Kettering University

College of Engineering

Department of Electrical and Computer Engineering

CE-490: Senior Design

David Foster

Final Project Report: Bluetooth Traffic Scanner

2021-03-22

Team Members

Jack Madsen

Zachary Herman

Maxwell Lagassa

Table of Contents

Introduction

Our Senior Design Project, created by Jack Madsen, Zachary Herman, and Maxwell Lagassa, is a system of devices that detect exposed bluetooth endpoints of other devices and using this information estimates how many people are in a certain area. This is done through a "Node" device which is battery operated and designed to be deployed anywhere with a wifi connection. Additionally, we utilized a "Base Station" which is designed to receive, process, and post information received from the node device to a local web server for anyone on the same network to see. This information is useful to individuals in a variety of situations. Ranging from being Covid aware to those wanting to use a non-busy rec center weight room, this device is designed to give an estimate of the number of people in an area to anyone who wishes to seek this information. The requirements for our project are divided into two groups, the requirements for the node, and the requirements for the base station. The requirements we are setting for our node microcontroller include the following:

1. Portable and useable without a wall outlet to supply power
2. Must be able to scan and detect open bluetooth signals within the area
3. Must be able to display relevant information to a screen for debugging purposes
4. Must be able to transmit information such as devices found with open bluetooth endpoints over wifi to the base station microcontroller

The requirements we decided upon for our base station microcontroller include the following:

1. Must be able to receive information over wifi from the node controller

2. Be able to process the information once received, and appropriately calculate an estimate for the population within an area.
3. Be able to host a local web server and post population estimates and other information to said web server.

Our group believes that these requirements, if achieved, will produce a final product that will solve our problem of providing population information to anyone who is seeking it for whatever reason they need.

Bill Of Materials

One of the most important aspects of building any sort of project is the ability to effectively find and source materials that fit the specifications of the project. The parts we sourced were as follows:

1. RPI3-MODBP, Raspberry Pi 3B+, Raspberry Pi Foundation
    - Vendor: Amazon.com Incorporated, Price per 100 units : $3,896.00
2. . 8541613031, 3.5 inch TFT touch screen display and case for Raspbery Pi 3B+, Jun-Electron
    - Vendor: Amazon.com Incorporated, Price per 100 units: $2895.00
3. KeeYees Development Board, Esp-32 wifi and bluetooth development board, KeeYees Electronics
    - Vendor: Amazon.com Incorporated, Price per 100 units: $749.00
4. B06ZZ8Y89F, 18650 rechargeable Lithium Ion Battery, Victagen
    - Vendor: Amazon.com Incorporated, Price per 100 units: $2799.00

5. A1012590US, 18650 Battery Shield, Diymore
    - Vendor: Amazon.com Incorporated, Price per 100 units: $799.00

Project Implementation

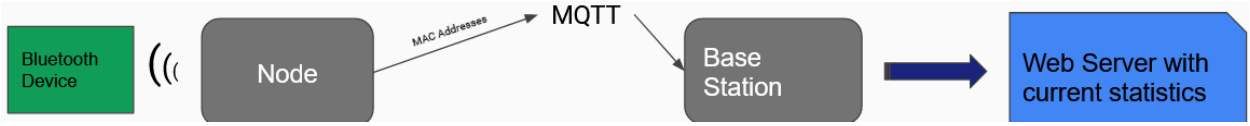
To accomplish the goals of this project various components were chosen to fulfill different roles that work together to produce our final product that achieves what we set out to. The specific parts and design choices we made to reach our end goal include the following.

In designing the architecture for our project it became clear that there would be two main components of our design: the Node and the Basestation. The largest computing demands came from the base station where it would be necessary to have it running several server applications to support our designs. The main server application to run on the base station was the MQTT broker that would be the backbone of the communication between devices. The MQTT broker of our choice was the freely available mosquitto server. Along with this we would also need to have the base station running a MQTT subscriber that would place the data from the node into a database. From here we needed the base station to have enough computing power to allow us to run a relatively basic http server to display our bluetooth data. With these applications in mind we chose to utilize the Raspberry Pi 3B+ due to the availability and performance it offered. On top of the Raspberry Pi 3B+ placed a touch screen to make programming easier and to display useful information about the MQTT data sent to the base station.

From here we moved our focus from the base station to the node setup. At the initial onset of our design phase it became apparent that we would need a microcontroller that had both bluetooth and wifi capabilities. Using the bluetooth capabilities we would scan the surrounding areas of bluetooth signals and use the number of devices found to make an educated guess on the

number of people in a given area. It was also important that it was a device platform that we were all familiar with. From these requirements we settled on the esp32 microcontroller as our node's microcontroller for its ease of programming in arduino and its availability. To power the esp32 while in the field we chose an 18650 battery and power delivery shield. This option was chosen due to its great power density and ease of rechargeability. An OLED screen was also implemented on the microcontroller to display helpful information such as the number of bluetooth devices found and the status of MQTT communications.

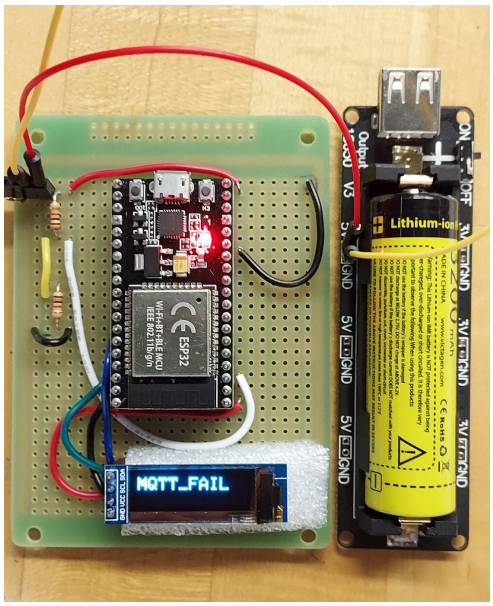
Product Specifications:



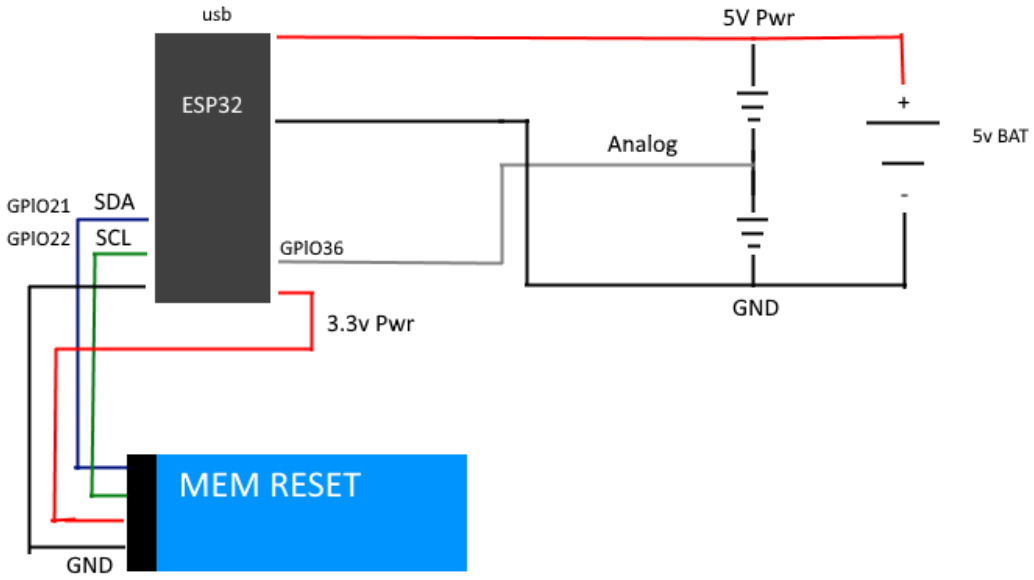
The range of the bluetooth scan is dependent on the surrounding environment but in an average sized classroom, every corner can be reached. The same can be said about the wifi connectivity, as long as there is a wifi connection within the room the node will connect. The battery life could last for days, especially if the OLED screen is turned, but it also can be directly connected to a usb charging cable thus it will never have to be recharged in such a case.

Hardware Utilized:

The above images show the Raspberry Pi 3B+ that was used as the base station in our product design. This particular model of Raspberry pi has had aftermarket heatsinks installed to allow for better cooling of the processor under the heavy server loads that we intended to run on it.

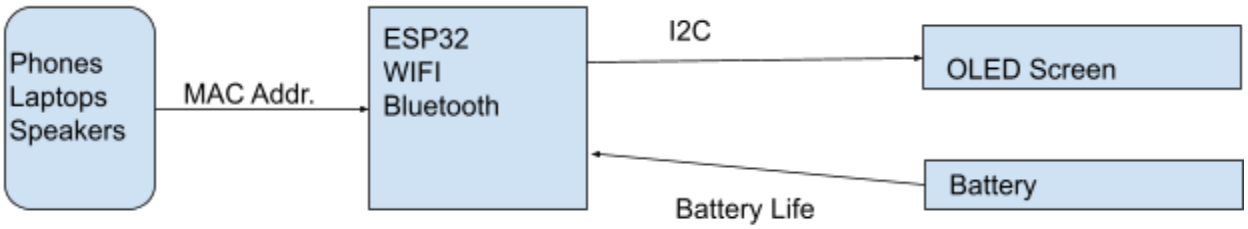


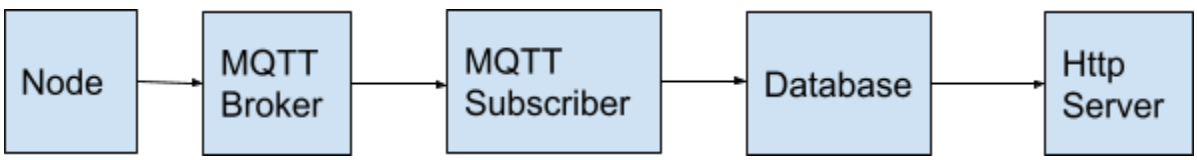
This is a finished node, all soldered together. The center board is the ESP32, powered by the battery slot next to it. Below is the I2C OLED screen.



This is the wiring schematic of the Node, and it includes the parts required to read battery life. The two resistors in the diagram do not specifically matter, as long as they are both equal to each other. Our node specifically has 1k resistors, but 10k and 5k will perform just the same.

Software Utilized:





The Above charts illustrate the software architecture that we chose for our project. The most important and interesting aspects of our project to note are that the MQTT broker, MQTT subscriber, and Http server are all run on the Raspberry Pi 3B+. Whie this is completely fine for our small use case, if this project were to be scaled to a commercial level with hundreds or even thousands of nodes it would be necessary to use a more power computing system such as a cloud service to accomplish the desired task.

Our project uses three main professional standards when it comes to programming. We have implemented MQTT for server communication between our nodes and the server host. We use the standard Wi-Fi protocol to connect our nodes and server host to the internet from any location in the school buildings. The standard Bluetooth protocol is used to communicate wirelessly with surrounding devices and to read their MAC addresses to count new devices. Lastly, our project uses I2C to connect the note to its OLED screen for simple message displays on the nodes.

This project is useful within our world today due to the unfortunate circumstances that require individuals to be aware of others around them to evaluate what risk they are putting themselves and others in. With Covid-19 being a very influential factor in our daily lives our project can give a student, or anyone else, an idea of how many people they would be exposed to if they were to try and study in the library, or if they were to try and use the Kettering weight room. Information pertaining to knowing the population of an area has plenty of use cases and proves to be something that would be useful to our daily lives, which is why our group decided to pursue this project. Even outside of Covid related use cases, individuals just have preferences on whether or not they would like to be around a bunch of other people while trying to study or

relax. Our product provides these people with information that could help them make a more informed decision, before they even leave their residence.

Communications Protocol

The method of information transportation we are using to communicate data between our Node and Base Station microcontrollers is done utilizing the MQTT protocol. MQTT, standing for Message Queuing Telemetry Transport, is a lightweight protocol designed for a publish and subscribe network of information specialized for IoT device use. We utilized this protocol to transport each detected MAC address from the Node microcontroller to the Base Station and from there the data is analyzed and posted to a locally hosted web server. After every MAC address is detected within a set interval of time, these MAC addresses are then followed by an integer representing the total amount of bluetooth signals found by the node. This trailing integer acts as a delimiter to separate the different packages of MAC addresses. Our transmission of data follows a relatively simple format, the list of MAC addresses will come in on a regular interval after their detection and following the burst of addresses a single integer between 1 and 3 digits in length will be sent to indicate the total number of detected devices. In order to utilize this protocol the following static variables are required. The SSID, username, and password for the wifi network are required, as well as the MQTT local server ip address, MQTT username and password, as well as the topic being published or subscribed to. In order to utilize this method of communication an active wifi network connection on the same network is required to connect the interfacing devices and to transport information.

This protocol for transmitting and receiving requires two devices, and a broker (which can be one of the devices). When one device wants to communicate any data, they will post the

information to what's known as a “Topic”. A topic is comparable to a TV channel, where only the other devices that are connected to this topic will be able to receive and read the information that is posted there. These computers that are looking to receive this information from a topic are known as subscribers under MQTT standard, and those machines looking to send data to other devices are known as publishers. Within our project, the publisher would be the Node computer, and the subscriber would be the base station. As previously mentioned a broker is required to facilitate the transfer of information from one device to another and this process is hosted on our base station microcontroller. This interaction allows for a smooth communication between two devices on the same network and allows us to communicate the discovered device information from one device to another.

The benefits of this technology over others is that it is specifically designed for IoT devices to transmit small amounts of data over a local network from one device to another efficiently and reliably. MQTT has four essential benefits when it comes to evaluating its strengths. Firstly the data that is transmitted over MQTT protocol is packet agnostic, meaning that it does not matter what type of data is attempting to be transmitted as long as the receiving device knows how to interpret the information that is being sent. Secondly, this protocol has multiple quality of service options that are available that can be utilized to reliably send data over a network without the fear that it will be lost to the ether and gone forever, with the proper configuration whatever is sent will be received by the subscribing devices. Next, this type of communication with a publish and subscribe model is massively scalable in terms of IoT. With the ability to simply post information to a topic and have any device look for that information and choose to receive it, any number of devices can be used to send and receive data between one another. Finally, this type of communication is majorly robust in terms of requiring a current

connection to function. One or more of the devices could disconnect from the network or the broker for a good amount of time and then reconnect and very simply continue receiving and sending data as if they never left. For all of these reasons we chose MQTT to be our main transport protocol for communication between devices for our project.

Validation and Testing

The testing process during development of our project was tedious and methodical, as we would retest most, if not all of, the components whenever new code was added or edited. This didn't necessarily slow us down, rather it made sure that each and every part of the project was compatible with each other and did not consume more space than we had available on the ESP32, or we were able to send to our remote server. For a generalized step by step process of how we tested components during a change, we'd start with:

1. Testing the physical hardware
    a. Check OLED output
    b. Operating alone off of battery power
    c. Check Wifi scan and connectivity
    d. Check Bluetooth Scan
    e. Reset button does work
2. Test the software on the ESP32
    a. Make sure it was reading a fairly accurate number of devices in the area
    b. Make sure it was attempting to connect to the server (if the server was not currently running)

    c. Make sure the ESP32 is able to send information such as MAC addresses to the Server

3. Test the software of the Server and Host
    a. Able to accurately count bluetooth MAC addresses without duplicating results
    b. Able to read messages from the Node

Testing our final product was difficult due to the unknown IoT devices that are placed throughout Kettering's Academic Building that could interfere with our estimate of the population within an area. Nonetheless, we tested the capabilities of our project in as contained an environment as we could achieve within the bounds of Kettering's campus. The main topic that is to be examined when attempting to test the capabilities of our project is the ESP32's capability to scan for exposed Bluetooth Low Energy devices that are within its range of detection. We confirmed that this scanning functionality was working as intended on multiple occasions by sitting in a room together as a team and analyzing the devices and amount of devices detected. We did this by simultaneously turning on and off our bluetooth devices we brought with us to try and influence the number of devices that were detected.

Since it isn't uncommon for individuals to have either no bluetooth devices activated, or several active at once, we needed to perform testing that would give us an idea of how to create a formula that would help minimize the amount of general error these scenarios could cause when we are trying to estimate the population of an area based on these signals. To give us a standard idea of what this formula would look like we ran a couple tests within the Senior Design lab, taking readings of how many devices were active during a normal class period where people were using things such as their laptop or smartphone, and then manually counting how many

people could be accounted for from those device readings. From this analysis we were able to come to the conclusion that the ratio of open bluetooth devices and actual population of a room hovered somewhere around 3 devices per every 2 people, thus, creating a very simple equation where (population) = ⅔ * (bluetooth devices found). Of course this will never be exactly correct at determining the population of a given area, so for our estimated population we suggest an estimated range of people who are in a certain area. For example if we detect that there are 30 devices within a room or area, we would expect to see between 18-25 people when they are counted in person.

The final form of our project meets all requirements provided to us from the Project Description for this assignment. Our project utilizes two microcontrollers, the ESP32 and a Raspberry Pi 3b+, our microcontrollers have bidirectional communication capabilities utilizing MQTT and wifi connectivity. Each of our devices utilizes MQTT as previously stated, and the ESP32 utilizes bluetooth and wifi connectivity as well, and the Raspberry Pi also displays metrics of our project on a locally hosted web server. These attributes of our project satisfy the requirements for multiple embedded devices, bi-directional communication for each, and the requirement for two types of I/O components for each microcontroller. In addition to this our Node microcontroller, utilizing the ESP32, runs on battery power so that it can be deployed to any environment, satisfying the last requirement for one of our devices to run on battery power alone. From this testing procedure, and the fact that our project satisfies all requirements set out for this assignment, we have concluded that our project is satisfactory in terms of the outlines that we have been provided.

Process Implementation

Our project was primarily broken down into four major sections, software development on both the Node and the Base station, and the physical design, including the 3d printed case, of the final product of our node and base station computers. Each of these components obviously overlap into each of the other components, but these sections are how we thought most logically broke the project into manageable pieces.

Jack's responsibilities pertaining to this project included the research and software development of the node microcontroller and each of its technologies. Aside from the research and development of how to implement the oled screen attached to our node, Jack developed every other component of the node computer. These technologies include the program that cyclicly scans for bluetooth low energy devices within the area, data collection and storage storage, wifi connectivity, MQTT communication publishing technologies, and configuring troubleshooting memos to the oled screen. To summarize the total amount of research done for these portions throughout the project, an estimated time of 10 hours has been spent researching background information on how to implement each part. In terms of actually developing and implementing these technologies an estimated time of 15 hours was spent on coding, troubleshooting, and implementing these different modules of our project.

Max's responsibilities focused mainly on the implementation of the base station device software. Since the base station was mostly ready for use on arrival barring software installations, most of the focus was on the software development and configuration. Starting with the research phase of the project, Max spent about 10-12 hours researching software capabilities to be implemented on the project. Furthermore, around 13-16 hours were spent in the software development process configuring the MQTT server, designing and programming the MQTT

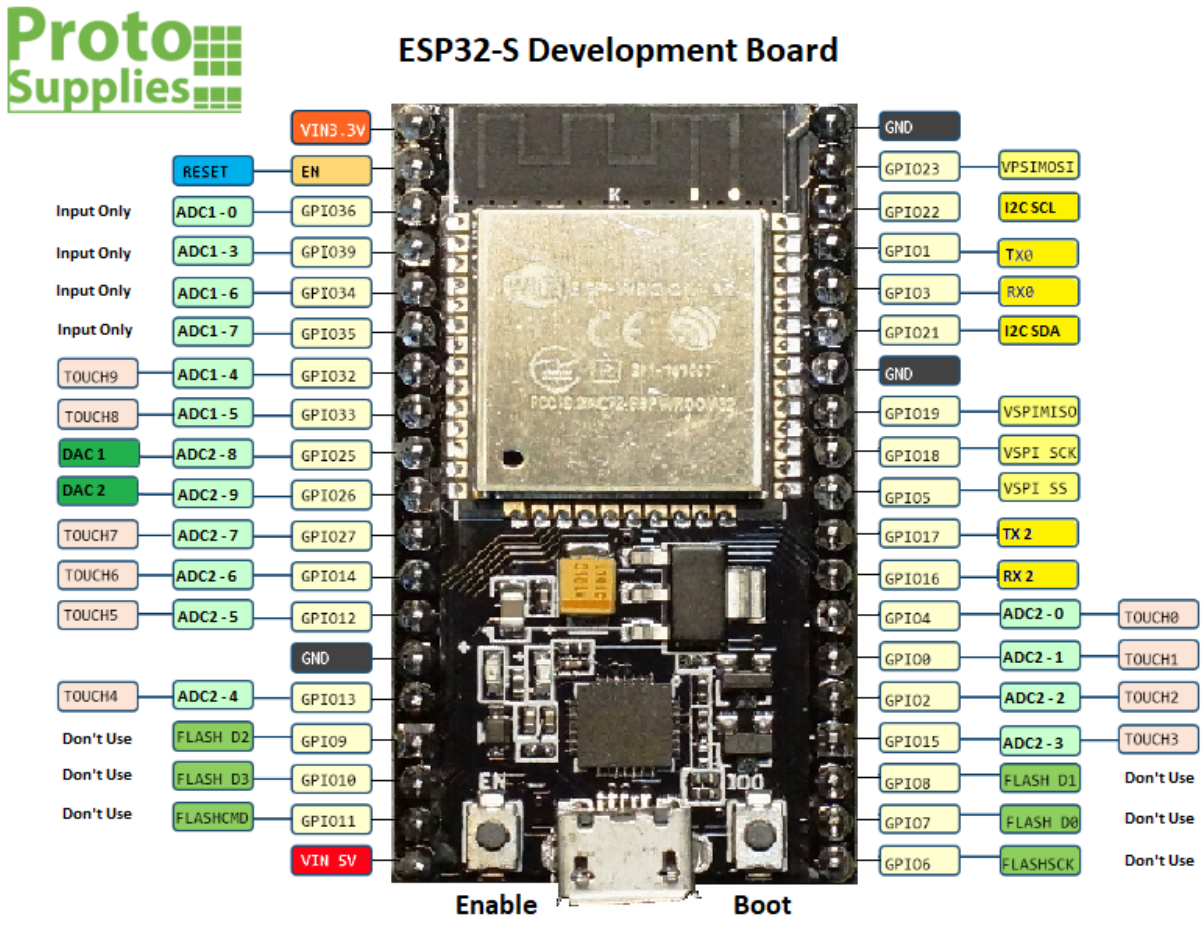
subscriber/database schema, and finally programming the local area network http server. Along with this Max also spent about 2-3 hours designing and 3D printing the housing in which the node device was kept for our demonstration.

Zach's responsibilities included many of the miscellaneous tasks of this project. These include working on the OLED screen, reading battery life, and soldering together all the components to a node. The OLED screen had to be outputting useful information to anyone looking at the node, this information includes whether there was a memory reset of the node, or how many devices that were read in the last scan. Reading battery life required two resistors to read the voltage coming off of the battery, the input to the ESP32 reading this incoming voltage being analog. Lastly, the soldering of all the parts together required a prototype be put together on a solderless board, following various diagrams for the individual components, then putting it all together under a single schematic.

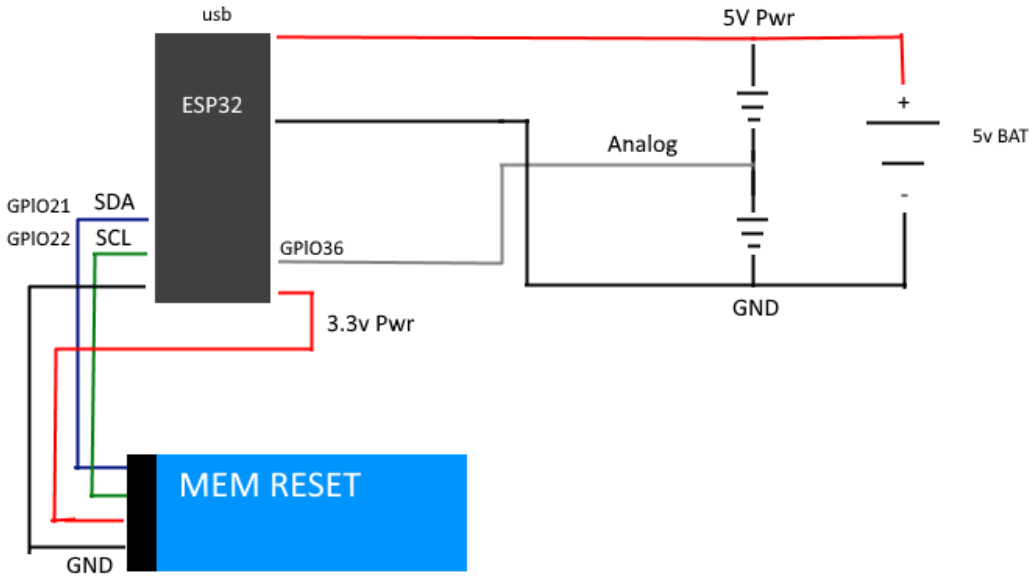
Due to circumstances out of our control, our group did not physically meet in class for the first five weeks of this semester. Sixth week we met to collect our parts that had been delivered. From there, we've met at least twice, if not three times a week to program and wire the project together, spending anywhere between 4 to 8 hours a week assembling the project. For the times we didn't meet in person, and for the first five weeks of this semester, we met virtually to discuss our plans, our progress, or to assist each other in whatever part of the project we may have been stuck on. This brings our total "man-hour-time" to an estimated 30 hours from each member equalling around 90 hours of total work during our regularly scheduled meeting times throughout the term.

Appendices and References

ESP32 Pinout Diagram (used for Reference)



Found at this page: https://protosupplies.com/wp-content/uploads/2019/04/ESP32-S-Pinout.png



This is the wiring schematic of the Node, and it includes the parts required to read battery life. The two resistors in the diagram do not specifically matter, as long as they are both equal to each other. Our node specifically has 1k resistors, but 10k and 5k will perform just the same.

For I2C functionality:

https://randomnerdtutorials.com/esp32-esp8266-i2c-lcd-arduino-ide/

For OLED functionality:

https://randomnerdtutorials.com/esp32-ssd1306-oled-display-arduino-ide/

For Battery life readings:

https://dr-mntn.net/2021/07/power-up-the-esp32-cam-on-the-input-voltage-pin-5v

Bluetooth Low Energy scanning reference project:

https://github.com/espressif/arduino-esp32/blob/master/libraries/BLE/src/BLEScan.h

MQTT Arduino publishing and subscribe reference:

https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/