OmniDrum
UCF Senior Design 1 - Final Report

Group 41 Members

Wylde Simpson - Electrical Engineering
Soufiane Louiba - Electrical Engineering
Jameson Palmer - Computer Engineering

Review Committee

Dr. Borowczak - ECE
Person 2 - Info
Person 3 - Info

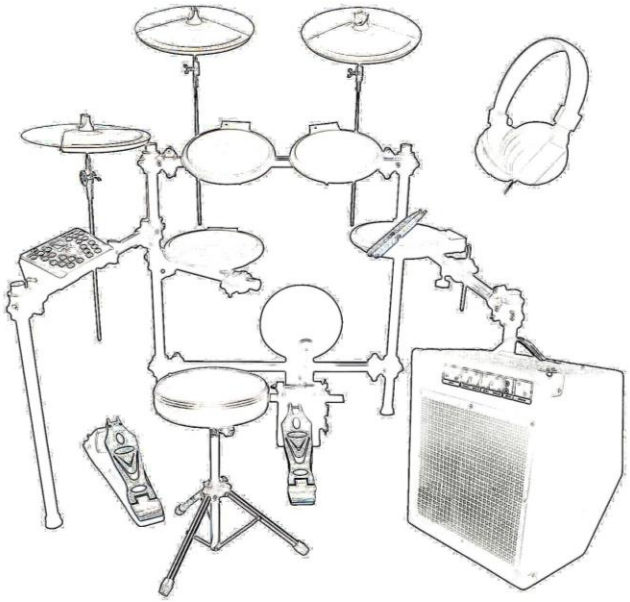


Figure 1.1 Project Mockup
by Jameson Palmer

Table of Contents

List of Tables

List of Figures

1. Project Narrative

1.1 Project Background & Context

Electric drum kits have been commercially available since the 80's and are chosen by many musicians for a number of reasons. Electric drum kits are composed of various pads that imitate conventional drums in shape and size. These pads are sensitive to being struck with a drumstick and will send a signal to a central unit that plays a drum sound according to which component is being struck. These pads can also be sensitive to the velocity of the strike and change the dynamics of the sound to match the playing style of the user.

Musicians choose electric drum kits for a variety of reasons. Electric drum kits are much quieter than conventional drums which is useful during practice in settings such as apartment buildings or other locations where creating loud sounds may not be acceptable. Another reason someone might choose an electric drum kit is because of their versatility. The sounds produced by the kit can be changed and usually electric drum kits will come with a variety of sounds built in which allows the user to experience the sounds of many different types of drums without having to own each one individually.

Conventional drum kits can consist of instruments that wear with use or can have serious damage if misused, mishandled, or damaged in transport. These instruments can combine to a significant financial cost. This cost can be an increased liability should anything get damaged. Digitizing the sounds each instrument makes not only allows someone to play these recordings back, they can be preserved through years and decades with no worry of sound degradation unlike their acoustic counterparts. Cost of replacing damaged components with this electric drum kit is reduced by the components used and modularity in the design.

Electric drum kits are also useful for recording and composition purposes. Generally with a conventional drum kit, recording someone playing the drums can be complicated and expensive. It may require buying multiple high quality microphones, mixing equipment, and even acoustically treating the room being used. The costs here quickly add up and may be above the budget of many players. Electric drum kits can be plugged directly into recording equipment without the need for complicated setup which reduces the entry cost to recording drums. Overall there are many reasons that musicians may want to purchase an electric drum kit as opposed to a conventional one.

1.2 Project Motivation

Quality electric drum kits can be very expensive to buy which is why some companies have made budget electric drum kits. These kits - while much lower in price compared to high quality kits - still cost a lot of money to the average young musician and are generally of lesser quality and may have less functionality. Our goal is to make an electric drum kit with a low enough cost that it can be bought

by someone on a restrictive budget, yet still has the functionality required to perform up to amateur standards. Many less experienced musicians are younger and do not necessarily have the money to purchase expensive equipment. Instruments are also cumbersome and heavy. The total weight of all drums, bells, and symbols would be drastically reduced. This project boasts a smaller footprint which would bring greater accessibility to players who might find it difficult to make an area for practicing.

Nowadays as music production software becomes available to an increasing market of people, many younger and less experienced musicians are getting access to the tools required to make their own music. Programs such as Digital Audio Workstations (DAWs) can allow beginner musicians to make professional-sounding music with many of their built in tools and recording functionalities. However even with music production tools becoming cheaper and more widespread, live drum recording tools are still harder for beginners to access due to budget constraints. Many of these people are only able to use sequencers built into their music production software, which generally are not as capable of making complex and interesting rhythms. Our project aims to provide an option that both allows for beginners to develop their skills and knowledge with drums to practice on, as well as providing a tool that can actually be used alongside music productions software to allow for players to incorporate their own drumming into music they want to create.

1.3 Project Functionality

Users of **OmniDrum** will be able to attach a number of sensors to different surfaces of their liking which will correspond to a certain component of the drum set. These sensors will detect vibrations and send a signal to a central hub which will include a power supply board, microcontroller for processing information, speaker for standalone practice, amplifier board for the speaker, an optional headphones port, and a MIDI Out port.

The central hub will connect to each of the sensors and play sounds out of the speaker corresponding to which sensors receive input. The sensors may also have the function to detect the magnitude of the vibrations and thus the velocity of the strike. This will allow the hub to adjust the volume and sound according to how hard the surface was struck. The user will also be able to choose between a variety of built-in libraries with different styles of drum kits to adjust the sounds to their liking. By default the drum sounds will be played out of an onboard speaker with a knob for volume adjustment. This allows the device to be used as a standalone drum kit allowing the user to practice without the need for a separate computer. There will also be a jack to connect a pair of headphones to the device in case the user needs to be silent or prefers to listen through their own device. When the headphone jack is being used, the onboard speaker will be muted. The headphone jack can also be used to connect the **OmniDrum** to an audio interface and allow its sound to be recorded.

Another main function of the device will be its ability to output MIDI information based on the player’s inputs. MIDI (Musical Instrument Digital

Interface) is a ubiquitous communication standard used to allow musical instruments to communicate with computers and other devices. MIDI allows these devices to transmit what notes are being pressed, the velocity of the notes, and more information regarding adjustments made to parameters. Most common DAWs today are heavily reliant on the MIDI standard for creating a seamless bridge between hardware and software. Certain MIDI note values are commonly used to represent different drum hits and sounds. The **OmniDrum** will be able to communicate what drums are being hit and with what velocity to a DAW allowing the MIDI information to be recorded. It should be noted that while MIDI transmits note values and velocity, it does not transmit audio. This allows the user to choose any drum sound they have on their computer and assign it to each MIDI note being transmitted from the **OmniDrum**, making it an even more versatile product. The MIDI capability allows the **OmniDrum** to be used in the workflow of an amateur musician to produce complex and interesting drum rhythms that can be directly incorporated into their music. This is also especially helpful for traditional drummers to get their ideas out of their heads into their projects in a more familiar and natural way, as opposed to clicking around on a sequencer with limited capabilities.

2. Objectives & Requirements

The design of this project focuses on an array of sensors which represent instruments. Responsiveness is a major goal as each sensor is monitored to retrieve sensor values. The array should be designed in a way where each sensor can be acted on immediately and independently. The resulting behavior should appear to have a parallelized connection to each of our sensors. Multiple instruments are expected to operate simultaneously and have their output weave together in a concurrent playback. The time between an instrument being struck and the resulting output needs to have minimal delay. This project will have a design that strives to minimize delay while providing the simultaneous input and output operations.

2.1 Goals and Objectives

Basic Goals:

1. The device will be able to detect input from sensors and play a corresponding sound
2. The device should be able to play at least 3 sounds at once
3. The device should be able to output MIDI signals corresponding to which sensors are triggered
4. Sensors should be able to be placed on most hard surfaces and detect a hit

Advanced Goals:

1. The device should be able to detect the velocity of each hit and play a different sound depending on the velocity
2. Sensitivity of the sensors should be adjustable through a potentiometer
3. The device should have a headphones/line out jack, which mutes the onboard speaker when in use
4. Integrate an amplifier for headphone usage listed above. The amplifier may be disabled as to not interfere when connected to another sound system to not add unnecessary gain.
5. The device should have multiple sets of built-in sounds

Stretch Goals:

1. Stylization such as waveform or frequency spectrum display
2. Modularity such that an instrument may be added/removed with ease.
3. **Bump/crosstalk detection** to prevent sensors from picking up vibrations on the frame from accidental knocks or another struck sensor.
4. **Demo mode** where sample drum routines could play on their own while lighting up the instrument that is played. A user may follow along the routine by striking an instrument when it flashes.

2.2 Requirements & Constraints

Requirement & Constraints	**Description**
Polyphony	Must be able to play at least 3 sounds at once
MIDI Output	Must output correct MIDI information notes 35 through 45
Vel. Sensitivity	Must be able to detect 3 levels of velocity
Detection Rate	Must be able to detect hits that are at least 100ms apart
Low Latency	Latency must be no more than 100ms
Low Power Draw	Target power for circuit board should be less than 15 Watts
Audible Output	External speaker should achieve at least 65 dBA

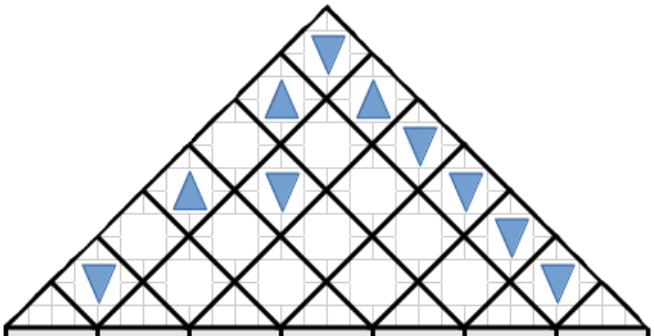
Table 2.1 Requirements & Constraints

2.2.1 Software

Objectives for software design are to implement data storage management. An external storage device is used to contain data for the project. Implementation for accessing external storage is a basic need for the purpose of this device. Software will take files from this storage and use the data for following processes. We must devise the steps necessary for communicating from the microcontroller to the external device. The software support of filesystems and file directory structure is an inherent component to the design.

Content from storage includes the audio assets that will play through the device. Knowledge and tradeoffs of audio codecs needs to be found to determine which codec fits the platform we are designing for. The embedded system will manage multiple files simultaneously along the process of decoding. They will each take CPU and memory resources away from the rest of the system. The real-time behavior of the system must be developed in a way that does not conflict with decoding our audio resources. Concurrent playback is a goal for our output signals which adds complexity to task management.

House of Quality Table:



		Latency	Power	Sensitivity	Polyphony	Audible Output	Detection Rate	Cost
		-	-	+	+	+	+	-
Latency	-	↑↑	↓		↑↑		↑↑	↓
Power	-	↓	↑↑	↓	↓	↓↓	↓	↑
Audio Quality	+	↑	↓	↑	↑↑	↑		↓↓
Durability	+							↓
Cost	-	↓	↑	↑	↓	↓↓	↓	↑↑
Targets for Engineering Requirements		<= 100 ms	<= 15 Watts	>= 4 levels	>= 3 level mux	>= 65 dBA	>= 10 Hz	<= $115 prototype

Figure 2.1 House of Quality
by Jameson Palmer

2.3 Block Diagrams

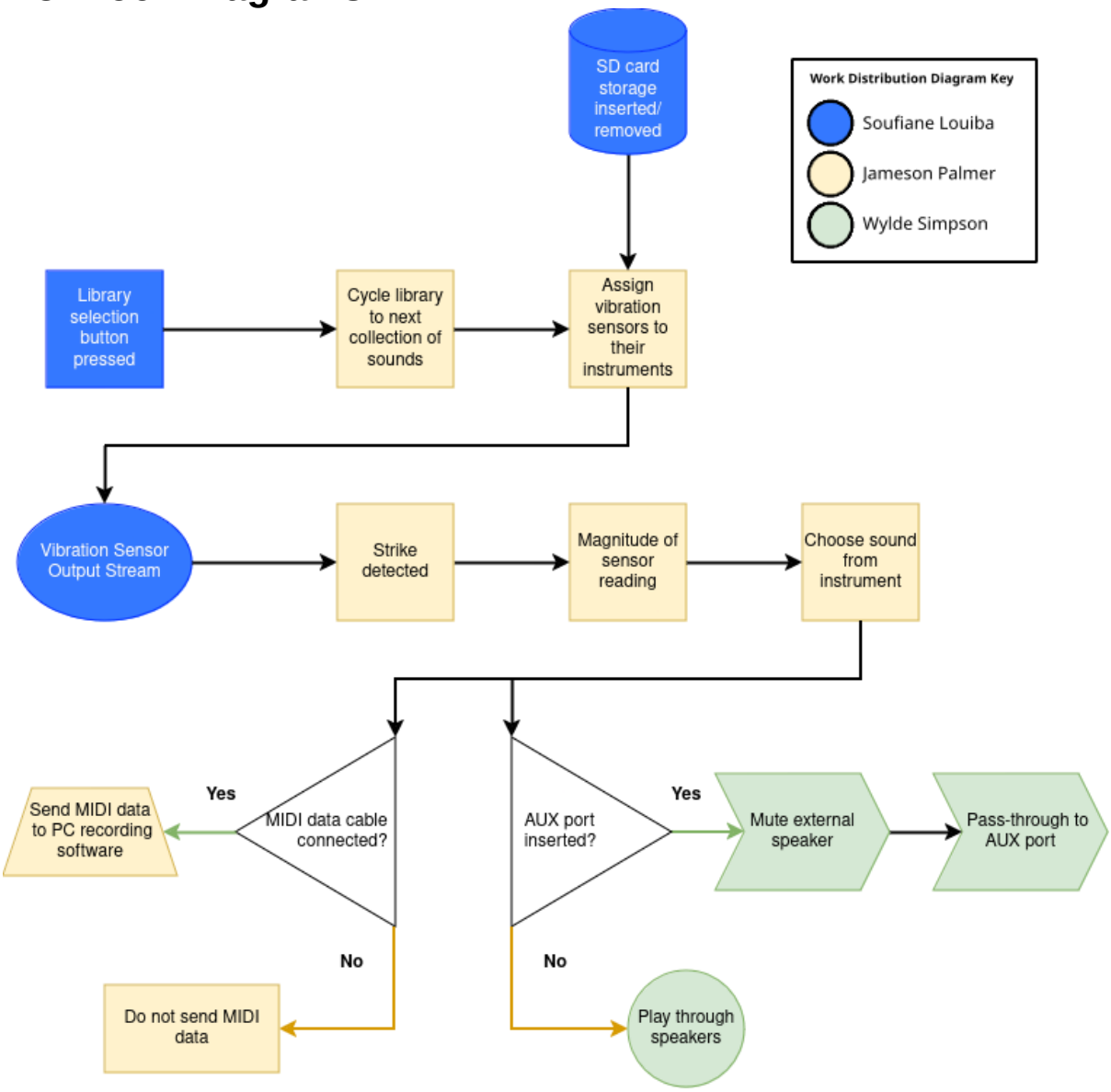


Figure 2.2 Software Flowchart
by Jameson Palmer

This diagram shows a simplified explanation of the software stages the device will follow. Software functionality is divided between these stages for better management while the code is developed. Some stages have intersection with other software functionality which helps to visualize how tasks should be arranged during operation. Determining task priority and stages of execution are detailed in their respective sections.

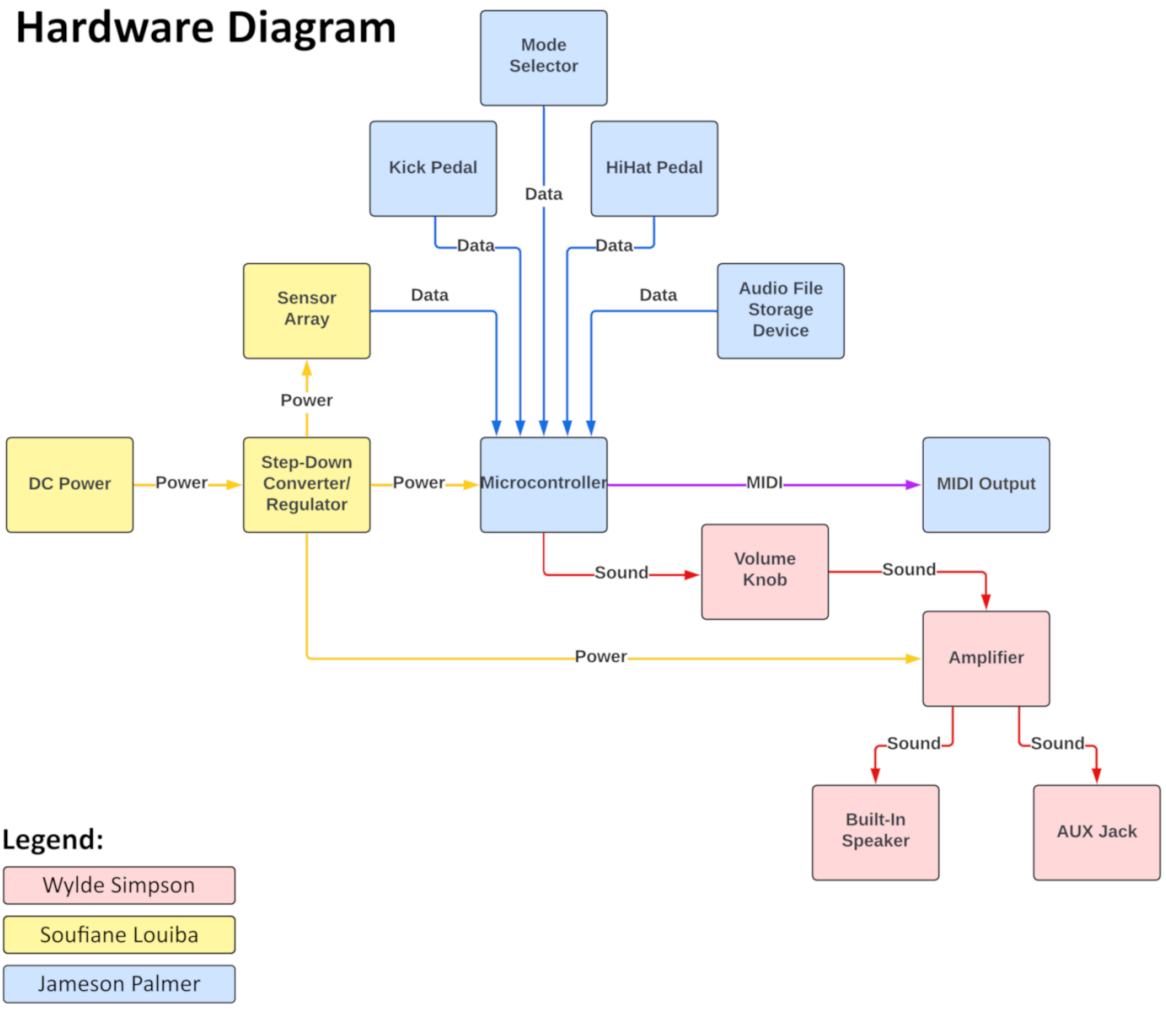


Figure 2.3 Hardware Flowchart
by Wylde Simpson

Certain tasks require collaboration by different members of the team to complete. The tasks listed here generally depict who is in charge of what, but some tasks will be worked on by multiple people in order to ensure the design works cohesively and properly.

3. Research & Part Selection

3.1 Research

3.1.1 MCU

3.1.1.1 Purpose

For this project we will be using a microcontroller to both detect input from the user as well as play the desired files at the correct time on an audio output device. The MCU will also have the job of outputting MIDI information so that the user may use the device to record the rhythms that they play onto an 3rd party piece of software capable of receiving and storing MIDI information. In order to accomplish this we need to find a microcontroller that can read audio files from a mass storage device and then transmit that data as an analog signal from an internal digital-to-analog converter, or transmit the file as a digital signal to an external digital-to-analog converter so that it may be amplified and played out loud.

Our MCU needs to be able to output multiple audio files at the same time. While playing, the user will be hitting multiple components of the drum kit at the same time and will need audio files to be able to play simultaneously. The microcontroller will be reading these files from an external storage device that will be discussed in a later section. It will also need to be able to cycle through different ‘profiles’ which will serve as separate collections of drum sounds that a user may choose from in order to play different types of drum kits.

MIDI and its role in the musical field will be discussed in a later section. Most microcontrollers should be able to transmit MIDI data over a GPIO pin. MIDI is an asynchronous digital communication standard consisting of basic 8-bit strings to transmit information on what notes are being played on an instrument and with what velocity, among other things. The MIDI standard is used ubiquitously within the world of digital music production.

3.1.1.2 External Storage Device

For our project to work correctly, we need our MCU to be able to pull audio files from an external storage device. A common way of accomplishing this would be connecting our MCU to a microSD card. Using a microSD card would allow us to easily reorganize and alter the files on the card due to the interoperability between microcontrollers and personal computers that they offer. We will be able to access the files on our own computers to troubleshoot and upload the sound files before placing the microSD card into the reader. Many microcontrollers have the ability to interface with microSD cards. There are different ways of doing this; the most common is by using SPI to allow for the microcontroller to communicate with the microSD card. Other forms of communication include SDIO and SDMMC. The microcontroller we choose will need to support these in some way.

3.1.1.3 GPIO Pins

In order for our project to work, our MCU needs to be able to receive several inputs and transmit many outputs. This means that the MCU that we choose will need to have a sufficient number of GPIO pins. We will need around 6-8 input pins for our sensors, as well as more for other various buttons, LEDs, and audio output channels. This should be achievable by most microcontrollers on the market, however in the next section we will discuss the specific capabilities required of these pins.

3.1.1.4 MIDI

After doing some research, it appears that supporting MIDI communication will actually be quite easy. MIDI communication is asynchronous at 31250 baud. Given that our microcontroller supports UART, we should be able to easily set up MIDI communication. Most microcontrollers have UART support so finding this should not be a problem.

3.1.1.5 ADC

The sensors that we are using for the project will output analog voltage that will need to be converted into a digital signal to be understood by the microcontroller. To do this we need ADCs on our microcontroller. We need at least as many ADCs as inputs that we will use. Our project will use at least 6 analog inputs and possibly more depending on what needs to be done. In this case we should consider a chip with at least 8 ADCs and an absolute minimum of 6.

3.1.1.6 DAC

We need to be able to get our microcontroller to be able to play sounds out of a speaker. In order to drive the speaker a separate amplifier circuit will be constructed. However this does not solve the issue of getting the audio signal out of the microcontroller. For that we will need to use a DAC. These can be either a separate IC used in conjunction with the microcontroller, or built into the microcontroller. In fact, using multiple DACs is a way of implementing simultaneous playback as the DACs can be tied together. These DACs need to be both external and buffered in order to be used properly. The easiest implementation would be to choose a microcontroller that has onboard DACs, preferably 3.

3.1.2 Audio Output Device

3.1.2.1 Types of Output Devices

The goal of this component of the system is to convert the electrical signal output from the MCU into auditory feedback for the user of the device. Converting

an electrical signal into a mechanical sound can be accomplished using different types of devices including electrodynamic speakers, piezoelectric buzzers, or even outside audio devices like headphones or in-ear monitors.

Electrodynamic speakers are a well known method for turning electrical energy into acoustic energy through the use of electrodynamic transducers. These generally consist of three parts: the motor, diaphragm, and suspension. An electrical signal that has been appropriately amplified is applied to the motor which causes it to ‘push’ and ‘pull’ on the diaphragm. The purpose of the diaphragm is to help radiate and project the vibrations into the air in such a way that they are both loud enough and provide an accurate frequency response. The shape of the cone used has a significance on the frequency response of the output and thus the quality of the sound. The concept of frequency response curves is discussed in the next section. The final part of an electrodynamic transducer is the suspension, the purpose of which is to support the diaphragm and attach it to the rest of the speaker assembly while still allowing it to flex and vibrate in the desired fashion.

Figure 3.1 Example of an Electrodynamic Speaker
Image Source: Author

Piezoelectric buzzers function somewhat differently than electrodynamic speakers. To produce sound piezoelectric buzzers have a membrane that flexes based on the voltage applied to it. An advantage of piezoelectric buzzers is their low cost. They are relatively small and may require an added diaphragm to project their output. The most common use for piezoelectric buzzers is for alert tones and other applications that require only a single tone. They usually receive a digital signal in the form of a square wave at the desired frequency. These are not too useful for outputting analog audio with more harmonic content but they can be used for that purpose. Even though these buzzers generally receive a 1-bit digital signal, we can use pulse width modulation (PWM) to modulate the input signal based on the frequency of the desired signal. This way a 1-bit buzzer can be used

to output varying frequencies. While 1-bit buzzers do have this capability, the output is usually plagued with much harmonic distortion and has a very limited frequency range.

Figure 3.2 Example of a Piezoelectric Buzzer
Image Source: https://www.shutterstock.com/image-photo/electronic-piezo-buzzer-1101886475

Other devices such as headphones and in-ear monitors are very similar in that they employ transducers except much closer to the ears of the users and at much quieter volumes. While we do intend to allow the user to use these devices in our final design, they will not be provided to the user and are not intended as the default listening method. We intend to use some form of a loudspeaker so that the sound from the device will be audible to the user within the room that they are in; at the same time we intend to offer a connector jack for the user to plug in their own peripheral audio device if they choose to.

For our project, the best option would be to use an electrodynamic speaker due to it having a preferable full-spectrum frequency response as compared to the other options. It also gives us the ability to have the sound fill the room so that the user and other people may hear what they are playing without having to provide their own listening device.

3.1.2.2 Frequency Response

The term "Frequency Response Curve" refers to the curve derived from relating the average amplitude of a signal and the frequency of that signal. A desirable frequency response is one that is as flat as possible such that it outputs all frequencies with the same average amplitude and in turn provides the most accurate recreation of the sound. That being said there are many cases where a flat frequency response curve may not be the most desirable to the user. While flat

frequency responses are the most useful for accurate audio replication, recreational listeners often prefer a more ‘V-shaped’ curve that has an emphasis on lower bass frequencies as well as higher frequencies such as cymbals and sibilance, along with less emphasis on mid-range frequencies. This is in part due to the fact that human ears do not have a flat response curve for which they pick up different frequencies and usually emphasize mid-range frequencies the most. In general, all people’s ears have slightly different frequency response curves based on the physical structure of the outer ear, inner ear, and even head shape.

The image below shows an example of a frequency response curve measured from a certain pair of studio headphones. The target curve in this case is not ‘flat’, but represents what the source believes would be the most desirable response curve to compensate for the natural variations in sensitivity of human ears.

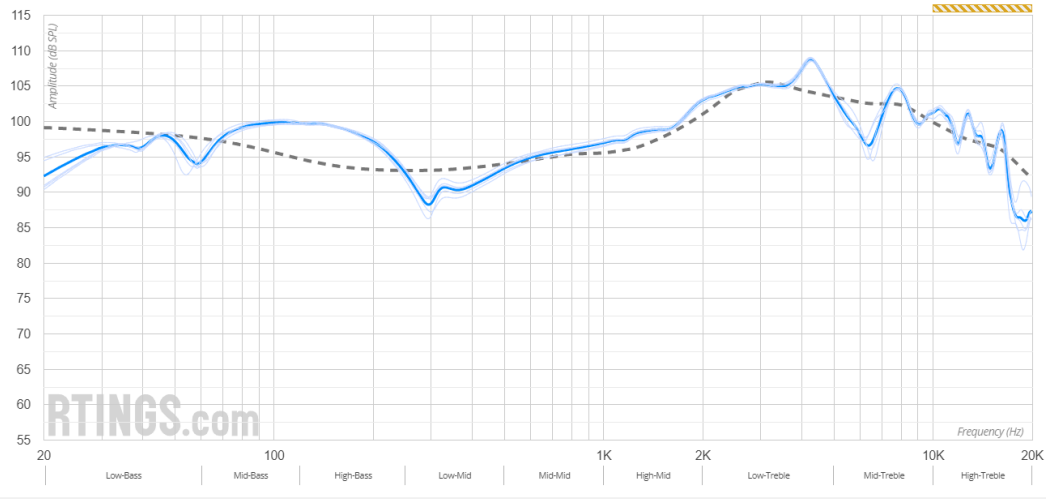


Figure 3.3 Raw FR L Graph for Audio-Technica ATH-M50x Headphones
Image Source: https://www.rtings.com/headphones/1-8/graph/25500/raw-fr-l/audio-technica-ath-m50x/295

It should be noted that this description is a simplification of a very complicated topic. Which specific response curves sound good and in which application varies greatly between different situations, and acoustic measurements can be taken in a variety of ways. For our purposes the frequency response curve allows us to get a general idea of the sound quality of the type of speaker we choose. The goal is to use a speaker that provides decent replication of drum sounds so that the device may be used without the user needing to provide their own playback device. While an excellent frequency response and high sound quality are preferable, that is not the primary focus of the project.

3.1.2.3 Speaker Wattage and Impedance

When choosing a speaker to use in our project, there are certain specifications that must be considered. We need to consider the wattage of the speaker we want to use as well as the impedance. The wattage of the speaker,

while not solely responsible for the overall loudness of the speaker, is a good estimate for how loud the speaker will be. The wattage listed on speakers is usually the peak power output, rather than the continuous power output. This is important for understanding how much power a speaker can be expected to use on average. It is harder to estimate the exact average power of a speaker since it changes with the volume and harmonic content of the output, but generally the RMS wattage is a fraction of the peak wattage. For the purposes of this project, a speaker around 1-5 watts would be appropriate. Speakers of this range are able to fill an entire room while requiring a modest amount of power. While this is not an appropriate amount of power for a live performance, the user will be able to plug in an external playback device to the output port and amplify the sound further if needed. 1-5 Watts is an appropriate amount of power for a practice setting.

The impedance rating of a speaker also has an effect on the sound output as well as the specifications of the amplifier that will need to be paired with it. Common impedance values for speakers are 4Ω, 6Ω, or 8Ω, though there are more possible values. Lower impedance is generally regarded as being better because it means that the speaker is able to get louder with less distortion than a speaker of higher impedance, however it can be more expensive as well because the amplifier needs to be able to provide higher amounts of power. For our purposes the impedance of the speaker will not matter too much so long as we can match it with a proper amplifier. If the speaker is within the power constraints of the project it should serve the purpose adequately.

3.1.3 Amplifier

3.1.3.1 Why Amplify?

In our design, the MCU will pull files from a microSD card and transmit them to a Digital-to-Analog converter. After the digital signal is converted to analog, it needs to be amplified to an appropriate level to be handled by a speaker. The required amount of amplification that the circuit will need depends on the specifications of the speaker. The amplifier used cannot be too powerful or it may damage the speaker and possibly even more of the circuit. Amplifiers need to be matched to speakers depending on both wattage and impedance. The exact specifications of the amplifier will depend on the choice of speaker and will be covered in a later section.

3.1.3.2 How to Amplify

In order to amplify our signal coming from the DAC we are going to use an amplifier IC. There are many different types of amplifier ICs with different functions and number of channels as well as amount of gain and output power. The choice of chip will be related to the output device we choose. Usually the ICs are not used standalone, but have a small amount of circuitry around them to help stabilize the input signal and power supply. For our specific amplifier we will need 1 mono channel as well as a volume control potentiometer. The IC itself should also have

some built-in protections such as short circuit protection. We would also like the amplifier to minimize total harmonic distortion in order to preserve sound quality

3.1.3.3 Low-Level & High-Level Signals

Low-Level and High-Level are terms used to generally describe the strength and/or type of signal passing through a device or cable in the world of audio. A low-level signal is the type of signal that will come out of the Digital-to-Analog converter in our circuit. Low-level signals are signals that have not been amplified such as signals coming out of electric guitars, microphones, and unpowered mixers. High-level signals are signals that have been amplified and would go into loudspeakers or PA systems. The distinction between these types of signals is important because plugging a low-level signal into a device that is expecting a high-level signal will not work due to the fact that the low-level signal is simply not strong enough to drive the device. High-level signals going into a device expecting a low-level signal have the capability to damage the equipment because of the fact that they are much more powerful than the device can handle.

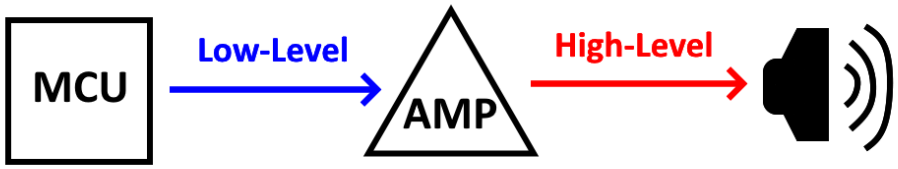


Figure 3.4 Low-Level VS High-Level
Image Source: Author

While the term ‘high-level’ describes a signal that has been amplified, it is not specific about how powerful the signal is. A signal coming out of a 30 watt amplifier is much different than the signal coming out of a 1000 watt amplifier. These terms are context dependent and have different meanings based on what the situation is. High-level in our project may be a signal amplified to drive a 3 watt speaker. In the context of car A/V however, this might mean the signal is amplified to drive 200 watt speakers.

3.1.4 Auxiliary Output Jack

3.1.4.1 Audio Jack Form Factors

There are many different form factors for audio output jacks. We will need to decide on a specific one to mount to the main housing of the device. The purpose of the AUX jack is to provide the user with the ability to connect their own personal playback devices. These can range from headphones to external speakers to even connecting to a USB audio interface or other recording device. This can be especially useful if the user would like to play the instrument quietly and without using the internal speaker, or in a situation where there is too much ambient noise for the internal speaker can be useful. It is also useful if the user would like to directly record what they are playing into a software program or some other device. Some examples of audio jack types are the well known 3.5mm jack, the ¼" jack, the XLR connector, as well as RCA cables.

The 3.5mm jack, commonly referred to as the 'Headphone Jack' on many modern devices, is one of the most ubiquitous audio jacks found on consumer products. Some examples of the male 3.5mm connector are shown below, as well as examples of the female 3.5mm jack. The 3.5mm jack can be found on computers, mobile devices, toys, instruments, and certain peripherals. Its widespread use makes it a very accessible type of connector to use in our design as most people would have some type of listening device that incorporates a 3.5mm audio connection.

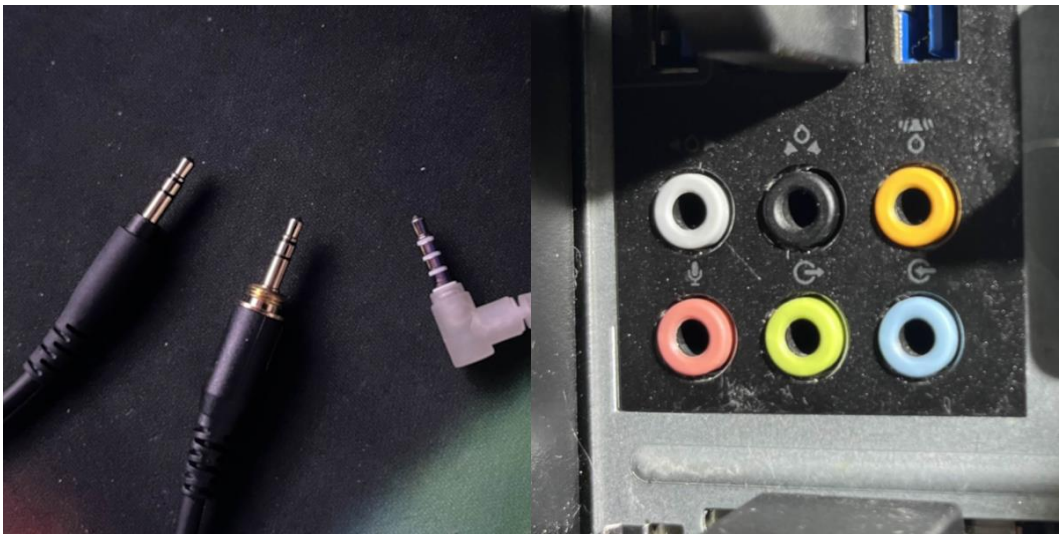


Figure 3.5 Male & Female 3.5mm Audio Connectors.
Image Source: Author

Another form factor is the ¼" jack; cables for this kind of jack are colloquially referred to as "Guitar Cables", "Instrument Cables", or sometimes "Line Cables" as they are commonly used to carry high-level or 'Line' level signals. However, ¼" cables are also very common for other uses with sound technology. The ¼" is primarily prevalent in the sound/music space as it is used to connect guitars to amps, instruments to mixers, headphones to interfaces and monitoring equipment as well as patch signals in modular synthesizers and Effects Racks. The ¼"

connector and jack appears as an oversized 3.5mm connector and jack, and it functions in practically the same way. The bigger jack is preferred in these situations due to it being more rugged and able to handle higher currents than 3.5mm cables. Most studio headphones used for more professional audio work have this type of connectors as they are expected to be plugged into a mixer or USB audio interface where ¼" jacks are the norm. It is possible that a user of the drum kit would be more likely to have listening devices that use this connector assuming they have other audio equipment, but also the 3.5mm connector cannot be ruled out since the product is aimed at a low price point and a user buying equipment on a budget is more likely to be listening with simple headphones or earbuds. This must be taken into consideration when choosing which connector type to implement into our design.

Figure 3.6 Male & Female ¼" Audio Connectors (male 3.5mm for reference).
Image Source: Author

The XLR connector is a form factor commonly used in stage settings. This connector type most commonly contains 3 pins, though a 5 pin version does exist but is much less common in audio-related situations and is found more often in lighting control scenarios for transmitting DMX information. XLR audio cables are generally used for connecting microphones to other devices such as mixers or PA speakers. The use of pins for XLR audio transmission is very similar to that of balanced 3.5mm and ¼" connectors such that there is a ground pin, a hot pin for carrying the audio signal, and a cold pin used for noise reduction. While XLR audio connections are useful in the world of sound, they generally do not fit our specific purposes and are unlikely to be useful.

Figure 3.7 Male & Female XLR Audio Connectors
Image Source: Author

The final form factor researched is the RCA connectors and jacks. RCA connectors have been around for a very long time and get their name from the Radio Corporation of America which invented the design in the 1930s. The design consists of a central protruding conductor surrounded by and separated from an outer barrel-like conductor. The central conductor carries the analog signal while the outer conductor acts as ground. These connectors have been used for many audio related purposes in the past such as connecting to radios, record players, and tape decks, as well as connecting TV sets and early home computers to external audio devices. They were also used to transfer video over both Composite video connections and later Component video connections before being succeeded by HDMI. Use of RCA cables has dissolved over the past few decades aside from specialized uses today such as Tape In/Tape Out recording on mixers as well as use in connecting car stereo systems to amplifiers and other DSP related components.

Figure 3.8 Male & Female RCA audio connectors
Image Source: Author

3.1.4.2 Tip, Ring, and Sleeve

It can be seen in **Figure 3.X** that the number of separate contacts on the male connectors of the 3.5mm and ¼" jacks varies. These contact areas are referred to as the Tip, Ring, and Sleeve. 3.5mm and ¼" connectors can be set up in different configurations such as TS, TRS, TRRS, and even the very uncommon TRRRS. The number of contact areas is representative of the number of individual wires running through the cable. The sleeve portion of the cable is dedicated to carrying ground. A cable set up in the TS configuration is used for carrying a mono signal, or a single channel of audio, where the tip carries the audio signal and the sleeve carries ground. This is the most basic configuration and lacks the advantages of more advanced configurations.

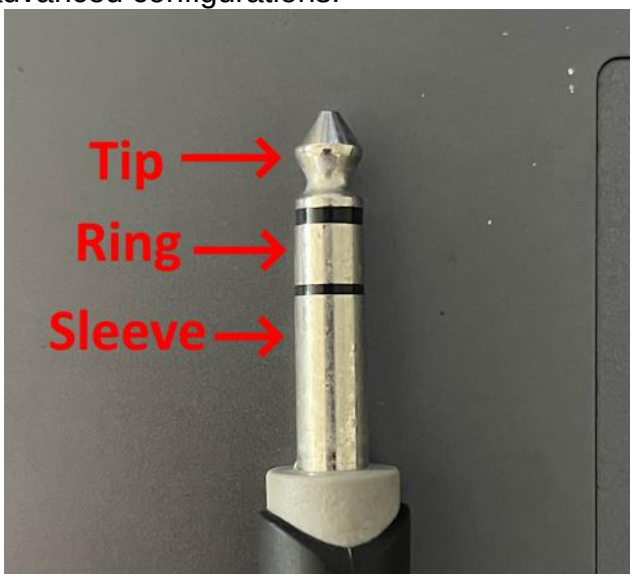


Figure 3.9 ¼" Audio Cable, Tip Ring Sleeve Example
Image Source: Author

Cables with the TRS configuration have three conductors and can be used for different purposes. One option is to use it for carrying a stereo signal, where the tip carries the left channel while the ring carries the right channel. Another use for TRS cables is carrying a single 'balanced' mono channel. In this case the tip carries the 'hot' audio signal while the ring carries a 'cold' neutral channel. As interference is picked up by cable, it has an effect on both the hot and cold conductors. At the other end of the cable, the polarity of the signal on the cold channel containing the isolated noise is inverted and the inverted noise is then added to the hot signal to phase cancel the noise in the hot signal. This technique is exceptionally effective at reducing interference noise in applications where the length of the cable is significant enough to pick up interference from the environment.

Further configurations such as the TRRS configuration are common on earbuds or headsets which include a microphone. In the case the tip carries the left audio channel, the ring adjacent to the tip carries the right audio channel, the ring adjacent to the sleeve carries the outgoing signal transmitted from the microphone, and the sleeve acts as ground. Further configurations such as TRRRS are very niche and usually only applicable in specialized applications. A cable with this configuration carries 4 conductors as well as ground and could be used to carry 4 unbalanced mono channels, 2 balanced mono channels, 2 pairs of stereo channels, or a number of other combinations of signals.

A useful feature of this form factor is that it allows for some intercompatibility between different cable configurations. For example, many computers contain a TRRS headphone jack as well as a separate microphone jack to account for the many different types of listening devices that could be connected. A pair of stereo headphones with a TRS connector can still be used normally even when plugged into the TRRS jack because the contact for the ring adjacent to the sleeve on the female connector would connect to the sleeve contact of the male connector on the headphones. This would appear to the computer as being shorted to ground when the device is plugged in and tell the computer that the listening device connected does not have a TRRS microphone connection, without transmitting any digital data between the devices. A mono listening device with a TS connector would function similarly in this scenario and only play the left channel from the computer, which isn't preferable but still allows for the listening device to function. However, it should be noted that certain combinations, such as this, can result in damage to devices if they are old or badly designed. Most devices today are designed to protect against possible damage from shorting certain connections, but not all devices are, so it is best to use the correct connectors when possible and when you know that the combination will be accepted by the device.

3.1.4.3 Headphone Amplifier

For the auxiliary output to function correctly, it needs to be amplified to the required level to drive headphones. Headphones generally have much higher impedance than loudspeakers and will require a different type of amplifier to be able to drive them. This amplifier will be a separate amplifier than the one used to drive the loudspeaker. It will require a different IC to amplify the signal properly and will have its own volume control

3.1.4.4 Muting the Internal Speaker

A common feature of many musical devices is the ability to mute the external loudspeaker when a device is plugged into the auxiliary jack, without the need for the user to flip a switch. We would like to incorporate this feature in our design to streamline the user experience. This can be done by using a 'switched' jack for the headphone output jack which will ground or power a pin on the loudspeaker amplifier IC that will shut it off so that the audio will only go to the headphones.

3.1.5 MIDI

3.1.5.1 What is MIDI?

MIDI, which stands for Musical Instrument Digital Interface, is a communications standard that was invented in 1983 for use with synthesizers and other digital instruments. The MIDI standard allows for musicians to connect instruments to computers or other instruments in order to communicate what notes are being played, with what velocity, and other information such as Control Change codes and Program Change codes on different individual channels. Over the years MIDI has become extremely successful and is almost everywhere in the world of digital music.

In our specific case, we will be using MIDI to allow the user to communicate which drums they are playing with a computer program they may own. Digital audio workstations (or DAWs) have support for communications with MIDI hosts. If the user chooses to, they will be able to connect to a DAW and record which drums they hit in real time so that they may play them back later or edit them.

It should be noted that MIDI does not transmit audio data; it transmits purely digital information. Different notes in the MIDI standard are communicated with values from 0 to 127. The reason MIDI is useful in our scenario is that it will let the user record when they play certain drums at certain times so that they can record various rhythms they may come up with. Each component of the drum set will be assigned a ‘note number’ so that the computer can display when each individual drum was played. The user can then assign a drum sound file on their computer to each ‘note’ such that when they hit that specific drum, a chosen sound plays in the computer. This allows the user to endlessly customize the sounds of their entire drum kit, record it into the computer, and adjust the mixing and effects on the drum kit to eventually use it in a musical project in conjunction with other instruments they may record. The user will have to provide their own computer and DAW in order to utilize this feature, adding MIDI capability to our device only gives it the ability to connect to this kind of system in the first place.

3.1.5.2 How does MIDI communication work?

MIDI communication is done asynchronously at a baud rate of 31250 Hz. Information is sent in 8-bit packets containing note values, velocity values, and further information allowing a musician to change parameters on different devices remotely. MIDI cables generally consist of two 5-pin male ends which plug into female jacks at either end. The constraints posed by the requirement for MIDI capability are minimal and we should be able to easily find a microcontroller that supports this.

3.1.5.3 What note values to use?

MIDI note values range from 0 - 127 which gives us a lot of options for choosing which notes should correspond to each component of the drum kit. Luckily, the music production community has decided common note values for different types of drum sounds. Although these are not strictly the rule, many brand name products stick to these values as their ubiquity allows for widespread compatibility with other products. According to zendrum.com the commonly used MIDI drum values are as follows

35	Acoustic Bass Drum
36	Bass Drum 1
37	Side Stick
38	Acoustic Snare
39	Hand Clap
40	Electric Snare
41	Low Floor Tom
42	Closed Hi-Hat
43	High Floor Tom
44	Pedal Hi-Hat
45	Low Tom
46	Open Hi-Hat
47	Low-Mid Tom
48	Hi-Mid Tom
49	Crash Cymbal 1
50	High Tom
51	Ride Cymbal 1

Figure 3.10 Common MIDI Drum Values
Image Source: https://www.zendrum.com/resource-site/drumnotes.htm

These values encompass the different components of the drum set that we plan to implement for the user. This should allow the user to be able to easily use our drum kit with drum kit presets that they may have in their DAW, as well as customize the sounds of it to their liking, enabling more versatile use cases of the drum kit. It should be noted that our project will not incorporate all of the instruments listed here, and that the list of common MIDI drum note values continues past the scope of this list.

3.1.6 Types of Sensors

3.1.6.1 Applicable Sensors

The user interface for our drum kit will consist of several modules such as the cymbal, hi-hat, snare, and bass. The designs for each module will incorporate

a sensor to detect a user's strike on the instrument. Translating a strike into digital communication with the mainboard will be the duty of whichever sensor best applies. What can be detected from a strike on the instrument? The plane of each instrument will encounter a vibration. Acoustic instruments are fundamentally designed for vibration frequencies to create music so a vibration sensor would seem to be a good candidate for our design. Each strike on a surface could have its force measured as well. The pressure or force placed on the surface area could be connected to a force sensor which can measure many different levels of force exerted. Another alternative would be to consider the strike surface being connected to an actuator such that it may depress from a hit. At greater force on the instrument the actuator would depress at a greater velocity. Measuring the velocity could then provide us enough information to determine the force of a strike at many different levels.

3.1.6.2 Force Sensor

A force sensor, also known as a pressure sensor, measures the amount of force applied to it. Many of these devices work by converting the mechanical energy applied to them into electrical energy. Some subtypes of this kind of device include strain gauges and piezoelectric sensors. Strain gauges are often made of metal foil wrapped by wire which encases a plastic or ceramic cylinder. Under stress the electrical changes with the deformation of its shape. For piezoelectric sensors, they are often made of a crystalline structure such as quartz. Ceramics can also be made in a molecular structure that has a piezoelectric characteristic. The piezoelectric effect is an electric charge generated by a force on this material which will bend the crystalline structure. The deviation of these molecules will shift positive and negative charge along the crystal. The amount of charge generated is proportional to the force applied to the material and this charge can be measured through an ADC. Measuring the charge through an ADC can then allow us to measure the amount of force. Both of the described sensors would need power supplied to them. Although the piezoelectric effect creates an electric signal, many of the available sensors require a power source for the added circuitry. This should be considered when wiring our instrument modules to the controller board as there should be wires for power as well as data.

3.1.6.3 Velocity Sensor

These devices can observe displacement over time. Some implementations of velocity sensors include optical sensors which measure light as it is deflected off of a target. Light can be directed back to a photodiode which generates electric current when subject to photons. A dedicated beam of light is integrated with this type of sensor. When in motion, the displacement changes the refraction of this light beam on the photodiode so a rate of change can be used to calculate the velocity of our target. For our purposes a velocity sensor would use a linear encoder which can detect velocity along one direction. When an instrument is struck the sensor can observe a piston as it is depressed inward. The target of our

sensor would be the end of the piston. Our implementation would need to have a low latency response so detecting velocity with a light beam could minimize the time for a response from our instruments. Due to the need for an integrated light source, this type of sensor would need active power delivery to operate. Implementation of this device should consider wires to dedicate a power source as well as wires to receive measurement data.

3.1.6.4 Vibration Sensor

These sensors could be used to measure the shaking of a strike surface on our drum kit. These can implement piezoelectric sensing technology or accelerometers. The sensor could be attached to the strike surface which is our target to be measured. It would detect the changes of motion of the target over time. Piezoelectric materials would have stress placed on their shape to create a charge across the material. Accelerometer-based implementations could detect the rate of motion over time to measure a frequency of a strike. These accelerometers can also use the piezoelectric materials.

3.1.7 Sensor Design

3.1.7.1 Piezoelectric Sensors

Piezoelectric sensors are one of the most important components of our project; these sensors constitute an end product combining functionality with cost-effectiveness. The term “Piezo" originates from the Greek word, meaning, press. This sensor consists of two defined zones on a 27mm-brass-disc diameter: the inner circle contains a slice of quartz crystal that generates a negative output voltage, while the outer circle generates a positive output voltage.

Piezoelectric occurs when a mechanical pressure is applied on the christal; in our case, when the drum pad is struck with the drumsticks. The hit causes the piezo sensor to activate, producing an alternating current (AC) at the output, which runs through the cable to the module (or “brain”) and triggers it to produce a sound, every sound in the OmniDrum has its own sample which is different from a pad to another, the user then will have the option to play the sound back through either speaker or headphones. The strength of the electrical signal is directly proportional to the force of the hit, making the sensor highly responsive to varying levels of pressure. A light tap produces a weaker signal, resulting in a softer sound, while a harder hit generates a stronger signal, leading to a louder sound. This makes the sensor highly responsive to different playing dynamics, letting the drummer control the sound’s volume and intensity naturally.

One of the stand out reasons for choosing this sensor is its effectiveness. At less than $7 for a pack of 50, it provides a high-performance, low-cost solution, making it ideal for our project. This combination of affordability and precision makes the Piezoelectric sensor an excellent choice for applications where sensing force and pressure are critical.

In a typical circuit with piezoelectric sensors, which generate small voltages when deformed, a pull-down resistor connected between the sensor's output and ground ensures that when the sensor is not generating voltage, the output is reliably pulled down to 0V (or the low logic level). This allows for cleaner and more consistent voltage readings when the sensor is triggered, reducing noise or false readings.

In our case, we will use 1M Ohm resistors as pull-down resistors in parallel with the piezoelectric elements. The purpose of it is to stabilize the voltage readings from the piezo elements. Without the pull-down resistors, the output from the piezo sensors could float when inactive, leading to unreliable or noisy readings.

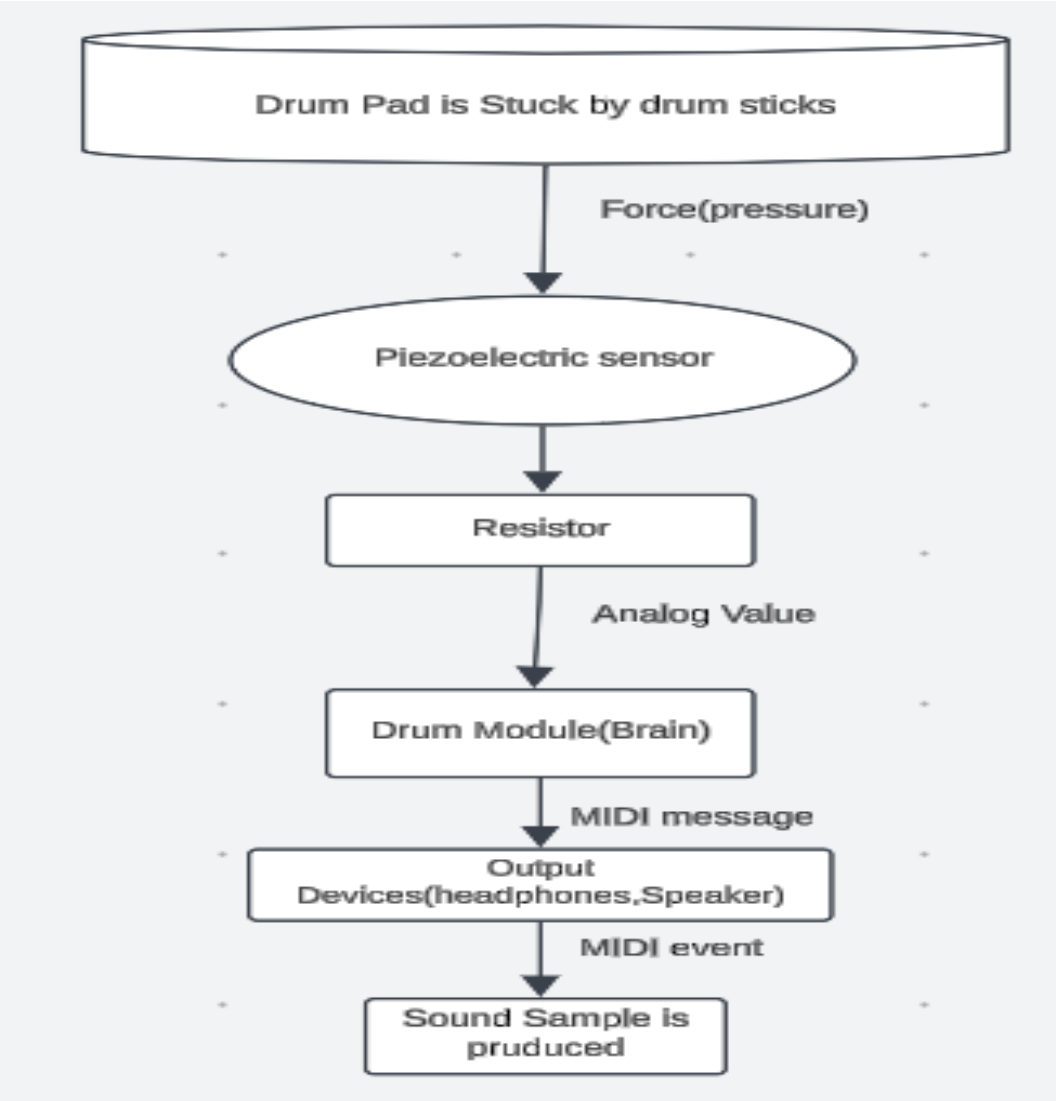


Figure 3.11 System Flow Design
Image Source: Author

3.1.7.2 Sensor Setup

There are many types of Drum Pads, with varying applicability depending on the usefulness of the different playing techniques. Hence, the number of piezo sensors will differ from the most basic pad to the ones that contain multiple zones. They can be broken down into the following types of electronic drum pads with their respective piezo sensor requirements:

3.1.7.3 Single-Zone Drum Pad

These are the most basic electronic pads, designed to identify strikes in just a single zone. These pads are destined for toms, kicks, and other drums that do not demand more than one striking zone. Requires 1 sensor to be placed under the Center of the pad to detect any strike on the pad surface

Examples: Such as kick drum pads and tom pads in basic kits.

3.1.7.4 Dual-Zone Drum Pad

Such pads are made to detect hits in two different areas, usually the drumhead (center) and the rim, which could serve well for snare drums.
Two sensors are required to be placed one: Head sensor placed under the center of the pad, and the Rim sensor placed at the edge to register Rim shots.

example:Dual-zone snare drum pads, dual-zone tom pads.

3.1.7.5 Triple-Zone Drum Pad

Such pads are made to detect hits from three separate zones; these pads can find application in more complex playing techniques, much like those on a ride cymbal. The sensors are usually placed under the center of the pad(the drumhead), rim, and either bell or edge.

The table below shows different types of drum pads that produce one to three different sounds, therefore, our group is going to follow this table to place the sensors in their designated location.

Pad Type	**Number of sensors**	**Pad placement**
Single-Zone bass drum/kick Drum)	1	Under the Center of the Pad
Dual zone Drum Stool/Throne	2	Center of the and Rim of the Pad
Triple Zone Hit-Hats	3	Center, Rim, and Bell of the Pad

Single-Zone Cymbal Pad	1	Center of the pad
Dual zone Cymbal Pad	2	Bow and Edge
Triple Zone Cymbal Pad	3	Bow,Edge, and Bell
High Hat Pad (Single Zone)	1	Bow
High Hat Pad (dual Zone)	2	Bow and Edge
Kick Drum Pads	1	One Zone
Percussion Multipads	1 per pad	Single zone

Table 3.1 Sensor set up for different types of drum pads

3.1.7.6 Sensor connection to the Control Board

A single zone pads will have one sensor, the positive lead will be connected to the analog input pin of the microcontroller and the negative will be connected to the ground on the MCU, a 1M ohm resistor will be placed in parallel to the piezo sensor as pull down resistor to dampen the voltage spikes and bring it down the safe levels to ensure a good read of the voltage produced by the sensor. The piezo sensor and the resistor can be soldered together to ensure a solid connection of the two, this will keep the analog voltage reading at zero volts until then until the piezo sensor sends a signal (voltage) basically preventing a false triggering. After adding a resistor we will connect the positive side to the microcontroller analog input, let say A0 and connect the negative side to the ground(GND).

We are also considering adding a diode like a Schottky diode to protect the microcontroller from a high voltage spike generated by the Piezo connected in parallel with the piezo sensor, the diode's cathode will be connected to the positive lead of the piezo sensor and the anode connected to the ground, if necessary we will add a small resistor of about 10K ohms connected in series with the piezo sensor to limit the current flowing into the analog input for to protect the microcontroller.

For the double and triple zone pads that contain more than one piezoelectric sensor we will follow the same way of connecting the single zone pad. A pad that has two sensors, each sensor will be wired to a different analog input in the control board.

A final circuit design of connecting the OmniDrum piezo sensors to the microcontroller using one diode and one resistor for the piezo sensor will look like the figure below adding more sensors.

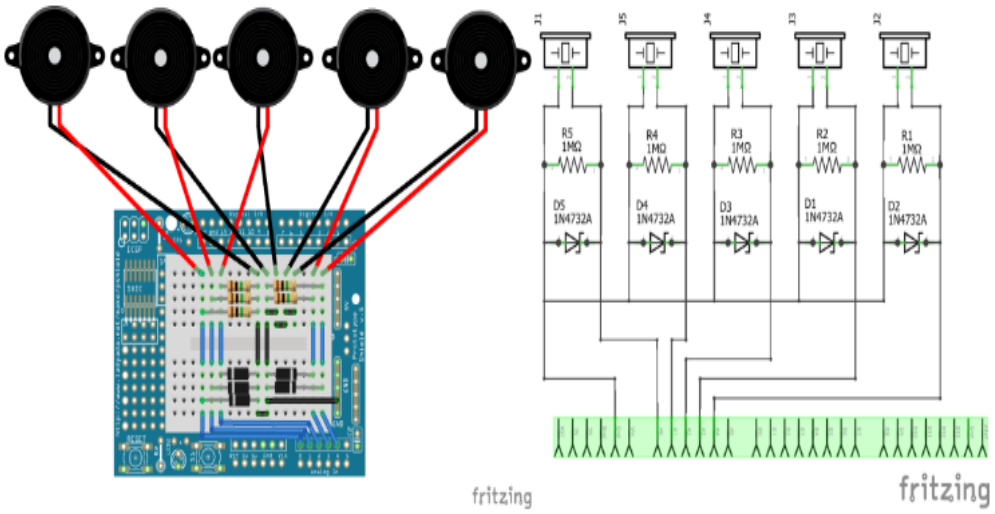


Figure 3.12 Circuit Design of Piezo Sensor to the Arduino MCU
Image Source:(Arduino Piezo MIDI Controller – Simple DIY Electronic Music Projects)

3.1.8 Power Supply

3.1.8.1 Power Specifications (Voltage and Current)

Our goal is to maintain the Omnidrum efficiency while keeping the total power consumption below 15 watts, to do that we need to focus on both the power supply and the design of the circuit

Our group will determine the voltage and current that are required for our OmniDrum system. In most of the electronic drum kits, the power requirements are usually between 9V and 12V DC, while the current supply will depend on the number of components (drum pads, sound module,displays,microcontroller, and amplifier). A calculation or estimation of the total current consumption for our system will need to be performed according to these components. The power consumption for Omnidrum could be broken down as follow

3.1.8.2 MSP430

The choice of a microcontroller will be based on our project objectives of efficiency along with the low power consumption also maintaining a low failure rate, for our OmniDrum we're considering two types of microcontrollers.

Because we used this microcontroller in the past classes like embedded systems and Junior Design 1 this might be a good choice for our project, but let's look at the power consumption of the MSP430, the operating input voltage of the MSP430 ranges from 3.6V to 15V.

The power requirements for each MSP430 family are listed below in Tables 3. The power given is based on the amount of current the core consumes per

megahertz (MHz). The Analog IMAX column indicates the amount of current added if the additional functional blocks are used.

Table 3. MSP430x2xx Family Power Requirements[1]

DEVICE FAMILY	PIN NAME	VOLTAGE (V)		CPU I_{MAX} [2] (µA/MHz)	ANALOG I_{MAX} (µA)	COMMENTS
		MIN	MAX			
F20xx	V_{CC}	1.8	3.6	370	Comp_A+ = 60 ADC10 = 1200, ADC10_I_{REF} = 400 SD16_A + I_{REF} = 1700 RefBuffer = 600	20x1: Comp_A+ 20x2: ADC10 20x3: SD16_A
F21x1	V_{CC}	1.8	3.6	410	Comp_A+ = 60	
F21x2	A_{VCC}, D_{VCC}	1.8	3.6	350	Comp_A+ = 60 ADC10 = 1200, I_{REF} = 400	
F22xx	A_{VCC}, D_{VCC} [3]	1.8	3.6	550	ADC12 = 1200, I_{REF} = 400 OA = 290	22x2: ADC10 22x4: ADC10, 2 OAs OA currents for a single amplifier
F23x0	A_{VCC}, D_{VCC} [3]	1.8	3.6	550	Comp_A + = 60	
F23x, 24x[1], 2410	A_{VCC}, D_{VCC} [3]	1.8	3.6	445	Comp_A + = 60, ADC12 = 1000, I_{REF} = 700	224x1: Comp_A+ 23x, 24x, 2410: Comp_A+, ADC12
F241x, 261x	A_{VCC}, D_{VCC} [3]	1.8	3.6	560	Comp_A + = 60, ADC12 = 1000, I_{REF} = 700 DAC12 = 1500	241x: Comp_A+, ADC12 261x: Comp_A+, ADC12, two DAC12s DAC12 outputs not loaded; DAC12 currents for a single DAC

Table 3.2 MSP460x2xx family power consumption

Image Source:
https://www.ti.com/lit/an/slva335c/slva335c.pdf?ts=1729720208179&ref_url=https%253A%252F%252Fwww.google.com%252F

Since the MSP430 requires only 3.3V input Voltage we will need to use a step down voltage regulator between the power source which could be an AC adapter or a battery and the MCU. The TPS62122 is a highly efficient solution (up to 96% efficiency) capable of driving75-mA loads. 11-µA quiescent current makes the TPS62122 an ideal choice in systems concerned over battery life.
The TPS62122 could be purchased for about 5 dollars, in addition to the low cost of this device, it provides additional protection to the MSP430 during the shutdown process. When VOUT has reached regulation (3.3V in this example), the TPS62120 signals the MSP430 through the PG (power good) pin. The TPS62120's SGND pin provides additional protection during the shutdown process. During shutdown mode, SGND provides a discharge path from the output capacitor to ground. This feature prevents lingering output voltages from discharging through the MSP430.

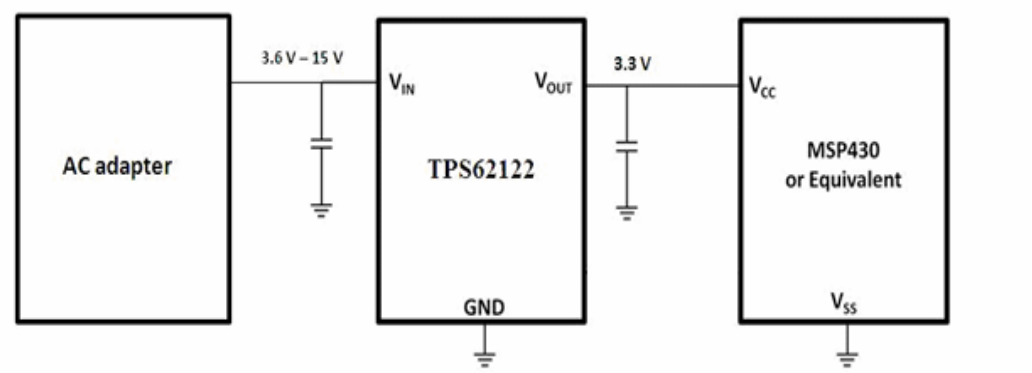


Figure 1. TPS62120 and MSP430 Simplified Block Diagram

Figure 3.13 TPS62120 and MSP430 Simplified Block Diagrams
Image Source:
https://www.ti.com/lit/an/slva335c/slva335c.pdf?ts=1729720208179&ref_url=https%253A%252F%252Fwww.google.com%252F

3.1.8.4 Linear and Low Dropout (LDO) Regulator with reverse current protection

The TPS709-Q1 low dropout voltage regulator has low voltage input ranging 2.7Volts to 30 volts which cost only about a dollar, it provides a regulated output voltage ranging 1.2volts to 5 volts. The LDO is capable of providing a steady output voltage without being impacted by the change in the input voltage which minimizes the noise making a good candidate for our design. Since the power consumption plays a big role in our project, using this device might be an issue for our design because of the heat dissipation due to the large difference between the input and the output voltages

The power dissipation of an LDO P(loss) is:

P(loss) = (Vin - Vout) Iout

Efficiency = Vout/Vin

This creates a reduction of efficiency by Vout/Vin
So, if our input voltage is 12V and output voltage which will be the voltage required by the MCU about 3.3V for the MSP430

Efficiency reduction = 3.3/12 = 0.275 = 27%

The excessive of the power dissipation could also damage the LDO
The LDO works better when the output Voltage is closer in value to the input Voltage or if the power consumption is not top priority for the design.

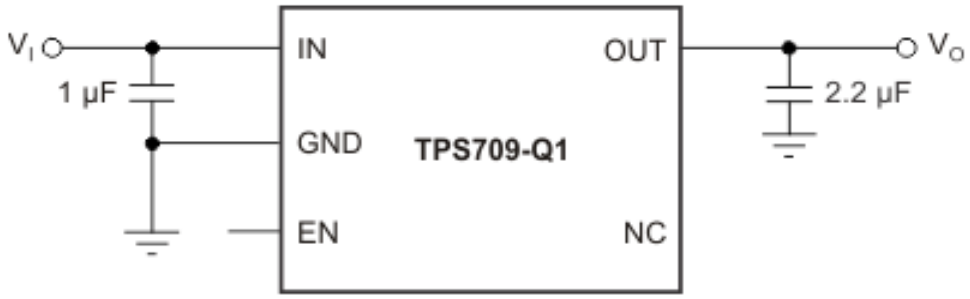


Figure 3.14 Low Dropout Voltage Regulator (LDO) TPS709-Q1
Image Source: [TPS709-Q1 data sheet, product information and support | TI.com]

3.1.8.5 Arduino

When it comes to choosing the right microcontroller, the size can play a big factor in reducing power consumption because each additional electrical device in the PCB consumes power to operate, therefore we need to look at the smaller one that fits our project. Other methods used to lower the power consumption involve reducing the clock speed and enabling the low power mode.
The table below shows the power consumption of different Arduino microcontrollers with different sizes using the three methods mentioned above.

Microcontroller	Reference 9V	Reduce Clock Speed 9V		Reduce Clock Speed and Operation Voltage 3.3V		Enable Low Power Mode 3.3V		Enable Low Power Mode 9V	
Result of measurement	Absolute	Absolute	Relative	Absolute	Relative	Absolute	Relative	Absolute	Relative
Arduino Nano	22.1 mA	18.5 mA	-16%	3.4 mA	-85%	3.4 mA	-84%	4.8 mA	-78%
Arduino Pro Mini 5V	14.6 mA	10.0 mA	-32%	3.7 mA	-75%	1.6 mA	-89%	3.2 mA	-78%
Arduino Pro Mini 3.3V	5.1 mA	3.8 mA	-25%	3.7 mA	-27%	1.6 mA	-69%	3.2 mA	-38%
Arduino Uno	98.4 mA	42.8 mA	-57%	11.6 mA	-88%	11.5 mA	-88%	27.9 mA	-72%
Arduino Mega	73.2 mA	61.8 mA	-16%	16.7 mA	-77%	11.9 mA	-84%	26.9 mA	-63%
Average Reduction			-29%		-70%		-83%		-66%

Table 3.3 Power Consumption of Different Arduino Microcontrollers
Image Source: [Guide to reduce the Arduino Power Consumption - DIYI0]

From these results we can see that the Arduino Pro mini is the most power efficient but the only problem with it is that it only has 6 analog inputs, so if we were to pick this for our project then we will have to add a multiplexer to expand the number of the analog input that will suit the number of the drum pads that we will use for the OmniDrum. One example MUX that can be added to the microcontroller isTexas Instruments CD74HC4067 16-Channel multiplexer which costs only $1.05.
Source (16 Channel Multiplexer - COM-00299 - SparkFun Electronics)

3.1.8.6 AC Power Adapter

A stable solution would be to use an AC to DC adapter, which converts the high voltage AC down to the level of DC voltage required by the OmniDrum. The AC adapter must have the same voltage and current specifications required for the OmniDrum kit with a clean supply to prevent electrical noise for a good sound quality.

3.1.8.7 Internal Power Regulation

Some components of the OmniDrum may require different voltages. In such cases, a voltage regulator circuit is needed. For example, where portions of a system (sensors or controllers) require a voltage of 5V while the main supply is at a voltage of 12V, a step-down voltage regulator can be used to convert voltage to the required level (a set ratio).

3.1.8.8 Protection and Safety

Fuses will be added in our power circuit to prevent excessive current from damaging the circuit in case of shorting or an overcurrent situation. The fuse will be just a little larger than our total expected current rating of the system. We will also use diodes in the power supply which can save some of the sensitive electronics from getting damaged due to reverse polarity.

3.1.8.9 Energy Efficiency

Low Power Components: If your products are battery-powered, then you would want to use components that consume less power to prolong battery life. Using efficient amplifiers or LED lights that don't suck too much power can minimize energy use. Sleep Modes: If your drum kit has a microcontroller, then putting your peripheral unit(s) into the sleep mode whenever it isn't used helps to minimize power consumption on the drum.

3.1.8.10 Existing Product Examples

Most commercial electric drum kits, like Roland and Yamaha, use 9V or 12V common external power adapters which have certain current ratings (usually 500mA-2A), which forms a yardstick of reference to design your own.

3.1.9 Software Languages

Which language best suits the needs of our project can be determined on many factors such as hardware choices, software support, granularity, or abstraction of the hardware from software. The C language can be simple and powerful for developing at the lower level, having control over memory and device hardware. Code can also be developed to release some of this control. Java can use the Java Virtual Machine to abstract this level of granularity in our code. With the JVM our program can manage memory efficiently and provide interfaces to the hardware in a more organized manner. Both are widely adopted languages and have contrasting benefits. The more modern features of Java may be applicable to some function of our project while taking away the control of the program. Using C can have better integration with the operating system we choose. Some portion of our program may be developed in C to account for what amount of control we wish to retain. The amount of support available from these languages can aid in reducing development time.

3.1.9.1 C

The C programming language is a fundamental tool for programming, offering a unique set of features that make it an excellent choice for special projects. Some explanations for why C is such a popular language include simplicity and low-level control of the program. C can be easy to use because of how simple it is. Other languages that include abstract concepts or complex syntax can take focus away from the goal of the program. An advantage of C is also the ability to provide direct access to hardware components which makes it an excellent candidate for low-level programming on embedded systems.

This language does not have a lot of object-oriented features included. This lack of object-oriented features would mean developing our own functions or using premade libraries for some solutions. This gives a greater control to the programmer as they develop low-level code. It is a popular language for embedded and semiconductor programming and an excellent choice for our hardware development. Due to how long C has been available, it lacks some of the modern features common for object-oriented programming. An alternative solution for object-oriented projects or inheritance from modern languages would need to be created from scratch or sourced from a number of premade libraries. This would be necessary to achieve a similar result to the modern languages available.

With how long C has existed there are many premade libraries available for developers to use. These libraries may already include the necessary code to perform simple to complex tasks which can save time for a developer. Writing our own algorithms or data structures from scratch can take a significant amount of

time particularly when a program is using multiple different variations of these types.

A key advantage is easy direct access to the hardware through C. This allows developers to adjust memory in different hardware that is attached to a C program. This language can provide direct access to registers, memory, and flags in an embedded microcontroller making it a great choice for developers. Code written in C requires compilation to machine code instructions. A separate computer would be needed to compile the code into an executable format before it is flashed onto the embedded microcontroller.

Developing code for our microcontroller with ArduinoIDE can allow us to use the C language. This IDE supports a wide range of microcontrollers and has a large user base contributing even more support to devices. Embedded devices can benefit from the libraries directly available from ArduinoIDE. Common tasks such as input and output may vary between microcontrollers and having a collection of premade libraries through this IDE will speed up development for our chosen hardware. A debugging tool is also included to support us during development. This debugging tool can assist with testing the program with break points where we may find values for our variables. ArduinoIDE includes many features which aid in swift development of our project.

The conclusions to be made are how the C programming language is excellent for direct hardware access or solutions for complex problems. Its simplicity makes it a popular choice for programmers. The key advantages of its simple syntax, direct access to hardware components, and abundance of premade libraries offers a unique set of strengths that make it an excellent choice for specific projects using embedded systems.

3.1.9.2 Java

The Java programming language stands out as a versatile tool which is widely adopted across industries. It is a language with strong focus on object-oriented and class-based code. Some of the many advantages of Java are portability of code, organization, external libraries and tools, and official documentation.

Perhaps the greatest advantage of Java is its portability. Unlike other languages being specific to certain platforms and devices, Java’s binary code can run on any device with a Java Virtual Machine. This means that Java programs can be easily installed on different environments. There is a partner component called the Java Development Kit which can be used during development and debugging. This kit includes a feature-rich set of tools for testing and finding bugs in a Java application.

The support for object-oriented programming helps to merge concepts of classes, objects, abstraction, polymorphism, encapsulation, and inheritance. Modern languages introduce many such concepts to developers to aid in creating organized code. These ideas can also produce greater resiliency to errors and protect from unauthorized access to data. There are built in security features

providing extra protection against a theoretical threat or vulnerability. Concepts of inheritance allow developers to reuse or recycle existing code in a way that can reduce the overall size of the code base. It facilitates the creation of organized code, making it easier to look through. In a large codebase with many components being added to the project the program can operate seamlessly when developed in an organized fashion.

Throughout the lifetime of Java there are many libraries, frameworks, and tools available. Adoption by industries has led to Java getting broad support. Industries that have chosen this language for their applications have an interest in finding bugs and improving the language overall. Many industries produce publicly available libraries to draw more interest in Java thereby expanding outside support of the language. It is not only enterprises that are interested in the language. Free source codes are available from countless FOSS projects which also share an interest in making the language all around better for everyone.

Java is an excellent choice for a versatile language that can handle complex projects. The portability alone is an attractive option. At scale this language can manage complicated applications efficiently. Even for small scale projects there is enough flexibility included to perform tasks with ease. A wide adoption through industries and devices, Java is positioned well as a leading programming language.

3.1.10 Sound Library

The drum kit will have resources for media playback which requires a selection of sound files saved on persistent storage. The persistent storage medium is in the form of an external flash storage device. Our circuit board must incorporate an external flash card reader for the system to mount and read this data. Software will be a limiting factor to the many formats of filesystems and audio files that will be supported. At the filesystem level the drum kit should follow a structured pattern to access all of these resources. Once the data is accessible to the device our data will be processed by the media codecs for playback. As with filesystems, there are many audio codecs that can be supported by the device. This support should be simplified to a select codec for quick implementation in the controller board software.

3.1.10.1 Directory Structure

Sound files will be saved in external flash storage. The directory structure for these files should not use spaces in the directory paths or file naming. Each instrument should map to the proper sound file on this device. Some devices may be mapped to the same sound profile such as two cymbal instruments or multiple snare drums. We will be implementing a directory structure such that a profile has a sound file mapped to each instrument installed. There may be multiple profiles

saved to flash storage to cycle through using the controller board buttons. The modules of our drum kit consist of four distinct instrument types:

1. Cymbal
2. Hi-hat
3. Snare
4. Base

For each of the four instrument types there will be several strike levels which can have their own sound file. Depending on how hard an instrument is hit the level of the strike will be determined by the sensor's output. Within each profile will be directories for each instrument which themselves will have the required sounds for each level of strike.

```
PROJECT/
├── PROFILE_1/
│   ├── BASE/
│   │   ├── 1_HIGH
│   │   ├── 2_MID
│   │   ├── 3_LOW
│   │   └── ...
│   ├── CYMBAL/
│   ├── HI-HAT/
│   ├── SNARE/
│   └── ...
├── PROFILE_2/
├── PROFILE_3/
└── ...
```

The PROJECT folder will be in the top directory location of the device meaning it is not within any other folder on the device. All profiles reside within this PROJECT folder. Only profile folders should exist in this directory. If there are folders that contain anything unrelated to the sound library here the folder may cause unforeseeable errors. The number of profiles that exist here should be no greater than 16 or 32 as the user cycling through greater amounts may have a poor experience. Naming convention of any custom profile folder should not need to adhere to any convention although it can be greatly helpful to include a number either at the beginning or the end to have it arranged in a specific order in the selection display. The numbers should be unique starting from 1 up to the maximum profiles that are chosen to implement.

Within each profile folder the default configuration can assign automatically to the instruments. Default configurations include instrument names BASE, CYMBAL, HI-HAT, and SNARE. Absence of any of these folders may leave the correlating instrument unassigned unless other configuration data can be found such as the profile.conf settings file. This configuration file will reside within their

respective profile's folder. The sound files within each instrument folder need to be numbered starting from 1. The number for sound files should be at the beginning followed by a label which can be anything. The standard number of files to each instrument should be three although as few as one file is necessary for operation. Greater numbers of files can be included and will equally divide strike levels amongst them. The loudest strike should always be numbered 1 and goes softer as the count increases 2, 3, 4, etc.

3.1.10.2 Filesystem

The format that can be most suitable for our integrated computer would be the FAT32 filesystem. Nearly every device can use this format so a user may change the contents stored on the external storage. One of the biggest restrictions to this format is the maximum partition size of 2TB at a sector size of 512 Bytes and file size of 4GB. These restrictions are well beyond the needs of our storage device requirements. Contents in this storage device will only be read from. Because we are using this as read only memory (ROM) the implementation of FAT32 support in software should be greatly simplified. Even though we are only reading from the device this does not mean that the device is only capable of reading through the software. Aside from these reasons, the legacy factor of FAT32 also provides a plethora of documentation and resources. Source code for other project implementations of this file format is also widely available online if we wish to use a free software solution. The free library FATFS can be imported to handle filesystem detection and data transmission. With this library our storage can be mounted, the directory structure parsed, and do file operations on FAT32 filesystems.

Extending support to other filesystems requires different implementations which may take focus away from developing the rest of the project. Personal computers today do not all agree on standard filesystem support such as proprietary ones by Microsoft or Apple. NTFS from Microsoft and APFS from Apple are the common formats that each company uses when formatting disks. These filesystems include more modern features for data storage. The features are advanced and complex methods of ensuring data integrity and security. Adopting support for either format is at the discretion of the developer as each company may be content with supporting their own. What is common, though, is support for the older legacy formats. FAT and exFAT still have wide adoption throughout devices from personal computers to smartphones and even more. Because we are not interested in modern features such as checksumming, encryption, or compression we may forgo implementing support for any more filesystems.

3.1.10.3 File Format

Performance of our project can be significantly impacted by the choices of which file format and codecs we choose to use. There are several hardware constraints that will impact the decision for using compressed audio over uncompressed audio formats. The time to read from the flash storage will greatly

increase if we choose to use uncompressed data while on the inverse side if a compressed codec requires too much processing from the integrated computer our project could behave poorly. Codecs that implement multiple compression level schemes introduce greater complexities to write software for.

The contents of any given sound file should consist of just near a second of audio to play the sound of an instrument. For uncompressed audio this duration would result in a small amount of data. With this in mind it is considered to use the lossless and uncompressed audio codec of wav. The relatively small sizes of our instrument sound files and minimal hardware requirements fit the constraints of our design. The drum kit hardware will have minimal compute cycles and I/O available to handle decompressing other formats. Software requirements to implement this wav format will be simplified for these same reasons.

Audio buffering can assist in lowering I/O to the external storage by caching files from the current selected profile closer to the processing cores. When a profile is active we can precache the relatively small sound files in memory and with uncompressed audio there should not be a larger memory footprint from caching both the compressed data and the decompressed audio. The wav format has minimal characteristics for how data is encoded. These are details such as the sample rate, channel count, and bit depth. A common configuration for wav files includes 44.1 kHz sample rate, 16-bit depth, and stereo channel audio. For a one second sound file in this format the file size is approximately 90kB. The operating system for the controller board should be able to implement this common case at a minimum. The software can include broad support to wav format as development continues or through the use of freely available libraries to have greater support of the format.

3.1.10.4 SPI (Serial Peripheral Interface)

The SPI protocol will be used for the communication between our external storage and microcontroller. The card will be made available to the system through several software setups such as initializing the card and placing it into SPI mode. Communication between the card and microcontroller is paired during initialization steps. Once the device is ready to operate it is necessary to locate the filesystem and partition table. Our external storage should be in the FAT32 filesystem and can be discovered through the FATFS library. Before the FATFS library can detect and mount our storage the software needs to perform the SPI communication setup.

Input and output of data across SPI can be achieved at a block level where a request command for 512 Bytes of data can be retrieved. The storage device will respond with the requested block or blocks of data. The commands to request blocks can be for single blocks using CMD17 or continuous blocks using CMD18. Using a library such as FATFS can manage filesystem discovery and traversal. With this library our read requests can be achieved with simple file lookups.

While the system is running and our storage is removed the software should unmount the filesystem. We want the filesystem to remount when the card is reinserted. This would require us to perform the SPI setup on insertion and have

a monitoring task to check whether the card is inserted or not. Our monitoring task will poll the SD card reader for a response and make our software aware of when there is no storage attached. The filesystem needs to be unmounted when removed and place our project into a standby state where no file operations are done.

The monitoring task can flag the system to standby without the storage attached and continue monitoring the SD card reader until it is reinserted. On reinsertion our card follows the setup of SPI again before being automatically handled by the FATFS library. With these steps in our software we can implement the sound library through SPI using the FAT32 filesystem and resume operation seamlessly if the storage is removed and reinserted while powered on.

3.1.11 Controller Board

This is the central component of the design which connects all peripherals together. Software interfaces to the sound library and instrument modules will be orchestrated by the system software running on this board. These inputs are mapped together by software such that an instrument may correspond to sound files. These instruments can be remapped for each profile as a user cycles through the available resources of the sound library. Most, if not all, of the assignment will be done automatically by the operating system. User intervention should be at a minimum when selecting an active profile. The operating system will manage output of both audio and MIDI data as well. All of these inputs and outputs must be handled with minimal latency.

3.1.11.1 Instrument Assignment

For each of the instrument modules there should be a dedicated port. The instrument assignment can be set statically in software to these input ports on the controller board. Statically assigning the instrument to these ports can help in simplifying coherence between the hardware layout and the operating system software. Implementing dynamic instrument assignment based on whichever module is plugged in to a given port can be challenging. There would need to be a way to identify between each module.

An alternative to having the operating system detect which module is connected would be to allow the user to have control over which input port has a particular module. Hardware requirements for this implementation would include a display that shows the configurations of each port, a button to select and cycle through these ports, and a button to cycle through one of the four instrument types to assign. Improper configuration could cause issues between the types of modules as the sensors for each module may have different calibration levels. The values that get read from one instrument module may be different from another instrument such as cymbals and snares. Matching each port correctly should be simple from a user perspective regardless of these decisions.

Not every port needs to be used. In the case that some ports are left unused the instrument assignment of these ports can still be set but they are nullified in

software. There should be no signals received from an unused port. Even when a port is unused we can implement software to not take any value until an instrument module is inserted. While the software is running a user may plug and unplug these modules. The operating system should be aware of which plugs are active and which are disabled. Allowing each port to still have an instrument assigned can give plug-and-play functionality. The operating system should detect when a port changes between unused and used.

```
PROJECT/
├─ PROFILE_1/
│   ├─ BASE/
│   ├─ CYMBAL/
│   ├─ HI-HAT/
│   ├─ SNARE/
│   ├─ ...
```

Expanding the number of instruments in a profile may be as simple as adding another directory in the profile. Although some modules can be assigned in a way that they share the same sound file, there can also be a way to assign more unique combinations. If a user wishes to change the sound of one of their snare modules they may do so using the selection controls on the controller board. The snare module can be assigned to one of the other available instruments sound files or an added instrument they placed in the profile. The display will give information of which port is selected and can cycle through the directory names inside the active profile.

3.1.11.2 Profile Selection

A standard profile with four distinct instruments with three levels of strike detection can be approximately 1MB in size if using one second wav formatted files. This could allow for many profiles to be loaded onto our external storage device although we should have the number of profiles that can be cycled limited to 16 unique profiles. The controller board will have a display and button controls for navigating the available selections of profiles. There should be a way to cycle all available directories within the PROFILE folder. This cycle should work with a forward and back button. Although some profiles may be customized to include greater or fewer instrument and sound files a user should have all instrument assignments made when switching profiles. This can be difficult for when a user creates a profile that deviates from the standard file structure as some custom sound levels and instruments may be present.

```
PROJECT/
├─ PROFILE_1/
├─ PROFILE_2/
├─ PROFILE_3/
```

```
├─ ...
```

In the case that a profile does not follow the standard file structure the operating system should keep a profile's settings when it is cycled back. A persistent method of allowing profiles to keep their assignment settings would be to include a profile.conf inside any given profile. Between power cycles this persistent file may load the custom profile in the desired configuration. The user experience is important when considering how customizable any given profile may be.

In the absence of such configurations, the operating system should make a best effort to assign a profile to a default configuration. A default configuration should be set to PROFILE_1 with the standard file structure. If any instrument folder is missing from the standard profile structure the system should not reassign automatically but rather leave the instrument unassigned for the user to choose.

3.1.11.3 Strike Level Selection

This project will implement sensors to detect between several levels of strikes. The standard configuration should distinguish three levels of a strike and a level for no strike. Each of the four instrument modules may implement different sensing technologies that send out different values for the same forces applied to them. Calibration of these sensors will provide us with the threshold levels to divide our sound files.

Our lowest threshold should be between the point of the lightest strike detected and the lowest sensor reading. This boundary will be our dampening value to differentiate the lowest strike between bumps and knocks. No audio should play from these bounds. Each sensor should be hardened to crosstalk and bumps through software. The highest threshold should be just below the sensor values for the hardest strike on an instrument. When a sensor reads beyond this threshold the sound file for the high strike should play.

```
PROJECT/
├─ PROFILE_1/
│  ├─ BASE/
│  │  ├─ 1_HIGH
│  │  ├─ 2_MID
│  │  ├─ 3_LOW
│  │  ├─ ...
```

With both of these thresholds determined we can assign a number of strike levels that are available for the assigned instrument. Profile customization can add or reduce the number of strike levels present in their directory. In the case of only one strike level present we may use the lowest threshold as the value our sensor can read to play that sound. If two levels are present we may use these two existing thresholds. As more levels of strike detection are present the system may evenly

split between these two boundaries as it adds more levels of strike detection. The limitation for how many divisions can be done would be determined by the range of values a sensor may provide.

The operating system should be able to track how many levels of strike an instrument includes by the number of files in that instrument's directory. The file naming structure would need to follow standard conventions for the correct strike level to be used.

3.1.11.4 Operating System

Our integrated computer hardware will be a limiting factor for the operating system. With minimal hardware specifications the software that runs on the controller board should be minimal and light on compute cycles. Other requirements for the software include responsiveness and highly parallelized communication for input and output streams. This system will be monitoring the array of input ports for sensor data, reading data from external storage, reading audio codecs, multiplexing audio playback, and managing output data streams to midi and speakers.

The drum kit will need to have operations done in real time or give the behavior of real time responsiveness. Creating a real-time operating system from scratch can be challenging. Some freely available operating systems that can be used for the base of our system include FreeRTOS. Support for this lightweight embedded operating system is present across many popular microcontrollers. Using this for the base of our software will provide more focus on the custom software of our project.

At startup our software will boot into the default profile, PROFILE_1. Precaching can then bring sound files from external storage into RAM. The software will identify the directory structure of the active profile and bind instruments to the relevant inputs. Scanning through a profile should have case handling for any deviations from the standard structure, any added or missing files or folders for instance.

The input signals will be individually buffered to memory where their values can be used for branching instructions. The operating system scheduler should give a higher priority to the task which monitors these buffer values. Because we are buffering the sensor readings we can retain the values of multiple sensors and read back all buffers at once. This scheduled task is considered to have a soft time deadline. In the event of a soft deadline passing for a real time operating system the value may not be wasted but at a degradation of performance. If the task were to miss the target deadline to read the buffer we may return back to this task and take the value this buffer still contains. Buffers should be overwritten when their contents are read from this task or if a new input value is detected from the sensor.

Multiple instruments must be able to play over each other. When a number of instruments are struck in series the sounds should layer over each other. To enable continuous playback for each sound file we consider a multiplexer configured for stereo channel output. All source files should be downmixed to stereo. Audio reencoding may be implemented to transform audio with more than

two channels before sending it to the multiplexer. If a source is mono audio then the single channel should be sent to both channels to retain consistent playback from both sides. As more sounds are directed to the multiplexer their signals will be added over each other to give this layered audio effect.

Even though FreeRTOS is a complete operating system there are still edge cases that can cause errors for our project. Error handling should be dealt with gracefully while the system is still running. During software development we will use a detailed software flow chart explaining each case and branching direction as the software runs. A completed state machine diagram will help to minimize edge cases and reduce errors while the drum kit is running.

3.1.11.5 Control Panel

The user control over the device will be provided with a set of buttons and a display module that shows relevant information. The menu will consist of nested options for users to navigate through and select the configuration they desire. The most significant function for the control panel is the cycling through the sound libraries included profiles. As a user cycles through the available profiles the operating system will handle some autonomous control. The operating system can handle some assignments of a profile to the instrument modules. A user may wish to reassign or otherwise customize how the active profile is set up. In this case a user may navigate this control panel through options such as which instrument is assigned to an input.

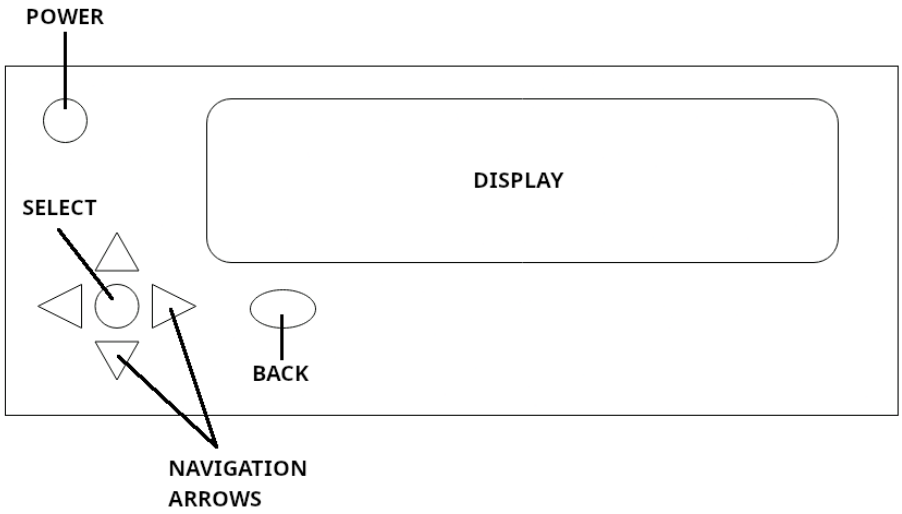
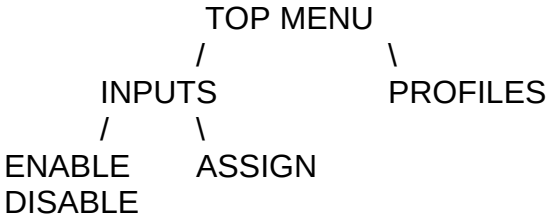


Figure 3.15 Control Interface and Display
By Jameson Palmer

The menu flow accounts for several submenus. As the user navigates further into any submenu the hierarchy of this navigation is designed with the given layers, or levels, below. Using the ‘BACK’ button will navigate this hierarchy up

levels. The top level is actually displaying both the inputs and profile that is currently active.



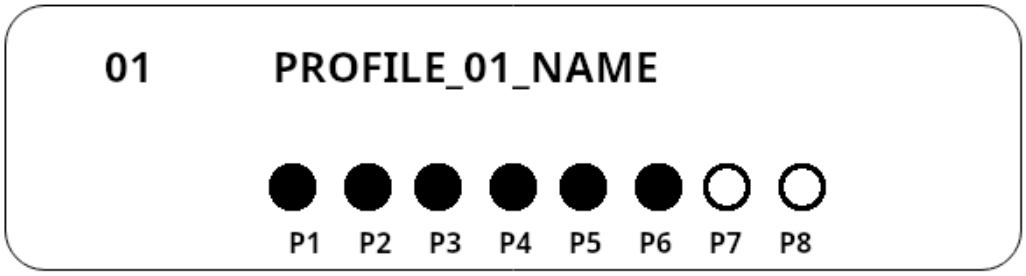


Figure 3.16 Top Menu Display
By Jameson Palmer

As the operating system pairs the sound library with instrument modules the status of these components will display along the screen. A status monitor is included to give a visual representation of the device configuration at a glance. A simplified approach to what information is displayed can help with ease of use and intuitive control. The display will show the input ports for our instruments and show which of these inputs are currently active. The arrow navigation buttons can highlight any one of these ports for additional options such as disabling an active port, reenabling a port, and changing which instrument a port is. For a port that does not have a wired connection the status should remain disabled.

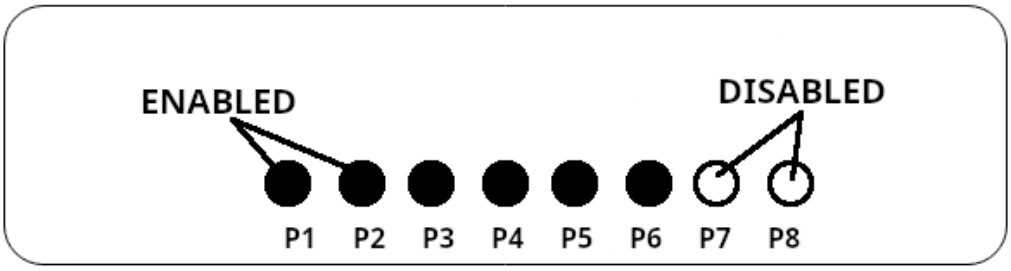


Figure 3.17 Input Port Status Monitor
By Jameson Palmer

Part of the customizability of our sound library allowed for including additional directories to represent more instruments. The standardized approach for a profile would have multiple instances of the same instrument to share the

same copy of its sound files. Take the case of our kit having two snare drums connected. Our inputs would be assigned as snare drums and, because of this, use the snare drum sounds from the library. A custom profile may assign one or both of these instruments to use a different instrument in a profile. Custom instruments should be given appropriate folder names in the sound library as this is what is shown on screen when choosing a custom instrument. On the screen we navigate to any of our input ports and enter a submenu by using the ‘SELECT’ button. A list of available instruments are now shown for the user to navigate. Hovering on an instrument and choosing ‘SELECT’ will reassign that input port. Choosing the ‘BACK’ navigation button closes the submenu.

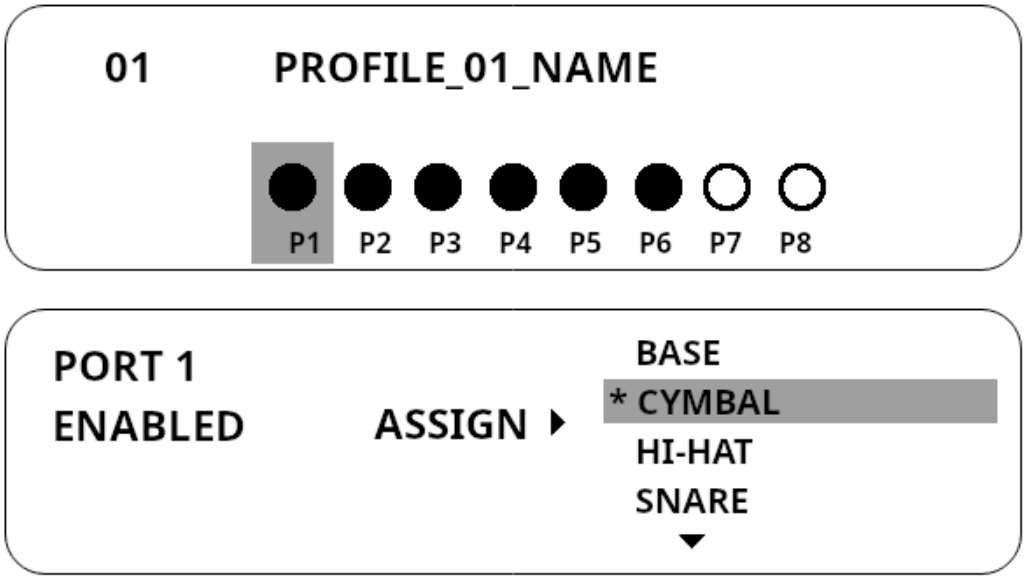


Figure 3.18 Input Port Assignment Submenu
By Jameson Palmer

Profile navigation is displayed on the second half of the screen. The folder name will display for the active profile and using the arrow navigation buttons can cycle through the included profiles in the sound library. Navigation will wrap around for the first and last profiles. Any number of customized profiles may be included into the sound library which may cause navigation to suffer as the number of profiles grows. An artificial restriction of 16 to 32 profiles is determined to round off edge cases and simplify our software implementation.

3.1.11.6 MIDI Data Output

We follow the common conventions of the MIDI code assignments when developing the software of our kit. The few instruments our design focuses around can be paired to these MIDI codes. The customizability of a profile brings a bigger challenge when developing for how these codes can be paired. A profile can include additional instruments that a user can assign. How does our software

assign MIDI codes to such instruments? From a software perspective we may implement expanded support to the common MIDI drum values. A folder name would need to match any of the names of these expanded instruments to function this way.

MIDI codes can be changed on the users recording software. As long as we can assign distinct code values to any instrument we can differentiate that device. The idea that each device can be assigned a unique MIDI value allows the software to handle customized profiles more gracefully. Our supported codes can consist of a selection of common drum values and a set of reserved values to be used when no automatic pairing can be made. There should be a reserve code available for any of the plugged in modules in the most extreme case that no automatic pairing is made.

It should be noted that when a profile is selected it not only assigns the instrument from the sound library but also assigns the MIDI code. When the drum kit plays the instrument it plays both the sound and the MIDI data. This may occur even if the MIDI cable is not used. This can help simplify the running software as it doesn't need to monitor whether a cable is connected or not. This can grant the behavior of plug and play functionality without increasing the sophistication of software. We can be sure that MIDI data is transmitting when a cable is connected because this method will always be sending data.

Some instruments may be modified with foot pedals which are also part of the input array. A foot pedal modifier should not play until a strike on the related instrument is detected. When a strike on the instrument is made this foot pedal modifier will be checked before branching further. The flow of software can determine whether to use the sound file and MIDI code of the instrument with or without the modifier applied. Sound files and MIDI codes are applied to the foot pedal.

MIDI notes should have a duration relative to the duration of associated sound files. When parsing data in the media library the files contain metadata information on how long the file plays for. This can be retrieved and used as a timer for how long a MIDI note is played. This metadata will be collected and stored in a structure when the profile is selected. By keeping track of the current profile content and metadata in memory the software can access and keep track of the filesystem without having to retrieve information from storage. This metadata is not going to change while the device is running and storage is mounted so the information should be retrieved once per profile.

Determine how to play simultaneous MIDI notes like concurrent audio.

3.2 Part Selection

3.2.1 MCU

The first thing that we need to decide on in order to begin forming our circuit is the microcontroller that we would like to use in our project. There are many different options each with different onboard peripherals that will be necessary for

us to implement the functionalities that we would like to. After further research online, some possible options were found and are in the table below.

Model	STM32F405RGT6W LQFP64	STM32L562QEI6Q LQFP64	STM32G473R LQFP64
GPIO Pins	51	52	52
12-bit ADC Channels	16	16	26
External DACs	2	2	3
UART/USART/LPUART	2/4/0	2/3/1	2/3/1
SPI	3	8	4
Memory	1024 MB	512 KB	512 KB
Price	$11.83	$6.94	$8.20

Table 3.1 Comparison of MCUs

The LQFP64 package size was chosen for all variants because it essentially guarantees that there will be enough GPIO pins. A concern with the GPIO pins is that some core features may overlap. For example, in the STM32G473R, one of the SPI pins overlaps with a DAC output pin. This would mean we cannot use SPI1 for the microSD card; luckily there is another pin with that same function on the LQFP64 package size which means it would still work for our purposes. This is a reason that choosing a larger package size would be beneficial. However, larger package sizes cost more and take up more space on the PCB. The LQFP64 package size provides a middle ground between these options.

Another benefit to using the LQFP64 package size is that the number of GPIO pins is less of a factor in choosing which chip to use. In the above chips, there is only one where the GPIO pin count is one less than the others. This means that the GPIO Pins section of the table has a minimal effect on the choice of MCU.

The number of ADCs available on the chip has a significant effect on which chip is chosen. ADCs will be used to measure values from our sensors in order to tell how hard they were hit and convert that into a value usable by the MCU. We would only need about 6 ADCs, but as was discussed earlier, pins can have overlapping capabilities so more is better simply to be sure that there will be available pins. For this category the STM32G473R beats the other two by having 10 more ADC channels available to be used.

Another thing that is very important to have on our MCU is a good number of external DAC channels. Many MCUs will have some number of both internal and external DACs. For our purposes we specifically need to consider external

DACs. It should be noted that the terms DAC and DAC channels are being used interchangeably; this is not true in the real world but makes sense for the purpose of our discussion. In reality many of these MCUs have one DAC with 2 DAC channels, functionally for us that means two separate output pins with independent audio channels. It is also important that these external DAC channels are buffered. Buffering the channel allows it to be easily connected to an output such as an amplifier without having to worry as much about pulling too much current or crosstalk between connected channels in the case of combining signals. All of the DAC channels being discussed in this section are buffered. Both the STM32F405RGT6W and the STM32L562QEI6Q have 2 external DAC channels while the STM32G473R has 3 external DAC channels. While two channels would help in creating polyphony, it would not be as good as having three channels. With two channels the likelihood of different sounds cutting each other off is much higher than it would be in the case of three channels, specifically in the case of a drum kit. In this case the STM32G473R wins against the other two choices.

The next item in the table is UART/USART/LPUART count. In order to transfer MIDI information we need to utilize UART. USART in asynchronous mode can also be used functionally as a UART channel. Since only one channel is required here it is very easy to accomplish this goal. Each of the three microcontroller options has the ability to accomplish this, so this category also becomes somewhat irrelevant in our decision.

The next category in the table is the amount of flash memory available on the MCU. It should be noted here that the sound files are going to be stored on a microSD card so their effect here is not as relevant, however we may still need to buffer our drum sounds. After some testing it has been found that a full drum profile without different sound files for each of the velocity levels could easily be stored within 512 KB. Regardless, the STM32F405RGT6W wins this category by having double the available flash memory as the other two.

The last category in the table is the price. The price difference between the choices is present although not too extreme. That being said the STM32F405RGT6W is quite higher than the other options which means it will need to be exceptionally more useful to be chosen. The STM32G473R, despite costing more than the STM32L562QEI6Q, does provide another DAC output which will help when implementing the code. Due to the fact that the exact method of implementation is not known yet it is best to choose an MCU with at least 3 DACs. There may be a way to combine the sound files digitally in real time and then only need a single DAC channel, however other methods of implementation may need to play the sounds on separate independent DAC channels. Seeing as that both the STM32G473R and the STM32L562QEI6Q share the same amount of memory, but the STM32G473R has an extra DAC channel. It seems worth the slight price increase to use it over the others. It is a good middle ground between both memory and price as well as an improvement in both ADC and DAC capability over the other two; this makes it the optimal choice for our project.

3.2.2 Audio Output Device

We need to decide on which audio output device to use with our device in order to give the user auditory feedback when playing. The options are shown in the table below.

Model	HY7040DS0830-H16.8	HYB30704R3W	SC700208-1
Price	$1.75	$2.00	$4.10
Rated Power	3W	3W	3W
Rated Impedance	8Ω	4Ω	8Ω
Output SPL	93dB ±3dB	116dB ±3dB	100
Total Harmonic Distortion	10%	5%	NA
Dimensions	71mm x 41mm x 17mm	70mm x 31mm x 17mm	71mm x 41mm x 25mm

Table 3.2 Comparison of Audio Output Devices

The three options for an audio output device given here are all loudspeakers. Piezo buzzers do not make good candidates because of their lack of audio quality and so only electrodynamic transducer speakers were considered. The rated power of the speakers is 3 watts; this value is appropriate for producing the volume of sound that we want. The loudness is about enough to fill a room while not requiring too much current.

The next thing that we took into consideration was the impedance of the speakers. Speakers with 8Ω impedance usually have better audio quality than 4Ω speakers however they require more current to be driven properly. Since we are using 3 watt speakers power isn't the biggest concern so we are able to make the tradeoff for improved audio quality.

All of the speakers listed have similar dimensions so regardless of our choice it won't make a difference. The output SPL of all of the speakers is also as loud as we would need it to be. Anything over 80 to 85 dB SPL is plenty loud enough to fill an entire room. Looking at the price, the HY7040DS0830-H16.8 is less than half the price of the SC700208-1. The HY7040DS0830-H16.8 also has a desirable frequency response curve. On the contrary, the SC700208-1 does not have any frequency response curve shown on its datasheet nor does it have total harmonic distortion listed. This makes it an unreliable choice as there is no information supporting that it can produce good sound.

Given this information it seems like the best option to choose is the HY7040DS0830-H16.8. With its cheap price it brings a good frequency response as well as an overall low power requirement and will get sufficiently loud enough.

3.2.3 Amplifier

Now that we have decided on a speaker, we need to choose the design for an amplifier based around that choice. Our chosen speaker has a nominal power rating of 3 watts and an impedance of 8Ω. Because of this we will need to choose an amplifier circuit that provides 3 watts to an 8Ω load. Luckily, many designs for this type of circuit can be found online with a parts list.

Model	TDA2003	PAM8403	LM386
Amplifier Class	AB	D	AB
Output Power	10W @ 4Ω	1.8W @ 8Ω	0.25W - 0.7W @ 8Ω
Supply Voltage	8V - 18V	2.5V - 5.5V	5V - 18V
Impedance Rating	1Ω - 8Ω	4Ω - 8Ω	4Ω - 32Ω
Lifecycle	Obsolete	Active	Active

Table 3.3 Comparison of Audio Speakers

The amplifier ICs above are all commonly used on amplifier circuits. They all have the ability to drive our loudspeaker but there are differences between them that have an effect on our circuit. The Amplifier Class category refers to the type of circuit being used to amplify the signal. Class AB amplifiers have a conduction angle that is between 180° and 360°. These amplifiers are very common and inexpensive but tend to be inefficient as well. Class D amplifiers use pulse width modulation and the conduction angle no longer plays such a large factor in the amplification process. Class D amplifiers tend to be more expensive but are much more efficient and produce less heat. For a low power application like this, it is smarter for us to use a Class D amplifier, the added expense is not too much at the scale of a single IC and would keep us within our power constraints while also not requiring a large heatsink. This leans towards choosing the PAM8403.

The output power of the amplifier has a somewhat related effect on how loud the amplifier can get; though these are not directly related, thinking about it this way is practical in our situation. The output power of the TDA2003 is 10W at 4Ω, so we can conclude it would be around 5W for an 8Ω load. This would exceed the power rating of our speaker and cause unsafe conditions. Though the signal can be driven lower than 3W, it risks the possibility of being turned up too high and causing a fire. The PAM8403 can drive 1.8W at 8Ω when supplied with 5V according to the datasheet. Though this is not the maximum amount our speaker is rated for, it would still be enough to drive the speaker to make a significant amount of noise. The amplifier power rating being lower than the speaker power rating is okay as it cannot overload the circuit and cause any danger. The LM386

has an output power of 250mW to 700mW at 8Ω depending on the input voltage. Though this would drive our speaker, the inefficiency of a Class AB amplifier limits its performance and would probably be too quiet to be acceptable in our application.

The input voltage of the TDA2003 is higher than the required supply voltage of the other two options and would still output too much power to our speaker which makes it undesirable at this point. The PAM8403 and the LM386 both can be powered with voltages near the supply voltage for our microcontroller. This means we could have an overall input voltage that is correct for our amplifier and then use an LDO to power the microcontroller. That makes it a win for both of them in this category.

The PAM8403 and LM386 can also both drive the 8Ω impedance of the speaker we have chosen. This makes them equally capable in this category as well. However, factoring in the output power and amplifier class, it seems likely that the PAM8403 is the best choice.

Lastly the lifecycle of the TDA2003 is obsolete meaning that it is no longer being produced. There are listings online for sale but it is possible that these could be counterfeit or damaged. It is not worth risking which makes the PAM8403 and LM386 both win in this category as well.

Given everything we have looked at, it makes sense to choose the PAM8403. Its higher efficiency gives it more output power than the other amplifiers with a low supply voltage. It is the closest choice to matching the wattage of our speaker without exceeding it. It is also readily available from many vendors.

3.2.4 Auxiliary Output Jack

In order to properly create an auxiliary output jack, we need to construct a headphone amplifier circuit. These circuits have different characteristics than the loudspeaker amplifier due to the fact that headphones have a much higher impedance than loudspeakers. This also means that we will need to choose a different chip for amplification. Below are some options available to us:

Model	LM386	TS482	LM4915
Amplifier Class	AB	AB	AB
Output Power	325mW @ 6V 8Ω	67.5mW @ 32Ω	250mW @ 32Ω
Supply Voltage	5V - 18V	2V - 5.5V	2.2V - 5.5V
Impedance Rating	4Ω - 32Ω	16Ω - 32Ω	32Ω
Lifecycle	Active	Active	Active

Table 3.4 Comparison of Output Jacks

The amplifier chips above are all possible candidates for the headphone amplifier. All of the amplifier chips listed are Class AB which means that the amplifier class is irrelevant here as it is the same regardless of what is chosen. The lifecycle is also Active for all options here as well, so there are no worries about acquisition.

This leaves output power, supply voltage, and impedance rating. Beginning with impedance rating, the ideal impedance we are assuming is 32Ω. The amplifier options are all able to drive this impedance. The designs of these amplifiers vary somewhat; the TS482 has the lowest output power of the amplifiers at 32Ω. The LM4915 is listed as driving 250mW at 32Ω whereas with some math we can conclude that the LM386 would provide around 80mW at 32Ω. The difference between these two options is in the circuit design. Although the LM4915 has a better output power, the LM386 has a far simpler circuit design. It only requires a couple components to function correctly. The amplifier for the headphones does not need to be as loud as the loudspeaker and as long as it is audible to the user it should be fine. Because of this, it would be optimal to use the LM386 for its simplicity.

3.2.5 Sensor

Here we need to decide which type of sensor we want to use in our implementation. These sensors have different ways of transmitting their information to the MCU, some of which require certain minor circuits to function correctly

Model	Force	Velocity	Vibration
Pins	2	4	3
Variance Type	Resistive	I2C	Digital
Price	$3.50	$9.97	$6.49/5pk
Required Connection	Analog IO Pin	I2C	Digital IO Pin

Table 3.5 Comparison of Sensors

The sensors listed above are all very different from each other in the ways that they transmit the information received. The velocity sensor in particular may be complicated to implement. The velocity sensor connects to the microcontroller through I2C. This is a bit more complicated to implement in our project due to the need for 6 independent inputs. Also, velocity may not be the best variable to

measure; the drum pads will not be moving a ton when someone is playing despite receiving force and vibration.

This leaves us with the force sensors and the vibration sensors. The vibration sensor may be useful to us because we can expect some vibration when a drum pad is struck. The issue here is that it only outputs a digital high or low. This may be a problem depending on the sensitivity of the sensor and could cause false positives. This also does not provide detail about how hard the sensor was struck which would not allow us to implement velocity sensitivity.

On the other hand, the force sensors provide an analog output as they simply decrease in resistance as pressure is applied. This would be easy to implement given the amount of ADC channels we have on the MCU. It would also be very simple to implement as the sensor only has two pins. Measuring force is probably the best way to receive input as it provides a direct measurement of how hard a drum pad is being struck, and is less susceptible to false positives. It should be noted that the MCU cannot measure the change in resistance, so a voltage divider will need to be created to implement a fluctuating voltage to be applied to the ADC pins. This will be discussed in a later section.

3.2.6 Power Supply

For this section, we need to decide on a way to provide power to our entire system. This requires us to choose an initial input voltage and then also decide how to convert it to the other voltages we may need in our circuit. Looking back at our circuit it is apparent that we need both 5V DC and 3.3V DC to supply different components. We also need to be aware of the amount of current we will be drawing. Looking at the datasheet for our chosen microcontroller we can see that the max current into the MCU is around 150mA, amplifiers can be estimated to need 500mA for the loudspeaker; the headphone amplifier could be estimated to use a much lower amount of current, for this we will say 100mA. That is probably far more than what we would need to drive it but this gives us a safety net as well. The force sensors will use negligible current and all of the indicator LEDs can be estimated to use 100mA altogether. MIDI output is usually only a couple mA, and a microSD card can pull up to somewhere around 160mA. This comes out to around 1A at 5V.

It is smartest for us to supply a main voltage equivalent to our highest required voltage. In this case that would be 5V. This can be supplied by an off-the-shelf AC DC converter. To be safe we can buy one that provides 1500mA. The only other voltage value needed is 3.3V. We can use an LDO to bring our 5V down to 3.3V without too much trouble. For this we will need to select and LDO.

Model	BA33DD0WT	LM3940	LT1085
Input Voltage Range	3V - 25V	4.5V - 5.5V	2.55V - 30V

Output Current	2A	1A	3A
Price	$3.12	$1.62	$10.13

Table 3.6 Comparison of Power Supplies

This simple table helps us choose which LDO we need for our application. Since the LDO will be outputting 3.3V, the current output is only relevant to the parts of our circuit drawing from the 3.3V line. This excludes the amplifier circuits which makes the required current output much lower than the overall circuit. We should not need over 1A for this part to function correctly. We will also be receiving input from a stabilized 5V DC source. This means a wide range of input voltages is not that useful to us because our voltage is already at one steady voltage. This makes the LM3940 look very appealing because of its low cost. It should be able to do everything we would need it to do in terms of driving the 3.3V line.

3.2.7 Development Environment: Languages and Repositories

The MCU of choice is the STM32 family microcontroller from STMicro. The manufacturer provides tools for developing with this controller on their website which includes an IDE and related testing software. STM32cubeMX is a companion software to the STM32cubeIDE which gives a graphical control of the components in the device. Setting up the pins of our controller can be done through this software which can help with connecting our peripherals. Initialization for our system can be generated into C code using this software. The initial configuration for embedded devices are often different due to hardware design and can be tricky to develop the proper setup procedure for different platforms. This provided software takes much of the complexity away with starting our development.

Although there is a provided IDE for our microcontroller we may also develop with VSCode or VSCodium, the open sourced variation of VSCode. Extensions may be installed to this program which contain the necessary resources to develop on the STM32 platform. Installing ‘stm32-for-vscode’ gives the needed functions to debug and compile code written for this family of microcontrollers. This extension may also work alongside the first party STM32cubeMX software. Code is expected to be developed in C language although C++ may also be supported.

The project behavior may have some complexity but overall each component in software can be simple enough to stick with C overall. This decision leaves most granularity of control with the programmer and unifies all aspects of the design into one language. Most of the low level code may be sourced from readily available libraries. Technologies that have existed for decades such as mounting and reading filesystems do not need to be made from scratch and are greatly discouraged in general. The C language can produce code that takes fewer cycles on the cores which can benefit the responsive feeling of the project. An emphasis on security is not necessary for this device as our use case does not

involve personal or sensitive data in any way. Security is not an inherent aspect of the C language like it is with many modern languages today.

Hosting the project code throughout the duration of its development is important for all members to view the software's development. It is important to have a history of the project with tools such as 'diff' for easily visualizing changes to the source code. The repository should be secure in that only members of the project team may view, add, delete, or download these sources while it is developed. A choice of remote hosting is GitHub for how accessible it is to all members. It includes all of the important features our project needs and is historically the one most used by our team.

3.2.8 Filesystem

Software for this project is intended to perform limited file i/o operations on an external storage device. The intent is for a user to be able to load their storage on a personal computer to customize audio files for the device. A format that is widely adopted across popular operating systems should be chosen. This is the greatest restriction to our choice of filesystem. Other formats may include more advanced features although even the simplest option we are comparing with, FAT32, contains enough features for the purpose of the device. There are features of operating systems that can detect or prevent data corruption, reduce data size, or have greater file and partition capacity.

File sizes are not a restriction in our choice as we are expecting a common profile with all its content to be in the range of megabytes. The more modern formats we are comparing can have a single file as large as hundreds of gigabytes to hundreds of terabytes. The restriction of 4GB is more than enough from the FAT32 filesystem. Sparse files that contain long segments of 'empty' data are able to be reduced by modern filesystems and returning more of the flash storage back to the user. This feature could only slightly improve our device although common file sizes would range to a few hundred kilobytes in size.

Data corruption is a serious concern for any device that is meant to write and store exact data. Journaling is meant to help with recovering a device from corrupted states. The filesystem logs its contents over time to revert to an uncorrupted state. External storage devices that get removed too early is the most common cause of corruption. Without a method of dealing with corruption we must develop our own solutions for when the device receives bad data.

Case sensitivity involves whether a filesystem may handle uppercase and lowercase paths and files as distinct. Without case sensitivity we can force each directory and file to be made distinct. No path or file within may share the same name but with different capitalization. This benefits our software to not have case sensitivity as it can reduce confusion that capitalization may introduce. File and folder structure is meant to conform to a specific layout in order to be properly discovered by the device. Even though NTFS can have case sensitivity it is common for the Windows operating system to enforce naming that emulates no case sensitivity.

Format	Case Sensitive	Sparse Files	Journaling	OS Cross Compatible
FAT32	No	No	No	Yes
exFAT	No	No	Optional	Yes
NTFS	Yes	Yes	Yes	Yes*
APFS	Yes	Yes	Optional	No**

* Not all platforms support writing.
** Not all platforms support reading.
Table 3.7 Comparison of Filesystems

3.2.9 Audio Format

Audio codecs can impact data transfer time, memory usage and CPU cycles. Latency is introduced with codecs that involve additional processing such as compression. Before a codec can produce audio some formats require a range of data to look ahead before it can be decompressed. The transfer speed from our storage device can have a negative impact on responsiveness as file size increases. With higher compression the responsiveness is negatively affected by CPU load. Larger file sizes will also take a larger footprint in memory although codecs that reduce file size with compression will need additional space in memory for the decompressed data. Tradeoff for memory footprint is determined to be negligible because of this. The chosen audio formats to provide compatibility for this project are WAV for its low CPU overhead followed by Opus for a balance between CPU and data resources.

Format	Compression	CPU Processing	Size

WAV	No	Low	High
FLAC	Lossless	High	Medium
ALAC	Lossless	High	Medium
MP3	Lossy	High	Low
AAC	Lossy	High	Low
Opus	Lossy	Medium	Low

Table 3.8 Comparison of Audio Formats

4. Design Constraints & Standards

4.1 Industrial Standards

4.1.1 PCB Design Standards

To produce our PCB we will be using the manufacturer JLCPCB. They were chosen due to familiarity with the service and past reliability. JLCPCB has certain guidelines that must be followed when creating a PCB design. These include using 1 - 2 oz copper for outer layers of the board, as well as a minimum trace width of 0.127mm. Other requirements are that tracks must be 0.127mm apart from each other and 0.3mm away from the edge of the board. Hole sizes must be within 0.3mm and 3.6mm and vias must be at least 0.3mm with 0.5mm pads.

Other than just the PCB guidelines of JLCPCB, there are other resources to consider when designing a PCB. IPC standards are relevant when designing PCBs to ensure a common understanding of their design and to make clear certain PCB design practices. We can reference IPC-2221 which has standards for

generic PCB designs as well as reference to specific documents that discuss certain types of printed circuit structures in depth. IPC-2221 recommends paying attention to certain details in order to design for reliability. This includes choosing materials that are able to tolerate temperature, humidity, and vibration. Also choosing materials with structural strength and that properly conduct electricity and retain signal integrity.

The standard also discusses considerations to be made regarding connecting the components on the board. Trace width and trace spacing is also discussed in the document in the context of choosing the correct trace sizes to handle the current required by the circuit. The dielectric material of the PCB is also important in controlling impedance and retaining the integrity of high speed signals. It also mentions practices to properly ground the circuit and reduce EMI. Further reading into the document mentions thermal management and making sure to use heatsinks for heat dissipation especially when working with high power components, as heat buildup can reduce the reliability and longevity of the device.

Related to all of those topics is also component placement which has an effect on heat dissipation and EMI. A specific example of component placement having an effect that will be used later in this project is placing capacitors close to the voltage input pins of a microcontroller in order to ensure voltage stability. Placing connected components near each other reduces possible interference issues encountered by the device and can reduce the overall complexity of the PCB design, which can reduce layer requirement and optimize for cost.

IPC-2221 goes on to discuss the topics of mechanical design and manufacturing constraints of the PCB. It is important to design a board in such a way that it is possible to be manufactured by the PCB manufacturer. Relevant constraints such as minimum trace width, minimum trace spacing, minimum hole size, and the minimum distance between vias among other things. These constraints are very important to pay attention to while designing the circuit because only noticing them once the design is completed and you are ready to send them off to the manufacturer could mean needing to go back and redesign major parts of the PCB.

Some final things described by IPC-2221 include environmental factors such as choosing materials that comply with environmental regulations, for example RoHS compliance. Though this also applies to choosing durable materials and coatings that will last in harsh conditions. This of course depends on the specific application of your device, but should be taken into consideration. Lastly it describes thinking ahead in terms of component placement about the assembly process as well as the troubleshooting process to make sure that these processes are not complicated and components can be placed on the board without issue. Also creating a detailed BOM as well as including design notes such as critical tolerances and requirements to inform other people who may have to work with your PCB design.

In conclusion, IPC-2221 will be useful for us to reference as we go through the process of designing our PCB to make sure that we make decisions that will make any future processes easier. We must pay attention to the design requirements of JLCPCB to make sure they are able to manufacture our circuit

board while choosing component placements that will make assembly and troubleshooting of our PCB easier. We must also make sure to choose components that are reliable so that our PCB is less likely to fail in the future.

4.1.2 SPI Standard

The Serial Peripheral Interface (SPI) specification defines a form of communication between devices where one acts as a master to multiple slave devices. This uses a 4-wire connection between devices with the pins functioning as the Chip Select (CS) pin, the Serial Clock (SCK, SCLK) pin, the Master-In-Serial-Out (MISO) pin, and the Master-Out-Serial-In (MOSI) pin. The functionalities of these pins and the usefulness of the SPI standard will be discussed in this section.

SPI is a full duplex communication standard that we will be employing in our project to enable communication between the microcontroller and the microSD card that will be storing our sound files. It is a synchronous protocol that times data transfers with clock pulses on the SCK pin. We will go through the pins to explain the purpose and significance of each one.

The Serial Clock (SCK, SCLK) pin, as earlier stated, provides a clock signal to all devices connected to the master device to keep data transfer synchronized. The clock signal is sourced from the master device and sent as a square wave on a shared line. Devices following this clock signal will do data transfers on either the rising edge or falling edge of the signal depending on its specific properties.

The Chip Select or Slave Select (CS/SS) pin is used to choose which slave device the master device is communicating with. Only one slave device may communicate with the master device at a time otherwise there will be communication issues. This pin is an active low pin, thus to enable a slave device for communication the pin must be driven low.

The Master-Out-Slave-In (MOSI) Pin is the pin used to transmit data from the master device to the slave devices. The line between all devices on the MOSI pin is shared, however it only has an effect on the slave device currently enabled by using the CS pin shown above. The Master-In-Slave-Out (MISO) Pin functions as the pin that slave devices use to transmit data to the master device. This line is also shared between all of the devices. Only one slave device is communicating over this line at a time as well, depending on which slave device is enabled. This line will have no effect on disabled slave devices.

With these connections made, the master device can begin communications with slave devices. This begins with the master outputting the clock signal. The clock signal used depends on what the slave and master devices are capable of handling. The master device should output a clock signal with a frequency within its own operating range but not exceeding the lowest maximum frequency that can be handled by any one slave device. It should also be greater than the lowest frequency that any slave device can handle. Once the master device is outputting a proper clock signal, the system then needs to drive the CS pin of any single slave device low so that it can be enabled for communication. Once the correct slave device has been enabled, the devices can begin sending

digital bits across the MISO and MOSI pins according to the speed of the clock cycle. The bits are of course transmitted and received in series relative to each other and are read in the order that they are received. In SPI, there are no stop or start bits and the datastream is continuous, the full duplex nature of SPI also allows for both devices to send and receive data simultaneously, making it a capable communication standard for many devices. The actual content of the messages is not dictated by the SPI standard, that is dependent on the specific devices being used and what data they expect to send and receive. The SPI standard simply defines the overall structure and format of the connections.

4.1.3 The MIDI Standard

MIDI (or Musical Instrument Digital Interface) is a music communication standard that has been around since 1983. To this day it is widely used and it can be found almost everywhere that you would need to digitally transmit note values and velocity values as well as numerous other functionalities. MIDI is an extremely capable communication standard that is also extremely simple to implement.

MIDI data is usually transferred through a cable with a male 5-pin DIN connector on either end. Though this connector has 5 pins, MIDI only uses 3 pins to transmit musical data. The innermost 3 pins are used to communicate data. In the case of MIDI output the central pin, pin 2, is connected to ground. Pin 4 is then connected to a +5V DC source with a 220Ω resistor in series with it. Lastly Pin 5 is connected to the UART Tx pin of a microcontroller and sends digital signals across the cable in order to transmit information. The other two pins (1 and 3) are unused for the MIDI standard but can be used by other devices for custom purposes wherever necessary.

Messages in MIDI are sent over UART at a baud rate of 31250 Hz. The messages consist of 8-bit segments transmitted over the UART line to convey different properties about the note including when it begins, when it ends, the velocity of the note, what MIDI channel it is being transmitted on, and other variables such as MIDI Control Change messages that may be controlled by different knobs or faders on the MIDI controller being used by the user to change the characteristics of the sound. For our purposes, we are concerned with how to start a MIDI note, end the MIDI note, define the correct note value, and define the velocity with which the MIDI note should be played.

MIDI messages consist of different bytes which serve different purposes. The first byte in a MIDI message is the Status byte. This byte defines the type of message being transmitted. Following the Status byte are Data bytes which convey further information about the specific message defined by the Status byte. Once a Status byte is sent, the system will assume all subsequent Data bytes will be in the context of the most recent Status byte received. This means that if you are transmitting only Note messages you do not need to resend the Status byte for each note.

To send NOTE ON and NOTE OFF messages on MIDI channel 1, a Status byte must be sent as the following: 1001 0000. The most-significant 4 bits are what define the message as a NOTE ON status, and the least-significant 4 bits define

the use of MIDI channel 1. MIDI channel 1 is the most commonly used channel for general purposes, other channels have uses when using multiple MIDI instruments in the same system.

The next bytes in our NOTE ON message define the pitch and velocity of our note. The pitch byte contains a most-significant 0 bit followed by 7 bits to define the note value. MIDI note values range from 0-127, giving plenty more octaves than are available on conventional pianos. Referring to the earlier discussion in section 3.1.5.3, the note value for a kick drum would be 36. In this case the pitch data byte we would want to send would be as follows: 0010 0100. This would tell the system to begin a MIDI note of value 36.

The last byte in this message would be the velocity byte. This follows the same structure as the pitch data byte except the least-significant 7 bits define the velocity value and not the pitch. MIDI velocity values also follow a 0-127 scale where 0 is the least – or quietest – velocity, and 127 is the most -- or loudest -- velocity. For example if we wanted to transmit a velocity value of 100, we would send a message as follows 0110 0100.

This completes the entire NOTE ON message providing the receiving system with the information it needs to begin a MIDI note. The message in its entirety would be sent as 0x90 0x24 0x64. This simple message would be sent over the UART channel at 31250 baud to our MIDI output jack where it would be transmitted to a computer or other MIDI receiving device to begin a kick drum sound. Notice that Status byte messages always have a most-significant bit of 1, while Data byte messages always have a most-significant byte of 0.

The next step is to send a NOTE OFF message to end the MIDI note we just began. If we do not send a NOTE OFF message, the system will continue to assume the note is being “held down” similar to a key on a piano. This will create numerous issues for a person trying to record the MIDI information. In order to send a NOTE OFF message we need to send another Status byte to change our MIDI Running Status from NOTE ON to NOTE OFF. The NOTE OFF Status byte is as follows: 1000 0000. In this case the most-significant 4 bits define the message as NOTE OFF and the least-significant 4 bits define the message as being on MIDI channel 1.

After this follows another two Data byte messages. These messages define the pitch value of the note being turned off as well as the release velocity of that note respectively. The pitch message would be the same as it was for our NOTE ON message as a 0010 0100 or 0x24. The release velocity message is generally set to 0. Release velocity is a rarely used value for most MIDI devices, and for our purposes will be set to 0. So for this byte it is simply transmitted as 0000 0000 or 0x00.

The entire NOTE OFF message would then be seen as the sequence 0x80 0x24 0x00. This message would be sent momentarily after the NOTE ON message and the entire pair of messages would allow us to properly transmit a MIDI note to a computer or other device where it will be recorded and not hang because it is not properly closed at the end of the note’s duration.

This is about as advanced as our usage of the MIDI standard will get, however the standard itself has far more capability than just this. More advanced

MIDI instruments would be able to transmit multiple notes on multiple channels as well as Pitch Bend messages which continuously change and affect the pitch of all other notes in ways according to the receiving instrument. There is also something known as MPE or MIDI Polyphonic Expression which allows multiple notes that are being played simultaneously to be pitch bent or modulated all individually from one another. It is clear why MIDI as an open standard has stood the test of time and is still found everywhere in the digital music space today as it is an extremely capable standard while also being surprisingly simple. This is what makes it such a good choice for us to use in transmitting our note information.

4.2 Practical Constraints

4.2.1 Time

For this project one of the main and most restrictive constraints we face is the constraint of time. This project comes with a strict deadline by which we must meet certain requirements. In this time we must be able to synthesize an entire project. A specific issue that was faced was finding a project that was complex enough to create an extensive report on, but also simple enough that it would be feasible within the semester. A way that we adapted to this was to find a base idea and expand upon it until it is of sufficient complexity.

Time becomes a very limiting constraint especially in regards to shipping times on certain products. When we learn that we need something new or need to purchase a replacement part, we must wait a significant number of days before we receive the part and are able to continue making progress. This is able to be remedied by paying for express shipping or faster shipping options so that we receive parts earlier, however this shifts the constraint over to cost which will be discussed later.

Since this is the first half of the Senior Design courses, a lot of time is spent researching how to do certain things and the methods to do them correctly. This takes up a significant amount of the time that we spend on the project as it is a required step that cannot be worked on later or pushed to the side. When we have an idea to do a certain thing a certain way, we then need to research if our idea is possible and can be executed in the way that we think it can. There is nothing particularly guiding our research except ourselves so the problem can sometimes be less about where to find the correct information and more so what the correct information even is. We needed to deep-dive into a list of different topics during our time researching to verify that the ideas we had were even along the correct line of thinking. If this were not an issue that was encountered during our research then we would be able to come to the conclusion of how to do something much faster and we would be able to explore options much more quickly.

What helped in this area was reading several forum posts online as well as seeing videos of people doing similar things to what we wanted to do and hearing what they said were difficult issues they faced and how to overcome them. The internet was an extremely valuable resource for us to get research done.

Other than that, time is especially a constraint because although we may budget a certain amount of time for something, we may encounter problems with

it during our work that make it take up a significantly larger portion of our time. Because of this we have had to give ourselves a safety net in terms of time spent researching or working on something to make sure that if a problem is encountered, there is some time set aside to remedy it while not causing other issues in our schedule.

One last issue that plays a role in both the constraints of cost and is the PCB design. Getting the PCB produced takes time for both the production and shipping of it. For this reason it is generally beneficial to avoid having to get the PCB remade. If components need to be changed or moved around on the board it will need to be remade, so we have put off having the PCB produced just yet until initial testing is over to reduce the number of iterations of the board that will be needed.

Overall, time is one of the prevalent constraints with this project and is the factor limiting us the most. Regardless, we have been able to make ample progress within the timeframe provided and are on track to have our researched design completed this semester. With the information we now have on what time estimates are realistic for which processes, it should be easier for us to determine our goals for the next semester

4.2.2 Economic

This project is being funded by the members of the group and is not receiving any external funding. Due to this fact, decisions must be carefully made when choosing how to spend money to minimize the economic impacts of the project. Although we have a set cost that we would like to achieve on the BOM for assembling the project, this does not take into account the costs of shipping, development tools, and even mistakes that may be made during development.

A specific example that can be made relates to ordering parts from vendors such as mouser.com, digikey.com, or taydaelectronics.com. When ordering from these vendors, after choosing the components that you need, you generally need to pay a flat rate shipping cost to your location regardless of how much you have ordered unless the weight of your order is particularly high. Due to this, the smartest option is to order all of the components you need in one order. However sometimes once parts are ordered and tested or assembled in the project, we might realize that we need more components than we had first thought. Sometimes the extra components required are only a few small components that can cost less than a dollar total. Despite the low cost of these extra components, the shipping cost will remain the same as it was during the first large order and will need to be paid again.

This can occur any number of times depending on how well and how thoroughly research for that part of the project has been done. To reduce the number of repeat orders we may need to make, we have been trying to research each part of the project thoroughly and double-check that our bill of materials includes everything we might need. This also extends to making sure the type of components we buy have the correct specifications that are relevant, determined

by checking the datasheet and corroborating that with research that has been done to ensure the part will be compatible with the rest of the project.

The economic constraint of the project is heavily related to the time constraint as well. Doing extra research and double-checking that we are doing things correctly takes extra time which is a limited resource to us. This also includes doing research so that the cost of the bill of materials meets our budgeting goal. Failing to do extra research however can cost us more time as well. As stated in the previous section, if we need to order more parts not only does that cost extra money that wouldn't have been needed had we ordered everything together, but it also takes more time for the parts to arrive and for us to test and incorporate them into our project. Then, the time expense of needing to test and incorporate those new parts plays into whether or not we should spend extra money on express shipping. From this it becomes obvious that the constraints of time and economy are intertwined with each other, and that decisions regarding them must be made with thorough consideration.

Other economic issues we might face could be related to exceeding our projected budget for the bill of materials. Avoiding this can be done by cross referencing multiple vendors to find who has the lowest price for certain components. These decisions must also be weighed with the cost of having multiple orders from separate vendors as the shipping costs increase with more orders. Certain vendors may be the only ones to have some components available, in that case it would be smarter to purchase other components from them as well considering that a shipping payment would already need to be made to them in the first place.

Overall there are many ways for us to consider minimizing the costs of our product, yet some mistakes may be made and only understood in the future. It is important for us to be frugal in this situation as the money for the project is being provided by us ourselves. Time and economic factors interact with each other and must be considered accordingly to ensure that we are making the best use of both our time and money.

4.2.3 Manufacturability

Another constraint that we will face in the time we spend working on and researching this project is the constraint of manufacturability. Our project needs to be something that can be manufactured with ease. This does not just mean a PCB that is able to be made by our manufacturer, but also that it is something that we can easily make ourselves without too much struggle. If the project takes an excessive amount of time for us to produce, or requires tools that we do not have consistent access to let alone any, then it is not easily manufacturable and may not even be feasible to produce.

We want to avoid any situations like this by being sure that what we are researching and designing can be created with relative ease and does not require exotic methods of production. This can be done when designing our project by thinking ahead about our ideas and making sure they are realistic and within the scope of this semester. This also ties back into the two previous constraint topics

of both time and economics. Keeping our design simple is beneficial to manufacturability as well as time and economics. If our design is more manufacturable, it is also more likely to be a more economical solution as well as take less time to produce. Our ideas must be kept within what is reasonable to be done by a team of three people in a semester.

Though this may be true, it is also still possible for us to design something that cannot be feasibly manufactured while also still being within our economic and time constraints. This mistake is especially possible for us to make if we do not do proper research on a topic or falsely assume something that we shouldn't have. We need to be careful to make sure that we are thoroughly researching all of our topics while also thinking ahead about how we are going to execute them. The issue here is that this may affect our time and economic constraints negatively. As was discussed previously, doing extra research can be costly to both time and economics, so doing more research to ensure manufacturability will take time regardless, and could also eat into the allotted research time that we need to make sure that our ideas are economically feasible as well. Though, ensuring manufacturability can also help ensure that our project is economical as well and can reduce the amount of time spent researching and producing it. This shows that a large factor in determining our overall efficiency is making sure that we are doing the correct research and learning the things we need to sooner rather than later.

4.2.4 Health & Safety

Health and safety is a lesser concern for our product but should still be considered. We want to make sure that our product does not pose any hazard for any of the users. This means designing a product that is safe to use and won't cause any danger for the user in any way. Possible ways that our product could be dangerous are either through the shape of it, through the hazard of electricity, or through using chemicals that are toxic to humans or could be a health risk in the long term.

Designing for this is not hard; the hazard of electricity is practically nothing due to the fact that we are using an AC/DC converter to bring power to the main board. So there is no AC power that can be dangerous to the user after the converter. Other than that we do not plan on using any hazardous materials such as paints or glues, that should not be an issue to the user either. Lastly, the shape of the product should not be an issue either. We plan to make sure that any sharp corners are sanded down and that we use finished wood or some other smooth material to construct the main enclosure.

4.2.5 Ethical

Another constraint of lesser concern to us is the constraint of ethics. The constraint of ethics is concerned with how the product will affect the end user or other people in society. Our specific product may actually be seen as an ethically beneficial thing, as it allows musicians on a limited budget to produce music in an easier way or with a better workflow than they would have otherwise. This could

help them finish projects that they otherwise wouldn't have, and have their voices and music heard by more people.

This point can be seen as ethically beneficial to people as it helps them achieve the things they want to. Ethics can also extend to the content of their music and the message that it promotes. Generally music is not promoting an ethically negative message, though if it is then that is out of the control of us the authors and is not something that we would be able to remedy. This point will also be discussed somewhat in the political section of this chapter. If the user decides to create music that promotes a political message or agenda using our product as a tool in their music production process, then that is out of our control and is their decision.

4.2.6 Political

Political constraints do not have a significant effect on our project. There are not necessarily any political applications that it could be used in. There were also no political intentions behind the design of any part of the project and it is intended simply as a tool to be used by amateur musicians to aid them in creating music. It is possible that it may be used by musicians in the creation of songs that may push a political agenda or advocate for change, but this choice of use is entirely a decision of the end user and does not depict the beliefs of any of the people that worked on the project. None of the authors of the project have intended for it to be used in any political way nor have we designed it for that purpose.

4.2.7 Environmental

The environmental constraint is concerned with the use case of the product and what is feasible given the real-world conditions it will be subject to. Our project will be intended to be used indoors at room temperature. Water is generally not a concern even though a spill could damage the project, but that is not something that we intentionally designed for. Something we needed to be concerned with is our project's ability to withstand shock and vibration. Different components of the drum kit are going to be struck with drumsticks with a decent amount of velocity. We did not design the project with the intention that the user is going to be hitting it as hard as they would hit a drum kit at an actual live event, but just as hard as they might hit a kit in a relaxed practice setting.

The drum pad enclosures which will be developed later will need to be made with this in mind, that they must be able to survive being hit repeatedly as well as still provide reliable readings to our central processing unit. Something that will help this central hub avoid withstanding excessive force is the fact that the drum kit components will be attached to it using only wires to transmit the data received from them. Not much force and minimal vibration will be transferred to this central hub which will be within the spec that the 3.5mm cable standard will be able to tolerate.

Lastly, the foot pedals that will be used along with this system have been designed for the use of being pressed by the user's foot. The only concern with

this would be the kick pedal being pressed hard if the user is using the kind of force that they would use at a real live event. We are hoping the fact that the kick drum sound effect will play at a constant volume and will not be velocity sensitive will help to reduce the likelihood of this happening

4.2.8 Sustainability

Sustainability also plays into the previous section with how much use the product can withstand. The project will be designed in such a way that it should be able to survive just fine under the usual conditions of use. The drum pads and cymbal enclosures that will be used will also withstand being struck by the drumsticks as long as it is not struck too hard. We intend to design the pads in a way such that the sensors do not get damaged from regular use.

The cables being used in the project should also be able to withstand being plugged and unplugged multiple times as long as it is done correctly. The connectors being used to hold in these cables should also be fine as they will be panel mounted and secured to the actual structure of the central processing unit's enclosure. We chose to use cables and connectors instead of hardwiring the components for this reason as well; so that under high stress situations when separating a component from the central hub such as something falling over, the components will disconnect from each other instead of damaging hardwired connections and coming separated from the PCB itself.

We will also need to secure the cables that are connecting to the force sensors to the enclosures that are housing them in some way, such that if there was excessive force put on those cables it would be transferred to the enclosure and not the sensors. The force sensors themselves are delicate and could rip or break if stressed too heavily. This could also dislodge the sensors and make their readings no longer as accurate as they should be. This could cause long-term issues for the continued use of the system and make it less sustainable.

5. Comparison of ChatGPT or Similar

ChatGPT can be a great source and can help speed up some project tasks like brainstorming ideas, summarizing, classifying and theming primary research insights, but the content of chatGPT should be always double checked for accuracy.

Case Study 1:

Question "How can I optimize the power consumption of an electric drum?"

"Optimizing the power consumption of an electric drum involves improving circuit efficiency, component selection, and software optimization. Below are actionable steps:

1 Efficient Circuit Design

- Use Low-Power Microcontrollers: Select microcontrollers optimized for low power, such as ARM Cortex-M0/M4 or low-power MSP430 chips. Ensure they have sleep modes for when the system is idle.
- Optimize Power Rails: Use efficient DC-DC converters instead of linear regulators to reduce power loss in voltage regulation.
- Component Selection: Choose ICs with low quiescent currents and MOSFETs with low on-resistance to minimize losses.
- Minimize Active Components: Reduce the number of continuously powered components. Use multiplexers to share resources like ADC channels or GPIO pins.

2 Power Management Techniques

- Implement Sleep Modes: Design the system to sleep during inactivity and wake on specific triggers (e.g., striking the drum).
- Dynamic Voltage and Frequency Scaling (DVFS): Adjust the system clock and voltage dynamically based on performance requirements.
- Control Peripherals: Power down or disable unused peripherals when not in use.

3 LED and Display Efficiency

- Use Low-Power LEDs: If the drum uses LEDs for feedback or aesthetic purposes, opt for high-efficiency LEDs with low current requirements.
- Optimize Display Power: For any display interface, use energy-efficient technologies like e-paper or low-power OLEDs. Reduce refresh rates or dim the display during inactivity.

4 Sensor Optimization

- Optimize Piezo Sensors: Piezoelectric pickups can be energy-intensive when amplified. Use efficient op-amp designs or digital sensors with low power demands.
- Threshold-Based Wake-Up: Ensure the sensors only trigger processing when a significant event occurs, avoiding unnecessary processing.

5 Firmware Efficiency

- Efficient Signal Processing: Optimize algorithms (e.g., FFTs for sound analysis) to minimize computational overhead.
- Event-Driven Programming: Design the firmware to process events only when necessary instead of polling continuously.
- Code Profiling: Profile your code to identify and reduce energy-intensive tasks.

6 Testing and Measurement

- Power Profiling: Use tools like the Joulescope or an oscilloscope to measure power consumption during operation and optimize bottlenecks.
- Iterative Design: Test early prototypes and make incremental improvements to the design.

7 Housing Design Impact

- Thermal Management: Optimize the housing for heat dissipation to ensure components work efficiently without requiring additional power-hungry cooling systems.
- Insulation: Use materials that reduce power loss in environmental extremes.” (Chatgpt 2024)

In this case Chatgpt provided us with a more complete and organized answer that would satisfy our goal of the power consumption limit combining efficient circuit design, advanced power management, optimized firmware, and smart housing design which can largely reduce the power consumption of our electric drum, ensuring a good performance.

Google searches are more general and may not exactly refer to what we are looking for, requiring more time to search and verify the information and sometimes the information can be irrelevant to our question, but it provides us with different sources and websites of real world experiences that others share. Google’s answer for the same question gives us a list of websites about the power usage of different existing electric drums.

Asking gemini google which is the AI-powered from Google using the latest AI technology from google the same question.

“Here are some key considerations for optimizing power consumption when designing an electric drum kit:

1 Component Selection

- **Microcontrollers:** Choose low-power microcontrollers like those from the ARM Cortex-M series. These offer a good balance of performance and energy efficiency.
- **Sensors:** Opt for sensors with low power consumption, such as capacitive touch sensors or piezo sensors.
- **Power Amplifiers:** Select efficient Class-D amplifiers for audio output. These amplifiers convert electrical energy into sound with minimal power loss.
- **Display:** If a display is necessary, consider using low-power OLED or e-ink displays.

2 Power Management

- **Power Supply:** Design a power supply with high efficiency to minimize power loss during conversion.
- **Battery Management:** If battery-powered, implement efficient battery management systems to maximize battery life.
- **Power Switching:** Use power switches to selectively power down components when not in use.
- **Low-Power Modes:** Utilize low-power modes for the microcontroller and other components during idle periods.

3 Circuit Design

- **Minimize Current Leakage:** Pay attention to circuit design to reduce current leakage, especially in analog circuits.
- **Optimize Signal Routing:** Minimize the length of signal paths to reduce power consumption.
- **Use Efficient Digital Logic:** Employ low-power digital logic design techniques.
- **Software Optimization:**
- **Efficient Algorithms:** Use efficient algorithms to minimize processing power.
- **Task Scheduling:** Optimize task scheduling to reduce unnecessary power consumption.
- **Power-Saving Modes:** Implement power-saving modes in the firmware to reduce power consumption during idle periods.

4 Additional Considerations

- **Wireless Technology:** If wireless connectivity is required, choose low-power wireless protocols like Bluetooth Low Energy (BLE).
- **Environmental Factors:** Consider the operating environment and temperature range. High temperatures can increase power consumption.
- **User Interface:** Design a user interface that minimizes power consumption, for example, by using touch sensors instead of mechanical buttons.

By carefully considering these factors, you can design an electric drum kit that is both efficient and enjoyable to play." (gemini.google.com 2024).

Compared to the google search engine, gemini google gives a more direct answer to our question, but focusing on different aspects than Chatgpt. Both systems provide important answers, but Chatgpt's concentrates upon the technicity of reducing power consumption through design and firmware optimization.

Case Study 2:

"What are the best microcontrollers for an electric drum project?"

"When selecting a microcontroller (MCU) for an electric drum project like your **OmniDrum**, consider factors such as real-time performance, input/output (I/O)

requirements, audio processing capability, and power efficiency. Here are some of the best microcontroller options based on these criteria:

1. ESP32

Why it's great:

- Dual-core processor (240 MHz) for handling multiple tasks, such as input processing and audio output.
- Integrated Wi-Fi and Bluetooth for wireless connectivity (e.g., MIDI over Bluetooth).
- Plenty of GPIO pins, including ADC for reading piezo sensors or trigger inputs.
- Low power consumption in deep sleep modes.

Downside: Limited RAM (520 KB SRAM) may require optimization for complex audio tasks.

Best for: Wireless MIDI drums, basic audio synthesis, and cost-efficient designs.

2. STM32 (e.g., STM32F4/F7 series)

Why it's great:

- High-performance ARM Cortex-M processors (e.g., Cortex-M4 or Cortex-M7).
- Built-in DSP (Digital Signal Processing) for audio synthesis and filtering.
- Extensive GPIO and support for ADC/DAC and high-speed peripherals.
- Low-power modes available for efficient designs.
- **Downside**: Slightly steeper learning curve due to feature richness.
- **Best for**: Advanced audio processing and real-time MIDI implementations.

3. Teensy 4.1

Why it's great:

- ARM Cortex-M7 (600 MHz), extremely powerful for audio applications.
- Built-in support for the **Teensy Audio Library**, making audio processing, synthesis, and effects easy.
- Ample GPIO, including high-resolution ADC and DAC support.
- Compact form factor and easy to integrate.

Downside: Relatively higher cost.

Best for: High-performance audio drums with complex effects and real-time synthesis.

4. Raspberry Pi Pico

Why it's great:

- Dual-core ARM Cortex-M0+ processor (133 MHz).
- Affordable and readily available.
- Support for MicroPython or C/C++ for rapid prototyping.
- Decent GPIO and peripheral support for drum sensors and MIDI interfaces.

Downside: Less suitable for heavy-duty audio processing.

Best for: Budget-friendly designs and simpler electric drum setups.

5. Nordic nRF52 Series

Why it's great:

- Built-in Bluetooth Low Energy (BLE) for wireless MIDI.
- Ultra-low power consumption.
- Decent processing power for input handling and MIDI.

Downside: Limited audio processing capabilities compared to STM32 or Teensy.

Best for: Wireless, battery-powered MIDI drum kits.

Comparison Table:

Microcontroller	Processing Power	I/O Pins	Audio Features	Connectivity	Power Efficiency
ESP32	Medium	High	Basic	Wi-Fi, Bluetooth	Medium
STM32 (F4/F7)	High	Very High	Advanced	Optional (via modules)	High
Teensy 4.1	Very High	High	Advanced	Optional (via modules)	Medium
Raspberry Pi Pico	Medium	High	Basic	None	High
Nordic nRF52 Series	Medium	Medium	Limited	Bluetooth Low Energy	Very High

„

(ChatGPT, 2024)

In this case ChatGPT prides us with a list of different microcontrollers giving details about the strengths of each one of them depending on the project priorities, but again this requires a deeper research to learn more about each proposed MCU. After searching in different websites and learning about the properties of different Microcontrollers, **STM32G473R LQFP64** was chosen to be the best choice for our OmniDrum due to its balanced capabilities and cost ensuring the reliability and scalability for the project.

Case Study 3:

“What sensors are ideal for detecting drum hits accurately?”

“Detecting drum hits accurately in an electronic drum system like your **OmniDrum** typically requires sensors that can reliably capture force, vibration, or changes in pressure. Here are the most suitable types of sensors:

1. Piezoelectric Sensors

How They Work: Piezoelectric sensors generate a voltage when deformed or subjected to vibration. They are highly sensitive to impacts, making them the most common choice for electronic drums.

Advantages:

- High sensitivity to vibration and force.
- Compact and lightweight.
- Low power consumption.

Use Case: Ideal for detecting hits on drum heads, cymbals, and pads.

2. Force-Sensitive Resistors (FSRs)

How They Work: FSRs change resistance based on the force applied to them.

Advantages:

- Can differentiate between soft and hard hits.
- Easy to integrate and cost-effective.

Use Case: Suitable for detecting pressure and softer touches, like rim shots or light drum rolls.

3. Capacitive Touch Sensors

How They Work: These sensors detect changes in capacitance caused by proximity or physical contact.

Advantages:

- Ideal for very light touch-sensitive inputs.
- Can be used for multi-zone detection.

Use Case: Good for integrating extra control zones or surface touches (e.g., triggering effects).

4. Laser or Optical Sensors

How They Work: Infrared or laser-based sensors detect the position and speed of a striking object.

Advantages:

- Highly accurate.
- Can measure velocity for dynamic response.

Use Case: Can be used in advanced setups where drumstick motion needs to be tracked.

5. MEMS Accelerometers

How They Work: Measure acceleration caused by vibration or impact.

Advantages:

- Compact and lightweight.
- Can provide velocity and impact angle data.

Use Case: Useful for unique designs or multi-zone drums.

Recommended Approach for OmniDrum

For the best accuracy and responsiveness:

- **Primary Sensor:** Use **piezoelectric sensors** for detecting drum hits and differentiating between light and hard strikes.
- **Supplementary Sensor:** Add **FSRs** to enhance sensitivity for softer hits or pressure-based input.
- **Advanced Features:** Consider **optical sensors** for tracking motion or integrating with dynamic visual feedback."(ChatGPT, 2024)

ChatGPT provided us with a great selection of the sensors that can be used for our OmniDrum explaining the use and the advantages of each case. Chatgpt suggests that we use primarily piezoelectric sensors. However, they may be more susceptible to environmental noise and crosstalk from other parts of the drum. After further research, we found that the force sensors are more ideal for our project due to their ability to minimize the unwanted noise and crosstalk from other parts of the drum.

6. Hardware Design

6.1 Main Power Supply Circuit

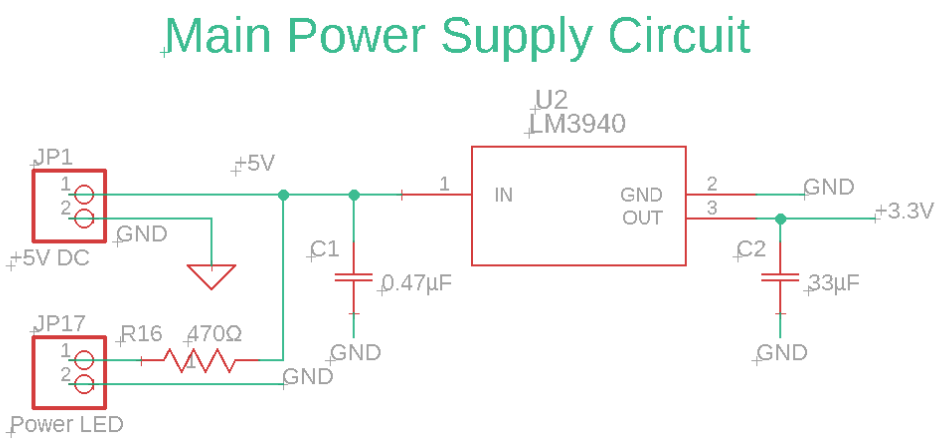


Figure 6.1 Main Power Supply Circuit Schematic
Image Source: Author

Shown here is the schematic for the Main Power Supply Circuit. The power will be provided at JP1 from a 5V AC/DC power supply plugged into a barrel jack which will be mounted on the outside panel of the unit. This will form our +5V line and our GND. The +5V line then goes down through a 470Ω resistor out to JP17. JP17 will power an LED that will be mounted near the DC barrel jack to indicate that the device is being powered. The +5V line will then go into the LM3940 LDO regulator which will output a voltage of 3.3V to power the MCU and other components. The input line of the LM3940 has a 0.47µF capacitor connected to it and ground as was recommended in the datasheet, there is also a 33µF capacitor on the output connected to ground. The 33µF capacitor is electrolytic with the negative side connected to ground, it was also recommended to have an ESR rating within a specific range, the ESR of this capacitor is 420mΩ.

6.2 STM32G4 Power Supply

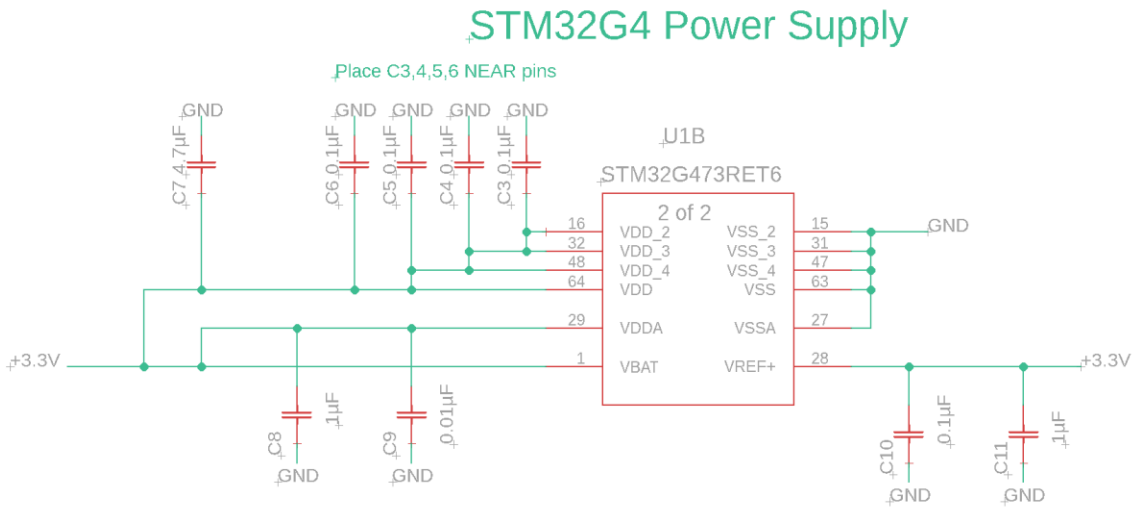


Figure 6.2 STM32G4 Schematic
Image Source: Author

This circuit shows the power supply scheme for the STM32G473RET6 microcontroller. The schematic split the microcontroller into two separate parts: one for the power supply pins, and the other for the IO pins. Above is the part for the power supply pins. The input voltage for the microcontroller is +3.3V. Several capacitors are shown connected to ground around many of the pins; this practice was recommended in the datasheet for the microcontroller in order to improve stability. It should be noted that capacitors C3,4,5,6 are explicitly recommended to be placed as close to their respective pins as possible for the best results. Due to this, the components shown here were specifically chosen to be SMD/SMT components. The rest of the components on the board are through-hole components due to simplicity of removing and replacing through-hole components when testing and fabricating. Through-hole components however would not have been able to fit as closely to the pins as desired. The capacitors here are all of the 1206 package size.

The VDD pins provide power to the main internals of the STM32G473RET6 while that VDDA pin provides power specifically to the analog portions of the microcontroller. The VDDA pins were given a different series of capacitors for stabilization, but receive voltage from the same +3.3V line. The VREF+ provides an external voltage reference for the analog circuits. There is an internal voltage reference for the analog circuits as well, however it is best practice to power this one. The VBAT pin was recommended to be given direct +3.3V without any capacitors required to be near the pins. The VSS and VSSA pins are all connected directly to ground.

6.3 Sensor Array

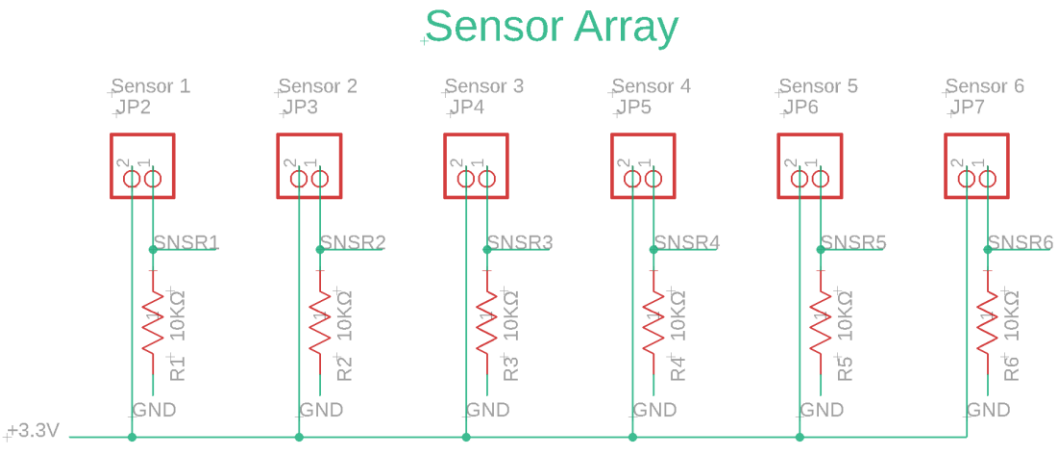


Figure 6.3 Sensor Array Schematic
Image Source: Author

The sensor array consists of the six sensor circuits shown above. The schematic shows JP2 - 7, acting as pin headers for the individual sensors. These pins will be connected to a series of panel mounted 3.5mm TS jacks. The jacks

feature two conductors and will allow us to easily connect and disconnect individual sensors to our main board. These force sensors will provide us with measurements of how hard each component of the drum kit is being hit. The leads of the sensors will be connected to a 3.5mm TS cable. This gives us an easy way to make the sensors removable from the device by using existing connectors with the same amount of conductors that we need. It also makes the sensors interchangeable in terms of which sensor controls which instrument on the drum kit.

The force sensors act as a variable resistor with two leads. We will connect one of these leads to an ADC pin on our microcontroller; each sensor will connect to its own individual ADC pin. According to the STM32G4 datasheet, the 12-bit ADCs read voltage values from 0V to +3.3V and output a digital value from 0 to 4095. The ADCs cannot sense a change in resistance, so we will need to make a voltage divider to enable the functionality of the sensors. To do this we will place a 10KΩ resistor in parallel with the ADC pin that will be connected to ground. When unpressed the sensor should be around 20MΩ. If we then apply +3.3V to the other lead of the sensor, we can expect the voltage at the ADC pin to be about 1.6mV. This should be seen as a near zero digital output value from the ADC pin. Conversely, when the sensor is pressed fully it should be near 0Ω in resistance. In this case the voltage seen by the ADC pin would be +3.3V and should register as a 4095 digital output value. This setup allows us to utilize the full range of the sensor as digital values from our ADC.

6.4 microSD IO

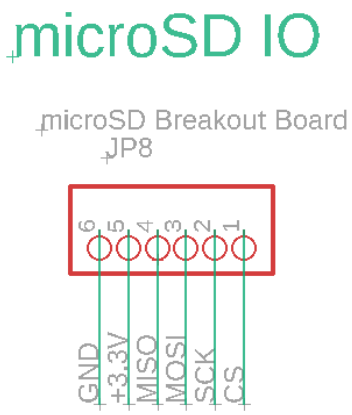


Figure 6.4 microSD IO Schematic
Image Source: Author

The microSD IO schematic shown above is very simple. The 6 pin header will connect to a breakout board that contains a microSD port so that it may interface with the microcontroller. The GND and +3.3V will be supplied by the main board and the other 4 pins will connect and communicate with the microcontroller over SPI.

6.5 Profile Select IO

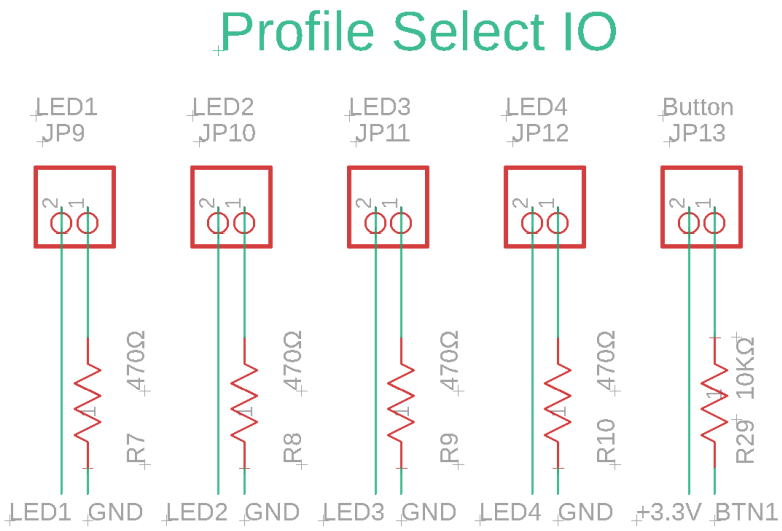


Figure 6.5 Profile Select Schematic
Image Source: Author

The Profile Selection IO facilitates the ability for the user to switch through the different sound profiles of the drum kit. These sound profiles will change the style of sound output. The system will default to the first profile on startup which will illuminate LED1. The button is set up similarly to the force sensors except it functions as a digital input. The pin should be set as a digital input with a pull down resistor. The button has 2 leads which when pressed will bridge together. One of the pins will be connected to the +3.3V line and the other will be connected to a 10KΩ resistor that then connects to the GPIO pin. This means that normally the input pin will be pulled low but when the button is pressed it will be pulled high. To the system this should be seen as a trigger for the profile to switch. The button will cause the system to rotate through the profiles in a round-robin formation. This is a momentary button that when released will immediately go back to the open position. Because of this the system must switch profiles only when it first sees the higher pin voltage and must not switch profiles again until the button has been released. If the button input bounces when pressed – as in it unintentionally quickly alternates between high and low – these inputs should not be taken into account for the profile switching. If this issue occurs it may be dealt with in the microcontrollers software.

When a new profile is chosen, an LED will light up to indicate which profile is currently selected. These LEDs are shown in the schematic and have a 470Ω resistor in line with them to limit their current. The input voltage for these pins will come from a GPIO that is set to an output pin in push-pull mode. Only one LED will be illuminated at a time to signify which profile is currently active. Each LED will pull approximately 7mA from the output pin. According to the STM32G4 datasheet, output pins are rated to supply no more than 20mA. All output pins combined may not source or sink more than 100mA combined. Because of this it is important to minimize current usage, so having only one LED illuminated at a

time would be desirable. There are four LEDs allowing for there to be four separate profiles. This could include three sound profiles and one MIDI output profile, or four sound profiles that all simultaneously output MIDI information and play the sounds associated with that profile.

The LEDs and the profile select button will be panel mounted as well, somewhere conspicuous to the user and easily accessed and visible. The ports JP9 - 13 may be populated with pin headers if using removable connectors is desired, or wires may be directly soldered from the LEDs and button directly to the PCB. The board should clearly denote which of the two pins is positive and which is negative so that the LEDs may be oriented properly.

6.6 Pedal IO

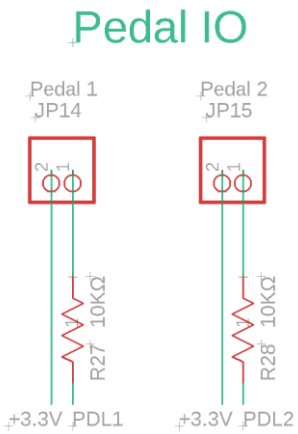


Figure 6.6 Pedal IO Schematic
Image Source: Author

The Pedal IO shown above functions similarly to the profile select button as a digital input. The pedals should be connected to the +3.3V line and a GPIO pin set as a digital input with a pull-down resistor. When a pedal is unpressed the digital input pin that it is connected to will read as low. When the pedal is depressed it will bridge the pins together and the input pin will read as high. The pedals will be connected using panel mounted ¼" TS jacks. Similarly to the profile select button, a 10KΩ resistor has been added in line with the pedal output as protection.

Most foot pedals used in musical applications have this connector. The connector has two conductors and functions as a switch. These foot pedals are used as actuators for certain devices in a musical setting. A common example usage would be a foot pedal used as a sustain pedal on a digital piano or MIDI controller. The pedal plugs into a ¼" TS jack on the digital piano or MIDI controller and when pressed down will sustain any notes being played. We are using these foot pedals for a similar application that will replicate the pedals used with drum kits.

The two pedals will have different purposes. The first pedal will be used to simulate a kick pedal on a drum kit. When pressed down, the system should use the pedal as an input trigger to play a kick drum sound in the same way that the force sensors are used. The second pedal will be used to simulate the pedal on a

hi-hat cymbal. This pedal will not trigger any sounds itself, but will be used as a check when the hi-hat force sensor is struck. If the hi-hat pedal is unpressed when the hi-hat force sensor is struck, the system should play an open hi-hat sound. If the hi-hat pedal is pressed when the hi-hat force sensor is struck, the system should play a closed hi-hat sound. It should be noted that usually the act of pressing the hi-hat pedal is used to make its own sound, however as the pedal is a digital input in our case there is no way to know how loud the user would like the sound to be and the sound may get in the way of other sounds when playing if it is too loud. If this sound were to be too quiet then it would be generally ineffective in any situation. Avoiding the use of this sound helps us save memory as well as cleans up the overall sound.

6.7 MIDI Output

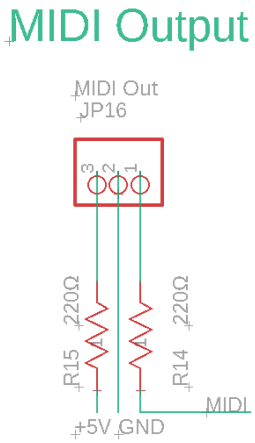


Figure 6.7 MIDI Output Schematic
Image Source: Author

Shown above is the schematic for the simple MIDI output that will be incorporated in our project. The 3 pin header will be connected to a female 5-pin DIN port on the side of the assembly. This port will connect to an external MIDI capable device to transmit the notes being played or to be recorded in a user's DAW. The MIDI output is powered by the +5V line and contains two 220Ω resistors. By design a 220Ω resistor should be placed across the +5V line to limit current. A 220Ω resistor is also placed on the UART transmitting line in order to protect it as well. These pins will connect to the MIDI output port either through connectors and pin headers, or wires directly soldered into the board.

6.8 Speaker Amplifier

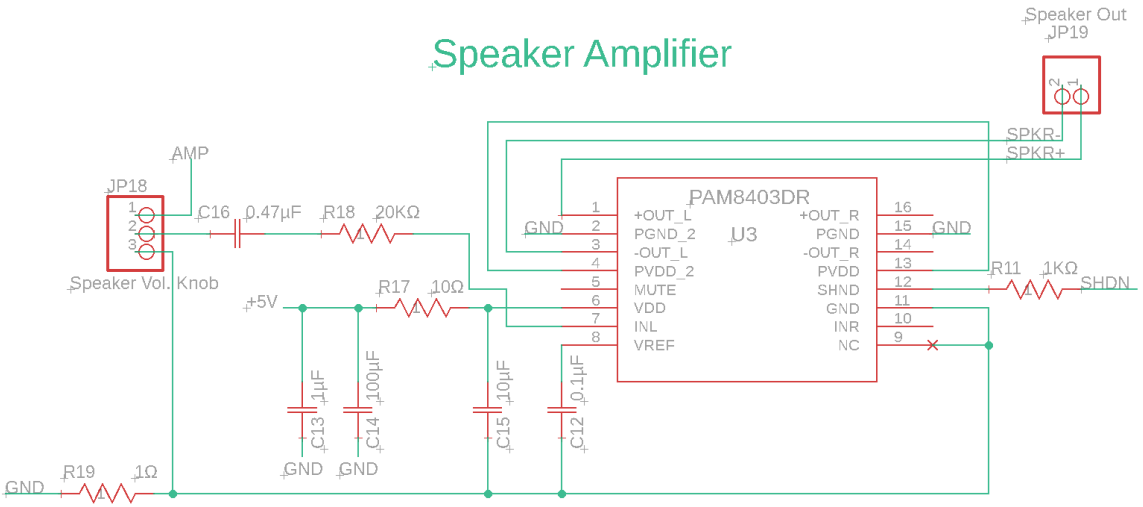


Figure 6.8 Speaker Amplifier Schematic
Image Source: Author

The speaker amplifier shown above uses the PAM8403 to drive the internal speaker by amplifying the sound coming out of the DAC pins of the microcontroller. The amplifier features many peripheral components to increase stability and to ensure the proper function of the amplifier.

The audio signal initially arrives from the 'AMP' line shown at the top left of the schematic. This line is connected to all of the DAC pins (Pins 18, 19, 20) on our microcontroller. The DAC pins on the microcontroller are buffered which means that they are protected from extraneous current and can be directly connected to the input of the amplifiers. This also allows us to tie the three pins together to add the signals so that all of the audio signals go to the same destination regardless of the DAC that is outputting. This also allows the use of multiple DACs simultaneously – if needed – to drive the amplifiers.

In this schematic, the AMP pin is connected to one of the leads of a potentiometer. This appears as a 1x3 pin header in the schematic because the potentiometer will be mounted to the outside of the device assembly. The potentiometer will connect to the main PCB at this point either through the use of a removable connector, or by soldering wires directly onto the board. This potentiometer acts as a volume knob for the speaker amplifier, allowing the user to adjust the volume to a desired level. The value of this potentiometer is 10KΩ; this was selected because it was the value shown on the example circuit diagram used as a reference for this amplifier circuit.

The other lead on the volume knob potentiometer connects to an isolated ground bus. A 1Ω resistor is used to separate the ground of the audio input device and the ground of the power source for the amplifier as a form of isolation. The example circuit schematic showed that a number of nodes needed to be connected to ground on the side of the 1Ω resistor connected to the audio input. This isolated ground line is visible at the bottom of the speaker amplifier circuit schematic and connects to a number of other components as well. It should be noted that the ground from the power source and the ground from the audio input source are the

same ground as both the power source and audio input source are on the same board and are not separate devices. Nonetheless, the resistor was included just in case it was needed anyway.

The ‘output’ of the potentiometer is connected to the main input of the amplifier. With this setup, the user is able to control how much of the input to the amplifier is influenced by the audio input source, and how much is influenced by ground.

As the audio signal travels into the input of the amplifier IC, it passes through a resistor and capacitor in series with each other. This forms an RC timing circuit which is used to limit the maximum frequency going into the amplifier input as well as control the maximum gain and keep the amplifier from saturating. This specific part of the circuit can serve a few different purposes but the general idea is to protect the amplifier.

Looking at the VDD pin, we can see that the +5V input must pass through a network of a resistor and some capacitors before entering the PAM8403. The first two capacitors in this network connect directly to ground. The values of them are 1µF and 100µF respectively. These capacitors are used to stabilize the input voltage of the amplifier to ensure it operates normally and is less affected by any possible noise it might otherwise encounter. After this the input power passes through a resistor and enters the PAM8403 in parallel with a 10µF capacitor attached to the isolated ground plane for stability.

Above this VDD pin is the MUTE pin. The MUTE pin functions similarly to the SHND pin which will be discussed later. This pin would allow us to mute the amplifier by pulling it low as it is an active low pin. When in use, the output of the amplifier will be temporarily muted, but the amplifier itself will remain powered on and active. For this application we do not have a use for this pin. By default the pin is pulled high by a pull-up resistor and so we do not need to worry about connecting it to anything and should leave it disconnected.

The pin above this is the PVDD_2 pin. This is the Power VDD pin which is shorted to another PVDD_1 pin on the other side of the chip. These pins provide supply voltage for the power side of the equipment whereas the regular VDD pin provides power for the analog section. These pins are not relevant to our application and are left shorted together.

Above those pins are the speaker output pins where our amplified audio signal is outputted. These pins will be used to drive our speaker at JP19. The 1x2 pin header at JP19 will connect to our internal speaker which will be mounted on the side of the assembly. These will be connected either by using wires with removable connectors on the end of them, or by soldering wires directly onto the main board. Between these two pins is the PGND or Power Ground pin. This pin is simply shorted to GND.

The other side of the PAM8403 is not used as much in our application because we are only utilizing one of its two channels. The input pin and both output pins on this side of the chip are left floating because of this. The PGND pin on this side of the board is still shorted to ground. There is another pin on the bottom left of the chip labeled NC. This pin is not connected to anything internally and does not serve a purpose. It has been tied to the isolated ground in order to reduce any

possible EMI it may cause. The GND pin on this side is of course tied to regular ground. The VREF pin will be discussed in the remaining part of this section.

The next pin to discuss on the PAM8403 is the SHND pin. This is the ‘shutdown’ pin. As stated earlier, the functions of the MUTE pin and the SHND pin are very similar. The SHND pin is an active low pin pulled high when not connected to anything. When pulled low it will shut down the bias circuitry of the amplifier chip, essentially turning it off. This is useful because unlike the MUTE pin, this will actually reduce the current draw of the chip significantly. Because of this added benefit, we have chosen to employ this pin to be used to achieve our goal of muting the internal speaker when headphones are plugged into the Phones jack of the device. The specifics of how this works will be described in the next section. This pin is connected to a pin on the microcontroller that functions as a push-pull digital output which will control the muting of the speaker amplifier.

Lastly, the VREF pin plays a very important role in enabling the functionality of the shutdown feature. This pin is connected to a capacitor which is then connected to the isolated ground. The capacitor here has a value of 0.1µF and determines how quickly the chip starts up or recovers from the shutdown state. It also helps to reduce noise from the power supply as well as reduce clicking or popping that may be heard when entering or leaving the shutdown state.

6.9 Headphone Amplifier

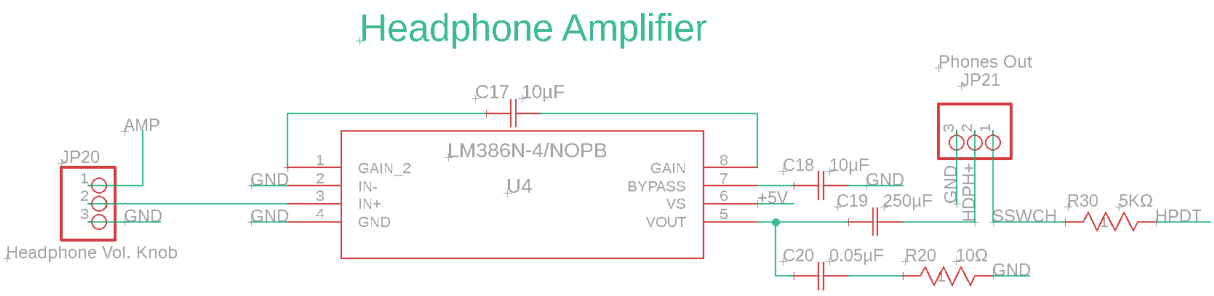


Figure 6.9 Headphone Amplifier Schematic
Image Source: Author

The headphone amplifier is the other component of the audio output system. This amplifier functions differently than the speaker amplifier as it drives a much higher impedance device. The assumed impedance of a device being plugged into the auxiliary output is 32Ω.

The signal flow for this portion of the circuit begins with the ‘AMP’ node on the left side of the circuit. This connects to the JP20 pin header; this 1x3 pin header will connect to a potentiometer that will be panel mounted to the outside of the assembly. This potentiometer will be used as a volume control knob for the user to adjust the output volume of the headphone amp to a desirable level. The other lead of this potentiometer is connected to ground. The amplifier input is attached to the wiper of the potentiometer and will change the mix between the ground signal and the DAC output signal that is flowing into the input of the amplifier. The value of this potentiometer has been chosen to be 10KΩ. This was what was

shown on the typical application circuit shown on the LM386 datasheet. The specific circuit being followed for this design is the typical application circuit that provides a gain of 200.

The power supply circuit of this amplifier is much simpler than the speaker amplifier. The VS pin on the amplifier is connected directly to the +5V line. The GND pin and the IN- pin are also both connected directly to ground.

The audio input from the potentiometer connects to the IN+ pin on the LM386 and gets amplified with a gain determined by the gain pins. The pins GAIN and GAIN_2 on the chip are biased using a capacitor to decide the value of the gain that is applied to the target signal. When not connected, the gain value of the LM386 is set to 20 by default. This is the lowest value and is the default in order to protect the user from accidentally damaging a device connected to the output in the case that they ignore the GAIN pins.

In our specific case, the gain pins are connected to each other by being connected to either side of a 10µF capacitor. This can be seen above the chip in the schematic above. Setting the chip to a gain of 200 gives us the maximum output power possible. We would like the output power to be at the maximum because our intended audio device for this jack has such a high impedance. From the datasheet we can see that even at the maximum gain that the LM386 can output, we still would not damage most headphones. Of course the user can plug whatever they want to into this jack so there is still the possibility of the user connecting a device that may be incompatible with the amplifier output and that could possibly get damaged.

The audio output of the amplifier comes out of the VOUT pin. This pin is then connected to a capacitor going towards the audio output jack, and a capacitor and resistor in series going to ground. The function of the resistor and capacitor in series here is to provide a load at higher frequencies. This stabilizes the amplifier and prevents oscillations from occurring. If removed, the oscillations could damage the amplifier.

Moving towards the output channel, there is a 250µF electrolytic capacitor between the amplifier output and the speaker. The positive terminal of the capacitor is facing the amplifier output pin while the negative terminal is facing towards the speaker input. This capacitor is used to block DC voltage from the speaker. This reduces distortion and protects the speaker.

Finally, at JP21 we can see the 1x3 pin header which will connect to the panel mounted internal speaker. The connection will be made either by wires with removable connectors on them, or wires that are soldered directly into the main board. Although the headphones output here is a mono ¼” TS jack, this connector has three pins instead of two. This is to facilitate the muting of the internal speaker when something is plugged into the Phones output jack. The Phones output jack is a switched jack that allows us to detect when something has been plugged into it.

A switched mono jack has four pins, two of them are used as connections to handle the conventional use of the jack. The other two pins are normally-closed switches that connect to the other two pins respectively when nothing is plugged into the jack, and open when something is plugged in. This allows detection of the

jack being used without the need to rely on an external circuit. Since the switches are actuated physically they will open when any device with a ¼" connector is plugged in.

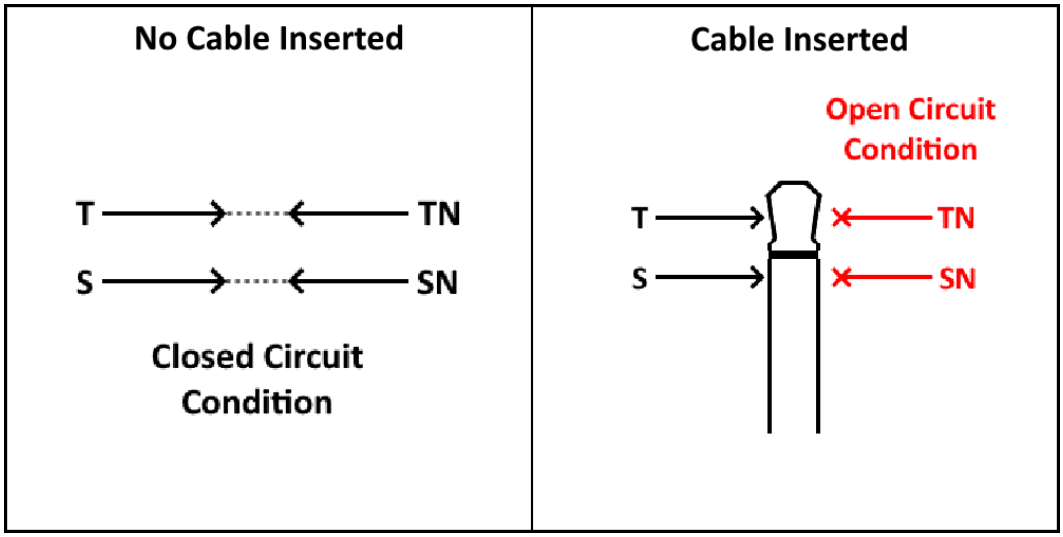


Figure 6.10 Switched Jack Functionality
Image Source: Author

The opening of the switches in the switched jack can be used to control the state of the SHND pin on the speaker amplifier. In order to shut down the speaker amplifier we need to drive the SHND pin low. To do this we must ground the pin in some way. Connecting the pin to the sleeve switch pin (SN in Figure 6.10) would allow us to ground the SHND pin, but only when nothing is plugged into the Phones jack. When something is plugged into the Phones jack the sleeve switch would open and the SHND pin would no longer be grounded. The internal pull-up resistor on the SHND pin would then enable the amplifier again. This would mute the speaker until headphones are plugged in, making it so that the headphones and internal speaker only play at the same time. This functionality would be the opposite of what is intended.

In fact, there is no way to directly use the switched pins on the Phones jack to pull the SHND pin on the speaker amplifier low. Somewhere in the flow of current, the voltage of the signal must be inverted so that the headphones and internal speaker function in a mutually exclusive way. This could involve using an inverter IC, however we can implement this simple functionality into the software instead using what we already have on the board. To do this we will need to use two GPIO pins set to digital input and digital output.

In order to make this functionality work we must first connect the sleeve switch pin of the Phones jack to a GPIO pin on the microcontroller. For this we will use Pin 42. Pin 42 will then be set as an input with a pull-up resistor. This way when nothing is plugged into the Phones jack the normally-closed switch on the jack will connect the sleeve switch to ground. This is because the negative terminal of the headphone audio pair of wires is already tied to ground.

Now, when something is plugged into the Phones jack, the switch will open and the input pin will no longer be grounded. Then the internal pull-up resistor will pull the input pin high. This means that in the software, something being plugged into the Phones jack will appear as Pin 42 going high.

It can be seen that there is a 5KΩ resistor between the sleeve switch pin and Pin 42. This resistor acts as a current limiter so that the input pin does not short to ground and damage the microcontroller. It also won't damage itself if the pin is set incorrectly. The value of this resistor has been calculated to specifically limit the current to an acceptable value while keeping the voltage seen at Pin 42 to still be considered a digital low by the microcontroller; the values used for this calculation are based off of the information in the STM32G4 datasheet.

Now to complete the amp shutdown functionality, we will need to use another pin as a digital output to drive the pin low. To do this we will use Pin 41 on the microcontroller. This pin should be set as a digital output in a push-pull configuration. Normally we would want to drive this pin high so that the amplifier is active, but the microcontroller pin can only deliver voltage up to 3.3V while the amplifier operates at 5V. This is the reason that a 1000Ω resistor has been placed between the two pins. This resistor should limit the current flowing to the output pin to only a few milliamps and keep the voltage at the shutdown pin high enough to register as enabled.

Then, the software should drive Pin 41 low when it sees Pin 42 high. This effectively functions as an inverter. In this case the voltage divider between the 1000Ω resistor and the internal pull-up resistor experienced by the SHND pin would drive the pin to a low enough voltage to disable the amplifier. The only thing other than this that needs to be done to complete this functionality would be to implement the logic in the software.

6.10 Overall Schematic

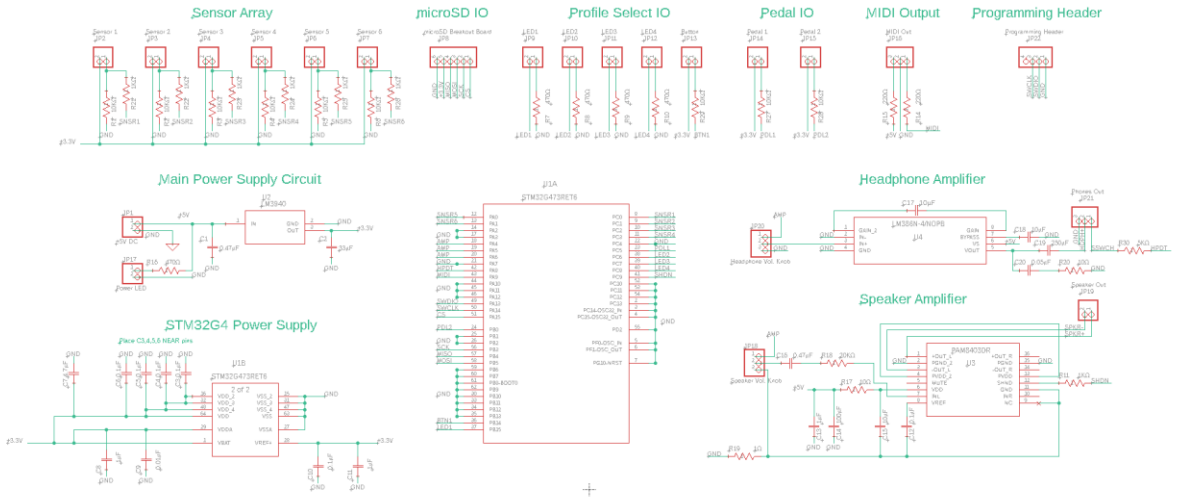


Figure 6.11 Overall Schematic
Image Source: Author

All together these components together create the overall schematic for our project. This is what will be included on the main PCB. The components will then need to be placed on the board according to their mounting style. The center of the image shows the IO pins of the microcontroller. Though it appears as if nothing is connected, the pins wires coming off them with the same label as many of the inputs and outputs on other parts of the circuit. In this way they are connected within the software but are not visually displayed in this way to reduce clutter. All unused GPIO pins are shorted directly to ground in order to reduce any issues that may arise from leaving the pins floating.

All of the pin headers shown in the circuit schematic will connect to some part of the general user IO. This includes indicators such as LEDs, the profile select button, the headphone jack, the sensor jack array. All of these things will be panel mounted and not attached to the board. This gives us more options on how to specifically customize the layout of our user interface so that it flows well and is easy to use.

6.11 Structure & External IO

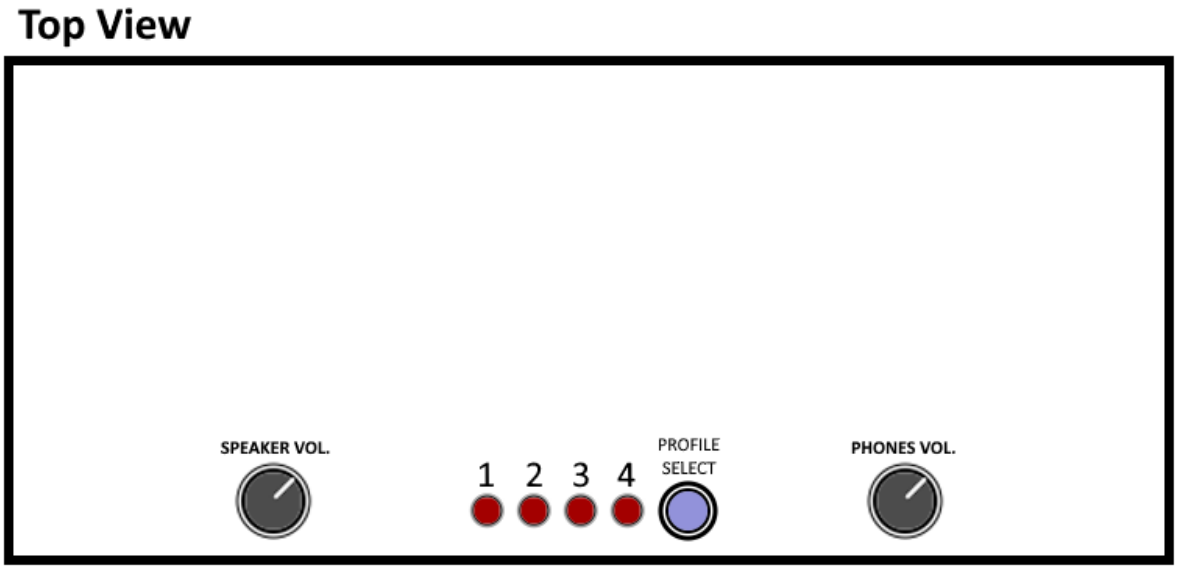


Figure 6.12 Top View of the Device
Image Source: Author

Above is a rough graphical representation of what the central processing hub will look like from above. It will take on the form of a rectangular box with various IO on the different faces of it. In the image above, the bottom of the box is the side that will be facing the user when in position. On the side near the user we can see four different LEDs that will indicate which profile is currently selected. As discussed in the earlier sections, only one of these LEDs will be illuminated at a time in an effort to not pull too much current from the microcontroller. These LEDs are on a circuit that will limit their current draw to less than 10mA. These LEDs will have visible labels above them to indicate which profile number they represent.

Near them will be a PROFILE SELECT button that will allow the user to round-robin through the different profiles by pressing the button momentarily and then releasing. This will change which profile LED is illuminated as well.

On opposing sides of the profile selection interface will be two knobs, one for individually controlling either the volume of the internal speaker or the volume of the headphones. These knobs are attached to the shafts of two 10KΩ potentiometers that are connected to the amplifier circuits of the two audio output channels. This allows the user to decide the volume for the two outputs separately so that they can adjust the volume independently to their liking. The PHONES VOL. knob should be placed somewhere near the PHONES port and should be closer to the PHONES port than the SPEAKER VOL. knob is. The SPEAKER VOL. knob should be placed somewhere near the internal speaker and should be closer to the internal speaker than the PHONES VOL. knob is. Both knobs should be clearly labeled so that the user understands which knob controls what, and they should be placed in a spot that they are not in the way of the user and are not easy to bump around and accidentally adjust unintentionally so that the user doesn't accidentally hurt their ears.

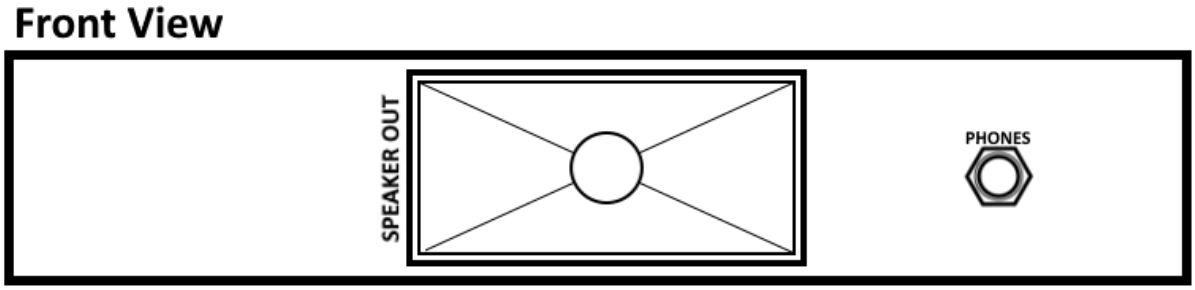


Figure 6.13 Front View of the Device
Image Source: Author

This image depicts the view from the front face of the device. In this case, the front face refers to the face that is pointing towards the user, and is what they would see in the front of the device looking head on. This face has only a few features including the speaker and the PHONES port.

The speaker is depicted by the rectangular shape in the center and is a rough estimation of what our internal speaker should look like. It is possible that the final design may involve a grill or protective covering for the internal speaker that protects it from being damaged by the user or any other foreign outside object that may puncture, damage, or dislodge it. This protective grill would still allow the sound from the speaker to be heard and escape the enclosure with minimal effect to the overall quality of the sound.

A label can be seen on the left side of the internal speaker on the diagram above. This label is optional and does not need to be included on the final design of the structure, but it can be if desired. The purpose it serves in the drawing is to denote what the rectangular shape is depicting as well as be in a position that makes it possible to be properly placed on the design as well.

The final thing that we can see on this face of the structure is the PHONES port. This port is a mono ¼” TS jack where the user can plug in any external headphones that they would like to use when practicing or recording with the device, possibly in a place where they would need to be quiet. Plugging something into this port will mute the internal speaker and only output audio through the PHONES port. The PHONES port should be placed in a location that is both easily accessible to the user and will not cause the cable for their headphones to get in the way of their movement, or get bent in an undesirable way. It should also be in a location near the PHONES VOL. knob so that the user can easily understand which knob will control its volume.

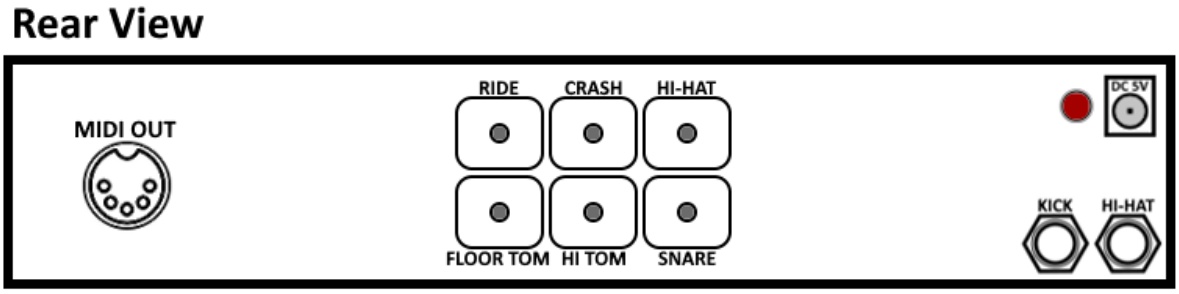


Figure 6.14 Rear View of the Device
Image Source: Author

This image depicts the rear view of the device, more specifically the face of the device that will be facing away from the user when they are using it. The majority of the IO is located here including the DC Power Input, the ¼” TS jacks for the Kick and Hi-Hat foot pedals, the array of 3.5mm jacks that will connect the sensors to the microcontroller, and the MIDI OUT port that will be used to send MIDI information to an external device.

We will begin by discussing the DC Power Input jack. This is a DC barrel plug that is used to power the overall system, it is a 2.1/5.5mm jack with a positive center that will be mounted in the top corner of one side of the rear face of the enclosure. The jack should be labeled “5V 1.5A” along with a symbol depicting the positive center and negative sleeve of the port somewhere near it so that the user knows the correct specifications of the DC power that the device needs to function correctly.

Near this DC Power Input will be an indicator LED that will turn on when the device is plugged in. This should be close to the power input to make it obvious what the purpose and meaning of the LED is. This LED will be directly powered by the DC Power Input and will not be controlled by the microcontroller. Its illumination will not indicate the function of the microcontroller but simply that power is making it into the system and onto the PCB.

Below these in the bottom corner (this would be the bottom left corner if viewed from the front face) are two ¼” TS jacks that are to be used for the Kick and Hi-Hat foot pedals. These are placed on what would be the left side to the user because the Hi-Hat is usually placed on the left within a conventional right-handed drum kit. The Kick port was also placed near here for convenience and to make

sense with the design, with the Hi-Hat placed outside as it would be further left. These ports will be used to both tell the system when to play a kick drum sound, and when to play an open or closed hi-hat sound file. When pressed down the connection is bridged and the microcontroller will see a high digital logic signal.

In the center of the rear face is the array of 3.5mm TS jacks that connect to the different components of the drum kit. Six 3.5mm TS cables will connect to the jacks and will route power to the sensors on all of the drum and cymbal enclosures. Each port will be labeled based on which component of the drum kit it connects to. The position of each port and which component it connects to is based on the usual location of each component in a conventional right-handed drum kit. Similar to the foot pedal jacks, the connections for these jacks will be bridged based on the force being used on the force sensor and give the microcontroller a reading of how hard the sensor is being pressed by the drumstick.

Lastly, on the left side of the rear (the right side from the user's reference point) is the MIDI OUT jack. This female 5-pin DIN jack will be used to connect a MIDI cable to an external 3rd-party capable of receiving MIDI information. This includes an audio interface that connects to a computer so that the MIDI information from the drum kit can be recorded in the user's DAW and used within their own music production process.

7. Software Design

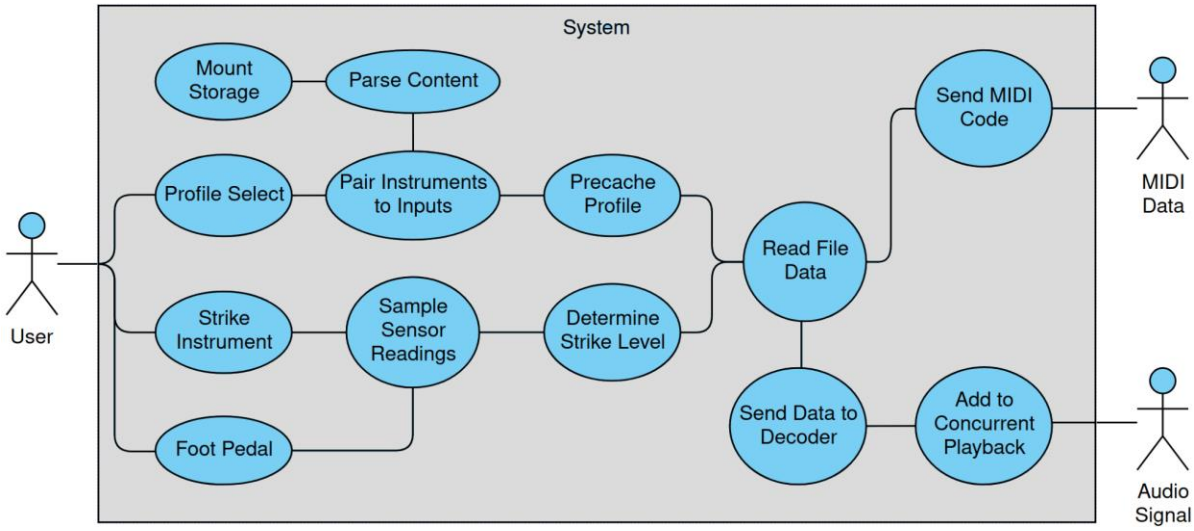


Figure 7.1 Software Use Case Diagram
By Jameson Palmer

A user interfaces with several modules of the device while several monitoring tasks are performed. A user action can be captured to registers where the task may iterate over these values. The captured value is used in branching behavior of the software to continue along the software flow. When the reading is

passed to further process the task may continue monitoring the rest of the register values. Separating the monitoring tasks from these added processing steps is meant to increase parallelization giving greater responsiveness to our array of sensors. User interaction is the driving force for changing the state of the software at any given time. After initializing the state of the device a user can interact with the instrument modules or switch profiles. Upon switching profiles the state must reinitialize with the content associated with the given profile.

Cycling through the profiles available using the profile select button can activate the precaching scheme several times which can introduce high i/o usage and delay. The precaching scheme is part of the initialization steps whenever the state is changed. The last instance of the profile switch takes precedence over other instances of the cache to ensure maximum utilization of memory resources close to the cores. The last iteration of precache should evict most or all of the contents of other profile sound files to contain large portions or the entirety of the active profile content. It is expected for a user to not immediately need the response of the instruments when the profile switches. Transferring data from the external storage is a significantly slower operation. When a user is switching profiles there is enough of a time window to perform access to external storage for the precache scheme. We expect the first usage of profile resources to be a cache miss and get transfers to external storage so a user would normally experience part of this delay regardless. Doing these i/o transfers as a batch will lead to fewer cache misses following the initialization.

The microcontroller cache policy should be set to most recently used caching schemes to give the greatest benefit of retaining useful data close to the cores. After precaching the microcontroller cache policy will determine contents of cache as the device is used with more frequent requests of data having a greater lifetime in cache. Data that is residing in cache but has gone stale or is not used as much as other rows of data will be evicted and the space is used for more frequently accessed rows. The entirety of a profile’s audio data may not be placed inside of cache. The limited memory space of the microcontroller must share its resources with the running software, codex, buffers, and the cache space. Instances of data transfer from external storage may be mitigated by retaining beginning segments of files while a retrieval is sent for the remainder. Audio files are arranged in sequence with playback times so the beginnings of these files can have access through the cache in the time necessary to retrieve following sequences.

7.1 External Storage Mounting and Parsing

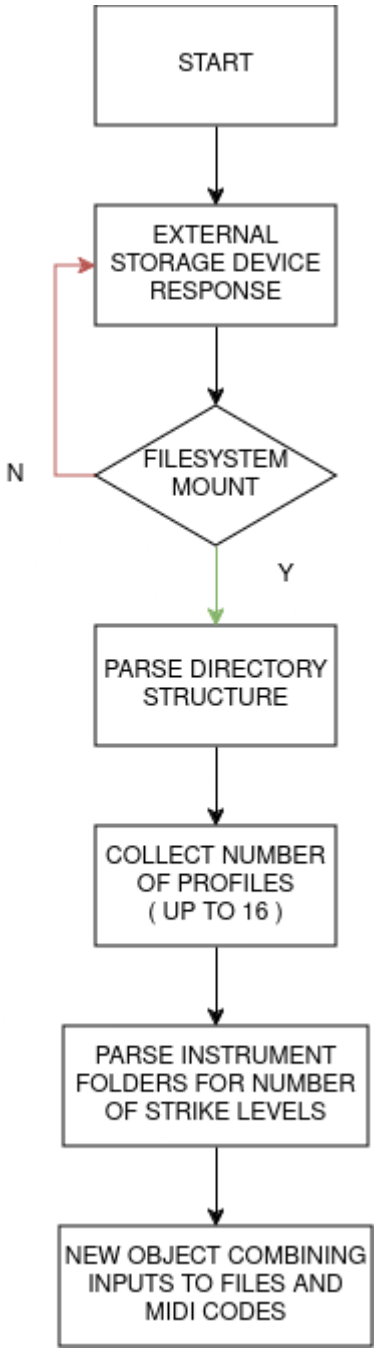


Figure 7.2 Procedure to Collect Profiles from External Storage
By Jameson Palmer

7.2 Monitoring of the Sensor Array

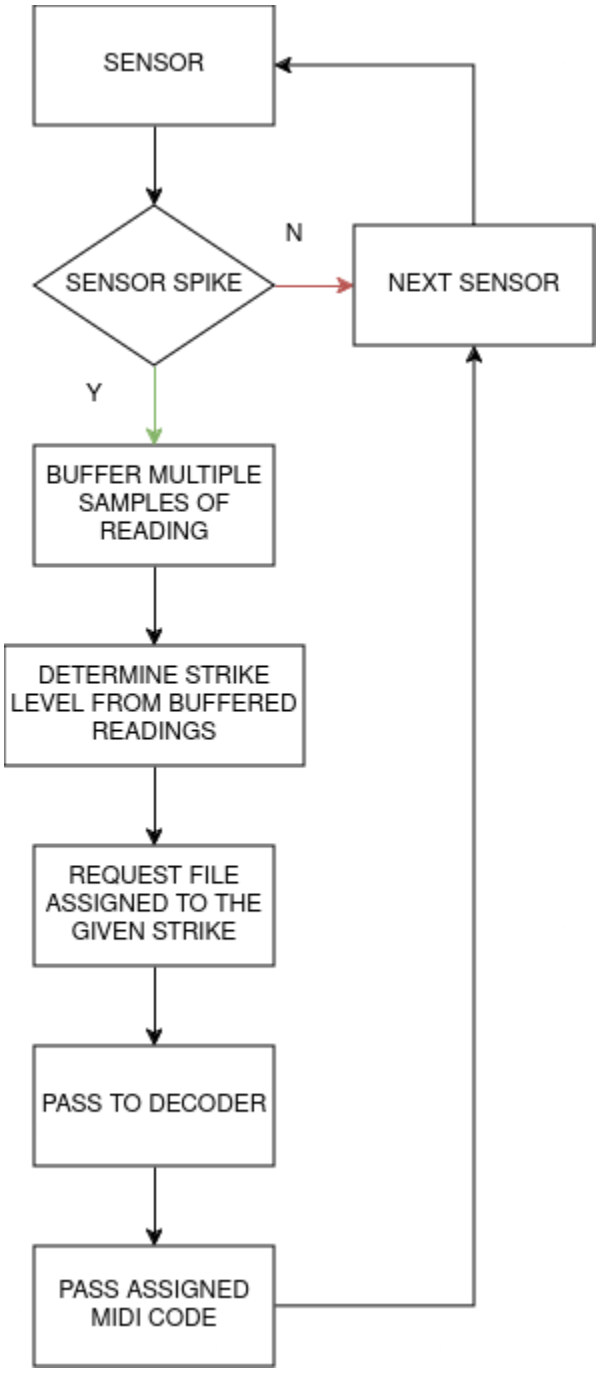


Figure 7.3 Procedure to Collect and Use Sensor Readings
By Jameson Palmer

7.3 Standby Mode, Idle

Without external storage available the device will not receive a response from the storage monitoring task. In the event the storage card is not inserted or removed the device will enter a standby mode until a response is received from an external storage device. Without storage the current mounted filesystem will be unmounted and tasks which are used for instruments will be suspended while the system awaits a response. Upon reinsertion the storage communication is then reestablished by the SPI initialization steps. The filesystem is then remounted and sent through the parsing stages. In the event a storage device is inserted that does not have a supported filesystem the software will await a properly formatted device to be present. Parsing and coursing the directories will initialize the state of our system again to ensure previous mounted data is no longer valid. A fresh state of the system is created before resuming suspended tasks.

7.4 Runtime

7.4.1 Profile Switching

Parsing the storage will present the structure of our sound library. The number of profile folders available will be determined and the first 16 profiles that are discovered will be taken. A structure array will contain the available profiles in an object-like fashion where the current profile will cycle through the array and wrap around to the first index. Whenever the profile select button is pressed the index will increment and take the information for that index to the profile setup steps. Information in each index contains the pertinent information of how many strike levels per instrument and the associated files or MIDI codes. Assignments made from previous profiles are overwritten or set NULL where applicable. This is to ensure the system state is associated to one profile.

7.4.2 Strike Level Detection

Within each profile the information of how many sound files get associated with an instrument is saved. This value is used to divide the range of sensor values evenly and assign the sound file for each range. Calibration of sensor readings will be done to determine a MIN and MAX strike to find what range of sensor readings gets divided. A strike value passing a given threshold determines the level of strike that is played. To ensure our thresholds are appropriately placed the MAX strike should have an offset so the sensor readings may pass this threshold rather than have a MAX strike lie on this threshold. Without this offset a strike which should be detected at a higher level may be read as a strike from a level below.

8. System Fabrication/Prototype Construction

8.1 PCB Design

In order to fabricate the device we will first have our PCB manufactured at JLCPCB, once the PCB has been fabricated and received we will need to solder our SMD/SMT components onto the board as well as then solder our through-hole components into the board as well. If this process is done correctly we can then go about attaching the rest of the external panel-mounted components to the main board with wires or by using removable connectors. These external panel-mounted components will be mounted in the enclosure that is created to house the parts of the central processing hub. Fabrication of this hub has not been strictly determined yet, but an example of the general idea is presented in Chapter 6.

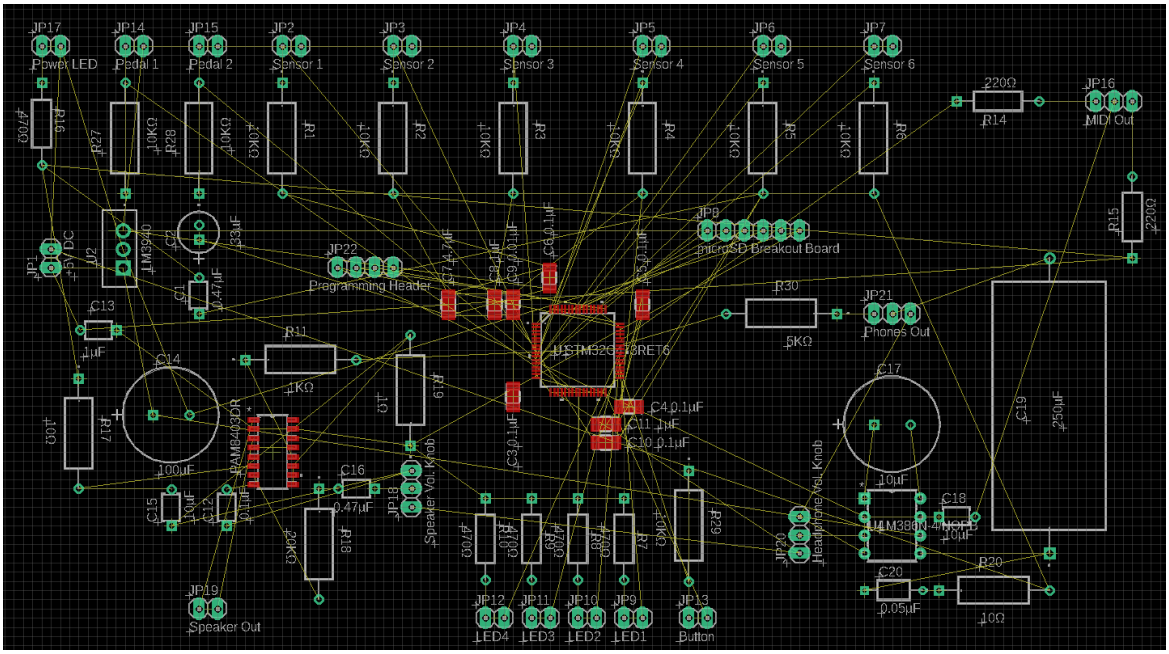


Figure 8.1 Tentative PCB Design
Image Source: Author

Shown above is a tentative PCB design that we have created. This is a rough draft of what the actual PCB will look like and where different parts will be organized. This image only shows the relative locations of the components and not the exact traces between the components. The reason for this is because as research continues it is possible that we may realize that we have made an error and either need to add, remove, or relocate one or more components. This remains a possibility as we test different components of our system independently and it is a smart idea to save money and time completing testing before actually ordering the entire PCb and placing the components.

Nonetheless this acts as a good example of the approximate dimensions of the board and also displays the size of the components relative to each other. This lets us get a good idea of generally how far apart components will need to be from each other and how much space we will need to fit all of the traces onto the board without issue.

8.2 Enclosure Design

Once fabricated, this PCB will be mounted and placed inside the enclosure. The general shape of the enclosure and position of the user IO has been decided, but the specific material that will be used to construct the enclosure has yet to be decided. It is possible to use wood or 3D print the material depending on which application is best. This will be decided at a later point after testing is complete and the electrical design has been finalized. The method for labeling the user IO also has yet to be decided but will most likely be done with a conventional label maker.

The enclosure that will be used to hold the force sensors into the drum pads and cymbals also has yet to be decided. These enclosures need to safely hold the sensors while also allowing them to receive and transmit accurate readings as the user strikes the drum kit components. These enclosures also need to be able to withstand the user striking them as they play, at least to a normal relaxed practice level of force. Lastly, they must firmly hold the sensors in place so that pulling on the cables coming out of them does not damage, disconnect, or dislodge the sensors in any way.

8.3 Prototyping

Prototype construction for hardware will be done using a basic breadboard and other components that are available to verify the function of major system components. This includes components such as the speakers and the force sensors among others. The speaker can be tested using an existing amplifier circuit or by constructing one out of our components on a breadboard, though this may prove difficult as the amplifier ICs are SMD/SMT. The force sensors can be tested by constructing a circuit that involves powering an LED and watching to see the LED turn on as the sensor is pressed down. The force sensors can be further verified using a multimeter.

Other components can be tested on a breadboard by constructing circuits in which we can vary the voltage and current through them in certain ways and use indicators to ensure we are getting the desired effects. Some components can be verified by simply measuring across them using the different functions on a multimeter. Once it is verified that all components are working, they should be verified to work together appropriately as well.

9. System Testing & Evaluation

9.1 Hardware & Software Testing

In terms of testing, hardware will be tested first in order to ensure that the parts we have ordered will function correctly when paired with our software. Since the software must be developed alongside at least some already working hardware to ensure that it is correct, it makes sense to do major hardware testing first. Some methods for this testing have been described in the section above. Hardware testing will also need to be done first to ensure that we do not actually need to reorder any components and that our circuit design is error free and does not risk damaging itself from a design flaw.

Both hardware and software design testing will be done by putting the system through a variety of test states and situations. This way we can be sure that the system does not damage itself, get stuck, or fail during any normal user situation. There is also the issue of edge cases; one thing that we will have to do is figure out what the possible edge cases of our system are and test them to ensure that if the user does something unexpected that the system will still function properly.

Software testing will involve testing for edge cases as well as testing the longevity of the program through extended use in various situations consecutively to ensure no issues such as memory leaks are present in the code. This will also test that the software is correctly interfacing with the hardware and that all external components are being addressed correctly. A specific example of something that will need to be tested is the force required to cause sounds to play with varying levels of volume/velocity. Calibrating sounds based on how hard the user generally hits the drum pads and therefore presses the force sensors will need to occur so that we can split the general force values into different levels of velocity. This will need to be done individually with each drum kit component because a user may strike them with varying amounts of strength.

9.2 Performance Evaluation

Some aspects of the performance of our project that will need to be evaluated are the output volume from the speaker, the latency between striking the drum pad and hearing a sound, and the overall power consumption of the device. These individual factors can be tested in a variety of ways and will be a good indicator of the quality of our project and how well we were able to execute our idea.

First of all, testing the speaker requires the use of a decibel meter to verify that we are reaching the desired output volume that we aimed to achieve. This volume should be able to fill a room as well as be usable and audible when other household devices are being used. It is not intended to be used in an actual live event, so we do not require that it gets loud enough to fill a small venue with the expected noise that you would encounter there, however it should be usable in most household situations. Using a decibel meter from varying distances may also help to verify that the device is performing loud enough to be usable in most household situations. For anything louder than that the headphone jack can be used to bring the sound directly to the user’s ears.

The second thing that we may want to test is the latency from the time that a drum kit component is struck and the time that a sound is played from the device. Using a specific test fixture setup may be a way to test this, as in creating a device that strikes the device and then calculates the time between when it hits the drum pad and the time that it picks up a sound using a microphone. The intent behind wanting the device to run at a low latency is so that the user does not get confused when playing it. When an instrument is being played, if there is a long amount of time between when a note is struck and when it is heard it can become disorienting for the person playing it. A good way to simulate this is using a MIDI controller with a DAW on a computer and digitally adding time between when a MIDI note is detected and when a sound is played. Generally, latency over 100ms becomes noticeable and disorienting, but anything further below that is not too much of an issue. As long as we are able to keep this latency within a comfortable range it should not be too detectable by a user. The goal with latency reduction is moreso to reach a level of ease and comfort than it is to reach a quantified performance metric.

The last testable metric we have to evaluate performance is the overall power consumption of the device. This can be tested using a variety of different devices. We can use a tool such as a digital multimeter to test the current passing through the DC Power Input of the device and multiply that by the +5V of input power that we will be using. We can also use a device called a Kill-O-Watt to directly measure the wattage of power being pulled from the wall before our AC/DC power converter. We really shouldn’t have too much to worry about here with making our goal of using under 15W because of the fact that our overall circuit shouldn’t consume that much power in general. However, this would still remain a valuable metric.

9.3 Overall Integration

The end goal for the project is to create a drum kit consisting of a central processing hub and drum kit components as peripherals. Once powered, the microcontroller will read incoming analog voltage signals from the force sensors that will rise when struck. Once these signals rise to a certain threshold, the microcontroller will poll the sensor and determine the value of the impact based on the maximum force value that it sees. It will then play a sound file with a velocity according to the received voltage value. This sound will be played out of either an internal speaker or to headphones that the user will have plugged into the device. It will also output a MIDI signal for each of the notes that are played. The device will also use foot pedals to determine when the kick drum is hit and whether or not the hi-hat is open or closed. A microSD card will contain the sound files to be used by the program, and there will be multiple different drum sound profiles so that the user can play sounds of a preferred style.

9.4 Future Plans

This is the first half of a two-semester long project where we have completed the initial design of our device after having come up with an idea and doing research to figure out how to make it a possibility. Our future plans for the next semester involve testing our hardware and making sure that it works so that we can get our PCB manufactured. Once we have the PCB we will then develop the software to control our system alongside the process of developing the enclosures and fixtures to hold the different parts of our system. Most of the hard work along the way will be figuring out how to implement solutions as well as troubleshooting any problems we may run into.

10. Administrative Content

10.1 Budget Estimates

This project is being funded by the group members involved in designing and making it. The table below provides estimates for the cost of the materials required to complete the project. The project aims to create a low cost solution to the problem of creating a functional entry-level electric drum kit.

Part	**Estimated Cost**
PCB	$30
Microcontroller	$3
IO Jacks	$3
Main Housing	$20
Sensors	$10
Pedals	$15
Speaker	$3
Amplifier Circuitry	$3
Power Delivery Circuitry	$5
Connecting Cables	$15
Other Components (Indicators, Buttons)	$3
Storage Device	$5
Total	**$115**

Table 10.1 Estimated Budget

10.2 Parts List/BOM

Categorized Parts List		
Component	Value	Qty.
MCU Section		
MCU	STM32G473RET6 LQFP64	1
Capacitor	0.01µF	1
Capacitor	0.1µF	5
Capacitor	1µF	2
Capacitor	4.7µF	1
Speaker Amp		
IC	PAM8403	1
Potentiometer	10KΩ	1
Hardware	Potentiometer Knob	1
Capacitor	0.47µF	1
Capacitor	0.1µF	1
Capacitor	1µF	1
Capacitor	10µF	1
Capacitor	100µF	1
Resistor	1Ω	1
Resistor	10Ω	1
Resistor	20KΩ	1

Speaker	HY7040DS0830-H16.8	1
Headphone Amp		
IC	LM386	1
Connector	Switched ¼” TS Jack	1
Potentiometer	10KΩ	1
Hardware	Potentiometer Knob	1
Capacitor	0.05µF	1
Capacitor	0.1µF	1
Capacitor	10µF	1
Capacitor	250µF	1
Resistor	10Ω	1
Resistor	10KΩ	1
MIDI Output		
Connector	Female 5-Pin DIN	1
Resistor	220Ω	2
Sensor Array		
Sensor	MF01A-N-221-A01	6
Hardware	Foot Pedal	2
Resistor	10KΩ	8
Connector	3.5mm TS Jack	6
Hardware	3.5mm TS Cable	6
Connector	¼” TS Jack	2
Power Supply		
Hardware	AC/DC Converter +5V DC	1

	1.5A	
Connector	Female DC Barrel Jack	1
Regulator	LM3940	1
Resistor	470Ω	1
Capacitor	0.47µF	1
Capacitor	33µF	1
Indicator	5mm/3mm LED	1
Profile Selection		
Hardware	Momentary Button	1
Indicator	5mm/3mm LED	4
Resistor	470Ω	4
Resistor	10KΩ	1
microSD Card Interface		
Hardware	microSD to SPI Breakout Board	1
Hardware	microSD Card	1
Connector	6-Pin Male Header	1
Connector	6-Pin Female Header	1
Programming Header		
Connector	4-Pin Male Header	1

Table 10.2 Categorized BOM

Condensed BOM				
Component	Value	Notes	Qty.	Total Cost
Capacitor	0.01µF	SMD/SMT	1	$0.10

Capacitor	0.1µF	SMD/SMT	5	$0.50
Capacitor	1µF	SMD/SMT	2	$0.17
Capacitor	4.7µF	SMD/SMT	1	$0.27
Capacitor	0.05µF	Thru-Hole MLCC	1	$0.36
Capacitor	0.1µF	Thru-Hole MLCC	2	$0.36
Capacitor	0.47µF	Thru-Hole MLCC	2	$0.62
Capacitor	1µF	Thru-Hole MLCC	2	$0.66
Capacitor	10µF	Thru-Hole MLCC	1	$0.54
Capacitor	10µF	Electrolytic	1	$0.57
Capacitor	33µF	0.33Ω - 0.5Ω ESR Electrolytic	1	$0.24
Capacitor	100µF	Electrolytic	1	$0.26
Capacitor	250µF	Electrolytic	1	$3.26
Resistor	1Ω	Thru-Hole	1	$0.03
Resistor	10Ω	Thru-Hole	2	$0.05
Resistor	220Ω	Thru-Hole	2	$0.03
Resistor	470Ω	Thru-Hole	5	$0.06
Resistor	1KΩ	Thru-Hole	1	$0.10
Resistor	5KΩ	Thru-Hole	1	$0.10
Resistor	10KΩ	Thru-Hole	10	$0.50
Resistor	20KΩ	Thru-Hole	1	$0.05
Potentiometer	10KΩ	Panel Mount	2	$0.78
IC	PAM8403	PAM8403DR	1	$1.13
IC	LM386	LM386N-4	1	$1.52
Regulator	LM3940	TO-220 Package	1	$1.62
Connector	6-Pin Male Header	None	1	$0.08

Connector	6-Pin Female Header	None	1	$0.07
Connector	4-Pin Male Header	None	1	$0.05
Connector	Female DC Barrel Jack	Panel Mount	1	$0.55
Connector	3.5mm TS Jack	Panel Mount	6	$1.50
Connector	¼” TS Jack	Panel Mount	2	$0.90
Connector	Female 5-Pin DIN	Panel Mount	1	$2.00
Connector	Switched ¼” TS Jack	Panel Mount	1	$0.49
Speaker	HY7040DS0830-H16.8	Panel Mount	1	$1.75
Hardware	Potentiometer Knob	Volume Control	2	$1.18
Hardware	3.5mm TS Cable	3ft Min.	6	$20.00
Hardware	AC/DC Converter +5V DC 1.5A	+5V 1.5A Inner + | Outer -	1	$7.99
Hardware	microSD to SPI Breakout Board	6-Pin	1	$2
Hardware	microSD Card	FAT32	1	$10
Hardware	Momentary Button	Panel Mount	1	$0.25
Hardware	Foot Pedal	None	2	$15.00
Indicator	5mm/3mm LED	Thru-Hole	5	$0.15
Sensor	MF01A-N-221-A01	None	6	$23.70
MCU	STM32G473RET6	LQFP64 Package	1	$8.20
Total				**$109.74**

Table 10.3 Condensed BOM

10.3 Milestones

In order to stay on track our team has set certain milestones in accordance with what is due throughout the course. Designing and testing is hard to foresee accurately and will probably vary in reality compared to what is listed in this table.

Hard deadlines should stay the same unless changed by the SD1 coordinators. These milestones cover only SD1 and up to the end of the Fall 2024 semester.

Time	Milestone
Week 3	Finish D&C document
Week 4	Meet with SD coordinators and begin Draft Paper
Week 5	Research specific parts, refine theoretical design
Week 6	Work on draft paper, select final components and begin design
Week 7-9	Testing components, rough PCB design, draft paper
Week 10	Submit first draft paper, continue testing/designing
Week 11-12	Submit draft paper revision
Week 13-14	Submit final draft
Week 15	Submit mini demo video

Table 10.4 Milestone Schedule

10.4 Work Distributions

Task	Primary	Secondary
Kit Assembly and Housing Design	Wylde	Soufiane
Instrument Sensor and Housing Design	Soufiane	Wylde
Sound Library Storage	Soufiane	Jameson
Connection between Sensors and Control Board	Soufiane	Wylde

Strike Level Detection	Jameson	Wylde
Sound Library Instrument Selection	Jameson	Soufiane
Sensor Instrument Assignment	Wylde	Jameson
Sound Library Mode Switching	Jameson	Soufiane
Auxiliary port passthrough	Wylde	Soufiane
MIDI cable data path	Jameson	Wylde
Control Board PCB Layout	Wylde	Jameson
Power Supply	Soufiane	Jameson
Control Board Software	Jameson	Wylde

Table 10.5 Work Distributions

Declaration: We hereby declare that we have not copied more than 7 pages from the Large Language Model (LLM). We have not utilized LLM for any purpose at this time.

11. Conclusion

Over the course of this paper we have gone from the concept of a project to doing full research on how to execute it, then choosing the hardware that we need in order to create it, and finally designing a circuit and the concepts of a physical structure and software program to drive it. This has required extensive research and great amounts of time to understand, and the project still has yet to be proven to function. Although we have done so much work on this project, we are still only halfway through the entire process and still have a whole semester to finalize and construct the design as well as get it to a functional and presentable state.

The next semester will see us work on our program and test it on the actual hardware we are going to be using, as well as finishing up the final PCB design and having it manufactured. During this process we will be figuring out how to construct the fixtures that will be used for the drums and the enclosures that will hold the sensors in place so they do not get damaged or disconnected. This will take place over the course of a few months and we are sure to run into some problems during this time. With the knowledge that we have acquired over this past semester we will hopefully be able to split up our time more wisely in terms of having a safety net in case something goes wrong. Previously we did not know how much time to allocate to specific parts of the research, however now we have

done an analysis of what constraints we have to deal with and how they will affect us.

One of the hardest parts of the research and design process was figuring out specifically which microcontroller to use for the project. We needed to cross reference many different types of microcontrollers and their capabilities to see which one had all of the features we would need. This also included doing research to figure out how microcontrollers perform certain tasks. Once we understand how they perform those tasks we are then able to search for microcontrollers with the required specifications to do those things.

Another hard part of the project was figuring out how to get specific inputs and outputs into and out of the microcontroller safely. This mainly manifested as protecting the microcontroller from sourcing or sinking too much current. This required deep looks at the datasheets and manuals for the microcontroller in order to determine its capabilities. Then we had to design circuits around these capabilities to ensure that our circuit was within the specifications. Even now as we realize our circuit we still need to do real world testing to determine if we were correct in our calculations.

We predict that the next semester will be easier than this one now that our idea is off the ground. Now that we understand what exactly our goal is, we have a lot more direction within the project and should be able to lay out our goals and get to them early on rather than having to figure out what it is that we need to do in order to realize any of our ideas.

This semester has been very informative in showing us what it is like to work with other engineers, especially ones from other disciplines. Dividing tasks so that everybody can focus on their strengths has been useful, however more issues arrive when we must connect these ideas. This has taught us why it is so important that engineers have some baseline knowledge within other disciplines so that communication of ideas is easier and so that we have an idea of what is actually possible in each other’s fields. Otherwise it is possible that one of us may have an idea for how to implement something and design our part of the project around that, only to later realize that it isn’t even a feasible implementation.

Overall, we are confident with how our project has ended up and are ready to enter the next semester and finalize our design while accomplishing what is needed so that we can verify our implementation. With our newfound knowledge, we should be able to organize ourselves in an effective way in order to get done what needs to be done in a more timely manner.

Appendix

References:

42 Piezo Buzzer Images, Stock Photos, 3D Objects, & Vectors | Shutterstock, Shutterstock, www.shutterstock.com/search/piezo-buzzer.

Campbell, Scott. “Basics of the SPI Communication Protocol.” *Circuit Basics*, circuitbasics.com, 14 Nov. 2021, www.circuitbasics.com/basics-of-the-spi-communication-protocol/.

EmbeTronicX. “SD Card Interfacing with STM32 - STM32 SPI.” *YouTube*, YouTube, 3 Oct. 2022, www.youtube.com/watch?v=xbLkp_pTWbo.

“General MIDI Drum Notes.” *MIDI Drum Chart*, zendrum.com, www.zendrum.com/resource-site/drumnotes.htm.

“Headphones v1.8 - Graph Tool.” *RTINGS.Com*, rtings.com, www.rtings.com/headphones/1-8/graph/25500/raw-fr-l/audio-technica-ath-m50x/295.

“How to Interface SD Card Using SPI in STM32.” *ControllersTech*, controllerstech.com, 24 Sept. 2024, controllerstech.com/sd-card-using-spi-in-stm32/.

“Lm386.” Texas Instruments, https://www.ti.com/lit/ds/symlink/lm386.pdf?ts=1732638357489

“Lm3940.” Texas Instruments, https://www.ti.com/lit/ds/symlink/lm3940.pdf

“PAM8403-247318.” Diodes Incorporated, https://www.mouser.com/ds/2/115/PAM8403-247318.pdf?srsltid=AfmBOoq5JOmVQgSXmgj7009gIm42bYmWdBN9q8-EsHbHalkg35UzcRLQ

Predictable Designs. “Tutorial: How to Design Your Own Custom Microcontroller Board.” *YouTube*, YouTube, 15 May 2018, www.youtube.com/watch?v=tgUx0Gm7-RM.

Predictable Designs. “Tutorial - How to Design Your Own Custom Microcontroller Board (Part 2).” *YouTube*, YouTube, 22 May 2018, www.youtube.com/watch?v=jN6rc2Zd-h8.

“R/Askelectronics on Reddit: What Does This RC Series Do in This Amplifier? Does It Change the Sound If I Vary These Values?” *Reddit*, reddit.com,

www.reddit.com/r/AskElectronics/comments/wjc8md/what_does_this_rc_series_do_in_this_amplifier/.

“Rm0440-Stm32g4-Series-Advanced-Armbased-32bit-Mcus-Stmicroelectronics.” STMicroelectronics,

Saleem, Kiran. “Simple PAM8403 Audio Amplifier Circuit.” *Circuits DIY*, circuitsdiy.com, 30 July 2024, www.circuits-diy.com/simple-pam8403-audio-amplifier-circuit/.

Science Buddies. “How to Use a Force Sensor with an Arduino (Lesson #23).” *YouTube*, 25 Sept. 2023, youtu.be/r7oWtcE6QQc?si=Yjr-jmNfqbtknLAw.

self-study electrical engineer. “SD CARD READER With STM32.” *YouTube*, 4 Dec. 2020, youtu.be/fRfRwMyNE7A?si=XeSMHynNSSWSymXb.

“SPI (Serial Peripheral Interface).” *SPI (Serial Peripheral Interface) - UTAT Space Systems Documentation 1.0 Documentation*, UTAT Space Systems, utat-ss.readthedocs.io/en/master/communication-protocols/spi.html.

“Stm32g473cb.” STMicroelectronics, https://www.st.com/resource/en/datasheet/stm32g473cb.pdf.

“STM32L562E-DK.” *STMicroelectronics*, www.st.com/en/evaluation-tools/stm32l562e-dk.html#st_all-features_sec-nav-tab.

“The MIDI Specification.” *MIDI Specification*, people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2000/selan/midi.html.

RTINGS Terms of Use:

3.2 ACCESSING THE MATERIAL

User is entitled to access and print, all material published on the Site, including without limitation documentation, text, pictures, sounds, graphics, articles, video’s and audio clips (collectively the “Material”) for User’s personal private and non-commercial use only, provided that User does not:

(a) download or print any Material in a systematic or regular manner to create a database (electronic or paper form);

(b) remove any notices relating to the ownership of copyright or other intellectual property rights in the Material;

(c) modify, translate, reverse engineer, reproduce, decompile, disassemble or create derivative works of any of the Material;

(d) rent, lease, sub-license, loan, copy, commercially exploit or give or transfer any rights in the Material in any form, to any person or entity without RTINGS prior written consent.

RTINGS status as the owner of the Material must always be acknowledged. If User prints off or otherwise copies any part of the Site or Material in breach of these Terms, User’s rights to use the Site will cease immediately and User must return or destroy any copies of the Material.