ECE:4880 Lab 1 – Temperature Sensor System

Team: Four Engineers Walk into A Bar

Members: Jacob Sindt, Ryan Griffin, Emma Taylor, Tony Longo

1. Design Documentation

When first embarking on this temperature sensor lab, we found it necessary to first construct a block diagram of how all the pieces of the system will function together. The way the overall system works is as follows: The ESP32 microcontroller is powered via an external battery pack using a microUSB connector. This power connection can be toggled off and on using a single pole double throw switch. The temperature sensor connects directly to the microcontroller and sends data through an IO port. This data from the ESP32 can then be displayed to an LCD screen by pressing and holding a push button and is also sent to an external website via WIFI to be displayed on a dynamic graph. The block diagram for this lab can be seen in figure 1.

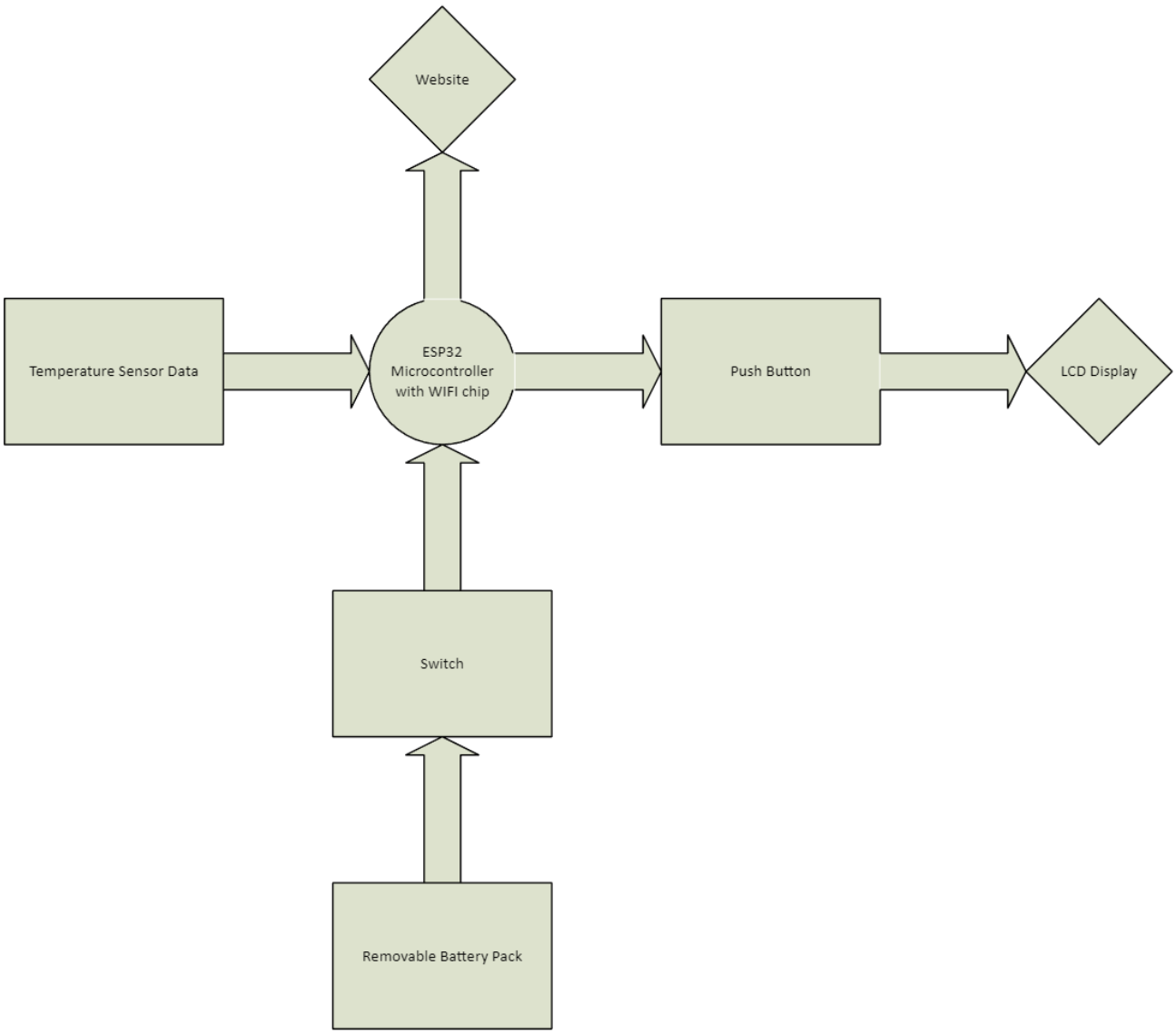

Figure 1: Overall temperature system block diagram.

1.1 Thermometer Design

When choosing a temperature sensor for this lab, we had to find a sensor that would fulfill the requirements necessary for the system. We ended up utilizing the DS18B20 temperature sensor for many reasons. The first reason for this choice was because this sensor is waterproof and comes with a 3-meter-long cable. This will satisfy our waterproof and 2-meter-long cable requirements. Another reason for this design choice was due to the simplicity of the sensor itself. We would only need one IO pin to read the data off this sensor which we were fond of.

1.2 Microcontroller Design

In the early development stages of this lab, we decided to use an Arduino UNO in conjunction with a WIFI shield to provide access to the website. As we progressed through the design process, we decided to make a switch to using an ESP32. The reason for this design choice was because the ESP32 has WIFI capability provided on the board, which we decided would be beneficial for integration purposes. Along with this, the ESP32 can be written to using the same Arduino IDE that was being used for the UNO. Another design factor that led to choosing this controller was the abundance of libraries available that could be used to provide internet connectivity between our system and our website.

1.3 LCD, Switch, and Button Design

LCD Design:

For our device, we were tasked to create a display that showed the current temperature the thermometer was reading while the pushbutton was pressed. We designed our display with a liquid crystal display, the LCD1602, running the device in its 4-bit mode. We decided on using the device in 4-bit mode to save space for other I/O pins on our microcontroller. At the top of figure 2 is the wiring for the LCD (LCD U1). The LCD pin function descriptions are listed below:

- Pin 1 (Vss): Power supply (GND). It is directly connected to the ground pin from the microcontroller.
- Pin 2 (Vcc): Power supply (+5V). It is directly connected to the 5V power supply pin from the microcontroller.
- Pin 3 (Vee): Contrast adjustment pin. It is connected to the center pin of a 10 KΩ potentiometer. This allowed for the user to manually adjust the contrast of the LCD.
- Pin 4 (RS): Digital bit that is used as an output to the microcontroller. Controlled the functionality of the LCD screen. If the voltage at pin 4 is high (logic bit of 1), then the data transferred to/from the device is treated as characters. If the voltage at pin 4 is low (logic bit of 0), then the data transferred to/from the device is treated as commands. Pin 4 was connected to I/O pin 14 on the microcontroller so that we could change from writing commands and writing characters into the LCD.
- Pin 5 (R/W): Digital bit that controls the data transfer direction. If the voltage at pin 5 is high, then the microcontroller reads from the LCD. If the voltage at pin 5 is low, then the microcontroller writes to the LCD. For our purposes, we did not need to read from the LCD and because of this, we terminated pin 5 onto the ground rail.
- Pin 6 (E): Digital bit that initiates data transfer. For writing, data transferred to the LCD on high to low transition. For reading, data is available following low to high transition. Pin 6 was connected to I/O pin 27 so that the user can strobe the value at pin 6 which will allow the user to begin transferring data to the LCD.

- Pins 7 through 14 (DB0, DB1, …, DB7): Digital data bus lines. These 8 pins are where data is transferred from the microcontroller to the LCD, or from the LCD to the microcontroller when used in read mode. The LCD can be used in either an 8-bit mode or a 4-bit mode. When used in 8-bit mode, the LCD uses all 8 data lines to transfer data (DB0 through DB7). In this setup, the interface requires 10, and sometimes 11, I/O pins on the microcontroller. The data transferring happens in one step, meaning that all the data is transferred all at once. When used in 4-bit mode, the LCD uses only 4 data lines to transfer data (DB4 through DB7). In this setup, the interface requires 6, and sometimes 7, I/O pins on the microcontroller. The data transferring happens in two steps, meaning that it takes two separate nibbles on the same 4 data lines. This process transfers the higher order nibble and then the lower order nibble. For our system, we designed the LCD to be used in the 4-mode and connected DB4, DB5, DB6 and DB7 to Pins 26 25, 33 and 32, respectively.
- Pin 15 (LCD LED +): If the voltage on this pin is greater than +4.2 Volts, the backlight for the LCD is turned on. This in conjunction with the status of pin 16, turn the LCD backlight on and off. Pin 15 was connected to a 220 Ω resistor that led to the 5V power supply rail. The 220 Ω resistor was used to step down the voltage at pin 15 to limit the continuous forward current to levels that wouldn't damage the device.
- Pin 16 (LCD LED -): Power supply (GND) for backlight. This connects the backlight circuitry to the ground pin on the microcontroller. This in conjunction with the status of pin 16, turn the LCD backlight on and off. Pin 16 was directly connected to the ground pin of the microcontroller.

Switch Design:

For our device, we were tasked to create a switch interface on the device that allows the user to turn the thermometer system on and off. When the switch is "off", the display screen cannot display temperatures and the temperature data is not available on the internet. In the bottom right corner of figure 2 shows the wiring of the switch (S1). The switch we used is the 7000 series toggle switch. The switch is a 3-pin device that changes the connections between the middle pin and the other two pins by changing the angle at which the toggle switch is pointed. For the switch, the middle pin was connected to I/O pin 34 on the microcontroller. The other two pins were connected to either side of the supply rail, one to +5v and the other one to ground. This setup, using the microcontroller, allowed us to determine the position of the switch, by the voltage it sees at the I/O pin. In one position the I/O pin would see +5V and in the other position the I/O pin would see ground.

Button Design:

When choosing a button for our system, we decided to design our system using a larger form factor button over smaller buttons. The reason for this was due to the need to have our button be easily accessible as it protrudes from the box so we could easily display to the LCD, whereas if we used a smaller sized button, it would be hard to access it through the walls of the box. Other than the need for a larger button, we used a normally open push button to enable us to display to the LCD when the button was pressed in and made a connection.

The circuit schematic for the system hardware can be seen in figure 2 and the bill of materials (BOM) for the entire temperature sensor system can be seen in table 1.

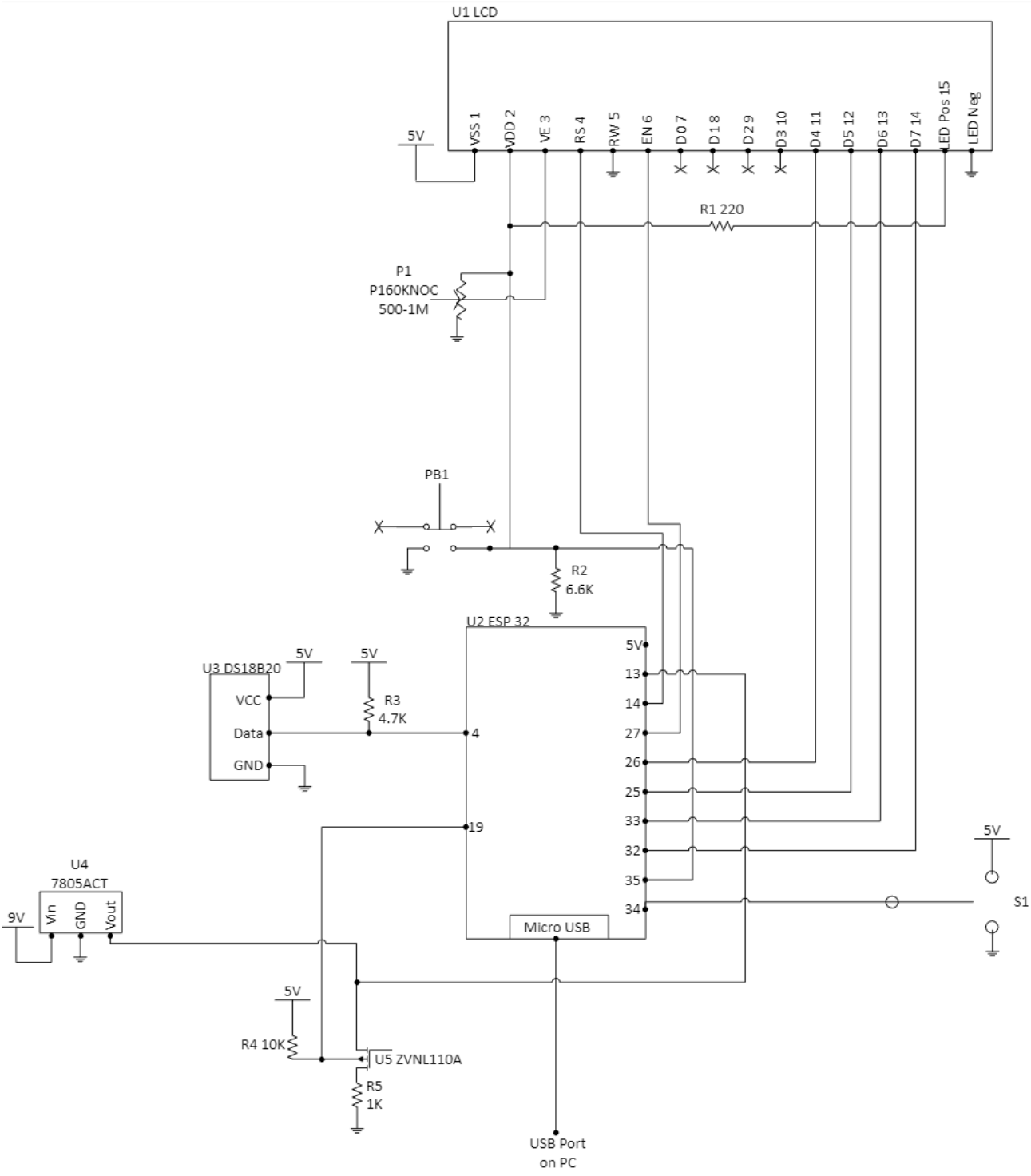

Figure 2: Circuit schematic depicting how the ESP32 interacts with the push button, switch, temperature sensor, and LCD display.

Table 1: Bill of materials (BOM) for the temperature sensor system

ITEM NO	QTY	DESCRIPTION	REFERENCE DESIGNATOR
1	1	Liquid Crystal Display (LCD), LCD1602	U1
2	1	500 - 1MOhm potentiometer	P1
3	1	220 Ohm resistor	R1
4	1	Single pole push button	PB1
5	1	6.6kOhm resistor	R2
6	1	DS18B20 temperature sensor	U3
7	1	4.7kOhm resistor	R3
8	1	ESP32 microcontroller with WIFI chip, ESP32-WROOM-DA	U2
9	1	Single pole double throw switch, 7000 Series	S1
10	1	ZVNL110A MOSFET	U5
11	1	1kOhm resistor	R5
12	1	10kOhm resistor	R4
13	1	LM7805 ACT linear voltage regulator (5V)	U4
14	1	16.5x16.5x10 cm 3D printed hollow box	NA
15	1	16.5x16.5 cm 3D printed lid	NA
16	2	3cm long screws with 0.5cm diameter	NA
17	1	3 pin keyed connector (both male and female)	NA
18	AR	26 AWG wire	NA
19	1	8x5.5cm breadboard	NA
20	AR	0.5cm diameter heatshrink	NA
21	AR	Electrical tape (black)	NA
22	AR	Hot glue	NA

Notes: AR stands for 'As Required'

1.4 Hardware for Virtual Button Design

For our microcontroller, the ESP32-WROOM-DA, the output voltage on the pins was 3 V. This voltage value is not ideal when powering on the display, the LCD1602. The display is dim when powered by the 3.3 V pin of the ESP32 so we decided to implement a LM7805 voltage regulator to create an output voltage of +5V that would power up the LCD display with a similar voltage level to when the LCD is powered up by the push button.

The LM7805 voltage regulator is a 3-pin device that operates to keep a fixed output voltage of +5V (+-0.1 V). For the 3 pins, one is connected to an input voltage, one is connected to ground and the last pin is designated to be the output pin. This output pin, when supplied with the appropriate input voltage, is fixed at a value of 5V, while the pin connected to ground helps dissipate the excess power. For our design, we hooked up an external 9 V battery to the input of the LM7805 voltage regulator. We did this because we wanted to use an external supply source to help aid in this function, but we still wanted the display to run on 5 V so it appeared to be running on 5 V similarly to when the push button is held.

To create the software interface with the LCD, we decided to use a transistor to implement the logic of an output pin coming from the ESP32. We implemented the ZVNL110A DMOS FET with the voltage regulator to make sure that when the virtual button is pressed, the voltage at the gate of the FET will be high meaning that the output of the voltage regulator will be getting to the supply rail of the LCD. When the virtual button is not pressed, the voltage at the gate of the FET will be low meaning that the output of the voltage regulator will not be at the supply rail for the LCD. We decided on going with a DMOS FET since it was convenient and easy to use. Our implementation for this virtual button can be seen in the schematic in figure 2. It is worth noting that the ESP32 IO pins are open drain which led us to the design decision of pulling the gate high by default and pulling it low when we wanted to open the MOSFET.

1.5 Software Design

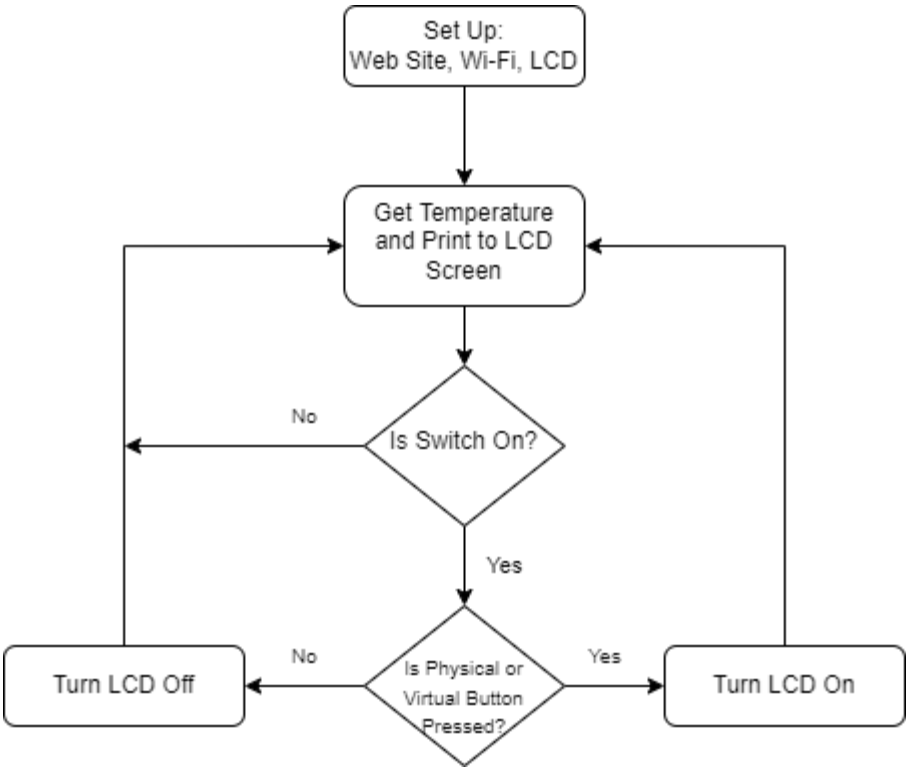

Figure 3: Software Structure

Software Design Description:

The software of the system was run on a single microcontroller, the ESP32-WROOM-DA, running a single uno file. Our uno file saved the HTML for the webpage as a string literal which it then used to in junction with the client.println() function from the EspAsyncWebServer library to write the HTML code to a hosted webpage on a local IP address. After writing the HTML and associated JavaScript, the main loop() of the code would be executed which would monitor for signals from the remote web page and from the local button interfaces. These signals allowed us to communicate to the user about changes in the system from the remote interface or the local temperature probe.

Website Design:

The website was designed using standard HTML and JavaScript with one imported library, the base highcharts.js package. With this the layout of the webpage can be seen below in Figure 3, the design is centered on the graph and the temperature that it displays. Below the graph the user will find buttons and input boxes where they can do the following; turn on and off the LCD by holding the Turn on LCD button, change the graph units to Fahrenheit, enter a number to send alerts to, change the minimum and maximum temperatures that trigger that text message alert. All the buttons have JavaScript functions that are assigned to them to handle their individual use cases. When communication back to the main loop of the Arduino code was needed an HTTP Get request which would be received on the Arduino and processed. The graph was updated though an interval function running every second that would execute a HTTP Get request that would look for new values for temperature.

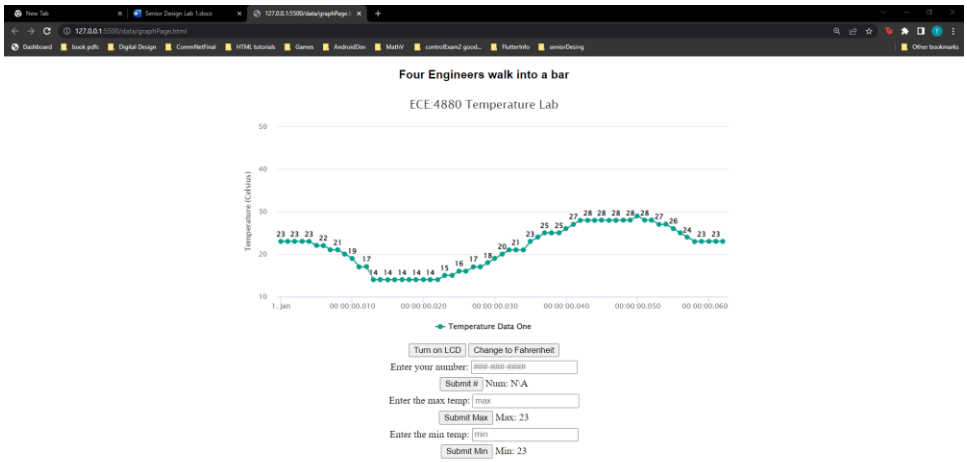

Figure 4: Website layout

Arduino Code Description:

The code is written in C++ using Arduino IDE. It is a single file containing HTML and JavaScript that's been converted to C++ for the website layout and function. In the set-up function, the GET and POST calls were linked to the JavaScript action using ESPAsyncWebServer.h's method server.on(). LCD, button, and switch pins were initialized. Wi-Fi connection was established using WiFi.h. After the connection was established the website server was started. In our main loop, the temperature from the probe was written on the LCD screen, then we checked the states of the button and switch. If the switch status was HIGH we then checked the button status. If the switch was LOW, we did nothing else. Inside switch HIGH, if the button was HIGH we turned the LCD on. If the button was LOW we turned the LCD off. To turn the LCD on and off we used a function that wrote pin 19 to be HIGH or LOW. If the pin was HIGH it would trigger the hardware to set the LCD display to on, if the pin was LOW it would trigger the hardware to set the LCD display to off. When the temperature of the probe was being converted to a String we checked if the value was 0 to indicate the probe was unplugged, this would display an error message to the LCD, and whether the temperature was within the ranges that required a text to be sent. If the temperature required a text, the messages were sent using HTTPClient to POST a packet to our IFTTT applet which utilized a webhook and SMS. If IFTTT received a post with a message and number, it would text the number the message from a pre-established phone number.

2. Design Process and Experimentation

When finalizing the final design of this system, there were a lot of design choices that needed to be thought out, tested, and agreed upon. When first testing the WIFI capabilities of this project with the Arduino UNO, we realized it would be much simpler and cheaper to use an ESP32 instead. An ESP32 would be capable of doing everything we need the UNO to accomplish in addition to providing us with access to an external website. The ESP32 also came with a large variety of libraries that worked in unison much more cleanly as they were designed knowing the Wi-Fi module in use. For example, the

ESPAsyncWebServer contains nearly all the libraries we needed to complete the remote requirements of this project, making the declaration and implementation of its components easier as they all use the same syntax. Another aspect of this system that was not required but we felt necessary for user efficiency was to use a potentiometer instead of a constant voltage contrast on the LCD display to enable the adjustment of the display level.

To enable the website to communicate with the LCD display to turn on or off the data, we initially decided to use an IO port that would simply supply the 5V needed to turn on the LCD from an Arduino UNO. When switching to an ESP32, we realized that the output voltage of each IO port was only 3V, which was not enough to power the LCD. With this issue, we had many viable solutions come to mind. Our first approach was to use a MOSFET attached to the power rails to turn off or on the LCD power using the IO port at the gate. When testing this method, we found that there was a 2V voltage drop across Vds which led to an insufficient amount of voltage needed to power the LCD. The next design we chose to test was to use an OP AMP to boost the level of the IO port's output voltage as it would be directly connected to the LCD's power. Our issue when testing was due to the need for a negative supply rail to power the OP AMP, which we did not have access to without using additional hardware. With these two methods tested and rejected, we finalized our design by using a linear voltage regulator in addition to a MOSFET with its gate tied to 5V and the IO port. When testing this method, we were getting the results we desired where when we set the IO port high, the LCD would stay on and when we set it low the LCD would turn off. One issue we faced through this design decision was strange behavior with the MOSFET holding a voltage value when the IO port would change between states. We were not able to troubleshoot this issue and this issue appeared to arise sporadically.

When first designing our system, we decided to use a perf board to enable the mechanical stability of the system when performing a drop test. As we were testing our circuit that was soldered to the perf board, we noticed that there were issues with the connectivity between the LCD display and the traces on the board. We believe that the main root of this issue was that the heat on the soldering iron was set at too hot of a temperature which had damaging effects on the overall system. Because of this concern, we decided to move our circuit back to a breadboard and to stabilize the wires with hot glue. When testing our circuit on the breadboard, we found there were no issues related to sparce connectivity.

In the preliminary stages of design for this system, we initially decided to use a connector that required crimping of wires to pins for the temperature sensor. We instead decided to use a simpler connector with wires already attached. The reason for this was due to the simplicity of stripping the wires of the temperature probe and the wires of the connectors, soldering two wires together and placing heat shrink over top instead of going through the process of crimping each wire to a pin.

3. Test Report

Table 2: Test report for the temperature sensor system

Test	Test Procedure	Expected Outcome	Pass /Fail	Measured
Thermometer Cable Length	Unplug the temperature sensor cable and	The length of the thermometer cable should be 2.0 +/- 0.1 meters.	Pass	Our cable length was 2.0 meters.

	measure its length with a measuring tape.			
Thermometer sensor system's box construction	Turn the box upside down and confirm the system still functions as expected. Then drop the box off the workbench onto the ground and verify its ability to function as expected.	The box of the system shall be enclosed and when dropped from the workbench the system shall still operate normally with the exception that the thermometer cable becomes unplugged. The box can also be turned upside down with the same result.	Pass	Our system worked without error after turning the box upside down and dropping it.
Sensor becoming unplugged	Unplug the sensor then plug it back in. Verify the system works as expected without using user intervention.	When the sensor is unplugged then plugged back in again, the system should begin normal operation without user intervention.	Pass	Our system began normal operation when the sensor was unplugged then plugged back in.
Temperature data display with switch off	Turn the switch on the box off. Verify that temperatures are not being displayed to the LCD or website graph.	When the switch is off, the thermometer system cannot display temperatures and temperature data is not available from the internet.	Pass	Our system did not display temperatures when the switch was off.
Temperature data display with switch on	Turn the switch on the box on. Verify that with the push button held the LCD displays the correct temperature from the temperature sensor in degrees Celsius and can visually be seen along with no noticeable (under 20 ms) delay.	With the switch on, when the pushbutton is pressed and held, a display on the LCD shows the correct temperature in degrees Celsius with no noticeable delay and shall be readable in normal lighting conditions.	Fail	Our system did not display temperatures to the LCD, but we were able to turn the LCD on and display random symbols.
Display when sensor is unplugged	Unplug the temperature sensor. Verify that an error message is sent to the website and to the LCD saying the sensor is unplugged.	When the temperature sensor is not plugged into the box, the display shall notify the user that there is an error.	Pass	Our system displayed to the user ‘UNPLUGGED’ on the LCD when the sensor was unplugged and sent the same message to the website.

Website temperature display with switch on	Turn on the switch. Verify that real-time temperature data in degrees Celsius (or Fahrenheit) is updated once a second on the website.	With the switch on, the real-time temperature in degrees Celsius or Fahrenheit (controlled by the user) is updated once a second on the website.	Pass	Our system updated the graph on the website once per second.
Website temperature display with unplugged sensor	Unplug the temperature sensor. Verify an error message appears instead of real-time temperatures being displayed on the website.	If the sensor is unplugged, an error message shall appear instead of the real time temperature.	Pass	Our system sent an error message to the website when the sensor was unplugged.
Website temperature display with switch off	Turn the switch off. Verify a message saying “no data available” is displayed on the website.	If the switch is off, a message "no data available" should appear.	Pass	Our system displayed to the website “no data available” when the switch was unplugged.
Website virtual button	Press the virtual button on the website. Verify that the LCD can be turned on and off in less than 1 second.	By the user, the temperature display on the box can be turned on or off from the website with a button response in less than 1 second.	Fail	Our system was not able to turn the LCD on and off via a virtual button.
Website temperature graph with the switch on	Turn the switch on and connect the computer to the website. Verify a graph of the past 300 seconds of temperature appear within 10 seconds of the start of the software for the website.	With the switch on and the computer connected to the internet, a graph of the past temperatures spanning 300 seconds prior shall be present within 10 seconds of the start of the software on the computer.	Pass	Our system displayed a graph on the website containing the last 300 seconds worth of data within the 10 second window.
Website graph's layout	Verify that the graph changes from units of degrees Celsius to Fahrenheit when hitting the virtual unit change button. Verify that the top of the temperature graph on the website corresponds to 50 degrees Celsius and the bottom corresponds to 10 degrees Celsius.	The graph on the website shall have a method to switch between degrees Celsius and Fahrenheit. The top of the graph should also correspond to 50 degrees Celsius, and the bottom corresponds to 10 degrees Celsius.	Pass	Our website was able to switch between units of degrees Celsius and Fahrenheit.

Website graph's scroll	Set up the website to collect temperature data on its graph. Verify that new temperature data points are added to the right side of the graph and the oldest are pushed to the left in a 'scrolling' manner.	The temperature graph shall scroll horizontally with the latest temperature on the right side of the screen and the oldest scrolling left off the screen with new temperatures being added once a second.	Pass	Our website's graph was able to scroll left by adding new temperatures to the right side of the graph.
Website graph's scalability	Using the graph adjustment button, verify the size of the graph is scalable with the mouse.	The physical size of the graph shall be scalable with the mouse.	Fail	Our website did not scale the physical size of the graph; however, we implemented a way to change the size of the vertical axis.
Website graph's horizontal axis	Verify that the horizontal axis of the website's graph depicts seconds ago from the current time.	The label of the horizontal axis on the graph shall say "seconds ago from the current time".	Pass	Our graph's horizontal axis displayed seconds ago from the current time.
Website's graph with data missing	Unplug the temperature sensor while taking measurements on the website's graph. Verify that a section of no data points appears on the graph.	When there is data missing, the graph shall have an obvious indication of no data (a large section of zero data points).	Pass	Our graph graphed a straight line of null data when there was no data available.
Website graph's ability to scroll with no data points	With the temperature sensor unplugged, verify the website's graph still scrolls to the left. Plug the sensor back into the box. Verify that the graph plots the correct data points while still scrolling to the left.	When there is no data from the temperature sensor, the website's graph shall continue to scroll, and the data shall be shown as missing (no data points). When the data resumes, the graphing of the points shall start again with plotting real data points.	Pass	Our graph continued to scroll once every second.
Website graph's ability to update the graph with switch being turned from off to on	Turn the switch off. Then turn the switch on and verify the website updates its graph with the correct data points within 10 seconds of flipping the switch.	With the computer on and the switch off, when the switch is turned on the website shall have its graph updated with correct data points within 10 seconds of the switch turning on.	Pass	Our graph was able to update with the correct data within 10 seconds of turning the

				switch on from off.
Text message for upper temperature limit	Turn the switch on and activate the website. Expose the temperature sensor to a temperature greater than 25 degrees Celsius and verify that a text message is sent to the specified number describing the temperature exposure.	When both the computer and switch are on, a text message saying "temperature probe detects a temperature above the 25 degrees Celsius limit" shall be sent to a specified phone number whenever the temperature exceeds 25 degrees Celsius.	Pass	Our website sent a text message to a phone saying that the temperature was above the specified range.
Text message for lower temperature limit	Turn the switch on and activate the website. Expose the temperature sensor to a temperature less than 23 degrees Celsius and verify that a text message is sent to the specified number describing the temperature exposure.	When both the computer and switch are on, a text message saying "temperature probe detects a temperature below the 22 degrees Celsius limit" shall be sent to a specified phone number whenever the temperature drops below 22 degrees Celsius.	Pass	Our website sent a text message to a phone saying that the temperature was below the specified range.

4. Project Retrospective

4.1 Project Outcome Summary

The components of our project were well built and performed well at the tasks they needed to complete but when we tried to put the pieces together, we struggled to get components to communicate well together. For example, the website and ESP32 communicated well together, and we were able to send information back and forth cleanly, however when we attempted to display that information to the LCD, we had difficulties getting a clean display. We struggled with debugging the project and finding the root problem causing our LCD to not display information consistently. The display had worked fully functional when we first wired it up, it was one of the first tasks we worked on, but once we put the whole system together, we couldn’t figure out why we were running into this problem.

Even though we ran into some problems with successfully getting the LCD to display the temperatures, we still managed to successfully implement most of the project's requirements. One of the most important requirements we were tasked with was to have our microcontroller successfully transfer data

to the website. One big issue we had was not being able to connect to the internet using an ESP8266. We were struggling to implement this into our project and found out that there may be an easier way to connect our microcontroller to the internet. We changed our design from using an ESP8266 to an ESP32. Making the decision to change to the ESP32 was easy after having some struggle with the ESP8266. Progress was fast and we were able to successfully implement our internet connection after we decided on the ESP32.

Overall, we were able to successfully meet most of the projects' required criteria. We made many design choices in the development process of the prototype. We believe that this lab was a great project that made each of us research and learn about how to integrate all these systems together all on our own without much help from the professors. While working on this project we all developed better documentation skills, which is necessary for a team of engineers to do when designing hardware and software for a prototype.

Pictures of the actual temperature sensor system working can be seen in the appendix.

4.2 Factors that held us back

We could have made the testing process faster by testing the website hosting and communication prior to the board being wired by just testing with simple debugging statements to see if values were being set at the correct times. We could have tested this using a simple LED to see if signals were being sent from the MCU at the correct time. Instead of this we were waiting to test the software until sizable portions of the wiring were complete which led to uncertainty in where errors were coming from. In the future our team will implement a more standard testing procedure and provide documentation about what features of the project have been tested and verified before handing the project over to another member.

One major setback regarding the hardware development of the system was implementing the circuitry for the virtual push button. The overall circuitry of the system was all designed prior to being built. After we designed how we wanted to build the circuit, we built the whole circuit all at once without stopping to check if each subsystem worked correctly. This led to many issues having to be troubleshooted at the same time. Most issues were slight wiring mistakes, but these minor setbacks could have been avoided if we took our time and checked our circuitry as we built it.

The ESP output pins supply a voltage of 3 V’s. This value is not high enough to power on the LCD. This caused issues when dealing with the virtual button press since we needed to design a voltage step up circuit that could properly turn on the LCD. This setback was detrimental to our success in developing the virtual button. This would not have been much of a problem if we had used an Arduino, which has output pins that supply a voltage of 5 V’s but adding the Arduino to our design would have caused more complexity to the hardware and the software. We took this into account when we first chose to use the ESP as our microcontroller.

4.3 Team Member Roles

Jacob Sindt:

Designed the software and hardware necessary to interact with the temperature sensor. Helped troubleshoot all hardware issues related to the system. Integrated the push button, switch, potentiometer, and LCD into the mechanical box. Soldered the perf board in early development. Wrote documentation for aspects that occurred throughout the lab.

Ryan Griffin:

Purchased necessary hardware using the senior design $100 credit card given to each team. Kept a current balance list for the credit card. Helped design the hardware and integration between multiple systems. Contributed to the soldering of the perf board in early development. Researched about the temp probe and alerted the probe to fit the requirements given for the project including creating pin connectors and fitting the correct length.

Emma Taylor:

Created the git hub to manage code between users. Wrote the base structure of the ESP32's code to allow for other members to add modular function calls. Managed pin mapping and monitoring the state of input pins and writing the correct state to output pins. Assisted in connecting wired components to the ESP32 and adding the website to the ESP32's code. Wrote all code to trigger and send SMS.

Tony Longo:

Designed the website and worked with Emma to setup the hosting of the website with the C code on the ESP32WROOM. This consisted of using HTML and JavaScript to control the UI and make sure information was being passed from the website to the server successfully. Additionally, it required learning a new library, HighCharts.js to make a more visually appealing graphical display of the temperature data.

4.4 Workload Distribution

The main dynamic of this project was divide and conquer in the beginning and then we slowly came together to fit our separate parts into one project. Emma managed the main code for the project and helped to fit the pieces into it. Tony focused on the website, Jacob and Ryan worked on the hardware, case and documentation.

4.5 Project Management Process

During our project, multiple management processes were implemented to set the goals of the development of our prototype. For the hardware portion of our project, we used a waterfall development approach, while for the software portion of our project, we used an agile development approach.

Waterfall (Hardware development):

The waterfall approach uses fixed milestones that can easily be predicted before they are designed. We know what we want and design and the possible approaches to how to get there. This is the moment where we make a design choice and test it until it fits our fixed criteria. This approach was used most predominantly in the designing of the hardware of the project. Our development was predictive, we had a predefined set of milestones. Our first set of milestones was to create each individual system separately and our next set was to integrate each system together one by one until the hardware functioned as 1 cohesive unit.

Agile (Software development):

The software was developed in an agile-like process where we focused on a modular function that we wanted to implement and would plan, develop, and test it before moving onto the next function. For example, the website was designed initially just to work as a display where the user could see a graph with test data. After the visuals of the graph were implemented, we went back and added user inputs. This process was applied across the entire software.

4.6 Gantt Chart

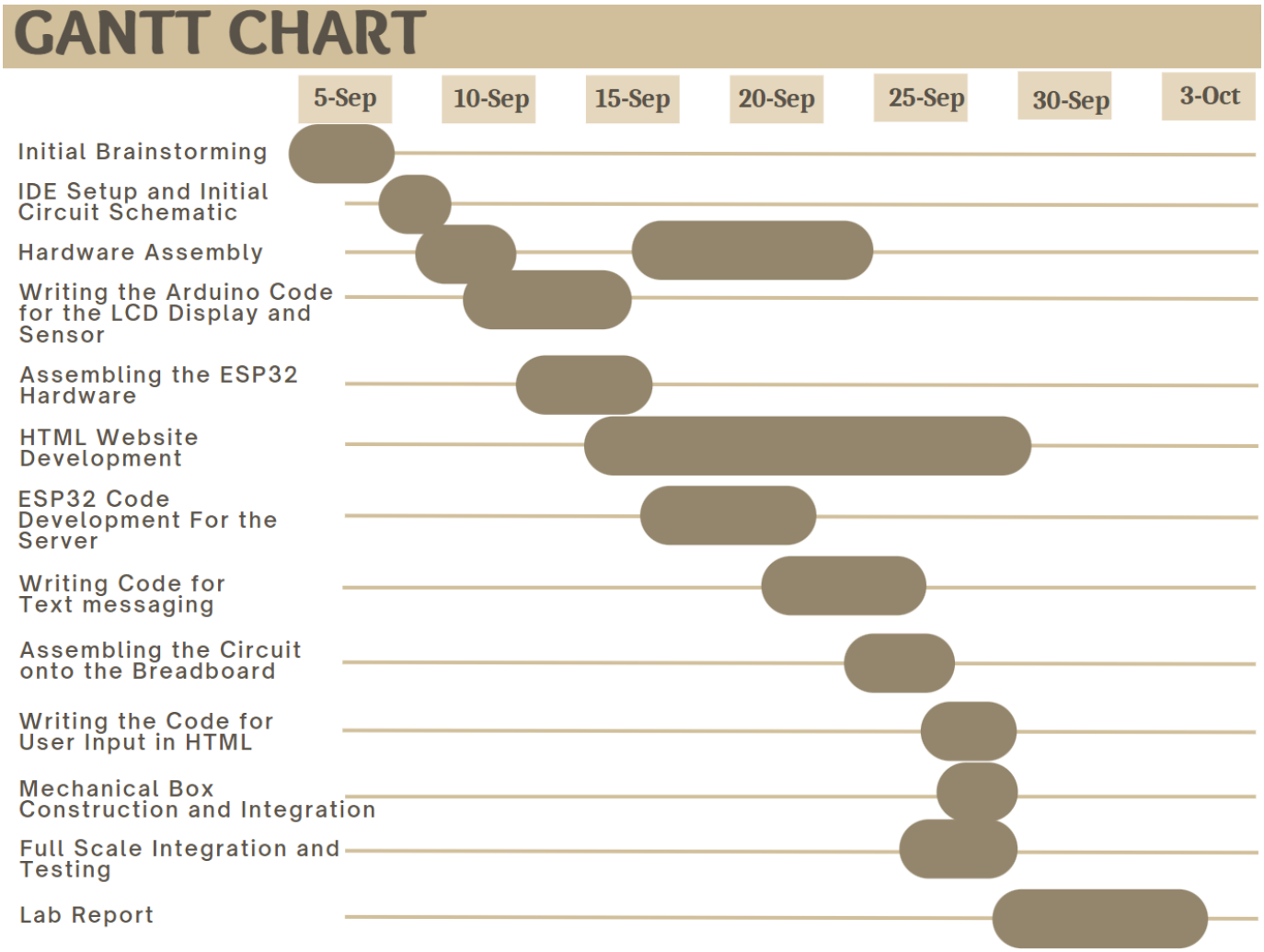

Figure 5: Gantt chart to track our progress throughout the lab.

5. Appendix and References

Flow Chart Reference:

This is the site we used to create our flow chart in Figure 3.

Libraries:

HighCharts:

HighCharts JS: HighCharts.js is an advanced graphing library that can simplify the design and display of simple bar graphs to complex stock charts. Additional information on how to implement the base library can be found here.

ESPAsyncWebServer:

EspAsyncWebServer is a comprehensive library that contains most of the packages needed to host a web server from an ESP32.

AsyncTCP:

AsyncTCP is a base library for EspAsyncWebServer to allow it to have an easier time connecting to an ESP32's MCUs for a network environment

LiquidCrystal:

LiquidCrystal is a base Arduino library that allows for easier communication with alphanumerical LCDs. This library can be found on the arduino reference website and is based on the Hitachi HD44780, and can work in with 4 or 8 bit mode.

WebServer:

WebServer is a library contained in the arduino-esp32 library. Used to use functions like server.begin() and initiate HTTP requests

WiFi:

WiFi is a library contained in the arduino-esp32 library. It is used to connect to local Wi-Fi networks using the standard name and password connection.

HTTPClient:

HTTPClient is a sub-library of arduino-esp32 which can be used to control client behavior that are hosted through an ESP32.

DS18B20:

DS18B20 is a Arduino library for the temperature sensor we were using. With this library discover and read attached sensors with a few lines of code.

HTTP_Method:

HTTP_method is a library that helps to identify key terms in the arduino-esp32 library so that its libraries can communicate more easily.

Tutorials:

ESP32/ESP8266 Plot Sensor Readings in Real Time Charts – Web Server:

This tutorial was used as a starting point to learn about HighCharts and how to send/receive data from the Arduino and the webpage.

Hardware Datasheets:

DS18B20:

DS18B20 is a programmable resolution 1-wire digital thermometer that provides 9-bit to 12-bit Celsius temperature measurements. The DS18B20 was the thermometer device we were using to read temperature values.

LCD1602:

LCD1602 is a “standard” 14-pin liquid crystal display that is Hitachi HD44780-compatible. We used the LCD1602 as the display for the box of the project. When the switch was “on” and the button was pressed, the LCD would display the temperature read from the probe.

PUSH BUTTON SWITCH:

The generic datasheet for the push button used in our temperature system is located here. The push button was used to interface with our system. The pushbutton helped the user turn on the LCD screen under the right test criteria.

LM7805 Voltage Regulator:

The LM7805 +5V voltage regulator is a 3 terminal positive voltage regulator. We used this in conjunction with our virtual push button to power on the LCD when the virtual button is pressed.

ESP32-WROOM-DA:

ESP32-WROOM-DA is a microcontroller that includes Wi-Fi, Bluetooth, and a Bluetooth LE MCU module. We used the ESP32 to connect to the internet, helping us create the website for the temperature data.

7000 Series Toggle Switch:

The Toggle Switch is a single pole double throw (SPDT) that changes the connections between the three pins by changing the position of the toggle switch. The switch was used as a digital input to the microcontroller and either input high voltage or ground.

ZVNL110A:

ZVNL110A is an N-Channel enhancement mode vertical DMOS FET. The ZVNL110 FET is used as an enable line for the output of the voltage regulator. This FET turns the LCD on when its gate voltage is high.

IFTTT:

IFTTT is a site that creates applets that allow one action to trigger another. We used this to trigger an SMS text message by sending it a JSON object using POST.

Figures:

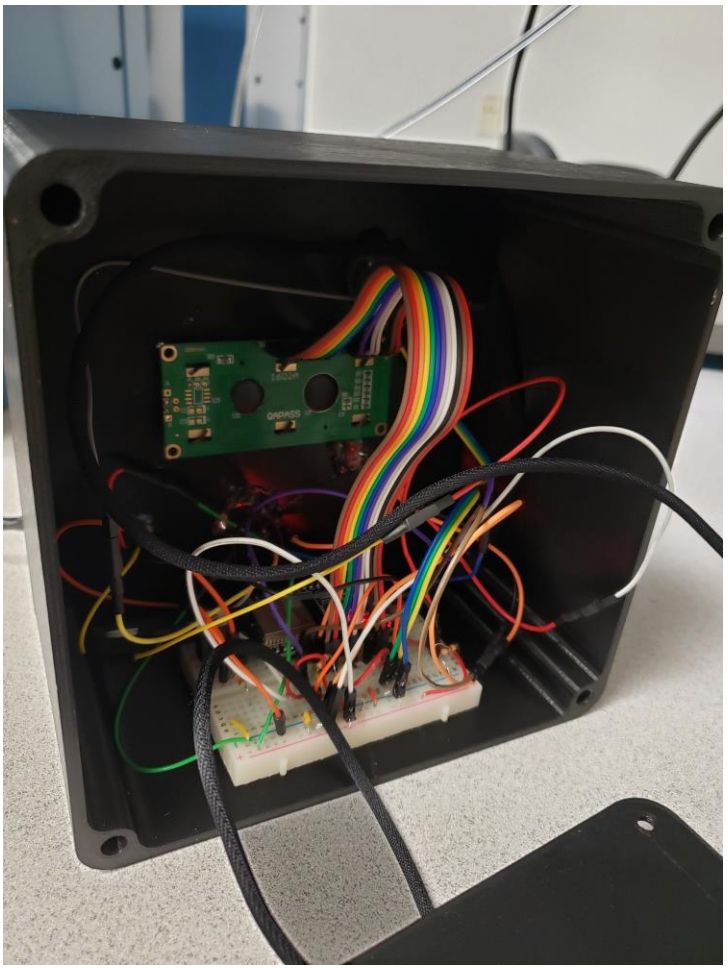

Figure 6: Temperature sensor system wired together and placed inside of the mechanical box.

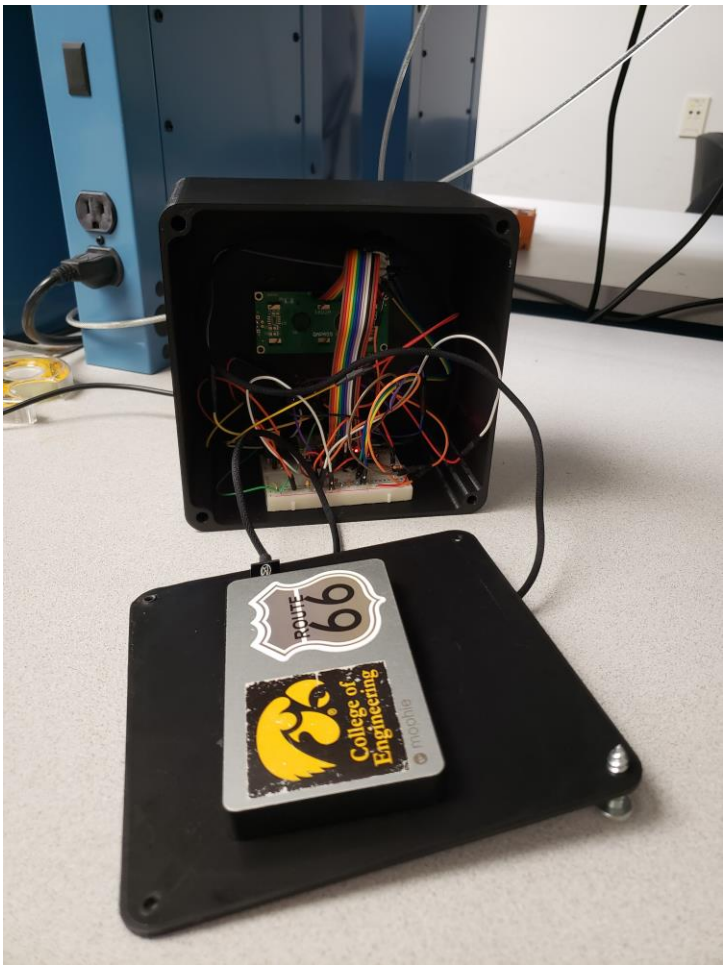

Figure 7: Detachable battery pack supplying power to the ESP32 to power the temperature system.

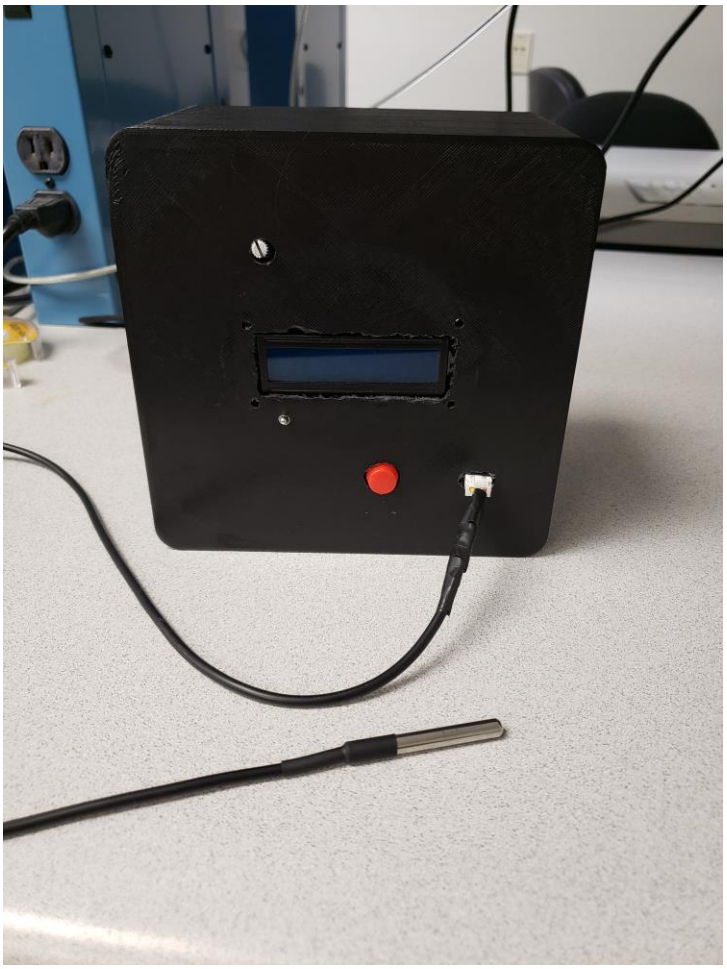

Figure 8: Front display of the temperature system showing the temperature probe plugged into the box along with the interactable button, switch, and potentiometer.

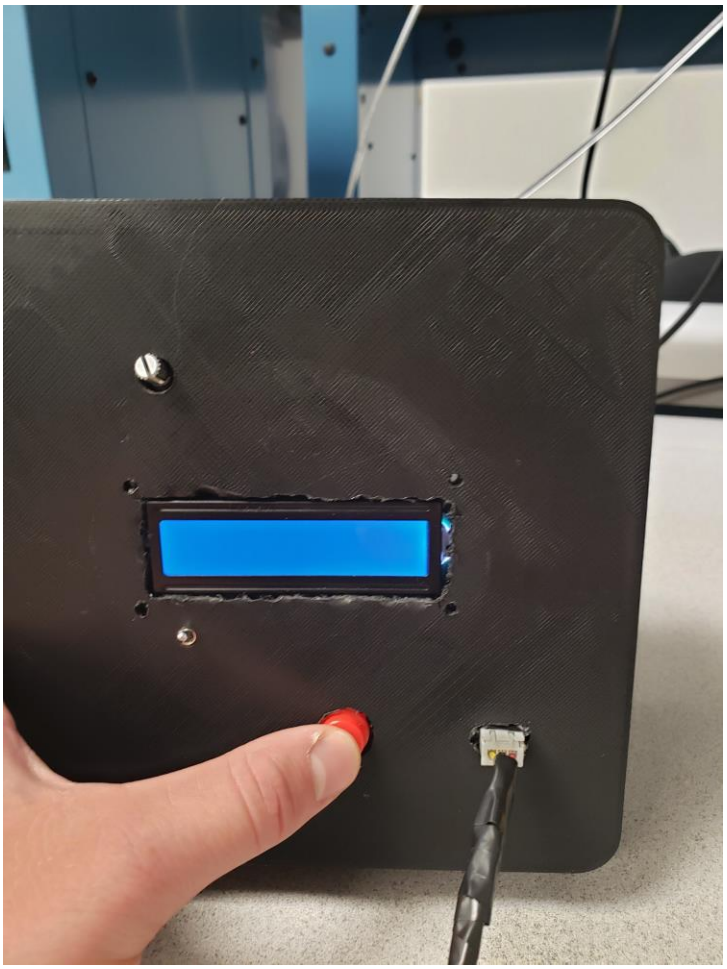

Figure 9: Temperature system powering the LCD by holding in the push button. To the right of the push button is the connector for the temperature sensor probe.