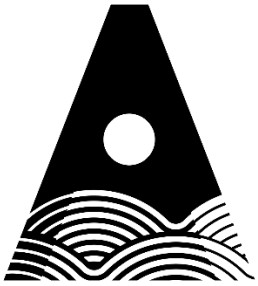

Ollscoil
Teicneolaíochta
an Atlantaigh

Atlantic
Technological
University

# Micro Synth

by

Charles Mulder

S00232211

Bachelor of Engineering

in

Electronic and Computer Engineering

Project supervisor: Fabian Connolly

10 May 2025

# Table of Contents

# Abstract

This project details the design and implementation of a microcontroller based digital synthesiser, the Micro Synth. The system receives musical input from a MIDI 1.0-compliant keyboard and generates corresponding audio output via a PIC16F1718 microcontroller. The microcontroller receives MIDI messages, interprets note-on and note-off events, and outputs audio signals using wavetable synthesis. Three waveform types are supported: sine, sawtooth, and square waves. Audio signals are amplified through an LM386 audio amplifier and filtered using a 2-pole Butterworth low-pass filter before being delivered to a speaker. Ethical considerations such as hearing loss, environmental impact, and electronic waste were explored, and sustainable design choices, such as breadboard-based assembly, were prioritised. The final system demonstrates a cost-effective and educational approach to embedded digital audio synthesis.

# List of Figures

# List of Tables

# List of Equations

# Student Declaration

By inserting my name below, I declare that this material, which I now submit for assessment, is my own work and that any assistance I received in its preparation is fully acknowledged and disclosed in the document. To the best of my knowledge and belief, all sources have been properly acknowledged, and the assessment task contains no plagiarism.

I understand that plagiarism, collusion, and/or copying is a grave and serious offence in the Institute and am aware that penalties could include a zero mark for this module, suspension or expulsion from the Institute.

I acknowledge that this assessment submission may be transferred and stored in a database for the purposes of data-matching to help detect plagiarism. I declare that this document was prepared by me for the purpose of partial fulfilment of requirements for the Degree Programme I am registered on. I also declare that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study either at IT Sligo or another college.

Signed: Charles Mulder ..... Date: 10 May 2025

# Introduction

Modern music production increasingly relies on digital audio technology, with affordable synthesisers enabling musicians to explore a wide range of timbres. This project combines a personal interest in music with embedded systems design by developing a microcontroller based digital synthesiser called the Micro Synth. The goal is to explore the fundamental building blocks of digital audio synthesis and demonstrate how real-time audio generation can be implemented on resource-constrained hardware.

The device uses the MIDI 1.0 protocol for input. MIDI (Musical Instrument Digital Interface) allows electronic musical instruments to communicate, using standardised messages to convey information such as note pitch and velocity. The Micro Synth accepts MIDI input from an external keyboard and produces audio output using wavetable synthesis.

The main objectives are implementing MIDI parsing, creating efficient waveform generation using lookup tables, and amplifying and filtering audio signals for speaker output. Design decisions also considered ethical factors such as environmental sustainability and user hearing safety.

The report begins by presenting the concept selection process, followed by an ethical analysis considering the device's personal and environmental impact. The following section details the hardware components, including the MIDI input interface, microcontroller setup, LED indicators, amplifier, and low-pass filter. The software section covers MIDI interpretation, wavetable synthesis, oscillator design, and sampling rate generation. The report concludes with a summary of outcomes and design limitations.

Figure 1 shows a high-level block diagram of the processes and components that make up the Micro Synth.

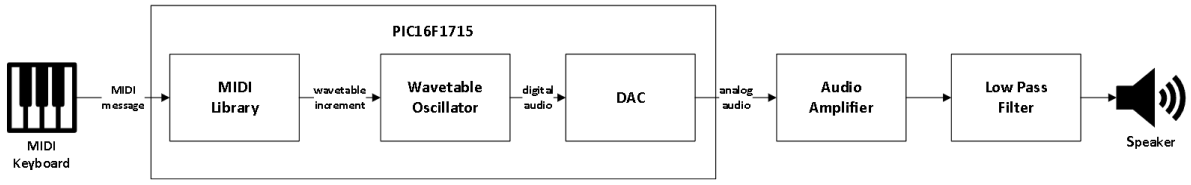


*Figure 1: Block diagram*

# Concept Selection

Three project concepts were considered, all stemming from a passion for music and an interest in electronics. The potential projects were a digital synthesizer, a MIDI keyboard, and a musical doorbell.

*Table 1: Project concept selection using a weighted Pugh matrix*

| **Characteristic** | **Weight** | **Digital Synthesizer** | **MIDI Keyboard** | **Musical Doorbell** |
|---|---|---|---|---|
| Ease of hardware implementation | 3 | S | - | + |
| Ease of software implementation | 2 | S | + | ++ |
| Familiarity | 1 | S | + | -- |
| Cost effectiveness | 2 | S | -- | - |
| Project interest | 5 | S | - | -- |
| Abundance of resources | 4 | S | ++ | + |
| Likelihood of success | 5 | S | S | + |
| Public appeal | 4 | S | S | -- |
| **Total +** | | **0** | **4** | **5** |
| **Total -** | | **0** | **-4** | **-7** |
| **Total Score** | | **0** | **0** | **-2** |
| **Weighted Total +** | | **0** | **11** | **16** |
| **Weighted Total -** | | **0** | **-12** | **-22** |
| **Weighted Score** | | **0** | **-1** | **-6** |

Table 1 shows a variant of the Pugh matrix used to select the strongest concept and understand the reasons for the choice (Wijnia, et al., 2009). The column marked *Characteristic* lists the criteria identified as critical for the project's successful outcome. The column marked as *Digital Synthesizer* serves as the reference or datum against which all other concepts are compared and is therefore ranked as *S* for all characteristics. The *MIDI Keyboard* and *Musical Doorbell* columns are ranked against the *Digital Synthesizer* as relatively more or less likely to satisfy each criterion.

The *Total* + and *Total* – rows contain each project's total minus and plus signs. A tie resulted between the *Digital Synthesizer* and *MIDI Keyboard* projects, as indicated by the *Total Score* row. To break the tie, a *Weight* column was added to rank the importance of each characteristic, which also serves as a score multiplier. The *Weighted* + and *Weighted* – row shows the scores after applying the weight multiplier. The result is communicated in the *Weighted Score* row, indicating the *Digital Synthesizer* as the project most likely to succeed.

# Ethical Considerations

In keeping with the code of ethics prescribed by Engineers Ireland, engineers have a responsibility to consider both the short and long-term impact of their products on individuals, society and the environment. The following sections explore the ethical implications of the Micro Synth's use, manufacture and disposal.

## Hearing Loss

An audio device can damage hearing. The Hearing Research journal (Reynolds & Bielefeld, 2023) relates music-induced hearing loss to noise-induced hearing loss, with louder volumes and longer exposure time increasing risk. Further evidence presented by (Schmuziger, et al., 2006) and (Kähäri, et al., 2003) suggests that musicians are at a higher risk than the general public. A further study by (Ergun, et al., 2024) comparing the preferred listening volume and resultant sound intensity levels across music genres found that electronic music had the highest dB values. Therefore, as the Micro Synth is most likely to be used by electronic musicians, the evidence suggests a higher-than-average risk of hearing loss, which must be communicated to the end user.

## Environmental Impact

Mining elements to produce the components required for electronic devices consume natural resources and harm the environment. An article published in the Journal of Environmental Management (Worlanyo & Jiangfeng, 2021) lists the destruction of natural habits, water pollution and air pollution as significant environmental impacts of mining the minerals required for electronic components. Another study by (Wang, et al., 2023) reported that semiconductor manufacturing consumes large quantities of water and energy, with water identified as the primary barrier to the industry's sustainable development. Given these environmental costs, the impact of having a printed circuit board manufactured overseas cannot be justified for a student project that requires only a single unit.

## Waste

All electronic devices inevitably end up as waste. The 2024 global e-waste report compiled by (Baldé, et al., 2024) for the United Nations Institute for Training and Research (UNITAR) found that 62 million tonnes of e-waste was produced in 2022. When compared to 2010, this represented an increase of 82%. According to current trends, the report predicts a further 32% increase to 82 million tonnes by 2030. To curb the trend, the European Union passed the WEEE (Crowe, et al., 2003) directive promoting collecting and recycling unwanted electronics. A decade later, the global e-waste report shows that only 22.3% of e-waste was formally collected and recycled in an environmentally sound manner, with recycled rare earth elements only satisfying 1% of the global demand. Assembling the Micro Synth circuit on a breadboard promotes easy reuse and repurposing of components, offering a practical way to reduce waste and contribute to more sustainable electronic design practices.

# Hardware

Figure 2 shows the complete circuit schematic with a PIC16F1718 at the core surrounded by four smaller circuits: a MIDI-in circuit, three LED status indicators, an audio amplifier circuit, and a low-pass filter. The following sections will discuss each of the components.

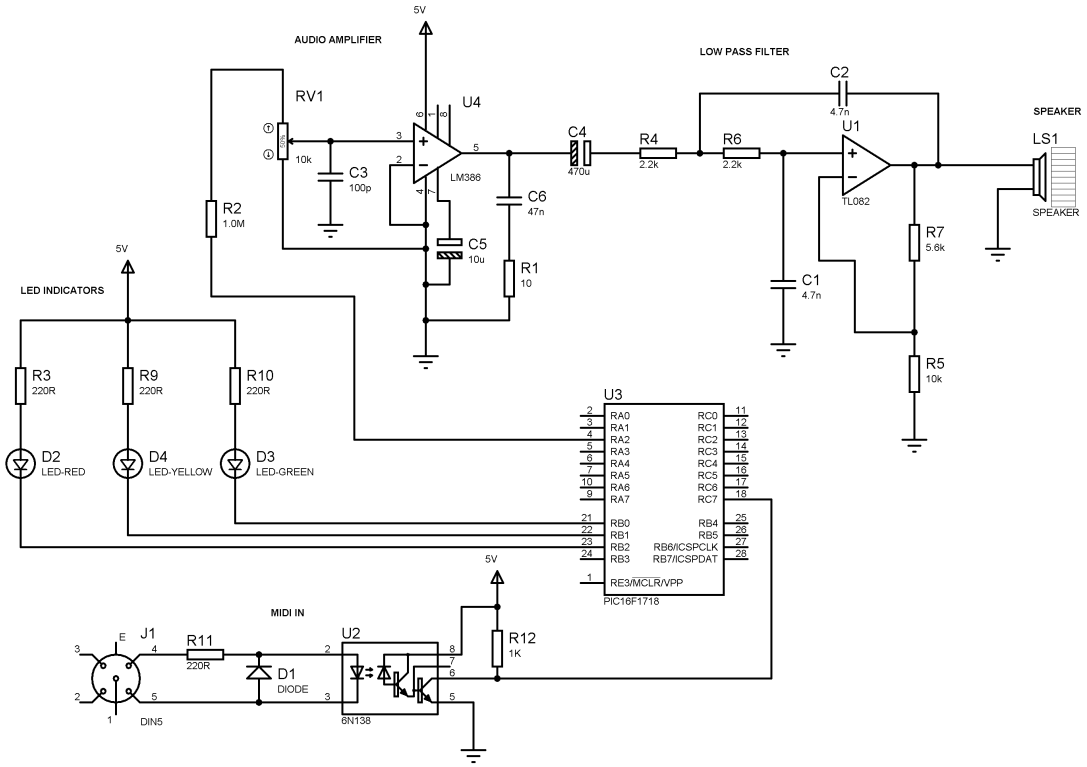


*Figure 2: Complete circuit schematic*

## MIDI-In

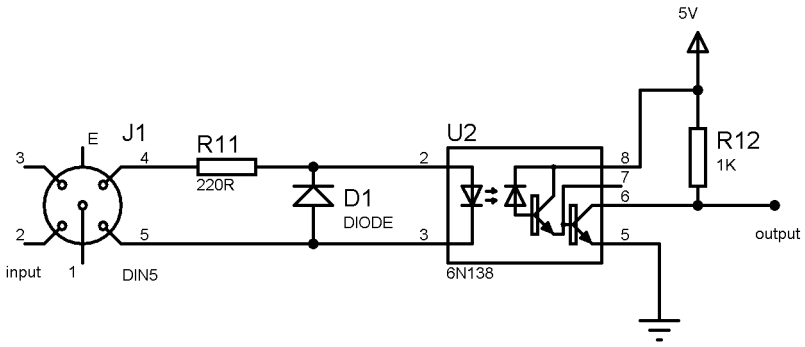


*Figure 3: Schematic of MIDI input*

Figure 3 is a schematic of the MIDI-input component designed by the (MIDI Manufacturers Association, 1996). MIDI messages are received from an external MIDI keyboard via a DIN5 connector. The digital signal is transferred to pin 18 of the PIC16F1718 via a 6N138 opto-isolator. The opto-isolator provides electrical isolation between the MIDI device and the PIC. Thereby protecting the circuit from noise or hum caused by ground loops.

## Microcontroller

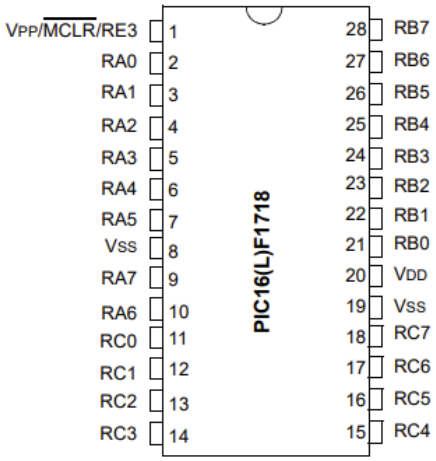


*Figure 4: Pin diagram of PIC16F1718*

Figure 4 shows the 28-pin diagram of the PIC16F1718, which was selected due to its 8-bit digital-to-analogue converter (DAC) to output audio, Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART) to receive MIDI messages and enough program memory (28kB) to store the required wavetables. The smallest possible microcontroller was selected to minimise energy consumption and wasted resources. Special note should be made of a 10 µF decoupling capacitor between pin 20 ($V_{DD}$) and ground to filter noise from the power supply and prevent voltage dips.

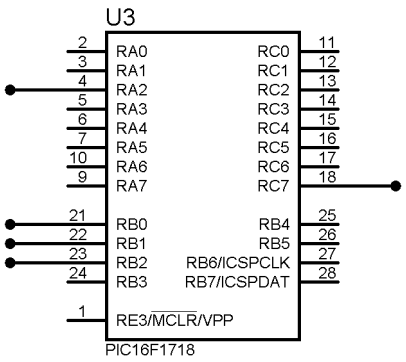


*Figure 5: Schematic of PIC16F1718*

Figure 5 shows the connected pins. Pin 18 receives MIDI messages via the EUSART module. Pin 4 outputs an audio signal generated by the 8-bit DAC converter. Pins 21, 22 and 23 control a green, yellow and red LED to communicate the device's status to the operator.

The PIC is programmed and debugged using a MPLAB PICkit 5 In-Circuit Debugger. An arrow on the connector indicates pin 1 on the PICkit. Table 2 shows the required mapping from the pins of the PICkit to the pins on the PIC16F1718.

*Table 2: PICkit 5 connection to PIC16F1718*

| **PIC16F1718** | | **PICkit 5** | |
|---|---|---|---|
| **Pin** | **Description** | **Pin** | **Description** |
| 1 | MCLR/VPP | 1 | NMCLR |
| 20 | $V_{DD}$ | 2 | $V_{DD}$ |
| 8, 19 | $V_{SS}$ (ground) | 3 | Ground |
| 28 | PGD (ICSPDAT) | 4 | PGD |
| 27 | PGC (ICSPCLK) | 5 | PGC |
| - | Do not connect | 6 | Do not connect |

## LED Indicators

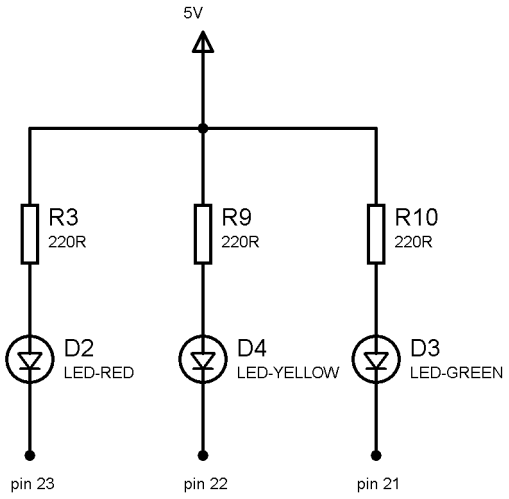


*Figure 6: Schematic of LED indicators*

Figure 6 shows the 3 LED indicators. All three LEDs will flash when powered up to ensure the indicators work. The green LED will turn on when a MIDI note-on message is received and turn off when the corresponding MIDI note-off message is received. The yellow LED indicates an EUSART framing error, which occurs when a stop bit is not received at the expected instant. A framing error does not prevent the device from receiving additional MIDI messages. The framing error will automatically be reset when the next character is received, thereby turning off the yellow LED. The red LED indicates an EUSART input buffer overrun, which will stop the device from receiving further MIDI messages. In this case, a device restart is recommended.

## Audio Amplifier

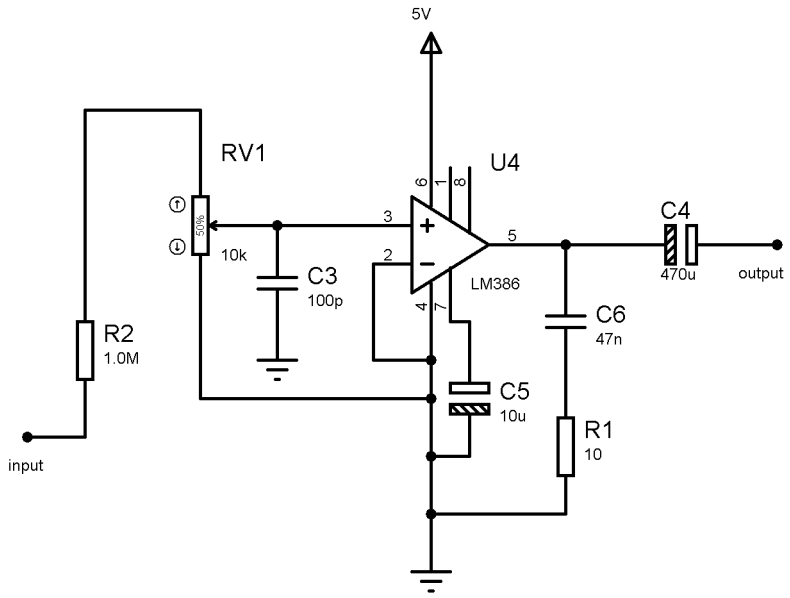


*Figure 7: Schematic of audio amplifier*

Figure 7 shows the audio amplifier circuit with an LM386 at its core. The (Texas Instruments, 2023) datasheet reveals that the gain is set to 20 but can be adjusted between 20 and 200 by connecting an external resistor and capacitor between pins 1 and 8. The specified input voltage range is between -0.4V and 0.4V. During testing, it was observed that the sine waves were clipped, indicating that the input voltage was exceeding the maximum limit. Through trial and error, it was found that adding a 1MΩ resistor at R2 effectively reduced the input voltage, thereby preventing clipping at the output.

## Low-Pass Filter

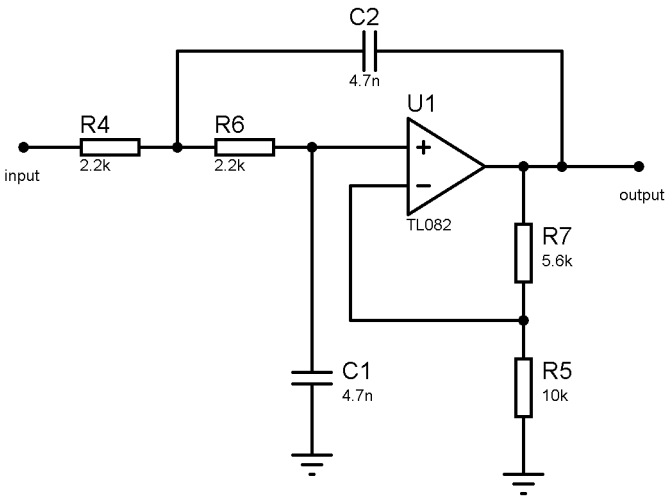


*Figure 8: Schematic of low-pass filter*

Figure 8 shows a modified Sallen-Key low-pass filter as provided by (Smith, 2003) in his book on digital signal processing. The sampling rate has been set to 29.4kHz due to the limitations imposed by the PIC16F1718's slow floating-point arithmetic. The filter's cut-off has been set at the Nyquist frequency of 14.7kHz. A Butterworth low-pass filter has been selected due to its absence of passband ripple and the relatively fast roll-off.

*Equation 1: Input resistor value for Sallen-Key low-pass filter*

$$R = \frac{k_1}{C f_c}$$
$$= \frac{0.1592}{4.7nF \times 14.7kHz}$$
$$= 2.3k\Omega$$

*Equation 2: Feedback resistor value for Sallen-Key low-pass filter*

$$R_f = R_1 k_2$$
$$10k\Omega \times 0.586$$
$$5.86k\Omega$$

Equation 1 calculates the value of the $R_4$ and $R_6$ resistors for the non-inverting input of the operational amplifier. Equation 2 calculates the value of the feedback resistor $R_7$ with $R_5$ arbitrarily set to 10kΩ. Table 3 shows the constants used for calculating the resistor values of a Butterworth filter.

*Table 3: Constants used for resistor calculations of Butterworth low-pass filter*

| **Poles** | **Stage** | **$k_1$** | **$k_2$** |
|---|---|---|---|
| 2 | 1 | **0.1592** | **0.586** |
| 4 | 1 | 0.1592 | 0.152 |
| 4 | 2 | 0.1592 | 1.235 |

Figure 9 plots the filter's simulated frequency response, showing that roll-off will start around 3kHz, attenuating frequencies within the audible spectrum. Therefore, this low-pass filter will not be sufficient for general audio applications. However, considering that a sine wave only contains a single frequency component and the highest pitch produced by the Micro Synth will be 523.251 Hz, the 2-pole Butterworth filter will suffice.

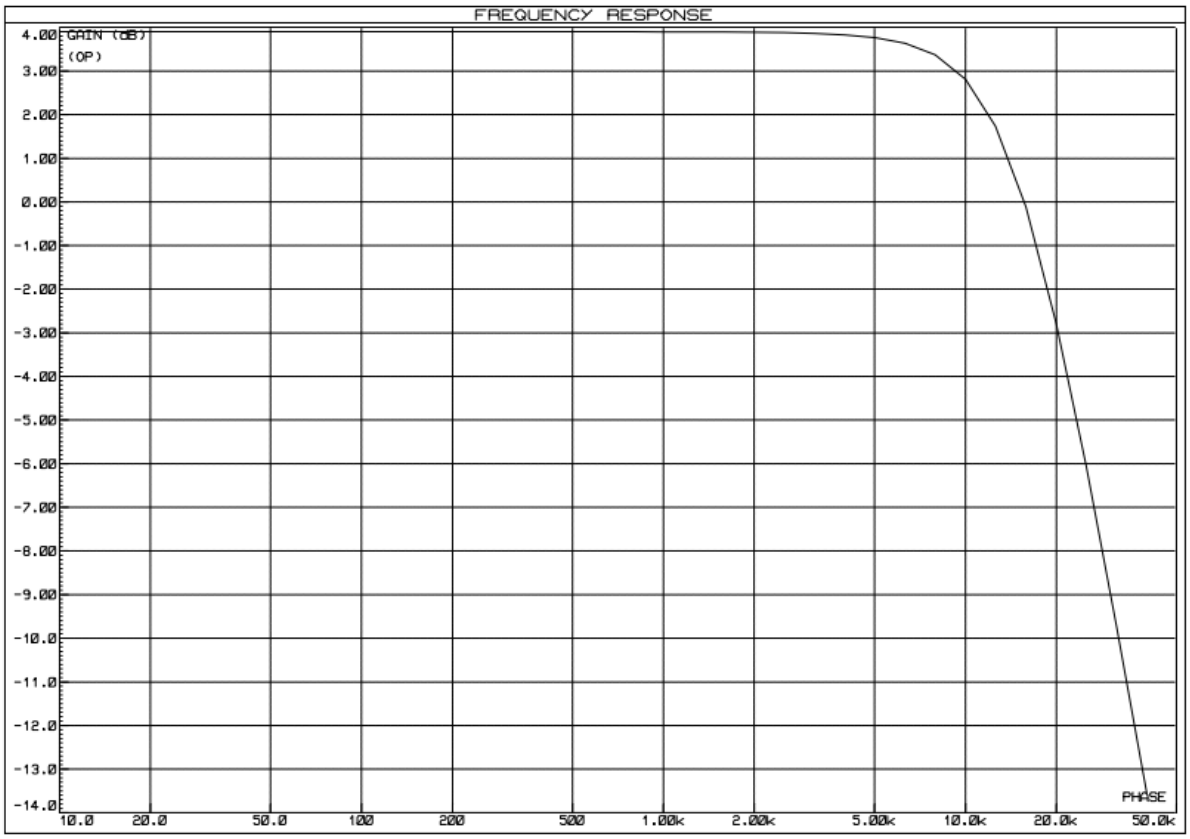


*Figure 9: Simulated frequency response of 2 pole Butterworth filter with cutoff frequency set to 29.4kHz*

## MIDI Keyboard


*Figure 10: MIDI Tech MIDI  Start Music 25*

Figure 10 is an image of the keyboard used to drive the circuit. The keyboard is MIDI 1.0 compliant and unable to produce sound independently. It has two transpose buttons that can transpose 3 octaves down and 4 octaves up. The transpose buttons alter the MIDI note numbers, as per Table 4 below.

*Table 4: MIDI note values for the transposed musical note C on the MIDI Start Music 25*

| Transposition | Keyboard note | MIDI note | Music note |
|---|---|---|---|
| --- | | 0 | |
| --- | | 12 | |
| --- | | 24 | |
| -- | Low C | 12 | |
| -- | Middle C | 24 | Lowest C on piano |
| -- | High C | 36 | |
| - | Low C | 24 | |
| - | Middle C | 36 | |
| - | High C | 48 | |
| none (default) | Low C | 36 | |
| none (default) | Middle C | 48 | |
| None (default) | High C | 60 | Middle C on piano |
| + | Low C | 48 | |
| + | Middle C | 60 | |
| + | High C | 72 | |
| ++ | Low C | 60 | Middle C on piano |
| ++ | Middle C | 72 | |
| ++ | High C | 84 | |
| +++ | Low C | 72 | |
| +++ | Middle C | 84 | |
| +++ | High C | 96 | |
| ++++ | Low C | 84 | |
| ++++ | Middle C | 96 | |
| ++++ | High C | 108 | Highest C on piano |

# Software

The source code is available online at github.com/charlesmulder/microsynth.

## MIDI

The (MIDI Manufacturers Association, 1996) states that MIDI messages are transferred at a rate of 31.25 (±1%) kBaud using asynchronous EUSART transfer with a start bit, 8 data bits, and a stop bit. As is common with asynchronous transfer, the start bit is a logical 0; the stop bit is a logical 1 and bytes are sent LSB first.

*Table 5: BAUD rate formulas (Microchip Technology, 2020)*

| Configuration bits | | | EUSART Mode | BAUD Rate Formula |
|---|---|---|---|---|
| **SYNC** | **BRG16** | **BRGH** | | |
| 0 | 0 | 0 | 8-bit/async | $\frac{FOSC}{64(SPBRG+1)}$ |
| 0 | 0 | 1 | 8-bit/async | $\frac{FOSC}{16(SPBRG+1)}$ |

The Baud Rate Generator (BRG) on the PIC16F1718 is an 8-bit or 16-bit timer that supports both the asynchronous and synchronous EUSART operation. Table 5 shows the configuration bits and baud rate formula for 8-bit asynchronous transfer. Equation 3 shows the baud rate formula rearranged to calculate the value of the SPBRG register.

*Equation 3: SPBRG value for desired Baud rate*

$$SPBRG = \frac{FOSC}{64 \times Baud\ rate} - 1$$

Table 6 shows the required values when setting the period of the baud rate timer via the SPBRG register. Due to the speed of the virtual COM ports on Windows, a baud rate of 9.6 kBaud is used during Proteus simulation. However, as specified by the (MIDI Manufacturers Association, 1996), a baud rate of 31.25 kBaud is used when interacting with a MIDI device.

*Table 6: Baud rate calculations*

| FOSC | Desired Baud Rate | Multiplier | SPBRG | Calculated Baud Rate | Error |
|---|---|---|---|---|---|
| 16MHz | 9.6kHz | 64 | **25.04** | 9,615.38 | 0.15625 |
| 16MHz | 31.25kHz | 64 | **7** | 31,250.00 | 0 |

The Micro Synth implements a subset of the MIDI 1.0 specification: note-on and note-off messages. Table 7 shows the binary format of a status byte for a note-on and a note-off MIDI message, each followed by 2 data bytes. It is worth noting that a note-on message with a velocity of 0 is equivalent to a note-off message.

*Table 7: MIDI status bytes for channel voice messages*

| Status Byte | Data Bytes | Description |
|---|---|---|
| 1000nnnn | 2 | Note-off |
| 1001nnnn | 2 | Note-on |

*nnnn: N-1, where N=channel*

Figure 11 shows the two types of MIDI messages compatible with the Mico Synth: status bytes and data bytes. Status bytes establish the purpose and meaning of the data bytes that follow. The most significant bit of a status byte is set to 1, whereas the most significant bit of a data byte is set to 0.

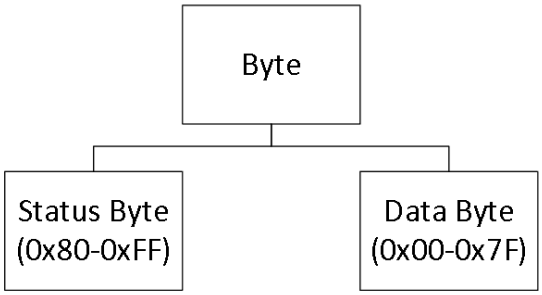


*Figure 11: MIDI message types*

The Advanced Linux Sound Architecture (ALSA) provides audio and MIDI functionality to the Linux operating system. The ALSA utilities *aconnect*, *aseqdump* and *amidi* were used to examine the content of the MIDI messages sent by the MIDI keyboard. Please see the appendix for detailed instructions on connecting and viewing the MIDI output of a virtual keyboard called *vkeybd*.

Table 8 shows the output when pressing and releasing middle C twice, followed by pressing and releasing concert pitch A once. MIDI channel 1 was used for the first sequence, and channel 2 was used for the second.

*Table 8: MIDI data when pressing and releasing one key at a time*

| Note | Hex | Decimal | Event | Channel | Data |
|---|---|---|---|---|---|
| C | 90 3C 2E | 144 60 49 | Note-on | 0 | Note 60, velocity 46 |
| C | 80 3C 40 | 128 60 64 | Note-off | 0 | Note 60, velocity 64 |
| C | 90 3C 2E | 144 60 46 | Note-on | 0 | Note 60, velocity 46 |
| C | 80 3C 40 | 128 60 64 | Note-off | 0 | Note 60, velocity 64 |
| A | 90 45 18 | 144 69 24 | Note-on | 0 | Note 69, velocity 24 |
| A | 80 45 40 | 128 69 64 | Note-off | 0 | Note 69, velocity 64 |
| C | 91 3C 26 | 145 60 38 | Note-on | 1 | Note 60, velocity 38 |
| C | 81 3C 40 | 129 60 64 | Note-off | 1 | Note 60, velocity 64 |
| C | 91 3C 1E | 145 60 30 | Note-on | 1 | Note 60, velocity 30 |
| C | 81 3C 40 | 129 60 64 | Note-off | 1 | Note 60, velocity 64 |
| A | 91 45 20 | 145 69 32 | Note-on | 1 | Note 69, velocity 32 |
| A | 81 45 40 | 129 69 64 | Note-off | 1 | Note 69, velocity 64 |

When multiple notes are played, one after the other, without releasing any keys, the MIDI keyboard omits sending the same status byte for each key and only sends data bytes instead. The MIDI specification refers to this feature as a running status, enabling a complete MIDI message to only consist of data bytes.

Table 9 shows the output when pressing and holding middle C, followed by pressing and holding concert pitch A. The middle C key is released first, followed by the key for concert pitch A. Notice that a status byte is not present for note A.

*Table 9: MIDI data when pressing and holding multiple keys*

| Note | Hex | Decimal | Event | Channel | Data |
|---|---|---|---|---|---|
| C | 90 3C 2A | 144 60 42 | Note-on | 0 | Note 60, velocity 42 |
| A | **00** 45 1D | **0** 69 29 | Note-on | 0 | Note 69, velocity 29 |
| C | 80 3C 40 | 128 60 64 | Note-off | 0 | Note 60, velocity 64 |
| A | **00** 45 40 | **0** 69 64 | Note-off | 0 | Note 69, velocity 64 |

Figure 12 shows the software flowchart of the code executed on each EUSART receiver interrupt, triggered when a MIDI key is pressed or released.

The yellow LED is turned on when a framing error occurs, which happens when a 0, instead of a 1, is received in the stop-bit location. The microcontroller automatically clears the framing error on reception of the next character, which turns off the yellow LED.

The red LED is turned on when the input buffer, which can hold two characters at a time, is overrun by a third. In this case, no additional characters will be received until the continuous receive enable bit is toggled; in the case of the Micro Synth, a power cycle is required.

Status bytes contain the MIDI channel number and dictate the required data bytes. When a status byte is received, the channel number is stored. When a data byte is received, it is stored until the required number of data bytes has been received. Note-on and note-off status bytes require two data bytes.

In the case of a note-on status byte, the green LED is turned on. The MIDI note number in the data byte is converted into the required audio frequency, which is then converted into a suitable wavetable sampling increment and stored for access by the audio sampling rate interrupt.

The Micro Synth is monophonic, meaning that previously sounded notes are overwritten by new notes. As a result, it is only possible to play melodies and not harmonies. When the data byte contains a midi note-off message, and the note is the same as the note being played, the sampling increment is reset to 0, and the green LED is turned off.

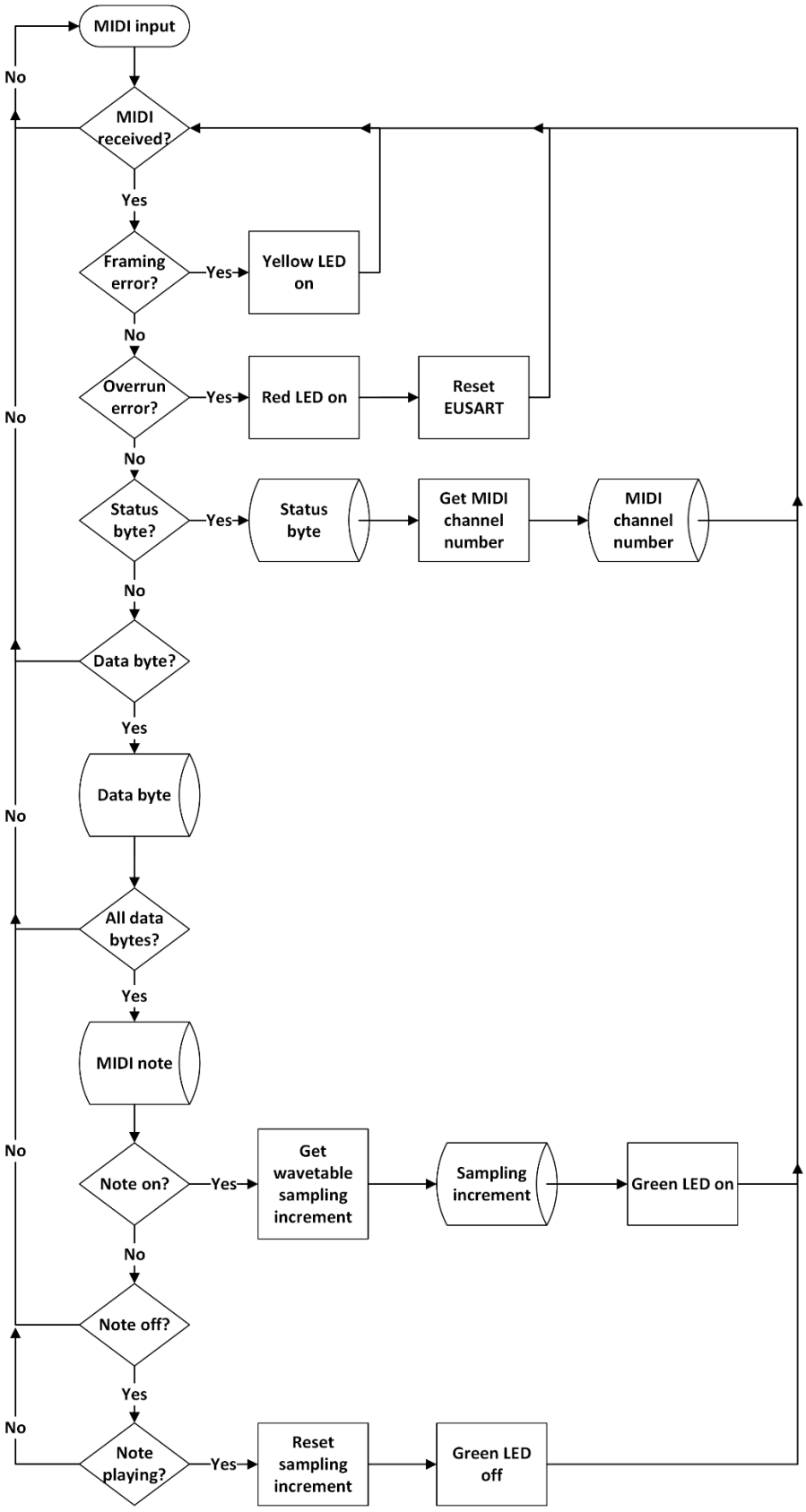


*Figure 12: MIDI input software flowchart*

## Wavetables

A wavetable is an array of values representing a single period of a periodic waveform. The wavetable array acts as a lookup table, enabling the microcontroller to retrieve samples in real time. Wavetables are efficient because the audio samples are only calculated once. In the case of the Micro Synth, the calculations were run in Excel but could also have been run on the microcontroller during initialisation. However, doing so would have prolonged the startup time.

The frequency of the synthesised tone depends on the rate at which values are read from the wavetable. Assuming a desired tone of A4 at 440 Hz with a sampling rate of 44,100 Hz. Since a wavetable represents a single period of a sine wave, it must be played back 440 times per second to produce the correct pitch. This means the wavetable should be read once every $\frac{44{,}100}{440} \approx 100.23$ samples. If the wavetable contains exactly 100 samples and is read at a constant rate of 44,100 samples per second, it will repeat 441 times per second, producing a tone of 441 Hz, which is slightly higher than the target frequency.

While the microcontroller's oscillator frequency and the audio sampling rate are fixed, the playback rate can still be adjusted by modifying the step size, which determines the following index to read from the array. A larger step size causes faster traversal of the wavetable, producing a higher frequency and pitch; conversely, a smaller step size results in a lower frequency. This step size is referred to as the sampling increment (SI). Equation 4 defines the SI for a wavetable of length $N$, desired frequency $f$, and sampling rate $f_s$.

*Equation 4: Wavetable sampling increment*

$$SI = \frac{N \times f}{f_s}$$

(Dodge & Jerse, 1997) refers to the index into the wavetable as the phase (ϕ). For each audio sample, the phase is updated by adding the sampling increment corresponding to the desired note frequency. Table 11 shows that the sampling increment is typically not a whole number. As the phase is an index into an array, it must be a positive whole number with a value no more than the length of the wavetable minus 1. When the phase overflows, it is wrapped around to the beginning of the wavetable by resetting the phase to zero. Truncation, rounding, and interpolation are solutions recommended by (Dodge & Jerse, 1997) when the phase is a decimal number. However, the PIC16F1718 microcontroller has slow floating-point arithmetic, making performing phase updates with fractional precision at the audio sampling rate impractical. Therefore, the sampling increment has been rounded to the nearest integer. As a result, the frequency of the notes produced by the Micro Synth has a varying margin of error, which is often noticeable when compared to the desired musical note.

Table 10 shows a wavetable length of 4,096 and a sampling rate of 29,400 Hz. The sampling increment of some adjacent lower notes is the same, producing the same tone. With no transposition, the lowest note on the MIDI Start keyboard is MIDI note 36, the musical note C with a frequency of 65.406 Hz and a sampling increment of 9.11, which can be rounded to 9. The following note on the keyboard is MIDI note 37, musical note C# with a frequency of 69.296 Hz and a sampling increment of 9.65, which can be rounded to 10. However, when going one note higher to MIDI note 38, musical note D with a frequency of 73.416 and a sampling increment of 10.23, rounding down will result in the same sampling increment as note C#. Two different and adjacent notes on the keyboard will sound the same.

*Table 10: Rounded sampling increments for C#, D and D#, E are identical, resulting in the same pitch.*

| Wavetable length (N) | Sampling rate (fs) | Note frequency (f) | Sampling increment (SI) | Rounded sampling increment | MIDI note nr | Musical note |
|---|---|---|---|---|---|---|
| 4,096 | 29,400 | 65.406 | 9.11 | 9 | 36 | C |
| 4,096 | 29,400 | 69.296 | 9.65 | **10** | 37 | C# |
| 4,096 | 29,400 | 73.416 | 10.23 | **10** | 38 | D |
| 4,096 | 29,400 | 77.782 | 10.84 | **11** | 39 | D# |
| 4,096 | 29,400 | 82.407 | 11.48 | **11** | 40 | E |
| 4,096 | 29,400 | 87.307 | 12.16 | 12 | 41 | F |

As a countermeasure, the Micro Synth starts from MIDI note 48 and ends at MIDI note 72. Table 11 contains the rounded sampling increments for the Micro Synth note range. Notice that the rounded sampling increments are all different and will therefore produce different pitches. However, the difference between the rounded sampling increments is smaller for lower notes, resulting in more deviation from the desired pitch.

*Table 11: Range of the Micro Synth*

| Wavetable length (N) | Sampling rate (fs) | Note frequency (f) | Sampling increment (SI) | Rounded sampling increment | MIDI note nr | Musical note |
|---|---|---|---|---|---|---|
| 4,096 | 29,400 | 130.813 | 18.22 | **18** | 48 | C |
| 4,096 | 29,400 | 138.591 | 19.31 | **19** | 49 | C# |
| 4,096 | 29,400 | 146.832 | 20.46 | **20** | 50 | D |
| 4,096 | 29,400 | 155.564 | 21.67 | **22** | 51 | D# |
| 4,096 | 29,400 | 164.814 | 22.96 | **23** | 52 | E |
| 4,096 | 29,400 | 174.614 | 24.33 | **24** | 53 | F |
| 4,096 | 29,400 | 184.997 | 25.77 | **26** | 54 | F# |
| 4,096 | 29,400 | 195.998 | 27.31 | **27** | 55 | G |
| 4,096 | 29,400 | 207.652 | 28.93 | **29** | 56 | G# |
| 4,096 | 29,400 | 220 | 30.65 | **31** | 57 | A |
| 4,096 | 29,400 | 233.082 | 32.47 | **32** | 58 | A# |
| 4,096 | 29,400 | 246.942 | 34.40 | **34** | 59 | B |
| 4,096 | 29,400 | 261.626 | 36.45 | **36** | 60 | middle C |
| 4,096 | 29,400 | 277.183 | 38.62 | **39** | 61 | C# |
| 4,096 | 29,400 | 293.665 | 40.91 | **41** | 62 | D |
| 4,096 | 29,400 | 311.127 | 43.35 | **43** | 63 | D# |
| 4,096 | 29,400 | 329.628 | 45.92 | **46** | 64 | E |
| 4,096 | 29,400 | 349.228 | 48.65 | **49** | 65 | F |
| 4,096 | 29,400 | 369.994 | 51.55 | **52** | 66 | F# |
| 4,096 | 29,400 | 391.995 | 54.61 | **55** | 67 | G |
| 4,096 | 29,400 | 415.305 | 57.86 | **58** | 68 | G# |
| 4,096 | 29,400 | 440 | 61.30 | **61** | 69 | A (concert pitch) |
| 4,096 | 29,400 | 466.164 | 64.95 | **65** | 70 | A# |
| 4,096 | 29,400 | 493.883 | 68.81 | **69** | 71 | B |
| 4,096 | 29,400 | 523.251 | 72.84 | **73** | 72 | C |

Table 12 shows that increasing the sampling rate to 44,100 Hz causes notes D and D# to share the same rounded sampling increment, resulting in identical output pitches.

*Table 12: Rounded sampling increments for D and D# are identical, resulting in the same pitch.*

| Wavetable length (N) | Sampling rate (fs) | Note Frequency (f) | Sampling increment (SI) | Rounded sampling increment | MIDI note nr | Musical note |
|---|---|---|---|---|---|---|
| 4,096 | 44,100 | 130.813 | 12.15 | 12 | 48 | C |
| 4,096 | 44,100 | 138.591 | 12.87 | 13 | 49 | C# |
| 4,096 | 44,100 | 146.832 | 13.64 | **14** | 50 | D |
| 4,096 | 44,100 | 155.564 | 14.45 | **14** | 51 | D# |
| 4,096 | 44,100 | 164.814 | 15.31 | 15 | 52 | E |
| 4,096 | 44,100 | 174.614 | 16.22 | 16 | 53 | F |

The length of the wavetable can be reduced to use less program memory. However, Table 13 shows that reducing the wavetable length to 2,048 results in notes C and C#, D and D#, and E and F sharing the same sampling increment. The combined results show that longer wavetables and lower sampling rates improve pitch accuracy by enlarging the sampling increments.

*Table 13: Rounded sampling increments for C, C# and D, D# and E, F are identical, resulting in the same pitch.*

| Wavetable length (N) | Sampling rate (fs) | Note Frequency (f) | Sampling increment (SI) | Rounded sampling increment | MIDI note nr | Musical note |
|---|---|---|---|---|---|---|
| 2,048 | 44,100 | 130.813 | 6.07 | **6** | 48 | C |
| 2,048 | 44,100 | 138.591 | 6.44 | **6** | 49 | C# |
| 2,048 | 44,100 | 146.832 | 6.82 | **7** | 50 | D |
| 2,048 | 44,100 | 155.564 | 7.22 | **7** | 51 | D# |
| 2,048 | 44,100 | 164.814 | 7.65 | **8** | 52 | E |
| 2,048 | 44,100 | 174.614 | 8.11 | **8** | 53 | F |

## Oscillators

(Welsh, 2006) states in his Synthesiser Cookbook that the most common waveforms on analogue synthesisers are the sawtooth, square, and triangle waves. Considering this, the Micro Synth supports sine, sawtooth, and square waveforms. Table 11 shows the highest note frequency supported by the Micro Synth is 523.251 Hz.

### Sine wave

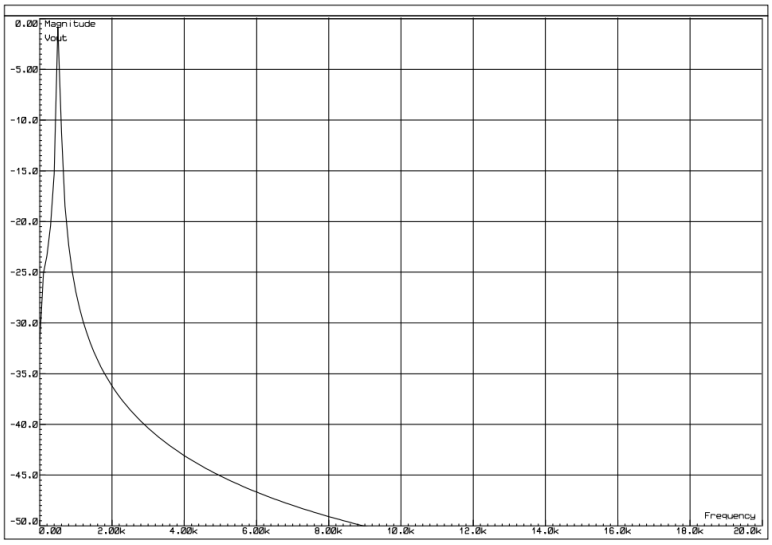


*Figure 13: Simulated frequency spectrum of a 523Hz sine wave*

Figure 13 shows a Proteus-simulated 523.251 Hz sine wave frequency spectrum. The main lobe is centred on the fundamental frequency of 523.251 Hz. Even though side lobes are present due to spectral leakage, no harmonic multiples of the fundamental frequency are present. As described by (Smith, 2003), the sampling frequency must be twice the highest frequency. Therefore, the minimum sampling rate must be set above 1.05 kHz.

*Equation 5: Sine wavetable samples*

$$x(n) = \sin\left(\frac{2\pi \times k \times n}{N}\right) \quad for\ n \in \{0, 1, \dots, 4095\}$$

Equation 5 generates a single period of a sine. In this context, $k$ represents the number of periods, $n$ is the sample number, and $N$ is the length of the wavetable, which is 4,096 for the Micro Synth. For a single period, $k$ is set to 1, and the sample number $n$ increments from 0 until $N - 1$.

*Equation 6: Scaled sine wavetable of unsigned chars*

$$y(n) = [\ (x(n) + 1)\ \times 127.5\ ] \quad for\ n \in \{0, 1, \dots, 4095\}$$

The datasheet indicates that the PIC16F1718 features an 8-bit Digital-to-Analogue Converter (DAC) that accepts unsigned 8-bit values ranging from 0 to 255. Equation 6 demonstrates how to map floating-point values commonly associated with the sine function (which range from -1 to 1) to integer values that fit within the DAC input range. To achieve this, 1 is added to the previously calculated sine wavetable sample value at the same index, which is then multiplied by 127.5 before rounding the result. In the case of the PIC16F1718, the result is assigned to a constant wavetable array to be stored in code memory. Figure 14 shows a plot of a scaled sine wave period in the range 0 – 255 with 4,096 values, which is also the wavetable length used by the Micro Synth.

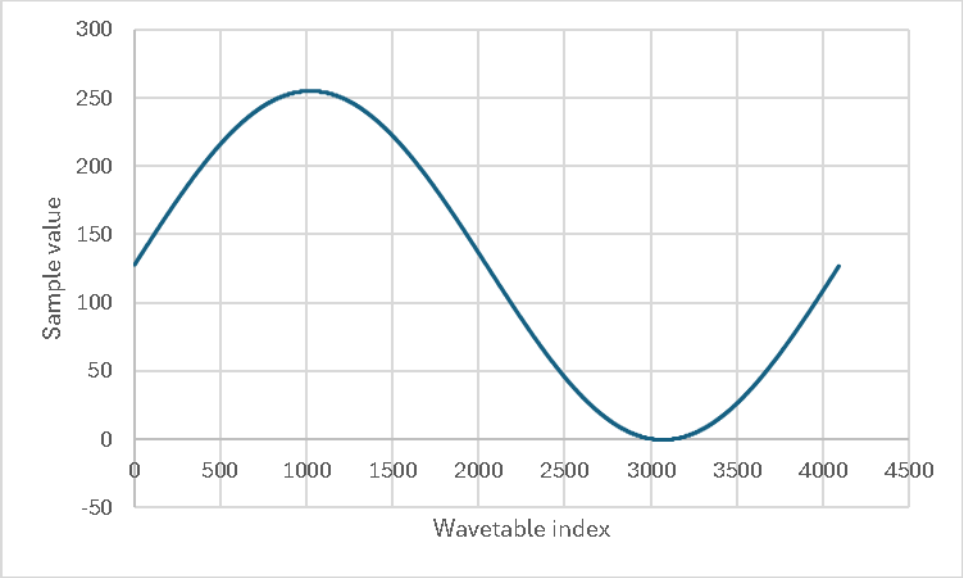


*Figure 14: Scaled sine period of unsigned chars*

## Sawtooth wave

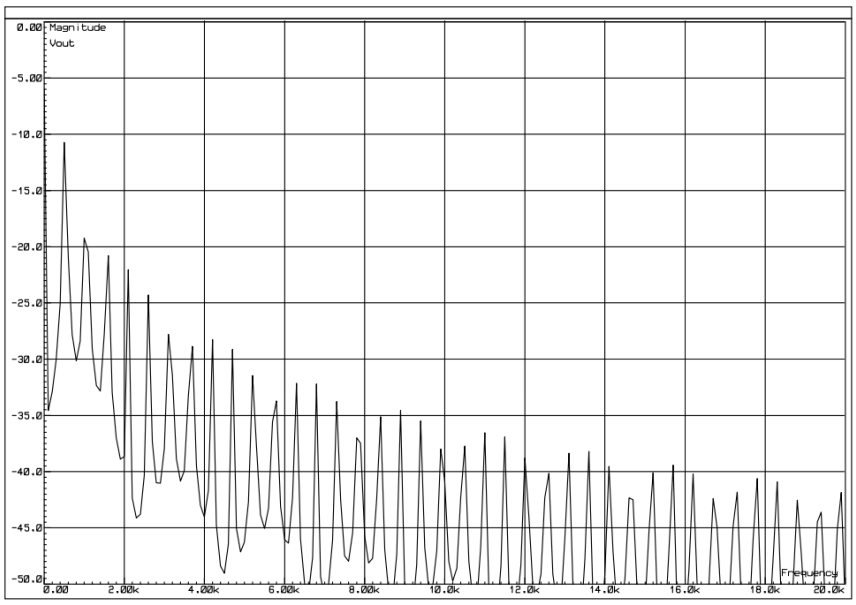


*Figure 15: Simulated frequency spectrum of a 523Hz sawtooth wave*

Figure 15 shows a Proteus-simulated sawtooth wave frequency spectrum with the fundamental frequency at 523.251 Hz. Harmonic partials with progressively decreasing amplitudes are present at odd and even multiples of the fundamental frequency: 523.251 Hz, 1,046.502 Hz, 1,569.753 Hz, 2,093.004 Hz, 2,616.255 Hz, 3,139.506 Hz, 3,662.757 Hz, 4,186.008 Hz and so on. The Micro Synth's sampling rate is 29.4 kHz, which means any frequencies above 14.7 kHz will be aliased. Therefore, the last unaliased harmonic partial will be at 14,651.028 Hz, which is the reason for selecting the low-pass filter's cut-off frequency of 14.7 kHz.

*Equation 7: Scaled sawtooth sample values*

$$x(n) = \left\lfloor \frac{n}{\frac{N}{2^b}} \right\rfloor \quad for\ n \in \{0, 1, \dots, 4095\}$$

Equation 7 produces a single period of a sawtooth wave, with the resulting values assigned to a constant wavetable array stored in code memory. In this context, $n$ denotes the sample number, $b$ refers to the number of bits supported by the DAC (which is 8 bits for the Micro Synth), and $N$ represents the length of the wavetable (4,096 samples for the Micro Synth). The result is rounded down to the nearest integer. Figure 16 plots the resulting sawtooth wave period in the range 0 – 255 with 4,096 values.

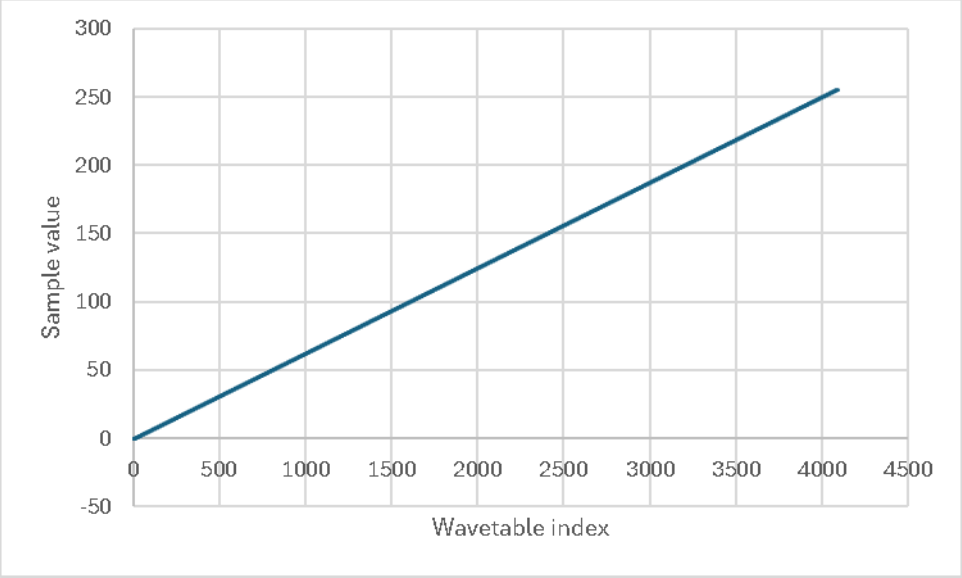


*Figure 16: Sawtooth period of unsigned chars*

## Square wave

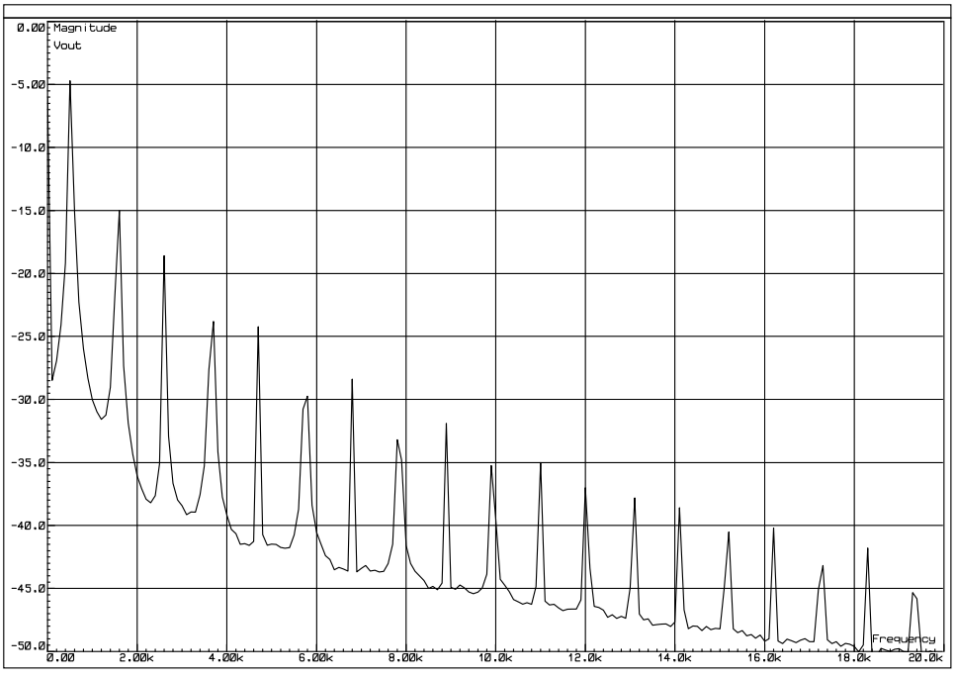


*Figure 17: Simulated frequency spectrum of a 523Hz square wave*

Figure 17 shows a Proteus-simulated square wave frequency spectrum with the fundamental frequency at 523.251 Hz. Harmonic partials with progressively decreasing amplitudes are present only at odd multiples of the fundamental frequency: 1,569.753 Hz, 2,616.255 Hz, 3,662.757 Hz, 4,709.259 Hz, 5,755.761 Hz, 6,802.263 Hz, 7,848.765 Hz and so on. The Micro Synth's sampling rate is 29.4 kHz, which means any frequencies above 14.7 kHz will be aliased. Therefore, the last unaliased harmonic partial will be at 14,127.777 Hz, which is the reason for selecting the low-pass filter's cut-off frequency of 14.7 kHz.

*Equation 8: Scaled square wave sample values*

$$x(n) = \begin{cases} 255, & 0 \leq n < 2048 \\ 0, & 2048 \leq n < 4096 \end{cases} \quad for\ n \in \{0,1,\dots,4095\}$$

Equation 8 produces a single period of a scaled square wave. In this context, $n$ denotes the sample number from 0 to 4,095. The result is a value of 255 for indexes from 0 to 2,047 and a value of 0 for indexes from 2,048 to 4,095. The square wave does not require a wavetable, as the calculation is not computationally expensive. The result for each wavetable index can be returned as the result of a function instead, thereby consuming data memory instead of code memory. Figure 18 plots the resulting square wave period in the range 0 – 255 with 4,096 values.

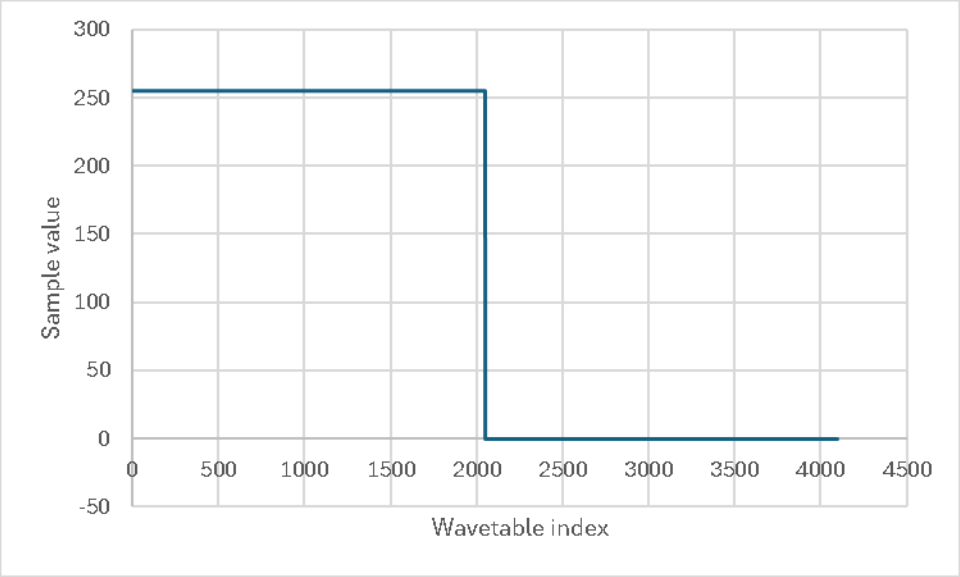


*Figure 18: Square wave period of unsigned chars*

## Sampling Rate

Equation 9 defines the method for calculating the required number of machine cycles to approximate a sampling rate ($f_s$) of 29.4 kHz. The system clock ($F_{OSC}$) of the PIC16F1718 is configured to run at 16 MHz. According to the datasheet, the instruction clock ($F_{cyc}$) runs at one-quarter of the system clock, resulting in 4 MHz. Consequently, the time per machine cycle ($T_{cyc}$) is 0.25 $\mu s$. The inverse of the sampling rate yields the time elapsed between audio samples ($T_s$), calculated as 34.01361 $\mu s$. To calculate the number of ticks Timer0 must wait before triggering an interrupt, it is necessary to determine the number of machine cycles per audio sampling period, which is 136.05 cycles. As Timer0 is an 8-bit timer overflowing on the 256th tick from 255 to 0, the TMR0 register is set to $256 - 136 = 120$.

*Equation 9: Cycles required for 29.4 kHz sampling rate*

$$F_{OSC} = 16MHz$$
$$F_{cyc} = \frac{F_{OSC}}{4} = 4\ MHz$$
$$Tcyc = \frac{1}{F_{cyc}} = 0.25\mu s$$
$$f_s = 29.4\ kHz$$
$$T_s = \frac{1}{f_s} = 34.01361\ \mu s$$
$$cycles = \frac{T_s}{T_{cyc}} = \frac{34.01361}{0.25} = 136.05$$

Figure 19 illustrates the software flowchart executed for each Timer0 interrupt. A sampling increment is evaluated during each interrupt to determine whether a key is pressed, indicating that a note should be produced. The MIDI channel number variable selects the desired waveform, while the phase variable tracks the current position within the waveform's wavetable. This phase value, combined with the MIDI channel number, is used to retrieve the appropriate audio sample, which is then sent to the input of the DAC. Next, the sampling increment increments the phase, so it is ready for the following sample. Should the phase become larger than the wavetable length, then the phase is wrapped around and reset to 0. This process is repeated each time Timer0 triggers a new interrupt, ensuring continuous audio sample generation.

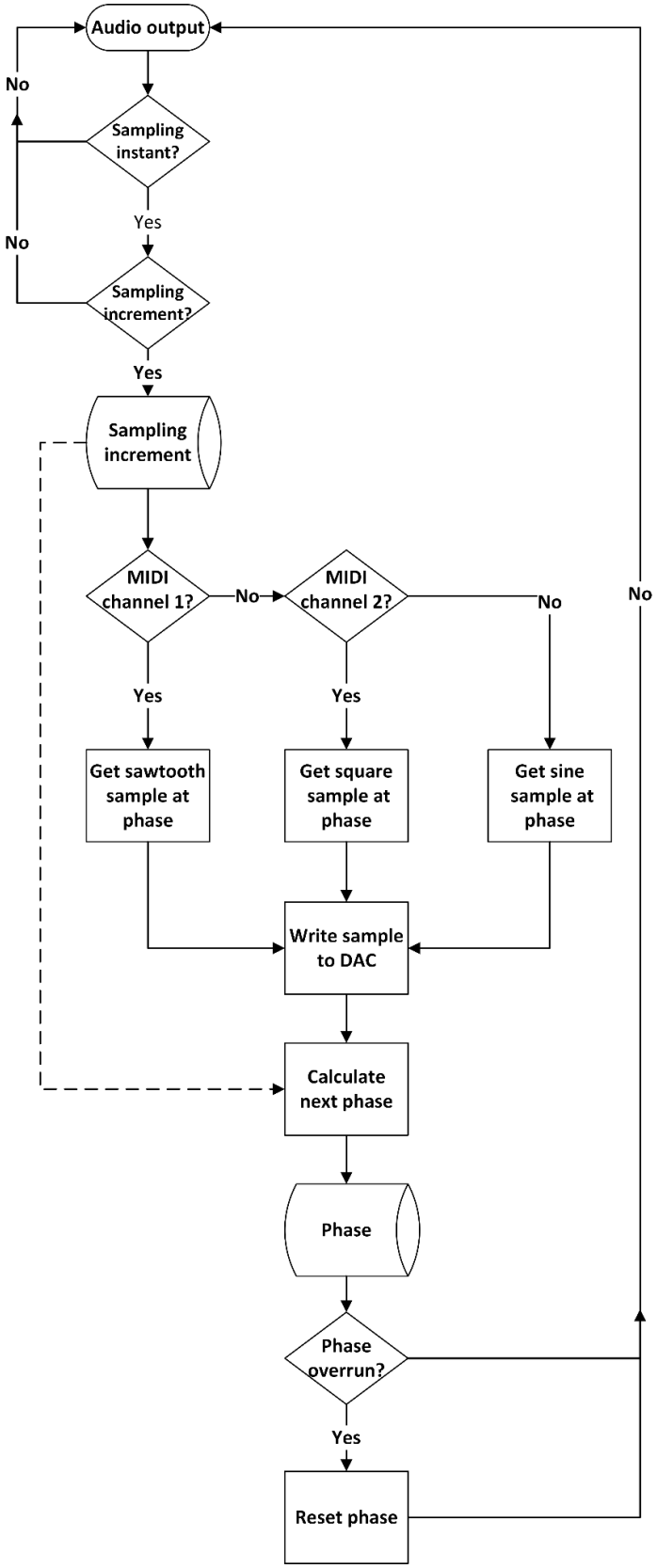


*Figure 19: Audio output sampling rate software flowchart*

# Conclusion

The Micro Synth project integrates MIDI communication, digital audio synthesis, and embedded hardware design into a compact and functional prototype. Using a PIC16F1718 microcontroller, the system translates MIDI note events into real-time audio signals via wavetable synthesis, supporting sine, sawtooth, and square waveforms.

A sampling rate of 29.4 kHz was achieved using Timer0 interrupts, with waveform playback driven by integer-based sampling increments. Although this fixed-point method introduces pitch errors, especially at lower frequencies, it balances accuracy and computational efficiency well.

From an ethical standpoint, the project prioritises sustainability by avoiding printed circuit boards and instead assembling the system on a breadboard to allow component reuse. It also highlights the risk of hearing loss from prolonged high-volume exposure, especially in electronic music contexts, and encourages responsible use.

Overall, the Micro Synth demonstrates that low-cost microcontrollers can effectively generate musical tones in real time and provides a strong foundation for future extensions, such as polyphony, envelope shaping, and real-time controls.

# References


Ankit, et al., 2021. Electronic waste and their leachates impact on human health and environment: Global ecological threat and management. *Environmental Technology & Innovation,* p. 102049.

Baldé, C. et al., 2024. *Global E-waste Monitor,* Geneva/Bonn: International Telecommunication Union (ITU) and United Nations Institute for Training and Research (UNITAR).

Crowe, M. et al., 2003. Waste from electrical and electronic equipment (WEEE). *Dimitrios Tsotsos. quantifies, dangerous substances and treatment methods.*

Dodge, C. & Jerse, T., 1997. *Computer Music: Synthesis, composition and performance,* s.l.: Schirmer.

Ergun, O., Cakmak, E. & Alniacik, A., 2024. Recreational music exposure and hearing health in young adults. *European Archives of Oto-Rhino-Laryngology,* p. 4373–4378.

Goodship, V., Stevels, A. & Huisman, J., 2019. *Waste electrical and electronic equipment (WEEE) handbook.* s.l.:Woodhead Publishing.

Kähäri, K. et al., 2003. Associations Between Hearing and Psychosocial Working Conditions in Rock/Jazz Musicians. *Medical Problems of Performing Artists,* pp. 98-105.

Kysela, J. & Iwai, T., 2020. *Advanced Linux Sound Architecture (ALSA) project homepage,* s.l.: s.n.

Microchip Technology, 2020. *PIC16(L)F1717/8/9.* Chandler, Phoenix, USA: Microchip Technology.

MIDI Manufacturers Association, 1996. *MIDI 1.0 Detailed Specification*. Los Angeles, California, USA: The MIDI Manufacturers Association.

Reynolds, A. & Bielefeld, E. C., 2023. Music as a unique source of noise-induced hearing loss. *Hearing Research,* Volume 430, p. 108706.

Schmuziger, N., Patscheke, J. & Probst, R., 2006. Hearing in Nonprofessional Pop/Rock Musicians. *Ear and Hearing,* pp. 321-330.

Smith, S. W., 2003. *Digital Signal Processing: A Practical Guide for Engineers and Scientists.* Massachusetts, United States of America: Newnes.

Texas Instruments, 2014. *TL08xx JFET-Input Operational Amplifiers.* Dallas, Texas: Texas Instruments.

Texas Instruments, 2023. *LM386 Low Voltage Audio Power Amplifier.* Dalls, Texas: Texas Instruments.

Wang, Q. et al., 2023. Environmental data and facts in the semiconductor manufacturing industry: An unexpected high water and energy consumption situation. *Water Cycle,* pp. 47-54.

Welsh, F., 2006. *Welsh's Synthesizer Cookbook.* USA: Fred Welsh.

Wijnia, Y. et al., 2009. The Pugh Controlled Convergence method: model-based evaluation and implications for design theory. *Research in Engineering Design,* pp. 41-58.

Worlanyo, A. S. & Jiangfeng, L., 2021. Evaluating the environmental and economic impact of mining for post-mined land restoration and land-use. *Journal of Environmental Management,* Volume 279, p. 111623.

# Appendix

*Code snippet 1: Connecting vkeydb and aseqdump to view formatted MIDI messages*

```
# start vkeybd
vkeybd

# list MIDI input devices
aconnect -i
client 129: 'Virtual Keyboard' [type=user,pid=256540]
    0 'Virtual Keyboard'

# start aseqdump
aseqdump

# list MIDI output devices
aconnect -o
client 128: 'aseqdump' [type=user,pid=256323]
    0 'aseqdump'

# connect vkeybd to aseqdump
aconnect 129.0 128.0

# list MIDI connections
aconnect -l
client 128: 'Virtual Keyboard' [type=user,pid=257175]
    0 'Virtual Keyboard'
        Connecting To: 129:0
client 129: 'aseqdump' [type=user,pid=257185]
    0 'aseqdump'
        Connected From: 128:0
```

*Code snippet 2: Connecting vkeybd and amidi to view raw MIDI output as hex*

```
# start vkeybd
Vkeydb

# list MIDI input devices
aconnect -i
client 129: 'Virtual Keyboard' [type=user,pid=256540]
    0 'Virtual Keyboard'

# amidi: View Raw MIDI devices
amidi -L

RawMIDI list:
...
virtual {
        @args.0 MERGE
        @args.MERGE {
                type string
                default 1
        }
        type virtual
        merge $MERGE
}
```

```
# amidi: Listen on virtual port and print MIDI data as hex
amidi -p virtual -d

# aconnect list MIDI output devices
aconnect -o
client 128: 'Client-128' [type=user,pid=262183]
    0 'Virtual RawMIDI '

# connect vkeybd to amidi
aconnect 129.0 128.0

# list MIDI connections
aconnect -l
client 128: 'Client-128' [type=user,pid=262183]
    0 'Virtual RawMIDI '
        Connected From: 129:0
client 129: 'Virtual Keyboard' [type=user,pid=262390]
    0 'Virtual Keyboard'
        Connecting To: 128:0
```

# The Final Countdown

Arranged by Charles Mulder

Composed by Europe