MIDI Music Box

By

Jeremy Lee

Sean Liang

Tyler Shu

Final report for ECE 445, Senior Design, Fall 2023

TA: Gregory Jun

December 2023

Team 22

Abstract

This final report comprehensively describes the design process and implementation of a MIDI music box which reads music instruction through a digital data format called MIDI and processes the MIDI data, sending analog signals to a speaker to play sound. The planned capabilities of the project include playing 4 different waveforms, eight notes of polyphony, playing sounds up to 15KHz, and volume up to 20 watts.

After many design stages, the final project was powered by a DC power jack and took in MIDI data through a USB port, outputting sounds that correspond to the MIDI data. The project was ultimately able to play the four different waveforms, eight notes of polyphony, sounds up to 15khz, but could only provide at most 10 watts of power to our speaker. Additionally, the project adjusts the volume of each note played based on the intensity of the note's keypress on the controller.

Table of Contents

1. Introduction

Music is a very popular activity, but oftentimes there is an entry barrier as musical equipment tends to be expensive and inaccessible. MIDI controllers are a popular tool for music, and low-end controllers do not have any sort of audio playback. As such, it is very difficult for a beginner to obtain a usable instrument. Our goal was to make a simple music box that can be connected to a MIDI controller and output sound without the use of any external music production software.

1.1 Performance Requirements

With emphasis on project functionality at the core of any development, the team set several high-level requirements at the beginning of the process in order to provide benchmarks and expectations for project functionality:

1. The project must be able to synthesize at least **four different tones** (waveforms), including (but not restricted to): **Triangle, Square, Sine, and Sawtooth**.
2. The project must be able to **produce at least eight note polyphony** - that is, the project must have the ability to combine at least eight different musical parts together to form a harmony.
3. The project must be able to **produce pitch in the frequency range of C2 (65.4 Hz) - C5 (523.5 Hz)**, with additional capability of **reaching up to 15 KHz** including harmonics for synthesized tones.
4. The project must be able to **drive a speaker up to 20W**.

These requirements provided realistic expectations for the project's functionality, with emphasis on sound variation through the different tones, polyphony, and full range of frequencies enabling the user to produce music in full, in addition to the power requirement providing the user with adequate playback volume. These high-level requirements acted as guidelines for development, and drove the development process throughout the project's lifetime.

2. Design

2.1 High-Level Operations and Block Diagram

To create active playback for MIDI Devices, the team developed a device which mainly utilizes a RPi3 (Raspberry Pi 3 Model B) and a DAC (Digital to Analog Converter) to convert the digital MIDI data into analog signals able to be amplified and utilized by traditional speakers for playback. The project's operations starts with the RPi3, which takes in MIDI Data Input from the external MIDI Controller connected via the RPi3's USB Port and transforms the digital data into a digital signal to be converted into an analog signal by the DAC. The RPi3 determines which waveform (like a Sine Wave or Square Wave) to transform the digital data into by reading two switch circuits connected to the RPi3's GPIO Pins. Additionally, the RPi3 maintains a constant rate of output by reading input from an external clock circuit provided by an LM555-Timer Clock Circuit, executing and producing output only on the rising edge of the clock. The RPi3 transmits the digital signal to the DAC using the SPI (Serial Peripheral Interface) protocol with every clock cycle, based on the aforementioned rising edge.

The DAC then sends the analog signal to a potentiometer, which will determine the overall gain of the produced wave, after which the signal is amplified by the Volume Amplifier and subsequently output by the Speaker. The entire system is powered by a 12V DC-DC Power Jack connected to an external supply, which is converted to 5V and 3.3V for proper use. The final high-level block diagram is shown in Figure 2.1 below, and provides an overview of the project's operations.

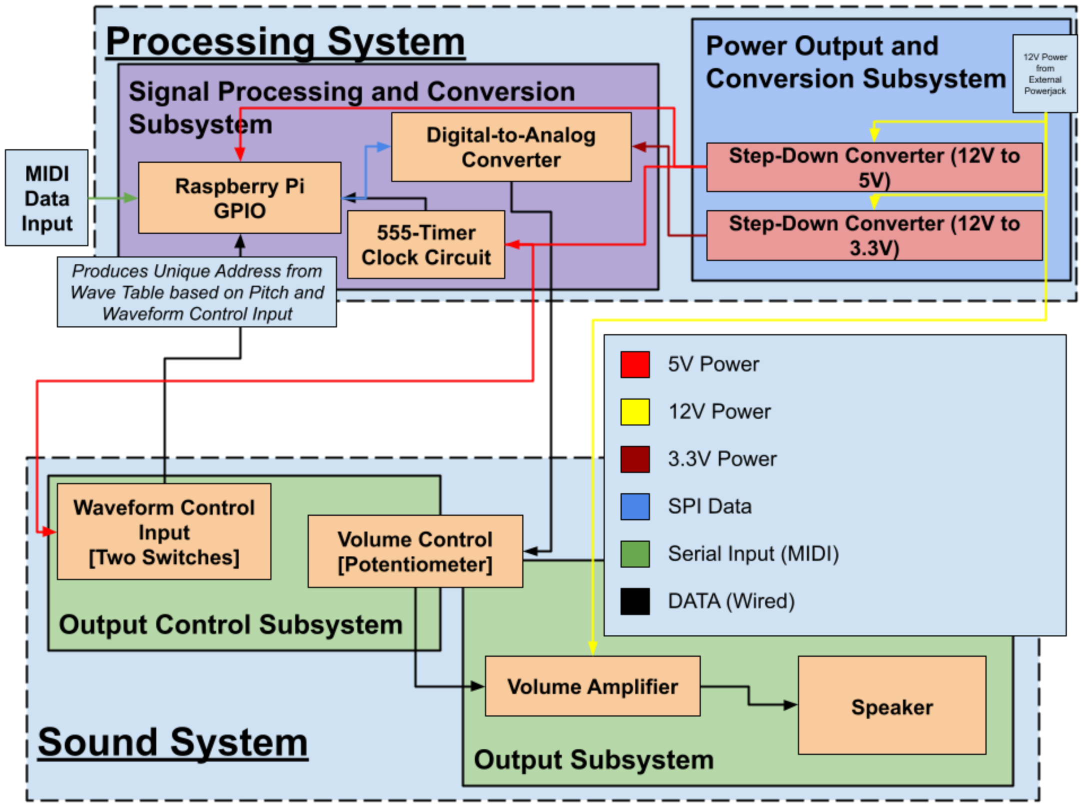

Processing System
Signal Processing and Conversion Subsystem
Power Output and Conversion Subsystem
12V Power from External Powerjack
Digital-to-Analog Converter
Step-Down Converter (12V to 5V)
MIDI Data Input
Raspberry Pi GPIO
555-Timer Clock Circuit
Step-Down Converter (12V to 3.3V)
Produces Unique Address from Wave Table based on Pitch and Waveform Control Input
5V Power
12V Power
3.3V Power
SPI Data
Serial Input (MIDI)
DATA (Wired)
Waveform Control Input [Two Switches]
Volume Control [Potentiometer]
Output Control Subsystem
Volume Amplifier
Speaker
Sound System
Output Subsystem

Figure 2.1: Final High-Level Block Diagram

The block diagram is split into two main systems, each containing two subsystems. The Processing System contains the Signal Processing and Conversion Subsystem (SPCS) which handles the signal processing required to generate waveforms representative of our audio feedback, as well as the Power Output and Conversion Subsystem (POCS), which is responsible for powering the entirety of the project. The Sound Systems contains the Output Control Subsystem (OCS), which provides an interface for users to control the project's output, whether that be through determining which waveform to output or the waveform's gain. The system also includes the Output Subsystem (OS), which is responsible for amplifying the waveform to an audible level and outputting the waveform via a speaker. The R&V (Requirements and Verification) tables for each subsystem are shown in Appendix A at the end of this document.

2.2 Block-Level Changes

Several block-level changes were made during the project's development, mainly involving the Processing System. The original block diagram for the SPCS included the usage of a filter applied to the digital signal based on MIDI Velocity, which was subsequently integrated into the RPi3's software operations. Additionally, the Timer Clock Circuit was added to the SPCS, and was not originally part of the design - this component was integrated once initial testing raised synchronization issues and inconsistent sampling rates for signals that were output by the RPi3. Without the Timer Clock Circuit, the RPi3 would output digital signals to the DAC based on the speed of the RPi3's code's operations, which would vary significantly based on the level of polyphony being produced due to larger amounts of mathematical operations being performed. By including the clock, the RPi3 would output at a constant rate, determined by the speed of the clock.

The entirety of the POCS was also redone - the initial design utilized a Lithium-Ion battery and a voltage regulator as opposed to a DC-DC Power Jack and Step-Down Converters. The design was changed due to the potential inconsistencies of running a battery, with charge depleting over time, thus warranting the usage of the DC-DC Power Jack and a Wall Adapter to provide more consistent power input.

Additionally, the OS originally included the possibility of a 3.5mm jack instead of an external speaker, with the removal of such to simplify project design and usage.

2.3 Power and Output Conversion Subsystem

Upon analysis of the project's power requirements, it was clear that the project would require 3 different voltage lines: a 3.3V line to be utilized as the DAC's Voltage Reference, a 5V line to be utilized as the primary power supply for almost all components, and a 12V line to be utilized as the power supply for the Volume Amplifier. Thus, the team opted to utilize a DC-DC Jack which connected to a wall adapter which would output 12V, and utilize a 12V to 5V Step-Down Converter as well as a 12V to 3.3V Step-Down Converter to provide power to the corresponding voltage lines. The wall-powered approach was chosen over a battery approach due to the more consistent nature of the wall adapter's voltage over a battery's lifetime depletion, as well as the nonessential nature of portability in this application

2.4 Signal Processing and Conversion Subsystem

The SPCS contains components essential to the Signal Processing Applications of the project. In the following sections, we will discuss the various hardware and software components that drive the operations of the SPCS.

2.4.1 Raspberry Pi

We decided to use a Raspberry Pi 3 to take in MIDI data, perform calculations and output digital data to our DAC. The RPi3 was chosen due to the data capacity limitations of most commercially available microcontrollers, lacking the storage required to store the 4 waveforms mentioned previously. Additionally, with a focus on speed of operations, the RPi3 became a suitable candidate for the project, with a more than sufficient processing speed and memory capacity. If the project were given more time to develop, the team would have explored the usage of other microcontrollers such as the Raspberry Pi Pico to optimize the size and cost of the device.

2.4.2 MIDI Data

In order to create sound, we need to read MIDI data, sent serially from a MIDI controller. MIDI data consists of individual messages, each with a status byte and up to two data bytes [1]. As seen in Figure 4.6, the MIDI messages for “Note On” and “Note Off” contain three bytes: a status byte which contains message type and the MIDI Channel (which is unused), a data byte containing the Note Number (or pitch), and another data byte containing the note velocity (intensity of keypress) [2]. Each keypress only sends a single MIDI message upon the note’s onset and termination.

2.4.3 Wavetable Oscillator

To generate sound waveforms, the project employed a technique known as Wavetable Synthesis. 1024 samples of one full period of an arbitrary wave are stored into memory, sampled based on the “phase” of the target waveform. The wavetable acts as a circular buffer, with values being sampled from the table based on the pitch of the note, which determines how fast the phase changes every sample, also known as the phase increment [3]. The phase increment p for our program can be calculated with the following equation:

$$p = \frac{1024f}{f_s} \quad (1)$$

Where f is the desired frequency of the output, and f_s is the sampling/playback rate. The frequency of a note in Hz given the MIDI note number n can be calculated as [4]:

$$f = 440 \bullet 2^{\frac{n-69}{12}} \quad (2)$$

These two equations allow us to use the pitch value from our input MIDI data and convert it into a digital signal. Since the phase increases every sample and the wavetable is circular, we are able to modulo the phase by 1023, to consistently select values within the wavetable.

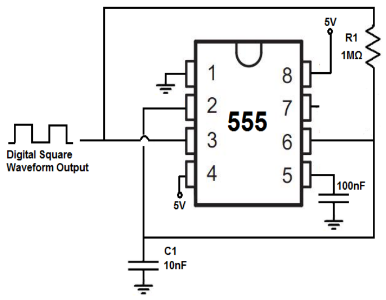
2.4.4 Software Implementation

The software component was originally implemented in Python, and ported over to C++ later in the development process. For our Python Implementation, we used the PyRtMidi library to interface with our MIDI controller and read MIDI messages. We utilized three dictionaries for the current phase increment, phase, and output. Each note is used for the key in all three dictionaries. Once a “Note On" message is received, the phase increment is calculated using equations (1) and (2). Then, every sample the calculated phase increment is added to the current phase, and the phase acts as a read pointer to the wavetable. Since the phase increment is usually not a whole number, the current phase is rounded to the nearest integer, and the value of the wavetable at that phase is written into the output dictionary. Finally, all the values from the output dictionary are summed, resulting in the final output value for the sample. When we receive a “Note off” message, all values paired with the note are deleted. This ensures that we only sum the values of currently active notes for the final output. For sending the output to the DAC, we used the spidev library.

We later switched to C++ as the Python Implementation was too slow for what we wanted to achieve. While using the Python Implementation, the software struggled to achieve above a sampling rate of ~15KHz, with Polyphony causing the sample rate to drop as mathematical operations increased. Switching the implementation to C++ yielded impressive results - with a maximum sampling rate of 39KHz reached, the software was significantly faster. The C++ implementation employed a similar method as the Python Implementation, but opted to frontload the Frequency and Phase Increment calculations in the interest of program efficiency and speed. The implementation also included several other features: software to read GPIO pins connected to external switches to determine output waveform, implementation to synchronize the program with an external clock to keep the sampling rate constant, and capacity to include note velocity into sound intensity. The implementation also used the same RtMidi library as the Python Implementation, utilizing its base C++ version, as well as using the WiringPi library to implement GPIO Operations and the SPI Protocol, as opposed to Python's spidev library.

2.4.5 555-Timer

With the need for the RPi3 to send data at a fixed, consistent sampling rate, the team implemented a clock circuit that would output a square wave at a predetermined frequency. The RPi3 implementation would read the value from a GPIO pin and wait for a rising edge, at which the program would execute the majority of the implementation and output a signal to the DAC. Figure 2.2 shows the schematic of the clock circuit, as well as its predetermined output frequencies based on component values.



To create a 6Hz signal, R1= 10MΩ and C= 10nF.

To create a 600Hz signal, R1= 100KΩ and C= 10nF.

To create a 134Hz signal, R1= 470KΩ and C= 10nF.

To create a 1.7KHz signal, R1= 33KΩ and C= 10nF.

To create a 43KHz signal, R1= 1KΩ and C= 10nF.

To create a 180KHz signal, R1= 150Ω and C= 10nF.

To create a 252KHz signal, R1= 100Ω and C= 10nF.

Figure 2.2: LM555-Timer Clock Circuit and Predetermined Output Frequencies

2.4.6 DAC

With no onboard DAC available on the RPi3, the team opted to use an external DAC to achieve any conversion requirements. Though DACs are available on some microcontrollers, most commercially available microcontrollers do not include DACs, and those that do are typically expensive - thus, the usage of an external DAC circumvents these issues, allowing the project to utilize the flexibility and speed of the RPi3. The team chose a DAC with a high enough bit resolution - without the proper resolution, the project would face significant quantization issues, causing the audio output to be inaccurate or unclean. The team decided to utilize a 10-bit resolution for a sufficient but simple implementation, utilizing the common SPI data interface for easier project understanding and portability, as long as innate compatibility with the RPi3's features.

2.5 Output Control Subsystem

The OCS contains components essential to providing the user with operational freedom in regards to the project's outputs. In the following sections, we will discuss the various hardware components that drive the operations of the OCS.

2.5.1 Potentiometer

To achieve the full variable gain effect, the team utilized the P160KNP-0QC15B10K Plastic Linear 0-10kΩ Potentiometer that was adjustable by hand and connected to the Volume Amplifier. The full range of resistance provided by the Potentiometer dampened the signal given by the DAC prior to amplification, creating a gain effect as desired. This feature was integrated with the goal of giving the user the ability to adjust the project's volume as needed.

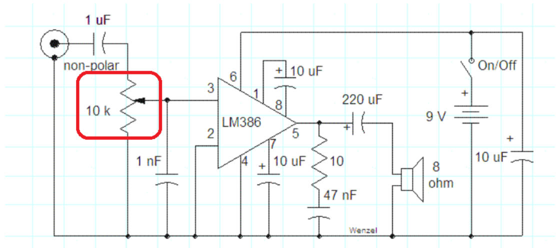


Figure 2.3: Volume Amplifier Circuit with Potentiometer

2.5.2 Switches

To achieve waveform variability, the team utilized two 2-pin SPDT switches to implement a small state machine, as depicted by Figure 2.4. With a 2-pin Switch Implementation, a resistor was required, leading to a dissipation of heat when the switch was closed and potentially a system failure - therefore, the team opted to utilize a 3-pin SPDT witch instead, allowing each switch to toggle between two different values - 1 or 0 - to utilize as inputs to our State Machine.

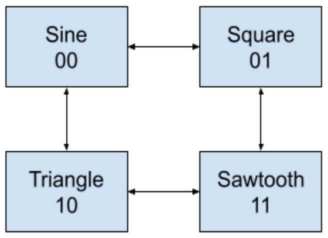


Figure 2.4: State Machine for Waveform Variability

2.6 Output Subsystem

As the final terminal of the project, the OS utilizes an Audio Amplifier Circuit, seen in Figure 2.3, and a Speaker to achieve the project's final output. The circuit shown in Figure 2.3 is able to achieve gain values between 20 and 200, utilizing the 12V line to sufficiently power the Volume Amplifier. As seen in Figure 2.3, the subsystem utilizes the LM386 in combination with the Speaker and a Potentiometer to achieve sound with variable gain.

3. Verification

In the following sections, the testing and verification of each subsystem (and whether they achieve the high-level requirements) will be discussed. The R&V table for each subsystem is included in Appendix B for reference.

3.1 Power and Output Conversion Subsystem

To verify that the results of the POCS matched the requirements set within Table 1 in Appendix A, the team measured the output voltage of the step-down converters using a multimeter. The requirement - which stated that each voltage needed to stay within 0.15V of the target voltage, was initially met and maintained within initial circuit implementation, but subsequent testing in later weeks showed varying results outside of the required range, possibly due to environmental and temperature factors of part degradation over time. Figure 3.1 displays the voltage measured through each step-down converter at the time of the failed testing.

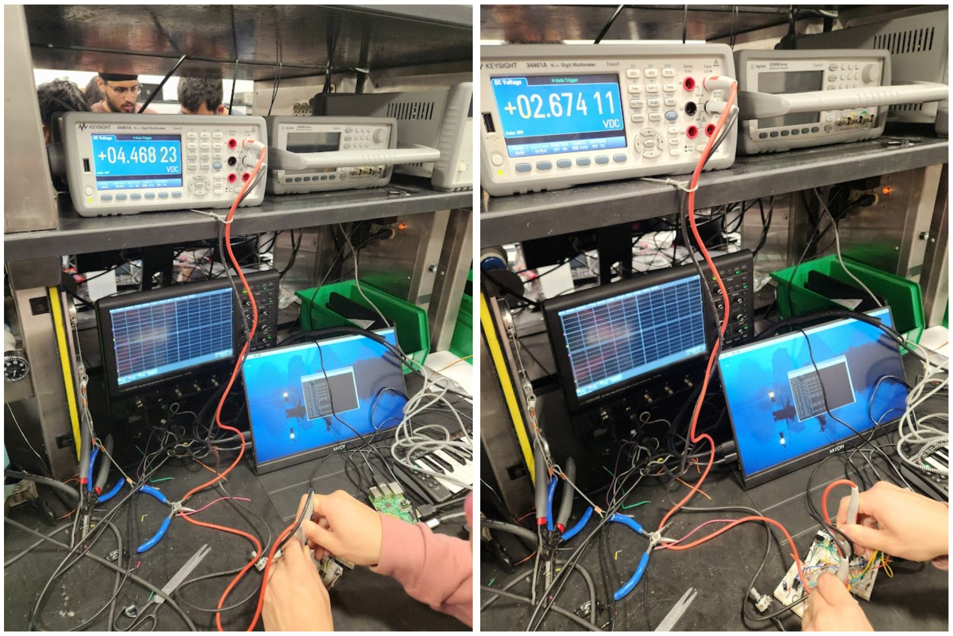


Figure 3.1: Voltage Outputs of the 5V Step-Down Converter (Left) and 3.3V Step-Down Converter (Right)

3.2 Signal Processing and Conversion Subsystem

To verify the program's output signals matched the requirements with Table 2 in Appendix A, an oscilloscope was utilized to analyze the waveforms produced by the DAC. During testing, the team found that the maximum sample rate of the DAC was approximately 40KHz, as shown in Figure 3.2, enabling the program to output frequencies up to around 20KHz according to the Nyquist-Shannon Sampling Theorem, which states that a "signal sampled at a rate F can be fully reconstructed if it contains only frequency components below half that sampling frequency: F/2" [7]. In other words, the maximum frequency signal f that can be reconstructed with our sampling frequency f_s is:

$$f = \frac{f_s}{2} \qquad (3)$$

Additionally, upon playing the lowest note on our MIDI Keyboard, a pitch with a frequency of approximately 4Hz was measured. The results of this testing can be seen in Figures 3.2 and 3.3. These results meet two high-level requirements, indicating that the project is able to output signals of up to 15KHz, including signals between the predetermined C2-C5 range. In the final design, the note frequency range was expanded beyond the C2-C5 range to include the full MIDI output, which ranged from A0-G#9.

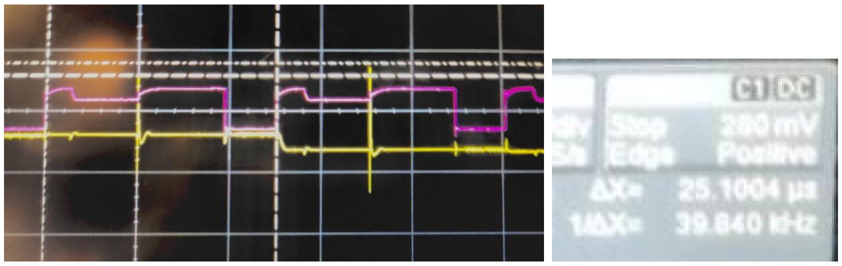


Figure 3.2: Playback Rate of Generated Signal

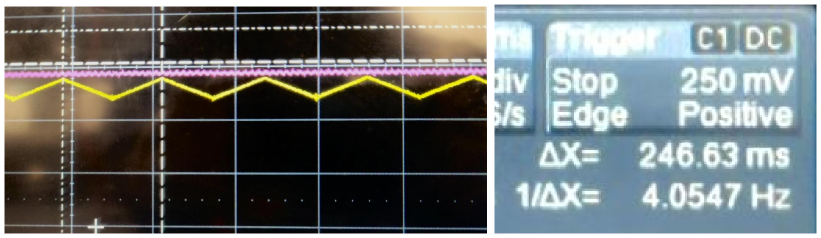


Figure 3.3: 4 Hz Note

The project was also able to accurately output our four desired waveforms - Sine, Square, Sawtooth, and Triangle, fulfilling yet another high-level requirement. Figure 3.4 shows the various waveforms produced by the project.

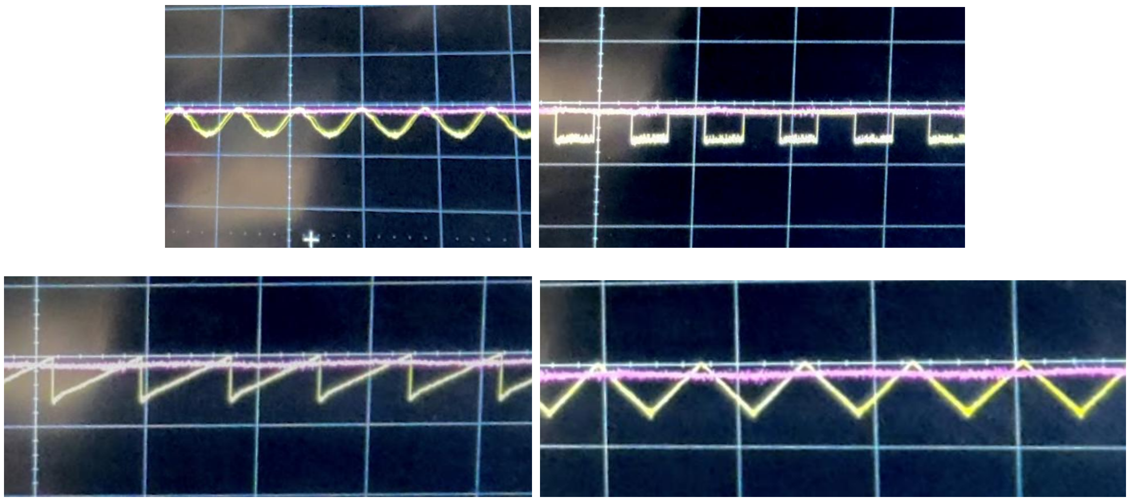


Figure 3.4: Waveforms. Sine Wave (Top Left), Square Wave (Top Right), Sawtooth Wave (Bottom Left), and Triangle Wave (Bottom Right).

3.3 Output Control Subsystem

To verify the OCS's functionality matched the requirements with Table 3 in Appendix A, the team applied an Oscilloscope to the DAC's output to determine the type of waveform being output and the amount of gain achieved. As seen in previously Figure 3.4, we were able to create all 4 waveforms and choose our waveform depending on the switch. Additionally, the team verified the functionality of the potentiometer that controlled the gain of the amplifier circuit by reading its input wave and output wave on an oscilloscope. Despite initial claims of the amplifier circuit stating a max gain of 200, we were only able to achieve a gain of 100 before voltage clipping occurred, resulting in unusable sound. We calculated gain using the following:

$$Gain = \frac{Peak-to-Peak\ output}{Peak-to-Peak\ input} \qquad (4)$$

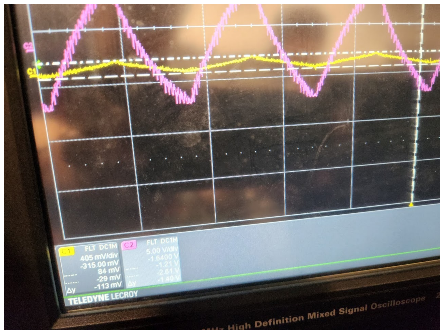


Figure 3.5: Triangle Wave Input and Output of the amplifier circuit

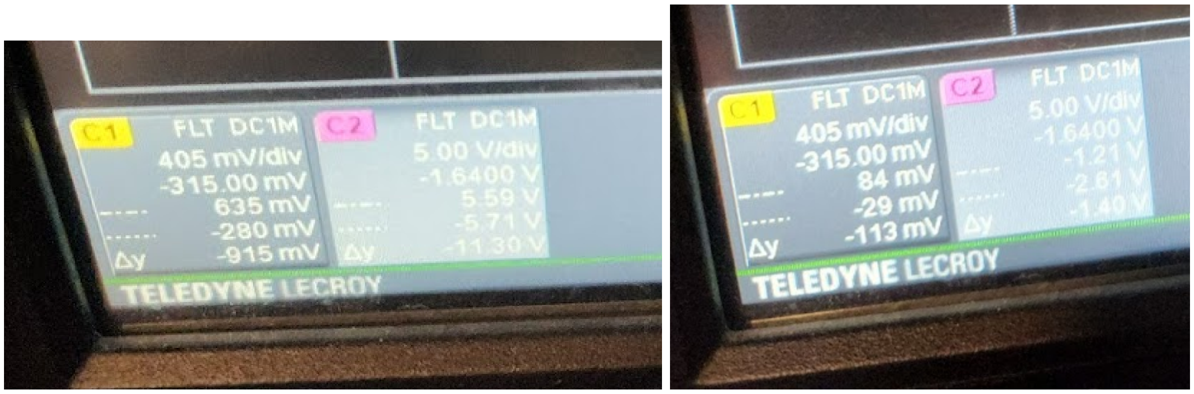


Figure 3.6: Peak-to-Peak voltage values of the input and output wave used for calculating gain.

3.4 Output Subsystem

To verify the OS's functionality matched the requirements with Table 4 in Appendix A, an Oscilloscope was applied to the Amplifier Circuit, as well as a Multimeter. As previously seen in Figure 3.6, the Amplifier Circuit successfully applies gain to the input wave, but experiences clipping at gain values above 100, failing the requirement included in the table. However, as previously seen in Figure 3.2 and calculations given by Equation 3, the project *is* able to output frequencies up to 15KHz, as indicated by the requirements. Although these two requirements were confirmed through the usage of an Oscilloscope, the final requirement was confirmed through the usage of the Multimeter - upon measurement of the speaker's wattage, the team found that the speaker struggled to exceed 10W at maximum volume, lower than the requirement's indicated 20W. Although the OS fell short in one aspect, the subsystem's functions remained sufficient enough for project operations to remain relatively unaffected.

4. Costs and Schedule

The following sections details the various costs of development - whether that be through parts or labor - that were incurred through the project's lifetime.

4.1 Parts

Various parts costs were incurred during the project's lifetime, throughout different parts of development. While a large portion of the overall cost came from the shipping of parts, some parts were significantly more expensive than others - namely, the RPi3 and Speaker were some of the costliest items utilized throughout the project. Table 5 in Appendix B shows the costs of parts that were ordered throughout the project, and indicate the total cost of development from a hardware perspective of the project.

4.2 Labor

In addition to the parts cost of the project, a labor cost was estimated using the following formula for each member of the project, as well as estimated labor for other individuals such as the machine shop:

$$Ideal\ Salary\ (Hour\ Rate)\ \times\ Actual\ Hours\ Spent\ \times\ 2.5 \qquad (4)$$

where *Ideal Salary* is estimated to be a \$20 per hour rate. The total hours per individual is given by Table 6 in Appendix B. See such for more detail.

4.3 Schedule

Figure 4.2, 4.3, 4.4, and 4.5, located in Appendix C, provide detail on the development process that took place throughout the project's lifetime. Figure 4.2 shows the overall development schedule of every section included in the same image, while Figure 4.3 shows the schedule regarding the project's Conception and Research, Figure 4.4 shows the schedule regarding the project's Software Design, and Figure 4.5 shows the schedule regarding the project's Circuit Design and Test Results Verification. These schedules represent the start and completion dates of various tasks set out during the duration of the project.

5. Conclusions

The following section provides the team's reflection on the project such as accomplishments, uncertainties, ethical considerations, and future work.

5.1 Accomplishments

Overall the project was able to achieve the main functionality central to the user experience, with the full range of musical frequencies, waveform variability, and multi-note polyphony achieved in full. Although the project was unable to drive its speaker at the target wattage, its sound produced - although quite noisy - is still more than sufficient for the user from a volume-experience perspective. Further refinement of the project could produce quality results.

5.2 Uncertainties

Though the project worked almost in full, the quality of sound produced by the project did not match expectations. The output sound contained audible noise, and did not resemble the intended waveform when sufficiently amplified past a value of 100, caused by Voltage Clipping due to the Amplifier's limitations. The precision of the clock also presented uncertainties - the variations in its rate caused inconsistencies within the playback rate, which ultimately decides the output of the project, producing notes that deviated from their intended frequencies. Additionally, inconsistencies in power caused by the Step-Down Converters forced the project to utilize the RPi3's Micro-USB port plugged into an appropriate charger for power, as opposed to being powered by the project itself. Overall, component failures throughout the project as well as inadequate circuit design to prevent such led to the large amount of uncertainty and inconsistency experienced by project developers.

5.3 Ethical and Safety Considerations

Our project follows the IEEE Code of Ethics to promote the highest standards of integrity, responsibility, and professionalism during the project's development. While there are no blatant ethical violations, potential breakings of the code can occur. The team focused on II of the code, treating "all persons fairly and with respect, to not engage in harassment or discrimination, and to avoid injuring others”. The project’s complexity would likely lead to frustration for each team member and potentially violation of the code. Additionally, the team strived “to ensure this code is upheld by colleagues and co-workers” since each team member had responsibilities outside the senior design class, leading to a lack of support to colleagues to follow the code. Consistent monitoring of personal integrity, proper documentation of additional rules and expectations to follow, and accountability were implemented throughout the project’s development

In addition to ethical concerns, safety concerns were identified at the start of the project where potential misuse of the project and lab equipment can lead to harm such as electrocution, burns, and unpleasant noise to the ear. Many precautions were taken such as avoiding the adjustment of the system while in operation, starting at a low volume before increasing volume, and ensuring the operation in low voltage conditions.

5.4 Future Work

Though the project’s applications are relatively niche, the project’s applications areas are extremely essential to increasing accessibility to electronic music synthesis methods. Several improvements could be made to significantly improve the project’s performance. First, utilizing a DAC with a higher-bit resolution would produce sound more accurate and consistent to the sound found within the modern music range, increasing the precision of our device as well as possibly reducing the noise. Secondly, improving the amplifier circuit, through increasing its amplification capabilities to achieve the full gain effect of 200 and reducing the noise it creates, would significantly improve the sound quality output by the project as well as its volume range. Lastly, incorporating a different, cheaper microcontroller, like the Raspberry Pi Pico, would significantly reduce the cost of the project, simplifying the project’s design and size as well as making the project more accessible to the public. While the project has been proven to be relatively capable, further refinement and development could yield promising results and elevate the project to commercial viability.

6. References

[1] "Expanded MIDI 1.0 Messages List (Status Bytes)", web page. Available at: https://www.midi.org/specifications-old/item/table-2-expanded-messages-list-status-bytes

[2] Breve, Bernardo & Cirillo, Stefano & Cuofano, Mariano & Desiato, Domenico. *Perceiving space through sound: mapping human movements into MIDI*. pp. 49-56, 2020.

[3] G. P. Scavone, "Wavetable Synthesis", web page. Available at: https://www.music.mcgill.ca/~gary/307/week4/wavetables.html

[4] G. Scavone, "Midi/Frequency Conversion", web page. Available at: https://www.music.mcgill.ca/~gary/307/week1/node28.html

[5] "How to Build a Clock Circuit with a 555 Timer", web page. Available at: https://www.learningaboutelectronics.com/Articles/555-timer-clock-circuit.php

[6] "Audio Amplifiers", web page. Available at: http://techlib.com/electronics/audioamps.html

[7] "Nyquist Frequency", web page. Available at: https://www.gatan.com/nyquist-frequency

[8] "IEEE Code of Ethics", web page. Available at: https://www.ieee.org/about/corporate/governance/p7-8.html

Appendix A: Requirement and Verification Tables

Table 1: Power and Output Conversion Subsystem Requirements and Verification

Requirements	Verification
- 5V Power Supply should vary from 4.85V - 5.15V - 3.3V Power Supply should vary from 3.15V - 3.45V	- Voltage read from a multimeter should maintain the given range for 30 seconds

Table 2: Signal Processing Conversion Subsystem Requirements and Verification

Requirements	Verification
- The Raspberry Pi must be able to read Serial input from its serial ports utilizing the MIDI protocol, at the rate determined by the protocol (31250 bits per second) - The DAC must contain a resolution of a minimum of 10-bits - The DAC must be able to output waveforms with frequencies within the target range, up to 15KHz - The DAC must be able to produce 4 different waveforms (Sine, Square, Triangle, Sawtooth)	- Verify Serial reading by passing in test input with predefined waveform, and verifying based on output audio - Utilize all bits of DAC Components capable of 10-bits. Evaluate based on waveform clarity with Oscilloscope - Verify DAC Frequency Range and waveform shape using Oscilloscope and test input

Table 3: Output Control Subsystem Requirements and Verification

Requirements	Verification
- Changing the switches changes the output sound and output waveform - Intensity of sound should be adjustable using a dial provided by the subsystem	- Measure waveform for shape and intensity using an Oscilloscope to verify both requirements

Table 4: Output Subsystem Requirements and Verification

Requirements	Verification
- The amplifier must be able to receive and amplify waveforms provided by the **Output Control Subsystem,** and output an amplified waveform with a gain between 20 and 200 to the speaker - The Speaker must be able to output frequencies within the range defined by the high-level requirements, which reach up to 15KHz - The Speaker must be able to output sound with a power of 0W to 20W	- Utilize Oscilloscope to analyze spectrum of sound, reading the original unamplified wave and the amplified wave and visually determining the gain - Utilize Oscilloscope to analyze the range of the Speaker, with test input - Utilize Multimeter to assess power output of Speaker, with test input

Appendix B: Parts and Labor Costs

Table 5: Parts Costs

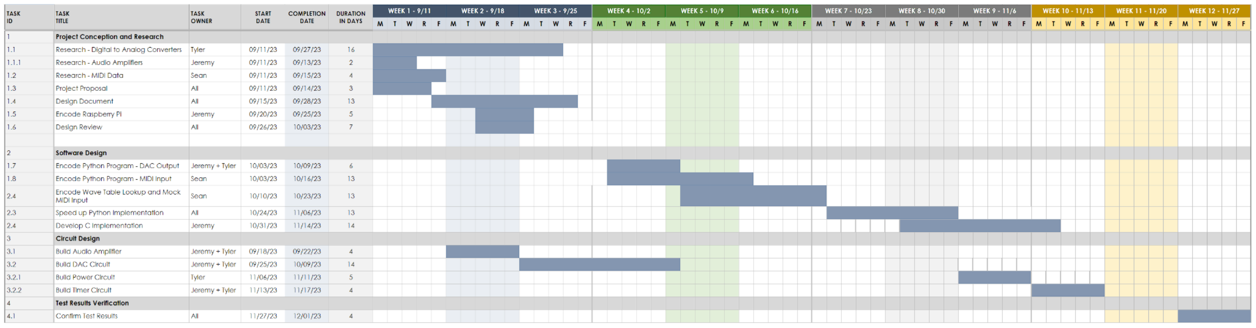
Description	Manufacturer	Part #	Quantity	Cost ($)
Raspberry Pi 3 Model B Board	Raspberry Pi		1	65
SSOP-16 TO DIP-16 SMT ADAPTER	Chip Quik Inc.	PA0182-ND	3	11.07
SMOOTH FLOW LOW TEMP SOLDER PAST	Chip Quik Inc.	315-NC191LT10-ND	1	7.95
SPEAKER 8OHM 800MW TOP PORT 88DB	Soberton Inc.	433-1104-ND	3	5.25
IC AMP CLASS AB MONO 325MW 8SOIC	Texas Instruments	LM386MX-1/NOPBCT-ND	3	3.33
IC DAC/AUDIO 24BIT 200K 16 SSOP	Texas Instruments	296-41373-1-ND - Cut Tape (CT)	5	9.6
IC AMP CLASS AB MONO 700MW 8DIP	Texas Instruments	296-43959-5-ND	4	5.12
SPEAKER 8OHM 800MW TOP PORT 88DB	Soberton Inc.	433-1104-ND	1	1.75
IC DAC 10BIT V-OUT 8DIP	Microchip Technology	MCP4911-E/P-ND	4	7.88
CAP CER 10UF 50V X5R 1206	Murata Electronics	490-12456-1-ND	10	4.05
Varying Capacitors and Resistors	N/A	N/A	30	3
SWITCH TOGGLE SPDT 3A 120V	E-Switch	EG2447-ND	4	11.88
DC DC CONVERTER 5V 2.5W	CUI Inc.	102-4244-ND	3	8.43
DC DC CONVERTER 3.3V 1.65W	CUI Inc.	102-4243-ND	3	8.43

LM386N-3/NOPB IC AMP CLASS AB MONO 700MW 8DIP	Texas Instruments	296-43959-5-ND	1	1.28
SPEAKER 8OHM 800MW TOP PORT 88DB	Soberton Inc.	433-1104-ND	1	1.75
IC DAC 10BIT V-OUT 8DIP	Microchip Technology	MCP4911-E/P-ND	1	1.97
OPTOISO 5KV DARL W/BASE 8DIP	Lite-On Inc.	160-1795-ND	3	2.31
SMALL SGNL DIODE DO35 100V 175C	onsemi	1N4148FS-ND	3	0.3
XTAL OSC XO 20.0000MHZ HCMOS TTL	ECS Inc.	X964-ND	2	6.3
XTAL OSC XO 24.0000MHZ HCMOS SMD	ECS Inc.	XC1953TR-ND	2	1.72
DC DC CONVERTER 15V 15/-5.0V 1.4	Mornsun America, LLC	2725-QA053C-1505R3-ND	2	8.84
IC ADC/AUDIO 24BIT 96K 14TSSOP	Texas Instruments	296-31708-2-ND	2	6
IC DAC 10BIT V-OUT 8SOIC	Microchip Technology	MCP4911-E/SN-ND	2	3.66
IC DAC 16BIT V-OUT 10MSOP	Analog Devices Inc.	505-AD5683RARMZ-RL7 TR-ND	4	30.04
ESP32-S2-MINI-1 DEV BRD	Espressif Systems	1965-ESP32-S2-DEVKIT M-1-ND	2	16
XTAL OSC XO 4.0000MHZ HCMOS TH	CTS-Frequency Controls	110-MXO45HS-3C-4M000 000-ND	2	6.42
XTAL OSC XO 4.0000MHZ HCMOS TH	CTS-Frequency Controls	110-MXO45-3C-4M000000 -ND	2	4.24
Total				301.46

Table 6: Labor Costs

	Rate	Hours	Total ($)
Jeremy Lee	20	121	2420
Sean Liang	20	110	2200
Tyler Shu	20	115.5	2310
Other	20	10	200
Total		356.5	7130

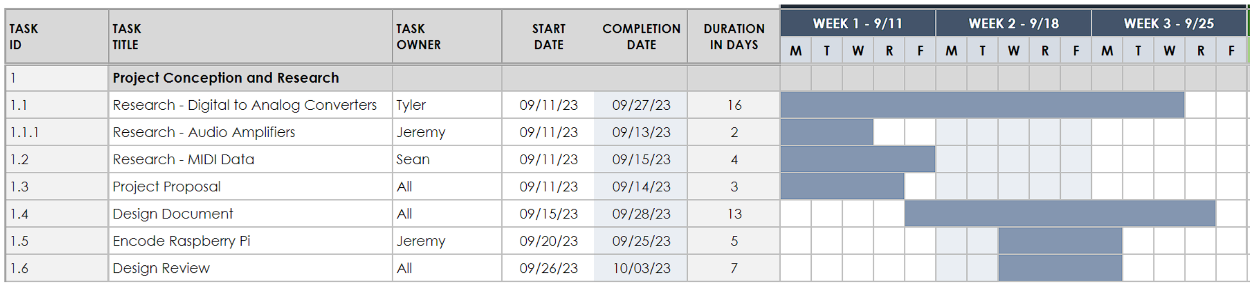
Appendix C: Project Schedules



TASK ID	TASK TITLE	TASK OWNER	START DATE	COMPLETION DATE	DURATION IN DAYS
1	**Project Conception and Research**				
1.1	Research - Digital to Analog Converters	Tyler	09/11/23	09/27/23	16
1.1.1	Research - Audio Amplifiers	Jeremy	09/11/23	09/13/23	2
1.2	Research - MIDI Data	Sean	09/11/23	09/15/23	4
1.3	Project Proposal	All	09/11/23	09/14/23	3
1.4	Design Document	All	09/15/23	09/28/23	13
1.5	Encode Raspberry Pi	Jeremy	09/20/23	09/25/23	5
1.6	Design Review	All	09/26/23	10/03/23	7
2	**Software Design**				
1.7	Encode Python Program - DAC Output	Jeremy + Tyler	10/03/23	10/09/23	6
1.8	Encode Python Program - MIDI Input	Sean	10/03/23	10/16/23	13
2.4	Encode Wave Table Lookup and Mock MIDI Input	Sean	10/10/23	10/23/23	13
2.3	Speed up Python Implementation	All	10/24/23	11/06/23	13
2.4	Develop C Implementation	Jeremy	10/31/23	11/14/23	14
3	**Circuit Design**				
3.1	Build Audio Amplifier	Jeremy + Tyler	09/18/23	09/22/23	4
3.2	Build DAC Circuit	Jeremy + Tyler	09/25/23	10/09/23	14
3.2.1	Build Power Circuit	Tyler	11/06/23	11/11/23	5
3.2.2	Build Timer Circuit	Jeremy + Tyler	11/13/23	11/17/23	4
4	**Test Results Verification**				
4.1	Confirm Test Results	All	11/27/23	12/01/23	4

WEEK 1 - 9/11, WEEK 2 - 9/18, WEEK 3 - 9/25, WEEK 4 - 10/2, WEEK 5 - 10/9, WEEK 6 - 10/16, WEEK 7 - 10/23, WEEK 8 - 10/30, WEEK 9 - 11/6, WEEK 10 - 11/13, WEEK 11 - 11/20, WEEK 12 - 11/27

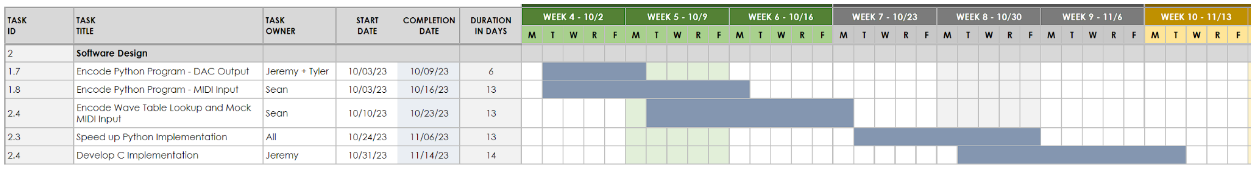
Figure 4.2: Overall Development Schedule



TASK ID	TASK TITLE	TASK OWNER	START DATE	COMPLETION DATE	DURATION IN DAYS
1	**Project Conception and Research**				
1.1	Research - Digital to Analog Converters	Tyler	09/11/23	09/27/23	16
1.1.1	Research - Audio Amplifiers	Jeremy	09/11/23	09/13/23	2
1.2	Research - MIDI Data	Sean	09/11/23	09/15/23	4
1.3	Project Proposal	All	09/11/23	09/14/23	3
1.4	Design Document	All	09/15/23	09/28/23	13
1.5	Encode Raspberry Pi	Jeremy	09/20/23	09/25/23	5
1.6	Design Review	All	09/26/23	10/03/23	7

WEEK 1 - 9/11, WEEK 2 - 9/18, WEEK 3 - 9/25

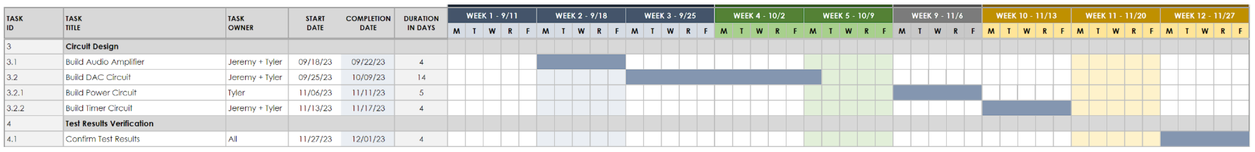
Figure 4.3: Development Schedule - Project Conception and Research



TASK ID	TASK TITLE	TASK OWNER	START DATE	COMPLETION DATE	DURATION IN DAYS
2	**Software Design**				
1.7	Encode Python Program - DAC Output	Jeremy + Tyler	10/03/23	10/09/23	6
1.8	Encode Python Program - MIDI Input	Sean	10/03/23	10/16/23	13
2.4	Encode Wave Table Lookup and Mock MIDI Input	Sean	10/10/23	10/23/23	13
2.3	Speed up Python Implementation	All	10/24/23	11/06/23	13
2.4	Develop C Implementation	Jeremy	10/31/23	11/14/23	14

WEEK 4 - 10/2, WEEK 5 - 10/9, WEEK 6 - 10/16, WEEK 7 - 10/23, WEEK 8 - 10/30, WEEK 9 - 11/6, WEEK 10 - 11/13

Figure 4.4: Development Schedule - Software Design



TASK ID	TASK TITLE	TASK OWNER	START DATE	COMPLETION DATE	DURATION IN DAYS
3	**Circuit Design**				
3.1	Build Audio Amplifier	Jeremy + Tyler	09/18/23	09/22/23	4
3.2	Build DAC Circuit	Jeremy + Tyler	09/25/23	10/09/23	14
3.2.1	Build Power Circuit	Tyler	11/06/23	11/11/23	5
3.2.2	Build Timer Circuit	Jeremy + Tyler	11/13/23	11/17/23	4
4	**Test Results Verification**				
4.1	Confirm Test Results	All	11/27/23	12/01/23	4

WEEK 1 - 9/11, WEEK 2 - 9/18, WEEK 3 - 9/25, WEEK 4 - 10/2, WEEK 5 - 10/9, WEEK 9 - 11/6, WEEK 10 - 11/13, WEEK 11 - 11/20, WEEK 12 - 11/27

Figure 4.5: Development Schedule - Circuit Design and Test Results Verification

Appendix D: MIDI Data

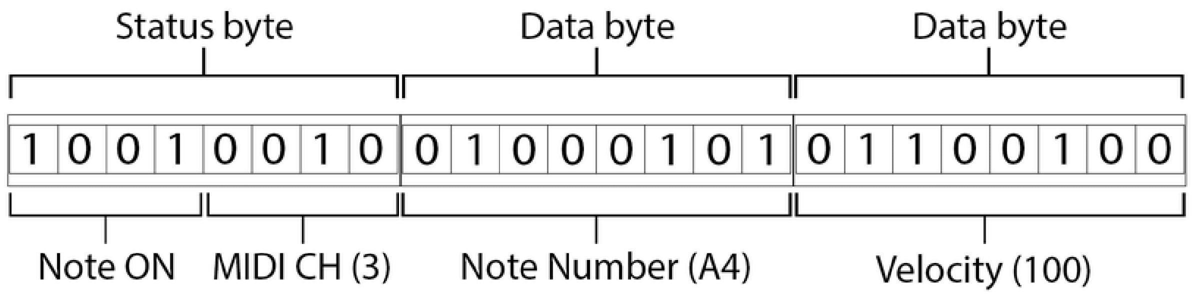


Figure 4.6: Example "Note On" MIDI Message

Appendix E: Schematics

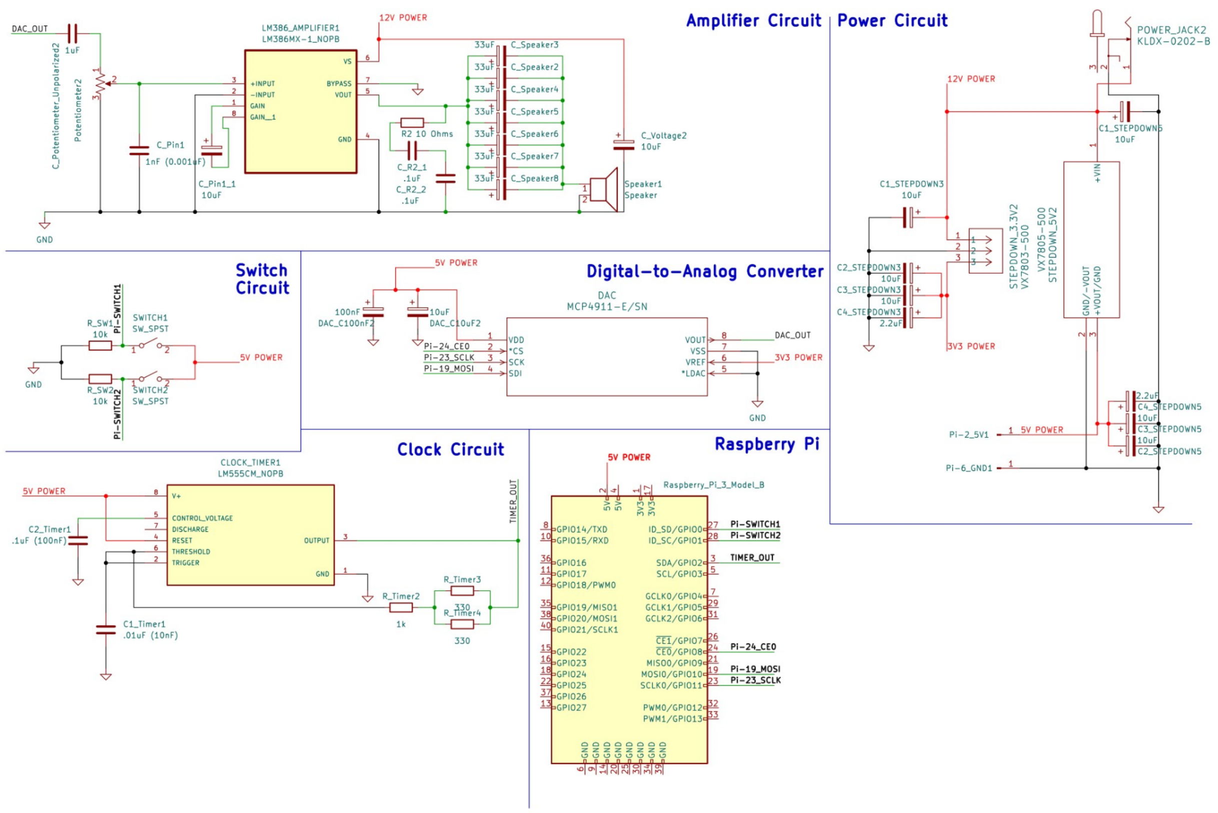


Figure 2.5: Final Schematic Design