Arduino Based Interactive Audio Reactive Lighting System

Third Year Individual Project – Final Report

April 2020

Lucas Francisco Millard Ferreira Leão

9627445

Supervisor: Dr Patrick Gaydecki

School of Electrical and Electronic Engineering

Abstract

This project focussed on developing an audio reactive lighting system. The purpose of the system is to react to sounds in its vicinity, both those created ‘live’ by individuals or instruments, or from recordings.

A microphone was used to sample the sound, allowing for easy interaction with the project. Addressable LEDs were used to create an aesthetically pleasing lighting display composed of unique patterns that were dependent on the sound input.

An original method was developed for creating an Arduino based sound-to-light system during the course of this project. An Arduino Due, digital microphone and Adafruit Neopixels were used. The microphone used the I2S protocol to communicate with the Arduino, which was set as the master. The Neopixels were used to create the lighting display. The lighting display was mounted on a wooden frame that allowed it to be stood on a table or hung on a wall. The system has two modes. In the first mode the system reacts to the volume of the audio input. In order to do this an RMS calculation was performed on the microphone samples, resulting in a single value for the energy of the audio input. In the second mode, different sections of the lighting display react to set frequencies, which means that the lighting display reacts differently depending on the pitch of the audio input. For the system to be able to differentiate frequencies in the audio input, infinite impulse response filters were used. The filters used in this project were the digital equivalent of analogue LCR filters. They have a frequency response centred around a resonant point that falls off either side, allowing a range of frequencies to pass through, but reducing the energy of these frequencies by an increasing factor the further they are from the resonant point. All signal processing in this project was performed in real time by the microprocessor.

The project was completed successfully and fulfils the design brief, which was to create an audio reactive lighting system that can be exhibited and interacted with.

Contents

Word Count: 7963

1. Introduction

The project consists of a microcontroller used to drive an LED lighting system that reacts in real time to a live audio input. It is a system that was designed to be exhibited and interacted with, principally for the purposes of entertainment. The aim was that when exhibited, people would be able to approach the system and make sounds to try to get the lighting system to change colour, flash at different speeds and display different patterns.

The project was designed to be an installation piece. The student's career goals are related to the live events industry. Many engineers in the live events industry produce installation pieces as part of their work, by producing this the student hopes to use it to add to their professional portfolio as an example of the work they can do and give them some exposure in the industry.

The microcontroller used is an Arduino Due. Arduino are a company that make open source hardware and software used by both amateurs and professionals for a variety of applications [1]. The Arduino Due is based on the Atmel SAM3X8E ARM Cortex-M3 [2], a 32-bit ARM microprocessor that operates at 3.3 V. The audio signal is sampled by a digital microphone, the SPH0645LM4H, which communicates with the microcontroller over I2S [3]. One of the aims of the project was to process the audio signal in software. The aim was for the software to analyse the frequency and energy, obtain a value for the BPM and a representation of the rhythm of the audio input.

The LED lighting system is made up of addressable LEDs, the SK6812RGBW, made into strips produced by Adafruit [4].

The datasheets and user guides for the microcontroller, microphone and LEDs were key to completion and success of the project. They supplied information and data on how to use these components.

The section on the Atmel data sheet regarding the Synchronous Serial Controller (SSC) peripheral used for I2S communication [5] was required in order to understand how to set up communication between the microphone and the microcontroller. It states that enabling the clock for the SSC peripheral must be done through the Power Management Controller peripheral with the Peripheral Clock Controller. The SSC has an independent receiver and transmitter for I2S on the device and it can operate as a master or a slave. To receive or transmit the system requires three signals; the data signal, the clock signal and the frame sync signal. Each of these signals are highly programmable allowing for good flexibility of use. The Peripheral Clock Controller can be set by writing to a single register and enables the clock to be sent to peripherals on the device.

The datasheet for the microphone states that it also communicates over I2S using 3 signal lines: a clock signal, a word select signal (the equivalent of the frame sync on the microcontroller) and the data signal. There is also a channel select signal for when two microphones are used to sample a left and right signal. The microphone only operates as a slave so must receive the clock and word select signal from an external source. The data is transmitted in the form of 24 bit 2's complement and has a precision of 18 bits. It operates at 1.6 – 3.6 V [3] so can be driven by the microcontroller.

The LEDs are designed to be used with Arduino microcontrollers and Adafruit have produced a library for this purpose [4]. Data can be sent over any of the digital output pins on the microcontroller (the data is sent over unipolar NRZ [6]). It operates at 5V logic so will need a buffer to boost the signal from the microcontroller. It will also have to receive its power from an external source. The guide gives details on best practise and how best to wire up the LEDs to allow for safe operation.

The notes for the Digital Signal Processing course at the University of Manchester were another important resource [7]. Chapters 11, 12, 20 and 21 give examples of different designs and implementations of DSP systems. Chapter 21 focusses on I2S and gives an explanation of how devices communicate over this protocol. Chapter 8 and 9 consider filter design, relevant to this project as it uses software-based filters to analyse the audio signal. These notes allowed the student to develop an understanding of the fundamentals before applying the concepts to their project.

1.1. Aims and objectives

The project had three main aims:

- To produce a lighting system using addressable LEDs.
- To produce an intuitive control surface for the lighting system.
- To analyse an audio signal in four different ways and use this analysis to affect the lighting system in real time.

The lighting system was designed to be exhibited, so had to be visible from a distance to a crowd of people. It was designed to be used in different scenarios and environments, and encourage people to interact with it. A block diagram for the system can be seen in Figure 1.

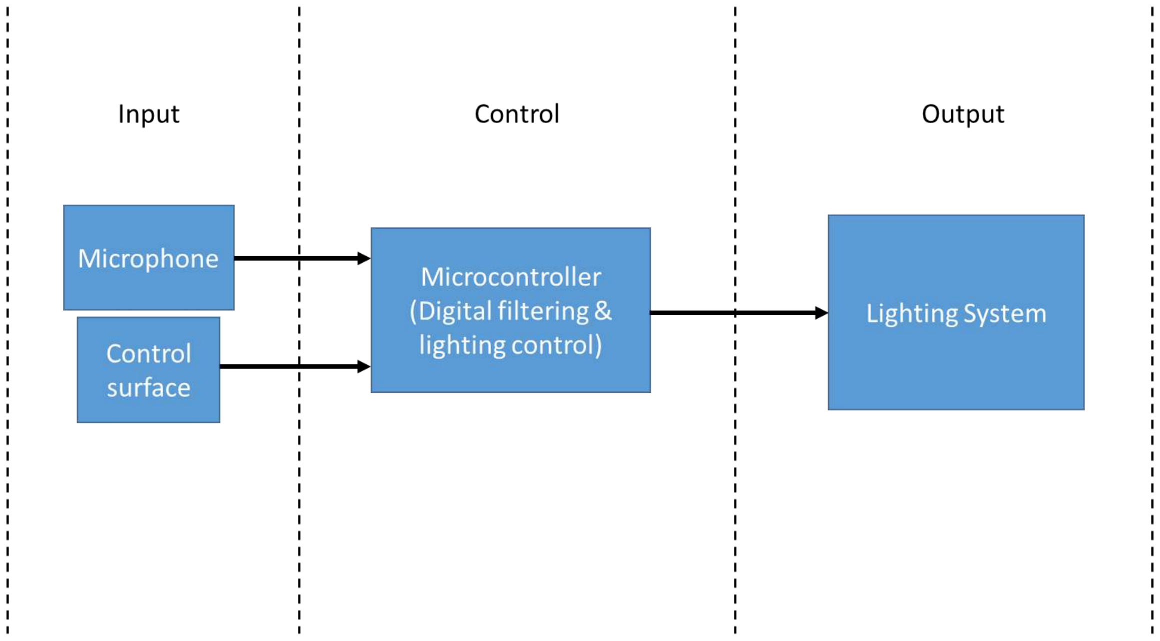

Figure 1: System block diagram.

The control surface was designed to allow the user to control how the audio input affects the lights. It was designed to be intuitive and allow for the same audio input to have several different effects on the lights depending on what the user wants to achieve. The design for the control surface can be seen in Figure 2.

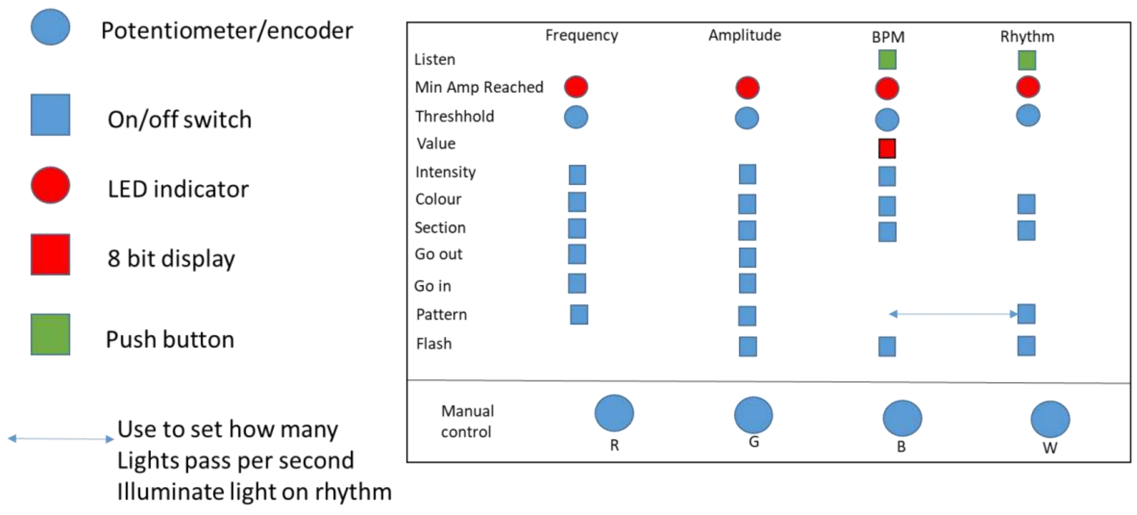

Figure 2: Control surface design.

The aim was to analyse the audio signal in four different ways, which were then to be used to control different effects for the lights. Figure 2 shows the four ways the audio was to be analysed across the top and how it would then affect the lighting system along the side.

A user would switch on one effect or less per audio characteristic, and this is the effect that they would have on the system. For example: if the user set the system such that the frequency affected the colour of the lights, the lights would change colour depending on the pitch of the sound that was sampled.

The project involved the physical construction and design of the lighting system and the user interface. The design and implementation of the communication between the lighting system, microcontroller and microphone and the audio analysis. This software was written in C and assembly. The ultimate goal of the project was to create a useable system that fulfilled the aims and objectives stated above.

1.2. Motivations

This project is a bespoke project proposed by the student. The student wanted to do a project based around their interests and related to the career they want to pursue. The student is currently a freelance lighting engineer and involved, as a technician, with various student-led performances at the University of Manchester. They hope to pursue a career as an audio-visual design engineer. By producing an audio-visual system as part of their third-year project they can demonstrate to potential employers or clients their skills as an engineer and designer and can include their project in a portfolio of work. As a freelancer the student has to regularly advertise and sell their skills to potential clients and one of the key ways they do this is by having evidence of the skills they have to offer.

2. Literature review

2.1. Similar projects

Collins English dictionary defines art by saying “art consists of… objects which are created for people to look at and admire or think deeply about.” [8] In this sense the project can be considered art. It exists to be admired. Although it can be interacted with and affected by the person using it, it has no other purpose than to be admired.

A community exists that creates art in this way and this project contributes in a small way to that.

When visiting an art gallery, it is rare that there are installations that can be interacted with. Traditional art galleries consist of paintings, photos or sculptures that are meant only to be looked at, protected by a thin steel cable and human's innate desire to do the right thing. However, there is art that exists for people to interact with. The Tate Modern's turbine hall has a long history of exhibitions that exist for that reason, from basking in the sun of Olafur Eliasson's 'The Weather Project' [9] to sliding down the slides of Carten Höller's 'Test Site' [10]. It was even written about the 'Test Site' exhibition that "the work must not be understood as an object but as a platform for interaction" [11].

There are several art pieces that bring together engineering, light, sound and interactive art like this project does. In the late 90s Golan Levin created the 'Audiovisual Environment Suite' [12], an interactive software environment where the user can create animations by drawing lines and squiggles that the software animates based on what shape was drawn and how it was drawn. The software can then use these animations to generate an audio output. The 'Audiovisual Environment Suite' was exhibited at 40 international venues during the early 2000s and was awarded a distinction at the Prix Ars Electronica in 2000 [13].

A leading figure in the world of Electronic art is Walter Giers. He was described in an Electronic Beats documentary as a man who wanted to "bring everything together in an interdisciplinary way" [14]. He was interested in the use of sound and music in art, as a form of expression rather than just something used for pleasure. Walter Giers' most famous piece is an interactive picture called 'Handbild/Hände' with two large metal hands. The piece produces a sound when the circuit, broken between the two hands, is closed by someone touching them. The sound changes depending on how you touch them.

These examples show there is a market for, and a community creating, pieces of art that bring together electronics, sound and light and allow people to interact with them. It proves that the project has value despite not necessarily having a utilitarian purpose.

2.2. DSP processes on Arduino

Arduino produce open source hardware that is used by a huge international community of hobbyists, students and professionals alike. There is a vast array of projects published as examples or tutorials of what is achievable using Arduino hardware. There are examples of DSP projects done using Arduino hardware however DSP is not an area that has been widely investigated on the Arduino platform. This project expands the knowledge base that makes up the Maker and Arduino communities by investigating the use of DSP processes and I2S communication on the Arduino Due.

A GitHub user, called Shajeebtm, posted a project where the Arduino FFT library was used to produce a 32-band audio spectrum visualiser using the Arduino Nano [15]. A stereo input to the Arduino's onboard ADC was used and the system outputted to a 32x8 LED matrix display. The system had a 38.6 kHz sampling frequency and the FFT operation was performed on 64 samples. The FFT library can deal with more samples than this, but was described as slow by Shajeebtm. The FFT library was also used for a project posted on the Arduino Project Hub. A music reactive strip was created, similar to the project discussed in the report [16]. An Arduino Nano, WS2812B LED strip and electret microphone were used. The electret microphone produced an analogue signal that was input to the onboard ADC. The FFT library was then used to split the sound into octaves and this information dictated how the strip behaved. These projects were similar to the project discussed in this report but there are key differences. Analogue inputs were used for the sound in the other projects, meaning that the processor had to perform the conversion to a digital signal, slowing the system down, whereas in this project the microphone provides the microcontroller with a digital signal. Both also used the FFT library available for Arduino, an inefficient library for the purposes of the project discussed in this report as the system did not need to analyse the signal in such a complicated manner.

The Arduino Due has also been used for some DSP purposes. An Arduino community member produced a digital, semi modular synthesiser that is capable of generating a variety of waveforms and effects, which is now sold as a commercial product [17]. The Arduino generates the digital form of the signals and then communicates with an audio codec (WM8731) over I2S, which performs the digital to analogue conversion [18]. This project proves that it is possible to communicate over I2S using the Arduino Due despite it not being officially supported. I2S, however, is supported by the microprocessor at the heart of the Due.

The project discussed in the report has many similarities with previous projects, however there are some key differences that make the project unique. It is a sound to light system and there are many examples of these. However, sound to light systems on an Arduino often use analogue inputs and the Arduino FFT library, unlike this project. Furthermore, there are very few projects that use the I2S communication protocol on the Due, most probably because it is not explicitly supported. This proves how this project has developed a new method for creating a sound to light system using the Arduino Due.

2.3. Open source hardware and the maker movement

The maker movement is described by Make magazine as a "tech-influenced DIY community**"** [19] and is arguably the next step in grass roots engineering and innovation. The importance of communities like this and contributing to them should not be undervalued, since they are driven by the same fascination and desire that lead to the popularisation of the personal computer and the emergence of the Silicon Valley giants.

An edition of the Harvard Education Review argues for the importance of the Maker Movement in education [20]. In fact, the act of producing something has been valued in education for many years, the difference is that now it can take the form of a more tech-based approach. Gone are the days of learning how to sew in a Home Economics class, students instead learn about programming and 3D design, meaning students can go from being consumers to being producers.

Dale Dougherty, who founded Make magazine and The Maker Faire, believes that this community does not only give individuals the opportunity to learn new skills but also gives companies opportunities to access ideas and skills that exist outside themselves [21]. He argues that the opportunity for individuals to become independent self-employed developers going after small niches, that are often overlooked by companies for not being financially viable, may play a big part in the industry in future.

The importance of contributing to these communities should not be understated. The ability to learn, innovate and create should not just be available to companies with millions to pump into R&D or even universities: it should be an opportunity that is available to anyone anywhere and this philosophy is one that will benefit us all.

3. Technical details

3.1. Project method

As mentioned in Section 1.1, the aims of the project were to produce a lighting system and complementary intuitive control surface, which analysed an audio signal in order to affect the system in real time.

To achieve these aims, choices for both the hardware and software were considered at the input, control and output stages of the system.

3.1.1. Input

The project needed an audio input to control the lighting system, for which a digital microphone was chosen. A microphone is a good choice as it makes the system immediately accessible to people; users walking past will see the system react to their sounds and will feel encouraged to interact with it. It should be something people can enjoy without having to dedicate much time or effort to it.

The digital microphone chosen communicates with the microcontroller over I2S, a three-wire synchronous serial interface [22]. It is a low-cost method of sampling a live audio signal with a frequency response that is fit for purpose. Additionally, it is low profile so has little impact on the physical design. It can be purchased in the form of a breakout board, allowing for easy physical implementation: particularly useful in the prototyping phase where using just the microphone would have been outside of the scope of the resources the student had access to. Lastly, I2S is an industry standard communication protocol, and the microprocessor used, an ATSAM3X8E in the form of an Arduino Due microcontroller, has a peripheral dedicated to serial synchronous communication [5]. This made I2S an ideal option for the communication protocol.

The microcontroller had to act as the master for the I2S communication. The only open-source library for I2S communication on an Arduino Due available was designed to configure the microcontroller as a slave [23]: therefore, the software for this protocol was written in assembly language. As only a very small part of the software had to be written in this way, the most efficient method was to write it within the Arduino development environment along with the rest of the software, which was written in C using Arduino libraries.

3.1.2. Control

A method had to be developed to use an audio input to control the lighting system. At the control stage the system had to process the signal coming from the microphone. This signal had to then be translated into a lighting effect. This effect had to be clearly linked to the sound input so the user could understand how their interactions impacted the lights. It also had to create an aesthetically pleasing pattern.

An aim for the project was to analyse the audio signal in four different ways however, this was not achievable within the timescales of the project. Instead, two aspects were chosen to be considered: the energy and frequency of the sound. Both parameters would be known to any user, making the system accessible to a wide audience.

A switch mounted on the control box enables the user to choose which parameter, frequency or energy, controls the lights. The control box houses the Arduino and has the microphone mounted on it. It is a plastic enclosure [24] that was modified for purpose. This was a cheap and efficient way to produce a prototype.

As a digital microphone was used, the input signal was measured in the form of a discrete signal in the time domain. This signal was analysed through the software written for the microcontroller. Functions in the software were used to calculate the energy of the input signal, and digital band pass filters were applied to the signal to analyse the proportion of the various frequencies within it. Analysing the signal in this way is efficient, precise and cost-effective. It reduces the need for hardware, which not only cuts down on costs but also makes the system more reliable and avoids any delay that would be introduced from converting the digital signal to an analogue signal and running it through a circuit. The reaction speed of the system is a vital part of making it work well in real time.

3.1.3. Output

The lighting system uses Adafruit Neopixel RGBW LED strips mounted on a wooden frame. Since the system is designed as a display, it must be aesthetically pleasing. Neopixels are individually addressable [4], meaning each LED in a strip can be controlled independently without being wired separately. This allowed for a wide variety of patterns and designs to be developed. RGBW LEDs were chosen rather than RGB LEDs, because white LEDs give a crisper white than a red, green and blue mix does.

Neopixels are wired up with 3 pins: power, ground and data. They are connected in series, and can be bought in strips where they are already connected to one another. In the lighting system the data line follows the strips round: the address of each Neopixel is the same as its position along the data line. By connecting the Neopixels such that they are addressed in a logical order, the process of writing the software for the lighting system was made more efficient.

Unlike the data line, power was distributed to the LEDs at several points on the strip. This was to avoid ‘brownout’: an effect where LEDs are dimmer further along the strip and the red component of their colour is increased. This is due to the fact that the red LED on a Neopixel can operate at a lower voltage than the green and blue.

Adafruit provide an Arduino library for use with their Neopixels, which was used when writing the software for this lighting system. This library ensures that the code written in the software is translated into the communication protocol that the Neopixels use. Additionally, the large online community providing information and support made Neopixels a strong and sensible choice.

Wood was used for the frame as it was within the budget constraints and easy to modify at the prototyping phase. The ability to modify the frame easily was important as it allowed cables, used to distribute power, to be hidden easily.

3.2. Design, development and implementation

3.2.1. Neopixels

The most appropriate choice of Neopixels were RGBW LED strips mounted on a black coloured flex with a density of 60 LEDs per metre [25]. They were supplied by the Pi Hut [26] in 1 m lengths and came with a waterproof silicon casing. As detailed above, RGBW LEDs were chosen for an aesthetically pleasing effect. The black flex was chosen for the same reason as the LEDs were mounted on a black background. The density of LEDs was chosen such that each line of LEDs had at least 2 LEDs. This ensured even the shortest sections looked in place. Strips were used as they allowed the student to cut them to any length needed, and then arrange them in the desired pattern.

RGBW Neopixels are based around SK6812RGBW LEDs [6]. Each SK6812RGBW consists of a controller and 4 separate LEDs, one for each colour. They operate at 5V and when data is sent at 800 kbps, as it is in this system, they have a refresh rate of 30 frames per second. If a Neopixel is running at full brightness it draws 80 mA [6]. The project used 118 pixels. Together these can draw 47.2 kW of power. In order to supply this power, the power supply chosen was rated for 10 A at 5 V [27].

The LEDs are driven using PWM (pulse width modulation) that is built into each pixel. The pixel's communicate using 8-bit data per channel, or 32-bit data per pixel, over the Unipolar NRZ protocol. By having PWM control built into each pixel a microcontroller can cease communication with said pixel and the pixel will maintain state. This means that through the use of shift registers it is possible to control a strip of pixels over a single digital output pin. This allowed for a simple and neat physical implementation of pixels designed as outlines of a shape.

The microcontroller used in the project runs at 3.3 V logic. As the Neopixels operate at 5 V, a logic level shifter, the SN74AHCT125 by Texas Instruments, was used [28]. The circuit diagram, Figure 3, shows how this was implemented.

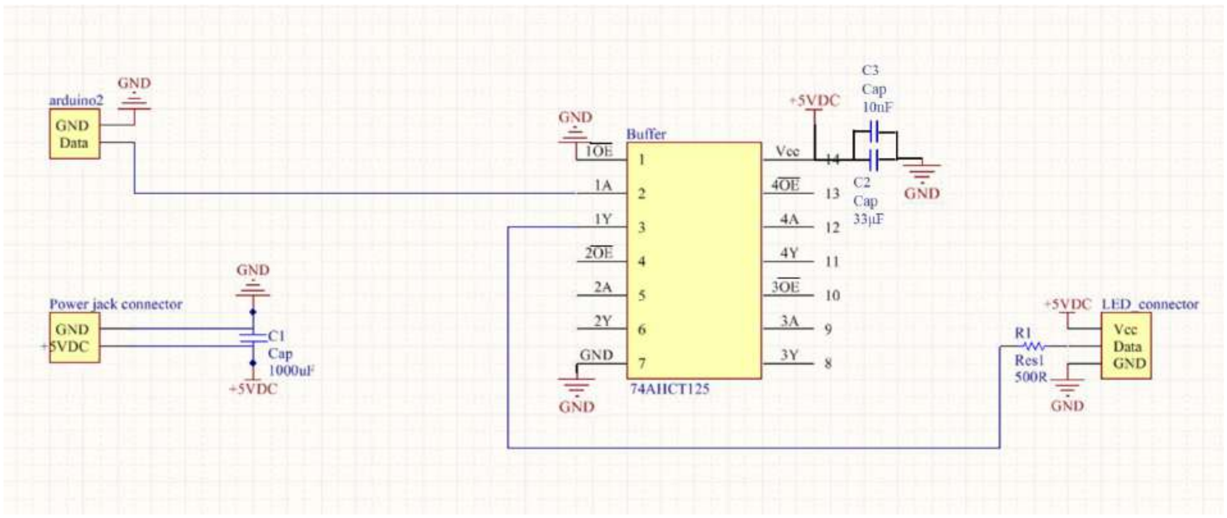

Figure 3: Logic level shifter implementation.

Two decoupling capacitors were used with the SN74AHCT125, as when first assembled into a stripboard the LEDs did not operate in a stable manner. It was then discovered that the stripboard was not grounded properly. Therefore, potentially the decoupling capacitors were unnecessary, however they have not been removed since the system works well with them included. The resistor protects the LEDs in the event of a spike in the voltage level on the data line, as recommended by the manufacturer [4]. The 1000 µF capacitor over the power supply is also recommended by the manufacturer. Pin 1 on the logic level shifter is driven low to enable the output.

Initially the lights colour varied proportionally to the input from the microphone. This made the system very susceptible to background noise. To correct this the system was modified such that the lights reacted when volume or certain frequency levels hit a set threshold. Once the threshold is hit the lights follow a pattern. Implementation of different patterns was facilitated through the use of the library created by Adafruit to simplify the communication between an Arduino and Neopixels. Hysteresis was used to improve the lighting system's stability. The turn on threshold is higher than the turn off threshold, such that the system is not constantly switching between the two states when close to the threshold.

In the first mode the system reacts based on the energy of the audio input: the volume. If the threshold is reached the lights will all turn on. They have one set colour, magenta, and the brightness varies at a constant rate. This will continue while the energy of the audio input remains above the turn off level. A flow diagram representing the software for the Neopixels, in mode 1, can be seen in Figure 5.

In the second mode the lighting system reacts based on the pitch (frequency) of the audio input. The outermost lights react to low pitches of sound, the innermost lights react to higher pitches of sound and the central lights react to a middling pitch. They change colour as they are activated by the audio input. The colour is set using the HSV (hue-saturation-value) colour model. The LEDs follow the HSV circle round when they are activated, Figure 4.

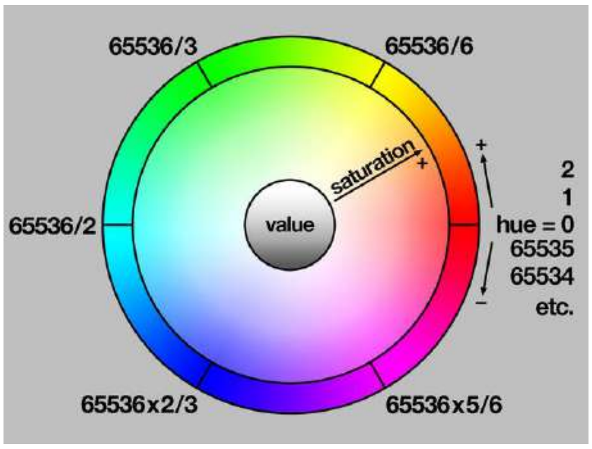

Figure 4: HSV colour model [4].

The LEDs closer to the centre of the design have a reduced saturation level set. These states are set for each group of LEDs one after another, where there are three groups: one representing the low, one the mid, and one the high pitches. This can be done because, once set, the LEDs maintain their state, even when they are not being communicated with. A flow diagram representing this software is shown in Figure 6.

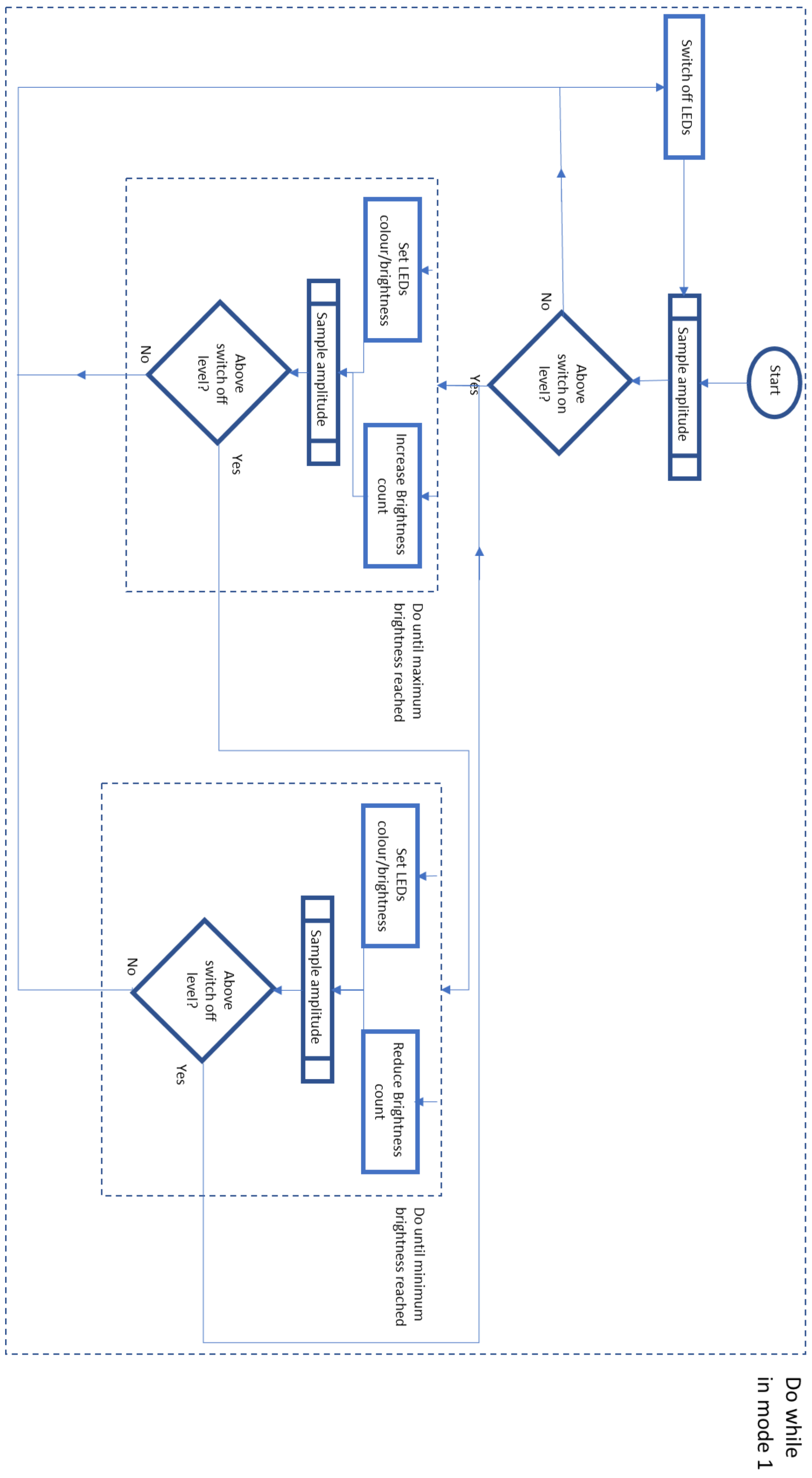

Figure 5: System in mode 1.

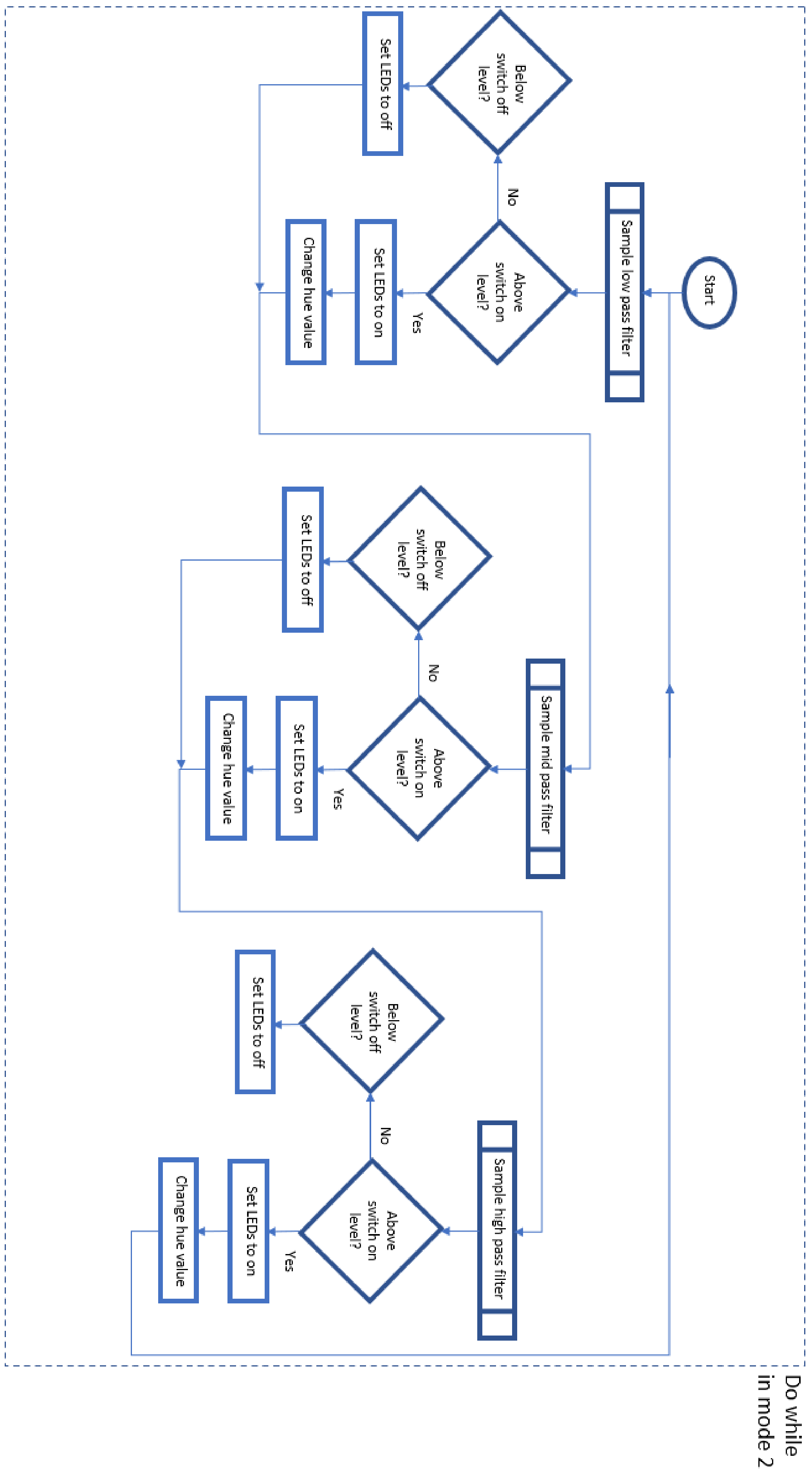

Figure 6: System in mode 2.

By using colours that change on each iteration of a loop, with sections of the lighting system that operate independently, the user can create a variety of different patterns and colour mixes depending on the sound they create. This ensures that the system rarely repeats itself, making it engaging and more entertaining and immersive than if the system followed the same pattern every time. Figure 7 shows the system in mode 2.

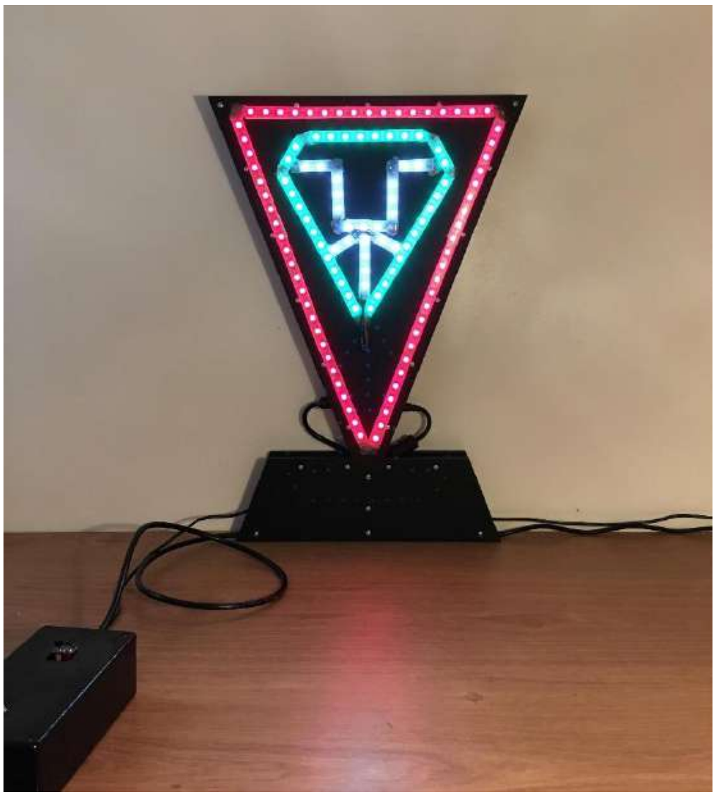

Figure 7: The system operating in mode 2.

3.2.2. Digital microphone

The microphone used in the project is the SPH0645LM4H-B [3]. Adafruit manufacture this microphone in the form of a breakout board [22], allowing for ease of use in a prototype. It communicates over I2S and outputs a discrete representation of the sound wave it measures.

The microphone is omnidirectional, important for the interactive element of this project.

When the system is in use the microphone operates at 2 MHz, this means it is in “active mode”, which is the normal mode of operation. There is also a “sleep mode” but it is not used in this application. Figure 8 shows the expected frequency response of the microphone.

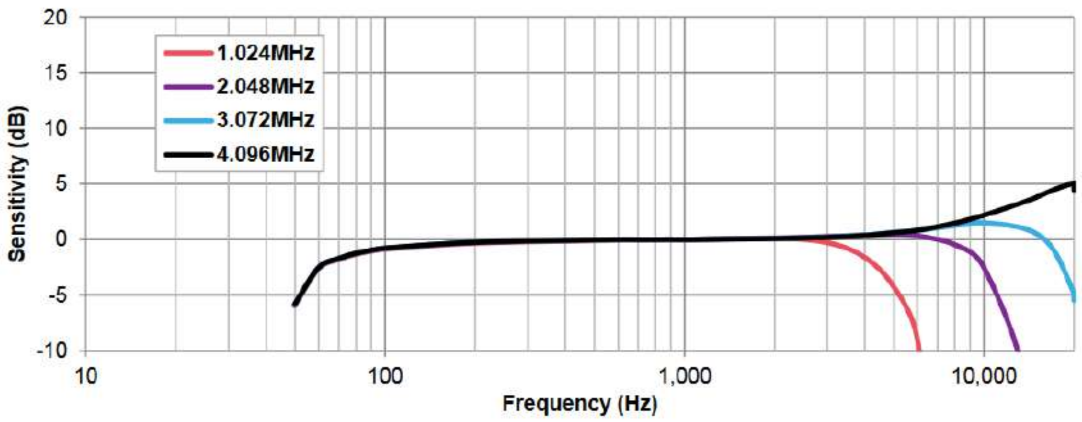

Figure 8: Microphone frequency response [3].

The microphones sensitivity starts to drop away from 0 dB (unity gain) above 7 kHz and below 100 Hz. This is a frequency range that is fit for purpose as the sounds that are expected to be registered are within human vocal range and the range commonly used in music, approximately 100 Hz – 4000 Hz [29]. Only particularly low bass notes may not be registered.

3.2.3. Microphone communication

The microphone communicates with the microcontroller over I2S (Inter-IC Sound). I2S operates using 2 control signals: the continuous serial clock (SCK) and the word select signal (WS). Both are sent by the master; data is then sent over a 3rd line (SD) [29]. The clock signal sets the bit rate and the word select signal sets the word length. Figure 9 shows a diagram of the I2S communication protocol.

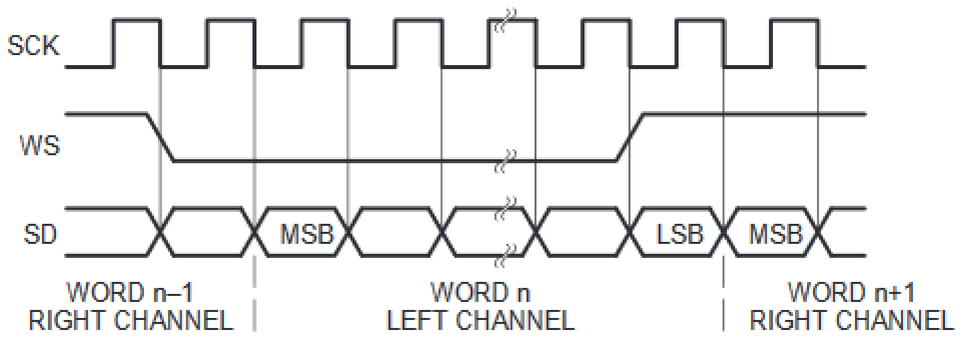

Figure 9: I2S communication protocol [22].

The microphone over-sampling is set to 64 bits therefore the WS signal's frequency was set at 1/64 of the clock's frequency. Data is sent in the form of 24-bit 2's complement with a precision of 18 bits. The data is sent with MSB first and the final 6 bits of the word are ignored. The select pin determines at what point in the WS signal the data pin is driven by the microphone, this is so that in the case of 2 microphones being used, the data is sent by each one at a different point in the cycle. However, for this project this functionality was not necessary as only one microphone was used. Since the select pin is permanently driven low a new data word is transmitted at a rate of 1/64 of the SCK signal. As the SCK rate is approximately 2 MHz the sample rate of the system is 31.25 kHz. This gives a Nyquist point of approximately 15 kHz, greater than that of the fall off region of the sensitivity of the microphone. The sample rate therefore does not affect the maximum frequency the microphone can sample.

The Atmel Sam3X8E, the chip the Arduino Due is based around, has a peripheral that is designed for I2S communication. However, there is no library for I2S communication on the Arduino Due with the Arduino configured as the master. Therefore, the most efficient way to write the software for the I2S communication was to write it in assembly language. A flowchart representing this section of the program is shown in Figure 10 and the code for it is as follows:

```
//set up microphone
  REG_PIOB_WPMR = 0x50494F00; // DISABLE WRITE PROTECT PIO REGISTERS
  REG_PIOB_ABSR = 0b00000000000000000000000000000000;  //SET AS PERIPHERAL A FOR
SSC
  REG_PIOB_PER = 0b00000000000000000000000000000000; //ENSURE PIO ISNT ENABLED
  REG_PIOB_PDR = 0b00000000000001110000000000000000; //DISABLE PIO ON SSC PINS
  REG_PMC_PCER0 = 0b1<<26; //ENABLE CLOCK FOR SSC
  REG_SSC_WPMR = 0x53534300; //DISABLE WRITE PROTECT TO SSC REGISTERS
  REG_SSC_CR = 0b0000000000000001; //ENABLE RECEIVE
  REG_SSC_CMR = 21; //SET CLOCK DIVIDER
  REG_SSC_RFMR = 0b00011000000101111000000010010001;   // GOING FROM THE LSB
UP:     SET DATA LENGTH (WORD LENGTH) TO 18       SET NO LOOP       SET MSB SAMPLED
FIRST        SET 1 DATA WORD PER FRAME
  REG_SSC_RCMR = 0b00011111000000000000010001000100; //GOING FROM THE LSB UP:
SET MCK AS CLOCK IN FOR SSC    SET RECEIVE CLOCK AS CONTINUOUS     SET GATE AS
CONTINOUS      SET TO START RECEIVING DATA ON FALLING RF   SET NO STOP CONDITION
SET NO DELAY PERIOD FOR RF DIVIDEDR
 pinMode(22,OUTPUT);
 digitalWrite(22, LOW); //SET SELECT LINE AS LOW
```

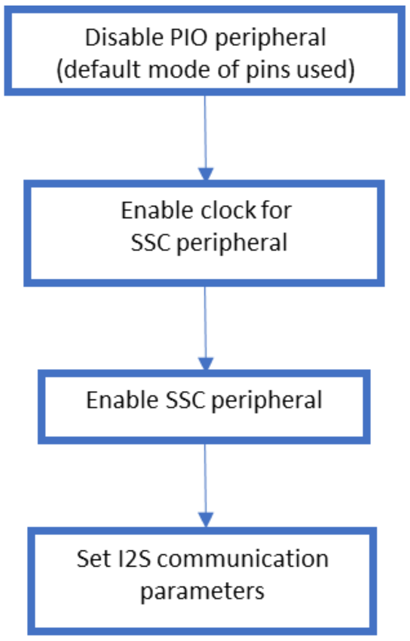

Figure 10: Flowchart, I2S communication setup.

The code shown above runs once after the system is powered on. It sets the appropriate registers to enable the SSC peripheral, which is the peripheral that handles I2S communication. It ensures that the microphone is constantly communicated with while the system is powered up.

Figure 11 is a flowchart that demonstrates how a value coming from the microphone is sampled. To avoid repeated samples the software checks a new value is available, and once there is, a new sample is saved. To check whether there is a new sample available the program looks at a register that is set once an I2S sample has been read.

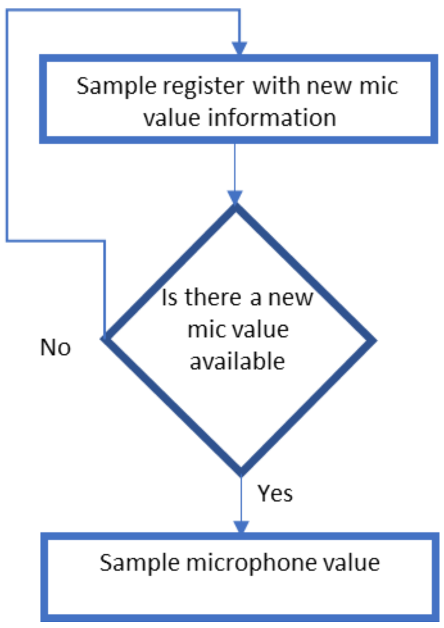
Sample register with new mic value information
Is there a new mic value available
No
Yes
Sample microphone value

Figure 11: Flowchart, sampling an I2S value.

3.2.4. Signal processing

The system processes the signal in different ways for each mode. In mode 1 the system calculates the energy of the input signal. In mode 2 it calculates the components of different frequencies within the input signal. Since the microphone outputs a digital signal, a discrete signal in the time domain, the most appropriate method to do this was to perform the signal processing in software.

The system uses root mean square (RMS) calculations to calculate the energy that is carried by a signal [31]. The RMS of a signal, x_{RMS}, is given by:

$$x_{RMS} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} x_n^2} \tag{1}$$

where: x is the sample value, N is the number of samples and n is the sample number.

It does this for the signal before and after filtering. By calculating the energy within the input signal before any filtering is done the system is able to measure how loud the signal is that the microphone is sampling. By calculating the energy after filtering the signal the system is able to measure the energy that specific frequencies have, or what component of the sound input is made up of those specific frequencies.

To calculate the energy of the input signal a function was written called *all_RMS*. This function first samples the audio input 500 times. Since the sample rate is 31.25

kHz it takes 0.016 s to sample the input 500 times. The function then adjusts the samples by -253,000 to avoid overflowing the variable in which they are stored. The function then calculates the RMS of the samples and returns this value. This value is used to represent the energy that the input signal has. The flowchart in Figure 12 gives a visual representation of what the function does.

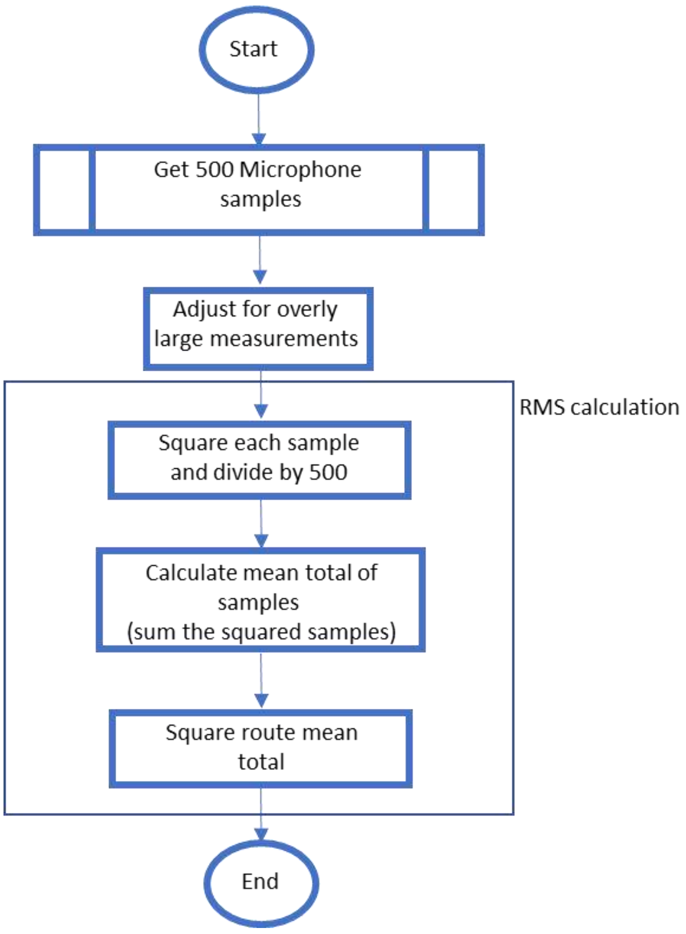
Start
Get 500 Microphone samples
Adjust for overly large measurements
RMS calculation
Square each sample and divide by 500
Calculate mean total of samples (sum the squared samples)
Square route mean total
End

Figure 12: The all_RMS function.

The sample values had to be adjusted to avoid overflow. The values that the microphone outputs are of the order of 250,000 (approximately 2^{18}). The microcontroller used is a 32-bit microcontroller. Therefore, the largest value that the microcontroller can store without overflowing is 2^{32}-1. The samples had to be squared to perform the RMS operation. 2^{18} squared is 2^{36}, which is a number so large that, if the program were to do this, the sample value would overflow and output a value not representative of the sample made: therefore the sample values had to be reduced. The choice to reduce each sample was based on experimental data collected by the student. The graphs in Figure 13 show the microphone sample values at different frequencies and volumes before and after adjustment. The lowest sample value recorded was 253,409. It was decided that the best way to scale the values down to a range unlikely to overflow was to adjust each sample by -253,000. Once each sample value was squared, they were divided by the number of values used in the operation. By doing the division before adding the squared values together the possibility of overflow was again reduced.

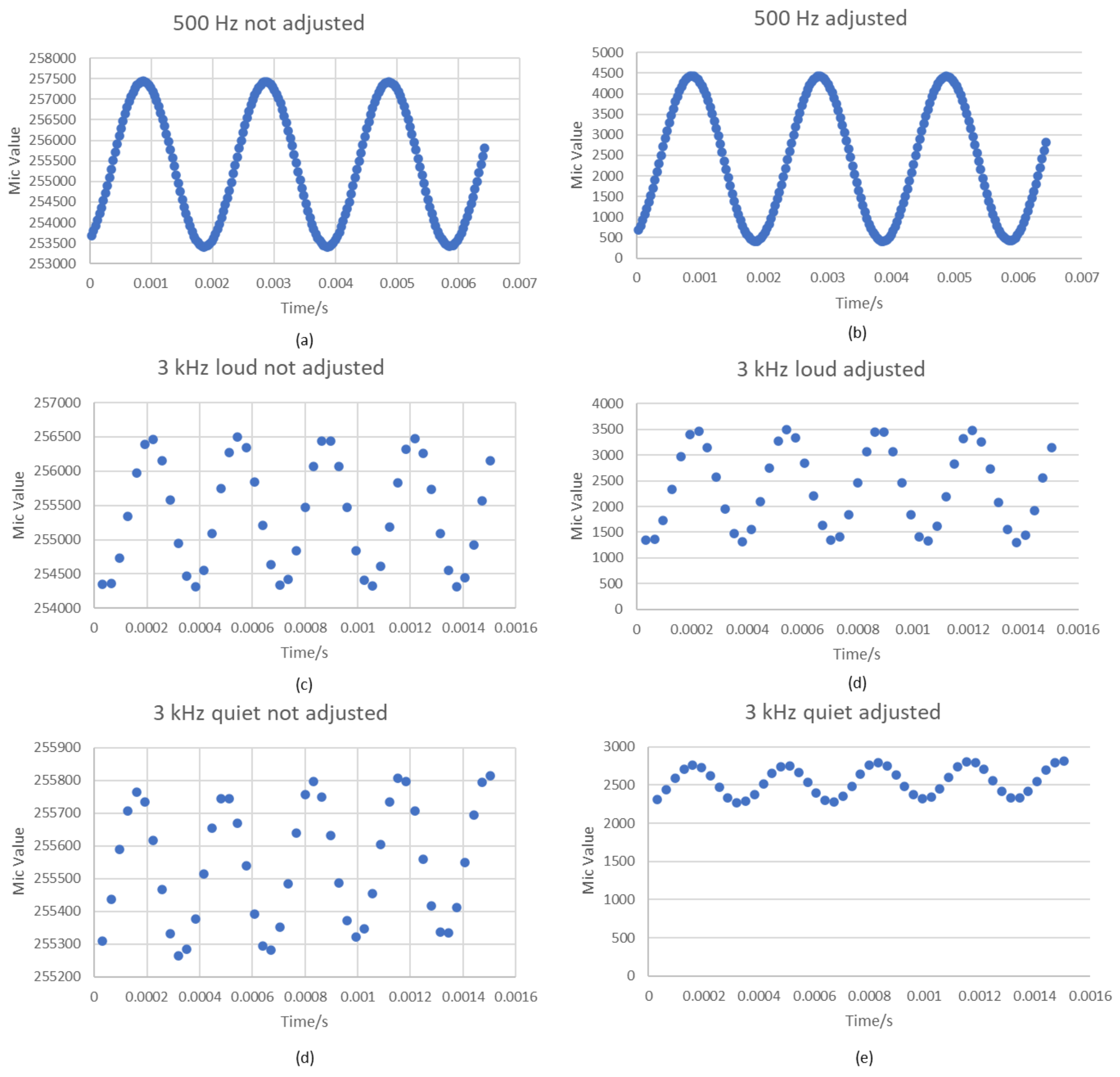

Figure 13: Adjusted and not adjusted microphone values at 500 Hz (a and b) and 3 kHz (c through e)

The system used three band pass filters to separate out the low, mid and high frequency components of the input signal. The physical implementation of the lighting system lends itself to being split into three sections, one for each frequency group. Second order passive infinite impulse response (IIR) filters were used to do this.

Second order passive filters were chosen as they give a frequency response that is fit for purpose. LCR second order filters are band pass filters with a frequency response that peaks at a certain frequency then rolls off either side of this. The roll off rate can be at the design stage of the filter implementation. These filters filter a signal such that a range of frequencies remain in the output of the filter. However, the filter ensures that as the frequencies stray further from the resonant frequency, the energy the signal carries is reduced.

The filters were designed using the program *Signal Wizard* [32], which allows the user to input values for the resistor, capacitor and inductor of a filter and select the type of filter they want, it then shows the user the frequency response. The filters used were LCR band pass filters, with peaks at 220 Hz, 1125 Hz and 2521 Hz for the low, mid and high range band pass filters respectively. The frequency responses for the low, mid and high range filters, as designed in *Signal Wizard*, are shown in Figures 14-16.

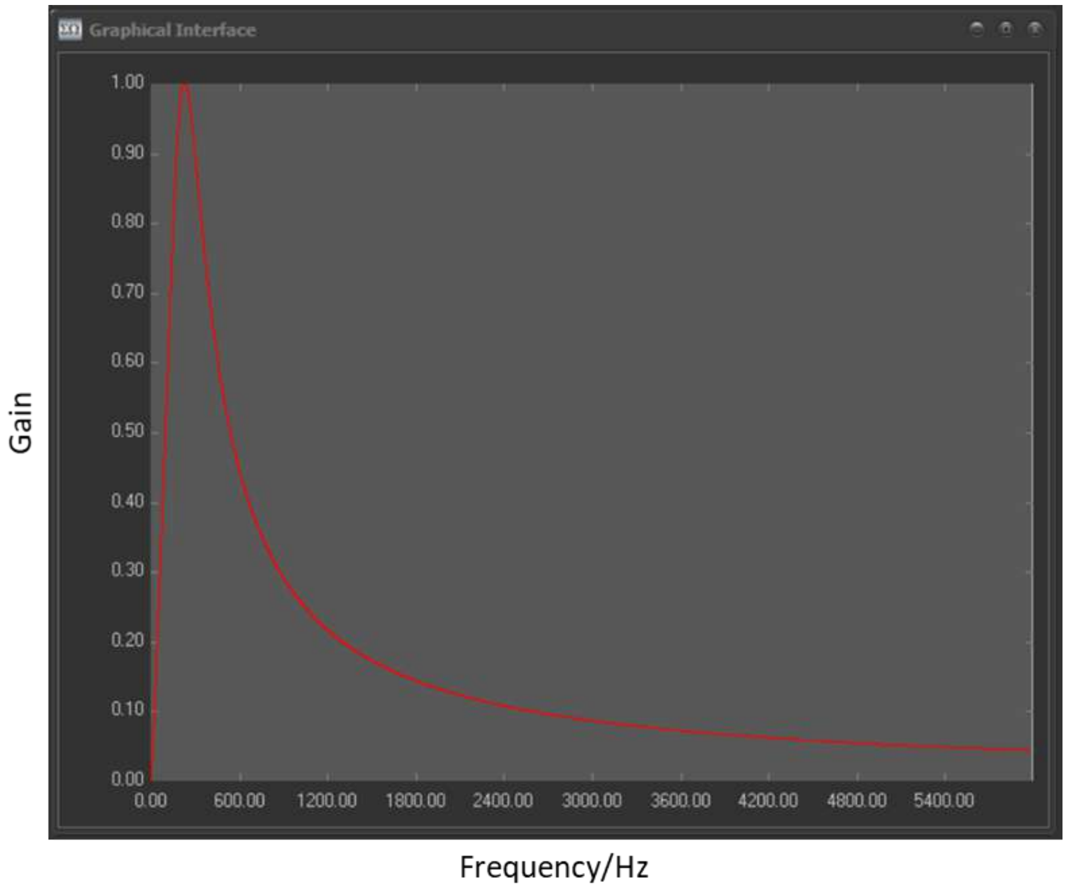

Figure 14: Low frequency band pass filter response.

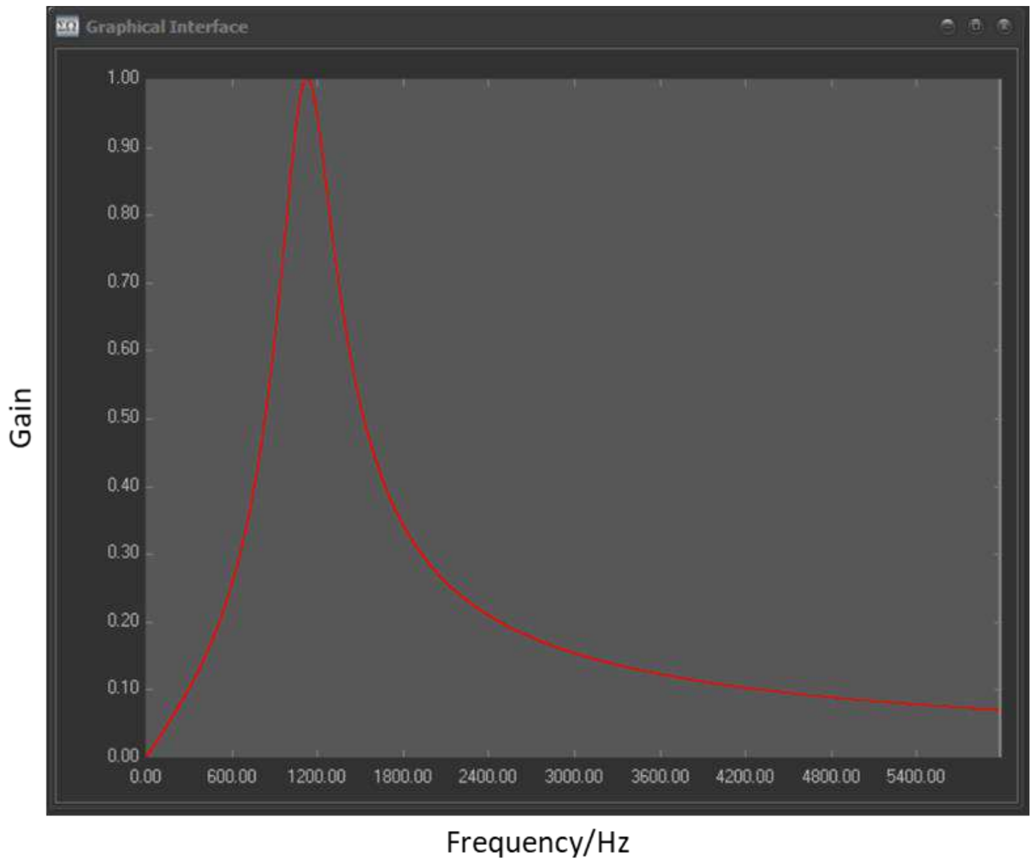

Figure 15: Mid frequency band pass filter response.

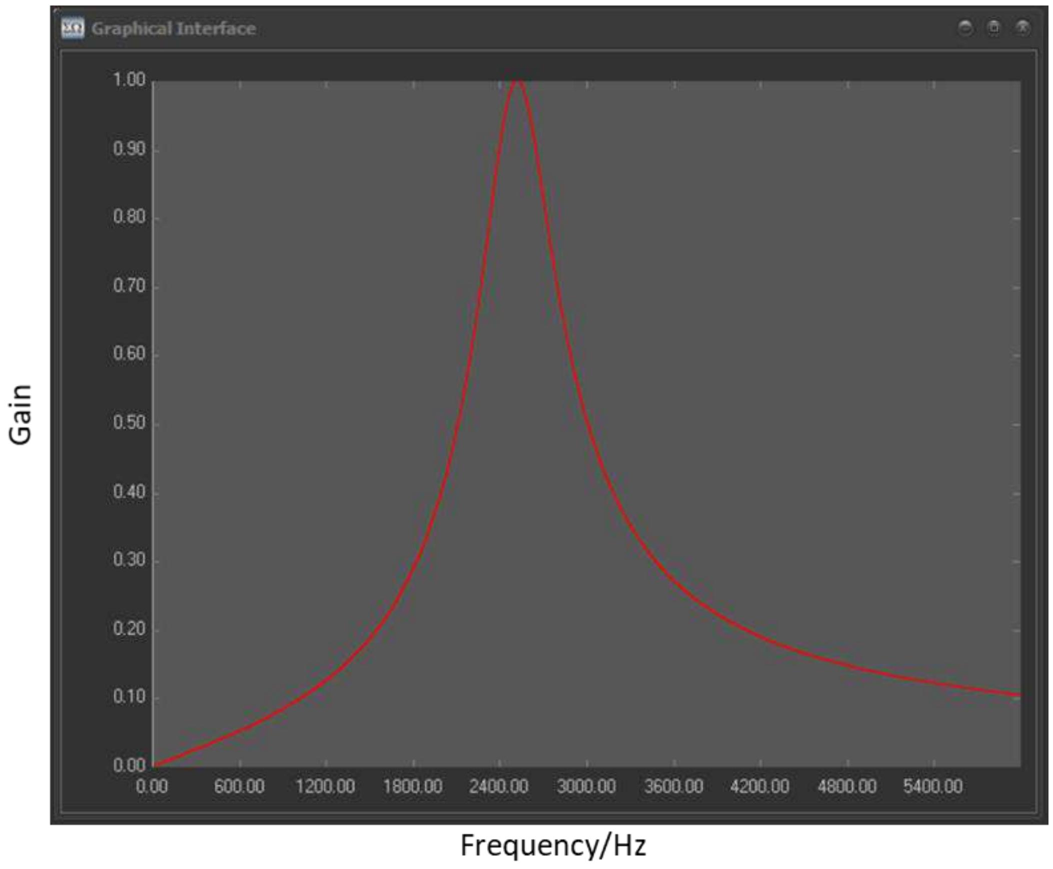

Figure 16: High frequency band pass filter response.

These filters were chosen because they covered key sections of the audio spectrum, as determined in audio engineering theory. These are shown in Figure 17. The filters cover parts of each of the three decades, which is a concept used to determine which frequencies the human ear can easily differentiate. Using these filter values emphasises to the user the fact that the system reacts to different pitches.

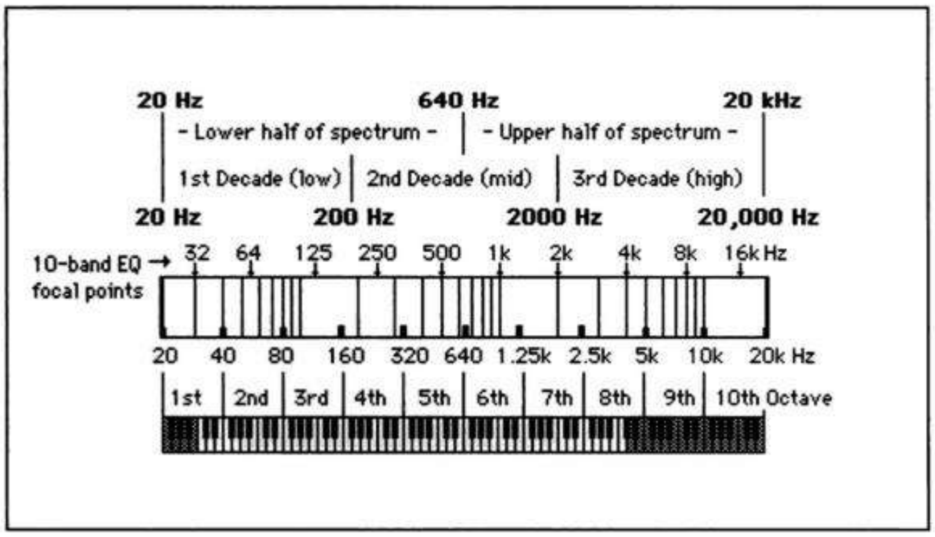

Figure 17: The audio spectrum [33].

IIR filters were chosen over finite impulse response (FIR), filters because they require significantly less processing power. A feature of IIR filters is that they are recursive, which can lead to instability. However, the fact that they are recursive means that IIR filters can be designed with significantly less coefficients compared to FIR filters, which typically have several hundred coefficients, with the same filter performance. There is less data for a processor to store and fewer operations to apply the filter to an incoming signal [7]. Since the microcontroller chosen was not a DSP focussed chip and therefore not optimised for multiply-accumulate operations (MACs), and the program performs several operations simultaneously, it was important that the computational load from the signal processing was as low as possible.

The discrete transfer function equivalent of an analogue filter's transfer function is used to implement the IIR filters used in the system. The bilinear z-transform (BZT) can be used to obtain this. It is used to remap the transfer function from the Laplace space to the z-space. This transfer function is used to implement the filters designed in *Signal Wizard*.

The BZT is performed by replacing s, the Laplace transform variable, in the Laplace transform of an analogue filter transfer function with the expression:

$$s = \frac{2}{T}\left[\frac{z-1}{z+1}\right] \tag{2}$$

where: T is the sampling period and z is the z-transform variable.

Distortion occurs, particularly at higher frequencies, when the BZT is used to transform continuous time to the discrete time equivalent. To compensate for this a prewarping factor was used. This prewarping factor, Ω*, is given by:

$$\Omega^* = \frac{2}{T}\tan(\Omega T/2) \tag{3}$$

where: Ω is the resonant frequency of the filter.

The system uses three 2nd order LCR filters which are implemented as IIR filters. The resonant frequency, Ω, of these filters is given by:

$$\Omega = \frac{1}{\sqrt{LC}} \tag{4}$$

where: L is the value of the inductor and C is the value of the capacitor.

The filters used in the system have the Laplace transfer function:

$$H(s) = \frac{sRC}{s^2LC + sRC + 1} \tag{5}$$

where: R is the value of the resistor.

Using the BZT substitution, in Equation (2), and replacing Ω, in Equation (4), with Ω*, from Equation (3), the transfer function for the IIR filter can be derived:

$$y[n] = \varepsilon_0 x[n] - \varepsilon_0 x[n-2] - \varepsilon_1 y[n-1] - \varepsilon_2 y[n-2] \tag{6}$$

where: y[n] is the output, x[n] is the input and ε_0, ε_1 and ε_2 are the identities:

$$\varepsilon_0 = \frac{a}{a+b+1} \qquad \varepsilon_1 = \frac{2-2b}{a+b+1} \qquad \varepsilon_2 = \frac{b-a+1}{a+b+1} \tag{7}$$

where: The identities a and b are:

$$a = 2\frac{RC}{T} \qquad b = \left(\frac{2}{T\Omega^*}\right)^2 \tag{8}$$

where: The variables R, C, T and Ω* are as stated above [7].

A filter function was written to output a value for the energy in the audio signal once it has passed through an LCR filter. The coefficients for the filter, shown in Equation (7), are calculated on each iteration of the program In order to do this, the resonant frequency, prewarping factor, a and b are calculated, using Equations (4,3,8) respectively. The flowchart in Figure 18 gives a pictorial representation of this process.

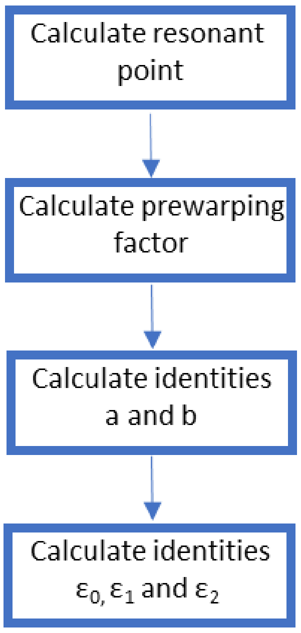
Calculate resonant point
Calculate prewarping factor
Calculate identities a and b
Calculate identities $\varepsilon_{0,}\ \varepsilon_1$ and ε_2

Figure 18: Calculating the coefficients for the filter.

The function was written in this way so that it could be used to implement second order LRC filters with any values of inductor, capacitor and resistor. As the system uses three filters it uses this same function three separate times, with different values for the inductor, capacitor and resistor.

Once the filter coefficients have been calculated, the microphone is sampled 300 times. These samples are adjusted for reasons discussed earlier in this section, the filter is applied to these samples and then the RMS of the output of the signal is calculated. The flowchart in Figure 19 outlines this process.

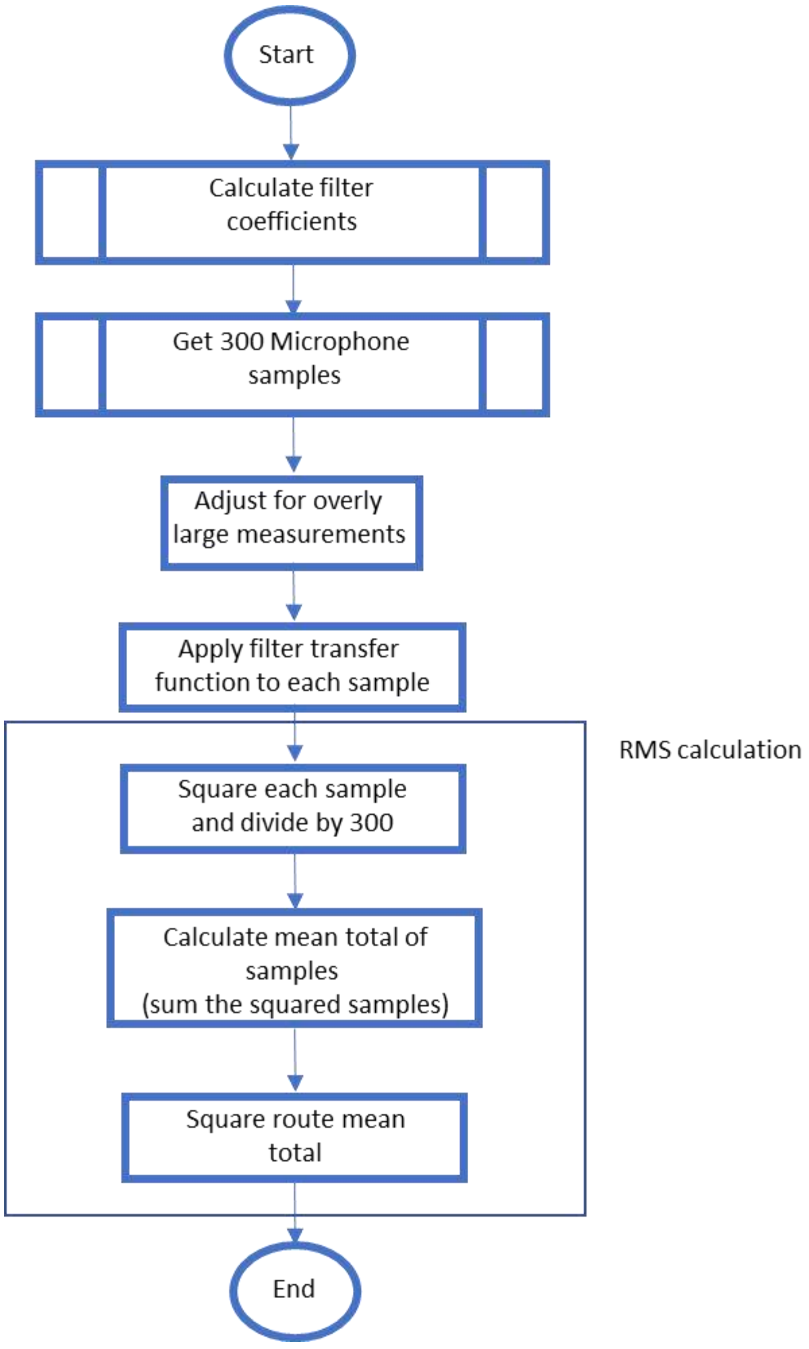
Start
Calculate filter coefficients
Get 300 Microphone samples
Adjust for overly large measurements
Apply filter transfer function to each sample
RMS calculation
Square each sample and divide by 300
Calculate mean total of samples (sum the squared samples)
Square route mean total
End

Figure 19: The filter function.

3.2.5. Physical design and implementation

The full system consists of 2 major parts: the control box and the lighting display. This is pictured in Figure 20. The control box and lighting display are powered separately and connected over a cable that carries the control data for the lighting display.

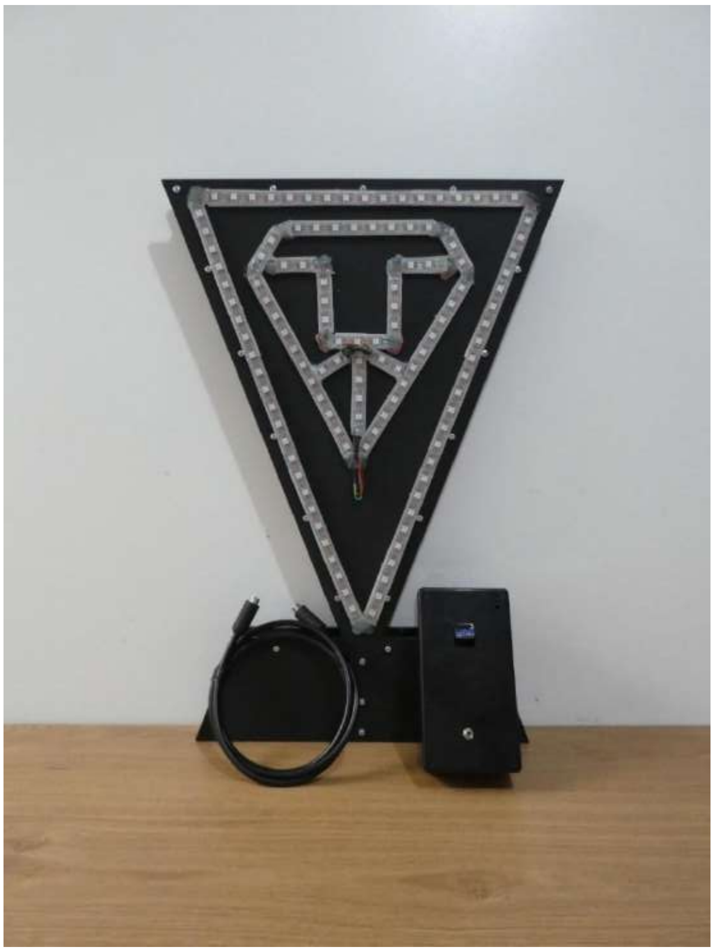

Figure 20: The full system.

On top of the control box there is a switch, used to change mode, the microphone, and a hole to allow for access to the reset button on the Arduino. On the front there are 2 micro-USB ports, used for programming the Arduino, as well as the power socket and a 4-pin mini DIN connector that outputs the data for the lighting display. components are all labelled and shown in Figures 21a-b.

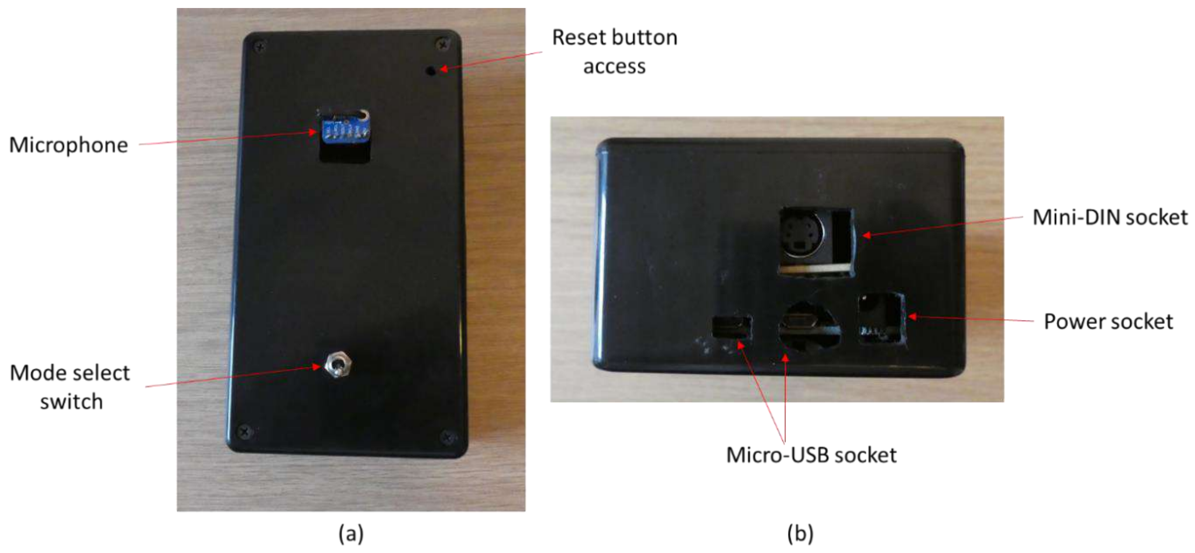

Figure 21: Top view (a) and side view (b) of the control box.

The box itself is a plastic enclosure that was modified for the purpose of the project. The control box houses the Arduino Due with a shield, made on stripboard, that the microphone and switch are connected to. Making a shield on stripboard allowed the student to quickly and cheaply modify the design as the project progressed. The shield and Arduino are pictured in Figure 22. It was not possible to transfer this design to a PCB due to the time constraints of the project. The Arduino is powered using a supply that is rated for 2 A at 9 V, as is recommended by the manufacturer [2].

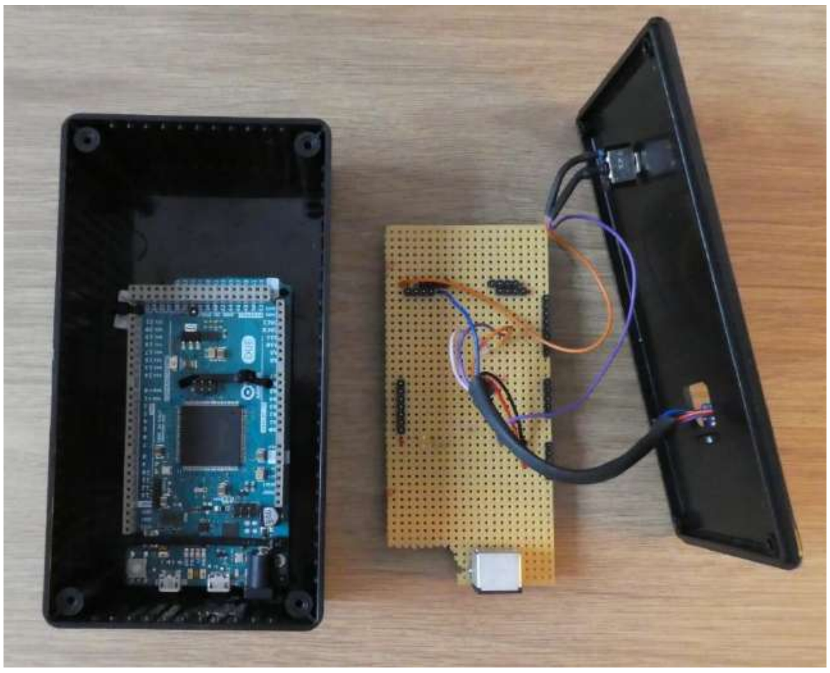

Figure 22: Control box contents.

The control box connects to the lighting display over a coaxial cable connected using 4-pin mini DIN connectors that transmits the data for the lights. 4-pin mini DIN connectors were chosen as they have a low profile and were available for a cheap price from a trusted supplier. The data for the lights takes the form of a digital signal sent at a rate of 800 kbps over the Unipolar NRZ protocol. Unipolar NRZ goes high if the bit being sent is 1 and low if the bit being sent is 0 [34], therefore the control signal for the lights has a maximum frequency of 800 kHz. A coaxial cable was used because it has a shield and was easily available to the student when the system was being constructed. The shield reduces the chance of the signal being interfered with, and the wrong data arriving at the lighting display, or the signal interfering with other electronic equipment in its vicinity.

The LED strips are mounted on a wooden frame. The wooden frame consists of two triangular panels made of MDF, one in front of the other, connected by hex pillars. The LED strips are mounted on the front inside their silicon casing, and attached to the frame with adhesive strips. The frame was painted black to minimise its visual impact. The frame has a stand, constructed in a similar way to the frame, so it can be placed upright on a table or other surface. It also has wall lugs on the back so it can be mounted on a wall. Including a stand and wall lugs ensures that the lighting display can be easily displayed in a variety of locations. The fully constructed frame is shown in Figure 23.

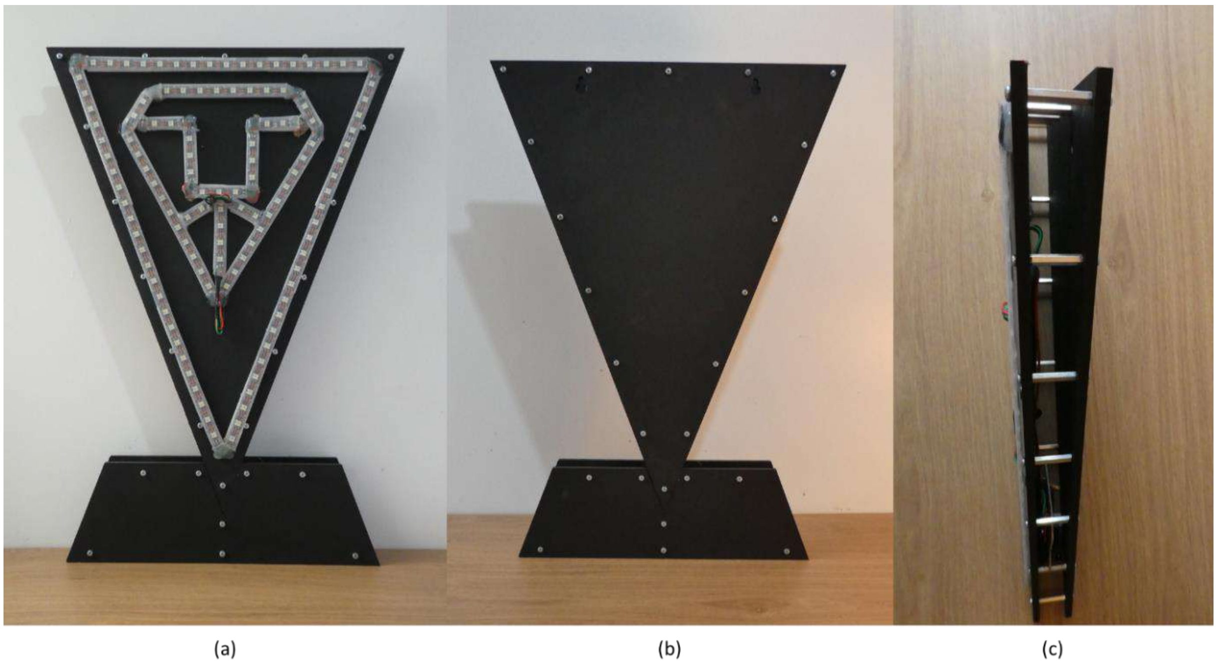

Figure 23: Front (a), rear (b) and side (c) of the frame.

A stripboard with the circuit for the logic level shifter, detailed in section 3.2.1 of this report, was mounted between the two sheets, with connectors for power and data. The stripboard was mounted in place with Velcro, meaning it could be removed and reattached easily during the prototyping phase. Any long cables were run between the two panels ensuring minimal impact on the frame's appearance. A photograph of the strip board and the cable runs is shown in Figure 24.

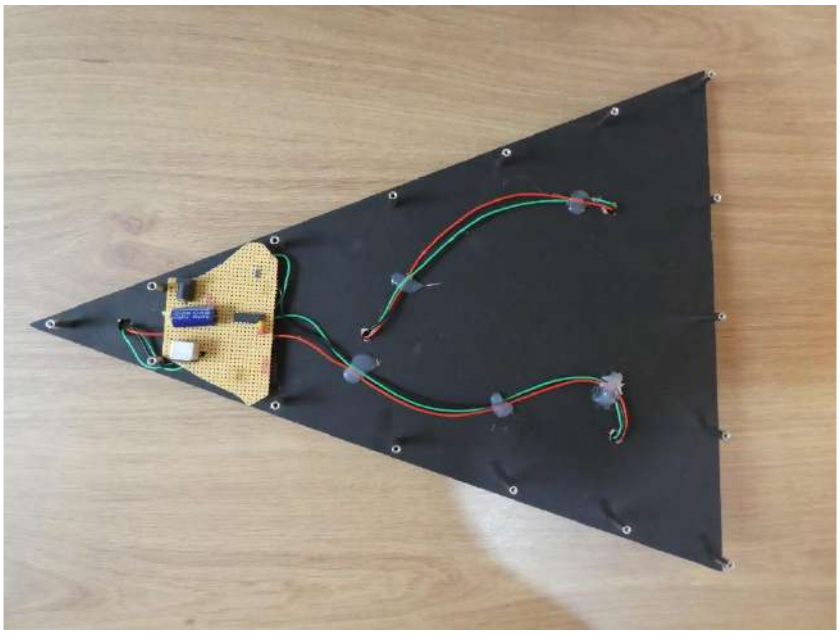

Figure 24: Centre of frame, one panel removed.

Where the strips had to be connected with wires, hot glue was used. The hot glue covers the wires, it diffuses their appearance so they have minimal visual impact. It also stops any moisture getting inside the silicon casing protecting the LED strips. One of these connections is photographed in Figure 25.

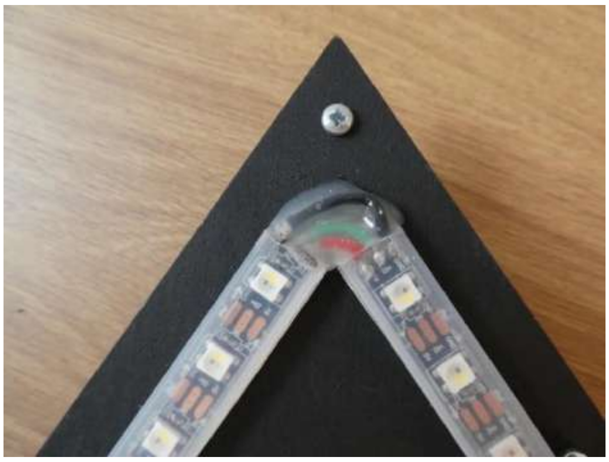

Figure 45: Connection between two strips

4. Conclusions

4.1. Testing and system capabilities

The system works well. In mode 1 the lights switch on when a sound is detected by the microphone, they remain on while this sound is constant, and switch off when the sound stops. The delay between the sound being detected and the lights coming on is not noticeable whilst the system is operating. The system is limited by the volume of the sound. If the microphone measures a sound above a certain volume, the troughs of the sound wave can underflow after the program adjusts the values. The reason for the adjustment of the values is detailed in Section 3.2.4. Figure 26 shows the effects of underflow on the sound wave measured. A logarithmic scale was used for the mic values so that the trend can be seen. The measurements shown in Figure 26 were taken when a 500 Hz tone was played through a portable speaker at a high volume close to the microphone. It is unlikely the system will have to detect a sound like this as it is too loud to be comfortable for the human ear.

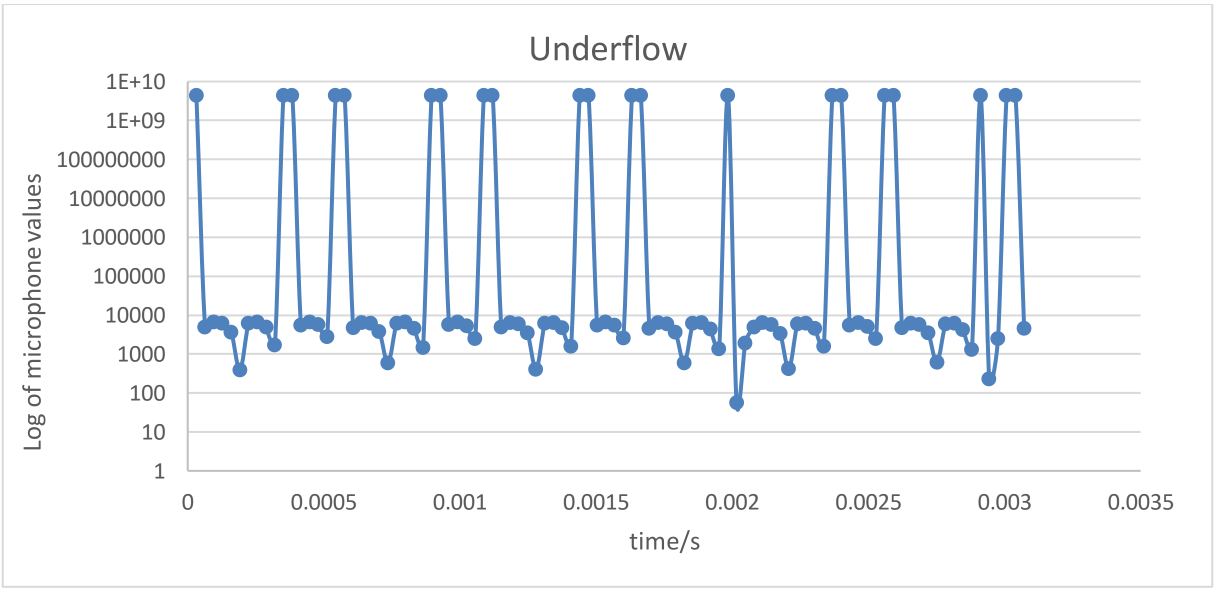

Figure 26: Microphone values showing the effects of underflow on the system.

In mode 2 the system is able to detect frequencies, and the lights turn on for the frequency they are designed to do so for. It can also detect different frequencies in a song composed of multiple simultaneous sounds, reacting as designed to create an interesting and exciting pattern that evolves constantly. As with mode 1, any delay between the sound being detected and the lights reacting is not noticeable, and it is clear that the way the lights behave is affected by the pitch of the sound that the microphone detects. Mode 2 can also be affected by the underflow limitation discussed above. Figure 27 shows the frequency response of the system in mode 2. The system was tested by playing individual tones at different frequencies, applying the digital filters to the sampled signal and calculating the energy that the signal had after filtering. This data, along with experimentation using different tones and songs, were used to set the turn on and turn off levels for each of the sets of lights. Figure 27 clearly demonstrate that the system is able to distinguish different tones when sampling an audio signal.

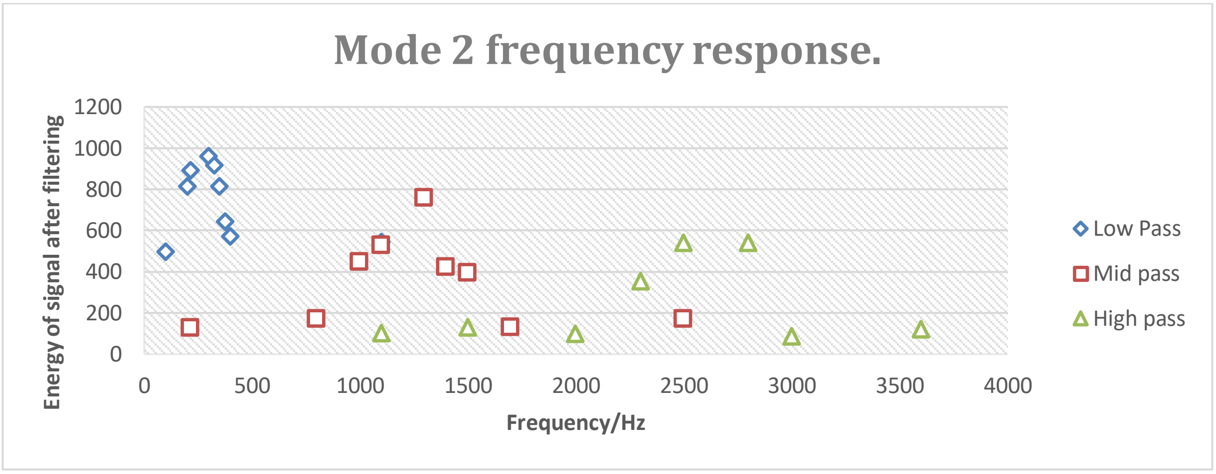

Figure 27: Frequency response of the system in mode 2.

4.2. Success or failure

Since the system uses addressable LEDs as a key component of its functionality, this aim was easily fulfilled in the project. Another aim was to provide an intuitive control surface for the system. This was also achieved: although the control surface is more simplistic than what was originally designed, the user still has control over the mode of the system in the final project. Additionally, the system is simple to interact with. The final aim of the project was originally to analyse the audio signal in four different ways, however within the timescales given the student managed analysis of only two: the energy and frequency.

The system can be used as a display and reacts in an aesthetically pleasing and exciting way to an audio input. It is easy to install and set up and meets the design brief. It can be marketed and used as intended in the real world. Some examples of uses are as entertainment and as an educational tool for people learning about music and audio engineering. For these reasons the project can be considered a success.

5. References

[1] "Arduino - Introduction", *Arduino.cc*, 2020. [Online]. Available: https://www.arduino.cc/en/guide/introduction. [Accessed: 28- Apr- 2020].

[2] "Arduino Due | Arduino Official Store", *Store.arduino.cc*, 2020. [Online]. Available: https://store.arduino.cc/due. [Accessed: 28- Apr- 2020].

[3] *SPH0645LM4H-B Datasheet*, 3rd ed. Itasca: Knowles, 2017.

[4] P. Burgess, *The Magic of NeoPixels*, 1st ed. Adafruit, 2013.

[5] *SAM3X/SAM3A Datasheet*, 1st ed. San Jose: Atmel, 2015, pp. 519-525, 528, 574-591.

[6] K. Zhu, *SK6812RGBW specification*, 1st ed. Dongguan: DONGGUANG OPSCO OPTOELECTRONICS CO., LTD, 2015.

[7] P. Gaydecki, *Digital Signal Processing, Lecture Notes*. Manchester: The University of Manchester, 2019, pp. 86-114, 232-242.

[8] "Art definition and meaning", *Collinsdictionary.com*, 2020. [Online]. Available: https://www.collinsdictionary.com/dictionary/english/art. [Accessed: 28- Apr- 2020].

[9] O. Eliasson, *The Weather Project*. London: The Tate Modern, 2003.

[10] C. Holler, *Test Site*. London: The Tate Modern, 2006.

[11] M. Windsor, "Art of Interaction: A Theoretical Examination of Carsten Höller's Test Site", *Tate Papers*, no. 15, 2011. Available: https://www.tate.org.uk/research/publications/tate-papers/15/art-of-interaction-a-theoretical-examination-of-carsten-holler-test-site. [Accessed 28 April 2020].

[12] G. Levin, *Audiovisual Environment Suite*. Exhibited at various venues, 2000.

[13] G. Levin, "Audiovisual Environment Suite", *Archive.aec.at*, 2000. [Online]. Available: https://archive.aec.at/prix/showmode/35396/. [Accessed: 28- Apr- 2020].

[14] *Walter Giers - Electronic Art*. Berlin: Electronic Beats, 2017.

[15] Shajeebtm, "README.md", *Arduino 32 band audio spectrum visualizer/analyzer*, GitHub, 2019.

[16] buzzandy, "Music Reactive LED Strip", *Arduino Project Hub*, 2017.

[17] "OCS-2 - nozoïd", *nozoïd*, 2017. [Online]. Available: http://nozoid.com/ocs-2/. [Accessed: 28- Apr- 2020].

[18] chnry, "OCS-2: A Digital, Semi Modular Synthesizer", *Arduino Project Hub*, 2017.

[19] "Maker Movement - Maker Faire", *Maker Faire*, 2020. [Online]. Available: https://makerfaire.com/maker-movement/. [Accessed: 30- Apr- 2020].

[20] "The Maker Movement in Education: Designing, Creating, and Learning Across Contexts", *Harvard Educational Review*, vol. 84, no. 4, pp. 492-494, 2014. Available: 10.17763/haer.84.4.b1p1352374577600 [Accessed 28 April 2020].

[21] D. Dougherty, "The Maker Movement", *Innovations: Technology, Governance, Globalization*, vol. 7, no. 3, pp. 11-14, 2012. Available: 10.1162/inov_a_00135 [Accessed 28 April 2020].

[22] L. Ada, *Adafruit I2S MEMS Microphone Breakout*. Adafruit, 2017.

[23] delsauce, "README.md", *Arduino Due HiFi*, GitHub, 2018.

[24] "1591DSBK - Plastic Enclosure, Multipurpose, Plastic, 50 mm, 80 mm, 150 mm, IP54", *Farnell*, 2020. [Online]. Available: https://uk.farnell.com/hammond/1591dsbk/box-abs-black-150x80x50mm/dp/1426574?ost=1426574&ddkey=https%3Aen-GB%2FElement14_United_Kingdom%2Fsearch. [Accessed: 28- Apr- 2020].

[25] A. Industries, "Adafruit NeoPixel Digital RGBW LED Strip - Black PCB 60 LED/m", *Adafruit*, 2020. [Online]. Available: https://www.adafruit.com/product/2837?length=1. [Accessed: 28- Apr- 2020].

[26] "Adafruit NeoPixel Digital RGBW LED Strip - Black PCB 60 LED/m", *The Pi Hut*, 2020. [Online]. Available: https://thepihut.com/products/adafruit-neopixel-digital-rgbw-led-strip-black-pcb-60-led-m. [Accessed: 28- Apr- 2020].

[27] "Adafruit Industries LLC 658", *Digikey.co.uk*, 2020. [Online]. Available: https://www.digikey.co.uk/products/en?keywords=1528-1519-ND. [Accessed: 28- Apr- 2020].

[28] *SN54AHCT125, SN74AHCT125 - Datasheet*, 4th ed. Dallas: Texas Instruments, 1995, pp. 1-2.

[29] C. Goss, "Octave Notation", *Flutopedia.com*, 2019. [Online]. Available: http://www.flutopedia.com/octave_notation.htm. [Accessed: 28- Apr- 2020].

[30] *I2S Bus Specification*. Philips Semiconductors, 1986, p. 1.

[31] "Introduction to Signal Levels", *Discovery of Sound in the Sea*, 2020. [Online]. Available: https://dosits.org/science/advanced-topics/introduction-to-signal-levels/. [Accessed: 28- Apr- 2020].

[32] P. Gaydecki, *Signal Wizard*. Manchester: Signal Wizard Systems, 2020.

[33] S. Stark, *Live sound reinforcement*. Boston, Mass.: Course Technology, 2008, p. 20.

[34] G. Rubner, *Data Networking - Lecture Notes*. Manchester: The University of Manchester, 2020, Lecture 15.

6. Appendices

6.1. Appendix – COVID-19

Due to the nature of this project the student had to make use of many of the facilities that the University offers, particularly for the hardware development aspect of the project. The student was able to mitigate the effects of the University closure due to COVID-19, by ensuring that all the hardware development was completed by the 17th March. As the coronavirus pandemic developed, they focussed their efforts on ensuring the hardware development for their project was complete, spending the majority of the week before the closures in C34 to do this. Due to the efforts of the student the hardware for the project was completed to as good a standard as it would have been whether the University had been open or not.

The quality of the report has been affected by the closure of the University facilities. The student was unable to access resources and material that the library would have had available. These would have helped justify decisions made when developing the project. Furthermore, the student was unable to access software, such as Altium and MATLAB, that would have enabled the student to produce better quality circuit diagrams and graphs. Not having access to these resources meant the student had to edit or create content in software that was not designed for the task. An example of how this effected the quality of the report is the circuit diagram in section 3.2.1. The student took a screenshot of an existing circuit diagram that they produced in Altium and had to edit it in PowerPoint. Not only was this time consuming but it also created an unprofessional finish for the diagram.

Therefore, the students project was primarily affected in terms of shifting timescales and of the quality of the report.

6.2. Appendix – Source code

```
#include <Adafruit_NeoPixel.h>
#include <Arduino.h>
#define LED_PIN     6
#define LED_COUNT  118
#define BRIGHTNESS 50
// Declare our NeoPixel strip
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRBW + NEO_KHZ800);

//overall RMS

double all_RMS(){

  uint32_t mic_val[500];
  uint32_t mic[500];

  float out[500];
  float squared[500];
  double mean_total=0;
  double RMS=0;

  uint32_t read_ready = 0;

  //read mic value
  for (int i=0; i<500; i++){
    label:
    read_ready = 0;
    read_ready = REG_SSC_SR>>4;
    read_ready = read_ready & 0b1;
    if (read_ready==1){
     mic[i] = REG_SSC_RHR;
    }
    else {
    goto label;
    }
    }
```

```
//adjust for overly large value
 for (int i; i<500; i++){
   mic_val[i]= (mic[i] - 253000);

}

  //RMS
for (int i=0; i<500; i++){
  squared[i] = sq(mic_val[i])/500;

}
mean_total=0;
  for (int i=0; i<500; i++){
  mean_total = mean_total+squared[i];

}

  RMS = sqrt(mean_total);

  return RMS;

}

//Filters



double low_pass(){                  // band pass filter for frequencies be centred around
200 Hz
  uint32_t mic_val[300];
  uint32_t mic[300];
  float ind = 0.05;
  float cap = 0.00001;
  float res = 80;

  float t = 0.000032;
  float omega = 1/(sqrt(ind*cap));
  float omega1 = (2/t)*tan(omega*t/2);



  float a = (2/t)*res*cap;
  float b=sq(2/(t*omega1));
  float c=a/(a+b+1);
  float d=(2-(2*b))/(a+b+1);
  float e=(1+b-a)/(a+b+1);

  float out[300];
  float squared[300];
  double mean_total=0;
  double RMS=0;

  uint32_t read_ready = 0;



//read mic value
  for (int i=0; i<300; i++){
    label:
    read_ready = 0;
    read_ready = REG_SSC_SR>>4;
    read_ready = read_ready & 0b1;
    if (read_ready==1){
     mic[i] = REG_SSC_RHR;
    }
    else {
    goto label;
    }
    }
//adjust for overly large value
for (int i; i<300; i++){
  mic_val[i]= (mic[i] - 253000);
}
    //filtering

    out[0] = c*mic_val[0];
    out[1] = (c*mic_val[1])-(d*out[0]);
  for (int i=2; i<300; i++){



    out[i] = (c*mic_val[i]) - (c*mic_val[i-2]) - (d*out[i-1]) - (e*out[i-2]);

  }
  //RMS
for (int i=0; i<300; i++){
  squared[i] = sq(out[i])/300;

}
mean_total=0;
  for (int i=0; i<300; i++){
  mean_total = mean_total+squared[i];

}

  RMS = sqrt(mean_total);

  return RMS;
}



double mid_pass(){                  // band pass filter for frequencies be 1.2 kHz
  uint32_t mic_val[300];
  uint32_t mic[300];

  float ind = 0.02;
  float cap = 0.000001;
  float res = 50;
  float t = 0.000032;

  float omega = 1/(sqrt(ind*cap));
  float omega1 = (2/t)*tan(omega*t/2);

  float a = (2/t)*res*cap;
  float b=sq(2/(t*omega1));
  float c=a/(a+b+1);
  float d=(2-(2*b))/(a+b+1);
  float e=(1+b-a)/(a+b+1);

  float out[300];
  float squared[300];
  double mean_total=0;
  double RMS=0;

  uint32_t read_ready = 0;

  for (int i=0; i<300; i++){
    label:
    read_ready = 0;
    read_ready = REG_SSC_SR>>4;
    read_ready = read_ready & 0b1;
    if (read_ready==1){
     mic[i] = REG_SSC_RHR;
    }
    else {
    goto label;
    }
    }

for (int i; i<300; i++){
  mic_val[i]= (mic[i] - 253000);

}
```

```
    out[0] = c*mic_val[0];
    out[1] = (c*mic_val[1])-(d*out[0]);
  for (int i=2; i<300; i++){
    out[i] = (c*mic_val[i]) - (c*mic_val[i-2]) - (d*out[i-1]) - (e*out[i-2]);
  }

for (int i=0; i<300; i++){
  squared[i] = sq(out[i])/300;

}
mean_total=0;
  for (int i=0; i<300; i++){
  mean_total = mean_total+squared[i];

}

  RMS = sqrt(mean_total);

  return RMS;

}

double high_pass(){                  //band pass filter for frequencies below 2.5kHz
  uint32_t mic_val[300];
  uint32_t mic[300];

  float ind = 0.04;
  float cap = 0.0000001;
  float res = 130;
  float t = 0.000032;
  float omega = 1/(sqrt(ind*cap));
  float omega1 = (2/t)*tan(omega*t/2);
  float a = (2/t)*res*cap;
  float b=sq(2/(t*omega1));
  float c=a/(a+b+1);
  float d=(2-(2*b))/(a+b+1);
  float e=(1+b-a)/(a+b+1);
  float out[300];
  float squared[300];
  double mean_total=0;
  double RMS=0;

  uint32_t read_ready = 0;

  for (int i=0; i<300; i++){
    label:
    read_ready = 0;
    read_ready = REG_SSC_SR>>4;
    read_ready = read_ready & 0b1;
    if (read_ready==1){
     mic[i] = REG_SSC_RHR;
    }
    else {
    goto label;
    }
    }

for (int i; i<300; i++){
  mic_val[i]= (mic[i] - 253000);


}


    out[0] = c*mic_val[0];
    out[1] = (c*mic_val[1])-(d*out[0]);
  for (int i=2; i<300; i++){


    out[i] = (c*mic_val[i]) - (c*mic_val[i-2]) - (d*out[i-1]) - (e*out[i-2]);


  }

for (int i=0; i<300; i++){
  squared[i] = sq(out[i])/300;

}
mean_total=0;
  for (int i=0; i<300; i++){
  mean_total = mean_total+squared[i];

}

  RMS = sqrt(mean_total);

  return RMS;
}

void setup() {
  // set up strips
  strip.begin();           // INITIALIZE NeoPixel strip object (REQUIRED)
  strip.show();            // Turn OFF all pixels ASAP
  strip.setBrightness(50); // Set BRIGHTNESS to about 1/5 (max = 255)

//set up microphone
  REG_PIOB_WPMR = 0x50494F00; // DISABLE WRITE PROTECT PIO
REGISTERS
  REG_PIOB_ABSR = 0b00000000000000000000000000000000;  //SET AS
PERIPHERAL A FOR SSC
  REG_PIOB_PER = 0b00000000000000000000000000000000; //ENSURE PIO
ISNT ENABLED
  REG_PIOB_PDR = 0b00000000000001110000000000000000; //DISABLE PIO ON
SSC PINS
  REG_PMC_PCER0 = 0b1<<26; //ENABLE CLOCK FOR SSC
  REG_SSC_WPMR = 0x53534300; //DISABLE WRITE PROTECT TO SSC
REGISTERS
  REG_SSC_CR = 0b0000000000000001; //ENABLE RECEIVE
  REG_SSC_CMR = 21; //set clock divider
  REG_SSC_RFMR = 0b00011000001011110000000010010001;   //LSB UP      SET
DATA LENGTH (WORD LENGTH) TO 18       SET NO LOOP      SET MSB
```

```
SAMPLED FIRST        SET 1 DATA WORD PER FRAME

  REG_SSC_RCMR = 0b00011111000000000000010001000100; //LSB UP       SET
MCK AS CLOCK IN FOR SSC    SET RECEIVE CLOCK AS CONTINUOUS     SET
GATE AS CONTINOUS       SET TO START RECEIVING DATA ON falling RF   SET
NO STOP CONDITION SET NO DELAY PERIOD FOR RF DIVIDEDR
  pinMode(22,OUTPUT);
  digitalWrite(22, LOW);
  Serial.begin(9600);

  //Run turn on sequence
    strip.setPixelColor(0, 255, 0, 0,0);
    strip.show();
    delay(200);
  for(int x=1; x<118; x++){
    strip.setPixelColor(x-1,0,0,0,0);
    strip.setPixelColor(x,255,0,0,0);
    strip.show();

  }
  strip.clear();
  strip.show();
  pinMode(27,INPUT_PULLUP);

}

uint32_t magenta = strip.Color(255,0,255,0);

int off_level=3050;
int on_level=3130;
int switch_value;

void loop() {
int switch_value;
switch_value = digitalRead(27);
```

```
double RMS;
uint32_t colour;
uint16_t col1 = 54913;
uint16_t col2 = 54913;
uint16_t col3 = 54913;
uint32_t nosat;
uint32_t somesat ;
uint32_t lotssat ;

while(switch_value == 1){

int col_val=20;        //start value for colour (0-255)

int col_count=0;       //count for iterations of the loop
RMS = all_RMS();       //sample RMS of audio in
if (RMS>on_level){     //lower off value than on value allowing to reduce switching
between off and on when close to the threshold
while(RMS>off_level){
  while (col_count<117){

    colour = strip.Color(col_val,0,col_val,0);    //get 32 bit value for colour
                                                  //the numbers input into the call function represent
RGBW

    col_val=col_val+2;
    strip.fill(colour,0,118);          //send colour info to all leds
    col_count++;
    RMS = all_RMS();
    if(RMS<off_level){
      break;
    }
    strip.show();                //Tell LEDs to activate with any data they have received
  }
      if(RMS<off_level){
      break;
      }
      while (col_count>0){

    colour = strip.Color(col_val,0,col_val,0);

    col_val=col_val-2;
   strip.fill(colour,0,118);
    col_count--;
    RMS = all_RMS();
    if(RMS<off_level){
      break;
    }
    strip.show();
  }
      if(RMS<off_level){
      break;
      }

 RMS=all_RMS();

}
switch_value=digitalRead(27);         //check mode switch
if (switch_value==0){
  break;
}
}

while(RMS<on_level){
  strip.clear();           //clear any data saved by LEDS

  strip.show();
 RMS=all_RMS();

}

switch_value=digitalRead(27);
}
```

```
int low;
int mid;
int high;

while (switch_value == 0){
  low=low_pass();              //sample low pass filter
  if (low>900){
    col1=col1+50;              //set hue value
    nosat = strip.ColorHSV(col1);   //get 32bit colour saturation and value default to
255
    strip.fill(nosat,0,65);       //send data to strips, the numbers used by the call function
                                //represent the colour, the address of the LED to start at and the
number of LEDS to set respectively
    strip.show();
  }
  if (low<800){

    strip.fill(0,0,65);
    strip.show();
  }

  mid=mid_pass();
  if (mid>550){
    col2=col2+50;
    somesat = strip.ColorHSV(col2,190,255);
    strip.fill(somesat,78,34);
    strip.show();
  }
  if (mid<400){
    strip.fill(0,78,34);
    strip.show();
  }
    high=high_pass();
  if (high>400){
    col3=col3+50;
    lotssat = strip.ColorHSV(col3,100,255);
    strip.fill(lotssat,65,13);
    strip.fill(lotssat,112,6);
    strip.show();
  }
  if (high<350){
    strip.fill(0,65,13);
    strip.fill(0,112,6);
    strip.show();
 }
  switch_value=digitalRead(27);
}
}
```