ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

TimeTable Productivity Device

Team #47

BEN XIE
(bx3@illinois.edu)
PRANAV GOEL
(pranavg4@illinois.edu)
HONGRU WANG
(hongru2@illinois.edu)

TA: Pooja Bhagchandani

May 4, 2022

Abstract

This report describes the design process and components used to develop our TimeTable Device. We will introduce our device's purpose along with the context for its creation. We then will talk about the high-level requirements as well as how they and their sub-level requirements were met. Since problem-solving is part of the design process, we will then discuss what went wrong and other challenges we faced. We will also discuss the cost analysis of our project. We will conclude with a breakdown of what went well, what went wrong, and what we should do going forward along with any ethical considerations along the way.

Contents

1 Introduction

1.1 Purpose

1.1.1 Problem

High productivity is something many people try to achieve with little success. Managing tasks is a vital skill to continuously optimize productivity and creating to-do lists is one of the most powerful methods used to accomplish this. This system is often used via apps people access from many devices like phones or laptops. However, apps can ultimately lower productivity rather than increase it. Checking your to-do list can be a multi-step process that takes your focus away from your original task and results in you doing something else. It can allow you to get distracted because less productive apps become easily accessible.In addition, having your to-do list hidden on your tablet or phone makes it easier to ignore, especially when notifications are often dismissed for no reason other than cleaning up the lock screen. This makes it easy to forget to check the app or interact with it entirely. Also, there could be many factors in one's working environment that can lower productivity, such as air quality [1], temperature [2], and humidity [3]. Many people don't even realize that something is wrong and will continue working, attributing their lack of focus and concentration to internal factors such as lack of sleep or stress [4]. In summary, we need a better method of keeping track of daily tasks. We also need something that continuously monitors one's working environment and informs them of issues affecting optimal productivity conditions.

1.1.2 Solution and Function

To give people a better way to monitor their daily tasks, as well as their working environment, we built a desktop device that can display a to-do list as well as monitor environmental factors such as CO_2 levels, temperature, and humidity. This provides a constant reminder of what needs to be done on your desk, making it easier to check and difficult to ignore. We are able to utilize sensors onboard the device to collect environmental data around the workspace.

This device uses an E-ink screen to display tasks due to its readability and low idle power consumption. Because E-ink displays have low refresh rates, we had individually addressable RGB LEDs to communicate some information, such as the task status, in real-time. We also included a rotary encoder and button to interact with the device physically. The primary function of these inputs is to change the status of the tasks between to-do, in-progress, and finished. The status for each task will be shown by the LEDs along the edge of the display. We also included a variety of sensors in the device to measure the environmental factors that were outlined above. Everything connects to an ESP-32 MCU which handles controlling the device as well as communicating with a server that contains all of the tasks. The tasks are sent to the server via a website.

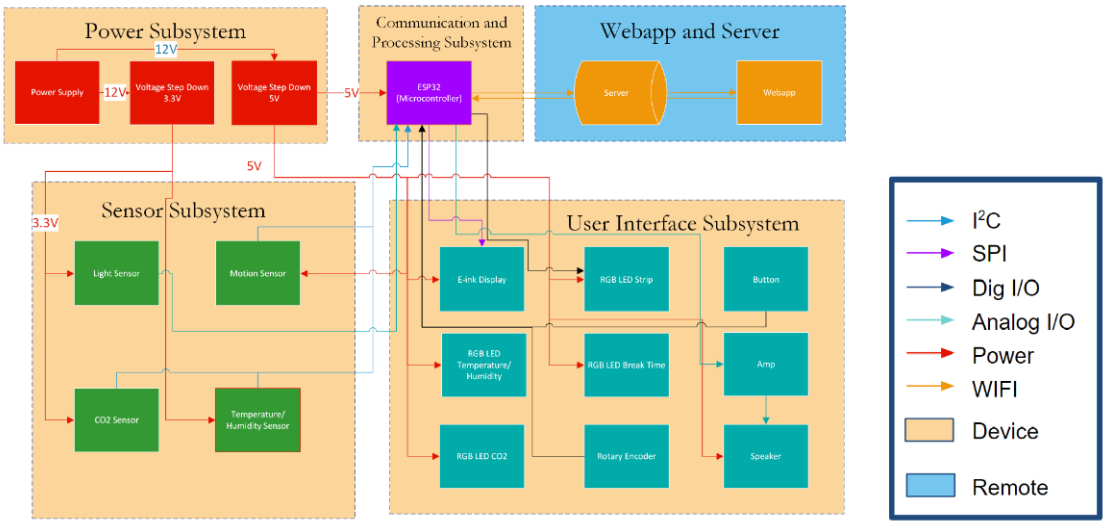

Figure 1: Block Diagram

1.2 Subsystem Overview

This system has 5 subsystems - Processing and Communication, User Interface (UI), Sensing, Power, and Web-App/Server. This section will give a brief description of each of these. Implementation details will be provided in section 2.

The Processing and Communication subsystem is the brain of our device. It controls everything in the UI and Sensing subsystems and handles communication between the device and the Web-App/Server subsystem.

The UI subsystem is responsible for allowing the user to interact with the device. This includes showing multiple tasks on a display, selecting tasks, and changing their state between to-do, in progress, and done, as well as displaying their respective task state. We also wanted a way to display sensor data in an easy-to-decipher visual format.

The Sensing subsystem measures environmental conditions and sends them all to the Processing and Communication subsystem to be routed to the UI and Web-App/server subsystems. We need it to measure Temperature, Humidity, and Air Quality, which have all been shown to impact productivity.

The Power system is responsible for providing a regulated 5V and 3.3V power source to the rest of the device from a 12V input.

The Web-App and Server subsystem is primarily used to input and store tasks. The user will be able to type in the task's name and the due date, and that information will automatically be sent to the server. The server will then send all the tasks to the device over Wi-Fi to be displayed. The website will also need to read the stored environmental data from the server and provide an easy to interpret visualization.

2 Design

2.1 Subsystem Design Considerations

2.1.1 Processing and Communication

The Processing and Communication subsystem needs to be able to drive the E-ink display over SPI, talk to our sensors through I^2C, as well as have multiple other GPIO inputs and outputs for the various buttons, encoders, and LEDs. We also need to be able to connect through Wi-Fi, either through a built-in antenna or an external component. Finally, we wanted something well documented and well supported, to make it easier when designing the PCB and writing the firmware.

The obvious choice here was the ESP-32. This microcontroller is used in a wide variety of applications, has abundant support and documentation, is compatible with Arduino, and most notably, has Wi-Fi and bluetooth capabilities built-in [5]. This microcontroller comes in a variety of versions, we chose the ESP32-WROOM-32E because it was in stock, and met all of our requirements for our device. The pinout for our module can be seen in the schematic below (Figure 2). One problem we ran into for our first PCB was that PIN 35 on the ESP-32 is input only, but it was assigned to the LED output. This was temporarily fixed by moving the LED pin to use the JTAG connector, which we never used. For our second PCB, we swapped the LED pin and the rotary encoder B pin, which resolved the issue.

With the microcontroller chosen, we needed to create a way to program it. This was done through the UART0 pins on the ESP32. We also included JTAG pins in case we needed them for debugging purposes.

The ESP32-WROOM-32E module has many components built into it, so we didn't have to worry about the oscillator, flash memory, and many other capacitors and passive components [5]. However, we did need a way to control the BOOT and ENABLE pins on the module. When ENABLE is triggered, the ESP-32 reboots, and if BOOT is activated while that happens, the ESP-32 boots into flashing mode, allowing us to upload new firmware over UART. Both of these pins are pulled high during normal operation, and are pulled low using a button. The button also has a debouncing circuit around it in order to prevent unwanted triggers of the pin.

Finally, we also included a similar button that allows us to reset the Wi-Fi password stored on the device. Unlike the BOOT and ENABLE buttons, this functionality was not built in, but we included it in the Processing and Communication subsystem because it relates to the Wi-Fi system in our device.

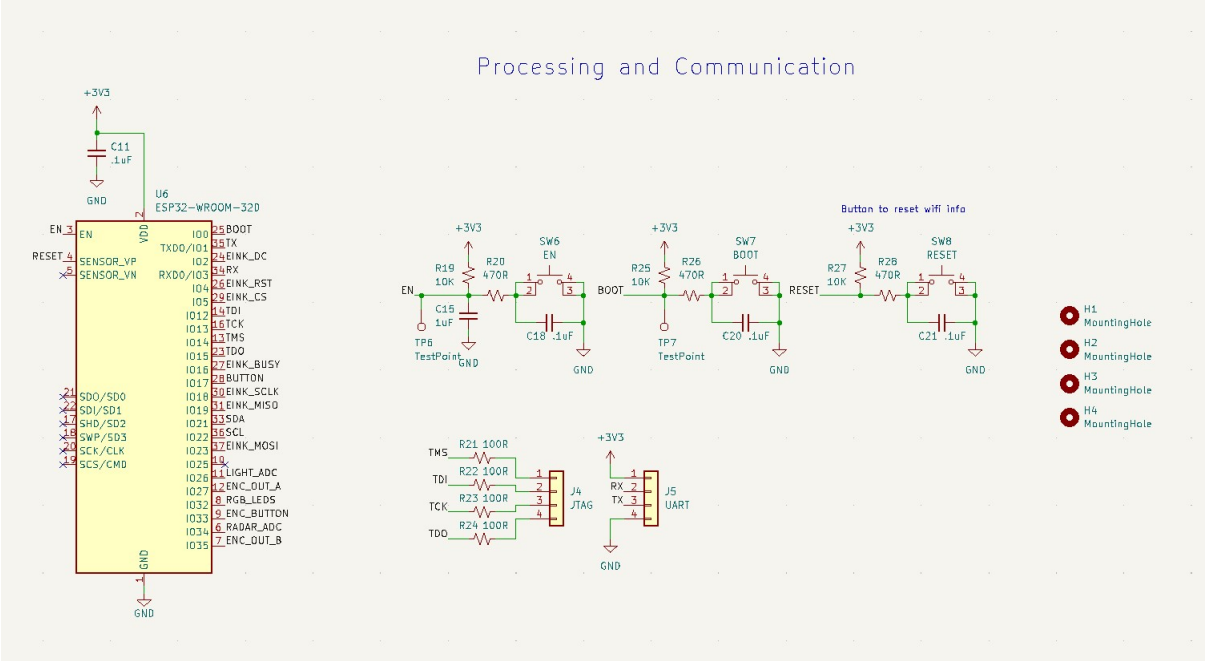

Figure 2: Processing and Communication Schematic

2.1.2 User Interface

The UI subsystem was arguably the most important subsystem in our device. It is responsible for displaying all of our tasks, their corresponding task status, and the environmental data we collect. It also needs to accept user inputs to select and change task statuses.

We decided to build our device around a 7.5-inch E-ink display, which is driven primarily through an SPI interface [6]. We chose an E-ink display over an LCD because E-ink is easier to read, and we did not need the benefits of an LCD screen. The tasks displayed remain largely static, so we don't need a fast refresh rate. That would have added a time constraint to our ESP-32 to meet the refresh rate requirements of the display and made the firmware more difficult to write.

We consulted the E-ink display's datasheet [6] to find out what pins we needed. On the connector to the E-ink display, we had the required SPI pins (MOSI, SCLK, and CS), as well as additional required pins: DC, RST, and BUSY. These pins were specific to the display we chose, and their functions can also be found in the datasheet.

For displaying the task statuses and the environmental data, we chose to use the SK6812 individually-addressable RGB LEDs [7]. These allow us to easily create a row of 14 LEDs on the left side of our screen, one for each task displayed. The LEDs will turn Red, Yellow, and Green, corresponding to To-Do, In Progress, and Completed. These LEDs will be attached to the PCB via a connector. There will also be three similar LEDs on the PCB itself, which are used to display the environmental information. These LEDs had the

footprint flipped on our PCB, which made soldering them on much harder. However, we were eventually able to get all of them connected and working.

For selecting and changing task statuses, we used a rotary encoder with a button built-in [8]. Turning the encoder changes the selected task, and pushing it in advances the task status. Another separate button was included to toggle between showing task statuses and environmental data on the sidebar of RGB LEDs. Both of these required debouncing circuits, as shown on the schematic (Figure 3).

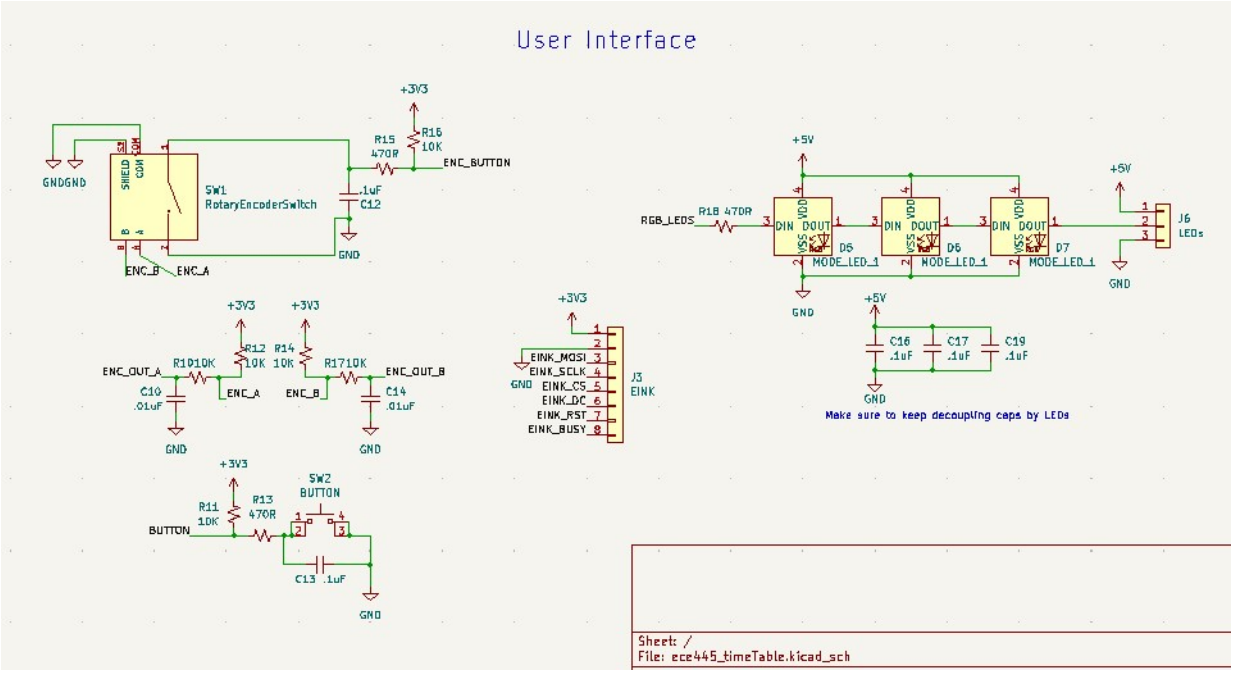

Figure 3: User Interface Schematic

2.1.3 Sensing

The sensing subsystem included sensors for Temperature, Humidity, Air Quality, ambient light, and motion.

For measuring Temperature and Humidity, we chose the SHT40 sensor [9], which communicates over I^2C. For Air quality, we chose the SGP40 [10], which returns an air quality index between 0 and 500, with 0 representing We chose to use I^2C because it was easy to scale to multiple sensors, and only required two traces on our PCB, making routing much easier.

The ambient light was measured with a NPN phototransistor. We had a connector for the phototransistor on our PCB in series with a current-sense resistor. The current pushed through the phototransistor is proportional to the amount of light hitting it, and by using the current-sense resistor with an analog to digital converter, we can measure the brightness in the room. However, because the first phototransistor we bought detected only infrared light, we needed to order a new phototransistor that detected visible light

late into the design process. This phototransistor never arrived, so we were not able to integrate it into our product.

Finally, for motion sensing, we chose to use the RCWL-0516 radar module [11]. We chose this over the standard PIR motion sensor because the RCWL worked through thin layers of plastic and even metal, which we tested in the lab. This allows us to hide the motion sensor inside our enclosure. The sensitivity was also able to be adjusted by soldering a resistor onto the PCB. Through testing, we decided to use a $700\,\mathrm{k}\Omega$ resistor to tune the sensitivity to be high enough to detect small motions close by, but low enough to not detect people walking by.

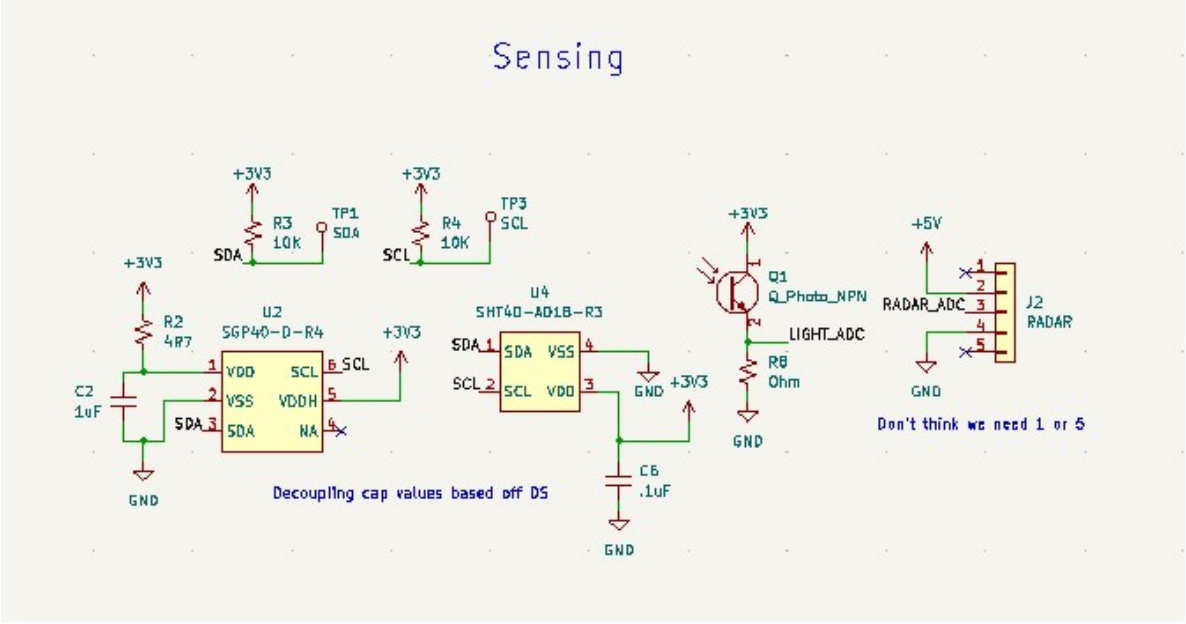

Figure 4: Sensing Schematic

2.1.4 Power

The final subsystem on our PCB was the Power subsystem. This was responsible for delivering clean 5V power to our radar module and LEDs as well as 3.3V to our sensors, microcontroller, and various buttons.

We decided to have a $12\mathrm{V_{DC}}$ input to our PCB from an external power supply. 12 volts was chosen to keep the current low, which opened up our options with the power supply, cable, and connector. This was then stepped down using a switching buck regulator. We decided to use a switching regulator to minimize power loss. If we had used a linear regulator, almost three quarters of the power would have been lost as heat, as opposed to the 90% or higher expected from switching regulators.

$$\frac{3.3\mathrm{V_{DC}}}{12\mathrm{V_{DC}}} = 27.5\% \tag{1}$$

Also, noise wasn't a huge concern for the devices on the 5V rail, so the small amount of ripple from the switching wasn't a concern. However, to step down 5V to 3.3V, we chose to use a linear regulator. That is because linear regulators are simpler, require less components, and are cheaper.

Component selection was pretty straightforward. We added up the maximum expected power draw for each major component, and added a safety factor of 1.5x to determine the minimum current our power components needed to supply. The ESP32 was by far the biggest power draw on the 3.3V power rail at 500 mA. In total, the maximum current was 600 mA, 900 mA with the safety factor. On the 5V rail, we had 17 RGB leds, each drawing 60 mA at maximum power.

$$17 \cdot 60 = 1020 \text{ mA} \tag{2}$$

Adding the load from the linear regulator means that our switching regulator needs to be able to supply at least 2A of current.

We chose the ADP2302ARDZ-5.0-R7 regulator from analog devices [12]. This supplies 2A of current, which meets the power requirements. For the linear regulator, we chose the AZ1117 regulator from diodes incorporated, which supplies a maximum of 1A at 3.3V. This also meets the requirements.

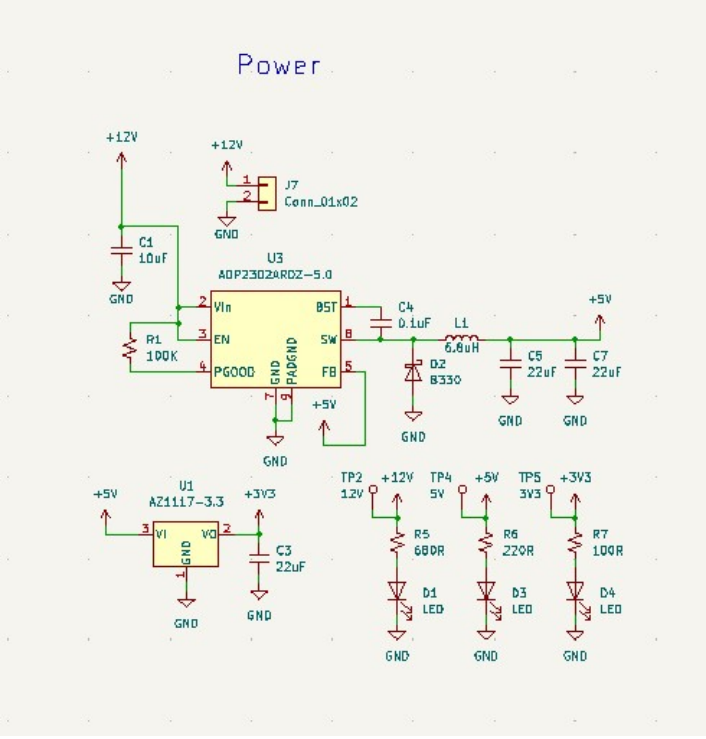

Figure 5: Power Schematic

2.1.5 Software and Server

We used React.js as the framework to develop the front end because its concept of Hooks. Hooks allow as to extract stateful logic such as the status of tasks from the task component so it can be tested independently and reused without changing the component hierarchy, which in turn allows us to share Hooks among both the tasks component and the progress component.

We use Firebase as the backend service because realtime databases can work offline. We can cache the data in device memory, and after reconnecting to the internet, synchronize it. We can call the Realtime Database from our client-side Javascript code, and display dynamic data in the static web-app. In addition, we deployed the web-app through Firebase Hosting Service, which allows us to deploy or test new versions with command in Firebase CLI.

We also developed a Chrome Extension as a shortcut for users to add tasks which is currently published on Chrome Web Store.

3 Cost and Schedule

3.1 Cost

3.1.1 Parts

All the parts except for the E-ink display and filament were ordered twice to in case the parts get destroyed when soldering them onto the PCBs. View the part cost breakdown in Appendex A, Table 1.

$$\Sigma \text{ Cost per Part} = \$210.04 \tag{3}$$

Based on equation 3 we can see that the total cost of all the parts is $210.04.

3.1.2 Labor

Labor costs were based around an hourly salary of $40/hour which is the average salary of a new ECE graduate from UIUC. This was found based on the average UIUC Electrical Engineering salary of $76,079 and the average Computer Engineering salary of $92,430. It also assumes that each team member will work 10 hours per week for 15 weeks. Therefore,

$$3 \text{ team members} \cdot \frac{\$40}{\text{hour}} \cdot \frac{10 \text{ hours}}{\text{week}} \cdot 15 \text{ weeks} \cdot 2.5 = \$45,000. \tag{4}$$

3.1.3 Total

$$\text{Total Cost} = \text{Parts Cost} + \text{Labor Cost} = \$210.04 + \$45,000 = \$45,210.04 \tag{5}$$

Based on equation 5 we can see that the total cost of this project is **$45,210.04**.

3.2 Schedule

See schedule in Appendix A, Table 2.

4 Design Verification

You can reference all requirement and verifications in Appendix C [13].

4.1 Processing and Communication

By connecting the ESP-32 to Wi-Fi, we were able to utilize Firebase's API to interact with the database. We quickly found that it took less than a second to call the data from the server to the MCU. The timing largely depended on when we made the call to the firebase database. This is because each function has to be finished one at a time. When a slow function is running, like writing to the E-ink display, the firebase function is blocked until the first one finishes. This can lead to a delay of up to five seconds. This may seem like a small number but, in practice is far too long to stop all the other device functionalities. Our solution to this was to utilize multiple cores in the ESP32 (multi-core diagram found in Appendix B, Figure 10). This allowed the Firebase calls to happen simultaneously with device interactions, significantly increasing the speed. With these enhancements, we were able to easily meet the requirement of sending and receiving data within 2 minutes. We verified this by checking changes in the database vs. the serial output for the device.

4.2 User Interface

We used a boolean value in the Firebase Database as an indicator as to when we need to trigger the E-ink display. We manually set this to "True" when testing the refresh speed, then measured how long the E-ink display took to finish updating. We found it took approximately 5 seconds to update, clearing the 15 second requirement. For displaying the tasks themselves, getting multiple tasks to render at the same time required looking into how graphics work on the E-ink display. An E-ink display works by physically moving black ink particles to the front or back of the screen to display "black" (black particles in the front) and "white" (black particles in the back). This is done by using "positive" and "negative" electric fields to interact with the particles [14] (See Figure 6).

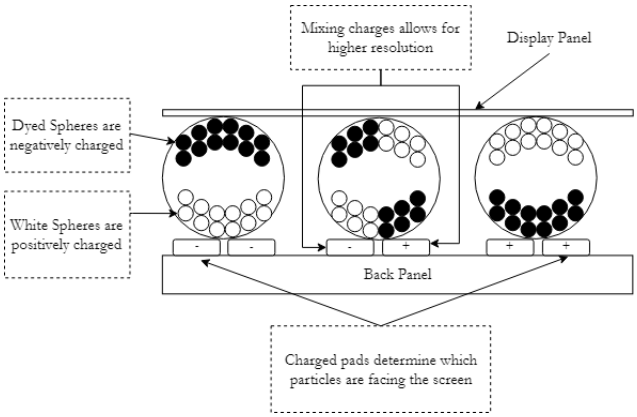

Figure 6: How an E-ink Display Works

Keeping this in mind, we had to find a way to first place all the tasks we wanted to display in the correct location and then render everything. We used a library called GxEPD2 by ZinggJM [15] to handle the placement and rendering of text and our custom background. This worked by having us essentially tell the display, relative to its dimensions, the location to place each task, and in what font. Then "write" each line by telling each pad where we want the charges to be. Then it finally rendered everything by setting those charges and moving the ink to the correct locations. A similar concept was applied to the custom background to have that render alongside the actual tasks. This allowed us to complete the requirements as now we only have to render once and display all the tasks specified simultaneously.

For the LEDs, we found that the footprints we used had a few orientation issues. We tried to use soldering paste instead of the regular solder to decrease the difficulty but found that this allowed for the LEDs to short. So we had to solder everything manually, fixing most of the hardware issues. Then we checked if all the LEDs chained together were able to turn on and change to specific colors - both the same colors and different colors. We found that all of these operations were successful, completing the verification for that requirement. Since the LEDs worked, it was fairly simple to test if the Rotary Encoder and Mode buttons also worked. We had each LED turn brighter when it was "selected". Then as we turned the knob we had the next LED inline become brighter instead. Checking if each detent switched the LED to the correct brightness in both directions allowed us to confirm that this requirement was passed. Similarly, for the Mode button, we paired it with certain LEDs and checked if pressing the button turned them on. This requirement was also passed.

4.3 Sensing

Verifying the temperature, humidity, and air quality sensors was a relatively straightforward process. We checked the output values of the sensors with the expected values. We found that the temperature sensor ran a little bit hot. This was fixed by calibrating it to match the correct temperature (simply subtracting the expected difference). This kept the new output values close to the actual values. A similar process was used for the other two sensors as well.

The motion sensor technically passed our requirements as it was able to detect movement within 2 meters. However it was too sensitive so it was picking up movement we were not looking for. The ambient light sensor was unverified as we never received it. Both of these requirements will be elaborated further in the uncertainties section (5.2).

4.4 Power

This was relatively simple to verify. We built test pads along with the 3.3, 5V, and 12V rail so that we can use a multi-meter to check if the voltage is correct. The following images show the results of this test.

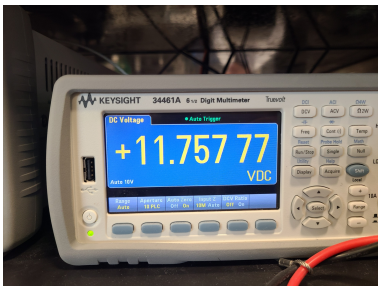

Figure 7: 12V Multi-meter Readout

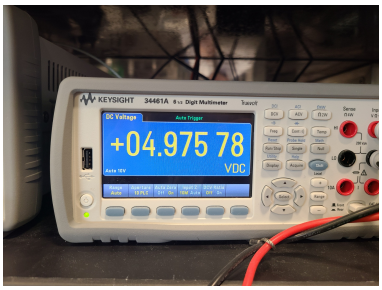

Figure 8: 5V Multi-meter Readout

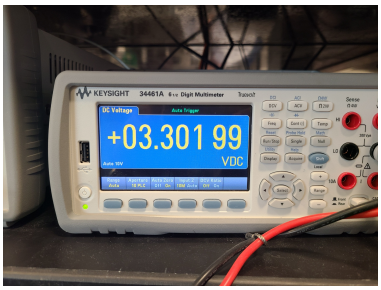

Figure 9: 3.3V Multi-meter Readout

The 3.3V output had an error of only .06% and the 5V output had an error of .48%. It is important to note that the 12V input doesn't have to be very accurate so long as the requirements for 5V and 3.3V are still being met. Our buck converter was actually able to take up to 20V of power, which allowed for consistent power delivery for 5V and 3.3V even if the 12V rail fluctuated. Because both of the outputs are well within 2% error, we were able to verify the power requirements.

4.5 Web Application, Server, and Chrome Extension

The web application is used to store tasks as input for the device, modify the tasks' content, status, or due date, display the current environment data collected from the sensors, and show the user's productivity as a ratio of tasks created and tasks completed. The Firebase Realtime Database connects our device with the web application and the Chrome Extension. The Chrome extension is used as a shortcut to add tasks without opening the web-app. The logic behind is very straightforward: the user inputs the data such as task information through the web application and Chrome Extension into the database, and the device then retrieves the data of task information from the database. Similarly, the device stores the environment data from the sensors in the database, then the web application retrieves the environment data and renders it.

The first requirement of the subsystem is that the user can store at least 12 tasks from the web application to the firebase. Because of the size of our device, we managed to allow the user to input at most 14 tasks. We verified that the database immediately stored the input from the front-end. And the device retrieved the task information from the server within ten seconds. The second requirement is that the environment data including the temperature, humidity, and AQI values collected from the sensors on the device gets transmitted to the server. We verified that as long as there is a change in the temperature, humidity, or AQI values, it would be reflected in our database immediately. The Firebase real-time database is very responsive and useful to transmit data between the device and the web-app.

5 Conclusion

There are many takeaways we had while developing our project. We developed skills in teamwork, time management, cost management, and salesmanship. It was important that we prioritized the main functionality of our design - creating a Minimum Viable Product (MVP). Focusing on the details of the MVP allowed for the resulting product to be at a point where someone could use it. We added extra features to the device like over-the-air updates (OTA) and multi-core processing to the MVP because of this. Our focus on creating a high quality MVP gave way for the accomplishments but also resulted in a few uncertainties as discussed below.

5.1 Accomplishments

Our first high level requirement specified that our device must display at least 12 of the tasks at the same time and within a minute. Our device was able to render 14 tasks at the same time along with our own custom design for the background. We were also able to display the tasks well within a minute of adding a task. Oftentimes it would only take 10-15 seconds before it renders. The longest time it took before rendering was around 40-45 seconds. The rendering time itself took 5 seconds or less. As such, our project successfully satisfied this requirement.

Our second high level requirement was to take environmental data on air quality, temperature, and humidity and send it at least once to the server every 15 minutes. We were able to measure the data and send the data over in almost real time. The delay would be at most 30 seconds and at least 2-3 seconds before that data was sent over to firebase. We tested this by printing out when it reads data and timing from that point till when the firebase database updates. This means our project successfully satisfied this requirement.

There are also many accomplishments outside of the high level requirements. The main two successes were getting multi-core processing and OTA updates to work for the ESP-32. A big part of achieving the high level requirements was due implementing multi-core processing. We were worried about different calls blocking each other, causing massive time delays. By using both ESP32 cores we were able to separate out database related functionality and local functionalities. For example, getting the E-ink display to work required calling tasks from the database before rendering them out. It takes some time to actually render the tasks. So while it renders, in a single core we wouldn't be able to do anything else. This would significantly delay other features like task selection which should be instantaneous (because it is done locally on the device). By separating these on each core, we can have both of these tasks run at the same time. For conflicts, we used FreeRTOS to schedule/prioritize threads, speeding up the processes significantly. Refer to the multi-core diagram in Appendix B, Figure 10 for a visual representation of the process. This is what allowed the user interface to be so responsive and still meet our high level requirements relating to the database side of the project.

For OTA, we created a portal from which we can upload our firmware directly. This was then uploaded to a storage database in Firebase. We also had a version management setup in Firebase's realtime database. On the ESP32, we were constantly checking that database to see if the most recent firmware version matched the current version. If it did not, then we pulled the firmware from the storage and flashed that. Refer to the OTA Diagram in Appendix B, Figure 11 for a detailed breakdown of the OTA process. Getting OTA to work enabled us to speed up our development time significantly, made our product more reliable, and ultimately extensible for the future. OTA proved to be extremely useful to program the ESP32 when UART became unreliable. With OTA, we were able to update the ESP32 from anywhere so long as it remained connected to Wi-Fi.

Another accomplishment was getting the device to connect to Wi-Fi as long as a network is available, making the device extremely portable. The size of the device also contributed to its portability. Because we 3D printed the chassis, we were able to iterate upon the design to better fit the hardware. We were also able to load in more complex custom E-ink Designs to the display, allowing for a more professional aesthetic on the device. Our accompanying web-app and extension was also a major success. Adding tasks became quick and efficient. The web-app provided an interface to view environmental data.

5.2 Uncertainties

Unfortunately we were unable to meet our third requirement which was to determine if a user was present within 5 minutes of the current status changing. This was a direct result of being unable to tune the motion sensors satisfactorily. The radar motion sensor works by sending a high signal if it detects movement and a low signal if it doesn't. Changing from high to low or low to high gave us rising and falling edges to detect. If an edge is detected, we know that the user's status has changed. We were taking advantage of this by setting an interrupt to call certain actions when that detection happens. Unfortunately, the signal initialized at a high signal and never fell. The sensor we were using had 360 degree vision and could work through plastic. This meant that it was extremely easy for it to detect movement from anywhere, not just the user sitting in front of it. Because the signal never changed to low, the interrupt would never be called and therefore nothing would ever happen. The documentation for the device indicated that we could adjust the sensitivity range by adding a resistor. We determined that the motion sensing was a nice feature that ultimately wasn't as important to the MVP as some of the other features we included. However, in the future we would likely find a more adaptable radar motion sensor or use a different type of motion sensor such as infrared instead. In order to make the motion sensing work, we needed something that is less sensitive and can have distinct states that determine if a user is present or not.

We also were unable to implement the ambient light detector into the circuit. This was purely a procurement issue as our first sensor detected only infrared light, which is useless for detecting visible light. The replacement sensor we ordered two weeks before our demonstration never arrived. Because we didn't have the component, we were unable to make the circuit.

We also found our 12V power plug was unreliable. Although it successfully delivered 12V, there were occasional power surges. This resulted in two filtering capacitors connected to the 12V power rail to fry. This caused our PCB to stop working, as it directly shorted our 12V power rail to ground. We determined that the rest of the circuit was fine by directly delivering 5V from the test point and found everything else was working fine. After replacing the capacitors, the circuit worked correctly again. However, we decided to no longer use the 12V power supply. We later discovered that the power supply was discontinued for its unreliability, so an easy solution would be to use a higher quality power supply.

5.3 Ethical Considerations

The user of our project will be directly involved with the operation of the device; We adhered to the guidelines from Section I.1 of the IEEE Code of Ethics that "to hold paramount the safety, health, and welfare of the public. . ." [16] and the safety regulation in Sections 1910.302 - Electric utilization systems by Occupational Safety and Health Administration [16]. It is important that we must ensure a safe and reliable product. There are several components in our product that, if mishandled or constructed poorly, might pose a risk to the user's safety. The power supply and circuits are some of the most important components to consider. The risks associated with the unstable power supply will be reduced by a rigorous power electronics design, and we would make sure that the users use our product in a safe environment. After testing the power delivery for 12V, 5V, and 3.3V, we felt that we met regulations. Furthermore, we considered that our product must be available to everyone per Section II of the IEEE Code of Ethics [16] and that our users can equally operate our device. The mission of our project is to give people a better way to monitor their daily tasks and their working environment. The idea behind our device was to provide a simple, accessible, and tactile solution to task management. This, by its very nature, should make it available to everyone as actually using requires very little work on the users end.

5.4 Further Work

5.4.1 Device

Much of our future work on the device consists of further refining and polishing our design. We can use a frame with different materials and better form to hold the hardware better, be visually more aesthetic, and feel more professional upon use. Adding Screen Protection (perhaps an acrylic sheet) would also be nice to make the device last longer under heavy use. Adding a battery to the device instead of a plug would make it even more portable than it currently is. Another potential improvement is to have better protection against faulty 12V adaptors. Also, there is a short delay when the device retrieves data from the database, which can be improved by having faster response time. In addition, we want to add more visually satisfying cues for users. For example, we could have a line cross out a completed task as well as the LED indicating its completed.

5.4.2 Web Application and Chrome Extension

Because we only have one device, we didn't develop any user authentication. User authentication would allow users to use multiple devices. We could verify the user's identity and limit the access to the resources in the database. Integrating third party APIs with our device would make it even more useful to many people by naturally becoming an already existing system. For example, we could have tasks added in the web-app show up in Google Calendar as events while still being synchronized to the user's other devices. If more users and devices are connected, we can use AWS Elastic Load Balancing and AWS EC2 to spread user traffic across many instances to decrease the possibility of performance issues. Currently, only the real time environmental data is rendered in the web-app. It would be more useful to show the trends of environment data so that we can make meaningful connections to productivity. For example, we can associate the percentage of task completion with each environmental factor over a large period of time. We could define optimal conditions for each individual by recommending conditions where that percentage was highest. Another possible feature is to have a chatbot to add a task, which can be implemented by AWS Lex and AWS Lambda Function. For the Chrome extension, only adding tasks are supported. It would be more convenient to users if they can update/delete tasks through the extension. We believe that making the task adding functionality as non-invasive to your regular workflow is extremely important. That was why we created the extension. That being said, a quick improvement could be to create automatic hotkeys that quickly open the extension so you could add a task quickly and without leaving the page you are currently on.

References

[1] J. G. Allen, P. MacNaughton, U. Satish, S. Santanam, J. Vallarino, and J. D. Spengler, "Associations of cognitive function scores with carbon dioxide, ventilation, and volatile organic compound exposures in office workers: A controlled exposure study of green and conventional office environments," *Environmental Health Perspectives*, vol. 124, no. 6, pp. 805–812, 2016. DOI: 10.1289/ehp.1510037.

[2] S. S. Lang. "Study links warm offices to fewer typing errors and higher productivity." (Oct. 2004), [Online]. Available: https://news.cornell.edu/stories/2004/10/warm-offices-linked-fewer-typing-errors-higher-productivity.

[3] A. Hedge, *Linking environmental conditions to productivity - cornell university*, Jun. 2004. [Online]. Available: http://ergo.human.cornell.edu/Conferences/EECE_IEQ%5C%20and%5C%20Productivity_ABBR.pdf.

[4] R. A. HENNFNG, S. L. SAUTER, G. SALVENDY, and E. F. KRIEG, "Microbreak length, performance, and stress in a data entry task," *Ergonomics*, vol. 32, no. 7, pp. 855–864, 1989. DOI: 10.1080/00140138908966848.

[5] *Esp32-wroom-32e datasheet*, v1.4, Espressif, 2022. [Online]. Available: https://cdn-shop.adafruit.com/product-files/1138/SK6812+LED+datasheet+.pdf.

[6] "7.5inch e-Paper HAT." (2022), [Online]. Available: https://www.waveshare.com/wiki/7.5inch_e-Paper_HAT (visited on 03/28/2022).

[7] *Sk6812 technical data sheet*, SK6812, Rev. No. 01, Adafruit, 2022. [Online]. Available: https://cdn-shop.adafruit.com/product-files/1138/SK6812+LED+datasheet+.pdf.

[8] *Pec11r series - 12 mm incremental encoder*, PEC11R, REV. 01/22, Bourns, 2022. [Online]. Available: https://www.bourns.com/docs/Product-Datasheets/PEC11R.pdf.

[9] *Datasheet sht4x*, SHT40, Version 3, Sensirion, 2022. [Online]. Available: https://sensirion.com/media/documents/33FD6951/624C4357/Datasheet_SHT4x.pdf.

[10] *Datasheet sgp40*, SGP40, Version 1.2, Sensirion, 2022. [Online]. Available: https://sensirion.com/media/documents/296373BB/6203C5DF/Sensirion_Gas_Sensors_Datasheet_SGP40.pdf.

[11] *Rcwl-0516 information*, RCWL-0516, IC Station, 2021. [Online]. Available: https://github.com/jdesbonnet/RCWL-0516.

[12] *Nonsynchronous step-down regulators*, adp2302, Rev. A, Analog Devices, 2012. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/adp2302_2303.pdf.

[13] H. W. Ben Xie Pranav Goel. "Productivity Enhancement Device (TimeTable)." (2022), [Online]. Available: https://courses.engr.illinois.edu/ece445/getfile.asp?id=20031 (visited on 03/28/2022).

[14] T. Carmody, *How e ink's triton color displays work, in e-readers and beyond*, Nov. 2010. [Online]. Available: https://www.wired.com/2010/11/how-e-inks-triton-color-displays-work-in-e-readers-and-beyond/.

[15] ZinggJM, *Gxepd2*, https://github.com/ZinggJM/GxEPD2, 2022.

[16] IEEE. "IEEE Code of Ethics." (2016), [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html (visited on 02/08/2020).

Appendix A Tables

Table 1: Cost Breakdown of each Part.

Cost				
Description	Part #	Manufacturer	Quantity	Cost
0.1uF Capacitor	581-08056D104KAT2A	Kyocera AVX	2	$0.60
0.01uF Capacitor	77-VJ08Y100V103K	Vishay	4	$1.16
3.0 Amp 30 Volt Diode	625-B330LA-E3/61T	Vishay	2	$0.94
10uF Capacitor	963-TMK212BBJ106MGHT	Taiyo Yuden	2	$0.64
1uF Capacitor	581-08053C105JAT2A	Kyocera AVX	4	$1.44
22uF Capacitor	963-JMK212BBJ226MG8T	Taiyo Yuden	6	$1.62
Ambient Light Sensors	720-SFH3711	ams OSRAM	2	$1.50
0.1uF Capacitor	581-08056D104KAT2A	Kyocera AVX	20	$4.12
ESP32-WROOM	356-ESP32WRM32E164PH	Espressif	4	$13.20
Tactile Switches	506-FSM12JH	TE Connectivity	8	$1.20
Lighting Series	1528-1709-ND	Adafruit Industries LLC	4	$23.80
2POS 2.54MM Connectors	WM4111-ND	Molex	2	$0.58
Phototransistors 850nm	475-1419-ND	OSRAM Opto Semiconductors Inc.	2	$1.84
4.7 Ohms Resistors	RNCP0805FTD4R70CT-ND	Stackpole Electronics Inc	2	$0.20
10K Ohms Resistors	RNCP0805FTD10K0CT-ND	Stackpole Electronics Inc	24	$2.40

Cost Continued (Table 1)				
Description	Part #	Manufacturer	Quantity	Cost
680 Ohms Resistors	RNCF0805DTE680RCT-ND	Stackpole Electronics Inc	2	$0.38
220 Ohms Resistors	RMCF0805FT220RCT-ND	Stackpole Electronics Inc	2	$0.20
470 Ohms Resistors	P470AZCT-ND	Panasonic Electronic Components	12	$9.72
Red 625nm LED	28-LSM0805412VCT-ND	Visual Communications Company - VCC	6	$2.28
30 V 3A Diode	B330-FDICT-ND	Diodes Incorporated	2	$1.16
8POS 2.54MM Connectors	WM4117-ND	Molex	2	$1.34
6.8 µH Inductor	445-180836-1-ND	TDK Corporation	2	$1.12
100K 1/8W Ohms Resistors	RMCF0805FT100KCT-ND	Stackpole Electronics Inc	2	$0.20
100 1/4W Ohms Resistors	RNCP0805FTD100RCT-ND	Stackpole Electronics Inc	10	$1.00
1.2M Ohms Resistors	P1.2MGCT-ND	Panasonic Electronic Components	2	$0.20
Air Quality Gas Sensor	1649-SGP40-D-R4CT-ND	Sensirion AG	2	$21.00
Humidity Temperature Sensor	1649-SHT40-AD1B-R3CT-ND	Sensirion AG	2	$6.00
LDO Voltage Regulators	AZ1117IH-3.3TRG1	Diodes Incorporated	10	$3.21

Cost Continued (Table 1)				
Description	Part #	Manufacturer	Quantity	Cost
10 Ohms Resistors	CRCW080510R0FKEAC	Vishay	10	$0.67
Rotary Encoder	PEC11R-4220F-S0024	Bourns Inc.	2	$3.50
Switching Voltage Regulators	ADP2302ARDZ-5.0-R7	Analog Devices Inc.	2	$6.92
E-ink display	B07Z25LWTS	Waveshare	1	$73.02
Filament	B08SM24833	JAYO 3D Store	1	$22.88

Table 2: ECE 445 Semester Schedule

Schedule			
Week	Ben Xie	Pranav Goel	Hongru Wang
01/31/2022	Brainstorm project ideas and write proposal	Brainstorm project ideas and write proposal	Brainstorm project ideas and write proposal
02/07/2022	Find parts for device and start preliminary BOM	Create the Block Diagram and Visual aid for Project	Create tolerance analysis
02/14/2022	Create schematic draft for main PCB	Gather supplementary materials (research) for Design Document and calculate requirement values	Start web front end developing
02/21/2022	Start working on Firmware - Get firebase/WiFi working	Create PCB Layout for main board	Create Schematics/PCB design for side LED module
02/28/2022	Create preliminary physical design	Verify PCB Layout with TAs	Order parts

Schedule Continued (Table 2)			
Week	Ben Xie	Pranav Goel	Hongru Wang
03/07/2022	Get CO2 sensor working and test accuracy	Get LEDs to work with Sensor Readings/Modes	Connect front end to firebase, write APIs
03/14/2022	Spring Break - Catch up to schedule	Spring Break - Catch up to schedule	Spring Break - Catch up to schedule
3/21/2022	Design and fabricate enclosure, solder PCB	Get E-Ink display working, debug first circuit, solder PCB	Get Motion Sensor working
3/28/2022	Clean up codebase	Verify Design Document Requirements	Debug web app
4/04/2022	Polish/Clean up mechanical design	Polish Front End on both Device and Website	Polish client side for better user interface design
4/11/2022	Work on presentation, debugging	Work on presentation	Work on presentation
4/18/2022	Mock Demo - Fix last minute issues	Mock Demo - Fix last minute issues	Mock Demo - Fix last minute issues
4/25/2022	Demo/Mock Presentation/Start Final Paper	Demo/Mock Presentation/Start Final Paper	Demo/Mock Presentation/Start Final Paper
5/02/2022	Presentation/Final Paper	Presentation/Final Paper	Presentation/Final Paper

Appendix B Diagrams

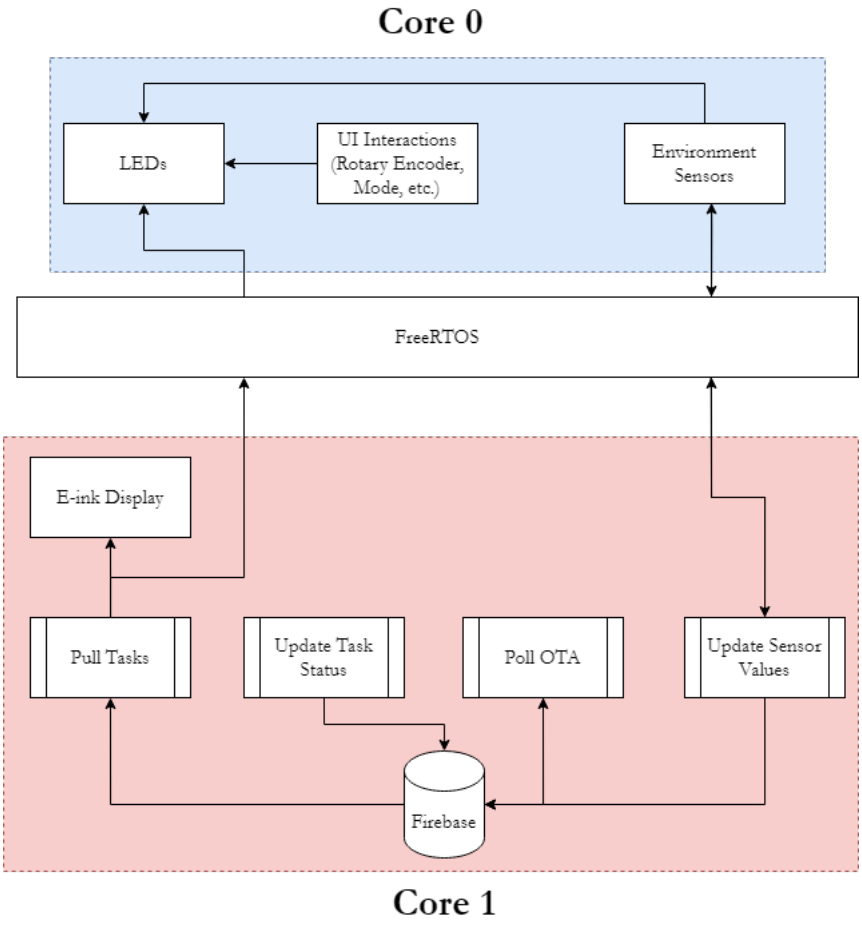

Figure 10: Multi-core Implementation Flowchart

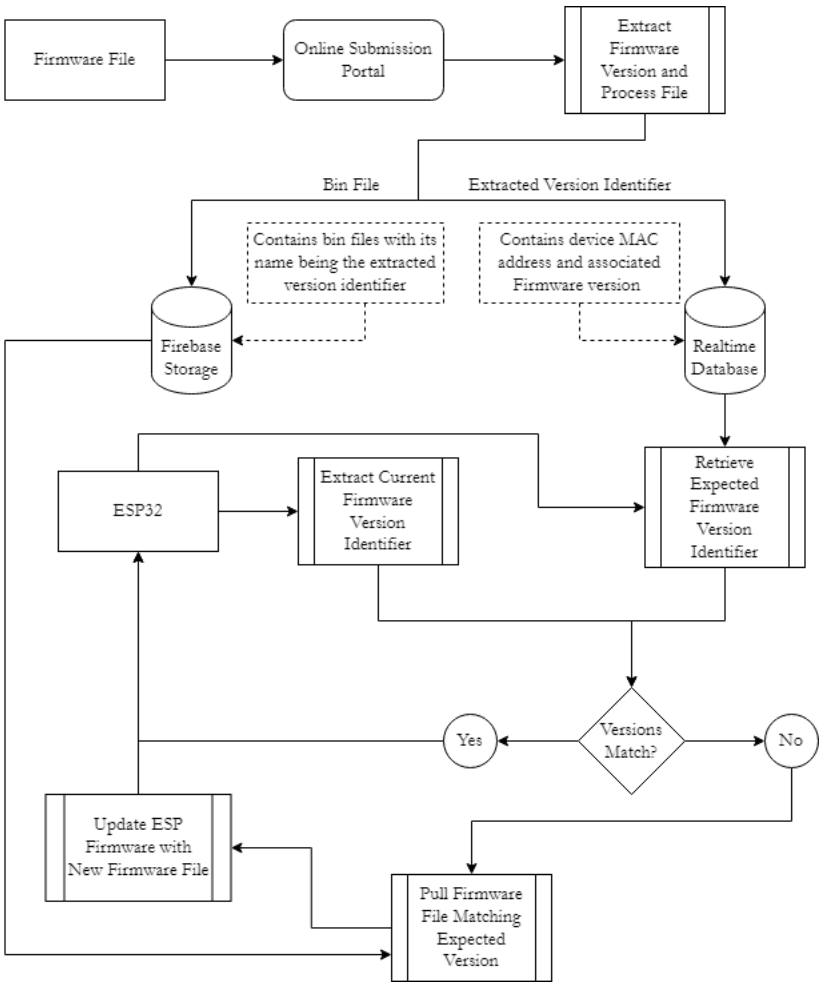

Figure 11: OTA Implementation Flowchart

Appendix C Requirements And Verification

Requirement	Verification
The buck converter provides fixed 5.0V ± 5% from a 12V source	Measure the output voltage using an oscilloscope, test operations that can use power such as screen update and sensor usage and ensure the output always stays within the expected range.
The LDO regulator provides fixed 3.3V ± 2%	Measure the output voltage using an oscilloscope, test operations that can use power such as screen update and sensor usage and ensure the output always stays within the expected range.

Table 3: Power Requirements & Verification

Requirement	Verification
The e-ink display must be able to completely refresh in under 15 seconds.	Trigger an update manually and measure the amount of time it takes to finish refreshing
The e-ink display must be able to show at least 12 tasks.	Add 12 tasks to the database, verify that all of them show up on the display
Each RGB LED needs to be able to show at least 6 colors (Red, Orange, Yellow, Green, Blue, Violet)	Run the sample NeoPixel code on the ESP-32 and verify that all colors show
The speaker must be able to produce a sound above 50 dB	Measure the volume of the speaker using a Decibel Meter
The device must be able to detect each detent of the rotary encoder as it turns	Write a simple program that changes the LED that is lit based on the rotary encoder and verify that each detent changes the LED by one.
The device must be able to register button presses both from the encoder and the tactile button	Write a simple program that turns the LEDs on and off based on button presses, verify that each button works

Table 4: User Interface Requirements & Verification

Requirement	Verification
The air quality sensor must be able to detect changes in air quality that may affect humans. (A CO_2 change of 500 ppm.)	Bring the sensor outside, take a note of the readings. Bring it back inside, and verify it on the air quality scale (Figure 7). A normal CO_2 reading outdoors is 400ppm, and complaints of drowsiness start at around 1000ppm [1].
The measured temperature needs to be within ±1°C of the actual temperature between 10°C and 40°C.	Compare measured temperature with a thermometer.
The measured relative humidity needs to be within ±3% between 25% and 75% relative humidity.	Compared measured humidity with a hygrometer.
The motion sensor must be able to detect movement within 2 meters of the device.	Output a message via Serial output when motion is detected. Verify that happens when someone moves within 2 meters of the device.
The ambient light sensor needs to be able to output a value proportional to the actual light intensity.	Put the device in a room with a dimmable light bulb. Output the ambient light level via serial output and ensure it changes as the light bulb dims.

Table 5: Sensing Requirements & Verification

Requirement	Verification
The MCU must be able to receive server information within 2 minutes of data sent	Write code so that the ESP-32 outputs any data received from the server over Serial and ensure it meets the time requirements
The MCU must be able to send relevant data to other subsystems within 2 minutes of it being called for	Ensure that the server updates within 2 minutes of a task status update on the device. Ensure that the e-ink display updates within 2 minutes of receiving new tasks.

Table 6: Processing and Communication Requirements & Verification

Requirement	Verification
The server must be able to store at least 12 tasks simultaneously.	Verify by performing adding tasks through the client-side and check if the tasks count is 12 in firebase.
The server must be able to store at least 24 hours of sensor data.	Check if the tasks are stored in the firebase real-time database or check the dashboard.

Table 7: Web App Requirements & Verification