Blind Me With SciEEnce

Senior Design Final Report

Caitlin Gruis
Enrick Hinlo
Christopher Ravasio

Table of Contents

1 Introduction

This report outlines the design process, testing, construction, and finished product of Blind Me With SciEEnce's final Senior Design project, which is a cumulation of the Electrical Engineering undergraduate experience and allowed the team to utilize the knowledge and skills acquired the past four years at the University of Notre Dame. The group consisted of Caitlin Gruis, Enrick Hinlo, and Christopher Ravasio.

Waking up, particularly on a weekday morning to get ready for school or work, is a difficult task that many people do not enjoy. The room is dark with the blinds closed, making it difficult to roll out of bed and begin the day. As they wake up and begin getting ready for their day, there is one significant question that can dictate how a person will dress, consider transportation, or address various home maintenance issues such as home temperature or blinds orientation: "What is the weather like outside?" Overall, positively influencing someone's morning routine and addressing some of the issues previously stated can help their day start off on the right note.

Additionally, people spend time in the morning and evening opening and closing the blinds in their house. Many modern homes can have numerous windows or be situated in high or hard to reach places, making it difficult or impractical to adjust them. Furthermore, when people are away from their houses for long periods of time, such as on vacation, leaving the blinds permanently in either the open or closed position is an indicator to burglars that no one is home. Lastly, leaving blinds open during the day can allow the sun to heat a room past the thermostat set temperature, resulting in higher energy cooling costs. Conversely, leaving blinds closed early

in the morning can lead to a missed opportunity for the home to be more efficiently heated at the beginning of the day.

Our project seeks to make the morning experience enjoyable and stress-free for the user by providing solutions to the problems listed above. The final product provides the user with accurate, real-time weather data, assists the user in getting out of bed due to the blinds illuminating the room with natural sunlight, and automates the user's home by rotating based on user-preferences. The blinds operate via a rotational motor and will open or close at a specific position manually set by the user. Furthermore, the user can put all of the blinds in automatic mode, which sets the position based on the amount of incoming sunlight. The solar charged weather station placed outside the home will take various data readings throughout the day, analyze the corresponding data, and notify the user of the weather. It is equipped with various sensors needed to gather pertinent weather data including light, temperature, and humidity. The user-interface is provided in the form of an iOS mobile application, making the entire experience quite user-friendly. The blind controls and weather station interfaces are easily accessible and make for a much more pleasant and prepared morning. With this setup, problems with forgetting to adjust blinds, having windows in hard to reach places, or adjusting blinds while away at work or on vacation can be solved.

Overall, our final design met our expectations. We were able to achieve all of the main goals we set out to complete. These goals were feeding weather data to a user in an app, using that app to remotely open or close a set of blinds, and allowing the blinds to open or close based on lighting conditions read by the weather station. Probably the largest component of our initial design we were unable to incorporate was opening or closing the blinds in order to regulate

indoor temperature based on an indoor temperature sensor. We found it difficult to program a hierarchy regarding whether temperature or luminosity should take precedence in whether the blinds are in an open or closed state. Despite this alteration to our design, the product still primarily functions as we imagined it would in the conceptual stage. Additionally, the accuracy of our systems also fell within our expectations, especially the temperature and luminosity sensors. If there was one element of the design that we found less accurate or reliable, it was our MQTT wireless connection.

2 Detailed System Requirements

The final overall system requirements are driven by the following:

First and foremost, the blinds system must be operated both automatically and manually. This requires the mobile app subsystem to accommodate this requirement as well. The automatic mode involves responding to changes in outside light levels and opening or closing accordingly. The weather station should wake up from sleep-mode periodically, sense the current light reading, publish that via MQTT to a specified topic, and the phone should subscribe to said topic and rotate based on various light cases preset in the code. The manual mode involves sending user-specified angles to set the exact position of the blinds through a servo motor. The mobile application would need some sort of slider tool or other tool that can quickly set and change specific values for the blinds to read. If multiple blinds were to be installed in the house, both the manual and automatic modes would be needed to accommodate this feature as well. Various parameters and cases in the code would need to be written to publish and rotate through various topics and motors.

Next, to be able to rotate the blinds effectively, a mechanical system consisting of a servo motor that provides sufficient torque to rotate the blinds, fits appropriately within the blinds frame, and doesn't require excessive power consumption is necessary for the system is necessary. A standard size, generic high torque motor would be a sufficient solution for this requirement. It's speed, torque rating, and size make it an optimal choice. A servo motor that only rotates 180 degrees is also a necessary component of this requirement so as to not destroy the blinds when the hex rod is being rotated. A servo-spline shaft coupler provides the connection between the servo and hex rod. Other mechanical requirements for our project include a sturdy and suitable wooden frame to demonstrate our product and a transparent enclosure for our outdoor weather station. These would need to be specifically designed to hold all components and provide enough air-flow and light for proper sensor readings.

In order to update the user to current local weather status, a weather station is needed that provides the user with real-time weather data, including temperature, barometric pressure, light, and humidity. The application includes logic that makes sense of some of the raw data like lux level, and uses it to tell the user whether it is sunny, cloudy, night-time, etc. The station should be placed outside near the window associated with it, and requires an appropriate protective covering. This covering needs to be mostly clear to let light in and have holes to get accurate humidity readings. To avoid excessive power use and be able to run off a battery, the weather station, in particular the sensors, needs to operate at a low power consumption and send only periodic updates to the user. When the sensors are not active, the entire system enters sleep mode to conserve power.

Because the outdoor board runs off a battery and we would also like to market the product as an efficient and cost-saving device, an energy efficient powering system is needed. Parts for the outside board need to operate at a low voltage and current consumption conducive to a LiPo battery and solar charging set-up. This will be achieved by only using components that require 3.3 volts. The solar charger will consist of a solar panel and an associated circuit that regulates the charging current into the battery from the panel. This battery will be fed into a DC to DC converter that maintains the output voltage at a constant 3.3 volts used to power the system. The indoor board needs to be able to supply both 5 and 3.3 volts due to the requirements of the ESP module and the servo motor. A wall wart supplies power from an electrical outlet, and is fed into a DC to DC converter to achieve both of these voltages for different components.

A user interface in the form of a mobile phone application that allows the user to access all other systems is also essential to the project. The phone application will use the publish/subscribe features of the MQTT protocol to communicate between different subsystems. The application will have the ability to manually rotate the blinds based on user control, as well as be able to put the blinds in automatic mode to respond to changes in the light sensor. The app will have another section devoted to the weather station, and needs to be able to subscribe to data coming in from the light, temperature, pressure, and humidity sensors. Finally, the app should include a feature that allows the user to add multiple blinds that they can control, as well as change the current weather station that it is connected to.

The last system requirement is a wireless interface based on MQTT protocol through the ESP module that provides smooth communication between devices. MQTT protocol is a publish-subscribe protocol that is able to send and receive messages from a server. The server

that our project must connect to is an online Amazon Web Server. The ESP and the mobile phone application both need to be able to connect to this server, as well as publish and subscribe messages through it successfully. It must consist of multiple topics under the the top level topic “sciEEnce/”, and each of the topics must have unique names in order to avoid collisions when communicating.

3 Detailed Project Description

3.1 System Theory of Operation

The overall system involves a user and two controllers. Two ESP-12(E) microcontrollers operate a weather system station and a blinds and motor system respectively. The weather station system is located outside the house, and takes current weather data using temperature, light, humidity and pressure sensors. This information is sent back to the phone application that the user is interacting with to inform the user of the current weather. The blinds and motor system is located inside the house, and autonomously rotates the blinds based on the level of light seen by the weather station. The servo motor is controlled via a PWM signal from the board. On the user facing side of the system, there is the phone application. The phone application is receiving information from the weather station about the current weather, and it will also allow the user to manually control the rotation of the blinds or put them in automatic mode. The app automatically brings up Wi-Fi access points the first time it is used to let the user pick the network they want to connect to, and brings up the menu again anytime it is disconnected. The application also allows the user to add additional blinds or weather station systems to create a system of automated blinds if they so desire.

3.2 System Block Diagram

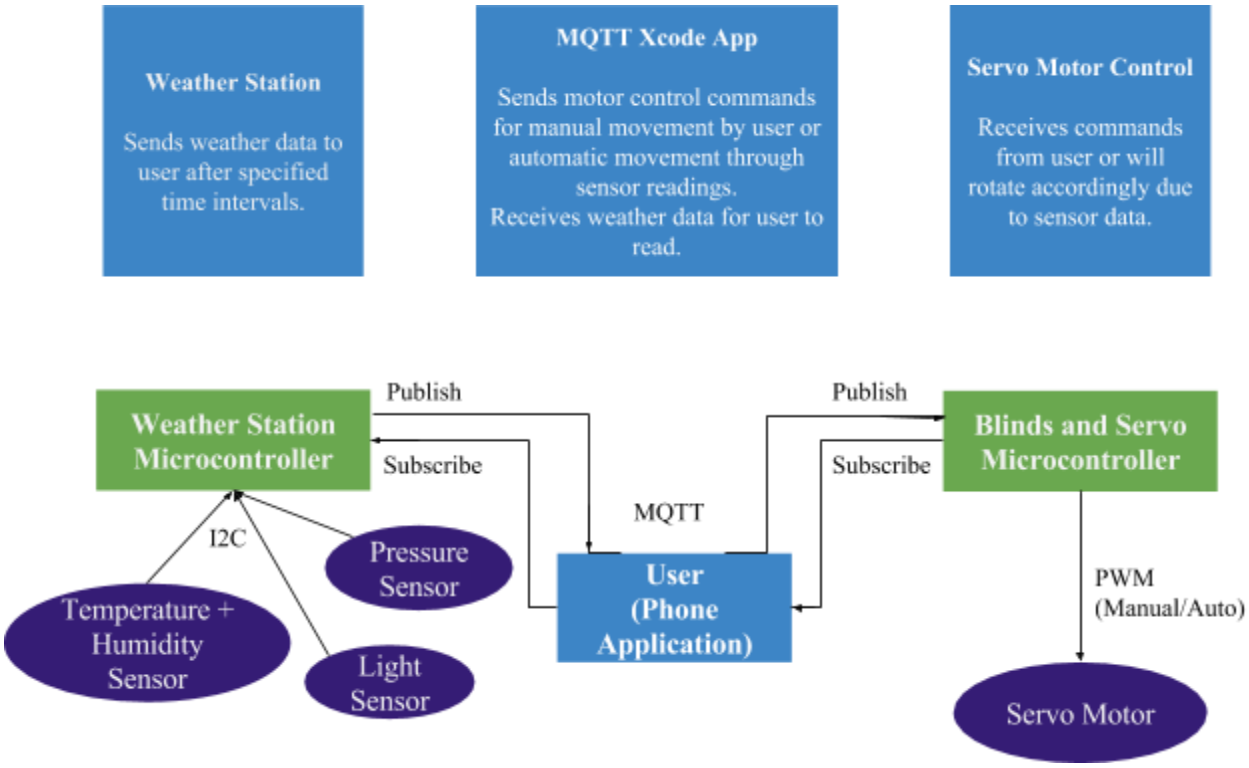

Figure 1. Overall System Block Diagram

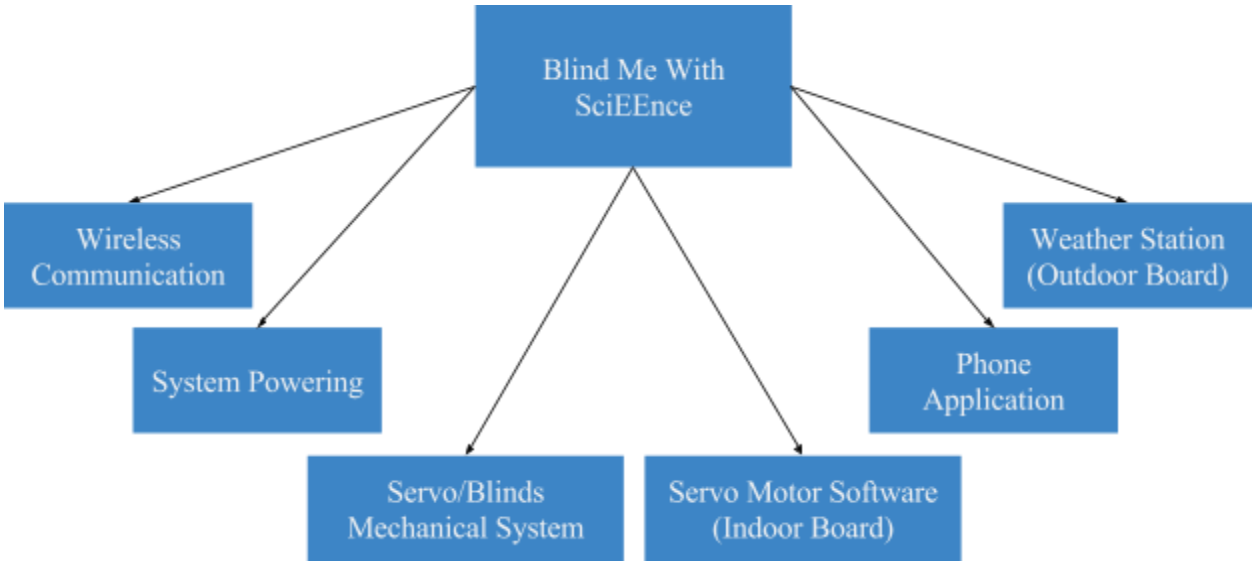

Figure 2. Subsystem Diagram

3.3 Detailed Design/Operation of Weather Station (Outdoor Board)

Subsystem Requirements and Schematic Explanations

To get weather information for our phone app and control the blind system based on light levels, we needed to design a circuit board that would be mounted outdoors and serve as a weather station. In order to obtain accurate weather data, sensors that measure temperature, barometric pressure, luminosity, and humidity were the major components needed for the board. Sensors were researched, and eventually chosen, based on the criteria of being reasonably priced and easily accessible when taking into account the budget of the project and the price of the final product. The associated schematics for these sensors can be seen below in Figures 3, 4, and 5 along with their associated part number. Looking at the schematics, we can see that each sensor is connected to Vcc to provide power and has this signal brought to ground through decoupling capacitors. All three sensors operate at low currents and 3V - 3.6V, so that they consume minimal power and can run on the same supply voltage as the ESP chip.

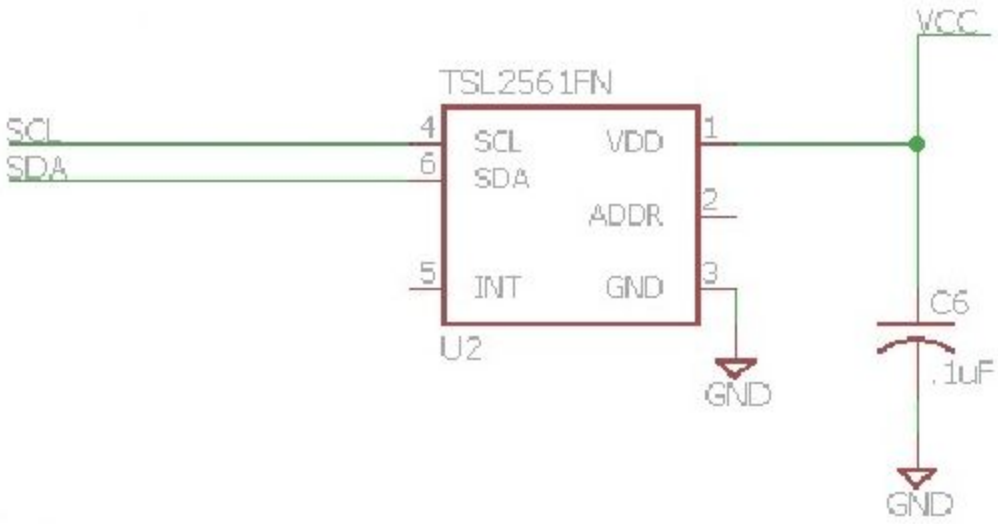

Figure 3. Light Sensor Schematic

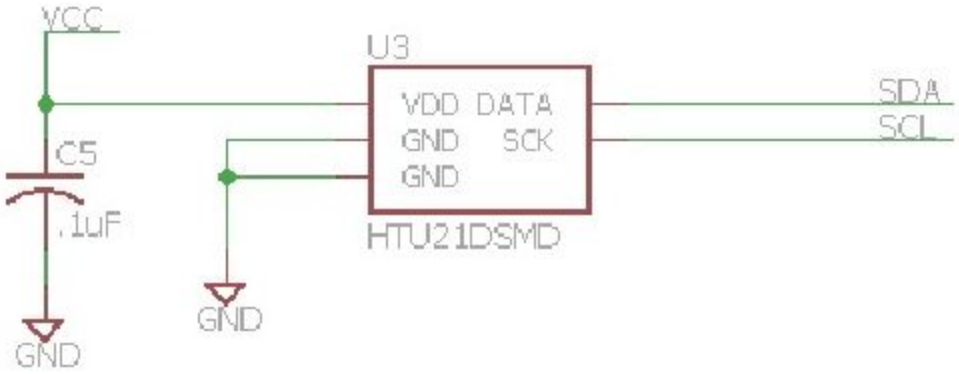

Figure 4. Humidity and Temperature Sensor Schematic

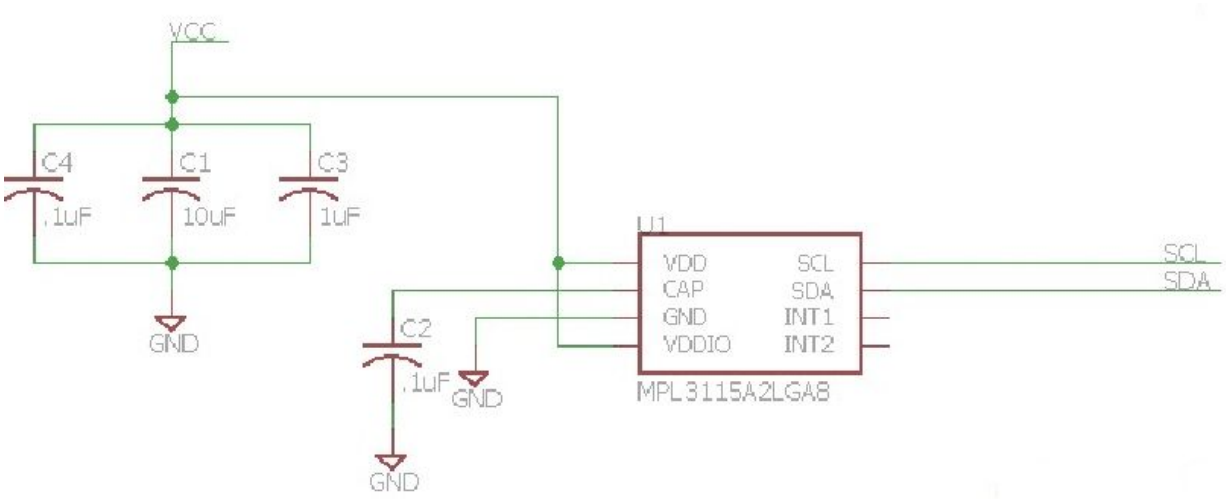

Figure 5. Barometric Pressure Sensor Schematic

Each sensor needed to have I2C capability due to the limited number of I/O pins on the ESP microcontroller. I2C was a good choice for this, since it only requires two signal lines (SCL and SDA). Further, multiple devices can be connected in parallel to the same two pins. Given these capabilities and our previous experience with I2C, we decided this would be the best method of transmitting data. Each SDA and SCL line is clearly marked on each sensor's schematic. Looking below at Figure 6, we can see the pins on the ESP to which the SCL and SDA lines are connected.

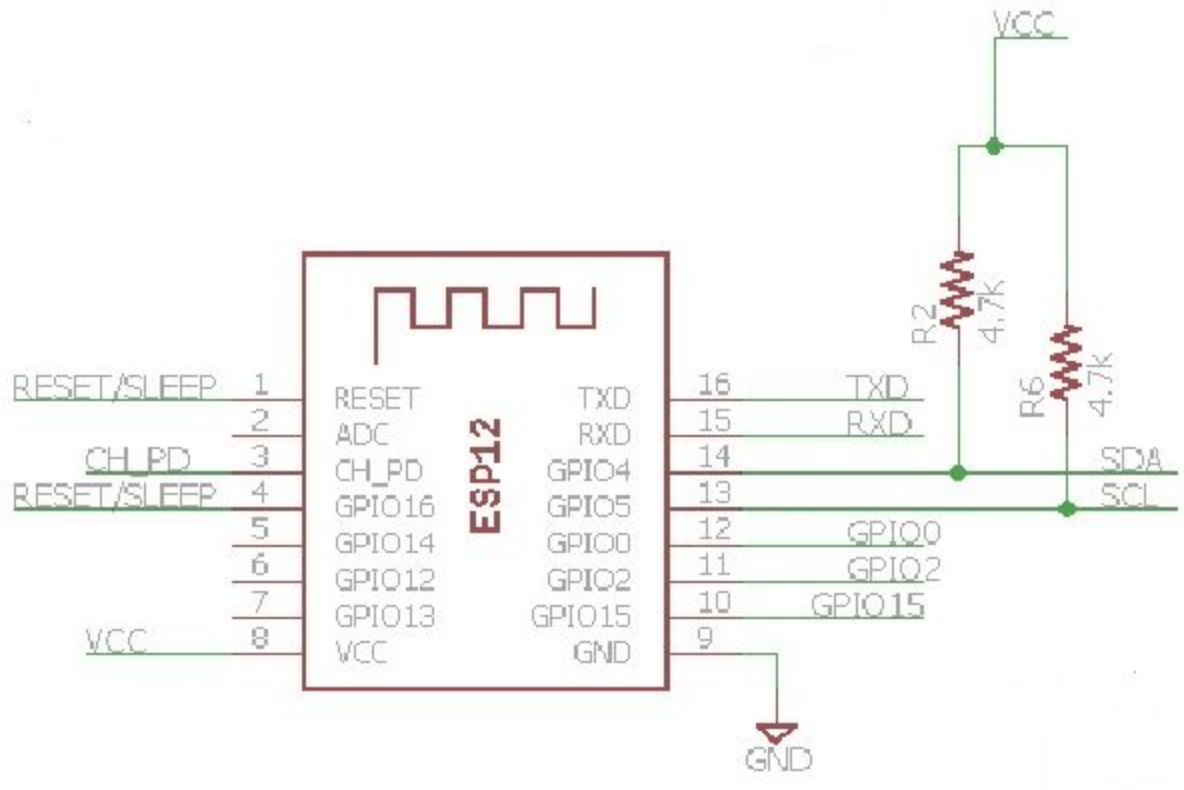

Figure 6. Weather Station ESP Schematic

The SDA line from each sensor is connected to the same node and fed into GPIO4 on the ESP, while the SCL lines are connected to GPIO5. Each of these pins are also connected to Vcc through pull-up resistors. Each sensor connects using an open drain connection to the bus, meaning that the load at the sensor itself is either open circuit or ground. This ensures that there aren't any electrical signal conflicts between sensors, since no device could assert a high signal while any other one is asserting a low. A high signal occurs when the sensor goes open circuit. Thus the pull-up resistors are present so that when none of the sensors are asserting a low they are all pulled up to a logical high signal.

To prevent damage from outside elements, the final board was encased in a protective covering that allows for accurate sensor readings. We purchased a plastic, clear case from

digikey that we then drilled several small holes into to allow for more accurate humidity and temperature readings. We also used a dremel to cut a larger hole into the casing to allow our solar panel cord to feed into the case and connect to the board. Details on the solar panel are covered further in the powering subsystem. The casing and solar panel were then placed in a simple bird feeder, so that the weather station could be easily hung in a tree by the user.

Software Flow

The figure below shows the software flow for the weather station code. The code was programmed in C in the Arduino IDE, and uploaded to our final outdoor board design.

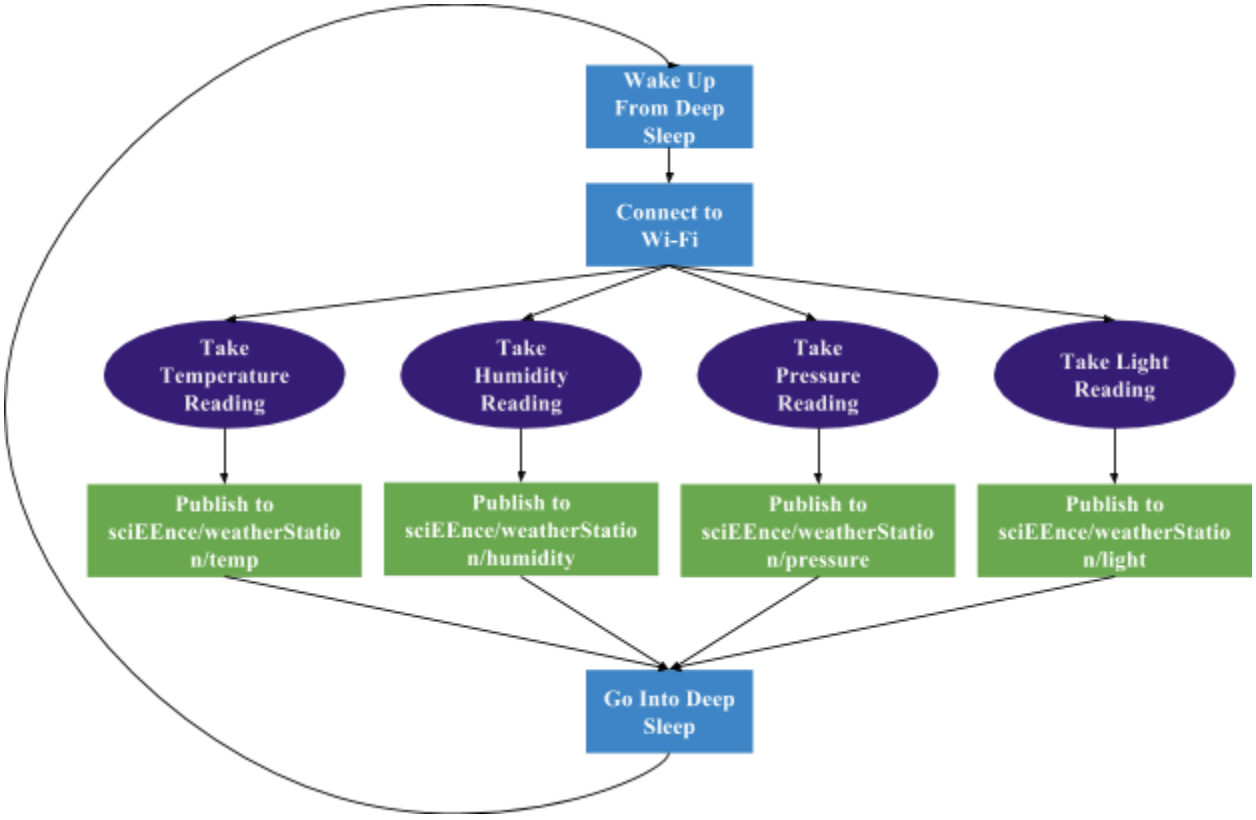

Figure 7. Weather Station Software Flow

When the board turns on, it attempts to connect to Wi-Fi. If it can connect to its last known settings, it does so. If it cannot, it creates an access point that the user can connect to in order to configure the Wi-Fi and connect to an appropriate connection.

Once the Wi-Fi is connected, the code grabs data from each of the weather sensors: temperature and humidity from the HTU21D sensor, barometric pressure from the MPL3115A sensor, and light from the TSL2561 sensor. This data is obtained from an I2C line by calling each of the respective sensors addresses. After the data is obtained, it is published to the MQTT server. Each type of data has its own topic named after it that it publishes to.

After the data is published, the command *ESP.deepSleep(30*1000000)* is run. This puts the ESP to sleep for 30 seconds, in order to save battery power while data is not being taken. After 30 seconds, the ESP wakes up, and the process begins again.

Subsystem Testing

The testing for this system occurred very gradually over a long course of time. The earliest testing was done on individual breakout boards for the ESP and sensors. As each breakout board arrived for the sensor we planned to use in the final design, we would test its functionality and feasibility immediately by trying to get it to work with an ESP breakout board. We ran very simple code that received the data from the sensor and printed the serial output to the user in real time. In this manner we made sure that each sensor worked as we expected it to and with the board. Through this process we learned that the original light sensor we chose was not as sensitive to light as we desired, and that we would also be better served with a different model that also had I2C capability. Once we verified that the sensors worked individually, we

connected all three sensor's breakout boards to the ESP breakout, duplicating the circuits that we would be implementing on the final board design. In this way we tested and verified that all of our sensors would work together properly and not cause problems when integrated as a single system. We found that data could be read simultaneously from all three sensors.

Testing this subsystem on the final board was probably the most arduous process of our entire project. When we powered up and ran our code on the outdoor board for the first time, we found that none of the weather sensors were giving us data readings. The ESP was unable to find the sensors even when given the exact address to access their data from. We tested a variety of possible errors. We verified that our code was fine by again testing with breakout boards. When they worked we reexamined our board layout. We found that the GPIO4 and GPIO5 pins were switched on the actual ESP hardware as compared to where they are in the Eagle library package. But even after correcting this in our code by switching the signal lines with each other, the sensors still didn't work. Deciding it was a hardware issue on our board, we cut traces into our board and isolated the sensors from each other. After testing a variety of combinations, we found that the pressure sensor was failing and causing the other two sensors to fail as well. We hypothesize that the pressure sensors we ordered were unable to withstand the heat of soldering and broke during this process. We were able to salvage the subsystem and regain full capabilities by taking the pressure sensor from our breakout board and soldering it to our outdoor board.

3.4 Detailed Design/Operation of Servo and Blinds Mechanical System

Subsystem Requirements

In designing the mechanical system of the product, a standard size servo was necessary to not only fit in the blinds mount, but also provide enough torque to turn the entire blinds contraption. While a continuous servo matched in size and rotational power, it was very difficult, in terms of software, to control its speed and precision. Therefore we decided to stick with the standard high torque servo due to these specifications and its ability to be powered by 5V. Mounting brackets and adhesive were needed to keep the servo in place as it performed rotational movements. To meet this requirement we attached two wooden pieces that came with the blinds and sanded it down to an appropriate size for the blinds casing. We then applied velcro adhesive to the mount and servo in order to make a secure connection and prevent the servo from moving as it rotated the hex rod. A connection, in the form of a shaft coupler, was required to connect both the hex rod of the blinds and the spline of the servo. At first we thought it would be necessary to create a custom 3D printed servo-spline attachment. However, after searching online, we were able to find a custom servo part that matched the spline specifications of the servo and the shape of the hex rod. We also acquired reasonably priced and accessible blinds so as to stay within the project budget. We based the sizing of our servo, mount, and PCB on the specifications of the purchased blinds. If this project were to be refined, a custom set of blinds would need to be created for robustness, stability, and aesthetics. Lastly, a wooden frame was constructed to accommodate the blinds for demonstrational purposes. Multiple 1x6 and 2x4 planks were used to create a frame that was sturdy. For aesthetics, the frame was stained and

glossed for demo day. The picture below displays our final mechanical system with the servo attached to the hex rod and secured by the wooden mount, the PCB fitting snugly into the frame, and the wooden frame encompassing the entire mechanical system.

Figure 8. Servo and Blinds Mechanical System

Schematic Explanation

There were only a few connections that needed to be made for the mechanical portion of the blinds to work. We decided to implement a molex connection due to the secure connection it provides between the servo and the board. Each servo comes with only three wires that need to be connected: power, ground, and signal. Since each servo requires a minimum of 5V, the power pin was connected directly to the voltage input of our 5V regulated wall wart plug-in. To send signals and exact servo angles, we used GPIO13 since it was an available pin on the ESP 12 module.

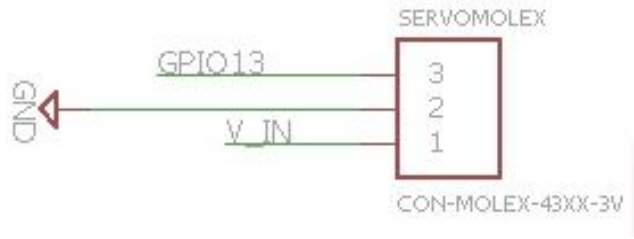

Figure 9. Servo Motor Molex Connector Schematic

As you can see below, we made connections from the molex connection to the GPIO13 pin of the ESP 12 module. As mentioned previously, power for the servo comes from the 5V regulated wall wart plug-in directly. Overall the connections necessary for effectively powering the motor were quite simple and most of the controlling came from the software.

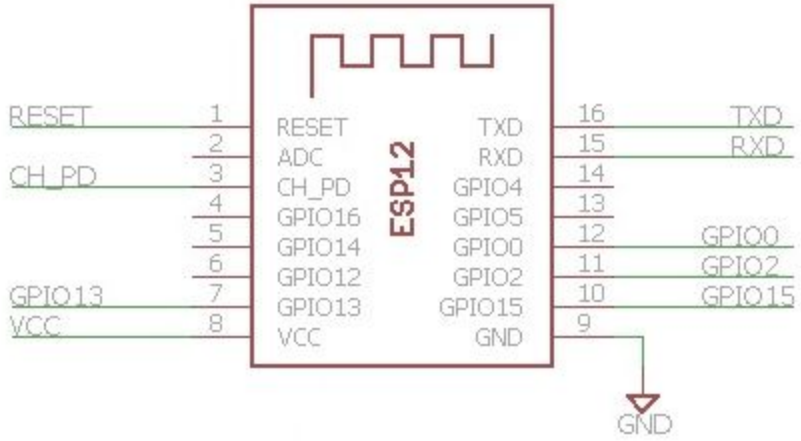

Figure 10. Servo Motor ESP Schematic

Subsystem Testing

The testing for this subsystem was quite simple and successful. Our first step was to decide what kind of motor we would use to power and rotate our blinds system. After searching for various home automation projects and forums online, it was decided that a servo would be best due to its size, low power requirements, and high torque specifications. It also seemed quite

easy to connect the servo to the blinds at first sight. The next step was deciding between a standard high torque servo and a continuous rotation high torque servo. As previously described, we ended up choosing the standard high torque servo due to the precision of being able to write exact angles in the code. After deciding on the motor system, we needed to combat the challenge of integrating the servo with the hex rod of the blinds. Once a custom servo spline attachment was acquired, a successful connection was made and the servo was able to provide enough torque to rotate the entire setup since the blinds were not too large or heavy. The final steps of testing this subsystem involved mounting the servo, constructing the frame, and integrating it with the other subsystems such as the software, weather station, and mobile application. Overall, this entire subsystem was executed successfully and proved to be effective in fulfilling our overall system requirements.

3.5 Detailed Design/Operation of Servo Motor Software (Indoor Board)

Subsystem Requirements

In order to create an automated set of blinds, finely tuned and robust software, particularly in the case of the servo motor, was required in the creation of this product. A standard, high torque, non-continuous rotation servo motor was necessary for setting the exact position of the blinds through a PWM output. A continuous rotation servo motor could have been chosen instead, but this would have complicated the code and made it slower and less efficient. The software needed to be able to accept information from both the user (in manual mode) and the weather station (in automatic mode) to change the position of the blinds. In addition to these things, some safety considerations were also required. In order to avoid breaking or over-rotating

the servo and blinds, significant testing was needed to properly adapt the position of the servo to the various position of the blinds. Being able to control the speed of the servo was also a must to prevent the blinds from breaking and over-rotating.

Software Flow

The figure below shows a flowchart of the code and logic implemented in the indoor blinds board.

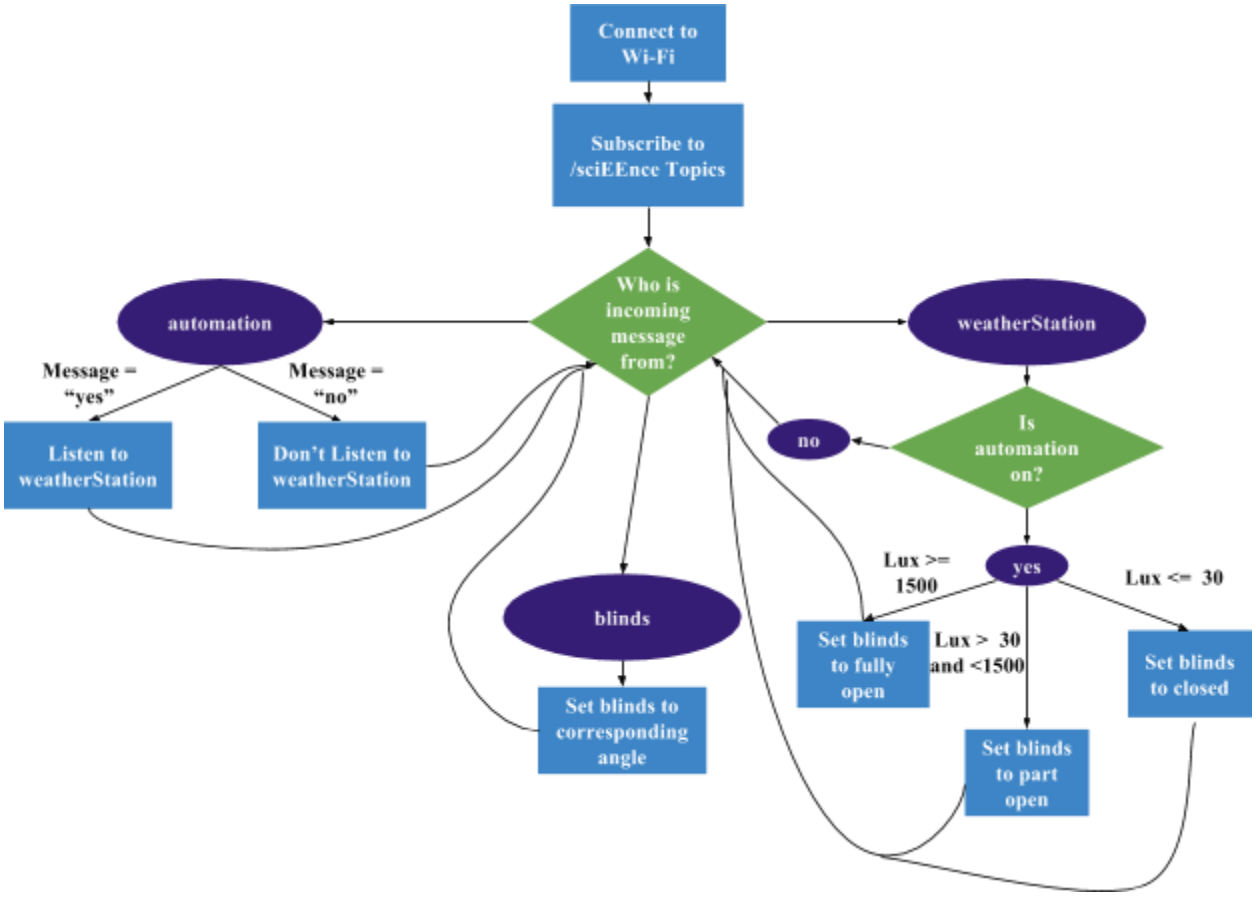

Figure 11. Blinds Code Flowchart

When the indoor board first turns on, it attempts to connect to Wi-Fi. If it can connect to it's last known settings, it does so. If it cannot, it creates an access point that the user can connect

to in order to configure the Wi-Fi and connect to an appropriate connection. Once the Wi-Fi is connected, the code subscribes to the "/automation", "/blinds", and "/weatherStation" topics under the "/sciEEnce" hierarchy.

If it receives a message from "automation", it then figures out if the message is "yes" or "no". If the message is "yes", it knows to listen to messages from the weather station next time they come in. If the message is "no", it knows to stop listening to data from the weather station.

If it receives a message from "weatherStation", it must first take a look at if automation is on or not. If it is off, the message is ignored. If automation is on, it takes a look at the lux value in the message and adjusts the blinds accordingly.

If it receives a message from "blinds", it looks at the value in the message and adjusts the angle of the blinds to the corresponding value.

In all cases, after the message has been received and deciphered, the code goes back and listens for the next message to come in.

Subsystem Testing

To test the servo motor software, we familiarized ourselves with the Servo.h library in Arduino. Since we decided to use a standard servo instead of a continuous rotation servo, setting the angle was not a problem due to the *Servo.write* statement. Delays were added in the manual rotation of the blinds to ensure that the servo motor controlling the blinds was not moving too quickly. Once we were comfortable with being able to write specific angles to the servo, we continued our testing by pairing it with the MQTT protocol and Wi-Fi communication. By setting specific publish and subscribe topics, we used MQTT Lens to verify that the angles were

being sent over the web server. After testing basic Wi-Fi communication, we further integrated the subsystem by pairing it with the iOS mobile application. A slider tool was implemented in conjunction with the delays to be able to control the servo between open and close at a decent rotational speed. Once the basic structure of the subsystem was complete, we continued to expand on the subsystem through weather station integration (light sensor automatic control) and multiple blinds functionality. Due to the convenience of the publish-subscribe protocol functionality, multiple blinds can be subscribed to the "All Blinds" topic and all move at the exact same time as opposed to having each motor move to its exact position in subsequent order. To make the automated feature more robust, we tested different light values outdoors and created various cases to represent nighttime, sunny, and very bright weather states. Overall, the servo software subsystem testing proved to be successful and effective in designing and finalizing our demo day product.

3.6 Detailed Design/Operation of System Powering

Subsystem Requirements

Because of the different energy and power requirements for our two different circuit boards, we used two different systems to provide each of them with power. Due to the nature of the outdoor board and the infeasibility of running wires through a window, the outdoor board needed to function independently off of a single 3.7V, 850mAh, 3.1Wh LiPo battery, a 2W solar panel, and a charging circuit that connects the two and protects the battery. These values were determined in order to ensure that the battery would not discharge faster than it is being charged by the solar panel. We wanted to incorporate solar power into our outdoor board so that the user

would rarely have to worry about changing the battery, and it would be a more cost efficient system than having to buy new AA or AAA batteries much more frequently. We also found that using a battery, instead of tethering the system to a wall wart, allowed the user much more flexibility in where and how they mounted their weather station. The only other required components to the powering systems are DC to DC converters to step down our power supplies to the 3.3 volts necessary to power the ESP chips and sensors.

Schematic Explanation

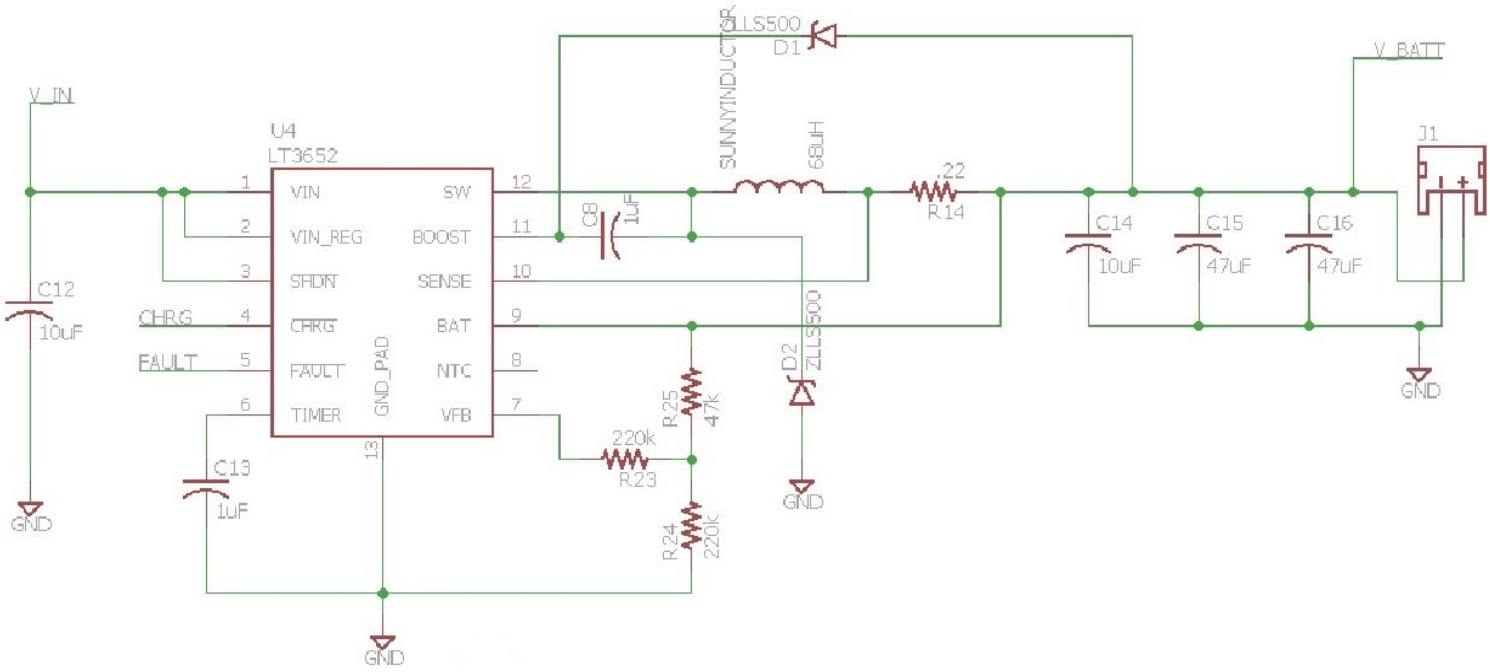

Figure 12. Solar Charger Schematic

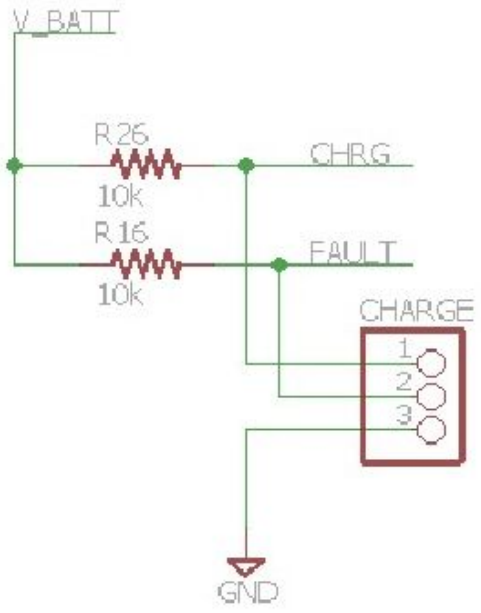

Figure 13. Solar Charger Battery Connection Schematic

Looking at Figure 12 we see the circuit and IC that we used to regulate the charging of our battery via the solar panel. The Vin line is the input voltage from the solar panel, connected to ground through a decoupling capacitor. In order to maximize the charge capability of the circuit, this voltage should be at least 6 volts. When our panel is in sunlight it easily attains this level, but in overcast and lower light environments it drops below this threshold. When this happens the circuit works to maintain the current charge level of the battery, neither draining nor charging it. It does this by monitoring the solar cell voltage, and drawing less current when it senses that the cell is being forced to do too much work. The battery being charged is connected to the V_Batt line. Figure 13 shows us how the batteries voltage is monitored and regulated by the IC. The voltage at V_Batt is connected to the CHRG line through a resistor. The CHRG line then connects to a pin in the IC, seen in Figure 12, that takes the current battery level into account when setting the charge current.

The package and circuit connecting the panel to the battery operates using a maximum power point transfer (MPPT) principal. This essentially means that the charger provides as much current to the battery as it is capable of drawing from its power supply, but no more than the battery can handle. The IV curve of a solar panel has a fairly constant voltage as current increases, only dropping slightly over time, before reaching a point where they decline increasingly rapidly with even small changes in current. An example can be see below.

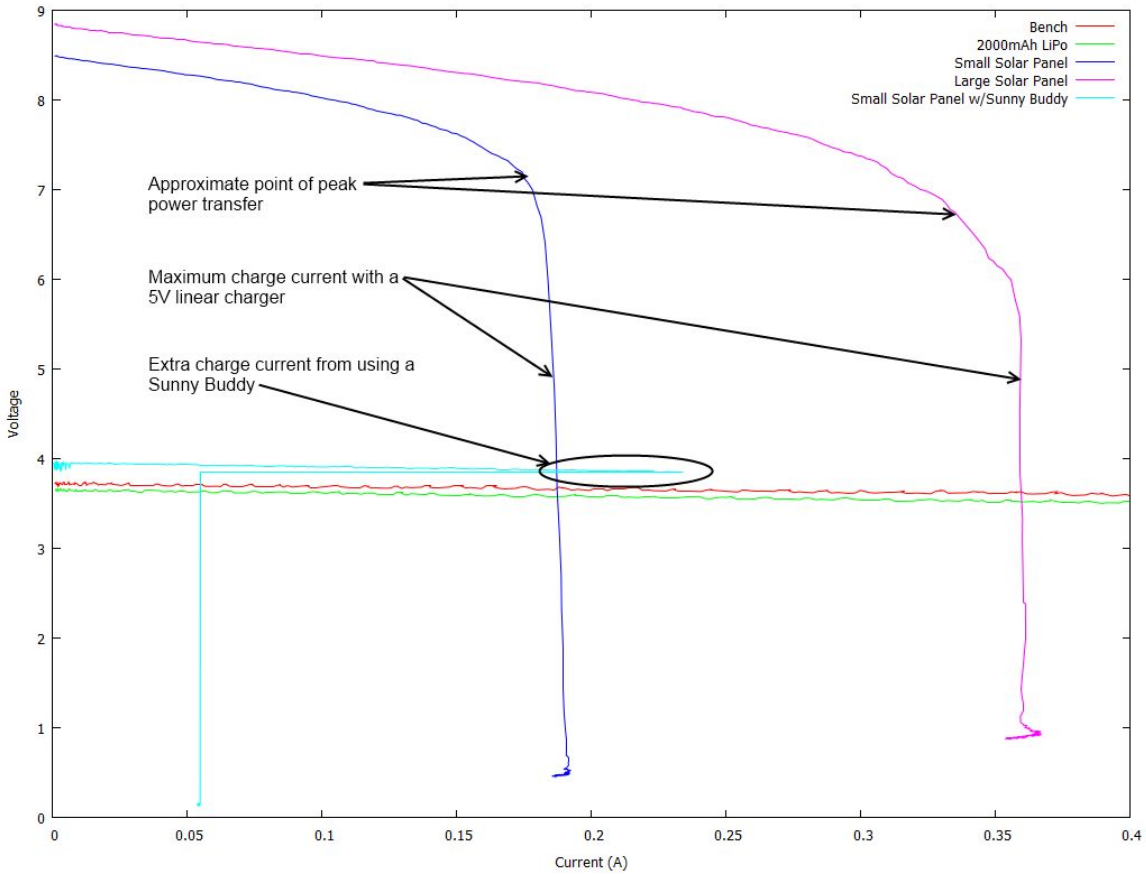

Figure 14. Solar Panel IV curves

The charging circuit locks in the current into the battery at the knee of the solar panel's IV curve, where the maximum power is converted to the load. Notably, in the figure above the charge circuit is able to provide 240 mA compared to 180mA from a panel with no circuit

attached, an increase of 33%. Our charge circuit is thus able to more efficiently charge the battery while also ensuring that it does not get overcharged as well.

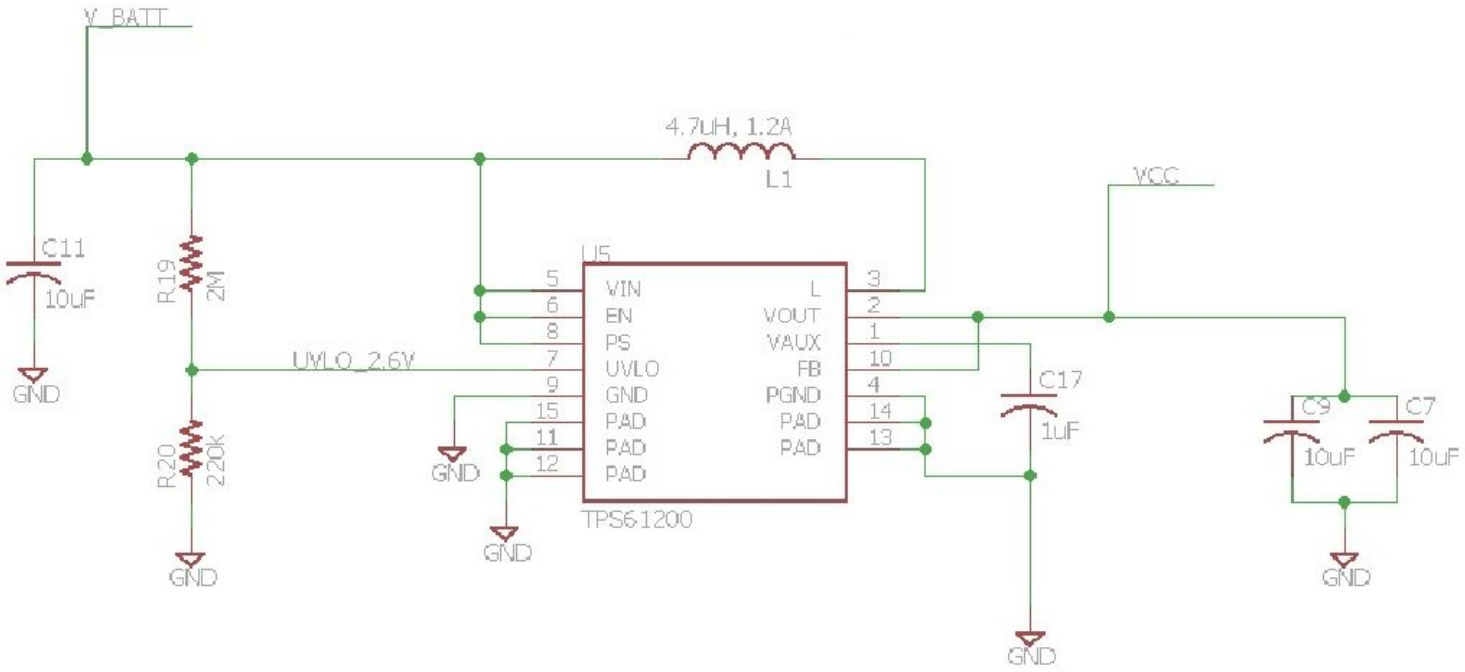

Figure 15. Outdoor DC to DC Converter

The final major component of the outdoor board power system is a DC to DC converter to regulate the voltage powering our ESP and sensors. The circuit can be seen above in Figure 15. This IC and accompanying circuit takes in the voltage of the rechargeable LiPo battery on the V_Batt line. This voltage can vary anywhere from 3.7 to 4.2 volts. This specific converter, the TPS61200, then steps the voltage down to 3.3 volts so that it can be used to provide power to the other components on the board through the Vcc line. These connections can be seen in Figures 3, 4, 5, and 6.

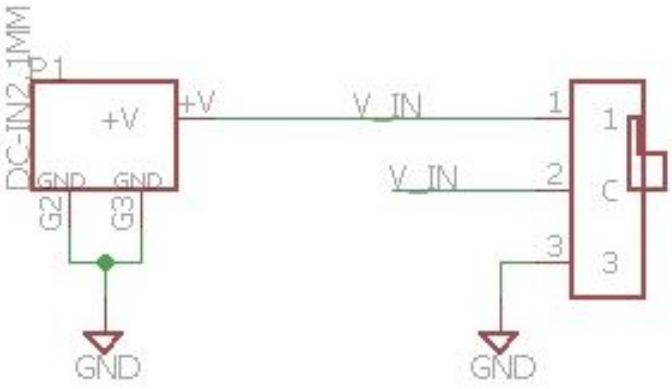

Figure 16. Wall Wart Input Schematic

Meanwhile on the indoor board, to avoid the difficulties of turning WiFi on and off repeatedly, we decided that we should keep it constantly powered. Thus, the board received its power from a 5V wall adapter with a microUSB connection. This can be seen in Figure 16 above, where the DC input from the wall wart is linked to V_in. This input is used to directly power the servo motor seen previously in Figure 9. It is also fed into a DC to DC converter that is identical to the one seen on the outdoor board, seen below in Figure 17. The 5 volt input from the V_in line is stepped down to 3.3 volts on the output line Vcc. We then use this line to power the ESP on the indoor board, seen previously in Figure 10.

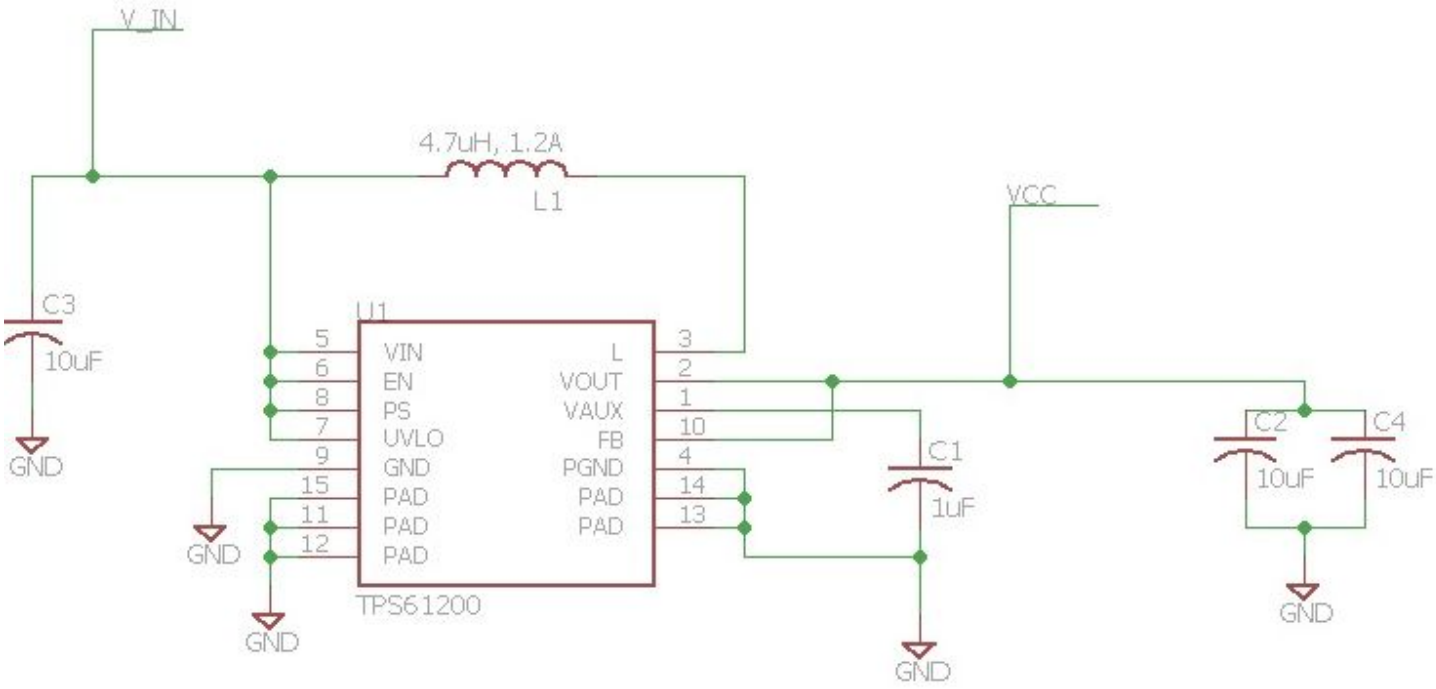

Figure 17. Indoor DC-DC Converter

Subsystem Testing

The testing of the power subsystem was fairly simple and easy for both boards. Some of the earliest testing we engaged in involved the solar panel, battery, and associated charging circuit on the breakout board. At a very early date we made sure that the circuit could be used to recharge the battery via the solar panel we were working with. We also consulted Professor Fay on how the charging circuit worked as well as what calculations to perform in order to figure out how big our battery and solar panel needed to be. As we tested our weather system with breakout boards, we would test the power system alongside it by using our battery and solar panel to power the prototyped weather station. At no point did we encounter difficulties with providing power to the weather station or draining the battery too fast. When we got the final board we

carried out this same process, using our solar panel and battery to provide power while testing other systems, and we encountered no complications.

Testing the indoor board power system was even simpler. Once we got our wallwart, we used it to power our servo motor to make sure it could provide enough juice. Once we had to test the final indoor board, simply providing power to the board and checking that the voltage levels at many pins were correct was enough to verify that the final system was working, along with testing the other functionalities associated with that board.

3.7 Detailed Design/Operation of the Phone Application

Subsystem Requirements

This project includes a phone application that acts as the user interface for the entire product. It is developed in the latest version of Xcode and programmed in Swift in order to run on iPhone devices. Swift was chosen as the programming language because it was familiar to multiple members of the group, and also due to the popularity of iPhones. It also features a user-friendly interface for intuitive control of the blinds and easily readable information from the weather station. On the blinds side, the user is able to either manually control the blinds or put them into automatic mode via the phone app. On the weather station side, information about the current temperature, humidity, barometric pressure, and luminosity are displayed to the user. The user is also able to add new blinds or switch which weather station they are connected to by entering in a name and registration code. The code overall makes use of the MQTT protocol, and subscribes to information from the weather station and publishes information to the indoor blinds system. MQTT was chosen due to the simplicity of working with the protocol, and the fact that it

was able to be easily integrated with the rest of the software in this project.

Software Flow

The diagram below shows each of the screens in the phone application, as well as the segues between each of them.

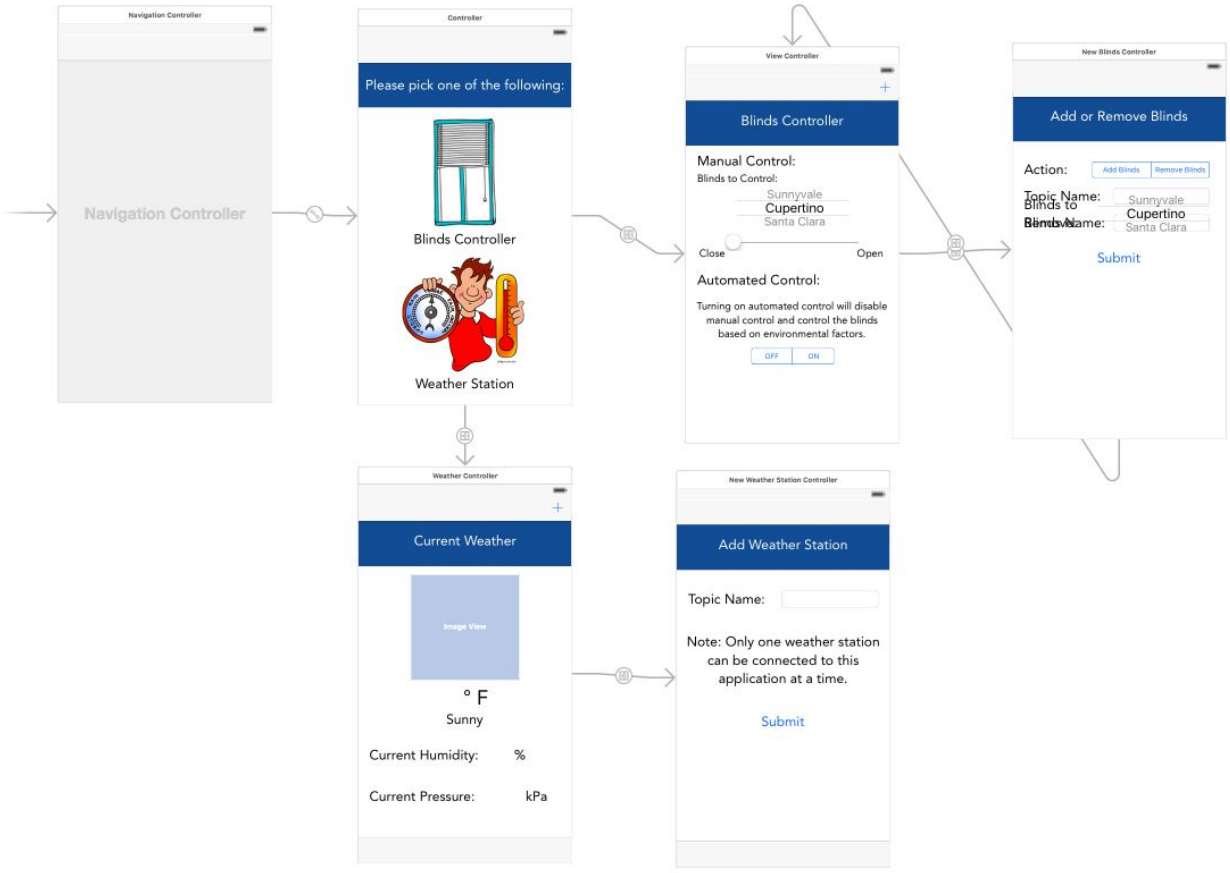

Figure 18. Mobile Application Software Flow

When the user first opens the application, they are met with a screen that asks them to click on either the Blinds Controller button or the Weather Station button.

If the user chooses to click on the Blinds Controller button, this brings them to a new screen where they have two options: manual control of the blinds, or putting the blinds in

automatic mode. Manual control of the blinds involves first selecting which set of blinds you want to control via a rotating picker view. The user can select “all blinds”, which will move all sets of blinds they have added to the list, or select a single individual set of blinds. Once they have chosen a set of blinds, they can use the slider below it to rotate the blinds between open and closed. Every time the slider is moved it publishes a message to the MQTT server. This message is an exact angle position which the blinds are subscribing to. When the blinds receive that message, they rotate to this angle position. The other option for the user is automatic control. If the user selects the “ON” button under automatic control, it disables the manual control, and publishes a message to the blinds that they now need to listen to the light sensor on the weather station. With automatic mode on, the blinds then adjust automatically based on the light reading that is coming in.

From this blinds controller screen, the user has the option to hit the “+” button in the top right hand corner. If this button is pushed, it brings the user to a new screen where they can add or remove blinds. Adding blinds involves typing in a registration code under “Topic Name” and then typing in whatever they want to name the blinds under “Blinds Name” (i.e. “Kitchen Blinds”). The topic name is a code given to each set of blinds that the phone app can publish data to. The blinds are subscribing to this topic name and will thus respond to it. Removing blinds simply involves selecting a set of blinds that have been previously added from a list that pops up. Once the user has appropriately added or removed a set of blinds, they can hit submit to return to the previous screen.

Back at the home screen, the user also has the option to select the Weather Station button. If this is pressed, it brings the user to a new screen that displays data incoming from the outdoor

weather station. It shows the current temperature, humidity, and barometric pressure, as well as a picture graphic that states whether it is sunny, partly cloudy, cloudy, or night time. This picture changes based on the reading coming in from the light sensor. Also, similarly to the blinds controller, there is a "+" button in the right hand corner of the screen that allows the user to change which weather station they are connected to.

Subsystem Testing

The code for the mobile application was tested through the use of a lot of print() statements to the console to make sure that the code was performing as expected. Since the mobile application essentially tied together both the blinds system and the weather station, it was important that both of these codes were functional before this system could really be tested. MQTT Lens was also used to make sure that the app was able to send and receive messages correctly. Below is a screenshot of the console in Xcode receiving messages from the weather station.

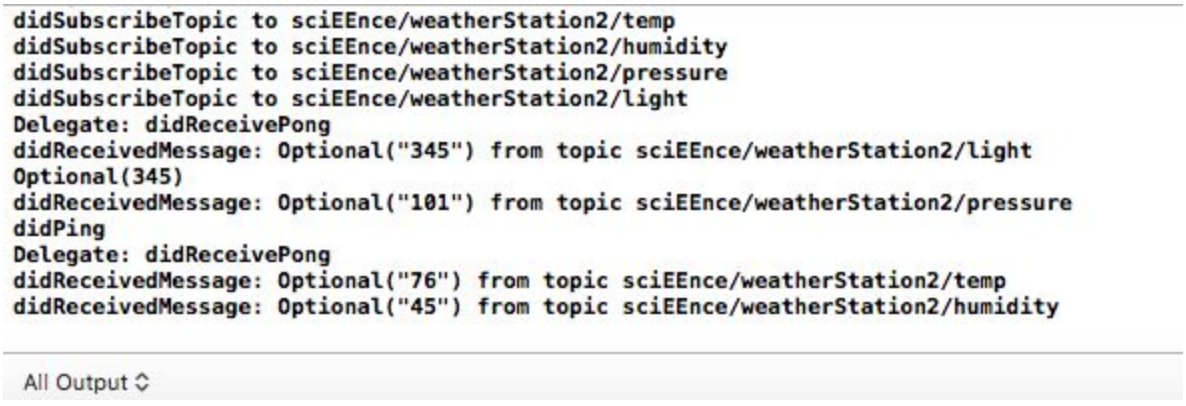

Figure 19. Mobile Application Testing

3.8 Detailed Design/Operation of Wireless Communication

Subsystem Requirements

As this year's theme for Senior Design was "Internet of Things", wireless communication was an integral part of this project. The underlying interface of the wireless communication needed to be the MQTT protocol. This was due to its publish and subscribe features and the amount of information we had and consequent ease of use with the ESP8266, Arduino IDE and iOS phone application. MQTT was the simplest and fastest way to get all of our devices to talk to each other. There was a lot of information out there online about the protocol, and it was easy to connect our devices through it. Amazon Web Services was chosen to be used as our cloud-based MQTT server. This server acted as a mediator between the ESP clients and the phone application. The cloud based server was chosen because it eliminated the need for a physical server in the form of a Raspberry Pi, and allowed users to be able to connect to our devices from anywhere without worrying about where the server was located.

The weather station needed to publish to unique topics under "sciEEnce/weatherStation#", and the mobile app needed to subscribe to these same topics. Similarly, the mobile application needed to publish to "sciEEnce/blinds" in order to communicate servo positions to the blinds system, and the blinds needed to subscribe to this topic. Each set of blinds was hardcoded to subscribe to specific topics in order to differentiate between the various blinds in a user's home. This was done within the Arduino ESP blinds code.

Software Flow

The flowchart below shows the wireless communication between devices based on MQTT protocol. It shows what messages each device is publishing as well as subscribing to.

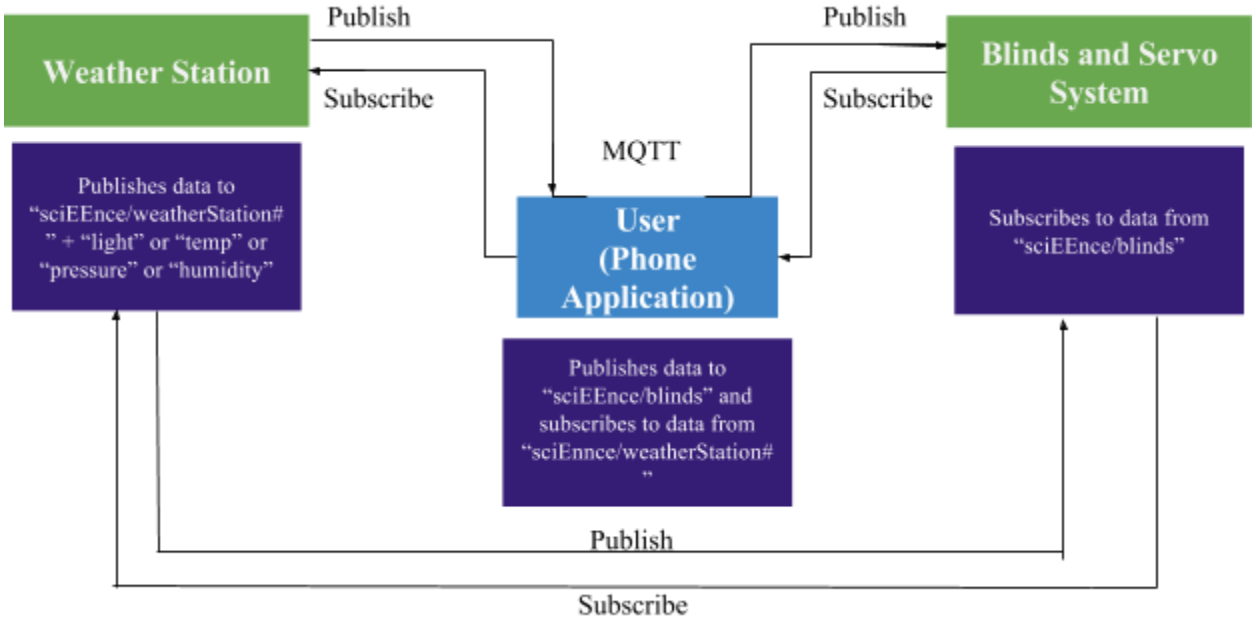

Figure 20. Wireless Communication Flowchart

The mobile application is the center of the wireless communication, as it receives messages from the weather station and also publishes data to the blinds and servo system. The weather station publishes data via "sciEEnce/weatherStation#" plus the corresponding sensor to the mobile application, while the blinds and servo subscribe to data from "sciEEnce/blinds". This data either sends servo angles telling the servo motor what position to, or tells it to go into automatic mode. In the case of automatic mode, it begins listening to the messages being published by "sciEEnce/weatherStation/light" in order to set the positions of the servo.

Subsystem Testing

The wireless communication subsystem was tested by using an application called MQTT Lens. This is a downloadable Google Chrome application that allows you to publish and subscribe to selected topics on a server of your choosing. By publishing and subscribing to topics such as “sciEEnce/blinds” or “sciEEnce/weatherStation”, the team was able to tell if messages were properly being communicated between systems. Below is a screenshot of MQTT Lens publishing data to one of our sciEEnce topics.

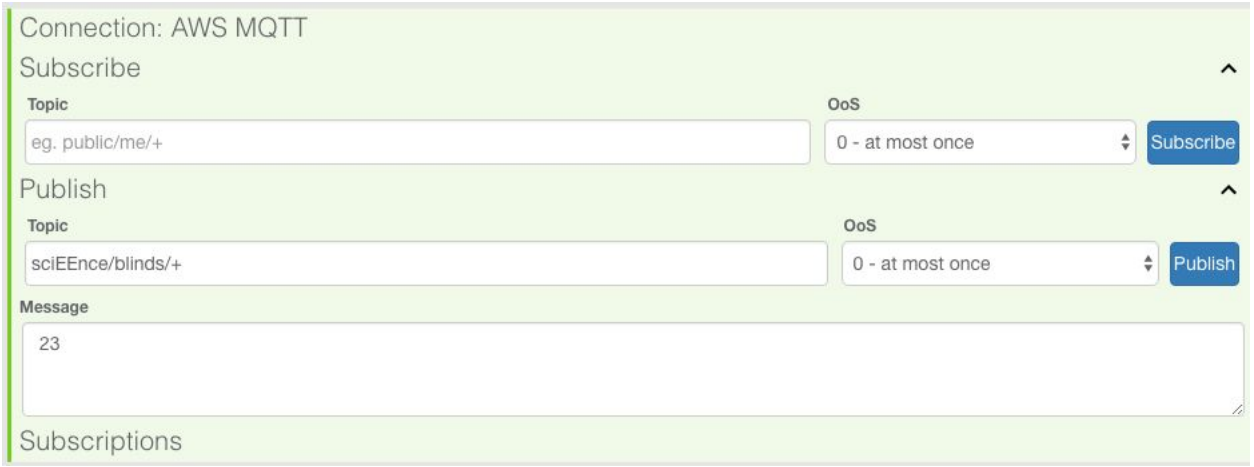

Figure 21. Wireless Communication Testing

3.9 Interfaces

Overall, the iOS mobile application serves as the main interface by which the user is able control the servo manually, view various weather parameters as data is being read, add multiple blinds and weather stations, and set automatic mode for the entire system. The communication between all the subsystems is done through the MQTT publish-subscribe protocol.

4 System Integration Testing

Integration testing began after all the individual subsystems proved to be working effectively and successfully. This involves verifying the functionality of the MQTT publish-subscribe protocol, integrating Wi-Fi communication with the mobile application, combining the servo software with the mechanical system, testing the effectiveness of the sensors, and determining the best method for powering our overall system. Once all the separate subsystems were complete, the first step we took to integrating all the subsystems was testing the Wi-Fi communication, the backbone of our overall system.

The Wi-Fi communication subsystem and the mobile application subsystem go hand in hand as the main interface by which the user is able control the servo manually, view various weather parameters as data is being read, add multiple blinds and weather stations, and set automatic mode for the entire system. Most of our integration testing was done through MQTT Lens. By coding simple and subscribe and publish features in the app, we were able to test simple communication between our web server and our app. Since the mobile application essentially tied together both the blinds system and the weather station, it was important that both of these codes were functional before this system could really be tested. Due to the instant feedback provided by the MQTT Lens, we were confident in our ability to finalize the other subsystems with the Wi-Fi mobile app.

Once the main interface was complete the next main subsystem to integrate was the servo software and the mechanical system. This involved being able to move the motor manually through a slider tool via Wi-Fi. To make sure that commands were being properly sent over

Wi-Fi, MQTT Lens was once again implemented for testing purposes. With our project specific topics, we were able to successfully integrate the software and hardware of the motor without having to deal with communication or connection issues from the other groups on the web server.

After the basic structure of the subsystem was complete, we continued to expand on the subsystem through weather station integration (light sensor automatic control) and multiple blinds functionality. Due to the convenience of the publish-subscribe protocol functionality, multiple blinds can be subscribed to the "allblinds" topic and all move at the exact same time as opposed to having each motor move to its exact position in subsequent order. To make the automated feature more robust, we tested different light values outdoors and created various cases to represent nighttime, sunny, and very bright weather states. We also finalized the weather station view of the app by testing different parameters outside and coding a user-friendly interface for the user. This weather view acted much like a simple weather app that updated the user with current weather data such as temperature, humidity, light readings, and barometric pressure. Overall, the servo software subsystem testing, weather station integration, and Wi-Fi interfacing proved to be successful and effective in designing and finalizing our demo day product.

Another phase of the subsystem integration was verifying the power connections on the PCBs we designed for the product. After the boards were fully soldered, we checked the connections with a DMM, ensure they were programmable, and proceeded to test the other subsystems. The only requirement of the indoor board power-wise was that the servo needed 5V and the ESP 12 module needed 3.3V. The outdoor board was much harder to test an integrate

due to the nature of LiPo battery powering and solar charging. To ensure the functionality of our circuit board, we tested different voltage parameters outside. The requirements for this board included not only the 3.3V to all components, but making sure that the battery was properly charging and discharging as well. Once the voltage values were verified, we were able to complete the final phase of integrating the Wi-Fi app, the mechanical system, and the weather station.

The final steps of testing the mechanical subsystem involved mounting the servo, constructing the frame, and integrating it with the other subsystems such as the software, weather station, and mobile application. Overall, this entire mechanical subsystem was executed successfully and proved to be effective in fulfilling our overall system requirements. Once all subsystems were fully integrated, further testing was done to verify the effectiveness of the Wi-Fi mobile app with all other subsystems such as the weather station and the mechanical setup. We made sure that all connections, electrical and mechanical, were secure by controlling the rotation of the blinds multiple times. We verified the functionality of the weather station by capturing user-controlled and environmental data. Lastly we tested the strength of our Wi-Fi connections by testing our fully integrated system at various distances.

In conclusion, our final product meets our overall system requirements laid out in the previous sections. The best demonstration of fulfilling our requirements was when we were able to send the servo exact positions over Wi-Fi and through our mobile application. This one feature was the main requirement of our system. We further expanded on this accomplishment by fulfilling the automated requirement of the blinds. This was evident through the blinds being able to rotate without having to control the slider tool. Being able to put our weather station to sleep

and take data after certain time intervals showed the robustness of our outdoor system and its low-power functionality. Using 3.3V - 5V components also demonstrated that our system was of a low-power nature. The combination of the LiPo and solar charging features showed that our system was energy-efficient and user-friendly. Lastly, the functionality of our mobile application, the backbone of our product and the main system interface, displayed its inter-connectedness and accessibility.

5 To-Market Design Changes

Due to budget and time limitations, our functional prototype differed significantly from the product we would have liked to take to the market. Some further development and design decision making would be required for a truly automated, Internet of Things, user-friendly, and commercially available product. Most importantly, there were many design decisions we ended up leaving out of the product due to the complexities and difficulties associated with their implementations. Some additional features we would have liked to add include an indoor temperature sensor for the blinds setup. This be used to help intelligently heat or cool the home via sunlight. Overall the house could cool down or warm up more efficiently, cutting down on energy costs. Other additional features include sensing the precise angle of light hitting the blinds and rotating appropriately or implementing timers for the user to rotate their blinds at set times of the day. A rain and wind sensor could also be included to provide more accurate weather readings to the user at the beginning and throughout the day. Furthermore, incorporating some speakers or sound functionality could help the user wake up in the morning along with the opening of the blinds. Lastly, a final addition to the product would be the raising and lowering of the blinds rather than the rotation feature by itself.

There are also some changes we would have liked to make to our existing design outside of any additional features. Firstly, nicer and sturdier blinds would have made the entire setup more aesthetically pleasing and more robust. Some mistakes were also made on the board design such as the proper implementation of switches to control power on and off. The application is currently available for Apple devices, but a version for Android would also need to be developed to allow for the same level of use for Android consumers. A larger solar panel would also be more ideal for efficient charging since our current one peaks at 6V. A higher quality of service level (QOS) would be better for quicker connections and precise communication between subsystems. Lastly, there were some features we decided to leave out at the end such as weather history and clothing to wear for the day which would have made the product more user-friendly and useful.

6 Conclusions

Overall, Blind Me With SciEEnce was a success. While there is always room for improvement, it is important to keep in mind that this is just a first attempt at a prototype. Even though the path to a final product was filled with various challenges and setbacks, hard work and innovation paid off at the end of the day. The final product can take weather readings from the outdoor station and publish them to an MQTT server, rotate the blinds automatically and manually via the phone app, efficiently charge via a solar charger and LiPo battery setup, and so much more! The basic requirements for the project were completed, and while minor adjustments can be made for improvement, the overall functionality of the Blind Me With SciEEnce final product is effective and successful.

7 Appendices (Attached/Found on Website)

Appendix A: Hardware Schematics and Component Data Sheets

Appendix B: Software Listings

Appendix C: User/Installation Manual