Low-Power Wireless Info Display

Aishwarya Rao, Amith Lohitsa, Malvika Shetty, Rishank Nair, Yatian Liu

EECS 473 Fall 2020

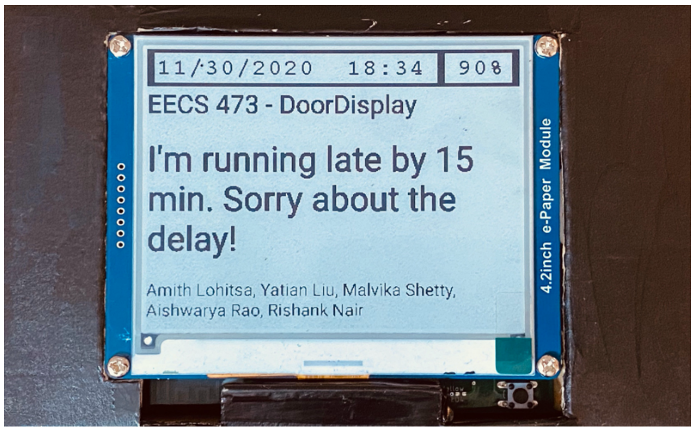

11/30/2020 18:34
90%
EECS 473 - DoorDisplay
I'm running late by 15 min. Sorry about the delay!
Amith Lohitsa, Yatian Liu, Malvika Shetty, Aishwarya Rao, Rishank Nair
4.2inch e-Paper Module

Figure 1: Product at EECS 473 Expo

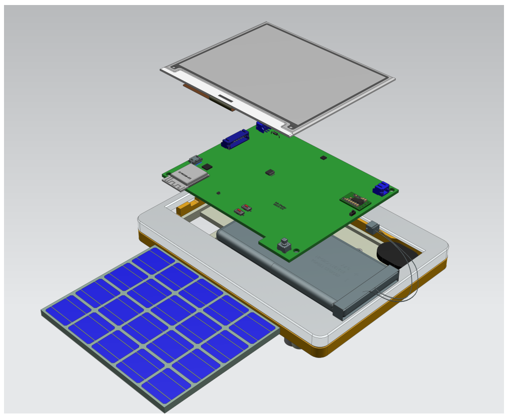


Figure 2: CAD rendering of final product

I Introduction and Overview of Project

In a large university, plans change last minute, rooms are reserved when they were not supposed to be, and students and faculty run around campus for surprising meetings. There is no central communication platform between teachers and students for quick and visible updates. If a lecture hall is taken for a networking event, a professor could send out an email to let students know where office hours have been moved. However, emails can easily get lost in a flurry of mobile notifications, and sometimes students are already on the way and won't be checking their emails. Online forum posts may help, but those too go straight to a crowded email inbox. What if a professor is stuck in traffic and will not make it to his morning office hours in time? A student could arrive and wait for the professor for 30 minutes, continually knocking on the door of an empty office. What if a faculty meeting is running overtime or there was an emergency? There may not be time to leave a note or it may not be physically possible. Anything can happen in this sort of environment.

Currently, a common solution to these problems is mounting tablets on the walls near entrances and doors. This requires large amounts of device configuration and setup, in addition to being quite expensive. They also need to be charged every day or constantly plugged in, which can add up on an electric utility bill and increase maintenance cost. Why purchase an expensive, full-size tablet just to keep it stationary near a door? To make door information displays cheaper and easier to deploy and maintain, we designed and built a low-cost alternative that can be quickly and easily configured from a mobile application. Our internet-connected door display allows the professor to tell his students and colleagues where he is or what is going on when he cannot communicate directly with them through a software platform. Instead of waiting for their meeting to start, the individuals can come to the door, see a new status, and proceed from there. The information on the screen can be updated from any distance remotely in the form of simple text, Markdown-formatted text, and images. This makes our product useful beyond the university setting, like in office spaces, conferences, and primary/secondary schools. It can even be used as an electronic picture frame when important updates are not needed, or a way to share fun facts with individuals who come across the display. Any building with a need to dynamically display information at a low-price tag is a potential home for our versatile product.

Originally, our intent was for this device was to make it low-power and self-sustaining. In other words, with very little current consumption, this product should be able to last indefinitely without user charging. Throughout the course of the project, we felt that this goal was challenging yet attainable. The preliminary math suggested it was possible and we proceeded full steam ahead. Nevertheless, in the days approaching the EECS 473 Expo, we discovered our power numbers were higher than we desired. At the EECS 473 Expo, we were able to demonstrate all three modes of the product (simple text, Markdown-formatted text, images) while powered by battery and solar panel. We felt the demo was a success; we also understood our power numbers still posed a sizable threat to our goal with a deadline approaching. Unfortunately, by the deadline for this project, we were not able to declare the device "self-sustaining." Additionally, an idle power consumption of about 37 mW is higher than we were aiming for. Throughout this report, we will detail the reasons for this outcome as well as discuss our takeaways and what we would like to have done better. However, it is also important to emphasize our accomplishments during this project and how we successfully created a viable product with room for improvement. While adding some additional functionality along the way, we were able to accomplish all but the one (low-power, self-sustaining) fundamental and important design criteria.

II Description of Project

The main intent of our project was to create a low-power solution. Since this project was developed with the intent of complete remote access, we based it on the concept of IoT. The system allows a user to remotely update a display mounted on a door outside an office cubicle or a conference room using a smartphone application. The application communicates with the device via an online web server. The device polls the server periodically through a WiFi connection to retrieve information from the real time database and updates its displaying content accordingly. When it is not polling the server, it enters deep-sleep state to reduce power consumption. Low-power, compact, and inexpensive components and low duty cycles make this device feasible in terms of energy, cost and physical design.

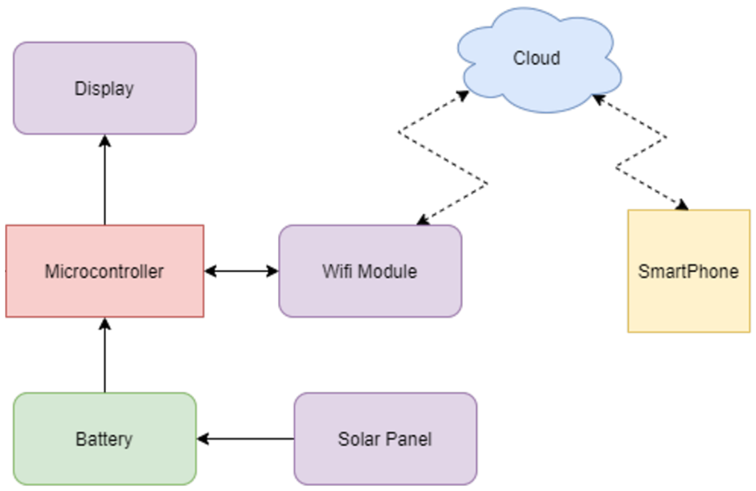


Figure 3: System architecture diagram.

II.1 Design Constraints

1. Several physical design constraints had to be kept in mind while constructing the device. Mounting it on a door or wall meant that it had to be light-weight. Size was also a major deciding factor in how the product would look since it has exposed elements like the solar panel that are vital to its overall functionality. Due to the possibility of an application in which the device is mounted on a transparent door, it would be preferable to make it as compact as possible.

2. With one of our main goals being self sustainability, low power was a major driving factor in the design process. The device is going to be placed indoors at all times where sunlight is scarce. Solar panels can harvest limited amounts of energy under indoor lighting, making low power a large design constraint.

3. With an antenna on the device, special precautions like clearance from other metal components and ample exposure had to be followed.

4. Every project group had the added constraints of budget and time. While we were given a generous amount to spend on our project, keeping it low cost was still a priority. Low cost is also one of our major design goals. The component that was most affected by this constraint was our display. Since E-Paper displays are relatively expensive, price limited the size of it.

II.2 System Architecture

Figure 3 shows our system's architecture. The system is composed of three main elements – the hardware device and its software, a real-time database hosted by Google Firebase servers and a user-end smartphone application. The application can be used to upload images, simple text or Markdown text (advanced formatting) to the cloud. The device polls the database periodically with the period being inversely proportional to the remaining battery percentage when the battery percentage is lower than 50%.

The Firebase real-time database has the form of a JSON file and we can read and write any of its entries. We use three string entries to store information about the type of displaying content, the content itself as well as a timestamp indicating when this update was made. Images are encoded in Base64 strings since the JSON database cannot store binary data directly.

On the display software side, the data, once received by the device, is processed, decoded if necessary, and relayed to the display via SPI. The embedded software incorporates a Courier New font library by STMicroelectronics to render simple text directly on the display. Images and Markdown captures are received as a byte stream storing the value of each pixels and directly relayed to the display. Lastly, we used an open-source WiFi manager made by Tony Pottier to allow the user to easily configure the device's connection to their network from their phone.

On the Android app side, the three different types of displaying content are handled differently. For text, a plain text string is uploaded since the display can render the string locally. For image, since we use an E-Paper display with fixed size, it can only display 1-bit grayscale (i.e. black-and-white) images with fixed resolution and we need to convert regular colored images of arbitrary size to fit the display. We first use the OpenCV library for android to convert a standard 32-bit ARGB image to an 8-bit grayscale image and resize it to match the resolution of the display. White borders are added so the aspect ratio of the image also matches the display. Then, a standard algorithm for converting high bit-depth images to low bit-depth images called Floyd–Steinberg dithering is used to converted the 8-bit fixed size grayscale image to a 1-bit grayscale image of the same size. The values of each pixel of the converted image is then represented using a byte array in a format required by the E-Paper display, and the byte array is encoded as a Base64 string and uploaded to the server. For Markdown, things are a bit easier. We found an open source library released under the Apache-2.0 License called Markwon that can render Markdown source code into Android-style formatted text, and the text can be displayed using an Android component called TextView. The Android API provides a function to capture a snapshot of a TextView, and we just feed the snapshot into the previous image workflow to get a Base64 string and upload it to the database.

The device itself contains an ESP32 chip, a CP2102N USB-UART bridge, an SPV1050 energy harvesting PMIC, a battery charging IC and 3.3 V output switching mode voltage regulator. The ESP32 chip contains a WiFi module that is responsible for receiving, processing and relaying data to the display. Due to limited energy supply from the solar panel, we have incorporated a power harvesting PMIC into the design. This PMIC slowly trickle charges the battery over time. Although the device consumes little current when the it's idle, during updates the power consumption spikes up which is why we have a battery to act as a buffer and supply sufficient current.

We also want to keep track of the state of charge of the battery. To do so, we use the ADC of the ESP32 and divide the voltage to keep it in the permissible range of ADC input. In order to avoid input impedance issues, we have implemented a unity gain amplifier circuit using an operational amplifier. A micro-USB port along with a USB-UART bridge IC allows us to program the device and charge the battery if required.

In the event that a viewer is unsure of whether or not they are seeing the most recent message on the display, they would have the option of pressing a push button that immediately refreshes it for them. The screen displays the image or text that was uploaded, the timestamp of the update, and the battery percentage.

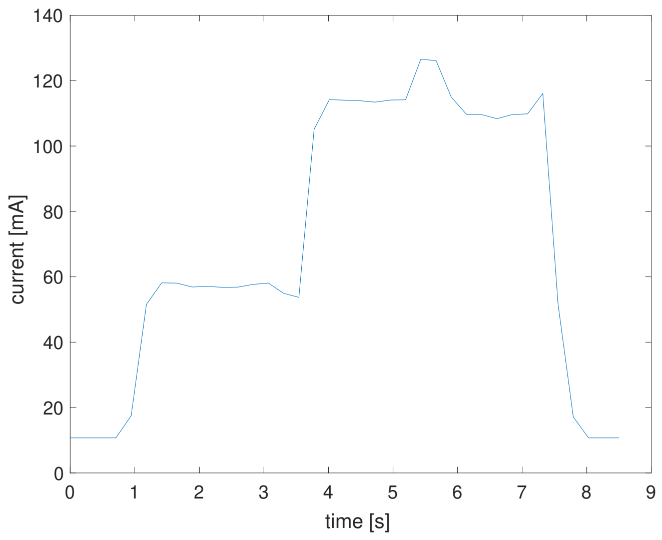


Figure 4: Power consumption curve for one active cycle.

II.3 Power Analysis

Figure 4 According to our final proposal, we had estimated that the power consumed by the display integrated with the ESP32 dev-kit module would be a total of 29.083 Joules per hour. When we conducted a power analysis on our prototype, we found that the device consumed a steady current of about 6 mA and about 180mA during the spikes caused due to the refresh of the display.

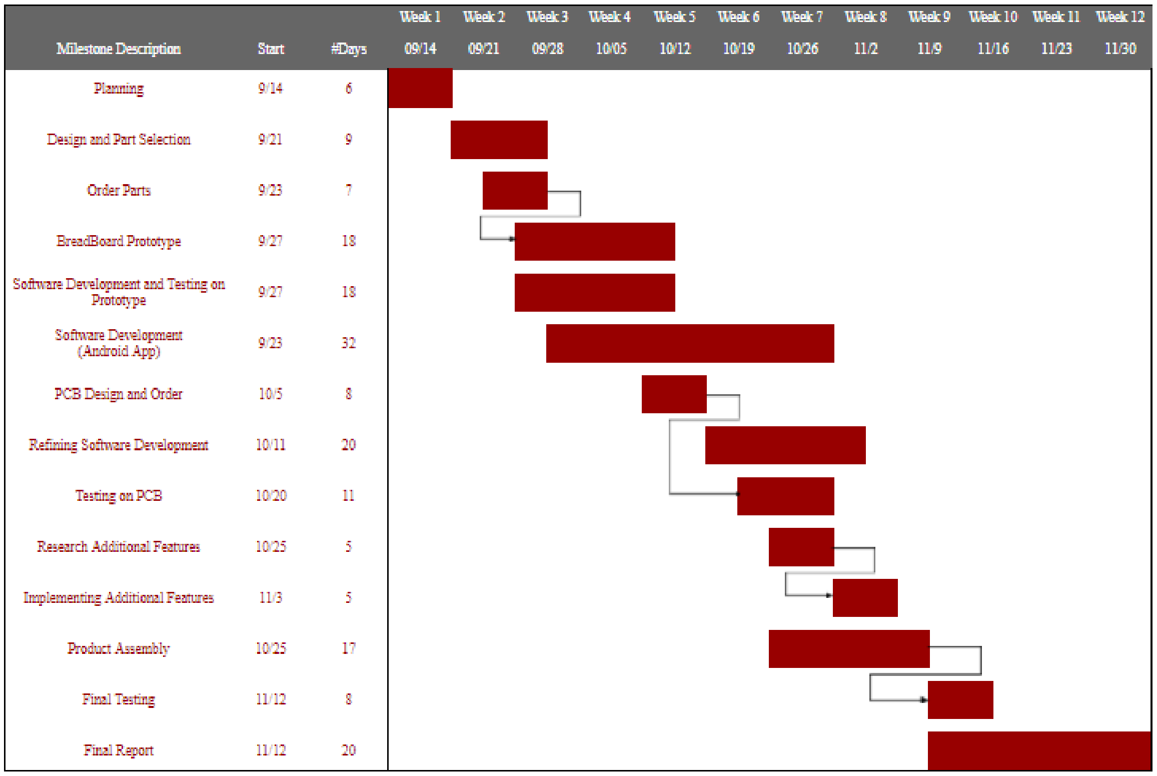
In the final stage of our power analysis, we tested our PCB for power numbers and compared them to that of the prototype and the ideal expected numbers in the proposal. Figure 4 shows how the value of current consumed spikes when the display refreshes. We were also able to obtain around 2mA of current from the solar panel under daytime lighting.

We found that the device consumed a constant current of about 10.7 mA and and 50-100 mA while refreshing the display. Considering the display refreshes every 15 minutes, the calculated average energy consumption was about 150.3 Joules per hour.

We tried reducing the power numbers by having the ESP32 module and the E-paper display enter deep sleep mode. We also use a switching mode Polulu regulator instead of a linear regulator to dial down the battery voltage to the board. Additionally, we used ESP-IDF's RTC GPIO isolation functions to disconnect our GPIO pins from internal circuits, thereby reducing current leakage. When we did this for all GPIO, this actually increased current consumption. Doing this for only the Op-Amp enable pin resulted in successful current reduction.

At this point, we feel that the software consumes as minimal power as possible. The culprit for increased power consumption is likely to be found in our hardware design. One reason why there is a discrepancy between the actual power numbers associated with the PCB and those that were expected could be due to the fact that the length of signal lines on the PCB had to be high enough to connect the spaced out components. With another 2 weeks dedicated only for power analysis we could have probably reduced the power consumption of our product.

III Milestones, schedule, and budget



Milestone Description	Start	#Days
Planning	9/14	6
Design and Part Selection	9/21	9
Order Parts	9/23	7
BreadBoard Prototype	9/27	18
Software Development and Testing on Prototype	9/27	18
Software Development (Android App)	9/23	32
PCB Design and Order	10/5	8
Refining Software Development	10/11	20
Testing on PCB	10/20	11
Research Additional Features	10/25	5
Implementing Additional Features	11/3	5
Product Assembly	10/25	17
Final Testing	11/12	8
Final Report	11/12	20

Figure 5: Gantt chart in the final proposal.

III.1 Milestones and schedule

Figure 5 shows the original Gantt chart in our final proposal. In addition, our milestone deliverables are shown below. The bold items are objectively demonstrable.

Milestone 1 (10/15)

1. **ESP32 module can interact with the e-paper display, including unformatted text and image display.**
2. **ESP32 module can connect to WiFi using hard-coded AP SSID and password.**
3. **The smartphone application is able to send unmodified image and unformatted text to the server. The server-to-display part will be implemented later.**
4. The first version for PCB design has been completed and ordered.

Milestone 2 (11/5)

1. **The server-to-display communication is completed. The user can send text or image to the server and the display should be able to refresh remotely.**

2. **The colored image-to-grayscale image conversion and Markdown formatted text-to-image conversion algorithms are completed.**
3. All hardware peripherals have complete code libraries running on the prototype. The individual functional tests for the libraries on the PCB have started.
4. The final version of PCB is assembled and has passed continuity & isolation electrical testing. Additionally it should have no faulty components or assembly defects.

In the prototyping stage we were quite close to our original schedule and milestone goals. We selected and ordered parts on around Sep. 30, and the embedded software for basic text and image display worked successfully on the breadboard prototype by Oct. 13. Hard-coded WiFi connection and text and image uploading were also working before Oct. 15. Also, for Milestone 2, pulling text from the server worked on Oct. 21, image conversion worked on Oct. 31st, and Markdown conversion worked on Nov. 8.

Since our project is not mechanically heavy and there are not many hardware peripherals, the prototyping stage work is mainly on software side. The main peripherals that need software interfaces are the E-Paper display and the WiFi module. The E-Paper display's official documentation is poor and hard to read, but there are many third-party software libraries available online, so we mainly referred to these libraries when implementing our own library. Amith did this part and it took considerable time and effort. However, for the WiFi module, the ESP32 SoC we used is very well supported. Its official documentation is quite readable, it has a massive built-in software library for WiFi setup and network communication, and there are many good quality third-party projects for it, so getting the WiFi setup and server communication to work did not take too much time and, overall, we stuck with our schedule. For the Android app, Rishank is familiar with Android development, so we got a working interface quite quickly. For the underlying logic of image conversion and Markdown conversion, Yatian found the OpenCV library for Android, a standard algorithm for converting high bit-depth images to low bit-depth images that is not hard to implement, and an open source library that can display Markdown text on Android, so most of our work involved getting familiar with the libraries' interfaces and implementing the standard algorithm. This took about 10 days to finish.

However, our progress falls behind the schedule in the PCB stage. We ordered our first version of PCB on Nov. 3, but we didn't finish testing until Nov. 29. Nov. 3 is the latest day to order first version PCB as required by the staff, so our completion is late but acceptable. We intended to finish the PCB design more quickly, but some problems made it slower. The first PMIC we found for the solar panel was not suitable for our purpose so we found a new PMIC and getting it ordered and tested took some extra time. In addition, we used both a solar panel and a battery to power the system and this brought some potential problems, so we waited for the meeting with Steve to get more suggestions.

Worse problems occurred after the PCB was ordered. We used a chip called CP2102N with QFN28 package to program the ESP32 SoC since this chip was also used on the development kit we bought. However, it turned out that the QFN28 package version of this chip is obsolete and we could not buy it from any major electronic components distributor such as DigiKey and Mouser. In addition, the communication among us was not very efficient and we did not notice this problem until around Nov. 15. We found a UK company that offered this chip and we ordered a number of them on Nov.

16, but for unknown reason there were no updates for the DHL tracking number and the package seems to be lost. As of Dec. 8, we have not received the shipment. To solve this problem, we tried to desolder the CP2102N chip on our development kit and resolder it onto our self-designed PCB. We had no experience of doing it and did not attach small nozzles to the hot air gun to concentrate the heat, so the first two trials burnt the chips. The third trial finally succeeded but that was on Nov. 29, and we have no time for a second version PCB that improves the system's current consumption.

III.2 Budget

The hardware components we used are not expensive and we did not buy many extra things beyond our initial plan so we stayed well in our budget. The detailed list of all the parts for one display is given in Section VII, and the total cost is \$112.73. The most expensive components we bought is the E-Paper display which cost \$33.27 each. Therefore, we only bought three displays but bought a set of five for all the other components, which led to a total cost of $\$112.73 \times 5 - \$33.27 \times 2 = \$497.11$. In addition, we bought development kits initially for prototype testing, and that costs about \$200 in total. As said in the last section, since our CP2102N order was not arriving, we ordered 3 more ESP32 development kits to desolder from, and that added $\$10.99 \times 3 = \$32.97$. Therefore, our final budget is about \$730.08. Additionally, we were granted an extra \$400 for equipment to use outside the lab. We used approximately \$250 of this on a digital microscope, hot plate, hot air gun, solder paste, and flux pen.

IV Lessons learned

Most parts of our project went well. We met all the fundamental and important design criteria in our final project proposal except for the one that the display will hopefully be self-sustaining, and the power consumption is still low enough to give a battery life of more than one month. In addition, we add the functions of measuring the battery's voltage level using ESP32's internal ADC, calculating a corresponding battery percentage and increase the refresh period proportionally when the battery percentage is low. Also, we successfully demonstrated our objectively demonstrable deliverables at both of the milestone meetings. Although the first PCB was successfully assembled later than we would have liked, it worked on the first attempt, which showed our team's ability to design a working PCB. The Android app's interface is aesthetic and easy to use, and the embedded software on ESP32 was solid.

The only part that was less than ideal was PCB component ordering, which is mentioned in Section III. We started ordering the PCB components a bit late, and we did not check whether all the components on our PCB can be bought easily online before we ordered the PCB, and this led to the unexpected problem regarding the CP2102N chip. Also, the shipping of components was more unpredictable than we expected possibly due to the COVID pandemic and this worsened the timing of our component ordering. Remote communication likely contributed to the failure to notice these issues; it was not as efficient as in-person meetings, and such problems were noticed with greater delay.

If we could send a short memo back in time to our group when we first started, we would like to stress on the importance of checking the components' availability before finalizing PCB design, ordering the components early, and communicate more frequently when problems occur. If we could have a working first version PCB on around Nov. 18, there will be much more time for use to test the power consumption of the PCB and possibly make a second revision to get better power numbers.

On the hardware side, Rishank, Malvika and Aishwarya gained experience in designing PCBs, and soldering both through hole and surface mount components. They learned the significance of differential pair signals and the constraints involved in placing them on the PCB. They also learned how to use low-power PMICs. They discovered how to use internet functionality in the smart phone application.

Amith learned a great deal about designing and writing embedded software for tightly constrained applications. Constant optimization and thorough analysis were needed throughout the project to ensure the Info Display application was consuming as few resources as possible. Amith also learned about RESTful API while coding HTTP requests and gained knowledge of another popular embedded systems toolchain and SoC.

Yatian learned a lot about parts selection, Android image processing and PCB power analysis. Reading loads of datasheets are not fun, but it is a necessary skill for an embedded engineer to know the functions of different chips and choose the ones they need. Image processing is an important skill and Android programming is also quite practical. In addition, the knowledge of PCB power analysis is quite useful for future low-power embedded projects.

V Contributions of each member of team

Team Member	Contribution	Effort
Aishwarya Rao	Aishwarya worked on designing the circuit for the hardware portion of the system. During this process, she took into consideration the electrical requirements (voltage, power, etc.) of each component. Once the circuit design was completed, she helped develop the prototype circuit using passive elements, IC's and a perforated breadboard. Overall functionality was tested in tandem with more features being added on the software side of things. Her work included designing, soldering, and programming the PCB once we had a working prototype. She helped conduct electrical testing on the assembled PCB and incorporate the various hardware components to create the final product design.	20%

Amith Lohitsa	Amith implemented the embedded software code on the ESP32. He wrote the E-Paper display library, which provides functions to interact with the display, and the high-level project application. His work included interacting with the real-time database using HTTP requests, integrating the open-source WiFi manager and Base64 encoder/decoder, monitoring battery levels, and designing the frame graphics for the display. Throughout the project, he worked to ensure application resource consumption was as low as possible.	20%
Malvika Shetty	Malvika researched and shortlisted hardware components for the project. She worked on the Circuit Design for the product combining the various ICs, the display, solar panels and other passive components . Once the circuit was finalized, she helped in assembling the hardware prototype and testing its correct functioning. She made the eagle schematic for the PCB design and also manually routed the entire PCB before placing the order for the same. Once the PCB arrived, she carefully soldered the SMD components and tested for bridges. She worked on electrical testing and obtaining power consumption numbers for the product. She then helped assemble the entire product along with creating a casing for the same.	20%
Rishank Nair	Rishank worked on integrating the back end and UI in the smartphone application. He helped design the circuit and conducted electrical and functional testing. He also researched how to work with the PMIC. His work also included laying out the schematic for PCB design on Eagle and soldering both surface mount and through hole components onto it. Once we got the PCB assembled and the additional hardware components like the display and battery connected, he tested the device as a whole for functionality.	20%
Yatian Liu	At the initial stage, Yatian was in charge of comparing and selecting parts. He searched for various microcontrollers with WiFi connectivity and compared their functions and power consumption and also compared different PMICs for the solar panel, finally choosing the ESP32 SoC and the SPV1050 PMIC after discussing with other teammates. After that, he helped investigate further on the different sleep modes of ESP32 to determine whether server pushing or display polling is more energy efficient. He also implemented the Android app code for image processing and Markdown-to-image conversion.	20%

VI Cost of Manufacture

Table 2: Capital Expenses

Item	**Cost**	**Cost/Device**
Bench Digital Multimeter	$1200.00	$12.00
Hot Plate	$26.06	$0.26
Hot Air Gun	$49.99	$0.50
Microscope	$56.99	$0.57

Table 3: Component Costs

Item	**Cost/Device**
PCB Manufacture	$9.00
Display	$33.27
Battery	$17.00
IC	$7.01
Voltage Converter	$14.94
Op-Amp	$1.31
Diode	$2.90
Tactile Push Button	$3.41
JST Connectors	$2.17
Switch	$1.53
Transistors	$0.42
LED	$0.70
Micro-USB Receptacle	$0.45
Capacitors	$10.52
Resistors	$6.34
Inductors	$0.28

Due to the obsolete component issues mentioned in Section III, we were unable to get a cost of manufacture from `https://circuithub.com/`. Given time, we would have sourced more readily available components. This would have made finding a quote much easier. However, we were able to consult an outside resource who was able to generate a CAD rendering of our product as well as a thorough cost of manufacture. Tables 2 and 3 outline the costs for 100 Info Display devices. We have determined that the cost to produce this amount is **$124.58**.

VII Parts

The parts list for one display is shown as follows.

Type	Part Number/Value	Quantity	Cost per unit	Total cost [$]
PCB Manufacture	All PCB	1	9	9
Display	E-Ink 4.2inch display	1	33.27	33.27
Battery	10000mAh Li-On	1	17	17
IC	ESP32-SOLO-1	1	3.4	3.4
	SPV1050	1	3.05	3.05
	MCP73831T-2ACI/OT	1	0.56	0.56
	CP2102N-A01-GQFN28	1	1.48	1.48
Voltage Converter	Polulu LDO	1	6.99	6.99
	AMS1117-3.3	1	7.95	7.95
Opamp	OPA358	1	1.31	1.31
Diode	LESD5D5.0CT1G	3	0.78	2.34
	BAT760-7	1	0.53	0.53
	1N4004	1	0.03	0.03
Tactile Push Button	DTSM-3	2	0.73	1.46
	10-XX	1	1.95	1.95
JST Connectors	2 pin 2mm PH	2	0.7	1.4
	8 pin 2mm PH	1	0.77	0.77
Switch	Sliding Switch	1	1.53	1.53
Transistor	SS8050-G	2	0.21	0.42
LED	Red	2	0.16	0.32
	Green	1	0.2	0.2
	Yellow	1	0.18	0.18
Micro-USB Receptacle		1	0.45	0.45
Capacitors	0.01uF	1	0.48	0.48
	0.1uF	7	0.68	4.76
	4.7uF	3	0.24	0.72
	10uF	2	0.3	0.6
	22uF	3	0.44	1.32
	47uF	4	0.66	2.64
Resistor	0K	3	0.1	0.3
	1K	3	0.43	1.29
	2K	2	0.43	0.86
	2.5K	1	1.37	1.37
	10K	4	0.35	1.4

	22.1K	1	0.26	0.26
	47.5K	1	0.15	0.15
	500K	1	0.11	0.11
	1.5M	1	0.1	0.1
	2.7M	1	0.1	0.1
	6.2M	1	0.1	0.1
	8.2M	3	0.1	0.3
Inductor	22uH	1	0.28	0.28
Total Cost of Product				112.73

VIII References and Citations

VIII.1 Soldered PCB

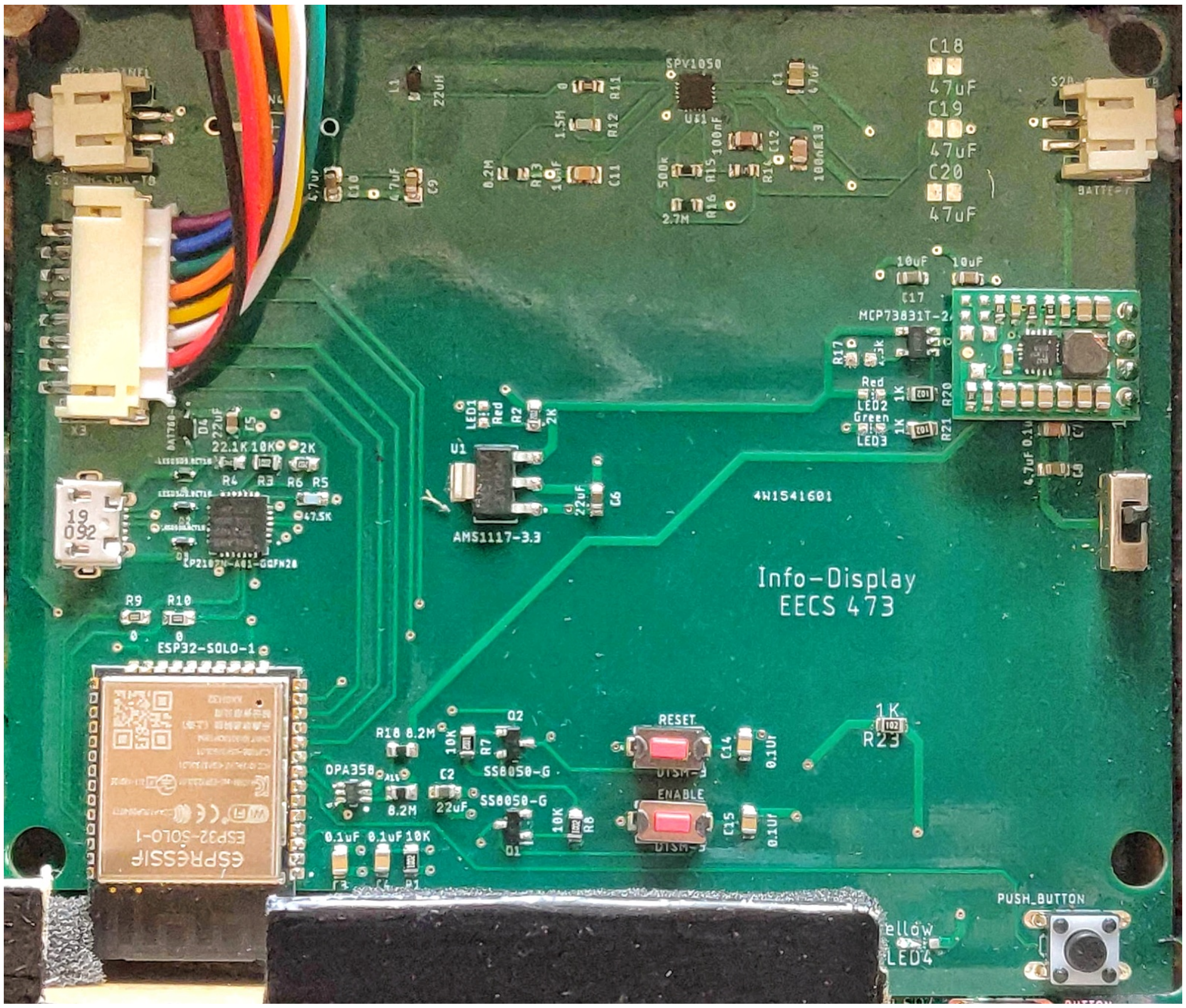


Figure 6: Soldered PCB

VIII.2 Info Display API

```
/**
 * @brief HTTP event handler that stores data into the global buffer.
 *
 * @param evt Internal pointer to event containing data.
 * @return esp_err_t Returns ESP_OK error code upon successful retrieval of data.
 */
esp_err_t _http_event_handle(esp_http_client_event_t *evt);

/**
 * @brief Event callback function to initialize and execute update cycle.
 *        Triggered on successful WiFi connection and retrieval of IP address.
 *
 * @param pvParameter
 */
void infodisplay_cb(void*);

/**
 * @brief Begins the WiFi manager and establishes event callback
 *        for when IP is found.
 */
void infodisplay_system_init();

/**
 * @brief Initializes the e-paper display and enables wakeup by pushbutton
 */
void infodisplay_epd_init();

/**
 * @brief Main execution function for update cycle.
 *        Retrieves information from real-time database, parses, and pushes to display.
 */
void infodisplay();

/**
 * @brief Retrieves image from the real-time database, decodes the data,
 *        and loads it to the interal buffer.
 */
void infodisplay_get_image();

/**
 * @brief Retrieves message from the real-time database and loads it
 *        to the internal buffer.
 */
void infodisplay_get_message();

/**
 * @brief Disconnects from WiFi, forgets WiFi credentials, and starts access point.
 */
void infodisplay_reconnect();

/**
 * @brief Configures the ADC and gets battery percentage reading.
```

```c
 *        Puts this percentage to the real-time database.
 *        Enables timer wakeup inversely proportional to battery reading.
 */
void infodisplay_battery();

/**
 * @brief Creates a frame for viewing timestamp, battery percentage,
 *        and content from the real-time database and loads this frame
 *        into the internal buffer.
 */
void infodisplay_frame();

/**
 * @brief Executes the push of all data (frame, timestamp, battery percentage, content)
 *        to the e-paper display. Closes HTTP client and WiFi connections.
 *        Enters deep sleep mode for both the e-paper display and ESP32.
 */
void infodisplay_push();
```

VIII.3 E-Paper Display API

```c
// global declaration of spi_module
spi_module_t spi_module;

/**
 * @brief SPI module initialization sequence
 */
void epd_spi_init();

/**
 * @brief EPD initialization sequence
 */
void epd_init();

/**
 * @brief Display hardware reset
 */
void epd_reset();

/**
 * @brief Set device to deep sleep mode
 */
void epd_sleep();

/**
 * @brief Send "command" bytes to EPD
 *
 * @param uint8_t command byte to send
 */
void epd_cmd(uint8_t);

/**
 * @brief Send "data" bytes to EPD
 *
 * @param uint8_t data byte to send
 */
void epd_data(uint8_t);

/**
 * @brief Send a byte of data to EPD
 *        Used by epd_cmd() and epd_data()
 *
 * @param uint8_t byte to send
 */
void epd_send(uint8_t);

/**
 * @brief wait for busy signal to become high
 *        (active-low signal)
 */
void epd_wait();

/**
 * @brief clear internal buffer
 */
void epd_clear_buffer();

/**
 * @brief load refresh transition LUTs
 */
void epd_set_LUT();

/**
 * @brief set a bit in display buffer
 *
 * @param int x position
 * @param int y position
 */
void epd_load_pixel(int, int);

/**
 * @brief set a byte in display buffer
 *
 * @param int x position
 * @param int y position
 * @param uint8_t byte to load
 */
void epd_load_pixel_byte(int, int, uint8_t);

/**
 * @brief print a char in display buffer
 *
 * @param char char to load
 * @param int font size
 * @param int x position
 * @param int y position
 */
```

```
void epd_load_char(unsigned char, int, int, int);

/**
 * @brief load a string in display buffer
 *
 * @param char input buffer
 * @param int font size
 * @param int x position
 * @param int y position
 */
void epd_load_string(unsigned char*, int, int, int);

/**
 * @brief load all input into SRAM and refresh display
 *
 * @param char display buffer
 */
void epd_display(unsigned char*);
```

VIII.4 Libraries Used

- WiFi Manager
  - https://github.com/tonyp7/esp32-wifi-manager
- Base64 Encoder/Decoder
  - http://web.mit.edu/freebsd/head/contrib/wpa/src/utils/base64.c
  - http://web.mit.edu/freebsd/head/contrib/wpa/src/utils/base64.h
- Fonts
  - https://github.com/STMicroelectronics/STM32CubeF7/tree/master/Utilities/Fonts
- Android Markdown library
  - https://github.com/noties/Markwon

VIII.5 Documentation and References

- PCB Schematics
  - https://dl.espressif.com/dl/schematics/esp32_devkitc_v4-sch.pdf
  - https://www.st.com/resource/en/datasheet/spv1050.pdf
  - https://www.mouser.com/datasheet/2/268/20001984g-846362.pdf
- ESP32 ESP-IDF API Reference
  - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/index.html

- ESP-IDF Github Repository
  - `https://github.com/espressif/esp-idf`
- E-Paper Display
  - Device Documentation
    * `https://www.waveshare.com/wiki/4.2inch_e-Paper_Module`
    * `https://www.waveshare.com/w/upload/6/6a/4.2inch-e-paper-specification.pdf`
    * `https://www.waveshare.com/w/upload/2/20/4.2inch-e-paper-module-user-manual-en.pdf`
    * `https://www.smart-prototyping.com/image/data/9_Modules/EinkDisplay/GDEW0154T8/IL0373.pdf`
  - Code Reference Repositories
    * `https://github.com/ZinggJM/GxEPD2`
    * `https://github.com/waveshare/e-Paper/tree/master/Arduino/epd4in2`
- Android API
  - `https://developer.android.com/reference`
- Floyd-Steinberg Dithering
  - `https://en.wikipedia.org/wiki/Floyd%E2%80%93Steinberg_dithering`
  - `https://github.com/kehribar/Dithering-OpenCV`
- OpenCV for Android
  - `https://opencv.org/android/`

VIII.6 Acknowledgements

Thank you to the EECS 473 staff for working with us during this incredibly difficult semester. We are so glad to have experienced EECS 473 in the greatest capacity possible because of your efforts. We would also like to thank Steven Rogacki from the University's CLASP Department for his consultation on our PCB Design and power constraints. Additionally, we send thanks to Abhilash Rao for his CAD rendering of our final product and assistance in establishing a solid cost of manufacture. Lastly, we would like to thank our friends and family for supporting our endeavors during this project and encouraging us along the way.