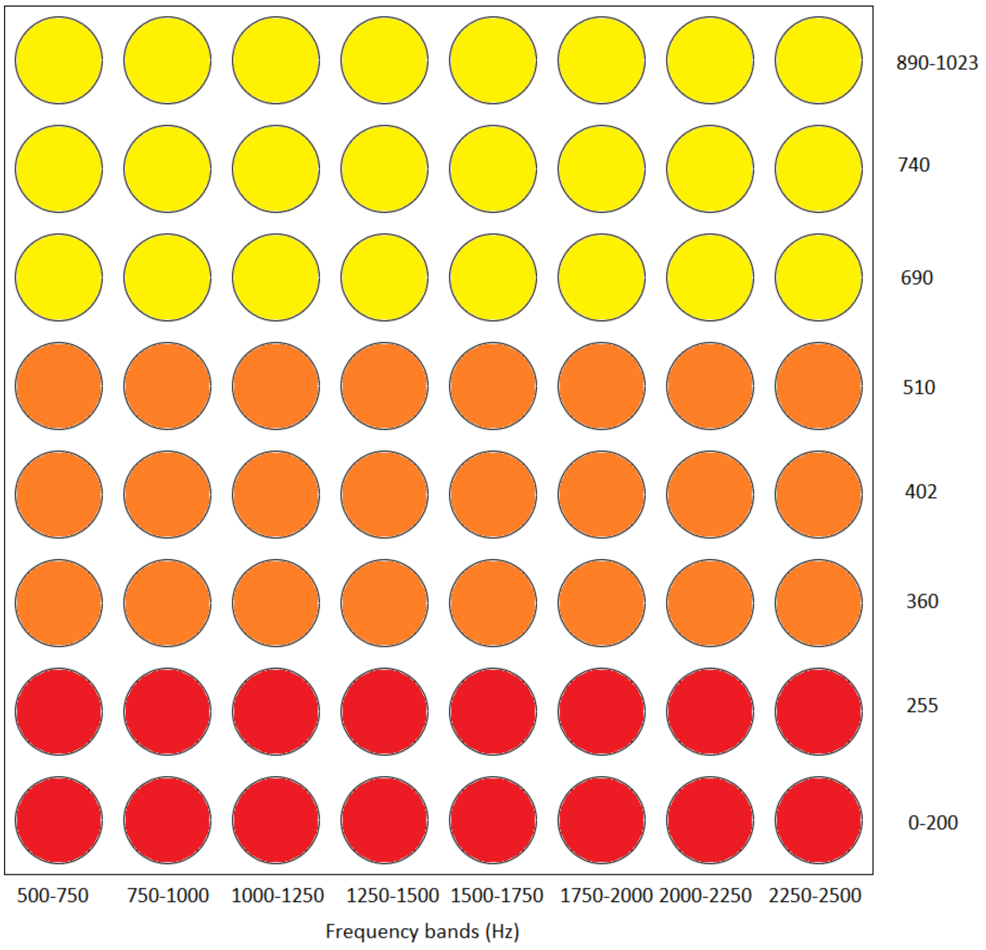


Figure 4. Planned Neopixel board

Procedure

The specification for this project is to create a three-band equaliser that has positive and negative gain. The second is to use an AVR micro controller to create eight digital filters using Goertzels algorithm and the third is to output these bands on to a neopixel. A speaker is also needed to output the sound. The diagram below shows a block diagram of the proposed final design and the rest of this section shows how to create these parts. The arrows show which way the audio signal is traveling, and they also show how the aux cables are connected. This means after the audio signal leaves the equaliser it splits with one signal going to the AVR and the other will go to the speaker. The speaker has not been built and is a pre-built component.

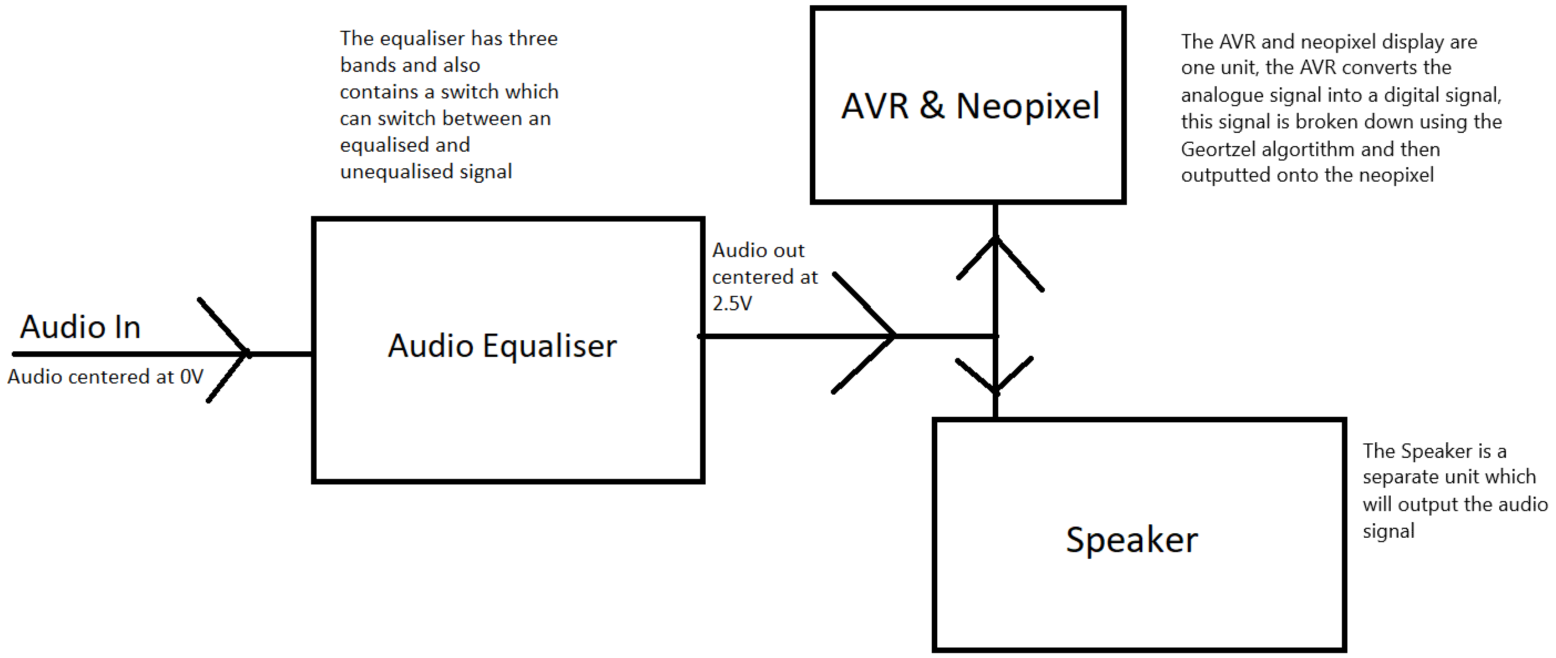

The equaliser has three bands and also contains a switch which can switch between an equalised and unequalised signal
AVR & Neopixel
The AVR and neopixel display are one unit, the AVR converts the analogue signal into a digital signal, this signal is broken down using the Geortzel algortithm and then outputted onto the neopixel
Audio In
Audio centered at 0V
Audio Equaliser
Audio out centered at 2.5V
Speaker
The Speaker is a separate unit which will output the audio signal

Figure 5. Product design Block diagram

Equaliser

The first step in designing the equaliser was to design three filters. These filters would be used to control the bass, mid and treble frequencies. For the bass an active low pass filter (figure 6) was designed, this was because the bass filter was designed to control all frequencies below 250Hz.

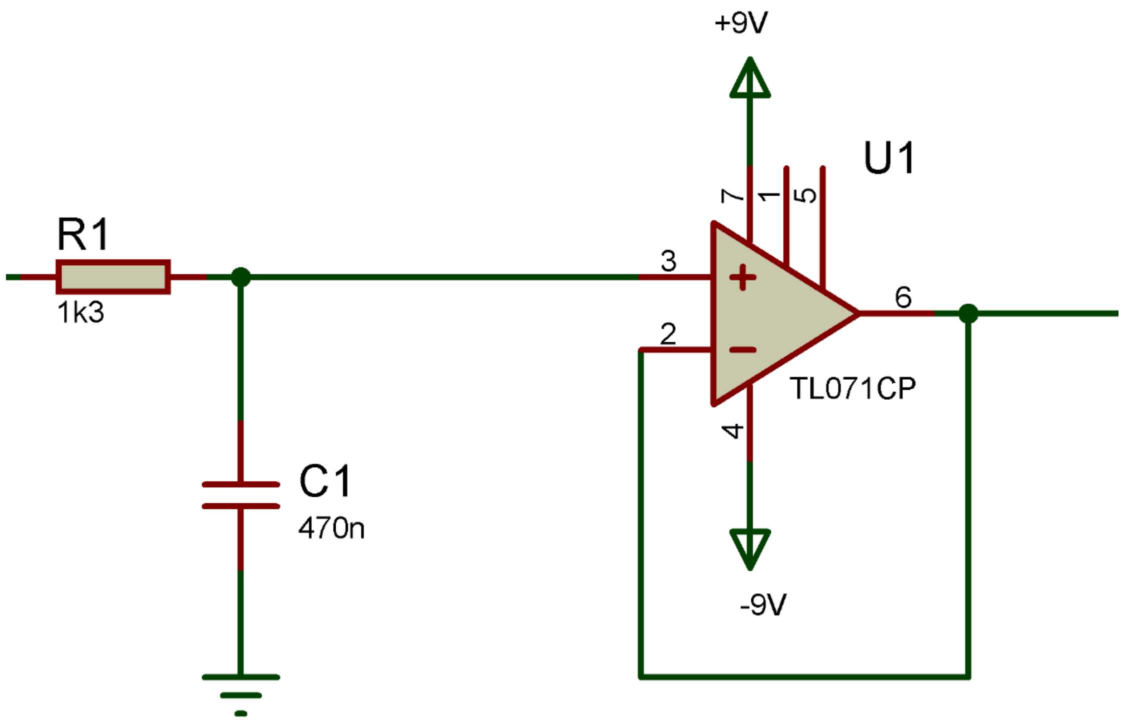


Figure 6. Low Pass Filter

The calculation to get the cut-off frequency is described below.

$$f = \frac{1}{2\pi RC}$$

Selecting frequency to 250Hz and resistance to 1300Ω and then rearranging the equation to give capacitance gives the following result.

$$\frac{1}{2\pi(250)(1300)} = 4.89 \times 10^{-7}$$

470nF is the closest capacitor to the above answer, then calculating the cut-off frequency with the real values is given below.

$$\frac{1}{2\pi(1300)(470 \times 10^{-9})} = 260.5Hz$$

The treble frequencies were controlled by an active high pass filter (figure 7), this filter allowed frequencies above 2000Hz to pass through.

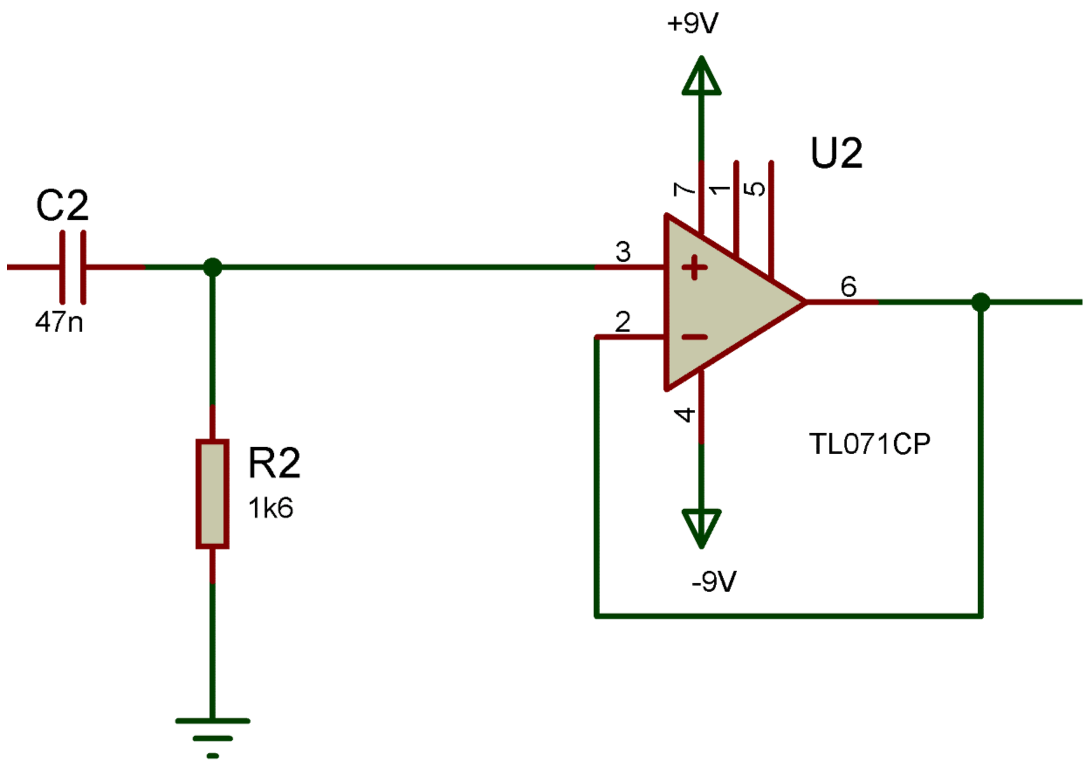


Figure 7. High Pass Filter

The calculation to get the cut-off frequency, resistor and capacitor values was the same method as the low pass filter, below shows the cut-off frequency calculation with the values from figure 7.

$$\frac{1}{2\pi(1600)(47\times 10^{-9})} = 2116.4Hz$$

The final filter controls the mid frequencies, an active band pass filter (figure 8) was selected with the low cut off frequency value being the same as the high pass filter cut off frequency and the high cut-off frequency being the same as the low pass filter cut-off frequency. This means that only frequencies above 260Hz and below 2116Hz would pass through this filter. However, when this was simulated it resulted in the gain being too high when all three bands were maxed out. For this reason, R7 was changed from 1.6kΩ to 2.4kΩ and 1kΩ resistor was added between pin 2 and ground, with a 330Ω resistor was added between pin 2 and 6. The 1kΩ and 330Ω resistor results in this filter having a 2.47dB gain on the audio signal before it reaches the potentiometer. The changes in the resistor also make the low cut-off frequency around 1410Hz.

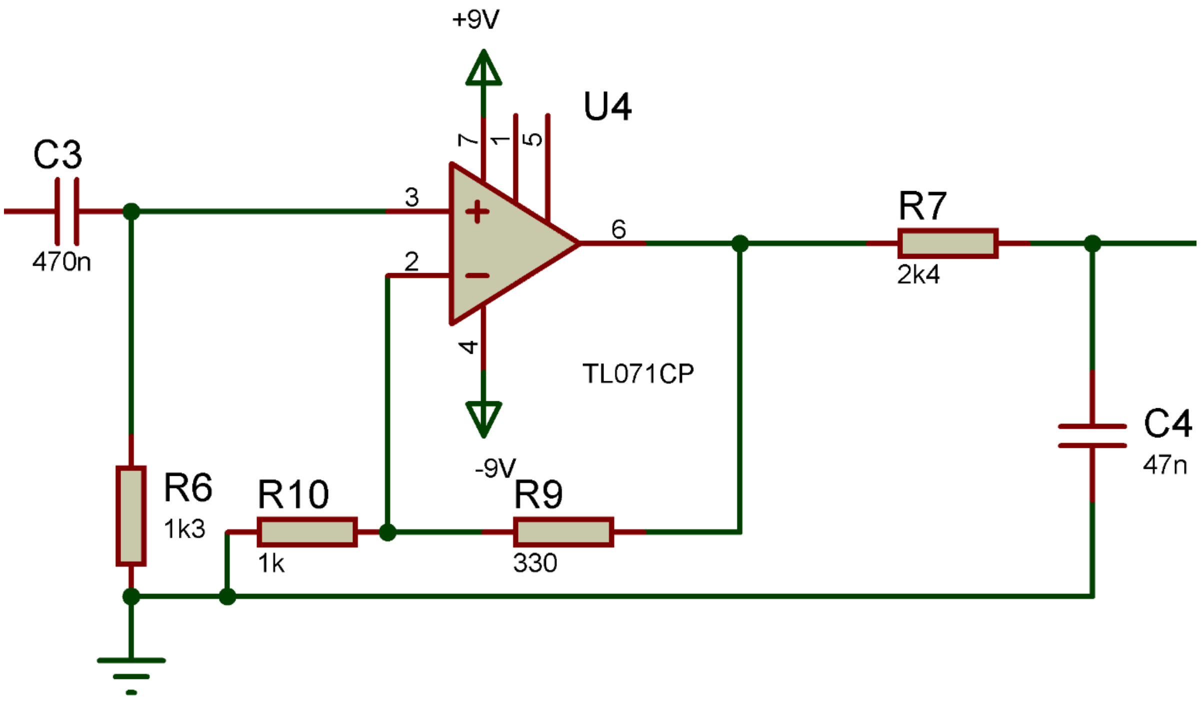


Figure 8. Band Pass Filter

All three of these filters are using a TL071 op-amp, the op-amp is what makes these filters active instead of passive. This op-amp was chosen as it is cheap and was readily available at university. It is dual supply, as it is shown in all the figures that each op-amp has a positive power supply at pin 7 and a negative supply at pin 4. 9V was chosen as this was the lowest voltage that is easy to attain and would result in no clipping. Also 100nF capcitors were on all power pins to act as decoupling capacitors, this is done to reduce noise.

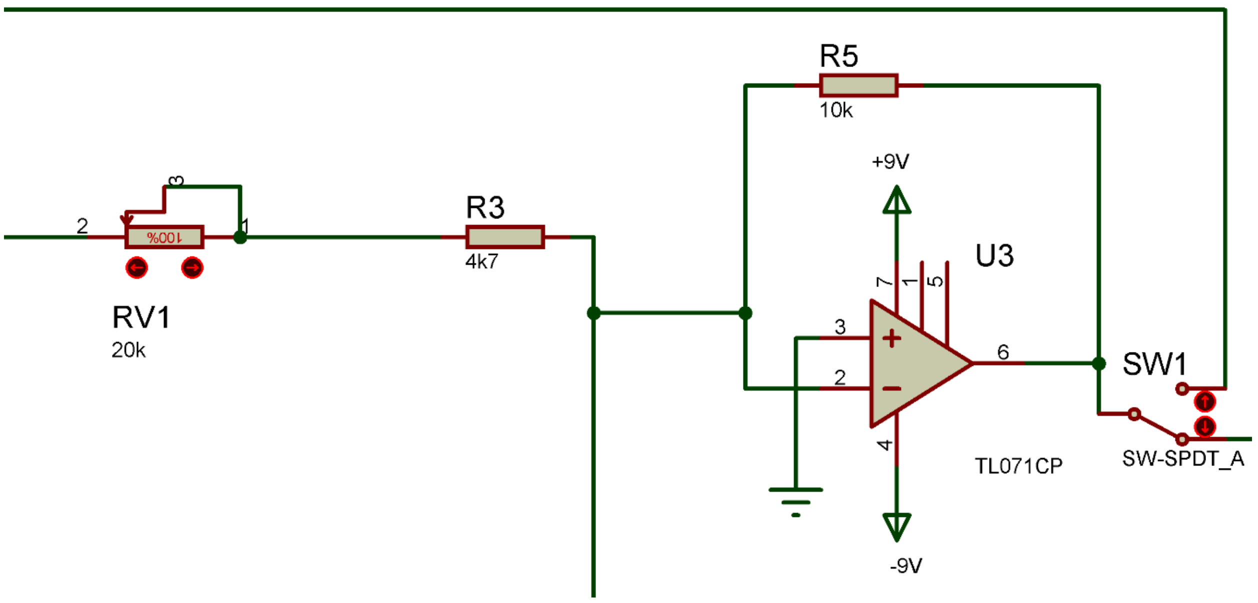


Figure 9. Potentiometer and Summing Amplifier

The output of these three filters were then connected to a 20kΩ potentiometer, these potentiometers each had a 4.7kΩ resistor connected to each of them (figure 9), the output of the potentiometers were then connected to a summing amplifier (figure 9). A summing amplifier is necessary as this combines the three filter outputs into one output. The 10kΩ resistor and the 4.7kΩ resistor are used to determine the gain of the filters. The calculation below shows how the gain was determined. The 20kΩ potentiometer determined the range of the gain, its gives a +/- 6dB range.

$$20\log\left(\frac{10000}{4700}\right) = 6.55dB$$

Originally a 12dB gain was going to be used but this would make the output voltage 10V, this was not feasible as the output voltage could not be more than 5V due to the AVR input not being able to read anything higher than 5V, it is for this reason the gain is only 6dB.

In figure 4 SW1 is used to switch between an equalised signal and a normal signal, this switch needed to be at this part of the circuit because if it was at the beginning then it would cause a short circuit.

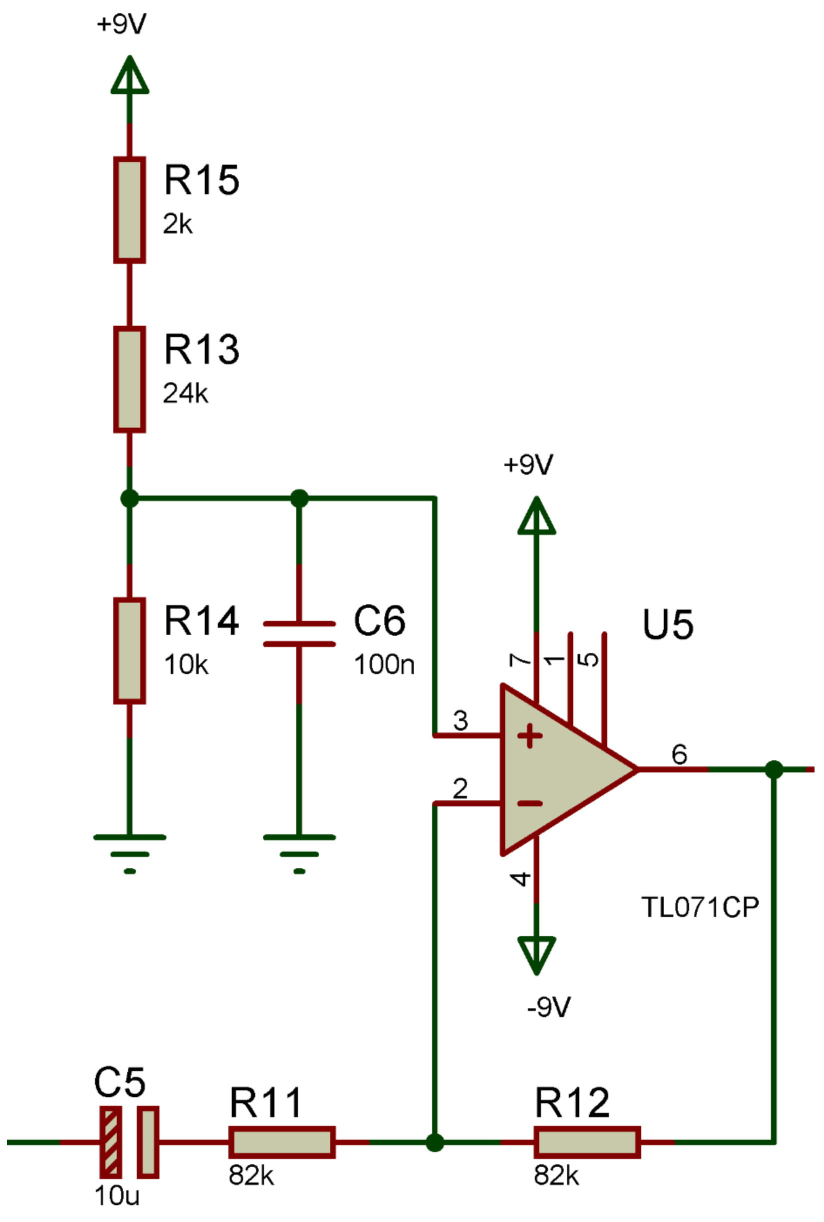


Figure 10. Potential divider

Thereafter the output of the summing amp goes into another op amp which has a potential divider that lifts the signal by 2.5V. This is done because AVR cannot read a negative voltage, lifting the signal by 2.5V makes the centre of the signal become 2.5V instead of 0V. The bottom 82kΩ along with the 10uF capacitor are used to create a high pass filter that allows any frequency through. The resistors R13, R14 and R15 were determined by the calculation below. The 100nF capacitor is a decoupling capacitor.

$$9 \times \left(\frac{10}{36}\right) = 2.5V$$

From this it shows that at 10kΩ, and 26kΩ were needed to get 2.5V. however a 26kΩ was not available so a 24kΩ and a 2kΩ resistor were used in place.

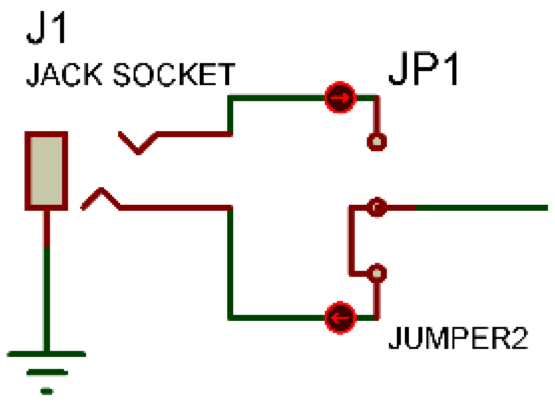


Figure 11. Headphone jack and Jumper

Figure 11 shows the headphone jack connector and a jumper; the jumper is used to select between left audio and right audio channel. There are two of these one at the beginning of the circuit and one at the end.

The circuit diagram (Figure 12) shows the entirety of the equaliser circuit design, the audio signal comes in from either left or right audio channel, and then depending on the position of SW1 the signal will be equalised or unequalised. If it is equalised the signal goes through all three filters and then into the summing amp, the potentiometers can change the gain of these filters, afterwards the signal passed thought the final op amp which centres the signal around 2.5V instead of 0V, this signal is then passed into the output.

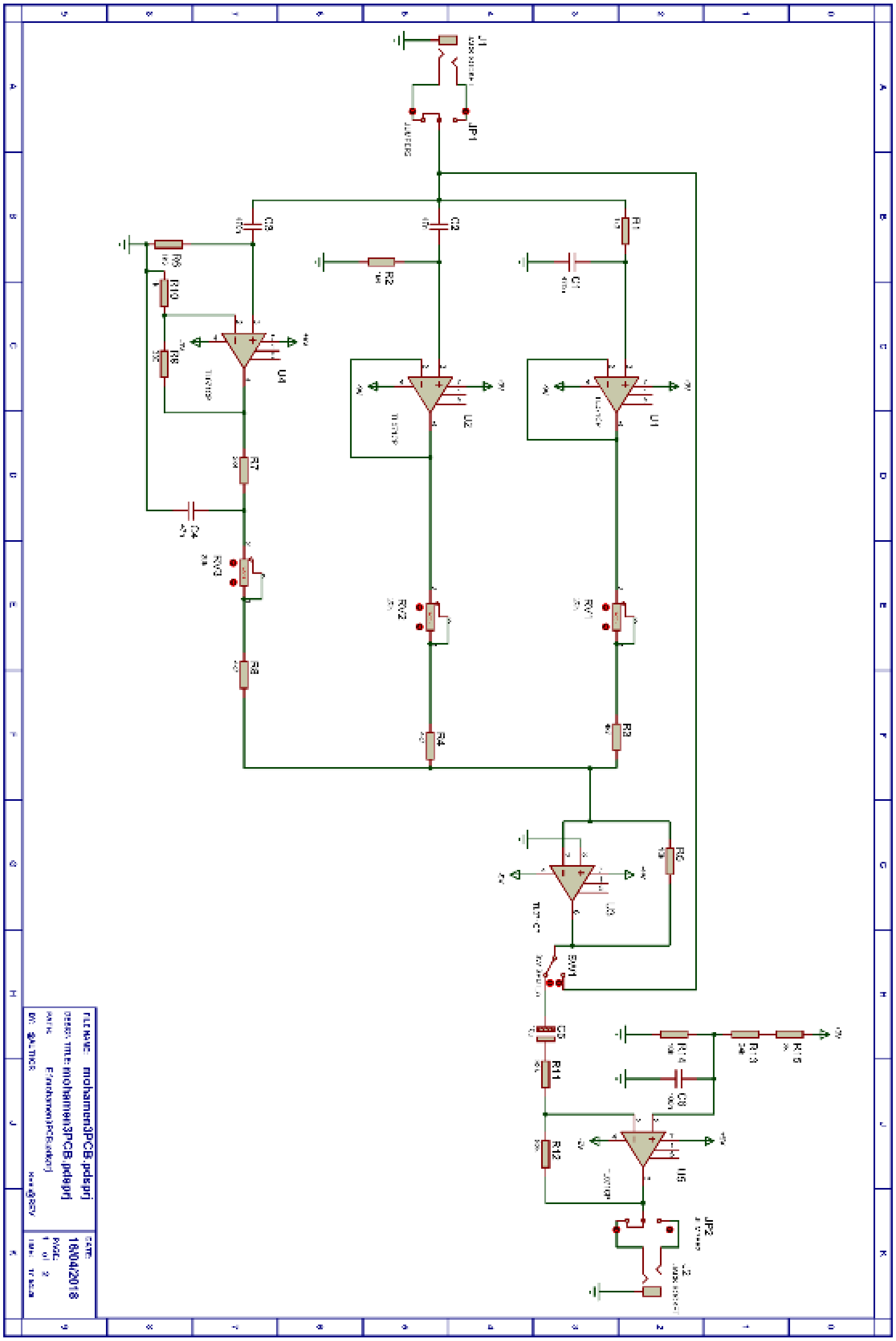


Figure 12. Equaliser Circuit diagram

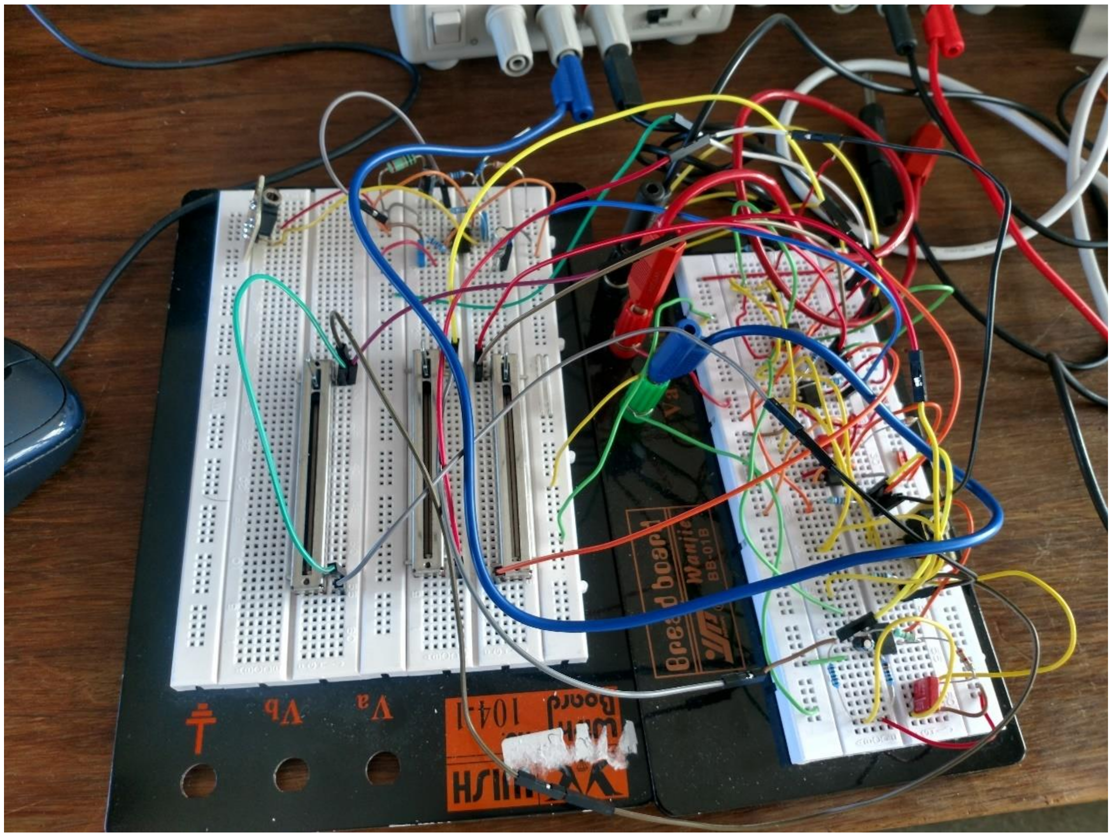
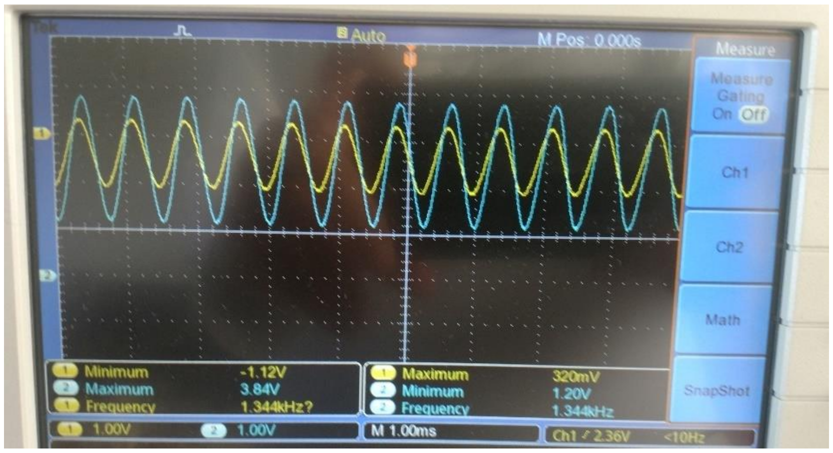


Figure 13. Breadboarded Circuit

The circuit was built on breadboard (Figure 13) to check if it gave the same results as the simulation, a dual 9V power supply was used to power the circuit and 100kΩ slider potentiometer were used as a testing component. After building a circuit that only had the three filters and the summing amplifier (right breadboard), it was tested. It was tested by using a signal generator as the input and then connecting both the input and output to an oscilloscope (figure 14). After confirming this worked the potential divider was added as well. As figure 14 shows the input (yellow signal) was centred around 0V but the output (blue signal) shows centring of 2.5V.



```
1 Minimum      -1.12V       1 Maximum     320mV
2 Maximum      3.84V        2 Minimum     1.20V
1 Frequency    1.344kHz?    2 Frequency   1.344kHz
1 1.00V        2 1.00V      M 1.00ms      Ch1 ∫ 2.36V   <10Hz
```

Figure 14. Oscilloscope

After the testing on breadboard was complete, the circuit was designed as a PCB, this was designed in proteas as well. Figure 15 shows the equaliser PCB; the red wire means that this wire will be a physical wire. The holes in the corners are used to mount the PCB onto the case. The switches and power are wired into the circuit, so junction boxes are used to achieve this. Two of these circuits are printed, one for left audio signal and the other for the right audio signal. These two circuits are connected to the PCB shown in figure 16.

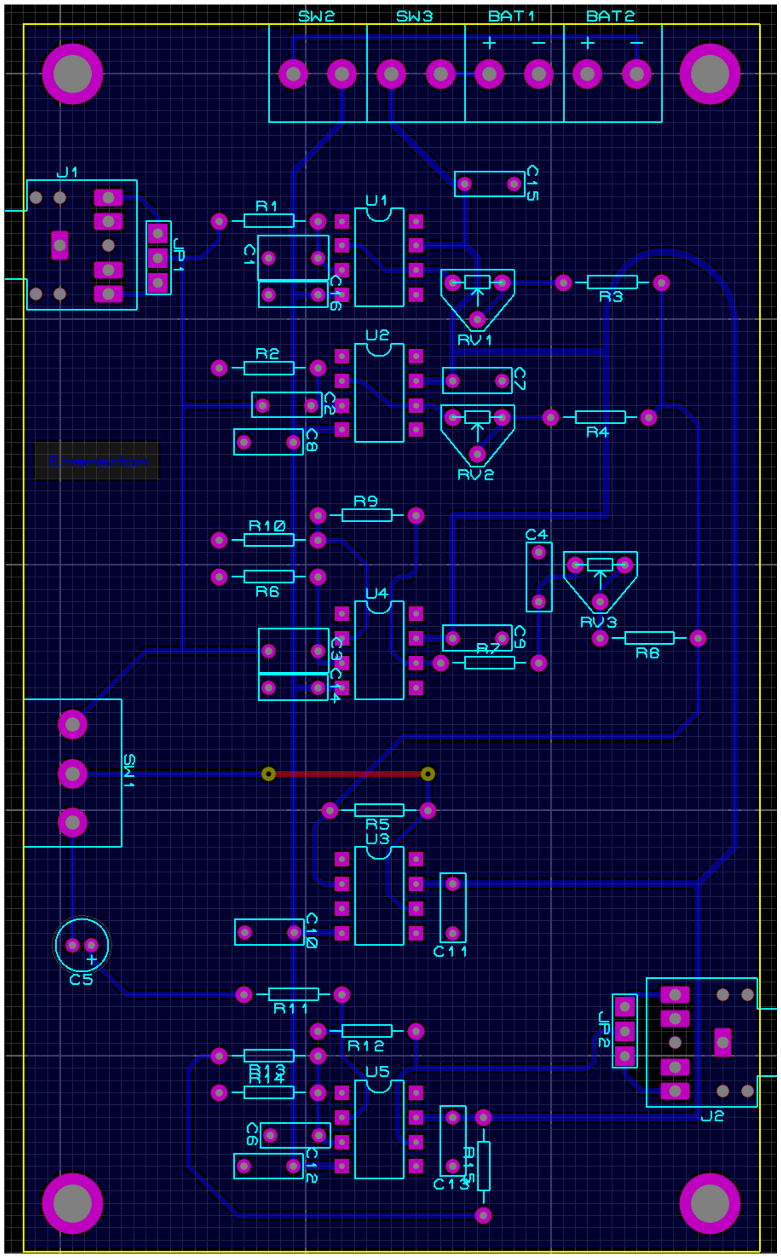


Figure 15. Equaliser PCB

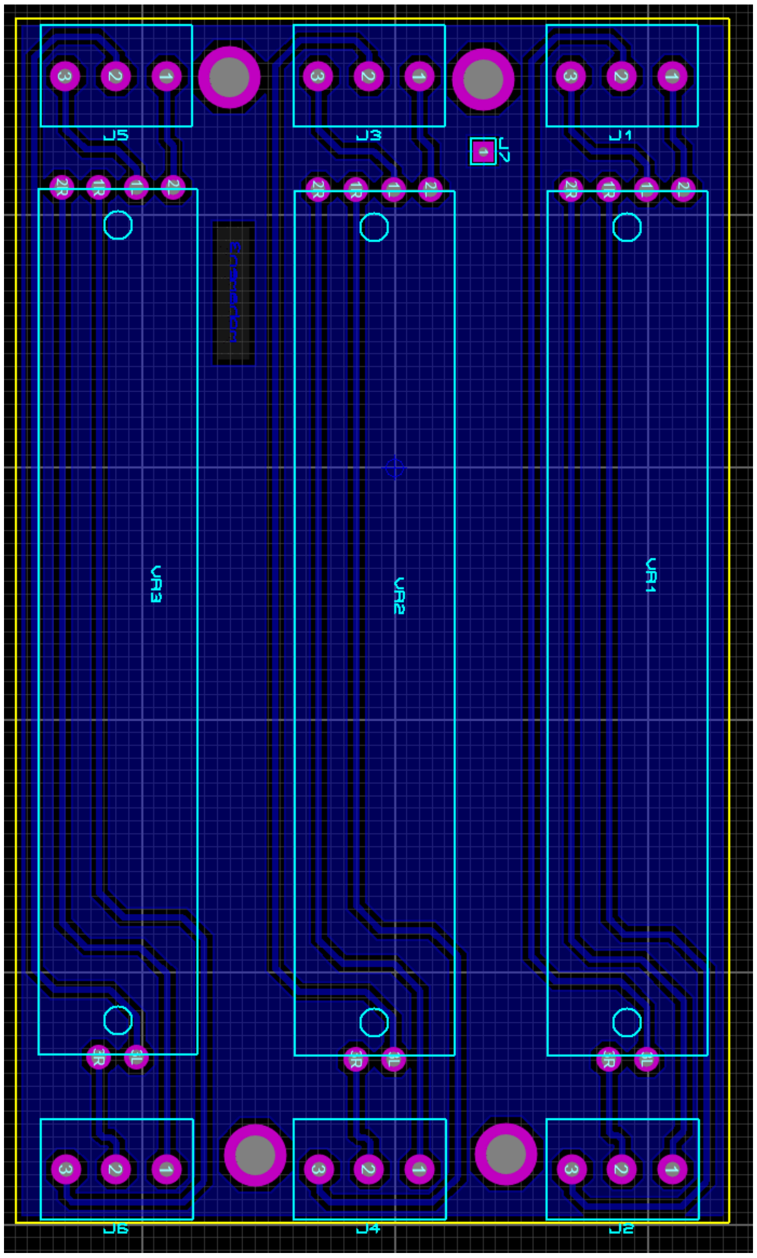


Figure 16. Potentiometer PCB

These three PCBs are connected to each other, via the junction boxes on the potentiometer PCB, wires come out of the equaliser PCB from RV1, RV2 and RV3, these wires go into their respective boxes on the potentiometer PCB. The result of this is shown in figure 17.

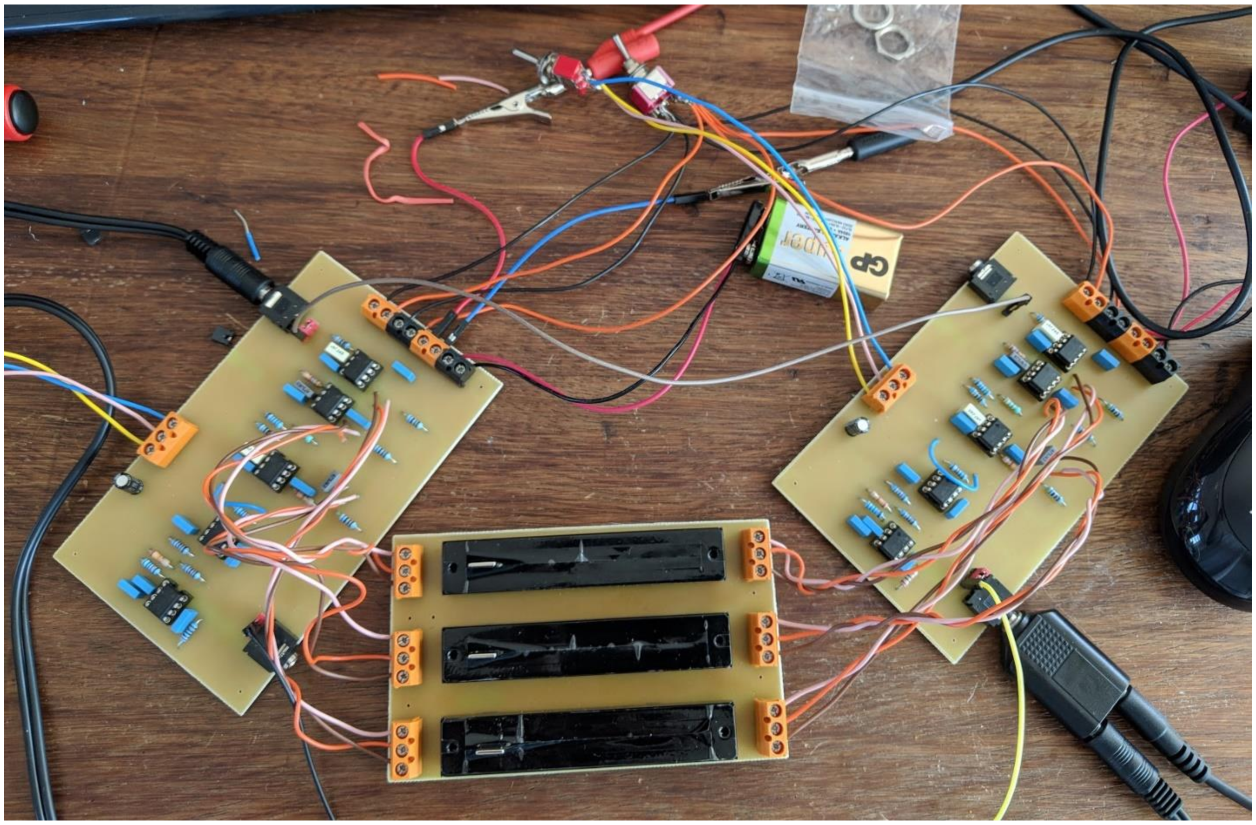


Figure 17. Final Equaliser PCB

This figure shows how the equaliser looks on circuit, the board on the left controls the left audio and the board on the right controls right audio, these boards both connect to the potentiometers (more info in appendix B). These slider potentiometers are dual gang which means they are perfect for stereo audio. Also shown is the splitter where one output goes into the speaker and the other goes into the AVR. This entire circuit is powered by a 9V power supply.

AVR

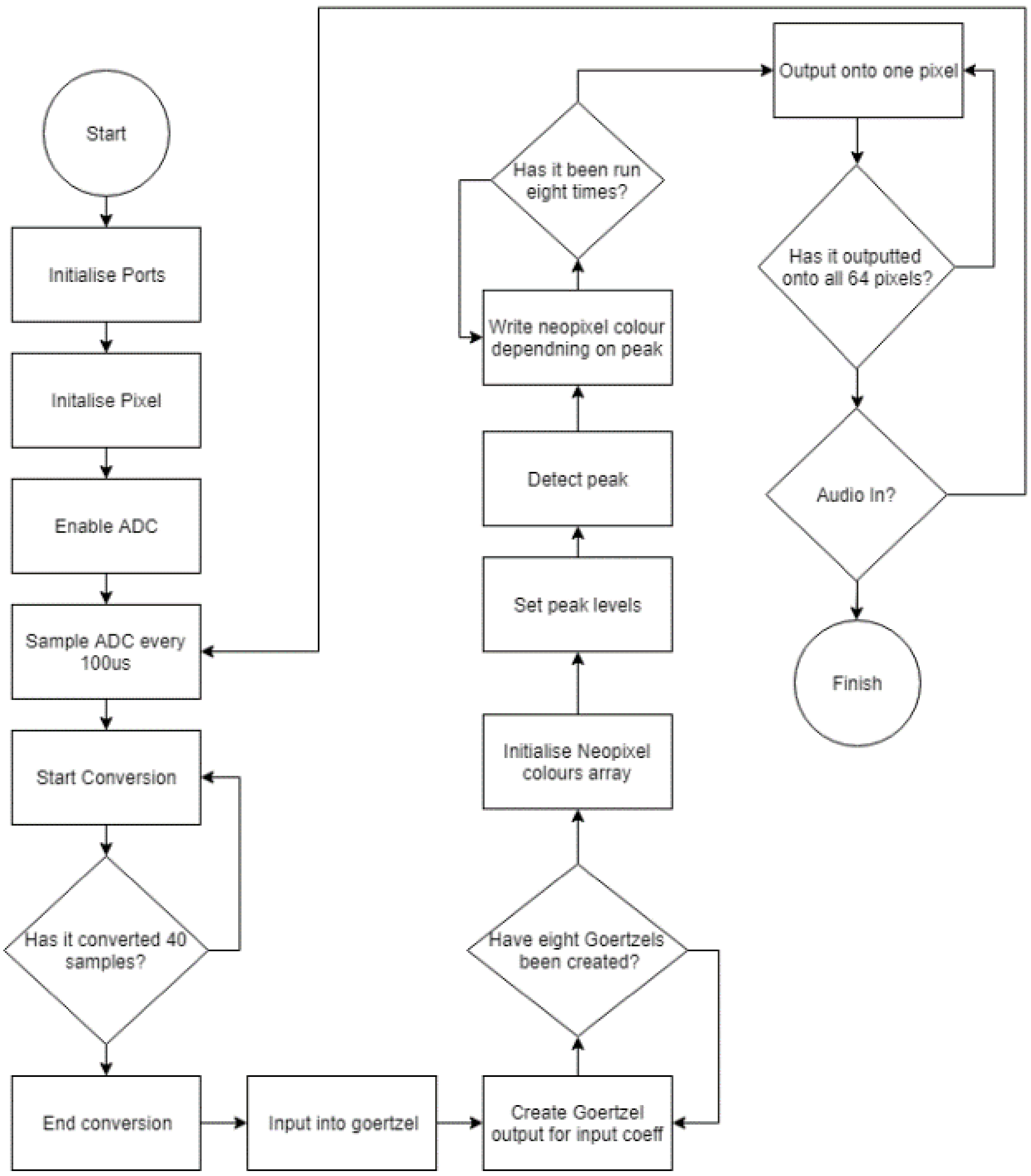
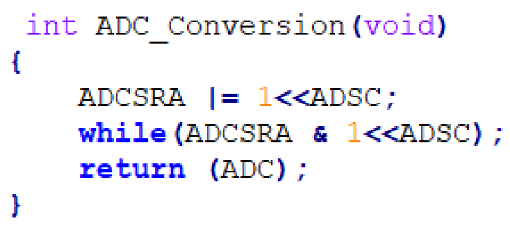

Start
Initialise Ports
Initalise Pixel
Enable ADC
Sample ADC every 100us
Start Conversion
Has it converted 40 samples?
End conversion
Input into goertzel
Create Goertzel output for input coeff
Have eight Goertzels been created?
Initialise Neopixel colours array
Set peak levels
Detect peak
Write neopixel colour dependning on peak
Has it been run eight times?
Output onto one pixel
Has it outputted onto all 64 pixels?
Audio In?
Finish

Figure 18. Code Flow Chart

The second part of the project was to create eight band pass filters using the Goertzel algorithm, this was achieved using an AVR micro controller and the Atmel AVR programme [10]. The AVR board is shown below with the schematic being available in appendix A. Figure 18 above shows a flow chart which is how the code works. To start with the ports being used and ADC needs to be initialised. Then the ADC runs for 40 samples and this is entered in to the Goertzel filter. Then eight Goertzels are created with the eight different inputs. The eight outputs are read and then depending on the output of them the value for the neopixels are written and they are written one by one.

To start with the clock frequency was set to 20MHz and the ADC Conversion function was added, this function is used to convert analogue signal to digital.

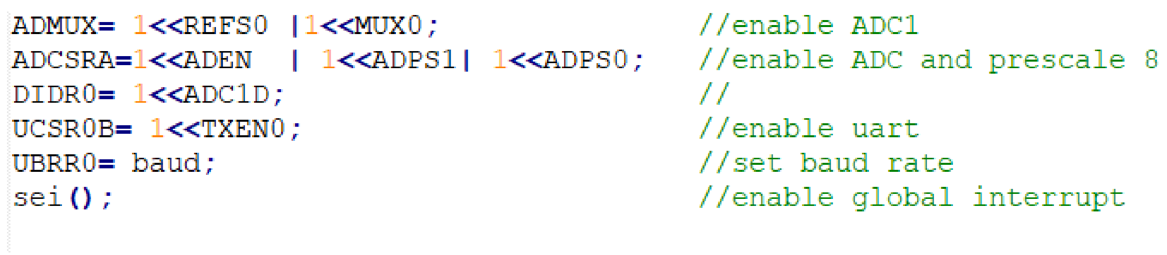


```
 int ADC_Conversion(void)
{
    ADCSRA |= 1<<ADSC;
    while(ADCSRA & 1<<ADSC);
    return (ADC);
}
```

Figure 19. ADC Conversion

This function is necessary as it converts the analgoue audio signal into a readable digital signal.

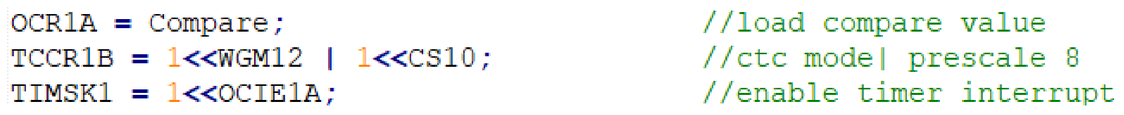
Thereafter initialisation of certain ports needed to be added in the main (see below).



```
ADMUX= 1<<REFS0 |1<<MUX0;                //enable ADC1
ADCSRA=1<<ADEN  | 1<<ADPS1| 1<<ADPS0;    //enable ADC and prescale 8
DIDR0= 1<<ADC1D;                         //
UCSR0B= 1<<TXEN0;                        //enable uart
UBRR0= baud;                             //set baud rate
sei();                                   //enable global interrupt
```

Figure 20. ADC and UART ports

The first three lines in figure 20 are used to enable the ADC, and then to pre-scale it by 8 this means the ADC clock frequency is at 250kHz as this is 20MHz. It takes 13 clock cycles to do one conversion. The next two lines are used to enable the serial monitor, this is used to monitor the Goertzel output. The last line is to enable interrupts.



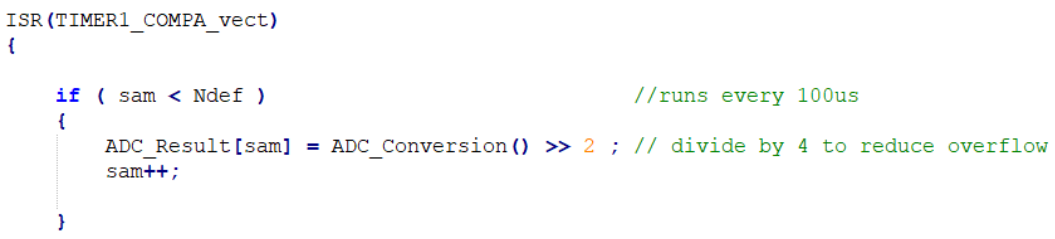
```
OCR1A = Compare;                         //load compare value
TCCR1B = 1<<WGM12 | 1<<CS10;             //ctc mode| prescale 8
TIMSK1 = 1<<OCIE1A;                      //enable timer interrupt
```

Figure 21. Timer ports

The timer interrupt is used to activate ADC conversion to get 100 samples every 100us. This is done by using compare timer mode on the AVR. The compare value is what is used to get 100us, it is calculated via the equation below.

$$Compare = \left(\frac{t \times f_{clock}}{prescale}\right) - 1$$

The sampling rate is 10000Hz and to get the time the eqution $t = \frac{1}{f}$. This make t =100us. F_clock is equal to 2000000MHz and 1<<cs10 means there is no prescale. Therefore the compare value is 1999.



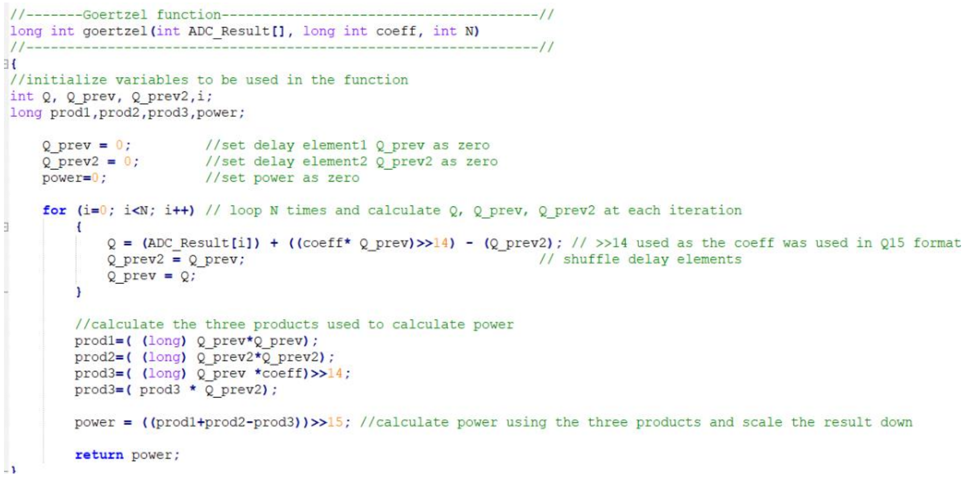
```
ISR(TIMER1_COMPA_vect)
{

    if ( sam < Ndef )                               //runs every 100us
    {
        ADC_Result[sam] = ADC_Conversion() >> 2 ; // divide by 4 to reduce overflow
        sam++;

    }
```

Figure 22. Timer interrupt

Figure 22 activates every 100us, this was set previously using the compare value. It runs the ADC conversion 40 times to create 40 samples that act as the input for the Goertzel function (read on). Ndef is set at 40 in the beginning. The input is scaled down to prevent input overflow, as if the input is too high the filter will not work properly.



```
//-------Goertzel function---------------------------------------//
long int goertzel(int ADC_Result[], long int coeff, int N)
//---------------------------------------------------------------//
{
//initialize variables to be used in the function
int Q, Q_prev, Q_prev2,i;
long prod1,prod2,prod3,power;

    Q_prev = 0;          //set delay element1 Q_prev as zero
    Q_prev2 = 0;         //set delay element2 Q_prev2 as zero
    power=0;             //set power as zero

    for (i=0; i<N; i++) // loop N times and calculate Q, Q_prev, Q_prev2 at each iteration
        {
            Q = (ADC_Result[i]) + ((coeff* Q_prev)>>14) - (Q_prev2); // >>14 used as the coeff was used in Q15 format
            Q_prev2 = Q_prev;                                        // shuffle delay elements
            Q_prev = Q;
        }

        //calculate the three products used to calculate power
        prod1=( (long) Q_prev*Q_prev);
        prod2=( (long) Q_prev2*Q_prev2);
        prod3=( (long) Q_prev *coeff)>>14;
        prod3=( prod3 * Q_prev2);

        power = ((prod1+prod2-prod3))>>15; //calculate power using the three products and scale the result down

        return power;
}
```

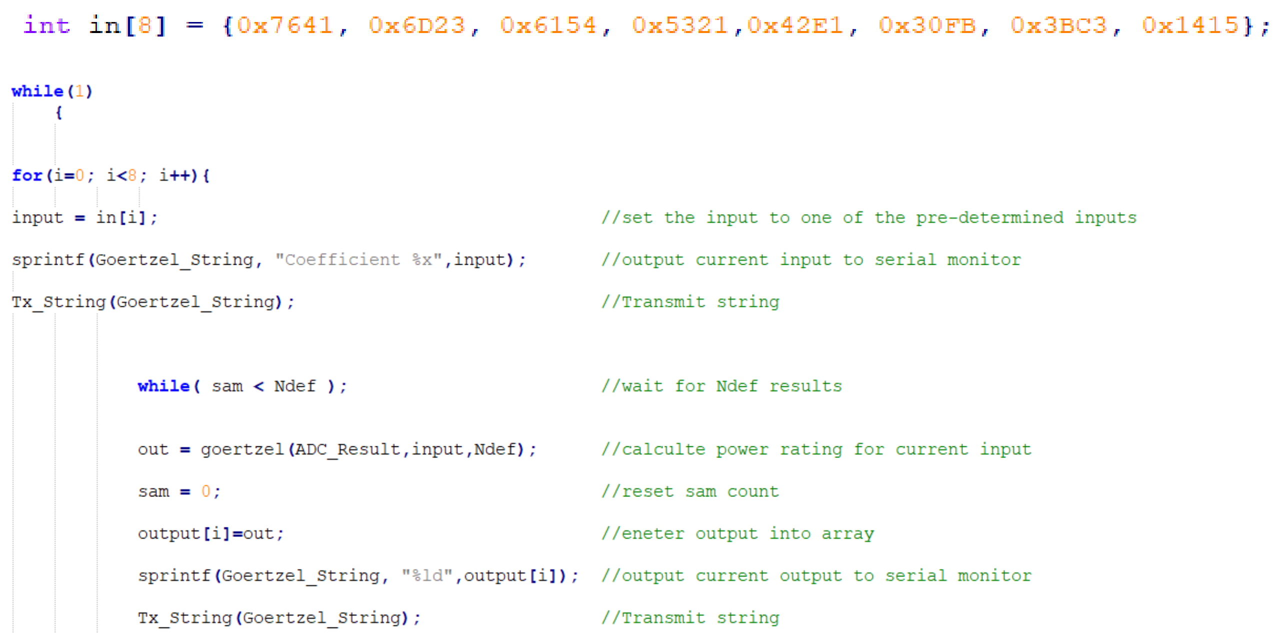
Figure 23. Goertzel function

Figure 23 shows the Goertzel function, this function was written by Omayma Said [12], it has been adapted for us in the project. This filter has three inputs, the first being the 40 ADC results which are contained in an array, the second being the input frequency which is pre-calculated as a Q15 coefficient via the calculation below and the last being the number of samples which is determined by Ndef.

$$f(dec) = 2\cos\left(2\pi \times \left(\frac{f\_tone}{fs}\right)\right)$$

Fs is 10000Hz, and for examples f_tone could be 1250Hz. This would give a decimal number which would then be converted into Q15 format.

The output of the goertzel function is also scaled down as it needs to be lower than 1023, due to the neopixel not being able to read anything higher than that.



```
 int in[8] = {0x7641, 0x6D23, 0x6154, 0x5321,0x42E1, 0x30FB, 0x3BC3, 0x1415};

while(1)
    {

for(i=0; i<8; i++){

input = in[i];                                          //set the input to one of the pre-determined inputs

sprintf(Goertzel_String, "Coefficient %x",input);       //output current input to serial monitor

Tx_String(Goertzel_String);                             //Transmit string


            while( sam < Ndef );                        //wait for Ndef results

            out = goertzel(ADC_Result,input,Ndef);      //calculte power rating for current input

            sam = 0;                                    //reset sam count

            output[i]=out;                              //eneter output into array

            sprintf(Goertzel_String, "%ld",output[i]);  //output current output to serial monitor

            Tx_String(Goertzel_String);                 //Transmit string
```

Figure 24. Goertzel loop

Figure 24 above demonstrates how the eight goertzel outputs are created. The first line means that this takes place in the while loop. The for loop is set to run eight times for the eight different inputs. The input coefficient is set depending on the current number for i. The while loop is there to prevent the current ADC results being overwritten by new ADC results, once this has been done a goertzel output is created and the output is stored into an array, this output is transmitted and the for loop then increments by one until eight geortzel outputs have been created. These outputs are then inputted onto the neopixel display.

Neopixel

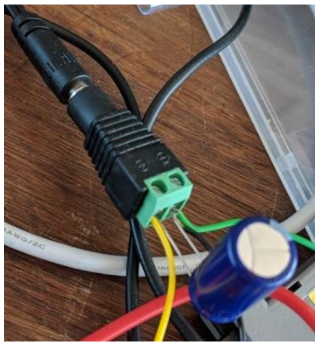
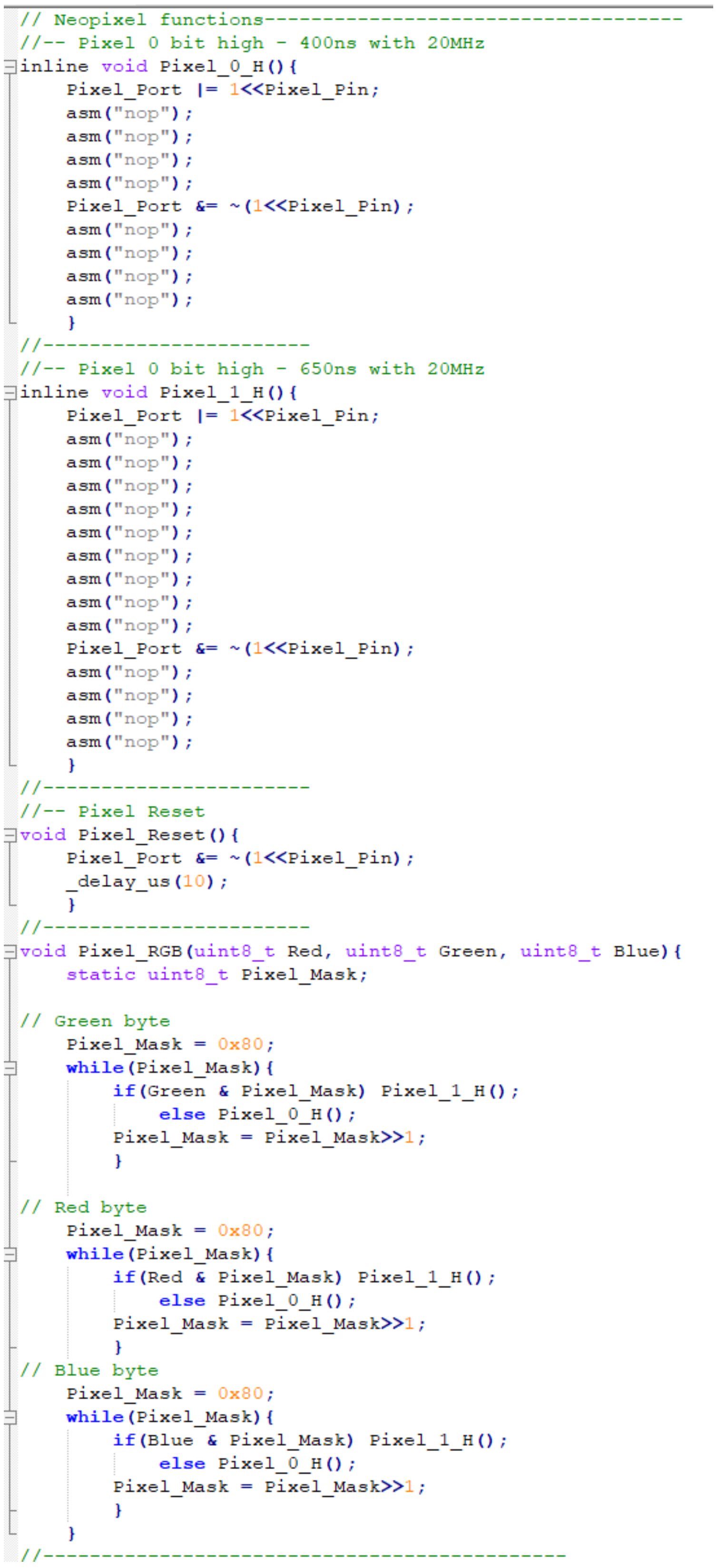


Figure 25. Neopixel power

The last part of the project was to output onto a neopixel display, first the neopixel needed to be connected. The figure on the left shows that a 5V 2A power supply was used to power the neopixel, this is because the neopixel needs a constant 5V DC supply to work and as there are 64 LEDs 2A current is used. A 1000uF capacitor is connected to between the positive and the negative, this prevents the onrush of current form damaging the pixels [12]. and the ground of the neopixel is also connected to the AVR ground.

A 390Ω resistor is connected between the data pin and the AVR to ensure the first pixel is not damaged by the voltage spikes [12]. Neopixel functions needed to be added. Figure 26 shows the functions; the first function determines how often the neopixel lights up for. The pixel reset function is self-explanatory and the pixel_RGB function is needed to set the colour of the neopixel.

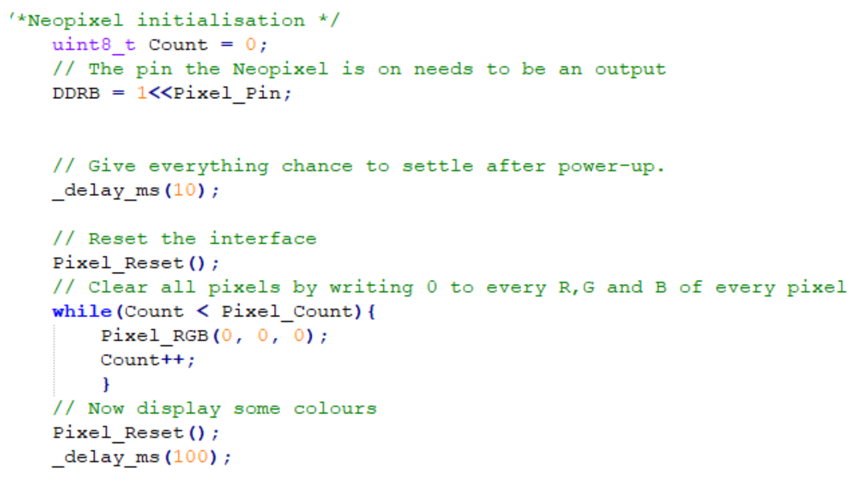


```
// Neopixel functions------------------------------------
//-- Pixel 0 bit high - 400ns with 20MHz
inline void Pixel_0_H(){
    Pixel_Port |= 1<<Pixel_Pin;
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    Pixel_Port &= ~(1<<Pixel_Pin);
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    }
//-----------------------
//-- Pixel 0 bit high - 650ns with 20MHz
inline void Pixel_1_H(){
    Pixel_Port |= 1<<Pixel_Pin;
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    Pixel_Port &= ~(1<<Pixel_Pin);
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    }
//-----------------------
//-- Pixel Reset
void Pixel_Reset(){
    Pixel_Port &= ~(1<<Pixel_Pin);
    _delay_us(10);
    }
//-----------------------
void Pixel_RGB(uint8_t Red, uint8_t Green, uint8_t Blue){
    static uint8_t Pixel_Mask;

// Green byte
    Pixel_Mask = 0x80;
    while(Pixel_Mask){
        if(Green & Pixel_Mask) Pixel_1_H();
            else Pixel_0_H();
        Pixel_Mask = Pixel_Mask>>1;
        }

// Red byte
    Pixel_Mask = 0x80;
    while(Pixel_Mask){
        if(Red & Pixel_Mask) Pixel_1_H();
            else Pixel_0_H();
        Pixel_Mask = Pixel_Mask>>1;
        }
// Blue byte
    Pixel_Mask = 0x80;
    while(Pixel_Mask){
        if(Blue & Pixel_Mask) Pixel_1_H();
            else Pixel_0_H();
        Pixel_Mask = Pixel_Mask>>1;
        }
    }
//---------------------------------------------
```

Figure 26. Neopixel functions



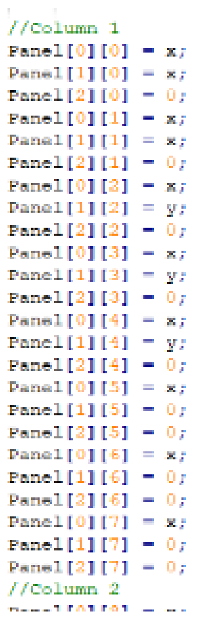
```
'*Neopixel initialisation */
    uint8_t Count = 0;
    // The pin the Neopixel is on needs to be an output
    DDRB = 1<<Pixel_Pin;


    // Give everything chance to settle after power-up.
    _delay_ms(10);

    // Reset the interface
    Pixel_Reset();
    // Clear all pixels by writing 0 to every R,G and B of every pixel
    while(Count < Pixel_Count){
        Pixel_RGB(0, 0, 0);
        Count++;
        }
    // Now display some colours
    Pixel_Reset();
    _delay_ms(100);
```

Figure 27. Neopixel Initialisation

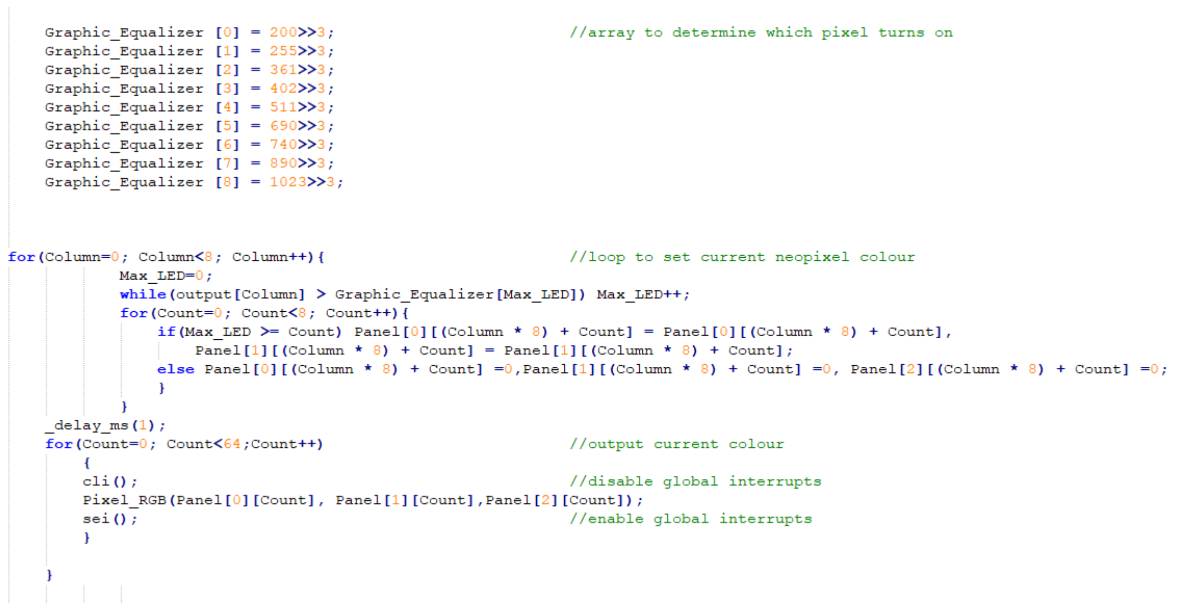
Above shows the initialisation of the neopixel, firstly the neopixel is set as an output on port B of the AVR, then the pixel is reset to remove any old data.



```
//Column 1
Panel[0][0] = x;
Panel[1][0] = x;
Panel[2][0] = 0;
Panel[0][1] = x;
Panel[1][1] = x;
Panel[2][1] = 0;
Panel[0][2] = x;
Panel[1][2] = y;
Panel[2][2] = 0;
Panel[0][3] = x;
Panel[1][3] = y;
Panel[2][3] = 0;
Panel[0][4] = x;
Panel[1][4] = y;
Panel[2][4] = 0;
Panel[0][5] = x;
Panel[1][5] = 0;
Panel[2][5] = 0;
Panel[0][6] = x;
Panel[1][6] = 0;
Panel[2][6] = 0;
Panel[0][7] = x;
Panel[1][7] = 0;
Panel[2][7] = 0;
//Column 2
```

Figure 28 shows part of an array which is 3x64 long. It contains the brightness setting for each pixel, the number on the left corresponds to RGB and the right number shows what pixel is being set.

Figure 28. Colour setting code



```
    Graphic_Equalizer [0] = 200>>3;                   //array to determine which pixel turns on
    Graphic_Equalizer [1] = 255>>3;
    Graphic_Equalizer [2] = 361>>3;
    Graphic_Equalizer [3] = 402>>3;
    Graphic_Equalizer [4] = 511>>3;
    Graphic_Equalizer [5] = 690>>3;
    Graphic_Equalizer [6] = 740>>3;
    Graphic_Equalizer [7] = 890>>3;
    Graphic_Equalizer [8] = 1023>>3;


for(Column=0; Column<8; Column++){                    //loop to set current neopixel colour
            Max_LED=0;
            while(output[Column] > Graphic_Equalizer[Max_LED]) Max_LED++;
            for(Count=0; Count<8; Count++){
                if(Max_LED >= Count) Panel[0][(Column * 8) + Count] = Panel[0][(Column * 8) + Count],
                    Panel[1][(Column * 8) + Count] = Panel[1][(Column * 8) + Count];
                else Panel[0][(Column * 8) + Count] =0,Panel[1][(Column * 8) + Count] =0, Panel[2][(Column * 8) + Count] =0;
                }
            }
    _delay_ms(1);
    for(Count=0; Count<64;Count++)                    //output current colour
        {
        cli();                                        //disable global interrupts
        Pixel_RGB(Panel[0][Count], Panel[1][Count],Panel[2][Count]);
        sei();                                        //enable global interrupts
        }

    }
```

Figure 29. Neopixel loop

The graphic equaliser array determines how high the output needs to be to turn a led on, if it's below 200 then the first neopixel will turn on, if its below 1023 than all eight pixels in one column will turn on. The graphic equaliser numbers have all been shifted right by 3 (divided by 8) as the output from the Goertzel was also scaled down to the point where the max output was not more than 110.

The loop is used to set each pixel's colour. If the first output has a power rating of 65 then the loop will leave the first four pixels the same as the panel array setting and the second four pixels are set to zero. After repeating these eight times for each power rating the next loop begins.

This last loop is used to output the settings of the panel array onto the neopixel. Firstly, global interrupts must be disabled otherwise the neopixel would be interrupted by the timer interrupt. After this the first pixel is displayed and then interrupts are enabled again. This runs 64 times for all 64 pixels.

Results

This section shows the results of the project, the equaliser results were gather by having the potentiometers in different positions and then calculating gain by doing the calculation below and then plotting graphs based on the results.

$$Gain(dB) = 20\log\left(\frac{Vout}{Vin}\right)$$

Equaliser

Simulation Results

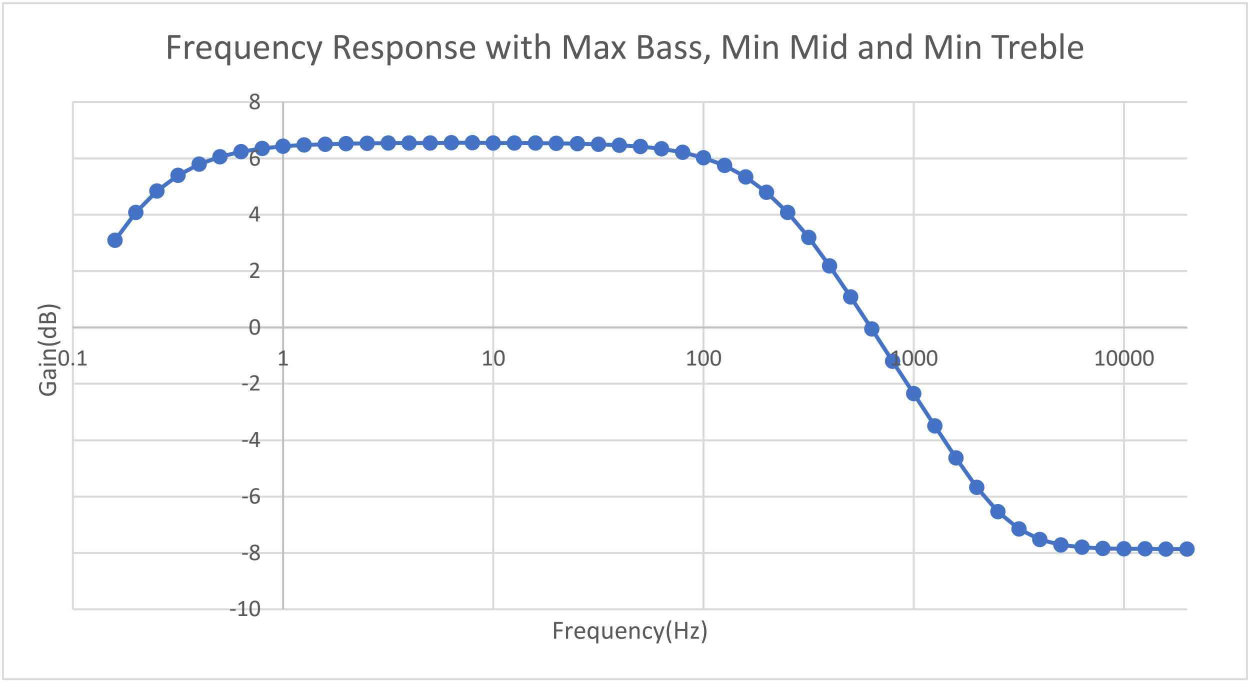


Figure 30. Graph showing Frequency Response with Maximum Bass

Cut-Off 290Hz Max Gain 6.54dB

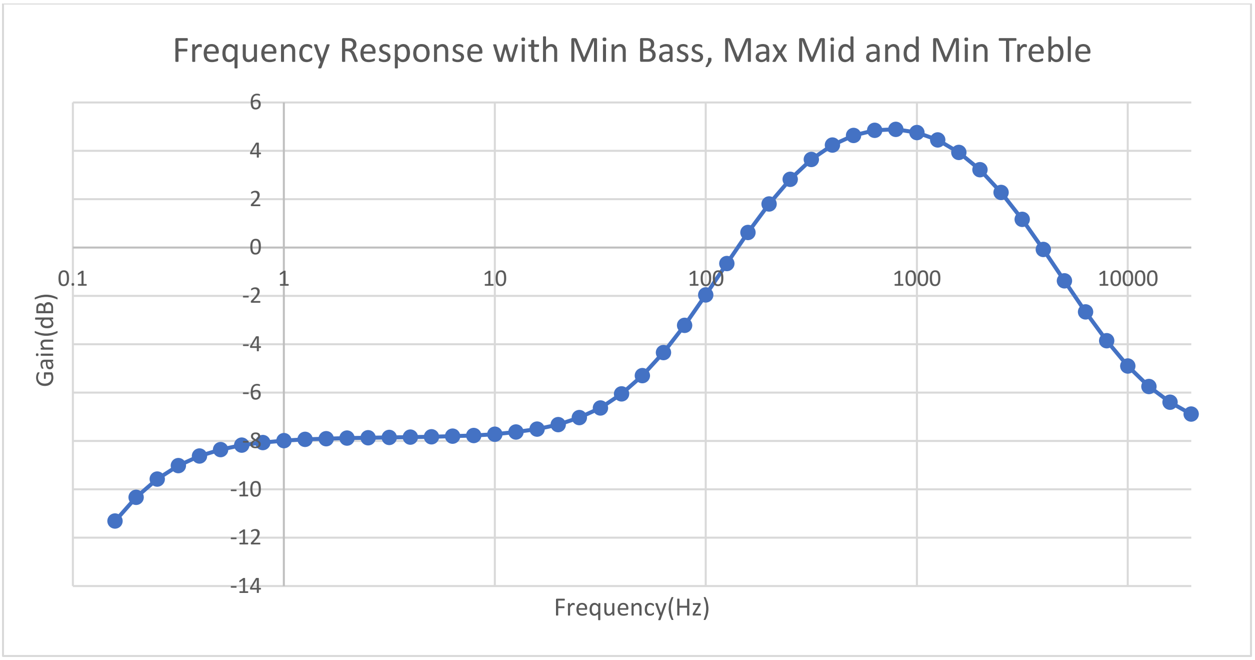


Figure 31. Graph showing Frequency Response with Maximum Mid

High Cut-Off 205Hz Low Cut-Off 2.77kHz Max Gain 4.9dB

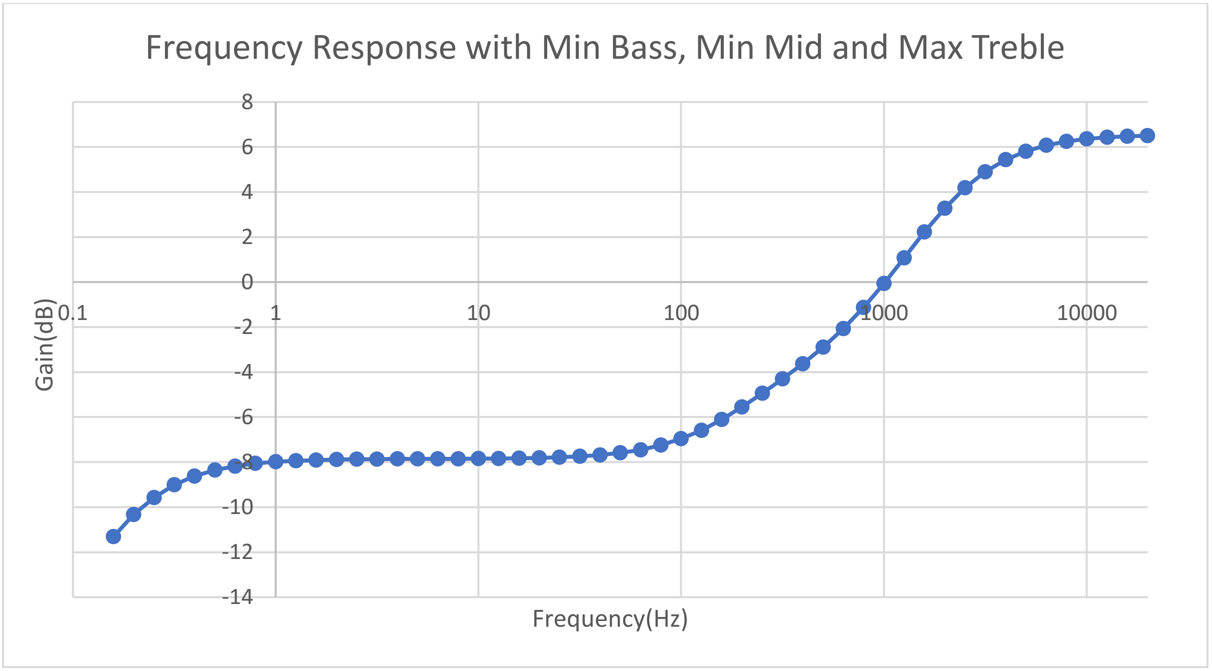


Figure 32. Graph showing Frequency Response with Maximum Treble

Cut-Off 2.15kHz Max Gain 6.54dB

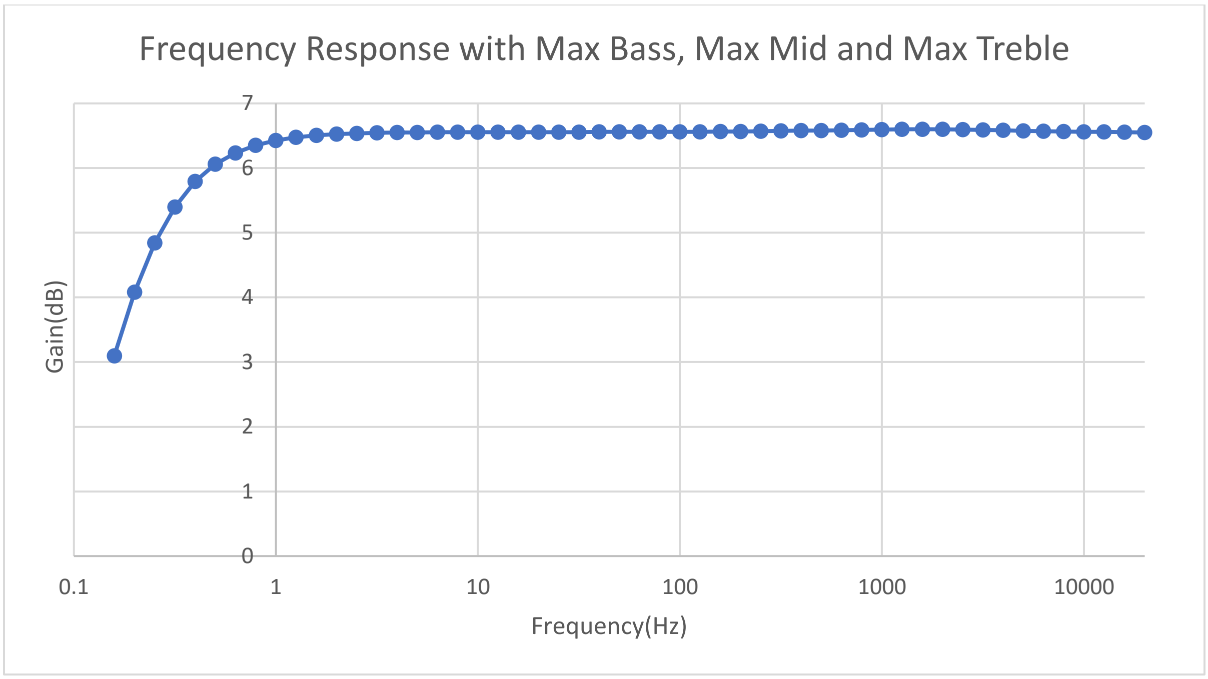


Figure 33. Graph showing Frequency Response with all bands at Maximum Gain

High Cut-Off 0.18Hz Low Cut-Off 470kHz Max Gain 6.6dB

PCB Results

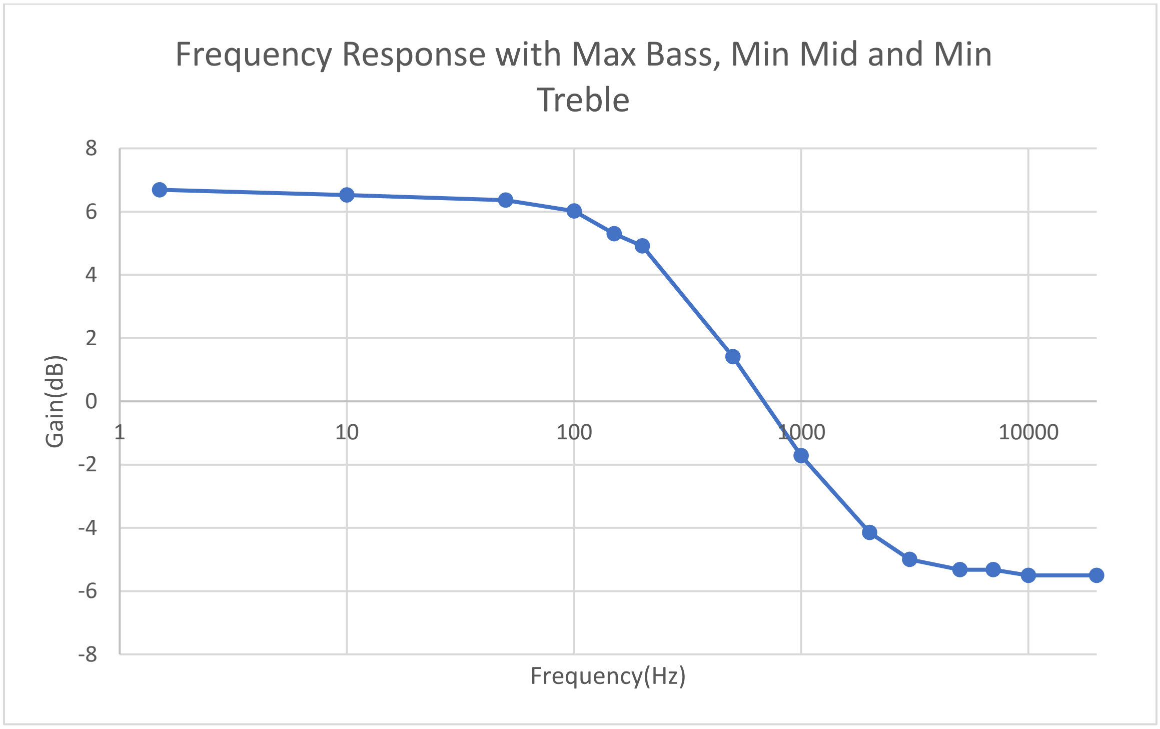


Figure 34. Graph showing Frequency Response with Maximum Bass

Cut-Off 275Hz Max Gain 6.68dB

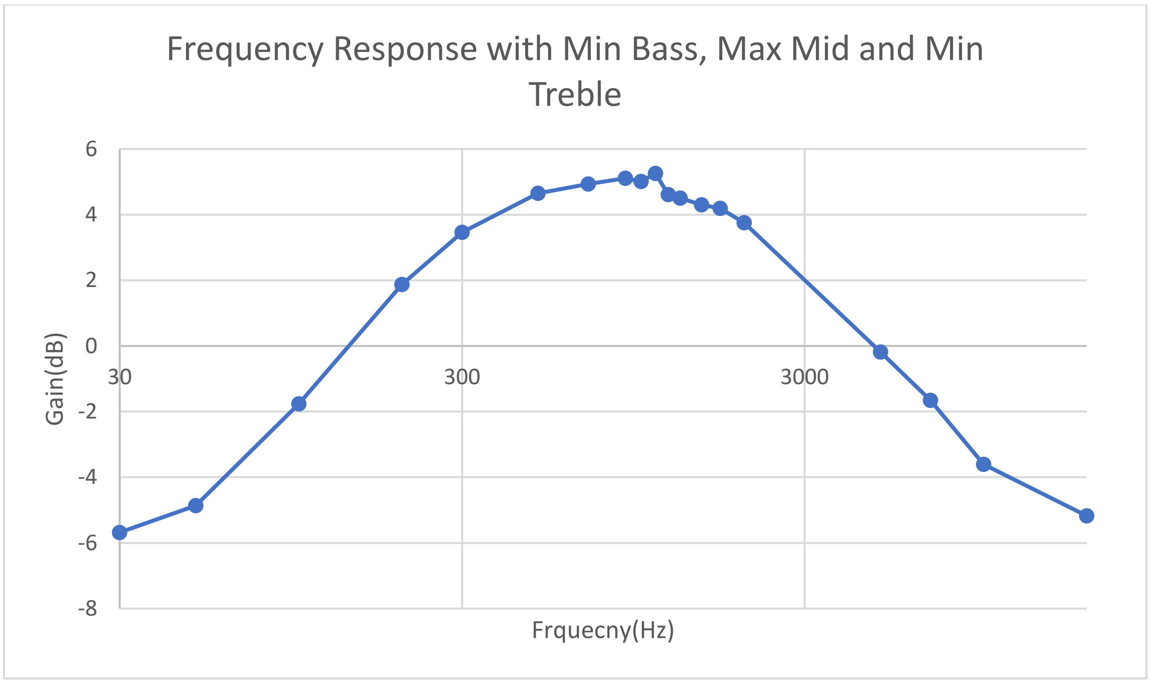


Figure 35. Graph showing Frequency Response with Maximum Mid

High Cut Off 285Hz Low Cut Off 2250Hz Max Gain 5.25dB

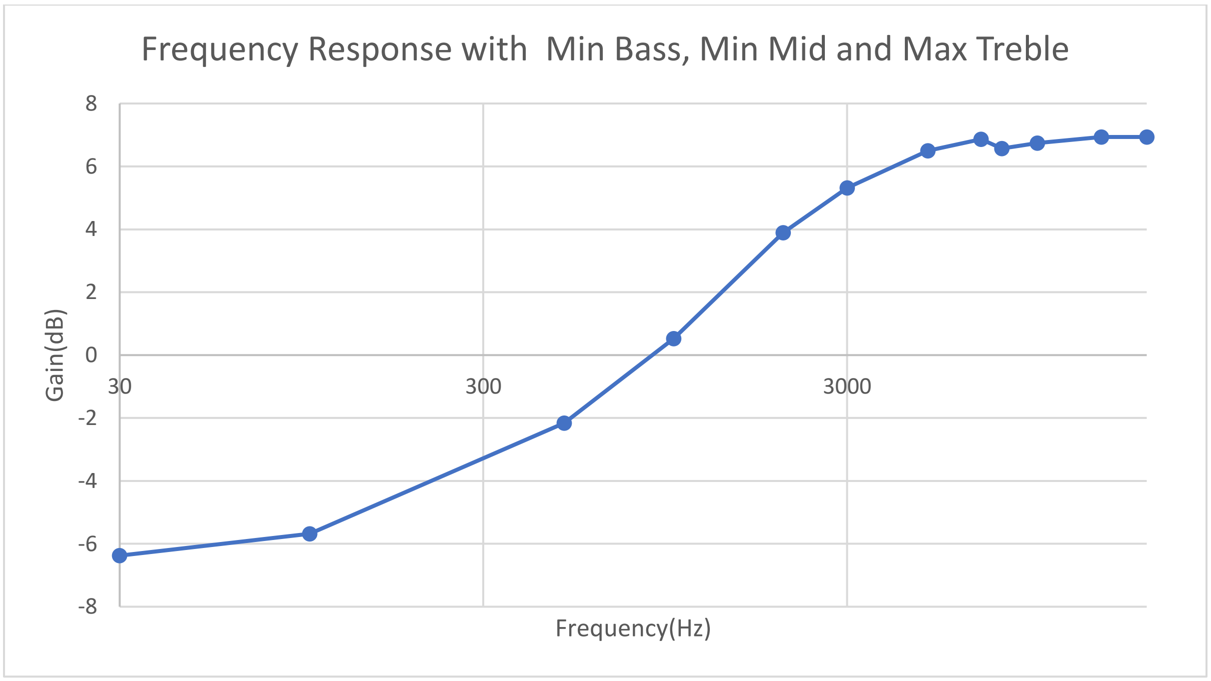


Figure 36. Graph showing Frequency Response with Maximum Treble

Cut Off 2020Hz Max Gain 6.93dB

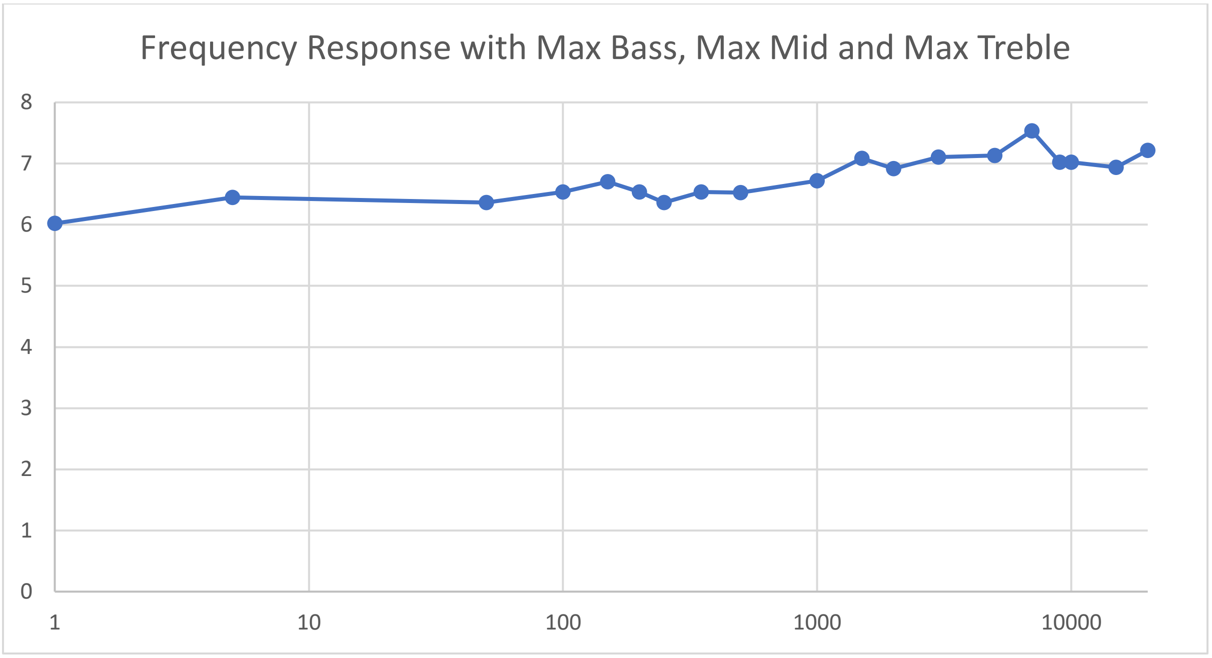


Figure 37. Graph showing Frequency Response with all bands at Maximum Gain

Max Gain 7.53dB

Goertzel

The filter results were gathered by reading the power of the output using the serial monitor and then recording the result that each filter gave. The frequency was increased by 50Hz until enough results were gathered. Below are some of the filter graphs along with a graph that's shows six of them.

Selected Results with a 2V input and 250Hz bandwidth

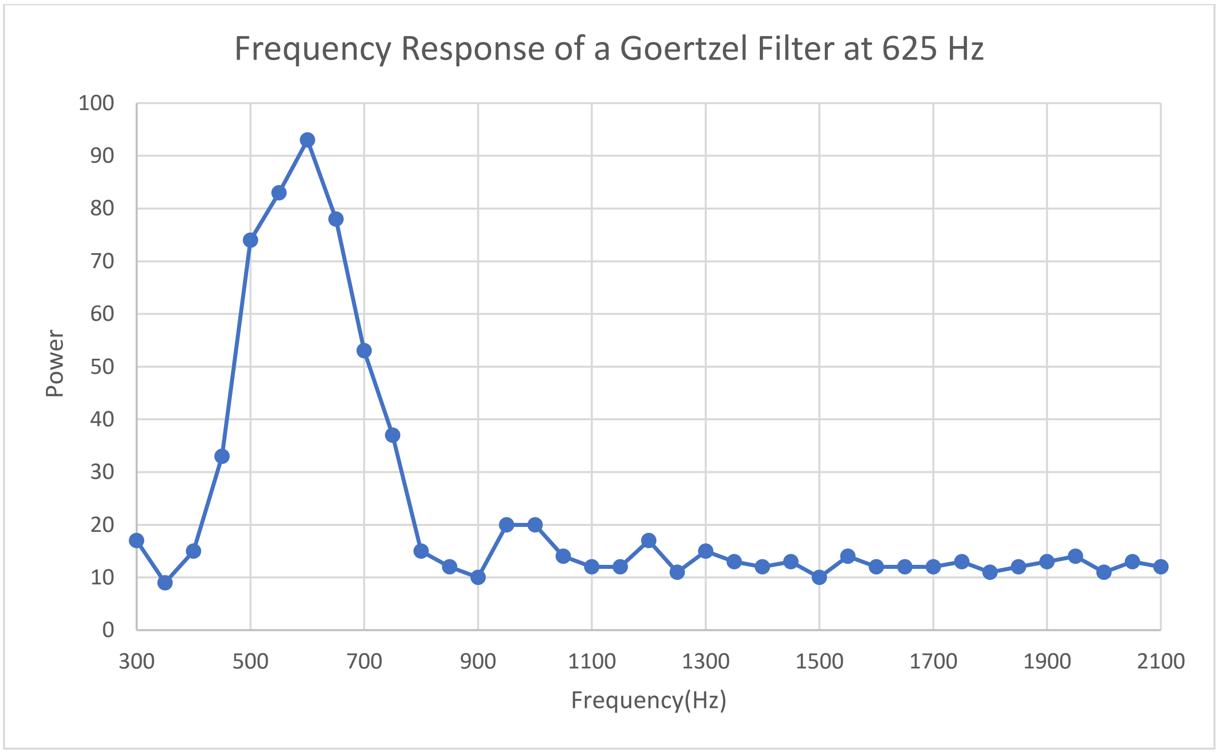


Figure 38. Graph showing Frequency Response of a Goertzel Filter at 625Hz

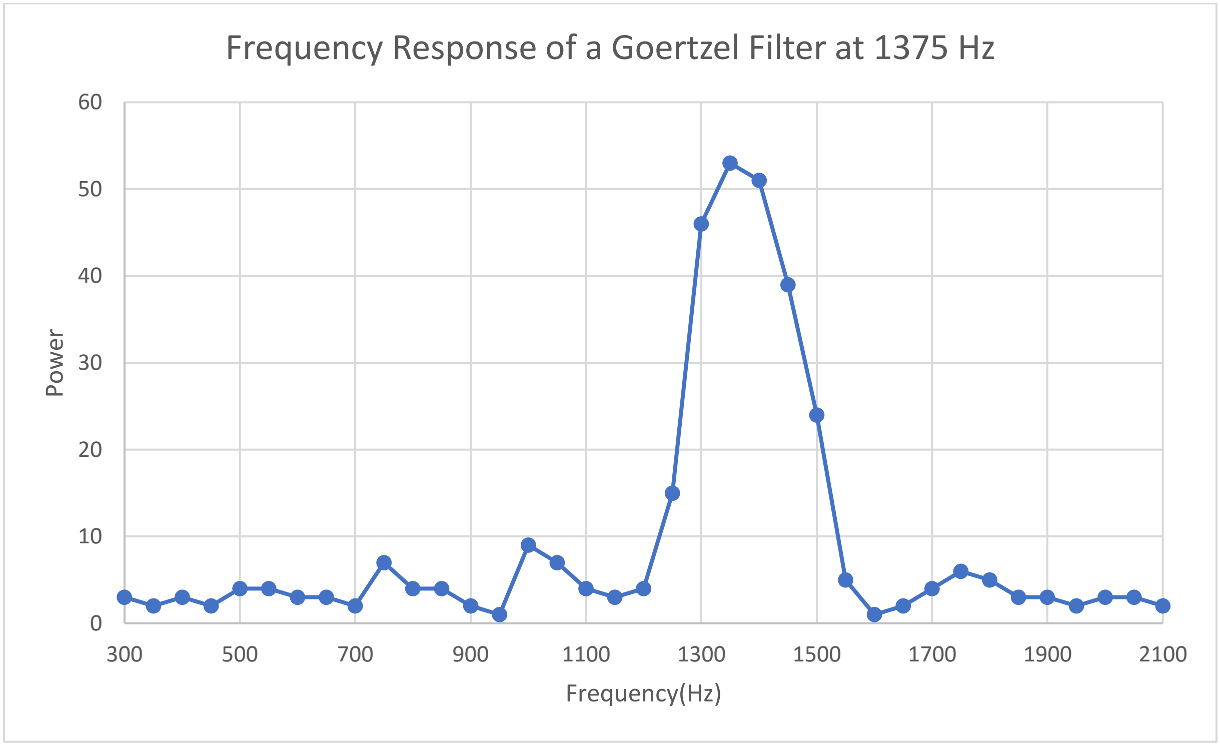


Figure 39. Graph showing Frequency Response of a Goertzel Filter at 1375Hz

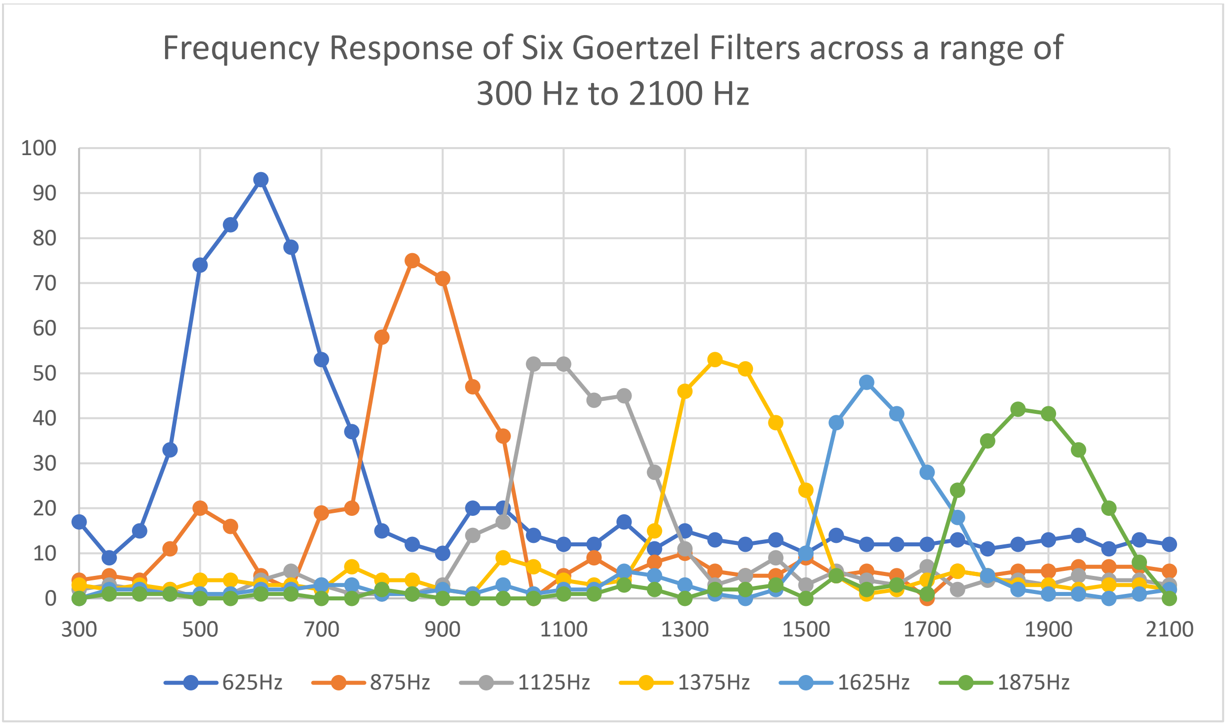


Figure 40.Graph showing Frequency Response of Six Goertzel Filters across a range of 300Hz to 2100Hz

3V Input with a 250Hz bandwidth

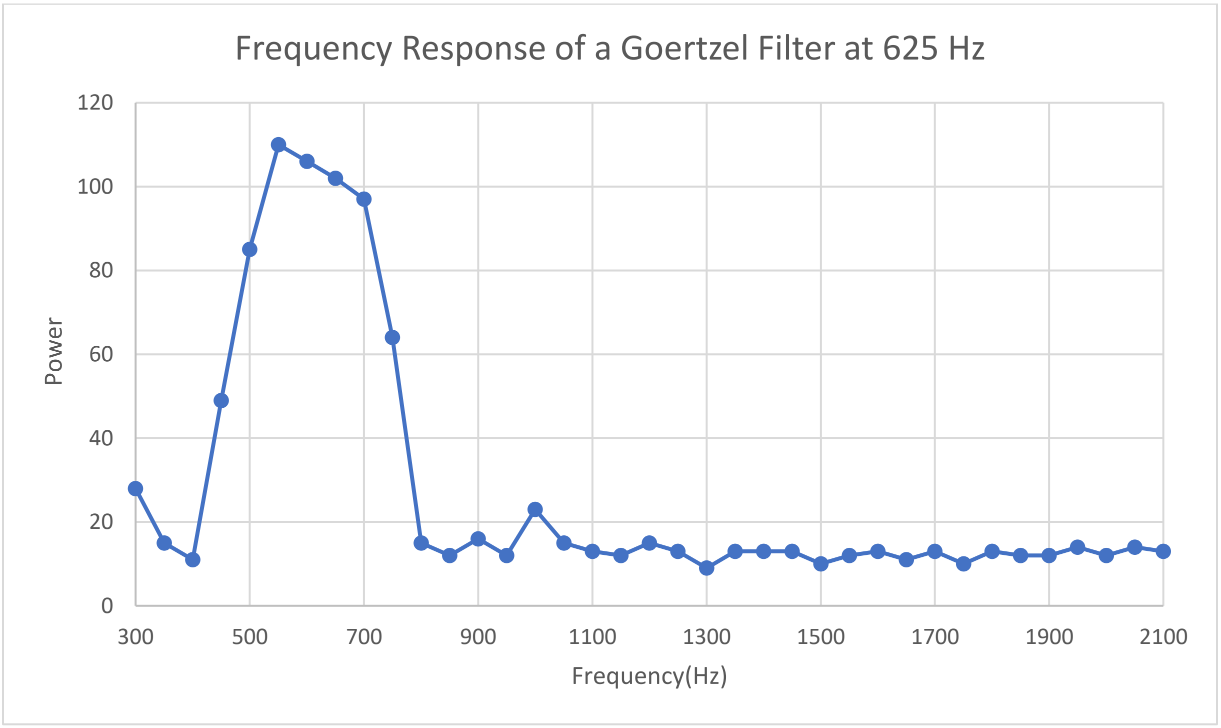


Figure 41. Graph showing Frequency Response of a Goertzel Filter at 625Hz

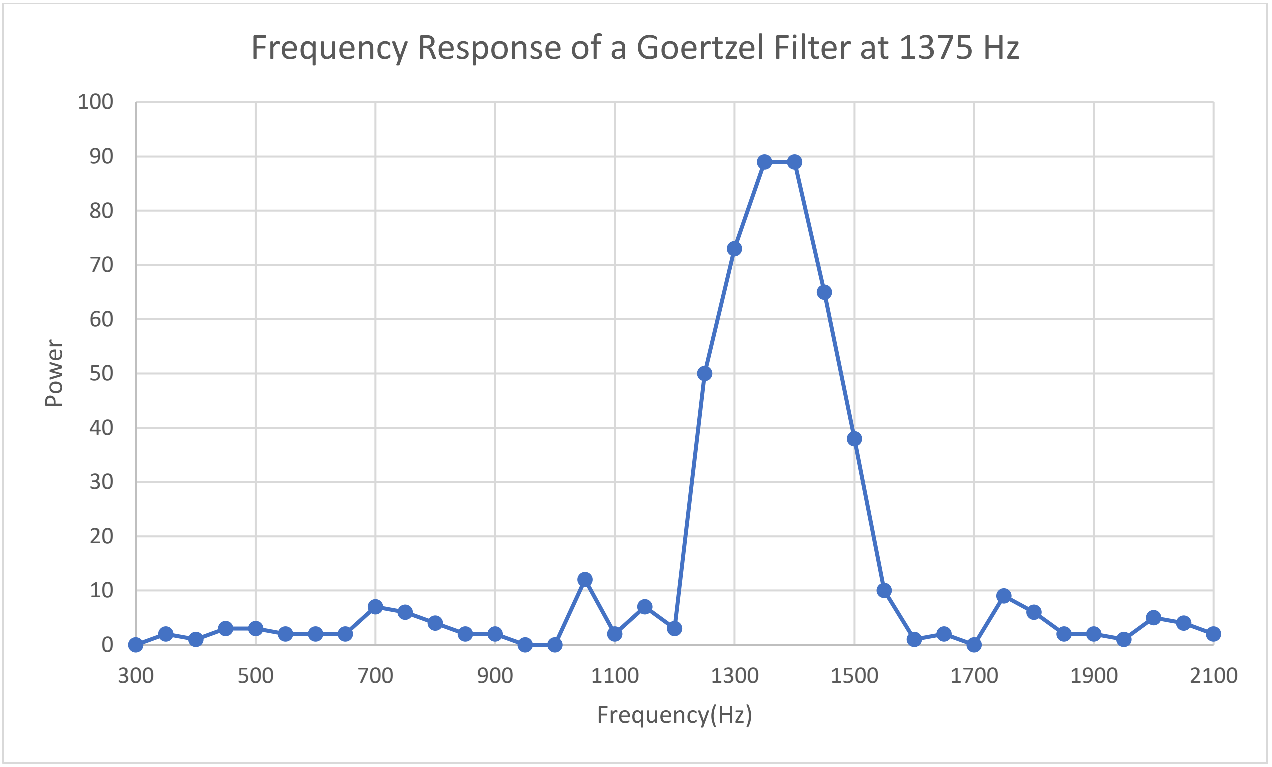


Figure 42. Graph showing Frequency Response of a Goertzel Filter at 1375Hz

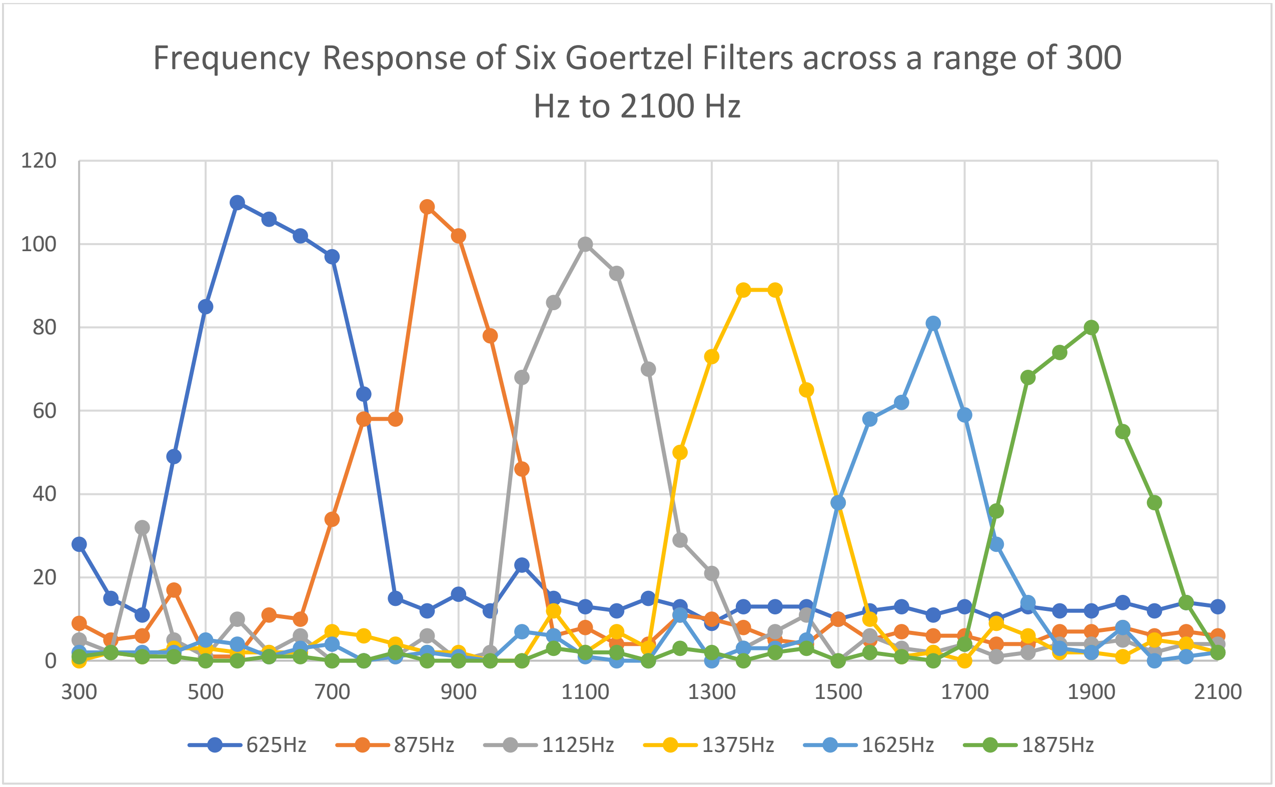


Figure 43. Graph showing Frequency Response of Six Goertzel Filters across a range of 300Hz to 2100Hz

Neopixel Response with a 950Hz Signal

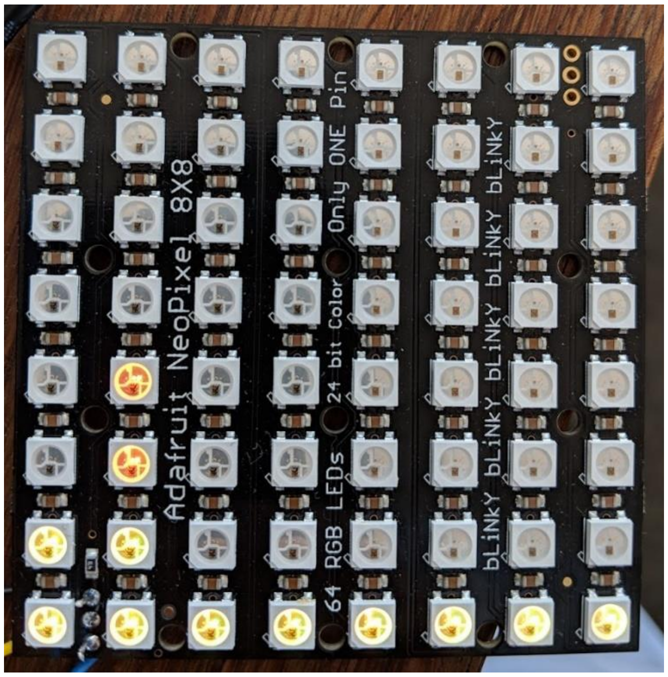

Adafruit NeoPixel 8X8
64 RGB LEDs 24 bit Color Only ONE Pin
bLiNkY bLiNkY bLiNkY bLiNkY bLiNkY

Figure 46. Neopixel Response with a 1V input

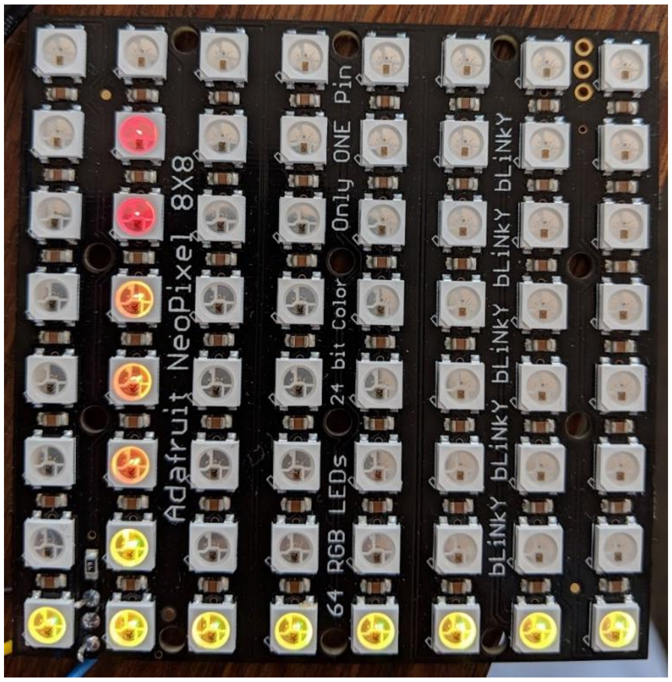

Adafruit NeoPixel 8X8
64 RGB LEDs 24 bit Color Only ONE Pin
bLiNkY bLiNkY bLiNkY bLiNkY bLiNkY

Figure 45. Neopixel Response with a 2V input

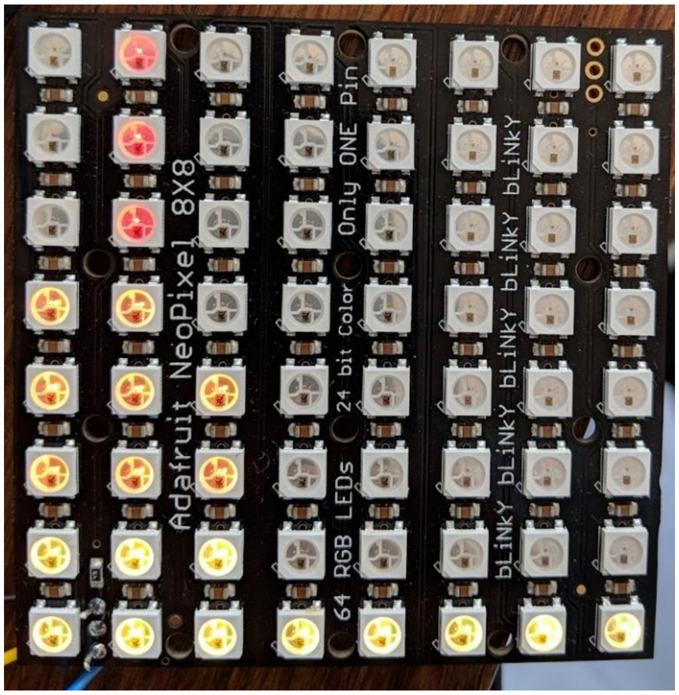

Adafruit NeoPixel 8X8
64 RGB LEDs 24 bit Color Only ONE Pin
bLiNkY bLiNkY bLiNkY bLiNkY bLiNkY

Figure 44. Neopixel Response with a 4V input

Discussion

All the results have been collected, so in this section they will be discussed and analysed to see what conclusions can be made.

Equaliser

The PCB results were gathered via connecting a signal generator to the input and an oscilloscope to the input and output, the voltage in and out were read by the oscilloscope and the gain was calculated from that.

From the graphs provided it is clear to see that the simulation results are like the final results on PCB. Comparing bass band result shows that on simulation the max gain was 6.55dB whilst on PCB it was 6.68dB, whilst the cut off frequencies were 290Hz and 275Hz respectively. Similar results are shown in the graphs depicting the mid band, with simulation maxing out at 4.89dB and the PCB having a top gain of 5.25dB. The results for the treble bands were also similar. One thing to note is that all cut off frequencies were different than their calculated cut-offs in both the simulated results and the PCB results, this probably was happening due to the bands interfering with each other. Also looking at the mid band frequency it is shown that the max gain is around 1.5dB lower than the other bands. This was done to make sure the gain was around 6.5dB when all bands were maxed out. This mid band causes interference with both the bass and treble bands as their cut off frequencies are really close so to make up for it the maximum gain of the mid band was lowered. The range of the equaliser is also shown on the bass and treble graphs. Both bands reach +/- 6dB(±0.5dB). This means the range of the equaliser is 12dB from these results it shows that the equaliser worked as expected.

The advantages of this design were that it's a cheap design the most expensive part being the 9V power supply. Compare this to what the market offers with a lot of similar equalisers being around £50 [14]. Another advantage of this design is its simplicity, there are only three bands (bass, mid and treble), this is good as the purpose of this equaliser was to help people hear lower and high frequencies better. If it was designed to be more like a normal graphic equaliser then it would be overly complicated, not fit for the intended purpose and a lot more expensive.

The disadvantages of this design are that it was limited to 6dB gain only, even though the design can provide a 12dB gain with a 15V input. The reason being that the calculated values were based on a 2V input, that limited the output to ~4.5V because the output of the equaliser goes into an AVR and an AVR cannot read values above 5V. Another disadvantage of this design was that the design was not portable. It was originally intended to make the product portable and run off batteries, but this was an issue because the potential divider needs a constant 9V input which a battery cannot provide.

Goertzel Filter

The Goertzel results were gathered using a serial monitor, the serial monitor would first show the frequency it was checking and then it would show its power. Using the signal generator as the input the signal would travel through the equaliser and the output of the equaliser is the input of the AVR. Then using the serial monitor, the results for the starting frequency of 300Hz were gathered and then the signal generator frequency is increased by 50Hz and the next results were gathered.

Looking firstly at figure 38 shows a filter at 625Hz its peak is around 98, as the band width is 250Hz this means the filter has a range of 500Hz – 750Hz and this shows as the readings in that range are at least double the readings outside that range. This is shown even more clearly in figure 39 with the 1375Hz filter which peaks at around 53 but the lowest value in its bandwidth is at least triple the power of the other ranges. Another thing to note is that outside the filter range the power still fluctuates, this is expected as all the filters produce side lobes, but the code is programmed to make these as insignificant as possible, so these side lobes don't have a higher magnitude than 10 apart from the first 625Hz filter.

One other thing to note was that frequencies below 500Hz did not work properly, some functioned fine when the band width was around 100Hz but a 100Hz band width was too small for what it is meant to achieve so it is for this reason that the first filter was 625Hz.

The last graph (figure 40) shows six filters each 250Hz apart from each other. It shows how the filters interact with each other, all the filters work in a similar manner, i.e. the further from the centre frequency of the filter the lower the power is. This has some implications as it means that the filter does not provide equal power for the entire range, however this was expected and cannot be avoided. From the graph it is also shown that as the frequency increases the power decreases, this is most significant in the first three filters. This could explain why the filters below 500 Hz do not work properly as the magnitude may be too large for the AVR to store properly.

The other three graphs show how the filters react to an increase in voltage input. The filters worked in the same way as before, but the magnitude is now increased on all filters. However, in figure 43 it is shown the difference in magnitude between the first and last filter is only around 40 whereas in the 2V graph it was around 50. This shows that the filters at higher frequencies work better with higher voltages without drastically affecting the lower frequencies.

The advantages of this design were that the flexibility is a lot better than a previous project. The previous project was restricted by the MSGEQ7, that chip determined the number of bands and the bandwidth. However, with this design the number of bands, band width and the centre frequencies of the filter are all flexible. Another reason this design is better is that it is cheaper there was no need to design an extra board for the MSQEQ7, the only board was the AVR board.

The disadvantages of this design were that it struggles to do any of the lower frequencies, as previously mentioned the filter struggled to work with low frequencies, this is bad as it meant none of the bass frequencies could be displayed which is a very important band when it comes to music audio. Another disadvantage is the filter has fluctuating power within its frequency range, this means that some frequencies power is not calculated properly.

Some of these disadvantages can be overcome, the problem with low frequencies was that the AVR could not store the value of the result of the Goertzel filter for these values. However, it did work when the band width was smaller, so next time if the lower frequencies had a smaller band width and the higher frequencies had a larger band width then more frequencies could be displayed. Another way to circumvent this issue is to use a more powerful micro controller that could store larger values.

Neopixel

The Neopixel is used to display the result of the Goertzel filter, with there being eight bands and eight levels. From the code its shown the first light would come on if it was lower than 200 and the eighth light would come on if the magnitude was lower than 1023. However, these numbers were bit shifted to the right by 3 (divided by 8) as the magnitude of the filter was scaled down, so the neopixel vales were adjusted as well.

The three figures (44,45 and 46) show how the neopixel responds with three different inputs. Firstly, looking at figure 46 it shows that with a 1kHz signal and a 2V input then 7 of the lights turned which is working as expected. In figure 44 this input is decreased to 1V and because the magnitude is lower with a lower input voltage the neopixel output also drops to four lights. Finally figure 45 shows a 4V input, the neopixel has now lit up on three of the filters, this is because the side lobes magnitude for the second filter have gotten so large that they are now outputting on to the neopixel as readings of the first and third filter. The 4V output is rarely reached when inputting a music audio signal so this overflow would not happen when music is the input and not a signal generator.

The advantages of using neopixel is that it's easy to display eight bands or less and also the pixels can be set to be any colour that is required. The neopixel is also incredibly easy to use with its one data pin for all 64 LEDs. Another advantage is that if more pixels are needed then the neopixels can be chained, this means the neopixels can display as many bands as necessary (depending on the AVR abilities).

The disadvantages are that neopixels are expensive with one costing around £20 to £30 (see appendix B for parts cost) depending on the retailer and this price can balloon if a lot of shields are needed to display multiple bands. They also draw a lot of current with one Neopixel shield needing 3 amps to display all 64 LEDs white at maximum brightness.

Conclusion

In conclusion the project was successful as all three main objectives were complete along with one secondary objective. The equaliser was successfully able to affect the gain of the bass, mid and treble frequencies as shown by comparing the simulation results to the PCB results. The project demonstrated the creation of Goertzel filters for any band width needed and did create eight Goertzel filers as shown by the Goertzel filter results. However, one downside to the Goertzel filters was its inability to create filters for lower frequencies. The Last objective was completed as well. As the results showed that the neopixel was responding to any changes the equaliser was making to the output.

Recommendations

One improvement to the project would be to get the Goertzel filter to work properly with lower frequencies. This could be achieved by further refining the code so that different filters could have different band width or by using a micro controller that could handle larger values.

Another improvement would be to implement Bluetooth communication, this would be useful to cut down on the number of wires needed to send the audio signal.

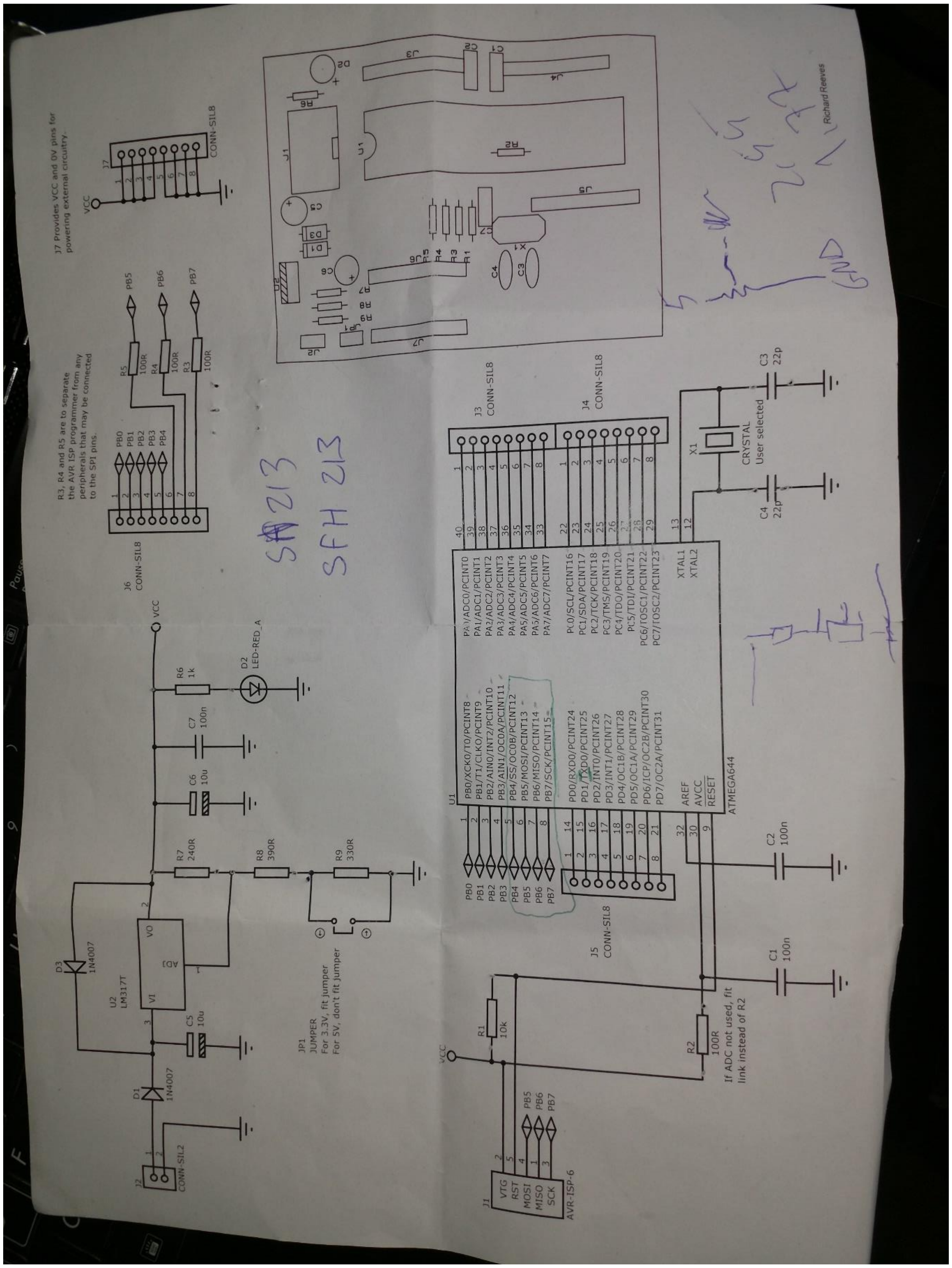
One more improvement could be to create a method to control the neopixel colour, this would make the product more useful for any end user.

References

1. Smith, S. (2018). Human Hearing. [online] Dspguide.com. Available at: http://www.dspguide.com/ch22/1.htm [Accessed 22 Apr. 2018].
2. Yamahaproaudio.com. (2018). What are equalizers? | PA Beginners Guide | Self Training | Training & Support | Yamaha. [online] Available at: http://www.yamahaproaudio.com/global/en/training_support/selftraining/pa_guide_beginner/equlizer/ [Accessed 22 Apr. 2018].
3. Banks, K. (2018). The Goertzel Algorithm. [online] Embedded. Available at: https://www.embedded.com/design/configurable-systems/4024443/The-Goertzel-Algorithm [Accessed 22 Apr. 2018].
4. Industries, A. (2018). Adafruit NeoPixel NeoMatrix 8x8 - 64 RGB LED Pixel Matrix. [online] Adafruit.com. Available at: https://www.adafruit.com/product/1487 [Accessed 22 Apr. 2018].
5. Microchip.com. (2018). ATmega164PA - 8-bit AVR Microcontrollers - Microcontrollers and Processors. [online] Available at: http://www.microchip.com/wwwproducts/en/atmega164pa [Accessed 22 Apr. 2018].
6. Nave, R. (2018). Op-amp Varieties. [online] Hyperphysics.phy-astr.gsu.edu. Available at: http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/opampvar5.html [Accessed 22 Apr. 2018].
7. Sparkfun.com. (2018). MSGEQ7 Datasheet. [online] Available at: https://www.sparkfun.com/datasheets/Components/General/MSGEQ7.pdf [Accessed 22 Apr. 2018].
8. Brown, S., Timoney, J. and Lysaught, T. (2018). An Evaluation of the Goertzel Algorithm for Low-Power, Embedded Systems. [online] Eprints.maynoothuniversity.ie. Available at: http://eprints.maynoothuniversity.ie/6567/1/JT-Goertzel-Algorithm.pdf [Accessed 22 Apr. 2018].
9. Rouse, M. (2018). What is Nyquist Theorem? - Definition from WhatIs.com. [online] WhatIs.com. Available at: https://whatis.techtarget.com/definition/Nyquist-Theorem [Accessed 22 Apr. 2018].
10. Arar, S. (2018). Fixed-Point Representation: The Q Format and Addition Examples. [online] Allaboutcircuits.com. Available at: https://www.allaboutcircuits.com/technical-articles/fixed-point-representation-the-q-format-and-addition-examples/ [Accessed 22 Apr. 2018].
11. Chen, C. (2018). Modified Goertzel Algorithm in DTMF Detection Using the TMS320C80. [online] Ti.com. Available at: http://www.ti.com/lit/an/spra066/spra066.pdf [Accessed 22 Apr. 2018].
12. Said, O. (2018). OmaymaS/DTMF-Detection-Goertzel-Algorithm-. [online] GitHub. Available at: https://github.com/OmaymaS/DTMF-Detection-Goertzel-Algorithm-/blob/master/Detect_DTMF_July2014.c [Accessed 22 Apr. 2018].

Appendices

Appendix A – AVR Schematic



Appendix B

Part	Supplier	Qty	Cost each (£)
Neopixel Shield	Farnell	2	27.89
ATMEGA164P	Farnell	2	2.29
Slider Potentiometer	Rapid	3	4.62
5V power Supply	Farnell	1	£17.81
9V Power Supply	Farnell	1	£5.08

Appendix C

Original Specification

Primary

- Design, build and characterise a 2-band tone control, as a self-contained module.
- Use the Goertzel Algorithm to act as band pass filters to break down signal into different bands using an AVR.
- Output Audio signals spectral content onto a neopixel board.

Secondary

- Connect audio via Bluetooth.
- Adjustable frequency ranges for display.
- Change colour of display.