# Laboratory # 2 ECE:4880,

# Principles of ECE/CSE Design Fall 2021

# Team CASH

Cole Brooks, Hongyu Zeng, Ann Thomas, & Steve Wasiswa

### Design Documentation:

This portion of the lab report will describe the hardware and software design of the system. It will include relevant visualizations and refer to pictures of our final system in an attempt to help readers better understand our finished design before we begin to explain the design process and engineering decisions made in getting to this final design.

The challenge posed in this lab was straightforward at its face: design a receiver for a transmitter that already existed. The idea was to simulate a realistic situation that many of us will face in the future. Restrictions such as being unable to change the design of the transmitter are very common in industry, and we need to be ready for that. Some of the major sub-challenges for this lab included dealing with noise, and signal amplification. All of these challenges will be discussed in detail in later portions of this report.

Our goal in solving these problems was to meet all the customers’ requirements as well as provide a clean -looking product, and a straightforward and simple user experience.

Designing such a system required three components: Software, hardware, and a web application. All components would need to work together to provide a professional grade product and good user experience. In the following subsections, we will elaborate on each of these components.

Hardware:

The hardware portion of our final system (*Appendix Figure 3*) is a straightforward solution based on the needs of the customer. It is a box containing everything necessary to run the system as required. On the box, users will find a pushbutton and a toggle switch *(Appendix Figure 4)*, an LCD, a connector on a yellow cable, and a USB compatible with any USB battery or power bank a user may have.

Inside of the box is an Arduino Nano 33 IoT used as the brains of the system, as well as a 2N7000 N Channel Mosfet used for remotely controlling the LCD of the system, and all the wiring to connect all hardware components together.

Connected to the box, users will need some kind of powerbank or battery. The one that we used consistently for this project was a cheap and widely available powerbank from Walmart. A link to the exact powerbank used will be included in the appendix of this report. Users will also notice a long wire connected to the system as shown in figure 4. This long wire is terminated by the temperature sensor we use to get data into the system.

More information on each of these components can be found in the design process and experimentation portion of this lab report.

Software:

Diagram

Description automatically generated

Diagram

Description automatically generatedDiagram

Description automatically generated

The software components come from different sources since each software component solves a problem different from others. The following are the source codes along with the problems they solve:

- A component with code within Arduino handles the needed logic to process different operations such as reading temperatures, connecting to the database, connecting to WIFI, etc.

- A component with code that takes care of graphing data with two different units (Celsius and Fahrenheit) and turning on the LCD remotely.

- A component with code, which sends a message to the target user using Twilio API. And it happens when the temperature goes beyond or below the limit range.

- A component with a database (Firebase) that stores temperature data, phone number for alerts, thresholds for alerts, toggling display or not, and whether the alert has been sent for the current temperature out of bounds issue.

- A component with a web application(interface) to allow user interaction such as changing phone number for alerts, pressing the button virtually, and changing the temperature interval for alerts.

Above is a block diagram of the big software components.

Web Application:

As shown in the diagram of the software components, the web application is just another portion of the software. We have included a diagram for the web interface along with its subprocesses to allow readers to have a better understanding of our web application.

Diagram

Description automatically generated

The above diagram shows that our web interface needs three major components to meet customer’s needs. These components are the input section to allow users to change numbers and temperature intervals, the button to help users switch from Celsius to Fahrenheit units and vice-versa, and the remote pushbutton to help users control the LCD. The main goal of these three components of our web application, shown below, is to facilitate user interaction with our system.

## Design Process and Experimentation:

This portion of the report will describe the process by which we arrived at the final design for the Lab. We will compare multiple design choices, evaluate them critically for cost, performance, and complexity, and we will cover the process by which we determined which solution we would choose to go with.

This portion, like in the sections before it, will be split up into three sections to cover each component of our solution.

Hardware:

This portion of the report will describe the process by which we arrived at the final design for the Lab. We will compare multiple design choices, evaluate them critically for cost, performance, and complexity, and we will cover the process by which we determined which solution we would choose to go with.

The hardware portion of this lab seemed quite intimidating. We had never before needed to connect a microcontroller to the internet, let alone implement all of the additional functionality on top of that. Additionally, every project we had worked on in the past came with strict hardware requirements, so the specific devices we used were never a decision we had to make.

After diving in to this project, we soon realized that we could view this experience in a different light. We could look at it as if the lack of strict requirements was a challenge, or we could look at it as freedom granted to us in this project to figure out what works well and is simple to implement.

After our first meeting, we noted the main components that we needed for the hardware portion of this lab that needed to be sourced:

1. A microcontroller to be the brains of the entire hardware setup
2. A display module for that microcontroller
3. A temperature sensor that met as many of the requirements as possible
4. A pushbutton and toggle switch
5. A case to keep everything in.

We almost immediately decided that the best option as far as a microcontroller goes would be an Arduino board. An Arduino based solution makes the most sense for this project for a variety of reasons. Arduino boards are well documented and widely used for internet of things projects. Not to mention that they are cheap, small in form factor, and more powerful than necessary for anything we would need them to do for this project. They also have plenty of output pins so we would not be restricted in data outputs down the line.

After narrowing down our choice to some type of Arduino, we took to the internet to figure out which specific board we would use. Originally, we planned to use the Arduino Uno because we already had a few of them between the group, and we were quite comfortable using them. After some consideration, though, we realized that the Uno was larger than we would like, and it would need extra hardware to connect to the internet. After doing some more research we found the perfect candidate: the Arduino Nano 33 IoT.

The Nano 33 IoT is a small but powerful board created by Arduino. Arduino notes that the Nano 33 IoT was developed specifically to provide an easy point of entry to basic IoT and pico-network applications which was exactly what we planned to use it for.

The display was an easy decision to make because we already had access to a number of LCD 1602 modules, and the experience to get them integrated quickly and without introducing visual bugs into the complexity of the project.

Next we needed to figure out which temperature sensor we were going to use. This was figured out quickly. The first result for the search term ‘Arduino Temperature Sensor’ was the DS18B20 which fit our needs perfectly. It was already a physically robust temperature sensor, plus it came in a 2 meter form factor and only needed one data line to accurately measure temperatures well beyond the requirements for this project.

After the difficult hardware portions of the lab had been vetted and sourced we turned out attention to some of the smaller pieces. The pushbutton and toggle switches came next. Both of these components were basics that a member of the team already owned, so it was an easy decision to go with those.

The last part that we knew we would have to source was the container that we would use to house the project. This was solved quickly for the same reason as the pushbutton and toggle switch. One of the members of our group had a collection of cases designed specifically for Arduino project. We planned to use this container along with an electric router to fabricate the housing unit for this project.

After starting our hardware implementation, we realized quickly that we had missed a few significant hardware components:

1. Means for toggling the screen remotely
2. Means for powering the system
3. Means for allowing the temperature sensor to be plugged in and unplugged from the system

Our plan for toggling the screen remotely was to use a relay connected to the LCD and a pin from the Arduino to remotely control the LCD. The first issue we ran into with that implementation was that the relay we had access to did not respond to the Arduino Nano’s lower logic voltage. *(3.3 rather than the typical Arduino 5 V logic).* We opted to try and figure out a way to make it work. After some research we found that it would be possible to control the 5 V sensitive relay with 3.3 V only if we had some kind of amplifier circuit involved. That would quickly become complicated and our box was almost full as it was, so we needed to figure something else out. The final issue we would have faced with a relay based solution was that the Arduino I/O pins are not designed to be high current sources or sinks. Rather they were designed to be logical high or logical low with the voltage and ground pins of the board absorbing most of the current. After checking the spec on the Nano 33 IoT we found that the current measured via multimeter coming out of the LCD was too high for any I/O pin on the board to sink. We needed to find a different solution.

Some more research lead us to a mosfet based solution, and the Engineering Shop recommended the 2N7000. We took their word and used that specific mosfet as a switch to ground the LCD via logical high or low on the board. This solution proved to be a solid engineering design choice for this project.

Next we needed to figure out how to power the board. Originally we were planning on using a 9 volt battery inside of the case to power the electronics. One weakness we faced in using the Nano 33 IoT was that the LCD operates on 5 volts, and this specific board has been stepped down and operates only at 3.3 volts. There is a solution to this problem that requires one to solder a portion of the back of the microcontroller to allow for access to the 5 volts coming from the power supply. Unfortunately, doing this limits the board to being powered by the micro-usb alone, meaning that our vision of using a simple 9 volt battery would not work.

This was one area where our design fails a requirement. We were supposed to enclose the battery within the box for this system, but ours instead plugs in to the USB that protrudes from the box.

We decided that making the project compatible with any USB compatible battery would not only solve our problem, but add convenient functionality for any user planning on deploying this system. It would be much easier this way to change the battery as needed, because instead of unscrewing the case and fumbling with all of the wires, one could simply unplug the old battery, and plug a new one in. If given more time, we would redesign the case for more efficient use of space, and find a smaller USB battery to power the system from within the box. For that solution, we would add an extra USB port for charging the battery after the system ran out of charge.

Finally we needed to find a means for the temperature sensor to be plugged into and unplugged from the rest of the system. This would make the design more robust and modular, so users could replace the sensor with a length that suited their own needs, or even replace damaged sensors without the need to source the rest of the system again.

We deferred to the expertise of the Engineering Shop again for this issue. We were given a 4 pin port that worked very well for our temperature sensor. We soldered the female end into our circuit, and the male end to the temperature sensor allowing us to plug and unplug the temperature sensor as we needed.

Software:

Web Application:

One of the main challenges that engineers and developers face when developing a product is finding the right tools or technologies that will allow them to deliver a finished product. Our team also faced that challenge during the development process for our design.

First, we decided to use the python web framework Django, a friendly framework for building a web application, to serve our backend and frontend. We still did not know how to use it to our advantage to produce graphs on the web interface.

After our initial setup with the server using Django, we came up with a list of three choices of JavaScript libraries. And these choices were D3.js, Chart.js, and Plotly.js. Eventually, we chose to go with Chart.js since we set up our first server using Django, and Django works better with Chart.js.

Next, we chose Chart.js as our main library for graphs, and we still had to figure out a way to serve a webpage; Django has some of the easiest ways to do that. Initially, we decided to use Bootstrap for our web page styles and JavaScript to add some interaction for other features such as the remote button for the LCD, the switch button for the temperature units, etc.

During the development process, we had difficulty trying to connect the Arduino to the server. We could not move forward with the front end since some of the front features did depend on the backend. In the meantime, we used dummy data to make plots to gain some experience using Chart.js.

Unfortunately, we faced an enormous challenge trying to connect the Django server to the Arduino without relying on the serial port for the data transfer. Due to this challenge, we had to try different technologies.

At some point, we tried using Node.js for the web application due to its popularity with these kinds of tasks; again, we faced the same challenge of trying to connect the Arduino to the server without relying on the serial port for the data transfer. And when we succeeded, we faced other challenges with different libraries. In

Next, we implemented two standalone python scripts to transfer data from the Arduino to the python server script and serve a webpage using HTML, CSS, and JavaScript. For the frontend, we had to make sure the web application has the following components:

- A switch button for uses to switch between temperature units (Celsius and Fahrenheit)

- A means for users to change phone numbers and temperature interval

- A pushbutton for users to control the LCD

After playing around with Chart.js, we could implement the switch button and other customers’ requirements on the graphs such as zoom feature, tick marks, interval limits, etc. Meanwhile, we were also able to implement a pushbutton to allow users to control the LCD.

We had to rely on TinyDB to store our data since we utilized the python web socket server. Unfortunately, using such a less powerful database for queries slowed down the data transfer between different units. Therefore, affecting the display of the graphs.

Next, we had to find a way to solve the issue, and we did so by instantiating a web socket from the client-side using JavaScript to communicate with the server. Then, we got rid of the database. Even though we changed our approach, there were some delays during data transfer.

Finally, we implemented a feature to allow users to change their numbers using JavaScript and CSS; we decided to use Firebase to serve as a server and database. With Firebase, we were able to get rid of most of our issues, if not all. We connect our frontend to Firebase to test the app with real data, and every feature we implemented using dummy data worked fine when we connected our frontend to the Firebase. At the end, we delivered a web application using one file for HTML, one file for our JavaScript codes, and one file for our CSS codes

## Test Report:

This section will include a table that describes all tests conducted to verify the completeness of our prototype. This section will be grouped slightly differently than the rest of the report. Rather than splitting this section into three categories (Hardware, Software, Web Application), this section will be split into two: Arduino & Web Application. We do this because we deemed the operation of the hardware is inseparable from the operation of certain parts of the software.

Arduino:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test** | **User Requirement Address** | **Expected Result** | **Actual Result** | **Pass/Fail** | **Date** | **Correction Action** |
| Length of Temperature Sensor | System has a thermometer sensor at the end of a 2.0 meter cable | Sensor that satisfies customer criteria is connected to the system | As expected | Pass | 9/3 | None |
| Physical  Robustness  Of  Temperature  Sensor | Temperature sensor should be of ‘nice mechanical construction’ (physically robust)   * Capable of bouncing   Sensor is not damaged when placed in ice water | Sensor can be dropped onto a hard surface or placed into ice water with no noticeable decreases in performance or accuracy. | As expected | Pass | 9/3 | None |
| Arduino System Physical Requirements | Box contains a display, a button, a battery, and a power switch. | System has all components required by customer | As expected | Pass | 9/10 | None |
| Enclosure Requirement | Arduino system is enclosed in some way | Arduino system has an enclosure to protect it from physical trauma. | As expected | Pass | 9/5 | None |
| Arduino System is physically robust | When dropped to the floor with cables connected, the connectors or cables should not break. | Arduino system should continue to operate as expected after being dropped from a typical height desk | As expected | Pass | 9//10 | None |
| Temperature Sensor Interchangeability | If the sensor has been unplugged and is then plugged in, the system should resume normal operation without user intervention. | Arduino System should notify user when temperature sensor becomes unplugged, and resume normal operation after the temperature sensor is plugged in again. | System requires users to close an alert before resuming normal operation | Fail | 9/25 | JavaScript implemented to automatically close the alert screen |
| Display Delay | The correct temperature should appear on the display when the button is pressed with no noticeable delay ( < 20 ms ) | Arduino system should display correct temperature within 20 ms of user pressing the red push button | As expected ( time to display was so short it was unmeasureable ( < 1 ms ) | Pass | 9/20 | None |
| Display Readability | Display should be clearly readable under normal indoor lighting conditions for all temperatures within normal range of the device | Display should be readable under a variety of light sources (Indoor with all lights on, indoor with all lights off, outdoors during the day, outdoors at night) | As expected | Pass | 9/25 | None |
| Pushbutton Operation Type | The pushbutton operates in a ‘momentary contact’ way | Pushbutton should turn display on while pressed, and display should turn off when the pushbutton is released | As expected | Pass | 9/3 | None |
| Display Notification for unplugged sensor | Display notifies user when temperature sensor is unplugged or damaged | User should see clearly on the display that the temperature sensor has become unplugged or damaged | As expected | Pass | 9/10 | None |

## Project Retrospective:

This section of the report will include a summary of the outcome of our project. This will include a critical assessment of the effectiveness of the choices described in the ‘Design Process and Experimentation’ section. This section will also include a discussion of factors that contributed to less-than-complete success in meeting our goals and an explanation of how we will change these factors in the future. We will also summarize the role of each team member in this lab, and the workload distribution between team members. We will discuss project management processes that were implemented and outline the timeline for this project.

Overall, we were proud of the outcome of this project. In just over a month, we were able to produce a prototype that satisfied almost all the needs of the customer with very few bugs. We felt that our solution met most of our goals. We satisfied almost all of the requirements set forth by the customer, we delivered a cleanly designed hardware portion for this project, and we designed and implemented a user interface that is straightforward and pleasant to interact with.

We will admit, however, that there were indeed failures in this project. We failed to deliver a product with a power supply included inside of the box. Additionally, some functionality when the Arduino system does not have power is not quite as desired by the customer. Next, we will discuss factors we believe contributed to these failures.

The biggest factor leading to the failures we experienced in this project was time management. As seniors, we’re all taking heavy course loads and have very little time to spend on such high intensity projects as this one. This was especially difficult because what little free time each group member did have was usually out of sync with the free time of the others.

To mitigate the issue of time management in the future, we have agreed to save a timeslot on Saturday mornings in which we will discuss progress, and next steps for the lab. Additionally, we will implement remote scrum/standup meetings in which we give short reports on our productivity at least three times each week (Monday, Wednesday, and Friday). We believe that these consistent check ins will lead to higher participant accountability and diminish the chances that we feel a time crunch as we approach deadlines for the next lab.

Another factor contributing to the failures experienced in this lab was simply inexperience with each other. None of the participants knew each other before starting this lab, leading to difficulty communicating early on. This issue mitigated itself as we proceeded through the lab, resulting in more team chemistry and higher productivity when we were working together.

As discussed in previous sections, there was an issue where halfway through the lab we realized that our implementation was not efficient, so we needed to basically start over on the interfacing between the Arduino system and the web application. This failure in planning lead to a lot of downtime in a critical part of our process. Additionally, some of the hardware did not operate as expected leading to more engineering decisions than necessary when we should have been focused solely on implementation.

To mitigate this issue, we will implement a meeting at the beginning of labs in which we all get onto the same page about design decisions early on. After that, we will deal with issues as they come, and do our best to avoid new engineering decisions until the project has been fully implemented. We want to remain as agile as possible to avoid boxing ourselves into a corner, but we should slow down a bit when making dramatic changes to the codebase. From now on, large changes will need to be discussed and thought over thoroughly before being committed to.

There were four members of our team, and each contributed their own unique expertise and work to the success of the project. Next, we will discuss in detail the contributions of each person.

Cole Brooks contributed most to the hardware portion of this lab but acted as an overall floater. He made most of the hardware decisions after doing most of the hardware research, purchased all hardware required for this lab, and worked with Hongyu to produce the software required to run the hardware. After all of that, Cole acted as a floater, helping others where they needed throughout the lab. He also tested the hardware portion of the system.

Additionally, Cole implemented one of the most impactful changes made to the system. Cole took point on the conversion from a Web Socket based approach to an approach that utilized Google Firebase instead as a means of interfacing the Arduino system with the Web Application.

Cole also acted as the administrator for the version control system. The entire project can be found on Cole’s GitHub page. He created the repository and did all the setup to provide for an easy development experience for the group.

Finally, Cole assisted with bug squashing in the final days of the project. When group members had issues which they were unable to solve, Cole was in calls with them helping to diagnose and fix issues.

<Hongyu’s contributions>

Ann worked on creating the frontend of the project. She created the web app user interface using Html, CSS, and JavaScript. Specifically, she learned and utilized that Chart.js library in order to create the graph for the project. Her part included creating the webpage itself, creating the graph, and adding required features such as a zooming capability and the ability to update in real-time.

Ann also worked on integrating the frontend with the server and backend portion. She worked with Cole on this to merge the two components and make edits so that the chart would update in real-time with the temperature data sent by the Arduino.

Steve set up the initial server using Django to server our backend and frontend. Even though the Django server is the right for the rapid development of web applications, it turned out to be more complex with our project when trying to connect the server to the Arduino without relying on a serial port for data transfer. Then we abandoned the Django server, including all initial web pages and user authentication systems. He helped set up a second server using Node.js, which worked with the serial ports. However, a system that was implemented to transfer data without utilizing a serial port was not fully tested due to some technical problems related to Node.js from one computer to another computer.

Steve helped to choose the right tool or technology for the graphs, but most of the work was done by Ann.

Steve helped with the implementation of the initial remote button, which was working with one state. Then, he implemented it using two states, false and true. Eventually, we had to reinstate one state toggle button for the web application to meet our customer’s needs

The project management process that we implemented was a modified remote agile methodology. This methodology focused on team members being able to work at their own time and pace and contribute what they could where they could. It still focused on the core frameworks of agile, as we consistently provided updates to the group in a pseudo-scrum fashion.

Workload assignment happened very early on in the project. As early as our first meeting we knew approximately what each team member would be responsible for. After that, each day we each provided an update to keep the other group members informed in our progress.

Our primary means of communication was the popular messaging application Discord. Being able to communicate on discord contributed to our success because we were able to keep our thoughts organized into channels, we were able to quickly share files and links with each other, and we were able to have a backlog organizing the progression of the project. We will definitely continue to use Discord as our primary means of communication in the future.

For version control, as mentioned previously, we used GitHub. GitHub is a website for programmers designed specifically to incentivize collaboration on code. The primary benefit with GitHub was it’s version control system which allowed for easy code retrieval, seamless merging of new features, and the ability to work quickly without compromising the integrity of the project. We will definitely be using GitHub again in the future.

In the future we would like to do a better job committing to the agile methodology. Toward the end of the project we got lazy and quit meeting up and giving updates on our statuses. This lead to each of us misunderstanding where the rest of the project was at, and lead to some overlap in work that could have been avoided if we only communicated a little bit better. The best way to solve this issue is going to be to double down and work hard to keep to a timeline, and schedule meetings far in advance to avoid time conflicts.

## Appendix & References:

Diagram, schematic

Description automatically generated

Figure : Hardware Schematic

Diagram

Description automatically generated

Figure : Hardware Wiring Diagram

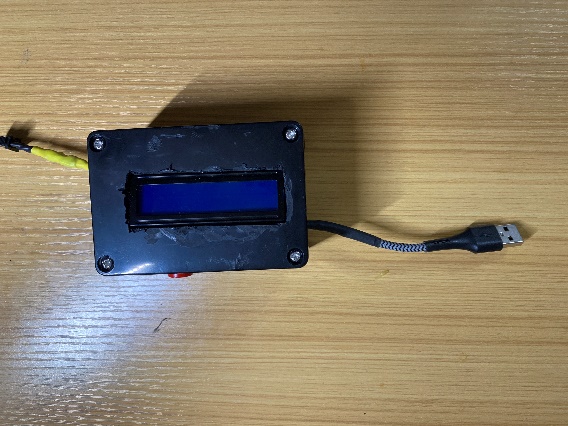


Figure : The final hardware system

A picture containing floor, indoor, wood

Description automatically generated

Figure : Top view of the hardware system



Figure : An image of the powerbank used for this lab. <https://www.walmart.com/ip/Power-Bank-50000mAh-Portable-Charger-External-Battery-Pack-With-Dual-USB-Outputs-For-IPhone-12-Mini-Pro-Max-IPad-2020-Samsung-AirPods-And-More-Ack/879758536?wmlspartner=wlpa&select>

A picture containing bicycle, light, electronics

Description automatically generated

Figure : System with temperature sensor connected

Vitor Freitas (<https://simpleisbetterthancomplex.com/tutorial/2020/01/19/how-to-use-chart-js-with-django.html>)

Geeks for Geeks (<https://www.geeksforgeeks.org/data-visualization-using-chartjs-and-django/>)

Chart.js (<https://www.chartjs.org/docs/latest/>)