Affordable Portable MIDI Keyboard Synthesizer

By

David Gutzwiller

Richard Engel

Sujay Murali

Final Report for ECE 445, Senior Design, Spring 2023

TA: Akshat Sanghvi

3 May 2023

Project #54

Abstract

This report serves as a complete overview and in-depth description of Group 54's Affordable Portable MIDI Keyboard Synthesizer project. It provides an overview of our goals and accomplishments, a complex technical breakdown of the system and the subsystems within, verifications and data for each subsystem, a cost analysis of the project, and considers the ethical issues and societal implications that may arise.

Table of Contents

1 Introduction

Right now, the world of music production is incredibly saturated. It seems harder than ever for new musicians to break into this space; not only is new software and instrumentation extremely expensive, but often needlessly complicated, with too many buttons or features. As a result, beginners in this field might find it difficult to get the hang of making professional music and bring their own talent and creativity to the field. That is why we decided to make an economical, small device that is simplistic, but still has enough elements to help someone learn about basic musical production skills.

After a semester of challenges and pitfalls, our group created an affordable MIDI keyboard synthesizer, made for less than $100 in parts and designed for new professionals. *Our first requirement* was to successfully implement buttons for keys, switching octaves, and knobs to change volume, pitch, and ADSR (everything a professional synthesizer needs for a fraction of the price). *Our second* was to use a single lithium-ion battery to keep it powered for over three hours while turned on (so it does not need to constantly be cable-powered), and *our third requirement* was that our synth can be plugged in to a computer to record MIDI notes onto any musical software or DAW. Using thorough tests throughout the building process, we verified and ultimately met all these requirements, and built a user-friendly product.

While browsing on social media circles pertaining to music, we noticed lots of people are looking to break into making music, but frustrated at the current selection of instrumentation available, either because these products are hundreds (or even thousands) of dollars, or are simply not conducive to someone who does not yet have years of experience and knowledge on music theory. Our synthesizer could be just the solution to help them get their feet wet and make some incredible tunes.

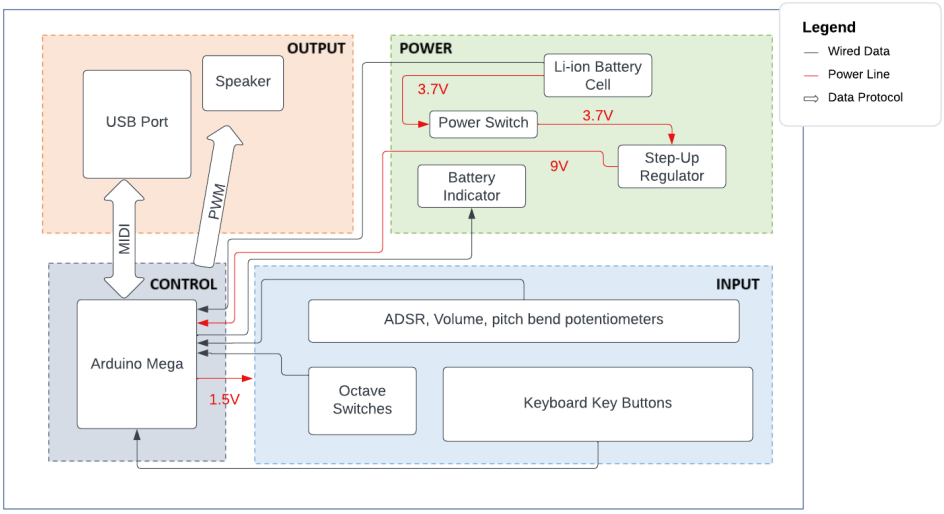

Figure 1: Synthesizer Block Diagram

Above, we have the different components of our design. Our input subsystem is everything that a user interacts with while playing our synth: the keys, the buttons to switch octaves, and the knobs that control aspects of the note. After connecting these to an Arduino, we tested to make sure that each input correctly affects the sound as it is designed to, and that the Arduino successfully outputs the sound to a speaker so that we can hear the correct notes play at a good volume. We aimed for C3 (131 Hz) to B6 (1975 Hz) to be audible, and that full volume yielded a 60 db sound, enough to hear over normal conversation. After verifying this, we used software to help the Arduino output a MIDI signal through its USB port, so one could plug it into a laptop and record notes on a musical software (we tested this on Ableton, a popular DAW, and confirmed that the serial port output of the Arduino was correctly being translated to MIDI). Lastly, the power subsystem allowed the entire device to be powered by a battery, which, after turned on by a switch, supplied voltage to a step-up converter that provides a constant 9+ V to the Arduino. Using a voltmeter, we verified that our power circuit gave our device enough power to last 3+ hours without needing to be plugged in.

2 Design

2.1 Procedure

Overall, we aimed for a simple, streamlined design that allowed for functional user inputs into an Arduino Mega, and played sound out of a speaker. If the Arduino was plugged into a laptop, one could play on our device and record music onto a computer software. Lastly, the entire device must be powered by a battery for at least three hours of on-time. To make our design as compact as possible, we set the inputs and Arduino up on a breadboard, which was desirable as it allowed us to make one connection between all inputs to the Arduino, keeping it simple and close-packed. We initially planned on having all inputs wire through our PCB, but fitting 14 keys and multiple potentiometers through our 10 cm x 10 cm PCB and *then* into the Arduino would have been cumbersome and led to a messier design. Our ultimate idea of moving the power subsystem onto the PCB and laying that underneath our breadboard, resulting in a neat design that was easy to debug if necessary.

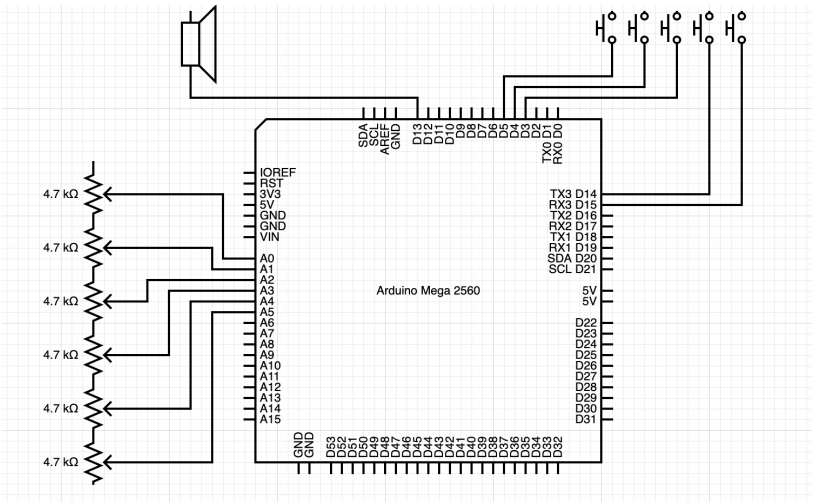

Figure 2: Generalized Layout of Arduino Mega and I/O Pins

Figure 2 shows a generalized design. Buttons connect via a wire to the digital pins of the Arduino, and potentiometers to the analog pins. Digital pin 13 sends an output signal to the

speaker. From here, we could upload software to the Arduino to implement basic piano functions. Initially the design included plans for waveform generation knobs. After developing the software to get ADSR functional, we decided that the waveform inputs are not significantly different from custom ADSR envelopes.

2.2 Details

2.2.1 Power

The power subsystem block was largely integrated into the PCB design. We planned to use a 3.7 Volt Lithium Ion battery to supply power to the microcontroller. However, the Arduino Mega 2560 has a recommended input voltage of between 7V - 12V. Therefore, we also use a step-up voltage regulator in order to boost the voltage from 3.7V to 9.25V, which is within the recommended range for the board. This circuit design also includes a power switch so that the battery can be disconnected when not in use. The Arduino also reads the battery voltage as an analog input and maps its operational voltages to a series of LEDs. These LEDs effectively display the battery life in 10% increments.

Our power subsystem draws power from a 3.7V 2.6Ah Lithium-Ion battery cell. This battery plugs directly into the PCB, with the negative end going to ground and the positive end going to the rocker switch. The rocker switch is simply wired up to the PCB, allowing us the freedom to place the rocker switch wherever we need to in the keyboard's housing. This is also the case with the battery given the lengths of its wires. When the rocker switch is switched on, this connects the battery to our step-up voltage regulator.

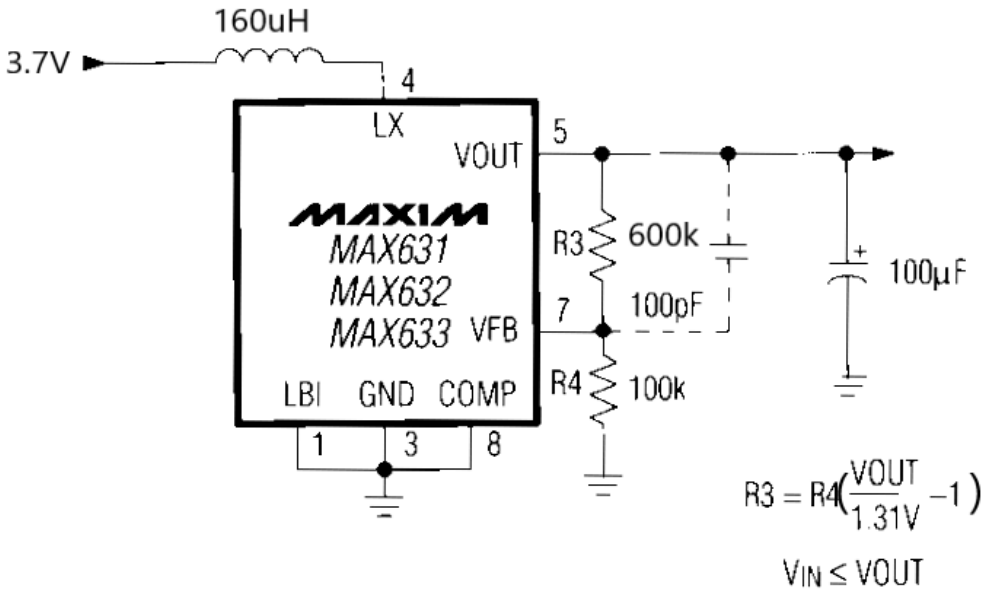

Figure 3: Adjustable Output Configuration and Equation

The step-up voltage regulator that we chose is the MAX632ACPA, which normally increases the input voltage by +12V [5], but also has an option for setting a custom output voltage using the configuration shown in Figure 3, along with the equation being used to find the correct resistor values. If we pick R4 = 100kΩ and we want an output voltage of ~9V, this results in R3 ~= 587kΩ. We can use a 600kΩ resistor for R3 for simplicity, which ends up with a theoretical output voltage of 9.17V (although our actual output voltage was closer to 9.25V). This configuration results in an output voltage that falls comfortably within our required range.

The final piece of our power subsystem is the LED bar graph battery life indicator. This is simply wired up to the Arduino, with each LED being connected to a digital output pin from the Arduino and connected to ground with a resistor. The current battery voltage is read and fed into an analog pin on the Arduino. The Arduino then outputs either on or off values to each LED

depending on the voltage value read from the battery. This LED bar graph displays the battery life of the battery in 10% increments, as shown in Figure 4.

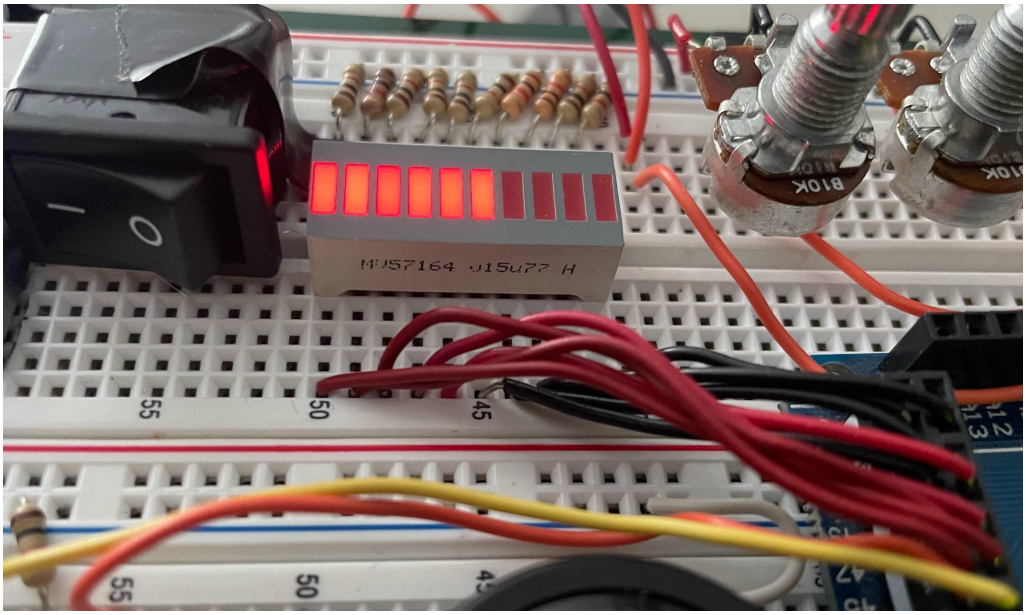

Figure 4: LED Bar Graph Battery Indicator; 60% battery life

2.2.2 Input

Our input subsystem consisted of twelve keys, two octave switches, and six rotary potentiometers. Our keys and octave switches are Adafruit tactile switches, all connected to the digital I/O pins on the Arduino. These switches are seated on a breadboard due to limitations of the PCB size. Twelve of these switches are programmed to cover one octave of the piano keyboard, or 12 semitones, and the other two are programmed to switch between a total range of four octaves. The bottom note of this four octave range is C3, and the top note is B6. In order to get the correct frequencies for these notes, we utilized the standard A-440 Hz tuning [7]. These are produced either on the speaker using the Arduino's PWM functionality, or sent as MIDI messages to a laptop or other device plugged into the Arduino. Both of these functionalities will be described in detail in the next section.

There are also six 10kΩ potentiometers connected to analog pins 0-5. We eventually picked rotary potentiometers with wires soldered rather than breadboard potentiometers due to the additional freedom of placement that they brought. These potentiometers all control a different aspect of the sound output. One potentiometer controls pitch bend, which basically bends the current note being played by up or down a half step. Another potentiometer controls output volume. The remaining four potentiometers control the sound's Attack Sustain Decay Release - or ADSR - envelope. This basically describes the lifetime of a note being played, with the attack referring to how quickly the note rises to its peak, the decay being how quickly it tapers off into the volume determined by sustain, and release being how quickly the note fades into nothing once the key is let go. This can be visually seen in Figure 5.

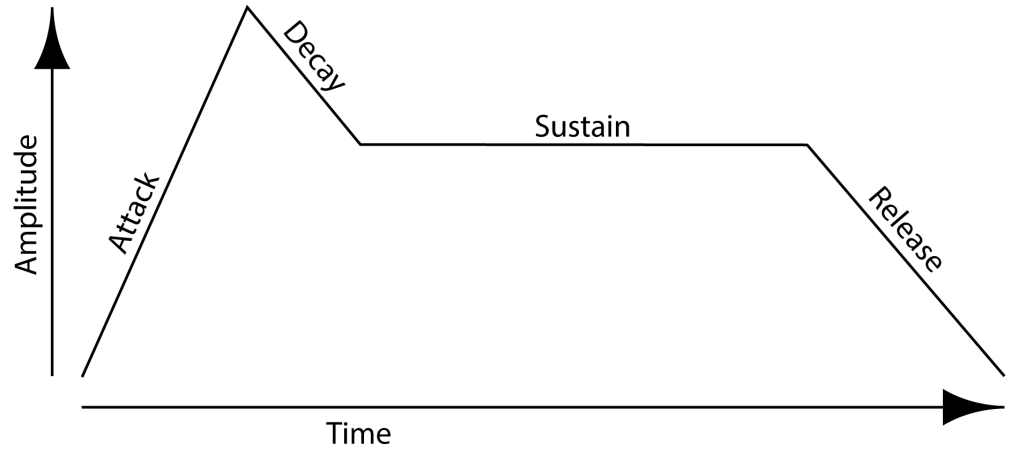

Figure 5: Amplitude vs. Time graph of an ADSR envelope

2.2.3 Control

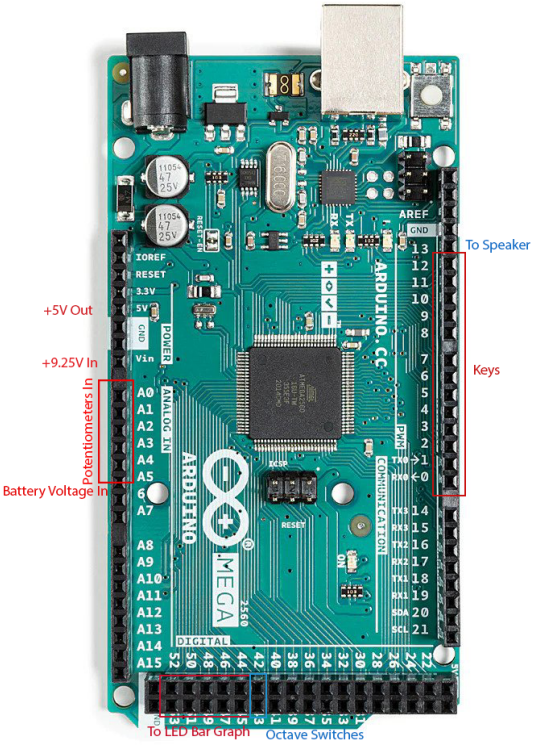

Figure 6: Pin Connection Layout on Arduino

The control subsystem consists of an Arduino Mega 2560 microcontroller board, which has a built-in ATMEGA2560 microcontroller chip, as well as all of the necessary PCB components for functionality. The control subsystem has three main functions. First, it must take inputs from the input subsystem and translate these into modified PWM signals to send to the built-in speaker. Second, it must use these same inputs to generate MIDI messages to send to the built-in USB port so that an external device can connect and use this keyboard with a DAW. The serial port of the Arduino should output a binary number, which is translated to a MIDI message using our software, and a computer should recognize that MIDI message as a musical note [2], recording it onto Ableton, our testing software. Third, it must take the voltage from the battery and use this to

display the current battery life on the LED bar graph. Figure 6 shows how all of these different inputs and outputs are wired to the Arduino.

2.2.4 Output

Our aim for this project was for the keyboard speaker to be at least 60 dB, since that is a little above normal conversational volume. We also wanted the speaker to have a frequency range that covered 150 Hz - 1900 Hz, since that would cover about 4 octaves of range, which is reasonable for a keyboard instrument. The output consists of two main components: the built-in speaker and the Arduino's USB port. The built-in speaker produces the sounds that are generated by the Arduino's PWM functionality as described in the previous section. The speaker can produce these sounds in a range from C3-B6, and it can produce these sounds at over 60dB, which meets our goal. The USB port is programmed to deliver MIDI messages via SPI to an external device to allow the control of a DAW. This functionality is also described in detail in the previous section. Beyond that, that's pretty much all there is to the output subsystem. In future iterations of this design, we would probably include a Digital to Analog converter to allow for more complex sound processing - potentially different instrument sounds and polyphonic functionality - an amplifier to allow for greater volume control, and a stronger speaker with a higher impedance that can work with said amplifier.

3 Verification

3.1 Input Subsystem

The requirements for the input subsystem were to make sure each key successfully plays its corresponding note, the octave switch buttons shift up and down an octave, and the input knobs successfully manipulate the sound as expected. Most of the verifications could be done through listening alone, but there are provided data and measurements to confirm them.

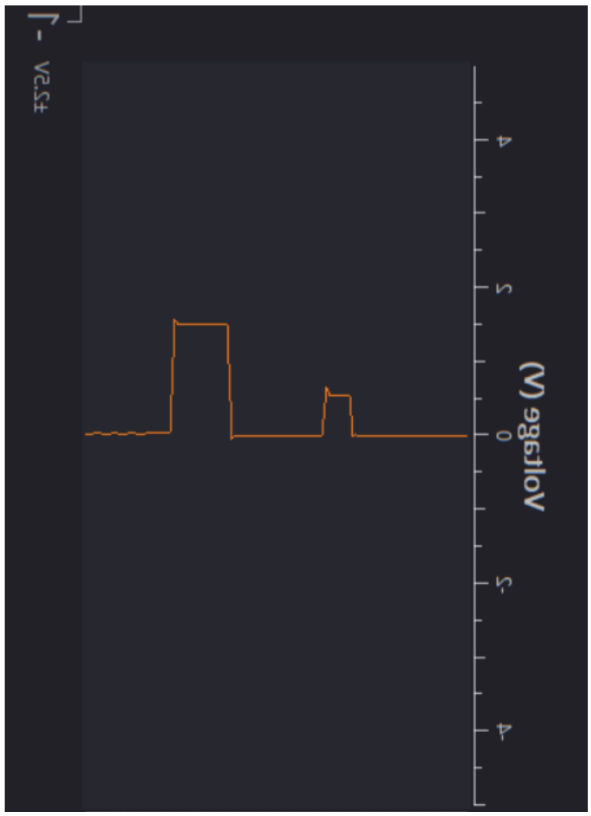

Figure 7: Voltage measured to speaker when volume knob is maximum, half, and zero

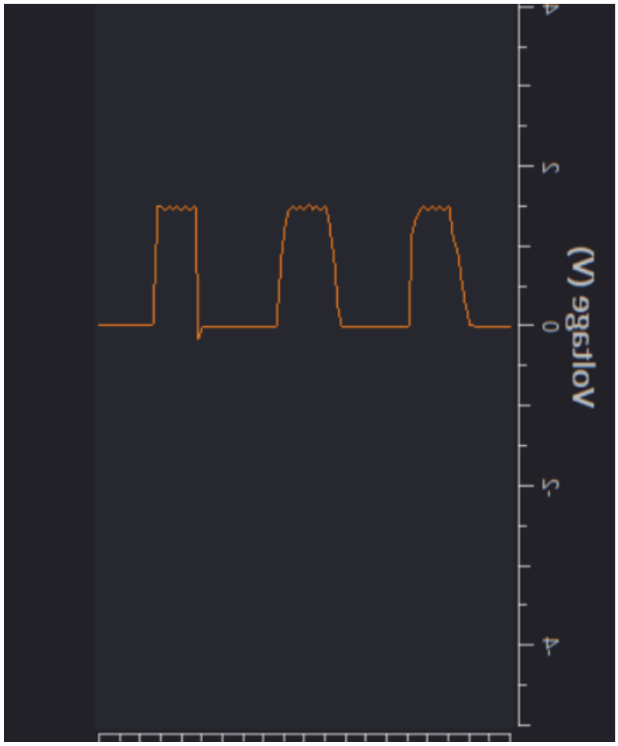

Figure 8: Voltage measured to speaker when the release is zero, half, and maximum value

Figure 7 shows that the volume knob successfully changes the voltage that drives the speaker, which means that it controls the volume of the speaker. Figure 8 displays how the release affects the volume over time, which showcases how ADSR functionality works properly. The falling edge of each voltage spike becomes less steep from the left spike to the right spike. This slanted falling edge shows how the volume level fades out as the release knob is turned.

3.2 Output Subsystem

The two main components responsible for the output subsystem are the speaker, and the arduino board to send MIDI messages. The speaker was verified by using a decibel measurement as well as a tuning measurement. These two measurements verify that the speaker is both loud enough to comfortably hear and also accurate enough to produce the correct notes. Figure 9 shows the decibel reading of the speaker when a note is playing. 63 decibels is similar to casual conversation volume. The frequency of a C3 note that our keyboard produces can be seen in

Figure 10. The measured value is 131 Hz which is 0.19 Hz away from the ideal value of 130.81 as stated by (MTU).

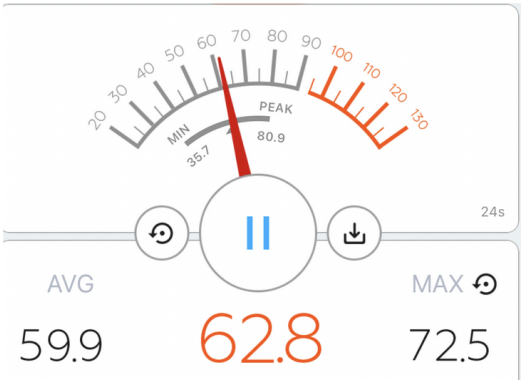

Figure 9: Decibel reading of the speaker at maximum volume

Figure 10: Frequency of a C3 note produced by the speaker

3.3 Microcontroller Subsystem

Our basic requirements for this subsystem was that it can successfully transmit sound to the speaker and MIDI messages through the USB port according to the input keys and knobs, and it does this without issue. After setting up an IAC driver to allow the keyboard to send a MIDI signal to our computer, we used Ableton to test, and successfully recorded musical notes onto a MIDI channel. We could change instruments and loop different tracks onto one another, which was verification that our synth successfully sends MIDI output through its USB port.

3.4 Power Subsystem

We wanted to make sure that the power subsystem both provides a consistent voltage between 7V and 12V to the Arduino board, and that it has a battery life that lasts at least three hours. With the usage of our step-up regulator, we were able to achieve a consistent 9.25 volts being delivered to the Arduino as shown in Figure 11, which is well within our requirement.

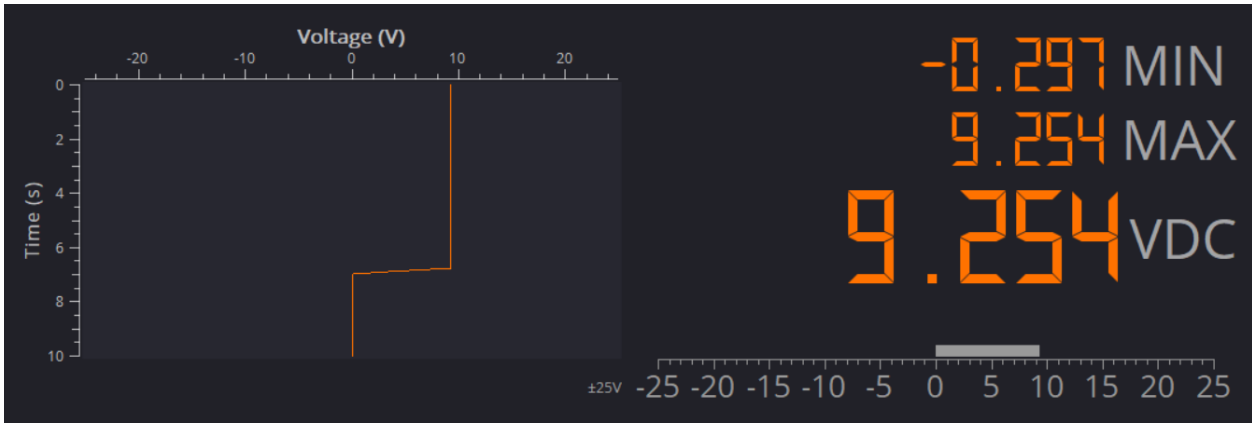

Figure 11: Input voltage measuring at ~9.25V

The power subsystem successfully provided enough voltage to the arduino to power the board for at least 3 hours. The board used 25 different pins in our final implementation. The battery we used was rated at 2700 mAH. If each used pin drew 20 mA, then the battery can last over 5 hours.

4 Costs

So, how did we do in terms of cost? We had three people working on this for about 50 hours total, so if we assume a $40/hr wage, which is about the average computer engineering wage, we can calculate initial labor costs during the design phase.

$40/hr x 50 hrs x 3 engineers x 2.5 = $15,000

So we have an initial design labor cost of about $15,000. The total cost of the components that we used for our project was $87.97, as can be seen in Table 1. Table 2 shows some additional components that we would potentially add in with further development, bringing the total component cost to $102.15. Considering a $20 shipping cost, the total would be $122.15. This results in a total initial design and development cost of ~$15,122.15.

Table 1: Components Cost Table

Description	**Manufacturer**	**Quantity**	**Extended Price**
ARDUINO MEGA2560 ATMEGA2560	Arduino	1	$38.72
SWITCH ROCKER SPST 6A 250V	NKK Switches	1	$3.27
2.6Ah, 3.7V, Li-Ion, 18650, JST	Adafruit	1	$9.49
MAX632ACPA Step-up Voltage Regulator	Maxim Integrated	1	$7.14
10 SEGMENT LED BAR GRAPH - BLUE	SparkFun Electronics	1	$2.25
Rotary Potentiometer - 10kΩ, Linear	SparkFun Electronics	6	$6.30
Micro Round 8Ω Mylar Speaker 93 dB 1.5 Watt 1.5" Wires	Jameco ValuePro	1	$2.35
PCB	PCBWay	1	$0.50
Acrylic Housing (Acrylic Sheet 12’’ x 48’’)	EStreet Plastics		$12
Adafruit Colorful Buttons	Adafruit	15	$5.95
Total Cost			$87.97

Table 2: Additional Components Cost Table

BQ25180 programmable linear charger	Texas Instruments	1	$2.78
PCM5252 32-Bit Stereo Differential-Output DAC	Texas Instruments	1	$8.24
TPA2014D1RGPT Amplifier 1.5 Watt	Texas Instruments	1	$2.39
PJ-102A Barrel Jack	CUI Devices	1	$0.77
Total Cost			$14.18

If this product were to move beyond development into mass production, the cost per keyboard would change. Once the code and design is all developed, it would take maybe 10 minutes to apply solder paste onto each PCB and upload the code onto the Arduino. Using the labor cost calculation ($40/hr x 1/6 hrs * 2.5), this would result in a labor cost of about $16.67 per keyboard. Money can be saved by ordering parts in bulk. If we consider the same cost for the Arduino, PCB, and acrylic housing - $38.72, $0.50, and $12 respectively - and add together the cost of the other components per keyboard unit when ordering them in bulk as in Table 3, this results in a cost of $90.13 for each unit. Considering the labor and $20 shipping, this results in $126.80 for each unit developed.

Table 3: Bulk Components Cost Table

Item	Manufacturer	Max Quantity	Extended Price	Price/Keyboard
Rocker Switch	NKK Switches	1,000	$2,099.79	$2.10
2.6Ah, 3.7V, Li-Ion Battery	US Electronics Inc.	100	$875.00	$8.75
MAX632ACPA Step-up Regulator	Maxim Integrated	1,000	$4,500.00	$4.50
10 Segment LED Bar Graph	SparkFun Electronics	100	$203.00	$2.03
10kΩ Rotary Potentiometers	SparkFun Electronics	100	$95.00	$5.70
8Ω Micro Speaker	Jameco ValuePro	2000	$3300.00	$1.65
Colorful Buttons	Adafruit	1500	$476	$4.76
Linear Charger	Texas Instruments	6000	$7,440.00	$1.24
DAC	Texas Instruments	1000	$6,750.00	$6.75
Amplifier	Texas Instruments	5000	$5,350.00	$1.07
Barrel Jack	CUI Devices	2400	$861.72	$0.36
Total Cost/Keyboard				$38.91

5 Conclusions

5.1 Takeaways

All in all, we were successful in making a device that costs less than $100 in parts and still has working features that a beginner can use to increase their knowledge and experience with music production. We satisfied all of our high level requirements; our keyboard has correctly working inputs, it plays the desired sounds through a speaker and sends MIDI to a computer when plugged in, and our PCB correctly uses a battery to power it for over 3 hours, allowing it to be portable. Using our own equations and creative designs, we configured our synth to be user-friendly and easy for a beginner to learn how simple note manipulation and configuration with DAWs work, all at a very reasonable price point. With our functioning software and a simple, optimized design, the product could easily be mass-produced.

Due to space constraints, limitations with our Arduino, and lack of knowledge on our part, we faced challenges in our design, some of which we could not fix. We could not implement chords or different tones other than a digital PWM sound. A future design could include a programmable linear charger, soldered onto the PCB and attached to a USB port, that would allow the user to plug the synth in to recharge the battery (rather than having to replace the battery entirely). We could use a Digital to Audio converter as an intermediary between the Arduino and the speaker, to allow us to create chords and an analog piano sound. Instead of plugging our device into a computer through the Arduino USB to record MIDI notes, we could connect the Arduino to a MIDI pin and use a MIDI-to-USB cable, reducing the latency between our keys and the software. Lastly, we could also connect the speaker to an operational amplifier to greatly increase our max volume.

5.2 Broader Impacts

In our completion of this project, we have shown that it is possible to create an effective musical device at a cheaper price than what is usually available on the market. Many options out in the market can cost hundreds or even thousands of dollars, with most of that money simply coming from branding. We have shown that it can be possible to create an extremely powerful instrument at an entry-level price point. With further development and refinement, our keyboard could end up creating some serious competition in the music production market for the better. According to 6AM group, a reputable music news source, one of the biggest barriers to entry in the musical industry is financial [6], and we can help turn the tides in the favor of new, talented musicians.

5.3 Ethical and Safety Considerations

We utilized the Harvard Lithium-Ion Battery Safety Guidelines Document [3] to ensure that we were careful to avoid thermal runaway or injury while using our battery. We also used Makerspace: “How to Solder Properly” [4] to ensure we were soldering wires together carefully, and cleaning the soldering tip as necessary. Throughout the process of designing and building our device, we took care to follow the ACM code of ethics, whether that be to build and design robust and usable systems [1], ensuring that our synth had functioning inputs and a user-friendly design, and accepting professional review at all stages, taking constructive criticism [1]. Only with the help of our professors and mentors would we have been able to correctly power the system and achieve our requirements. The experience taught us about the careful considerations of engineering, and steadily working towards our goal.

References

[1] “ACM Code of Ethics and Professional Conduct.” *Code of Ethics*, Association of Computing Machinery, 2018, https://www.acm.org/code-of-ethics.

[2] Amandaghassaei and Instructables, “Send and receive Midi with Arduino,” *Instructables*, 28-Oct-2017. [Online]. Available: https://www.instructables.com/Send-and-Receive-MIDI-with-Arduino/. [Accessed: 03-May-2023].

[3] *Laboratory safety guideline - ehs.harvard.edu*. (n.d.). Retrieved February 23, 2023, from https://www.ehs.harvard.edu/sites/default/files/lab_safety_guideline_lithium_ion_batteries.pdf.

[4] Makerspaces.com, “How to solder: A complete beginners guide,” *Makerspaces.com*, 31-Oct-2021. [Online]. Available: https://www.makerspaces.com/how-to-solder/. [Accessed: 03-May-2023].

[5] Maxim, “CMOS Fixed/Adjustable Output Step-Up Switching Regulators,” MAX631/632/633 Datasheet.

[6] S. Schossberger, “Barriers to entry: Electronic music's hidden gatekeepers,” *6AM*, 24-Nov-2021. [Online]. Available: https://www.6amgroup.com/barriers-to-entry-electronic-musics-hidden-gatekeepers/. [Accessed: 03-May-2023].

[7] *Tuning*. Frequencies of Musical Notes, A4 = 440 Hz. (n.d.). Retrieved March 24, 2023, from https://pages.mtu.edu/~suits/notefreqs.html

Appendix A Requirement and Verification Tables

Table 4: Power Subsystem Requirements and Verification

Requirements	Verifications
• The power system successfully supplies between 7V-9V of power to the microcontroller	• Ensure that when the battery is not powered on that the Vin to the microcontroller is 0 Volts • Ensure that when the battery is turned on the microcontroller Vin is greater than or equal to 9 Volts
• The battery life can last at least 3 hours from a full charge	• Measure a 9 Volts or greater as input to the microcontroller at any moment within 3 hours of turning the battery on
• LED bar graph can display battery life in 10% increments	• Monitor the LED bar graph as the battery drains

Table 5: Input Subsystem Requirements and Verification

Requirements	Verifications
• Each piano key should send a distinct signal (12 total signals) to the Microcontroller Subsystem	• Check using the Arduino that each signal is received and distinct
• Octave changers and pitch bend wheel should change both the synthesized audio and the MIDI output by one octave or one whole tone either up or down respectively.	• Verify functionality on the synthesized notes through examination of the speaker audio • Verify proper MIDI functionality through examination of the MIDI messages output from the microcontroller

Table 6: Microcontroller Subsystem Requirements and Verification

Requirements	Verifications
● The microcontroller successfully performs digital audio synthesis and outputs the data in SPI form	● Sensor data is processed and converted to SPI by using an oscilloscope with a high-impedance multimeter. We check the SPI bus signals on the oscilloscope for different input signals to ensure that the differences are accurately reflected.
● The microcontroller successfully generates MIDI messages based on the input	● Connect the microcontroller to a Windows computer via USB. Using Ableton, see if MIDI messages are being translated through the microcontroller onto the computer

Table 7: Output Subsystem Requirements and Verification

Requirements	Verifications
● Speaker can produce sounds between 150Hz and 1900 Hz	● In a largely volume-free room, point the speaker towards microphone plugged into the laptop. Use this to measure frequency response of speaker at lowest and highest setting.
● Speaker can produce sounds up to at least 60 dB at full volume.	● Computer software like TrueRTA or REW can test volume readings of our speaker very accurately, so we will do the same microphone testing with our driver to check the volume