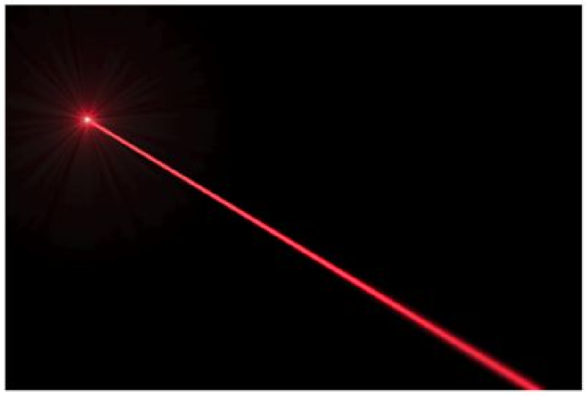
Laser Guitar Instrument



+

Final Document

Group 1

Alexander Truong-Mai - Computer Engineer

Jacob Legler - Photonic Science and Engineering

Jonathan Spurgeon - Electrical Engineer

Juan Gamero - Computer Engineer

Table of Contents

Indexes

List of Figures

List of Tables

1. Executive Summary

Our inspiration for the Laser Guitar Project came from a senior design project done in the past by a group of Electrical Engineering and Photonics students titled “Laser Musical Instrument”. This project was intriguing to our group as bringing the world of engineering into the world of music offered a wide range of creative opportunities. The Laser Musical Instrument designed by the previous group was modeled after an Optical Theremin, which played sound that changed in pitch depending on the intensity of the light reflected off of the user. While this project did succeed in creating a laser musical instrument, the Theremin is not a very popular instrument, and likely would not appeal to a large margin of people. Their design was also monophonic, which prevented the user from playing chords, which are essentially the building blocks of modern music. While brainstorming our group came up with the idea of designing a laser musical instrument that would appeal to a larger audience. This presented us with the following challenge: To design a laser musical instrument that can be played similarly to more popular stringed instruments such as the guitar, and provide a more instrumental and musical experience for the user than other laser instruments found in today’s market.

In order to accomplish this, the instrument should be polyphonic so that the user is able to play chords. This opens up the opportunity for the user to play songs as if they were using a real guitar. This would provide an investment opportunity for musicians everywhere, as an affordable laser instrument that is housed in the form of an instrument as popular as the guitar has never been made before. Also, there is an appeal to people who do not play guitar, but have always wanted to learn. Learning to play guitar can be a painful experience, as pressing down on metal strings hard enough to play a note can be both difficult and strenuous on the player's fingers. By eliminating the need for metal strings, the laser guitar will draw an audience of beginner musicians who wish to learn guitar without the pain of pressing down on metal strings. Another issue that this tackles in the world of music is the price of today’s instruments.

Most high end electric guitars run for hundreds of dollars, not including necessary equipment such as pedals and amplifiers. This is a major roadblock for people who may want to get into music, but do not have sufficient funds to purchase an instrument. Because this involves generally cheap resources, as photodiodes and laser diodes often run for less than ten dollars, we have an opportunity to appeal to an even larger market of musicians by designing an instrument that is far cheaper than what it costs for most guitars today, provided we properly manage our budget. Finally, the instrument should sound good. While this may sound obvious, it would be entirely possible to play a simple sine wave at the proper frequencies of a musical scale, however this would not sound good. Instead, we should sample actual instruments and have our speakers play these samples so that our instrument sounds like an acoustic or electric guitar.

Marketing strategies could potentially present our product as a futuristic musical instrument that is fun for beginner and advanced musicians and runs for a cheaper price than anything like it on the market. Our team consists of two Computer Engineering majors, an Electrical Engineering major, and a Photonic Science and Engineering major. By allowing our computer engineers to focus on the software development of our project, while our Electrical and Photonic engineers focus on the hardware of the project, we are provided with a well balanced team capable of tackling this project in a creative, musical, and financially sound manner.

2. Project Description

This section will give a high level overview of our project. It will cover what exactly we are trying to accomplish and why, as well as some basic diagrams and specifications we hope to achieve. Along with everything, there will be detailed goals set forth and milestones for us to accomplish.

2.1. Project Motivation and Goals

For years, instruments have been used as a form of entertainment and storytelling. There are a wide range of instruments from the violin to the guitar, and even older ones that indigneous people used such as the Didgeridoo. Making music today involves more digital systems and softwares to mimic traditional sounds from instruments. Guitars though are still widely used in bands and solo artists during concerts; however, learning the guitar is a struggle for many beginners due to the fact that it can be physically painful to play guitar strings with your bare hands over a period of time. With today's technology, we can solve this problem by removing the strings. By removing guitar strings entirely, we can replace them with lasers. Using lasers as the strings of the guitar will provide for a user friendly, fun, innovative, and new musical experience for musicians everywhere.

The motivation for this project is to develop an instrument with new, unique features that uses the knowledge we have gained from our studies at the University of Central Florida (UCF). This instrument will provide a way for beginners to learn to play the guitar without the painful struggle of fretting strings for the first time, and also have the feel and musical capabilities of a normal guitar. With our instrument, we will also satisfy the requirements for Senior Design 1, and develop an open-ended solution to the advancements and visual effects we have for our guitar.

For the goals of our project, we will have three sectors separating them by short term, long term, and stretch goals. Starting with the long term goal, this consists of one main goal to achieve an accurate sounding guitar with lasers acting as the guitar strings. Building up to this, there are short term goals; consisting of implementing the laser diodes, build design, photodetectors, software implementation, printed circuit board layout, etc. The end objective is an accurate, portable guitar that is reasonably priced for musicians or the general public that are interested in purchasing it. The stretch goals consist of ambitious ideas that can be implemented but not necessary for the design as a whole; sustaining notes being played based on frequency of strums.

As for implementing each necessary component, there will be two sets of strings consisting of either three or four laser diodes. The first set will be across the head of the guitar and the second across the neck. For the head of the guitar, the software implementation will detect which laser will be broken from the photodetector. Along the neck, the amount of reflected light at the point the laser is broken by the player's finger back to the photodetector will determine which note to play. To tie all of these strings together, a speaker and battery will be chosen with research to fit our needs and a microcontroller will be chosen from a list of researched ones with the necessary features. The printed circuit board layout will be designed with the microcontroller, ADC, DC-DC converter, etc. The idea for this project is based on a previous senior design project from UCF called "Laser Musical Instrument". The features and new software/hardware implementations we will be adding will help our design look and mimic an actual guitar.

2.2. Requirement Specifications

The design of our instrument will result in a stringless guitar that uses laser light as the strings. The user requirements include the size and feel of the instrument, and general eye safety for the user. To make an instrument that feels natural to play, the size should be no longer than that of a bass guitar, and the spacing between frets should be reasonably small so that the user can play comfortably. Eye safety will be accounted for by using nothing more powerful than a class two laser with proper eyewear included if necessary. In terms of engineering requirements, we must ensure that the correct notes are being played based on which laser is being blocked by the user's finger, and the returned intensity signals the microcontroller to play a certain note. This note will be stored and played when the corresponding string is strummed on the strumming set of strings. To do this, two separate sets of laser systems will be designed.

The first is a strumming system. This system simply tells the microcontroller to play a note when the beam is interrupted. A stretch goal of the system would be to determine the volume and sustain of the note played based on the velocity of the strums. The second system is the fretting system. This system uses a series of photodiodes in conjunction with a laser diode to determine what note will be played by the strumming system. The photodiodes will be placed along the neck of the guitar in series so that they carry the same current but different voltages. The voltage collected by each individual photodiode will be either high or low, depending on the amount of light incident on the photodiode. This amount of light can be controlled by the user by interrupting the laser beam that is fired across the fretboard. In terms of project requirements, this means we need to have a system which can fire a laser beam in a controlled area of space and collect any light reflected by the user's finger. The system should react to this action fast enough so that the act of playing the instrument feels natural to the player, and the system should be able to work under settings of various lightings.

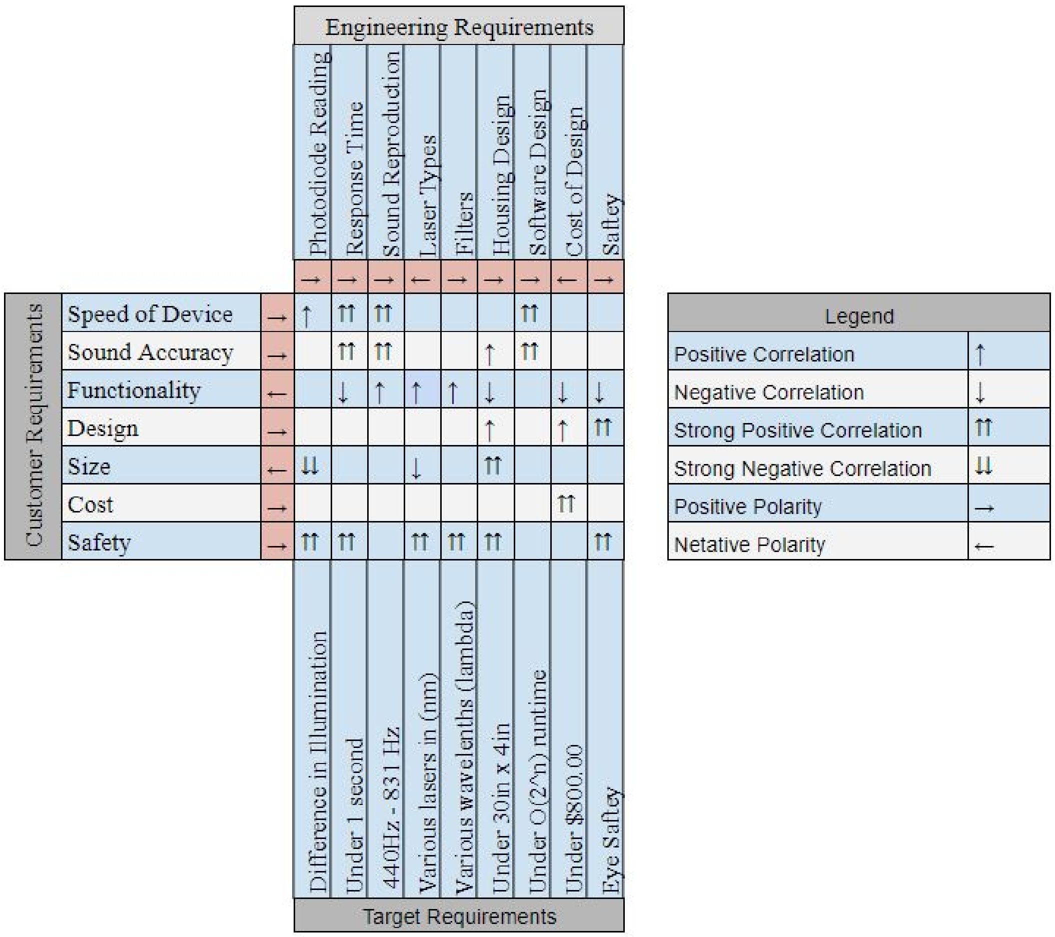
Next, we must implement this idea across several strings. To prevent noise from adjacent strings, we are given the choice of a few different options. First, we can consider using narrow band filtering to only collect light from the specified wavelength of our laser diodes. In this case, each photodetector will be paired with a bandpass color filter which only allows light of the respective string's wavelength to pass on to the photodiode. The laser diodes will use light in the visible spectrum, with wavelengths of roughly 450nm (Blue), 550nm (Green), and 633nm (Red), so they can be easily filtered. This will also provide a more intuitive experience for the user as each string will essentially be color coded. The filters should also help with ambient light, as a good portion of the light will be filtered out of the system. This idea would function well under various lighting conditions, and would prevent the noise from adjacent strings as intended, however narrow band filtering is an expensive process, and may not be the best option to consider. Another option would be to modulate our laser diodes and detect this modulation using the series of photodetectors. This would allow us to differentiate between adjacent strings by varying the rate of modulation of our laser diodes, and because it is very unlikely any ambient light will be modulating at the same frequency we are trying to detect, this will also allow us to solve the problem of ambient light. Finally, we can potentially design our housing in such a way that light can only come in from in front of the photodiodes. This is the cheapest option as it only requires us to modify the physical housing of the instrument and does not require any further electrical equipment. We will initially test different housing options with multiple strings and make our decision on whether or not to modulate the strings after our initial testing. Due to the cost of narrow band optical filters, this option is likely off of the table.

Requirements	**Description**
Size	Bass Guitar or smaller (<36”x4” Fretboard)
Weight	< 5lbs
# of Strings	4
Notes Per String	4 (Open Strings + 3 Frets)
Polyphony	4 Notes at once
Frequency of Notes Played	Proper Spacing between notes based on musical scale
Safety	Class 2 Lasers
Responsivity	Able to detect which fret and string is being held down
Response Time	< 50 ms

Table 1: Project Requirement Specifications

2.3. Quality of House Analysis

The House of Quality diagram below displays the desires and needs of the customer while satisfying engineering specifics.The target audience for our design will ultimately be anyone who enjoys music, and more specifically to those with an interest in guitar. In order to meet the expectations of our users, there is a list of user requirements to consider; speed, sound accuracy, design, cost, size and functionality. The speed of the device when the user plays notes and pitches should output sound promptly. The sound accuracy of the device is something customers will look for to gain the satisfaction of hearing their notes and pitches played out correctly. User’s also have specific desires when it comes to the design, size and cost of the instrument, and thus these requirements are taken into consideration. Lastly, the functionality and the safety of the device will be essential to customers. Some user’s want different functionalities such as the whammy bar and volume knob, while some may not need it. Safety is included in this list due to the usage of lasers, and therefore, it is essential that the user’s feel safe when using the instrument.



			Engineering Requirements								
			Photodiode Reading	Response Time	Sound Reproduction	Laser Types	Filters	Housing Design	Software Design	Cost of Design	Saftey
			→	→	→	←	→	→	→	←	→
Customer Requirements	Speed of Device	→	↑	⇈	⇈				⇈		
	Sound Accuracy	→		⇈	⇈			↑	⇈		
	Functionality	←		↓	↑	↑	↑	↓		↓	↓
	Design	→						↑		↑	⇈
	Size	←	⇊			↓		⇈			
	Cost	→								⇈	
	Safety	→	⇈	⇈		⇈	⇈	⇈			⇈
Target Requirements			Difference in Illumination	Under 1 second	440Hz - 831 Hz	Various lasers in (nm)	Various wavelenths (lambda)	Under 30in x 4in	Under O(2^n) runtime	Under $800.00	Eye Saftey

Legend	
Positive Correlation	↑
Negative Correlation	↓
Strong Positive Correlation	⇈
Strong Negative Correlation	⇊
Positive Polarity	→
Netative Polarity	←

Figure 1: House of Quality

Target engineering requirements for our project include the functionality and responsiveness of the device. In terms of hardware, each of the photodiodes used for measuring the returned intensity should be able to detect the difference in intensity from one distance on the fretboard to the next in order to return a signal to the microcontroller that tells the program what note to play. The notes that are able to be played should all be within frequencies in the chromatic scale, so that they sound musical. Similarly, the strumming strings must detect the presence of the user's finger in order to tell the microcontroller when to play the selected notes. All detections should work under both dark and light conditions, the device should be small enough to be portable, while the spacing between strings should be large enough so that the player's fingers do not get in the way of each other when selecting which string to play. From the software perspective, a program must be written that selects which note to play based on the returned intensity of the light reflected off the user's fingers. This note will not be played by the speakers until the user strums the second set of strings, called the strumming strings. Lastly, eye safety is a big factor to consider. This can be achieved by containtaing the laser within the housing of our design. Also labels should be included to tell the customer to not look into.

2.4 Block Diagram Overview

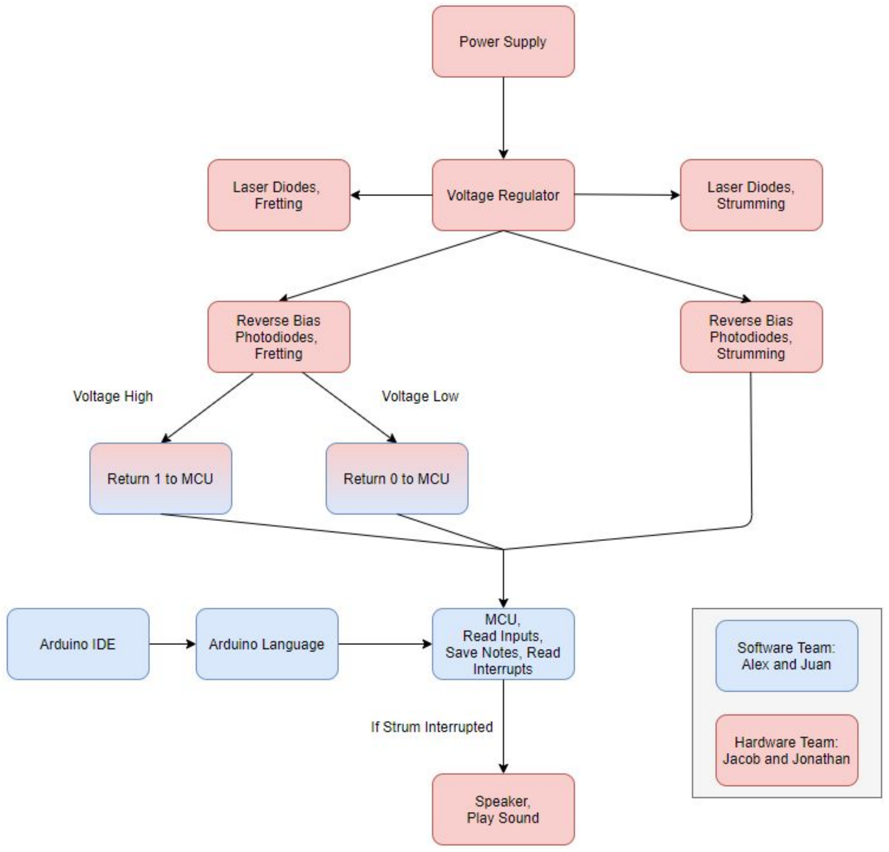


Figure 2: Block Diagram Overview

In the block diagram overview, the modules and division of tasks given to each individual member of the group based on their respective colors. The blocks were made by looking at the objectives and requirements of the project. Each module represents a function of the system needed to complete the project. The specific functionality of each of the blocks is described in further detail in the following sections.

2.4.1 Hardware Component Block Diagram

From Figure 2, Figure 3 only focuses on the hardware components of the block diagram. By focusing on only the hardware of the block diagram, this gives the group a one sided perspective of how the hardware components should be connected and be taken into consideration.

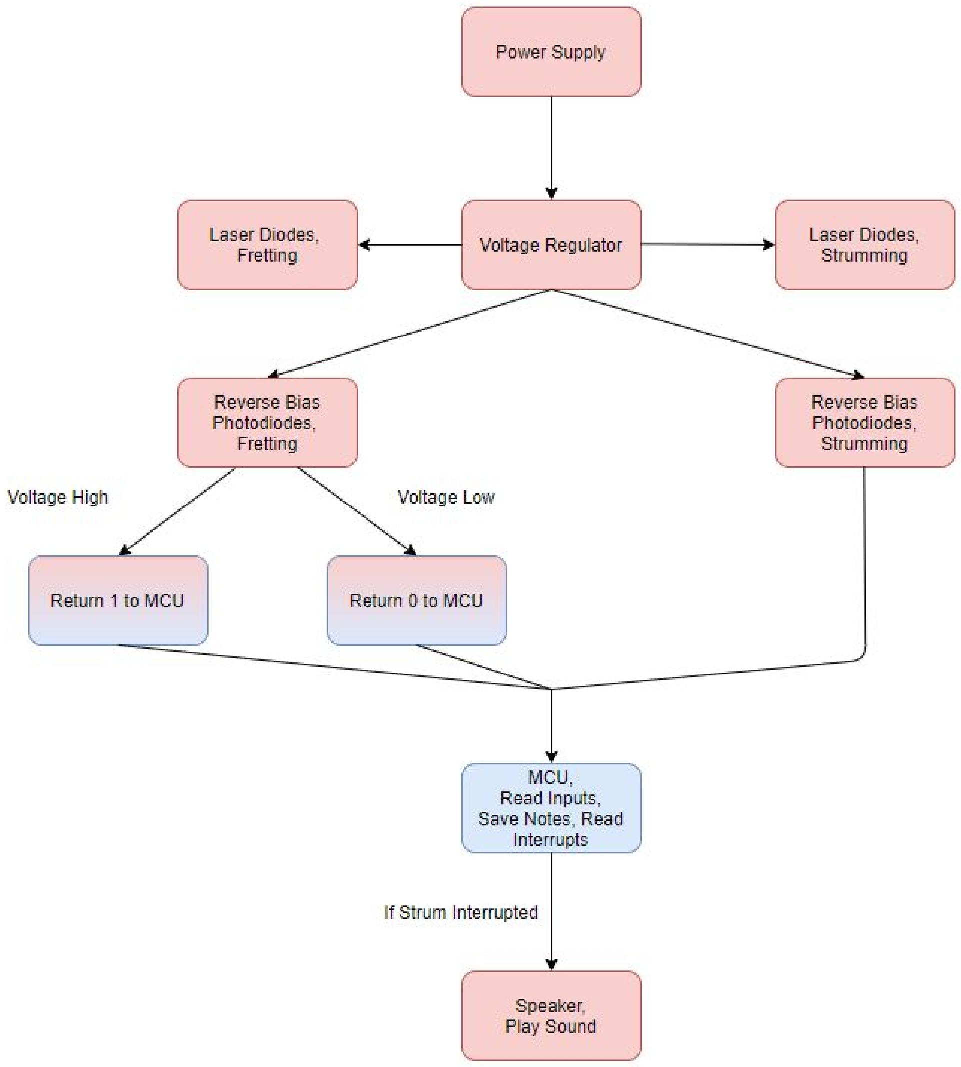


Figure 3: Block Diagram, Hardware

The goal of the hardware implementation is to allow the microcontroller to detect which string is being played, and where it is being fretted to determine what note to play. To do this, two sets of lasers to emulate one string on a guitar. We will call these sets the “fretting” sets and “strumming” sets. The strumming set of strings will be placed at the head of the guitar and will operate by detecting when the laser is interrupted. If a strumming laser is interrupted, then the note determined by the corresponding fretting laser will be played. The fretting laser system will operate by detecting the amount of light reflected off of the user's finger back to the photodiodes which will read the intensity of reflected light. Each fret will have its own photodiode, which will return a 1 to the MCU when the voltage is high, meaning the fret is being held down and reflecting light to the photodiodes. The photodiodes which are not reading any reflected light will return a low voltage, or value 0 to the MCU. This information will be used to save which note should be played by the speaker. The entire system will be powered by a lithium ion battery connected to a voltage regulator, which will power the laser diodes and provide a reverse bias to the photodiodes.

2.4.2. Software Component Block Diagram

From Figure 2, Figure 4 only focuses on the software components of the block diagram. By focusing on only the software of the block diagram, this gives the group a one sided perspective of how the software components should be connected to the hardware.

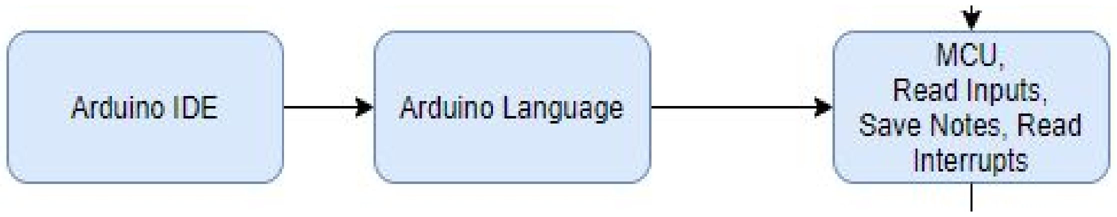


Figure 4: Block Diagram, Software

The goal of the software implementation of the project is to produce the correct frequency/pitch according to the laser intensity when another laser is strummed/plucked. When reading the laser from the sensors, there are two lasers to record and analyze. The first laser is the sturm/pluck from the “strumming” set , and the second laser is the fret “fretting” set. Both of these laser detection will need its own algorithm to process because they are two different systems of lasers. By reading the laser intensity, we can sign a frequency/pitch. When the laser is strummed/plucked we can use this reading as a trigger to play the frequency/pitch to the speaker. This will be explained more in details in Section 6.

3. Research and Selection

This section is focused primarily on all of our research and findings. It covers all of the relevant concepts that were taken into consideration such as music theory, hardware parts and technologies and any relevant software. The section is concluded with all of our part selections and chosen technologies based on the research that was conducted.

3.1. Music Theory and Acoustic Guitar Design

When designing a musical instrument, it is important to take in consideration what exactly makes instruments sound good, and how they provide a user-friendly way of making music. A basic understanding of music theory and how it directly applies to guitars is therefore an important factor in the design of our project. Music theory will apply more to the software design, as it will provide our software team with the proper sound information our device should output to closely resemble a musical instrument. Guitar design, on the other hand, applies more to our hardware design, as we will have to design the housing of our hardware in such a fashion that it feels as natural to play as a bass guitar or ukulele would, with each of the strings playing the same notes that would be played on the real instrument. The following paragraphs outline the key factors of music theory that we will consider for the software design, and how the design of acoustic guitars will impact our hardware design.

<u>Notes and Scales</u>

For the purposes of this project, a very basic understanding of music theory is required so that our instrument will play notes that sound musical. A good place to start is the concept of notes and scales. A note simply refers to a symbol denoting a musical sound. Notes are commonly symbolized alphabetically with the letters A-G. This alphabetical pattern repeats, as notes with fundamental frequencies in a ratio equal to any power of two (i.e. half, double, or four times), are given the same note name. For example, taking the note middle C, which resonates at 262 Hz, and doubling its frequency to 524 Hz would result a note also called C, that is said to be one “octave” higher than middle C. Similarly, playing a note at half the frequency of middle C, or 131 Hz, would also be called C, and would be labeled as one octave lower. The name octave comes from these being eight intervals away from each other on the musical scale. A scale is just a series of notes ordered by fundamental frequencies. Playing frequencies in a musical scale ensures that the sounds played will sound nice to the listener. By programming our MCU according to the frequencies in scales, this information will be used to play the proper frequencies in our laser instrument. The chart below shows the fundamental frequencies of the notes C-B, which make up the most popular scale in all of music, the C Major scale.

Note Name	Fundamental Frequency
C	261.63 Hz
D	293.66 Hz
E	329.63 Hz
F	349.23 Hz
G	392.00 Hz
A	440.00 Hz
B	493.88 Hz

Table 2: Frequencies of The C Major Scale

There are also notes which lie in between the frequencies of some of the notes of the C Major scale. These notes have fundamental frequencies halfway in between notes separated by 48 Hz, and are named after the notes directly above or below them. For example, the note between C and D resonates at a frequency of 277.18 Hz, and is called C# (C Sharp) or Db (D flat). The chromatic scale consists of the seven notes of C Major, and all sharp/flat notes, for a total of twelve notes, as shown in the chart below.

Note Name	Fundamental Frequency
C	261.63 Hz
C# or Db	277.18 Hz
D	293.66 Hz
D# or Eb	311.13 Hz
E	329.63 Hz
F	349.23 Hz
F# or Gb	369.99 Hz
G	392.00 Hz
G# or Ab	415.30 Hz
A	440.00 Hz
A# or Bb	466.16 Hz
B	493.88 Hz

Table 3: Frequencies of the Chromatic Scale

All the frequencies of sound that can be played on most instruments are some ratio equal to a power of two of the notes in the chromatic scale. This is valuable to know for both our hardware and software teams. For hardware, we can use this knowledge combined with the knowledge of what notes are played on standard ukuleles to assign the proper notes to their corresponding pins on the MCU based on their location on the fretboard. As far as software is concerned, we can use this knowledge in such a way that these frequencies are playable in the same way they would be on a standard bass guitar or ukulele.

Combining these factors together will allow us to construct an instrument that feels natural to play.

Chords

The next topic of music theory that is relevant to our project is the concept of chords. A chord is a combination of three or more notes played in unison. The most common types of chords are major and minor chords. Major chords consist of three notes; the first, called the root note, can be any note in the chromatic scale. The next note for a major chord would be two intervals away from the root note on the root's respective major scale, and is called the Major 3rd. Finally, the third note in a major chord is found four intervals away from the root note on its major scale, and is called a Perfect 5th. For example, a C Major chord would consist of the root note C, it's Major 3rd E, and it's Perfect 5th, G. Minor chords are built in the same way as major chords, except the Major 3rd is replaced with a Minor 3rd. A minor third is simply the flatted Major 3rd. So, a C Minor chord would consist of the root note C, it's minor 3rd E^b, and it's perfect 5th, G. This is relevant to our project because one of the main draws to stringed instruments in general is that it allows the player to play chords on just a single instrument. On the guitar, this is accomplished by arranging the strings in such a way that the user can easily play the notes of major and minor chords by simply fretting the strings in the proper order. By mimicking the strings of a ukulele or bass, our project will allow the user to play chords very similarly to how one would on an acoustic instrument. This brings us to how the design of acoustic instruments will impact our project.

Acoustic Guitar Design

As mentioned in the previous paragraph, guitars are designed in a way that allows the user to easily play chords by fretting the strings in the proper order. Guitar style instruments have strings that are assigned a root note, and by fretting the string this note goes up one interval in the chromatic scale per fret (i.e. fretting the first fret of the G string results in a G#). We will use this same method to determine what notes will be played by our instrument. There are many different guitar style instruments, with varying sizes and amounts of strings. For the purposes of our project, we chose to design a four stringed instrument. This is because of the popularity of the bass guitar and ukulele, the simplicity in their design, and the variance in size this gives us, as the ukulele is one of the smallest guitar-style instruments, and the bass guitar is one of the largest. That being said, the chord chart shown in Figure 5 will be used to determine what notes will be played by each string, where the open strings are tuned to play the notes G, C, E, and A, respectively.

With this foundation of music theory and physical instrument design we were able to establish an outline of how our project will function from an engineer's standpoint. Our knowledge of notes and scales will allow us to properly program the software so that we can accurately replicate the proper tonality of our musical instrument while observing the design of acoustic four stringed instruments allowed us to envision the housing and functionality of the hardware design of our instrument. We will use this knowledge as an outline of how we plan to create the optimal hardware and software design that provides the user with an instrumental experience. Figure 5 below shows an example chord chart for Ukuleles which we can emulate the proper tonality of our instrument.

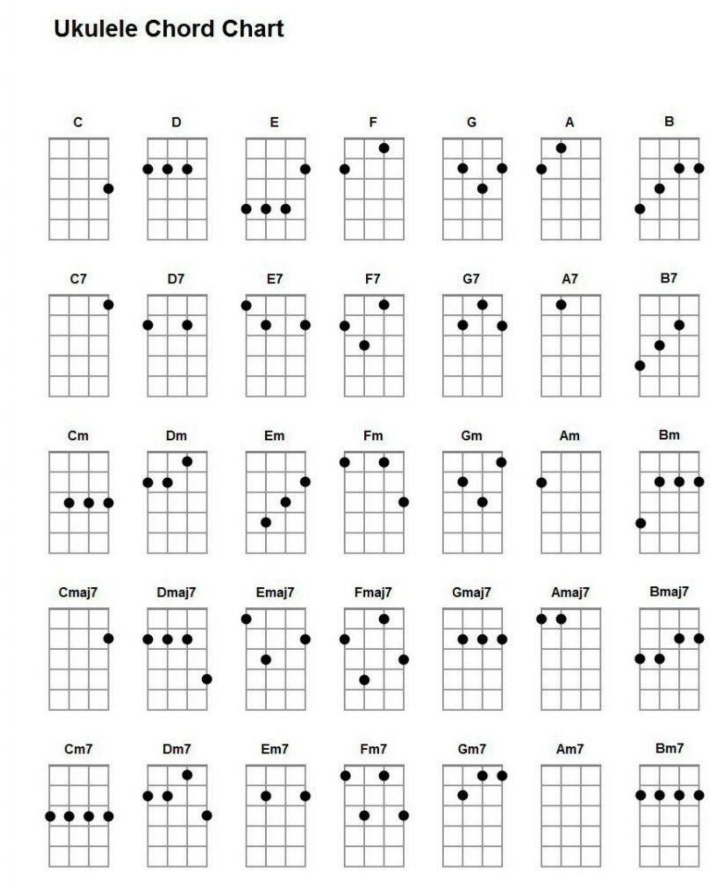


Figure 5: Chord Chart for Ukuleles in Standard Tuning

3.2. Existing Products

One of the most well-known laser instruments is the '*The Laser Harp*' by Laser Spectacles, Inc. Their design can be described as a MIDI note being sent to a synthesizer when the beams are detected as off. The lasers themselves are under 4.95 milliwatts. It is described as having an extremely fast response time with exceptionally clean lasers. Laser Spectacles, Inc. is not just a company that makes and sells their laser harp, but also performs shows around the United States. A typical laser harp they make sells for around $7,000, custom harps can be made for either a higher or lower price.

With further research, the only laser instrument available is as a harp. A company by the name of KromaLaser sells different forms of laser harps to the masses. Currently there are two Embedded RGB laser models and three RGB laser projectors. Another interesting product they sell is Laser Pedalboard, which attaches to a piano and shines different lasers based on the foot pedals you're using. The same principle applies to these models that we will be researching, in which individual lasers are detected when breaking the plane with the photodetector and measuring the amount of photons to play a note.

A revolutionary patented product codenamed ‘*Beambow’,* uses a combination of two or three lasers and measures the speed, direction, and position in which the beams are cut. A great feature of this product is the ability for a musician to customize the noises to their instrument and create new gestures when blocking the laser beams. Also, a note will not be played when the lasers are blocked, since the speed in which you break the lasers is being measured when the photodetector measures the photons again. More traditionally, our design and many other designs just use one laser to measure a break, but with the three lasers in Beambow, it is said to create a virtual line in space.

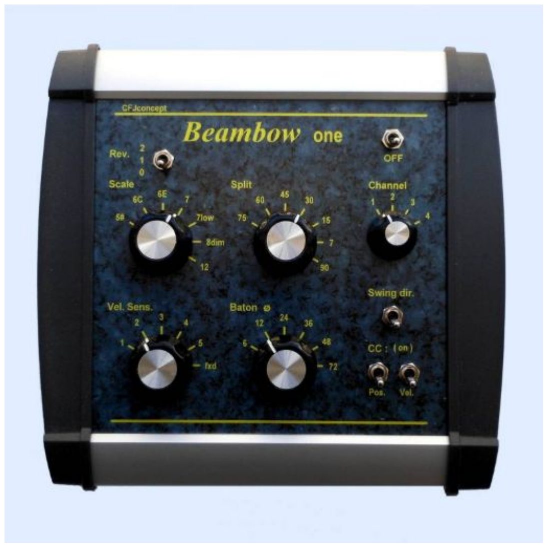


Figure 6: Beambow Controller Unit [D4]

With the ‘Beambow’ unit above, a musician can set different sounds and notes to play depending on the speed of the swing and the direction you swing at. Along with this unit is the laser system, which is odd because they are separated from each other. The combination of these two is a computer software to set and track points in which you break the plane. As with the Laser

Harp, our guitar laser idea was thought of from a previous UCF senior design project that revolved around building a laser harp. Approaching their design, we thought of what we can change and implement to make ours new and unique. Which birthed our concept of the laser guitar.

Another fascinating company is '*Prolight'*, where they sell a wide range of products from lasers, controllers, to smoke machines. From the beginning, *Prolight* began when the founders observed that their laser labyrinth project could be modified to make sounds when a laser beam is broken. Their first model was a framed design, but it turned out to be too cumbersome, so they built a frameless harp, known as a '*Prolight Laser Harp Controller LH1*'. The advantages of this model allow for any audio or video to be played when one of the lasers is broken, with the help of *Pangolin* software to capture all of the events. With this model, it is very consumer friendly with a controller and any ILDA compatible laser. Both the controllers they currently offer range in price from 900 to 1000 US dollars. The lasers are important and the boxes they sell start at 500 US dollars and can go all the way up to almost 5500 US dollars. The output of the highest end is 6 watts, with the beam size varying between units. To go along with the laser show, you can also purchase a smoke machine which creates a haze effect when the lasers project through. Overall, you can easily spend a few thousand dollars to set up your own laser harp rig.

The Laser Harp	**KromaLaser**	**Beambow**	**Prolight**
- Sell custom Laser Harps - Few thousand dollars - Performing crew	- Online seller - Embedded designs - Controllers - Least expensive	- First real laser-based MIDI controller - Patented - Uses multiple lasers as virtual space - Sophisticated	- Online retailer - Over seas -Controllers/Lasers /Smoke - Very Expensive - Reliable

Table 4: Comparing/Contrasting the different offerings

Overall, our design will be vastly different from these products, but still use the principles that they all use. Instead of having our lasers to act as a harp with/without a frame, our device will be very portable and can be carried around for a long period of time. It won't be a full-blown guitar, but a miniaturized version with less strings and less notes that can be played. The products described give us assurance that musical instruments that use lasers to interact with the user are feasible to build, and already have a market of users interested in this sort of product. Our purpose with this project is to make a unique product, that's consumer friendly and potentially enjoyed by thousands, while not breaking the budget of the average individual.

3.3. Relevant Technologies

This section focuses on our research of relevant technologies for our project. These relevant technologies consist of various tools and compliments that can be used in our project from both a hardware and software perspective.

3.3.1. Relevant Technologies: Hardware

The following section outlines our research on technologies that are relevant to our project from a hardware perspective. This includes our source of light, the laser, and the way it will detect this light, the photodetector, as well as the way we intend to provide power and portability to our device.

Lasers

To provide the user with an instrumental experience when using our product, our design will have strings made from lasers which can be fretted and played as if it was a real guitar. In order to understand how this will work, some basic background knowledge on lasers and photodetectors is necessary. First, we will look into the laser. A laser is a device that emits light through optical amplification based on the stimulated emission of photons. Lasers are more applicable than other light sources because the light emitted is coherent. The spatial coherence of laser light allows for the light to be focused into a tight spot, while staying narrow over long distances. This is important for our purposes because it allows us to simulate the strings of a guitar using laser light. Lasers also have high temporal coherence, which means they can emit a single color of light. This is valuable to know in case we choose to filter the light between adjacent strings based on color using narrow bandpass filters. This choice would greatly increase the cost of our system though, so it is still under consideration and will be looked into further when designing the system with several strings.

Photodiodes

Now moving on to the photodetectors, which will be used to return a measurable voltage to the microcontroller and determine which notes to play, and when to play them. Photodetectors contain a P-N junction which converts incident light into an electrical current. There are several different devices that operate as photodetectors, including photoelectric devices such as phototubes, semiconductor devices such as photodiodes and phototransistors, and photovoltaic devices such as solar cells. When used in conjunction with the laser, the photodetector becomes a very powerful piece of technology. For our purposes, we will use photodetectors to tell our microcontroller when a laser beam on the strumming set of strings is interrupted so the system will play a note. Similarly, for the fretting strings, a series of photodetectors will be used to tell the microcontroller which note to play based on where the beam of light is being interrupted.

Regulators

During this section, regulators provide a vital role to sufficiently provide power accurately to all the necessary components. Whether it will be boosting up or down, the batteries selected will need additional support to supply their power. Each component will require different levels of power, so multiple outputs might be necessary. In the end, the lithium ion batteries will plug into the printed circuit board to the regulator before power is distributed. It's wise to not supply too little or too much as well, and our regulator will act as a means of controlling this factor.

ADC/DAC

Here the ADCs, and DACs can either be internal or external, with advantages/disadvantages to both. In our case, we will currently be using the internal modules to the ATMEGA2560, with more than enough pins. If for some reason we lose accuracy or experience noise, external is always an option. As said before, each analog input(lasers to photodetectors) will have a digital signal to represent what note to play corresponding to the fretboard. Then take that signal and convert it back to an analog signal to the amplifier.

3.3.2. Relevant Technologies: Software

The main objective from the software side of our project is to program the microcontroller accordingly in order to produce the correct frequency/pitch and notes. Since the microcontroller is a core component to reproducing a sound, the microcontroller needs to be programmed in a way that each laser will be assigned a frequency/pitch. The lasers that represent each frequency/pitch will be positioned on the neck of the guitar. Similarly, the lasers on the head of the guitar will be used to program the microcontroller in such a way that when a laser is broken it will act as a trigger to output the respective frequency/pitch based on the corresponding laser on the neck of the guitar.

As the microcontroller is a core complement to achieve our objective, it is crucial that our research includes various technologies and programming languages are most common when working with microcontrollers. These technologies include, but are not limited to the program language, and the program environment. By researching these technologies, it brings to light various obstacles and solutions to take into consideration. When seeing these obstacles and solutions, it will ameliorate both the implementation and testing phase when designing the project.

Language Hierarchy

In the world of programming there is now an abundance of programming languages, each of which are designed for different purposes and tasks. These languages are typically classified into three main categories, machine, low-level and high-level languages. Machine languages can be thought of as "the native tongue of the computer" [1]. From Figure 7 below we can see that Machine language can be seen in this manner because it is the language closest to hardware. A program written in machine code is composed of a sequence of ones and zeroes known as binary digits, which represent simple operations that are supported by the computer. These programs can be executed directly on the CPU, while higher-level languages have to be translated into machine code before they can be executed.

There are different viewpoints when it comes to categorizing languages as low-level. People often include languages anywhere from machine code to C, due to all the advancements in technology opinions are now mixed. Generally speaking low-level languages can be described as using little to no abstraction in comparison to a computer's instruction set architecture. When discussing low-level languages, the most commonly thought of language is assembly. Assembly was made in efforts to make programming more feasible to humans. It replaces binary code with pseudo-english words and hexadecimal values. Translation to machine code is done by an assembler which is simpler than having to rely on a compiler. Assembly is not commonly used anymore, however, for education purposes it can be very beneficial.

High-level languages can be described as being heavily abstract and more english-like which makes it easier for programmers to learn and understand. C#, Java and Python are all good examples of higher-level languages. The main benefits from utilizing high-level languages is their simplicity, readability and portability. The syntax makes for code that is easy to write, and maintain and due to the heavy abstraction these languages generally do not have any dependencies on the machine. On the downside, high-level languages are slower in comparison to low-level languages because of the time it takes to translate into machine code. This is something that should be considered when it comes to programming small devices such as microcontrollers.

By researching the different levels of languages, we can now better understand how languages are classified and which ones may be the best suited for any given use-case. For our use-case we can see that a low-level language may be the best fit. Low-level languages gives the programmer more freedom to control memory and the code is easily translated into machine code. This results in less use of power and memory which can be advantageous, given the size of microcontrollers and their lack of processing power and memory capacity. On the contrary, high-level languages are at a disadvantage for this same reason, however, microprocessors have improved over the years. Thus, high-level languages now have use cases in which they can be applicable.

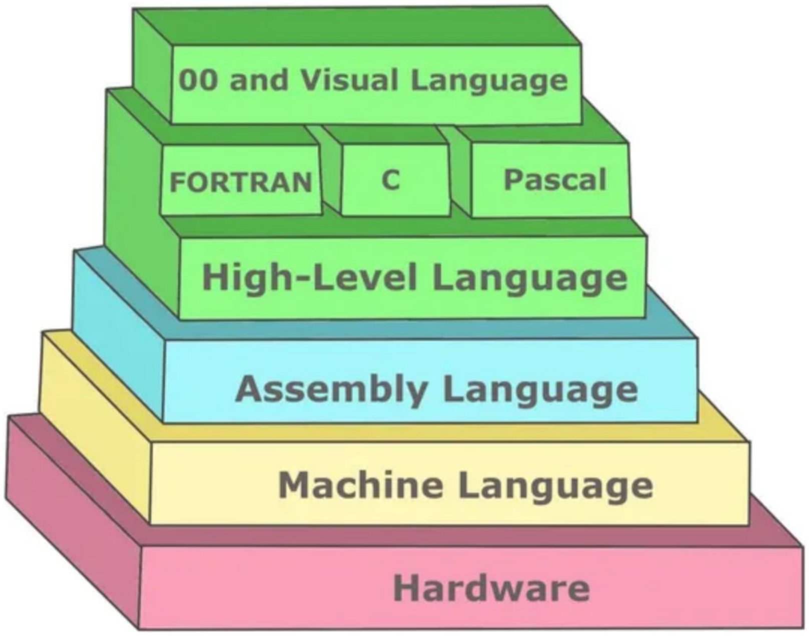


Figure 7 : Language Hierarchy [1]

Programming Languages

In the earlier stages of microcontrollers developers were limited in the range of low-level to mid-level programming languages such as C and Assembly. These languages were most commonly used because they are closer to machine code and therefore are more specific to the architecture and hardware of the system. Over the years as technology evolved, so have microcontrollers. Microcontrollers have the ability to not only program at a low-level, but also in high-level programming languages. Developers now have a wide range of languages to choose from besides C and Assembly. The high-level languages that are now supported by microcontrollers or embedded systems are Python, C++, Java, C#, and Verilog.

When selecting a programming language for our microcontroller, the project needs to consider what type of languages can be supported by our microntroler, and if there are any constraints to the language. Some of these constraints are, but not limited to efficiency, functionality, runtime, and framework.

C Language

The C language was originally said to be a high-level language as seen in figure 7 above. However, in todays' modern day many people have started to consider C as a lower-level language. This is highly in part to the fact that memory management is in the hands of the programmer, unlike Java or Python in which this is handled for you.

C is one of the most popular languages when it comes to microcontrollers and embedded systems. Most electronic gadgets that people use on a day to day basis such as cell phones, cameras, stereos, etc developed in C. Some of the reasons why C is popular when it comes to embedded systems are memory management, performance, portability and bit manipulation. C's memory management and performance are essential for microcontrollers and smaller embedded systems where memory is limited. Having the ability to control it allows you to make it as efficient as possible and the fact that C compiles easily into optimized machine code improves the performance of an embedded system as a whole. Portability and bit manipulation also play a pivotal role in embedded systems as the portability allows for the code to be moved to different operating systems without a problem and bit manipulation makes it straightforward to program the hardware where flipping bits in registers is recurrent. These capabilities of C make it extremely popular.

C++

C++ is considered to be an intermediate-level language, as it includes capabilities of both high and low-level language features. C++ was created as an extension of the C language and can be considered as a superset of C. This means that almost anything done in C can be done in C++, however the same can not be said when going the other way. Many of C++'s features, including classes, automatic resource cleanup, parametric polymorphism, and domain specific libraries such as Arduino syntax. Having C++'s low-language capabilities allows developers to work with various embedded systems. Being able to have low-language capabilities allows libraries of the language to easily be used to control the hardware of the device. Without these libraries, then it would be difficult to directly change special registers to control everything.

Not only does the language provide domain specific libraries, it also gives the ability to use abstracts and object oriented paradigms. The ability to use high-level language paradigms and functions gives the developers more power to create without the tradeoff of affecting the infrastructure of a system directly, unless explicitly doing so. The data structure of C++, like C, is algorithm-based, and therefore it is a great fit for solving any challenging programming problems that are commonly faced when dealing with embedded development. C++ also has processor independence, and microprocessors today come loaded with C++ compilers starting at $1 USD [2].

Assembly

Assembly is a low-level programming language and there are different variations which depend specifically on the processor architecture. “Assembly Language is a pseudo-English representation of the Machine Language.”[15]. Assembly uses pseudo-English words such as ADD, SUB, and MOV, along with hexadecimal codes to create what is known as instructions. Each instruction is equivalent to one line of binary code and they rely on an assembler to make the conversion over to machine code. On the other hand high-level languages translate into many binary instructions and they rely on a compiler to make the translation over to machine code. For this reason assembly executes much faster and takes up less memory as the memory management is entirely up to the programmer.

With the advancement of technology, assembly is not used as much to program microcontrollers as it once was because memory is now cheaper, CPUs have become more powerful, and assembly lacks portability. However, programming a microcontroller in assembly can be very beneficial for any developer. When programming in assembly the programmer must know all about the hardwares architecture and the CPUs registers. This will allow the developer to grasp a deep understanding of the microcontroller functionalities and have more control to make use of all the features it has to offer.

Java

“Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.” [3]. Java allows developers to “write once, run anywhere” (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation [3].

With the ideology of “write once, run anywhere”, we can take our java code and implement it to various devices that have java installed on it. Java code is run using Java Virtual Machine (JVM) which allows Java code to be executed on any device with Java on it because java has its own virtual machine, Java’s runtime environment ensures that applications of different systems (e.g JVM and the Microcontroller’s architecture) are not interfering with one another's process by checking the code in the JVM before execution. If Java’s code attempts to alter the microcontrollers core behaviors, it won't be run [4]. The java code is independent from hardware and the architecture of our microcontroller.

Python

Python is one of the most popular programming languages today mainly because of its readability which makes it easy for beginners to learn. It is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together [5]. Python's uniqueness of dynamic typing and semantics allowed many users to create libraries for specific tasks. These libraries range from artificial intelligence and machines learning to its web applications, but also its embedded system capabilities.

Having these various libraries, python scripts can be developed that set the embedded system into different states, set configurations and test all the possible perturbations and interactions that the system would have with the external world[4]. These Python scripts allow automated testing for our system to undergo. From these automated testing we can see the results of bugs or non-conformances. A big reason as to why python is not popular in the realm of embedded development is due to its runtime speed as a result of its high abstraction. With the popularity of python on the rise, MicroPython was created. MicroPython is an efficient implementation of Python 3 with a small subset of the python standard library and it is optimized to run on microcontrollers[20]. With this technology python will look to gain popularity over C and C++ which are the most commonly used for embedded systems.

Verilog

Verilog is known as a Hardware Description Language (HDL) that focuses designing, and modeling embedded systems. The syntax is very similar to that of C, however the languages they used for different purposes. Verilog supports different levels of abstractions such as the behavioral, register-transfer, and the gate level which help design and model embedded systems.These levels of abstraction require different coding methodology and implementation method because of how sensitive and close they are to the architure of the hardware.

Verilog is mainly used when it comes to programming Field Programmable Gate Arrays (FPGA). Field Programmable Gate Arrays are integrated circuits that can be configured or programmed to perform a given task. They are similar to microcontrollers however, have more flexibility to be configured into anything the programmer may be trying to accomplish.

Programming Language Comparison

Our research from a software perspective consists of many available programming languages. Each programming has their strengths and weaknesses. In order to narrow down what programming language we will use, we created a table of all of our researched programming languages to make our choice. Our choice of a selected programming language is discussed in 3.5.2 Software selection. The table consists of key attributes from each language. The table below shows this comparison.

Languages	**Advantages**	**Disadvantages**
C	● Dynamic memory allocation ● Compiled language ● Easy to learn ● System programming ● Built-in Function	● Not Object Oriented base ● No Run time checking ● No Strict type ● No Namespace
C++	● Object Oriented ● Self-memory management ● Multi-level language ● Hardware specific libraries	● Bloated Machine Code ● Slow ● Emphasis on instructions ● No built in threads ● No garbage collector
Assembly	● Fast execution time ● Memory efficient ● Hardware oriented ● More control over programs	● Difficult to understand ● Long and convoluted codes ● Compatibility with different architectures can be an issue
Java	● Object Oriented ● Platform independent	● Large Code size ● Usually slow ● Single Paradigm ● No low-level programing
Python	● Easy to learn and read syntax ● Scripting for automated testing ● Object Oriented ● Asynchronous Coding	● Slow execution ● Weak mobile computing ● Design restrictions ● Not memory efficient ● Runtime errors
Verilog	● Hardware modeling language ● Handles Bidirectional devices ● Is SystemVerilog	● No High-level construct ● No custom types ● Poor asynchronous signals

Table 5: Languages Advantages Vs Disadvantages

Software Environments

Programming languages need a compiler in order to translate human-readable source code to executable machine code. When a code is written in a specific language that is already compiled, it gives the hardware instruction on what it needs to achieve, and how it should perform based on the source code.

An environment that a specific programming language can work and compile with is an Integrated Development Environment (IDE). Integrated Development Environment are software application suites that consolidate basic tools required to write and test software.

For every programming language, there is an environment that supports it. There are an abundance of Integrated Development Environment that are available to developers, many having a compiler built into them. Not only do Integrated Development Environment have built-in compilers, but they offer various tools specific for the programming language it supports. Having an environment centralizes the codes and its dependents in an area, while having tools to navigate and modify code quicker. Integrated Development Environment also provides debugging tools to help catch errors and fix any bugs within the code. This both time and resources from being wasted when fixing it. By researching what type of software environment our code will be built from, it will save resources for when we will test and implement the code.

To go along with an Integrated Development Environment to work in we must also have an environment which will facilitate working on a code base with a team. Such environments are known as version control systems. These systems allow for code to be kept organized when there are many contributions from different team members while also keeping a history of it.

Eclipse IDE

The Eclipse IDE is famous for our Java Integrated Development Environment (IDE), but we have a number of pretty cool IDEs, including our C/C++ IDE, JavaScript/TypeScript IDE, PHP IDE, and more [6]. Eclipse was mostly written in Java, therefore it would be well suited for Java based code. Not only does it support Java, it also supports other programming languages such as C, C++, PHP, and many more. In order to use other languages with eclipse, all the user has to do is install the language plugin within the IDE.

Visual Studio Code

Visual studios is considered to be a hybrid environment. It is considered a hybrid environment because it is in between an IDE and a text editor. Being a hybrid type of environment is what gives Visual Studios Code its notability to still be considered a software environment. Visual studio is lightweight, and powerful cross platform editor. It has a large ecosystem of plugins which developers can use for their specific needs. By allowing picking what is needed, the environment can be lightweight, but still perform the desired task needed. Making the environment operate as a language-agnostic code editor for any language. Some of the plugins consist of, but not limited to debugging, syntax highlighting, code completion, refactoring, and workspace customization.

Code Composer Studio

Texas Instruments (TI) developed an Integrated Development Environment that supports their microcontroller (TI's based Microonctroller) and embedded processors portfolio called Code Composer Studio (CCS). Texas Instruments description of Code Composer Studio is that it "comprises a suite of tools used to develop and debug embedded applications. It includes an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler, and many other features."[12]. Code Composer Studio was designed as an Integrated Development Environment for both low-lew and embedded applications.Code Composer Studio also combines the Eclipse software framework with more advanced embedded programming and debugging tools. Code Composer Studio can also connect with MATLAB, Simulink, and with Simulink. These IDEs are for other programming languages that have embedded development tools.

Vivado Design Suite

Vivado Design Suite is one of Xilinx Hardware Description Language (HDL) software that emphasizes on synthesis and analysis of Hardware Description Language Design. The software is considered an IDE because its high-level synthesis with toolchains allow written C and C++ language code to be converted into programmable logic. Using Vivado Design Suite gives developers the backbone for advanced, generation-ahead hardware, software, and system development using All Programmable Abstractions [7]. Using Vivado Design Suite developers should experience accelerating system implementation and integration, and comprehensive hardware debugging.

Arduino IDE

The Arduino IDE is an open source software platform that allows you to write code and upload it easily to Arduino devices. The platform was written in Java and is supported by Windows, Mac and Linux operating systems. The languages supported by the Arduino IDE are C, C++ and Arduino's own native language which are based on C and C++. Programs written on this platform are called sketches, which are compiled into binary files that are then transferred directly to the board through the configured port. Given that the IDE is native to

Arduino, if an Arduino microcontroller is decided upon, it would be favorable to use this environment.

MPLAB X IDE

The MPLAB X IDE is a software environment that gives the user a wide range of freedom when configuring, programming and debugging a broad selection of microchips and microprocessors. The MPLAB X IDE has a variety of tools and features that allow for developers to efficiently debug projects such as a data visualizer and an input/output view. This Integrated Development Environment is supported by Windows, macOS and Linux and is free to use.

Atmel Studio 7

Atmel Studio 7 is the software environment platform created specifically for programming and debugging all AVR and SAM devices. AVR and SAM devices are different families of microcontrollers that were developed by Atmel. Studio 7 provides a user friendly environment to develop applications written in C/C++ or assembly. It also allows you to import Arduino code, which are called Arduino sketches, as a C++ project. Atmel Studio also provides a wide range of libraries, tools and extensions that can be added through its online store the Atmel Gallery. This platform is free to use and is only supported by Windows operating systems.

Git/Github

When working on projects with multiple developers things can get messy. Each developer's contributions need to be integrated into one single project. Therefore, it is essential to have a version control system that can facilitate team collaboration. That is where Git and Github become relevant. Git is a version control system that allows you to manage and store the history of your source code, while also keeping track of who made which changes. It also allows you to go back to any of the previous versions which can be critical when trying to add more functionality or features to a project that may cause the entire code to break or fail. Github provides an online source that provides the same functionality git has while also acting as a host system for git repositories. Git and github will be critical in our project as it allows our team to have an environment in which everyone can access the code, keep track of the code history and also monitor all the contributions and changes made by the team.

3.4. Strategic Components and Part Selection

This section focuses on what components or parts have been taken into consideration from our research in section 3.3. The parts chosen will ensure that our project design works smoothly and efficiently, while being mindful of our budget and keeping the device relatively cheap.

3.4.1. Battery Selection

The battery selection for our project will have to be widely available at a low cost and provide enough power to the device for a session to last at least an hour. Our chosen battery will need enough current with a low self-discharge and a long cycle life. The voltage of the battery also plays a vital role, needing to be able to provide enough potential to power all the electronics. As for powering the speaker/amplifier, a wall outlet will most likely be used while the electronics use the battery technology chosen. Preferably, maintenance will not be an issue but if avoidable will be the best option. With our final selection, we will purchase each cell individually or a pack that contains the cells already configured with our specifications.

Lead Acid

These batteries are the most widely available technology in batteries today. With having an incredibly low energy-to-weight ratio and low energy-to-volume ratio, these batteries can provide high currents which makes them suitable for electric vehicles. With these drawbacks, an advantage is actually their large power-to-weight ratio. During disposal, you will have to take the battery to a landfill to properly dispose of the lead used in the battery. A description of the charged state of the battery uses the chemical energy of the potential difference between the lead and PbO_2. The electrical energy is then produced from the interactions of the H_2O molecules produced from the H^+ and O^{2-} ions.

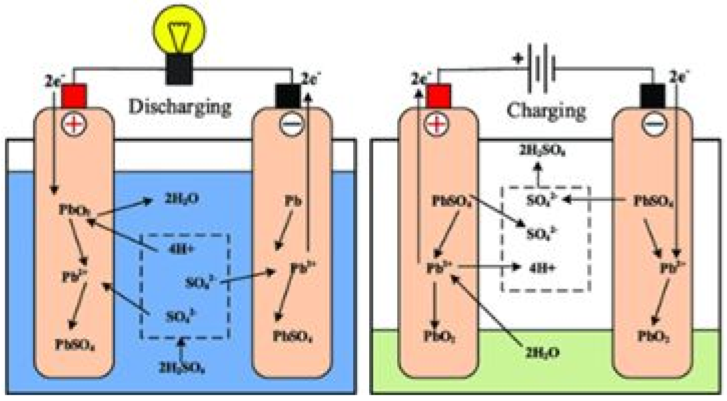


Figure 8: Lead Acid charging and discharging chemistry [D5]

Lithium Ion:

With this type of battery, they have a high energy density, the ability to self-discharge with low maintenance. Several disadvantages include the cost, the ability for protection and the ageing of the battery. The main advantage compared to NiMH is the decreased self-discharge rate of about 1%-2% per month, paired with its high energy density, allowing longer uses in between charges. The technology used consists of lithium ions moving from a negative electrode through an electrolyte to the positive electrode, during charging and discharging. If punctured, the electrolyte will cause an explosion and start a fire.

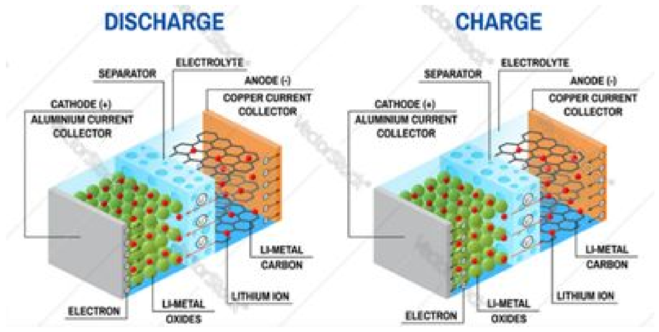


Figure 9: Charge and discharge chemistry for lithium ion [D6]

<u>Nickel Metal Hydride</u>

With the beginning stage of NiMH batteries, they were unstable due to the alloys in the cells. Development occurred and created new hydride alloys to improve the stability and increase the popularity. With having a higher energy cell density than NiCd but is less durable. With a benefit of being more environmentally friendly, limitations include a higher self-discharge with a decrease in performance at higher altitudes. Often needing higher maintenance from having to do a full discharge to prevent any crystalline structures. The chemical reaction to produce electrical energy starts with a positive electrode using nickel oxide hydroxide and the negative electrode uses a hydrogen-absorbing alloy.

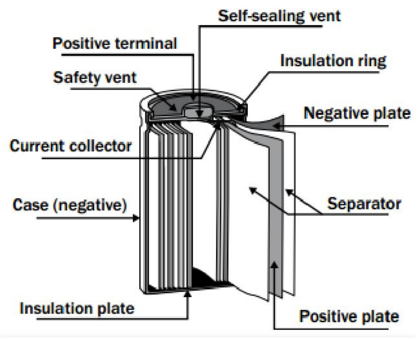
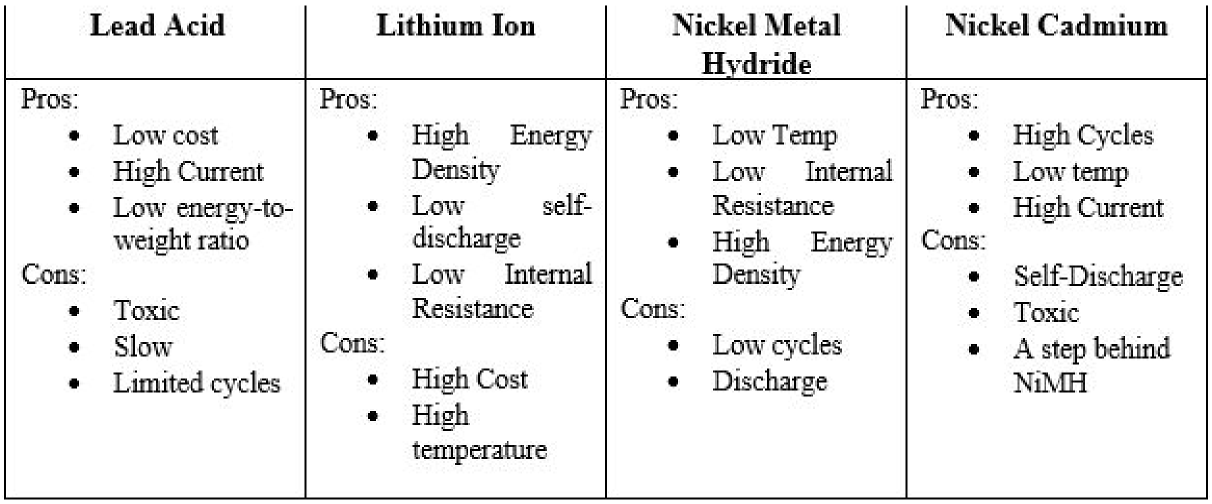


Figure 10: Nickel Metal [D7]

Nickel Cadmium

These batteries are another type of rechargeable battery that uses cadmium as the electrode and nickel oxide hydroxide. A great advantage is how inexpensive they are, and low maintainability required. A downside is the low energy density and self-discharge higher than normal. With around 100 cycles per cell, they have to charge again rather quickly after one use. They also contain toxic materials, which need extra precautions when disposing of them. Overall, these batteries are great for a cheap alternative, but do not meet the requirements for having a long-lasting battery. The negative properties within these batteries are something that we must consider, in order to improve our functionality of how we power our components without the cost of health and financial issues.



Lead Acid	Lithium Ion	Nickel Metal Hydride	Nickel Cadmium
Pros: • Low cost • High Current • Low energy-to-weight ratio Cons: • Toxic • Slow • Limited cycles	Pros: • High Energy Density • Low self-discharge • Low Internal Resistance Cons: • High Cost • High temperature	Pros: • Low Temp • Low Internal Resistance • High Energy Density Cons: • Low cycles • Discharge	Pros: • High Cycles • Low temp • High Current Cons: • Self-Discharge • Toxic • A step behind NiMH

Table 6: Various Battery Pros and Cons

Our Process:

From the different technologies above, we have chosen to work with Lithium Ion, for its high energy density to allow for a long play time of the instrument. The two important factors with batteries are the voltage and the mAh rating. For the voltage, we don't need it to be too high, but high enough to allow for a step-down voltage to power the electronics. When it comes to the storage capacity, the mAh rating determines how long the battery will operate. Some comparisons here are for phones, more specifically the iPhone X and XS have a battery around 2700 mAh. Now, this is a perfect scenario for the phone since all their electronics are optimized in house. For our case, we would probably be looking to double that mAh, which will be plenty of a charge to allow our device to play for many hours. The more charge and power our battery has enables the user who is using our devices to play for longer periods of time without the constraints of high power consumption components in our project.

The batteries can now be a single pack, or we can use multiple cells and plug them into a battery holder. Through some research, the cost of buying 4 cells at 3.6V each is about a quarter to third of the cost for a single lithium ion battery pack. With this, 4 18650 lithium ion cells will be utilized at 3.6 volts. The configuration will use a 4-cell battery holder which will give the pack 5000mAh, which will be plenty for our design. Another alternative would be to use 3 cells for a total of 10.8V, which will be suitable. With the voltage range we have, each component on the PCB will need an operating voltage, usually around 5V for the parts we have chosen now. Utilizing a buck converter instead of a linear regulator to step down our voltage will be more efficient while producing less heat and extend the lifetime of the battery cells.

The configuration of the battery cells is very crucial to our set up. To achieve our desired terminal voltage, each cell will be configured in series. With four cells, pairs will be configured in parallel to achieve 5000mAh, from 2500mAh cells. This combination is called 4s2p, which is very common in electronics today. Most importantly, each cell must be the same voltage and capacity, if not there will be an imbalance from the odd one out. With a series configuration, the terminal voltage is only as strong as the weakest cell, exhausting more quickly. Our goal is to achieve a 2s2p configuration as shown below to increase our capacity.

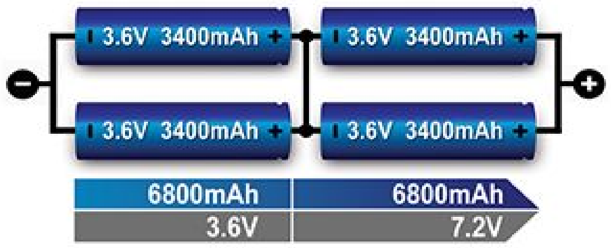


Figure 11: 2s2p battery setup **Terminology/Safety [D8]**

Another option instead of using four cells is to separate the ukulele into different sections according to the power supply. Starting with the fretboard design, there will be a total of four lasers, with each having four photodiodes for the notes. Each string will use one lithium ion cell battery at 3.6 volts to supply the photodiodes. As well for the one photodiode on the head of the guitar. With this, instead of having a total of four cells powering the PCB directly leading to all the electronics, four cells will be used to power each string separately. As with the lasers, five volts is needed which will use two cells and step down the voltage. There are a total of eight lasers, which only need power, which will utilize two 3.6V lithium ion cells in series. Next is the MCU which will have its own two cell power supply. With this, instead of being top heavy from one-point powering everything, the supplies are spread out which will hopefully provide a consistent and reliable supply of power.

Within the industry, specifications are made on the cells in series first and then the cells in parallel. As mentioned above, that can be seen as 2s2p. You have two cells in series and two in parallel. The connection of these cells first starts with making pairs in parallel, once that is done the two parallel connections can now be placed in series. This is most common for lithium ion cells, but other technologies such as NiCd may start with series. It's common to maintain its chemical nature. The safety of these cells can be made with positive temperature coefficient switches or charge interrupt devices. Commonly in larger than 3-cell packs. Working to switch the cells off when experiencing excessive pressure or current. Keeping in mind that cells may turn off early which will increase the load current among the other cells. Best practices also include keeping each cell clean, never mix batteries, correct polarity, and remove batteries when not in use.

The first method involves a positive temperature coefficient thermistor. As the temperature increases of the battery the resistance also increases for the PTC. Defined into two categories there is one made of silistors and a switching mode one. The important characteristic of the silistor is the use of silicon which gives a linear characteristic. Unlike this, the switching mode characteristic is nonlinear. Specifically, when the switching mechanism is heated, the resistance decreases initially until the critical temperature is reached. Once the temperature passes that threshold, the resistance increases drastically. With this, there are two modes: self-heating and sensor. While in self-heating, a current is passed through the thermistor and the resistance increases as described above. During the sensor mode, a minimum amount of current is passed through the thermistor, which neglects the self-heating portion. Once the device starts heating up, the resistance begins to increase as well.

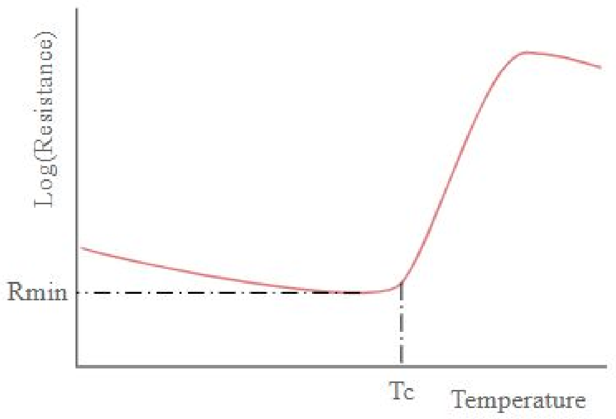


Figure 12: PTC graph of resistance as temperature increases [D9]

The second method involves charge interrupt devices. As the name suggests, this is a fuse, that when triggered turns off the electrical circuit. This can be triggered by many factors, ranging from high temperature, voltage, or pressure. As mentioned above, the same principles apply to accurately shut off the charge when unintended behavior occurs. An issue that can be observed is thermal runaway where the voltage threshold has been passed and short conditions occur. Overall, protecting your devices against in-rush current and overcurrent is obtainable with these applications.

3.4.2. Voltage Regulators

The use of a voltage regulator in our design will ensure that our printed circuit board and diodes will receive a constant voltage not exceeding their recommended thresholds. With our current design, we are using a battery pack, which provides a DC voltage as opposed to an AC voltage from a wall outlet. Which will make our design portable. With this mind, our focus will be on the technology used in DC voltage stabilizers. With the battery pack for our design, we will end up having to step down the voltage to the components on our printed circuit board, which will feature a Buck converter. There is also a wide range of regulators that will also be discussed below and then compare each one with their advantages and disadvantages.

Linear Regulators

The concept of a linear regulator uses a closed feedback loop to create a bias to sustain a constant voltage. Two types include a series and shunt. The series type is a simple where a variable element is in series with the load. By changing the variable resistance, the voltage will change accordingly. With an advantage of having a clean output, when it comes to step up the voltage, they are very unstable. The second type is a shunt regulator, which uses a variable resistance to supply a voltage to ground, unlike a series where a load is used to supply voltage to. The current however is diverted away from the load directly to ground, which is not as efficient. Used more commonly in low powered circuits, where the current is kept low as well. A low-dropout regulator can still regulate the output voltage when both the supply and output are close to each other, usually within the dropout voltage. The advantage of this is no switching noise because there is no switch, unlike the switching regulator. There are also fixed regulators that allow a fixed output voltage such as 5V or 9V, but also adding a Zener diode to the IC, you can vary the range of the output.

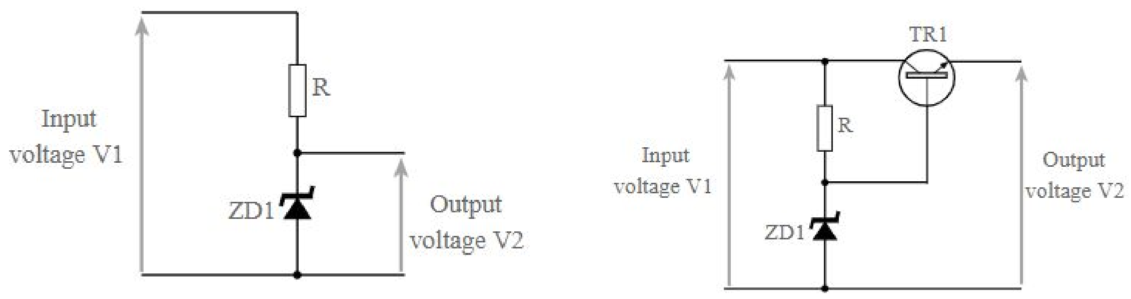


Figure 13: Depictions of a shunt and series voltage regulators

Switching Regulators

The idea here is to use a switching mechanism to transform the input voltage into a pulsed voltage, and then smoothed using capacitors. A MOSFET is used to turn on until the desired voltage is set, once so, the MOSFET turns off. Continually to do this operation makes these regulators very efficient. This is known as pulse width modulation; hence the duty cycle of the switch determines how much of a charge is transferred to the load. There comes a cost with the high efficiency and low heat generated, such as an increased noise and a much more complicated design than the other regulators discussed so far.

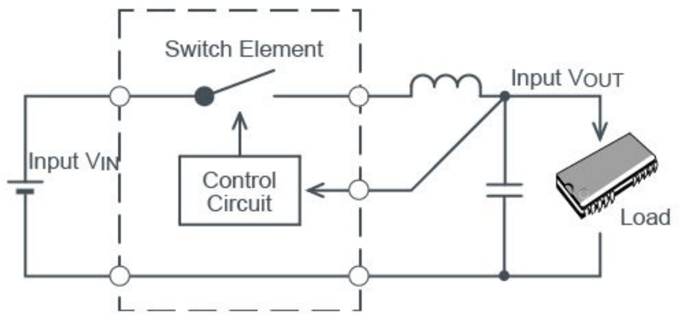


Figure 14: Common switching regulator

Buck Converter

A buck converter allows for a higher input voltage to be stepped down to a lower output voltage. Ideally, the current through the inductor is controlled via a transistor where when the current is on, there is zero voltage drop and zero current flow when the switch is off. The voltages also remain constant through the whole process. There are also two modes such as a continuous mode and discontinuous mode.

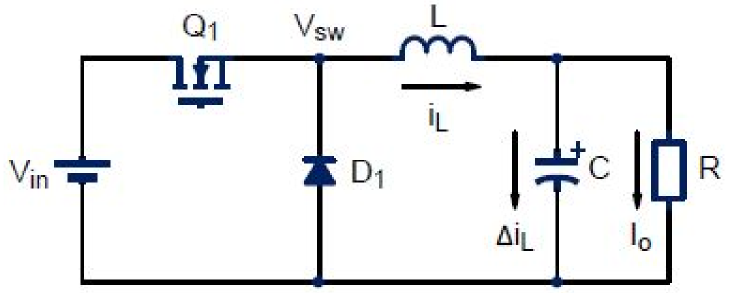


Figure 15: Common buck converter

Boost Converter

A boost Converter acts as the opposite of a Buck converter where the voltage is stepped up. In the figure below, the inductor resist changes to the current. While the switch is closed, the current flows through the inductor storing energy as a magnetic field. When the switch is open, the impedance is now higher resulting in a lower current. The magnetic field will now be destroyed and the polarity across the inductor is now reversed. Cycling the switch fast enough will result in not a full discharge and the voltage across the load will always be higher than the input.

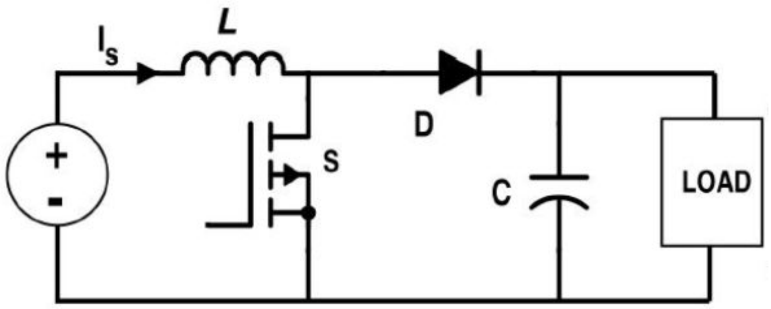


Figure 16: Common boost converter

Specifications

With each regulator, specifications such as the load regulation, dropout voltage, inrush current, and quiescent current are considered when designing a regulator. Importantly, the load regulation measures the change of the output with respect to a change in the load current. As mentioned, the dropout voltage sets the minimum voltage difference necessary for the supply and output to still supply current. Inrush current is the maximum amount of current supplied to the IC when it’s first powered on. Lastly, the Quiescent current is important if efficiency is an issue since the current is measured while no load is connected and is internal to the IC. To piece it all together you can test the transient response of each regulator, by suddenly changing the load current and comparing the data with those provided in the datasheet of the specific regulator.

<u>Our Process</u>

For our design we will be using a higher voltage provided by a lithium ion battery and step it down to a smaller voltage to power all of the electronics. When it comes to PCB design, you can choose either a surface mounted or through hole device. A through hole regulator typically deals with large voltages and currents while a surface mounted regulator deals with much smaller voltages. For the MCU itself, it will require a higher voltage than the electronics, so a linear voltage regulator will be used. According to the data sheet, the AtMega2560 requires an operating voltage of 5V. A perfect IC to do this is provided by adafruit (7805 TO-220), where an input voltage between 7 and 35 volts will step down to 5 volts, this will be used for testing purposes and a buck converter will be used instead of linear regulator for its efficiency and power usage. Below are different buck converters found through Texas instruments Webench tool. Which is a power designer tool to optimally pick different ICs to power electronics.

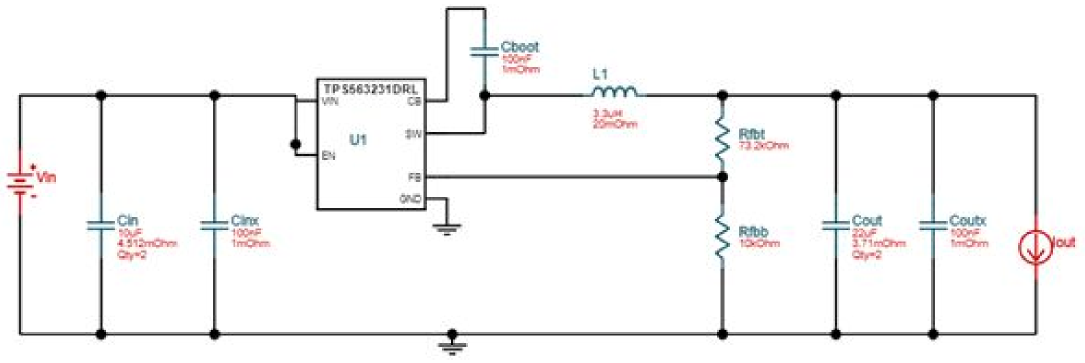


***Figure 17: schematic of TPS563231 IC* [D1]**

The input voltage range for figure above is 4.5 to 17 volts, which is perfect for our battery configuration of either 10.8V or 14.4V. Some important features are the switching frequency of 600 kHz, and an output range of 0.6 to 7V. There is also a low shutdown current of 12μA and the package for the IC is a 6-pin SOT563. Some common applications mentioned on the datasheet are a digital

TV power supply, surveillance, and networking home terminal. With a price of $0.83 on Mouser, this regulator will suit our needs.

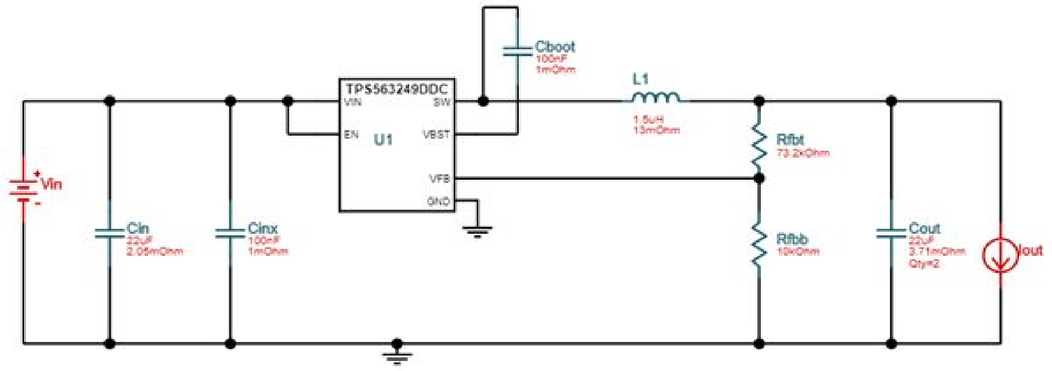


Figure 18: TPS563249 schematic IC [D2]

Similar to the TPS563231, the input voltage range is between 4.5 and 17 volts, with an adjustable output of 0.6 or 7 volts. It uses a forced continuous conduction mode instead of a pulse skip mode and a smaller low shutdown current of 10µA. The protection is also a hiccup mode to limit any overcurrent. Importantly, the switching frequency is much higher at 1.4-MHz which helps reduce component size but in return increases the cost to $0.95 on Mouser. With a SOT-23-THIN package, typical applications include a broadband modem, wireless routers and set-top boxes. This IC could be more useful for the higher switching frequency, but not necessary. This input voltage range could also limit us if we wanted to boost a 3.6 volt cell to five volts depending on our final design. Overall though, we can easily convert a higher voltage to a lower one based on our needs.

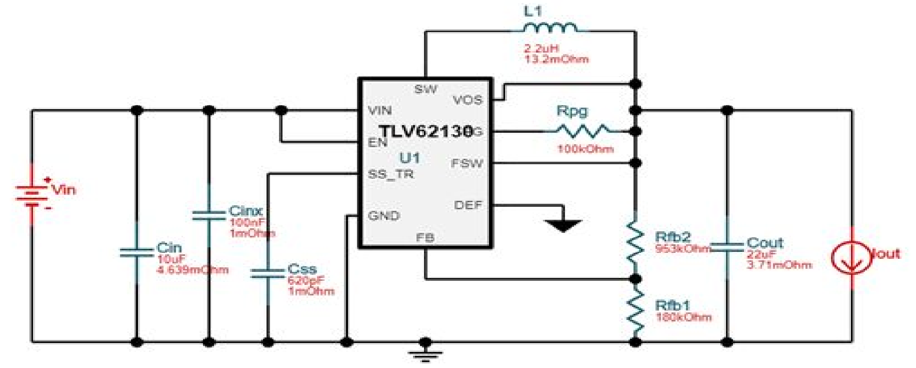


***Figure 19: TLV62130 schematic IC* [D3]**

The regulator above has an input voltage range like the other two of 3 to 17 volts and an output range of 0.9 to 5.5 volts. With an output current of up to 3A and a quiescent current of 19µA. It has both a short and over temperature protection and comes in a 3x3 QFN package. With a good power output, the typical applications are 12-V rail supplies, motor drives and set-top boxes. With a frequency of 1.35MHz and efficiency of 93.8%, the cost on Mouser is $1.49. Overall, this regulator has too many features that won't be utilized but makes for a great power/efficient PCB. During the testing and design process, if we switch up the power supply design to achieve greater long term results, the input voltage range is just low enough to allow for a one cell design to power a set of diodes. Currently, below figure is a depiction of the initial plan, but can see significant changes as we look for better results. With that, the TLV62130 offers great advantages over the other two ICs, while still giving a great price point and ability to test different battery pack methods.

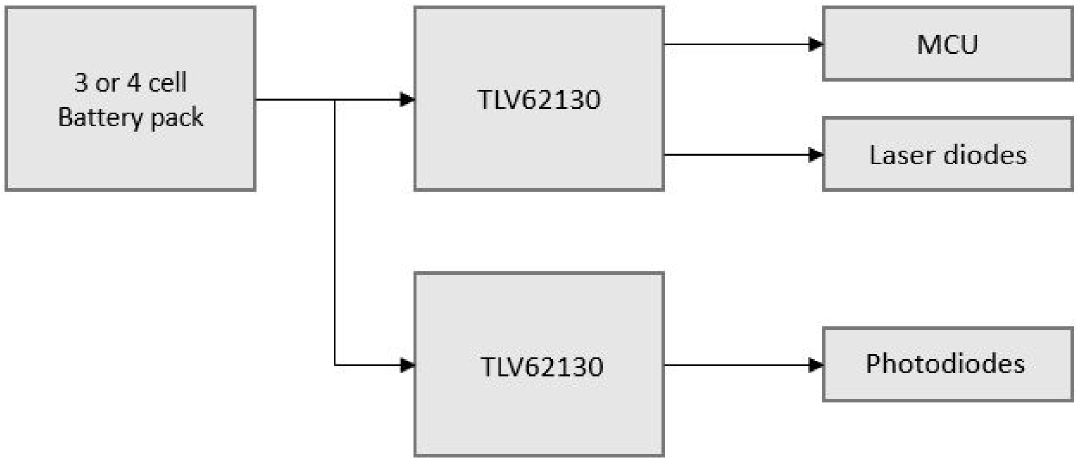
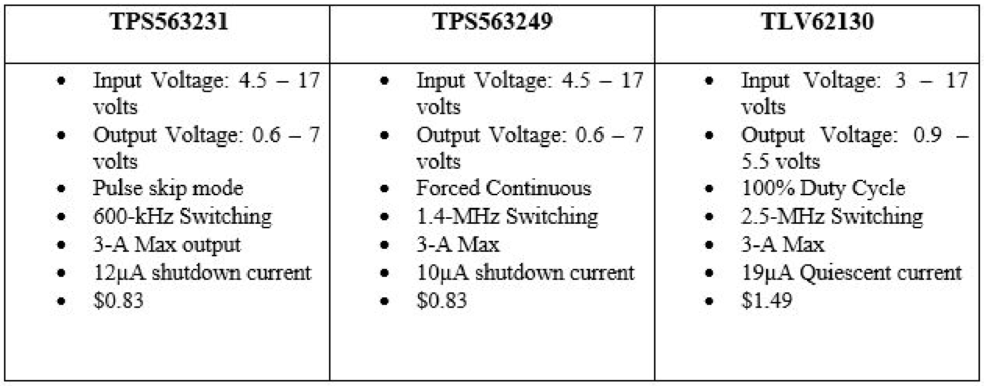


Figure 20: Diagram of power supply through regulators to electronics

In the figure above is the current plan for power distribution to current parts we have selected. With a 10.8- or 14.4-volt battery pack, the regulators chosen are the TPS63249, for its range in output voltage and the very fast switching frequency compared to the TPS563231, which is also relatively similar in cost. There are currently three components being tested before placement/wiring on/to the PCB, which are the MCU, laser diodes and photodiodes. Both the MCU and laser diodes have an operating voltage of 5V while the voltage for the photodiodes will be determined at a later point with more testing. With all three regulators being chosen for its advantages over being a switching regulator instead of linear regulator. The linear regulator chosen for testing purposes has less noise while the switching IC for final design replaces the ripple on the input for noise on the output, but is also easy to overcome. For the PCB, components can cause ground bounce and ringing, which can lead to bit rate errors for a large number of ICs, so not necessarily in our case for our limited design. For issues that do arise, bypass capacitors can be implemented between the power and ground pins on each IC. Overall, the signal integrity will remain strong for these ICs as more issues occur for a larger number of components.



TPS563231	TPS563249	TLV62130
• Input Voltage: 4.5 – 17 volts • Output Voltage: 0.6 – 7 volts • Pulse skip mode • 600-kHz Switching • 3-A Max output • 12µA shutdown current • $0.83	• Input Voltage: 4.5 – 17 volts • Output Voltage: 0.6 – 7 volts • Forced Continuous • 1.4-MHz Switching • 3-A Max • 10µA shutdown current • $0.83	• Input Voltage: 3 – 17 volts • Output Voltage: 0.9 – 5.5 volts • 100% Duty Cycle • 2.5-MHz Switching • 3-A Max • 19µA Quiescent current • $1.49

Table 7 - Voltage Regulator comparisons

3.4.3. Printed Circuit Board

With our project idea, a printed circuit board will be designed to encompass all the electronic components once testing has completed. A printed circuit board is essentially a custom board that allows electronic components to connect via conductive tracks to power a device. Consisting of laminated sheets and a non-conductive substrate. An important feature of a printed circuit board is to choose between having through hole or surface mount electronic components. For efficiency and low cost, surface mount allows components to be soldered onto metal caps instead of through the printed circuit board.

<u>Properties/Material</u>

The lamination process is done by curing layers of cloth or paper with resin under high pressure and temperature. Some important characteristics when determining the ratio between the cloth and resin is the dielectric constant, loss factor, tensile strength, and importantly thermal expansion. There are many variations used in industry today, but the most common is FR-4, which is woven glass and epoxy. With these substrates, it is important to mention that they can be woven or nonwoven, with woven being cheaper with a high dielectric.

Before printed circuit board boards were constructed, the process to construct circuits was known as point-to-point wiring. Failures often occurred due to aging, which caused a development into wire wrapping, where a wire is wrapped around a post for each connection. As technology evolved and silicon was used more and more, the printed circuit board was created to reduce costs to match the electronics. Below is an image of the composition of a printed circuit board where you have the substrate, copper, solder mask and silkscreen.

As mentioned above, the substrate is FR4, which is fiberglass, giving the printed circuit board its thickness. After the substrate, copper is applied to both sides, where the thickness varies but a common amount is 1 oz. With high power electronics, the amount may increase. On top of the copper is the solder mask, which is the green/red color you usually see on the printed circuit board. It helps insulate the copper to prevent contact with other metal. The silkscreen is then applied on top of the solder mask which denotes contacts as either power or creates letters or numbers. Usually letting the user know the function of each pin. Some common manufacturers include TechnoTronix, RedBoard Circuits, NexLogic, and A.C.T. Each company offers a wide selection of processes with great costs on many forms of PCB's. Ranging from industry such as military or oceanographic to the specific application.

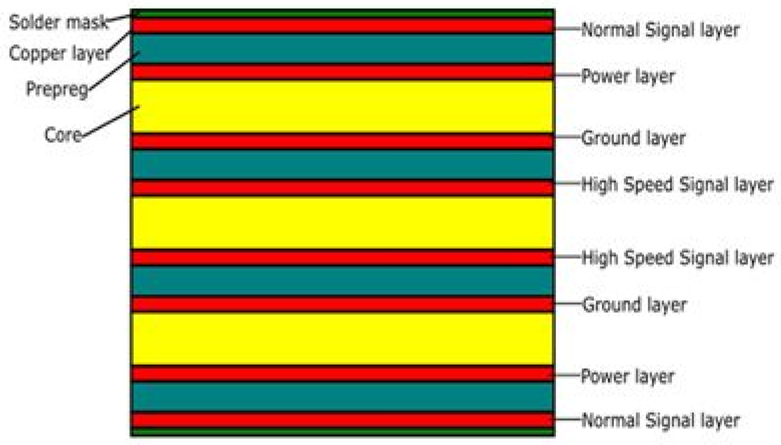


Figure 21: Common layers of a printed circuit board side view [D10]

Terminology

When it comes to printed circuit board designs, they all have one thing in common and that's the terminology used to describe different parts of the board. The first term is DRC, which is an acronym for design rule check. Some common checks are things such as traces touching incorrectly, or they can be too thin. The next term is annular ring, which essentially describes the ring of copper surrounded by a plated through hole. With that, a drill hole is where a hole should be drilled, which accommodates a plated through hole for through hole parts instead of surface mounted parts. An odd one is the finger, which are the exposed metal pads around the printed circuit board to allow connections from one printed circuit board to another. A pad is exposed surface metal to allow surface mounted electronics to be soldered on. A couple important terms for when actually soldering components to a board, include reflow which is melting the solder to form joints and solder paste which are balls of solder in a gel applied to the printed circuit board before electronics are placed. It is a substitute to the actually using solder, which requires more of a steady hand and additional parts such as a soldering iron to melt the solder iron onto the pads as you place the components. A final note that everyone has their own preference and may choose whatever method they desire to apply their electronics.

Manufacturing

In the beginning, printed circuit boards were constructed by hand from a photomask on a mylar sheet. From the schematic, the metal pads and traces were made on the mylar, with the use of self-adhesive tape. Today, computer software is used and starts with the printed circuit board data being read by the CAM (Computer Aided Manufacturing). From this process, if there are multiple printed circuit boards, they could be grouped together and prepare the printing process. Two of the processes involved are photoengraving and silk screen printing. The photoengraving involves removing a UV photoresist to create a mask, while the silkscreen uses etch-resistance inks to create the mask. Once imaging and etching is complete, all layers are laminated together with drilling taking place to electrical connect each layer. The boards are then coated with a solder mask and machined to the specified dimensions.

Types

When you're choosing a printed circuit board manufacturer, you have the option to choose from a variety of printed circuit boards such as having a rigid board with a single layer or having multiple layers with a flexible board. The rigid version is usually constructed from fiberglass as the substrate and is inexpensive, but less versatile. The flex models offer greater versatility with a higher cost, but if you're needing a printed circuit board to fit in a very confined space or adjust over time, these will work well. With a greater resistance to heat, thermal expansion is less of an issue. An alternative is a hybrid model, where multiple rigid printed circuit boards can be connected with the flexibility component from the flexible printed circuit boards, while keeping a great size and maintaining durability during its lifetime.

Our Process

During the testing phase and finalizations of components for our design, a printed circuit board will be designed to encompass everything. The software used during this step will be Autodesk Eagle, but there are a couple others such as KiCAD and Fritzing which are highly rated. Once testing and design is finalized, the selection of the PCb manufacturer will be chosen and our design will be optimized to allow the lowest cost.

3.4.4. Integrated Circuits

This section covers integrated circuits involving ADCs and DACs. The research covers a broad range of concepts involved to understand the electrical functionality of the process to convert between analog and digital signals.

Analog-to-Digital Converter

For an Analog-to-Digital converter, an analog signal like detecting when a laser is broken is then converted into a digital signal. As an example, on the fretboard of our design, there will be a set of lasers pointing into photodetectors. To recognize when the photodetector stops receiving photons, that analog signal can be categorized as a one and when the photodetector still receives photons, that will represent a zero. Along the fretboard, different notes can be played. Depending on the distance when your finger breaks the sight, the number of reflected photons will be picked up by the photodetector, which the microcontroller will assign a digital value corresponding to a note being played.

When understanding ADCs, here are some terms to explain its operation. The resolution of an ADC is the amount of binary values that can be produced over an analog range. When it comes to resolution, Dither is used to help improve the overall performance. Essentially, a small random noise is added to the input, which in turn helps the LSB from getting cut off. Calculating the accuracy so far has not influenced, but the most common errors deal with non-linearity and Quantization. Usually resolved by some type of calibration of the IC. The last important term to understand is the sampling rate, which is basically the rate at which new values are sampled. When a signal is sampled, the input is held constant for a single point during a conversion time while a sample and hold task takes place with a capacitor. Usually oversampling takes place when the ADCs performance can be greatly increased.

There are also different types of ADCs such as a direct-conversion and successive approximation. For direct conversion, the sampling rates are fast but with a low resolution, usually at 8 bits. Comparators sample the input and a logic circuit is used to create the output for the voltages used. The second approach is successive approximation, which also uses comparators. At each step through the process, an output range is compared to the input that is stored. The input voltage is compared to voltages above and below it to narrow down the field. There are also many applications for an ADC such as in DSP applications and sound recording.

When it comes down to our overall design, the purpose here is to measure four signals per each of the four strings, which encompasses a total of 16 notes for our ukulele design. The voltage drop across each resistor must also be a large enough value for the MCU to recognize and assign a note value. Overall, there are many advantages and disadvantages that play an effect when it comes to ADCs.

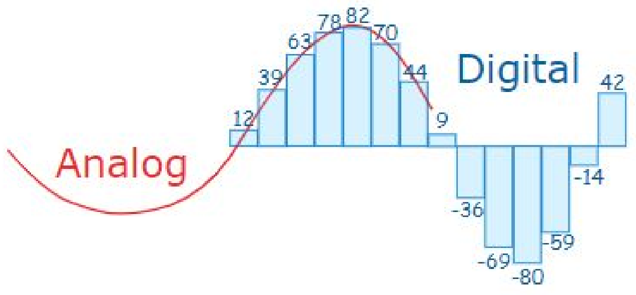


Figure 22: Digital output taken from analog input [D11]

Digital-to-Analog Converter

As with a Digital-to-Analog Converter, the opposite occurs from an ADC where you use the digital signal from the microcontroller and produce an analog output such as a musical note in our case. When discussing DACs, there are some figures of merit to consider, resolution and maximum sampling frequency. For our instance, our analog signal from the lasers on the fretboard and head of the guitar, will represent different digital signals processed by the microcontroller, including different notes per string. These digital signals will then take advantage of the DAC and play each note to the speaker of our choice. A good representation of a DAC would be a summing amplifier as shown below. The strings will represent each bit and different levels on each string will correspond to different notes.

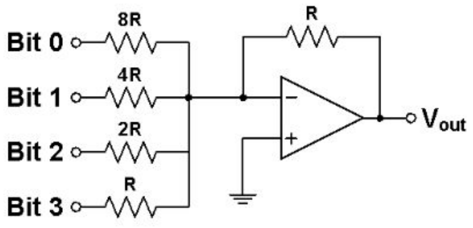


Figure 23: Summing amplifier shown to represent a DAC [D12]

The applications of a DAC range from audio to video and mechanical/communication. The mechanical is interesting where you have an actuator which has multiple positions to represent a bit. Communication as always is popular with mobile devices and cell towers. With applications, there are also different types of DACs, such as a pulse-width modulator, binary-weighted, and oversampling. The oversampling uses a delta-sigma modulation process, allowing a signal to be sampled more accurately. With the binary-weighted model, an op-amp is used for each bit, which mimics a summing amplifier as shown above. With its extremely fast response time and high precision, the downside is the lack of accuracy.

Pulse-Width Modulation

This technique allows for a signal that is in a high or low state to vary the time at which the state occurs, which essentially changes the proportion between each state. In the below figure, you can see when the signal is high and when it is low. The term duty cycle is a percentage that describes the amount of time a digital signal is on over a time interval. The 50% refers to the signal being half the time off and half the time on. While 75% is three quarters on and a quarter off. With 25% being the inverse of that. A good approach to visualize how to calculate a PWM signal, is to compare a sine wave with another waveform and look at the intersecting points of the sine wave in terms with your signal in its high state and low state. The duty cycle is then used and altered based on what the customer or engineer needs.

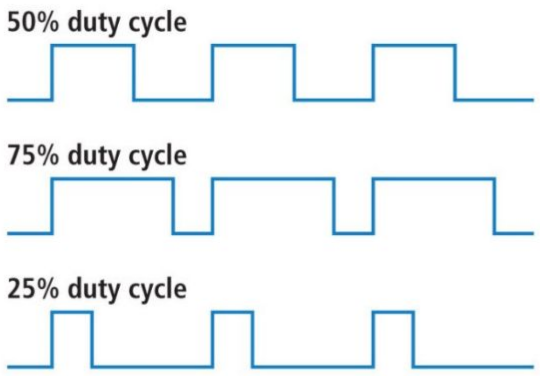


Figure 24: Differences of graph between common duty cycle [D13]

Architectures

When it comes to analog-to-digital devices, there are many architectures used for different practices such as successive approximation, delta-sigma, and pipeline. With delta sigma, oversampling and filtering is used with advantages such as high resolution and stability but lacks in speed and latency. Successive approximation samples the input signal and uses iterative processes to determine the digital signal. With it, there is high accuracy and low latency but has a low sample rate range compared to the other two methods. Lastly, pipeline converters under sample the signal and uses multiple stages to give a higher bandwidth and speed, but compromises with more power usage and lower resolution.

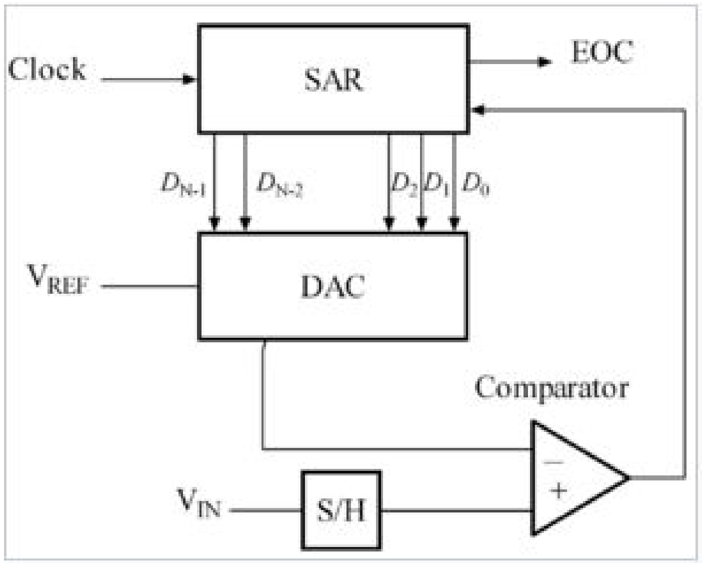


Figure 25: SAR depiction of architecture [D14]

In the figure above, there is a depiction of a common successive approximation ADC converter. The flow starts with the sample and hold block where the input voltage is measured and then fed into the comparator. The output of the DAC is used to compare against the input voltage, so the N bit number into the DAC will determine the voltage through a multiplication process depending on the current value.

For each clock cycle, the process is such that the input voltage is used to determine whether the output of the DAC is larger or smaller, which will result in a zero or one respectively. There is also a cool approach that uses capacitors, where the capacitors are then charged/discharged to feed a value into the comparator. Essentially working on the same scale as shown above.

For digital-to-analog devices, the architectures include delta sigma, string and R-2R. With R-2R architecture, it is one of the older methods with low noise and high performance, but often skews the timing during the data process. The string method is really interesting where it has a set of switches and resistors built around a voltage reference and buffer. This gives great advantages where it is cost effective with great DNL performance. Though, the resolution is limited, and the area is rather large compared to the others. As with delta sigma, mentioned above, it's a reverse ADC. With low cost, high resolution, and low power.

Overall, the ADCs and DACs play a vital role in recognizing a high voltage and playing a corresponding note in our case. Once the analog signal from the laser to photodiode is converted to a digital signal, the DAC will take the digital signal from the MCU and play the programmed note per string. Different architectures as mentioned will aid in this design process.

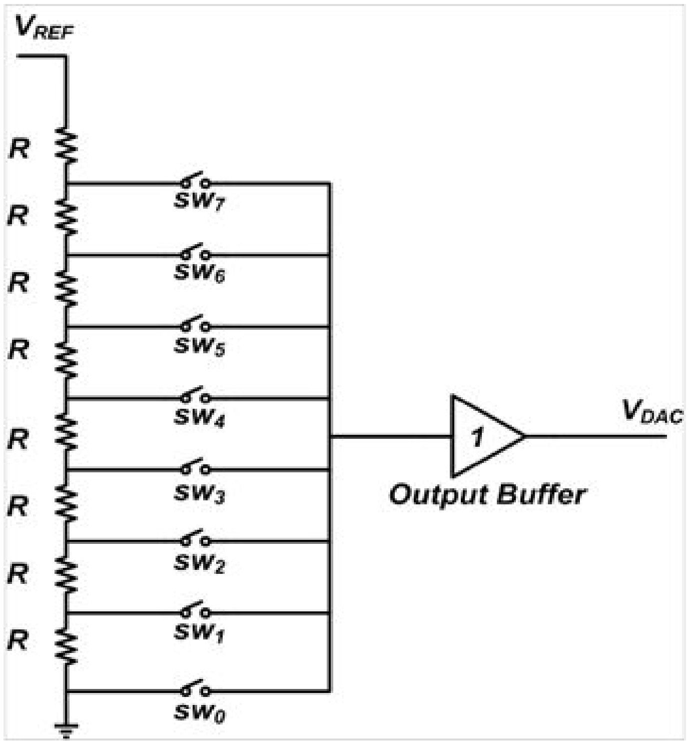
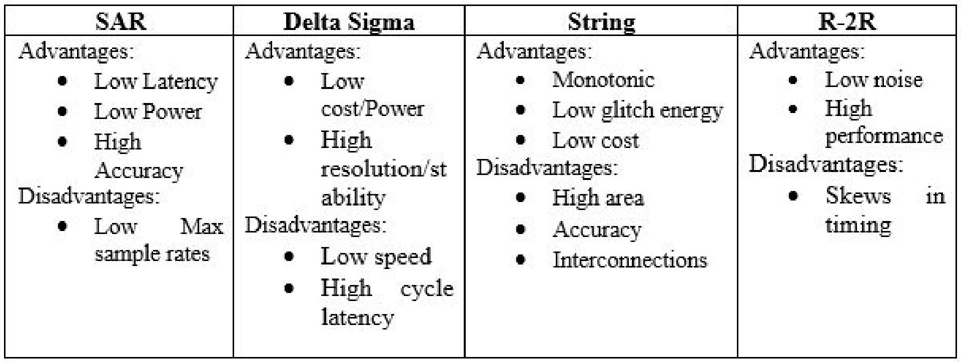


Figure 26: Depiction of the switch/resistor string architecture [D15]

In the figure above is the depiction of an eight switch/resistor string that produces eight different voltages. Also, since there are eight switches, it's a three-bit device. Depending on the reference voltage, each resistor value is the same which will give an output such as V_{ref} /2 or V_{ref} /4, by turning on the respective switches. The output buffer then helps prevent extra effects brought on from V_{DAC}. Though now the size will only increase as you increase the number of resistors 2^n and the switches.

Internal/External

With both external and internal ADCs, there are many compromises a designer must choose. Specifically, for an internal ADC such as those on a microcontroller, you're mixing an analog device with a digital one. The main drawback for internal ADCs is the slow speed/performance needed for high end analog applications. For reference, the mixed-signal processes used must be paired with slow logic at around 1MHz. How this works starts with the reference voltage being used form the power supply and to limit the amount of noise interaction to the digital logic. All in all, as a designer, if speed and accuracy is an issue, then an internal device may be appropriate, but if the application is very demanding with data, an external device will be sufficient.



SAR	Delta Sigma	String	R-2R
Advantages: • Low Latency • Low Power • High Accuracy Disadvantages: • Low Max sample rates	Advantages: • Low cost/Power • High resolution/stability Disadvantages: • Low speed • High cycle latency	Advantages: • Monotonic • Low glitch energy • Low cost Disadvantages: • High area • Accuracy • Interconnections	Advantages: • Low noise • High performance Disadvantages: • Skews in timing

Table 8 - Comparisons of Architectures for ADC/DAC

3.5. Architectures and Implementation Factors

This section focuses on what architectures and implementation factors to consider. The architectures and implementation factors consist of, but not limited to the following topics.

Abstraction Methodology

When designing various components within an embedded system, the methodology of developing an embedded system consists of abstractions. The term abstractions in context to embedded systems can be defined as the process of simplifying or modifying a complex system. By simplifying or modifying a complex system, the process gives developers information of what components of a system is doing or the overall design. The design of an embedded system consists of two components, the hardware and software. In order to understand how our microcontroller will work with our code, we need to understand abstraction methodologies and concepts of both hardware and software abstractions. This will provide an understanding of how written code is implemented to the hardware from a software perceptive. And from a hardware perspective, it will show how the hardware will process the given code written by the software.

Hardware Abstraction

A key component to an embedded system design is the hardware. The hardware of an embedded system is composed of many different components. These components can be, but not limited to the microcontroller, sensors, resistors, capacitors, transistors, and power supplies. The methodology of abstraction is relevant to the hardware of an embedded system because as a developer we must design and build these components. There will be various components that are dependent on one another which can eventually become massive. Having massive amounts of components can create complex systems which can be confusing for other developers to follow along. By understanding hardware abstraction, it can help break down complex hardware components into pieces. As components are built into subsystems of a device, the overall perspective of the design can be overwhelming. By breaking down complex hardware components into elements, developers can have a simplistic view on what components can be utilized by the software. Hardware abstraction can be defined as a layer of code that allows communication between a system's software and hardware. This layer is called the hardware abstraction layer (HAL). The hardware abstraction layer allows developers to write platform-specific and device-independent applications by providing operating system (OS) routines that have direct access to both hardware resources, and system peripherals.

Having the ability to gain direct access to a hardware's resources and a system's peripherals gives developers the freedom to customize their needs while maximizing the hardwares features. In order to write code that will be implemented to the hardware abstraction layer, is that it needs to first be accessed. The hardware abstraction level can be accessed from either the OS's kernel (computers core) or from a device driver. When the hardware abstraction level is accessed by either the kernel or driver, there are many techniques to append code to the hardware abstraction level. An efficient and reliable way of appending changes to the hardware abstraction level is virtualization. Virtualization is the process of running a virtual instance of a computer system in a layer abstracted from the actual hardware [8]. Virtualization uses a virtual machine, which is an operating system independent from the kernel and hardware to run code on. Using techniques such as virtualization, helps map out virtual resources of the hardware to the physical resources and uses the actual hardware for computations inside a virtual machine. When the virtual machine is running, the machine needs to communicate to the physical resources specified in the code. Once it has made a connection, the simulate takes over and multiplexes appropriately and appends the code.[S90].

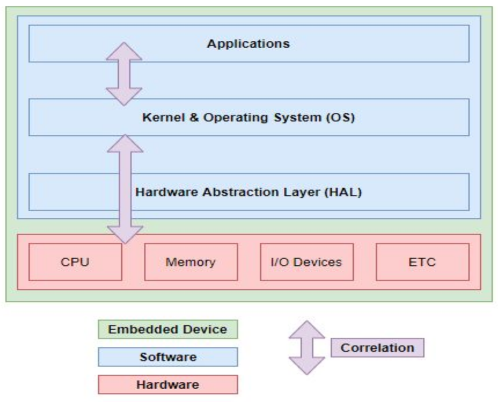


Figure 27: A Hardware Abstraction Layer Example.

In figure 26 is a visual representation of how the methodology of abstraction is implemented within an embedded system. The Applications, Kernel & Operating, and hardware abstraction layer comprises the software components of the embedded system. The CPU, Memory, I/O Devices, and ETC (Other hardware component) make up the hardware components. As presented, the hardware abstraction layer is positioned in between the “Kernel & Operating System” and “Hardware”. This is positioned in between them because the hardware abstraction layer is a layer of code that allows communication between a system’s software and hardware. There's no correlation arrows to indicate how it affects the kernel or hardware because it encapsulates most hardware-specific functions that are performed by the operating system. If another portion of the operating system wants to access a hardware device, it must refer its request to the HAL. The HAL handles communication between the kernel of the operating system and the hardware. [9]. The hardware abstraction layer helps developers avoid the need to fully understand how the hardware works. This pushes developer’s to focus more on the software to maximize the various capabilities that each hardware component has to offer.

All of this process happens seamlessly, through a software environment. The software environment will take the code written by the developers and append it to hardware accordingly. For our project design we must take hardware abstraction into consideration because without the methodology of hardware abstractions, the design would accumulate more constraints, and exhaust more resources than needed. The hardware abstraction layer removes the process of hard-coding drivers, kernels, and application programming interface to every component of a hardware device. When designing the hardware for the project, the hardware components should be strategically categorized based on if the component needs to be influenced or improved by software. If a component is heavily influenced by software, then the hardware abstraction layer will provide a fluid integration when we develop the code for it. This allows the team to shift our focus more on how the software should be written in order to achieve our requirement specification and meeting realistic design goals.

Software Abstraction

The second key component to an embedded system design is the software. The software of an embedded system is composed of varying software technologies and design implementations. These varying software technologies and design implementations can be, but not limited to databases, life cycle developments, servers, application programming interface (API), design patterns, time and space complexity, and programming languages. The methodology of abstraction is relevant to the software of an embedded system because as a developer designs and codes the software of the system, there will be various elements of codes that are dependent on one another. These dependencies are critical to the functionality of how the hardware will interpret it. An abundant amount of codes and dependencies can create complex design implementations which can be difficult to debug and refactor. By understanding software abstraction, it can help simplify complex code into readable and usable elements.

Large complex codes can hinder the readability of the code when being analyzed by other developers. Not only can it hinder the readability of the code, but it can also waste memory due to how large the code size is. These handicaps of writing large complex codes can be ameliorated by applying the methodology of abstraction to software. Unlike hardware abstraction which consist of one type of implementation, there are many types of implementations when discussing the scope of software abstraction. Software abstraction can be defined as the simplification of behavior and implementation of code. The simplification of behavior and implementation can be categorized into two types of software abstraction respectively; control abstraction and data abstraction. The software abstraction is categorized into two categories because the methodology of abstraction focuses on two different concepts when implementing abstraction into software. These two categories can be done individually or simultaneously. By understanding what control and data abstraction is, this can help the software development of the project to maximize resources under tight constraints.

One category of software abstraction is called control abstraction. Control abstraction is the process by which programmers define new control constructs, specifying a statement ordering separately from an implementation of that ordering [10]. By creating new control constructs, it hides implementation of complex execution flow and low-level processes. An execution flow is the process of running statements that contain code instruction on what to do. The execution flow process always executes the first statement of a program. After the first statement, the following statements within the program will execute one after another until it reaches the end of the program. This is the foundation of how code is executed within any compiler across all programming languages. Control abstraction allows developers to create a new way to control how statements are executed. Creating new constructs of ordering statements can allow developers manipulate the ordering of their code.

By creating new control constructs, it hides implementation of complex execution flow and low-level processes. Developers can also hide data information by using data abstraction. Data abstraction can be defined as providing only essential information about specific data within a set of code. Data abstraction can be applied by using programming paradigms such as Object Oriented Programming. Using Object Oriented Programming, a user can hide information within an object. An object can enclapules all of its variables and methods, and hides it’s code from a program. Then users can work an object and access its properties to perform a task without ever needing to declare new variables or functions that can increase the code size. The increase in code size can become unreliable and hard to maintain. By using data abstraction, developers can maintain their code while making it modular.

3.6. Parts Selection Summary

This section is our selection of parts and components. The section consists of the parts and components that we will actually use in our design from both a hardware and software perspective.

3.6.1. Hardware Selection

This section focuses on primary all of our hardware parts and components. Each of the hardware components and parts were selected based on our research and strategic component consideration.

Laser and Photodetector Selection

The hardware design of our project will be largely dependent on the detection and disruption of the laser strings. In order to obtain optimal performance from our strings, while still adhering to the design restraints of safety, size, and cost of the device, the type of laser and photodiode we use for this detection is imperative. There is an abundance of options for us to consider for both the type of laser and photodetector that we will use, so the following sections are dedicated to selecting the best possible option for what type of laser and photodiode we use in our project.

Laser Selection

The laser should be low cost, small, safe to use, and operate within the visible light spectrum for a more intuitive user experience. Our options include semiconductor lasers, gas lasers, fiber lasers, and dye lasers. Semiconductor lasers operate by using the concept of inversion to produce coherent laser light when the semiconductor is sufficiently pumped. In doing so, a very small, cheap, and efficient lasing system is made, that is available for purchase on any major shopping site such as Amazon. Gas lasers operate by optically resonating light through a tank of gas which acts as the gain medium for the light to become more intense and coherent. These types of lasers are much larger than semiconductor lasers, and because of the size constraints of our project alone they can be ruled out of the question. Fiber lasers allow for interesting functionality by allowing the user to guide the laser light through an optical fiber. This has interesting applications, but because our project is just a linear optical system, it is unnecessary for our needs. Finally, dye lasers operate by pumping laser light into an organic dye which acts as the gain medium for the light. Clearly, lasers are designed in several different ways, however when considering the applications of our project the only viable option is the semiconductor laser, which is small, low cost, low power, and operate within the visible light spectrum of 400nm-700nm. For now, we have opted to go with the GeeBat Mini 650nm Laser Diode, which can be found on Amazon in groups of 10 for just $5.99. These laser diodes provide the functionality necessary for our project, operating at 5mW with a beam diameter of 6mm. This provides a small enough beam to fit multiple on a fretboard, lies within the visible light spectrum, and has enough power to provide sufficient lighting for our photodiodes while not being too dangerous to use for our project. The following section outlines our options for Laser Diodes.

GeeBat Mini 650nm Laser Diode

The first laser diodes we considered are the GeeBat Mini 650nm found on amazon. These laser diodes have a working voltage of 5V and produce a beam with a diameter of 6mm. The operating current of this laser diode must be under 20mA to work. The beam has a power of 5mW and the diodes are available in groups of 10 for $5.99 on Amazon. These diodes were considered due to their availability, cost, and functionality. They operate within the visible light spectrum at a power low enough to be safe to use and are cheap enough to order in abundance, making them a strong contender for our laser diode.

Lights88 532nm Laser Diode

The next laser diode considered had similar features to the GeeBat Mini Diode, however there are some crucial differences to consider the use of this diode. First, having a wavelength of 532nm must be considered when using these in conjunction with our photodiodes. If this is closer to our photodiodes peak wavelength, this is worth taking into consideration. The operating current of this diode also ranges up to 250mA, giving us much more freedom with what power we choose to operate the laser diode at. The Lights88 Laser Diode can operate at powers from 5mW to 50mW, depending on how much voltage is applied to the diode. This is valuable in case we need to draw more power from our photodiodes, we would have the option of increasing the output power of our laser. This diode is significantly larger than the GeeBat Mini 650nm, at roughly 1 inch in length and 1cm in width, which can also impact our decision. The cost of the Lights88 Laser Diode is $13.34 for one diode, which is significantly more than the GeeBat Mini which is 10 for $5.99.

Civil Laser 405nm Diode Laser

The Civil Laser 405nm offers a good balance of pricing, output power and size, which are all crucial parameters to consider in our project. Similarly to the previous two diodes, the Civil Laser 405nm has an operating voltage of 5V. The output power of the Civil Laser 405nm is 20mW, which is a good middle ground between the other two laser diodes. The wavelength of 405nm is important to consider in conjunction with what photodiode we select, and the size of the diode is small enough to fit comfortably on our guitar, at a size of 12x15mm. This laser diode also runs for a very cheap price, at just $10 for an order of 5 diodes. This option was strongly considered due to meeting our requirements for size, price, and being within the visible wavelength spectrum, however we had to rule it out once we began looking for photodiodes and found that photodiodes with sensitivity in the 400nm range were very expensive. The final option we went with is described in the section below.

Final Laser Diode Comparison and Selection

The table below shows the types of laser diodes we chose from.

Laser Diode	Price	Size	Wavelength	Output Power
GeeBat Mini 650nm	$0.59	8x12mm	650nm	5mW
Lights88 532nm	$13.34	1inx1cm	532nm	5mW-50mW
Civil Laser 405nm	$2	12x15mm	405nm	20mW

Table 9: Laser Diode Comparison

After looking into each of our options for laser diodes, and taking into account the photodiode we selected, we chose to go with the GeeBat Mini 650nm. These laser diodes were not only the most readily available, as with Prime shipping they could be shipped overnight with Amazon, but they were also the cheapest and smallest of the laser diodes we looked into. The downside is that the maximum output power is only 5mW, however this actually helps us with keeping our device from becoming too bright and staying within our safety constraints. Because we ended up selecting the Vishay BPW34 photodiodes, the GeeBat Minis were also the best option due to the wavelength being closest to the Vishay's peak wavelength. This will help us achieve a more clear signal and make the software design easier for our software team. Another factor that we took into consideration is the size of the laser diode. Because the GeeBat Mini 650nm was the smallest option available, we were more inclined to pick this device. This was confirmed when we decided we wanted to use the body of a ukulele rather than the standard or bass guitar body, as this was the smallest body we had to choose from. Other options of wavelengths would require more expensive and harder to find options for photodiodes in most cases, so we decided to go with the 650nm option, where related components were found abundantly for cheaper price.

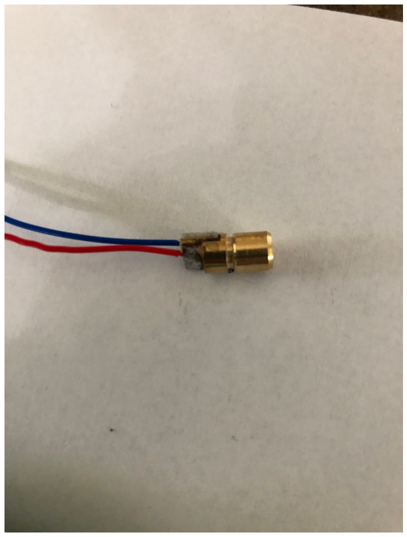


Figure 28: The GeeBat Mini 650nm Laser Diode

Photodetector Selection

The photodetector should be cheap, precise, sensitive to change in incident light, sensitive to light within the desired wavelength, and able to detect power up to our maximum laser output power. The various types of photodetectors include phototubes, photovoltaic solar cells, phototransistors, and photodiodes. By briefly observing the functionality of each of these options we can deduce which should be used for the optimal functionality of our project. Phototubes are a gas filled or vacuum tube that is sensitive to light. Incoming photons strike a photocathode, which knocks electrons onto an anode creating a current depending on the intensity and frequency of our light. Next we will observe the option of photovoltaic solar cells. These photodetectors create a flow of electrons by absorbing light emitted usually from the sun which excites electrons in a p-n junction causing electricity to flow.

Photovoltaic cells are generally expensive and offer lower efficiency than other options, such as the photodiode. Photodiodes operate by taking advantage of the photoelectric effect, where absorption of light causes an electron hole pair to be produced. If this absorption occurs in the depletion region of the p-n junction carriers are swept across the junction resulting in a photocurrent. Photodiodes can operate in either photovoltaic mode or photoconductive mode. A photodiode operating in photovoltaic mode has no external voltage applied to it and operates the same as the photovoltaic solar cells that were previously described. When the photodiode is reverse biased, only the very low dark current of the photodiode is allowed to flow until light is present on the photodiode. When light is incident on the photodiode, the photocurrent increases dramatically. For the purposes of this project, we will use PIN photodiodes in conjunction with our semiconductor laser to return information to the microcontroller. This is because PIN photodiodes offer high sensitivity at a very low cost.

By using our photodiodes in photoconductive mode we will be able to detect the presence or absence of light in our project. The photodiodes will not allow much current to flow when no light is present resulting in a low returned voltage reading to the microcontroller. When light is present on the photodiode in photoconductive mode current will flow resulting in a readable voltage being returned to the microcontroller. The following section outlines our options for photodiodes.

Vishay BPW34

The Vishay BPW34 is a PIN photodiode offered on Amazon in groups of 10 for just $7.99. The data sheet of the BPW34 lists the device as having a dark current ranging from 2-30nA, and a reverse light current of 40-50μA. By using a voltage divider with resistors on the order of 10-50kΩ, this would allow us to easily draw enough voltage to send a signal to our MCU given enough light incident on the photodiode. The rise time and the fall time of this photodiode are both 100ns. The spectral range of the Vishay BPW34 is from 410-1100nm, which unfortunately means it will not be compatible with the Civil Laser 405nm Laser Diode if we chose to go that route. The spectral sensitivity of this photodiode is shown in the figure below. The peak sensitivity of the BPW34 is 900nm, which is outside of the visible spectrum, and therefore not ideal for our purposes. However, the availability, cost, and size of this photodiode still makes it a strong contender for our project.

OSRAM SFH 2401

The OSRAM SFH 2401 is a PIN photodiode available in packs of 10 for $14.60. This photodiode was considered due to its enhanced sensitivity to blue and green wavelengths of light. The peak sensitivity of this photodiode is 950nm, and the spectral range is from 300nm-1100nm, which covers every option of laser diodes we are looking into. The dark current ranges from 1nA to 25nA. The spectral sensitivity of the device is shown in the figure below. The rise and fall times of this photodiode are both 0.04μs. The wider spectral width and an enhanced sensitivity to shorter wavelengths made this another contender for our project.

Final Photodiode Comparison and Selection

Taking into account the speed, price, and compatibility with our laser diodes, we decided to purchase the Vishay BPW34 Photodiodes for our project. This does unfortunately rule out the ability to use wavelengths of light below 430nm, however when looking into various photodiodes we found that finding a photodiode with high sensitivity at near ultraviolet wavelengths was significantly more costly than finding photodiodes with high sensitivity for near infrared wavelengths. This is what led us to choosing the Vishay BPW34 and the GeeBat Mini 650nm, as the cost and sensitivity of lasers and photodiodes operating at near infrared wavelengths was significantly lower than other wavelengths.

Photodiode	Cost	Rise/Fall Time	Spectral Range	Peak Sensitivity
Vishay BPW34	$0.79	100ns	410-1100nm	900nm
OSRAM SFH 2401	$1.46	40ns	300-1100nm	950nm

Table 10: Photodiode Comparison

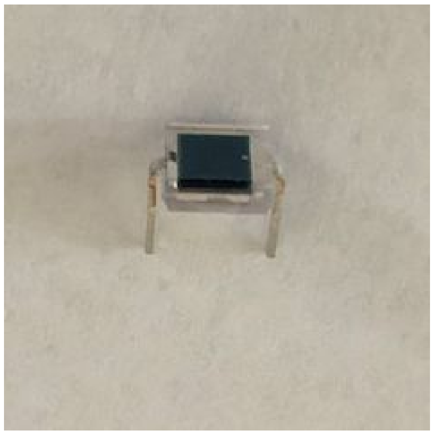


Figure 29: The Vishay BPW34 Photodiode

Speaker Selection

The speaker is another important component in our project, as we need to ensure the device provides a loud enough output so that when it is played it is audible to an audience in a small room. In terms of signals processing, our speaker will be in charge of receiving an electrical signal, processing the signal, and outputting an audible sound for the user to hear. The main factors that went into selecting our speaker are price, power consumption, size, availability, and sound quality. Taking these factors into consideration, we decided to purchase the MakerHawk Arduino speaker, which is designed to work with arduino speakers via their convenient 2-pin interface. They operate under either 3.3V or 5V and are found for $4.99 on Amazon. They are 31mm long, 28mm wide, and 15mm thick, which is small enough to ensure that we will have plenty of room to house them in our project. These speakers attach directly to pins of our microcontroller and provide high quality sound output according to the software programmed into the microcontroller.

Figure 30: The MakerHawk Arduino Speaker

Microcontroller Selection

The type of microcontroller that we use is crucial to our design due to the fact that it brings our software and hardware designs together to make the instrument play sound in a user-friendly way. The microcontroller will process the signal from the hardware, store this signal as memory in the software, and send the proper signal to the speaker output to play sound. Because the microcontroller impacts both the hardware and software design of the project, we carefully selected the best possible microcontroller for our purposes. From the hardware perspective, we need a microcontroller with enough I/O pins to account for the large number of photodiodes that will be used to transmit information.

Specifically, our current model involves the use of 16 photodiodes for the fretboard alone, with another 4 photodiodes added when considering the strumming strings as well. We are also considering modulating our laser diodes for the fretting strings, which would require an additional 4 pins for modulation. These strings need to communicate the information to a speaker, which adds at least one additional pin for a total of at least 25 pins to cover the strings alone. Another requirement of our microcontroller is that it includes a digital to analog (DAC) and analog to digital converter (ADC).

The DAC and ADC are necessary because our project requires us to gather light as an analog input and use this input as digital information and convert it back into analog sound at the output. While the amount of analog inputs would be a strict requirement if we were using the difference in intensity to determine the note being played, by using an array of photodiodes which simply return high voltage or low voltage we can use digital inputs more freely. From the software perspective, the microcontroller should be programmable in languages our computer engineering team is comfortable with, such as C, C++, or Python. Taking these constraints into consideration led us to the following list of options.

ATmega2560

First on our list is the ATmega2560. This microcontroller offers 54 I/O pins with 16 analog inputs. The 16 analog inputs are connected to a 16-channel analog to digital converter making them ideal for use with our laser strings. The total amount of 100 pins is more than enough to provide that all four strings with four photodiodes will have a usable input, while still accounting for our power supply and any other unaccounted-for inputs. The ATmega2560 is also very power efficient, drawing only 500 microamps when running in active mode at 1 megahertz and 1.8 volts. This power efficiency improves even more when the device is in sleep mode, drawing just 0.1 microamps. The ATmega2560 offers a clock speed of 16MHz at 4.5V-5.5V, which is fast enough to provide sufficient processing for a natural feeling experience.

The ATmega2560 is programmable with a language specific to Arduino which is based on C and C++ programming, which should ensure that picking up the language will not be a challenge for our software team. The price is currently listed as $15.99 on Amazon, which is reasonably cheap for the amount of features this microcontroller provides. The chart below summarizes the features of the ATmega2560.

Feature	**Value**	**Significance**
Price	$15.99	Cheapest option being considered
Clock Speed	16MHz at 4.5V-5V	Sufficiently fast information processing
I/O Pins	54 Analog I/O, 16 Analog Inputs	Enough pins to provide for all four strings
Power Efficiency	500μA at 1.8V, 0.1μA in Sleep Mode	Significantly more efficient than other options considered
ADC	16-Channel	Enough to Provide for all Four strings

Table 11: ATmega2560 Summary

AT91SAM3X8E

The next microcontroller we considered was the AT91SAM3X8E. This microcontroller offers an even faster clock speed than the ATmega2560 at 84MHz. On top of this, the AT91SAM3X8E comes preinstalled with a 2-channel digital to analog converter, which would allow us to output an analog signal without the need for an external DAC. This is very appealing because it would add ease to the workflow of outputting sound from a digital input that our instrument will provide. The pins on the AT91SAM3X8E are similar to that of the ATmega2560, offering 54 digital I/O pins and 12 analog inputs, with a total of 100 pins. Again, this is more than enough for the purposes of our project and allows us to be ensured that the number of pins will not be an issue. With all the similarities, and the advantage of the on board digital to analog converter, the AT91SAM3X8E seemed like the better choice for our project, until we looked into the power efficiency of this microcontroller. The AT91SAM3X8E draws 130 mA when running in active mode at 3.3V, which is significantly more than the ATmega2560's 500μA at 1.8V in active mode. In backup mode, it draws 25μA, which again is significantly more than the ATmega2560's 0.1μA in sleep mode. Also, the AT91SAM3X8E is currently running for $19.25 on Amazon, which is a bit more expensive than the ATmega2560.

Feature	**Value**	**Significance**
Price	$19.25	2nd most expensive option considered
Clock Speed	84MHz	2nd Fastest option being considered
I/O Pins	54 digital I/O, 12 Analog input, 2 Analog Output	Enough to provide for all four strings
Power Efficiency	130mA at 3.3V and 2.5μA in backup mode	Lower power efficiency than ATmega2560
ADC	16-channel	Same as ATmega2560
Additional Features	On board DAC	Eliminates potential need for external DAC

Table 12: AT91SAM3X8E Summary

Texas Instruments MSP-EXP430FR5994

The Texas Instruments MSP microcontroller offers 40 I/O pins with 8 analog I/O pins. This is again enough to suit all the needs for our project. The clock speed of the Texas Instruments MSP is 16MHz at 3.3V, which is slower than our other options. This microcontroller also draws a relatively large amount of power, at 100mA at 3.3V, which is less than the AT91SAM3X8E, but significantly more than the ATmega2560. The MSP offers a 12-bit Analog-to-Digital converter, which would be useful for our project's design. The table below summarizes the specs of the Texas Instruments MSP-EXP430FR5994.

Feature	Value	Significance
Price	$16.99	2nd cheapest option being considered
Clock Speed	16MHz at 3.3V	Same as ATmega2560
I/O Pins	40 I/O Pins, 8 Analog I/O Pins	Enough pins for our project, but less than other options
Power Efficiency	100mA at 3.3V	Better than AT91SAM3X8E and Raspberry Pi, but worse than ATmega2560
ADC	12-bit on board ADC	Eliminates need for external ADC

Table 13: Texas Instruments MSP-EXP430FR5994 Summary

Raspberry Pi 4 Model B+

The Raspberry Pi 4 Model B+ offers 40 general purpose I/O pins, which is enough for the purposes of our project. The clock speed of the Raspberry Pi 4 is 1.4GHz, which is by far the fastest of any of the microcontrollers we observed. The Raspberry Pi 4 Model B has a very high power consumption, at 200mA when operating at 5V. This microcontroller does not offer an on board DAC, which means a separate DAC would be required for our project. This microcontroller is also the most expensive of any of the microcontrollers we considered, at $35. The table below summarizes the specifications of the Raspberry Pi 4 Model B+.

Feature	Value	Significance
Price	$35	Most expensive of microcontrollers considered
Clock Speed	1.4GHz	Significantly faster than other microcontrollers considered
I/O Pins	40 General Purpose I/O pins	Enough to cover the needs of our project
Power Efficiency	200mA at 5V	Highest of Microcontrollers considered
ADC	None on board, but sold	Would increase cost of our

	separately	project

Table 14: Summary Raspberry Pi 4 Model B+

Our Choice – The ATmega2560

The table below summarizes the advantages and disadvantages of each of the microcontrollers for the comparison purposes. The selection of this microcontroller was largely due to the constraints the project's hardware design brought about. Because our system involves such a large array of photodiodes, we needed a microcontroller that could accommodate our needs. Our desire to make our project portable meant we needed to rely on battery power and therefore look into the most power efficient microcontroller possible. Breaking down our hardware before testing, we know we will have an array of 16 photodiodes for the fretboard alone, along with 4 more photodiodes for the strumming strings. These also must return a signal to at least 1 and at most 4 speakers, depending on the speaker's polyphonic capabilities. With just this knowledge, we know that we need a bare minimum of 24 inputs for our holistic system, and any extra inputs will just be reassurance that we have enough. The ATmega2560 offers enough functionality for our project, with 54 programmable I/O pins. On top of this, the ATmega2560 offers very low power consumption (1.8V: 500µA in active mode), and is relatively inexpensive for the functionality we are getting, at $15.99 on Amazon. Other options were either more expensive while still being less efficient, such as the Texas Instruments option, or just too expensive and offering too many unnecessary features, as in the Raspberry Pi 4 Model B+.

Microcontroller	**Advantages**	**Disadvantages**
ATmega2560	Good balance of speed, power efficiency, price and functionality	No significant drawbacks compared to other options considered
AT91SAM3X8E	Faster than ATmega2560, on board DAC	Relatively high power consumption
Texas Instruments MSP-EXP430FR5994	Good balance of price, efficiency, speed, and functionality	Doesn't bring anything to the table the ATmega2560 already had
Raspberry Pi 4 Model	By far the fastest	No on board ADC,

B+	processor of any options	expensive, high power consumption

Table 15: Summary of Microcontroller Options

3.6.2. Software Selection

Given that the microcontroller is at the root of our project and will ultimately be a key factor, our software selections were decided upon once the microcontroller was picked out. The microcontroller selected was the ATmega2560, which is most commonly found on the Arduino Mega 2560 microcontroller board. Our software selection is solely geared around this device and thus the programming language that will be used is Arduino's native language. This language was not in our programming languages research for the sole reason that it is derived from C/C++ and all C/C++ code will work in Arduino for the most part. The developers on the team are familiar with C/C++ and therefore, makes the Arduino language a good fit as a new language will not have to be learned. The Arduino language will give us the benefits that come with the C language such as memory management and bit manipulation, while also allowing us to incorporate C++ programming paradigm such as Object Oriented Programming.

In regards to the software environment we were able to narrow down our options to Arduino IDE, MPLAB X IDE and Atmel Studio 7. From our initial research these software environments are the ones that support the microcontroller we decided upon and work seamlessly with it. These three were downloaded and tested briefly in order to make a final decision. Atmel Studio 7 was eliminated due to the fact that it is only supported by Windows, and not all developers on the team own a windows operating system. We decided to go with the Arduino IDE because of its simplicity and it's high compatibility with the ATmega2560.

4. Related Standards and Realistic Design Constraints

The standards today provide specifications, procedures and examples for products and the use of materials. With standards coming from ANSI and IEEE, specifically for our design, the lasers are the key aspect, as well with the software and batteries. These standards help maintain characteristics and technical specifics of what a product or system entails. Listed below are a few that are useful for our project.

4.1. Software Standards

Following coding standards can have a significant impact in group projects and they are widely used in large organizations. Coding standards are generally specific to a programming language and can be described as a set of rules or guidelines, best approaches and conventions, that a developer must follow throughout the project. This will result in much cleaner and efficient code.

Implementing coding standards can help ensure that the source code is secure, dependable, maintainable and even result in better portability. Consistency throughout the code is one of the key reasons as to why following code standards reaps all these benefits. Consistency makes for more readable and reliable code. When the code is uniform and well documented throughout, developers are able to maintain and modify it without a hassle as it is easier to follow and understand. Inconsistent and faulty code makes the whole code base vulnerable and more susceptible to attacks, and this can be avoided by simply incorporating standards.

IEC/ISO 12207

This standard falls under systems and software engineering and represents Software life cycle processes. This is a one in all standard that defines all the processes associated with the development and maintenance of software systems. The current revision has a new process very similar to IEC/ISO 15288, where the total number of processes is down to 30 from 43. There also exists a difference between stages and processes. Where a stage is a "period within the life cycle of an entity that relates to the state of its description or realization", IEC 12207. A process is a "set of interrelated or interacting activities that transforms inputs into outputs", IEC 12207. With these definitions, this standard only defines processes, but does however consider stages such as development, production utilization, sustainment, or retirement. Some examples of processes include agreement, organizational and technical. The agreement is exactly how it sounds, an agreement is established between two parties. The organization allows an organization to maintain and control the life cycle process of the system implemented. Finally, the technical processes allow for an assessment of each project during its life cycle, such as its architecture, implementation, or design. With this, there are thousands of other standards related to many forms of software, but a couple more examples include ISO/IEC 14764, which covers the maintenance of life cycle processes and ISO/IEC 15288 which covers the processes associated with a system during its life cycle.

BARR-C2018

BARR-C2018 is a relatively new standard which was created specifically for applications and designs written in C/C++ that deal with embedded systems. It's main objective is to minimize the chances of writing code with bugs and defects [16]. BARR-C2018 has a wide range of rules and guidelines on various aspects of C/C++ programming. The rules cover things such as basic formatting, (braces, parenthesis, white spaces, alignment, commenting, etc), dealing with different data types (signed & unsigned int, floats, booleans, etc), procedure rules (functions, ISRs, etc), variable names and statement rules (conditional statements, switch statements, loops, etc).

Some of the more important rules that should be followed regarding the general formatting of our source code are:

- The line widths of every line should not exceed 80 characters
- Braces should always surround any blocks of code following, if, else, for, while etc.
- Each left brace and right brace should appear on its own line and should be lined up with each other
- Parenthesis should be used whenever applicable to ensure the execution order for sequences of operations are correct
- Parenthesis should be used whenever a logical operator is used in a statement.
    - e.g. if ((a > b) && (a < c))
- Comments should be clear and concise with correct spelling and grammar
- Comments should come before a block of code that implements some functionality or algorithm that needs explaining and should have the same indentation as that block of code
- All assignment (=, +=, -=, *=, /=, %=, &=, |=, ^=, ~=, !=)and binary (+, -, *, /, %, <, <=, >, >=, ==,!=, <<, >>, &, |, ^, &&, ||) operators should be preceded and followed by a space.
- Every indentation should be a multiple of 4 characters from the start of the line

Some of the more Important rules that should be followed in regards to data types and procedures are:

- The names of all new data types should only contain lowercase character and underscores
    - e.g. int max_count
- If the bits or bytes of an integer are important then fixed width integers should be used
    - e.g. int8_t, int16_t, int32_t, int64_t
- Procedures should not begin with underscores and should not be longer than 31 characters
- Function names should not have any uppercase characters
- Underscores should be used to split up words in procedure names
- Functions should be limited to a maximum of 100 lines of code
- Every function should declare a prototype in the module header file
- Interrupt service routines should be indicated by #pragma or by keywords that are specific to the compiler such as "_interrupt"
- Functions that will act as ISRs should have names that will end in "_isr"
    - e.g. "strum_isr"

Some of the more important rules that should be followed in regards to variables and statements are:

- The naming convention for variables follow the same rules that were covered for functions and procedures
- Global variables should start with the letter 'g' and pointers should start with the letter 'p'.
    - e.g. 'g_max_count' , "p_strum_one"
- Every variable should be initialized before it can be used.
- Local variables should be defined where you need them as opposed to being defined at the top of a function
- Using the comma operator when declaring variables should be avoided
    - (e.g. "int x, y;"
- There should not be nested conditional statements that go deeper than two levels.
- All 'if' statements that are followed by 'else if' should end with an 'else'
- For each switch statement the break for each case should align with the case name.
- Numbers should not be hardcoded as the initial start or endpoint in any loops
- Infinite loops should be implemented with "for (; ;)."

These sets of rules are the most important and relevant that should be followed to ensure our code stays consistent and robust if the BARR-C2018 standard is chosen.

GNU Standard

GNU is a unix-like operating system whose software is free and allows users to have more freedom than most. The GNU standard was primarily created to ensure an easy and clean installation of the GNU system. However, the standard also serves as a guide for writing clean portable and dependable code. This standard was structured towards programs written in C, but can also be useful for other programming languages. Much like BARR-C2018, the GNU standard also has a wide range of rules for programs written in C such as code formatting, commenting, use of constructs, naming variables, functions and files. Although GNU has many rules, it is not as strict and restrictive as BARR-C2018 is.

Some of GNUs standard in regards to formatting are as follows:

- Each line of source code should not exceed 79 characters
- Braces should be on their own individual line
- If all arguments do no not fit cleanly on one line then they should be split up into two. Arguments on the second line should be indented such that the first argument on the first line is aligned with the first argument on the second line.
- Spaces should be added before any open parentheses and after any commas
- Programs should begin with a comment briefly describing what the code does
- Each function should have a comment saying what it is doing

In regards to constructs, variables, functions and files some important rules to follow are:

- Names of variables and functions should be insightful and give some sort of information as to what the meaning of the variable or function is
- Abbreviations should be avoided, and in the case that they are used an explanation of what they mean should be given
- Each variable declaration should start on a new line
- You should not have assignments inside of if-conditions
- In the case of nested if-else statements, braces must be used
- Underscores should be used when naming variables, functions or files with more than one word
- Uppercase characters should be reserved for macros and enum constants and not variable, functions or file names

- Any variable with a constant integer value should be defined by using enum as opposed to '#define'

GNU has many other rules on top of the ones listed, however these are the most relevant rules that can be applied to our use-case [21].

Using Software Standards

For our project, we must use software standards to create a structure for everyone who is working on the software end of the project. From our previous research and sections we have decided to use the Arduino programming language which is a combination of C and C++. Thus, both GNU and BARR-C2018 standards could be applied and be beneficial. However, we have chosen to implement the BARR-C2018.

While GNUs objective is to write clean, portable and dependable code we believe that we will benefit more from implementing BARR-C2018. This standard was created specifically for programming embedded systems and ensuring that all bugs and errors are minimized when possible. Given that our project is structured around our microcontroller this was a big deciding factor. In addition BARR-C2018 also aims to improve the readability and portability of the code much like GNU, however it is more strict and restrictive as we can see from the set of rules listed for each. This will ultimately lead to more consistent and robust code.

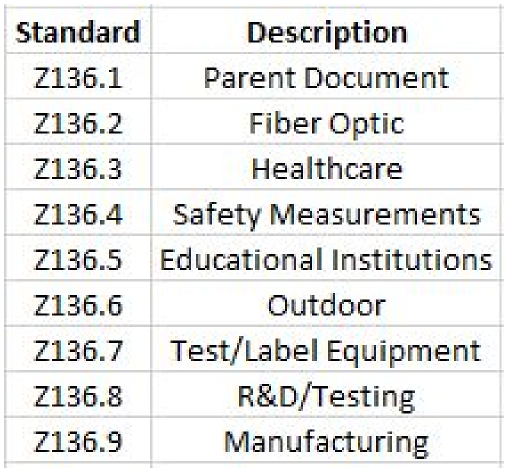
If a standard is not used within our software it will hinder our progression of implementing a workable code to our microcontroller. Thus, everyone contributing to the code base of this project will become familiar with BARR-C2018s rules and implement them when applicable. Following this standard will allow us to minimize any defects while also writing clean and consistent code where everyone can easily follow and make contributions.

4.2. Hardware Standards

By selecting the hardware components there are also standards for certain types of hardware. This section focuses on the standards of these hardware components. Specifically, we will cover what is expected for safety while using lasers and how batteries standards can affect our overall design.

ANSI Z136.1

With the American National Standards Institute, this standard represents the guidelines for the safe use of lasers that operate between 180nm and 1000μm wavelengths. The outline here includes laser light shows, lasers used outdoors for research, and military lasers. There are also acceptable levels of irradiation for airspace and to minimize laser interference with air crews. Within these standards are different classes of lasers, from 1 to 4. For class 1, these are lasers of 0.39 milliwatts, which means they are safe for long term viewing. Class 2 consists of lasers less than 1 milliwatt, which is safe for unintentional exposure. For class 3, these consist of 3R and 3B, with beams of up to 4.99 milliwatts and 499.9 milliwatts, respectively. Avoid intentional exposure or exposure all together. The last class is 4, which consists of 0.5-watt lasers. These can cause severe eye damage and also to the skin. Concerning eye damage, typically anything above 500 milliwatts is on the high scale and should be taken with caution. Some other standards from ANSI exist under Z136 where you have 0.4, 0.6, and 0.7. Z136.4 is a practice for safety measurements during hazardous situations. Z136.6 demonstrates how to use lasers outdoors safely and Z136.7 provides information on how to effectively use safety equipment such as glasses. [D16]



Standard	Description
Z136.1	Parent Document
Z136.2	Fiber Optic
Z136.3	Healthcare
Z136.4	Safety Measurements
Z136.5	Educational Institutions
Z136.6	Outdoor
Z136.7	Test/Label Equipment
Z136.8	R&D/Testing
Z136.9	Manufacturing

Table 16: Laser Safety Standards

<u>Using Hardware Standards</u>

From the hardware standards above, there is a variety, ranging across the proper use and safety of lasers. With ANSI Z136.1, the implementation and understanding of the use of lasers both indoors and outdoors. With nine different segments focusing on separate fields, understanding the parent document that outlines sections will give us a guideline to follow when dealing with lasers and the proper handling. Importantly, understanding situations and surroundings will help develop our housing design.

The next stop is to understand the implementation of ANSI Z87.1, which outlines protective eyewear across many fields. Specifically, eye safety while using lasers, the user and any bystanders will follow a protocol before any devices are powered on. Whether the laser is not necessarily powerful, damage to the retina is a major concern across the board. To also help, there are signs developed to warn everyone when they approach an eye safety zone as can be seen in Figure 31.

Lastly, the use of battery is crucial to our design, and that involves maintaining reliability over the lifetime. Table 17 lists out common standards for lithium ion battery cells and packs for a range of commercial or household uses. For each standard, implementing the proper use of our lithium ion cells in our design to maintain proper power to our electronics. When not in use, keeping the cells in a household climate away from extreme heat and the possibility of being punctured will maintain the life of the cells to continually use for our project design.

ANSI Z87.1

The industry standard ANSI Z87.1 introduces eye safety standards for many different fields of work including optical. The markings listed in this category are separated by impact/non impact, splash/dust, and optical radiation. The focus for our project design involves lasers, which are very harmful to the eyes but still can deal damage if direct eye contact is made, whether by accident or on purpose. For optical radiation protection, there are several rating numbers for different protection abilities. A 'W' is used for welding, 'UV' is ultra-violet, 'Heat' is Infra-red, 'Glare' is Visible light, 'V' is variable tint, and 'S' is special purpose. It is also important to note that labels and signs can alert people of the appropriate personal protective equipment (PPE) for the different scenarios.

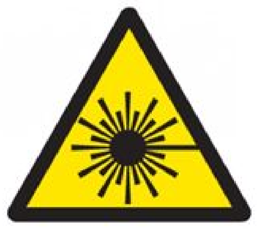


Figure 31 : Laser Safety Sign

The figure above is actually a perfect example of a sign that is common around work areas that involve lasers capable of dealing damage to the eyes or skin. For ANSI Z535, this system presents safety and accident prevention information that can be conveyed through signs like the one above. There are categories from one through six that follow safety colors, facility safety signs, safety symbols, product signs, safety tags and information. Some notable words involve "WARNING" and "CAUTION".

Battery Standards

Understanding these standards allows for the proper use and disposal of batteries that may or may not contain toxic materials. Also, general principles and specifications of charging and maintenance of a wide range of batteries to allow for longevity. With IEC 60086-2, this is a standard strictly for batteries in general, involving proper care and safety guidelines. ANSI C18.2M contain specifications on portable rechargeable cells, similar to the previous standard but targeted specifically to rechargeable batteries. What's important now is understanding the standards for Lithium Ion batteries and Nickel Metal Hydride. Importantly, BS EN 60086-4, lists the safety standard for lithium batteries, providing lists on the dos and don'ts when charging and storage. Nickel Metal Hydride, BS EN 61436, covers all the topics for secondary cells from safety as above and also use of/makeup of the technology.

Three of the most popular standards for lithium ion cells include UN/DOT 38.3, IEC 62133-2, and UL 2054. Each one covers different aspects of safety and standard when dealing with batteries/cells. UN/DOT 38.3 is set to allow lithium ion batteries to travel the world by air, rail, or truck. There are eight tiers of tests that cover altitude simulation, thermal tests, and vibration. IEC 62133 covers international compliance for the safety requirements of portable sealed lithium cells. There are four tests that cover molded stress tests, external short circuit, free fall, and overcharging. Finally, UL2054 is the most challenging to pass with a vast amount of tests. These include seven electrical, four mechanical, four battery enclosure, one fire, and two environmental. The standards listed below offer more of a general understanding of specific lithium ion batteries during testing and common use.

Standard	Info
IEC 60086-1:2000	Primary Batteries - General
UL 2054	Safety of Commercial and Household Battery Packs - Testing
IEC 600086-4:2000	Primary batteries. Safety standard for lithium batteries
UL1642	Safety of Lithium-Ion Batteries - Testing
IEC 61960-1:2000	Secondary lithium cells and batteries for portable applications. Secondary lithium cells

Table 17: Battery Standards

4.3. Design Constraints

When it comes to creating a product and brainstorming many ideas, there are always constraints in place either set by the company/individuals or society. From time to health and safety constraints, we will walk through what each one entails and how it impacts our design decisions. A quick brief consists of the cost of our design or the impacts of lasers on humans dealing with blindness or the toxic materials used inside some. As well with how our component selection can impact the environment through the manufacturing process.

Economic and Time Constraints

Usually within a company, an economic constraint consists of some type of cost that goes along with the research/manufacturing of a new product or existing ones. As a student, costs could include things like textbooks and transportation fees. During our time, the virus COVID-19 has affected our economy drastically, so when it comes time to know what to purchase and get all parts together for final build, we should order everything at least two weeks prior just in case of restrictions and extra precautions during shipping. With the production of our design, the cost could end up being substantially more than a practice guitar you can get from a music shop. In the end, the training options and intrigue from people that were not interested in musical instruments before but now are will be an end goal.

As with all time constraints, there is a set amount of time to achieve our goal. For senior design, there are two separate semesters where we do all the research in the first semester and build our project in the second semester. Since we are taking senior design 1 in the Summer as opposed to the Fall or Spring, we have 12 weeks compared to 15 weeks. Our time schedule is condensed which means we have less time to research and build a prototype going into Senior Design 2.

Environmental, Social, and Political Constraints

With our environmental constraints, since we are dealing with lasers, there is not much affecting the environment except for the fact that lasers can be blinding or the process of manufacturing and assembling of the lasers can be harmful without precautions. It’s also important to mention that with batteries, any toxic materials must be disposed of properly at a landfill. All manufacturing processes have some type of environmental impact, which may deal with radiation or a large amount of energy. All precautions are necessary, and follow any compliances set forth by an organization.

There is a limited list of social constraints, such as our product is used during a performance and an audience member has issues with looking at lasers which can cause seizures in epileptic patients. Specifically, with the manufacturing of our device, it could be held in place on parts we need, or they need to go through a rigorous test set by a social standard in another country. Importantly, it is the music being played and how it can be perceived based on the person or culture in another country.

Our political constraints are very existent, in the sense that the US has restrictions on the lasers that can be produced and sold on American land. Restrictions in place are with the wattage preventing illegal lasers in the US. This is the case since cases exist where people would shine lasers at helicopters and planes in hopes to blind the pilots. Another exhibit arises from safety on a consumer level. If our design ever gets mass produced for the public, even more rigorous testing must take place to ensure our supplier is reliable and manufacturing is consistent.

Ethical, Health, and Safety Constraints

These constraints are in place to hopefully protect the musician from potentially being blinded by the lasers or the audience. Safety is especially important and to maintain to the health of everyone involved in the project design. Ensuring everything is in place and all the electrical components are soldered correctly and grounded when needed. There are also industry ethics standards to follow such as being truthful and avoid deceptive acts. Proper handling of lasers and electronics/power to avoid any harm to another person. While working with printed circuit boards, any soldering taking place must have good ventilation and dexterity in fingers to place components on the board. Lead free solder is the best option and rosin flux must be taken seriously since breathing problems can arise. The ethical constraints are important to consider since there are standards set on the health and safety of everyone involved or potentially involved in our project. It is our ethical duty to ensure all the constraints listed here and other sections are taken seriously for the safety of others and the University.

Manufacturability and Sustainability Constraints

During this section, these constraints represent the limited amount of manufacturing services to us within our price range and being able to produce a quality product. With our design, specifically the printed circuit board and its components such as the microcontroller, voltage regulator and connections to the laser diodes/photodiodes. The connections from the printed circuit board to the diodes will be a considerable distance but should produce a significant delay from the plucking to the amplifier. To manufacture our design, the shape and dimensions of the guitar must be finalized and the number of strings we will incorporate. The purpose here is to insure that all the electronics will fit nicely and neatly within the guitar for easy use when needed. For all the electronics, the testing will be done with a breadboard and all the electronics chosen so far. For integration, the components will be soldered onto the printed circuit board, but with extra precaution since damage can occur and to insure that no shorts or any faults preexist on the printed circuit board.

For a long lasting, sustainable design, our focus will first be on the actual guitar design to ensure there is proper airflow and no overheating to keep the longevity of the electronics. We also want to keep the cost low, but with a sturdy frame. The next important factor is the play time of the instrument. With this, we want the musician or pupil to play for at least two hours, but hopefully have the overall design last off and on throughout the day.. To achieve this, the battery technology narrowed down to NiMH or Lithium Ion. Lithium Ion is the better choice for a single charge since it has a greater density, but the downside is the cost. Another great option would be to add fans to the body of the guitar if overheating and circulation becomes an issue. Overall, we need our design to be within budget and without setbacks, either from parts or faults, to ensure a smooth transition from our prototype to Senior Design 2.

5. Project Hardware and Software Design Details

The project hardware and software design details focus on the design details from both a hardware and software perspective. These perspectives are broken down into smaller topics to help build a bigger picture of large components.

5.1. Initial Design Architectures and Related Diagrams

Figure 32, shows a basic overview of how our hardware will be set up in its final stage of design.

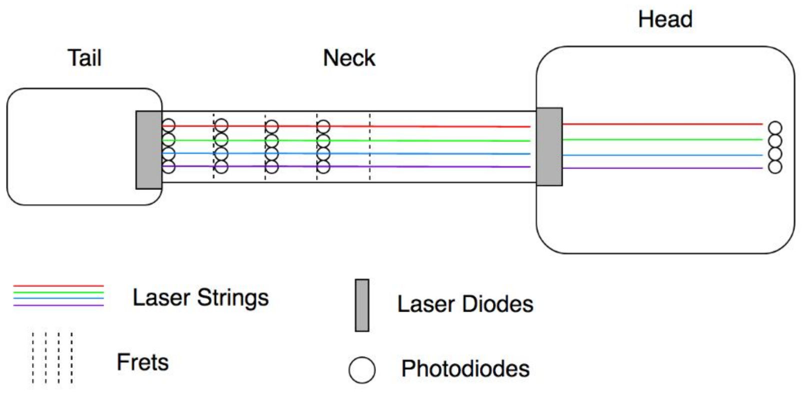


Figure 32: Project Hardware Overview

The figure above outlines the hardware of our project. The guitar can be separated into three main sections, the “Tail”, the “Neck”, and the “Head”. Each section provides a different functionality that is vital to the instrument's performance. The three sections will work together in unison to provide the user with an instrumental musical experience. The overall design and goal of our project was to design an instrument similar to the most popular string instrument, the guitar. There are already various instruments that use lasers as the strings, however these technologies are mostly attempting to mimic the harp or Theremin, which are far less popular than the guitar, bass, or ukulele. For our design, we chose to go with a four stringed guitar model, which would be most similar to a bass guitar or ukulele. The choice of using four strings was made because of the popularity of these instruments, as well as the fact that they are on opposite spectrums of size in terms of guitar style instruments, with the ukulele being one of the smallest guitar style instruments, and the bass guitar being one of the largest. From an engineering perspective, this gives us a lot of room to work with when testing our project, and increases our options in terms of portability and functionality. As can be seen in the diagram, the device currently has four frets, which will operate by returning a voltage from the reflected light gathered from the user's fingers playing the frets. We chose to have four frets in our model, because we want to be able to use this instrument to play chords like a real instrument. Having four frets allows the user to play any major or minor chord, as can be seen in the chord chart in the music theory section of this document, effectively allowing the user to be able to play any song. However, to provide an experience that is more true to the instrument, a full fretboard would be desired, so we have made additional frets past the first four a stretch goal of ours. Now, we will break the design overview into its three main sections: The Tail, Neck, and Head.

Starting with the Tail of the guitar, which will be the simplest of the three sections. The tail will contain only the laser diodes which will act as the fretting strings of the guitar, as well as potentially containing a power source for this set of diodes, if necessary. This section can also be used to house a small system for modulating the laser diodes, if we choose to go that route in the future. Fitting the aesthetic of a guitar is an important part of the appeal to our project, so having a tail that is small relative to the head will be important.

The neck of the guitar is likely going to be the most complex part from an engineering standpoint, as this is where the user will select their notes by blocking the light from the strings, as demonstrated in the diagram below.

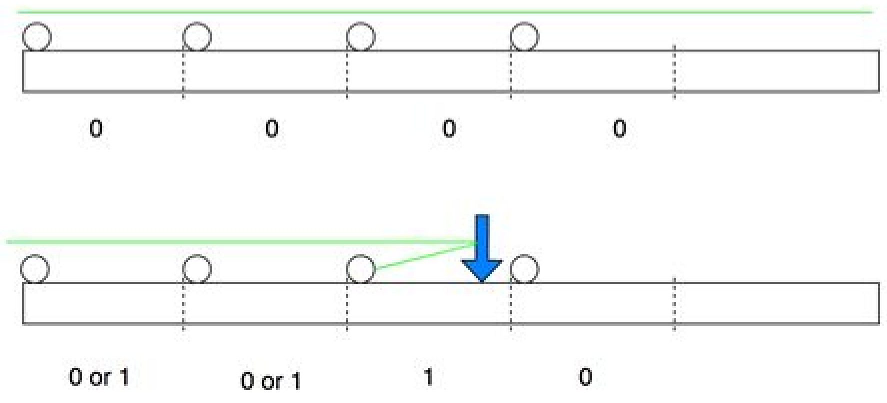


Figure 33: Guitar Fretboard Diagram

The diagram above shows a side view of how one of the fretting strings on the neck of the guitar will operate. The circles are photodiodes, the green line is the laser string, the blue arrow is an object, and the rectangle is the fretboard, with the dotted lines representing the frets of the guitar. A single photodiode will be placed at the beginning of each fret, where the user likely will not interfere with it since guitars are traditionally played with your fingers at the end of the frets. The photodiodes will either return a 0 for low voltages, or a 1 for high voltages. When no object is present to reflect the spatially coherent laser light into the photodiodes, the diodes will have a low voltage, and return a 0 to the microcontroller. When an object, such as the user's finger, is present on the fretboard, the diodes will collect the reflected laser light and return a high voltage, or value 1 to the microcontroller. These 1s and 0s will be used to determine which fret is being played by the user, and this information will be returned to the microcontroller which will store the note that is selected. Here, it will be important to consider the effects of noise from adjacent strings and ambient light. Our first design test will attempt to use the housing to prevent as much ambient light as possible from entering the photodiodes. With more testing, we may also consider narrow band filtering and laser modulation as alternative methods of dealing with noise.

This brings us to the head of the guitar, which will contain a set of strings which will play the notes selected in the fretting section. This design will be a bit simpler, only involving one photodiode per string. This photodiode will be on by default, activated by a second set of lasers that are on the head of the guitar. When the beam is interrupted by the user, the microcontroller will know to play a note. The head of the guitar contains the most space to put objects such as a power supply and the microcontroller, so these parts will go here. Depending on the speed of the photodiodes, we can potentially measure how long the photodiode was off to determine the velocity of the strums. This value can be used to determine the volume which each note is played at, and would be a nice feature that would add to the instrumental experience. For now, however, our goal is to be able to get the guitar to play the selected notes consistently. Measuring strum velocity is a stretch goal of ours.

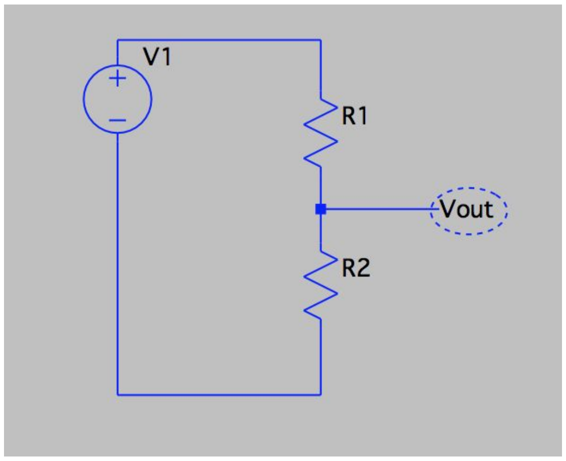


Figure 34: Voltage Divider

Equation 1: $V_{out}=V_1*(R_2/(R_2+R_1))$

In order to draw a readable voltage from our fretboard to the microcontroller we will use the concept of voltage divider. By arranging our photodetectors in series with a resistor and our voltage input, we will be able to measure the voltage at the node between the photodetector and the resistor using Equation 1 above in order to return an input to our microcontroller. This voltage will correspond directly to the amount of light incident on the photodetector, increasing when there is a lot of incident light, and decreasing when there is not a lot of incident light. Our system will determine what values are high and low when we begin our testing, and the microcontroller will read high voltages as a 1 and low voltages as a 0 to determine which note to store when the strumming strings are played by the user. The figure below shows a visual representation of the concept of voltage divider. Because we will be using a set of four photodiodes in series with one another, we will be able to apply this method to each of the photodiodes to find out whether the voltage is high or low. Having the photodiodes in series means they will share the same current, but have different voltages, so this voltage is what we will use to return information to the microcontroller, as demonstrated in the figure above.

<u>Hardware Schematics</u>

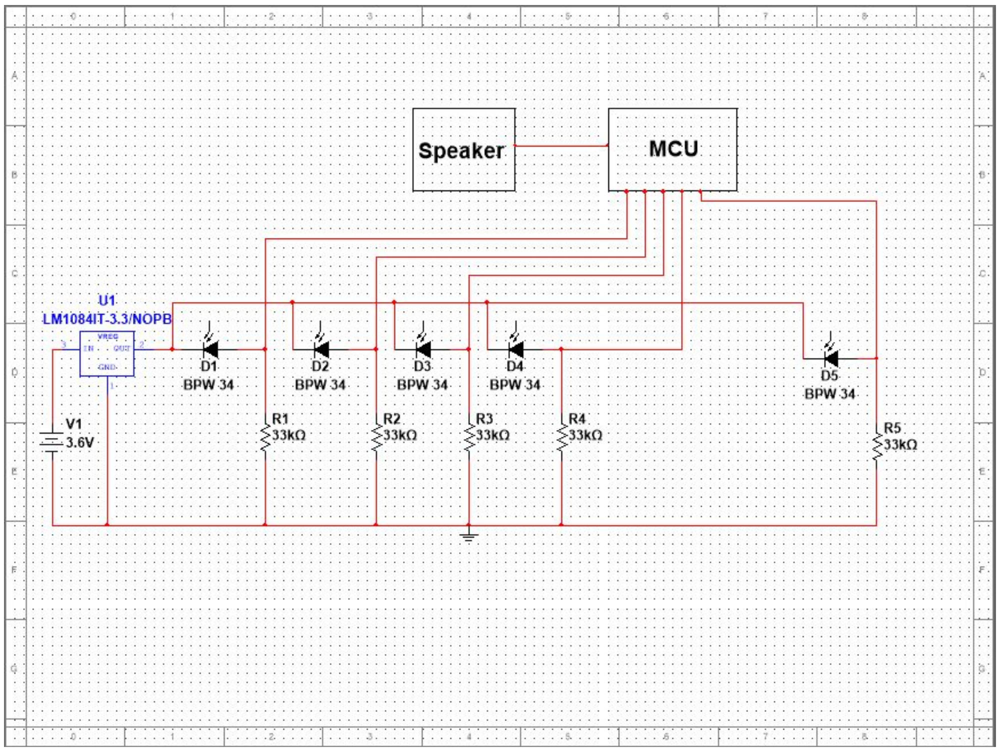


Figure 35: Single String Hardware Schematic

The figure above shows the electronic schematic for a single string of the laser instrument. The design is going to use five photodiodes which are reverse biased by 2V at a single node. The voltage source will be a single cell battery pack which provides 3.6V that is run through a voltage regulator to reverse bias the photodiodes with the proper amount of voltage. As mentioned previously, by reverse biasing our photodiodes they will be operating in photoconductive mode, essentially acting as current sources which will respond to the amount of light incident on the photodiode. The two sections of the single string design, that is the strumming section and the fretting section, which are seen in the diagram can be seen in the schematic as well. By breaking down the two main sections of the single string's electronics we can apply this to all four strings in the laser guitar to achieve the full design of our project.

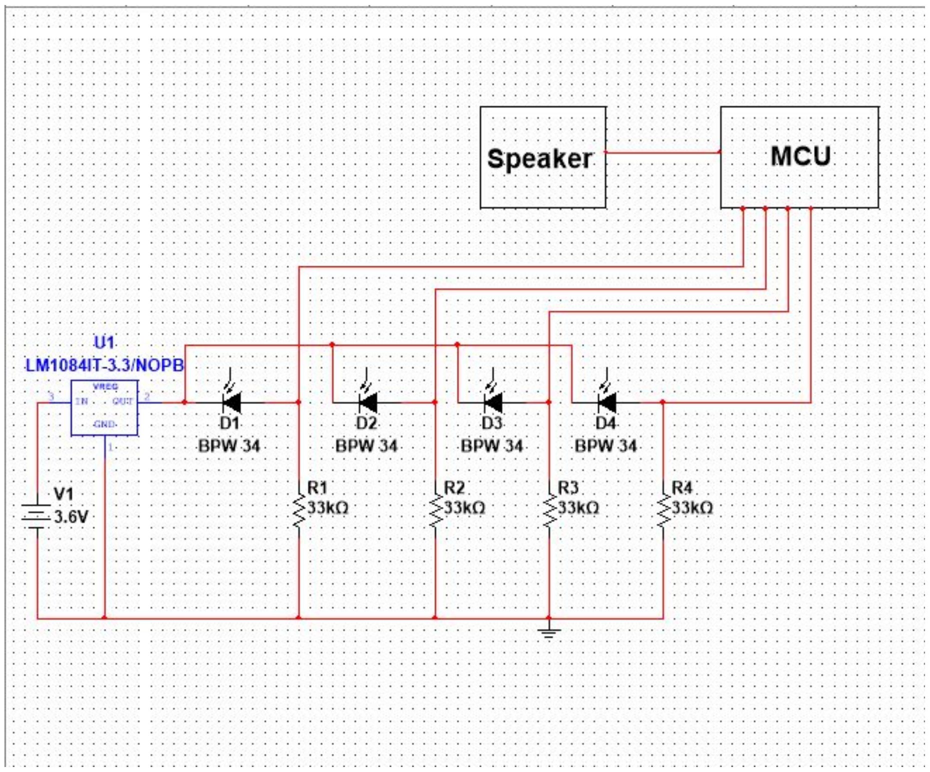


Figure 36: Single String Fretboard Schematic

The figure above shows the schematic for the fretboard of each individual string. The fretboard will operate by using our photodiodes to act as a current source which sends high current to our MCU when light is being reflected from the user's finger directly on to the photodiode. The current is then run through a 33kΩ resistor which will provide a voltage at the node between the photodiode and the resistor that our MCU will be programmed to read. In our initial testing, we were able to draw about 0.807V from our photodiodes when reflecting light off of the user's finger. Since our initial testing was done with 30kΩ of resistance, we decided to increase the resistance to slightly improve the amount of voltage being drawn from the system. In contrast, when no light was being reflected directly on to the photodiode, only an average of 0.0505V was measured. These measurements were taken in less-than-ideal conditions, as the room we were testing in contained a lot of ambient light, and our photodiodes were placed orthogonally to the light being reflected off the user's finger, so that they were not picking up as much light as they will be in our final design, where they will face the light source directly to achieve maximum lighting incident on the diodes. In order to keep our design as simple as possible, while still providing enough information to the MCU in order to determine what note to play, the MCU will be programmed to read the voltage as either "high" or "low". A high voltage reading lets the MCU know which fret is being held down by the user. When the fret is known, a note is stored in the

MCU, and that note is played when the MCU reads an interrupt from the strumming section of our design, as explained in the section below.

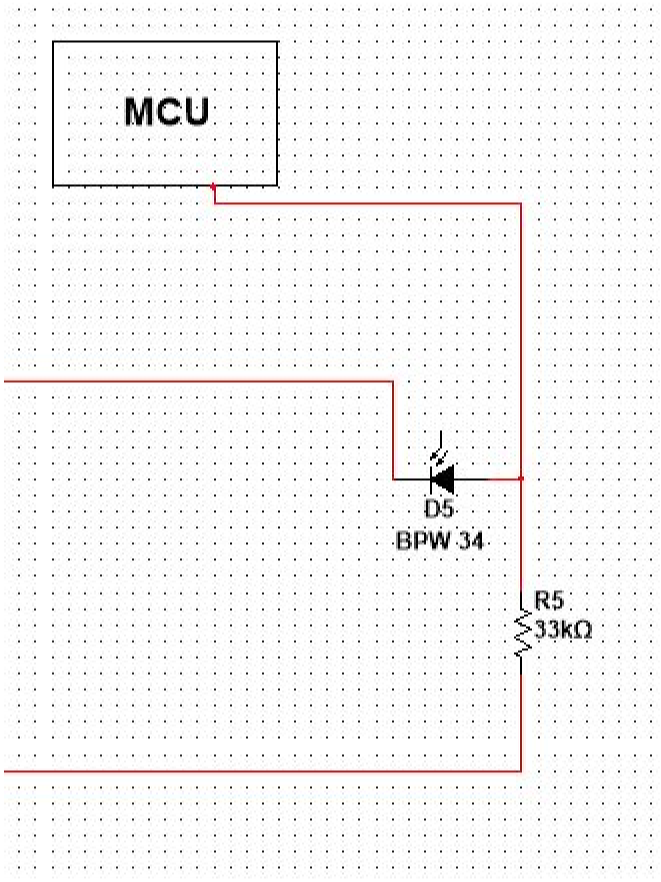


Figure 37: Single String Strumming Schematic

The strumming section of the guitar uses a single photodiode in series with a resistor to detect the light coming from the second set of laser diodes attached to the head of the guitar. Because we are using two sets of laser diodes, the fretting strings will not interfere with the strumming strings. The second set of laser diodes will be shined directly on the strumming photodiodes. Because of this, the strumming photodiodes will constantly have a high voltage value returning to the MCU. The MCU will be programmed to read when this high voltage is interrupted, and will tell the speaker to play a sound when an interrupt occurs. The two sets of photodiodes will both be reverse biased at the same node in order to save space and reduce the amount of battery packs required for each string thus minimizing the overall cost of the device. Once again we have opted to use a 33kΩ resistor. This decision was made in order to keep a relatively consistent voltage across each pin that is used in our design, across both the fretboard system and the strumming system. The interrupt and the fretboard will work in conjunction to provide a friendly user experience that feels like playing an acoustic instrument.

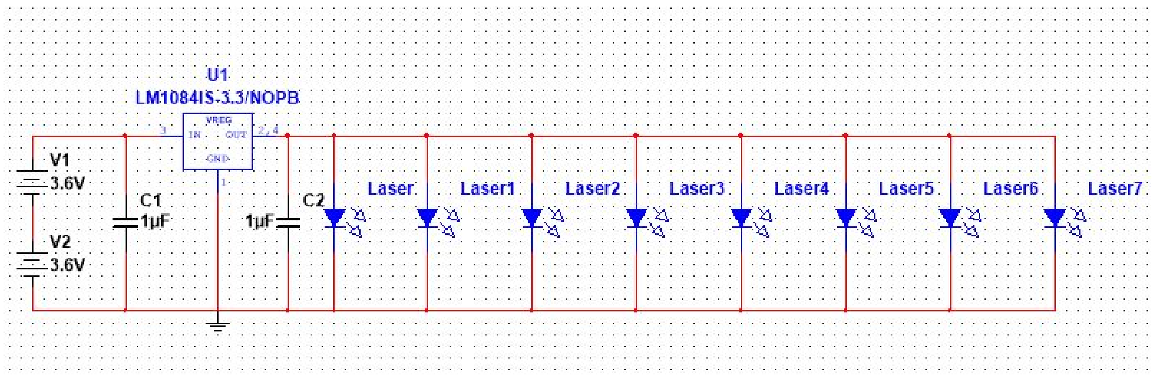


Figure 38: Laser Diode Schematic

In the above figure, this schematic represents the power supply to the eight laser diodes. The idea is to have two lithium ion cells feed a voltage regulator to have an output of five volts to each laser diode. Four of these will be placed on the fretboard and the other four on the head of the guitar. As stated above, the amount of light feeding into the photodiodes will determine exactly what note will be played and the string for the strumming section.

Another approach to this power supply can use a single cell to power each set of four laser diodes. With this, there will be less current drawn per cell which will increase the longevity of the power supplies. Now, instead of using a buck converter, we would have to use a boost converter. There are actually regulators that act as both a boost and buck converter, so this would not be too difficult to implement. Testing wise, we were able to provide five volts to a laser diode, which was then used to test the photodiode. From one to five volts, the intensity of the beam would increase, so with more testing with this range and the measurements taken from the photodiodes, we will determine the best scenario based on the environment to appropriately supply each laser diode. Keep in mind that each laser represents a single string, which is where all the testing begins to give us a better understanding of the overall four string/four note design.

The voltage regulator shown represents a buck converter where the 7.2V input voltage will be reduced down to 5 volts to power each diode. With this switching method, we will have an excellent efficiency rating where the heat dissipation is kept low. The power supply can also be fed off from the power to the MCU since both will need a 5 volt operating voltage. Overall, this is our initial schematic that will see improvements as we progress through the testing phase.

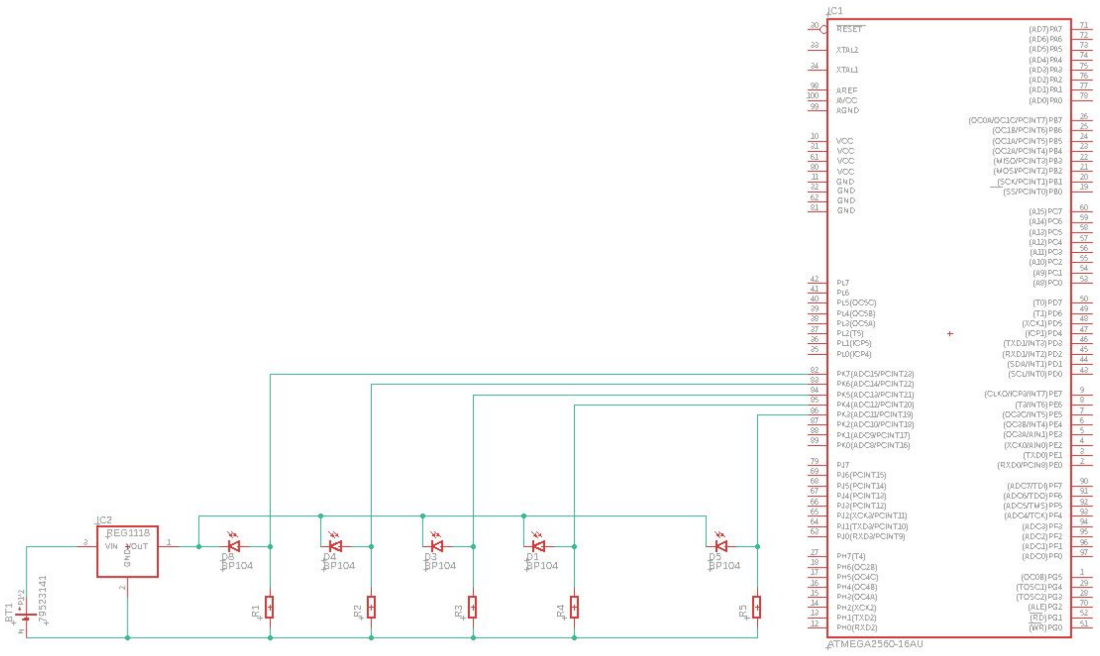


Figure 39: Single String MCU Pin Schematic

In the figure above, we have the ATMEGA2560 shown to the right and the schematic for a single string to the left. Pins 82 through 86 are used on the MCU to connect the photodiodes on the positive terminal. Once the diodes are reversed biased, the change in voltage across each resistor is measured as photons are introduced to the diode. From which a threshold is met to play a certain note from the MCU. The MCU itself will have a seperate two cell 7.2 volt power supply with a regulator supplying 5 volts. The purpose here is to give a better understanding of how the positive terminals of the photodiodes will connect to the MCU and utilize the internal ADCs.

This design encompasses a single lithium ion cell of 3.6 volt, which is converted to 2 volts through a buck converter. The representation of the buck converter is kept simple as the final representation will have eight pins instead of three. With the increased voltage drop across the resistors, that voltage is measured and compared to the programmed value that represents the specific notes.

Overall, this is a very good representation of our expectations of our final design with how we will measure and supply voltage to each photodiode. It is also important to mention that there are a total of 54 pins that can be used to measure and compare our voltage values. Changes can still apply as we progress along through our testing phase.

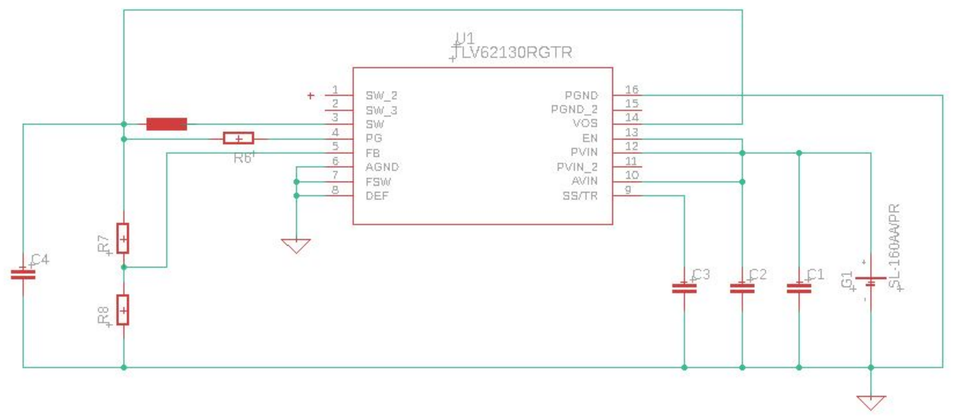


Figure 40: Switching Voltage Regulator Schematic

With the above figure being our representation of the switching voltage regulator we will be using. For our one and two cell power supplies, we will be able to step down 7.6 volts to 5 volts and 3.6 volts to 2 volts, to supply both the microcontroller and the photodiodes/laser diodes. This means that each string will use one cell lithium ion batteries with their own voltage regulator, while the MCU and laser diodes will utilize a two cell design since their operating voltage is higher. Our first thought was to have a four cell battery pack and supply 14.4 volts directly through to the voltage regulators feeding the electronics. Long term though, this can cause too much strain and reliability issues with the batteries. So the former creates a better vision of how everything will be encompassed. With its current package, not all the pins will be used, but this IC offers many advantages in switching frequency, heat dissipation, and protection protocols.

Housing Design

An important factor to consider is how we can optimize the housing of our laser instrument in terms of functionality, portability, and aesthetics. Because we are creating an instrument that plays like a guitar, the exterior housing should resemble a guitar. For the purposes of our project, we have decided to use a real ukulele to house the electrical components of our design. This is convenient because of the cheap price that ukuleles can be found for, and the fact that they are small and portable, but have a large enough body to fit all of our electrical components. Most of our electronics will lie in the head of the ukulele, however due to the nature of our design some of the components will need to lie in the fretboard. In order to account for this, it is likely that we will saw off the ukulele's original fretboard and replace it with our own, with all of the necessary electrical components already in place. The figure below shows the ukulele we have ordered to house our electronics.

Figure 41: The Martin Smith UK-222-A Ukulele

This ukulele will be used as the body of our design, with most of the electrical components lying in the head of the ukulele, and some of the components lying in the fretboard, which is likely to be sawed off and remade to suit our needs more precisely. This Ukulele was simply the cheapest available on Amazon at $22, since we only needed the body for our housing and the quality of the instrument is not crucial to our design.

In order to optimize the housing from an engineering standpoint we had to consider the optical system that our project is. The main issues to consider in our design are size of the device, and the noise that each string will have to avoid from ambient light and adjacent strings. In order to account for size, we will simply design our own fretboard that is large enough to provide for the needs of our project, since the fretboard on the UK-222 Ukulele is likely going to be too small to fit all four of our laser strings at a comfortable distance from each other. The specific size of our fretboard will be roughly 2.5 inches in width and In order to account for noise from ambient light and adjacent strings, we will surround our photodiodes with a simple plastic ring which will help prevent light from reaching the photodiode unless it is coming from directly in front of the photodiode. Other housing design considerations include using a polarizer at the end of our optical system to prevent noise from light hitting the end of the guitar fretboard, and adding electrical components to the tail of the ukulele such as a power supply for the fretboards laser diodes, if the extra space is necessary. These housing design considerations will be made final depending on the results of our multi-string testing in Senior Design 2.

5.2. Software Designs

The process of designing our software for our microcontroller to detect the laser input should be as fluid as possible. The fluidity of our process should not hinder the performance of the device and should meet our requirement specification. In order to make the design process easy without the trade off of device performance, we need to consider how the code should be used and how it should be designed. The requirement specification that the software design must consider is producing the sound under 0.50 milliseconds and reproducing the sound as accurately as possible. Although these two requirement specifications are independent from one another, the software that will be designed to achieve these requirements can create both design and programming obstacles.

The hardware implementation can be simplified down as each laser will be assigned a frequency/pitch. The lasers that represent each frequency/pitch will be positioned on the neck of the guitar. Similarly, the lasers on the head of the guitar will be used to program the microcontroller in such a way that when a laser is broken it will act as a trigger to output the respective frequency/pitch based on the corresponding laser on the neck of the guitar. In response to these actions occurring simultaneously the correct frequency/pitch will be played back by the speaker. By understanding the hardware components needed for our project, it allowed us to consider important design factors to advance the software design process. The factors to consider to ameliorate the software design process are programming paradigms, and design patterns. Both the programming paradigms and the design patterns should be considered because this will help keep the design within the requirement specification while minimizing impediments.

Although using programming paradigms and design patterns can ameliorate the software design process, we need to consider how the software will affect the performance of a device. Two factors that can affect the performance of a device when designing the software for it is space and time complexity. Without taking these two factors into consideration, we could be writing modular and efficient code, but it would not perform to our requirement specification. Space and time complexity work in conjunction with one another. Space complexity is defined as the amount of memory used when an algorithm is executed. And time complexity is defined as the amount of time it takes for an algorithm to complete its task. Space complexity and time complexity works in conjunction with one another because if an algorithm can execute quickly, then the space needed for the algorithm to achieve it's task is minimal. The algorithm does not need to consume more space of a given device to execute and vice versa.

In addition to programming paradigms and design patterns, Interrupt Service Routines (ISR) and low power modes will play key roles in our software design.

There are two methods in which microcontrollers can handle interrupts: using ISRS or by polling. The ISR or interrupt handler is what tells the processor what to do when an interrupt occurs. In our case the interrupt will come from the hardware in the event that a laser is broken. The I/O pin that is programmed to sense whether the laser has been broken or not, will raise a flag that triggers the ISR and tells the processor what should be done at that instant. Polling on the other hand is continuously checking the status of other devices and seeing if a flag has been raised to then do whatever action it needs. This can be very inefficient as it is constantly wasting execution time checking for an event to occur, which sometimes may not even occur and the microcontroller cannot do anything in the meantime. This is the benefit of using the ISRs as it wastes no time, and the microcontroller can be put into low-power modes until an event occurs.

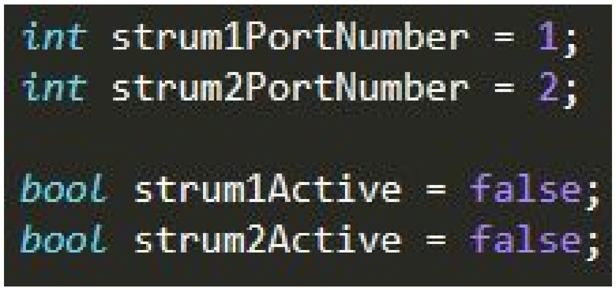
Low power-modes can vary depending on the device or microcontroller, however, all variations have the same goal at hand. They aim to conserve as much power and energy whenever possible. In order to accomplish this goal most microcontrollers have different levels of low-power modes which vary from least to most energy consumption. The low power mode in which the most energy is conserved generally disables certain features and functionalities as well as responds slower to certain events. On the contrary, the level in which the least energy is conserved, allows for most of the microcontrollers features and functionalities to remain active.

5.2.1. Programming Paradigms

The first design factor is to understand what programming paradigms are. A programming paradigm is a way or approach used to solve a problem in programming. We can use programming paradigms to solve problems using the tools and techniques allowed in a programming language. One of the reasons why we chose to program in the Arduino language is because it gives us the ability to use Object Oriented Programming.

Object Oriented Programming is a programming paradigm that models the organization of software design around objects[11]. Objects are instances of a class which contain various attributes and methods (functions). It can be considered a blueprint or prototype from which objects are created. From a class, we can create multiple objects from it and work on them individually without affecting other data or objects within the code. Using Object Oriented Programming provides the ability to reuse code and improve software maintainability

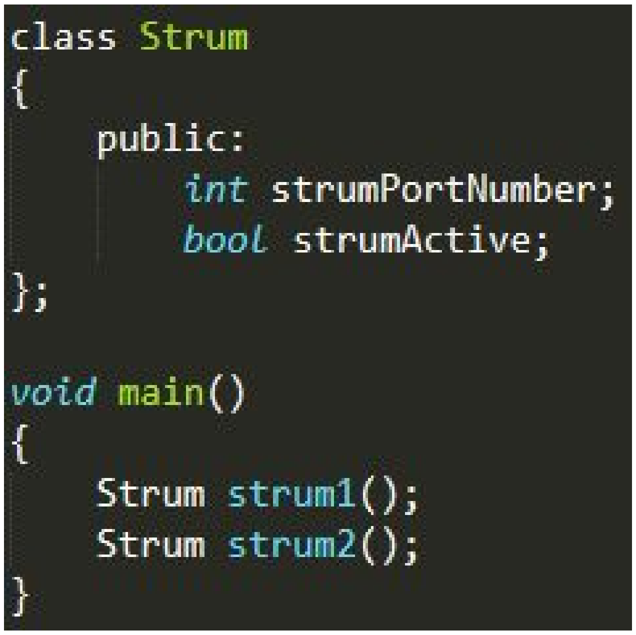
Given the capabilities of Arduino we can implement the programming paradigm of Object Oriented Programming. Objects are instances of class which we can create freely and manipulate as needed. For our project, objects are broken up into two sections. The first section are the lasers that will be used to strum, and the second section are the lasers that will emulate the frets on a guitar board. We can create a strum class and create multiple instances of it. By creating a strum class we can organize and create objects for each laser that are physically housed respectively. Similarly, we can also create a class for the second section of laster. Creating two different classes gives us the ability to reuse the same code multiple times without affecting the other sections. For example;



```
int strum1PortNumber = 1;
int strum2PortNumber = 2;

bool strum1Active = false;
bool strum2Active = false;
```

Figure 42 - Without using Programing Paradigms

Figure 42 shows an example of simple variable declaration. Over the course of our software design there will be multiple variable declarations for the strum and fret section of lasers. Multiple variables can be focused and implied to one section. The problem lies when there are multiple lasers within a section. We would need to keep track of all the variables and manipulate the variables accordingly. The code size of the software can be potentially large enough to exhaust our microcontroller resources to achieve a desired task.



```
class Strum
{
    public:
        int strumPortNumber;
        bool strumActive;
};

void main()
{
    Strum strum1();
    Strum strum2();
}
```

Figure 43 - Using Programing Paradigms

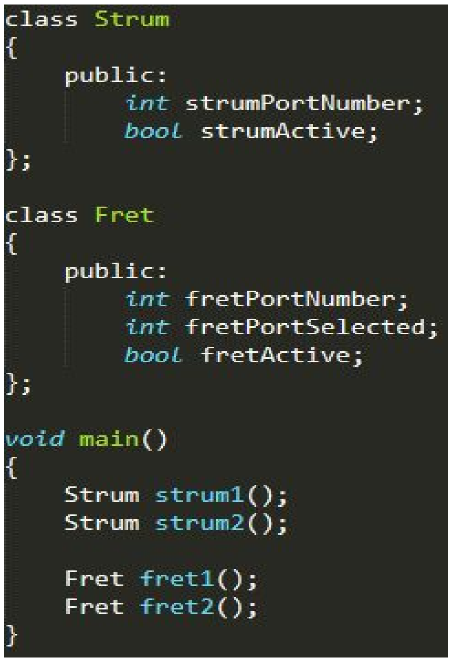
Figure 43 shows an example of Object Oriented Programming. Over the course of our software design, there will be multiple variable declarations for the strum and fret section of lasers. We can take multiple variable declarations and methods (functions) specific to a certain laser and make an object from it. By creating an object, we can encapsulate everything related to a laser and manipulate the data of a specific laser accordingly. To change a Strum's active status, all we need to is access the object's member variable, *strum1.strumActive = true*. We can repeat this process for any object we create. This helps us reuse code without the need to declare multiple variables for every laser.

5.2.2. Design Patterns

The second design factor is to use specific design patterns within our software. A design pattern is a repeatable solution to solve a problem in a software design or implementation. Design patterns can be viewed as a template of an idea that a user can apply to code. A design pattern isn't a complete design that can be converted directly to code.

Design patterns can be altered and implemented differently based on design needs and constraints. There are many types of design patterns that a developer can use such as Bridge, Decorator, Facade, Factory Method, Singleton, and much more. Selecting a programming language that can implement design patterns will be crucial to our project. Choosing the Arduino language, gives us the ability to use Object Oriented Programming capabilities which complements the practice of using design patterns. Object Oriented Programming compliments the practice of using design patterns because we can construct objects and create a pattern of how objects can be used. By using design patterns with objects, it can prevent subtle issues that can cause major problems and improve code readability for coders and architects familiar with the patterns [12].

Design patterns are not specific to a particular programming language. We can use design patterns with object oriented programming in Arduino. A design pattern that we'll be using is called a Facade pattern. A Facade pattern is a design implementation that uses a single class to represent an entire subsystem. The subsystems for our software design are our two sections; the strum, and fret sections. When using a Facade pattern, it is important to consider what system is needed and how we can interact with individuals and the components of that system. The Facade pattern can be expanded on to form another type of design pattern, but for our project we will only need to use one type of design pattern. Another type of pattern that we can encounter as a problem instead of as a solution is laval flow pattern. Lava flow pattern is when olde code is still within the code document. These old code can have other code that is dependent on it. As the code size increases, and the implementation of code becomes overwhelming to maintain. A build up of 'useless' code is within our software design which we don't need.



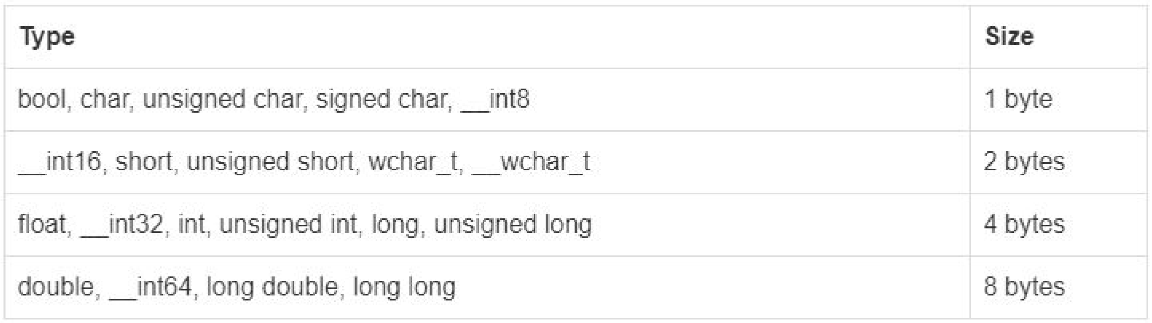
```
class Strum
{
    public:
        int strumPortNumber;
        bool strumActive;
};

class Fret
{
    public:
        int fretPortNumber;
        int fretPortSelected;
        bool fretActive;
};

void main()
{
    Strum strum1();
    Strum strum2();

    Fret fret1();
    Fret fret2();
}
```

Figure 44 - Using Facade Pattern

Figure 21 shows an example of a Facade Pattern. There are two distinct classes that encapsulate all of its member variables. These two distinct classes represent our subsystems, the strum and fret lasers. We can create multiple instances of each class without the concern of a variable sharing the same name or data affecting another class. If we needed to incorporate more components to hardware design, we can easily incorporate it within our software design. We can easily incorporate another component because we would be repeating the same process of creating a new subsystem by creating a new class into our code. Using the Facade pattern will give us the ability to easily organize our subsystems without the need to declare new variables. We can easily reuse code because Ardunio allows Object Oriented Programming.

5.2.3. Space Complexity

Space complexity is the amount of memory needed for an algorithm to achieve its desired task. Space complexity is important for our software design because if our algorithm for a specific task takes large amounts of memory space, then we could hinder our microcontrollers' performance. Not only does space complexity focus on the algorithm, but the data types used. The data type used within a code has a specific memory size. If an algorithm were to use a variable as an argument with a large memory size, then the algorithm can take longer to process the input because of how large it is. Also when creating objects, it is also important to take into consideration of what arguments are being passed to it.



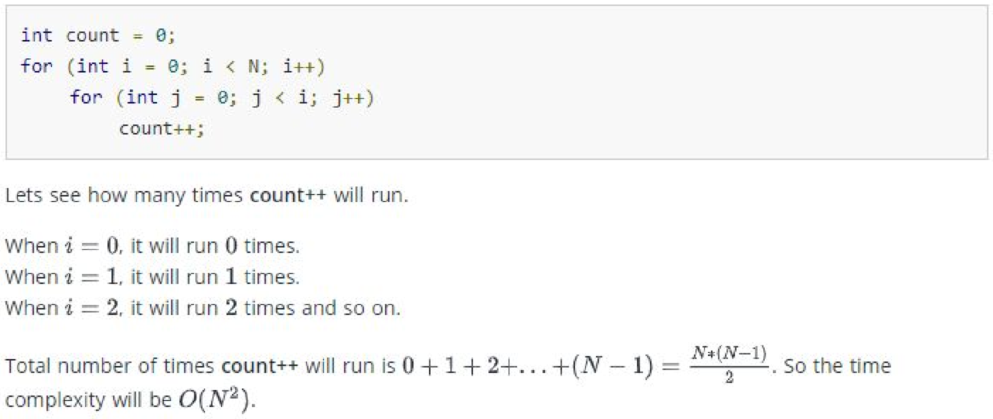
Type	Size
bool, char, unsigned char, signed char, __int8	1 byte
__int16, short, unsigned short, wchar_t, __wchar_t	2 bytes
float, __int32, int, unsigned int, long, unsigned long	4 bytes
double, __int64, long double, long long	8 bytes

Figure 45 - Data Type Sizes [11]

In Figure 45 shows a table of data types and their corresponding sizes. There are some microcontrollers that have different types of architectures, x32 or x64. When declaring variables within our code, it is important to note what type of data types are allowed on our microcontroller. Our code size may take up a lot of memory because of the variable declarations and the sizes of each object. The code size can take memory space away from the algorithm when it is executed.

5.2.4. Time Complexity

Time complexity is the amount of time it takes for an algorithm to achieve its desired task. Time complexity is important for our software design because if our algorithm takes a long period of time, then the software is not responsive to meet our requirement specification. Depending on our algorithms input, our algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input [S90]. If we have a very large input our algorithm has to take this into consideration.Time complexity can be measured using the notation of Big-O (e.g. O(n)).



```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = 0; j < i; j++)
        count++;
```

Lets see how many times **count++** will run.

When $i = 0$, it will run 0 times.
When $i = 1$, it will run 1 times.
When $i = 2$, it will run 2 times and so on.

Total number of times **count++** will run is $0 + 1 + 2 + \ldots + (N - 1) = \frac{N*(N-1)}{2}$. So the time complexity will be $O(N^2)$.

Figure 46 - Time Complexity of O(n^2) [13]

Figure 46 shows an example of how an algorithm's time complexity is calculated. The example's time complexity was calculated based on its input, N and i. The algorithm is trying to operate on the amount of time it takes to go through N and i. The algorithm is processing N and i simultaneously counting the number of iterations. This basic analysis can be expanded and applied to complex algorithms.

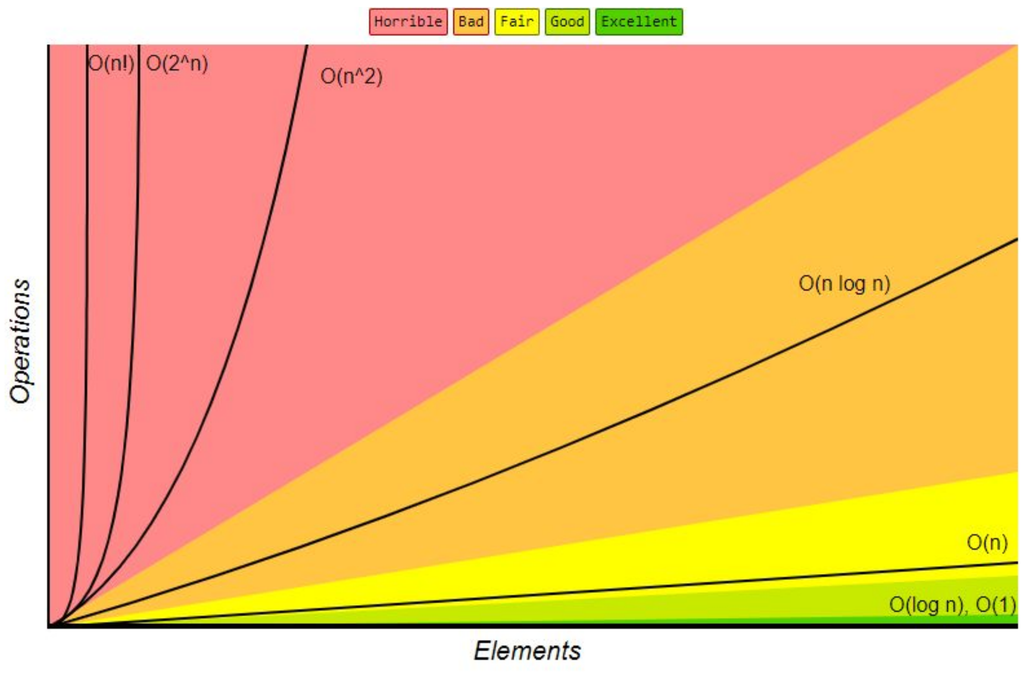


Figure 47 - Time Complexity Chart [14]

In Figure 47, O(1) is a constant time which is the absolute best. This means that the algorithm completes an operation instantly. O(n) takes the algorithm "n" amount of times to complete its takes. For each O() in figure 47, it describes how long an operation takes for an algorithms operation to complete. This analysis and process of calculating time complexity can be applied to our algorithms in our software design. There is no possible way for our algorithms to be O(1) because we would need to search for which I/O pin is being activated. We would need to search through "N" (54) I/O pins to find which one is activated and that is O(n). Our algorithm design for our software should be at max O(n^2). It should be max at O(n^2) because that is the run time for most searching algorithms. We are only searching for which I/O pins. There is no need for complex algorithms that exceed O(n^2).

5.2.5. Software Logic

In Figure 48, shows a flow chart of our thought process on how we would approach certain algorithm designs.

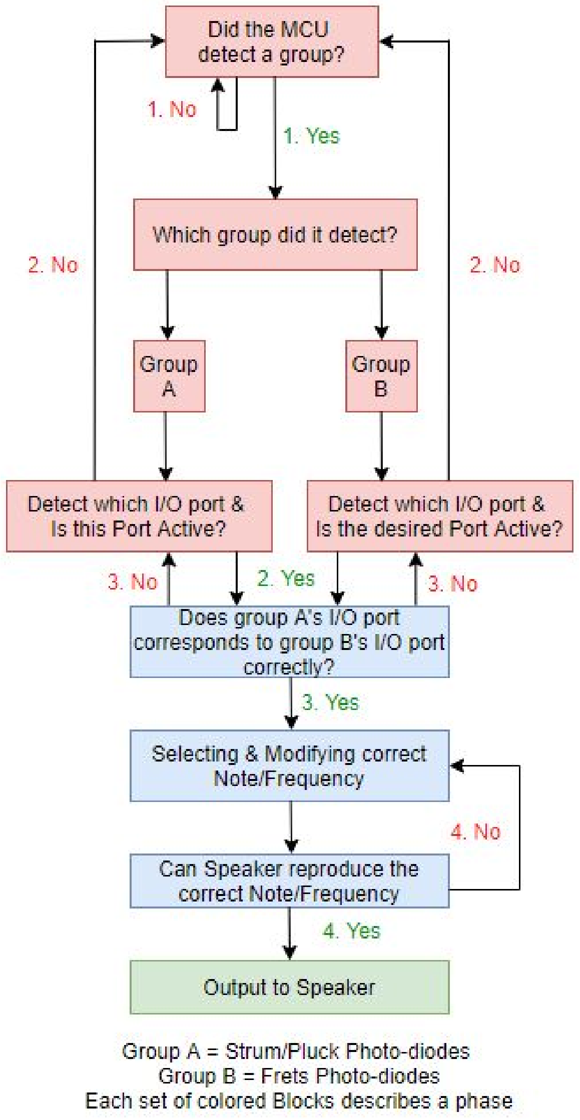

Did the MCU detect a group?
1. No
1. Yes
Which group did it detect?
2. No
2. No
Group A
Group B
Detect which I/O port & Is this Port Active?
Detect which I/O port & Is the desired Port Active?
3. No
2. Yes
3. No
Does group A's I/O port corresponds to group B's I/O port correctly?
3. Yes
Selecting & Modifying correct Note/Frequency
4. No
Can Speaker reproduce the correct Note/Frequency
4. Yes
Output to Speaker
Group A = Strum/Pluck Photo-diodes
Group B = Frets Photo-diodes
Each set of colored Blocks describes a phase

Figure 48 - Flowchart Logic

Figure 48 shows a flowchart diagram of our thought process on how we will approach our algorithm design for our microcontroller. The flowchart diagram will help all the members within the group understand how the microcontroller will achieve the desired task of our project. By understanding the logic behind how the hardware and software components will communicate with one another, it will ameliorate the process of both designing and implementing code to our hardware components.

The overall view of the flowchart diagram can be broken down into three different phases. Each of these phases have their own specific focus on the type of hardware that should be considered, and what part of our code design accommodates that specific hardware. Within each phase, there are steps to follow, and depending on which step is 'yes' or 'no' the direction path of the flowchart changes. The three different phases are color coded with different colors to show how one phase transitions into another, and how these phases are dependent on one another. The three phases are color coded as red, blue, and green.

All of the red blocks represent phase one of our logic. Within this phase, the hardware components that are taken into consideration is the microcontroller that we have selected and photodiodes. The microcontroller and the photodiodes must be taken into consideration within the first phase because without the microcontroller. This project would not be possible, and without the photodiode. The microcontroller would not achieve its desired task by detecting a laser and playing a corresponding sound when a laser is broken. The first step is to have our microcontroller detect whether a group is detected. There are two key groups; Group A and Group B. Group A will be the strum/pluck photodiode which will tell the processor to play the respective sound. Group B will be the fret photo-diodes and this will indicate what note or pitch should be played. If the microcontroller does not detect a group, then it will wait until a group is detected. Otherwise if there is a group that is the controller, then our algorithm will detect which group has been detected. This detection will dictate how step 2 can progress into phase two or not. At step 2, the microcontroller has been detected and therefore Group A and/or Group B has been detected. Group A and Group B have their own detection algorithm. Group A and Group B have their own detection because they are two different types of hardware components that are connected to our microcontroller.

Although our microcontroller detected Group A or Group B, our algorithm needs to figure out which photodiode is active for which respective group. This way the software can save the location of this port and use it throughout the software. If the algorithm does not detect a port within a group, then it will return to step 1 again. And wait for the microcontroller to repeat the same process of detecting which group is detected. If the microcontroller detects group A, this means that an interrupt was caused by a laser being broken and a photodiode is now active on the head of the guitar. This is representative of a strum/pluck. Not only does the microcontroller need to detect group A, it also needs to detect group B as well. At the neck of the guitar, when a laser is broken when a user is holding down a fret, an interrupt also occurs. Otherwise if the algorithm does detect a port within both groups, then the software design can move into phase two.

All of the blue blocks represent phase two of our logic. Within this phase, the hardware components remain the same, but now with a speaker component that we need to take into consideration. All of the photodiodes within each respective group that were detected from phase one. At step two, our algorithm will decide if Group A's photodiodes correspond to Group B's photodiodes. It is crucial to make this decision because just like on a guitar. When an individual holds down a fret on a guitar string, and then sturms the string that the fret is being held at. It will produce a certain note. We must emulate the fret and the strumming of that fret with the photodiodes. If the photodiodes of Group A and Group B do not correspond, then the algorithm must go back to phase one. If the photodiodes within Group A and Group B correspond like a strum of a fret like string on a guitar, then we can proceed to step three. In step three, there will be another algorithm that detects which note to play that corresponds to what photodiode port in Group B is active. This will emulate the sound of the fret that is being held down on a guitar. Once the selected note is picked by the algorithm, we can proceed to step 4.

Within step 4, we must ensure that the note selected by our algorithm is being emitted from the speaker correctly. In order to ensure that the note selected by our algorithm does emit from the speaker correctly. We must look at the range of frequencies that the speaker can produce. In order to test if a speaker can produce the correct notes, we must do various tests on the speaker. This will be explained more in chapter seven within this document. When a note is selected by the algorithm and is emitted by the speaker, if the sound is not properly produced, then we must change the code to emit the sound properly. If the sound is emitted properly, then we can go to phase three.

Phase three consists of one block, which is just emitting the correct note from the speaker. In conclusion the flowchart is a foundation for how the hardware and software should communicate with one another. Details of how this flowchart is translated into our software design is described in more detail in chapter six. In chapter six, the chapter discusses how the construction of both the hardware and software components of the project is constructed.

6. Construction and Coding

This section focuses on both the hardware and software construction of how the project is built. The construction of both the hardware and software construction is tentative and in progress of what needs to be completed. The hardware construction can be described in details in section 6.1, and the software construction can be described in details in section 6.2.

6.1. Software Construction

In figure 27 shows a UML diagram for our software construction. This construction is a foundation to how we will code.

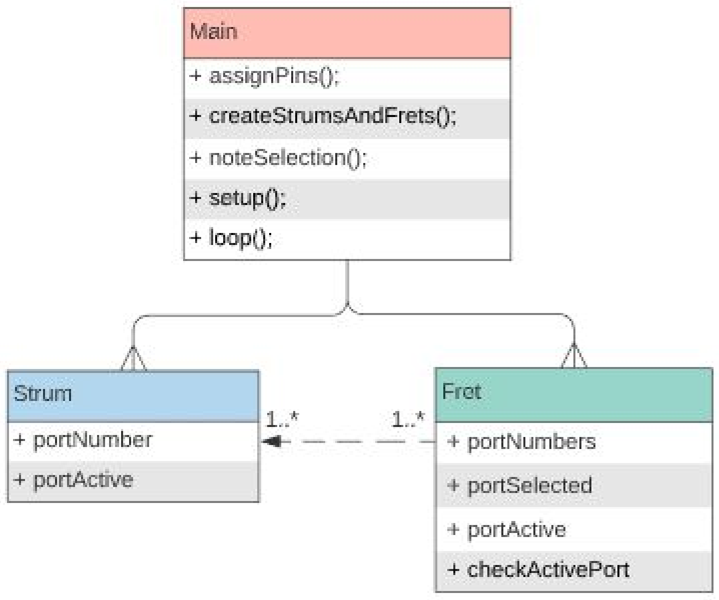


Figure 49: UML Diagram

The UML diagram also known as a Unified Model Language shows how classes are interacted with one another. This is a visual representation of our software design implementing Object Oriented Programming using a Facade pattern. The implementations of Object Oriented Programming and Facade pattern has been discussed in section 5.2.1 and 5.2.2. Within section 6.2, goes into detail of how we chose to design our software design using Object Oriented Programming with a Facade pattern.

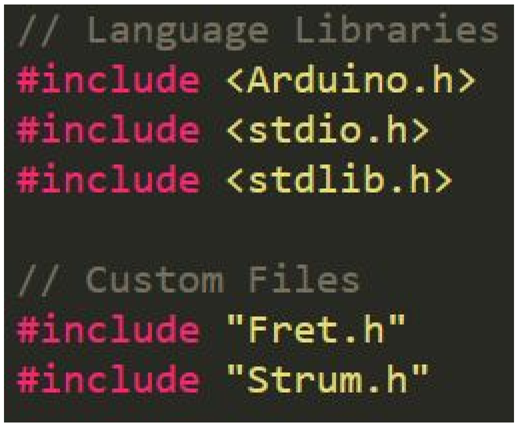
6.2. UML Diagram

The Unified Model Language diagram in Figure 27, is a visual representation of how the software implementation is designed. In Figure 27, there are three components; Main, Strum, and Fret. Main can be considered as the main file that will generate the other two compliments which are classes. Within each component within the Unified Model Language diagram, contains variables and functions specific to that complement. There are lines going into the Strum and Fret class. This line is known as an association line. The association line allows readers what component is associated with another component. The Main component is associated with both the Strum and Fret class. The symbol '+' within the Main component indicates the function is a public access modifier. Access modifiers are a type of access a class can have to a function or variable within a class. There are private, public, protected, and default access modifiers that a class can implement for each of its functions and variables it has.

The numbering label '1..*' in between the Strum and Fret class is an indication of many many components there can be. This numbering is translated as one to an unlimited amount of that specific complement there are. There can be one or as many Strum and Fret classes that we can create within our software. The association line that is coming from the Fret class to the Strum class is a dependency line. In our software our Fret class is dependent on the number of Strum classes there are. For some number of Strum classes that are created, there is the same amount of Fret classes as well.

6.2.1. Main

Main is the core for every program. It is required within our software implementation to run our code. Main is not necessarily a class, but is the main file that we will need to run all of our instruction and control our microcontroller. We chose to program in the Arduino language which is both C and C++ language. C and C++ languages require a main within their code design, therefore it is present within the Arduino language. For our software construction, we must import all of the necessary libraries and custom files. We must import all of the necessary libraries and custom files because there are already built in functions within these libraries and custom files that we can use. This can reduce the need to write custom functions that can hinder both space and time complexities that our microcontroller needs to complete our objective, and meet our design requirement specification.



```
// Language Libraries
#include <Arduino.h>
#include <stdio.h>
#include <stdlib.h>

// Custom Files
#include "Fret.h"
#include "Strum.h"
```

Figure 50 - Imported Libraries and Files

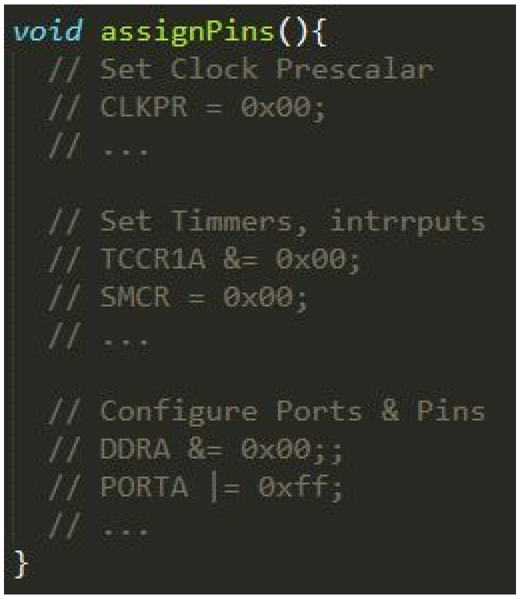
In Figure 50, shows the imports we need for our project. The importance can be categorised into two sections; the language libraries and custom files. The language libraries are libraries already created by Arduino. Each language library has its own unique functions that we can call and use. These files are used to create the Fret and Strum class respectively.

By creating these two custom files, we can represent each class within its own file without need to put these two classes within Main. This helps reduce the code size of main while keeping everything modular. It is crucial that we import these two files because we will not be able to implement Object Oriented Programming and use a Facade pattern. Not only will we not be able to use these techniques, but the code will not be able to recognize the syntax of what we are trying to infer in the code.

Within the Main file, not only does it include imports, it also contains the main method to run our code, and other functions necessary to run. The Main file consists of; assignPins, createStrumsAndFrets, noteSelection, setup, and loop.

assignPins

The assignPin is a public function that has no return type. The objective of this function is to enable certain functionality of what the microcontroller has to offer. Figure X below shows brief pseudo code of configuring clocks, timers, ports and pins.



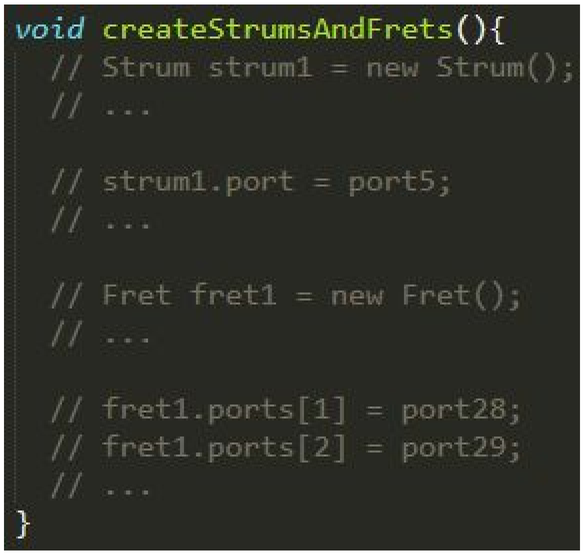
```
void assignPins(){
  // Set Clock Prescalar
  // CLKPR = 0x00;
  // ...

  // Set Timmers, intrrputs
  // TCCR1A &= 0x00;
  // SMCR = 0x00;
  // ...

  // Configure Ports & Pins
  // DDRA &= 0x00;;
  // PORTA |= 0xff;
  // ...
}
```

Figure 51 - assignPins Function

All microcontrollers contain a clock as it is an essential component that is used extensively when working with microcontrollers. These clocks may be internal such as a RC oscillator or external such as a crystal oscillator. Deciding which clock to use typically comes down to factors such as accuracy and cost. The microcontroller we have chosen has several clock source options, such as a crystal oscillator at 16MHz or the internal RC oscillator at 8MHz. Clocks play such a pivotal role in microcontroller applications due to the fact that many applications rely on clock signals to achieve their goals. Clock signals are used by the CPU and peripherals, as well as other use cases like the control of baud rate in serial communication signals or keeping track of time taken for digital-to-analog conversions and vice versa. Timers or counters are also a feature heavily used in microcontrollers and they typically follow the ticks of the clock. Our microcontroller comes with 6 Timer/Counters, which can be configured accordingly by referencing it's datasheet. In our project we want to make sure that our laser instrument plays sound quickly and at the appropriate time which is when a laser beam has been broken. Timer interrupts should be considered in the event that external interrupts are not being triggered quick enough to meet our requirement specification for response time. Implementing timer interrupts can help ensure that we play a sound quick and on time.

createStrumsAndFrets

The createStrumAndFrets function is a public function that has no return type. The objective of this function is to create instances of Strums and Frets, and assign their variables.



```
void createStrumsAndFrets(){
  // Strum strum1 = new Strum();
  // ...

  // strum1.port = port5;
  // ...

  // Fret fret1 = new Fret();
  // ...

  // fret1.ports[1] = port28;
  // fret1.ports[2] = port29;
  // ...
}
```

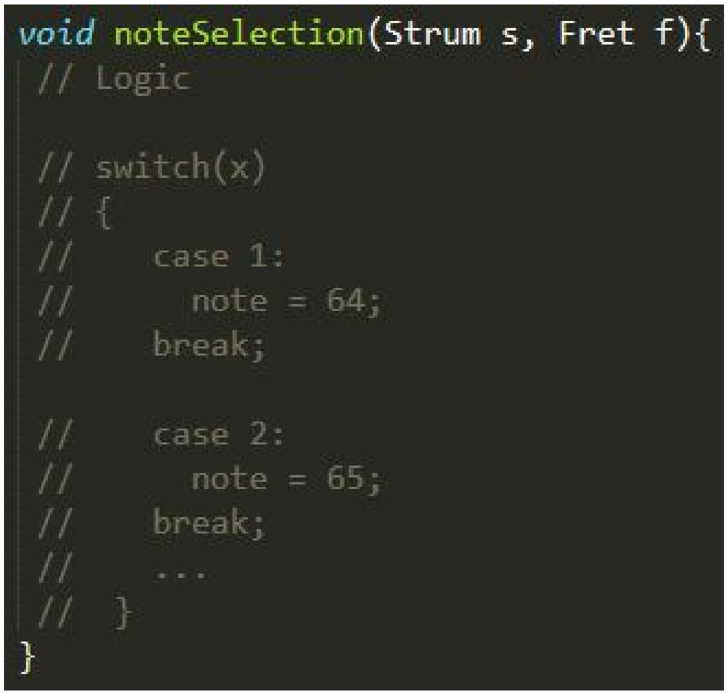
Figure 52 - createStrumsAndFrets Function

Within the createStrumsAndFrets function, we are creating multiple instances of the Strum and Fret class. By creating objects we can differentiate what photodiodes correspond to what laser system we have set up. After creating these objects we can assign all of its member variables accordingly. For the Fret objects, we have predetermined what port numbers each photodiode will be assigned to.The ports that are assigned for each photodiode can be grouped. The group of photodiode port numbers can be placed in an array. We can hardcode each index of a Fret object's array member variable with the port number. By hardcoding the port numbers at each index, the group knows what index to retrieve when needed.

noteSelection

The noteSelection function is a public function that has no return type. The objective of this function is to select a note based on the corresponding fret that is being played.

The function selection has two parameters, and will take a Strum object and a Fret object as its argument. We must take two objects as an argument because we need to access the member variables from the Strum and Fret objects. We will use the member variables from both of these objects and compare them. After the logic for the comparison occurs, we can use a switch statement to determine what to play from the speaker. We can have multiple switches for each note, at various frequencies for its cases. In Figure 53 below shows the basic foundation of how this function should be implemented.



```
void noteSelection(Strum s, Fret f){
  // Logic

  // switch(x)
  // {
  //    case 1:
  //      note = 64;
  //    break;

  //    case 2:
  //      note = 65;
  //    break;
  //    ...
  //  }
}
```

Figure 53 - noteSelection Function

setup

The setup function is the main function like C and C++ language. Since the setup is the main function of the program, the objective of the setup is to give instructions to the microcontroller.

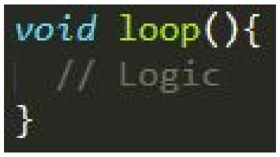
```
void setup(){
  // Functions
}
```

Figure 54 - setup Function

The setup function will just call our functions we have called or run any algorithm we give it. When setup is first initially called, it will execute the function assignPins, and createStrumsAndFrets. We must call these functions in this order because assignPins will assign all of the microcontrollers internal setup, ports and pins. If this function is not called first, then the microcontroller has no instructions to work with. Thus there will be no readings from the photodiode and nothing will happen.

loop

The loop function is what will keep the code iteration forever until a conflict occurs.



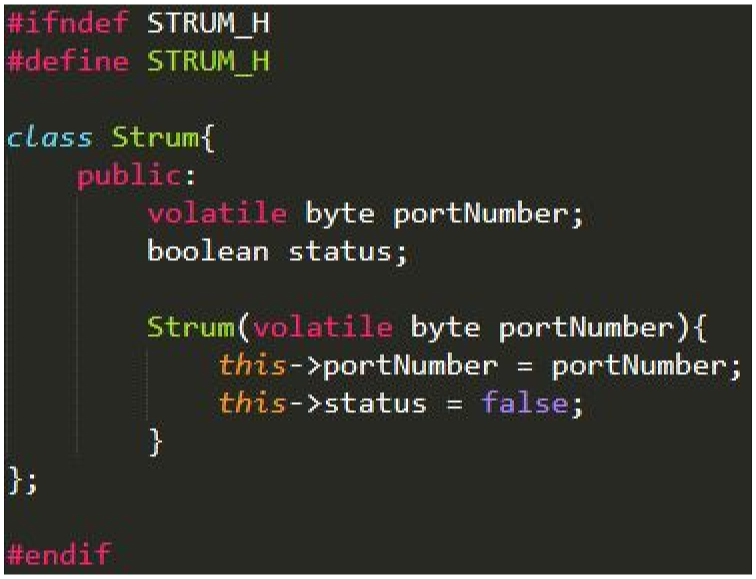
```
void loop(){
  // Logic
}
```

Figure 55 - loop Function

After the setup function is called, the loop function will be called next. The loop function will contain code to detect when a photodiode is activated. When a photodiode is activated, we can call the noteSelection function to play a sound to the speaker. Most of the algorithms to detect a photodiode activation will be here.

6.2.2. Strum Class

The Strum class will be a custom external file that will be imported to create all of the Strum objects.



```
#ifndef STRUM_H
#define STRUM_H

class Strum{
    public:
        volatile byte portNumber;
        boolean status;

        Strum(volatile byte portNumber){
            this->portNumber = portNumber;
            this->status = false;
        }
};

#endif
```

Figure 56 - Strum Class

The Strum Class is an external file that we must import for us to create a Strum object. In order to create a Strum class file we must create a header file. We can create a header file by wrap out code in between '#ifndef' and '#endif'. By wrapping the code in between '#ifndef' and '#endif', the compiler will compile the header file and all of its contents within it. If there are multiple header files with the same name, the '#ifndef' will prevent the compiler from compiling the same header file name. Creating a class is similar in Arduino language like any other language like Java or Python. We must start the file off with the syntax 'class' and the class name with a capital letter. Within this class, we can create member variables and a constructor.

The member variables of the Strum class will be public. The member variables will be public because we need to modify these member variables within Main. When we implement our algorithm to detect an interrupt, we can easily change the status of the Strum object without the need to redeclare a set variable in Main.We can easily change all member variables of a class by using the '.' operator and then use the member variable name. For example, we can easily change the status by coding 'Strum1.status = true'.

The Strum class also has a custom constructor that we can pass a portNumber as an argument. A constructor can be considered as the input of information to an object. The data that is being passed to our constructor is a portNumber. When passing a port number when declaring an instance of a Strum, we can quickly assign a port number to it, and the strum a status. We can give a Strum all of the information specific for a certain Strum objective. We can use this pattern, a Facade pattern for every Strum we create.

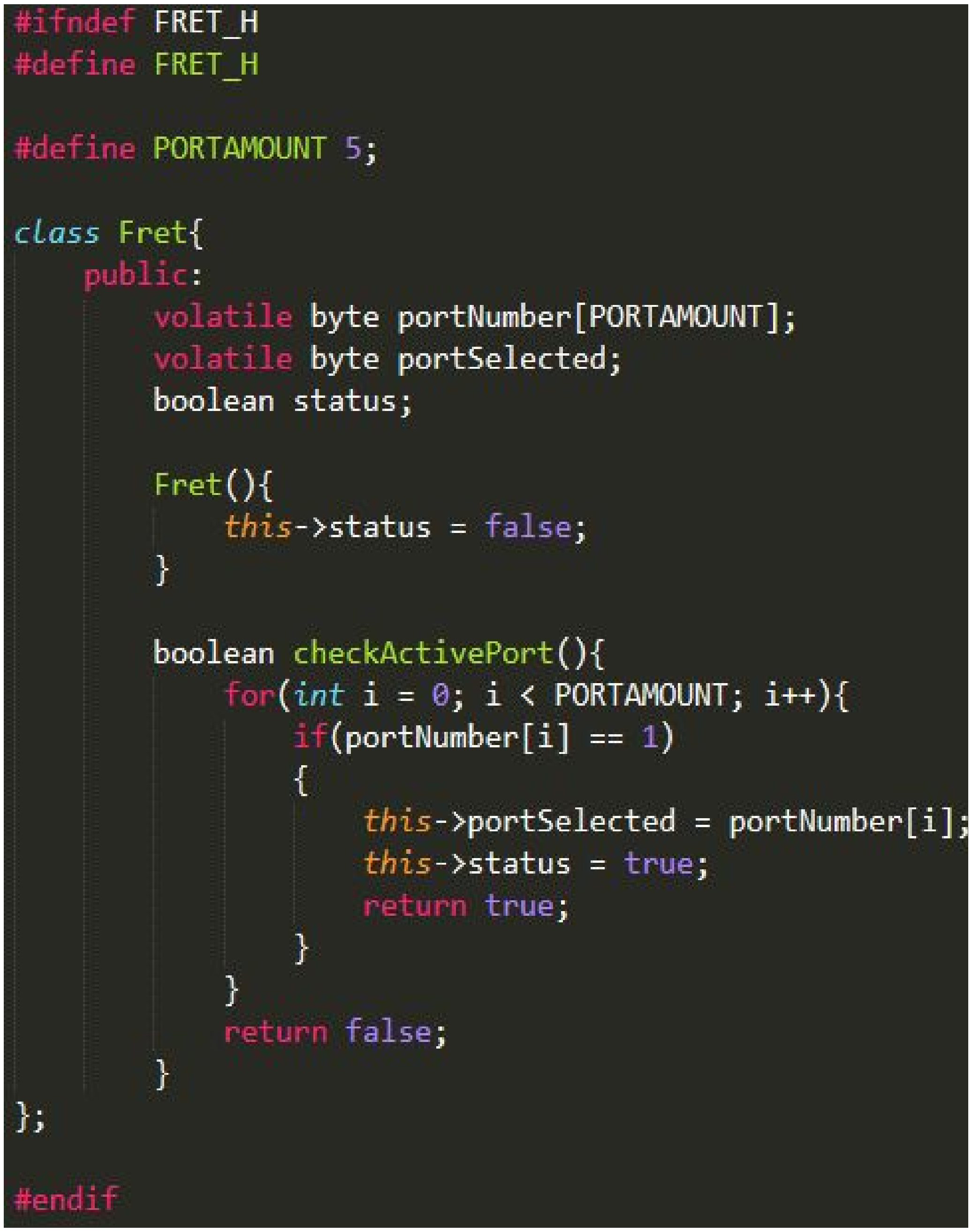
There are no methods (functions) within the Strum class. There are no methods within the Strum class because we only need to declare a port number to it and a status. The Strum class will represent a user strumming. When a user is strumming, the photodiode either detects the broken laser or not at a specific port. Thus, there is no need for methods within this class.

6.2.3 Fret Class

The Fret Class is an external file that we must import for us to create a Fret object. Similar to how we create a Strum class, the code must be wrapped in between '#ifndef' and '#endif'.

The member variables of the Fret class will be public. The member variables will be public because we need to modify these member variables within Main. When we implement our algorithm to detect an interrupt, we can easily change the status of the Fret object without the need to redeclare a set variable in Main similar to changing a Strum object. The member variable portNumber is an array of port numbers that we have hard coded within the createStrumAndFret function within Main.

Every element of the portNumber array has a port number that corresponds to all of the Frets on the guitar. This will help us group what ports are being used and occupied.



```
#ifndef FRET_H
#define FRET_H

#define PORTAMOUNT 5;

class Fret{
    public:
        volatile byte portNumber[PORTAMOUNT];
        volatile byte portSelected;
        boolean status;

        Fret(){
            this->status = false;
        }

        boolean checkActivePort(){
            for(int i = 0; i < PORTAMOUNT; i++){
                if(portNumber[i] == 1)
                {
                    this->portSelected = portNumber[i];
                    this->status = true;
                    return true;
                }
            }
            return false;
        }
};

#endif
```

Figure 57 - Fret Class

The Fret class also has a custom constructor that has no arguments. The constructor will only set the Fret's object status. No further action will be needed from the constructor unless there needs to be a redesign of how we want to implement our algorithm.

There is only one method within the Fret Class. This method is public and returns a boolean value. Within this method, we will check which port within our array is active. The algorithm for checking which port number has a time complexity of O(n) which is fast and linear. The algorithm consists of a for loop that will iterate up to the length of the array, which is the port amount of five. As we iterate through the for loop, if one of the ports in the array is active then, we will change the member variable portSelected the actual port number. And set the status of the Fret objective to active. This will give us information that the current Fret is active at a specific port number. We can easily call this method within the loop when we design our algorithm to detemer which Fret is being used.

6.3. I/O Programming

In this section we will cover the importance of our microcontroller schematics and data sheet. These two things are essential as they will ultimately be a guide on how to correctly program and configure the appropriate pins and registers.

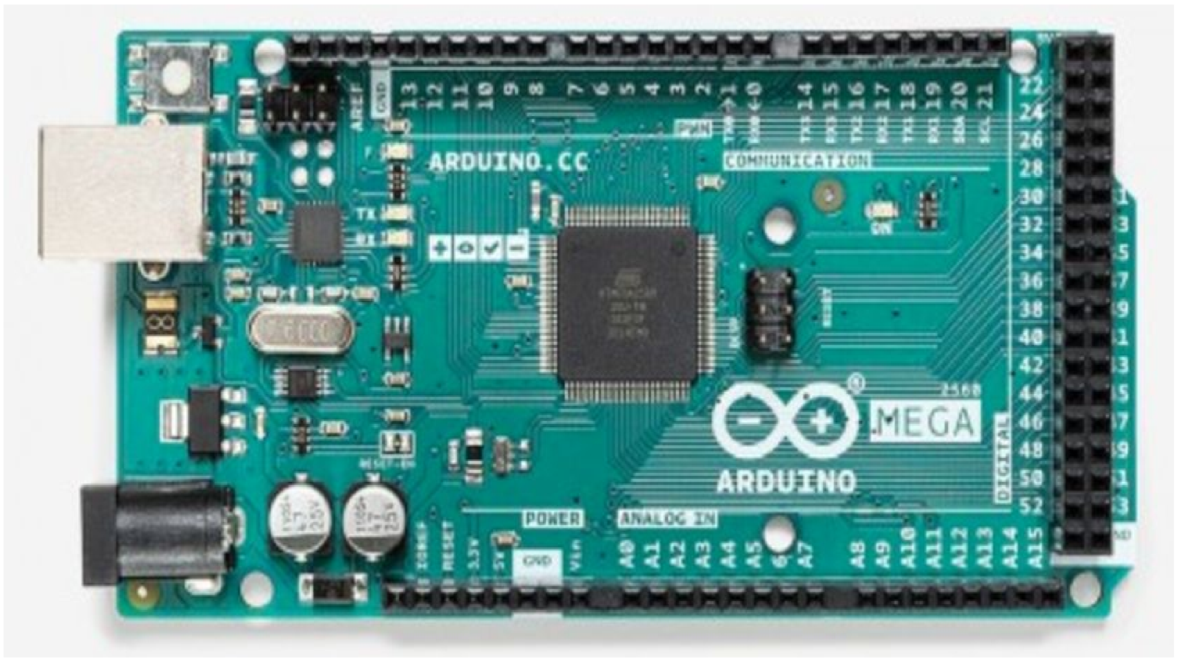


Figure 58 - Arduino Board Overview [19]

In Figure 58, is the Arduino Board Overview. We can see that there are 54 I/O pins that we can use. Based on these 54 I/O ports, each port has unique characteristics that we can program with when the slots are being used. The following figures below show the characteristics of each I/O port. These ports have, but are not limited to registers, timers, and regulators. By understanding what these ports are, and how they are used, it helps with hard cording the port numbers. The different functionalities of these ports and how to program them accordingly will be covered in more detail below.

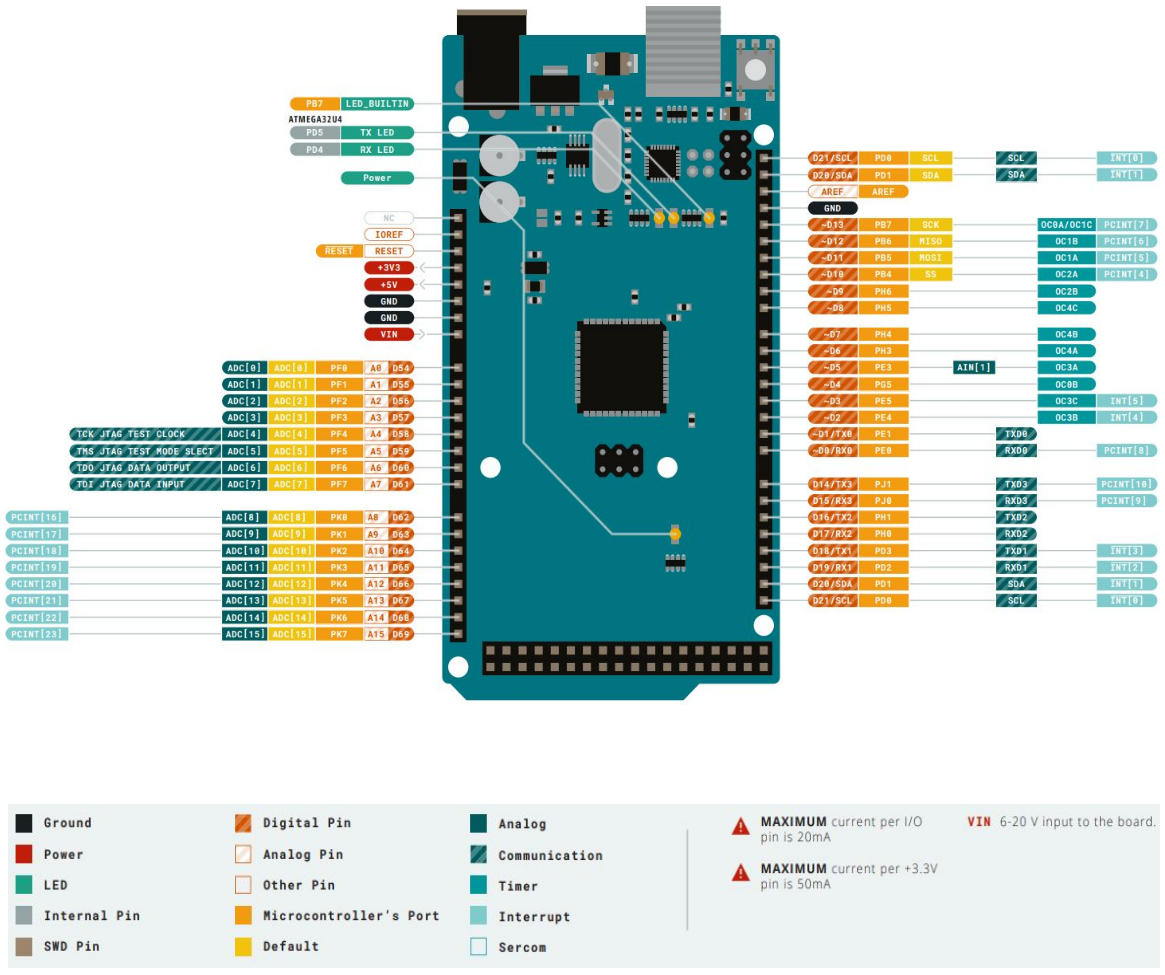

PB7
LED_BUILTIN
ATMEGA32U4
PD5
TX LED
PD4
RX LED
Power
NC
IOREF
RESET
+3V3
+5V
GND
VIN
ADC[0] PF0 A0 D54
ADC[1] PF1 A1 D55
ADC[2] PF2 A2 D56
ADC[3] PF3 A3 D57
TCK JTAG TEST CLOCK ADC[4] PF4 A4 D58
TMS JTAG TEST MODE SLECT ADC[5] PF5 A5 D59
TDO JTAG DATA OUTPUT ADC[6] PF6 A6 D60
TDI JTAG DATA INPUT ADC[7] PF7 A7 D61
PCINT[16] ADC[8] PK0 A8 D62
PCINT[17] ADC[9] PK1 A9 D63
PCINT[18] ADC[10] PK2 A10 D64
PCINT[19] ADC[11] PK3 A11 D65
PCINT[20] ADC[12] PK4 A12 D66
PCINT[21] ADC[13] PK5 A13 D67
PCINT[22] ADC[14] PK6 A14 D68
PCINT[23] ADC[15] PK7 A15 D69
D21/SCL PD0 SCL SCL INT[0]
D20/SDA PD1 SDA SDA INT[1]
AREF AREF
GND
~D13 PB7 SCK OC0A/OC1C PCINT[7]
~D12 PB6 MISO OC1B PCINT[6]
~D11 PB5 MOSI OC1A PCINT[5]
~D10 PB4 SS OC2A PCINT[4]
~D9 PH6 OC2B
~D8 PH5 OC4C
~D7 PH4 OC4B
~D6 PH3 OC4A
~D5 PE3 AIN[1] OC3A
~D4 PG5 OC0B
~D3 PE5 OC3C INT[5]
~D2 PE4 OC3B INT[4]
~D1/TX0 PE1 TXD0
~D0/RX0 PE0 RXD0 PCINT[8]
D14/TX3 PJ1 TXD3 PCINT[10]
D15/RX3 PJ0 RXD3 PCINT[9]
D16/TX2 PH1 TXD2
D17/RX2 PH0 RXD2
D18/TX1 PD3 TXD1 INT[3]
D19/RX1 PD2 RXD1 INT[2]
D20/SDA PD1 SDA INT[1]
D21/SCL PD0 SCL INT[0]
Ground
Power
LED
Internal Pin
SWD Pin
Digital Pin
Analog Pin
Other Pin
Microcontroller's Port
Default
Analog
Communication
Timer
Interrupt
Sercom
MAXIMUM current per I/O pin is 20mA
MAXIMUM current per +3.3V pin is 50mA
VIN 6-20 V input to the board.

Figure 59 - Arduino Schematic 1 [18]

Figure 49 above shows a pinout diagram of the Arduino Mega 2560, and displays Pins D0-D69.We can see that each pin is color coded which tells us what type of pin it is, which port it is in and what functionality it can provide. Pins are the physical elements that allow us to make electrical connections. Some pins can be configured to serve multiple purposes such as pin D21. Pin D21 is a digital pin meaning it can take digital signals as input or output them. However, it can also be configured to handle communication and interrupts.

All pins are a part of a port which are depicted by registers. Configuring these registers are what allow developers to control the state of pins, whether it be setting it as input, output or any of the functionalities that pin may support. There are a total of three registers that are configured in order to control the input/output pins: DDR, PORT, and PIN. Every port possesses its own set of registers. For instance, port F which can be found from pins D54 to D61 as seen in Figure 49 above, has registers DDRFn, PORTF and PINFn, with ‘n’ being the bit number. Port K found on pins D62-D69 has registers DDRKn, PORTKn, and

PINKn, and the rest of the ports follow this same methodology. This will be covered in more detail.

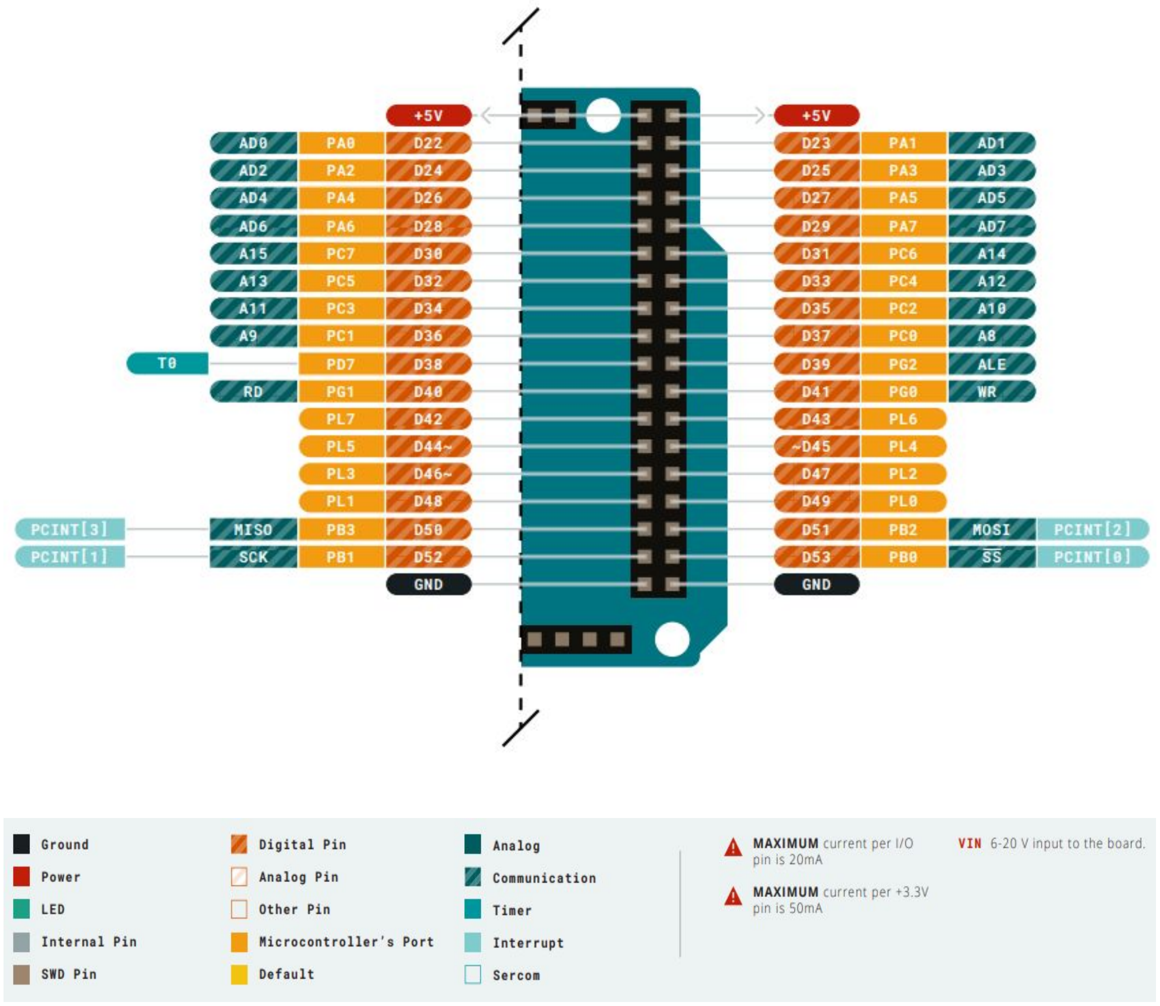
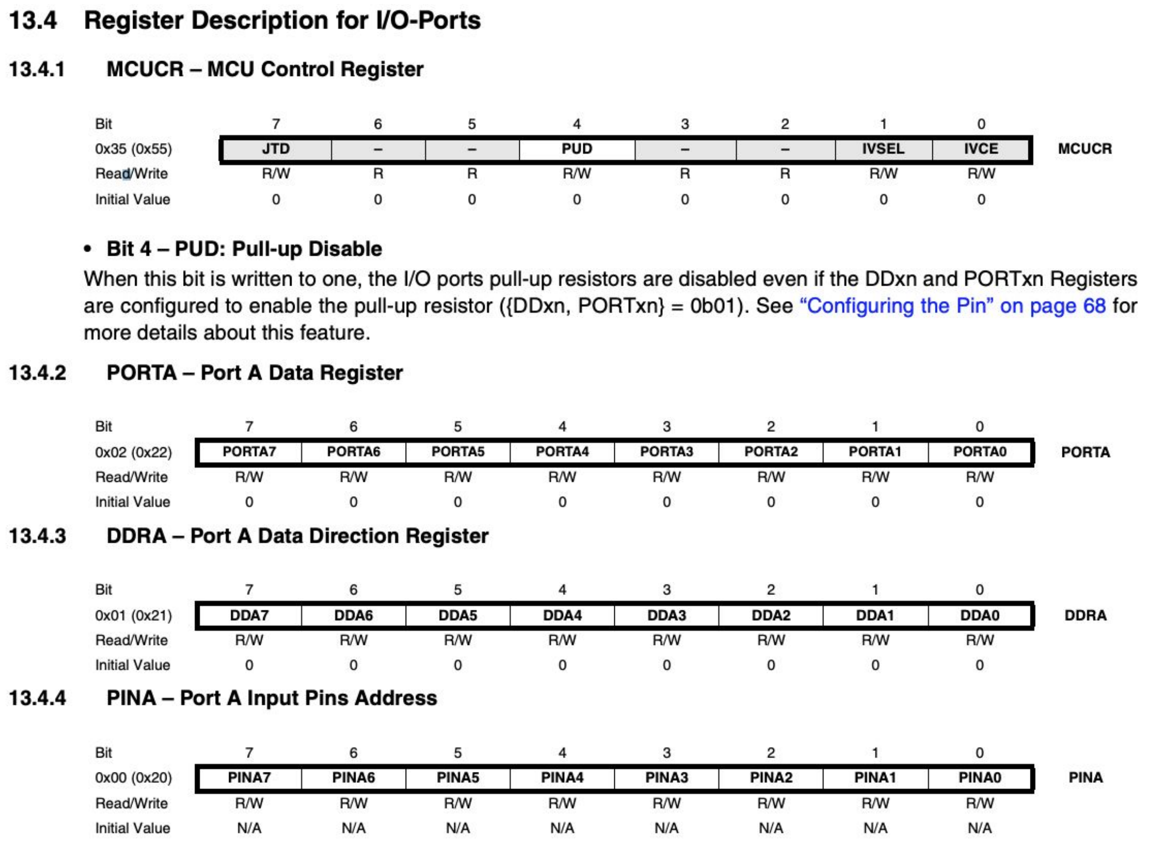


Figure 60 - Arduino Schematic 2 [18]

Figure 60 above shows another pinout diagram of the Arduino Mega 2560. Figure 60 is an extension of Figure 49 that displays and labels pins D22-D53. These are the bottom most pins if the microcontroller is being held vertically. The view in Figure 60 displays these pins with the microcontroller being held horizontally similar to the orientation in Figure 48 (Arduino board overview). We must be aware of all the pins made available to us, given that our project will require us to use many of the Digital pins as input and output in order to read from all our photodiodes, set up our lasers and be able to output the sounds to a speaker. FIgure X below gives us a better description of how these pins are actually configured in order to achieve the desired functionality. Although Figure 60 and Figure 61 gives us what I/O ports and pins are available to use, we also need to look at what registers are available at a specific port. We need to look at the register of a specific port because we want to maximize the features of each port to help improve our codes algorithm. We can see all of these specific registers within the datasheet [17].



13.4 Register Description for I/O-Ports

13.4.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R/W	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the I/O ports pull-up resistors are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-up resistor ({DDxn, PORTxn} = 0b01). See "Configuring the Pin" on page 68 for more details about this feature.

13.4.2 PORTA – Port A Data Register

Bit	7	6	5	4	3	2	1	0	
0x02 (0x22)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

13.4.3 DDRA – Port A Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x01 (0x21)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

13.4.4 PINA – Port A Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x00 (0x20)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

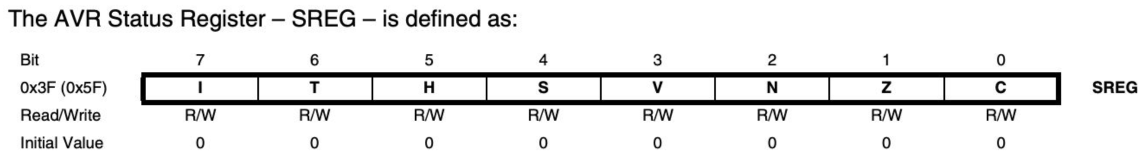
Figure 61 - Register Description of I/O Ports [17]

Figure 61 above gives us a general overview of the I/O ports from our datasheet. As was previously discussed before we can see that port A has the three registers,DDRA, PORTA, and PINA. Each register is composed of 8 bits. DDR is used to set the direction of the pin. If for example, DDRA1 is set to 1 this means that pin 1 has now been configured to be an output pin. On the other hand, if DDRA1 is set to 0 this means that pin 1 has now been configured as an input pin.

The PORT register is configured similarly to DDR. We use this register to set the output value. For example, if DDRA1 was set to 1 making it an output pin and PORTA1 is also set to 1 then the voltage for pin 1 will now be set to 5 volts. If PORTA1 is set to 0 and DDRA1 is still configured as an output pin then the voltage for pin 1 is now set to 0 volts. We must also consider what happens when a pin is configured as input and we try to set the PORT value of that respective pin. This introduces the concept of the pull up/ pull down resistor. The pull up/ pull down resistors are used so that any pin that is not connected or is left floating can be set to a default value. When pins are left free the

microcontroller might not know whether the input value is high or low and thus the pull up/ pull down resistors take care of this issue.
In the case that DDRA1 is set to 0 making it an input pin and and PORTA1 is set to 1 this will activate the pull-up resistor. If PORTA1 was set to 0 then the resistor would be pulled down in this case.

Lastly, the PIN register is simply used to read what value is contained by that respective pin. If the value at PINA1 is set to 1 we know that its value is 5 volts, and if it is set to 0 then we know the value is 0 volts. This methodology can be used to configure all other pins. However, it is important to note that some pins have different functionalities and therefore, we must check the data sheet in order to correctly configure the pins within our software implementation.


The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 62 - AVR Status Register [17]

Figure 62 above shows the AVR status register which stores information about the arithmetic instruction that was last executed. This information is important to keep track of as it allows us to change the flow of the program in order to achieve a desired outcome. An important thing to take away about this register and something that must be taken into account in regards to our project is BIT 7 which is labeled as ‘I’ above, is the Global Interrupt Enable. Interrupts will play a key role in our project and if the Global Interrupt Enable bit is not set to 1 then none of the interrupts will be enabled. Therefore, we must remember to set this bit in order to enable any other interrupts which have their individual enable configurations.

7. Prototype Testing

With the COVID-19 pandemic and social distancing changing the circumstances of testing for our project and group work in general in the year 2020, it was important that our group had a plan on how we would tackle this project safely and while working mostly from home. For the research phase of our project, most of the work was done individually, and in our biweekly meetings we would catch everyone up on our groups progress. For testing, however, our group needed to meet in person so we could all have access to the individual parts of our project. Thankfully, three out of our four group members were in Orlando for a period of time, so we were able to meet up with each other and work in person for our initial testing, while our fourth member was present on Zoom, assisting with software testing. However, distance did become a problem after our initial testing, since one of our members had to travel for work, leaving us with just two team members in our group available to meet for a large portion of Senior Design 1. The following section outlines how we went about testing our parts and software to prepare ourselves for completing our project in Senior Design 2 on the days we were all available to test in person.

7.1. Hardware Testing Environment

Once our parts were shipped, our group scheduled a meeting to begin our initial testing for the parts we received. We met at a group members house and began testing our Laser Diode, Photodiodes, and MCU using the lab equipment sent to us by UCF and a multimeter owned by one of our group members. UCF provided us with the Analog Discovery 2 portable USB laboratory to use for our hardware testing. In order to use the equipment sent by UCF, we needed to calibrate the Analog Discovery 2 using the multimeter we already owned. The figure below shows us calibrating our oscilloscope to use for further testing.

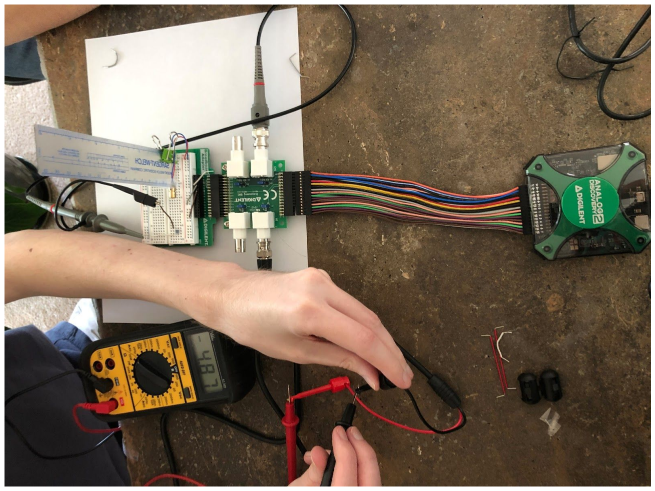


Figure 63: Calibrating the Analog Discovery 2

Once our equipment was calibrated and we had downloaded the necessary software, we were able to supply a voltage to our laser diodes and our photodiodes and begin testing our hardware.

7.2. Hardware Specific Testing

For Senior Design 1, our goal was to have all of our crucially necessary equipment ordered and tested by the end of the semester. We began by testing our laser diodes by using the Analog Discovery 2 to apply 5 volts to the diode. In doing so, we were able to observe a beam with a diameter of roughly 6mm, as advertised in the GeeBat Mini 650nm Specifications. The figure below shows the laser diode turned on with 5 volts applied.

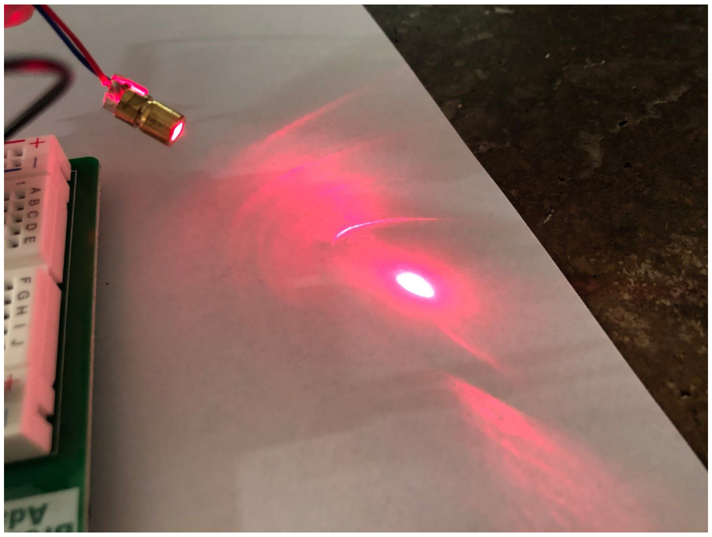


Figure 64: Laser Diode Testing

Unfortunately, without a power meter we had no way of testing the output power of our Laser Diodes, which would be valuable information to have. According to the specifications of the GeeBat Mini, however, the output power was assumed to be approximately 5mW for our testing purposes. For our final testing, we are hoping to have access to a power meter to provide more precise readings, however it is not fully necessary for the purposes of our project.

Next, We tested our photodiodes. First we tested the photodiodes in photovoltaic mode by measuring the voltage across the photodiode under dark and light conditions. Each of the photodiodes produced a voltage when exposed to light, meaning they were able to function in photovoltaic mode. This was reassuring at first, because we were unsure whether ambient light would be a major issue, however we saw a significant change in the voltage regardless of the ambient conditions. The figure below shows us measuring the voltage across the photodiode while operating in photovoltaic mode.

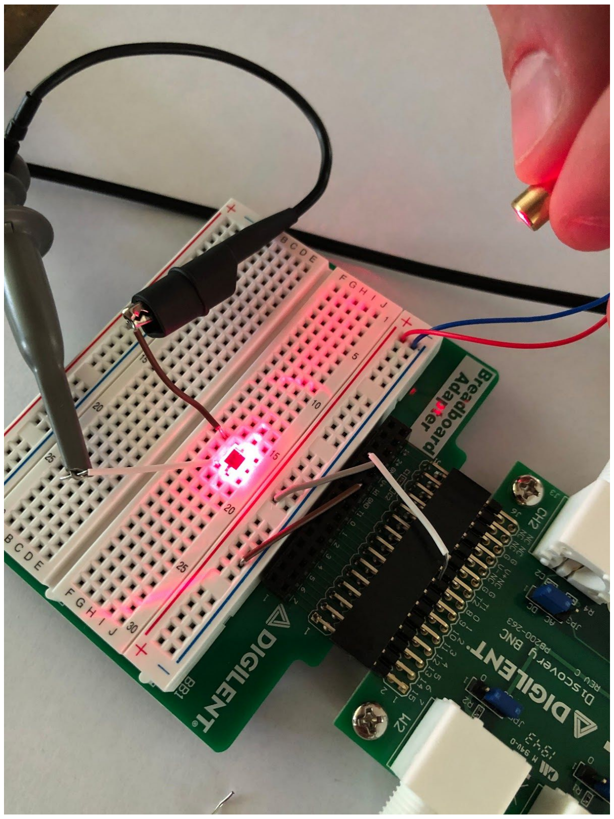


Figure 65: Measuring the Voltage Across Photovoltaic Photodiode

The room was fairly well lit, but we were still able to see a large discrepancy between values depending on if the laser was on or off. The photodiodes would provide an average of 26.3mV of electricity when the laser was off, and this value would jump to an average of 56.6mV when the laser was directly on the photodiode. The results of our tests are summarized in the table below.

Photodiode	1	2	3	4	5	6	7	8	9	10
Dark Voltage (mV)	26	27	26	28	25	24	27	27	26	27
Bright Voltage (mV)	57	56	56	57	58	57	56	55	56	58

Table 18: Photodiode Photovoltaic Voltages

While it was reassuring that each of the photodiodes worked in photovoltaic mode, it is more important to our project that they can operate in photoconductive mode. To test the photodiodes in photoconductive mode, we used the Analog Discovery 2 to apply a reverse bias on our photodiodes of -5V, and measured the current when no light was present, and when light was present. To make the testing closer to how the actual device will operate, the photodiode was placed in series with a 30kΩ resistor. We then mounted a laser diode on a ruler half a centimeter above our breadboard, and blocked the light using our fingers and measured the current produced by the light that was reflected off the users finger, rather than measuring the current with the laser light directly on the photodiode. The figure below shows the experimental set up.

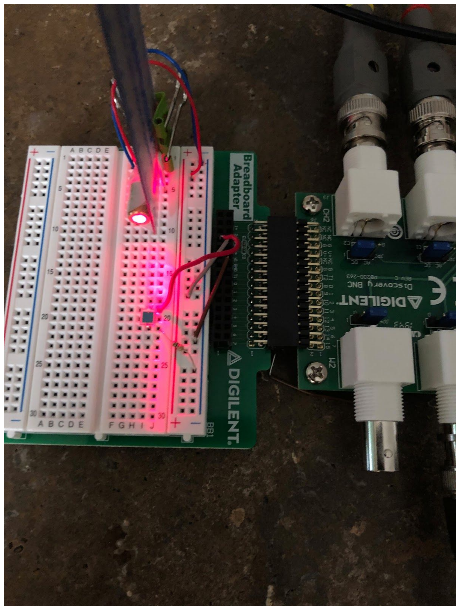


Figure 66: Testing The Photodiodes in Photoconductive Mode

The table below summarizes the results of our testing for each of the photodiodes operating in photoconductive mode. The current was measured by breaking the circuit between the photodiode and the resistor, and the voltage was calculated by multiplying by the resistance.

Photodiode	1	2	3	4	5	6	7	8	9	10
Dark Current (μA)	1.5	1.7	1.4	2.1	1.8	1.6	1.5	2.4	1.6	1.3
Bright Current (μA)	27	31	25	29	26	31	24	28	23	25
Dark Voltage (V)	0.045	0.051	.042	.063	.054	.048	.045	.07	.048	.039
Bright Voltage (V)	0.81	0.93	0.75	0.87	0.78	0.93	0.72	0.84	0.69	0.75

Table 19: Testing the Photodiodes in Photoconductive Mode

The tests were done by applying -2V to the photodiodes using the Waveforms application sent to us by UCF. This caused the photodiodes to act as a current source and create a voltage across the resistor placed in series with the photodiode. Again, due to the large amount of natural ambient light in the room we were testing in, some of the values for dark current are inflated, and the discrepancy between voltages would be greater in a dark setting. However, we were still able to achieve a consistent increase in voltage of just under 1V using a 30kΩ resistor. This is high enough for us to read a difference between the high and low voltage values in our software programming, however there are certainly improvements that can still be made to the design when looking into our initial tests. The paragraph in the section titled "Hardware Initial Testing Conclusions and Potential Improvements" discusses how we can potentially improve this voltage discrepancy to send more clear high and low voltage values to our MCU. Next, we considered the testing environment for our voltage regulator, as demonstrated in Figure 67 below.

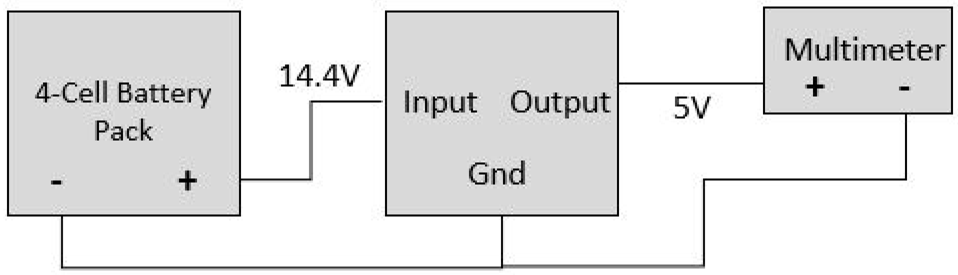


Figure 67: Voltage Regulator test environment

For the voltage regulator, the testing design will be a linear model while the final PCB mounted one will be a switching model that will go under further testing following. The objective here is to verify that the output voltage of 5V is indeed the output voltage, so that when we supply the MCU. It is receiving the correct supply amount. After confirming that we can draw a voltage from the photodiodes to our MCU we began testing our speaker in conjunction with our MCU to play sound from the strumming system of our project.

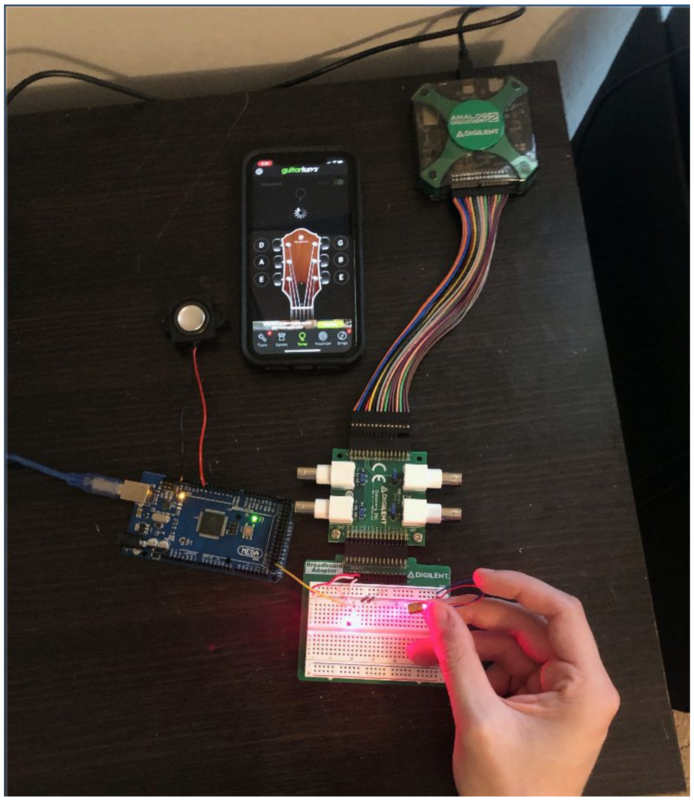


Figure 68: Strumming Interrupt Testing

In order to test our strumming system we designed a circuit consisting of our laser diode, photodiode, MCU and speaker, where the MCU is programmed to read the voltage coming from the node between the resistor and photodiode, which varies with the amount of light incident on the photodiode.This voltage was high when the laser was directly on the photodiode, and low when the laser beam was interrupted. To simulate the interrupts which would be caused by the user blocking the light with their finger, we simply moved the laser off of the photodiode so that less light was incident on the photodiode. The software coded into the MCU then ran a signal to the speaker which caused the speaker to play a note which corresponds to the note that will be played for an open string in our final design. The note played by the speaker was then measured by a guitar tuner app on our phone, to ensure that the frequency programmed into our MCU was the proper frequency for the note we were trying to play. For our first string, we played the note “G”, which is the note that will be played by the bottom string in our final design. The figure below shows a screenshot of the app used to measure the frequency emitted by our speaker, called the “GuitarTuna” app. The guitar tuner returned the note “G”, which confirmed that the noise coming from our speaker was playing at the proper frequency. The sound testing was not as clear as we would like in our final design, however the sound can be improved by using an amplification circuit to improve the signal.

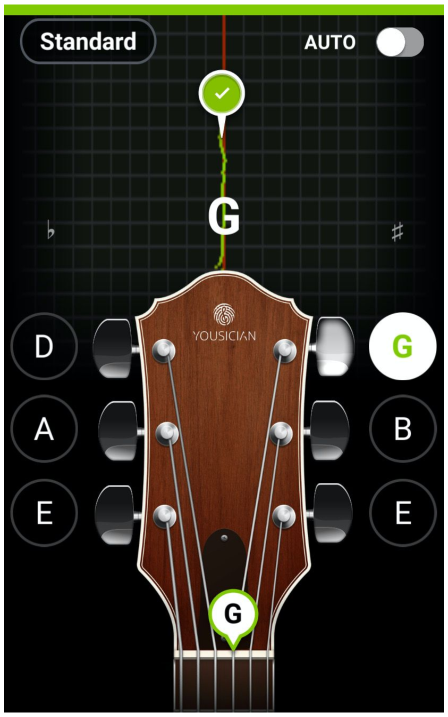


Figure 69: Tuning Our First String

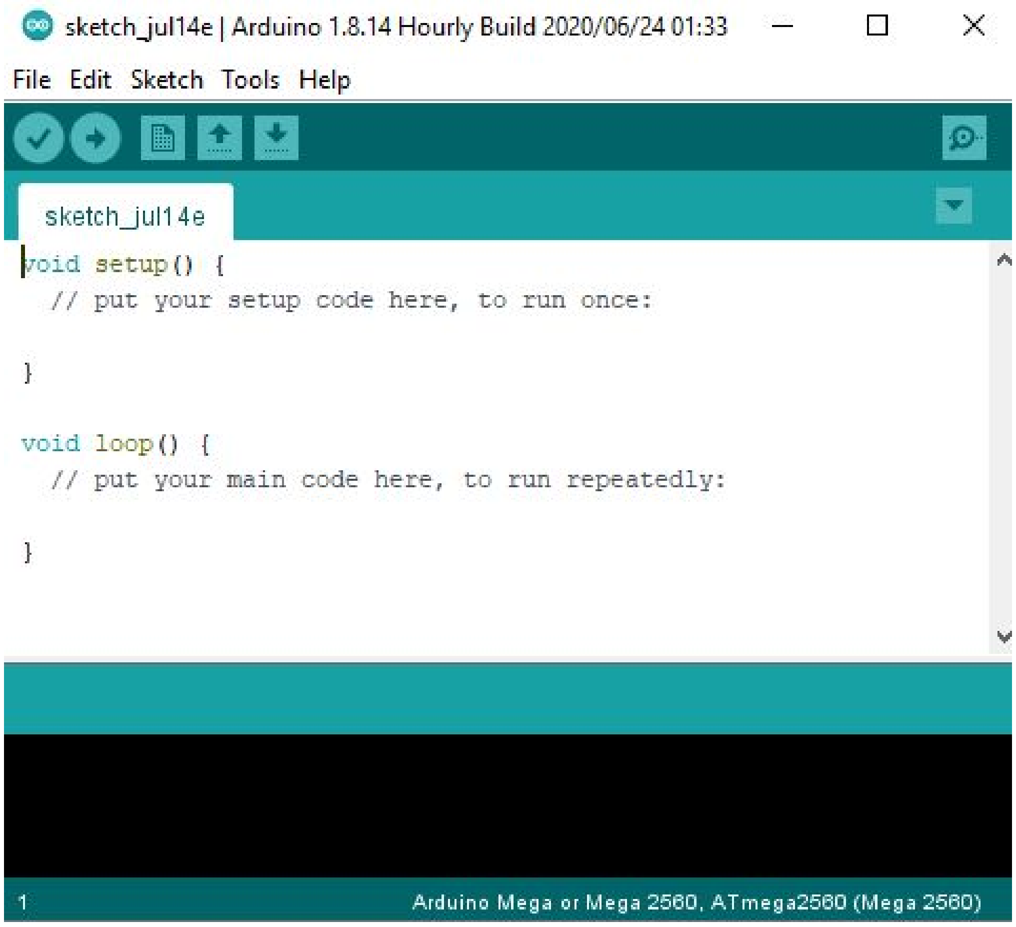
This app helped us ensure that the frequency being played by our speaker is the proper note in terms of the musical scale. In our future testing, we will use this app to ensure that each of the open strings plays sound at the proper frequencies that would be played by a real guitar or ukulele.

<u>Hardware Initial Testing Conclusions and Potential Improvements</u>

The results of our initial testing were reassuringly positive. All of our components worked well under less than ideal conditions, and we gained a lot of valuable information from our tests. Now, we have thought of several ways we can improve our system for future tests. First, we can mount our photodiodes vertically, rather than flat on the fretboard as we initially planned. This should cause the reflected light to hit the photodiode at a better angle resulting in a clearer and stronger signal. If the signal is still not strong enough, we can increase our resistance, but ideally we would like to keep our resistance relatively low. If necessary, we are also presented with the option to use multiple photodiodes in parallel to create a stronger bright current for each individual string. We can also use an amplifier circuit to improve the quality of the signal that is being sent to the speaker for a more clear sound. Finally, we can consider how we house our laser diodes and photodiodes. By designing a housing that decreases the amount of light that leaks from the laser diodes, and blocks light coming from behind and the sides of the photodiodes, we should be able to further improve the performance of our hardware.

7.3. Software Testing Environment

From our selected components, we chose to test our software with Arduino IDE. Figure 59 shows a new file being created.

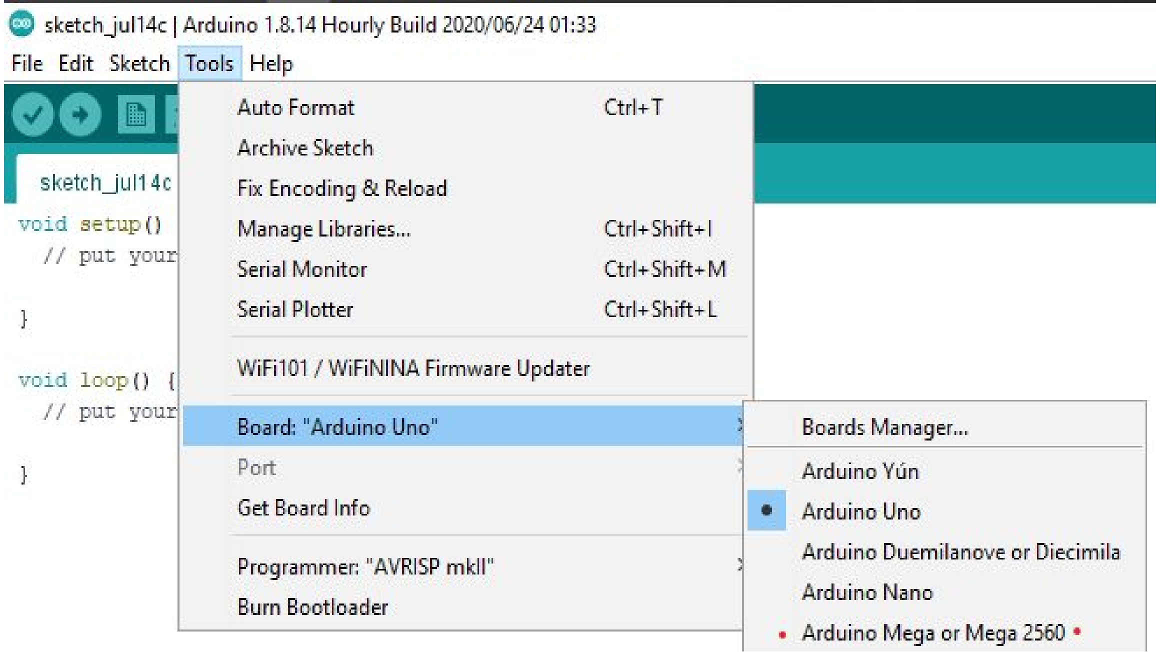

sketch_jul14e | Arduino 1.8.14 Hourly Build 2020/06/24 01:33
File Edit Sketch Tools Help
sketch_jul14e

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
1                    Arduino Mega or Mega 2560, ATmega2560 (Mega 2560)
```

Figure 70 - Arduino IDE

After researching on what Integrated Development Environment we would, we chose to work with Arduino IDE. Figure 70 is the default screen when creating a new file. In order to use our microcontroller, we must connect it to the computer and select the board drivers from within the Integrated Development Environment. Errors that could accrue within Arduino IDE is that it can only support C, C++ and .INO files. Any other file format will cause Arduino to crash. Also another key factor is setting up the drivers correctly for our microcontroller. Most embedded integrated development environments already have drivers for a specific microcontroller while others don't. Since Arduino IDE is developed for Arduino microcontrollers, Arduino IDE should have most and all of the drivers for its own microcontroller. When we first plugged our microcontroller into the computer and loaded Arudino to run test code it wouldn't work. Although the drivers were already preloaded into the Arduino IDE, we need to select it.

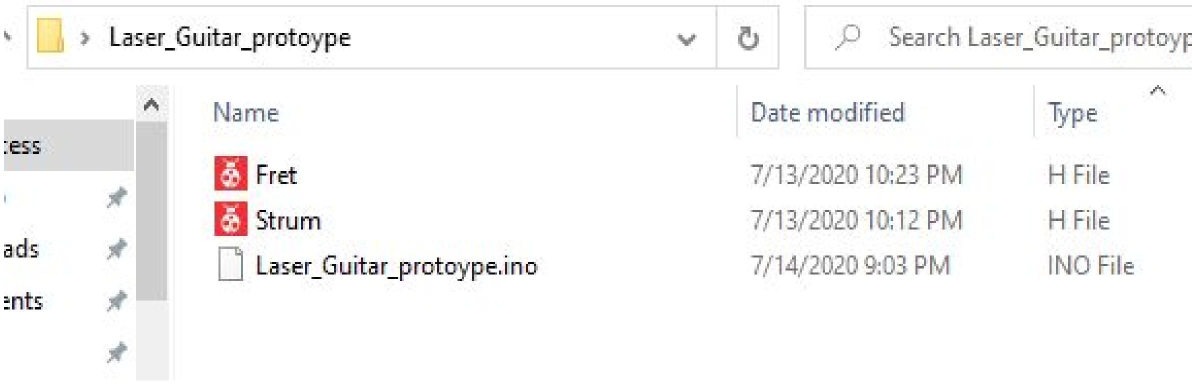

sketch_jul14c | Arduino 1.8.14 Hourly Build 2020/06/24 01:33

```
void setup() {
  // put your

}

void loop() {
  // put your

}
```

File Edit Sketch Tools Help
sketch_jul14c
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Boards Manager...
Arduino Yún
● Arduino Uno
Arduino Duemilanove or Diecimila
Arduino Nano
Arduino Mega or Mega 2560

Figure 71 - Microcontroller Selection

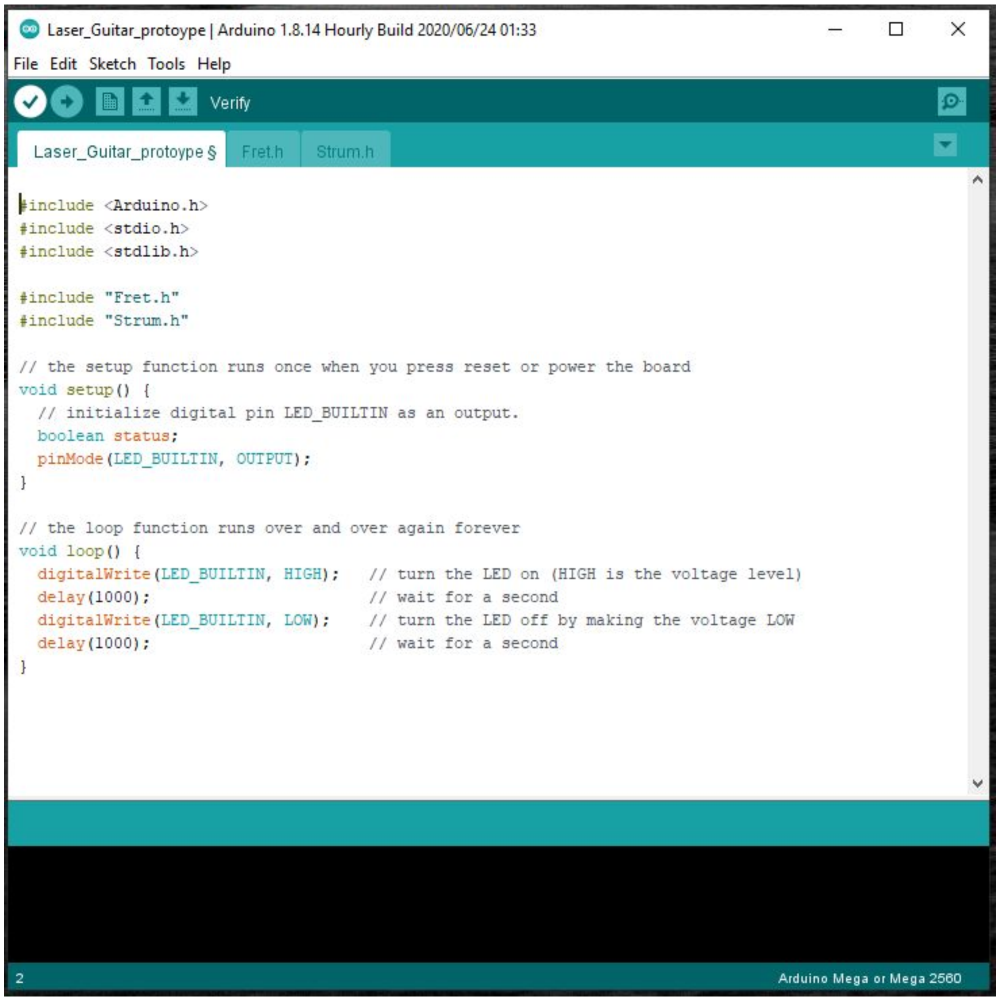
Figure 59 shows how to select the microcontroller with Arduino IDE. Since Arduino IDE is specific for Arduino based microcontrollers, there are various Arduino minctorncolter to select from. The red dots within Figure X shows that our microcontroller is available, and we can select it. By selecting our controller, Arduino IDE will automatically recognize all of the drivers peripherals of our controller when plugged in. This should always be the first step when connecting our microcontroller to our workstation.



Name	Date modified	Type
Fret	7/13/2020 10:23 PM	H File
Strum	7/13/2020 10:12 PM	H File
Laser_Guitar_protoype.ino	7/14/2020 9:03 PM	INO File

Figure 72 - File Folder

Figure 60 shows how many files and the types of files that will be needed for our project. We can see that there is only one .INO file which is the Arduino file. This Arduino file is what the Arduino IDE will compiler and implement the code to our microcontroller. This folder is our ‘sketchbook’ which is Arduino IDE’s project environment. We can have multiple sub folders, and the Arduino IDE would automatically detect these subfolders and create a working directory.


Laser_Guitar_protoype | Arduino 1.8.14 Hourly Build 2020/06/24 01:33
File Edit Sketch Tools Help
Verify
Laser_Guitar_protoype §
Fret.h
Strum.h
2
Arduino Mega or Mega 2560

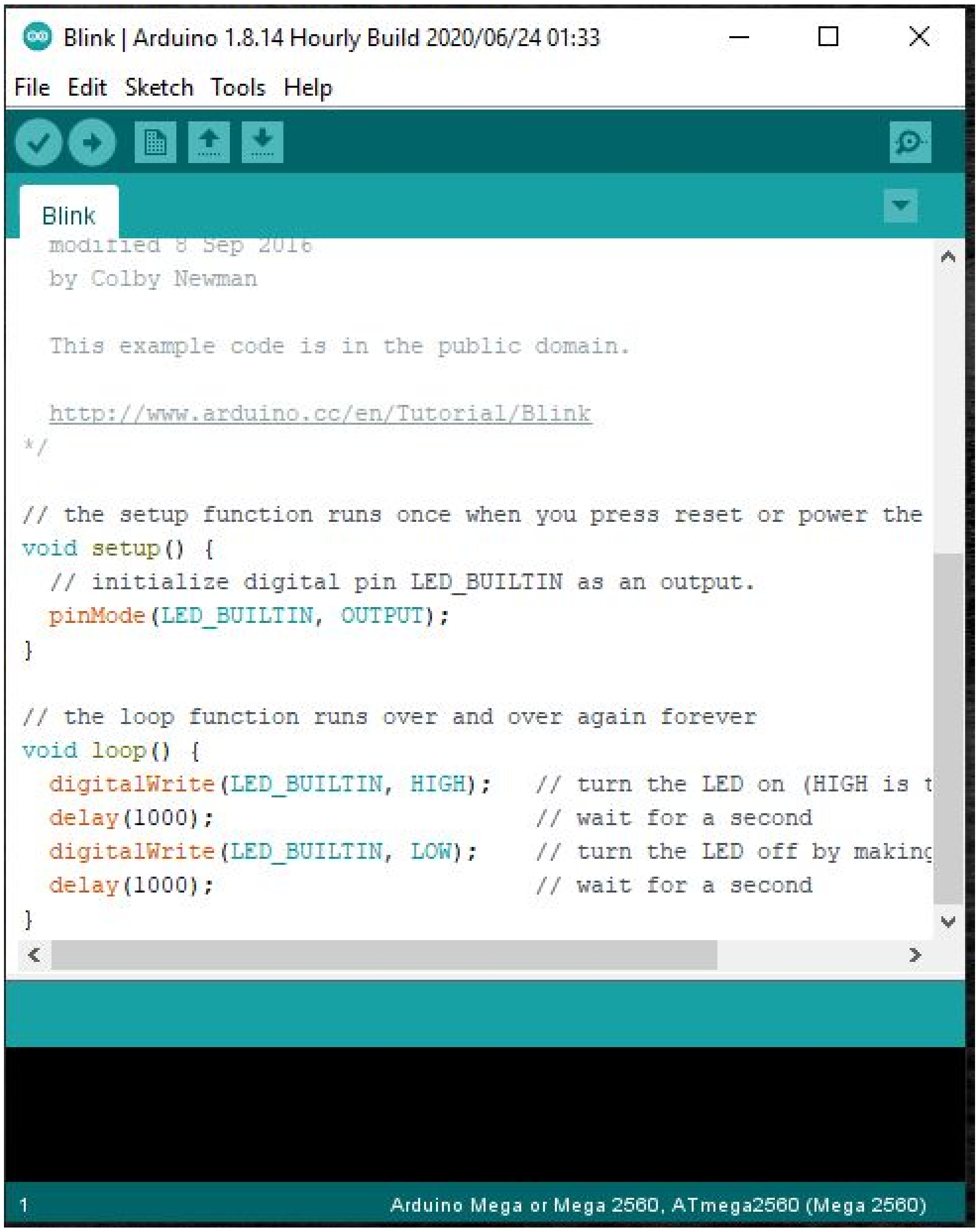
```
#include <Arduino.h>
#include <stdio.h>
#include <stdlib.h>

#include "Fret.h"
#include "Strum.h"

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  boolean status;
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Figure 73 - Project Sketchbook

Figure X shows how Arduino IDE will import the sketchbook. Arduino IDE has all of the files open within tabs that we can easily work from. As we change one or more files, Arduino IDE will automatically save these changes. Within the Arduino IDE we can compile the code and implement it to our microcontroller to see these changes live.

7.4. Software Specific Testing

Now that we have selected and understand the basic operations of our software environment. We can begin doing various types of tests for these basic operations. Figure 63 shows this.


Blink | Arduino 1.8.14 Hourly Build 2020/06/24 01:33
File Edit Sketch Tools Help
Blink

```
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is t
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by makinc
  delay(1000);                       // wait for a second
}
```

Figure 74 - Blink Example Code

Although Arduino IDE has the drivers and peripherals codec need to recognize our minctroller, we need to test whether or not the code that we will write will be applied to it. Arduino has example code that we can run on our microcontroller to see if the code was applied to it or not. The example code we used to test if the code applied to the microcontroller is called Blink. The objective of this code is to make the LED on the microcontroller blink at a specific interval, one second on and off.

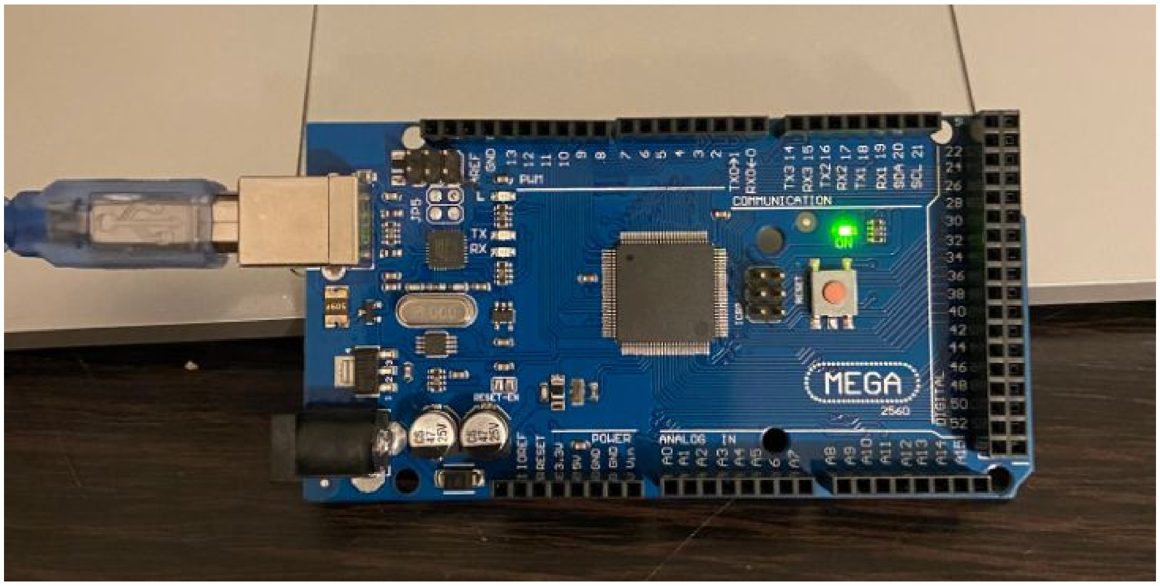


Figure 75 - Blink Example on MCU

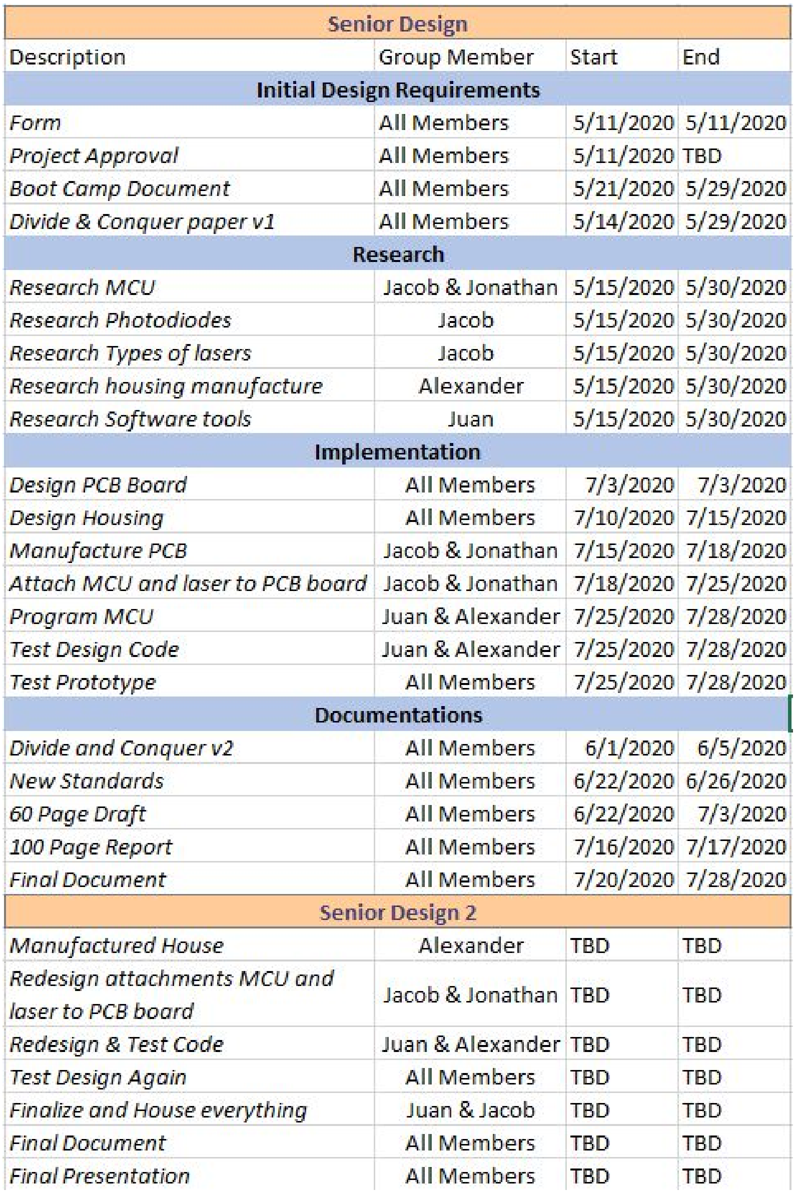
Figure 63 shows the result of when programming the Blink code example to our microcontroller. The LED is one for one second and off for one second .Now that we have an initial test of our Integrated Development Environment and the result of our microcontroller program. We can begin implementing our algorithm. There are several testing that we need to continue and improve on. That will be taken care of in Senior Design 2 when we have our final project built.

8. Administrative Content

The Administrative Content consists of milestone dates and budget outline. These milestones and budgets are subject to change based on time, resource, and money constraints. For the time being, this section has the following information.

8.1. Milestones

Our milestones can be seen below in Table 18. This Table is our projected outlook on what needs to be accomplished within a time frame.



Senior Design			
Description	Group Member	Start	End
Initial Design Requirements			
Form	All Members	5/11/2020	5/11/2020
Project Approval	All Members	5/11/2020	TBD
Boot Camp Document	All Members	5/21/2020	5/29/2020
Divide & Conquer paper v1	All Members	5/14/2020	5/29/2020
Research			
Research MCU	Jacob & Jonathan	5/15/2020	5/30/2020
Research Photodiodes	Jacob	5/15/2020	5/30/2020
Research Types of lasers	Jacob	5/15/2020	5/30/2020
Research housing manufacture	Alexander	5/15/2020	5/30/2020
Research Software tools	Juan	5/15/2020	5/30/2020
Implementation			
Design PCB Board	All Members	7/3/2020	7/3/2020
Design Housing	All Members	7/10/2020	7/15/2020
Manufacture PCB	Jacob & Jonathan	7/15/2020	7/18/2020
Attach MCU and laser to PCB board	Jacob & Jonathan	7/18/2020	7/25/2020
Program MCU	Juan & Alexander	7/25/2020	7/28/2020
Test Design Code	Juan & Alexander	7/25/2020	7/28/2020
Test Prototype	All Members	7/25/2020	7/28/2020
Documentations			
Divide and Conquer v2	All Members	6/1/2020	6/5/2020
New Standards	All Members	6/22/2020	6/26/2020
60 Page Draft	All Members	6/22/2020	7/3/2020
100 Page Report	All Members	7/16/2020	7/17/2020
Final Document	All Members	7/20/2020	7/28/2020
Senior Design 2			
Manufactured House	Alexander	TBD	TBD
Redesign attachments MCU and laser to PCB board	Jacob & Jonathan	TBD	TBD
Redesign & Test Code	Juan & Alexander	TBD	TBD
Test Design Again	All Members	TBD	TBD
Finalize and House everything	Juan & Jacob	TBD	TBD
Final Document	All Members	TBD	TBD
Final Presentation	All Members	TBD	TBD

Table 20: Project Milestones

Table 18 shows the expected initial begin dates and end dates for the milestones we currently have. The first half represents the Senior Design 1 milestones and the second half are the milestones for Senior Design 2. Everything currently listed is expected, but can change depending on how progress is moving online through the semester.

8.2. Division of Work

Not only does Table 18 show the planning of work in a time frame. In Table 19 shows the division of work between group members.

Task	Primary	Secondary
Code Planing and Design	Alex, Juan	Jacob
Photodetection and Laser System	Jacob	Jonathan
Power Supply and Voltage Regulation	Jonathan	Jacob
Optimizing Housing Design	Jacob	Jonathan
Sending/Recieving Signals to/from MCU	Jacob, Jonathan	Alex, Juan
Sending Signals to Speaker, Playing Sound	Alex, Juan	Jacob, Jonathan
PCB Design	Jonathan	Jacob, Juan
Test Sound Production	Jacob, Jonathan, Juan	Alex
PCB Manufacturer Selection	Jonathan	Jacob
Optimize Eye Safety Standards	Jacob	Jonathan
Code Test Cases	Juan	Alex

Table 21: Division of Work.

Table 19 shows the task of our project for each member within the group. There are Primary and Secondary individuals for each task. The Primary column consists of members who will primarily focus on the task. If there were any complications in terms of technical, personal or any types of conflicts that hinder the progression of the project. There is a Secondary person to assist. Individuals who are a Secondary will be working on other tasks and objectives of the project, but be on standby to help Primary individuals with the task that the Secondary is assigned to. The assignments were made based on each group member's individual strengths, where our Electrical Engineering and Photonic Science and Engineering majors focused mainly on tasks involving the hardware, and our two Computer Engineering majors focused on the tasks involving more software design.

8.3. Budget and Finance Discussions

The budgeting and fiance for our project can be seen within Table 20. From our research there are constant price changes for certain components. Due to COVID-19 the rarity of some components are hard to both find and buy because some of these manufacturers who make the components that we're looking for are closed. As an alternative to our part selection and budgeting of these components, we researched on what's currently available to us. Table 20 shows the current market price of our components.

Items	Quantity	Cost
400nm Laser Diodes	2	~$30
500nm Laser Diodes	2	~$30
600nm Laser Diodes	2	~$30
Photodiodes	9	~$50
400nm Bandpass Filter	1	~$50
500nm Bandpass Filter	1	~$50
600nm Bandpass Filter	1	~$50
MCU	1	~$50
Instrument Housing	1	N/A
PCB	1 or 2	~$250
Speakers	3	~$30
Miscellaneous Electronic Components	N/A	N/A
Approximate Total Cost		~$620

Table 22: Projected Budget

For the project there will not be any type of outside funding or sponsors from any parties. Instead, the project will be funded by the members of the group. Figure 4 below, shows the breakdown of what items we need, the quantity amount, and the cost. These items are subject to change based on design, safety, and implementation constraints that we may or may not encounter.

Items	Quantity	Cost
Geebat Mini 650nm Laser Diode	30	$17.94
Vishay BPW34 Photodiode	30	$23.70
MakerHawk Arduino Speaker	4	$19.96
ATmega2560 Microcontroller	1	$15.99
Martin Smith UK-222-A Ukulele	1	$22.00
Total Cost as of 7/27/2020		$99.59

Table 23: Current Cost as of 7/27/2020

The chart above shows the parts ordered for our projects initial testing, along with the quantity and total price of each of the components. A few things have changed from our initial project design, which brought our cost down below what we initially intended on spending. While we still need to order some costly components, most notably our PCB, we are comfortably below budget to where we do not need to worry about spending more than we initially intended. One of the main factors in reducing the cost of our project was the decision not to vary the wavelength of each of the strings. This decision was made because of the high cost of narrow band optical filters, as well as the high cost of laser diodes outside of the 600nm range, and photodiodes with high sensitivities for wavelengths in the visible spectrum below 600nm.

9. Appendices

Section 9 Appendices contains all evidence of copyrighted permissions, works cited and references.

9.1 Works Cited

[1] Chawla, Mohit. “Levels of Programming Languages.” Medium, The BitTheories, 18 Dec. 2017,
thebittheories.com/levels-of-programming-languages-b6a38a68c0f2

[2] Unknown, Author “Jacob’s Blog.” Bingo Embedded Group, Beningo Embedded Group, 18 Feb. 2018,
www.beningo.com/5-reasons-to-start-using-c-over-c

[3] Lokesh, Gupta. “Java Tutorial.” HowToDoInJava, 27 June 2020,
howtodoinjava.com/java/basics/java-tutorial

[4] Staff, Embedded. “Embedded Java.” Embedded.Com, 1 Mar. 2002,
www.embedded.com/embedded-java

[5] Unknown Author, What Is Python? Executive Summary.” Python.Org,
www.python.org/doc/essays/blurb

[6] Guindon, Christopher. “Eclipse Desktop & Web IDEs | The Eclipse Foundation.” Eclipse,
www.eclipse.org/ide

[7] “9 Reasons Why The Vivado Design Suite Accelerates Design Productivity.” Xilinx,
www.xilinx.com/publications/prod_mktg/vivado/Vivado_9_Reasons_Backgrounder.pdf

[8] Unknown, Author. “Virtualization Technology & Virtual Machine Software: What Is Virtualization?” VMware,
www.vmware.com/solutions/virtualization.html.

[9] Editor. “Hardware Abstraction Layer (HAL).” Network Encyclopedia, 29 Aug. 2019,
networkencyclopedia.com/hardware-abstraction-layer-hal

[10] Crowl, L. A., and T. J. LeBlanc. “Control Abstraction in Parallel Programming Languages - IEEE Conference Publication.” IEEE, 22 July 1992, ieeexplore.ieee.org/document/185467/metrics#metrics.

[11] Unknown, Author. “Paradigms.” Programming Paradigms, cs.lmu.edu/%7Eray/notes/paradigms.

[12] Unknown, Author. “Design Patterns and Refactoring.” Source Making, sourcemaking.com/design_patterns.

[13] Unknown, Author. “Time and Space Complexity Tutorials & Notes | Basic Programming.” HackerEarth, 30 Aug. 2016, www.hackerearth.com/practice/basic-programming/complexity-analysis/time-and-space-complexity/tutorial/#:%7E:text=Time%20complexity%20of%20an%20alg.

[14] Drowell, Eric. “Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell.” Know Thy Complexities!, www.bigocheatsheet.com.

[15] Ravi, et al. “8051 Microcontroller Assembly Language Programming.” *Electronics Hub*, Ravi, 25 Dec. 2017, www.electronicshub.org/8051-microcontroller-assembly-language-programming/

[16] Barr, Michael. *Embedded C Coding Standard*. Barr Group, 2018. https://barrgroup.com/embedded-systems/books/embedded-c-coding-standard

[17] Atmel Corporation. *Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V*. 1600 Technology Drive, San Jose, CA 95110 USA T, Atmel Corporation, 2014, ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

[18] Arduino.cc. Creative Commons, 17 June 2020. content.arduino.cc/assets/Pinout-Mega2560rev3_latest.pdf.

[19] Arduino. “Arduino Mega 2560 Rev3 | Arduino Official Store.” *Store.Arduino*, store.arduino.cc/arduino-mega-2560-rev3.

[20] George, Damien. *MicroPython*, George Robotics Limited, 2018, micropython.org/.

[21] Free Software Foundation, Inc. *GNU Coding Standards*. 2008, www.gnu.org/prep/standards/html_node/GNU-Free-Documentation-License.html.

[22] Pal, Sayan Kumar. “Coding Standards and Guidelines.” *GeeksforGeeks*, 2 July 2019, www.geeksforgeeks.org/coding-standards-and-guidelines/.

[23] Van Zegbroek, B. (2011). Principles of Semiconductor Devices. Retrieved June 03, 2020, from http://ecee.colorado.edu/~bart/book/book/chapter4/ch4_7.htm

[D1] https://www.ti.com/product/TPS563231

[D2] https://www.ti.com/product/TPS563249

[D3] https://www.ti.com/product/TLV62130

[D4] http://beambow.com/

[D5]https://www.researchgate.net/figure/Lead-acid-battery-chemistry-a-during-discharging-b-during-charging-and-c-LA_fig4_311305861

[D6] “UC San Diego Works to Build Batteries of the Future.” *Accelerating Microscopy*, 12 Apr. 2019, www.thermofisher.com/blog/microscopy/uc-san-diego-works-to-build-batteries-of-the-future/.

[D7] Jihad Tarabay, and Nabil Karami. “Nickel Metal Hydride Battery: Structure, Chemical Reaction, and Circuit Model.” *Undefined*, 2015, www.semanticscholar.org/paper/Nickel-Metal-Hydride-battery%3A-Structure%2C-chemical-Tarabay-Karami/5a915f5bf77684aaa66209969569f3be4592d91b.

[D8] “Serial and Parallel Battery Configurations and Information.” *Batteryuniversity.Com*, 2018, batteryuniversity.com/learn/article/serial_and_parallel_battery_configurations.

[D9] electronics notes. “PTC Thermistor: Positive Temperature Coefficient » Electronics Notes.” *Electronics-Notes.Com*, 2019, www.electronics-notes.com/articles/electronic_components/resistors/thermistor-ptc-positive-temperature-coefficient.php.

[D10]“Multilayer Pool - The Most Comprehensive Lead You To Know PCB Layers.” *www.Wellpcb.Com*, www.wellpcb.com/multilayer-pool.html

[D11] “Analog and Digital.” *Www.Mathsisfun.Com*, www.mathsisfun.com/data/analog-digital.html.

[D12] “Digital to Analog Converters - Analog and Digital Electronics Course.” *Electronics-Course.Com*, electronics-course.com/digital-analog-converter.

[D13] “Pulse Width Modulation - Learn.Sparkfun.Com.” *Learn.Sparkfun.Com*, learn.sparkfun.com/tutorials/pulse-width-modulation/duty-cycle.

[D14]“Successive-Approximation ADC.” *Wikipedia*, 20 May 2020, en.wikipedia.org/wiki/Successive-approximation_ADC.

[D15] All About Circuits. “The Operation and Characteristics of Voltage-Mode R-2R DACs.” *Allaboutcircuits.Com*, 11 Mar. 2019, www.allaboutcircuits.com/technical-articles/voltage-mode-r2r-dacs-operation-and-characteristics/.

[D16] “ANSI Z136 Standards.” *The Laser Institute*, 3 Oct. 2018, www.lia.org/resources/laser-safety-information/laser-safety-standards/ansi-z136-standards.

[D17] “Battery Technologies - Learn.Sparkfun.Com.” *Sparkfun.Com*, 2019, learn.sparkfun.com/tutorials/battery-technologies/all.

[D18] “Common Battery Types.” *Uiuc.Edu*, 2019, butane.chem.uiuc.edu/pshapley/GenChem2/C6/3.html.

[D19] electronics notes. “Li-Ion Battery Advantages / Disadvantages | Lithium Ion | Electronics Notes.” *Electronics-Notes.Com*, 2019, www.electronics-notes.com/articles/electronic_components/battery-technology/li-ion-lithium-ion-advantages-disadvantages.php.

[D20] electronics notes. “Series Voltage Regulator | Series Pass | Electronics Notes.” *Electronics-Notes.Com*, 2020, www.electronics-notes.com/articles/analogue_circuits/power-supply-electronics/linear-psu-series-regulator-circuit.php.

[D21] “Regulated Power Supplies.” *Learnabout-Electronics.Org*, learnabout-electronics.org/PSU/psu21.php.

[D22]“Difference Between Linear Regulator and Switching Regulator | Electronics Basics | ROHM.” *Www.Rohm.Com*, www.rohm.com/electronics-basics/dc-dc-converters/linear-vs-switching-regulators.

[D23] “What Is a Printed Circuit Board (PCB)? - Technical Articles.” *Www.Allaboutcircuits.Com*, www.allaboutcircuits.com/technical-articles/what-is-a-printed-circuit-board-pcb/.

[D24] “What Is a Printed Circuit Board (PCB)?” *Printed Circuits LLC*, www.printedcircuits.com/what-is-a-pcb/.

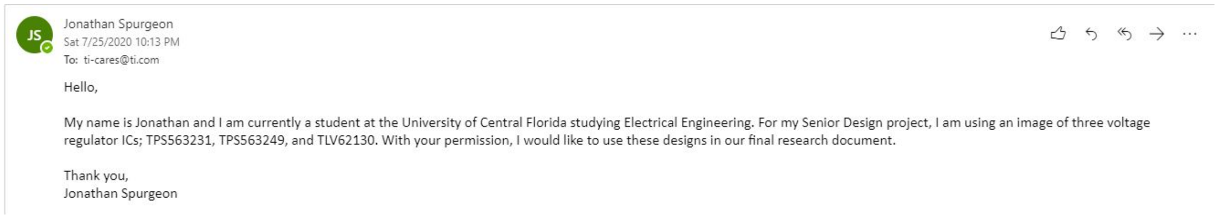
[D25] ”Glossary Definition for Switching Regulator.” *Www.Maximintegrated.Com*, www.maximintegrated.com/en/glossary/definitions.mvp/term/Switching%20Regulator/gpk/298#:~:text=A%20switching%20regulator%20is%20a

[D26] “Boost Converters.” *Learnabout-Electronics.Org*, learnabout-electronics.org/PSU/psu32.php

[D27]“ANSI Z87.1 Eye Protection Standard | Graphic Products.”*Www.Graphicproducts.Com*, ww.graphicproducts.com/articles/ansi-z871-eye-protection/

[D28]“Battery Standards - Lithium, Nickel Metal Hydride, Nickel Cadmium.” *www.Epectec.Com*, www.epectec.com/batteries/battery-standards.html.

9.2 Approvals Cited



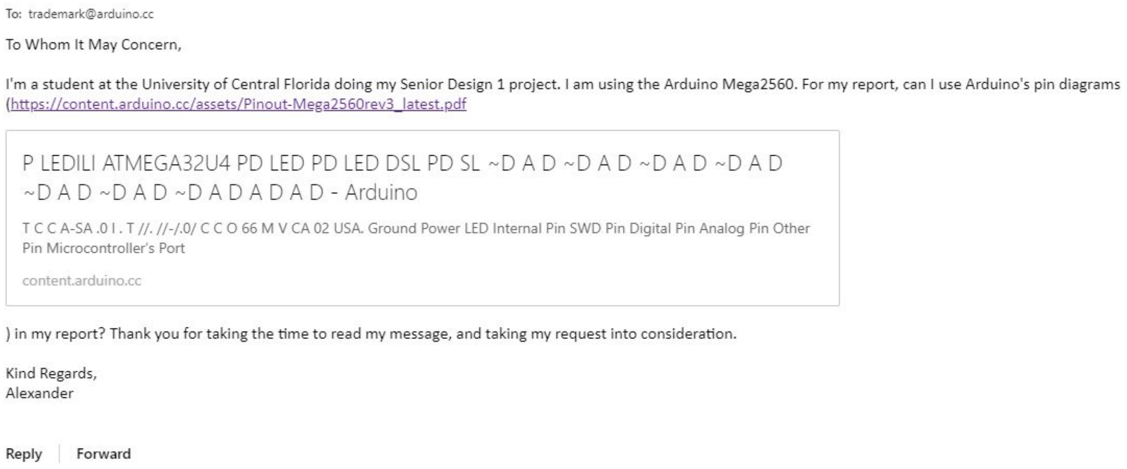
Jonathan Spurgeon
Sat 7/25/2020 10:13 PM
To: ti-cares@ti.com

Hello,

My name is Jonathan and I am currently a student at the University of Central Florida studying Electrical Engineering. For my Senior Design project, I am using an image of three voltage regulator ICs; TPS563231, TPS563249, and TLV62130. With your permission, I would like to use these designs in our final research document.

Thank you,
Jonathan Spurgeon

Texas Instrument Approval



To: trademark@arduino.cc

To Whom It May Concern,

I'm a student at the University of Central Florida doing my Senior Design 1 project. I am using the Arduino Mega2560. For my report, can I use Arduino's pin diagrams (https://content.arduino.cc/assets/Pinout-Mega2560rev3_latest.pdf

P LEDILI ATMEGA32U4 PD LED PD LED DSL PD SL ~D A D ~D A D ~D A D ~D A D ~D A D ~D A D ~D A D A D A D - Arduino

T C C A-SA .0 I . T //. //-/.0/ C C O 66 M V CA 02 USA. Ground Power LED Internal Pin SWD Pin Digital Pin Analog Pin Other Pin Microcontroller's Port

content.arduino.cc

) in my report? Thank you for taking the time to read my message, and taking my request into consideration.

Kind Regards,
Alexander

Reply | Forward

Arduino Approval

9.3 Data Sheets

[17] Atmel Corporation. *Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V*. 1600 Technology Drive, San Jose, CA 95110 USA T, Atmel Corporation, 2014, ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

[18] Arduino.cc. Creative Commons, 17 June 2020. content.arduino.cc/assets/Pinout-Mega2560rev3_latest.pdf.