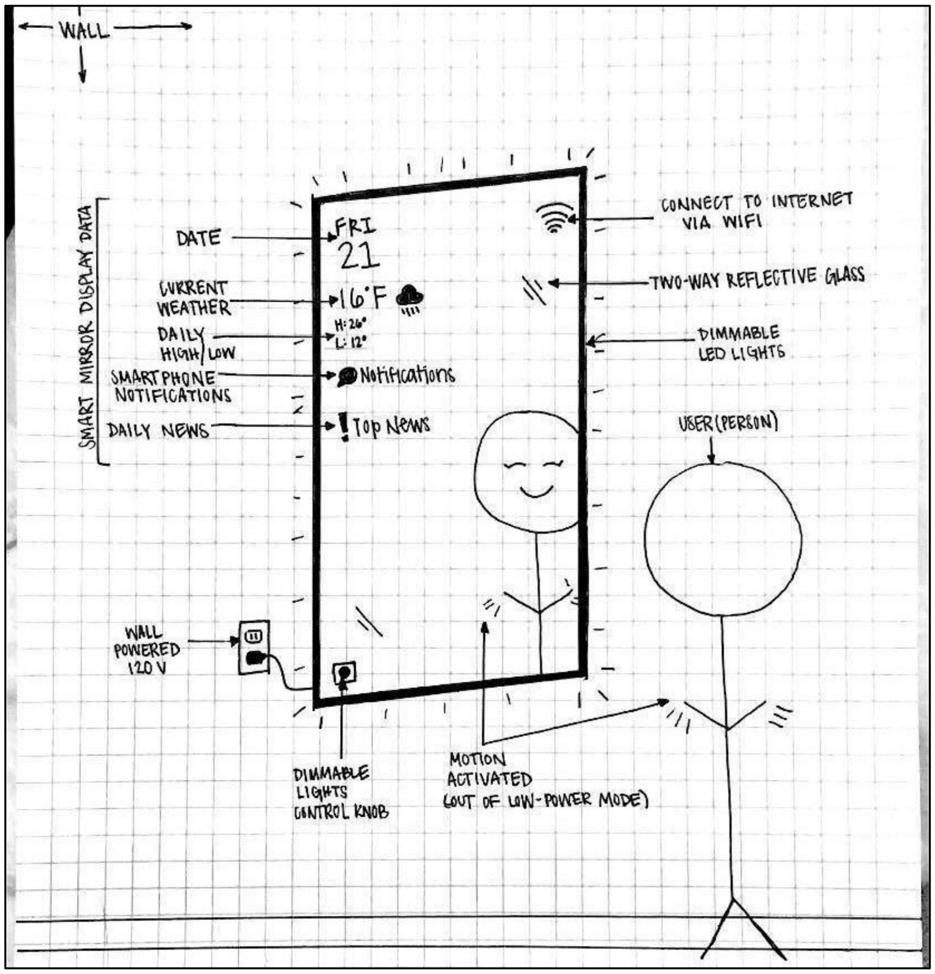
The Looking Glass

- Team 44 -

(Nicholas Schuch, Larissa Phillip, Seth Lee, Richard Rhee)

Design Document - Rev. 4



Product Sketch

TA : Yuhang Zhu

Professor : Mike Zoltowski

Contents

List of Figures

List of Tables

Revision Log

Date	Revision	Changes
01/19/2022	1	Initial Release
1/30/2022	2	Updated system block diagram - Removed AC wall adapter from power system, placed it outside our system blocks (since we won't be designing it), per feedback from Yuhang Zhu/R. Beasly from Solution Proposal Presentation Added glossary Added subsystem design sections
3/27/2022	3	Subsystem details - Updated specifications - Updated block diagrams/schematics - Added theory of operation Added system PCB model and schematics Updated glossary
4/29/2022	4	Updated requirements and stated final status of the project Updated glossary Updated system block diagram Added system photograph

Table 0.1: Revision Log

Glossary

API - Application Programming Interface
BLE - Bluetooth Low Energy
DEVKIT / DEVBOARD - Microprocessor Development Board
GPIO - General Purpose Input/Output pin
GUI - Graphical User Interface
HTTP - Hypertext Transfer Protocol
LDO - Low Dropout
MCU - Microcontroller Unit
NEMA - National Electrical Manufacturers Association
PIR - Passive Infrared
SPI - Serial Peripheral Interface

1. Introduction

1.1 Executive Description

As technology becomes rapidly more integrated and smart home devices become increasingly common, such as Google Nest and Amazon Alexa with their respective devices, there is always a central hub. The simple mirror is something that is a part of most peoples' morning routines, either in the bathroom or while getting prepared for the day. For most people, their morning routine also involves checking their smartphone for notifications, emails, weather, and time. The Looking Glass is a smart mirror designed to decrease the amount of time spent on our phones in the morning, and provide a one-stop quick look at the day ahead. The smart mirror will have a low power mode, and activate via motion sensors to display notifications, daily weather and top news stories. Many individuals get this information from their phone on computers, but we would like to integrate it into something we already use in the morning to increase efficiency of time. The Looking Glass would be the answer to this.

1.2 User Story

Steve, the student, is always peeking at his phone as he gets ready for his 7:30 lab in the morning. He is looking for a smart mirror system that will show him all the important information for his day ahead as he gets ready. He wants the system to display the date, time, and schedule for the day so that he knows how quickly he needs to get ready. He wants the system to display the current weather and the high/low temperatures for the day so he can get dressed accordingly. He also wants a short summary of his messages and emails as well as a few news articles so he can stay connected with current events. He doesn't want the system to use a lot of energy, so he wants the system to deactivate when he is not around. And activate when he is near the system. On the rare occasion when Steve has a date and must put on a tie, he wants to be able to see how bad the knot looks, so he wants the system to have lights that automatically adjust their brightness (Steve also thinks it would be great if the system wouldn't blind him when he gets near it at night).

2. Design Requirements

2.1 Requirements

1. The system shall display the current date.
2. The system shall display the current time.
3. The system shall display the current outside temperature.
4. The system shall display the current inside temperature.
5. The system shall display the daily low and daily high temperatures.
6. The system shall display the weather status for the day (e.g., sunny, rainy, partly cloudy, cloudy, etc.,).
7. The system shall display a user-created to-do list.
8. The system shall display current (daily) news articles.
9. The system shall display current notifications (e.g., phone notifications).
10. The system shall detect motion to activate.
11. The system shall have an additional means of activation that is not reliant on motion detection (e.g., push button or touch sensor).
12. The system shall be powered by a standard outlet (120V, USA.).
13. The system shall have a low-power mode (e.g., sleep-mode).
14. The system shall have a normal-operation, full-power mode.
15. The system shall connect to the internet.
16. The system shall retrieve information from the internet.

2.2 Factors Influencing Requirements

17. Public Health, Safety and Welfare
    a) Many people forget or do not have the time to check the weather in the morning, leading to being ill-prepared for the day. By providing a way to get this information hands-free and organically integrated in the morning routine, the user can prepare for the day effectively and efficiently.
18. Global Factors
    a) The system is currently designed to work with 120V wall-power outlets (i.e., USA-based). In the future, different designs or adapters will be necessary to allow use in

other countries with different power standards.

b) The system will gather world news from the internet to display to the user. This will help users stay connected with global current events.

c) The system will function anywhere with an internet connection, with no limits on deployment other than the power supply needs outlined above.

19.Cultural Factors

a) Mirrors are universally used around the world. This device provides additional functionality to a common household object which can integrate easily into people's lives.

20.Social Factors

a) Humans typically spend time in front of a mirror every morning (and throughout the day). This system improves that experience for people:

- More efficient by providing information that people would traditionally get through a newspaper or smartphone
- Check notifications and messages from socials and email

21.Environmental Factors

- The system will have a low-power (sleep) mode to limit power consumption when not in use.

- The system will be mounted on a wall, similar to existing mirrors, while only adding some additional depth from the wall.

22.Economic Factors

a) The system provides enhanced utility over a normal mirror. This increase in efficiency of time can be translated to productivity. As the saying goes, time is money.

b) The system will add minimal cost to the price over a regular mirror.

3. System Overview

3.1 System Block Diagram

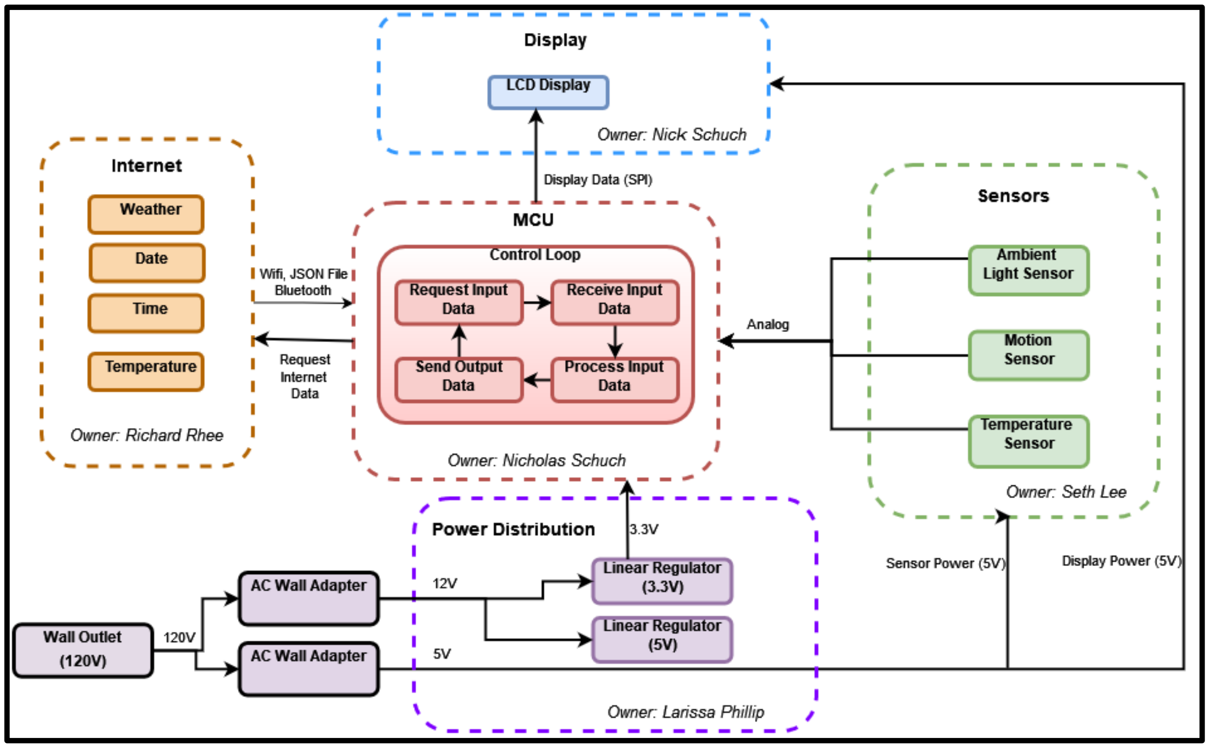


Figure 3.1.1: System Block Diagram

3.2 Integration Approach

Theory behind the system design, with reference to subsystem integration within your system – i.e., explain how it is supposed to work, but not whether it did actually work (for that info, see Section 7).

The integration approach can be split into two different plans: software and hardware.

The software, from the Internet subsystem and MCU subsystem respectively, was developed in parallel. Nicholas Schuch worked on the MCU software while Richard Rhee developed the Internet software. They collaborated when necessary (such as to compare implementations or to knowledge-share about the structure of their code), and made sure to always be working in the same environment (language, IDE, flashing routine, microcontroller, all the same). When it came time to integrate the two code-bases, it was discovered that the space on our ESP32 was being exceeded when trying to store all our code on it, so we switched from MicroPython to Arduino (a Python-base to a C/C++ base) in order to save space and also for further support/extensions/documentations). So both subsystems switched to Arduino IDE and C/C++ implementations, and integrations continued by combining the two codes. They shared the same code-structure from Arduino IDE (setup and loop function), so it became a matter of troubleshooting the order of initializations in the code in order to make sure that communication between the MCU and peripherals, and the MCU and the internet, were not interrupted. This needed to be done, because an improper initialization could lead to aberrant behavior.

For the hardware, The plan was to create a prototype for each subsystem independently and then integrate them together on the PCB, so that when the PCB was fully assembled, the system would be fully integrated. A back-up plan was put in place in case the PCB did not work when fully assembled. This back-up plan was to combine our individual breadboard prototypes of each subsystem into a new breadboard that contained the entire system. This breadboard would use the DevKit ESP32, as opposed to the ESP32 on the PCB.

3.3 System Photograph

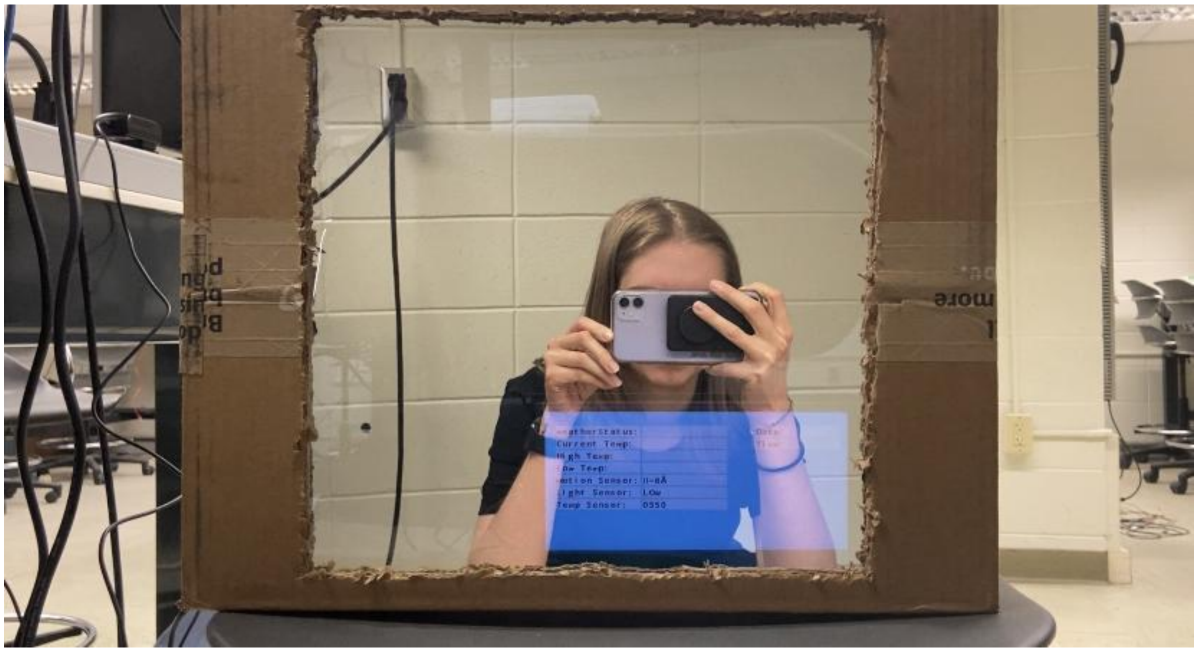

WeatherStatus:
Current Temp:
High Temp:
Low Temp:
Motion Sensor: [illegible]
Light Sensor: LOW
Temp Sensor: 0550
Date:
Time:

Figure 3.3.1: Functioning smart mirror

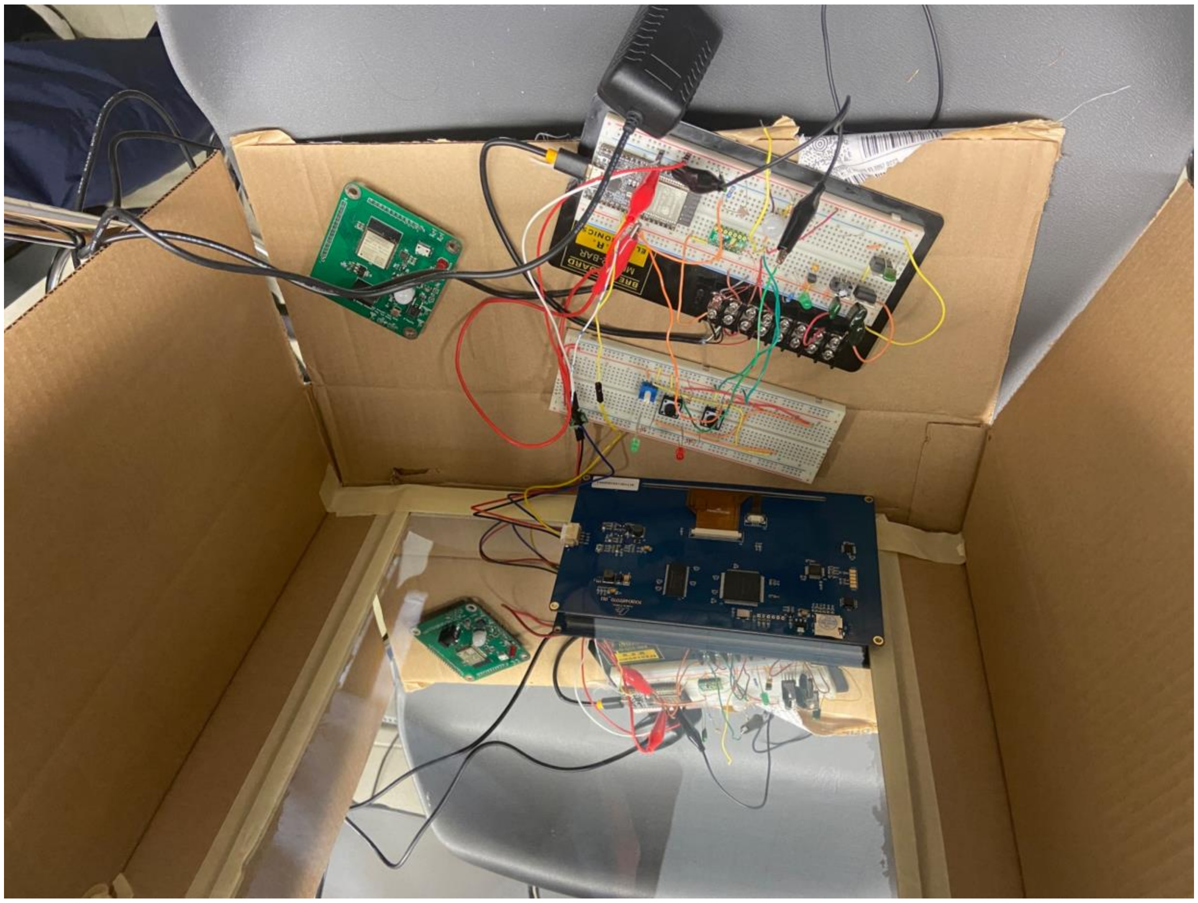


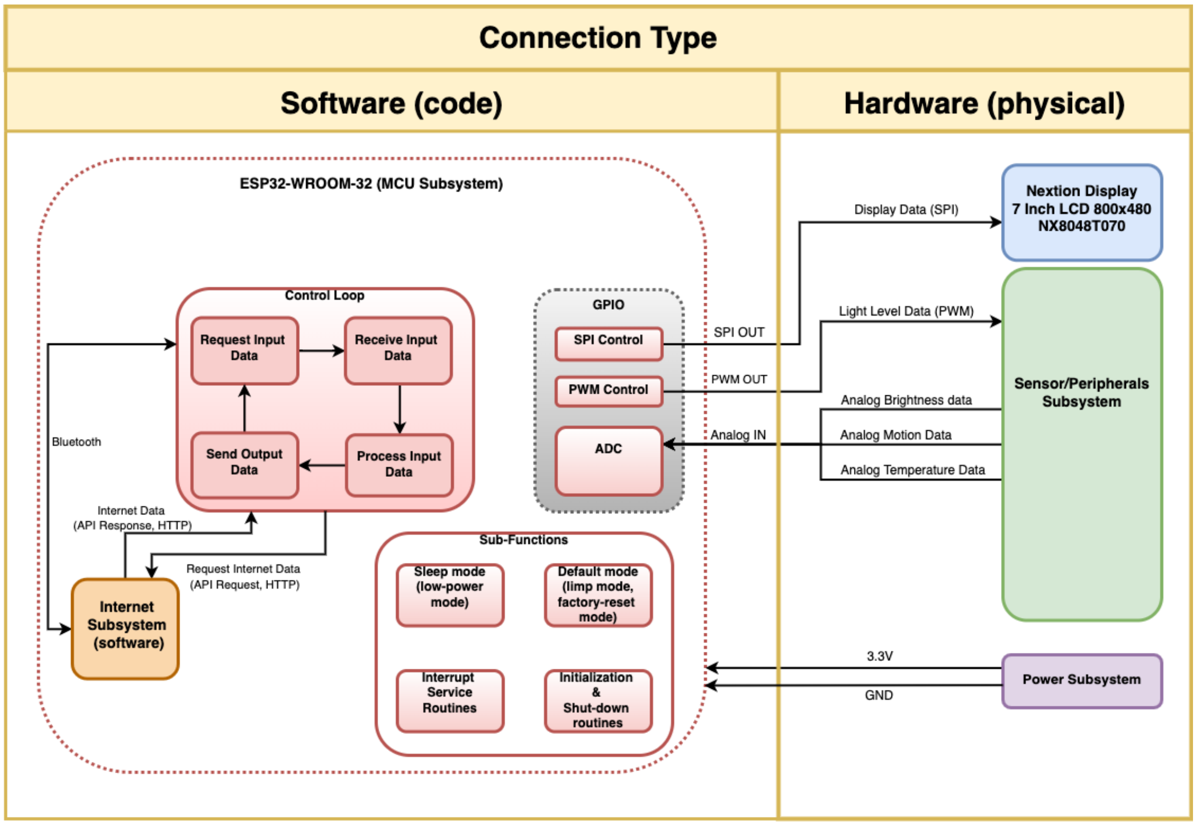
Figure 3.3.2: Inside the box

4. Subsystem Design

4.1 Subsystem 1 : Microcontroller Unit (MCU)

Subsystem Owner: Nicholas Schuch

4.1.1 Subsystem Diagrams



Connection Type
Software (code)
Hardware (physical)
ESP32-WROOM-32 (MCU Subsystem)
Control Loop
Request Input Data
Receive Input Data
Send Output Data
Process Input Data
GPIO
SPI Control
PWM Control
ADC
SPI OUT
PWM OUT
Analog IN
Bluetooth
Internet Data (API Response, HTTP)
Request Internet Data (API Request, HTTP)
Internet Subsystem (software)
Sub-Functions
Sleep mode (low-power mode)
Default mode (limp mode, factory-reset mode)
Interrupt Service Routines
Initialization & Shut-down routines
Display Data (SPI)
Nextion Display 7 Inch LCD 800x480 NX8048T070
Light Level Data (PWM)
Analog Brightness data
Analog Motion Data
Analog Temperature Data
Sensor/Peripherals Subsystem
3.3V
GND
Power Subsystem

Figure 4.1.1: MCU subsystem Block Diagram

4.1.2 Specifications

The MCU must facilitate the acquisition and delivery of various data to the user. That is to say, the MCU must gather information from the internet and other peripherals, and then process that information in order to decide how to display that information to the user.

1. The MCU must acquire (send & receive) data from the internet.
2. The MCU must acquire (send & receive) data from a smartphone.
3. The MCU must communicate with all peripherals.

4. The MCU must provide data & information to the display.
5. The MCU must provide a low-power mode (sleep-mode) to the system.
6. The MCU must control the brightness of the LED strip that illuminates the system.
7. The MCU must control the status of the system at all times.

We have selected the ESP32 as our MCU of choice. It meets all of the above specifications and was recommended by our mentor (Yuhang, TA) and investor (Prof. Zoltowski). MCU software was originally programmed in the MicroPython programming language using the Thonny IDE. This changed to using the Arduino IDE and the Arduino programming language (C/C++) after our midterm demonstration.

4.1.3 Subsystem Interactions

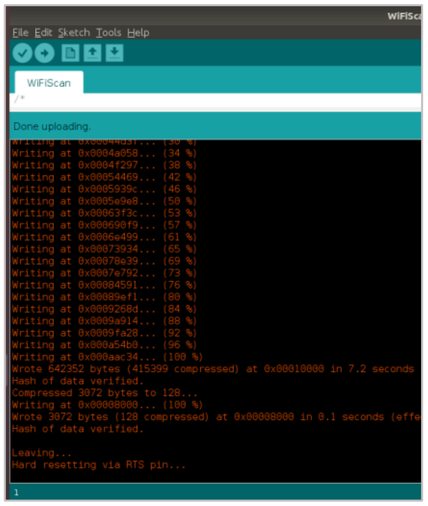
The MCU will act as the "brain" of the system. It will need to communicate with all the other components of the system.

1. Power Distribution:
    - The MCU will need to receive 3.3V and GND (0V) from the Power Distribution subsystem in order to function.
    - The MCU will not need to provide any information to the Power Distribution system.
2. Internet/Connectivity
    - The MCU and the Internet/Connectivity subsystems are intertwined. Simply put: the Internet/Connectivity subsystem is a large portion of the software that will be run on/by the MCU. It can be confusing to talk about inputs/outputs of a system when the systems are both parts of the same codebase.
    - The specifications for connecting to the internet and bluetooth can be found in the Internet and Connectivity subsystem design section.
    - The Internet/Connectivity subsystem will be implemented in Arduino (C/C++) and be stored on local storage of the MCU.
3. Sensors/Peripherals
    - The MCU will communicate with the display via SPI.
        - The MCU will provide the data for display to the user. This will be accomplished by sending visual data to the display with the Serial Peripheral Interface specification.
    - The MCU will drive & control the LED lighting of the system via Pulse Width Modulation (PWM).
        - PWM will be used to control both the brightness of the lights and their behavior (when they are on/off).
    - The MCU will receive analog data from the Motion Sensor via its onboard Analog-to-Digital Converter(s).
        - The MCU will take in the analog Motion Sensor data and then process that data to determine whether or not a user is present. This then determines whether the system should be awake or asleep.
    - The MCU will receive analog data from the Temperature Sensor via its onboard Analog-to-Digital Converter(s).
        - The MCU will take in the analog Temperature Sensor data and then process that data to display it to the user (convert the analog voltage value to a user-expected temperature value, then display that value).
    - The MCU will receive analog data from the Ambient Light Sensor via its onboard

Analog-to-Digital Converter(s).
        - The MCU will take in the analog Ambient Light data and then process that data to determine the necessary output LED light brightness (to be used in PWM in the section above).
    - The MCU will receive digital data from a pushbutton via a GPIO pin configured for digital input.
        - The MCU will detect a change in the digital value of the GPIO pin when the pushbutton is pushed, and then process that data to change from low-power to full-power mode as necessary.

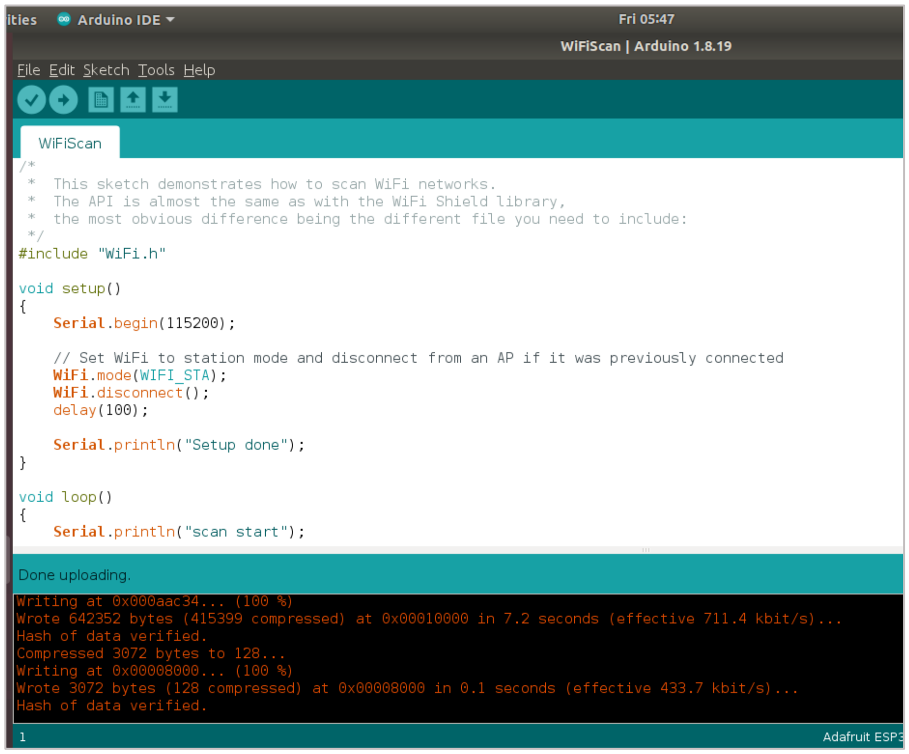
4.1.4 Theory of Operation/Algorithm

- We previously used the Thonny IDE and micro-USB to communicate & interact with the ESP32.
- We now use the Arduino IDE and micro-usb to communicate & interact with the ESP32.
- The basic control scheme of MCU subsystem is now based on the base control scheme of Arduino IDE. This consists of two functions:
    - A setup function
        - For initial configurations, this function runs once before the loop function starts to loop
    - A loop function
        - A function that loops to form the main operation of the MCU while the system is running/on
        - Most operations go here
- The basic flow of system is that the display is initialized and the SPI communication initiated, and then the ESP32 connects to WiFi. This all occurs in the setup function.
- After WiFi is connected, then the loop function is entered. The ESP32 then checks for inputs each time through the loop, and then, based on those inputs, generates some output. The inputs that the ESP32 checks for are: motion and push button input, and analog values from the sensors. The outputs that the ESP32 generates are: full-power mode vs. low-power mode (on or off), and displaying various values to the user.



```
Writing at 0x0004a058... (34 %)
Writing at 0x0004f297... (38 %)
Writing at 0x00054469... (42 %)
Writing at 0x0005939c... (46 %)
Writing at 0x0005e9e8... (50 %)
Writing at 0x00063f3c... (53 %)
Writing at 0x000690f9... (57 %)
Writing at 0x0006e499... (61 %)
Writing at 0x00073934... (65 %)
Writing at 0x00078e39... (69 %)
Writing at 0x0007e792... (73 %)
Writing at 0x00084591... (76 %)
Writing at 0x00089ef1... (80 %)
Writing at 0x0009268d... (84 %)
Writing at 0x0009a914... (88 %)
Writing at 0x0009fa28... (92 %)
Writing at 0x000a54b0... (96 %)
Writing at 0x000aac34... (100 %)
Wrote 642352 bytes (415399 compressed) at 0x00010000 in 7.2 seconds
Hash of data verified.
Compressed 3072 bytes to 128...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.1 seconds (effe
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

Figure 4.1.4.1: Flash process of the ESP32 with Arduino IDE with example WiFiScan program.



```
/*
 *  This sketch demonstrates how to scan WiFi networks.
 *  The API is almost the same as with the WiFi Shield library,
 *  the most obvious difference being the different file you need to include:
 */
#include "WiFi.h"

void setup()
{
    Serial.begin(115200);

    // Set WiFi to station mode and disconnect from an AP if it was previously connected
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);

    Serial.println("Setup done");
}

void loop()
{
    Serial.println("scan start");
```

Done uploading.

```
Writing at 0x000aac34... (100 %)
Wrote 642352 bytes (415399 compressed) at 0x00010000 in 7.2 seconds (effective 711.4 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 128...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.1 seconds (effective 433.7 kbit/s)...
Hash of data verified.
```

Figure 4.1.4.2: Code sample from the WiFiScan program that shows the setup and loop functions that make up the general code structure for Arduino IDE programs (in C/C++).

4.1.5 Core ECE Design Tasks

The core ECE design task in the MCU subsystem is Embedded System Design. Embedded System Design is a large task. Multiple courses at Purdue teach elements of Embedded System Design, especially in regard to Microcontrollers.

- ECE 362 - [Microprocessor Systems and Interfacing]
    - ECE 362 introduced the practice of programming a microcontroller for use with a variety of peripherals, and with a variety of communication protocols. Many of the labs and lessons from ECE 362 will be directly applied to this MCU subsystem. The use of the ADC, SPI, GPIO pins, and other microcontroller sub-functions were all learned in ECE 362.
- ECE 40862 - [Software for Embedded Systems]
    - ECE 40862 continued the microcontroller learning curve with more advanced concepts than ECE 362. These advanced concepts included connecting to WiFi, sending/receiving HTTP requests, and sending/receiving push notifications with a smartphone from a microcontroller. All of these will be instrumental in our system.

4.1.6 Schematics / Parts

- The MCU selected by our team is the ESP32-WROOM-32. This chip contains a plethora of peripherals, including all the peripherals we need for our system on-chip (WiFi, Bluetooth, ADC, PWM, and SPI).

- We will be programming our ESP32-WROOM-32 with C/C++ in the Arduino IDE. Previously this was the MicroPython programming language.
- Arduino submodules of interest:
    - ITEADLIB_Arduino_Nextion - Support library for interfacing our Nextion display with Arduino-based MCU
    - ARDUINO-ESP32-2.0.2 - Support library for ESP32 for Arduino development
        - Very extensive library with support for most ESP32 functionality
- MicroPython Modules of interest:
    - Machine - general board control (clock frequency, pinouts, sleep)
        - Pin, a sub-module of Machine, for specific pin access (GPIO/ADC/PWM/etc.)
        - Timer, sub-module of Machine, for access to hardware timer(s)
        - PWM, sub-module of Machine, for access to PWM functions
        - ADC, sub-module of Machine, for access to ADC control
        - SPI, sub-module of Machine, for access to SPI control
    - ESP32 - For ESP32 specific sub-functionality (i.e. built in sensors)
    - Network - WiFi/connectivity
- We will use the Arduino IDE and micro-USB to communicate & interact with the ESP32. Previously Thonny IDE and micro-USB.
- Preliminary development took place on the Adafruit HUZZAH32-ESP32 Feather Board (dev board), and then transition was planned to an integrated prototype system after the PCB is complete. However, due to the PCB not fully working, we instead relied on a back-up plan that utilized our prototype breadboards and the AdaFruit HUZZAH32-ESP32 Feather Board for our final system/demonstration.

Figure 4.1.6.1: ESP32 Dev-Kit (Adafruit HUZZAH32 - ESP32 Feather Board) [ESP-WROOM-32]

- GPIO pins will not be decided until the PCB design process is initiated and completed.
    - Current Pins used are on the devkit we are using to develop the project; the finalized PCB pins are also known, as seen in the system PCB/Schematic section.
    - Current Pinout of the HUZZAH32 Featherboard (devkit)
- The Display we ended up choosing for this system was the Nextion NX8048T070 LCD Display 800x480. We chose it primarily for its size. It is more difficult to find large LCD displays (larger than 2-3 inches diagonally), and so we went with one that had the features we needed and the size desired, with the lower cost. We wanted a bigger display because the system needed to display many different pieces of information and it needed to be readable from a few feet away from the display (normally a person is a few feet away from a mirror).

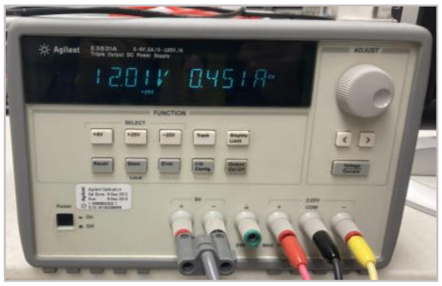
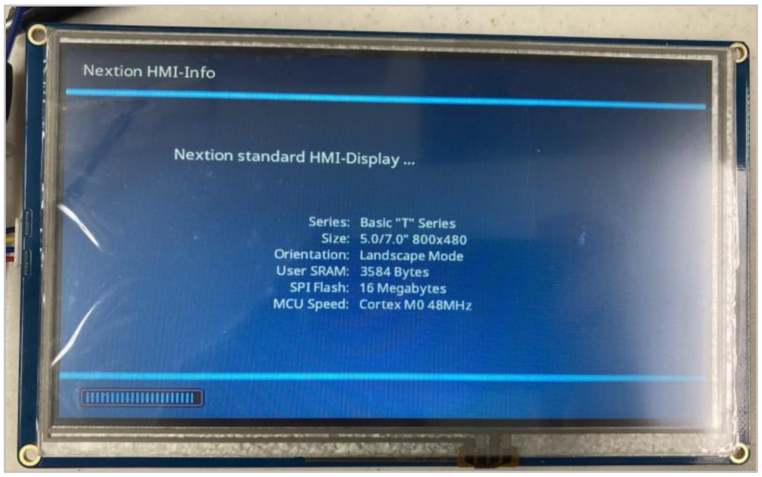


Figure 4.1.6.2: Standard current draw of the NX8048T070

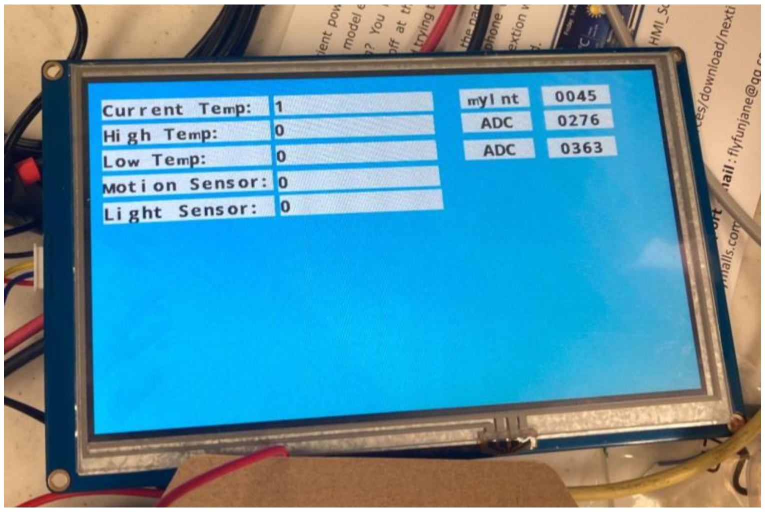


Nextion HMI-Info

Nextion standard HMI-Display ...

Series: Basic "T" Series
Size: 5.0/7.0" 800x480
Orientation: Landscape Mode
User SRAM: 3584 Bytes
SPI Flash: 16 Megabytes
MCU Speed: Cortex M0 48MHz

Figure 4.1.6.3: Standard Boot-up screen for the NX848T070 display.



Current Temp:	1	myInt	0045
High Temp:	0	ADC	0276
Low Temp:	0	ADC	0363
Motion Sensor:	0		
Light Sensor:	0		

Figure 4.1.6.4: NX8048T070 during testing to display the ADC values from the ESP32 to the user.

4.1.7 Specification Measurements

MCU Specifications Status: Expectation for Final Demo

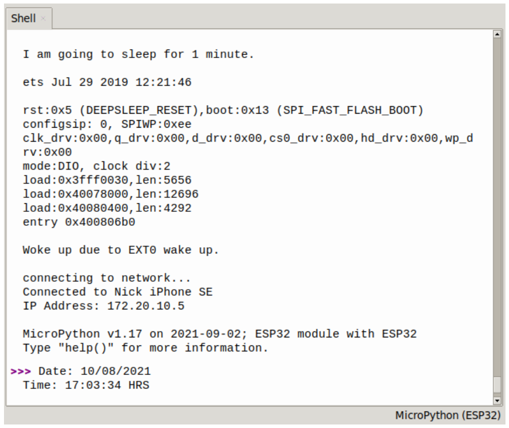
(Met / Unmet / Work In Progress (partially mett)

1. The MCU must acquire (send & receive) data from the internet.
2. The MCU must acquire (send & receive) data from a smartphone.

3. The MCU must communicate with all peripherals.
4. The MCU must provide data & information to the display.
5. The MCU must provide a low-power mode (sleep-mode) to the system.
6. The MCU must control the brightness of the LED strip that illuminates the system.
7. The MCU must control the status of the system at all times.

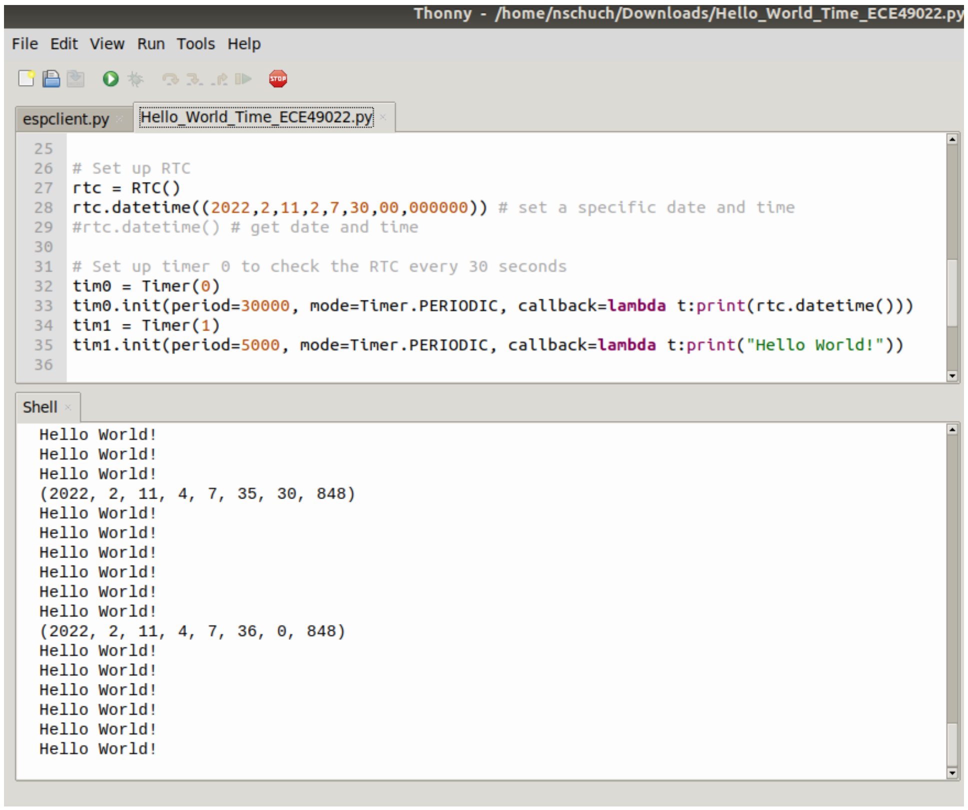
Note: The partially met (yellow) specification #3 is due to a technicality. The MCU does communicate with all peripherals that are in use (connected to the system, providing information), but it is not communicating with all peripherals because we currently do not support all peripherals (namely the LED strip was not implemented).

The MCU demonstrated meeting the above specifications in the Final Demonstration video. Below are some screenshots from the Midterm Demonstration to show sample outputs from the ESP32 to the terminal.:



```
I am going to sleep for 1 minute.

ets Jul 29 2019 12:21:46

rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_d
rv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:5656
load:0x40078000,len:12696
load:0x40080400,len:4292
entry 0x400806b0

Woke up due to EXT0 wake up.

connecting to network...
Connected to Nick iPhone SE
IP Address: 172.20.10.5

MicroPython v1.17 on 2021-09-02; ESP32 module with ESP32
Type "help()" for more information.
>>> Date: 10/08/2021
Time: 17:03:34 HRS
```

Figure 4.1.7.1: Example Thonny output during testing.


Thonny - /home/nschuch/Downloads/Hello_World_Time_ECE49022.py
File Edit View Run Tools Help
espclient.py
Hello_World_Time_ECE49022.py
Shell

```

# Set up RTC
rtc = RTC()
rtc.datetime((2022,2,11,2,7,30,00,000000)) # set a specific date and time
#rtc.datetime() # get date and time

# Set up timer 0 to check the RTC every 30 seconds
tim0 = Timer(0)
tim0.init(period=30000, mode=Timer.PERIODIC, callback=lambda t:print(rtc.datetime()))
tim1 = Timer(1)
tim1.init(period=5000, mode=Timer.PERIODIC, callback=lambda t:print("Hello World!"))

```

```
Hello World!
Hello World!
Hello World!
(2022, 2, 11, 4, 7, 35, 30, 848)
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
(2022, 2, 11, 4, 7, 36, 0, 848)
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

Figure 4.1.7.2: Example Thonny output during testing.

4.1.8 Standards

The communication protocols utilized by the system are standardized: SPI (display) and HTTP, standard: RFC 2616 (API requests for internet information).

4.2 Subsystem 2 : Internet/Connectivity

Subsystem Owner: Richard Rhee

4.2.1 Subsystem Diagrams

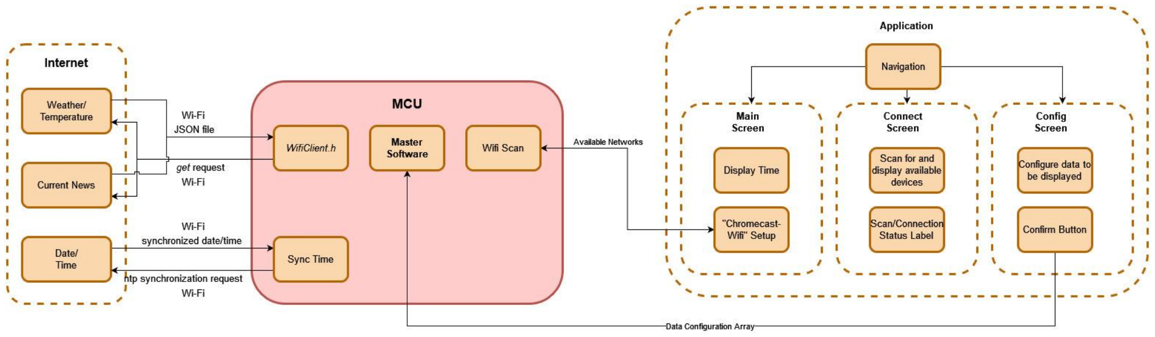


Figure 4.2.1: Internet/Connectivity subsystem Block Diagram

4.2.2 Specifications

As one of the selling points for this device is to save time during the morning routine as well as throughout the day, it should be relatively responsive and quick to gather, parse, and display the information. Because the focus of this subsystem is on gathering data from the web via a wifi connection, there are minimal specifications related to hardware.

Specifications:

1. The device must be capable of Bluetooth Low Energy.
2. The device must be capable of 2.4 GHz Wi-Fi.
3. The device must request and receive weather information, which includes current temperature, daily high/low temperatures, current weather condition, and displays the information to the user.
4. The device must have a GUI for the user to interact with.
5. The device must display current relevant news articles.

4.2.3 Theory of Operation/Algorithm

The Wi-Fi connection is first established by putting the ESP32 in “setup mode” when power is first provided. The ESP32 looks for a Bluetooth connection with a smartphone. Once a connection is established, the ESP32 scans for available Wi-Fi networks. The networks are displayed on the Android application, where the user can enter a number for the Wi-Fi network to connect to, then enter a password. Once the user selects a network and enters the correct password, the ESP32 establishes a Wi-Fi connection.
Once setup is complete and a Wi-Fi connection is established, the device will be in normal mode. The device is activated by either a motion sensor or a pushbutton press. The motion sensor can be activated by walking in front of the device, and the pushbutton can be pressed by the user. Once activated, the ESP32 will make two HTTP GET requests: one to OpenWeatherMap API to receive weather data, which includes current temperature, daily low and high temperatures, and current weather condition, and one to NewsAPI to receive current relevant news article titles with their authors and dates published. This data is received in the form of JSON, which is parsed and displayed for the user.

The device has an Android application which the user needs to interact with the device. The application has 3 pages (screens): the main screen, the connection screen, and the configuration screen. The main screen displays the current time. The connection screen is used for establishing a BLE connection to the ESP32 and a Wi-Fi connection. The configuration screen is incomplete, as the user-configuration for data to be displayed on the screen was not finished.

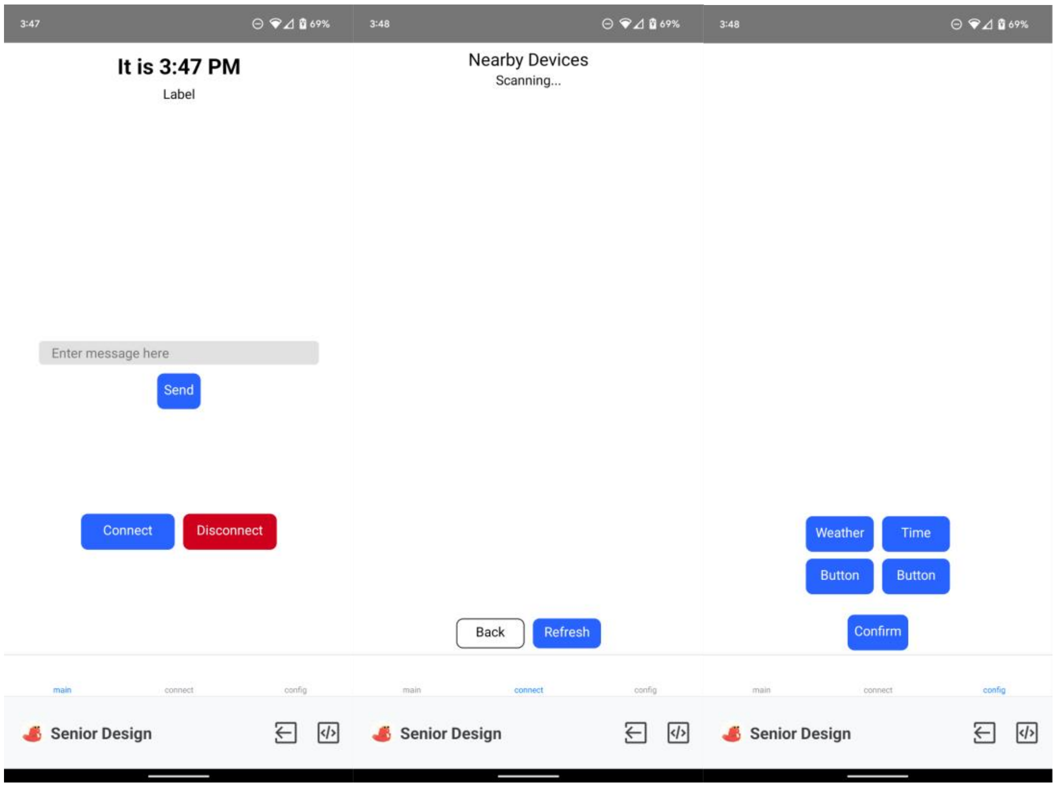


Figure 4.2.2: Android application GUI main screen, connection screen, and configuration screen in respective order

4.2.4 Subsystem Interactions

This subsystem will mainly be interacting with the microcontroller subsystem. The core function of this subsystem is to gather data from the web and the connected device, then parse the data to display to the user. The microcontroller has on-board Wi-Fi functionality which will be used to connect to the web. To set up Wi-Fi, the ESP32 will set up a BLE advertiser, which the user can connect to. Then, the user will connect the smart mirror to Wi-Fi through the phone. Once a connection is established, the smart mirror will be able to perform the various functions described in the specifications.

4.2.5 Core ECE design tasks

- Microcontroller Interfacing
    - ECE 362 (Microprocessor Systems and Interfacing)
        - The microcontroller uses on-board Wi-Fi and Bluetooth for various functionalities, which need to be configured. These protocols will enable our device to display information to the user.

- Data Collection and Parsing / Software Development
    - ECE 20875 / ECE 461 (Python for Data Science / Software Engineering)
        - There are different kinds of information that need to be gathered from the web, including weather/temperature and date/time. Our device software will interface with existing APIs to get this information. The data will then need to be parsed and organized to be displayed to the user.

4.2.6 Schematics / Parts / Algorithm

- ESP32 dev board - An ESP32 dev board will be used for prototyping and coding before developing our own custom board. The code for the dev board should be identical if not very close to the code to be used on the custom board, as they should use the same microcontroller.
- Arduino IDE (C++) - A widely used coding environment for microcontrollers. Arduino has a huge library for interfacing with hardware that others have already written.
- Wi-Fi router - Used for testing Wi-Fi connection and HTTP GET requests. In our case, we used a smartphone hotspot with no WPA2/WPA3 security protocols for ease for testing and demo.
- Smartphone - The user's smartphone will be the main way of interacting with the display, aside from walking in front of the mirror to view the information. The user will configure the smart mirror's Wi-Fi, as well as choose what information is displayed.

4.2.7 Standards

- Wi-Fi - Family of wireless network protocols commonly used for local area networking of devices and Internet access, allowing nearby digital devices to exchange data by radio waves.
- Bluetooth - Short-range technology standard that is used for exchanging data between fixed

and mobile devices over short distances using UHF radio waves in the ISM bands, from 2.402 GHz to 2.48 GHz and building personal area networks.

- Bluetooth Low Energy - A wireless personal area network technology, independent of classic Bluetooth, designed and marketed by the Bluetooth Special Interest Group. Original specification was developed by Nokia in 2006 as Wibree, and was integrated into Bluetooth 4.0 in December 2009 as Bluetooth Low Energy.

4.3 Subsystem 3 : Sensors/Peripherals

Subsystem Owner: Seth Lee

4.3.1 Subsystem Diagrams

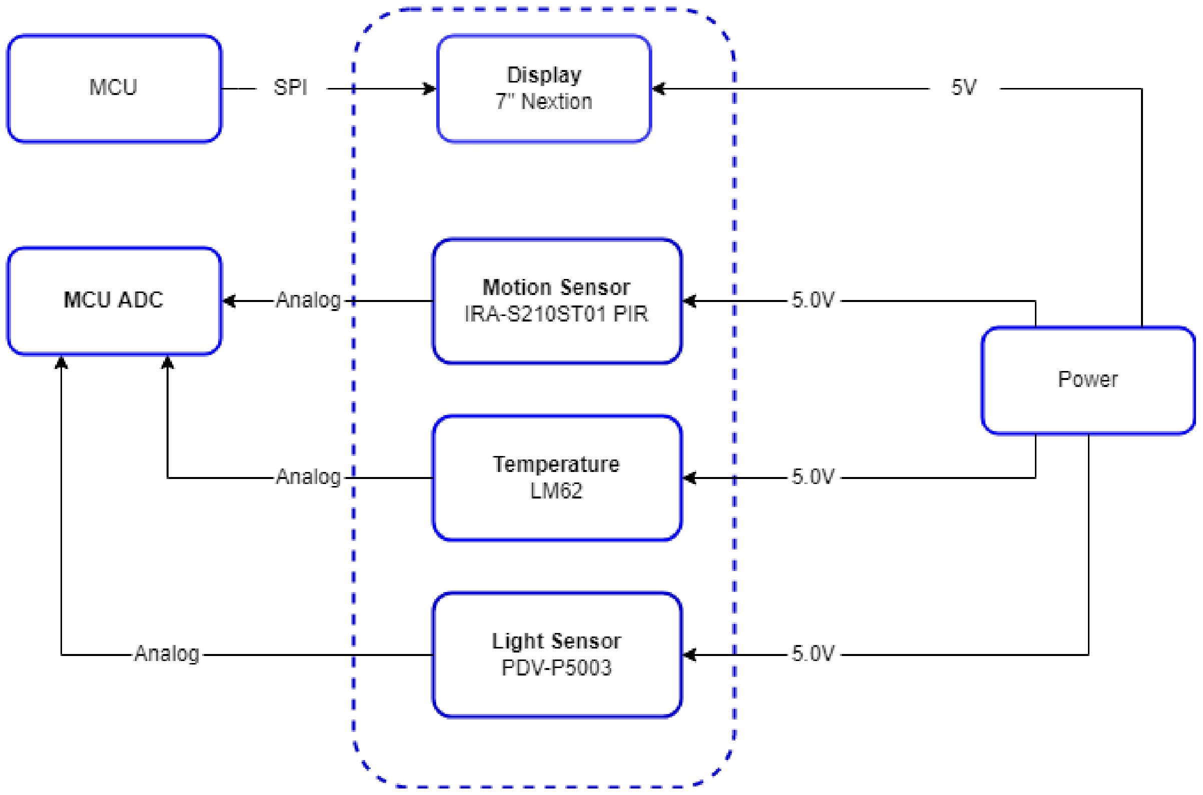


Figure 4.3.1: Peripherals subsystem Block Diagram

4.3.2 Specifications

The sensor modules gather information about the world around to facilitate other functions included on the device. The data will be sent to the MCU to determine what values need to be changed whether it is on the display or the intensity of light. More sensors or functionality may be added as necessary to enhance the device.

Specifications:

1. The temperature sensor must not have an error greater than 2ºC
2. The light sensor will have a resistance of 0.7 Ω/Lux
3. Each sensor must be directly powered by the power system and not MCU for stable power

4. Each sensor must communicate with the MCU.
5. The motion sensor must indicate movement at 1 meter.

4.3.3 Subsystem Interactions

1. Power Distribution:
    a. The sensors will mainly be using 5V voltage draw and the display is using the same.
    b. Currently there are no parts of the sensor system that will power any other parts of the system.
2. Internet/Connectivity:
    a. The sensors are not currently projected to connect to this subsystem because it does not require any information from the internet nor are we uploading any information to the internet.
3. MCU:
    a. The main information transfer will be between the MCU and the sensors with the light and temperature sensors updating information through requests by the MCU.
    b. The main communication between the sensors and MCU will be through sending an analog signal to the MCU's ADC because most of them are passive.
    c. The display is a TFT LCD and will receive data from the microcontroller with data to display through SPI.

4.3.4 Theory of Operation/Algorithm

Each sensor will be used in their own role. The temperature sensor will be giving an output voltage that corresponds to a certain temperature. The voltage will be received by the microcontroller through an ADC channel and converted into a temperature based on previous data and is displayed on the screen as a reference for the user.

The PIR sensor is used to detect movement. It sends an impulse of voltage when detecting movement. This signal is sent to the microcontroller through an ADC channel and will be used to tell the device to wake up from low power mode.

The light sensor is an ambient light sensor that tells the microcontroller how much light is being detected in the room. This was originally planned to be part of an adjustable light system. The photoresistor is placed in a voltage divider circuit, and the voltage variance corresponds to different light intensities. This voltage signal is sent to the microcontroller through an ADC channel and converted to a lux value using previous data.

4.3.5 Core ECE design tasks

IC selection

- ECE 20007/ ECE 20008: Electrical Engineering Fundamentals I/II / ECE 362: Microprocessor System & Interface
    - In these labs we learned how to read data sheets and how to compare different ICs for different specifications.

PCB Design

- ECE 49022 (Electrical Engineering Senior Design Projects)
    - We have covered a little bit about circuit design, but will probably be introduced more in the future.

4.3.6 Schematics / Parts / Algorithm

Temperature Sensor: TI LM62 Precision Centigrade Temperature Sensor
This sensor is very precise and has a wide range of temperature sensitivity. It may be a little unnecessary on the range, but it is good for understanding the current temperature inside. It also has a very low impedance, which correlates to a lower load on the system.

Motion Detector: Murata IRS-B210ST01 PIR sensor
This component can detect the whole human body at a 5m range that is more than necessary for our device but will work well with what we envision would be the final design. It also detects decently wide, about a 70º angle, which allows it to detect a varied amount of angles of approach to trigger the device to activate.

Light/luminance sensor: PDV-P5003
This is a photoconductive photocell that changes resistance based on luminance. We plan on using it by measuring changes in voltage to understand the level of luminance. It is precise with a precision of 0.7 Ω/Lux and within a range that is suitable for what we anticipate as the device interacts with human eyes.

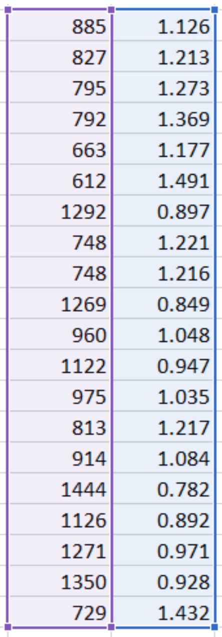
Display: Nextion Display 7 inch NX8048T070
This is our current display. Although it is somewhat small at 7", it will be enough for the prototype system we currently have in mind. It uses SPI to communicate which is similar to the display we used in ECE 362.

4.3.8 Specification Measurements

Specification 1: The temperature sensor must not have an error greater than 2ºC
This specification has been changed because the sensors that we planned to use with this are out of stock. Instead we have new replacement sensors. The original error was .75ºC as demonstrated below.

This was demonstrated through the use of the wall thermostat, the graph given in the datasheet, and the component in a circuit that was in similar conditions to the test one.
It showed that the temperature matched the expected output voltage.

Specification 2: The light sensor will have a resistance of 0.7 Ω/Lux



885	1.126
827	1.213
795	1.273
792	1.369
663	1.177
612	1.491
1292	0.897
748	1.221
748	1.216
1269	0.849
960	1.048
1122	0.947
975	1.035
813	1.217
914	1.084
1444	0.782
1126	0.892
1271	0.971
1350	0.928
729	1.432

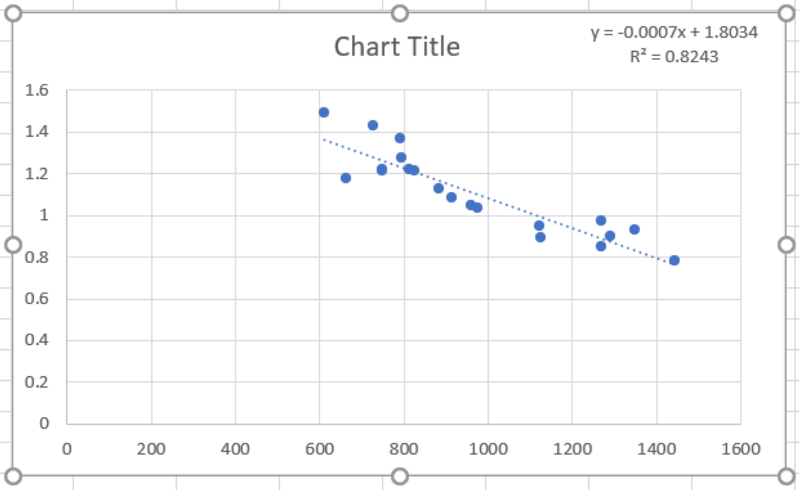


Table 4.3.2: Light Sensor recordings

4.3.7 Standards

I do not think that there are many standards that are addressed by this subsystem. The display may be the only part with reference to the standard SPI communication system.

4.4 Subsystem 4 : Power Distribution

Subsystem Owner: Larissa Phillip

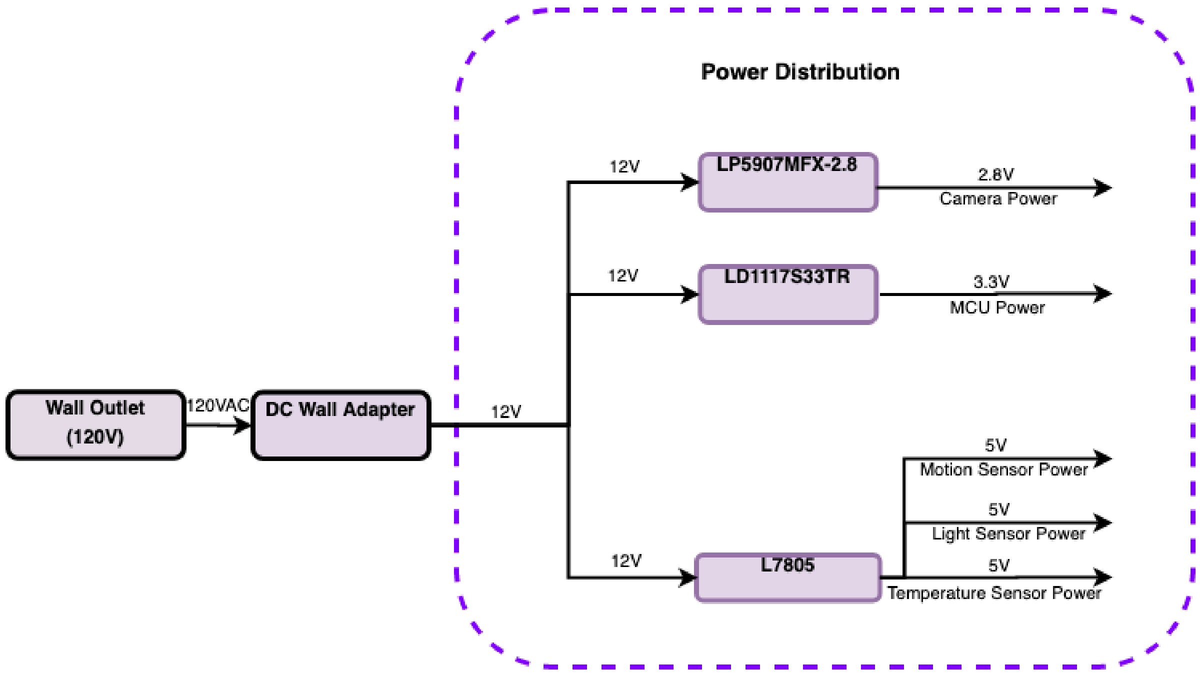


Figure 4.4.1: Power Distribution subsystem Block Diagram

4.4.2 Subsystem Interactions

- The power system will provide 3.3V to the MCU subsystem at 500mA.
- The power system will provide 5V to the light sensor at 50mA.
- The power system will provide 5V to the temperature sensor at 50mA.
- The power system will provide 5V to the motion sensor at 50uA.

All components mentioned above and found in figure 4.4.1 are explained further below in the algorithm section.

4.4.3 Theory of Operation/Algorithm

The power distribution system is designed to meet the power specifications of the various components found in the additional subsystems. The power distribution subsystem will provide the appropriate input voltage and power to each of the subsequent subsystems. As part of the user interface, the display will be powered by the 12V rail. The power system will provide 3.3V to the MCU subsystem at 500mA. 5V will be supplied to the motion, temperature and light sensors within the sensor subsystem to meet power requirements. Lastly, 2.8V will be available to supply the camera component of the sensor subsystem via a voltage regulator.

A 12V rail is connected to a DC wall converter for simplicity, stepping the 120 VAC down to 12V DC. Attached to the 12V rail is a pull down resistor and "heartbeat" LED to indicate system power. The LED will be on when the system has power. This rail is then fed into three different linear regulators to step down the voltage again to meet the power specifications of the remaining subsystems, including the microcontroller subsystem at 3.3V and sensor subsystem, consisting of a temperature sensor, a motion sensor and a light sensor, all requiring 5V. The power subsystem is also capable of supplying 2.8V for a camera subsystem, should this design move forward with this subsystem in the future.

The decision to go with a voltage regulator, instead of stepping down voltage with a simple resistor, was made based on the understanding that the load draw will vary. Because a resistor will follow Ohm's law, and voltage will hence vary with current, a voltage regulator is a safer option to maintain a constant output voltage. There is concern that voltage regulators may generate too much heat. Alternatively, a switching regulator could be an option, but causes concern for potential noise generation that may impact both the MCU and sensor subsystems. The largest power draw from the power distribution subsystem will come from the MCU subsystem, requiring 500mA of current during peak operation.

The power regulation for the 3.3V, 5V and 2.8V rails is done with a linear regulator, considered a simple design, due to its excellent stability and simplicity of integration over its efficiency. For comparison, a buck or switching converter may be more energy and cost efficient, but employs a longer, more complicated design process to achieve similar resulting characteristics. The microcontroller for the system does not list a specific maximum voltage ripple, but such a high-speed device should be provided with a stable power source. Additionally, the Vpp ripple will be minimized by filtering with a ferrite bead, and capacitors.

4.4.4 Specifications

1. The system power will be provided via a 12V DC wall adapter.

2. The power distribution system must provide 3.3 V at 500 mA to the ESP32-WROOM-32 of the MCU subsystem.

3. The power distribution system must provide 5 V at 50 mA for the motion sensor in the sensor subsystem.

4. The power distribution system must provide 5 V at 50 μA for the temperature sensor in the sensor subsystem.

5. The power distribution system must provide 5 V at 40 mA for the light sensor in the sensor subsystem.

4.4.5 Core ECE design tasks

- Power Distribution Circuit Design
  - ECE 20100 / ECE 20200 (Linear Circuit Analysis 1/Linear Circuit Analysis 2)
    - To come up with a complete solution to powering multiple subsystems with different power specifications required heavy reliance on analyzing linear circuits with power supply and passive elements. This is important when configuring the most efficient circuit design for the system.
- IC Selection
  - ECE 20700 / ECE20800 (Electronics Measurements Techniques/Electronic Devices and Design Laboratory)
    - Choosing the appropriate IC element to incorporate into the circuit design relied on the knowledge and skills gained during Electronics Measurements Techniques & Electronic Devices and Design Laboratory. This includes

researching various IC datasheets and reading and interpreting the power specifications accordingly to provide accurate values to meet the system's power requirements.

4.4.6 Schematics / Parts / Algorithm

- LP5907MFX-2.8 - 250 mA Low Noise Regulator (12V to 2.8V)
- LD1117S33TR – Fixed LDO Voltage Regulator (12V to 3.3V)
- L7805 - Linear Voltage Regulators 5.0V 1.0A Positive(12V to 5V)

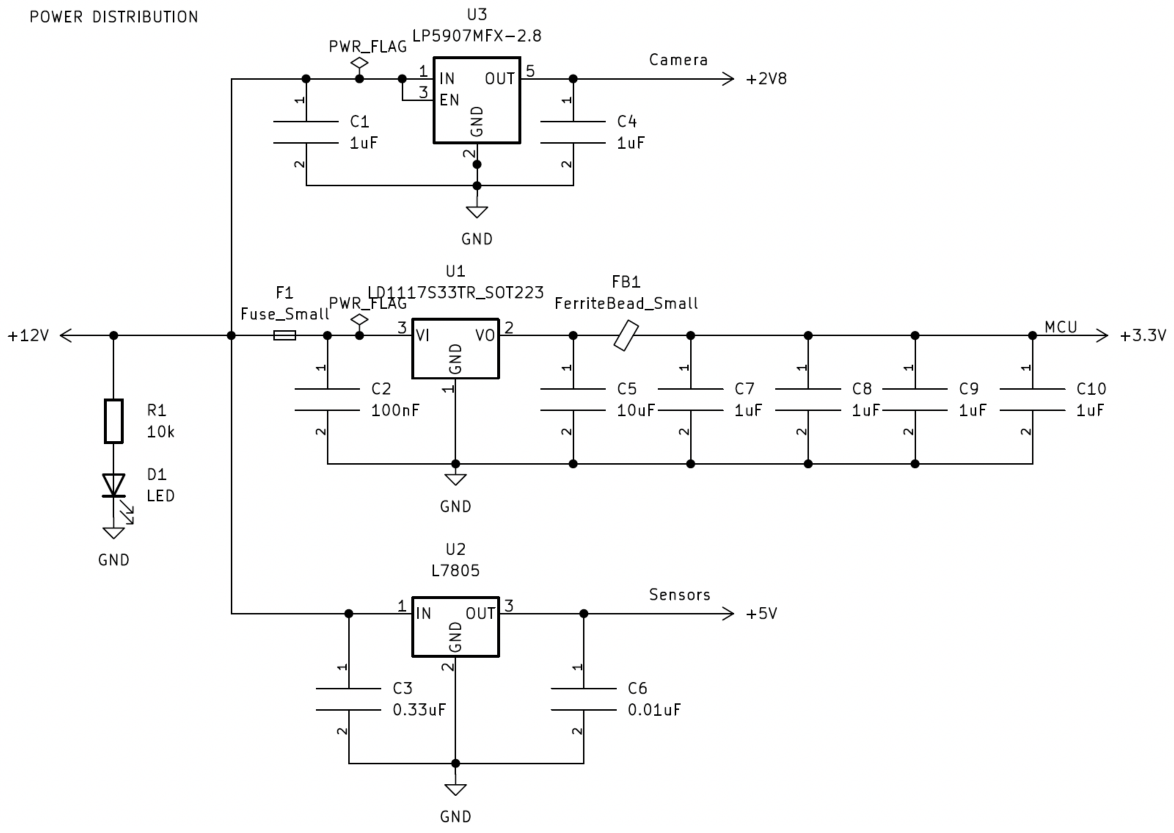
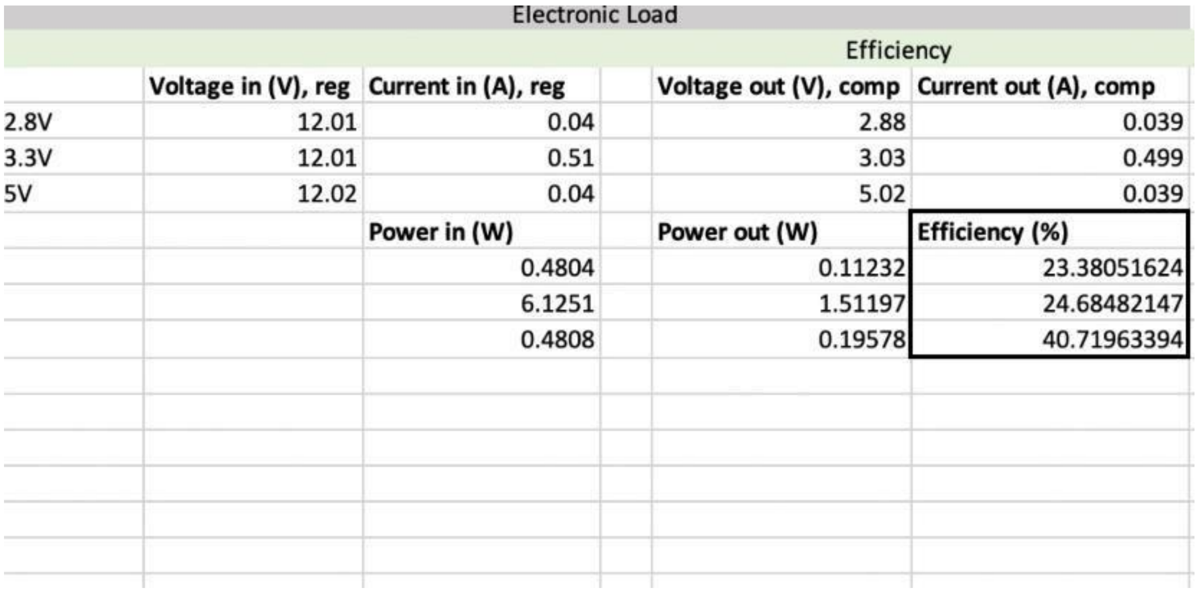


Figure 4.4.2: Power Distribution subsystem Schematic

4.4.7 Specification Measurements

Specification 1: The device operates with 12Vpp input. The table below demonstrates the system's ability to take 12 V input and output various voltages to meet the subsystem requirements.



Electronic Load					
				Efficiency	
	Voltage in (V), reg	Current in (A), reg		Voltage out (V), comp	Current out (A), comp
2.8V	12.01	0.04		2.88	0.039
3.3V	12.01	0.51		3.03	0.499
5V	12.02	0.04		5.02	0.039
		Power in (W)		Power out (W)	Efficiency (%)
		0.4804		0.11232	23.38051624
		6.1251		1.51197	24.68482147
		0.4808		0.19578	40.71963394

Table 4.4.1: Electronic load measurement and efficiency calculations

Specification 2: The power distribution system must provide 3.3 V at 500 mA to the ESP32-WROOM-32 of the MCU subsystem.

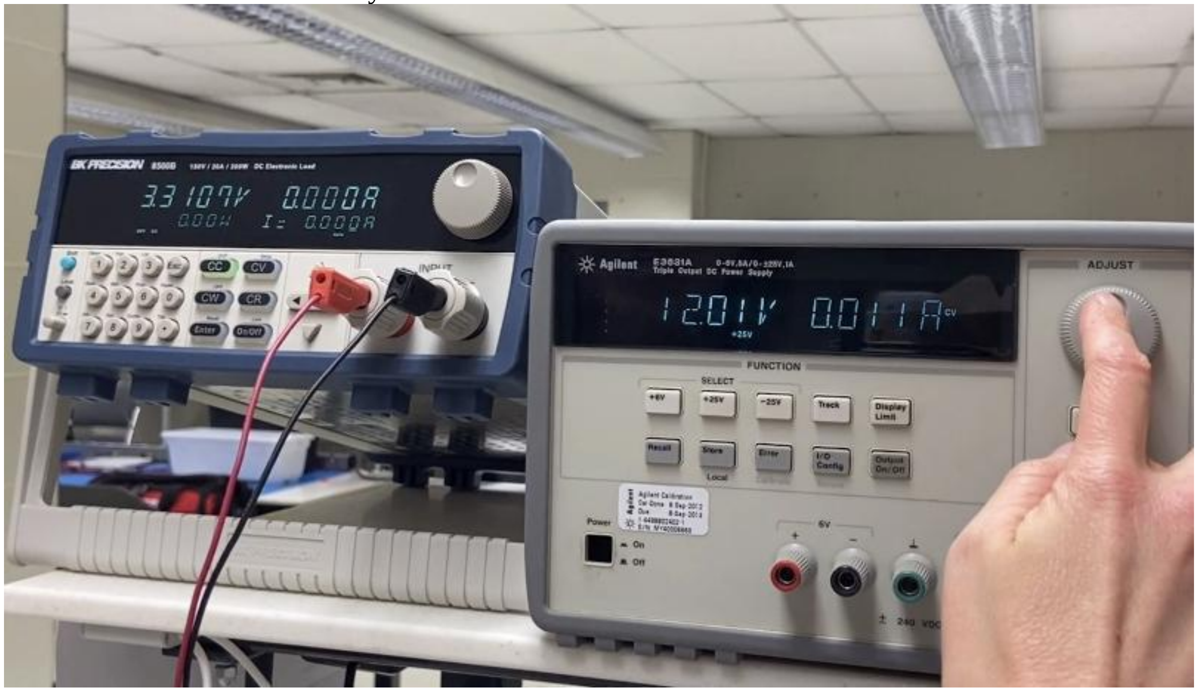


Figure 4.4.3: Simulated electronic load data for 3.3V linear regulator

Specification 3: The below plot proves the system provides stable output of 5V up to 200mA of load current, which is well above the expected current draw for the sensors.

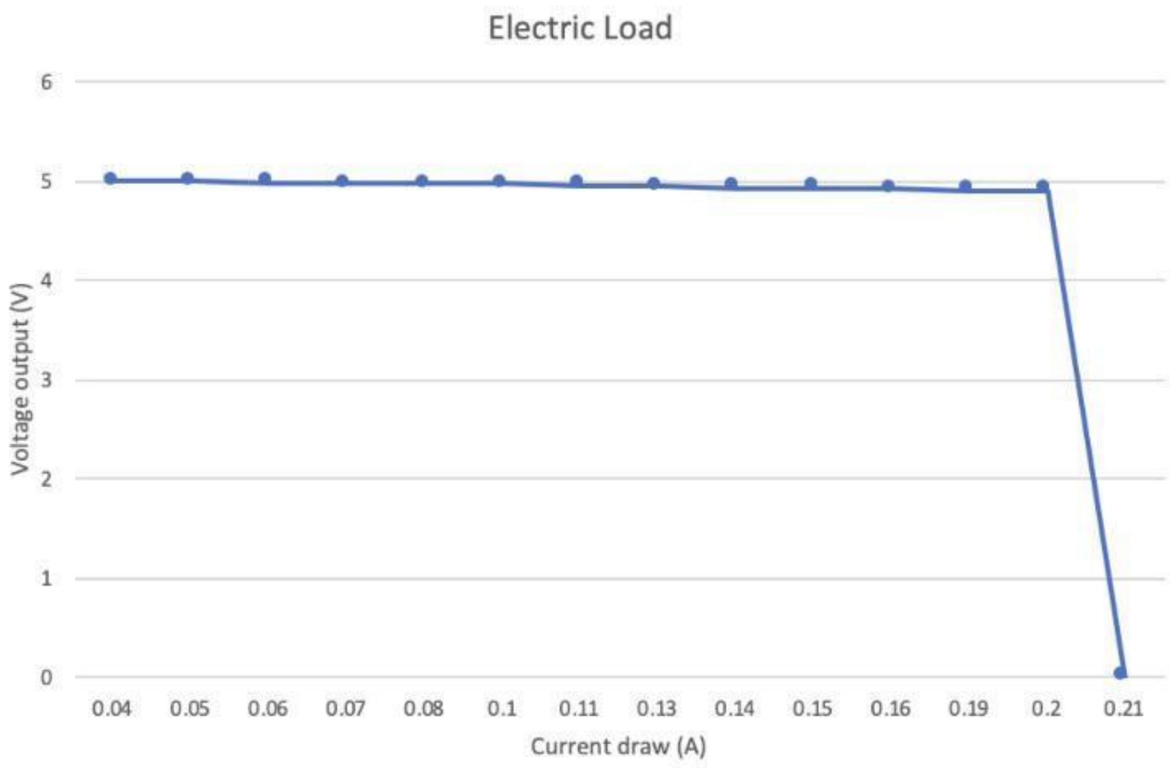


Figure 4.4.4: Voltage output(V) vs. current draw(A) for 5V regulator

Specification 4: As seen above for specification 3, the system is capable of providing 5V up to 200mA, and will meet the 50uA current draw required to power the temperature subsystem.

Specification 5: The below photo demonstrates that the system will continue to supply required 5V for the light sensor when the load current is 40 mA.

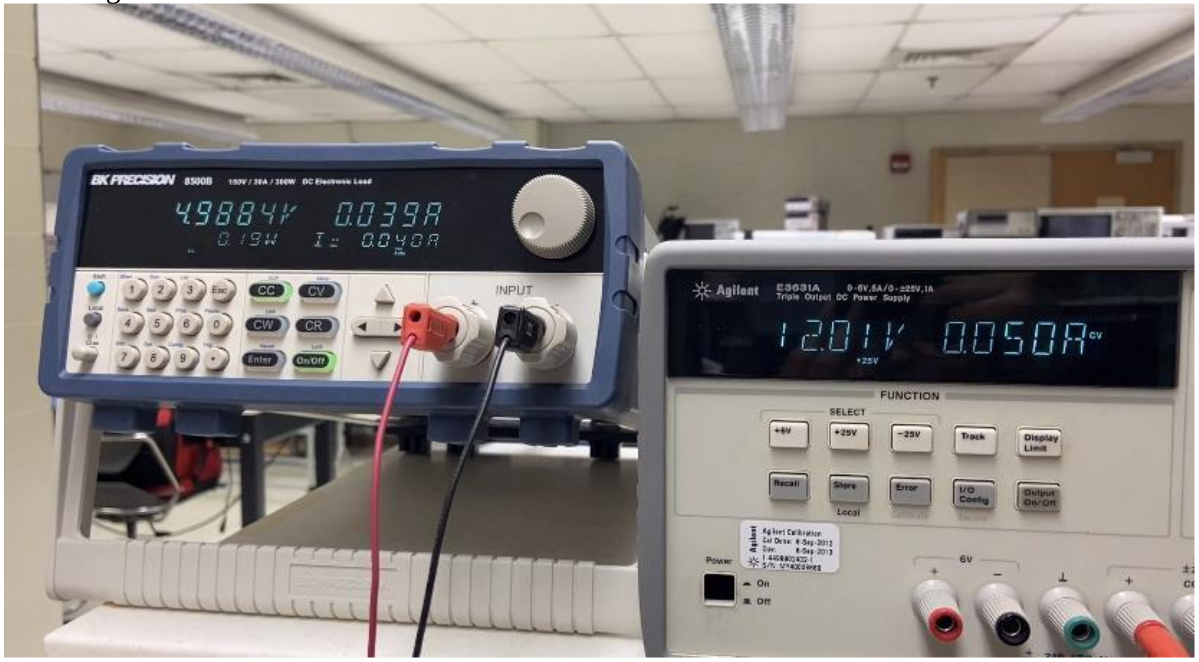


Figure 4.4.5: Simulated electronic load data for 5V linear regulator

4.4.8 Standards

American wall outlets provide a standard 120 VAC at 60Hz, according to the standards by the NEMA.

5. PCB Design

5.1 PCB Circuit Schematics

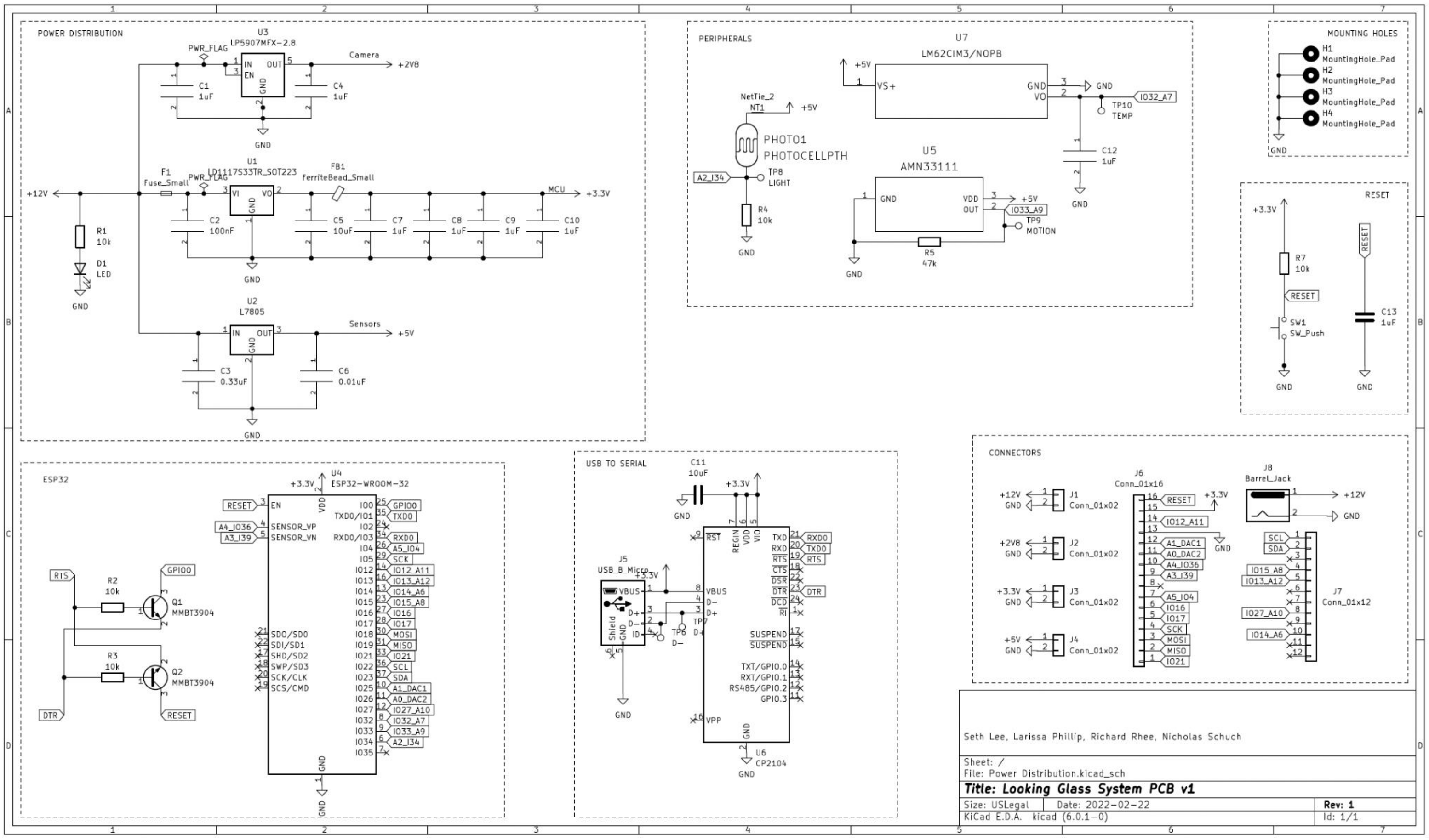

POWER DISTRIBUTION
PERIPHERALS
MOUNTING HOLES
RESET
ESP32
USB TO SERIAL
CONNECTORS
Seth Lee, Larissa Phillip, Richard Rhee, Nicholas Schuch
Sheet: /
File: Power Distribution.kicad_sch
Title: Looking Glass System PCB v1
Size: USLegal
Date: 2022-02-22
KiCad E.D.A. kicad (6.0.1-0)
Rev: 1
Id: 1/1

Figure 5.1.1: PCB System Circuit Schematic

5.2 PCB Physical Layout

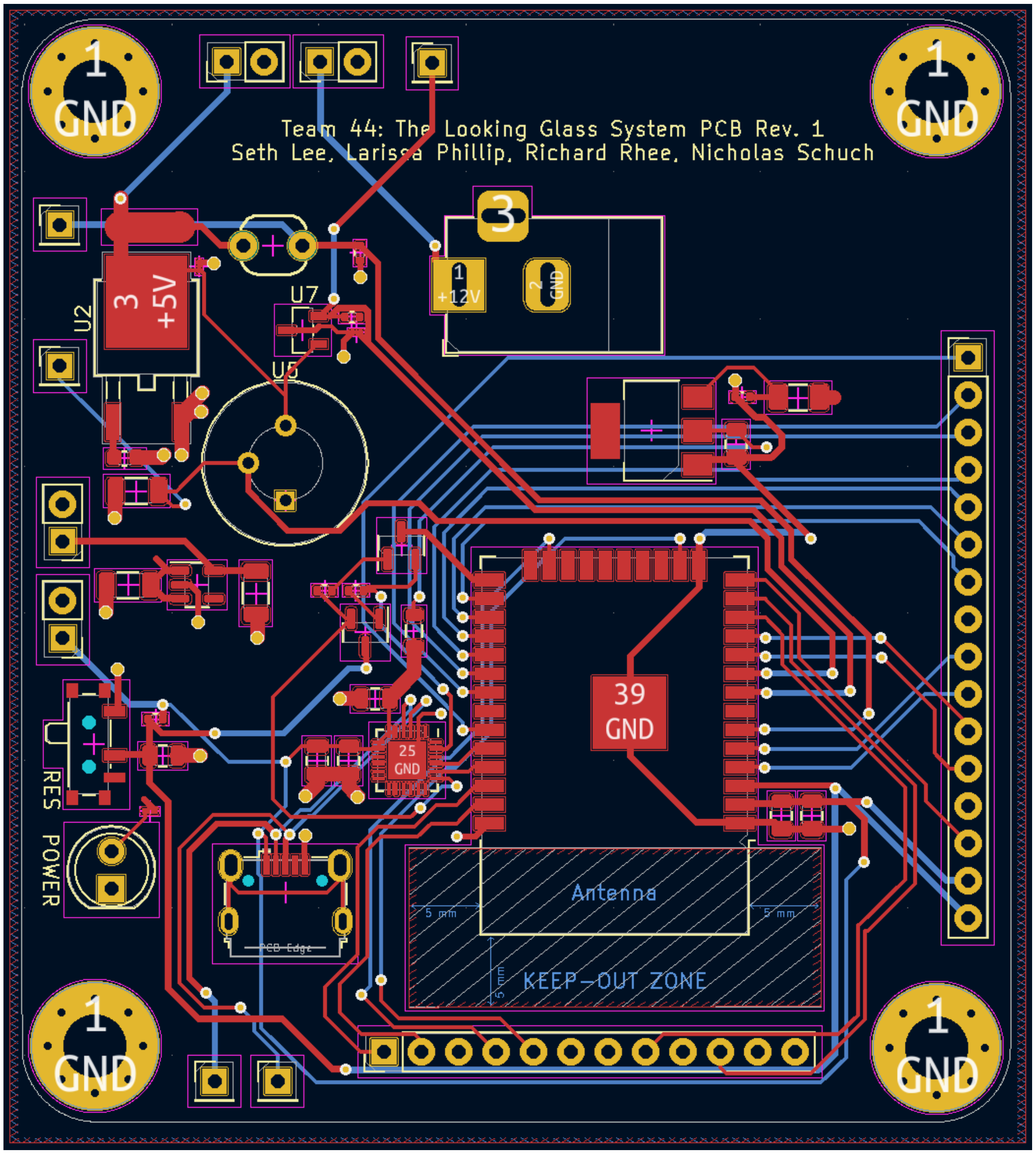

Team 44: The Looking Glass System PCB Rev. 1
Seth Lee, Larissa Phillip, Richard Rhee, Nicholas Schuch
1
GND
3
+5V
U2
U7
U5
1
+12V
2
GND
39
GND
25
GND
RES
POWER
Antenna
5 mm
5 mm
5 mm
KEEP—OUT ZONE
PCB Edge

Figure 5.2.1: PCB System Circuit Layout

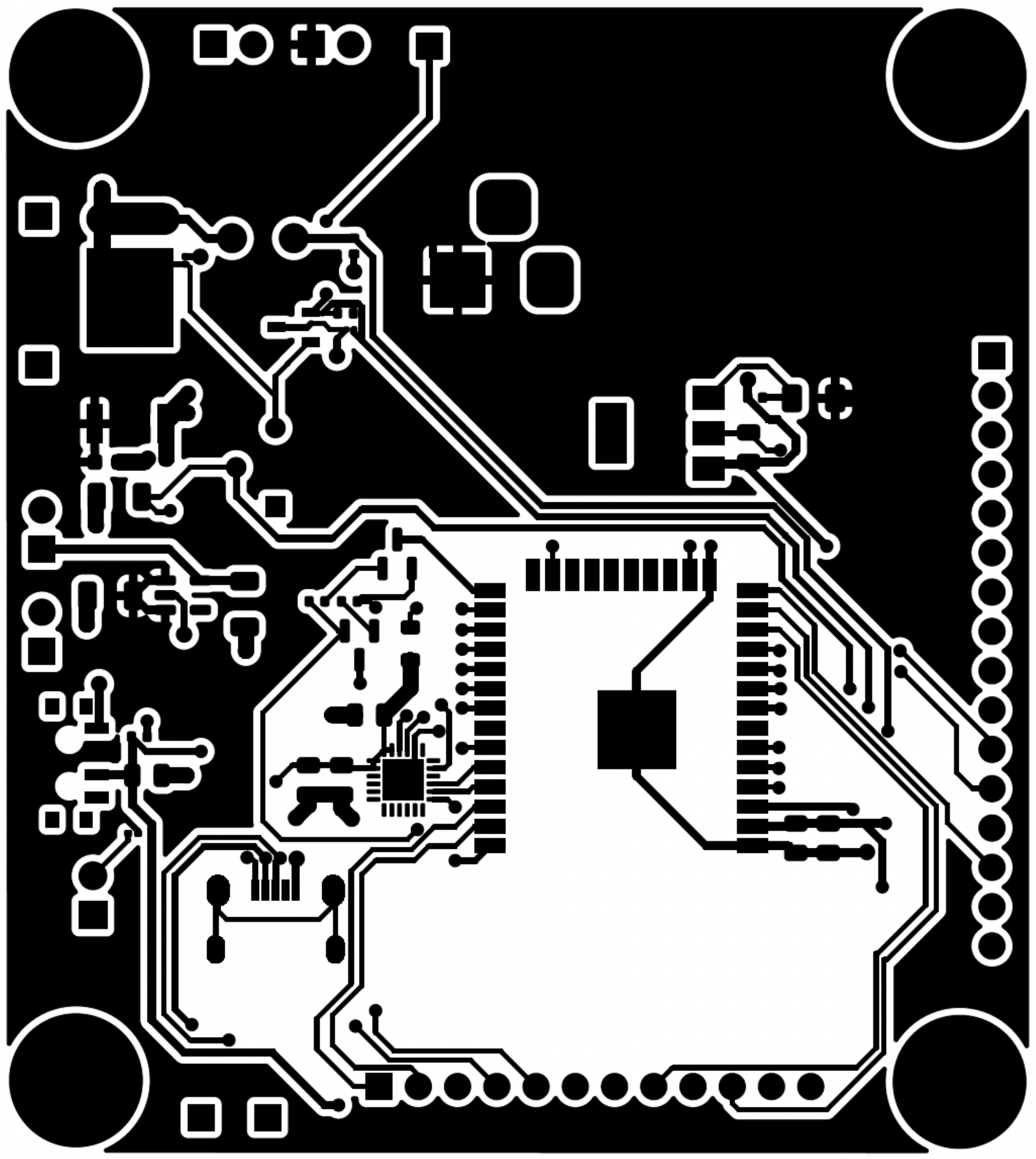


Figure 5.2.2: PCB Front Layer, +12V Plane

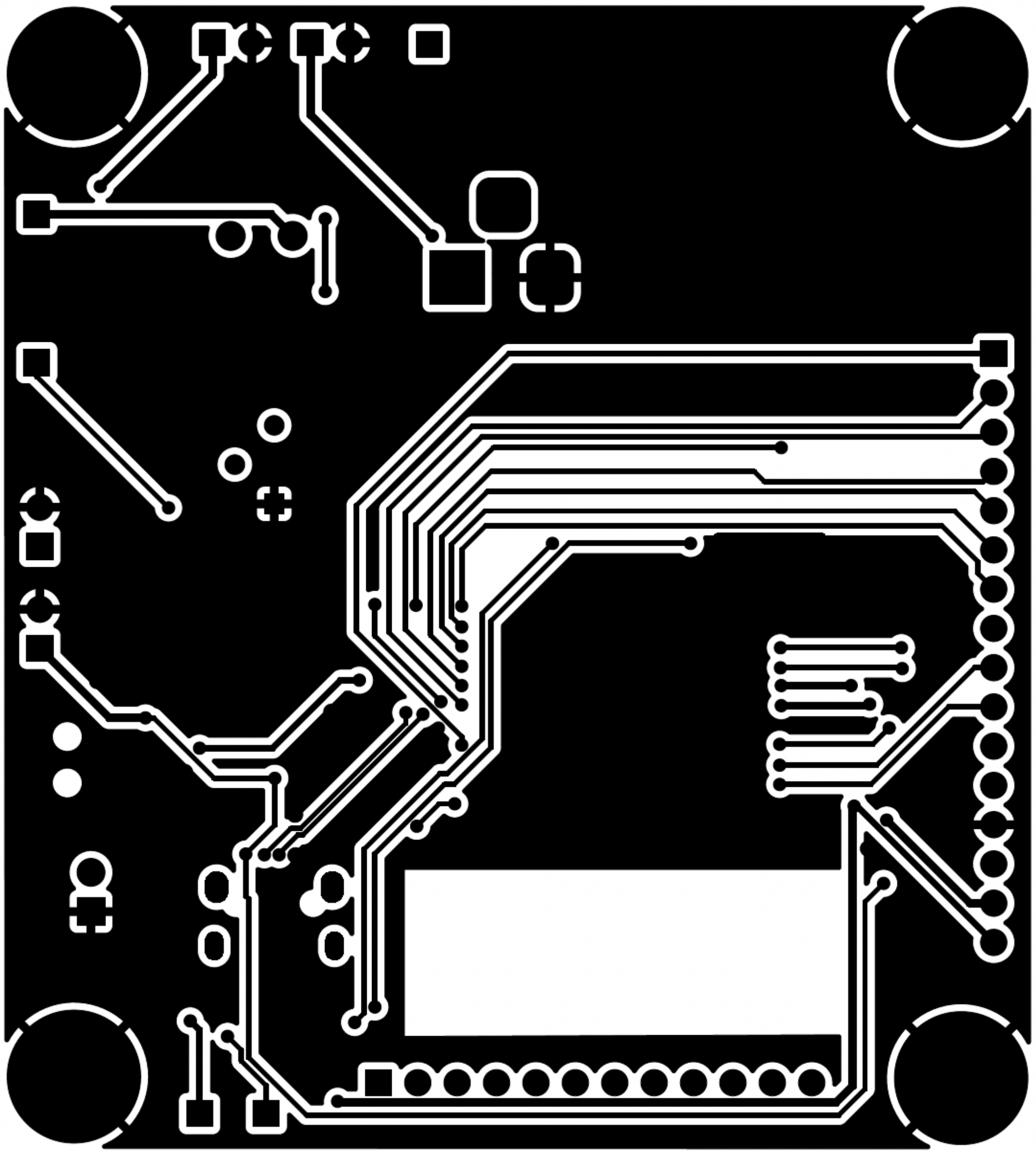


Figure 5.2.3: PCB Back Layer, Ground Plane

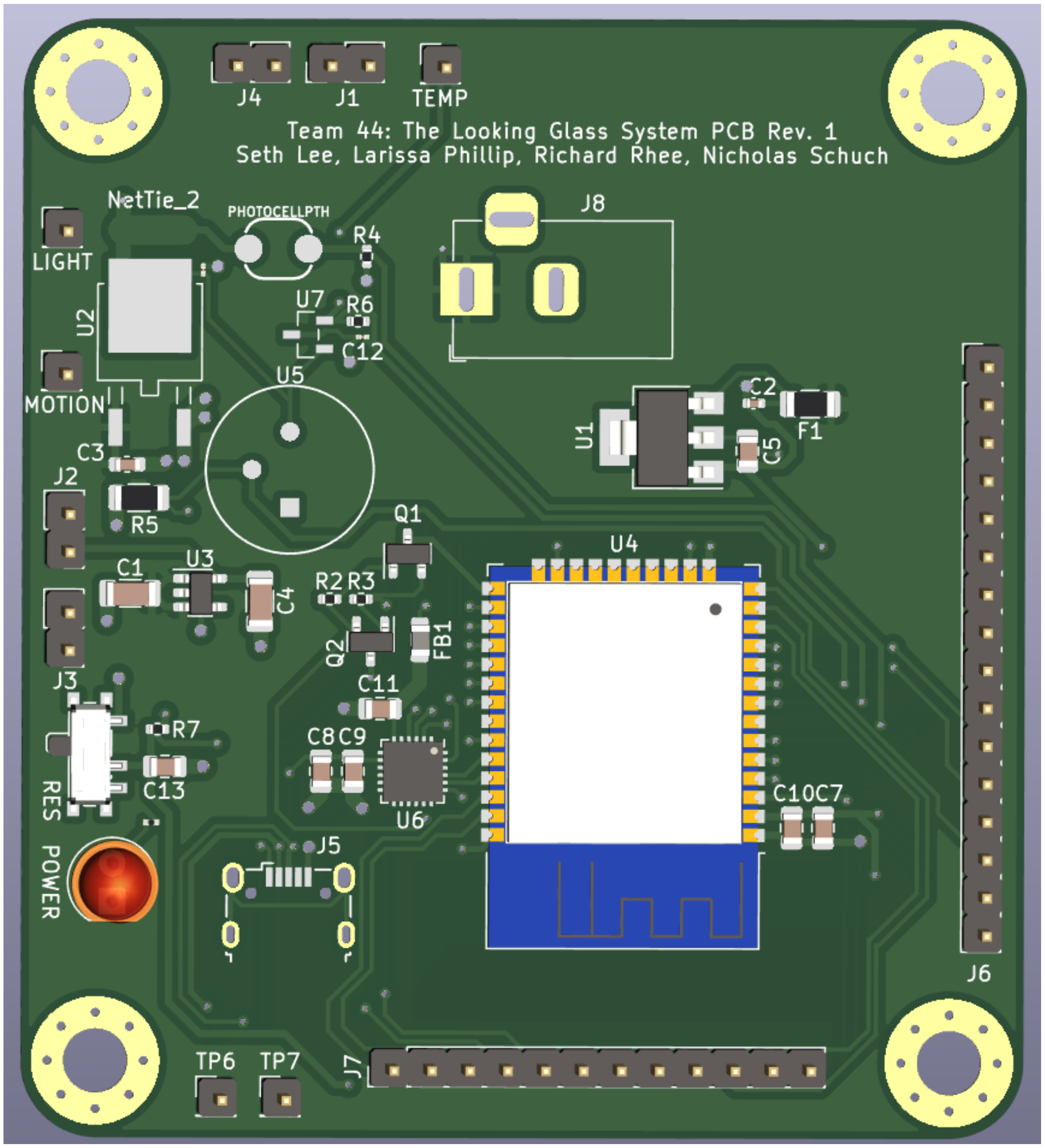


Figure 5.2.4: PCB Layout 3D View

6. Final Status of Requirements

Requirement 1. The system shall display the current date.

Met - The system displays the current date in MM/DD/YYYY format once activated with either motion sensor or pushbutton press.

Requirement 2. The system shall display the current time.

Met - The system displays the current time in HH/MM format once activated with either motion sensor or pushbutton press.

Requirement 3. The system shall display the current outside temperature.

Met - The system makes a HTTP GET request to OpenWeatherMap API, which returns a JSON with the current, daily minimum and maximum, and current weather condition, which is then parsed and displayed to the user.

Requirement 4. The system shall display the current inside temperature.

Partially Met - The temperature sensor was not implemented for use. We had to change sensors multiple times, and because of this, we ran out of time to collect adequate data to convert from the mV output displayed to a temperature.

Requirement 5. The system shall display the daily low and daily high temperatures.

Met - The system makes a HTTP GET request to OpenWeatherMap API, which returns a JSON with the current, daily minimum and maximum, and current weather condition, which is then parsed and displayed to the user.

Requirement 6. The system shall display the weather status for the day (e.g., sunny, rainy, partly cloudy, cloudy, etc.,).

Met - The system makes a HTTP GET request to OpenWeatherMap API, which returns a JSON with the current, daily minimum and maximum, and current weather condition, which is then parsed and displayed to the user.

Requirement 7. The system shall display a user-created to-do list.

Not Met - BLE 4.0 has a transmit limit of 20 bytes, which is far too little for communicating a to-do list.

Requirement 8. The system shall display current (daily) news articles.

Partially Met - The system was able to make a GET request and receive JSON of current relevant news, with date, author, title, and a short summary. However, we did not have enough time to implement parsing this data and sending this information to the display.

Requirement 9. The system shall display current notifications (e.g., phone notifications).

Not Met - Like requirement 7, there was a hardware limitation. We also did not have time to implement this functionality.

Requirement 10. The system shall detect motion to activate.

Met - Explanation

Requirement 11. The system shall have an additional means of activation that is not reliant on motion detection (e.g., push button or touch sensor).

Met - Explanation

Requirement 12. The system shall be powered by a standard outlet (120V, USA.).

Met - The system receives power from the standard wall outlet via a AC-DC wall adapter that converts 120V AC to 12V DC.

Requirement 13. The system shall have a low-power mode (e.g., sleep-mode).

Met - Explanation

Requirement 14. The system shall have a normal-operation, full-power mode.

Met - Explanation

Requirement 15. The system shall connect to the internet.

Met - The system is able to connect to the internet via 2.4 GHz Wi-Fi to perform various functionalities.

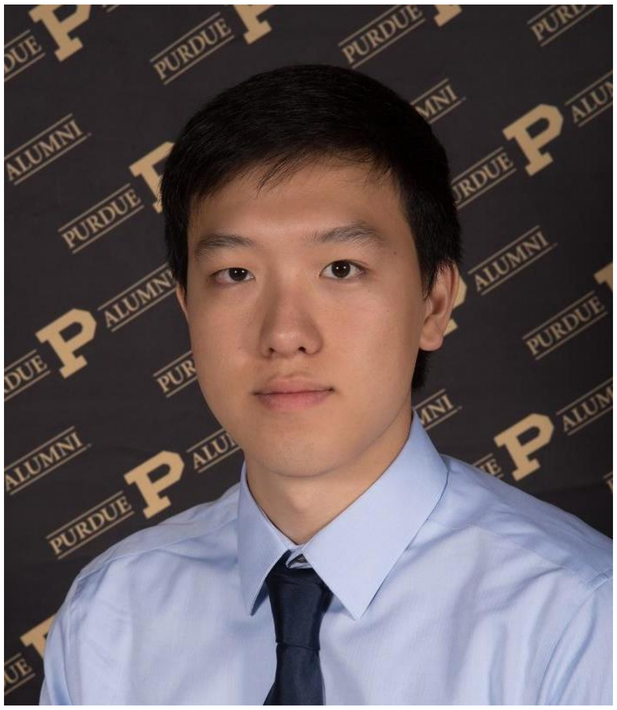
Requirement 16. The system shall retrieve information from the internet.

Met - The system is able to connect to the internet and retrieve information from the internet using HTTP/HTTPS GET requests to receive and parse JSON data.

7. Team Structure

Nicholas S. Schuch
Major: Computer Engineering
Contact: schuch@purdue.edu
Skills: Python, C, LabView, MATLAB, Autodesk AutoCAD, Autodesk Inventor, woodworking
Previous Projects: ECE362 Hydroponic system prototype w/STM32, various labs using ESP32 in ECE40862, 5-session co-op with Fiat Chrysler Automobiles, FCA, (Odometer software group – Model Harnesser in Simulink, Winter Testing on ESC module, Voice Recognition w/Indian speakers in Uconnect group, LabView SIL testing with BCM group, Scrum Development with ADAS group)

I am from Kent, Ohio (about 45 minutes south of Cleveland). I spent 5 rotations as a co-op at Fiat Chrysler Automobiles (FCA) in Auburn Hills, Michigan. I worked in 5 different groups in their electrical engineering department: Software Odometers, ESC testing, Voice Recognition in the Uconnect radios, LabView Software-in-the-loop testing with the Body Control Module, and scrum development in Advanced Driver Assistance Systems. I have always particularly enjoyed my projects at Purdue that utilized microcontrollers and embedded systems.



Richard Rhee
Major: Computer Engineering
Contact: rheer@purdue.edu
Skills: C, C++, Python, Java, Assembly
Previous Projects: Rogue in Java, compiler for MicroC, tic-tac-toe on STM32, software tools for TensorFlow research

I am from Bloomington, Indiana, but I have lived in Korea for 11 years. I did one semester of research with the Cam2 TensorFlow team at Purdue and played in the Purdue jazz bands for two years. My interests include consumer products, game development, and user interface design. Some hobbies I have include playing video games and rock climbing.

Larissa Phillip
Major: Electrical Engineering
Contact: phill236@purdue.edu
Skills: C, C++, Java, Python, MATLAB, Verilog, Circuit Design
Previous Projects: Lightning Protection, Ground Grid Analysis, Protection & Controls, EPICS (ISBVI Magnifier, ISBVI LEAP, Zoo Bird Exhibit)

I am from Austin, Texas—the live music capital of the world. Thus far in my career, I have spent 4 rotations working for Duke Energy as a Project Engineer in various departments, including the Substation, Protection & Controls, and Transmission Engineering groups. While at Purdue, I have devoted several semesters to different student-led projects through the EPICS program. Outside of my studies, I enjoy traveling and adventuring, mainly hiking. I spent the Fall 2021 semester abroad at Yonsei University in Seoul, South Korea, and otherwise found time to travel with my family during holidays.

Seth Lee
Major: Computer Engineering
Contact: lee3023@purdue.edu
Skills: C, Python, SQL,
Previous Projects: Data Mine Corporate Partners (Sandia AESOP, TMap), ECE 362 wav player on STM32

I was born and raised in Bloomington, IN, where IU is located. I have been part of the Data Mine Corporate Partnerships program for the past 3 semesters, working with companies while learning about Data mining and science. I currently have an interest in embedded systems and computer security. Besides academics, I have entertained myself with chess and music, previously playing piano and violin. Here at Purdue, I am part of the Cornerstone Christian Fellowship and Purdue bells.

Bibliography

https://en.wikipedia.org/wiki/Wi-Fi
https://en.wikipedia.org/wiki/Bluetooth
https://en.wikipedia.org/wiki/Bluetooth_Low_Energy