Magic Ears:
Wirelessly Synchronized LED Mickey Mouse Ears

ECE 445 Final Report

Team 8 - Kaitlin Vlasaty and Ian Napp
TA: Hershel Rege
Spring 2019

Abstract

Within recent years, there has been an increasing interest in wearable technology and the various ways it can upgrade our lives. We sought to improve an existing Disney product and find some Disney magic of our own. Magic Ears are a way for families to create entertaining light shows together and enhance their experience at a park. Magic Ears were developed from the traditional appearance of Mickey Mouse ears. The design was updated to include more LEDs and WiFi capability through an Android application. Through several iterations of testing and debugging, Magic Ears became a functional product. Users can connect their phones and have control over their own headset or a group of headsets. This project combines equal parts hardware and software to create an smart, visually pleasing product. It proved to be an excellent learning experience in terms of both technical knowledge and engineering skills.

Contents

1 Introduction

1.1 Motivation

Visuals are an important component of all Disney products and creations. Everything is carefully crafted to give the audience the best experience, and the theme parks are a great example. Parks and resorts are one of Disney's main revenue sources, with over 150 million park visitors worldwide in 2017 [1]. There is an abundance of products available, and at night anything with lights becomes especially appealing. One of these products is the iconic Mickey Mouse ear hat, which has been modernized to include LEDs. Originally these were standalone devices, but later a communication aspect was introduced. Disney released Glow with the Show (now called Made with Magic) products to go along with their light shows starting in 2012 [2]. The shows premiered in DisneyLand but are now integrated into all the parks. The translucent plastic ears attach to a hat, and each ear contains one LED. The hats are powered by two AAA batteries and weigh about 110 grams [3].

The controls are based on IR communication: focused emissions are sent to certain parts of the crowd that allow producers to create patterns and dynamic effects throughout the area [4]. However, original models of the ears had limited functionality outside of the parks. The version we purchased was controlled only by the power switch; the LEDs cycled through a predefined set of three colors — orange, yellow, and magenta. We were inspired by Disney magic to create our own version of these headsets through a project that combines our interests in LED wearables and wireless communication systems. We wanted to understand the process of developing a product, including the research, building, troubleshooting, and documenting required to turn an idea into reality.

1.2 Solution

Through the development of Magic Ears, we aimed to improve on the existing design. Magic Ears consist of a headband with ears lined with RGB LEDs. The user controls the lights using a phone application, and the communication happens over WiFi within a closed wireless ad hoc network. There are options to change the color or choose from a selection of light patterns. Our design adds more lights and more complex circuitry which allows users to create their own lighting experience no matter where they are. The original hat has one LED per ear, and therefore requires a large crowd of people wearing the hats to create the lighting effects. Our design includes seven lights per ear so that patterns are made with only one or a few headsets.

Magic Ears create a complete system with a straightforward user interface. The IR receiver in the original product has been replaced with the ad hoc WiFi network, under the SSID 'MagicEars', and can be joined by any mobile device. With the Magic Ears Android app, users can select one of 16 solid colors or 5 light effects. By pressing a button, the color is displayed on all headsets within the network. The command is processed within the control module, which is also responsible for maintaining the WiFi network. The lights are strips of WS2812B individually addressable LEDs that line the interior cavities of two circular 3D printed ears. The lights are powered by a lithium polymer (LiPo) battery with an output between 3.7 and 4.2 V. This gets

boosted via a regulator that controls the voltage for each component. The PCB is contained within one of the ear cavities, and the battery is housed in the other.

As stated in our original documentation, there were three high-level requirements that defined the success of our project. First, the LED-lined ears should be attached to a headband that is safe and comfortable to wear and weighs between 255 and 340 g. Our final product was even lighter than our expectations at 179 g. We also imposed size restrictions in terms of PCB size, ear diameter, and ear thickness. The final dimensions were all within the given ranges: the PCB is 62 mm by 62 mm, the outer diameter of the ear is 105 mm, and the thickness of the ear is 15.3 mm. The third requirement involved the IEEE 802.11b ad hoc WiFi network within a range of 90 to 150 m. We found the average range of the WiFi to be 114 m, as described in Appendix B.

In addition to testing the subsystems, it was important to verify that the overall solution was working as intended. We did this in stages, first combining the lights and WiFi control, then testing it on the PCB and finally incorporating the app. At each step we confirmed the functionality and debugged as necessary. We had three complete headsets assembled at the time of the final demo, with the WiFi modules all programmed to run the same code. When turned on, the three formed a network that we connected to using an Android phone. Using the Magic Ears app, we controlled the lights using the buttons and saw those commands reflected on the headsets' lights. The network will dynamically adjust if a headset turns off or goes out of range, or if a new headset is added. We wanted our design to be comparable to the original Glow with the Show ears in terms of size and cost. While not intended to be mass produced, we believe that Magic Ears would be appealing to consumers or anyone interested in wearable LED technology.

2 Design

2.1 Introduction

The block diagram for our headset is separated into four major subsystems: power, lights, control, and the mobile application. However, we have changed the placement of a few components to better represent their function in our design, as seen in Figure 1. The power block supplies the correct voltages to other components, charges the battery, and turns off the headset. We have updated the diagram to show that the voltage from the battery is variable until it reaches the voltage boost. The RGB LED light strips, wrapped facing outward along the inside edge of the ear, are individually addressable. The control block includes the WiFi module which also acts as the microprocessor for the lights. It includes the button used for activating the battery level and distress light effect code; the button's primary purpose is sending data signals to the microcontroller. Our last subsystem is the Android app, which controls the light instructions and shows how many headsets are on the network. Users can choose what color or sequence they want as shown in Figure 5.

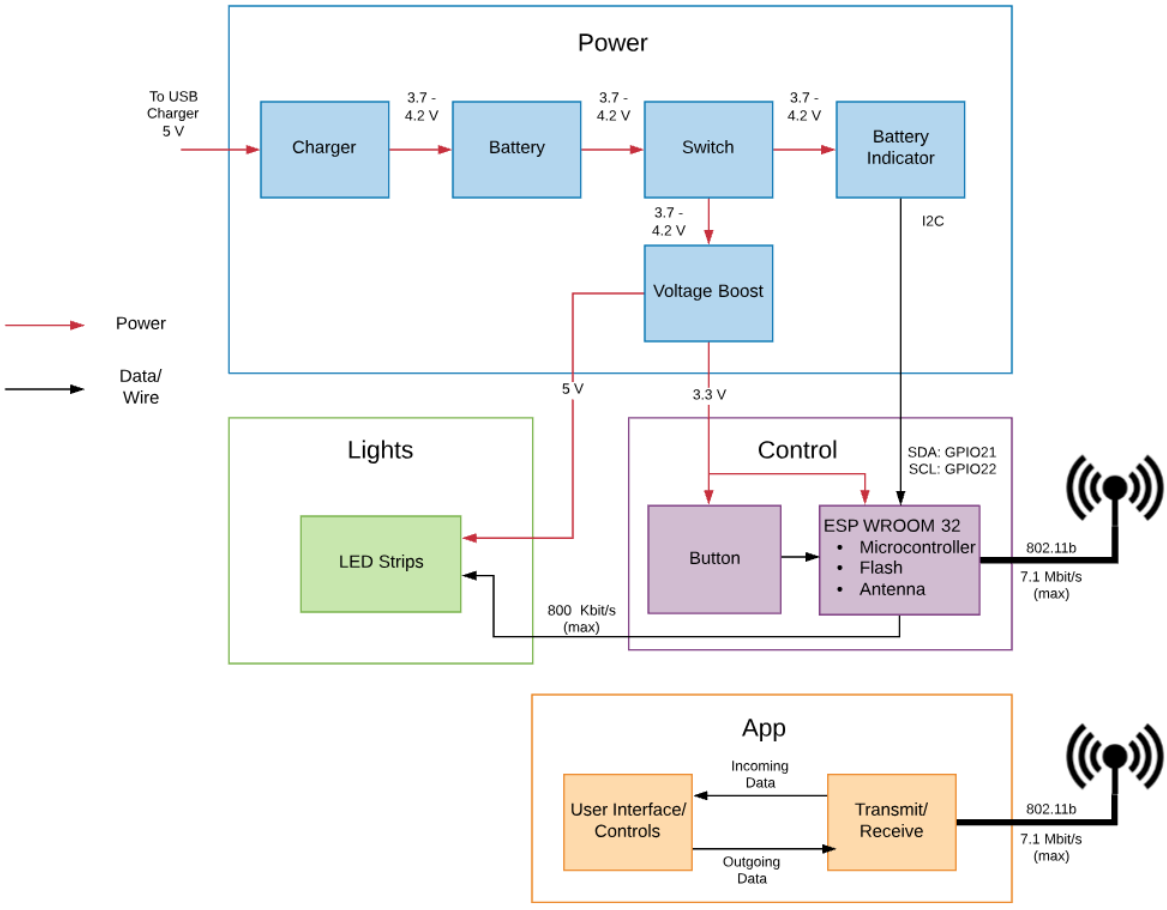

Figure 1. Block Diagram of Headset System Components

2.2 Physical Design

The physical design is an important aspect of our project. Since it is wearable, it needs to be light, balanced, and comfortable on the head. The features of the headset include two 3D printed ears attached to a headband by screws. Wires connect the lights and the battery running between them on the underside of the headband. Based on our objectives, the ear cavities are 105 mm in diameter and lined with LED strips. The 62 x 62 mm PCB fits snugly in one ear while the other houses the battery. The ears are made of PET-G transparent plastic which was chosen because it's more durable and easier to print with than ABS, another type of 3D printing plastic.

Since the Design Document, we have made several changes to the physical design to improve the look and ease of construction. Firstly, the ears are not interlocking pieces that snap together in the middle. Instead, we have made the back side removable like a lid so that the LED strips can be inserted more securely. This also helps with testing since the parts can sit in the ears more easily without being completely sealed. We also added a fabric strip to cover the wires and screws. This prevents hair getting caught in the headset and improves the look of the design. Finally, we changed the shape of the ears so that the rim is rounded instead of flat. This casts the light from the LEDs around the edge of the ears more efficiently and makes it easier to secure the strips in the ear, since they can be inserted snugly into the rounded part of the cavity.

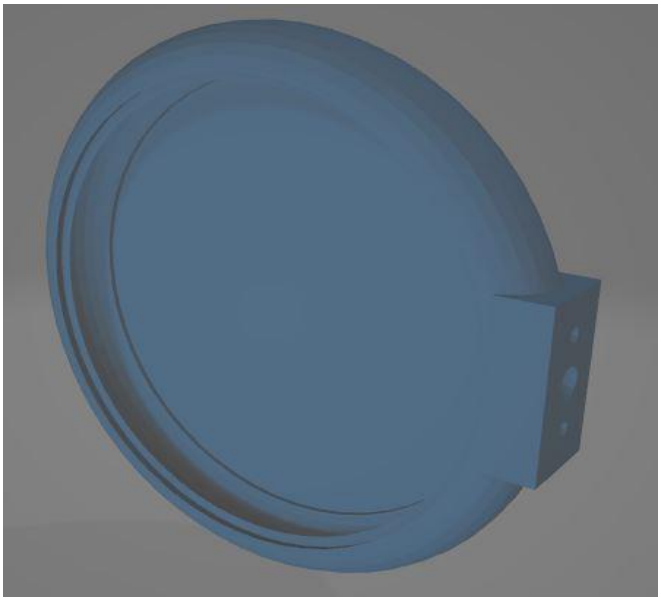

Figure 2. Final 3D Ear Model, Without Lid Piece

The most important part of the physical design is the printed circuit board (PCB) as it is the foundation for all other parts. To create a functioning PCB, we went through a series of assembly steps. First, we secured the PCB to a flat surface so that it wouldn't shift while we applied the solder paste. When ordering the boards, we also ordered an SMD solder paste stencil that had cutouts for each pad on the board. The stencil was placed on top of the PCB and lined up carefully. The stencil was secured using foam blocks taped down tightly against all four sides to reduce the amount it could move. We then spread solder paste above the pad cut

outs and used a plastic card to spread it evenly across the cut outs, making sure the stencil was held down so that the paste would be in the right spots.

After that we were able to simply lift the stencil and place the parts on the board according to the schematic. To solder down the parts, we placed the PCB in a toaster oven set to 450 °F for about five minutes or until all the solder had clearly melted and flowed onto the pads and parts [5]. We removed the PCB using an oven mitt and let it cool for a few minutes before it was safe to handle. It is worth noting that this toaster oven is not used for food and is clearly marked that it is contaminated with lead from the solder. We also made sure not to touch the solder paste with bare skin and clean any tools we used with rubbing alcohol. Once the PCB was cool, we checked for any imperfections like parts that did not attach correctly, accidental bridges, or connections that needed more solder. Finally, the power switch, which was a through hole component, was soldered to the board and some solder was added to the micro USB connector to secure it more firmly to the board. We also soldered the first seven-light strip to the board to test that it worked properly with the lights.

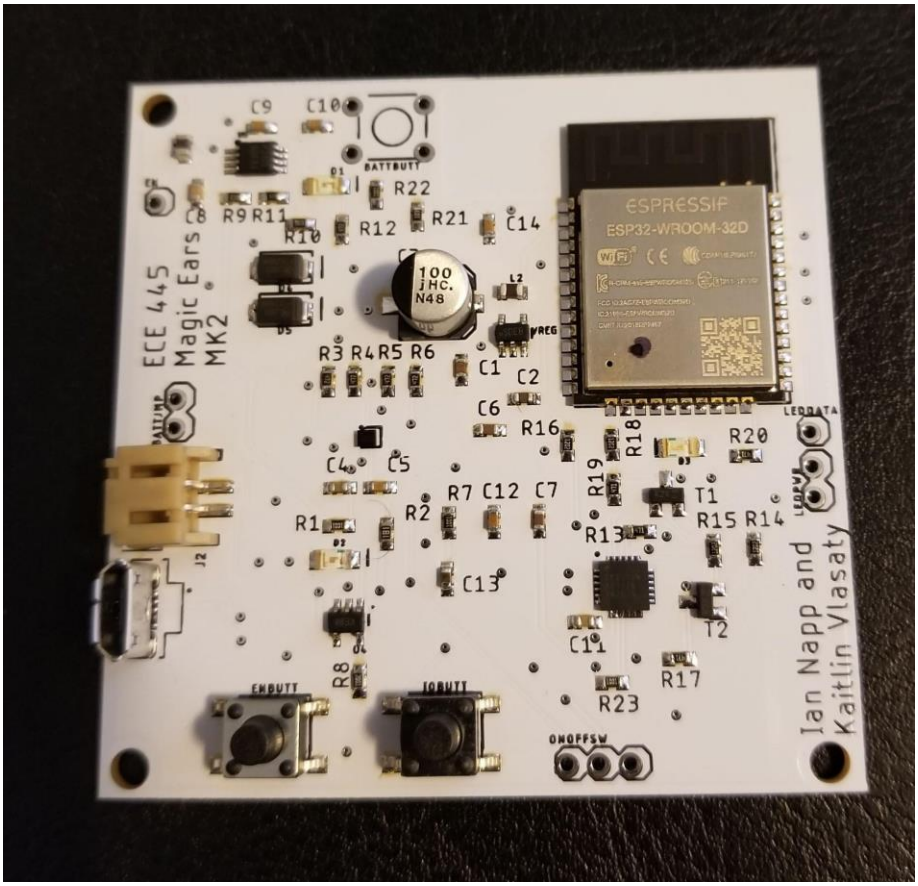

Figure 3. Constructed PCB

To assemble the headset seen in Figure 4, it was created from the bottom up starting with the headband. We drilled six holes in each headband total: four for the screws using a 2 mm drill bit, and two larger holes for the wires using a 4 mm drill bit. Next, brass inserts were sunk into the

3D printed ears. A soldering iron to heated them up enough to melt the plastic for insertion. They were originally sunk straight into the ears, but this forced plastic into the middle of the insert and made it hard to use the screws and equally hard to scrape out the plastic. We found that putting the screw into the insert first and then sinking both into the headset made the process cleaner. Next, we screwed one ear into the headband and the PCB was inserted into the cavity with the lights wrapped around the edge. To bridge the second set of lights in the other ear and the battery to the PCB, 15 mm long wires were cut to create the necessary connections. This was ultimately a little longer than what was needed but the wires were easily trimmed. The wires, first soldered to the end of the first strip of lights and the JST connector for the battery, were ran along the underside of the headband and up into the cavity of the second ear which was then attached. The remaining lights and battery were soldered to their respective wires, and the PCB was tested to make sure everything was still properly working. Finally, a piece of felt was cut and glued on to the headband, which masks the wires and keeps hair from getting caught in the screws.

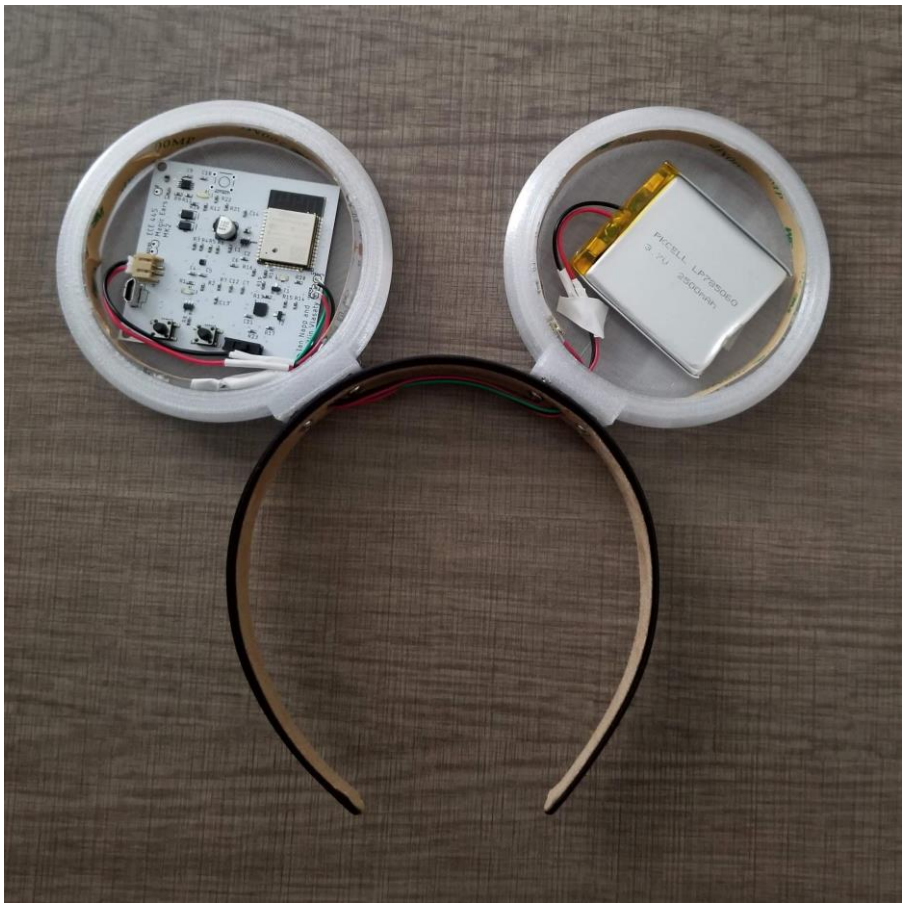

Figure 4. Constructed Headset, Without Felt Cover

2.3 Power Block

2.3.1 Charger

Lithium polymer batteries are not cheap or easily disposable, therefore they need to be recharged after use. This necessitates the use of a charging circuit and IC to safely provide power to the LiPo battery. This is done with the MCP73831 in our design; it can provide 4.2 V at a max current of 1.2 A from an input of 4.5 to 10 V. The battery can be charged from a common 5 V wall adapter or a USB port which provides the same power. The schematic for the charging circuit can be seen in Appendix D, Figure 8.

2.3.2 Battery

Since our design is portable, we need a battery to power it for a day's worth of use. Form factor is a constraint for our project, so we chose thinner, higher capacity lithium polymer batteries over AAA batteries. Originally, we planned on using two 3.7 V, 1000 mAh batteries in series to provide enough voltage to power the 5 V LEDs. Using two LiPo batteries proved to be tricky, especially in terms of charging, and we found that one 2500 mAh battery used in conjunction with a 5 V buck boost circuit could be used much more easily while taking up the same amount of space.

2.3.3 Switch

The switch was problematic to figure out since we kept changing what exactly we wanted it to do. When we made our first round PCBs, we found that we had shorted the traces for the switch in the layout and we could not turn off the power to the rest of the device. We also discovered that the charging circuit did not work under a load, even if it was just the WiFi module and microcontroller with the lights off. This prompted us to take a more careful look at the circuit and properly wire the switch for the second round. It currently functions to power the entire PCB with the battery, or 5 V USB connection when attached, when switched "on". It is only connected to the charging circuit to the battery when the switch is "off".

2.3.4 Fuel Gauge

The battery indicator was intended to display the power level of the battery in a fun and practical way. The IC could read LiPo battery voltage and send that data via I2C. The microprocessor would take that data and light up a number of lights on the headset based on the fuel percentage. The chip we used, the MAX17043, was where we had problems with the circuit. On the first round PCB, we used a TDFN-8 package that would have been easy to work with when it came to soldering. However, the IC with this pad layout was not available anymore and the only other option we had was the same chip with a 9-pin BGA package. This layout was very small and was very difficult to work with even with a solder stencil. We attempted to manually spread a thin layer over the pads but after heating the solder paste, found that it clumped together too much to work properly and could not get it working in the end.

2.3.5 Voltage Regulators

Since the output of the battery ranges from 3.7 to 4.2 V and the LEDs require 5 V to operate while the WiFi module can only handle a maximum of 3.3 V, we needed multiple circuits to handle stepping up or down the voltage. To increase the voltage to 5 V, we used the PAM2401 buck boost IC, and the LM3671 to step down to 3.3 V.

2.4 Lights Block

The lights we used for our project were Alitove WS2812B RGB LEDs. We chose these lights for their low power consumption and versatile color options. We used 14 LEDs in our design with seven per ear. We chose this number simply because it was the amount that best fit around the edge of the ear based on the ear's required diameter. These LEDs consume about 9 W/m or 0.3 W per LED which totals to 4.2 W for our 14 LED strip per headset. During the testing process, we noticed that the LEDs would flicker when powered. After some research, we found that this is caused by the 5 V power input washing out the data input if the data is fed by a 3.3V source like from our microprocessor. The solution was to place a small resistance, we used 10 Ω resistor, between the 5 V connector on the PCB and the LED strip to reduce the power it received by about 1 V, enough for the lights to get a clear data signal while keeping them on.

The lights receive control signals from the GPIO 16 pin of the ESP32 module. The code controlling the lights takes advantage of the FastLED library [6]. This library offers support for the WS2812B lights and provides a simple interface for creating lighting effects. Our final demonstration included 15 solid colors, a random color selector, and five effects. Two effects, Rainbow and Trace, use built-in FastLED functions; but we wrote the code for Fill, Red White & Blue, and Circles. The lighting options can be seen in the app display in Figure 5. The code for controlling the lights is located in Appendix C. When the system is running, a command is given to set the lights. The LEDs maintain their color or pattern until a new command is received.

2.5 Control Block

The control block is responsible for the WiFi connectivity as well as controlling the lights. The main component is the ESP32 WROOM WiFi module. It was chosen for its low cost and power consumption, making it ideal for wearable applications. The ESP32 handles maintenance of a mesh network with nearby headsets and listens for lighting commands. It broadcasts commands it receives to the rest of the network so that all headsets are displaying the same color or pattern. To accomplish this, our program uses the PainlessMesh library, which is compatible with the Arduino IDE and can create an ad hoc mesh network [7].

The commands are received via a web server that the modules host. It was originally an HTML form with a text box that broadcast any text that was submitted. In the final design, the main web page displays a list of currently connected nodes, but it is still capable of receiving commands. The web page can be reached from the app or any device using the gateway IP address of the network, found in the WiFi settings of the device. The subnet is always created within the Class A range of private IPv4 addresses, so it does not have public internet access [8]. The master node receives instructions from that address and uses them in two ways. It sets the lights on its own headset, and it broadcasts the message to all other nodes, so they can

update their lights as well. We wrote the controlLights function to handle cases where the given headset is the master node and when it is receiving the message from another node. It was important to have the exact same code in every headset, so they would all function the same regardless of which was the master. The master node is chosen as the one with the lowest ESP32 chip ID. If a headset is added or removed, the network will reselect the master when necessary.

The overall communication relies on a set of opcodes. They are all lower-case letters, and each one corresponds to a different light or effect. The web page or app sends a letter, and the ESP32 module reacts accordingly. We chose this convention to make the data rate requirements as low as possible. The data portion of the packet is only an eight-bit character, reducing the data rate to 8 bps if one command is sent every second. This convention is not practical for a system with many color options, as would be the case with color wheel or slider for example. In that case, the data could be updated to full 24-bit RGB values for solid colors and a numbering system for lights. This would provide greater flexibility without increasing the required data rate significantly. Either way, it is still nowhere close to the limit on the ESP32, at about 150 Mbps for 802.11n [9].

The original design also included a battery indicator button in the control block. Due to issues with the fuel gauge chip, it was not used in the final prototype. Had the fuel gauge been functional, the code on the ESP32 would have used a MAX1704X library [10]. This would take a battery level reading, as a voltage and percentage, when the button was pressed. The percentage would have been translated to an appropriate display on the lights, in a manner described in our Design Document.

2.6 App Block

The Android app gives the user the ability to control the lights wirelessly. Once connected to the Magic Ears network, any light color or effect can be selected and will be displayed on all the headsets on the network. Since the mockup in the Design Document, we have consolidated several screens to simplify the design. The app now consists of a main menu, for connecting the app to the network, and a control screen, for changing the lights. The primary constraint for the app was that it must be able to interact with the ESP32s in the headsets, and therefore had to be designed around the way that the ESP32s could send and receive data.

To use the Magic Ears app, the Android phone must first be on the Magic Ears WiFi network. As seen in Figure 5, the first screen is the main menu, where the user should press the ‘connect’ button. This brings up the control screen, which features a web view box and all the light control buttons. The IP address of the current gateway is also displayed here. The web view box shows the list of nodes present on the network, which is a page hosted by the web server on the ESP32s. The light buttons send the command out to the headsets, triggering the lights to change to the desired color or effect. The node list is also updated any time a light button is pressed. One important note is that users will not have internet access on their phones while connected to the Magic Ears WiFi network. Ideally, users will join Magic Ears, set their lights how they want, and then reconnect to the regular WiFi network they were using previously.

The backend of the app handles connecting to the web server and sending out the light commands. The IP address of the current network is read and uses to build the URL for the web server. It is also displayed in the web view box. All of the control buttons are linked to URLs and commands. The URLs are similar to the original one used to access the web server. It uses the gateway IP address and adds the correct letter to be sent based on the chosen color or effect. This doesn't change what is shown on the web view, but sends out the command to the master headset, which broadcasts the command to the rest of the network. An example URL is shown below, in this case it is sending the command for the color red to the given gateway address.

http://10.163.97.1/?BROADCAST=a

Occasionally, the mesh network will need to update and renegotiate its structure. This can happen when adding or removing a headset, which may change the gateway address. Users will be unable to send a command, and the web view will show a 'Webpage not available' error. This can be fixed by backing out to the main menu and pressing the connect button again, which updates the IP address that the buttons are using. The final app was somewhat simplified from our original designs due to time constraints. It was not the most critical part of the project and we had no prior experience with app development. We focused on making the app functional within its requirements and relatively straightforward to use.

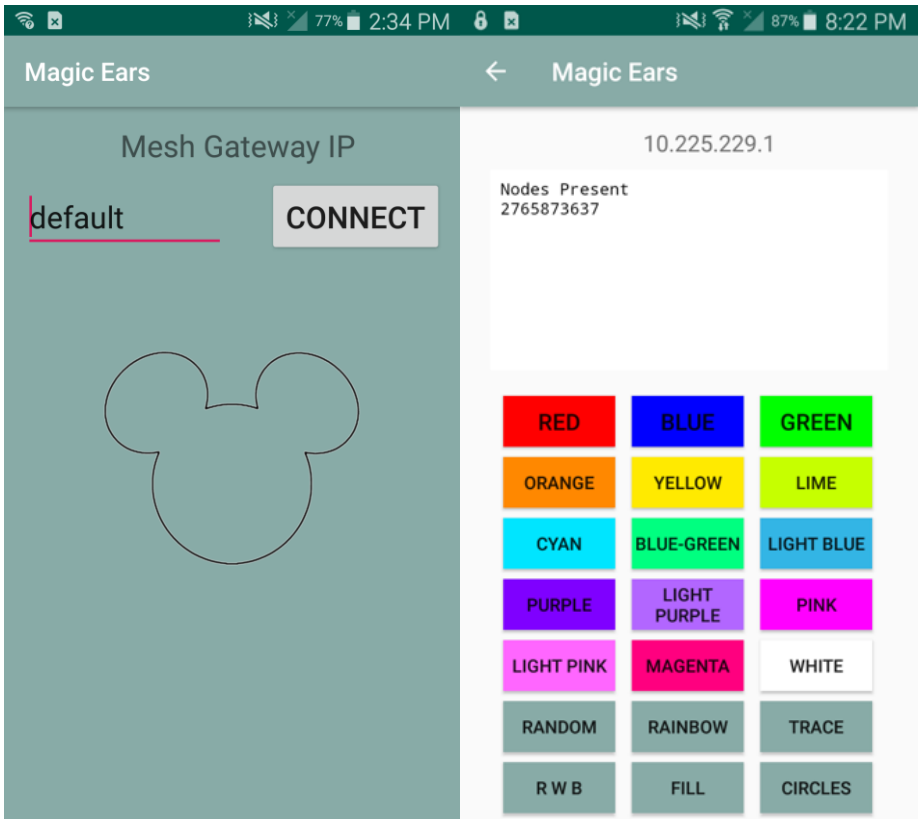

Figure 5. Final App User Interface

3 Cost and Schedule

3.1 Cost

We considered the individual cost to be the price we paid to assemble one of our prototypes. For some items, like the lights and building materials, this was a much larger quantity than needed. The bulk cost is the ideal cost to assemble a headset assuming an adjusted price for the exact quantities needed and bulk prices for electrical components.

Part Name	Part Number	Cost (individual)	Cost (bulk)
WiFi module	ESP-32 WROOM 32	$3.80	$3.80
LED Strip	ALITOVE 16.4ft WS2812B RGB LED Strips 150 Pixels	$25.99	$3.17
3D Printed Ears - 4 pieces per headset, 142 grams total	Translucent Clear PETG Filament 1.75 mm	$11.88	$2.81
Lithium Polymer Battery	Adafruit 328	$14.95	$14.95
PCB (PCBWay)		$7.80	$0.42
Battery Charger	MCP73831	$0.58	$0.43
Battery Indicator Subsystem	Maxim MAX17043	$9.95	$9.95
Voltage Regulator Subsystem	TI - TPS63060	$9.95	$9.95
Small components (button, switch, etc)		$10.00	$1.00
Headband	1 inch wide black headband	$1.67	$1.67
Building Materials (screws, felt, hot glue, etc)		$20.35	$1.11
Total		$116.92	$49.26

3.2 Schedule

Week	Kaitlin	Ian
2/25	Design document, research, and planning.	Design document, research, and planning.
3/4	Set up Arduino environment ESP32 testing using dev board and sample code. Confirmed viability of components.	Subsystem research and testing to make sure they could work with each other.
3/11	LED strip testing and debugging. Helped with PCB schematic and layout.	Created Overall schematic and PCB layout. Fixed problems with subsystem interconnections and part availability.
3/18 (Spring Break)	Wrote IPR. Started light demo with colors and some effects. Tested ESP32 range. Assembled and debugged first round PCB.	Wrote IPR.
3/25	Finished light effects and demo program. Helped fix issues and order second round PCB.	Created more first round PCBs for practice and any further testing/future use. Updated schematic and PCB layout for second round PCB order.
4/1	Assembled second round PCB. Defined light opcodes. Wrote code to integrate lights and WiFi controls for solid colors only. Started app development.	Assembled second round PCB's for use in final design. Soldered light strips on to use for testing effects and synchronization.
4/8	Investigated issue with flickering LEDs. Further app research. Worked on lights and WiFi code, including dynamic effects. Helped with plans for building prototype. Assembled two more PCBs.	Researched problem with lights flickering and added a resistor to the V+ input for the LED strips to add stability. Assembled two more PCBs, drilled holes into headbands, got materials together to create first prototype.
4/15	Prepared prototype and code for mock demo. Added 'alert' effect. Started preparing for final presentation and paper.	Created first round prototype for the mock demo. Made another PCB and two more prototypes for final demo.
4/22	Final demonstration. Presentation slides, writing final paper. Creating diagrams and graphics as needed. Finished app testing.	Final demonstration. Created presentation slides and gathered any graphics needed for it. Worked on final paper. Finished hardware verification.

4 Verification

4.1 Power Block

4.1.1 Charger

Verifying that the charger worked properly was a fairly simple process. First, we ran the battery down to around 3.7 V, the amount of charge left after running the LEDs for six hours, or a full day of use according to our requirements. The battery was measured with a voltmeter on the battery terminals to make sure we were starting at an almost drained battery. Then, the battery was connected to the PCB's charging circuit and left to charge using a 5 V wall outlet transformer. We measured and recorded the batteries voltage every half hour, making sure to remove the charger first to not skew the results. The battery took about four hours to charge, which met our requirement. The data we collected during the verification process can be seen in Figure 6.

4.1.2 Battery

Our project required that the battery be able to provide enough power to keep the headset functional for at least six hours of use. We confirmed this by first charging a headset to full charge and confirming this using a multimeter. It gave a reading of 4.15 V, close to the maximum voltage listed for the battery we used. Next, we left the headset running with the LEDs on and took measurements every half hour, which can be seen in Figure 6. The minimum voltage the battery can supply before it is considered dead is 2.8 V, so this was the threshold our battery had to stay above for six hours of use [11]. We found that after six hours, our battery still had 3.65 V which met our requirement and allowed for extra run time.

We did notice a few things while testing the longevity of the battery. The most notable observation we made was that after about two hours of use the LEDs began to flicker slightly. This flickering increased in severity as the battery drained, and by the end of six hours the headset was having a hard time holding a solid color. This could be due to the 5 V buck boost IC losing stability as the battery drops below a certain level, and a solution could be a less sensitive IC. However, this would require more testing beyond the scope of this project.

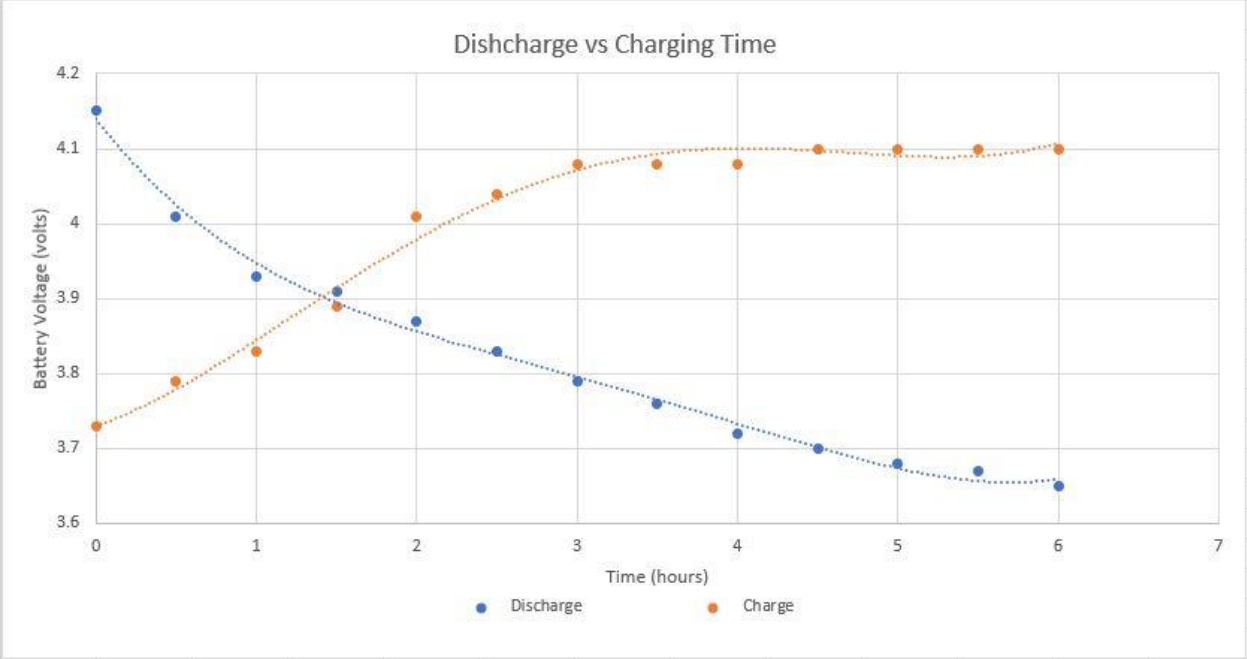

Figure 6. Discharging vs Charging Data

4.1.3 Switch

The verification of the switch was a very simple process. After soldering all the necessary parts to our PCB, we attached the switch and flipped it to the on position. We confirmed that the WiFi module and lights were receiving the right amount of power from the battery, and then flipped the switch to the off position. The boost circuit and module had indicator LEDs to show if they were in use and we used these to confirm they had no power when the switch was turned off. Further, with the switch still off, we plugged in a 5 V source to confirm that the battery could still charge with the switch in the off position.

4.1.4 Battery Indicator

The fuel gauge could not be verified due to the problems mentioned in Section 2.3.4. If we had the proper IC to use for our design, we do believe we could have met the requirements set for this part of the project. We had the right circuit layout and code available to test the fuel gauge with the module we were using. In the first iteration of the PCB we were having problems getting any readings, but we discovered that only specific GPIO pins on the module supported I2C, the protocol the IC used to send the fuel data [12]. We corrected this problem in the second iteration so that if a better pad layout was used it could be working correctly immediately.

4.1.5 Voltage Boost

Testing the requirements for the voltage boost overlapped with creating a constructed, functioning PCB. The LEDs require 5 V to operate which is the output voltage of the buck boost. If it was not working correctly the LEDs would not turn on. However, with all of the PCBs we made the lights turned on confirming it works. We verified this again by using a voltmeter on the input of the light strips to confirm they were receiving 5 V ± 10% as per the requirements, which they were.

4.2 Lights Block

A major issue with the lights that we were able to diagnose and begin to fix was an issue with LED flicker. We noticed this when testing first began with the LEDs and WiFi module dev kit. The LEDs at the end of the strip would constantly flicker and with certain light effects the flickering would become more severe. We originally thought that the data wire between the lights and PCB was unstable but after testing found no solution to the problem. Researching the issue more, we found that other people had this problem with WS2812B lights as well. The solution was not with the data wire but the 5 V power wire to the light strip. If data was sent from a 3.3 V source it could potentially be washed out by the power voltage coming into the strip and cause flickering. The answer was to add a resistor or diode in series with the light strip to reduce the voltage enough (about 1 V) so that it wouldn't interfere with the data coming in.

The other portion of the lights testing was writing the code used to control them. Initially, we wanted to determine the colors and effects that would be available in the app. We modified the DemoReel sample program from FastLED, which cycled through various light patterns. We tested solid colors first, which were 15 colors and an option to display a random RGB value. This required several iterations with different colors, because sometimes a color that looked good on a computer did not look good as an LED color. We also added the three new light effects, which also required some trial and error to create effects that were aesthetically pleasing. The code for the alert/distress effect was also written but not incorporated into the final design. However, this would be an easy feature to add. When finished, the Magic Ears light demo showcased all the lighting capabilities. This proved to be a modular way to test and verify the LED code before incorporating it with the rest of the project.

4.3 Control Block

The control block was testing in stages to ensure that it would function as intended. The first step was confirming that the ESP32 was the right choice, which was done using ESP32 WROOM development boards. We were able to test sample code provided by the PainlessMesh library; starting with a basic program that blinks the on-board LED once for each node on the network. Blinking twice proved that our two development boards were able to communicate, and we built up from there. The next stage used the web server functionality from the library, which allowed any node to host a simple web page. It uses an HTML form that submits any text entered into the text box. We confirmed via the serial monitor that the module would receive the text and broadcast it to the rest of the network.

The light control code and WiFi code were developed independently, but once they were functional they needed to be combined. We assumed this would be a challenge as the web server is asynchronous and the lights are controlled synchronously, but it went together better than expected. We assigned a unique lowercase letter to each color and effect and added a large case statement to handle all possibilities. When the PCBs were finished, we moved to testing the code on those and confirmed that everything still worked. We noticed that our boards could be programmed more quickly than the development boards, which would often fail and require several tries, while our boards mostly programmed successfully on the first attempt. The

initial testing was done using the web page but switched over to the app once that was almost finished. As of the final demonstration, the app was the primary way to control the lights. However, the original text box configuration can still be reached using the gateway's address, X.X.X.X/change, so the lights can still be controlled from a web page.

If the fuel gauge IC had been present and functioning in the final design, we would have needed code to handle that in the main program. When prompted by the button, the ESP32 would need to read the battery voltage or percentage from the fuel gauge via I2C. The reading would be translated into the corresponding color (green, yellow, or red) and number of LEDs to be lit and sent out as a command. This would override the most recent command being displayed on the lights at the time, acting as an interrupt signal.

One main issue with our first round PCB design involved the USB to UART CP2102N chip. We did not pay close enough attention to its connections and didn't fully understand them the first time. We intended to use it in its bus-powered configuration using the internal voltage regulator, as described on the data sheet [13]. This required the VIO pin to be set to 3.3 V, because the maximum voltage is limited by VIO + 2.5 V. By leaving VIO unconnected, the limit was 2.5 V, and it was being given 5 V when plugged in. When powered, the CP2102N heated up quickly and began smoking. The solution was to bridge the VIO and VDD pins together and make the connection schematically in the second-round layout design. This small fix allowed the chip to function as intended.

During the design process, we decided to redefine the data rate requirement. The initial number of 1 Mbit/s was based on capabilities of the chip and not what our actual system needed, which was much lower. We should have defined the requirement to be more specific to what was needed for our system. The actual data rate is very low, based on the light commands and the internal communication required to maintain the network that is handled by the library. Therefore, the data rate meets the requirements of the project even though the 1 Mbit/s suggested limit was never formally tested. The other requirement for the control module was the range over which two modules can remain connected. We found the average separation between the modules before they lost connection was 114.3 m, which was within our desired range. The procedure and data for this test can be found in Appendix B.

4.4 App Block

The requirements for the app involved its ability to control the lights while not introducing significant delay to the system. Originally it was planned to work on the user interface and backend components separately, but as development began it made more sense to work on both concurrently. It took several iterations to create a functional app, which involved learning about the functions and abilities available. The important components were the web view box and the URL builder that accessed the gateway of the phone's current WiFi network.

We ran into issues with constraining the buttons in the XML layout view. Even with the correct aspect ratio chosen, the app would look different in Android Studio than it would on the Galaxy S5 we used as a test phone. This made it difficult to place and constrain the buttons in a way

that they would display on the phone without overlapping each other. It would require further research and testing to redo the constraints properly. Ideally, the display should be universal and look about the same no matter the phone's screen size or resolution.

Testing both the functionality and the latency were somewhat dependent on the rest of the system working. However, once the rest of the project was working it was very straightforward to verify the app's performance. It was a natural progression of the design to confirm that the buttons in the app could control the lights on a headset, which they were able to do. In the event of the app not working, it is still possible to control the lights on a phone from a web browser on the Magic Ears WiFi network. Once the app was functional, we also tested the overall system latency. This was defined as the time delay between pressing a control button on the app and when the lights changed to that color. We took 10 measurements with the app and one headset and the average response time was 0.672 s. This was below our requirement of 2 s. Occasionally it would take a noticeable amount of time to change, but in most of the trials the lights changed almost instantaneously. In some cases, it was limited by human response time and how fast we could stop the timer once the lights had changed. When multiple headsets are connected, the master node will respond first. It causes some additional delay with several headsets as the command gets broadcast to the whole network, but the response time is still reasonable.

5 Conclusion

5.1 Ethical Considerations

This project was completed with ethical and safety issues in mind. We understand the IEEE Code of Ethics and accept the obligation to be honest and safe with our work [14]. With our project being used as headgear, we are aware of the well-being of the users. This is consistent with Code #1: “to hold paramount the safety, health, and welfare of the public…”. Additionally, this project was developed for personal use only. We have no intention of mass-producing or selling the headbands commercially, therefore we will not infringe on Disney’s copyright. In accordance with IEEE Code #6, “to maintain and improve our technical competence…”, we developed this project primarily as a learning experience to expand our technical abilities. It is imperative to be honest in reporting the data and results of all requirement testing. Data would be meaningless if it did not come from actual testing, and it would not be an accurate representation of our progress. Additionally, Code 7 of the IEEE Code of Ethics involves honest criticism of technical work, acknowledging errors, and properly crediting others. This is a standard that relates to the academic integrity policies of the University. Throughout the design process, we received feedback to improve our technical work.

Since our project uses batteries we acknowledge the risks involved with Lithium-Ion batteries. If too much power is drawn from the battery pack too quickly or the circuit is shorted, it can cause a spike in temperature beyond safe operating standards [15]. This can cause the battery pack to expand and possibly explode which could result in bodily harm. It has potential of starting an electrical or chemical fire, either of which can be dangerous and hard to extinguish. The electronics of the headset can also short and potentially cause health risks. As a result, safe operating procedures are essential. There are safeguards for the charging circuit to prevent the batteries from being overcharged, but the batteries must be used with caution by users who are well-educated on proper techniques. More rigorous safety testing should be done before Magic Ears are worn for an extended period, especially outside or in any weather conditions.

Any electronics using wireless communications must adhere to regulations set forth by the Federal Communications Commission (FCC). Since our headsets communicated via WiFi, they fall under Part 15 of the Electronic Code of Federal Regulations [16]. This states that private consumers with low power devices have the right to use WiFi frequencies near 2.4 GHz without the need to register the device with the FCC. Magic Ears create a closed network for controlling lights remotely. This means that in the event of a hack, the hacker would merely be able to change the color of the headsets and not be able to gain access to sensitive information, such as user location. The Association for Computing Machinery (ACM) Code of Ethics and Professional Conduct is also relevant because of the software components of the project. Specifically, ACM Code 2.9 requires that “systems are robust and usably secure” [17]. This relates to both hardware durability and the security of the WiFi communication. The whole system operates on a subnet within the range of private IP addresses. Because the devices do not have access to the public internet, hacking is a minimal concern. ACM Code 1.5 addresses respect for “the work required to produce new ideas, inventions, creative works, and computing artifacts.” We are incredibly grateful for the resources that people have developed and shared.

In our context these are the libraries and tutorials for using the technology found in our project. In the given time scale to develop our project, it would not have been possible without these resources.

5.2 Accomplishments and Results

Based on the amount of systems we were able to implement and have working we consider project a great success. We set out to make an aesthetically pleasing light up headset and we did just that. Almost all our requirements were met, and a few were exceeded, including two of the high-level requirements. The headset weighed nearly half the required weight amount and the PCB is smaller than we hoped, with room to shrink if the need arises. The final headset is almost fully functional except for the fuel gauge. This, however, is not critical to the function of the rest of the headset. It could be fixed with a pad layout that is easier to solder manually or through a process that can more precisely apply solder paste to a PCB, via robots for example.

Beyond this project's accomplishments, our team has learned a lot from this senior design project. Between the hardware and software components, there was a lot of new information that came at us that had to be researched first before we could begin to make a functioning product. We took it one step at a time and, with careful planning, developed useful skills that helped us complete this venture. We tackled problems and stepped out of our comfort zone, taking risks like trying to solder PCBs using a stencil for the first time, rather than soldering each piece individually. Although it was foreign to us, we trusted in our abilities and found that it was a much more efficient way to place parts and improved the quality of design, something we constantly strive to do. This example, among others, emphasize how we have furthered our expertise as engineers during this project.

5.3 Future Improvements

Given the modular nature of our project, it has a lot of room to evolve with better technology. Right now, LEDs are constantly getting better in design in terms of size and power consumption. Our team is happy with the lighting effects we were able to implement for our project, however, there is a lot of room to expand the grandeur of what the lights can do with smarter LEDs. Another area that could be improved is the 3D printed ears. 3D printing is a new technology that has a lot of room to grow and as new types of plastics become available for consumer use, our headset can benefit from these advances. For example, printing the broadside of the ear in a more opaque plastic would mask the PCB and battery, while keeping the edges clear to allow the lights to be seen.

Additionally, the app UI is an area that could be greatly improved. The layouts of the two screens should be made more universal and tested with different phones and resolutions. Our app was functional but could be expanded to be more user-friendly and provide more features in terms of controlling the lights. This would give users more flexibility to create a unique lighting experience. Since they are inspired by a Disney product, it would be interesting to wear them at a Disney park and gauge people's reactions to them.

5.4 Final Thoughts

This project proved to be an excellent learning experience. We grew as engineers and found a new appreciation for the standards of professionalism, project management, and documentation. It was satisfying to start something as an idea and have it evolve over time into a functional product. This was an opportunity to research topics we're interested in outside of class and learn in a different way. One important topic was soldering and all the techniques for doing that. With all skills, there is always room to improve and this project allowed us to do so. We were able to explore RGB LEDs, wireless communication, and hardware design; and combine them into an interesting project. This class was a great way to conclude our college careers, and Magic Ears are an embodiment of our hard work.

References

[1] "How Many People Visit Disney Parks Each Year?", *Disney News*, 2019. [Online]. Available: https://disneynews.us/disney-parks-attendance/. [Accessed: 30- Apr- 2019].

[2] E. Glover, "Disney California Adventure Park Guests Will ‘Glow with the Show’", *Disney Parks Blog*, 2012. [Online]. Available: https://disneyparks.disney.go.com/blog/2012/06/disney-california-adventure-park-guests-will-glow-with-the-show-at-disney-california-adventure-park/. [Accessed: 30- Apr- 2019].

[3] "Disney Mickey Mouse and Friends Light-Up Made With Magic Ear Hat", *Amazon.com*, 2019. [Online]. Available: https://www.amazon.com/Disney-Mickey-Mouse-Friends-Light-Up/dp/B01B26NXPE. [Accessed: 30- Apr- 2019].

[4] R. Niles, "Disney explains the 'Glow with the Show' Mickey hat technology", *Theme Park Insider*, 2019. [Online]. Available: https://www.themeparkinsider.com/flume/201208/3149/. [Accessed: 30- Apr- 2019].

[5] A. Spiess, "#192 Aldi Hack: Simplify SMD Soldering with a Cheap Aldi Oven. Nothing else", *YouTube*, 2019. [Online]. Available: https://www.youtube.com/watch?v=mjfnpjvw9jY. [Accessed: 30- Apr- 2019].

[6] "FastLED/FastLED", *GitHub*, 2019. [Online]. Available: https://github.com/FastLED/FastLED. [Accessed: 30- Apr- 2019].

[7] gmag11, "gmag11/painlessMesh", *GitHub*, 2019. [Online]. Available: https://github.com/gmag11/painlessMesh. [Accessed: 30- Apr- 2019].

[8]"IP address", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/IP_address#Private_addresses. [Accessed: 30- Apr- 2019].

[9] "ESP32-WROOM Datasheet", *Mouser.com*, 2019. [Online]. Available: https://www.mouser.com/datasheet/2/891/esp32-wroom-32_datasheet_en-1510934.pdf. [Accessed: 30- Apr- 2019].

[10] Porrey, "porrey/max1704x", *GitHub*, 2019. [Online]. Available: https://github.com/porrey/max1704x. [Accessed: 30- Apr- 2019].

[11] "Polymer Lithium-ion Battery Product Specification", *Cdn-shop.adafruit.com*, 2019. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/785060-2500mAh_specification_sheet.pdf. [Accessed: 30- Apr- 2019].

[12] "What pins are the I2C pins on the ESP32?", *Rntlab.com*, 2019. [Online]. Available: https://rntlab.com/question/what-pins-are-the-i2c-pins-on-the-esp32/. [Accessed: 30- Apr- 2019].

[13] "USBXpress Family CP2102N Data Sheet", *Silabs.com*, 2019. [Online]. Available: https://www.silabs.com/documents/public/data-sheets/cp2102n-datasheet.pdf. [Accessed: 30- Apr- 2019].

[14] "IEEE Code of Ethics", *Ieee.org*, 2019. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 30- Apr- 2019].

[15] "Lithium-ion battery", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Lithium-ion_battery. [Accessed: 30- Apr- 2019].

[16] "eCFR — Code of Federal Regulations", *Ecfr.gov*, 2019. [Online]. Available: https://www.ecfr.gov/cgi-bin/retrieveECFR?gp=&SID=752c648e754360a36c89c602d54ea9f9&mc=true&n=pt47.1.15&r=PART&ty=HTML#se47.1.15_1247. [Accessed: 30- Apr- 2019].

[17] "ACM Code of Ethics and Professional Conduct", *Acm.org*, 2019. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed: 30- Apr- 2019].

Appendix A Requirements and Verification Table

Table 1. System Requirements and Verification

Requirement	Verification	Verification Status
LiPo Charger		
1. LiPo battery fully charges to 4.2 V when supplied 4.2 V±1% at no more than 1.2 A.	1. A. Discharge battery completely. B. Recharge battery from output of the MCP73831 chip making sure voltage/current is not exceeded. C. When charge status pin goes high the charge cycle is complete, battery will be checked to make sure voltage is at 4.2 V.	1. Y
2. Battery charges within 7 to 10 hours.	2. During step 1.B, monitor time to make sure charging cycle is completed within desired time.	2. Y
3. While charging, temperature stays below 60°C	3. Also during step 1.B, monitor thermals with IR thermometer sensor to make sure IC does not heat up beyond 60°C.	3. Y
LiPo Battery		
1. The battery can provide up to 4.2 V at 2500 mAh for 6-7 hours, totaling at around 55 Wh.	1. A. Fully charge battery pack to 4.2V and connect to a load simulating LEDs and PCB. B. Drain battery for 7 hours with LED's as load. C. Confirm that the battery has more than 3 V of charge using a voltmeter.	1. Y
Power Switch		
1. Switch will toggle power to circuit on or off completely, as well as allowing the battery to charge properly.	1. Voltage measured using multimeter after the switch will read the battery voltage if the switch is on. If it is off then the voltage will read 0.	1. Y

Table 1 — continued from previous page

Requirement	Verification	Verification Status
Battery Indicator 1. Battery level must be accurate within ±5% of the available charge.	1. A. Test battery charge at any given amount greater than when it is dead. B. Plug battery into battery level indicator and test to see if the amount it is reading is within the given constraint.	1. N
2. LEDs can display the battery level in the manner described in procedure 1, accurate to the closest multiple of 7% rounded down.	2. A. Use code to display battery level on the LEDs. B. Count the number of LEDs to confirm that the displayed percent is consistent with the number of LEDs that should be lit according to the requirement.	2. N
Voltage Boost Regulator 1. Regulator provides a steady 5 V ± 10% output when supplied 4.2 V input.	1. A. Connect battery pack to the voltage regulator and confirm that the input is 4.2 V at 1000 mAh. B. Drain battery though regulator and measure output voltage with a multimeter making sure it stays within required range during discharge cycle.	1. Y
LED Strips 1. LED strip must consume at most 0.3 W ± 0.01% per LED or 9 W/m.	1. A. Connect lights to a power source at 5 V but not exceeding 6 V. B. Using a multimeter, measure the power consumption of a 1 meter segment of the strip. C. Confirm that each LED uses the required amount of power by dividing the total power consumed by the number of LEDs.	1. Y

Table 1 — continued from previous page

Requirement	Verification	Verification Status
Indicator Button		
1. Button can send a high signal when pressed.	1. Measure voltage to confirm that when button is not pressed a low signal is read and when pressed the voltage is read indicating an active high signal.	1. Y
WiFi Module		
1. WiFi module can receive data at a rate of 1 Mbit/s.	1. A. Establish a connection between two modules. B. Run test code that sends 1 Mbit of data, and start a timer when data starts sending. C. Stop timer when all data has been received, verify that it took 1 second or less.	1. See section 4.1.3
2. Two modules can maintain a connection up to 90 m apart.	2. A. Establish a connection around 1 m and confirm data can be sent both ways. B. Begin moving the modules apart, stopping every 30 m to confirm that the connection still works. C. Continue until the modules are at least 90 m apart.	2. Y
3. Microprocessor can accurately control the LED strips in both ears.	3. A. Run test code on microprocessor that cycles through lighting changes. B. Confirm that LEDs in both ears change according to the instructions in the test.	3. Y

Table 1 — continued from previous page

Requirement	Verification	Verification Status
Android App		
1. The app is able to connect to a headset and control its lights.	1. A. Use the connect button to join the network of a headset that is within 100 ft of the phone. B. Verify that the headset's chip ID is listed within the app. C. Use the app to change the color or light effect several times and confirm that it produces the correct result.	1. Y
2. The app does not introduce significant latency into the system, there should be no more than 2 seconds of delay.	2. A. Start a timer with precision of at least 100 ms when a solid color is selected on the app. B. Stop the timer when the lights on the connected ears change to the correct color.	2. Y

Appendix B Requirement Testing Data

WiFi Range Testing

One of the important requirements for the WiFi module is its reliable connection range. For testing, the 'startHere' sample code from the Painless Mesh library ran on two ESP32 WROOM development boards. The blue LEDs on each board blink twice (indicating two nodes on the network) when they are connected and blink once when they have lost connection. The testing was done with a clear line of sight and no interference from buildings. One module was left in place and the other was moved farther and farther away along a sidewalk. For the trial to meet the requirement, the modules had to maintain a connection at least 295 ft (90 m) apart.

When testing, a connection was first established with the modules next to each other and then we began moving them apart. In increments of 50 ft, the connection was checked and confirmed. The recorded maximum connection distance was the separation distance between the modules when they first lost connection. Sometimes walking a bit closer and allowing them to reconnect increased the range, but the measurement was maintained as the first dropped connection point. This data is displayed in Table 2. The first two trials were done with batteries that were not fully charged, which may have affected the range of the modules. The remaining eight trials were done with fully charged batteries. Based on 10 trials, the average maximum connection distance was 375 ft, which is within the expected range. Therefore, we concluded that the WiFi modules passed their range test requirements.

Table 2. WiFi Range Testing Results

Trial Number	Maximum connection distance (ft)	Maximum connection distance (m)	Within acceptable range?
1	225	68.6	No
2	325	99.1	Yes
3	450	137.1	Yes
4	350	106.7	Yes
5	500	152.4	Yes
6	450	137.1	Yes
7	350	106.7	Yes
8	450	137.1	Yes
9	350	106.7	Yes
10	300	91.4	Yes
Average	375	114.3	Yes

Appendix C Abbreviated Magic Ears Control Code

Note: Some functions have been shortened or removed in the interest of saving space.

```
//***************************************************************
// This program gets loaded onto the ESP32 chip of the
// MagicEars board. It is responsible for creating and maintaining
// the WiFi network between other nodes and any connected mobile
// devices. It also controls 14 WS2128B RGB LEDs based on the
// received character. MagicEars are currently capable of 16 colors
// and 6 lighting effects.
//
// The asynchronous web server portion of the code was based on
// the web server example provided by the PainlessMesh library.
// Several of the light effects were made possible with functions
// from the FastLED library.
//
// Written by Kaitlin Vlasaty and Ian Napp for ECE 445 Senior
// Design Spring 2019
//***************************************************************

#include "IPAddress.h"
#include <painlessMesh.h>
#include <painlessMeshSync.h>
#include <FastLED.h>
#include <stdlib.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

#define   MESH_PREFIX     "MagicEars"
#define   MESH_PASSWORD   "12345678"
#define   MESH_PORT       5555

#define DATA_PIN    16   // 2 for dev kit, 21 for MK1, 16 for MK2
#define LED_TYPE    WS2812B
#define COLOR_ORDER GRB
#define NUM_LEDS    14
CRGB leds[NUM_LEDS];
#define BRIGHTNESS          48
#define FRAMES_PER_SECOND  120
painlessMesh  mesh;
AsyncWebServer server(80);
IPAddress myIP(0,0,0,0);
IPAddress myAPIP(0,0,0,0);

// List of patterns to cycle through. Each pattern is defined as a separate function
PatternList patterns = {solid, rainbow, trace, rwb, fill, circles, alert};

uint8_t CurrentPatternNumber = 0;     // Index of current pattern number
uint8_t gHue = 0;                     // Rotating 'base color'
SimpleList<uint32_t> nodes;           // List of node chip IDs

void setup() {
```

```
  Serial.begin(115200);

  // tell FastLED about the LED strip configuration
  FastLED.addLeds<LED_TYPE,DATA_PIN,COLOR_ORDER>(leds,
NUM_LEDS).setCorrection(TypicalLEDStrip);
  FastLED.setBrightness(BRIGHTNESS);   // set master brightness control

  mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT);
  mesh.onReceive(&receivedCallback);

  mesh.setHostname(HOSTNAME);
  myAPIP = IPAddress(mesh.getAPIP());
  // Async webserver
  // Can be accessed from a control device to set the light colors from a web browser
rather than the app
  server.on("/change", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/html", "<form>RGB Value to Broadcast<br><input
type='text' name='BROADCAST'><br><br><input type='submit' value='Submit'></form>");
    if (request->hasArg("BROADCAST")){
      String msg = request->arg("BROADCAST");
      mesh.sendBroadcast(msg);
      char msg_char = msg[0];
      controlLights(msg_char);
      FastLED.show();
      FastLED.delay(100);
    }
  });

  // Web page to show nodes present, will also accept and process light commands
  server.on("/", HTTP_GET, [](AsyncWebServerRequest * request) {
  nodes = mesh.getNodeList();
  String message;
  SimpleList<uint32_t>::iterator node = nodes.begin();
    message = "Nodes Present\n";
    message += mesh.getNodeId();
    message += "\n";
    while (node != nodes.end()) {
      message += *node;
      message += "\n";
      node++;
    }
      request->send(200, "text/plain", message);
      if (request->hasArg("BROADCAST")){
      Serial.printf("Num nodes: %d\n", nodes.size()+1);
      String msg = request->arg("BROADCAST");
      mesh.sendBroadcast(msg);
      char msg_char = msg[0];
      controlLights(msg_char);
      FastLED.show();
      FastLED.delay(100);
    }
  });
  server.begin();
}
```

```
void loop() {
  mesh.update();

  patterns[CurrentPatternNumber]();         // Call the current pattern's function
once, which updates the lights data
  FastLED.show();                           // send the 'leds' array out to LED strip
  FastLED.delay(1000/FRAMES_PER_SECOND);    // insert a delay to control frame rate
  EVERY_N_MILLISECONDS( 20 ) { gHue++; }    // slowly cycle the "base color" through
the rainbow
}

// Function that gets called when the node receives a broadcast message from another
node. Mainly used to control the lights
void receivedCallback( const uint32_t &from, const String &msg) {
  Serial.printf("bridge: Received from %u msg=%s\n", from, msg.c_str());
  char msg_char = msg[0];
  controlLights(msg_char);
  FastLED.show();
  FastLED.delay(500);
}

// Function responsible for interpreting the control signal and changing the lights
void controlLights(char msg_char){
   switch (msg_char){
    case 'a':          setColor(255,0,0);             // Red
    case 'b':          setColor(255,128,0);           // Orange
    case 'c':          setColor(255,255,0);           // Yellow
    case 'd':          setColor(128,255,0);           // Yellow-Green
    case 'e':          setColor(0,255,0);             // Green
    case 'f':          setColor(0,255,128);           // Blue-Green
    case 'g':          setColor(0,55,255);            // Cyan
    case 'h':          setColor(0,128,255);           // Light Blue
    case 'i':          setColor(0,0,255);             // Blue
    case 'j':          setColor(127,0,255);           // Purple
    case 'k':          setColor(178,102,255);         // Light Purple
    case 'l':          setColor(255,0,255);           // Pink
    case 'm':          setColor(255,102,255);         // Light Pink
    case 'n':          setColor(255,0,127);           // Magenta
    case 'o':          setColor(255,255,255);         // White
    case 'p':          setColor(r_rand,g_rand,b_rand);// Random
    case 'q':          rainbow();                     // Rainbow
    case 'r':          trace();                       // Trace
    case 's':          rwb();                         // Red, White, and Blue
    case 't':          fill();                        // Fill effect
    case 'u':          circles();                     // Rainbow circles effect
    case 'v':          alert();                       // Alert light sequence
    default:           // Do nothing when unused character is sent
      }
}

void reset_lights(){
    for (int i = 0; i < NUM_LEDS; i++){   //turns all lights off
      leds[i] = CRGB::Black;
  }
}
```

```
void solid(){
  //do nothing
}

void rainbow(){
  fill_rainbow(leds, NUM_LEDS, gHue, 7);  // FastLED's built-in rainbow generator
}

void trace(){
  fadeToBlackBy( leds, NUM_LEDS, 20);     // a colored dot sweeping back and forth,
with fading trails
  int pos = beatsin16( 13, 0, NUM_LEDS-1 );
  leds[pos] += CHSV( gHue, 255, 192);
}

void rwb(){
  // sets lights in an alternating pattern of red, white, and blue
}

void fill(){
  fill_wave(leds, NUM_LEDS, &k);
}

void fill_wave(struct CRGB * arr, int numToFill, int * k){
  //turns all lights blue one at a time, then starts over
}

void circles(){
  fill_circles(leds, &curr_state);
}

void fill_circles(struct CRGB * arr, int * curr_state){
  //creates a circular rainbow effect, with colors rotating in both ears in synch
}

void alert(){
  set_alert(&l);
}

void set_alert(int * l){
  // Alert light effect, flashes on and off with red lights
}

void setColor(int r, int g, int b){         // Function to set all lights to the given
RGB value
  CurrentPatternNumber = 0;
  for (int m = 0; m < NUM_LEDS; m++){
    leds[m] = CRGB(r,g,b);
  }
}
```

Appendix D Layout and Schematics

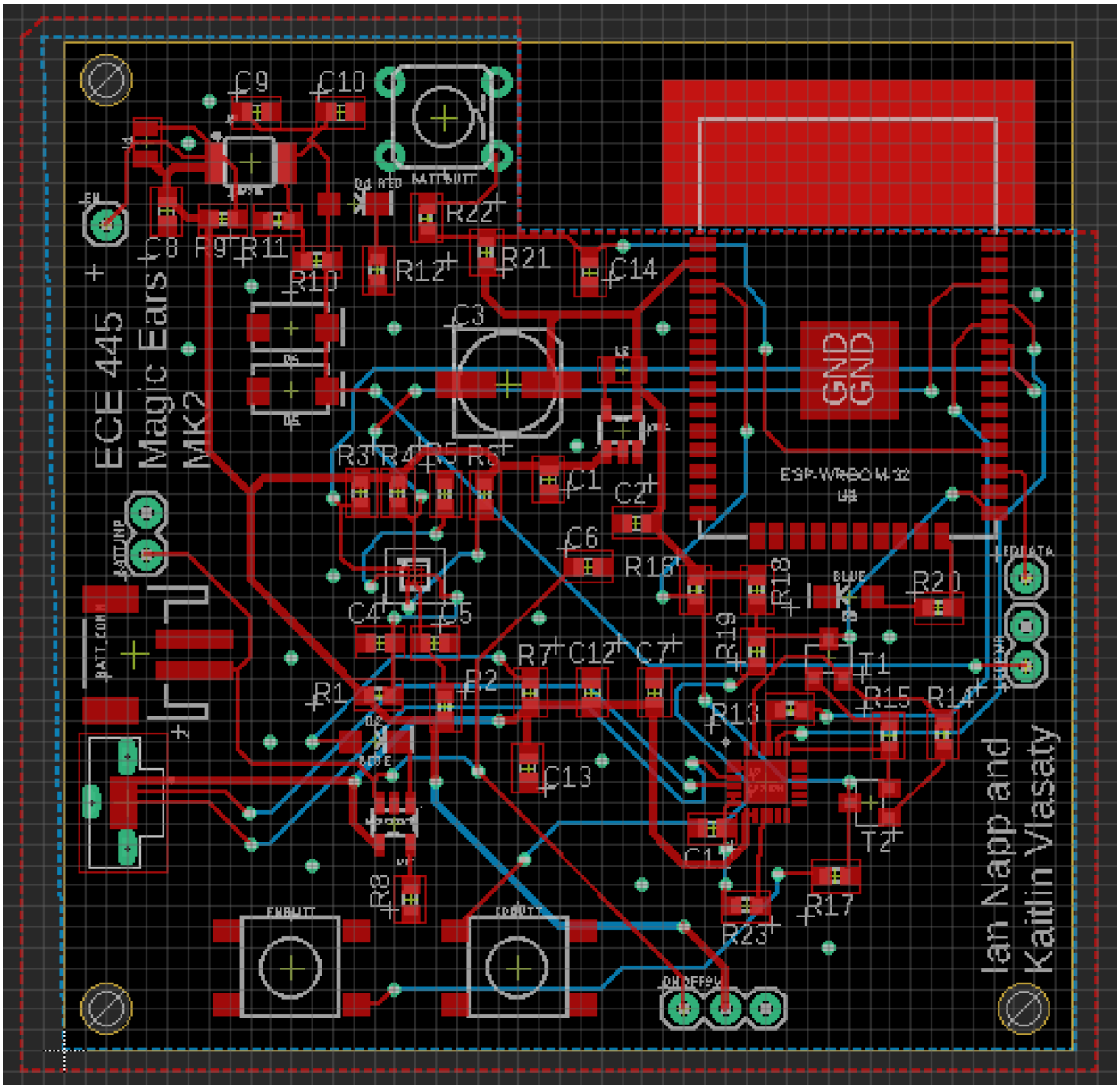

Figure 7. Final PCB Layout

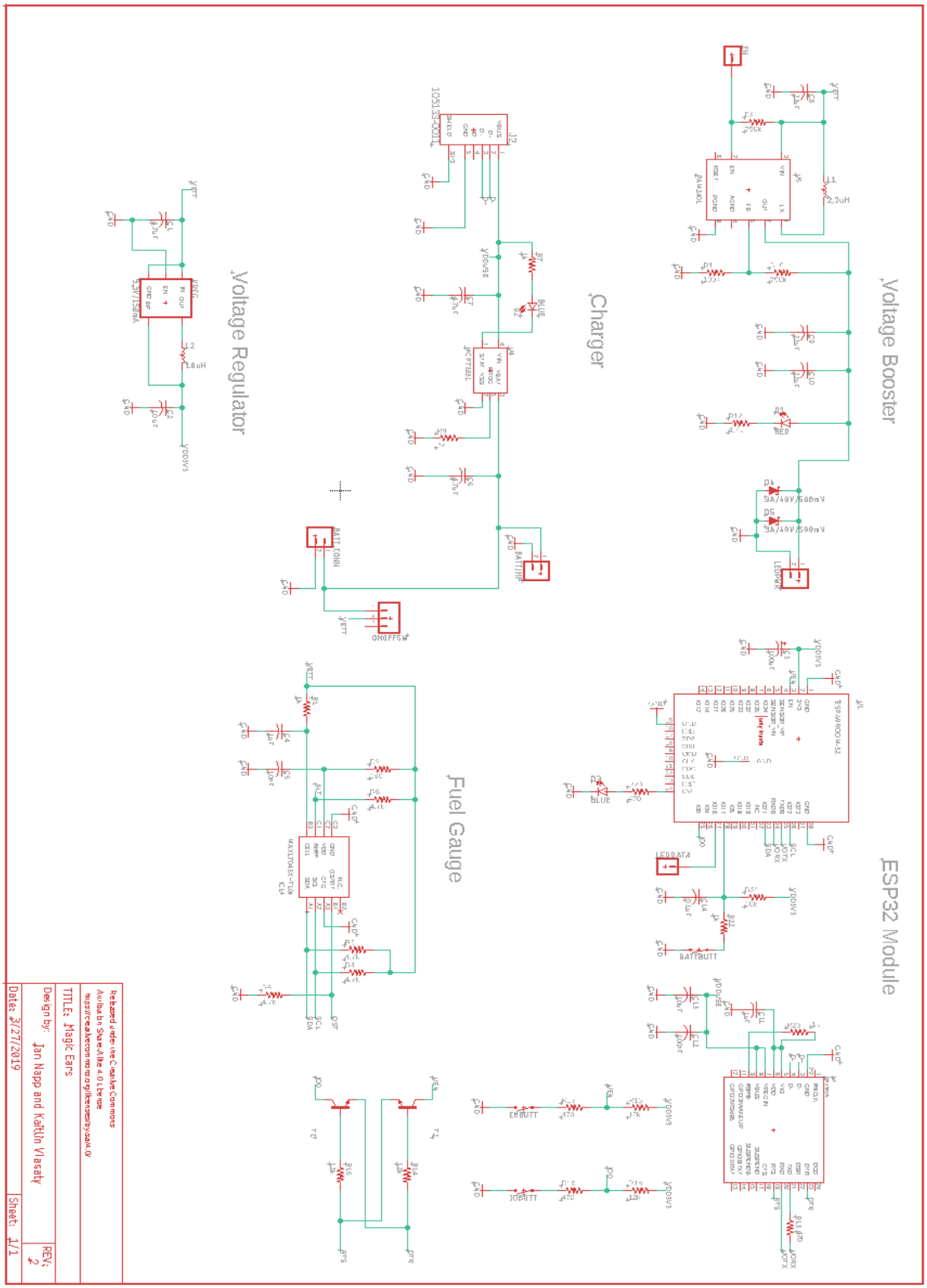

Figure 8. Final Schematic