Voice Activated Scorekeeper

ECE 445: Final Report

Team 31: Jason Hwang, Christopher Kalebich, Allan Zou

TA: Enliang Li

Fall 2019

Abstract

Hundreds of millions of people across the world play sports in their free time. According to a 2007 census by FIFA, there are 265 million people playing soccer across the globe [1]. A number which is likely to have grown in the past decade. And that is not counting other sports, like volleyball, basketball, baseball, etc. The majority of these players aren't professionals however. They play for fun in the evenings; casual games of pickup with local community members. These players have no effective way of keeping score, only relying on their memory which often proves unreliable. By developing a portable voice activated scorekeeper, we can help millions of people in the world keep track of the score quickly, easily, and reliably. Players will no longer forget what the score is, less time will be wasted trying to remember the score, and games can be played with more intensity and focus.

Table of Contents

1. Introduction

1.1. Problem & Solution Overview

Pickup games are very casual and quickly organized games with either strangers or a group of friends. Due to its casual nature, there is no referee or scorekeeping system; rather, the players are keeping track of the score using their memory. In fast-paced, or high-scoring games such as basketball or volleyball, this can result in players forgetting the score and having to ask other players what the score is. Not only is this disruptive to the flow of the game, it can also result in errors in scorekeeping.

Our goal is to eliminate this problem by creating a voice activated scorekeeper that will allow players to quickly and easily update and check the current score. Using specific keywords, users will be able to increment and decrement the score for two teams.

1.2. High-level requirements list

- Score correctly changes through keywords spoken by a user with at least 85% accuracy.
- Microphone should pick up audio commands $\leq$ 10 ft from the user with a background noise of 75 $\pm$ 5% dB.
- Battery should be rechargeable, lasting $\geq$ 4 hours.

1.3. Block Diagram

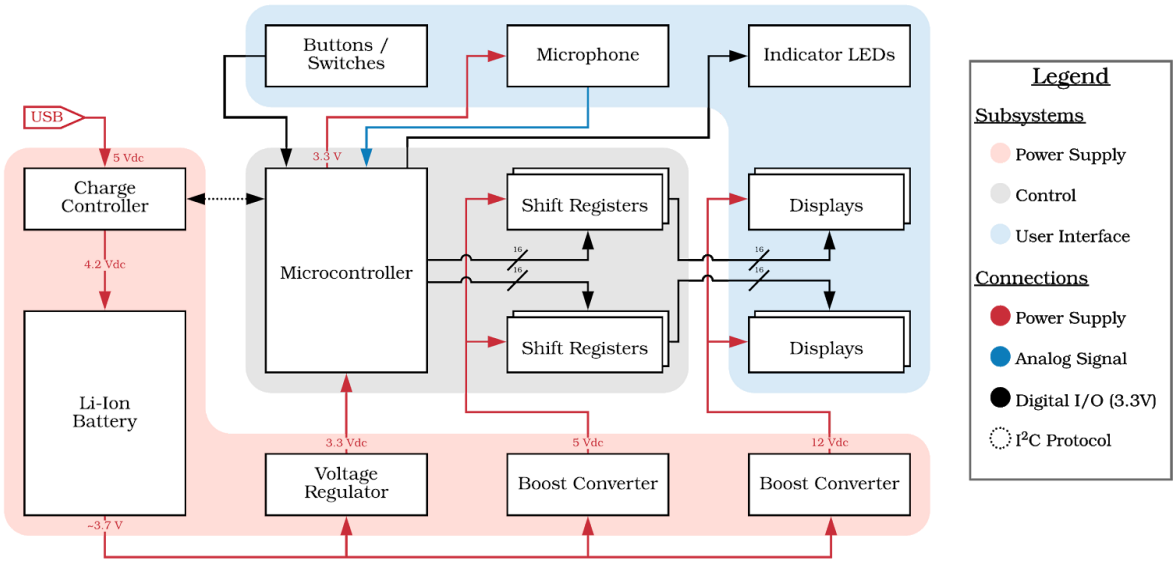


Figure 1. Block Diagram

The block diagram shown in Figure 1 contains three subsystems: power, control, and user interface. The power subsystem allows the battery to be recharged safely and in turn power the rest of the system for at least 4 hrs when supplying 3.3V. The seven-segment displays in the UI output subsystem must be large enough to be seen from the service line 30ft away. Since they will be larger than standard seven segment displays, the system utilizes a boost converter to increase the voltage from the voltage regulator in order to properly power the displays. Finally, the UI input subsystem contains a microphone which allows the system to capture the keywords spoken by a user and pass it to the microcontroller for processing and determining what action to perform. The microphone should still be able to receive user commands in an environment where the background noise is 75 db ± 5%. We arrived at this value by measuring the noise level of a gym while players were playing volleyball and finding the average noise level over a three-minute time frame.

1.3.1. Block-level Changes

In our final design, we had to separate the speech-recognition system from our microcontroller. Initially, our speech-recognition system was a tensorflow model which would reside in the flash memory of the microcontroller. Due to problems with tensorflow model not being able to correctly predict the right keyword despite being given data provided by the tensorflow team itself, we made the decision to switch to a different library. The new library we chose to use, uSpeech, is an Arduino-based library for voice recognition using phonemes. A phoneme is a distinct unit of sound such as ‘sh’, ‘f’, and ‘e’. Since this is an Arduino-based library, we used an Arduino Uno to interface between our microphone and microcontroller. Our final design operated by having uSpeech determine which phoneme is being said and then generating an interrupt on the microcontroller by briefly turning the corresponding teams increment/decrement pin high.

2. Design

2.1. Design Procedure

2.1.1. Microcontroller

The microcontroller, a STM32F401, will handle a variety of tasks. First, it will receive input from either the buttons, or the microphone. If input is received from the microphone, the microcontroller unit (MCU) will transform the analog signal to a digital signal. It then feeds the newly transformed signal into the Keyword Spotting (KWS) model stored in its flash memory. Depending on the keyword detected by the model, the MCU can turn on the indicator light to show users it is listening or update the game score. If the input is received from the buttons, the MCU can immediately update the score based on which button it received a signal from. In order to visually update the score, the MCU will communicate with the seven-segment display using shift registers. Finally, the MCU will also receive input from the battery management IC over I^2C to show users the charge status of the battery.

This microcontroller was chosen for a few reasons. First, it is an ARM Cortex M-4 based MCU. The Cortex M-4 contains a DSP extension to ARM's Thumb instruction set with additional instructions which accelerate DSP algorithms. In addition to faster DSP algorithms, by using an MCU with a Cortex M-4, is able to reduce manufacturing cost, development cost, and system complexity by removing the need for a separate DSP unit. This enables our system to rapidly modify the signal from the microphone so that it can be used as input for the KWS model. Secondly, this MCU has flash memory ranging from 128 to 512 kb which is necessary to store our KWS model. Deployment of a KWS application onto a Cortex M-7 STM32F746G-DISCO development board used ~70kb of memory [2], so we estimate that our application will use around the same amount of memory, while allowing us leeway to increase the size of the model.

Alternatively, other microcontrollers can also be chosen to implement our final design. Since the speech-recognition system in our final design is an Arduino library, it is possible to simply use an ATmega328P as the microcontroller for the project. This would greatly reduce development time and reduce code complexity. This is not recommended however, since the uSpeech library

has a very low accuracy so the final product would be unusable. Using this microcontroller would also limit further improvements to the product and possibly extend development time for future iterations of this product.

2.1.2. 8-Bit Shift Registers

The main purpose of these shift registers is to provide a controllable drain for our seven segment displays. Each bit stored in the registers will represent whether or not a certain segment of the display is on. In addition to acting as a current drain, the shift registers also greatly reduce the number of I/O pins necessary on the microcontroller. Since we have four seven-segment displays, this reduces the required pins from 32 pins to 4 pins. Furthermore, it is possible to daisy-chain these shift registers together. In our case, we daisy-chained two shift registers for each team, so we only used two pins, but it is possible to use only one pin to update the score by daisy-chaining all four shift registers together.

2.1.3. Microphone

The microphone we chose to use is the MAX9814 which comes with automatic gain control. The key reason we chose to use this microphone is for simplicity. The automatic gain control allows us to avoid tweaking the amplifier gain consistently.

There are a vast amount of alternatives that can be used. Microphone polar patterns can have a large influence on how well users are heard. There are three main types of polar patterns for electret microphones: omnidirectional, unidirectional, and noise-cancelling. Omnidirectional microphones are designed to receive sound from any direction; unidirectional microphones receive sound from a single direction; noise-cancelling microphones filter out sound from a certain direction. For our purposes, unidirectional microphones are best since the microphones are placed facing users. We chose not to focus on the polar pattern of the microphone since it is a fine detail that can be refined in future iterations of the product.

2.1.4. Seven-Segment Displays

The seven-segment display will indicate the current scores of two teams, and we chose our specific 6.5in. display because it would be easily visible from a distance.

2.1.5. Charge Controller

The charge controller will supply the battery with a constant 4.2 V supply. More importantly, the controller will monitor the input charge current and stop charging once the current drops below 3% of the battery's rated current.

2.1.6. Battery

When choosing a battery for our project, we decided to choose a lithium-ion battery because they tend to be lighter and smaller than other alternatives such as lead-acid.

2.1.7. 3.3 V Voltage Regulator

The voltage regulator will output 3.3VDC while receiving variable voltage input from the battery.

2.1.8. 5V Boost Converter

The voltage regulator will output 5VDC while receiving variable voltage input from the battery.

2.1.9. 12V Boost Converter

The voltage regulator will output 12VDC while receiving variable voltage input from the battery.

2.1.10. Speech Recognition

A main high-level requirement of our project is to control our scorekeeper using voice commands. Thus, speech recognition was extremely important. We decided to use Tensorflow Lite for its ability to train machine learning models to recognize only the specific keywords that we needed while maintaining a small enough footprint to be deployed on a microcontroller with

limited computation and memory resources. Trained Tensorflow models also demonstrated their ability to train reasonably accurate models sufficient for our needs.

2.1.11. Speech Recognition State Control

While using Tensorflow Lite to recognize the specific spoken keywords, we will use the state control to keep track of the spoken keywords and update the correct teams score accordingly once a full loop has been made.

2.1.12. Hardware/Software Integration

In order to make sure that all of our subsystems worked correctly together, we had to program our microcontroller using Mbed and C++. While using Mbed enabled functions, we were able to access low level capabilities without needing to explicitly set up specific settings such as clock frequency, registers, and others.

2.2. Design Details

2.2.1. Microcontroller (Schematic: Appendix A.1.)

To design the surrounding circuit for our STM32F4 microcontroller, we referenced its datasheet, [3]. The primary design challenges were choosing how to properly design its power supply and how to choose the correct General Purpose In/Out (GPIO) pins.

Designing the power supply scheme was as easy as reading and following the suggested method found in [3]. This included placing decoupling capacitors between the IC's voltage pins. Pins labeled Vdd were given the input 3.3V, while pins labeled Vss were connected to electrical ground. On our version of the STM32F4, there were four pairs of Vdd/Vss pins; each pair required a 100nF capacitor, and one pair, the main voltage reference, required at least a 4.7µF capacitor.

Additionally, we referenced the datasheet to determine which pins supported our needs. Specifically, we needed to locate those capable of analog-to-digital conversion (ADC), as we planned on processing the digital sample of the analog microphone's output. In the end, we found and used pins 8 and 9 for the signals ADC. The rest of the pins are simply GPIOs and can be used for normal control. The resulting pinout and schematic can be found in Appendix A.1.

2.2.2. 8-Bit Shift Registers (Schematic: Appendix A.2.)

The shift registers, Texas Instruments TPIC6C596, were chosen because each pin was capable of sinking more current than the MCU's pins, 250mA compared to 25mA. This was important because each segment of our seven-segment displays was rated to draw 140mA.

The resulting circuit became very straightforward to design after referencing its datasheet, [4], as the chip simply needed to be supplied 5V (pin 1) and grounded (pin 16). In addition, because the IC uses a serial-in and parallel out, we needed to supply signals from the microcontroller to control how the input data would shift along through the registers. To do this, we connected SRCK (pin 15) and SRCK (10) to the MCU. SRCK stands for shift register clock and it controls

how the data in, SERIN (pin 2), flows through the buffer registers. Once the desired data has been loaded, the RCK signal needs to toggle to shift the contents of the buffer registers to the main registers that control the output. Lastly, because the four displays will be paired to create two 2 digit numbers, we daisy-chained two shift registers together to create a large 16-bit register that controls two digits. In all, we used two pairs of shift registers, four in total.

2.2.3. Charge Controller (Schematic: Appendix A.5.)

The charge controlling IC that we used to charge our battery was the Texas Instruments BQ25896. To design the circuit, we referenced the part's datasheet [5] and followed the recommended circuit. The particular charge-sensing behavior of the controller was already embedded in the IC and was able to be later tweaked by communicating with a microcontroller over I2C protocol. As seen in Appendix B.5., the signals that communicate with the MCU needed to be pulled up to the main system voltage through a 10kΩ resistor. This is done to ensure that high logic satisfies the MCU's logic levels and does not float at an unspecified value.

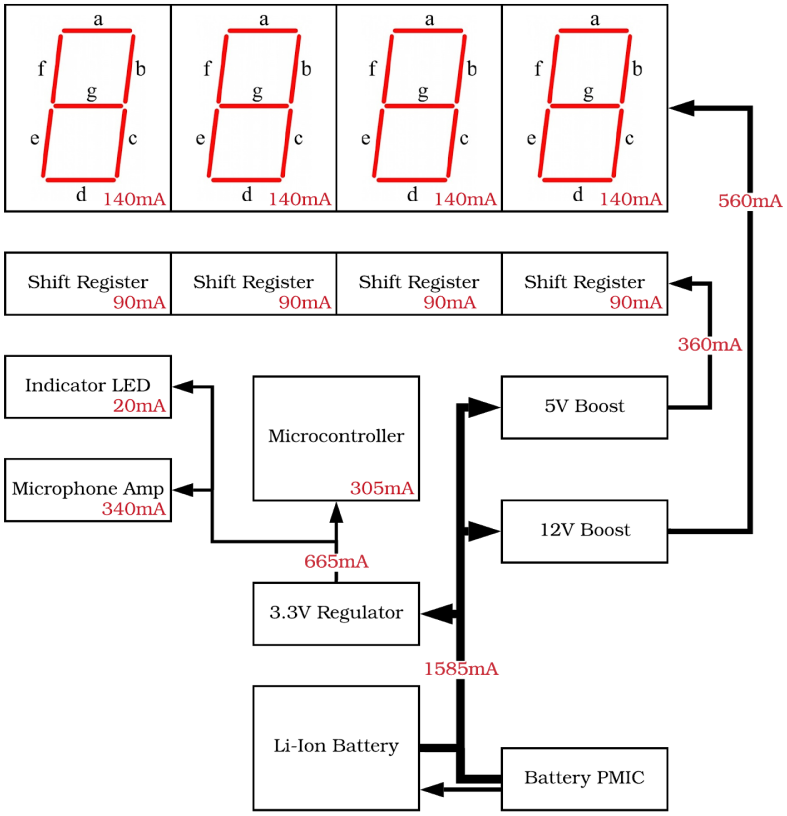


Figure 2: Maximum Current Draw Approximation

2.2.4. Battery

While we did not directly design a battery, we had to choose a battery that was able to provide enough current to last about 4 hours, as specified in our high level requirements. To do this we referenced the datasheets of all of our components and found the maximum current draw for each component. The resulting totals can be seen in Figure 3; these totals allowed us to choose and properly spec the battery in addition to the voltage converters.

As seen in Figure 3, we approximated the total current draw to be about 1585mA. Additionally, the common practice for discharging lithium-ion batteries is to withdraw current at a rate 0.2-0.25 times the battery's C-value, as we learned from Battery University's lesson BU-409: Charging Lithium-Ion Batteries [6]. The C-value of a battery is how many mAh the battery contains. Therefore, we calculated the C-value's upper and lower bounds as

$$Upper\ C = \frac{1585mAh}{0.2} = 7925mAh \quad (1)$$

$$Lower\ C = \frac{1585mAh}{0.25} = 6340mAh \quad (2)$$

Consequently, we chose a battery with a C-value of 6600mAh, resulting in a maximum discharge rate of (1585mAh / 6600mAh) = 0.240C.

2.2.5. 3.3 V Voltage Regulator (Schematic: Appendix A.6.)

To design the circuit for our Texas Instruments low-dropout regulator (TPS75233QPWP), we referred to [7]. The most important part was to properly adjust the output voltage. This was done by following the provided voltage divider equation from the datasheet. The resistor from the feedback (FB)/sense pin to ground was suggested to be 30.1kΩ. In our schematic, Appendix A.6., this resistor is R37. The resulting resistor, R36 in the schematic, was calculated to be

$$R36 = \left(\frac{V_{out}}{V_{ref}} - 1\right) \cdot R37 = \left(\frac{3.3V}{1.1834V} - 1\right) \cdot 30.1k\Omega = 53.6k\Omega \quad (3)$$

where Vref is the chip's internal reference voltage, given by the datasheet, [7].

2.2.6. 5V Boost Converter (Schematic: Appendix A.7.)

The Texas Instruments boost converter, TPS61089RNRR, operates similarly to the 3.3V regulator in the sense that the output must be set by a voltage divider between the output voltage, a feedback pin, and ground. However, this boost converter is a switching converter meaning that it requires additional components like an inductor to lessen the output current ripple. To size our inductor, we referenced the datasheet [8] and chose a value of 1µH based on the provided table. More importantly, we needed to compute the resistance values for the output voltage divider. Resistor, R43, in Appendix A.7. is the resistor between FB and ground; this value was chosen to be 100kΩ in order to keep the FB current low. As a result, the output resistance, R42, was calculated as

$$R42 = \left(\frac{V_{out}}{V_{ref}} - 1\right) \cdot R43 = \left(\frac{5.0V}{1.222V} - 1\right) \cdot 100k\Omega = 309.17k\Omega \quad (4)$$

where Vref is the chip's internal reference voltage, given by the datasheet [8].

2.2.7. 12V Boost Converter (Schematic: Appendix A.8.)

This boost converter is the exact same as our 5V boost converter except for the voltage divider resistances. Resistor, R46, in Appendix A.8. is the resistor between FB and ground; this value was chosen to be 113kΩ in order to keep the FB current low and because R=100kΩ resulted in an output resistance that was hard to find as a real component. As a result, the output resistance, R45, was calculated as

$$R45 = \left(\frac{V_{out}}{V_{ref}} - 1\right) \cdot R46 = \left(\frac{12.0V}{1.222V} - 1\right) \cdot 113k\Omega = 1.00M\Omega \quad (5)$$

where Vref is the chip's internal reference voltage, given by the datasheet [8].

2.2.8. Tensorflow Machine Learning Overview

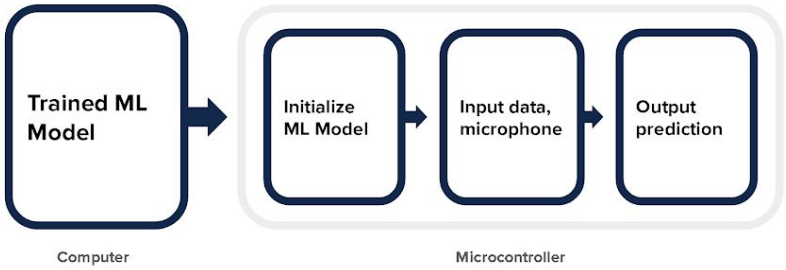


Figure 3: Keyword State Control

When training our ML model for speech recognition, we had to follow the general steps as shown in Figure 3. The first step is to train our model on the computer. As our team was using the specific keywords "one", "two", "three", "up", "down", "stop", "marvin", we had to train the model with audio files of those spoken words. Using the Google Speech dataset, we were able to train our model with the one-second examples of the words and reached a final test accuracy of 88%. After training the model, we then needed to convert the model to a format that was usable by our microcontroller. Therefore, we had to convert it to a Tensorflow Lite model, and finally a C++ array.

After training the model, you can then move from the computer to the microcontroller, which has its own specific set of instructions. On the microcontroller, you need to first initialize the model

that you trained with the training parameters like feature bins and window stride. After initializing, you can then get the input data from the microphone. The microphone returns the reading in terms of amplitudes, but our model requires analyzing the audio files in terms of frequencies, so we needed to convert the input data using fast fourier transform. With the amplitudes converted to frequencies, the model can then take the input features and perform the necessary calculations to assign scores to the keywords. Then, the keyword with the highest score is chosen as the final prediction. Upon the identification of a particular, our state control would then be able to be updated to the next state or update the score if all necessary keywords were spoken.

2.2.9. Speech Recognition State Control

In order to implement our state control, we chose to use Mbed OS which is an open-source RTOS by Arm. This allowed us to setup an easy to use development environment since it provided features such as prewritten interfaces and the ability to code using C++.

To drive our state control when using tensorflow, we need to first provide input data for the keyword spotting model. We sampled the microphone at a rate of 16kHz and stored the samples in an external buffer before passing the data to our model. Our model then converts the amplitudes by the microphone into frequencies using fast fourier transform, and splits the frequencies into a set number of feature bins. Then, it can run the final calculations, get its prediction, and then call specific functions when it recognizes a command.

When we switched to uSpeech, the design of our state control had to change. First, we needed to initialize constants for specific phonemes [9] . Once that is done, we then constantly check whether there is a sound being heard by the microphone. In our case, we were only able to utilize three phonemes effectively: ‘e’, ‘o’, and ‘sh’. We set ‘e’ and ‘o’ to indicate two teams and ‘sh’ to indicate an increment in the respective team’s score. Since the program is always listening for sounds, there needed to be a way to prevent the program from recognizing the other team in case

the user's pronunciation changes before incrementing the score. As a result, when the phoneme for a team is first heard, it sets a flag indicating that a team has been heard and which team it is. After hearing the 'sh' sound, that team will increment the score by generating a pulse which indicates to our microcontroller to increment that teams score. The flag is then reset.

2.2.10. Keyword Spotting Model

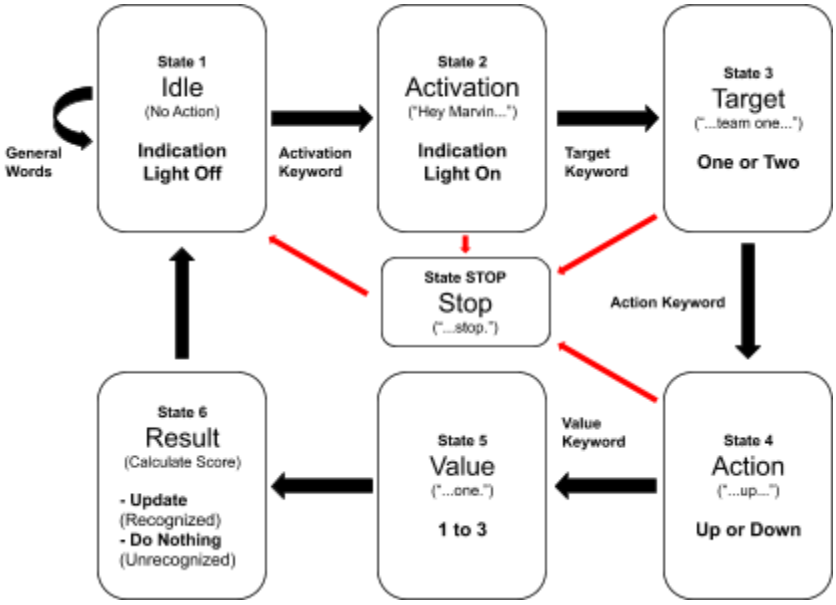


Figure 4: Keyword State Control

As shown in Figure 3, our keyword state control is responsible for keeping track of all the spoken keywords and updating the correct team with the specified amount. In summary, our keywords are recognized in a specific order. In the beginning, our device is in an idle state, but once the correct keyword for the next state is provided, it will move on to the next state. These states will specify which team, to increment or decrement, the amount to change, and then update the score.

Unfortunately, we were unable to use this state control since as mentioned, Tensorflow was unable to recognize any of the keywords. As such, we had to use uSpeech instead, which only provided us three spoken keywords: 'e', 'o', and 'sh' sounds. Thus our final state control was to determine the team by either 'e' or 'o' and increment the score when the 'shh' sound was heard.

3. Verification

3.1. Microcontroller (Requirements: Appendix B.1.)

To verify that our microcontroller (MCU) had the proper amount of flash storage and that it could properly communicate over I2C with the battery controller, we had to use a development board. Unfortunately due to a hardware bug in our 3.3V regulator, we were never able to put our STM32 on our own PCB. However, on the development board, we were able to verify that the MCU we chose had sufficient flash storage and could communicate over I2C.

3.2. 8-Bit Shift Registers (Requirements: Appendix B.2.)

To verify that our shift registers functioned properly, we soldered the chips to our board after we verified that our 5V boost converter was functioning properly. Then we connected the outputs to the displays through 15Ω current limiting resistors. Finally, we probed the pins to simulate the high and low toggle of the MCU's signals, and successfully verified that the shift registers shift properly and that the pins could safely sink the proper amount of current, 140mA.

3.3. Charge Controller (Requirements: Appendix B.5.)

The charge controller was simply verified by monitoring the input current to the battery. After the USB connector was connected to the board, and the charge control module was soldered, we could test the battery charging. Built into the charge controller is a pin that outputs the current charge state of the battery. We connected this pin to an LED, and monitored the status. If the battery was charging improperly, the controller was supposed to blink the status output at 1Hz. This never occurred for us, and we were able to observe that the controller properly cut off the charge current as the battery approached a full charge.

3.4. Battery (Requirements: Appendix B.6.)

To verify the battery, we simply tested the battery voltage at various charge percentages, and we found that its voltage output ranged from 3.65V to 4.1V, and as a result, its average voltage was 3.87V, slightly higher than its projected nominal voltage of 3.7V.

3.5. 3.3 V Voltage Regulator (Requirements: Appendix B.7.)

To verify that our 3.3V regulator met its requirements found in Appendix B.6., we first verified that the battery charging and output was proper, then we connected this voltage, Vbat, to the input of the regulator. Next we probed the output of the regulator and found the output voltage to be exactly the same as Vbat, the input voltage. This can be seen in Figure 4, where the blue signal is the battery voltage and the green signal is the regulator voltage. Despite attempts to debug and solve this problem, we could not identify the error. As a result, we could not meet our requirements, and additionally, we were never able to test the microcontroller on our own board.

3.6. 5V Boost Converter (Requirements: Appendix B.8.)

To verify the requirements for our 5V boost converter, we connected Vbat to the input of the converter. Next we probed the output of the converter with an oscilloscope. In Figure 5, the blue signal is the battery voltage, and the green signal is the boost converter voltage. We can see that our output voltage has an RMS value of 5.08V. Additionally our peak-to-peak ripple was 400mV. This means that our over-under voltage variation was

$$\pm v^{\sim}\ \% = \frac{peak-peak}{2(V_{max})} \cdot 100 = \frac{400mV}{2(5V)} \cdot 100 = \pm 4\% \quad (6)$$

where Vmax is our desired voltage of 5V. This ripple variation is successfully less than our required value of ±5%.

3.7. 12V Boost Converter (Requirements: Appendix B.9.)

To verify the requirements for our 12V boost converter, we connected Vbat to the input of the converter. Next we probed the output of the converter with an oscilloscope. In Figure 6, the blue signal is the battery voltage, and the green signal is the boost converter voltage. We can see that our output voltage has an RMS value of 12.2V. Additionally our peak-to-peak ripple was 1.20V. This means that our over-under voltage variation was

$$\pm v^{\sim}\ \% = \frac{peak-peak}{2(V_{max})} \cdot 100 = \frac{1.2V}{2(12V)} \cdot 100 = + \pm 2.5\% \quad (7)$$

where Vmax is our desired voltage of 12V. This ripple variation is successfully less than our required value of ±5%.

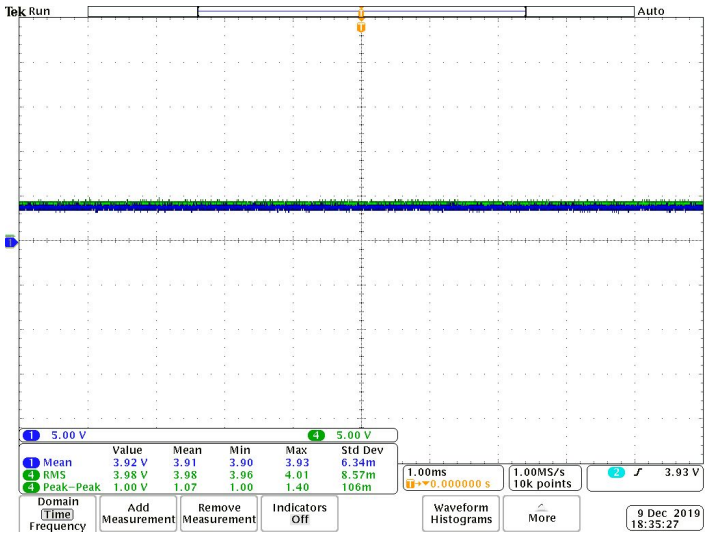


Figure 4: 3.3V Regulator Output

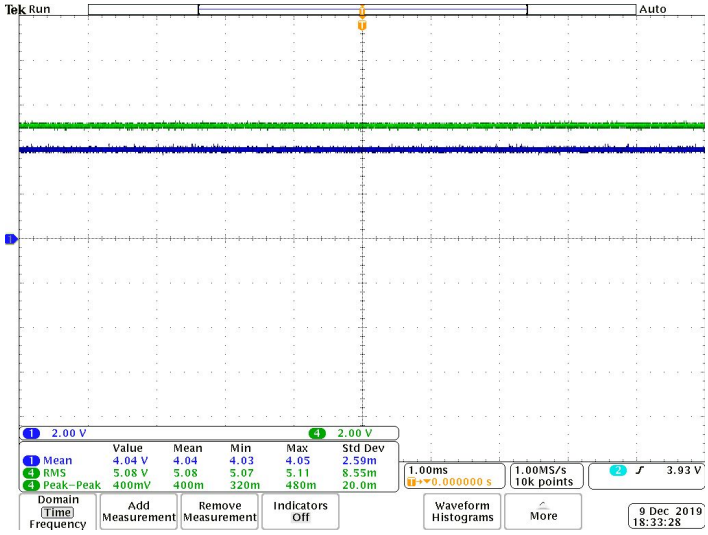


Figure 5: 5V Boost Converter Output

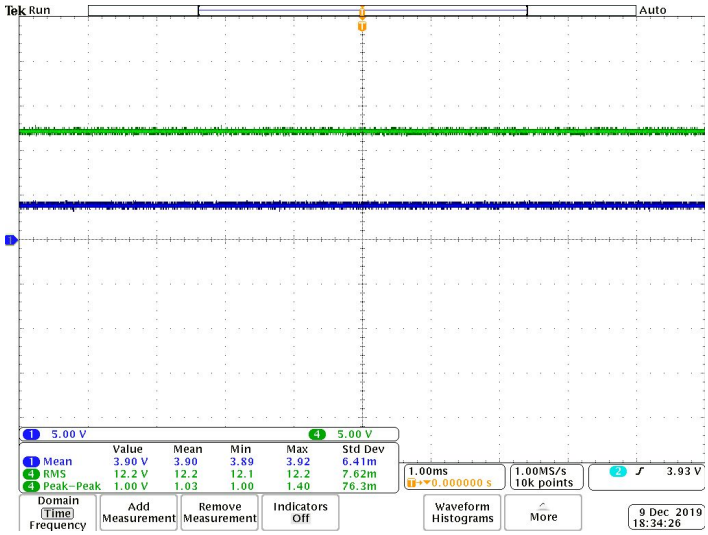


Figure 6: 12V Boost Converter Output

3.8. Speech Recognition State Control

When we first began, we had used Tensorflow Lite for our speech recognition. After training our machine learning model with the training data for our specified keywords, we had encountered issues with getting the incorrect predictions. Our first approach was to unit test our voice recognition system by breaking it down into its core components. Despite testing our main components such as the input data, microphone bias, and other parameters, we were unable to figure out what was wrong. It was only until we testing running Tensorflow Lite using the example model and example input data provided by Tensorflow that we discovered that Tensorflow did not run correctly on our microcontroller. After this discovery, we tried to run the model on other controllers such as the Arduino Uno and Nano, yet we did not have success recognizing our keywords and running the ML models. Therefore, as mentioned we transitioned to using the uSpeech library.

3.9. Keyword Spotting Model

Our Keyword Spotting model was revised after switching to the uSpeech library. The uSpeech library used phonemes, and as such our KWS used phonemes as well. After calibrating the uSpeech library as their mentioned documentation, we still found that we were unable to get great accuracy, and where therefore limited to only three phonemes: ‘e’, ‘o’, and ‘sh’. Therefore, our KWS was simple and used only those phonemes to recognize the team and add one to that team’s score. Verification of this functionality was done with debugging and printing directly to the computer’s console. While we did face issues with the team score not updating correctly or being decremented without warning, we finally were able to get the final system to work correctly and increment the correct team.

4. Costs

As seen in Appendix C as C.1., our estimated prototype cost is $148.58.

According to the IlliniSuccess 2017-2018 annual report [10], the average salary for Computer Engineering graduates was $92,430 and $76,079 for Electrical Engineering graduates. This translates to an hourly rate of $46.22 and $38.04 respectively. As a result, our total development cost can be seen is calculated in (6) to be $48,930.

$$(2 \bullet \frac{\$46.22}{hr} + \frac{\$38.04}{hr}) \bullet 2.5 \bullet \frac{10hrs}{week} \bullet 15\ weeks = \$48,930 \quad (8)$$

5. Conclusion

The biggest goal for future improvements would be finding a better method of voice activation. We looked to use Tensorflow because we were able to find examples of other projects using it successfully. If Tensorflow is not an option however, there are alternative solutions such as prebuilt voice recognition kits which come with set keywords. These kits can be implemented into our system similar to how we utilized uSpeech.

Although we were unable to achieve our goal using keywords to control the scoreboard, we were able to implement a simpler version using sounds instead. Through this project, we learned a lot about the process of making an idea a reality. We learned skills such as PCB design, part picking, and testing. We also faced many challenges and roadblocks in the process such as our tensorflow model failing. Despite the challenges we faced, we were able to find alternative solutions to our problems.

5.1. Ethics & Safety

Our projects main ethical concern is the voice recognition system. In recent years, users are becoming increasingly aware of the data that is being collected by large companies such as Google, Amazon, and Facebook; much of that data being collected without the knowledge and consent of the user. Amazon Echo devices have been an especially large privacy and safety concern ever since it was revealed that the device has recorded private conversations and even shared them [11]. This should not have been a surprise to consumers. Not because it is "obvious" that they are recording, but because these companies should have made sure that consumers were aware of the data being collected. According to the IEEE Code of Ethics #1 and #5 [12], engineers must ensure the safety of the public and improve their understanding of emerging technologies. Large tech companies are choosing to ignore these codes and leaving consumers unaware and oblivious to threats to their privacy. Because our project also utilizes a voice recognition system, we need to be cognizant of these facts.

Our project mitigates the dangers of voice recognition in the following manners. First, the length of time that the recording of the user will exist should be no more than a few seconds. By the time the score has been updated, the recording should no longer exist in the system. The system has a limited amount of memory (128kb), with the majority being used by the operating program and KWS model. The leftover amount of memory will likely range from a couple kb to around ten kb. If the system were to keep the voice recordings, it would quickly run out of memory. In addition, the system does not have internet connectivity, unlike Alexa. Alexa operates using a server to handle the voice processing and must upload the user's voice command to the server where memory is not an issue. Because our system is not connected to the internet, it cannot upload the user's voice recordings and must delete them to save memory.

5.2. Impact

While our project may not have much impact in global and economic contexts. It is able to positively impact society by providing a quality of life improvement to casual sport players, a demographic which encompasses most humans on Earth. Further improvements to the project such as reducing production costs will only increase its impact by reaching those in less developed nations where its current cost may be too expensive for the average citizen.

References

[1] M. Kunz, “265 million playing football,” FIFA magazine, 2007. [Online]. Available: https://www.fifa.com/mm/document/fifafacts/bcoffsurv/emaga_9384_10704.pdf. [Accessed: 11 Dec. 2019].

[2] Y. Zhang, N. Suda, L. Lai, and V. Chandra, “Hello Edge: Keyword Spotting on Microcontrollers,” *arXiv*, February 2018. [Online]. Available: https://arxiv.org/pdf/1711.07128.pdf. [Accessed: 22 Sep. 2019].

[3] “Arm® Cortex®-M4 32-bit MCU FPU,” Apr-2019. [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/datasheet/9e/50/b1/5a/5f/ae/4d/c1/DM00086815.pdf/files/DM00086815.pdf/jcr:content/translations/en.DM00086815.pdf. [Accessed: 2019].

[4] “TPIC6C596 Power Logic 8-Bit Shift Register,” Mar-2015. [Online]. Available: http://www.ti.com/lit/ds/slis093d/slis093d.pdf. [Accessed: 2019].

[5] “bq25896 I2C Controlled Single Cell 3-A Fast Charger,” May-2018. [Online]. Available: http://www.ti.com/lit/ds/symlink/bq25896.pdf. [Accessed: 2019].

[6] “Charging Lithium-Ion Batteries,” *Charging Lithium-Ion Batteries – Battery University*. [Online]. Available: https://batteryuniversity.com/learn/article/charging_lithium_ion_batteries. [Accessed: 2019].

[7] “TPS752xx-EP w/Reset TPS754xx-EP w/Power Good Fast Trans Response Volt Regulators,” Apr-2003. [Online]. Available: http://www.ti.com/lit/ds/symlink/tps75201-ep.pdf. [Accessed: 2019].

[8] “TPS61089x 12.6-V, 7-A Fully-Integrated Synchronous Boost Converters,” Jul-2016. [Online]. Available: http://www.ti.com/lit/ds/symlink/tps61089.pdf. [Accessed: 2019].

[9] A. Chakravarty, “uSpeech,” Github Repository. [Online]. Available: https://github.com/arjo129/uSpeech. [Accessed: 11 Dec. 2019].

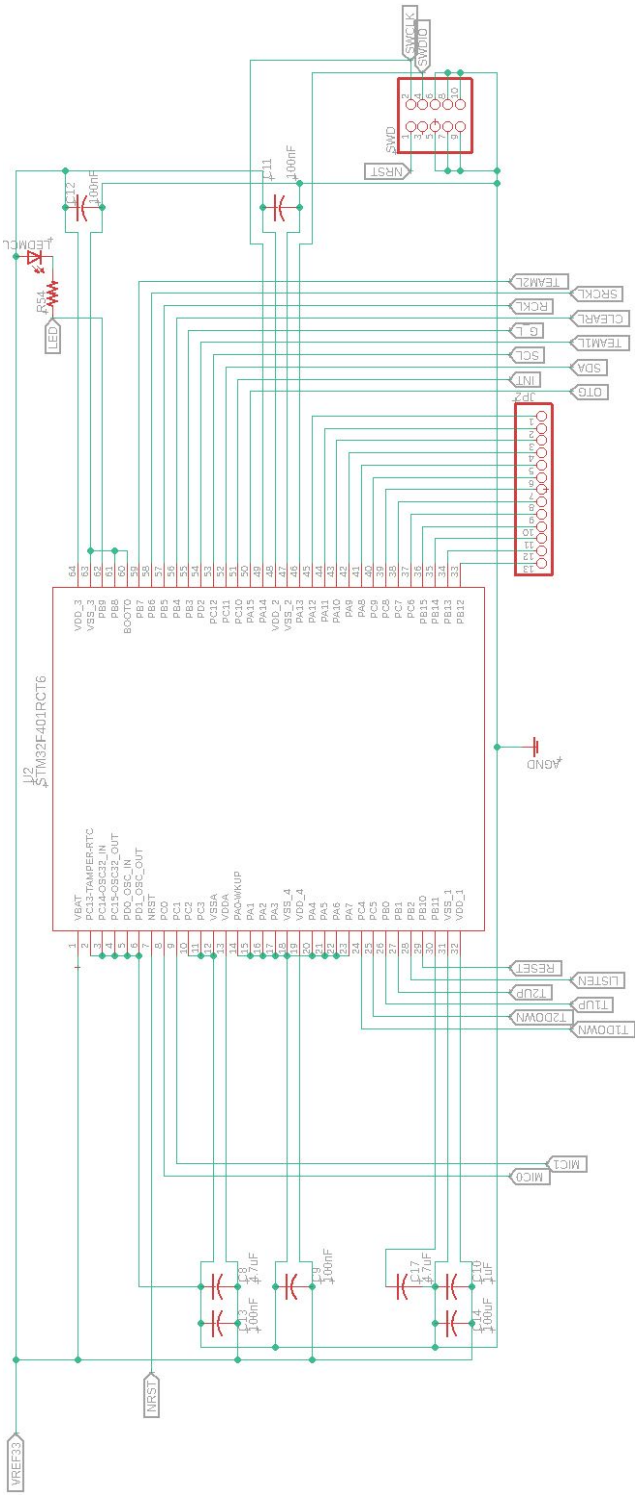
[10] “All Campus Undergraduate 2017-2018 Report,” Illini Success. [Online]. Available: https://uofi.app.box.com/s/ml9oh48vawrhm7e019kw5ix4j91j605p. [Accessed: 03 Oct. 2019].

[11] C. Osborne, “Amazon confirms Alexa customer voice recordings are kept forever,” *ZDNet,* July 2019. [Online]. Available: https://www.zdnet.com/article/amazon-confirms-alexa-customer-voice-recordings-are-kept-forever/. [Accessed: 03 Oct. 2019].

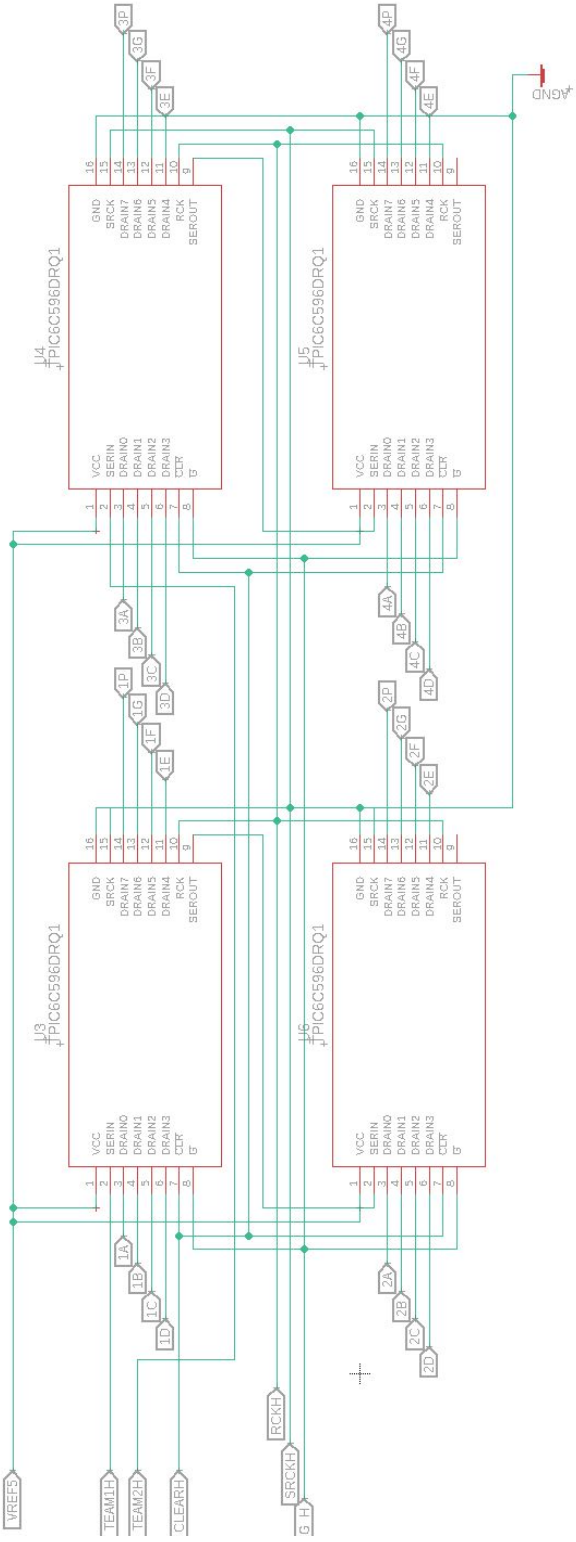
[12] “IEEE Code of Ethics.” *IEEE,* 2019. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 19 Sep. 2019].

Appendix A - Schematics

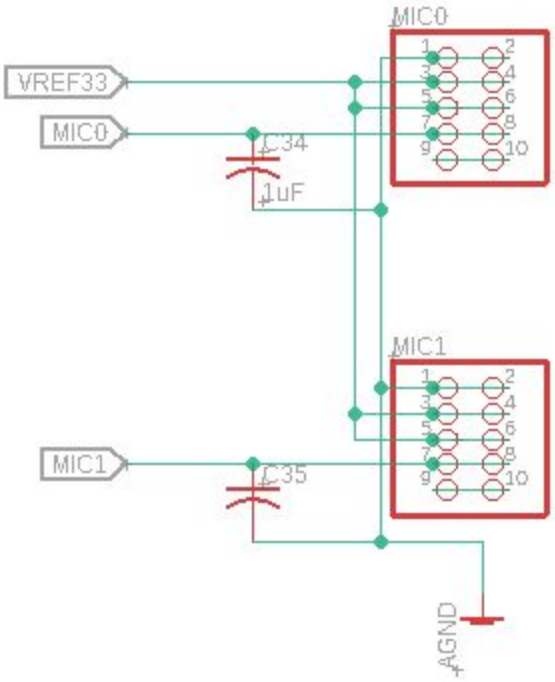
A.1. Microcontroller (STM32F401RCT6)



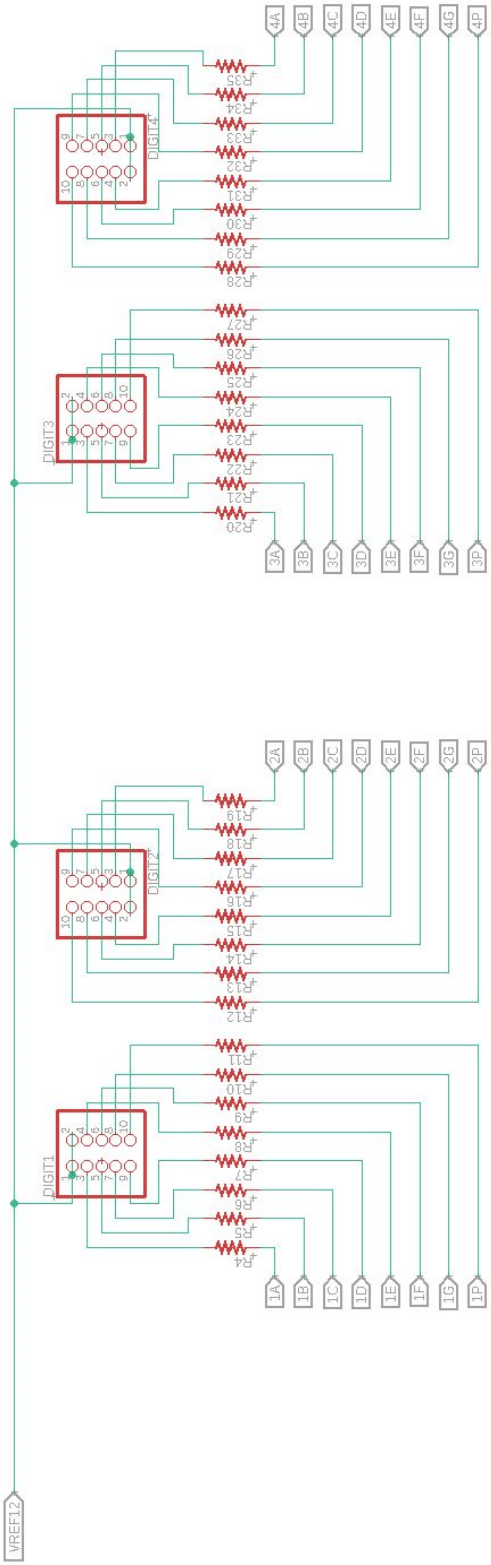
A.2. Shift Registers (TPIC6C596DRQ1)



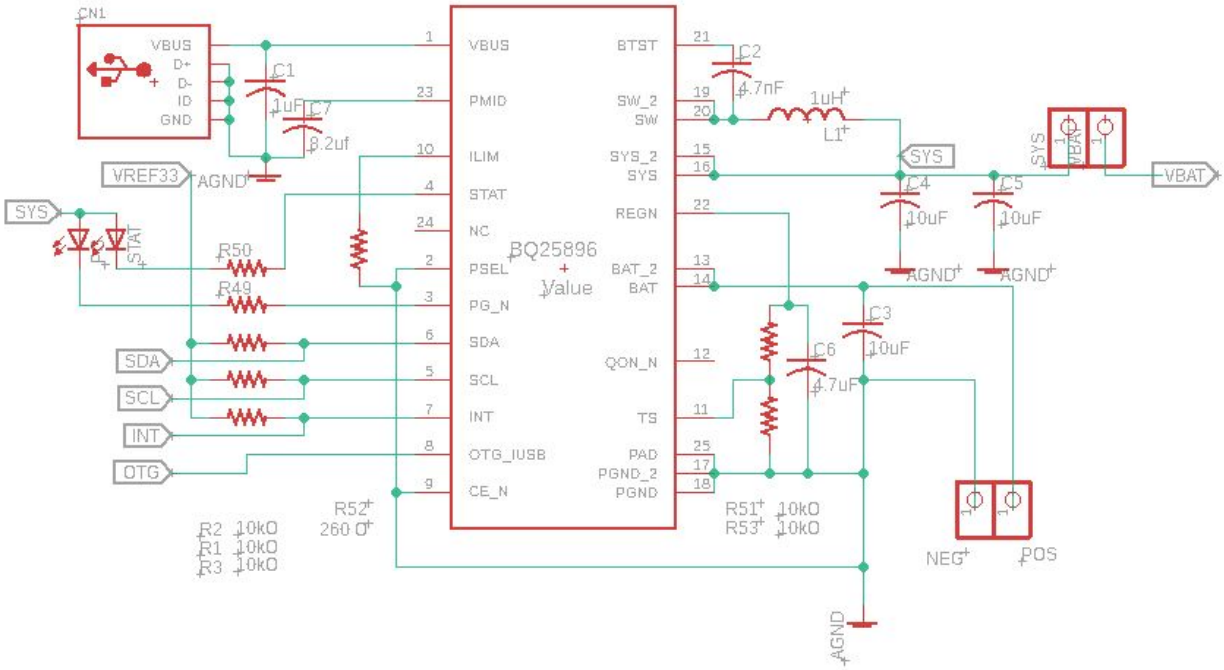
A.3. Microphone Connection



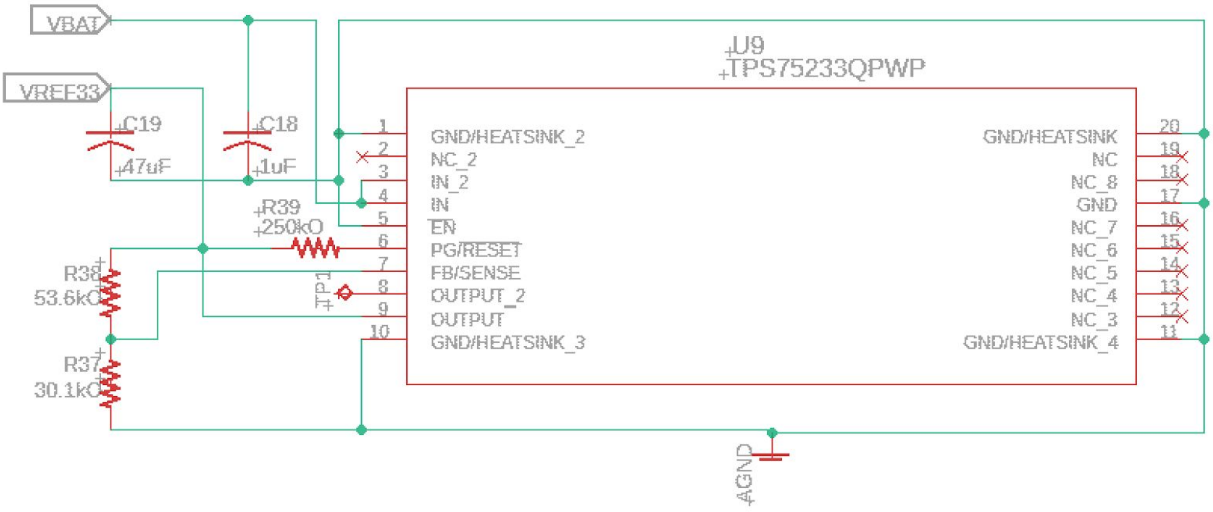
A.4. Seven-Segment Display Connections



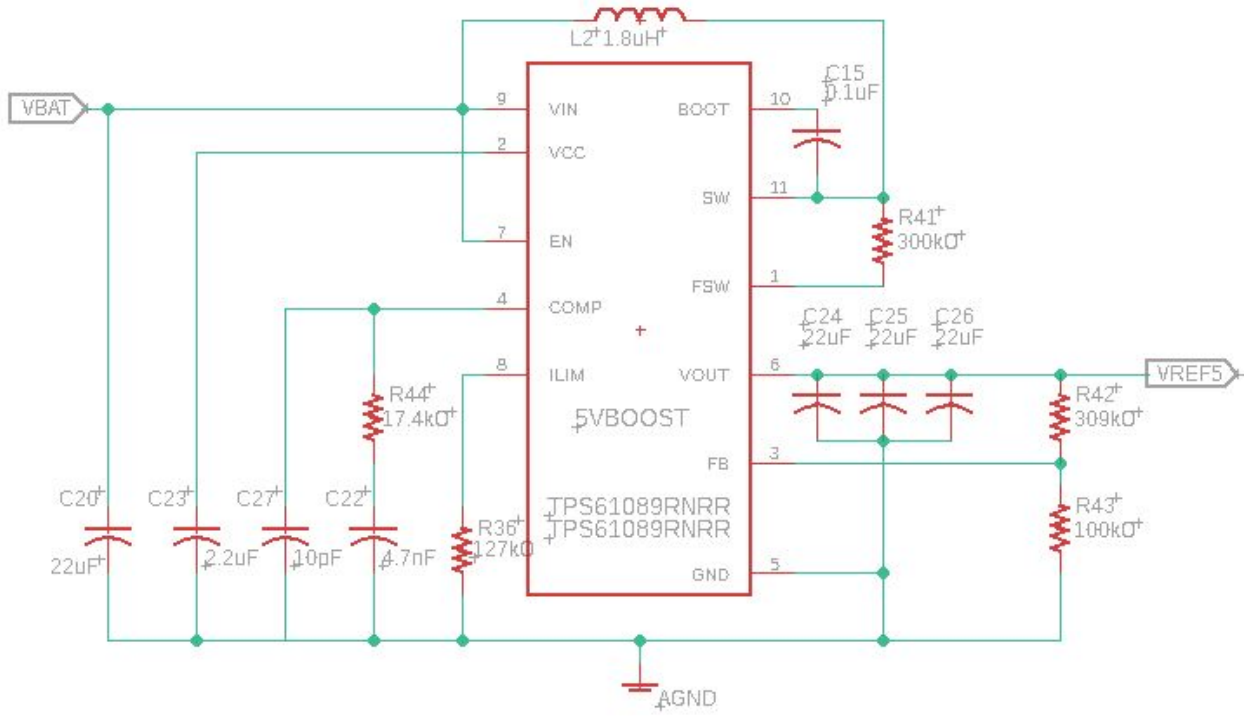
A.5. Battery Charging IC (BQ25896RTWT)



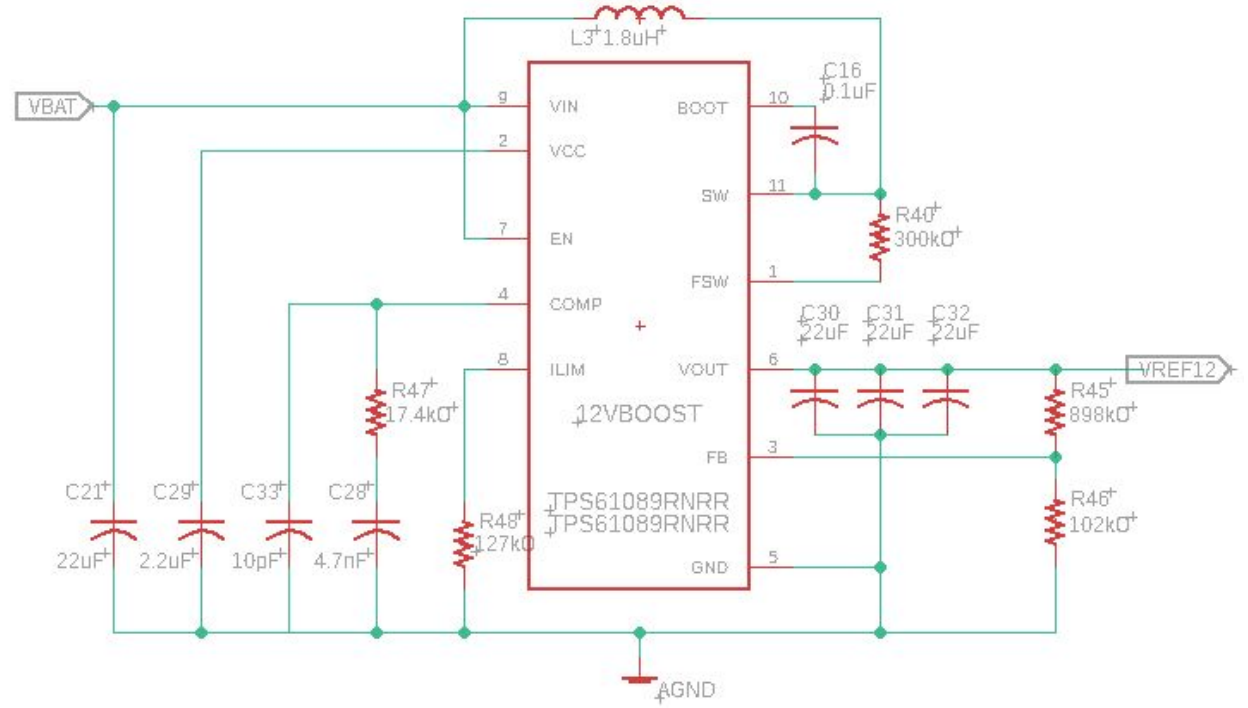
A.6. 3.3V Low-Dropout Regulator (TPS75233QPWP)



A.7. 5V Boost Converter (TPS61089RNRR)



A.8. 12V Boost Converter (TPS61089RNRR)



Appendix B - Requirements and Verification Tables

B.1. Microcontroller (STM32F401RCT6)

Requirements	Verification
1. Flash memory is ≥ 100kb	1. a. Connect MCU to PC with an ST-Link connector b. Compile and load a test project with 100kb of data into flash memory using Arm Mbed IDE c. Connect MCU to PC with a USB-to-Serial Converter d. Using a terminal (ie. PuTTY), connect to the board on the associated COM port e. Transmit data loaded in flash memory from the MCU using USART f. Verify that data received matches data loaded
2. Must communicate over I2C	2. a. Connect MCU to a chip which can generate data and communicate over I2C(ie. RTC, Temperature sensor, etc.) b. Connect MCU to PC with a USB-to-Serial Converter c. Using a terminal (ie. PuTTY), connect to the board on the associated COM port d. Transmit data received over I2C from sensor to terminal using USART e. Verify data received is data that can be generated by the chip

B.2. Shift Registers (TPIC6C596DRQ1)

Requirements	Verification
1. Must have Serial in Parallel Out	1. a. Feed 4 bits of data into the register through serial in b. Read data from parallel out on 4th clock cycle c. Verify data read is same as data entered
2. Each drain pin must be capable of sinking at least 140mA	2. a. Connect device to 5V power b. Insert serial data using microcontroller dev board c. Drive an output drain pin low with a 40Ω, 5V load attached d. Measure drain current with multimeter

B.3. Microphone

Requirements	Verification
1. Must pick up audio from ≤ 10 ft away	1. a. Drive chip at 3mA b. Connect chip output to an oscilloscope c. Measure 10 ft from microphone d. Play music/speak and verify that the signal is appearing on the oscilloscope

B.4. Seven-Segment Display (Sparkfun COM-08530)

Requirements	Verification
1. Each segment must illuminate when applied with 12V and drive at a maximum of 140mA	1. a. Supply display with 12V and drive current of 140mA b. Ensure segments are illuminated

B.5. Battery Charging IC (BQ25896RTWT)

Requirements	Verification
1. Must provide the battery with a constant charging voltage of 4.2 V.	1. a. Properly wire the chip with attached battery b. Measure the charging voltage across the battery with an oscilloscope
2. Must stop charging the battery once the charging current drops below 3% of the battery's rated current.	2. a. While charging, measure the battery input current with a multimeter

B.6. Battery (PkCell ICR18650)

Requirements	Verification
1. Must provide a nominal output voltage of 3.7 V ± 1%	1. a. Connect a simple load to the battery to drive 1A b. Measure the voltage across the load using an oscilloscope
2. Must provide a continuous discharge current of at least 1.5A (.23C) for four hours	2. a. Connect a load of 2.5 Ohms b. Measure the load current with a multimeter every ten minutes

B.7. 3.3V Low-Dropout Regulator (TPS75233QPWP)

Requirements	Verification
1. Must supply 3.3V ± 1% with a varied input voltage range of 3V to 5V	1. a. Connect output to an oscilloscope b. Input a range of DC voltages from 3V to 5V c. Monitor the output voltage with an oscilloscope
2. Ensure the chip can drive up to 2A	2. a. Vary the load across the output b. Monitor the load current using a multimeter

B.8. 5V Boost Converter (TPS61089RNRR)

Requirements	Verification
1. Must supply 5V ± 5% with a varied input voltage range of 3V to 5V	1. a. Connect output to an oscilloscope b. Input a range of DC voltages from 3V to 5V c. Monitor the output with an oscilloscope
2. Ensure the module can drive up to 3A	2. a. Vary the load across the output b. Monitor the load current using a multimeter

B.9. 12V Boost Converter (TPS61089RNRR)

Requirements	Verification
1. Must supply 12V ± 5% with a varied input voltage range of 3V to 5V	1. a. Connect output to an oscilloscope b. Input a range of DC voltages from 3V to 5V c. Monitor the output with an oscilloscope
2. Ensure the module can drive up to 3A	2. a. Vary the load across the output b. Monitor the load current using a multimeter

Appendix C - Cost Table

C.1. Cost breakdown of prototype

Power Subsystem							
Part Name		**Make**	**Part No.**	**Cost ($)**		**Quantity**	**Total ($)**
				Single	**Bulk***		
1	Battery PMIC	Texas Instruments	BQ25896	3.87	2.35	1	3.87
2	Battery	Pkcell	ICR18650	29.50	-	1	29.50
3	Voltage Regulator	Texas Instruments	TPS75233	10.81	6.13	1	10.81
4	5V Boost	Texas Instruments	TPS61089RNRR	2.94	2.94	1	2.94
5	12V Boost	Texas Instruments	TPS61089RNRR	2.94	2.94	1	2.94
Control Subsystem							
Part Name		**Make**	**Part No.**	**Cost ($)**		**Quantity**	**Total ($)**
				Single	**Bulk***		
1	Microcontroller	STMicroelectronics	STM32F401RCT6	6.27	3.32	1	6.27
2	Shift Registers	Texas Instruments	TPIC6C596DRG4	1.13	0.51	4	4.52
User Interface Subsystem							
Part Name		**Make**	**Part No.**	**Cost ($)**		**Quantity**	**Total ($)**
				Single	**Bulk***		
1	Microphone	CUI Inc	CMA-4544PF-W	0.83	0.38	2	1.66
2	Amplifier	Maxim Integrated	MAX9814	1.51	0.85	2	3.02
3	Seven-Segment Displays	SparkFun Electronics	COM-08530	18.95	-	4	75.8
4	Listening Light	Kingbright	WP154A	1.95	0.83	1	1.95
5	Power/Charge Status light	Kingbright	WP154A	1.95	0.83	1	1.95
6	Power Button	E-Switch	TL2201EEZA1CWHT	0.67	0.37	1	0.67
7	Increment Button	E-Switch	TL2201EEZA1CWHT	0.67	0.37	4	2.68
						TOTAL PARTS COST:	148.58
							*Price/piece (1000pcs)