

IoT Plant Sensor

FINAL REPORT

sddec-08

Client: Mark Easley

Advisers: Daji Qiao

Zachary Kauffman - Firmware/Backend

Daniel Phalen - Hardware

Mason Gil - Frontend/Backend

Walter Gilbert - Frontend

Thomas Smeed - Hardware

sddec21-08@iastate.edu

<https://sddec21-08.sd.ece.iastate.edu>

Revised: 12/8/2021 / Version 2

Executive Summary

Development Standards & Practices Used

Agile Development

Javascript Coding Standards

Git Conventions

2 - layer PCB

Continuous Testing

Summary of Requirements

- Design a PCB to allow the user to connect multiple analog sensors, like moisture, light, and temperature, while reducing the GPIO ports of the microcontroller
- Given the sensor adapter PCB and some basic instructions, a non-technical user should be able to get the sensor working and begin monitoring their garden/nursery
- The front-end interface needs to keep track of the value of the sensors and report if any level is above/below the acceptable level, reporting within 1 hour.
- The front-end interface needs to have available settings for different types of plants. Example: If a user has a garden with 3 plant types, they should be able to set different sensor levels for each
- The hardware should run for long periods of time (a few weeks) with minimal user interaction.
- The sensor module should be able to monitor moisture, temperature, and pH, all within 5% accuracy and < 5-minute delay.

Applicable Courses from Iowa State University Curriculum

CprE 288

CprE 488

ComS 309

SE 319

EE 201

EE 230

New Skills/Knowledge acquired that was not taught in courses

AWS

Project definition

Leadership skills

Table of Contents

1 Introduction	4
1.1 Acknowledgment	4
1.2 Problem and Project Statement	4
1.3 Operational Environment	5
1.4 Requirements	5
1.5 Intended Users and Uses	5
1.6 Assumptions and Limitations	6
1.7 Expected End Product and Deliverables	6
2 Project Plan	6
2.1 Task Decomposition	6
2.2 Risks And Risk Management/Mitigation	6
2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria	7
2.4 Project Timeline/Schedule	8
2.5 Personnel Effort Requirements	9
2.6 Other Resource Requirements	9
2.7 Financial Requirements	9
3 Design	9
3.1 Previous Work And Literature	9
3.2 Design Thinking	10
3.3 Proposed Design	10
3.4 Technology Considerations	11
3.5 Design Analysis	11
3.6 Development Process	11
3.7 Security Concerns	11
3.8 Design Plan	13
3.9 Design Changes From CprE 491	13
4 Testing	14
4.1 Unit Testing	14

4.2 Interface Testing	14
4.3 Acceptance Testing	14
4.4 Results	15
5 Implementation	15
5.1 Frontend	15
5.2 Backend	15
5.3 Hardware	16
5.4 Firmware	16
6 Closing Material	16
6.1 Conclusion	16
6.2 References	18
6.3 Appendices	19

1. Introduction

1.1 ACKNOWLEDGMENT

We would like to thank Texas Instruments and Mark Easley for sponsoring this project. Without their contributions, we would not have the necessary resources to complete this project.

We would also like to thank Dr. Daji Qiao for his technical advice and guidance throughout the project.

1.2 PROBLEM AND PROJECT STATEMENT

Gardening, farming, or even caring for a houseplant, takes care and time. As life gets busy, people (especially those new to growing plants) are especially prone to forget to water and care for their plants, causing them to die. Even professionals like those who run nurseries need to expend a lot of time and manpower to keep their plants healthy.

IoT sensors can make this easier for both groups. By planting sensor modules along with plants, users can take the mental work out of gardening, and let algorithms guide their actions. Rather than guess if their plant needs watering again, they can log onto the provided website and examine the plant's water consumption and current supply. If the days are getting longer and sunlight levels are changing, they can view sunlight charts and see if the plant should be moved somewhere else to get more/less sunlight.

The goal of our project is to develop an easy-to-use IoT sensor ecosystem to be used in conjunction with growing and monitoring plant life. The final project will consist of a sensor module that will have attached sensors for light, fertilizer, and moisture, which will connect to a microcontroller that will send the data to an AWS server. Users will be able to go on this website and view the status and relative position of their sensors, as well as generate graphs of each sensor's value over time. Recommendations on repositioning (for better light), watering, and re-fertilizing the soil will be generated based on this data. For home gardeners who likely don't tend to their gardens every day, they will be able to receive reminders when the moisture sensor detects less moisture than their plant requires.

Once completed, this project will be a massive boon for farmers and gardeners, both hobbyists and professionals alike.

1.3 OPERATIONAL ENVIRONMENT

The end product will be exposed to the elements, especially water. High temperatures and rain will likely be a frequent occurrence where the sensors are positioned. While these sensors are not expected to be positioned outdoors during the winter and offseason, they will still likely experience some cold temperatures, especially when used incorrectly and left for longer periods of time than intended.

1.4 REQUIREMENTS

Functional:

- The front-end interface needs to have available settings for different types of plants. Example: If a user has a garden with 3 plant types, they should be able to set different sensor levels for each
- Sensor status visible on front-end
 - Text format
 - Graph format
- Visual indicator on the sensor module of if the sensors are in the range
- Users can add multiple sensor modules
- Users can add up to 8 sensors per sensor module

Non-functional:

- The sensor module should be able to monitor light, moisture, temperature, and pH, all within 5% accuracy and < 5-minute delay.
- If a sensor reading goes out of bounds, a user should get an email within 5 minutes stating that fact.
- The hardware can run for upwards of a week without user interaction

1.5 INTENDED USERS AND USES

Our intended users are all who wish to grow plants and need some help in that process. This group can be broken down into a few subcategories.

- New to gardening. These users will likely not have large gardens to tend to and will rely more on the reminder and recommendation features (when they should water next, how much, etc), using only a single sensor module.
- Experienced gardeners with larger setups. These people know how to care for plants, but as they expand their gardens/hydroponics setup, they require a bit more help keeping everything straight. These users will likely have a few modules, so they need to be able to view all the data in the cloud and have an easy way to make sense of it. They will likely not need the recommendations features as much, but the reminders will still likely be used.
- Professionals. Nursery businesses or farmers who have lots of experience, lots of plants, and lots of sensors. These users have a very clear understanding of what they need to do and will likely be using the sensors as more of an error detection system, helping them know if there's a part of their field that needs extra attention. This group will not be our core focus.

1.6 ASSUMPTIONS AND LIMITATIONS

Assumptions: users will have a network connection available, users will have access to electricity/batteries, users will be able to procure the water and nutrients they need for their plants, maximum registered users will be 10

Limitations: we will be unable to test large-scale setups and thus the project will be focused on houseplants and ordinary garden plants.

1.7 EXPECTED END PRODUCT AND DELIVERABLES

Front-end interface

This will be built with React and connect to our AWS backend. It will display current and past sensor readings, recommendations and other useful information. This will be used by the user to monitor their sensors and plants. This should be ready by the start of the fall semester, with notable progress made by the end of the spring semester. Unit testing will be done as development proceeds.

AWS backend

Our microcontroller will stream the data over the network to our AWS backend, which will collect the sensor data and process it. The backend will store sensor data so that history graphs can be generated. It will also store user preferences and thresholds for alerts on water/light/nutrient needs. The backend timeline is similar to the frontend timeline and we expect to develop them at the same time. This should be ready at the start of the fall semester, with notable progress and preliminary testing done by the end of the spring semester.

Sensor adapter daughterboard

This device will allow the user to use common analog sensors, such as moisture, humidity and light sensors, that can be bought off the shelf, and will reduce the amount of GPIO ports in use on the microcontroller. The board will split the analog input to the microcontroller into 8 possible inputs. The daughter board will allow for cycling between the 8 sensors. Since the sensors have different pinouts, there will be a method for the user to switch between data, power and ground.

2 Project Plan

2.1 TASK DECOMPOSITION

Core tasks:

- Development of hardware PCB that connects sensors and stream data to a microcontroller
- Design of a front-end interface with AWS which will take the data provided by the aforementioned microcontroller, parse and store it, and present it to the user
- Designing a core firmware loop to bridge the hardware sensor readings to AWS and, by extension, our front-end view.

Integration tasks:

- Integration of the microcontroller to the sensor module
- Integration of the microcontroller to the cloud service
- Integration of the cloud service to our front-end application

2.2 RISKS AND RISK MANAGEMENT/MITIGATION.

Sensor and breadboard testing - Risk score: 0.5

- Since we are dealing with electrical component, there is a potential for shock
- testing the moisture sensor with water, makes the situation a bit more risk, since there is a possibility of it spilling

Soldering the PCB - Risk Score: 0.8

- using a soldering iron has the potential to burn the user
- fumes from the flux and solder pose a hazard to the user's health

PCB Testing with sensors - Risk Score: 0.2 (lower than breadboard since less wires are exposed)

- Since we are dealing with electrical components, there is a potential for shock
- testing the moisture sensor with water, makes the situation a bit more risk, since there is a possibility of it spilling

Database loss of user data - Risk Score: 0.2

- It's imperative that users can rely on their data being stored on the application.
- By using AWS we can reasonably count on this never happening.

2.3 PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

Our key milestones were

- Hardware prototype on the breadboard
- PCB design
- Front end React app
- Backend AWS server
- PCB integrated with sensors
- Firmware interacting with PCB
- Firmware interacting with AWS
- Software can interact with the sensor module/controls

Our evaluation criteria for this project included the following

- Ease with which users can sign up and log in to the website
- Reliability and speed of sensor updates to front-end
- Functionality provided by the history and recommendation features
- Reliability of the connection between the sensor module and AWS
- Accuracy of the sensor data values

2.4 PROJECT TIMELINE/SCHEDULE

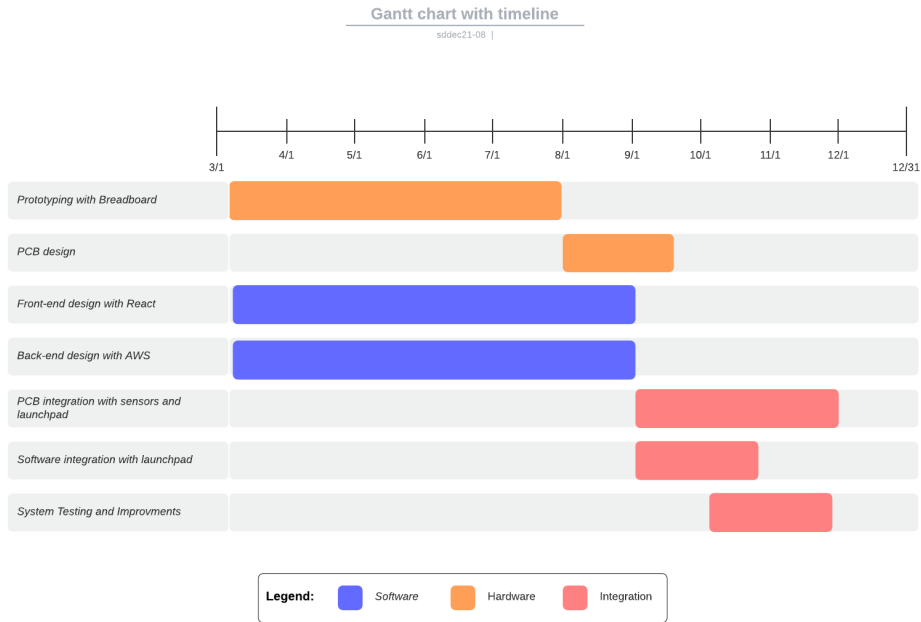


Figure 1

2.5 PERSONNEL EFFORT REQUIREMENTS

Hardware prototype on the breadboard	20 hrs	Breadboarding allowed us to be able to test the mux logic with the daughterboard before it arrived
PCB design	50 hrs	Had gone through a few redesigns due to flaws with parts and footprints used in the PCB
React app	60 hrs	Getting the proper libraries working took a generous amount of time, most of our time was spent cleaning up and debugging code after hastily hashing together user interfaces
AWS server	20 hrs	Was fairly simple, however, the complexity of AWS UI took a significant chunk of time to decipher.
PCB integration with sensors	5 hrs	Since we had the design breadboarded, it expedited the integration.

Software/hardware interaction	60 hrs	This was initially done using the breadboard due to the delays in shipping of the PCB. Took a bit of researching and trial and error to get the TI launchpad to recognize the sensors
Firmware control loop	40 hrs	We designed firmware logic for the TI launchpad to get the user's preferences, read the sensors, and send the sensor readings off to AWS.

Figure 2

2.6 OTHER RESOURCE REQUIREMENTS

AWS accounts and credentials

CAD software for PCB design

2.7 FINANCIAL REQUIREMENTS

Ordering the PCB and its components is where most of our funds went, since there had been 3 ordered PCB designs and 2 different sets of components, for the revision.

Sensors were also ordered for testing purposes, from Amazon and Adafruit. We made sure to use the sensors that made the best sense for our application and also based on the cost since we want to make sure that this project caters to our variety of demographics.

The AWS free tier should suffice for our initial implementation.

3 Design

3.1 PREVIOUS WORK AND LITERATURE

digiPlant[1] is the most notable result that will pop up if you start looking for other IoT plant sensors. This project is developed by Microsoft and is primarily focused on teaching the process of IoT, making it pretty simple but effective. It can integrate with Twitter and tweet out the status, which is a handy feature. Outside of that, there aren't many ways to monitor the plant's status remotely, and Twitter as your main remote monitoring mechanism doesn't provide all the information we would like to provide with our solution. [2]

There are many solutions out there similar to digiPlant. Focused on hobbyists, these guides help users set up their own IoT solutions, but are not very user-friendly. They only support one or two sensor nodes and require lots of knowledge on the part of the user. Our solution aims to be much more approachable.

Another commercial product on the market is the WANFEI Plant Monitor [3]. This is a sensor node with light, temperature, humidity, nutrient sensors, and Bluetooth capability. It's marketed as being easy to use, but the Bluetooth connection only lets users monitor when they are right there with their plants. Our solution will be accessible from anywhere and will support multiple sensor nodes.

3.2 DESIGN THINKING

Our project was presented to us relatively open-ended. Thus, most of the first few weeks of the semester were very much focused on defining the project. One of the first aspects of the design that was defined was “smart home”. It was clear from the start of the project that we wanted to focus our sensor use on the home and make people’s lives easier. Here we explored a few different ideas:

- Smart thermostat system with temperature sensors in every room and an app that let you see and control the temperature of each room
- Hydroponics vertical farming system with moisture, temperature, light sensors
- Pandemic focused biometrics sensors that would let companies track the locations and temperatures of employees in their building for the early phase transitioning back to the office

The next aspect that was defined was “plant growth”. During brainstorming, we had discovered hydroponics setups and that lined up nicely with the smart home focus. By focusing on residential agriculture, we make it easier for people to participate in the very therapeutic act of gardening in their own homes.

3.3 PROPOSED DESIGN

Proposed hardware design:

- PCB daughterboard for a TI Launchpad which has ports for several sensors
 - Since the board will have the capability to support multiple sensors, we will be able to use one board that will monitor all of light, moisture, temperature, and pH.
- Enclosure for the board with sensors exposed so that they can be placed while treating the daughterboard as a black box
 - Enclosing the board in a box with only sensors exposed may help some consumers be less confused about the setup process since they will only have to worry about the sensors and not the board itself.
- Launchpad will stream data to the AWS server
 - Using the AWS server will be our method of having the hardware report any imbalances in the monitored levels to the user(s) as well as allowing the user to check the levels at any time from any capable device.
 - The Launchpad will get preferences from AWS on boot, telling it which sensors it has connected to it, and on which port.
- LED which will get sent data from the server and display status
 - Receiving data from AWS, the TI launchpad will display some of the information to an led that is close to the system. This will allow the user to view the information without having to access the application.

Proposed software design:

- AWS backend will receive data from the microcontroller
 - Using the AWS server will be our method of having the hardware report any imbalances in the monitored levels to the user(s) as well as allowing the user to check the levels at any time from any capable device.
- React app will take the data from the backend and generate sensor graphs for the history of sensor values
 - This will allow the user to check the history and trends of the sensors in case there is a problem with the monitored values.
- React app will generate alerts when sensor values drop below a certain threshold
 - The user will be sent a notification from the react app that the plant(s) is/are low on water, not enough light, or other issues. This will allow the user to make sure that their plant(s) is/are staying alive.
- Data gets sent to status LED panel

- Receiving data from AWS, the TI launchpad will display some of the information to an LCD that is close to the system. This will allow the user to view the information without having to access the application as well as the battery level.

This design satisfies our requirements because it allows us to design an easy-to-use plant sensor module that lets users view their plant status from the cloud.

3.4 TECHNOLOGY CONSIDERATIONS

React: Slow but easy to build user-friendly apps on the web.

AWS: Easy to build IoT applications on, free to use for us. Powerful IoT tools and its IAM and Cognito offerings make it easy to allow users to make secure accounts and assign them the right permissions.

TI Launchpad: Powerful microcontroller with built-in WiFi features, making it a prime candidate for reading our sensors and sending them off to AWS.

Custom PCB: Be able to connect multiple sensors to the TI launchpad. Must be able to connect to commonly used analog sensors.

3.4 DESIGN ANALYSIS

The components of our software design have been successful. We have a clear framework built off of AWS connecting to both our frontend and Launchpad that works correctly for managing sensor data and user preferences. We have been able to deploy an Amplify app with AWS and use MQTT data to display real-time updates on the front-end, with graphing functionality pulling in older values from the database. This software skeleton we've designed uses AWS Cognito and lets users sign up, sign in, and gives them the proper permissions to view our AWS topics.

Additionally, we've been able to get real sensor data up to AWS via a Launchpad gateway that sets the mux select lines on our PCB and reads in the analog values, delivering them back up to AWS. This control loop has also been successfully implemented.

The PCB design went through a few iterations. The first design and schematic, which are in figures 13 and 14, were just tested using KiCad. This led to a revision, in figures 15- 17. We had ordered this PCB and the parts for it. Then after receiving the parts and realizing the issues with them, we had done a third revision with new components. We had ordered the PCB and the new parts. While we were still waiting for the PCBs, we were able to get parts so we could breadboard the new design and test the hardware and firmware integration. We had then received our PCBs, 2 and 3, right before Thanksgiving. Then we noticed a major flaw, the holes for pin headers were too small. This led to a PCB 3.5 since the only thing that really changed was the size of the holes for the pin headers. We are still waiting for this PCB to arrive.

3.6 DEVELOPMENT PROCESS

We used an Agile development approach because it encourages continuous improvement and evaluation. Since we've used Agile approaches in other courses and internships and prefer using it in practice to other development processes.

3.7 SECURITY CONCERNS

One of the main benefits of AWS was that it handles the authentication for our web application and hardware. AWS Amplify handles making accounts on the website, which are given proper access by applying for AWS IAM roles. They are only given read access to the sensor_data database and write access to the preferences database, which will only affect the data they own. The firmware project is built using AWS certificates, which ensures that it has proper access to the AWS IoT endpoint, which is secure from

other access attempts. The firmware also only has MQTT publish access, and our AWS Lambda function is responsible for checking the format of incoming messages, only writing to the database for valid data messages.

By relying on AWS this much, it is possible to have our web application suffer negatively should AWS itself be compromised.

3.8 DESIGN PLAN

Preliminary block diagram

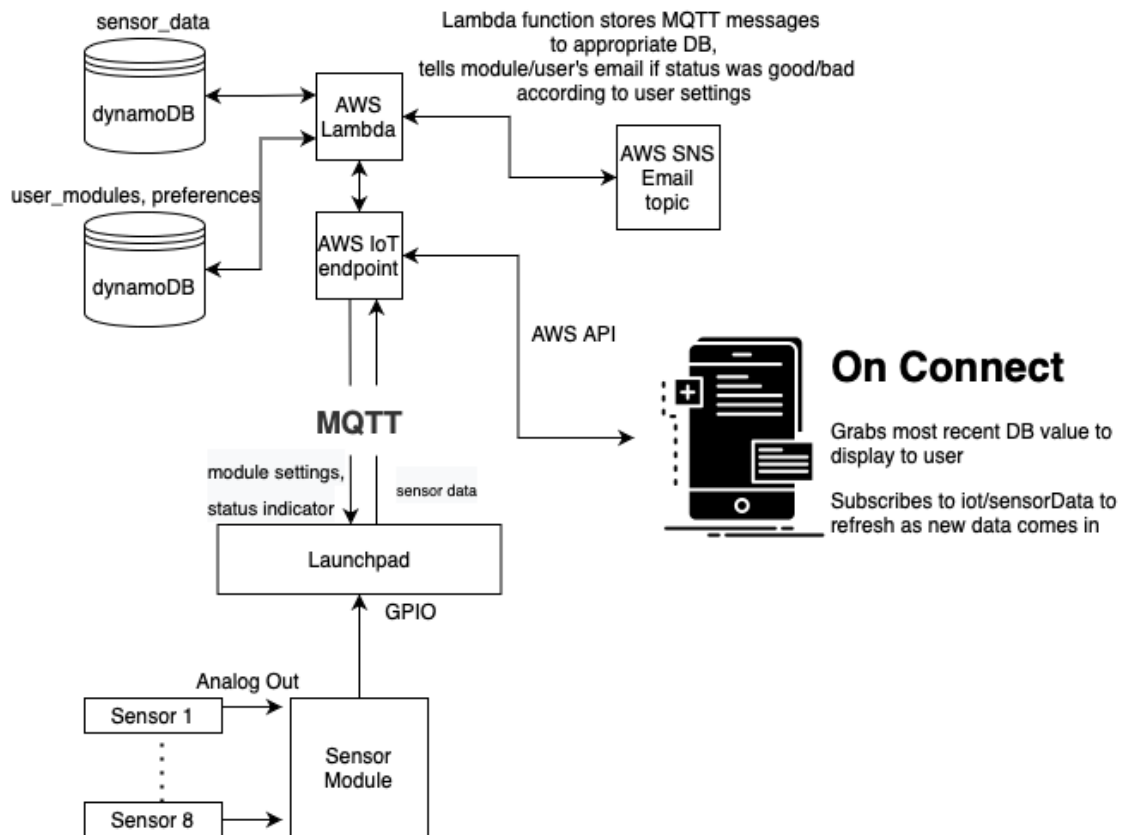


Figure 3

3.9 DESIGN CHANGES FROM CPrE 491

The core design of our project hasn't changed much from CPrE 491, although the core ideas have been solidified. The firmware part of our project took shape near the start of CPrE 492, where we changed direction from using a Raspberry Pi to using the TI Launchpad. Several aspects of the software design took shape this semester, such as using AWS SNS for email alerts, and how the firmware would handle knowing which sensors were attached to it at any given time. On the hardware side, our group moved away from making our own sensor and into making a daughterboard that will make it easier to connect off-the-shelf sensors to the TI launchpad while reducing the number of ports being used.

4 Testing

4.1 UNIT TESTING

AWS server: Building API test scripts where the AWS server's endpoints are called and expected to send back the correct data.

React Front end: The React app was tested by rigorously trying to break the user interface and writing a small number of tests for the small number of states/internal logic the app will hold.

Custom daughterboard: Using KiCad to make schematics and PCBs. We used this to figure out what parts to order for breadboarding the PCB before it arrives.

Microcontroller: We tested with the sensors to be sure that the microcontroller can read out the proper information from the GPIO ports.

Sensors: The sensors were calibrated, like for the pH or moisture, to make sure that they are working properly.

We had tested to make sure that sensors were being detected by the launchpad and in the code. After finding the value in the code, we were then able to find the value range for the analog output.

Hardware sensor modules were tested in isolation. They were tested in isolation to ensure that the proper sensor values were being taken from the sensors and outputted together to the microcontroller.

The backend was tested in isolation to ensure that, given fake data in lieu of the sensor module, it was eventually correctly read and processed to present to the frontend.

Similarly, the frontend was tested to ensure that it displays data correctly.

4.2 INTERFACE TESTING

Interface testing involved the intercommunication of the microcontroller with the sensor to the React Front end displaying the correct values for each sensor like moisture value. This was done through rigorous testing by having different controls, like a high moisture environment to a very dry environment.

Communication from the microcontroller with plant interfacing components like water pumps and lights was tested outside the system. This allowed us to set values for the flow rate of the water pump or the brightness of the lights.

The frontend/backend interface was tested to ensure that data from the backend is correctly displayed by the frontend and that no errors occur as data moves through the pipeline.

The hardware/software interface was tested to observe sensor values correctly read by the sensor module and displayed to the user via the React app.

4.3 ACCEPTANCE TESTING

We tried our best to demonstrate that the design requirements are being met in section 1.4, with the current global shipping issues and covid restraint made it a bit more of a challenge to get the hardware side completed; this issue mainly impacted the PCB. We were still able to make good progress on the software and firmware side since we were able to have breadboarded versions of the pcb.

4.4 RESULTS

Our project is largely successful, achieving both the functional and non-functional requirements, although the hardware side of our project would benefit from a bit more time. Our front-end interface accomplishes what it set out to do- users can correctly see the status of their sensors, both in text and graph form. They can modify the layout of their sensors and set the maximum values, and the firmware will pick up the preference the next time it boots up. The firmware reads in sensor values and is able to send them to the website correctly.

Our PCB faced some issues getting done on time. Due to shipping delays and Covid early in the semester, the first version of the PCB did not arrive until the second version had arrived, so we were not able to notice a flaw that persisted into the second version. We discovered a flaw in our second revision and ordered a third revision, but at this point, it was Thanksgiving break, which slowed down shipping enough that, as of December 7th, our team has not been able to work with it. The second revision, which we did test, was mostly functional, but the header layout was too small, so wires were soldered on directly, making it fragile for testing. Testing with a breadboard also validated that the core design worked properly.

5 Implementation

5.1 FRONTEND

Our frontend is built on the Javascript library React Native, along with a couple of other libraries and AWS Amplify to communicate with our AWS backend. React Native Paper is the UI component library we chose to use to build our application on as it has good documentation with numerous examples, and looks good to boot. For our chart library, we used React Native Chart kit as it had working examples to build off in its documentation; most other chart libraries were either barely documented or didn't work. The app consists of a sign-in page, sign-up page, view systems page, view charts/data page, and schedule alerts page. AWS Amplify helped us tremendously in communicating with the backend as it didn't require specific endpoints for manipulating data, and came with built-in user authentication and credential tools.

5.2 BACKEND

The backbone of our whole design is AWS, which provides many useful tools, such as AWS IoT, DynamoDB, Amplify, and Lambda.

AWS IoT is used for our MQTT communication between AWS and the Launchpad and generating the access credentials for the FreeRTOS project that the Launchpad runs. The endpoint for IoT core is used for all the core Launchpad communication - requesting preferences, receiving status updates, and sending back data. Through the IoT Core interface, we have several triggers set up that will forward the Launchpad's data to AWS Lambda (discussed later) and to DynamoDB for storing the sensor data. Additionally, the front-end is able to subscribe to this same MQTT topic and get the sensor data updates in real-time.

Lambda is used for processing data values as they come in. Whenever a new sensor data message is received, the Lambda function is invoked to check if the values are in/out of bounds with respect to schedules the user has set up. If a value comes in and a user has set a preference that they want to be alerted whenever a value comes in that drops below a threshold value, it will examine the conditions and send them an email if it's out of bounds. Additionally, every message triggers an acknowledgement from AWS to the Launchpad, telling it to set the led indicator red or green for bad/good status respectively.

AWS Amplify is the foundation of our website, and handles making user accounts and giving those accounts IAM access for reading/writing the appropriate DynamoDB databases.

There are three DynamoDB tables used in this project:

- User preferences (which sensors are with which modules, what the thresholds are)

- Sensor data (stores all the sensor values attached to a specific module at a specific timestamp)

- Schedule preferences (handles setting alert thresholds)

5.3 HARDWARE

The PCB daughter board is designed to hook up to 8 sensors to the TI launchpad, with 1 GPIO port being dedicated to the analog signal and three other ports for the cycling of the mux. The reason this is designed this way is so that the user can scale up the plants that are being monitored without having to add more TI Launchpads in the system because they will only need 1 GPIO port for every new daughterboard because the number of ports being used scales at a rate of $1(\text{GPIO Pins}) * (\# \text{ of daughterboards}) + 3$ (Logic Pins for cycling on the daughterboard) = # of GPIO Ports being used. The daughterboard also allows the user to plug-in sensors with three pins (power, ground, and analog out). The power and ground are taken care of by the USB- C power connector. The connection from the TI Launchpad to the daughterboard with the three logic pins allows the TI launchpad to cycle through the sensors. The PCB allows the user to adjust the pin header on the PCB from power, ground, and analog out, to the sensors. This helps ensure that the pins are plugged into the right part of the PCB.

5.4 FIRMWARE

The TI Launchpad used in our implementation runs FreeRTOS and a simple program written in C that handles connecting to the AWS IoT endpoint and subscribing to the appropriate “admin” MQTT topic (iot/<device name>/admin), based on the “Thing” name used in AWS to generate the certificates. Once connected, the Launchpad sends out an initial request to our AWS backend in order to determine which sensors it has at which port, as well as the maximum values for those sensors.

At this point, the Launchpad begins to execute a basic loop. It runs through every possible combination of select lines for the hardware mux (0-7), setting the select line and reading the analog input. Using the preferences it received from AWS, it converts the analog value into a sensor reading in line with the preferences and adds it to an outgoing MQTT message. The message then gets sent off to AWS IoT on the ‘iot/sensorData’ MQTT topic, which will trigger the Lambda function (and trigger a live update of the web app’s text view, which is subscribed to this topic). The Lambda function will check against all the user thresholds to see if any values are out of bounds, triggering an email if they are. The sensor values then get stored in the database for use and can be seen in the graphing view. Lastly, an acknowledgement is sent to the Launchpad, letting it know if the sensor values are good (board led should be green) or bad (led should be red).

6 Closing Material

6.1 CONCLUSION

With all of this in mind, our project was a success, based on the criteria presented by our functional and non-functional requirements. We were able to develop a solid cloud interface that is able to work with the

data from the sensors that we used. Our front end supports multiple accounts with any custom amount of systems and sensor interfaces. On the hardware side of the project, we were able to successfully design a custom daughterboard and improve it through multiple revisions.

For the PCB design, we had made a schematic of the board first with dip switches, a 16:2 mux, and pins. This design led to the first design of the PCB. The first PCB design went through a redesign after talking with Lee Harker and Mark Easley. We had remodeled the first PCB into our PCB 2 design, the main parts stayed the same, but had shrunk the size by a lot. Unfortunately, we were not able to breadboard this idea, since we were still waiting for the parts to arrive for testing. When we got the parts for the first PCB, which arrived later than expected, we were able to see a major flaw with the dip switch: the max current allowed through the switch. This led to a redesign, PCB 3, with triple pull, single throw switch, for the modular component. We were able to get the parts in rather fast, so we were able to breadboard the switches and mux, while we were waiting for the new PCB to arrive. The week right before Thanksgiving, we had finally received the PCB 2 and 3. When we received them, that is when we noticed the flaw with PCB; the holes for the pin headers were way too small. This led to a PCB 3.5 which was a correction to the pinholes. We had ordered this part, but due to shipping issues, it still has not arrived.

The hardware design and accompanying software are flexible enough to support any arbitrary analog sensor, with each user being able to have up to 8 sensors per sensor module, with as many sensor modules as they would like. The alerts system through AWS SNS is set up properly, and users can get emails when the relevant sensors are out of bounds of what they deem “safe” for the plant.

6.2 REFERENCES

- [1] ms-iot (2016). PlantSensor [source code] <https://github.com/ms-iot/PlantSensor>.
- [2] W. I. T. M. Nagase, “Plant App,” *Hackster.io*, 27-Dec-2017. [Online]. Available: <https://www.hackster.io/windowsiot/plant-app-1167ed#team>. [Accessed: 08-Mar-2021].
- [3]”WANFEI Plant Monitor Soil Test Kit Flower Care Soil Tester Smart Plant Tracker Intelligent Sensor Plants Detector Bluetooth Monitor for Light Moisture Fertility Temperature Level, for iOS and Android”, *Amazon.com*. [Online]. Available: https://www.amazon.com/WANFEI-Monitor-Flowers-Sensor-Moisture/dp/B07ZH7FQJ7/ref=sr_1_5?dchild=1&keywords=plant+sensor&qid=1618934778&sr=8-5. [Accessed: 21-Apr-2021]

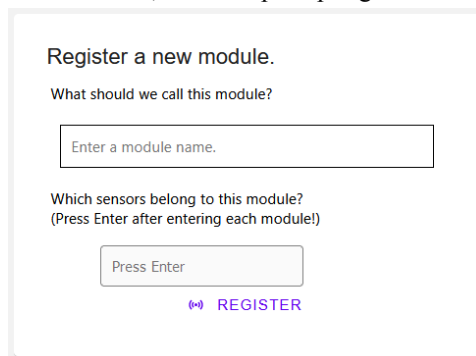
6.3 APPENDICES

Appendix I: User Manual

To begin using the sddec21-08 plant sensor system, first, one should get in touch with a member of the team (at sddec21-08@iastate.edu) to obtain the hardware with appropriate permissions to connect to AWS. One final step is required before launching the main control loop, which is to edit the appropriate configuration file (`demos/include/aws_clientcredential.h`) to tell the TI Launchpad the name of the SSID (line 63) it should connect to, and the appropriate password to use (line 69). At this point, launching the hardware loop will connect to the backend server.

In order to use the features provided by the web application, the user should go to the webpage at <https://main.d2mftyaz4wsvvq.amplifyapp.com> and follow the instructions to create an account. This is fairly simple: create a username/email, and then wait for a confirmation email with a code to come to the given email address. Once successful, the user will see a blank home page, with “user settings” and “sign out” options.

To register a new hardware module, go to the “User Settings” page. At this point, you will see a screen like the one below, which is prompting for a new sensor module name.



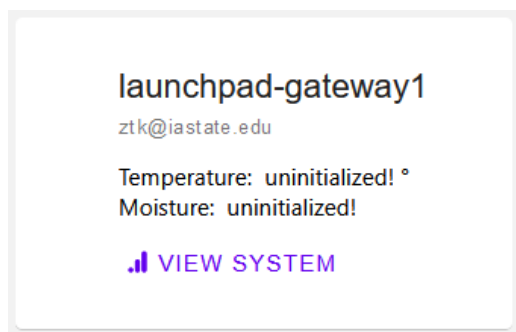
Register a new module.

What should we call this module?

Which sensors belong to this module?
(Press Enter after entering each module!)

[REGISTER](#)

Type in the name provided with the hardware you received, and enter any sensors that belong to it (moisture, temperature, etc). Then hit “Register” and refresh the page. Now the new home screen should include an entry for that sensor:



launchpad-gateway1
ztk@iastate.edu

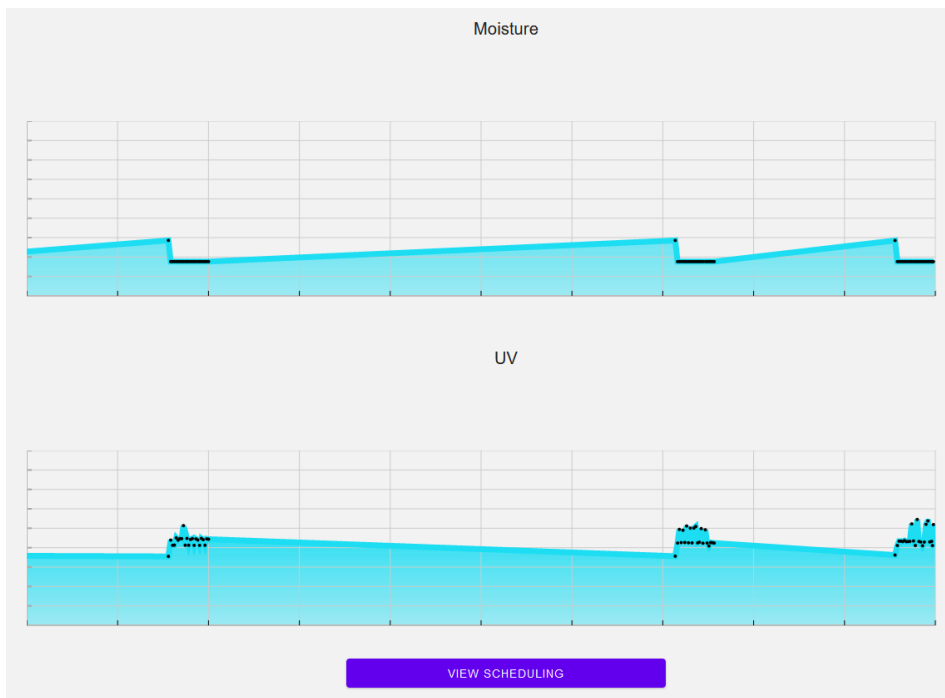
Temperature: uninitialized! °
Moisture: uninitialized!

[VIEW SYSTEM](#)

At this point, you can go back to the user settings page to add maximum sensor values or leave the default value of 100 intact (this will treat the sensor values as percentages). Launch the hardware loop using Code Composer Studio, and it will begin to run the main loop, using the data entered in the User Settings Page. Assuming not much time has passed (the web page will time out eventually), you will see the numbers on the home page begin to update, corresponding with the sensor values read by the Launchpad.

Graph View

To view the data coming in from the Launchpad in graph form, simply click the “View System” button at the bottom of each sensor module card on the launchpad. This opens up the graph page, which pulls all the corresponding data from the database for each sensor and shows how the values have changed over time. This is a great way to see where your plants are doing when it comes to moisture and other critical sensors.



Setting Alerts

To set alerts, advance one screen further by hitting the “View Scheduling” button at the bottom of the graph view page. This allows you to make thresholds for various sensors, triggering emails when the sensor value

passes above/below the threshold.

Schedule Title
Moisture Levels too high

Notify over Text Message

Notify over Email

Sensor Type
Moisture

Set Threshold
6

Notify when the sensor reaches over, or under the threshold?
Over Under

Notify by date:

[SUBMIT SCHEDULE](#)

Appendix II: Additional Figures

Software Figures

Sign in to your account

LOGIN

[Don't have an account? Sign Up](#)

Figure 4: Sign In Page

Register a new module.

What should we call this module?

Which sensors belong to this module?
(Press Enter after entering each module!)

REGISTER

Figure 5: User Settings Page

Home

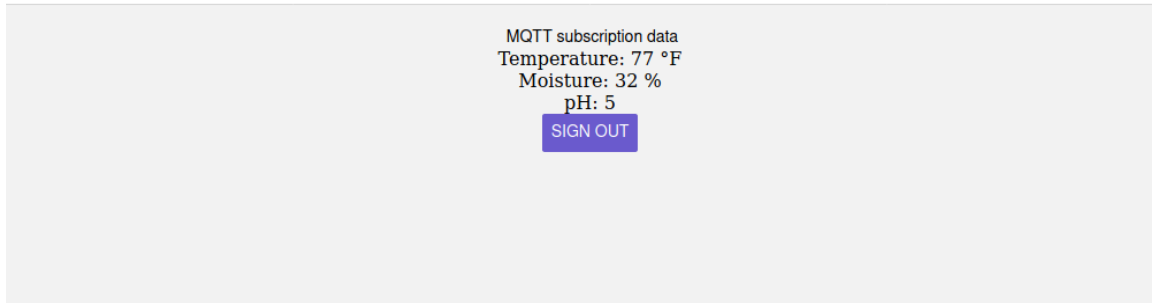


Figure 6: Home Page



Figure 7: AWS's MQTT Test interface which is publishing the data displayed in Figure 6

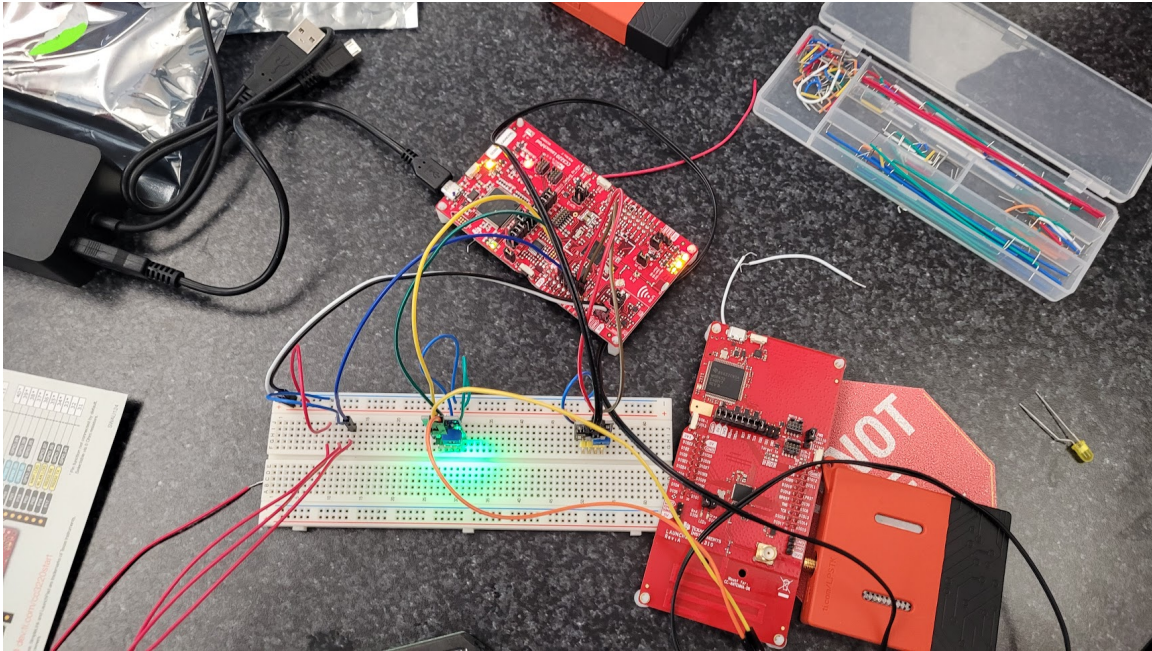


Figure 8: TI Launchpad wired to breadboard that leads to sensors

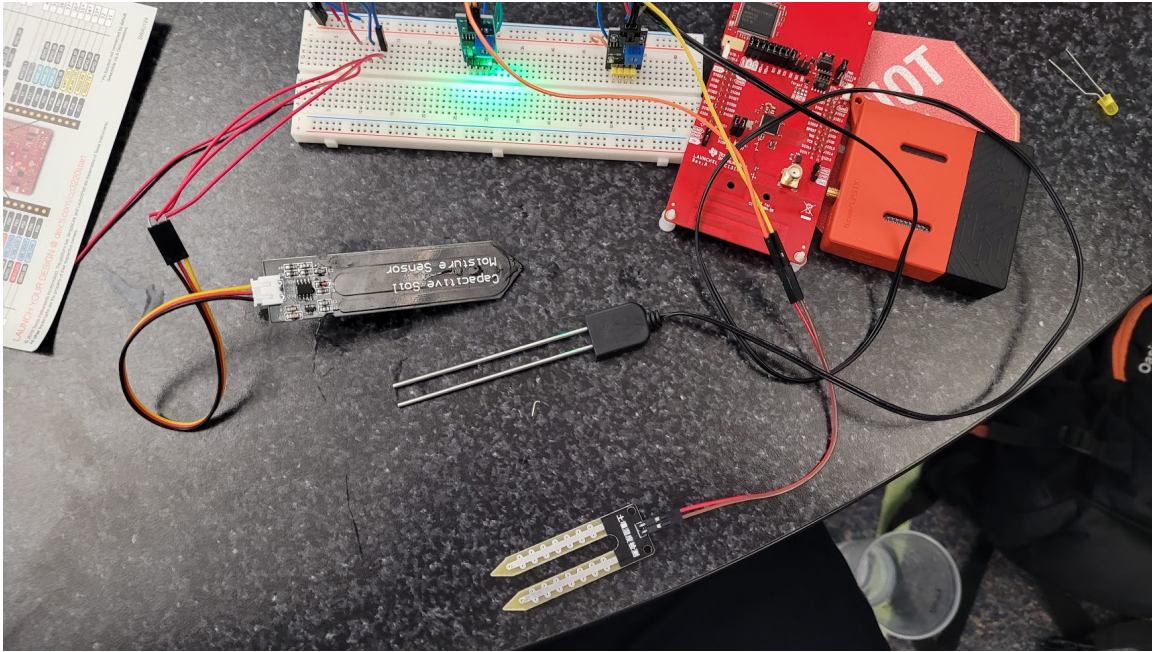


Figure 9: Sensors that are connected to the breadboard

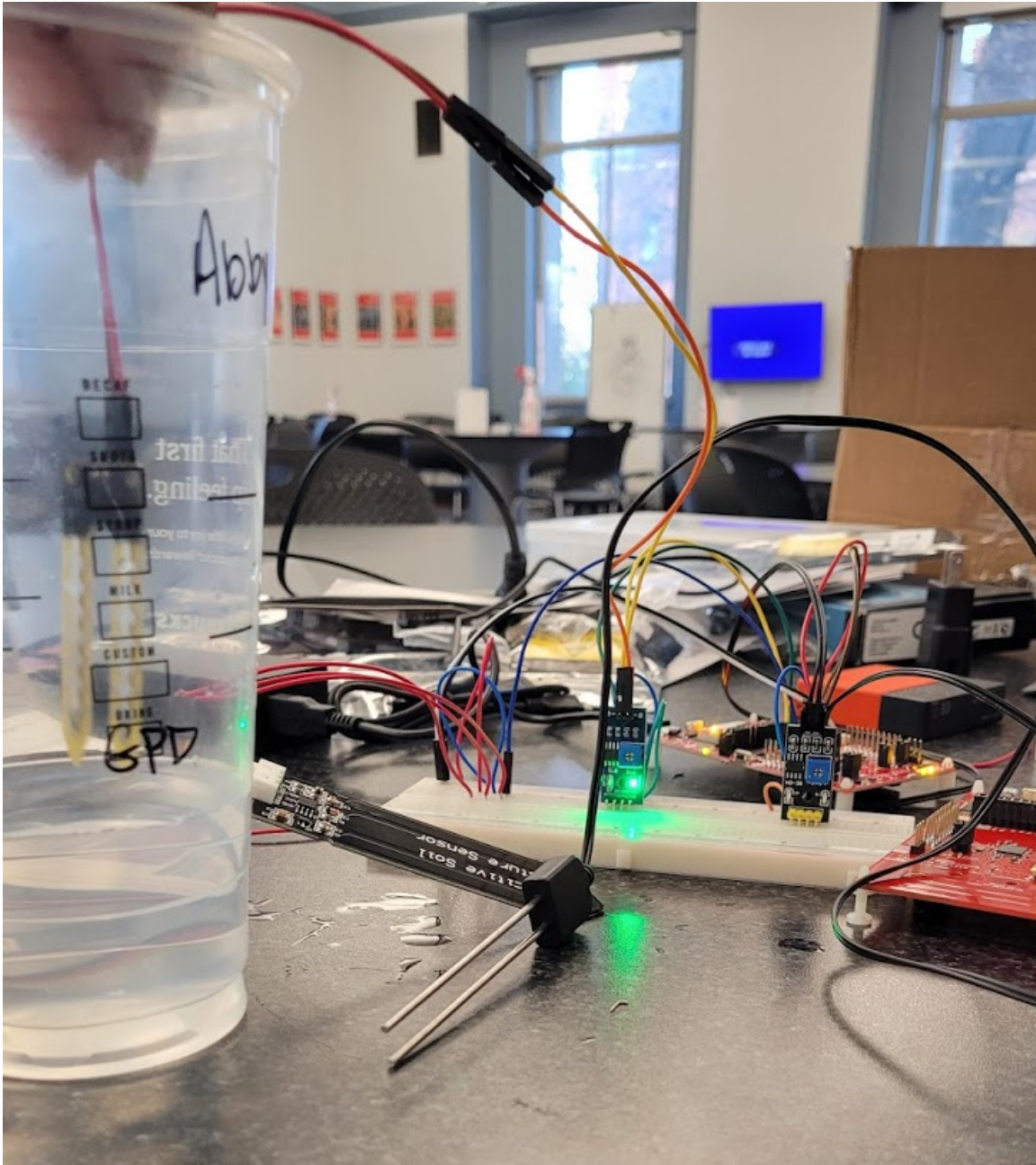


Figure 10: Resistive sensor not detecting moisture (**not in water**)

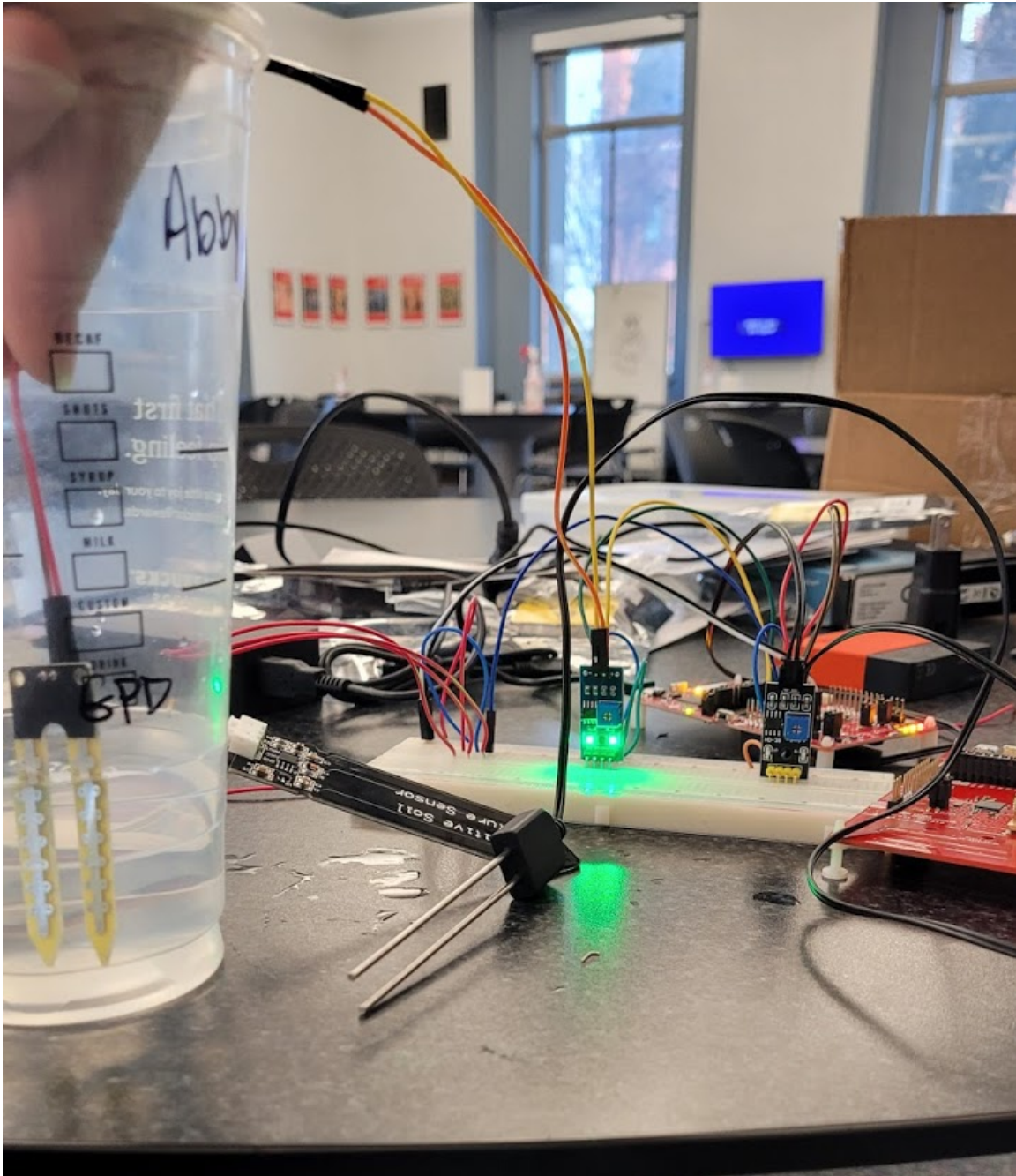


Figure 11: Resistive moisture sensor detecting moisture (in water)

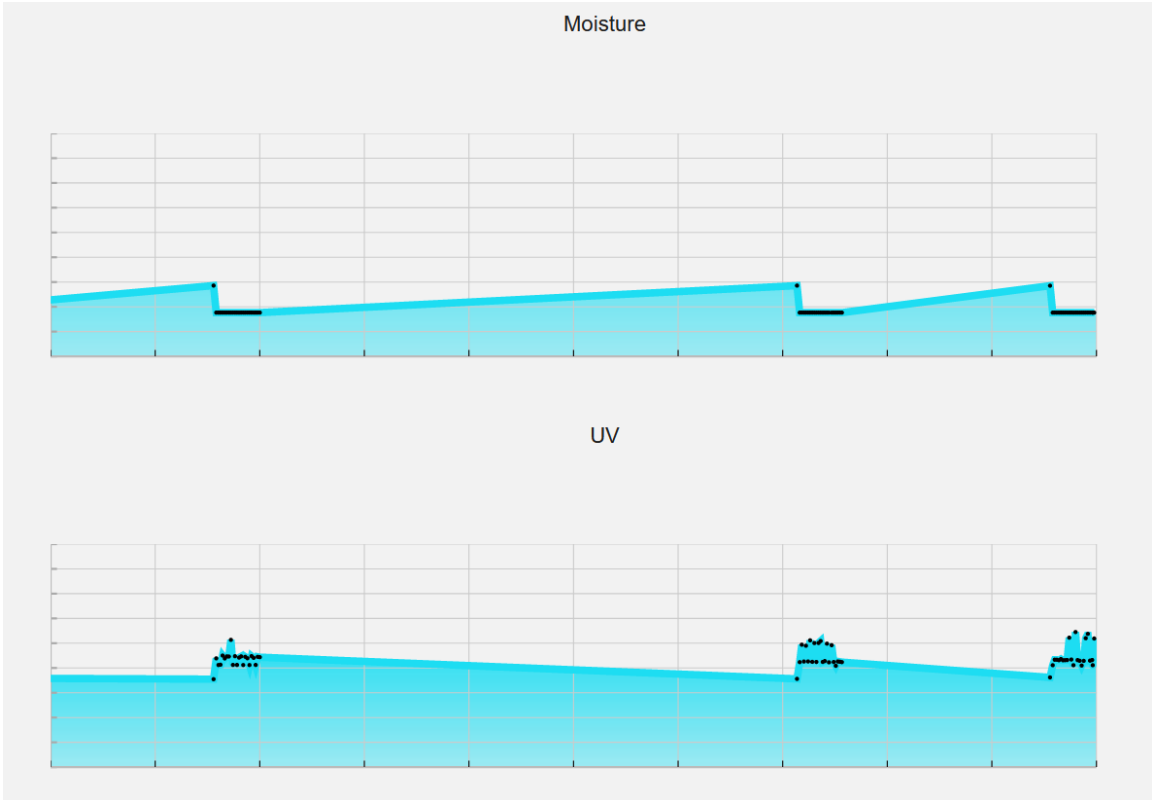


Figure 12: Graph view for 2 sensors

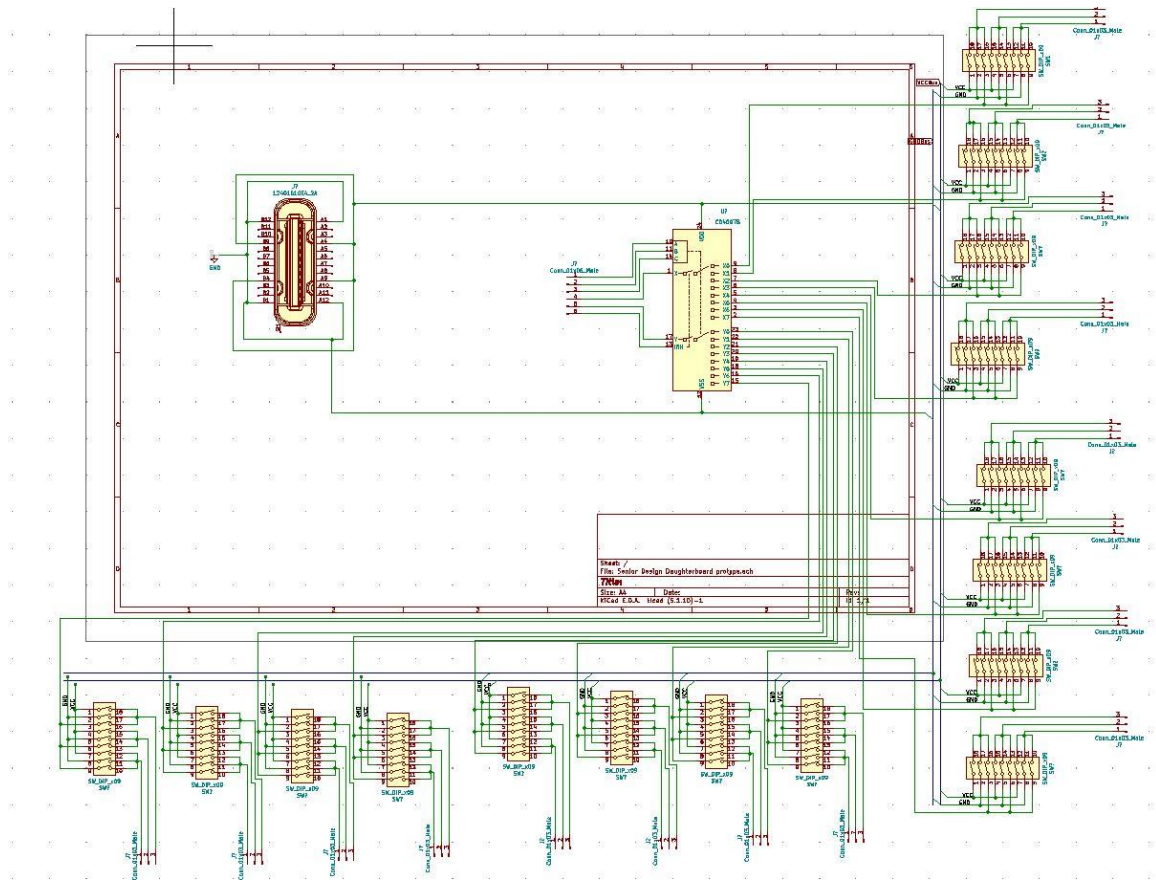


Figure 13: Schematic for first PCB design

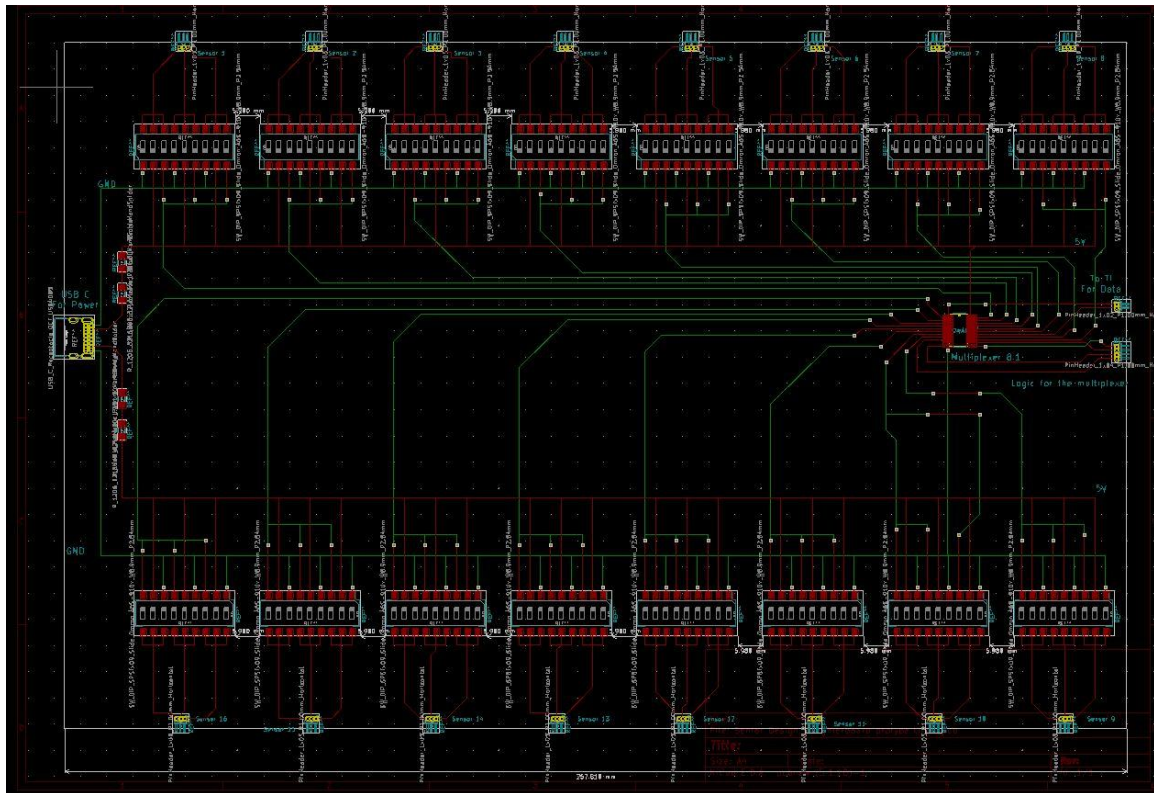


Figure 14: First PCB design

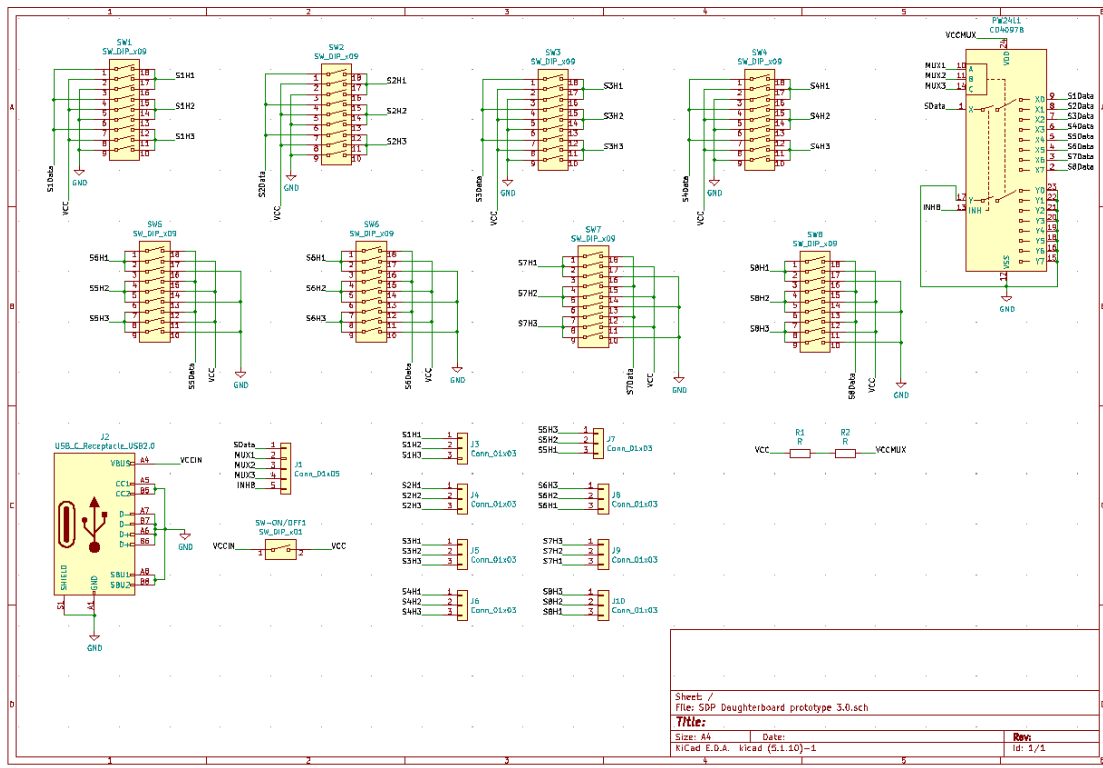


Figure 15: Schematic for PCB 2

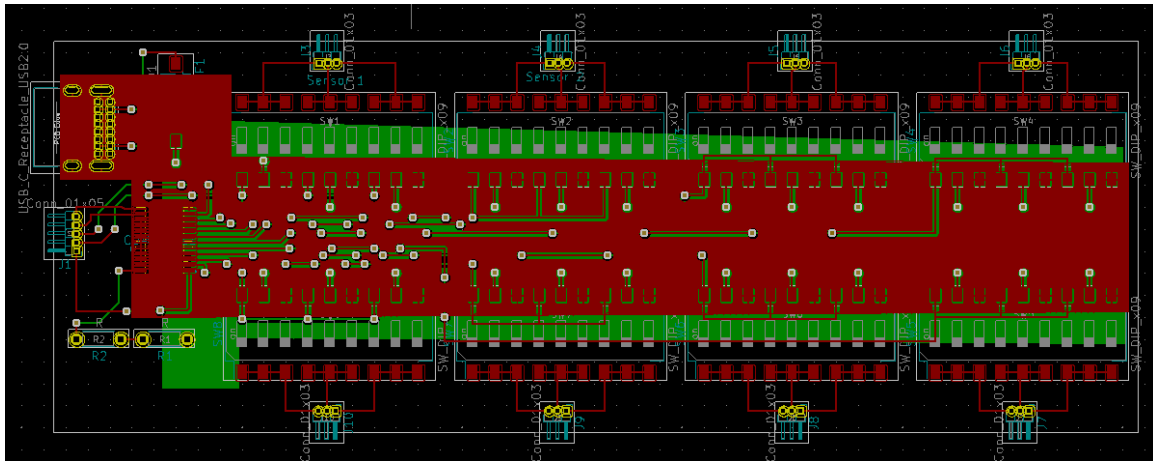


Figure 16: PCB 2 layout with copper pours

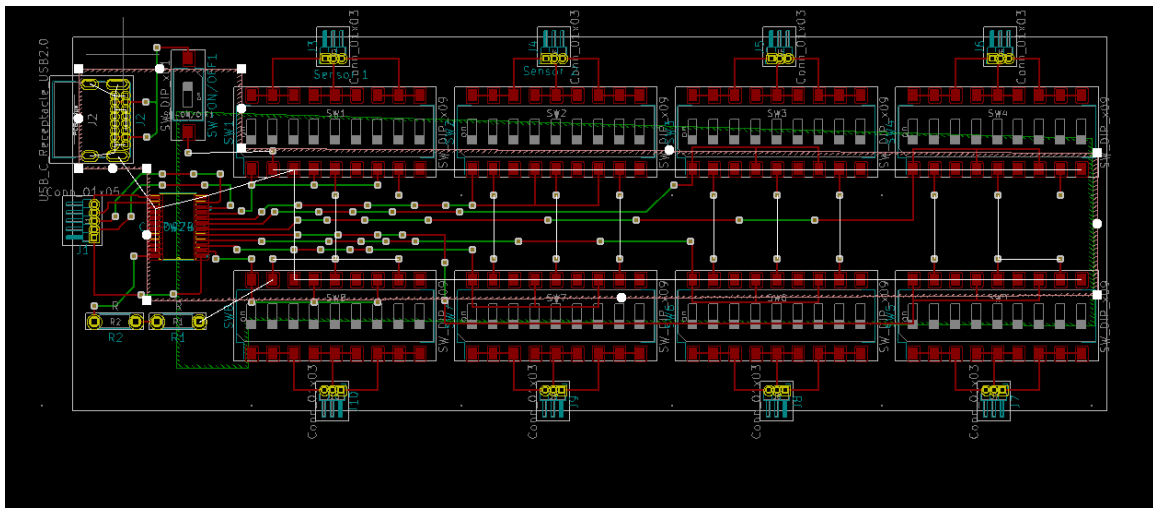


Figure 17: PCB 2 layout without the copper pour

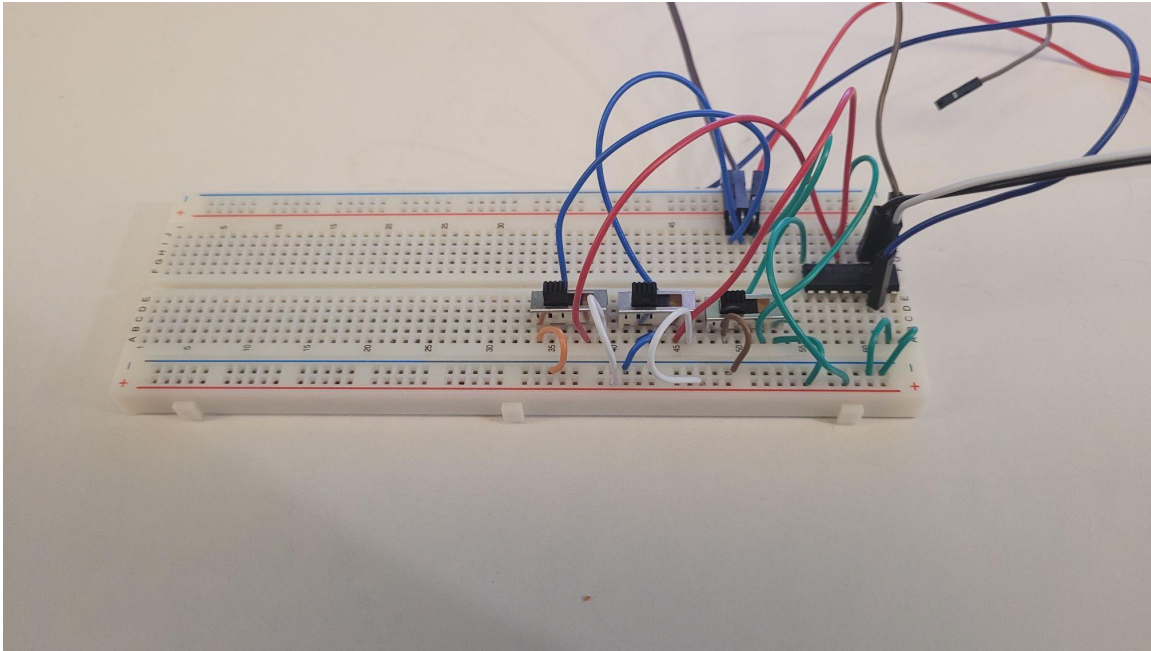


Figure 18: Breadboarding with new switches

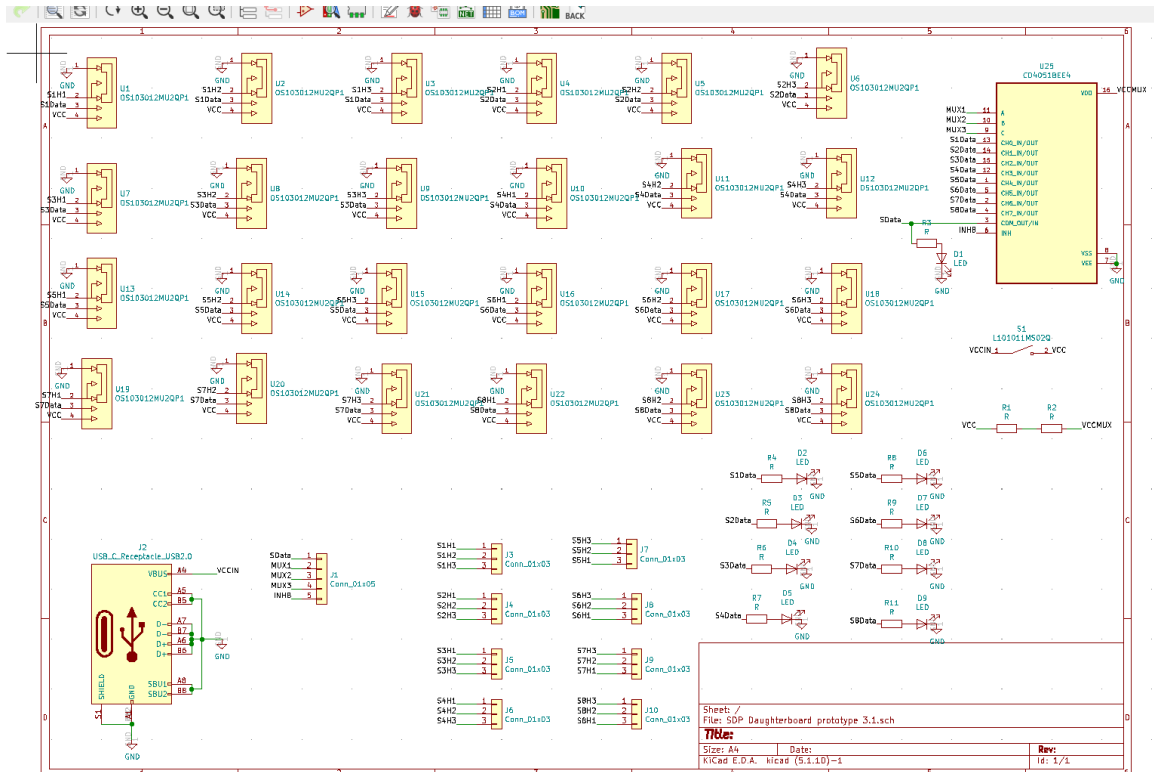


Figure 19: Schematic for PCB 3

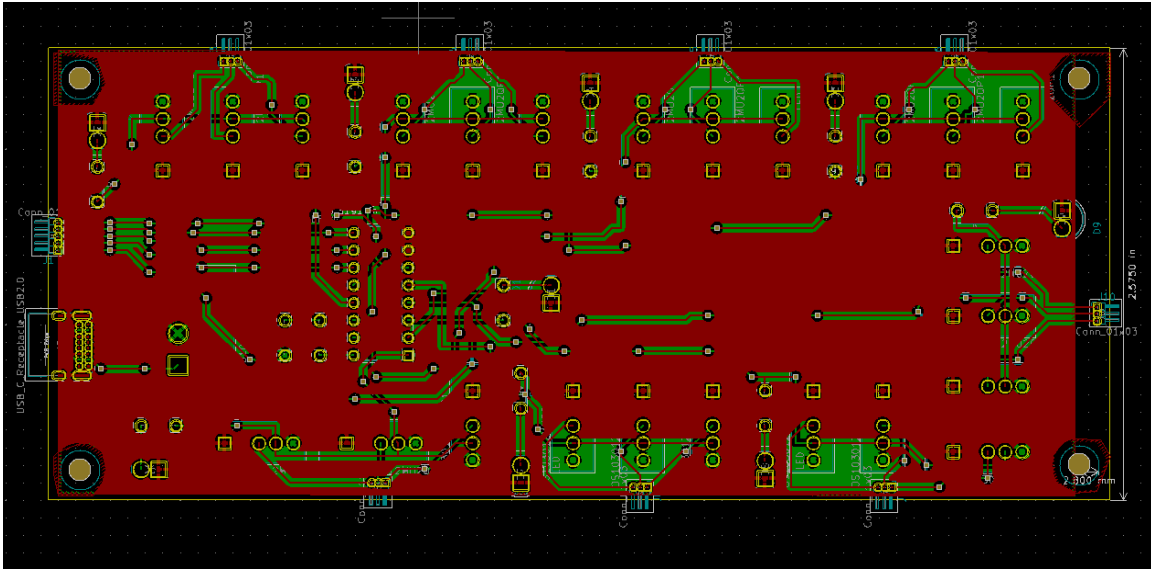


Figure 20: PCB 3 layout with copper pour (top view)

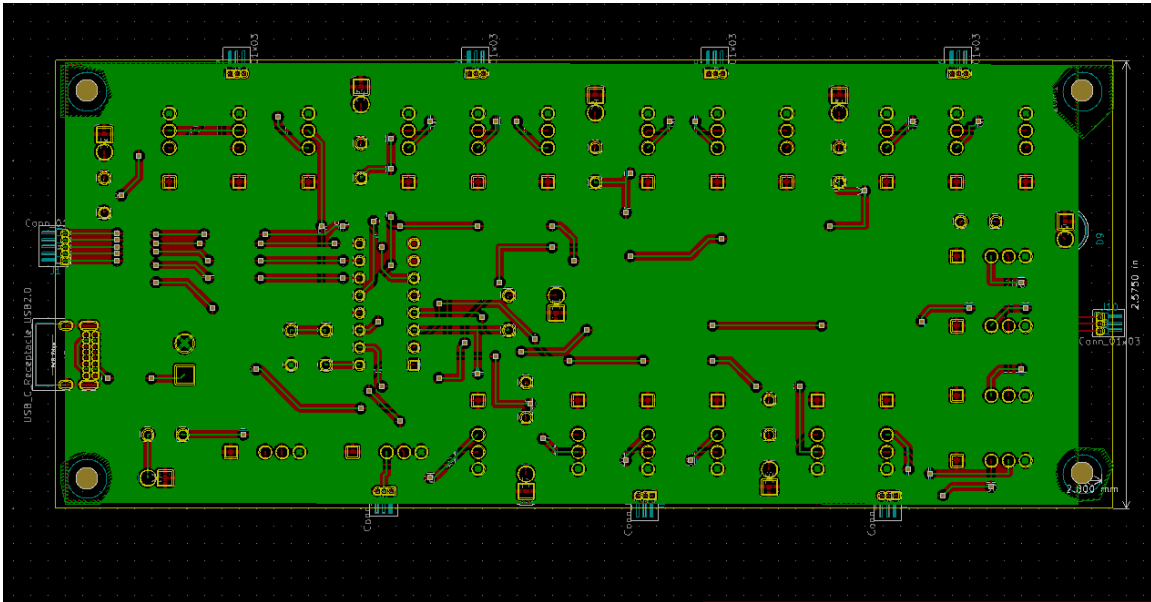


Figure 21: PCB 3 layout with copper pour (bottom view)

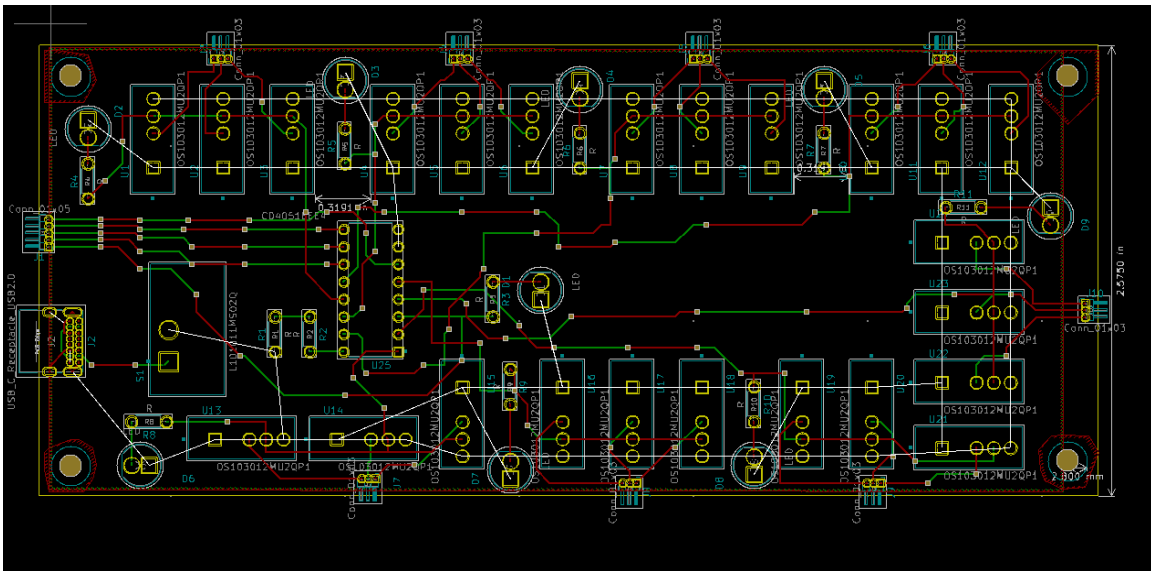


Figure 22: PCB 3 layout without copper pour

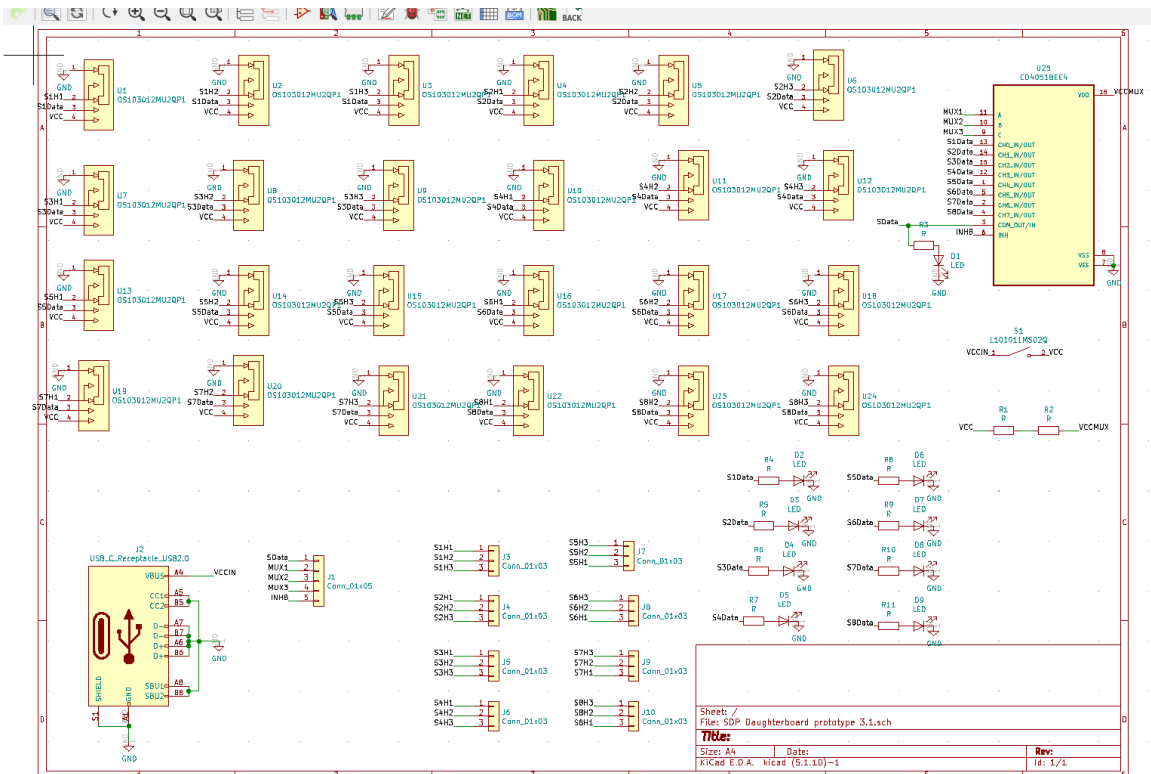


Figure 23: Schematic for PCB 3.5

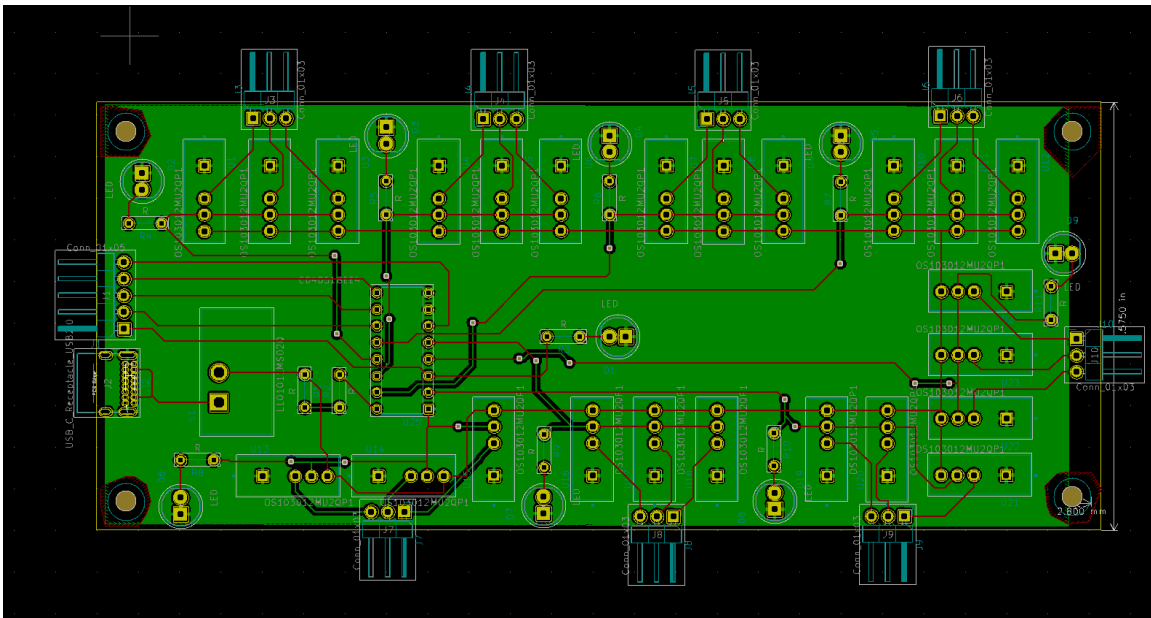


Figure 24: PCB 3.5 layout with copper pour (top view)

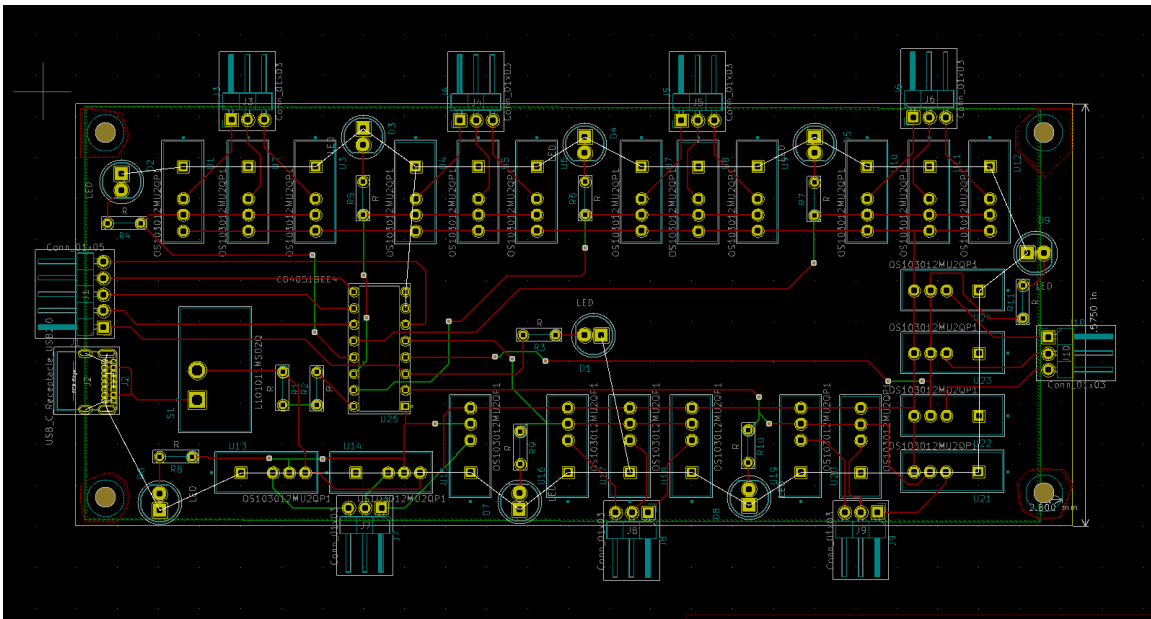


Figure 25: PCB 3.5 layout without copper pour

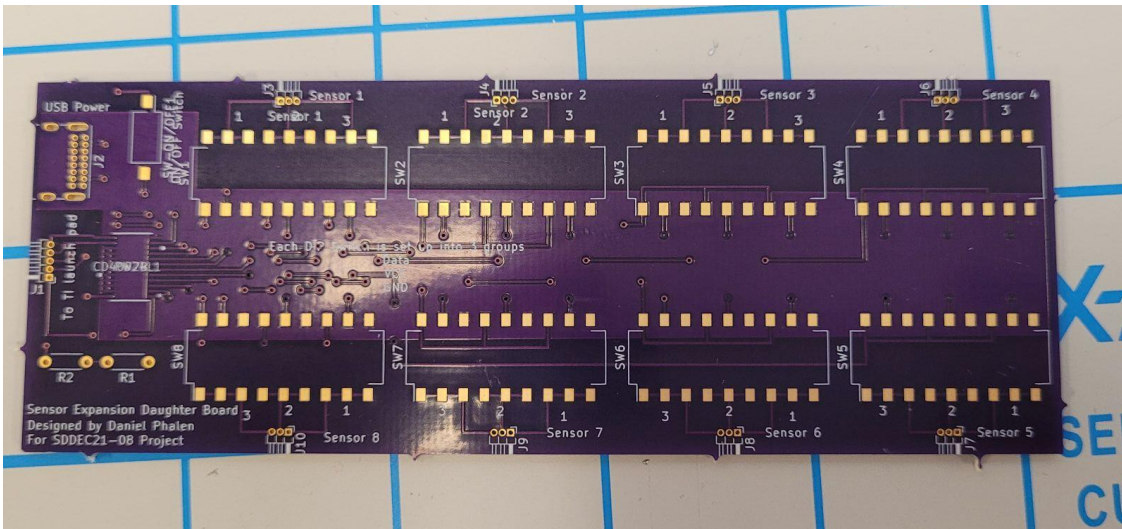


Figure 26: PCB 2

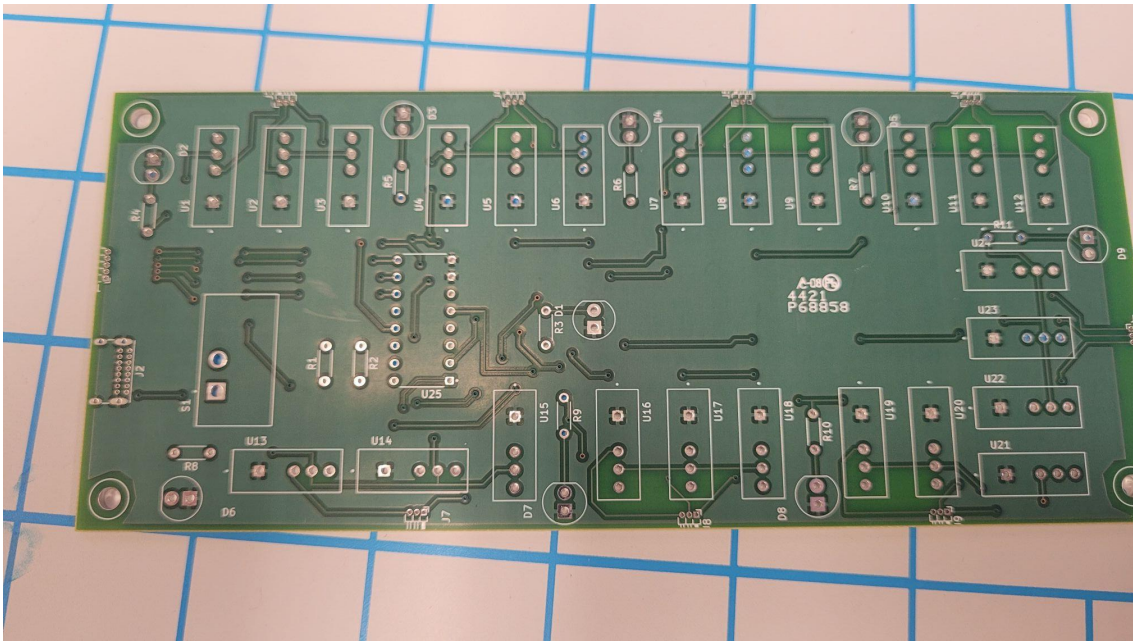


Figure 27: PCB 3