EE153 Final Project Writeup - Team M

Juan Garcia, Paul Wang and Asher Milberg

December 2025

1 Introduction

The goal of this project is to design, build, and program a wireless temperature monitoring probe and deploy it on the Tufts University campus. The node must be able to measure temperature at least every hour and have a timestamp accurate to within 5 minutes. The temperature accuracy must be within 2 degrees Celsius of the ambient air temperature. The probe will transmit the data through MQTT on the Tufts Wireless Network with a specified protocol decided upon by the students of the Fall 2025 EE153 Class. The goal is that the probe will continue to report temperature data for at least 6 months and withstand the weather conditions during that period of time. In other words, it will be able to work without any continued maintenance after deployment.

The major constraints for this project are that the BOM (Bill of Materials) must cost less than 20 dollars per node (assuming a quantity of 1000 nodes) and the system must use an ESP32-C3-WROOM-02 module with a custom PCB.

2 Design Overview

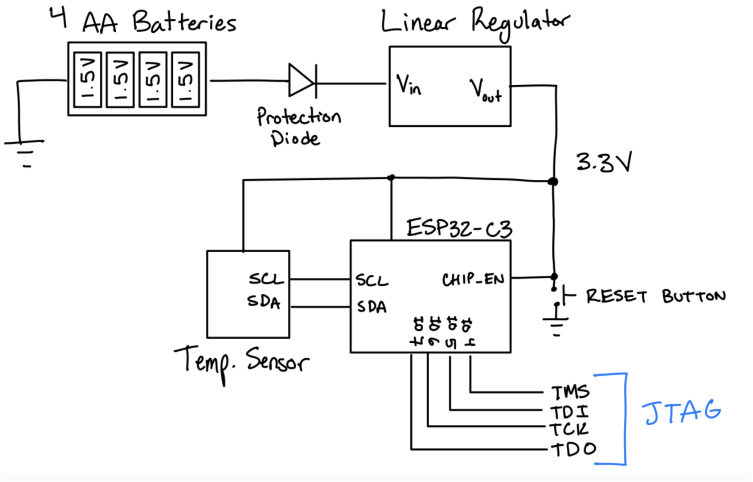


Figure 1: High-level Block Diagram

Pictured above is a high level block diagram of our system. The power comes from four 1.5 volt AA batteries wired in series. This runs through a protection diode that prevents against voltage spikes into an LDO (low-dropout) voltage regulator to step the voltage down from 6 volts to 3.3 volts. This then powers the temperature sensor and the ESP-32, which communicate with each other using I2C. We programmed the board with an ESP-Prog Board (FIGURE CORRECTION: UART was used instead of JTAG).

3 Design Description

The section will go into detail on PCB design, programming (UART through ESP-Prog), software and mechanical enclosure.

Here is a link to the GitHub repository that contains the files flashed onto the ESP-32.

3.1 PCB Design

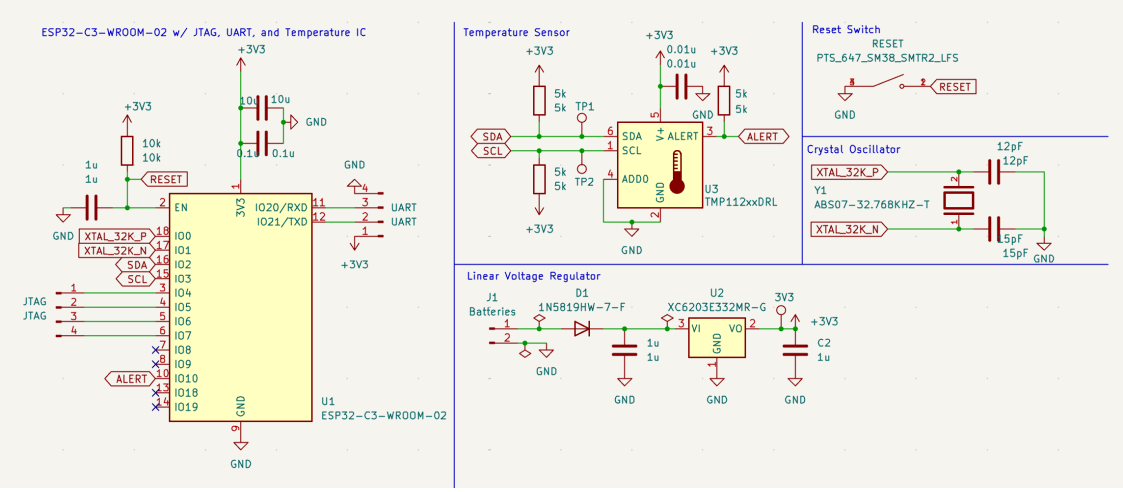


Figure 2: PCB Schematic

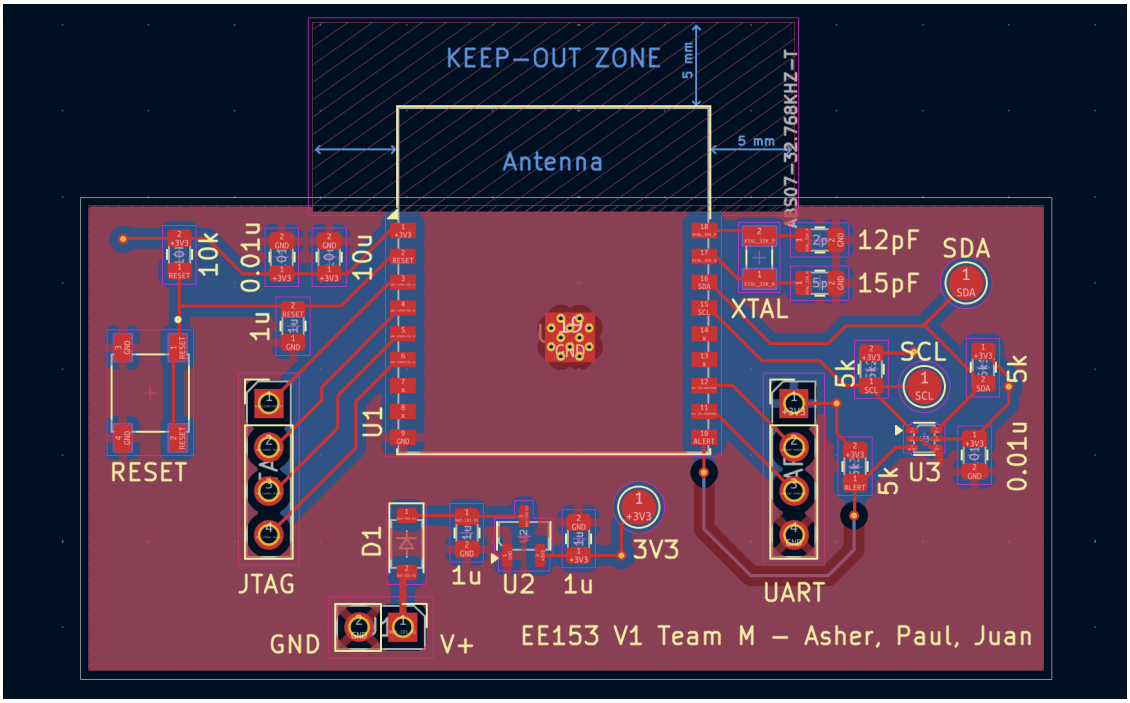


Figure 3: PCB Layout

Key Features:

- I^2C temperature sensor
- UART and JTAG interfaces for programming and debugging
- External 32.768 kHz crystal for accurate low-power RTC operation
- Onboard 3.3 V regulation from a battery supply
- Manual reset circuitry
- Top Plane: GND, Bottom Plane: 3.3V

The PCB layout follows RF and mixed-signal best practices, including a defined antenna keep-out region and short crystal routing.

3.1.1 Power Architecture

Power Input The board is powered by a battery pack connected at connector **J1 (V+)**. A protection diode (**D1: 1N5819**) provides reverse-polarity protection and prevents back-feeding into the supply.

Voltage Regulation A low-dropout linear regulator (**U2: XC6203E332MR-G**) generates the regulated 3.3 V rail used by all digital components. The regulator is stabilized with a 1 μF capacitor at both the input and output. The resulting +3.3V rail powers the ESP32, temperature sensor, and all pull-up resistors.

3.1.2 ESP32-C3 Core Subsystem

The ESP32-C3-WROOM-02 serves as the central processing and wireless communication unit.

Decoupling The ESP32 supply pin is locally decoupled using a 10 μF bulk capacitor and a 0.1 μF ceramic bypass capacitor to mitigate voltage droop during RF transmission bursts.

Reset and Enable Circuitry The EN (reset) pin is pulled up to 3.3 V using a 10 kΩ resistor. A momentary pushbutton shorts EN to ground to allow manual resets. A 1 μF capacitor provides a power-on reset delay, ensuring reliable startup behavior.

3.1.3 Clocking and Timing

An external 32.768 kHz watch crystal (**ABS07-32.768KHZ-T**) is connected to the ESP32 XTAL_32K_P and XTAL_32K_N pins. Load capacitors of 12 pF and 15 pF are used to meet the crystal's specified load capacitance. This clock enables accurate low-power RTC operation during deep sleep modes. The ESP32's internal RTC clock during deep sleep can lead to timing drift due to temperature and voltage variations. An external 32.768 kHz crystal provides much higher accuracy and stability.

To configure an external 32.768 kHz RTC crystal for the ESP32 in PlatformIO, first open a terminal in your project directory and run "pio run -t menuconfig" to launch the ESP-IDF configuration menu. Navigate to "Component config $\rightarrow$ ESP32-specific", then select "RTC Clock Source" and choose "External 32 kHz crystal". After saving and exiting, rebuild and flash your project; the ESP32 will now use the external crystal for more accurate RTC timing and deep sleep wake-ups.

3.1.4 RF and Antenna Layout

The ESP32-C3 module utilizes its integrated PCB antenna. A defined antenna keep-out zone is implemented on the PCB with no copper pours, signal routing, or ground planes. This preserves antenna impedance and radiation efficiency in accordance with Espressif reference guidelines.

3.1.5 Temperature Sensor Subsystem

Sensor Device Temperature measurement is implemented using the **TMP112xxDRL** digital temperature sensor.

I²C Interface The sensor communicates with the ESP32 over an I²C bus using SDA and SCL lines, each pulled up to 3.3 V with 5 kΩ resistors.

ALERT Interrupt The TMP112 ALERT pin is pulled up to 3.3 V using a 5 kΩ resistor and routed to an ESP32 GPIO. This pin can be configured as either a comparator output or an interrupt to signal temperature threshold events. While the alert signal was not used in the final version of this temperature probe, the possibility of using it for better error detection and logging could be explored in future iterations.

3.1.6 Debug and Programming Interfaces

UART Interface A UART header exposes TX, RX, GND, and 3.3 V for firmware flashing and serial debugging.

JTAG Interface A 4-pin JTAG header provides access to the ESP32-C3's JTAG signals, enabling low-level debugging and programming capability.

Problems Ultimately there were issues with using JTAG on the ESP-Prog. By default on the ESP32-C3-WROOM-02, the JTAG interface is only accessible through a USB connection. Using the 4-pin JTAG interface requires burning an eFuse which can only be done by programming the ESP32-C3 with some other tool. Thus, we had to use the UART capabilities to program the firmware and debug the software. In order to put the ESP32 into bootloader mode, we did have to add an additional wire that went to IO8 and IO9 (which correspond to pins 7 and 8 respectively on the ESP32-C3-WROOM-02). IO8 was pulled up to 3.3 V and IO9 was connected to the ESP-Prog's IO0 pinout. The reset button on the custom PCB was attached to the EN pin on the ESP-Prog. Finally, opposed to the traditional way of routing Rx → Tx and Tx → Rx, we found the ESP-Prog only worked when we routed: Rx → Rx and Tx → Tx.

3.1.7 Pinout Table

ESP32-C3 Pin	IO	Function in This Board
1	3V3	Regulated 3.3 V supply
2	EN	Reset / Enable (pull-up + button)
3	IO4	JTAG
4	IO5	JTAG
5	IO6	JTAG
6	IO7	JTAG
7	IO8	Pulled High for UART Bootloader
8	IO9	Pulled Low for UART Bootloader
9	GND	GND
10	IO10	ALERT (Temp sensor interrupt)
11	IO20/RXD	UART RXD
12	IO21/TXD	UART TXD
13	IO18	(Unused / spare)
14	IO19	(Unused / spare)
15	IO3	SCL
16	IO2	SDA
17	IO1	XTAL_32K_N
18	IO0	XTAL_32K_P

To put the ESP32-WROOM-C3 into bootloader mode for flashing, certain strapping pins must be set during reset. Pulling **Pin 7 (IO8)** high (3.3V) tells the chip to boot from the UART flashloader, while pulling **Pin 8 (IO 9)** low (GND) ensures the chip reads the strapping pins correctly during reset. This combination forces the ESP32 to enter boot mode, ready for programming.

3.2 Software

Strategy To maximize power efficiency while staying within the constraints outlined in Section 1, we decided to implement a batched measurement system. The probe wakes up from deep sleep once every hour, takes a measurement, stores it into memory, and goes back to sleep. At the end of each day the probe will send daily measurements to a server using MQTT. This design works because it minimizes the amount of time spent connected to Wifi, which is by far the probe's most power intensive process.

Code Architecture The code uses an internal **state** variable to keep track of what it needs to do every time it wakes up. Before going to sleep, it increments the state variable. Depending on the state, the probe perform one of three processes:

1. Startup (-1)
2. Measure and Sleep (0 to 23)
3. Send Measurements to Server (24)

Startup In order to keep accurate timestamps, the node connects to Wifi and sets the on-board time by using the course-provided **/time/epoch** MQTT topic. This is only done once to avoid spending unnecessary time connected to Wifi. The on-board time will never be synced or verified again unless the node is given a hard reset.

The TMP112xxDRL was chosen in part for its accuracy and for its one-shot power saving mode. To configure the sensor into shutdown mode, the startup state will communicate with the sensor over I2C to set the necessary register values in accordance to its data sheet. This only needs to occur once on startup.

Measure and Sleep When in one of the state values between 0 and 23, the node simply gets temperature from the TMP112xxDRL, stores it into memory, increments the state variable, and goes back to sleep.

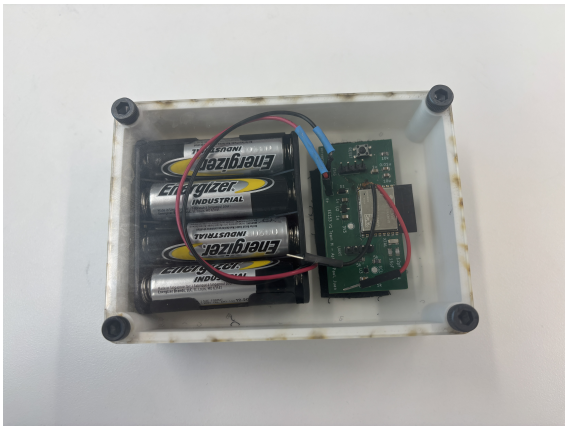
Send Measurements to Server After taking 24 measurements, we know one day has passed. The node formats its stored measurements into one large JSON payload and then connects to Wifi to send it. Once it's done, the node resets the state variable to zero and takes a measurement.

GitHub Repository As mentioned in Section 3, the final version of the code (with explanatory comments) can be found here: link.

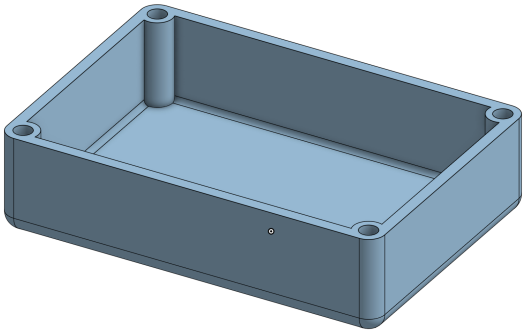
3.3 Enclosure

For the enclosure, we had three main considerations:

1. Weather-resistance and general housing protection
2. Ease of access and repair-ability
3. Accurate temperature readings and WiFi connection



(a) Finished enclosure with PCB inside



(b) CAD design of the enclosure

Figure 4: Enclosure design and implementation

The base of our enclosure was 3D printed with basic PLA filament. The base was measured to hold both the temperature sensor and the respective battery. Both components are held down with Velcro for extra security. For debugging and continued monitoring, we designed a laser-cut clear top made of acrylic. The top and base are fastened with flat head screws and hex bolts.

Our design achieved everything. The materials used for the enclosure accurate reading of outside temperatures, and consistent WiFi-connection. Additionally, the enclosure was really easy to close or open up. The enclosure is **NOT** perfectly waterproof, but it will prevent the majority of possible interruptions. We have placed it with cover for further protection.

4 Current Status

The sensor is currently functioning and deployed on the Tufts University campus. All the sensors functions with its original goal of communicating once every 24 hours is also functional. The temperature IC part is able to read temperature and communicate it to the ESP32 over I2C. The external oscillator is serving as the RTC clock source which enables accurate time keeping in deep sleep. The sensor is able to store data through deep sleep to send batch of 24 samples of data. The sensor is also able to send temperature data through MQTT to the server using the agreed upon protocol.

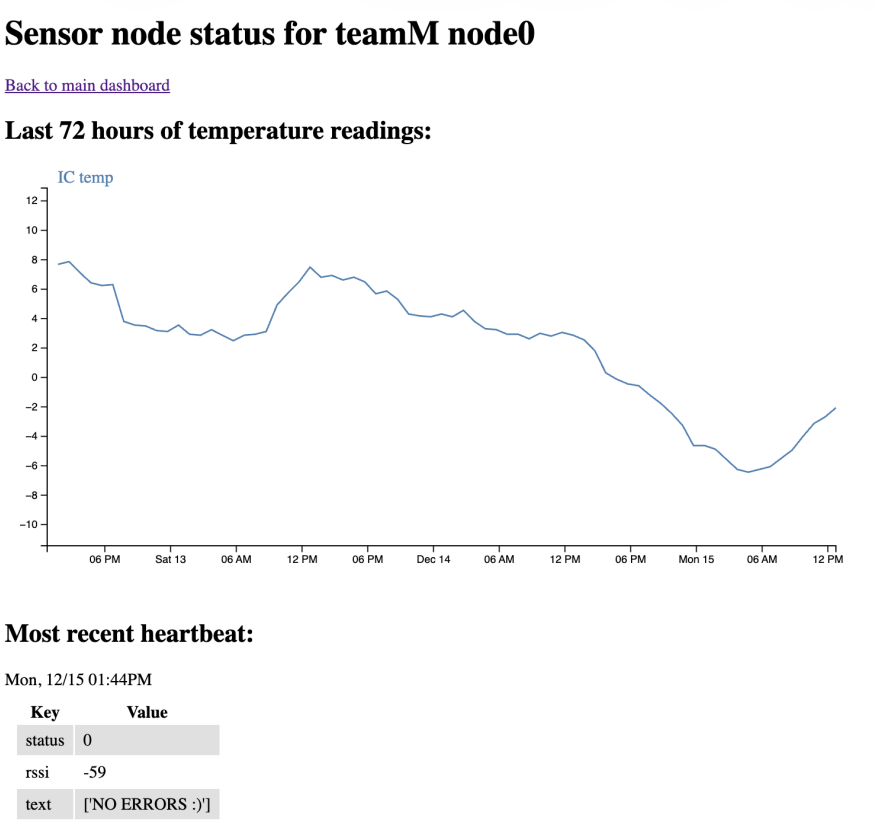


Figure 5: Last 24 Hours of Measurements

The sensor sends data everyday at **1:43 pm EST**. The figure above shows the most recent screenshot of the dashboard, which shows readings have peaks and dips in the temperature around times that would be typical of those characteristics. An infrared thermometer gun was used to verify that the readings shown above are accurate to the environment the probe has been placed in.

5 Deployment

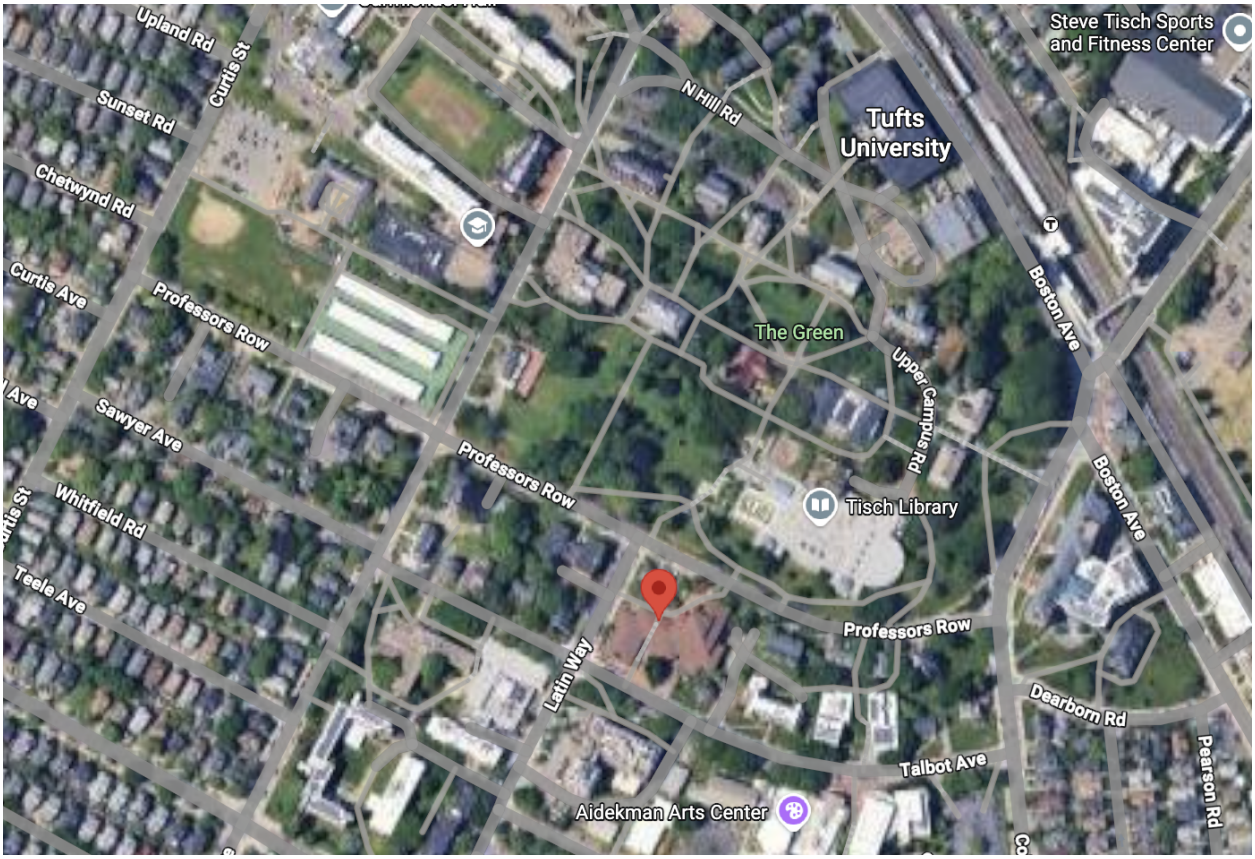


Figure 6: Current Location of Sensor

(a) Zoomed In View

(b) Wide View

Figure 7: On the Ground View of the Sensor

The sensor also has a message inside its enclosure that reads: "Please Do Not Move Me! -Humble Temperature Probe".

6 Lessons Learned

In the end, our sensor turned out functional and is currently reporting data. Time will tell how long the batteries currently in the enclosure will last it and hopefully it will for a while (or at least past when winter break ends). However, there are some things that we wished we had done.

From the start we should have been programming over UART instead of JTAG, as JTAG required additional setup and debugging. We learned only after ordering our PCBs that an eFuse needs to be burned in order for the 4-pin JTAG functionality to be used on the ESP32-C3-WROOM-02.

In addition, the process would have been easier if the system had a dedicated boot pin. We did find work-arounds by soldering jumper wires directly onto the ESP-32 (process described in the Design Description section), but this added complexity and could have been avoided if more research was done beforehand.

With more time, it would have been beneficial to implement robust error detection for Wi-Fi operations such as connecting, sending data, and subscribing to time updates, as well as to add structured error logging and make more effective use of the heartbeat table. A deeper understanding and thorough testing of the crystal oscillator would have improved confidence in timing stability, and additional testing of the TMP112 sensor within the final housing would have helped validate real-world thermal performance.

7 Bill of Materials

Index	Quantity	Part Number	Manufacturer Part Number	Description	Unit Price	Extended Price USD
1	3	1965-ESP32-C3-WROOM-02-N4CT-ND	ESP32-C3-WROOM-02-N4	RF TXRX MODULE BT PCB TRACE SMD	3.28000	9.84
2	3	893-XC6203E332MR-GCT-ND	XC6203E332MR-G	IC REG LINEAR 3.3V 400MA SOT23	0.63000	1.89
3	3	1N5819HW-FDICT-ND	1N5819HW-7-F	DIODE SCHOTTKY 40V 1A SOD123	0.31000	0.93
4	12	11-LR6XWA/BXU-ND	LR6XWA/BXU	BATTERY ALKALINE 1.5V AA	0.42000	5.04
5	2	163-165-ND	163-165	4 AA BATTERY HOLDER	2.99000	5.98
6	3	108-Y97KT23B2NAFPCT-ND	Y97KT23B2NAFP	SWITCH TACTILE SPST-NO 0.05A 12V	0.26000	0.78
7	5	296-47223-1-ND	TMP112AQDRLRQ1	SENSOR DIGITAL -40C-125C 6SOT	1.28800	6.44
8	10	RMCF0805FT10K5CT-ND	RMCF0805FT10K5	RES 10.5K OHM 1% 1/8W 0805	0.01300	0.13
9	10	RMCF0805JT5K10CT-ND	RMCF0805JT5K10	RES 5.1K OHM 5% 1/8W 0805	0.01400	0.14
10	10	1276-1096-1-ND	CL21A106KOQNNNE	CAP CER 10UF 16V X5R 0805	0.01600	0.16
11	10	1276-1029-1-ND	CL21B105KBFNNNE	CAP CER 1UF 50V X7R 0805	0.01300	0.13
12	10	738-CML0805X7R104KT50VCT-ND	CML0805X7R104KT50V	CAP CER 0.1UF 50V X7R 0805	0.05700	0.57
13	10	738-CML0805X7R103KT50VCT-ND	CML0805X7R103KT50V	CAP CER 10000PF 50V X7R 0805	0.05700	0.57
14	10	490-11190-1-ND	GJM1555C1H120FB01D	CAP CER 12PF 50V C0G/NP0 0402	0.08500	0.85
15	10	311-1101-1-ND	CC0805JRNPO9BN150	CAP CER 15PF 50V C0G/NPO 0805	0.04800	0.48

Figure 8: Total: $ 33.93

This total represents the total for the order which includes extras in case components are damaged or lost. The total cost per unit is $ 10.81, which is half of the allotted budget.