Fall 2021

Automated Blind Control

Daniel Nahra
The University of Akron, dan50@zips.uakron.edu

Matthew Lacek
The University of Akron, mlj121@zips.uakron.edu

Timothy Kurczewski
The University of Akron, tjk99@uakron.edu

William Daulton Baksa
The University of Akron, wdb21@zips.uakron.edu

Senior Design Project

Final Design Report

Automated Blind Control (A.B.C.)

Design Team Number 6

W. Daulton Baksa (Archivist)

Tim Kurczewski (Software Lead)

Matthew Lacek (Hardware Lead)

Daniel Nahra (Project Lead)

Faculty Advisor: Dr. Michael L. French

11/25/2020

Table of Contents

List of Figures

List of Tables

Abstract

Window blinds are a common feature in households and are adjusted multiple times each day by homeowners. This document provides a solution to saving time and energy costs to homeowners who must adjust their blinds manually by creating an autonomous blinds control system that provides the optimal sunlight into the home as expected for the time of the day and year. The human error of leaving the blinds shut during the cold winter months and leaving them open during the warm summer months will become a thing of the past with the implementation of these automated blinds. The automated blinds control system (A.B.C.) focuses on sunlight data entering the window and recording the inside and outside temperatures. The system makes decisions from the input data and adjusts the vertical and horizontal state of the blind slats to ensure proper energy savings of the room. The design incorporates a user-friendly webpage for customization and control from anywhere with an internet connection. Manual override will still be present to ensure operation of the system in the event of no available internet access. This project results in a cost-effective product that modernizes the home, creates energy cost savings, and is hands-free for homeowners. -DB, ML

1.0 Problem Statement

1.1 Need

A lot of power is expended in order to heat and cool homes and other buildings. Windows are a typical feature in the average building and about 30% of the energy used to heat a home is lost through windows. During winter seasons, sunlight falling on windows can also provide up to 76% of its energy as heat, reducing heating costs. During the summer season, blocking out light can reduce heat entering the building through windows, reducing cooling costs. It is estimated by the Office of Energy Efficiency and Renewable Energy that 75% of residential window coverings are left in the same position each day. Currently, window coverings are automated, but the homeowner still needs to control these with a remote. A system to automate and control window blinds based off light coming in and temperature desired is needed to save energy costs. – DB, TK, ML, DN

1.2 Objective

The objective of this project would be to design and prototype an automated, light and temperature sensing window blinds system. The device would detect temperature, both inside and outside, and incoming sunlight to determine proper window blind position for maximum energy savings. Including functionality for this device to be controllable from a user's cell phone will enable a user to change the automatic settings if they want to override the blinds for any reason. – DB, TK, ML, DN

1.3 Background

The Automated Blind Control, or A.B.C., is a system that would fully automate and control a window blind to allow for sunlight to enter a room when needed and shut out warm sun rays when not desired. By creating a user-friendly platform, a profile can be created of when to open and close the window blinds allowing for an automated system that keeps the temperature of the room at the desired level. This project will improve on existing technologies that accomplish a similar goal. The A.B.C. system will also incorporate an integrated battery backup system that can be charged as the system operates, allowing for the system to remain running,

even with the temporary loss of main power. Remote control abilities as well as automatic modes will be available through either a phone application or online website. -TK, DB

The light sensors would detect sunlight intensity, in the range that includes infrared and UV light, to determine the optimal angle of blind openings such that the desired amount of light enters through the blinds. The positioning of these light sensors needs to be determined such that they detect the incoming light properly (The Sunlight Beam Index [4]). The blinds will automate this process of opening and closing based on the amount of light unless overridden by user control. In addition to this, the light sensors will not react to ambient light from electric sources such as car headlights and streetlights. - ML

The use of a temperature sensor in our A.B.C system has been previously used in other systems but is useful in the fact that it can be used to sense the temperature in the room and outside. We will be using one to measure inside temperature and then will be getting the outside temperature from a Weather API on the internet. The temperature sensor can detect the temperature and relay that information to the controller of the system. The controller would then decide to open the blinds or close them depending on the temperature and what is relayed from the light sensor. The automatic sensing of the light and the temperature would allow for more energy efficient blinds and help the house stay warm or cool depending on the desired temperature. - DN

In order to allow for the A.B.C. system to be accessed from anywhere, an application will be created in order to give users access from any remote device directly to the blind system. Some of the current products on the market are controlled via a remote controller which limits the control to only when a user is close to the blinds and has access to the remote (US 2012/0193035 A1). Using an application instead of a remote control greatly reduces the hardware cost of the product and allows smart technology to go into the homes of ordinary people (Customary Homes to Smart Homes using Internet of Things and Mobile Application [6]). Currently the different ways of communication from a smart home system to a device are GSM, Bluetooth, or using a microcontroller with a communication module (A Comparative Analysis on Smart Home System to Control, Monitor and Secure Home[1]). Each method has its

benefits and disadvantages. Making use of the communication methods and picking the best method for the A.B.C system will depend on trial and error. - DN

The motors used will be low power to supply only the torque necessary to adjust the blind openings as well as move the blinds up and down as desired. Utilizing low-power motors will keep the power needs cheap as well as require little energy to keep the blinds operating over the course of each day. Ideally, the motors will only draw power when needed for adjusting the blinds so that power for the motors is only drained when the system is making its adjustments. - ML

For ease of installation and implementation with the common household, the system will obtain its main supply of power from a wall outlet using an AC-DC wall wart plug-in. This accessory allows for a typical 120V, 60 Hz connection from the system to the house and establishes a system rail voltage of 12V DC that can be stepped down and utilized with the essential sensors, motors and controllers. In case of a power outage where the home loses power, it is desired that the system still be able to operate for some duration of time until the normal source of power is restored. To ensure this, energy storage in a battery system is necessary. Solar power was investigated as a potential option but would require significant cost and a more complex system overall, due to the variations in solar irradiance that occur daily. This would likely cause the system to not be independent as a long-term solution (Cooperation and Storage Tradeoffs in Power Grids with Renewable Energy Resources [5]). Battery backup can still be utilized through an uninterruptable power supply (UPS) which charges while main power is provided to the system and is used to power the system the moment main power is no longer present. Battery protection will ensure that the UPS remains undamaged and reliable. Due to the desire for efficiency and cost-effectiveness, cheap and efficient battery systems and power devices will be used to maximize storage life (Feasibility Analysis of Different Energy Storage Systems for Solar Road Lighting Systems [2]). - ML

The end goal of the designed A.B.C system is to incorporate the theory behind all the products mentioned before and create a one-of-a-kind, fully assembled system. The focus will be on energy saving costs and ease of user interfacing. The cohesion of monitoring indoor and outdoor temperature along with UV light intensity will provide the system with the most accurate

information to provide the highest energy savings costs for the homeowner. An easily accessible user platform will be incorporated, and all desired system variables will have fully customizable capabilities. The A.B.C will be an affordable product for all homeowners, providing ease of control and constant savings with the use of green energy. - DB, ML

Initial research of the project revealed that a similar project was conducted in 2017 by a group of engineering students at the California Polytechnic State University (LIGHT SENSING AUTOMATED BLINDS [3]). Their design was realized by utilizing an Arduino microcontroller that controlled a DC motor. Their system was wall powered and opened and closed the blinds based upon the light intensity of inside and outside the room. No remote-controlled abilities were present in this design. - TK, DB

Further research yielded several patents that directly related to the problem at hand. US Patent 2012/0193035 A1 is for a "MOTORIZED BLIND CONTROL DEVICES, METHODS OF USE THEREOF" that was created August 2, 2012. The patent describes a motorized window blind that is controlled from a remote device. The window blind system uses a single axis motor that lifts and drops the window covering. The window covering is a single drape style blind that is not slotted. It does not allow for partial light emission when the blinds are down. A light sensor and a temperature sensor send a signal to the motor to open and close based upon defined thresholds. - TK

A second patent, US Patent 7,417,397 B2 is for a "AUTOMATED SHADE CONTROL METHOD AND SYSTEM" that was created August 26, 2008. The automated shade control system utilizes temperature and light sensor data to calculate a desired blind covering angle. It also includes solar radiometers to measure total solar radiation as well as local area daylight. The system has capabilities for manual remote override. A fixed blind system was used, and the motor control only controlled the angle of the blind opening. The blinds do not move, roll, or slide off the window surface. - TK

There are a few different companies today that offer similar products to the problem at hand. MySmartBlinds by tilt is the closest comparable product which offers window blind coverings, both in a roller style as well as slat style, with remote control operations. Alternatively, they also offer a conversion kit to allow for normal manual adjusting blinds to be

automated. This portion is just describing what the other companies offer, so I think it can stay (DB). An additional solar panel kit can be purchased to allow for the blinds system to be powered from solar technology as a backup for wall outlet power failure. MySmartBlinds has a phone app that allows users to open and close the window blinds remotely. - TK, DB

The current technologies for the types of problems we intend on providing a solution to contain some limitations. One problem with similar items in the market today is the purchase and assembly required of several components to achieve the final, automatic system. A system may only contain means of using outlet power and a remote control. This takes up outlet space as well as provides the possibility of losing or breaking the remote, disabling the automation factor completely and adding the need to purchase additional remote controls. - TK, DB, ML

Other systems contain a form of battery power as the source. These power sources would need replacing or recharging at some point. The addition of solar panels for battery recharging is available, but at an additional purchase and installation fees. Depending on the homeowner, this may also require additional contracted labor costs. This also means that the systems are not all inclusive and need to be modified in order to achieve the desired result. - ML

Current systems like MySmartBlinds do have a phone app but only allow interaction with the blinds system when within Bluetooth™ range. There is no web-based application that can be accessed remotely and directly to the blinds. On the project done by CPS University (LIGHT SENSING AUTOMATED BLINDS [3]), there is no remote-control feature at all. The system runs autonomously, and blinds cannot be manually opened or closed. -TK

The A.B.C system will be similar to these current technologies in several ways. The A.B.C will utilize light and temperature data acquired from sensors and the internet to determine the optimal amount of light filtering through the blinds, automatically adjusting to accommodate for changes in the direction of light. It will incorporate low power motors to control the position and angle of the blinds. Users will additionally have remote access to the system from a web page. The primary differences, however, occur with the improvements upon the already existing designs. The A.B.C. system will be all inclusive, requiring no additional purchases or installations to obtain extra features. This includes being independent of a smart home system. In the event of a main power supply failure, such as during a power outage, the A.B.C system

will contain a battery back-up system that transitions directly to main power with the usage of a uninterruptible power supply system. This ensures the system can be always operational whether the rest of the house has power or not. Stored energy can even alternatively be used to power other blinds in a network or other devices. -DB, ML

The ability to remotely control the blinds will deviate from current remote-based designs by implementation of a user application via a web page to control the blinds system directly from any location, without any additional features. This allows for users to adjust their blinds throughout the day without requiring proximity to the system or a remote-control device. A user would be able to login to any computer and adjust the variables of the A.B.C. accordingly, as well as view the current positioning and sensor data from this remote location. - DB, TK, ML

1.4 Marketing Requirements (DB, TK, ML, DN)

1. User can have hands free control of the blinds.
2. Blinds control is accessible from anywhere with internet connection.
3. Blinds system is adaptive for seasonal and daylight changes to provide home energy cost savings.
4. User has the ability for personalized control over blinds system.
5. System stores light and temperature data and keeps record of blind activity.

2.0 Engineering Analysis

2.1 Circuits (ML)

The primary circuitry involved with the A.B.C. system is a microcontroller connected with two motor drivers, each operating a separate motor, a power system coming from a standard US 120V AC outlet, with battery backup, and wired connections to necessary sensor hardware for measuring light and temperature. All components will be hardwired for both communication paths and for powering.

The overall system will have power provided from a standard US 120V AC outlet, as these are available in any typical building or household that the system would be installed in. A battery backup system will also be present, in the event of a power outage, so that the system can continue to function for a limited time until outlet power is restored. Since batteries are a DC

supply, however, in order for the system to run from a sole type of current, AC or DC, the AC input from the outlet must be converted to a DC power supply.

The choice to convert the AC input power from the outlet to DC appears to be a more logical option than converting DC battery power to an AC supply, since the system will also need to step the 120V AC supply down to a much lower voltage, and any power adapter circuits can be external from the system if necessary, so both stepping down the voltage and converting the source to DC power at the same time was analyzed further.

The essential components to manage the input power from the outlet source are a transformer and a rectifier circuit. Essentially, the transformer will step down the voltage from 120V AC to the desired voltage level using the ratio between coils on the two sides of the transformer with the equation

$$N1/N2 = V1/V2, \quad (1)$$

where N1 and N2 are the coil ratios and V1 and V2 are the input voltage and output voltage respectively. Once the voltage has been dropped by the transformer to a voltage that is desired, a rectifier circuit can be used to convert it to a DC voltage. There are both half wave and full wave rectifiers for converting AC to DC which both utilize diodes to eliminate the negative portion of the AC input. A full wave rectifier is slightly more complex than a half wave rectifier, but it also provides positive voltage the entire time, rather than only during the positive voltage cycles from the AC input voltage like the half wave rectifier. Finally, placing a capacitor at the output smooths out the voltage for powering the circuit at a DC level with a voltage ripple based on the capacitor value, but this is mostly constant and steady, with only a slight ripple.

A regulator is also needed as the last part of this circuit for providing power to the ABC system from the standard wall outlet. The regulator stabilizes the input voltage to ensure that it is the desired value for the microcontroller and other components to the ABC system. This is important to ensuring that the system does not receive too much power, which could damage or destroy the system.

In the event of a power outage, the battery backup will come into effect as mentioned previously. In order for this to occur without any halt in the system operation, an uninterruptible power supply (UPS) can be used. The UPS uses a transfer switch to quickly move from the main power supply to the battery power supply as necessary. If an interrupt occurs, the switch will activate, allowing the battery to begin powering the ABC system. While not running from battery

power, the supply may also be used to charge the batteries, allowing the UPS to not require the user to change the batteries after the event of a power outage or similar event that would lead to the use of the battery backup. The UPS requires battery cell protection circuitry to ensure that the batteries are neither overcharged nor over-discharged at any given time. Many batteries can be severely damaged if over-discharged, or in the event of overcharging, could even catch fire. Discharging batteries too quickly can also cause permanent damage to battery cells. In order to protect against this, a battery protection integrated circuit (IC) can be used. A battery protection IC can be connected to the battery cells to be charged along with two MOSFETs which will switch on and off in order to stop charging or discharging of the battery cells when the cells are at risk of an overcharging or over-discharging. Battery protection ICs can be matched to the desired maximum cell voltage such that it properly switches the MOSFETs on and off at the correct overvoltage and undervoltage thresholds of the chosen cell. The MOSFETs are selected based on the chosen battery protection ICs recommendations. This circuit is then placed in the UPS to ensure that the batteries stop charging once they achieve the threshold voltage, even while the system continues to draw power from the main power supply. Additionally, extended use of the backup system will not damage the battery, as they will stop supplying power when they reach the critical undervoltage threshold.

In order to have a useful battery backup system, the selection of battery to use for the battery backup was investigated. The battery selected must have a high energy density and low self-discharge rate. It is also advantageous to have a lightweight and small sized battery that can be easily placed inside the blind housing, and multiple cells can be placed together. Finally, the battery must be rechargeable, so that the system can continue to reuse the same batteries. Lithium ion batteries stand out amongst the many types of batteries as ideal for the ABC system, due to a number of these factors. Lithium ion batteries have a high single-cell voltage and a high energy density, with a much slower self-discharge than most other types of batteries available. A single cell generally has a voltage of 3.6 volts, meaning that very few cells would be needed to provide power to the system. They also have a self-discharge rate of between 0.35% to 2.5% per month, meaning that they can last a very long time if not used at all, which means that they are reliable at holding charge for a long period of time. The ability for the battery to be able to recharge additionally opens up the opportunity for the main power supply to recharge the battery backup when it isn’t in use, which could allow the user to almost never have to change the

battery, meaning continuous operation of the ABC system for a much longer lifetime. Compared to other battery types, such as lead acid batteries, the lithium ion batteries also have the advantage in size and weight, being considerably smaller and lighter, despite not having as high of a cell voltage.

In order to know how much power needs to be supplied to the system, the following equation can be used with DC supplies:

$$P = V*I, \quad (2)$$

where P is the power required per hour, V is the voltage supplied, and I is the current supplied per hour. Each component will have a power requirement per hour, based off the voltage and current draw that they need. For example, the motor power requirement can be calculated using the following equation:

$$P = T*w, \quad (3)$$

where T is the torque produced by the motor and w is the rotational velocity in radians per second. Most devices will have a current and voltage rating that power can be calculated from. In addition to this, the power needed will vary as each device will not be in continuous operation, but only when taking measurements or adjusting the system positioning. For example, the temperature sensor will only draw power when a reading is requested from the microcontroller, and thus will consume 0 power when not taking a reading. This is factored in to calculate how often the device will operate on battery backup to determine the battery rating in order to provide power for the course of a day. The following calculation can be used to determine how often the system will be used:

$$X \text{ uses/day} * x \text{ minutes/use} * 1 \text{ hour}/60 \text{ minutes} = \text{hours of usage per day.} \quad (4)$$

By knowing the hours of usage per day, the current rating for each device in amp-hours can then be used along with the voltage level required to determine the ratings required for the battery pack in order to remain operational for an entire day without the main power source by multiplying the power calculated for the device in Equation 2 by the hours of usage per day found from Equation 4.

The circuitry to connect the motor and motor driver with one another will be provided in the relevant data sheet for a selected motor and motor drive to ensure proper functionality. Motors have varying amounts of wires necessary to connect, as some have four, six, or even eight wires. The motor connection in this case will be for a bipolar stepper motor hybrid. This

means that the motor has four leads, with two driving the clockwise direction, and two driving the counterclockwise direction. These connect to the motor driver outputs of A0 and A1 and B0 and B1, which are sent high and low based on the direction bit and step bit commands from the microcontroller. The driver in turn will be connected to the microcontroller and power supply as listed in its respective data sheet. Similarly, the microcontroller, sensors, and communication modules will have proper wiring connections in their respective data sheets.

2.2 Electronics (DB)

The Automated Blind Control system will be dependent on the use of electronics to gather and distribute data to enable the system to analyze and make decisions. The electronics that the A.B.C system will primarily be using are a few different sensors for gathering light and temperature data and a microcontroller for the decision-making process and data storing. The overall thought process when researching what components would be useful for this project boiled down into several different requirements. The electronics that would be most beneficial for this design would be physically small, powered by low DC voltage, output digital or analog data for interpretation by the microcontroller, consume a low amount of power while in operation and are low in purchasing costs. Further discussions in this section are below focusing on the sensors that will be used for this project and the microcontroller discussion can be found in the Embedded Systems portion of this report.

Regarding the sensors that may be used for this project, there isn't much of a detailed approach for applying engineering analysis to them, but more of a compare and contrasting of different manufacturers and models that hit the general specifications above to see which fits the needs of the project the best. In taking this approach, it was established that there are many manufacturers that produce the electronics needed for this kind of project, but prior to the actual wiring and pretests of the prototype, understanding how these electronics work and their inputs and outputs is what must be finalized before purchasing.

The first example is the indoor temperature sensor needed to obtain the room temperature and transmit that reading to the microcontroller. While researching, low cost and a small physical size were the main filtering aspects. A few temperature sensors that have fulfilled those requirements and seem to be providing the output range and temperature accuracy needed are the LM35 Centigrade Temperature Sensor and the LM61 temperature sensor from Texas Instruments. The LM35 temperature sensor comes as an 8-pin SOIC (small outline integrated

circuit) which can be fitted onto a printed circuit board (PCB). This sensor operates on a DC range from 4-30V, only uses 60µA of current when operating and costs less than $1 USD. Most importantly, this type of sensor has a linear input temperature to output voltage scale that makes the interpretation of data very easy. Per every degree in Celsius that the temperature sensor reads, it outputs 10mV, such as if a room is 25°C (77° F) the sensor has an output of 250mV to the microcontroller.

The LM61 is also a valuable choice as it is a 3-pin small outline transistor (SOT), operates between 2.7-10V DC, draws 125µA current at maximum and is low in cost. The output is still an analog voltage linear with 10mV per degree Celsius, however, there is an offset of 600mV. Therefore at 0°C, the output of the sensors would be near 600mV, considering the product's accuracy. With the analog voltage input to the microcontroller from either of these sensors and proper programming with conversion equations, the data will be interpretated for manipulation of the blinds as well as stored for reference for the system owner.

The next form of data that is to be gathered for the operation of this design is the outdoor temperature. Ideally, obtaining the outdoor temperature similar to the indoor temperature would be the primary choice for ease of the project's operation. However, after more collaboration and discussion, it was determined that using a physical sensor on the outside of a window would not be logical. Providing power to the outdoor sensor and retrieving data from this sensor would require a physical path between the sensor and the housing of the window blinds. This would mean drilling through walls of the house to run wires to and from the outside and that is unpractical.

After some discussion, it was determined that the best way to obtain the outside temperature of the location where the A.B.C. system is installed would be taken care of in the software side of this project. With the combination of programming and an onboard Wi-Fi module in the housing of the A.B.C. system, the outdoor temperature would be obtained from the user setting their zip code on the website of the A.B.C. The temperature will then be pulled from an outside source and displayed on the webpage as well as used as a data input to the microcontroller. More specifics about the software behind gathering the outside temperature data will be discussed later in this report under the Computer Networks section.

The third main electronics portion of this project, specifically on the hardware side, is obtaining the intensity of sunlight hitting the area of the window. Sunlight is a major contributor

to why people use their blinds for both visibility and energy saving reasons. Taking the intensity of sunlight at the window is an important component of this project as it is a contributor in the heating of a room during the winter, when blinds can be open, and keeping the room cool in the summer, when blinds are closed. There are three ways of transferring heat: convection, conduction, and radiation. Sunlight through windows into a room is a form of radiation heat transfer. Radiation heat transfer is done not by the movement of particles through the air, but is done by infrared waves propagating from the sun to the Earth's surface. The absorption or reflection of these waves on a surface is what causes heat to be transferred from the waves to the bodies they hit (4). Infrared rays from the sun are in the wavelength range of 750nm to 2500nm, visible light is in the range of 400-750nm and ultraviolet rays are in the range of 300-400nm. Ultraviolet rays contribute more to damaging surfaces rather than providing heat transfer. UV rays are what causes humans to obtain skin and/or eye damage when exposed without using the proper protective gear such as sunglasses and sunscreen.

Knowing that sunlight will be a contributor in the operation of the A.B.C. system, it becomes imperative that this data be gathered effectively for input into the microcontroller for analysis. For this project, there are a few types of light sensors that can be utilized to obtain the information we are looking for and they boil down into being passive or active components. The difference between active and passive components is that active components need an external power source to operate, such as an operational amplifier, whereas passive components do not need an external power source, such as a resistor. An active sensor that would be useful for this design would be the OPT101 Monolithic Photodiode and Single-Supply Transimpedance Amplifier developed by Texas Instruments. This sensor can be used to obtain the kind of sunlight rays that are hitting the window that the A.B.C. system is installed on. The OPT101 has a spectral responsivity relationship between wavelength received and its output voltage. This analog output voltage will be transmitted into the microcontroller and with the proper programming, the data will be interpretated for manipulation of the blind system.

A passive component that can also be used for obtaining the light intensity data is a photoresistor or LDR (light decreasing/dependent resistance). A photoresistor is a photoconductive component where its resistance decreases as it is exposed to more light. With relevance to this design, the LDR can be used in a voltage divider circuit between a rail voltage,

such as 5V DC, and ground with the output voltage taken from after the photoresistor to the microcontroller. In testing, we can determine voltage output ranges that show if there are low, medium, or high levels of light hitting the LDR. These ranges can then be programmed in the A.B.C. and the voltage obtained from the photoresistor circuit output will tell the microcontroller what kind of light intensity the window is receiving. This information will then be interpreted for determining the positioning of the A.B.C. system.

The last signal that will be a part of the inputs of the system is the manual override input. This manual override will simply be a switch or pushbutton that sends an analog voltage signal to the microcontroller that indicates the user is locally at the A.B.C system and wishes to operate the system manually. In the programming of the A.B.C system, this manual override will take precedence over all current commands since it is at a higher desire than the other operations. This pushbutton allows the user to change the position of their system at any time they are not using the system's web site. A three-position switch is a logical choice in hardware for this manual override aspect that way the switch can either be used to lift the blinds, lower the blinds, or not operate the blinds at all.

2.3 Signal Processing (DB/DN)

As seen in Figure 3 of the Hardware Level 2 Block Diagram, the data transmitted from the inside temperature sensor will be an output of voltage, which is the form of analog data. For the microcontroller to have the power to understand this data, it must be converted into digital data. To do so, an analog-to-digital converter at the input of the microcontroller will be needed. In order to make sure the microcontroller chosen satisfies our requirements, a max temperature of about 110 degrees Fahrenheit, which is converted to about 43C°, was put into the below equation from the temperature sensor. The temperature sensor operates in a range of -55° C to 150° C. The equation for the conversion to voltage is:

$$V_{out} = 10mv/^\circ C \times T, \quad (5)$$

Where Vout is output voltage from the conversion, C is in degrees Celius, and T is the temperature in degrees Celsius of the environment. This shows that 43° C would be 430 millivolts by multiplying 10mv by 43. For 430 to be represented digitally we would need at least 9 bits or 2^9 which is 512. This shows us that their needs to be at least 9 bits for the

analog/digital converter. For simplicity of the microcontroller inputs, the output voltage of the light sensor for this design will be required to be no higher than the bits needed to convert the data from the temperature sensor. This will be obtained by either choosing the right active sensor or adjusting the passive sensor circuitry to ensure low voltage outputs to the inputs of the microcontroller. Further discussion of the microcontroller can be found later in this report in the Embedded Systems section.

2.4 Communications (TK/DN)

Communications for this project are relatively straight forward. The system needs a wireless communications protocol that allows for remote communications. In this application a Wi-Fi based system will work for the desired needs, as the commands could come from virtually anywhere with an Internet uplink. An ESP8266 module was found to be able to handle this type of connection. The ESP8266 is a System on a Chip (Soc) manufactured by Espressif. It contains a Tensilica L106 32-bit micro controller unit and a Wi-Fi transceiver. The ESP8266 must support 802.11b protocol to allow for Internet communications to send data to our user application. For the module to be powered, the voltage from Vdd will need to be converted to 3.3 volts instead of 5 volts. The Wi-Fi connection needed will be discussed in the Computer Networks section (2.6) below. For the module to communicate with the microcontroller, UART communications will be to be configured first. This includes setting the baud rate, which is 115200 for the ESP8266. The Wi Fi module should communicate with the main microcontroller via a 2-wire serial communication. The ESP8266 interacts with the microcontroller on URART using AT instructions. The microcontroller can send the ESP8266 a character string instruction and the ESP will react based on predefined instruction set. The instruction set can be found below in Reference [9] A depiction of this setup can be seen below in Figure 5. The main micro controller must also interface with two motor controllers, both via serial communication. The remaining temperature and light sensors communicate via serial or I2C.

2.5 Electromechanics (ML)

The system utilizes two motors and motor drivers to control the blind movement. In order to raise and lower the blind assembly, one of the motors will spool the strings connecting each slat. In order to do this, the motor must be capable of generating the required torque to overcome gravity as well as lift the blinds to their maximum height. A motor capable of lifting the blind assembly will also be capable of lowering it. In order to calculate the necessary torque, the force

on the blinds downward due to gravity must be calculated. This can be done with the following equation:

$$F = mg, \tag{6}$$

where F is the force exerted on the blinds by gravity, m is the mass of the blinds, and g is the acceleration due to gravity. The torque produced by the motor must include a force greater than the force exerted on the blinds due to gravity in order to lift the blinds. In order to calculate the torque produced by the motor, the following equation can be used:

$$T = rF, \tag{7}$$

where T is the torque produced by the motor and r is the radius of the motor shaft. In order to produce a higher torque, the radius can be increased using a gear ratio:

$$T_2/T_1 = n_1/n_2, \tag{8}$$

where T_1 is the torque produced by the motor and T_2 is the torque stepped up by the gear ratio. The gear ratio is represented by n_1/n_2. The chosen motor must be able to handle torque greater than the torque needed to oppose the force exerted by gravity on the blind assembly. This holding torque will keep the blinds from falling when completely raised, though designing the mechanical spool which holds the strings for the blinds could be done in order to hold the blinds when the motor is not energized.

The motor must also be capable of lifting and lowering the assembly quickly. This can be calculated by knowing the diameter of the shaft of the motor. With the diameter of the motor, the circumference can be found using the equation:

$$C = \pi d, \tag{9}$$

where C is the circumference and d is the diameter. Using the circumference, it is possible to calculate how fast the motor needs to rotate in order to lift the blinds up and down. By knowing how fast the motor rotates, in revolution per minute, rpm, the distance the entire assembly moves can be found. One revolution of the motor will move the string which lifts the blinds by a distance equal to the circumference. Thus, the linear distance traveled over the course of a minute can be calculated with the following equation:

$$y = Cw, \tag{10}$$

where y is the linear distance traveled and w is the speed in rpm. This speed can also be modified with a gearbox, without raising the rpm, as described previously.

While any type of motor can perform all of the above requirements, not all types of motors are capable of fulfilling another important feature of the lateral motor: that it is easy to know what position the blinds are at without additional hardware such as limit switches, which would add additional complexity, cost, and power consumption. DC motors do not have any way to determine how far they have traveled, however, there are a couple of options with motors in order to do this, such as stepper motors and servo motors. Servo motors are generally used in higher cost and higher performance motion implementations. This is due to the need for a position sensor, typically an encoder, and an error amplifier, both of which increase complexity. If rapid acceleration or high precision and accuracy were required, a servo motor would also be advantageous, but as this is not the case, the simpler stepper motor provides all of the same needs, such as the required accuracy, speed, and torque, but also typically for a lower price. Stepper motors also move one step at a time, which allows it to determine how far it has rotated, based on the number of steps per revolution. Each step can also be translated to an exact degree angle, with some error factor. A stepper motor comes with the number of steps it has per revolution, and the angle per step can be calculated using the following equation:

$$\text{Angle} = 360/(\text{number of steps}), \tag{11}$$

For example, a stepper motor that has 200 steps per revolution would take a step of 1.8 degrees each time it receives power. Many stepper motors also have the option to take half-steps, or quarter-steps as well, which can be controlled by the motor driver based on inputs from the microcontroller. These divisions of the steps can allow stepper motors to reach even higher accuracy, which is beneficial to controlling the blind slats to specific angles.

The rotational motor uses similar principles; however, the motor does not need to produce torque to combat the force of gravity. Therefore, the important equations for the rotational motor are the linear speed and the angles available based on the steps provided by the stepper motor.

Both the rotational and the lateral motors require a separate motor driver in order to ensure that they operate as desired and aren't overloaded with current. Some features of the motor drivers that need to be considered are the operating voltage range of the motor, the max continuous current per phase, and the micro-step resolutions available. The motor drivers must match to the motors being used and must be able to be wired together with the motor properly as described by the motor chosen. Another important feature when choosing the motor drivers is

that they can communicate with the microcontroller and have a simple interface with it. Ideally, a serial communication from the microcontroller to the motor drivers would allow for a simple interface that can provide all the necessary information to pass back and forth between the drivers and the microcontroller such that the drivers direct the motors as desired by the microcontroller and the drivers provide the microcontroller with accurate position data. In order to conserve power, the motor drivers will also have to handle low power motors.

2.6 Computer Networks (TK/DN)

With Wi-Fi chosen as the communication protocol used to interface to the internet, the system will communicate via Wi-Fi module to a local router. This router should have access to an Internet uplink. This allows for commands to be sent from any device via website. This allows for data to me sent from the Wi-Fi module to the router or the Internet to the Router. The Wi Fi router acts as the gateway from one system to another. There will be a web application that will be the source on the internet receiving the information from the Wi-Fi module. A preliminary version of the webpage can be seen below in Figure 12. On the webapp, users can see the most recent state the system was in as well as last sampled temperature and light values. The system should be able to send the sensor information to a computer on the local network. The main send and receive functions can be seen in Figure 09. This PC can then compile the sensor data and store it in a database. The database will then be able to be view up to a certain day on the web application. The framework of the application and a choice of web service will be used to run the application. Explored solutions currently include using Python Flask to write a web API. This API will be configured from the ESP8266 in HTML. This will then be displayed in format suitable for a web browser. In order to obtain the outside temperature, a weather API will be used. A website called OpenWeather allows for users to create an account and get a personal key that can be used to call to the website, and it will send back the needed information for the desired zip code. This weather API was chosen because of its easy ability to grab the current weather data for free. This was an easier method then an outside sensor once we were able to learn that we could obtain the temperature online and because we were not looking for the exact temperature right outside a window. IEEE 802.11b will be used as the network standard for this system.

2.7 Embedded Systems (TK/DN)

An embedded system must be developed to satisfy the following needs. First a microprocessor must be used to process signals from temperature, infrared and light sensors. These sensors are analog, so the need of a minimum 3 channel A/D convertor is needed. In the previous section it was proven that a 10-bit A/D convertor will be enough to satisfy the needed conversions. The microcontroller must also be able to interface with two motor controllers as well as have support to communicate with a Wi Fi module. The ability to send data over serial communication as well as I2C is mandatory. A minimum of 7 IO ports are needed. A low power processor is also important, as the battery system will only power a high draw system for a brief period. A microchip with a sleep mode would be ideal since the system sits idle for most of the day. In normal operation, the system only wakes up twice an hour in the 16-hour window, adjusts the system, and goes back to sleep. Surveying various projects of a similar degree, a 16-bit micro controller is needed with a minimum of 50KB program storage. The microcontroller will be programed using MPLab XIDE, which is based on the open-source NetBeans platform.

3.0 Engineering Requirements Specification (DB, TK, ML, DN)

Table 1: System Engineering Requirement Table

Marketing Requirements	Engineering Requirements	Justification
1,2	System will communicate wirelessly with external devices and be controlled via the Internet.	Wireless communication between user and the A.B.C. system enables hands free control and access from anywhere outside of the physical location of the system and is based on the functionality of commercially available WIFI modules.
1	Blinds assembly will raise and lower via motor control within 1 minute.	This is based on commercial stepper motor ratings for speed and step count per rotation.
1	Individual blind slats will rotate via motor control in less than 5 seconds, in less than 15-degree increments.	This is based on commercial stepper motor ratings for speed and step count per rotation.
3	System will measure inside and outside temperature every half-hour in a 16-hour window (6am-10pm).	Controlling the sunlight entering a room contributes to the heat transfer between the room and the outdoors. Depending on

		difference between indoor and outdoor temperature, the blinds should adjust.
3	System will measure sunlight intensity with focus on infrared and UV wavelengths in the range of 250-1000nm.	Infrared and UV rays from the sun are most damaging to the Earth's surface and to human eyes and skin and are all contained within this range of wavelengths on the light spectrum.
3	Based on season, system will open or close blinds to minimize or maximize sunlight in room to conserve energy costs.	Limiting or allowing sunlight to enter a room will contribute to the heat transfer through the window during months where the room is desired to be cooler, such as summer, and months where it is desired warmer, such as winter.
1	System will contain battery backup for minimum 16 hours of operation time in case of outlet power loss.	User remains in control of A.B.C. system even if home power is disconnected for at least one full day of operation, which is 16 hours.
4,5	System position and sensor values will be displayed on a web application with a precision of 1 decimal point.	Knowing current data of the A.B.C. system will allow the user to manipulate the system if desired as well as adjust future settings based on current information. The display should be accurate to the actual data without being overwhelming for the user.
4	System will allow the user to fully raise or lower the blind assembly via a switch or pushbutton.	Manual override is pertinent to allow the user to have total control of the system at any desirable time.
5	System will store light, temperature, and time data after each measurement for user reference.	Data will be accessible by the user to adjust future settings and action taken by the autonomous system.

Marketing Requirements

1. User can have hands free control of the blinds.
2. Blinds control is accessible from anywhere with internet connection.
3. Blinds system is adaptive for seasonal and daylight changes to provide home energy cost savings.
4. User has the ability for personalized control over blinds system.
5. System stores light and temperature data and keeps record of blind activity.

4.0 Engineering Standards Specification (DB,TK,ML,DN)

4.1 Safety

ANSI/WCMA A100.1-2018

IEC 62061

UL 50

NFPA 70

4.2 Communication

IEEE 802.11

I2C

Serial

4.3 Data Formats

ISO 8601

4.4 Design Methods

MPLab XIDE

4.5 Programming Languages

NetBeans

HTML

CSS

SQL

Python

4.6 Connector Standards

NEMA 5-15P

5.0 Accepted Technical Design

5.1 Hardware Design (DB,ML)

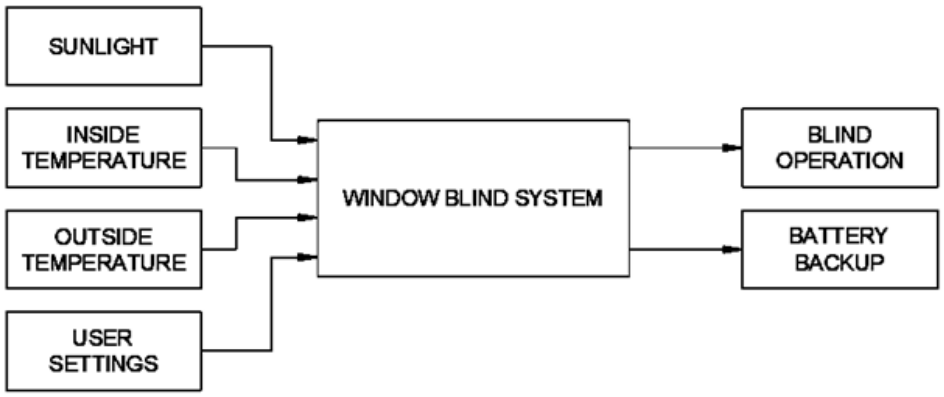


Figure 1: System Level 0 Block Diagram

Table 2: System Level 0 Block Diagram Functional Requirement Table

Module	Automated Window Blinds
Designers	W. Daulton Baksa, Tim Kurczewski, Matthew Lacek, Daniel Nahra
Inputs	Sunlight: Infrared and visible light waves from the sun Inside Temperature: Temperature measured inside the building Outside Temperature: Temperature measured outside the building User Settings: User choice for open/close for desired times, seasonal operation, and manual open/close
Outputs	Blind Operation: Blinds move to desired position based on automatic or manual inputs Battery Backup: Secondary power supply that activates when main power supply fails
Functionality	Automated light shade for a window. Allows users to choose operating mode with wireless control.

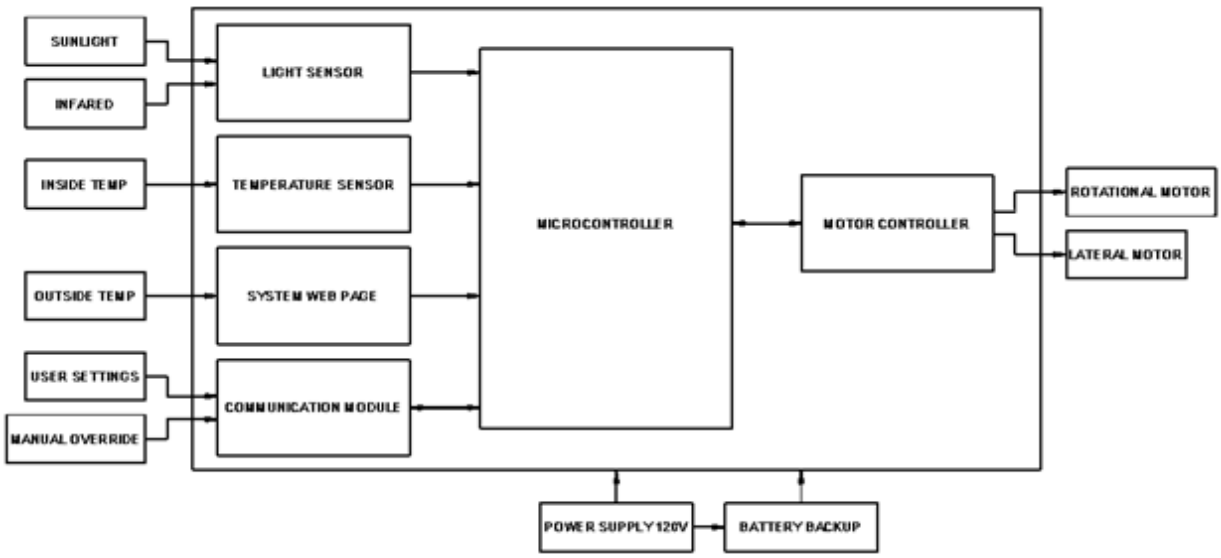


Figure 2: Hardware Level 1 Block Diagram

Table 3: Hardware Level 1 Block Diagram Functional Requirement Table

Module	Light Sensor
Input	Sunlight intensity and wavelengths in nanometers
Output	Voltage signal corresponding to input
Functionality	Obtain incoming sunlight and interpret it into an output electronic signal for the microcontroller
Module	Temperature Sensor
Input	Temperature in degrees Celsius
Output	Voltage signal corresponding to input
Functionality	Obtain inside and outside temperatures and interpret them into output electronic signals for the microcontroller
Module	System Web Page
Input	User settings and system's area code location
Output	Status of the system and outdoor temperature
Functionality	Main source of controlling the system as well as displaying current system data to the user
Module	Communication Module

Input	User commands from wireless external device and data from microcontroller
Output	Data sent to microcontroller and data sent to user application.
Functionality	Allows user to send commands to the system via a wireless connection to the microcontroller, as well as send data back to user from the microcontroller
Module	Microcontroller
Input	Digital signals from sensor modules, motor drivers, and communication module
Output	Signals to motor drivers and communication module for blind positioning and data storage. Waking sensors to read data.
Functionality	Reads and writes commands to all modules in the system and makes the logic decisions for blind positioning based from signal inputs
Module	Motor Controllers
Input	Position signals from microcontroller for lateral and rotational motors
Output	Voltage and current supply to motors and position signal to microcontroller
Functionality	Sends signals and power to the motors to reach desired positioning as well as sending signals to microcontroller to let microcontroller know the current system position
Module	Rotational Motor
Input	Current and voltage from motor controller

Output	Motor shaft rotation
Functionality	Turns the blind slats to desired angle
Module	Lateral Motor
Input	Current and voltage from motor controller
Output	Motor shaft rotation
Functionality	Raises and lowers blind assembly to desired height
Module	Battery Backup
Input	120 V AC power supply through AC-DC conversion
Output	Power needed to power system when main supply fails
Functionality	In the event that the main power supply is not on, battery backup will provide a limited period of power to the system
Module	120V Power Supply
Input	120V, 60Hz AC voltage from standard US wall outlet
Output	DC voltage to system.
Functionality	Normal/main power supply to the entire system.

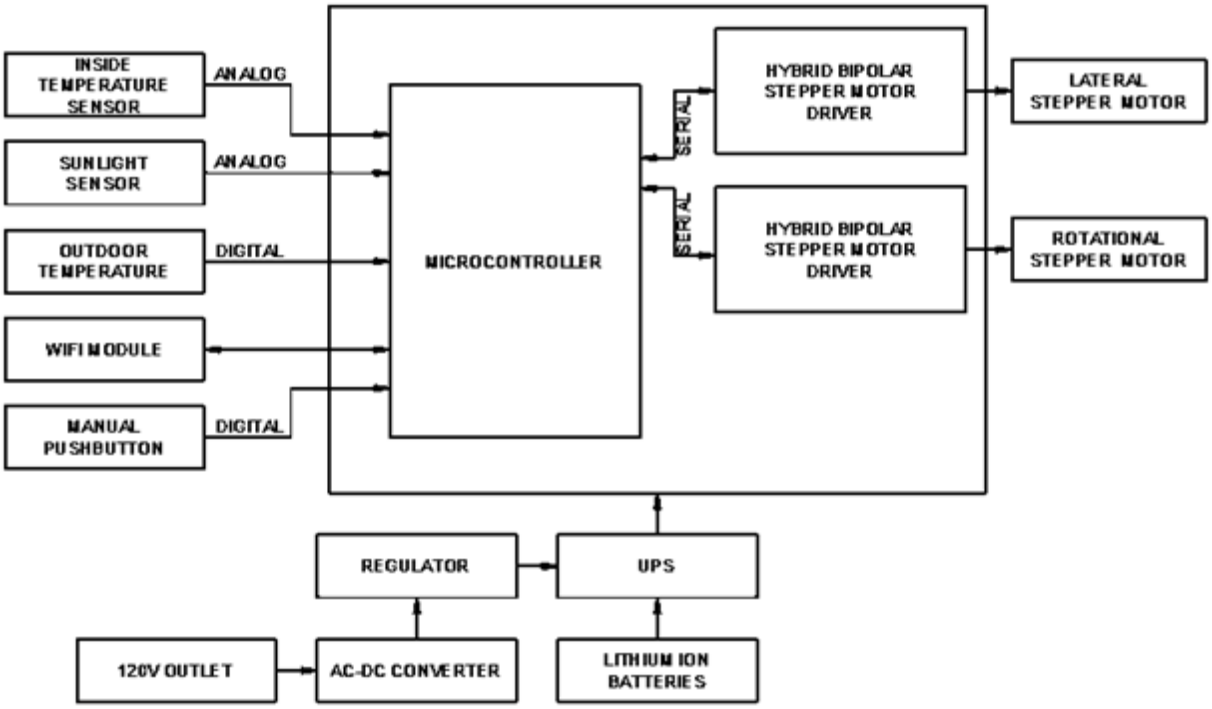


Figure 3: Hardware Level 2 Block Diagram

Table 4: Hardware Level 2 Block Diagram Functional Requirement Table

Module	Sunlight Sensor
Input	Sunlight infrared and UV wavelengths in nanometers ranging from 250-1000nm
Output	Voltage signal corresponding to infrared and UV wavelength input
Functionality	Obtain incoming sunlight and interpret it into an output electronic signal for the microcontroller
Module	Inside Temperature Sensor
Input	Room temperature in degrees Celsius

Output	Voltage signal (10mV per °C) corresponding to input i.e. 25°C = 250 mV output
Functionality	Obtain inside room temperatures and output electronic signals for the microcontroller interpretation.
Module	Outdoor Temperature
Input	Temperature in degrees Fahrenheit from internet
Output	Temperature value as digital data
Functionality	Obtain outside temperature and interpret them into output electronic signals for the microcontroller.
Module	WIFI Module
Input	User commands from wireless application and data from microcontroller.
Output	Data from user's web application is uploaded to microcontroller. Data from microcontroller is uploaded to user's web application for storage.
Functionality	Allows user to send commands to the system via a wireless connection to the microcontroller, as well as send data back to user from the microcontroller
Module	Manual Pushbutton
Input	ON or OFF position (logic 0 or 1)
Output	High or Low digital voltage signal
Functionality	Allows for manual override of the system at the system's direct location that takes precedence over other input signals and decision making.

Module	Microcontroller
Input	Inside and outside temperature, sunlight intensity, web application control, manual pushbutton control, motor driver lateral position and motor driver rotational position
Output	Wakes sensors for data gathering, wakes Wi-Fi module for data transmitting and receiving, energizes motor drivers for motor operation and blind positioning.
Functionality	Performs logic decisions based on input data for optimal blind positioning and transmits recorded data to user application.
Module	Hybrid Bipolar Stepper Motor Driver for Rotational Motor
Input	Position signal from microcontroller for the rotational motor.
Output	Voltage and current supply to motor until desired number of steps have been completed and position signal of the motor to microcontroller.
Functionality	Sends signals and power to the motor to reach desired positioning as well as sending signals to microcontroller to let microcontroller know the current system position.
Module	Hybrid Bipolar Stepper Motor Driver for Lateral Motor
Input	Position signal from microcontroller for the lateral motor.
Output	Voltage and current supply to motor until desired number of steps have been completed and position signal of the motor to microcontroller.
Functionality	Sends signals and power to the motor to reach desired positioning as well as sending signals to microcontroller to let microcontroller know the current system position.

Module	Rotational Motor
Input	Current and voltage from motor controller.
Output	Motor shaft rotation number of steps with each step equivalent to 1.8 degrees with precision of at least 15 degrees
Functionality	Rotates the blind slats to desired angle in ? degree increments.
Module	Lateral Motor
Input	Current and voltage from motor controller.
Output	Motor shaft rotation with torque of at least 5 mNm to lift or lower the blind assembly in under a minute.
Functionality	Raises and lowers full blind assembly to desired height within the entire length of the window.
Module	Uninterruptable Power Supply
Input	12V DC from main power supply
Output	14.4 V DC and 3.5 Ah current from Lithium Ion Batteries
Functionality	When the main power supply is no longer powering the system, the UPS switches power to the battery backup until main power returns.
Module	Lithium Ion Batteries
Input	4 standard 3.70V DC per cell batteries connected in series.

Output	14.4 V DC with 3.5Ah of supply power as the backup power system for a 16 hour duration.
Functionality	Provides enough power for the system over the course of a 16 hour period during which the main power supply is unavailable.
Module	120V Power Supply
Input	120V, 60Hz AC voltage from standard US wall outlet
Output	DC voltage to system
Functionality	Normal/main power supply to the entire system.
Module	AC/DC Converter
Input	120 V AC, 60 Hz Voltage
Output	12V DC primary rail voltage
Functionality	Takes the AC voltage from the outlet and converts it to a lower DC voltage for powering the system
Module	Regulator
Input	Unstable fluctuating DC voltage from the AC/DC converter.
Output	12V DC voltage at a steady rate.
Functionality	To ensure that the DC input voltage remains consistent for necessary system power.

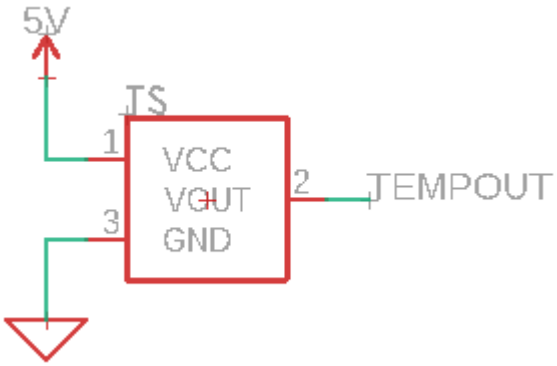


Figure 4: Temperature Sensor Schematic

***Description**: The temperature sensor receives 5 V input and a ground pin. It reads the temperature and outputs a voltage signal to the microcontroller which is interpreted using A/D conversion on the microcontroller.*

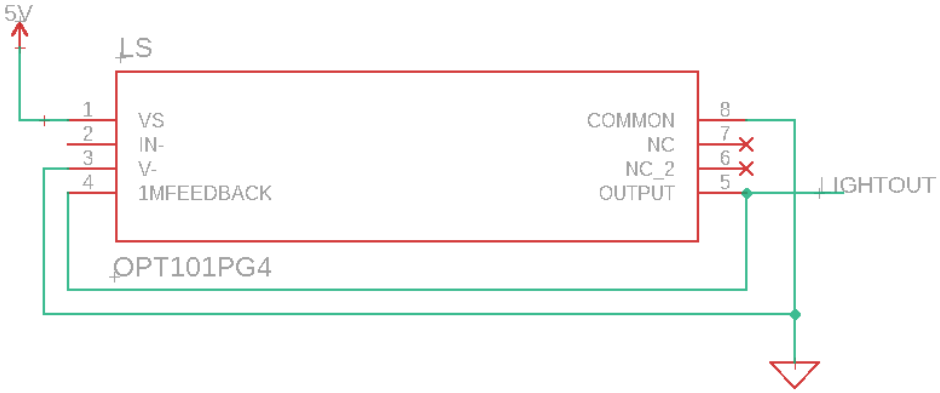


Figure 5: Light Sensor Schematic

***Description**: The light sensor receives 5 V input and a ground pin with pins 3,4,5, and 8 wired as shown. It absorbs the light against the window and outputs a voltage signal to the microcontroller which is interpreted using A/D conversion on the microcontroller.*

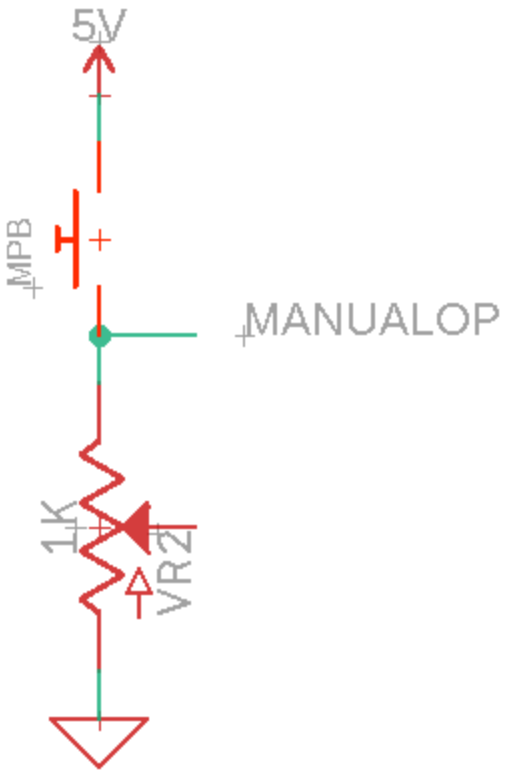


Figure 6: Manual Pushbutton Override Schematic

***Description**: The manual pushbutton signals to the microcontroller when the user presses the pushbutton by driving the ManualOp pin high. A pull-down resistor keeps the signal low when the push-button is not pressed.*

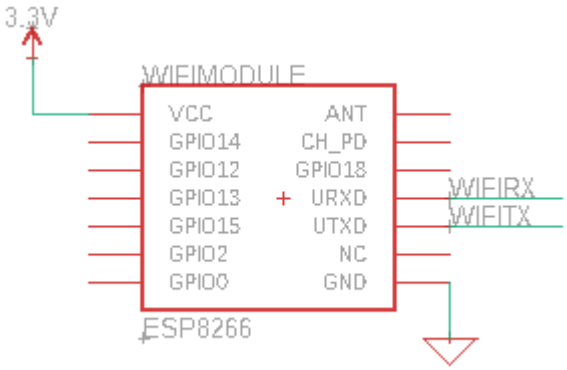


Figure 7: Wi-Fi Module Schematic

***Description**: The Wi-Fi module recieves a 3.3 V power supply and communicates with the microcontroller via UART communication.*

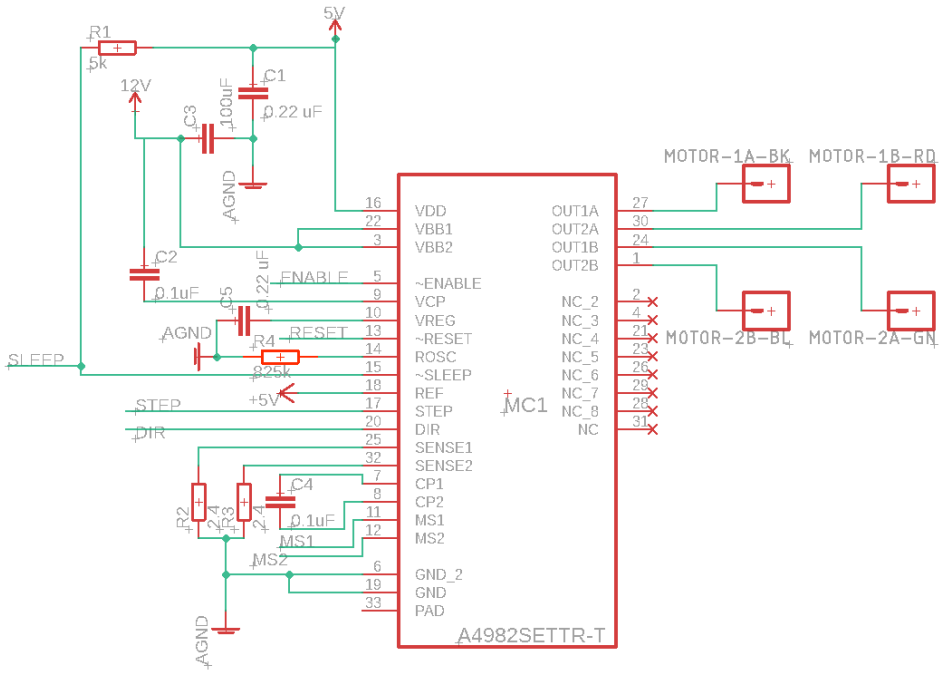
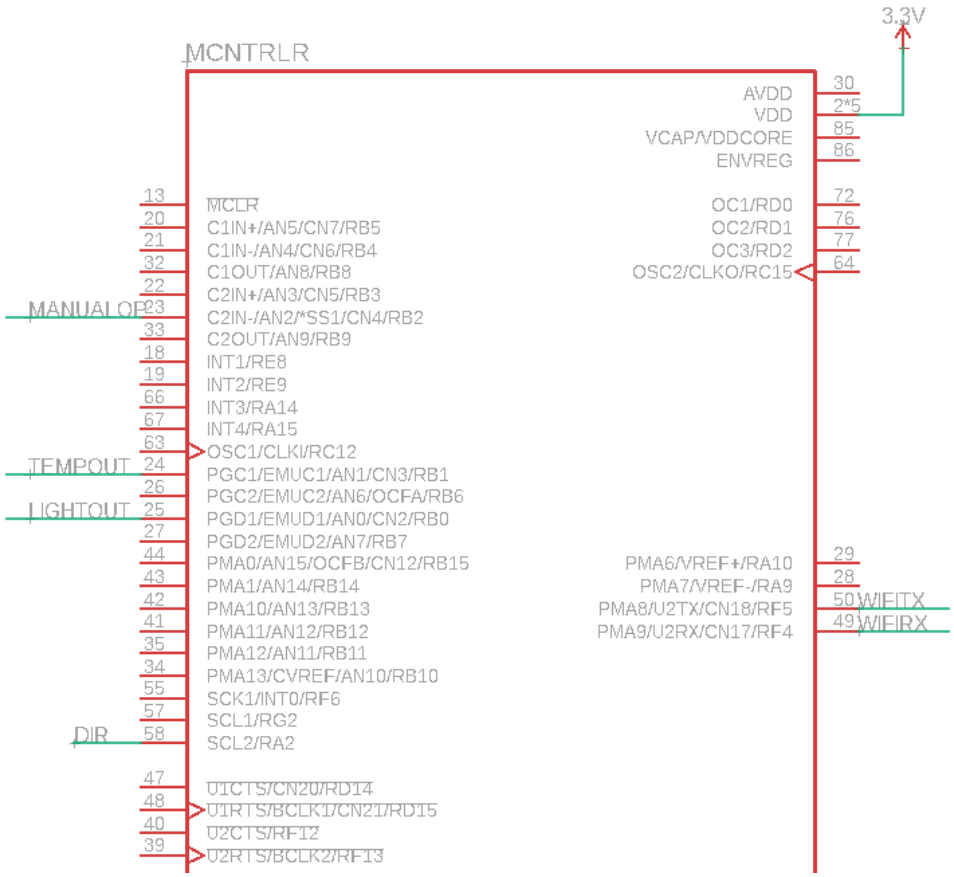


Figure 8: Motor Controller Schematic

***Description**: The Motor Controller receives input signals from the microcontroller for STEP, SLEEP, and DIR, as well as for microstepping with the MS1 and MS2 signals. The motor controller recieves the input main power rail, and provides the appropriate voltage and current necessary to drive the bipolar stepper motor from the main power rail using Sense resistors to limit the current. This circuit will be applicable for both the horizontal and vertical motor controllers.*


3.3V
MCNTRLR
AVDD
VDD
VCAP/VDDCORE
ENVREG
30
2*5
85
86
13
MCLR
20
C1IN+/AN5/CN7/RB5
21
C1IN-/AN4/CN6/RB4
32
C1OUT/AN8/RB8
22
C2IN+/AN3/CN5/RB3
MANUALOP
23
C2IN-/AN2/*SS1/CN4/RB2
33
C2OUT/AN9/RB9
18
INT1/RE8
19
INT2/RE9
66
INT3/RA14
67
INT4/RA15
63
OSC1/CLKI/RC12
TEMPOUT
24
PGC1/EMUC1/AN1/CN3/RB1
26
PGC2/EMUC2/AN6/OCFA/RB6
LIGHTOUT
25
PGD1/EMUD1/AN0/CN2/RB0
27
PGD2/EMUD2/AN7/RB7
44
PMA0/AN15/OCFB/CN12/RB15
43
PMA1/AN14/RB14
42
PMA10/AN13/RB13
41
PMA11/AN12/RB12
35
PMA12/AN11/RB11
34
PMA13/CVREF/AN10/RB10
55
SCK1/INT0/RF6
57
SCL1/RG2
DIR
58
SCL2/RA2
47
U1CTS/CN20/RD14
48
U1RTS/BCLK1/CN21/RD15
40
U2CTS/RF12
39
U2RTS/BCLK2/RF13
OC1/RD0
OC2/RD1
OC3/RD2
OSC2/CLKO/RC15
72
76
77
64
PMA6/VREF+/RA10
PMA7/VREF-/RA9
PMA8/U2TX/CN18/RF5
PMA9/U2RX/CN17/RF4
29
28
50
49
WIFITX
WIFIRX

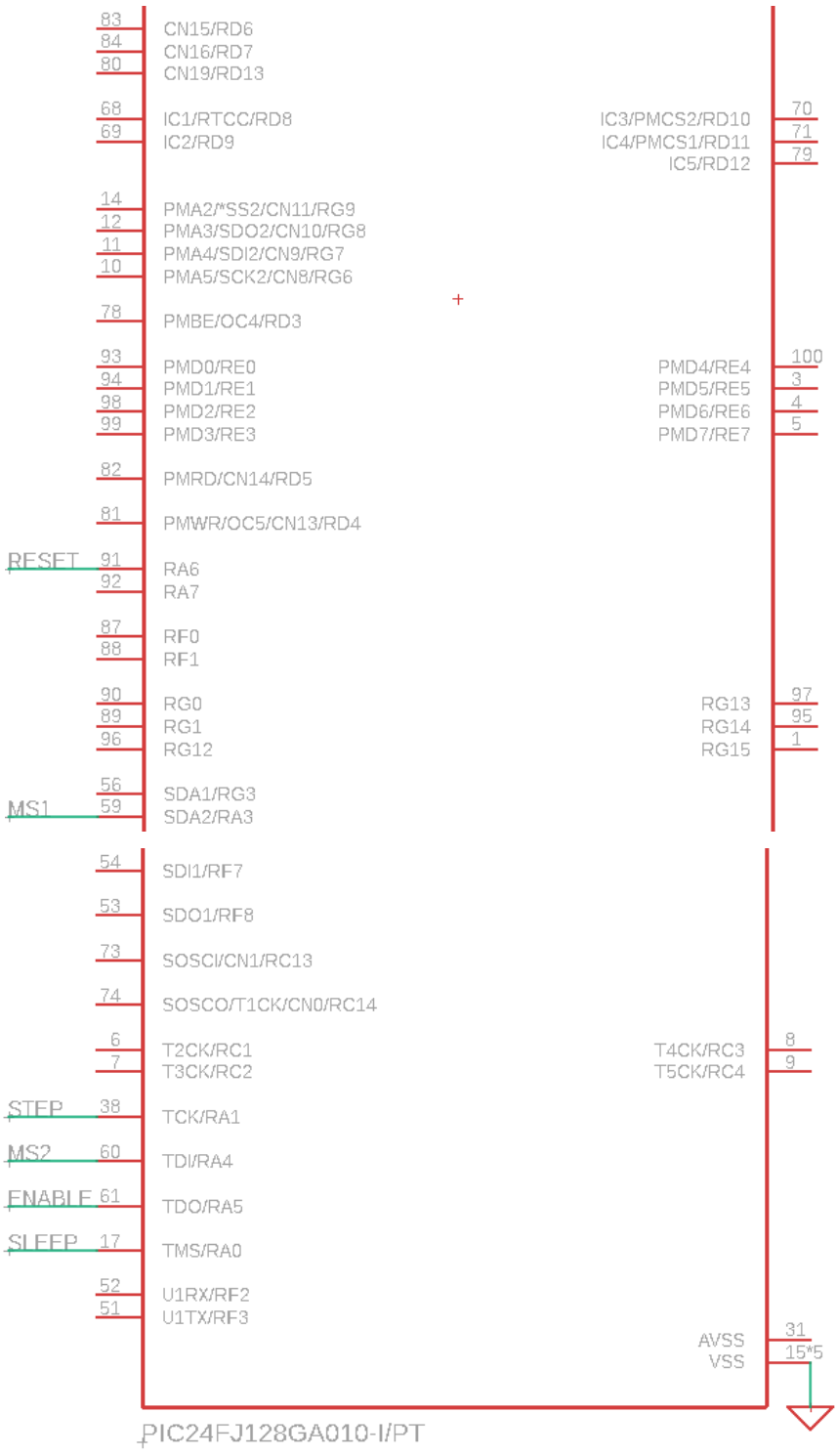


Figure 9: Microcontroller Schematic

***Description**: The microcontroller connects to each of the other devices. It sends out high signals determining when the motors rotate based off of the STEP, DIR, and SLEEP pins on the motor*

controller. The microcontroller also reads signals from each sensor and communicates via UART protocol with the Wi-Fi module.

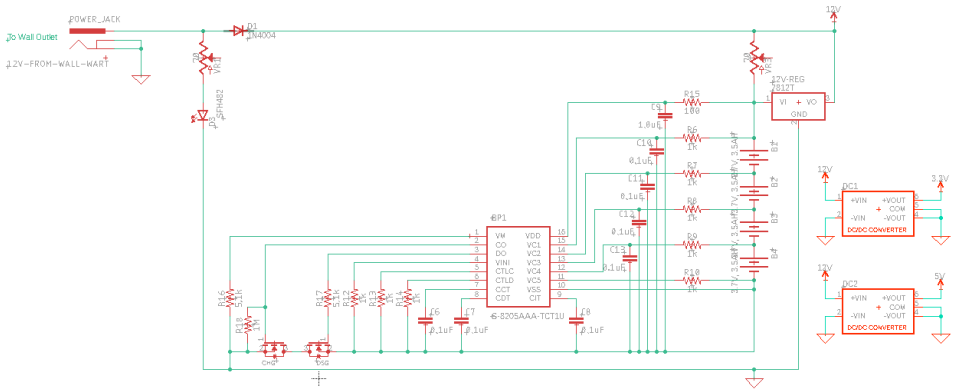


Figure 10: UPS and Rail Voltage Schematic

***Description**: The uninterruptable power supply will consist of (4) 3.7V, 3.5AH lithium-ion batteries in series that will be trickle charged via the voltage output of the wall wart. This charging will be monitored by a protection IC and the output voltage of the batteries will be regulated via the 7812T regulator. The 3.3V and 5V rails needed throughout the system will be established by DC-DC converters taken from the main 12V rail.*

5.2 Software Design (TK,DN)

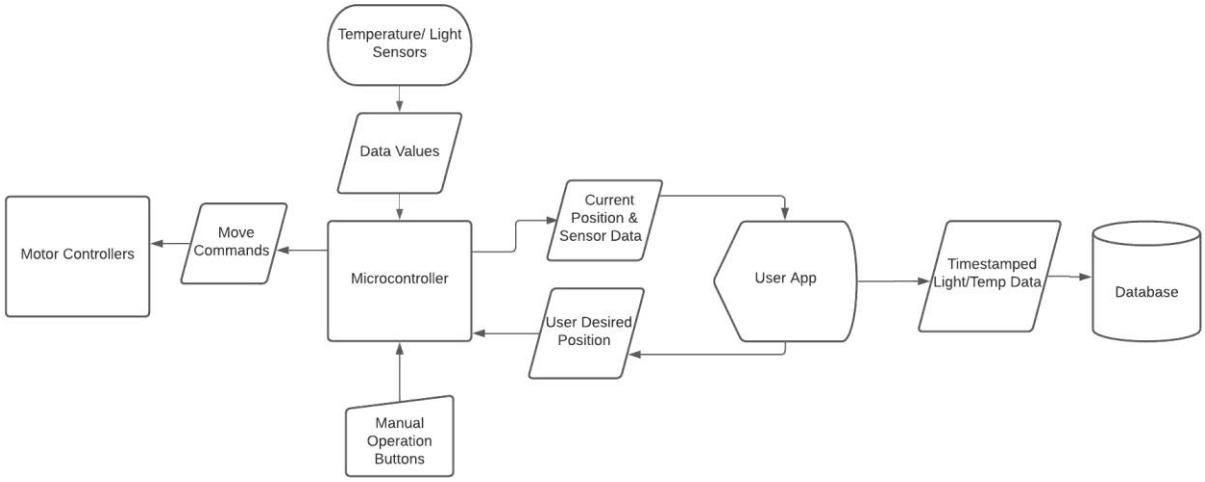


Figure 11: Software Level 1 Flow Chart

***Description**: The microcontroller will be receiving values from the Temperature and Light Sensor where then their data values can be converted and then understood by the controller. Once the microcontroller decides if it needs to do anything, it will send move commands to the motor controllers that have access to the motors to open, close, raise or lower the blinds. Then the microcontroller will also send the data and the position of the blinds to the User Application that will have a data storing system so the user can view data from previous days. The user can then also send commands that will be taken by the controller to change the motors if needed. Finally, if a manual Button is pushed then the microcontroller will decide if it needs to change the position or not. (DN)*

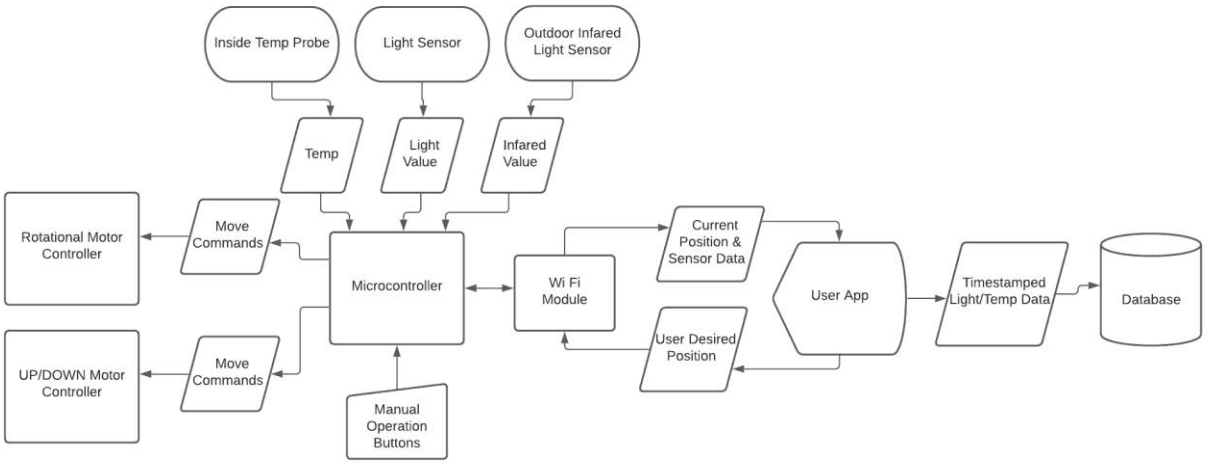


Figure 12: Software Level 2 Flow Chart

***Description**: The microcontroller sends messages to the User App through a Wi-Fi-Module. The temperature and light sensors have also been separated. The microcontroller also sends messages to two motor controllers now instead of just one. One motor controller for up/down movement, and one controller for rotational movement of the blinds. (TK)*

Table 5: Software Level 2 Functional Requirement Table

Module	Input	Output
Rotational Motor Controller	Rotation Commands	Rotates blind position between OPEN, CLOSED, and 45 OPEN (5V signal)

UP/DOWN Motor Controller	Movement Commands from microcontroller	Moves blinds UP or DOWN Blinds must be in closed position to operate (5V signal)
Microcontroller	Temperature value (Analog) Light Level Value (Analog) Outside Temperature value from User App User requested Movement commands from User App	Sends movement commands to motor controllers (5V Signal) Sends current state to User App Sends temperature and sensor values to User App
Inside Temperature Probe	Temperature level is measured on device	Sends temperature value to microcontroller
Light Sensor	Light Value measured from window pointing outside	Sends light level value to microcontroller
Infrared Light Sensor	Infrared radiation measured from window pointing outside	Sends infrared value to microcontroller
User App	Current position of blinds • Up/Down Displayed • Closed/Open Displayed Current Sensor Values - Inside Light Value Displayed - Inside Temperature Value Displayed - Infrared Light Value Displayed User interacts and can click buttons on screen to change system state	Sends movement commands to microcontroller Sends Data to Database (every 30 minutes)
Manual Operation Buttons	Process signal from button	Send electrical signal to microcontroller (5V)
Database	Timestamped Data from User App Light, Temp, Infrared values	View database data from a laptop or other device with internet access
Wi-Fi Module	Commands from User App Data from microcontroller	Data sent to Microcontroller Data sent to User App

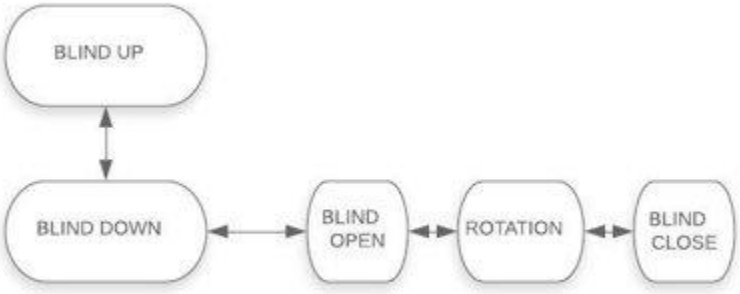


Figure 13: System State Machine Diagram

Description: *The system will have 4 main states: Blind up, Blind Down, Blind Open and Blind Close. The blind will make incremental steps in between open and close. The blind will only open or close if it is down first. And the Blind can only go to the Up position of the blind slats are in the open position. (TK)*

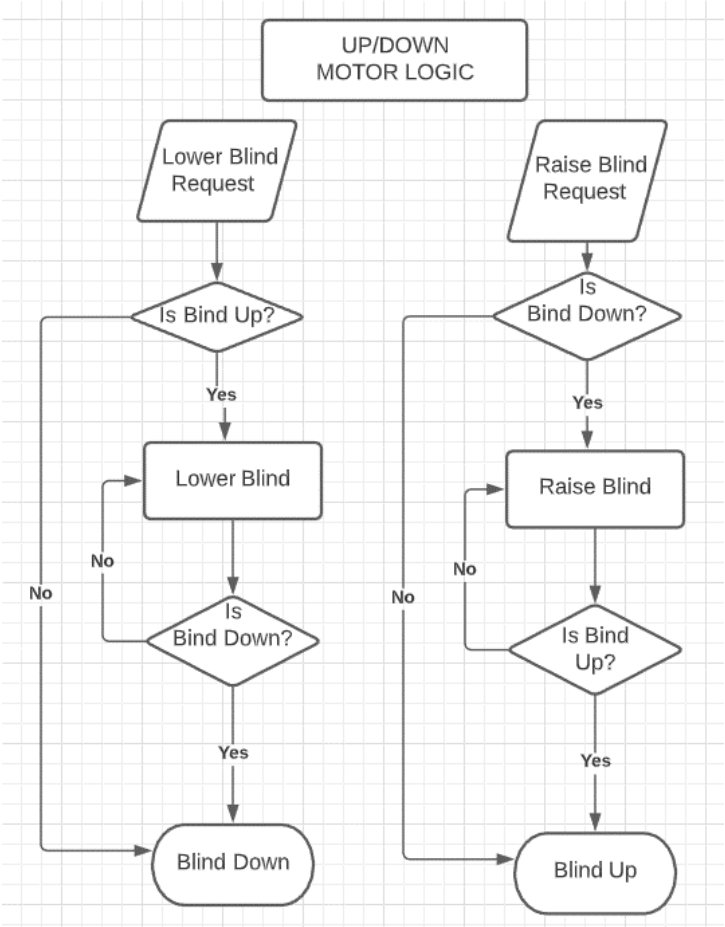


Figure 14: UP/DOWN Motor Controller Flow Diagram

Description: *Main motor movement decision making. The microcontroller will process the input from webpage, then call lower or raise commands to move the blind accordingly.*

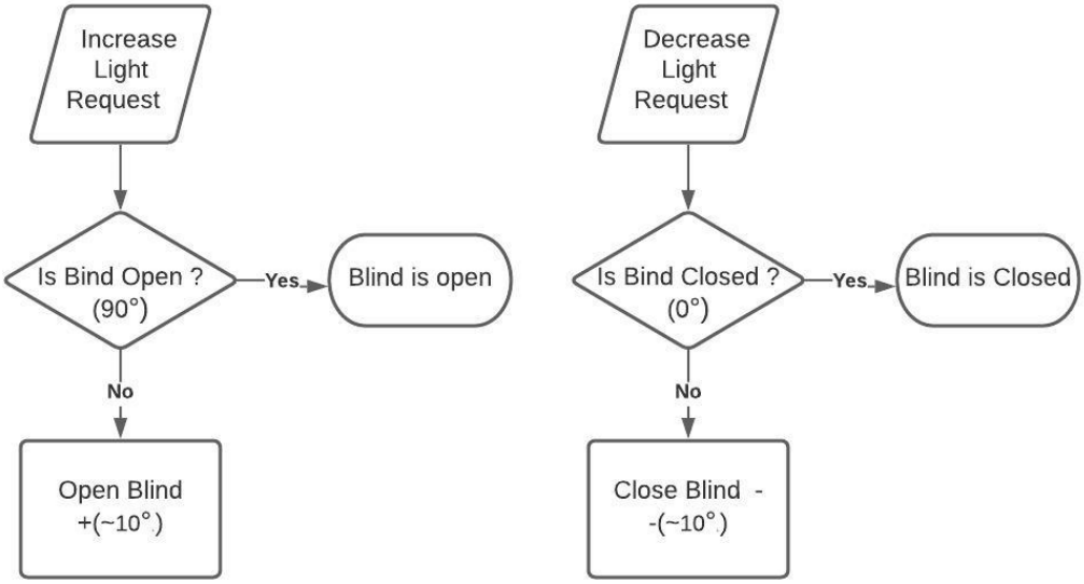


Figure 15: Blind Rotation Motor Controller Flow Diagram

Description: *When ABC receives a request from webapp asking for blind rotation*

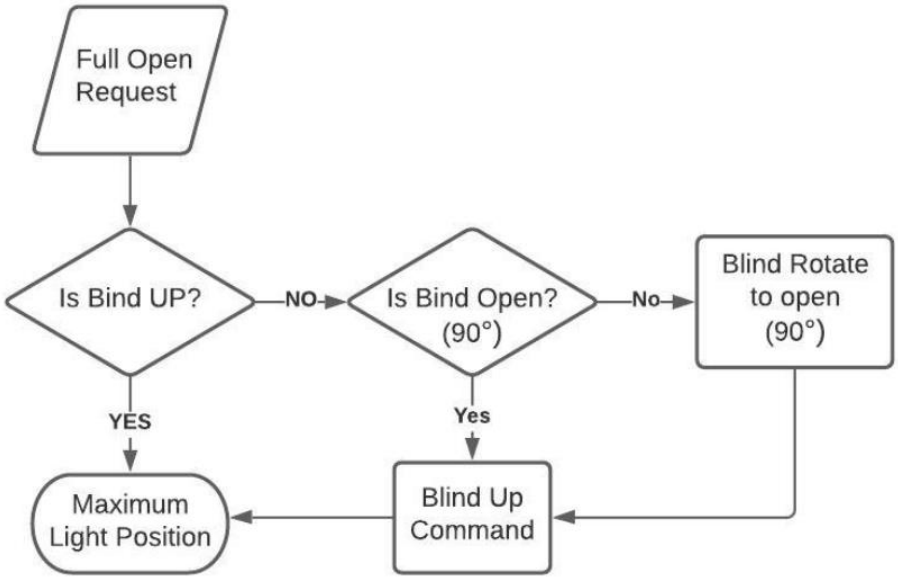


Figure 16: Blind Full Open Flow Diagram

Description: *When ABC receives an open request from webapp*

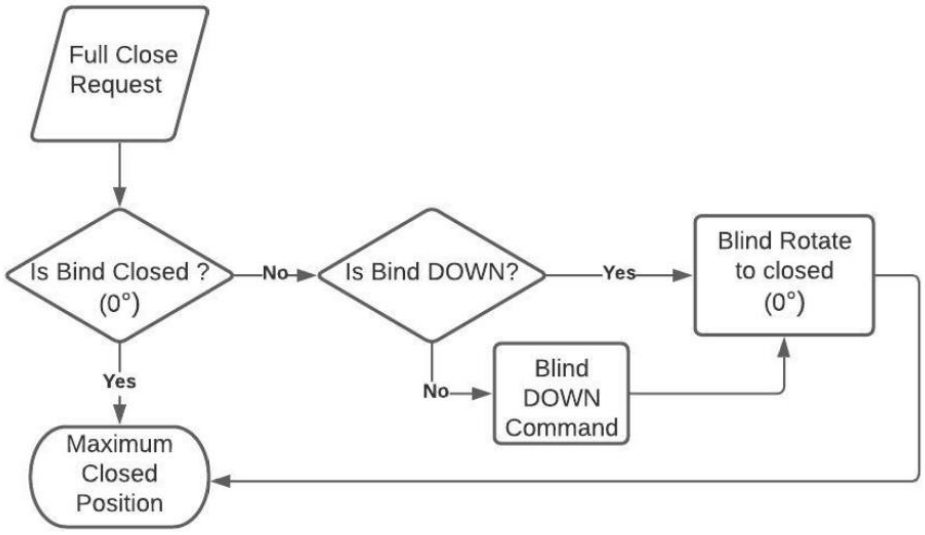


Figure 17: Blind Full Close Flow Diagram

Description: *When ABC receives a close request from webapp*

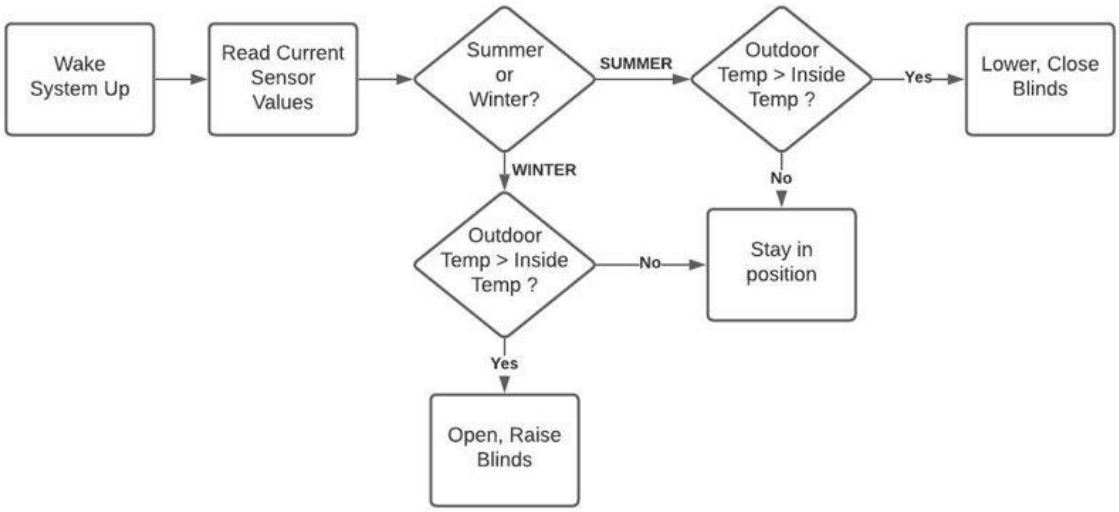


Figure 18: System Operational Diagram

Description: *Main decision-making diagram describing autonomous operation.*

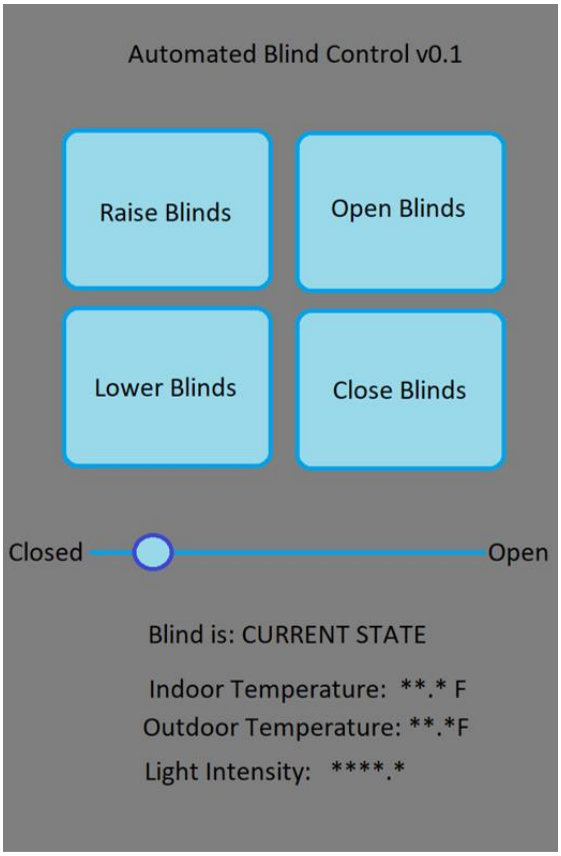


Figure 19: Web App Preliminary Depiction

Description: *A simple web interface allows the user to send commands to the ABC system. The webapp will also display the last known position of the system, as well as the last recorded temperature and light intensity.*

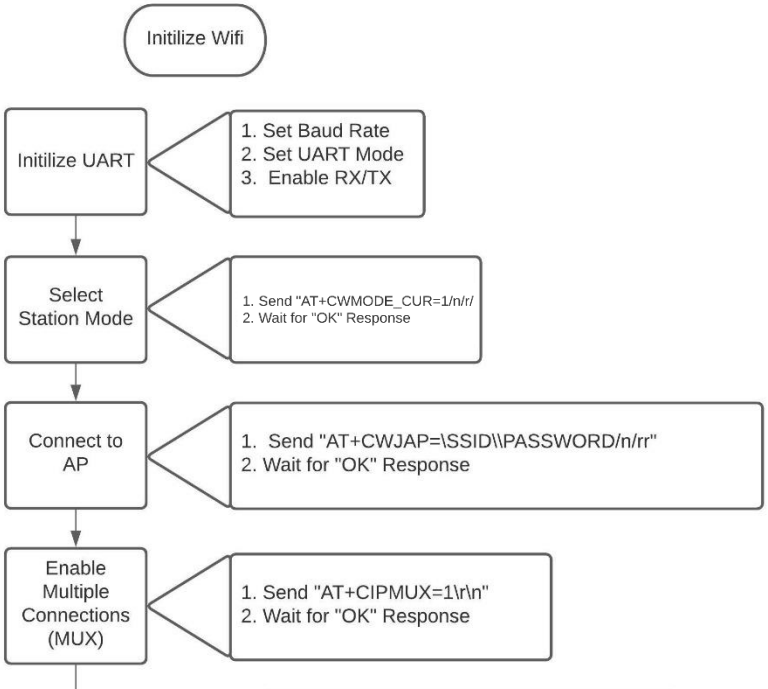


Figure 20: Wi-Fi Initialize Flow Chart

Description: *To communicate with the ESP8266, UART communications must be initialized. Then the ESP needs to connect to an Access Point, enable multiple connections, and then start the webpage.*

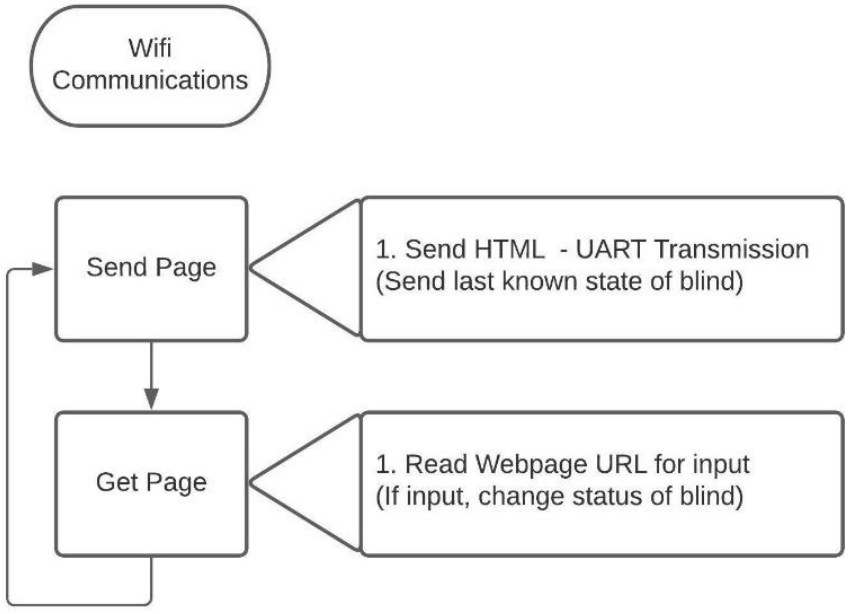


Figure 21: Wi-Fi Communications Flow Chart

Description: *The Wi-Fi module receives a string of characters from the microcontroller, processes it as a HTML page and displays it at the IP address. The Wi-Fi module sends requests back from the webpage to the microcontroller for input.*

```
#include "config.h"
#include "ESP.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int i;

int main(void)
{
    // initialize the device
    TRISA = 0;         // all PORTA pins as outputs
    TRISD = 0xFFFF; //Sets Port B to input

    InitPMP();
    ms_delay(400);
    InitLCD();
    ms_delay(400);
     SetCursorAtLine(1);
    putsLCD("PIC24 ON");
     ms_delay(400);
    ClrLCD();
    InitU2();
    SetCursorAtLine(1);
    putsLCD("UART ON");
    ms_delay(400);
    ClrLCD();
    Start_ESP();
    ms_delay(100);

    while (1)
    {
        //flash LEDS to confirm web connection, and loop is running
        for (i=0;i<=10;i++) {
        PORTAbits.RA0=1;
        ms_delay(100);
        PORTAbits.RA0=0;
        ms_delay(100);
```

Figure 22: Wi-Fi Main.C Code

Description: *The microcontroller is started, UART enabled, and calls to start the ESP8266. LEDs flash to show signal, and the main loop sends and receives page.*

```
/*
 ********************************************************************************
 *ORIGINALLY CREATED AS
 * File:    esp8266.h
 * Author:  Camil Staps <info@camilstaps.nl>
 * Website: http://github.com/camilstaps/ESP8266_PIC
 * Version: 0.1
 *
 * MODFIED AND IMPROVED BY
 * File:    esp8266_functions.h
 * Author:  Aswinth Raj B  <mailtoaswinth@gmail.com>
 * Website: circuitdigest.com
 * Version: 0.1
 *
 * See:   circuitdigest.com for more explanation
 *
 * This is the header file for the ESP8266 PIC16F877A library where ESP runs on 115200 baudrate.
 *
 *  *
 * MODFIED AND IMPROVED BY
 * File:    esp8266_functions.h
 * Author:  Phillipe Sedycias de Queiroz  <phillipesedycias@hotmail.com>
 * Website: www.philseque.com
 * Version: 0
 *
 * See:   circuitdigest.com for more explanation
 *
 * This is the header file for the ESP8266 PIC16F877A library where ESP runs on 115200 baudrate.
 *
 *  * MODFIED AND IMPROVED BY
 * File:    ESP.h
 * Author: Tim Kurczewski  <tjk99@zips.uakron.edu>
 * Version: 0.2
 *
 *  -Updated to work with XC16 compiler
 *
 * This is the header file for the ESP8266 PIC16F877A library where ESP runs on 115200 baudrate.
 *
 */



#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <xc.h>
#include "LCD.H"

#define ESP8266_STATION 0x01
#define ESP8266_SOFTAP 0x02

#define ESP8266_TCP 1
#define ESP8266_UDP 0

#define ESP8266_OK 1
#define ESP8266_READY 2
#define ESP8266_FAIL 3
#define ESP8266_NOCHANGE 4
#define ESP8266_LINKED 5
#define ESP8266_UNLINK 6
#define LED1ON 1
#define LED1OFF 2
#define LED2ON 3
#define LED2OFF 4
int len;
```

Figure 23: ESP.H Part 01

Description: *Main ESP file to define all functions. Reference comments in code*

```
68  ///***_____________________ESP module Function Declarations_____________________**///
69  void put_char(unsigned char);
70  unsigned char get_char(void);
71  int esp8266_isStarted(void);        // Check if the module is started (AT)
72  bool esp8266_restart(void);          // Restart module (AT+RST)
73  void esp8266_echoCmds(bool);        // Enabled/disable command echoing (ATE)
74  // WIFI Mode (station/softAP/station+softAP) (AT+CWMODE)
75  void esp8266_mode(unsigned char);
76  // Connect to AP (AT+CWJAP)
77  void esp8266_connect(char*,char*);
78  // Disconnect from AP (AT+CWQAP)
79  void esp8266_disconnect(void);
80  // Local IP (AT+CIFSR)
81  void esp8266_ip(unsigned char*);
82  // Create connection (AT+CIPSTART)
83  bool esp8266_start(char protocol, char* ip,char port);
84  // Send data (AT+CIPSEND)
85  bool esp8266_send(const char*);
86  // Receive data (+IPD)
87  void esp8266_receive(unsigned char*, unsigned int, bool);
88  /** Functions for internal use only **/
89  // Print a string to the output
90  void _esp8266_print(const char *);
91  // Wait for a certain string on the input
92  inline unsigned int _esp8266_waitFor(const char *);
93  // Wait for any response on the input
94  int _esp8266_waitResponse(void);
95  int _esp8266_waitResponse2(void);
96  //Manin webpage functions
97  void SEND_PAGE();
98  void GET_PAGE();
99  //Initilize ESP
100 void START_ESP();
101 //********_____________________End of Function Declaration_____________________********///
102
103
104 //***Initialize UART for ESP8266**//
105 void InitU2(void)
106 {
107     U2BRG =8;
108     U2MODE = 0x8000;
109     U2STA = 0x0400;
110 }
111 //________UART module Initialized__________//
112
113
114 //**Function to send one byte of date to UART **//
115 void put_char(unsigned char bt){
116     while(U2STAbits.UTXBF ); // Wait if transmit buffer full.
117
118    U2TXREG = bt; //Load the transmitter buffer with the received value
119 }
120
```

Figure 24: ESP.H Part 02

Description: *Starts UART2 for communication with ESP8266*

```
121
122  //**Function to get one byte of date from UART**//
123  unsigned char get_char(void)
124  {
125     // while(!U2STAbits.URXDA); // hold the program till RX buffer is free
126      while(!(U2STAbits.URXDA  == 1))
127      {
128      }
129
130      if(U2STAbits.OERR == 1) //if error, clear the error
131      {
132         U2STAbits.OERR = 0;
133      }
134      return U2RXREG; //receive the value and send it to main function
135  }
136
137  //**Function to send strings to ESP**//
138  void ESP8266_send_string(char* st_pt)
139  {
140      while(*st_pt) //if there is a char
141          put_char(*st_pt++); //process it as a byte data
142  }
143
144  //**Function to stations IP**//
145   void esp8266_get_stationIP()
146      {
147      char rex;
148         ESP8266_send_string("AT+CWLIF\r\n");
149         SetCursorAtLine(2);
150       putsLCD("IP:");
151      do
152         {
153           rex = get_char() ;
154           putsLCD(&rex);
155          }while(rex!=',');
156      }
157
158   //**Function to enable multiple connections**//
159   void enale_MUX()
160   {
161   _esp8266_print("AT+CIPMUX=1\r\n"); //Enable Multiple Connections
162   _esp8266_waitResponse();
163   }
164
165    //**Function to create server on Port 80**//
166   void _esp8266_create_server()
167   {
168      _esp8266_print("AT+CIPSERVER=1,80\r\n"); //Enable Server on port 80
169      _esp8266_waitResponse();
170   }
171
172  /**
173   * Check if the module is started
174   *
175   * This sends the `AT` command to the ESP and waits until it gets a response.
176   *
177   * @return true if the module is started, false if something went wrong
178   */
179  int esp8266_isStarted(void) {
180      _esp8266_print("AT\r\n");
181      return (_esp8266_waitResponse());
182  }
```

Figure 25: ESP.H Part 03

Description: *More functions defined*

```
/**
 * Restart the module
 *
 * This sends the `AT+RST` command to the ESP and waits until there is a
 * response.
 *
 * @return true iff the module restarted properly
 */
bool esp8266_restart(void) {
    _esp8266_print("AT+RST\r\n");
    if (_esp8266_waitResponse() != ESP8266_OK) {
        return false;
    }
    return (_esp8266_waitResponse() == ESP8266_READY);
}

/**
 * Enable / disable command echoing.
 *
 * Enabling this is useful for debugging: one could sniff the TX line from the
 * ESP8266 with his computer and thus receive both commands and responses.
 *
 * This sends the ATE command to the ESP module.
 *
 * @param echo whether to enable command echoing or not
 */
void esp8266_echoCmds(bool echo) {
    _esp8266_print("ATE");
    if (echo) {
        put_char('1');
    } else {
        put_char('0');
    }
    _esp8266_print("\r\n");
    _esp8266_waitFor("OK");
}

/**
 * Set the WiFi mode.
 *
 * ESP8266_STATION : Station mode
 * ESP8266_SOFTAP : Access point mode
 *
 * This sends the AT+CWMODE command to the ESP module.
 *
 * @param mode an ORed bitmask of ESP8266_STATION and ESP8266_SOFTAP
 */
void esp8266_mode(unsigned char mode) {
    _esp8266_print("AT+CWMODE=");
    put_char(mode + '0');
    _esp8266_print("\r\n");
    //_esp8266_waitResponse();
     if (_esp8266_waitResponse() != ESP8266_OK) {
        SetCursorAtLine(1);
         putsLCD("OK");
    }
}
```

Figure 26: ESP.H Part 04

Description: *More Functions defined*

```
/**
 * Connect to an access point.
 *
 * This sends the AT+CWJAP command to the ESP module.
 *
 * @param ssid The SSID to connect to
 * @param pass The password of the network
 * @return an ESP status code, normally either ESP8266_OK or ESP8266_FAIL
 */
void esp8266_connect(char* ssid,char* pass) {
    _esp8266_print("AT+CWJAP=\"");
    _esp8266_print(ssid);
    _esp8266_print("\",\"");
    _esp8266_print(pass);
    _esp8266_print("\"\r\n");
    _esp8266_waitResponse();
}

/**
 * Disconnect from the access point.
 *
 * This sends the AT+CWQAP command to the ESP module.
 */
void esp8266_disconnect(void) {
    _esp8266_print("AT+CWQAP\r\n");
    _esp8266_waitFor("OK");
}

/**
 * Store the current local IPv4 address.
 *
 * This sends the AT+CIFSR command to the ESP module.
 *
 * The result will not be stored as a string but byte by byte. For example, for
 * the IP 192.168.0.1, the value of store_in will be: {0xc0, 0xa8, 0x00, 0x01}.
 *
 * @param store_in a pointer to an array of the type unsigned char[4]; this
 * array will be filled with the local IP.
 */
void esp8266_ip(unsigned char* store_in) {
    unsigned char i;
    _esp8266_print("AT+CIFSR\r\n");
    unsigned char received;
    do {
        received = get_char();
    } while (received < '0' || received > '9');
    for (i = 0; i < 4; i++) {
        store_in[i] = 0;
        do {
            store_in[i] = 10 * store_in[i] + received - '0';
            received = get_char();
        } while (received >= '0' && received <= '9');
        received = get_char();
    }
    _esp8266_waitFor("OK");
}
```

Figure 27: ESP.H Part 05

Description: *More Functions defined*

```
/**
 * Open a TCP or UDP connection.
 *
 * This sends the AT+CIPSTART command to the ESP module.
 *
 * @param protocol Either ESP8266_TCP or ESP8266_UDP
 * @param ip The IP or hostname to connect to; as a string
 * @param port The port to connect to
 *
 * @return true iff the connection is opened after this.
 */
bool esp8266_start(char protocol, char* ip, char port) {
    _esp8266_print("AT+CIPSTART=\"");
    if (protocol == ESP8266_TCP) {
        _esp8266_print("TCP");
    } else {
        _esp8266_print("UDP");
    }
    _esp8266_print("\",\"");
    _esp8266_print(ip);
    _esp8266_print("\",");
    char port_str[5] = "\0\0\0\0";
    sprintf(port_str, "%u", port);
    _esp8266_print(port_str);
    _esp8266_print("\r\n");
    if (_esp8266_waitResponse() != ESP8266_OK) {
        return 0;
    }
    if (_esp8266_waitResponse() != ESP8266_LINKED) {
        return 0;
    }
    return 1;
}

// Send data (AT+CIPSEND)
/**
 * Send data over a connection.
 *
 * This sends the AT+CIPSEND command to the ESP module.
 *
 * @param data The data to send
 *
 * @return true iff the data was sent correctly.
 */
bool esp8266_send(const char* data) {
    char length_str[6] = "\0\0\0\0\0";
    sprintf(length_str, "%u", strlen(data));
    _esp8266_print("AT+CIPSEND=");
    _esp8266_print(length_str);
    _esp8266_print("\r\n");
    while (get_char() != '>');
    _esp8266_print(data);
    if (_esp8266_waitResponse() == ESP8266_OK) {
        return 1;
    }
    return 0;
}
```

Figure 28: ESP.H Part 06

Description: *More Functions defined*

```
357  /**
358   * Read a string of data that is sent to the ESP8266.
359   *
360   * This waits for a +IPD line from the module. If more bytes than the maximum
361   * are received, the remaining bytes will be discarded.
362   *
363   * @param store_in a pointer to a character array to store the data in
364   * @param max_length maximum amount of bytes to read in
365   * @param discard_headers if set to true, we will skip until the first \r\n\r\n,
366   * for HTTP this means skipping the headers.
367   */
368  void esp8266_receive(unsigned char* store_in, unsigned int max_length, bool discard_headers) {
369      _esp8266_waitFor("+IPD,");
370      unsigned int length = 0;
371      unsigned char received = get_char();
372      do {
373          length = length * 10 + received - '0';
374          received = get_char();
375      } while (received >= '0' && received <= '9');
376
377      if (discard_headers) {
378          length -= _esp8266_waitFor("\r\n\r\n");
379      }
380
381      if (length < max_length) {
382          max_length = length;
383      }
384
385      /*sprintf(store_in, "%u,%u:%c%c", length, max_length, get_char(), get_char());
386      return;*/
387
388      uint16_t i;
389      for (i = 0; i < max_length; i++) {
390          store_in[i] = get_char();
391      }
392      store_in[i] = 0;
393      for (; i < length; i++) {
394          get_char();
395      }
396      _esp8266_waitFor("OK");
397  }
398
399  /**
400   * Output a string to the ESP module.
401   *
402   * This is a function for internal use only.
403   *
404   * @param ptr A pointer to the string to send.
405   */
406  void _esp8266_print(const char *ptr) {
407      while (*ptr != 0) {
408          put_char(*ptr++);
409      }
410  }
```

Figure 29: ESP.H Part 07

Description: *More Functions defined*

```
411
412  /**
413   * Wait until we found a string on the input.
414   *
415   * Careful: this will read everything until that string (even if it's never
416   * found). You may lose important data.
417   *
418   * @param string
419   *
420   * @return the number of characters read
421   */
422  inline unsigned int _esp8266_waitFor(const char *string) {
423      unsigned char so_far = 0;
424      unsigned char received;
425      unsigned int counter = 0;
426      do {
427          received = get_char();
428          counter++;
429          if (received == string[so_far]) {
430              so_far++;
431          } else {
432              so_far = 0;
433          }
434      } while (string[so_far] != 0);
435      return counter;
436  }
437
438  /**
439   * Wait until we received the ESP is done and sends its response.
440   * @return a constant from esp8266.h describing the status response.
441   */
442  int _esp8266_waitResponse(void) {
443
444      unsigned char i;
445      unsigned char so_far[6] = {0,0,0,0,0,0};        //quantidade de respostas
446      unsigned const char lengths[6] = {2,5,4,9,6,6}; //tamanho das respostas
447      const char* strings[6] = {"OK", "ready", "FAIL", "no change", "Linked", "Unlink"}; //respostas
448      int responses[6] = {ESP8266_OK, ESP8266_READY, ESP8266_FAIL, ESP8266_NOCHANGE, ESP8266_LINKED, ESP8266_UNLINK};
449      unsigned char received;
450      int response;
451      bool continue_loop = true;
452
453      while (continue_loop) {
454          received = get_char();
455
456          for (i = 0; i < 6; i++) {
457              if (strings[i][so_far[i]] == received) {
458                  so_far[i]++;
459                  if (so_far[i] == lengths[i]) {
460                      response = responses[i];
461                      continue_loop = false;
462                  }
463              } else {
464                  so_far[i] = 0;
465              }
466          }
467      }
468      return response;
469  }
```

Figure 30: ESP.H Part 08

Description: *More Functions defined*

```
// sends the webpage in HTML format
void SEND_PAGE()
{
    _esp8266_print("AT+CIPSEND=0,382\r\n");  //Use character counter for data size --> https://www.invertexto.com/contador-caracteres
    ms_delay(300);
    _esp8266_print("<html><head><title>DT6</title><meta http-equiv=\"refresh\" content=\"4\" ></head><h1>DT6 Demo</h1>"); //98
    _esp8266_print("<body><p><form method=\"GET\"></p>"); //34
    _esp8266_print("<p><input type=\"submit\" name=\"B\" value=\"1\" > LED1 ON</p>");       //62
    _esp8266_print("<p><input type=\"submit\" name=\"B\" value=\"2\" > LED1 OFF</p>");    //63
    _esp8266_print("<p><input type=\"submit\" name=\"B\" value=\"3\" > LED2 ON</p>");       //62
    _esp8266_print("<p><input type=\"submit\" name=\"B\" value=\"4\" > LED2 OFF</p>");   //63


    if (PORTAbits.RA7){
        _esp8266_print( "<p>RA7 ON! </p></body></html>\r\n");  //33
    }
    else _esp8266_print("<p>RA7 OFF!</p></body></html>\r\n");  //33

    //close connection
   _esp8266_print("AT+CIPCLOSE=0\r\n");
   ms_delay(250);
   _esp8266_print("AT+CIPCLOSE=0\r\n");
   ms_delay(250);
}
// processes the commands from web URL. Swtich to determine how to change outputs based on web input.
void GET_PAGE(){
switch (_esp8266_waitResponse2())
        {
            case (LED1ON):
            {
            PORTAbits.RA1=1;
            }
            break;

           case (LED1OFF):
            {
            PORTAbits.RA1=0;
            }
            break;

            case (LED2ON):
            {
            PORTAbits.RA2=1;
            }
            break;

           case (LED2OFF):
            {
            PORTAbits.RA2=0;
            }
            break;
    }
}
```

Figure 31: ESP.H Part 09

Description: *Main send and receive page functions*

```
/*
Itterates through the response from ESP
If responce matches with any of the expected cases, return the const
for matched case
ex - if response == "B=4" , return LED2OFF ~ 4

*/
int _esp8266_waitResponse2(void) {
    char str[10];
    unsigned char i;
    unsigned char so_far[4] = {0,0,0,0};
    unsigned char lengths[4] = {3,3,3,3};
    const char* strings[4] = {"B=4", "B=3", "B=2", "B=1"}; //responses
    unsigned const char  responses[4] = {LED2OFF, LED2ON, LED1OFF, LED1ON};
    unsigned char received;
    char response;
    bool continue_loop = true;
    while (continue_loop) {
        received = get_char();
        for (i = 0; i < 4; i++) {
            if (strings[i][so_far[i]] == received) {        //match the responce with strings[]
                so_far[i]++;
                if (so_far[i] == lengths[i]) {
                    response = responses[i];   // if matched, responce is responce[]
                    continue_loop = false;  // leave the loop
                }
            } else {
                so_far[i] = 0;
            }
        }
    }

    SetCursorAtLine(1);
    sprintf(str, "%d", response);
    putsLCD(str);
    ms_delay(500);
    ClrLCD();
    return response;
}

// Starts the ESP module up for a web-application. Connects to AP, enables MUX, and starts the server on port 80.
void Start_ESP() {

    /*Put the module in AP+STA*/
    // Station Mode
    esp8266_mode(1);
    SetCursorAtLine(1);
    putsLCD("MODE 1");
    //Connect to AP Wifi
    esp8266_connect("TKWIFI","Letme1n!");
    SetCursorAtLine(1);
    putsLCD("CONNECT TKWIFI");
    //Enable Multiple Connections to Server
    enale_MUX();
    SetCursorAtLine(1);
    putsLCD("MUX ON");
    //Start Server on port 80
    _esp8266_create_server();
    SetCursorAtLine(1);
    putsLCD("SERVER ON");
    ms_delay(300);
    ClrLCD();
}
```

Figure 32: ESP.H Part 10

Description: *StartESP() is called in Main to start ESP8266*

```c
#include "mcc_generated_files/system.h"

void ms_delay(int n)
{
    T1CON = 0x8030; // Timer 1 on, Prescale 256:1
    TMR1 = 0;
    while(TMR1 < n*63);  // (1/(16MHz/256))*62.5 = 1 ms
}

void us_delay(int n)
{
    T1CON = 0x8010; // Timer 1 on, Prescale 8:1
    TMR1 = 0;
    while(TMR1 < n*2);  // (1/(16MHz/8))*2 = 1 us
}

int main(void)
{
    SYSTEM_Initialize();
    TRISA = 0;  // All PORTA pins as output
    AD1PCFG = 0xffff;   // All PORTA pins as digital
    //pin 60 must go high to leave sleep - RA4
    PORTAbits.RA4 = 1;
    //set enable and reset low to enable RA5 and RA6 pin 61 and 91
    PORTAbits.RA5 = 0;
    PORTAbits.RA6 = 0;
    //set M1 and M0 to low for full step
    PORTAbits.RA2 = 0;
    PORTAbits.RA3 = 0;
    //set the direction Pin 38
    PORTAbits.RA1 = 1;
    //set the step high to turn - RA0 Pin 17
    while (1)
    {
        PORTAbits.RA0 = 1;
        // 5 ms total time for 60 rpm
        int k = 0;
        ms_delay(4);
        PORTAbits.RA0 = 0;
        ms_delay(1);
    }

    return 1;
}
```

Figure 33: Motor_Control_Test.C

Description: *The motor controller demo code sets the clock and declares a function to have a millisecond delay time with an input for how long that should be. Then, the system initializes each of the TRISA bits to output to the motor controller. The sleep bit had to be driven high, and the enable and reset pins driven low to ensure that the motor would run. Then, for full step operation, M1 and M0 are set to low and the direction bit was set high. Finally, to rotate at 60 rpm, the step bit is set high for 4 ms, followed by an off time with the step bit set low for 1 ms. This results in the stepper motor running at 60 rpm continuously.*

```
46      Section: Included Files
47  */
48  #include <xc.h>
49  #include "mcc_generated_files/system.h"
50  #include <stdio.h>
51  #include <stdlib.h>
52  #include <string.h>
53
54  char ReadLCD(int addr)
55  {
56      int dummy;
57      while(PMMODEbits.BUSY); // wait for PMP to be available
58      PMADDR=addr;            // select the command address
59      dummy=PMDIN1;           // initiate a read cycle, dummy
60      while(PMMODEbits.BUSY); // wait for PMP to be available
61      return(PMDIN1);         // read the status register
62  } // ReadLCD
63
64  // In the following, addr = 0 -> access Control, addr = 1 -> access Data
65  #define BusyLCD() ReadLCD(0)&0x80    // D<7> = Busy Flag
66  #define AddrLCD() ReadLCD(0)&0x7F    // Not actually used here
67  #define getLCD() ReadLCD(1)          // Not actually used here
68
69  void WriteLCD(int addr, char c)
70  {
71      while (BusyLCD());
72      while (PMMODEbits.BUSY); // wait for PMP to be available
73      PMADDR = addr;
74      PMDIN1 = c;
75  } // WriteLCD
```

Figure 34: Temp Sensor Demo.c Part 01

Description: *Includes and Functions for LCD*

```
103 }
104
105 void InitPMP(void)
106 {
107     //PMP initialization
108     PMCON = 0x8303; // Following Fig. 13-34. Text says 0x83BF (also works)
109     PMMODE = 0x03FF;     // Master Mode 1. 8 bit data, long waits
110     PMAEN = 0x0001;      // PMA0 enabled
111 }
112
113 void SetCursorAtLine(int i)
114 {
115     int k;
116     if(i == 1)
117     {
118         CmdLCD(0x80);    // this sets cursor of LCD to upper left
119     }
120     else if (i == 2)
121     {
122         CmdLCD(0xC0);    // sets cursor to bottom left
123     }
124     else
125     {    // flash LEDs if neither line is set
126         for(k =1; k < 20; k++)
127         {
128             // flashes on and off at 5 Hz
129             PORTA = 0xff;
130             ms_delay(100);
131             PORTA = 0x00;
132             ms_delay(100);
```

main > while (1) >

Figure 35: Temp Sensor Demo.c Part 02

Description: *Functions for LCD*

```
136  void InitADC(int amask)
137  {
138      AD1PCFG = amask;     // select analog input pins
139      AD1CON1 = 0x00E0;    // auto convert @ end of sampling, Integer Data out
140      AD1CON2 = 0;         // use MUXA, AVss and AVdd used as Vref
141      AD1CON3 = 0x1F01;    // Tad = 2xTcy = 125ns. 31*Tad for conversion time
142      AD1CSSL = 0;         // no scanning required
143      AD1CON1bits.ADON = 1;   // turn on the ADC
144  }// InitADC
145
146  int ReadADC(int ch)
147  {
148      AD1CHS = ch;         // 1. select analog input channel
149                           // start sampling, automatic conversion will follow
150      AD1CON1bits.SAMP = 1;   // 2. Start sampling
151      while(!AD1CON1bits.DONE);   // 5. wait for conversion to complete
152      AD1CON1bits.DONE = 0;   // 6. clear flag. We are responsible
153      return ADC1BUF0;
154  } // Read ADC
155
156  void ms_delay(int ms)
157  {
158      T2CON = 0x8030;      // Timer 2 on, TCKPS<1,0> = 11 this 1:256
159      TMR2 = 0;
160      while(TMR2<ms*62.5);
161  } //ms_delay
162
163  int power(int val, int exponent)     // calculates a value raised to a power
164  {
165      int power = 1;
```

Figure 36:Temp Sensor Demo.c Part 03

Description: *Functions for ADC*

```
163  int power(int val, int exponent)     // calculates a value raised to a power
164  {
165      int power = 1;
166      int k = 0;
167      for (k = 0; k < exponent; k=k+1)
168      {
169          power = power * val;
170      }
171      return power;
172  }
173
174  int temp;
175  int temp2;
176
177  int main(void)
178  {
179      // initialize the device
180      SYSTEM_Initialize();
181      InitADC(0xFFF7);        // initialize the ADC and analog inputs
182      TRISA = 0xff00;         // all PORTA pins as outputs
183      int n = 4;                  // change for number of iterations to average
184      int sample = power(2,n);        // for the loop condition
185      float quant_res = 0.00322; // conversion ratio 3.3/1023
186      float Vout = 0;
187      float T_degC = 0;
188      float T_degF = 0;
189      char LCDchar[10];     // Create an array for 8 characters
190      InitPMP();
191      InitLCD();
192
```

Figure 37: Temp Sensor Demo.c Part 04

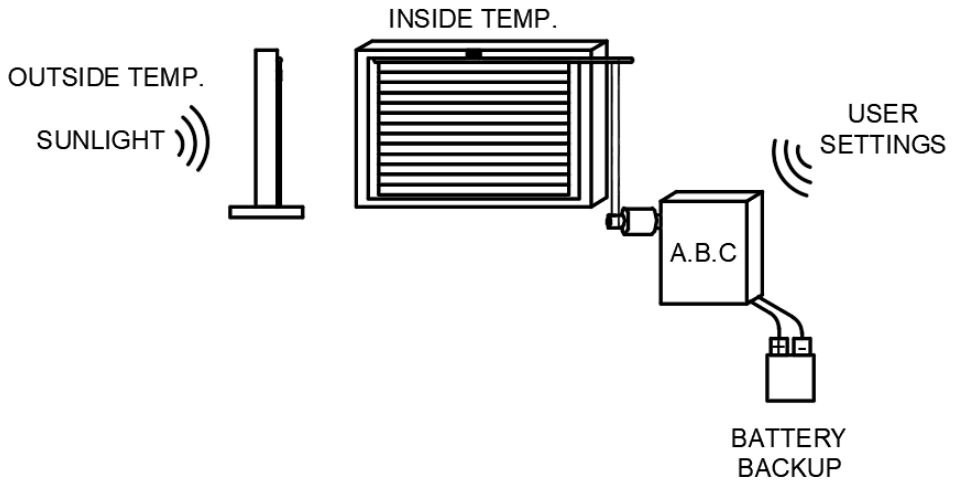
Description: *Initialize ADC converter*

```
190     InitPMP();
191     InitLCD();
192
193     while (1)
194     {
195         int i = 0;  // reset for loop variable
196         temp = 0;                    // reset temp value
197         for (i=0;i<=sample; i=i+1)
198         {
199             temp = temp + ReadADC(3);  // Read Temperature sensor on CH4
200             ms_delay(5);
201         }
202         temp = temp >> n;
203         temp2 = temp;
204         Vout = temp * quant_res;
205         T_degC = 100*Vout - 50;
206         T_degF = T_degC*1.8 + 32;// temp sensor conversion formula
207         PORTA = T_degF;
208
209         sprintf(LCDchar, "%0.4f", T_degF); // Convert T-degC to FP, put in LCDchar
210         ms_delay(32);
211         SetCursorAtLine(1);
212         putsLCD("Temperature is : ");
213         SetCursorAtLine(2);
214         putsLCD(LCDchar);
215     }
216
217     return 1;
```

Figure 38: Temp Sensor Demo.c Part 05

Description: *Main that changes Celsius to Fahrenheit and displays on LCD*

6.0 Mechanical Sketch (DB,TK)



7.0 Team Information

W. Daulton Baksa (DB), Electrical Engineering.

Tim Kurczewski (TK), Computer Engineering.

Matthew Lacek (ML), Electrical Engineering.

Daniel Nahra (DN), Computer Engineering.

8.0 Parts List (DB,TK,ML,DN)

8.1 Part Specifications

Qt	Reference Designators	Part Number	Description
1	WIFIMODULE	ESP8266 ESP-01	Wifi Serial Transciever Module
1	MCNTRLR	PIC24FJ128GA010-I/PF	Microchip 16 bit Microcontroller
1	D3	SFH482	Green LED
2	VR1,VR3,VR4	COM-09288	Potentiometer Resistor

2	DC1	VA-1205S1.5	1.5W, 12V-5V, single output DC-DC converter
2	DC2	VA-123R3S1.5	1.5W, 12V-3.3V, single output DC-DC converter
8	B1,B2,B3,B4	LG MJ1 INR 18650	3500mAh 3.7V Li-ion Battery
1	BP1	S-8205AAB-TCT1U	Battery Protection IC
2	MC1,MC2	A4982SETTR-T	Bipolar Motor Driver
1	POWER JACK	PRT-10288	Terminal Adapter, 2.1mm Jack to 2 Pos Terminal
1	D1	1N4007	Diode
1	12V-REG	L7812C	Positive Voltage Regulator
1	LS	OPT101PG4	Monolithic Photodiode Transimpedance Amplifier
4	R2,R3	General	2.4 Resisitor
2	R1	General	5 k Resistor
2	R4	General	825 k Resistor
8	R6,R7,R8,R9,R10,R12,R13,R14	General	1 k Resistor
2	R16,R17	General	5.1 k Resistor
1	R18	General	1 M Resistor
1	R15	General	100 Resistor
12	C2,C4,C6,C7C8,C10,C11,C12,C13	General	0.1uF Ceramic Capacitor
1	C9	General	1 uF Ceramic Capacitor
4	C1,C5	General	0.22 uF Ceramic Capacitor
2	C3	General	100 uF Bulk Capacitor
4	CHG,DSG	2N7002ET1G	N-Channel 60V 260mA
1	MPB	TTONC&K_KSS	Manual Pushbutton
1	TS	TMP36GT9Z	Temperature Sensor

8.2 Bill of Materials

Qt	Part Number	Description	Unit Cost	Total Cost
1	ESP8266 ESP-01	Wifi Serial Transciever Module	$5.99	$5.99
1	PIC24FJ128GA010-I/PF	Microchip 16 bit Microcontroller	$4.75	$4.75

8	LG MJ1 INR 18650	3500mAh 3.7V Li-ion Battery	$5.35	$42.80
1	S-8205AAB-TCT1U	Battery Protection IC	$1.47	$1.47
1	SB017	House Blinds (Alabaster Color)	$14.43	$14.43
2	1207	Pololu Bipolar Stepper Motor	$13.81	$27.62
1	L6R12-120	AC/DC Wall Mount Adapter 12V, 12W	$7.58	$7.58
1	PA0067-KIT	QFN-32 (0.5 mm pitch, 5x5 mm body) PCB and Stencil Kit	$26.79	$26.79
1	PA0067	QFN-32 TO DIP-32 SMT ADAPTER	$8.39	$8.39
2	A4982SETTR-T	Bipolar Motor Driver	$3.18	$6.36
1	PRT-10288	Terminal Adapter, 2.1mm Jack to 2 Pos Terminal	$2.95	$2.95
2	1N4007	Diode	EE Stock	$-
1	L7812C	Positive Voltage Regulator	EE Stock	$-
1	OPT101PG4	Monolithic Photodiode Transimpedance Amplifier	$7.35	$7.35
4	General	2.2 ohm Resistors	EE Stock	$-
2	General	5 kohm Resistor	EE Stock	$-
2	General	825 kohm resistor	EE Stock	$-
8	General	1 k Resistors	EE Stock	$-
2	General	5.1 k Resistors	EE Stock	$-
1	General	1 M Resistor	EE Stock	$-
1	General	100 Resistors	EE Stock	$-
12	General	0.1 u Ceramic Capacitors	EE Stock	$-
1	General	1 u Ceramic Capacitor	EE Stock	$-
4	General	0.22 u Ceramic Capacitors	EE Stock	$-
2	General	100 u bulk capacitor	EE Stock	$-
4	2N7002ET1G	N-Channel 60V 260mA	$0.18	$0.72

1	TMP36GT9Z	Temperature Sensor	$1.48	$1.48

9.0 Project Schedules (DN)

Midterm Design Gantt:

Task Name	Duration	Start	Finish	Predecessors	Resource Names
SDP I 2020					
Project Design	**98 days?**	**Wed 8/26/20**	**Wed 12/2/20**		
Midterm Report	**40.38 days**	**Wed 8/26/20**	**Mon 10/5/20**		
Cover page	40 days	Wed 8/26/20	Mon 10/5/20		
T of C, L of T, L of F	40 days	Wed 8/26/20	Mon 10/5/20		
Problem Statement	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**		
Need	40 days	Wed 8/26/20	Mon 10/5/20		All
Objective	40 days	Wed 8/26/20	Mon 10/5/20		All
Background	40 days	Wed 8/26/20	Mon 10/5/20		All
Marketing Requirements	40 days	Wed 8/26/20	Mon 10/5/20		All
Engineering Requirements Specification	40 days	Wed 8/26/20	Mon 10/5/20		All
Engineering Analysis	**34 days**	**Wed 8/26/20**	**Tue 9/29/20**		
Circuits (DC, AC, Power, …)	**34 days**	**Wed 8/26/20**	**Tue 9/29/20**		
Inside Temp Sensor	33.38 days	Wed 8/26/20	Mon 9/28/20		Daulton Baksa
Light Sensor	33.38 days	Wed 8/26/20	Mon 9/28/20		Daulton Baksa
Infared Light Sensor	33.38 days	Wed 8/26/20	Mon 9/28/20		Daulton Baksa
Batterys	33.38 days	Wed 8/26/20	Mon 9/28/20		Daulton Baksa,Matt Lacek
Electronics (analog and digital)	**34 days**	**Wed 8/26/20**	**Tue 9/29/20**		

Battery Charger(BMS)	33.38 days	Wed 8/26/20	Mon 9/28/20		Daulton Baksa,Matt Lacek
Solor Panel	33.38 days	Wed 8/26/20	Mon 9/28/20		Daniel Nahra,Daulton Baksa
Signal Processing	**34 days**	**Wed 8/26/20**	**Tue 9/29/20**		
Sensor communcation to MicroController	33.38 days	Wed 8/26/20	Mon 9/28/20		Tim Kurczewski,Daniel Nahra
Communications (analog and digital)	**34 days**	**Wed 8/26/20**	**Tue 9/29/20**		
Wifi Communcation	33.38 days	Wed 8/26/20	Mon 9/28/20		Daniel Nahra
Sensor Data to database	32.38 days	Wed 8/26/20	Mon 9/28/20		Daniel Nahra
Electromechanics	**34 days**	**Wed 8/26/20**	**Tue 9/29/20**		
Motor (Lateral and Horizontal Movement)	33.38 days	Wed 8/26/20	Mon 9/28/20		Matt Lacek
Computer Networks	**34 days**	**Wed 8/26/20**	**Tue 9/29/20**		
Web Application	33.38 days	Wed 8/26/20	Mon 9/28/20		Daniel Nahra,Tim Kurczewski
Embedded Systems	**33.38 days**	**Wed 8/26/20**	**Mon 9/28/20**		
Microcontroller	33.38 days	Wed 8/26/20	Mon 9/28/20		Tim Kurczewski
Motor Controller	33.38 days	Wed 8/26/20	Tue 9/29/20		Matt Lacek
Wifi Module	33.38 days	Wed 8/26/20	Tue 9/29/20		Tim Kurczewski
Accepted Technical Design	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**		
Mechanical Sketch	40 days	Wed 8/26/20	Mon 10/5/20		
Team information	40 days	Wed 8/26/20	Mon 10/5/20		All
Project Schedules	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**		
Midterm Design Gantt Chart	40 days	Wed 8/26/20	Mon 10/5/20		Daniel Nahra
References	40 days	Wed 8/26/20	Mon 10/5/20		All

Midterm Parts Request Form	47 days	Wed 8/26/20	Mon 10/12/20		
Midterm Design Presentations Day 1	0 days	Wed 9/30/20	Wed 9/30/20		
Midterm Design Presentations Day 2	0 days	Wed 10/7/20	Wed 10/7/20		

Final Design Gantt:

Task Name	Duration	Start	Finish	Predecessors	Resource Names
Project Poster	14 days	Wed 11/18/20	Wed 12/2/20		
Final Design Report	**50 days**	**Tue 10/6/20**	**Wed 11/25/20**		
Abstract	48 days	Tue 10/6/20	Mon 11/23/20		Matt Lacek, Daulton Baksa, Tim Kurczewski
Hardware Design: Phase 2	**51.38 days?**	**Mon 10/5/20**	**Wed 11/25/20**		
Modules 1…n	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Motors	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Research/ Testing	51.38 days	Mon 10/5/20	Wed 11/25/20		Matt Lacek
Implementation	51.38 days	Mon 10/5/20	Wed 11/25/20		Matt Lacek
Uninterruptable Power Supply	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Design	51.38 days	Mon 10/5/20	Wed 11/25/20		Daulton Baksa
Sensors	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Temperature Sensor	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Design	51.38 days	Mon 10/5/20	Wed 11/25/20		Daulton Baksa,Daniel Nahra
Implementation	51.38 days	Mon 10/5/20	Wed 11/25/20		Daniel Nahra
Light Sensor	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Design	51.38 days	Mon 10/5/20	Wed 11/25/20		Daulton Baksa

Implementation	51.38 days	Mon 10/5/20	Wed 11/25/20		Daulton Baksa
Schematics	51.38 days	Mon 10/5/20	Wed 11/25/20		Daulton Baksa,Matt Lacek,Tim Kurczewski
Software Design: Phase 2	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Modules 1…n	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Website	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Design	51.38 days	Mon 10/5/20	Wed 11/25/20		Tim Kurczewski,Daniel Nahra
Implementation	51.38 days	Mon 10/5/20	Wed 11/25/20		Tim Kurczewski
Wifi Module	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Design	51.38 days	Mon 10/5/20	Wed 11/25/20		Daniel Nahra,Tim Kurczewski
Implementation	51.38 days	Mon 10/5/20	Wed 11/25/20		Tim Kurczewski
PIC Microcontroller	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Design	51.38 days	Mon 10/5/20	Wed 11/25/20		Daniel Nahra,Tim Kurczewski
System integration Behavior Models	51.38 days	Mon 10/5/20	Wed 11/25/20		Daniel Nahra,Tim Kurczewski
Parts Lists	**51.38 days**	**Mon 10/5/20**	**Wed 11/25/20**		
Parts list(s) for Schematics	51.38 days	Mon 10/5/20	Wed 11/25/20		Tim Kurczewski,Daulton Baksa,Matt Lacek
Materials Budget list	51.38 days	Mon 10/5/20	Wed 11/25/20		Matt Lacek,Tim Kurczewski
Proposed Implementation Gantt Chart	51.38 days	Mon 10/5/20	Wed 11/25/20		
Conclusions and Recommendations	51.38 days	Mon 10/5/20	Wed 11/25/20		Daniel Nahra,Daulton Baksa,Matt

					Lacek,Tim Kurczewski
Final Parts Request Form	13 days	Sun 10/11/20	Sat 10/24/20		
Subsystems Demonstrations Day 1	0 days	Tue 11/10/20	Tue 11/10/20		
Subsystems Demonstrations Day 2	0 days	Tue 11/17/20	Tue 11/17/20		
Parts Request Form for Spring Semester	9 days	Mon 11/23/20	Wed 12/2/20		

10.0 Conclusions and Recommendations (DB, TK, ML, DN)

From our preliminary research and engineering analysis, we propose a viable and realistic solution to fully automating a window blind slat-style covering. The understanding of sunlight data received is still in the development stage as well to help specify what the microcontroller is looking for when analyzing the output of the light sensor. So far, proof of concept has been shown through the ability to turn a motor with the microcontroller, read in temperature and light sensor values, and control an LED from the web application. The next phases of this project will be realizing this design using parts shown earlier and creating the logic so that the A.B.C system will make decisions based on the inputs. A printed circuit board of all hardware will also be created. At the end of next semester, a functioning prototype combining all subsystems will be delivered.

(More conclusion & recommendations will be added once the project has been completed.)

11.0 References (DB, TK, ML, DN)

[1]

https://ieeexplore-ieee-org.ezproxy.uakron.edu:2443/stamp/stamp.jsp?tp=&arnumber=8537381

- Comparative Analysis of different methods of Smart Home Automation

[2]

https://ieeexplore-ieee-org.ezproxy.uakron.edu:2443/document/8752363

- Looking at energy storage due to change in weather conditions (variations in solar irradiance), what is the most efficient/cost-effective power device?

[3]

https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1400&context=eesp

- Previous SDP for automated blinds

[4]

http://web.a.ebscohost.com.ezproxy.uakron.edu:2048/ehost/detail/detail?vid=3&sid=605ef61f-53b2-4093-b74f-374ade02c842%40sessionmgr4007&bdata=JnNpdGU9ZWhvc3QtbGl2ZQ%3d%3d#AN=112801715&db=a9h

- Sunlight beam index, how sunlight affects room temp, etc

[5]

https://ieeexplore-ieee-org.ezproxy.uakron.edu:2443/document/6840292

- why we need to perform energy storage if using solar power: solar power isn't consistent and it's more cost-effective to store (Lyapunov model). Also, cooperation in distributed power generation units (sharing power with deficient areas from excess of others)

[6]

https://ieeexplore-ieee-org.ezproxy.uakron.edu:2443/stamp/stamp.jsp?tp=&arnumber=8358532

- About the use of a mobile application in a smart home system

[7]

https://patents.google.com/patent/US20120193035A1/en?oq=US+2012%2f0193035+A1

- US 2012/0193035 A1

[8]

https://patents.google.com/patent/US7417397?oq=US+7%2c417%2c397+B2

- US 7,417,397 B2

[9]

https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf

- ESP8266 AT Instruction Set

12.0 Appendices (DB,TK,ML,DN)

(More appendices will be added once the project has been completed.)

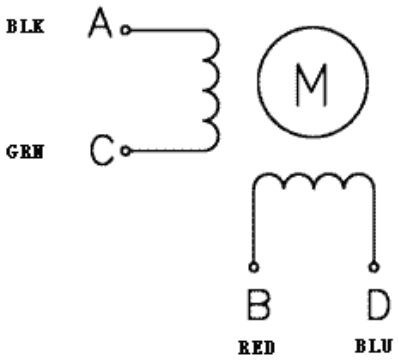


Figure 39: Bipolar Stepper Motor Schematic Wiring Diagram

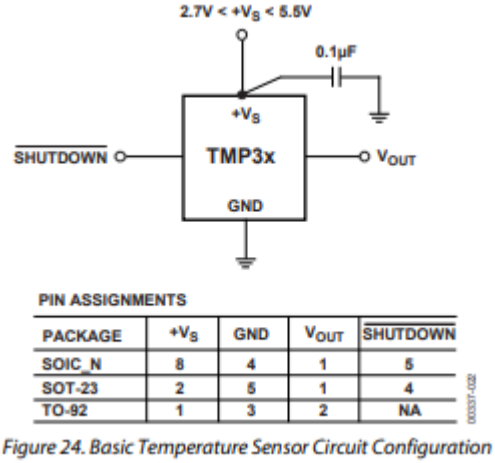


Figure 40: Indoor Temperature Sensor Power/Output

3.2. Commands

3.2.1. AT—Tests AT Startup

Execute Command	AT
Response	OK
Parameters	-

Figure 41: ESP8266 AT Test Command

4.2.1. AT+CWMODE_CUR—Sets the Current Wi-Fi mode; Configuration Not Saved in the Flash

Commands	Test Command: AT+CWMODE_CUR=?	Query Command: AT+CWMODE_CUR? Function: to query the current Wi-Fi mode of ESP8266.	Set Command: AT+CWMODE_CUR=<mode> Function: to set the current Wi-Fi mode of ESP8266.
Response	+CWMODE_CUR:<mode> OK	+CWMODE_CUR:<mode> OK	OK
Parameters	<mode>: • 1: Station mode • 2: SoftAP mode • 3: SoftAP+Station mode		
Note	The configuration changes will NOT be saved in the flash.		
Example	AT+CWMODE_CUR=3		

Figure 42: Set ESP8266 Mode

4.2.3. AT+CWJAP_CUR—Connects to an AP; Configuration Not Saved in the Flash

Commands	Query Command: `AT+CWJAP_CUR?` Function: to query the AP to which the ESP8266 Station is already connected.	Set Command: `AT+CWJAP_CUR=<ssid>,<pwd>,[<bssid>][,<pci_en>]` Function: to set the AP to which the ESP8266 Station needs to be connected.

Response	`+CWJAP_CUR:<ssid>,<bssid>,<channel>,<rssi>`
	`OK`
		`OK`
		or
		`+CWJAP_CUR:<error code>`
		`FAIL`

Parameters	`<ssid>`: a string parameter showing the SSID of the target AP.	• `<ssid>`: the SSID of the target AP. • `<pwd>`: password, MAX: 64-byte ASCII. • `[<bssid>]`: optional parameter, the target AP's MAC address, used when multiple APs have the same SSID. • `[<pci_en>]`: optional parameter, disable the connection to WEP or OPEN AP, and can be used for PCI authentication. • `<error code>`: (for reference only) ▸ 1: connection timeout. ▸ 2: wrong password. ▸ 3: cannot find the target AP. ▸ 4: connection failed. This command requires Station mode to be enabled. Escape character syntax is needed if SSID or password contains any special characters, such as , or " or \.
Note	The configuration changes will NOT be saved in the flash.	

Examples	`AT+CWJAP_CUR="abc","0123456789"`
	For example, if the target AP's SSID is `"ab\,c"` and the password is `"0123456789"\"`, the command is as follows:
	`AT+CWJAP_CUR="ab\\\,c","0123456789\"\\"`
	If multiple APs have the same SSID as `"abc"`, the target AP can be found by BSSID:
	`AT+CWJAP_CUR="abc","0123456789","ca:d7:19:d8:a6:44"`	

Figure 43: ESP8266 Connect to Access Point AT Command

5.2.15. AT+CIPMUX—Enable or Disable Multiple Connections

Commands	Query Command: `AT+CIPMUX?`	Set Command: `AT+CIPMUX=<mode>` Function: to set the connection type.
Response	`+CIPMUX:<mode>` `OK`	`OK`
Parameters	`<mode>`: ▸ `0`: single connection ▸ `1`: multiple connections	
Notes	• The default mode is single connection mode. • Multiple connections can only be set when transparent transmission is disabled (`AT+CIPMODE=0`). • This mode can only be changed after all connections are disconnected. • If the TCP server is running, it must be deleted (`AT+CIPSERVER=0`) before the single connection mode is activated.	
Example	`AT+CIPMUX=1`	

Figure 44: Multiple Connection AT Command