

Log Book

Rajdeep Gill 7934493

ECE 3760 A01

March 4, 2025

Contents

1	LOGBOOK #1 START	2
2	LOGBOOK #2 START	4
3	LOGBOOK #3 START	8

1 LOGBOOK #1 START

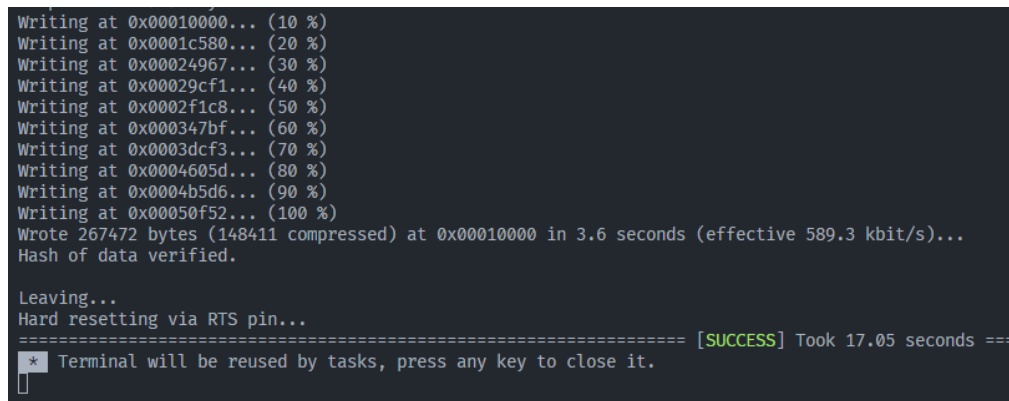
Lab 1 Entry

January 20, 2025

Received the board, soldered pins to the board and ensured that the board was working and no short circuits were present.

Installed platformio on VSCode and ensured the extension was working correctly by running a simple serial print program on the board.

The program was first built and then uploaded to the board, and got a successful message in the console as seen in Figure 1.



```
Writing at 0x00010000... (10 %)
Writing at 0x0001c580... (20 %)
Writing at 0x00024967... (30 %)
Writing at 0x00029cf1... (40 %)
Writing at 0x0002f1c8... (50 %)
Writing at 0x000347bf... (60 %)
Writing at 0x0003dcf3... (70 %)
Writing at 0x0004605d... (80 %)
Writing at 0x0004b5d6... (90 %)
Writing at 0x00050f52... (100 %)
Wrote 267472 bytes (148411 compressed) at 0x00010000 in 3.6 seconds (effective 589.3 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
===== [SUCCESS] Took 17.05 seconds =====
* Terminal will be reused by tasks, press any key to close it.
█
```

Figure 1: Successful upload message in console

Design Ideas

January 20, 2025

During the lab, discussed some ideas for the project. Talked about the basic requirements, what potential ways we can meet the requirements. Need to do more research on how curling actually works to get a better idea of what someone would need to communicate with their team members to ensure a successful game.

Currently thinking of the skip having a device that can communicate with the two sweepers, having a speed up and slow down button for the sweepers to adjust their speed. The skip would also have a button to indicate when to stop sweeping. On the sweepers side, they would have an LED or a speedometer esque led display to show them how fast they should be sweeping. Since different players might need to sweep at different rates, need to differentiate somehow between the two sweepers. Maybe have two of the same device, but with different colored LEDs or something to indicate which sweeper the skip is talking to. For example, a left and right sweeper device, that connects to the respective sweeper.

Individual Design Brainstorming

January 26, 2025

A rough sketch of the design I had come up with is provided below. Essentially two sets of controles will be present on the skip's device, one for each sweeper. Along with a display made of LED lights, or other visual indicators to reflect what the sweepers are seeing on there device. 3 buttons will be present, speed up, slow down and an immediate stop button. On the sweeper's device, a line of LED lights or other visual indicators to show a level for the sweeper to sweep at. The skip will be able to adjust the level of the sweeper, and the sweeper will be able to see the level they should be sweeping at.

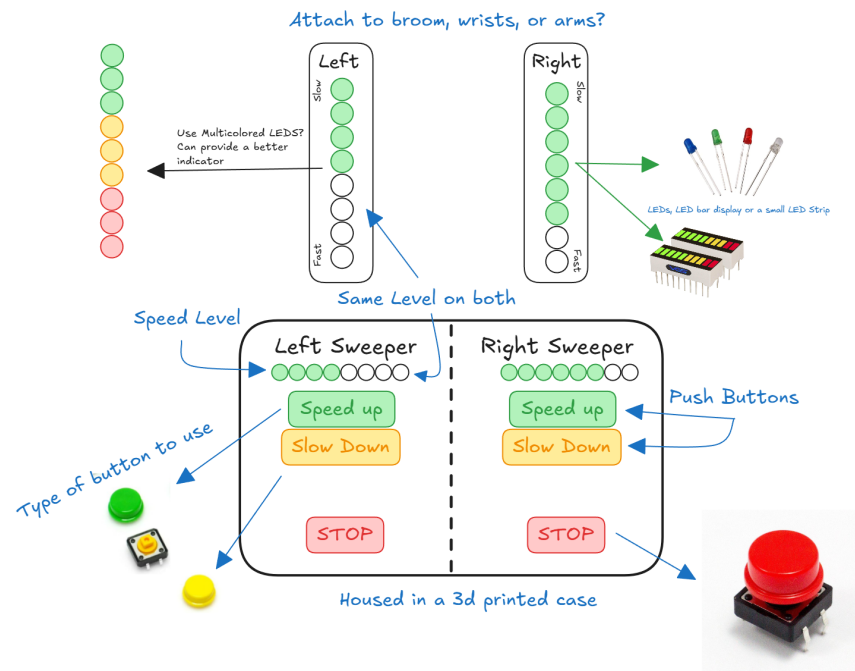


Figure 2: Rough sketch of the design

The communication between the devices can be done via WiFi as it would have sufficient range to cover the play area. A packet/message structure will need to be defined to ensure the messages are read correctly and by the proper device. A simple message structure could be as follows:

```
<sweeper_id, message_data>
```

Where **sweeper_id** can be a 1 bit for which sweeper, if more than 2 sweepers are present, then we could use 2, 3, 4 bits to represent the sweeper. The message data for each message can be as simple as the level of speed to set the sweeper to, and the sweeper would reply back with an ACK message of the current level they are at to ensure proper synchronization. A simple message-communication diagram is shown below.

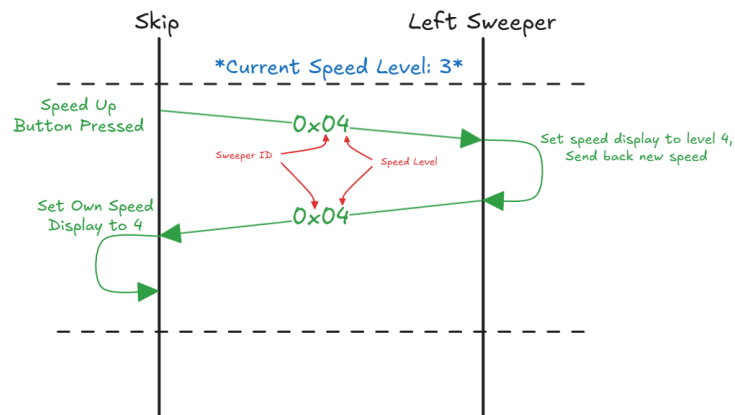


Figure 3: Message communication diagram

Both sweepers will actively listen to all messages and only act when the `sweeper_id` matches their own. This will ensure that the skip can communicate with both sweepers at the same time.

A Compact Design

January 28, 2025

A more compact device for the skip could be used, where a switch can be used to toggle between the two sweepers. This would reduce the size of the device, and essentially reduces the number of components to just half. On the sweepers device a small vibration motor can be used to indicate a message has just arrived. A rough sketch is shown below for the more compact design.

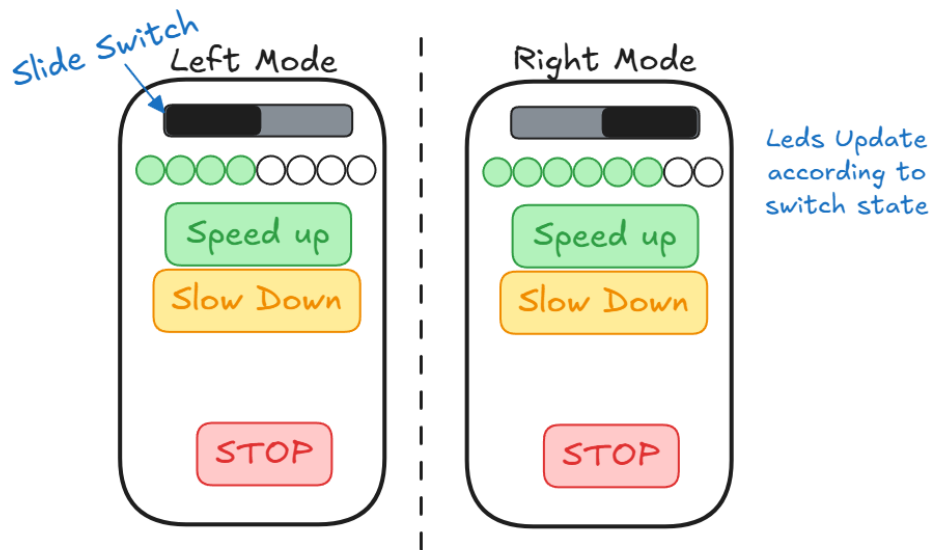


Figure 4: More compact design idea

The downside I can see with this is that the skip would need to swap between the two sweepers, which could be a bit cumbersome at times. However, the size reduction could be worth it, and reduced cost of the device. It also will have less LEDs so power consumption could be reduced as well.

2 LOGBOOK #2 START

Design Review 1 Reflection

February 3, 2025

After the feedback of the initial design review, we have altered our design. Our big oversight was the huge amount of granularity we had in our design. Coming up with multiple sweeping speeds is unnecessary and we have learnt that there is really only 3 levels, sweep hard, clean the ice and stop. Outside of this, the curlers will know what to do.

We also have decided to modify our design to contain simpler commands and share the same command to both sweepers. This means that if the skip wants the stone to curl left, we can send the curl left command to both sweepers, and the sweepers will know what to do to make the stone curl left. This will simplify the design and make it easier to implement.

Lab 2***February 3, 2025***

The goal of this lab was to get familiar with the ESP32 and the ESP-Now protocol. We first decoded the positions of a joystick and then controlled LEDs on a ring based on the position of the joystick. The controller was connected to the board at 2 different ADC pins and we read the values of the x and y direction to determine the location. At each position, 8 samples were taken and averaged to get a more resistant value. The x, y values were then used to decode where the joystick was pointing and the LEDs were lit up accordingly.

To light up the LEDs I had used the adafruit neopixel library, allowing me to light any of the 12 LEDs on the ring. The position of the joystick was encoded by a value of 1-9, which represented a position on a grid that was later used to communicate with another device using ESP-Now. The encoding scheme was as follows:

NW	N	NE
9	8	7
W	C	E
6	5	4
SW	S	SE
3	2	1

Table 1: Joystick encoding scheme

A function was created which took a direction value, 1 through 9, and lit up the corresponding LED(s).

In the last part of the lab, we had obtained the mac address of a differnet device to communicate via ESP-Now. We established a connection between the sender and receiver and then send a message containing the encoded joystick position to the receiver. The receiver would then light up the corresponding LED on the ring. The communication was successful and the LEDs lit up as expected. There was a short delay added as to not overload the network with messages.

Questions for Lab 2

1. The joystick was powered by the 3.3V and grounded by the GND pin. Two ADC1 pins were used to read the x and y values of the joystick. For the LED ring, the data pin was connected to a GPIO pin on the ESP32, and powered in a similar fashion to the joystick.
2. The folow of the program is as follows:
 - Connect with receiver via ESP-Now
 - Initialize the LED ring
 - Main Loop:
 - Sample x and y values 8 times and average them
 - Decode the position of the joystick into one of 9 positions
 - Send the position to the receiver via ESP-Now
 - Receive the position and light up the corresponding LED
3. To improve the usability of the device, a difference decoding scheme from the coordinates to position could be used. Currently there exists some deadzones where the joystick is close to one position but LEDs do not light up.

4. There were no major performance issues, aside from the added delay as to not overload the network. The LEDs lit up as expected and the joystick was able to control the LEDs. Doing a serial print, it was able to detect and decode the position in a very fast manner.
5. Technical questions:
 - a. The transmitter needs to know the MAC address of the receiver because ESP-NOW does not use IP addresses like regular Wi-Fi. Instead, devices communicate directly using their unique MAC addresses. This helps the sender know exactly which device to send data to. Without the MAC address, the transmitter wouldn't know where to send the message. There is an option to broadcast the message to all devices by using the mac address of 0xFF.
 - b. We are not able to determine if the message was received, only if it was sent successfully. To ensure the message was received, we could implement a handshake protocol where the receiver sends an ACK message back to the transmitter. This would ensure that the message was received and understood by the receiver.
 - c.
 - i. There is no way to authenticate that the packet was specifically meant for a target device if it is broadcasted. However, we can add extra details in the packet to ensure that the receiver knows that the packet was meant for them. This could be a simple ID, or a more complex encryption scheme to ensure that the packet was meant for the receiver.
 - ii. The transmitter would not know if the message was received by the target device by default. However, as discussed above, we can implement a handshake protocol.
 - iii. As more devices transmit on the network, we would have more congestion and sending messages could take longer and even fail if the network is too congested. This is why we had added an artificial delay to ensure that the network was not overloaded with messages.

Design Refinement

February 5, 2025

The new design idea contains a more simplified setup. 5 buttons are present on the skips device along with an LED ring. The sweepers device will be a simple case with the same LED ring. The commands will be sent to both sweepers at the same time. We will also show the command for a brief period before turning the LEDs off, which will allow for better power consumption. The different commands will light up different parts of the LEDs in various colors to make it as easy as possible for the sweepers to differentiate the different commands. The current command encoding for the LEDs is as follows. This is preliminary and could change as we test the design. The goal would be to make it as easy as possible for the sweepers to understand the command.

Command	LEDs
Sweep Hard	Green - Fully lit
Clean Ice	Blue - Top Half
Stop	Red - Fully lit
Curl Left	Blue - Left Half
Curl Right	Blue - Right Half

Table 2: Encoding for LEDs

PCB Assignment

February 9, 2025

The PCB assignment was done successfully without any errors. Initially I had picked the EasyEDA pro option, but then switched to the std edition as it was easier and more intuitive to use. The artistic element included on the PCB was a meme of a guy pointing towards the required information on the back of the board.

The design of the PCB is from class, a Human-Machine-Interface board with push buttons, an analog input and a screen. We utilize an OLED screen, 2 push buttons, a potentiometer and a status LED. The screen is controlled by an I²C interface, the push buttons are connected to two separate pins, and the potentiometer is connected to a different pin. A generic 8-pin header is also present to connect to a MCU to control the board.

PCB Assignment - Bill of Materials

February 11, 2025

The bill of materials with sources for the PCB we designed as follows:

Part	Quantity	Price	Total	Source
Capacitor	3	\$0.03442	\$0.10326	DigiKey
Header Pin	1	\$0.11134	\$0.11134	DigiKey
Push Button	2	\$0.62256	\$1.24512	DigiKey
LED	1	\$0.18192	\$0.18192	DigiKey
Resistors	5	\$0.00619	\$0.03095	DigiKey
Potentiometer	1	\$1.40968	\$1.40968	DigiKey
OLED Screen	1	\$11.99	\$11.99	Waveshare
Total			\$15.07	-

Table 3: Bill of Materials

The OLED screen was sourced from waveshare and is of the same specification as the one used in the PCB. It is a 1.54 inch OLED screen with a resolution of 128x64 pixels with an I²C interface. Other parts were sourced from DigiKey, and were picked to fit the footprint on the PCB and the requirements of the board.

3 LOGBOOK #3 START

Design Idea - Message Encoding

February 17, 2025

Figure 5 shows how each message would be displayed on the ring. Colors and position of the LEDs are used to indicate the message. Common colors are used, as green to sweep as it normally signals go, red to stop, and orange to clean. For the curl and line commands, the right and left halves are lit up respectively. For curl we use the color blue and yellow for, which can be easily differentiated by the sweepers.

One potential problem is that depending on the LEDs, orange and yellow may be hard to differentiate, so we may need to change the colors for the line and clean commands.

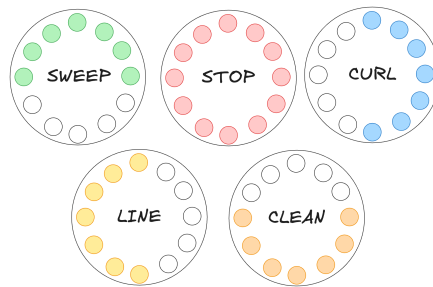


Figure 5: Message encoding on the ring

Design Idea - Sweeper and Skip Device

February 20, 2025

Figure 6 shows a potential design for a more refined skip device. It is a circular device with 5 buttons on the front and an LED ring around the edge. It would have a hook at the top to allow for a lanyard or string to be attached.

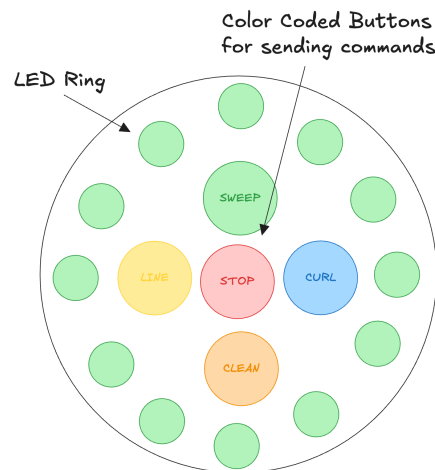


Figure 6: Skip device design

For the receiver side, the sweepers device could be similar to the curling timers that get attached onto the brooms. The idea is to have the device that can be positioned at various lengths on the broom, and can be easily seen by the sweepers. The led ring would be angled slightly towards the sweeper to make it easier to see. Similar design concept to Figure 7, found on the internet.



Figure 7: Curling timer design

Design Idea - Sweeper device sketch

Feb 22, 2025

Figure 8 shows a rough sketch of the sweeper device. As previously described, it would be a device that can be attached to the broom and would have an LED ring that would light up based on the command from the skip.

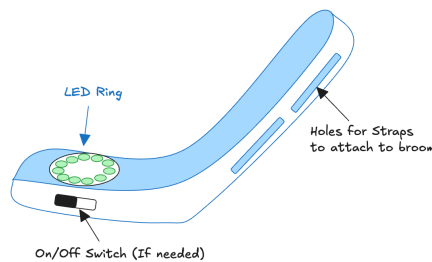


Figure 8: Sweeper device sketch

Alternatively, a plastic clip could be used, instead of straps to attach the device to the broom, however this is not a very important detail at the moment. The main focus is to have a device that can be easily attached to the broom and can be easily seen by the sweepers.

Lab 3

February 24, 2025

Lab 3 consisted of implementing power saving measures on the ESP32 with the joystick from lab 2. The goal was to see a significant improvement in the average current consumption with the power saving measures implemented. This was achieved by putting the ESP32 into a light sleep instead of a blocking delay, reducing the CPU clock frequency and WiFi power.

Light Sleep

Since the required transmission rate was every 150ms, a majority of the time was spent doing nothing, about 145ms of waiting and 5ms of processing. This was a perfect use case for light sleep, as the ESP32 could go into a this low-power idle state and wake up every 150ms to transmit the data. The delay was set to 145ms and once transmitted, the wifi was turned off and the ESP32 went into light sleep.

Reducing CPU Clock Frequency

Testing the power consumption at different CPU frequencies, it was found that the lowest frequency at which the radio was also operational showed the lowest power consumption. The frequency was set to 80MHz, which was the lowest frequency at which the radio was operational.

Reducing WiFi Power

The WiFi power was reduced to the lowest setting, which reduced the total power consumption of the ESP32.

Results

The results of the power saving measures are shown in Figure 9. The baseline power consumption was around 129.44mA, and with the power saving measures implemented, the power consumption was reduced to around 16.21mA. This was a significant improvement in power consumption and showed that the power saving measures were effective.

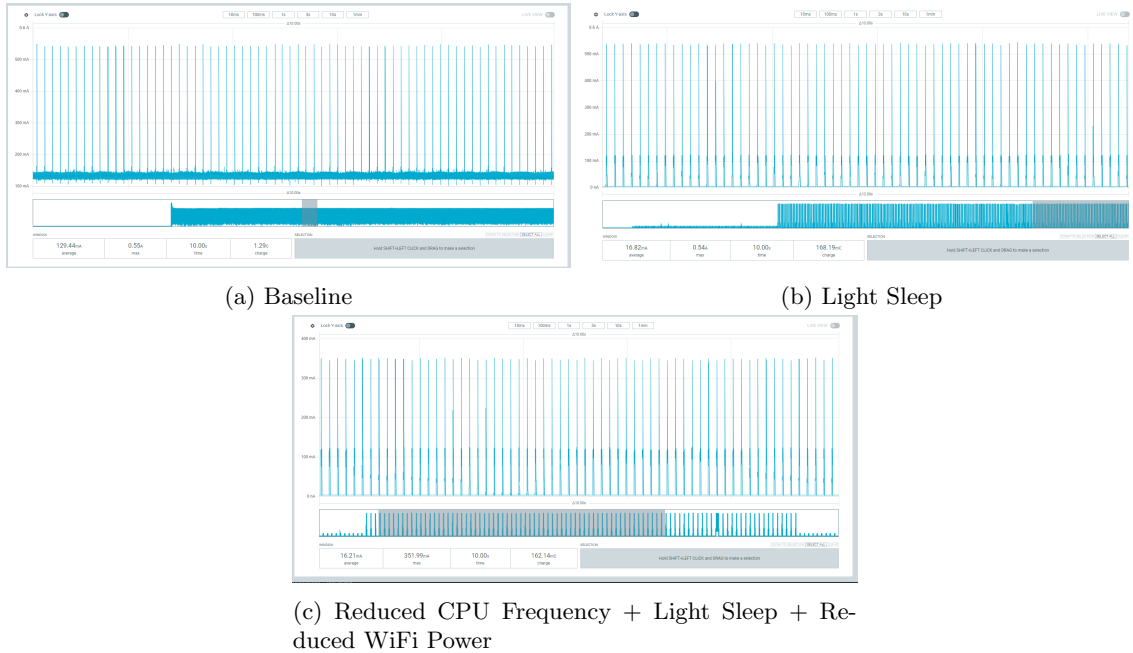


Figure 9: Current consumption

We will calculate the worst-case scenario for power consumption on our device. We have some push buttons which are negligible, a 12-LED ring which will consume 30mA per LED at max brightness, and 1 mA on idle.

The sender will include the 12-led ring which gives us an idle power consumption of 12mA, and adding a little bit more for the light-sleep mode, we can assume roughly 15mA on idle. When a push button is pressed, the LEDs will light up and the message will be sent to the receiver. Assuming all LEDs are lit up, and using only 1 channel at half brightness, that is a single color (r, g or b) at 50% brightness, we can calculate the power consumption as follows:

$$\text{LED Power} = 12 \text{ LEDs} \times 10 \text{ mA} / \text{LED Channel} \times 1 \text{ Channel} \times 0.5 \text{ Brightness} = 60 \text{ mA}$$

Since we only transmit when a button is pressed, the active power consumption will be slightly higher than 60mA, so we can assume around 65mA, which would include the power consumption of the ESP32 and the radio to transmit the signal.

On the receiver side, we only have the 12-LED ring. Without any complex power-saving measures, the receiver will continuously listen for messages and light up the LEDs. This would have the idle power consumption of 30-35mA, and when a message is received, the LEDs will light up. Assuming the same power consumption for the LEDs and slightly higher for actively listening for messages, we can assume around 75mA.

The reason for this 30-35mA idle consumption on the receiver is that we are always listening. To get past this limit, we would need to implement a synchronization method so that the receiver can sleep when it is not expecting a message. This would require something like a handshake protocol to synchronize the time and also utilize a consistent message sending rate.

If we de-age the LiPo battery to 80% of its capacity, we can calculate the battery life as follows:

$$\begin{aligned}\text{Sender Battery Life} &= \frac{500 \text{ mAh} \times 0.8}{65 \text{ mA}} = 6.15 \text{ hours} \\ \text{Receiver Battery Life} &= \frac{500 \text{ mAh} \times 0.8}{75 \text{ mA}} = 5.33 \text{ hours}\end{aligned}$$

This is the worst-case scenario where the device is continuously transmitting and receiving messages. In a real-world scenario, the battery life would be much longer as the device would not be continuously transmitting and receiving messages. The following plot shows the battery life of the device with various active percentages.

We plot the following equation, where a = active percentage:

$$\begin{aligned}\text{Sender Battery Life} &= \frac{500 \text{ mAh} \times 0.8}{a \cdot 65\text{mA} + 15\text{mA}(1 - a)} \text{ hours} \\ \text{Receiver Battery Life} &= \frac{500 \text{ mAh} \times 0.8}{a \cdot 75\text{mA} + 35\text{mA}(1 - a)} \text{ hours}\end{aligned}$$

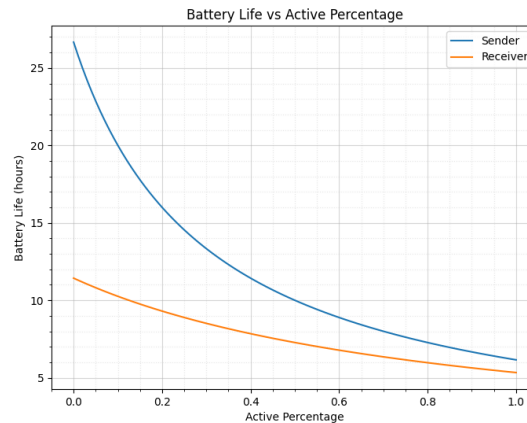


Figure 10: Battery life of the device

If we assumed that it would be active 50% of the time, we get a battery life of 10 hours on the sender and 7.27 hours on the receiver. This is a more realistic scenario where the device would not be continuously transmitting and receiving messages.

3D Design Idea

February 28, 2025

A 3D case was made for the idea in Figure 6. It would be a small hand-held circular device with 5 buttons on the front and an LED ring around the edge. Figure 11 shows the design of the device. An addition to this design is to add a hole on the side for a powerswitch, and depending on the battery type, a charging port. If a non-rechargeable battery is used, then the two halves can be twisted open like a screw. A ring for a lanyard is also present at the top of the device.

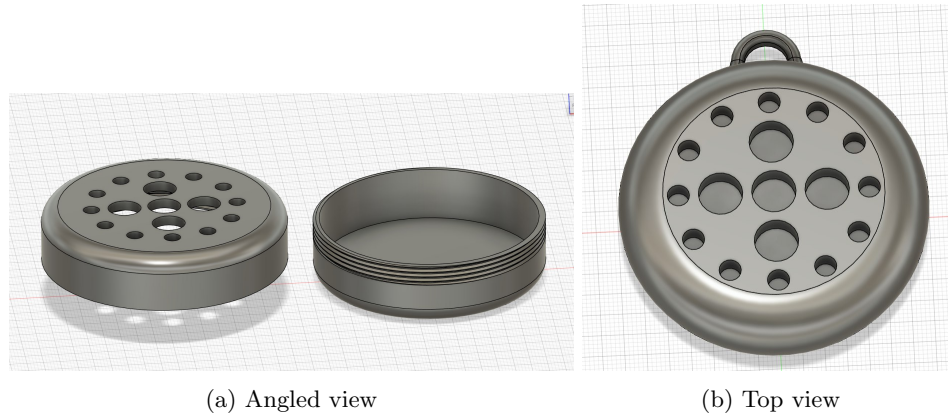


Figure 11: 3D design of the device

Lab 3 - Optimizations

March 4, 2025

Previously, the way the ESP-32 went to sleep was not optimized. It went to sleep after a fixed delay which caused issues when the data was not properly received. To ensure the sleep was done at the proper time, a callback was used to set off a flag when the data was successfully transmitted. Once the flag was set, the ESP32 WiFi would be turned off and the light sleep would start with a calculated sleep time.

The sleep time is also now dynamic, where we find the amount of time processing and sending takes and subtract that from the total time we want to sleep. This ensures that the ESP32 wakes up at the proper time to send the data. That is:

$$\text{Sleep Time} = 150\text{ms} - (t_{\text{end}} - t_{\text{start}})$$

Where t_{end} is the time the data was received and t_{start} is when the esp initially Woke up. These times are found by utilizing the `millis()` function.

The results of this is shown in Figure 12. This was also tested on a different board so the results are not directly comparable to the previous results, but were still significant. The power consumption was reduced to around 9.17 mA, which was a significant improvement over the previous results.

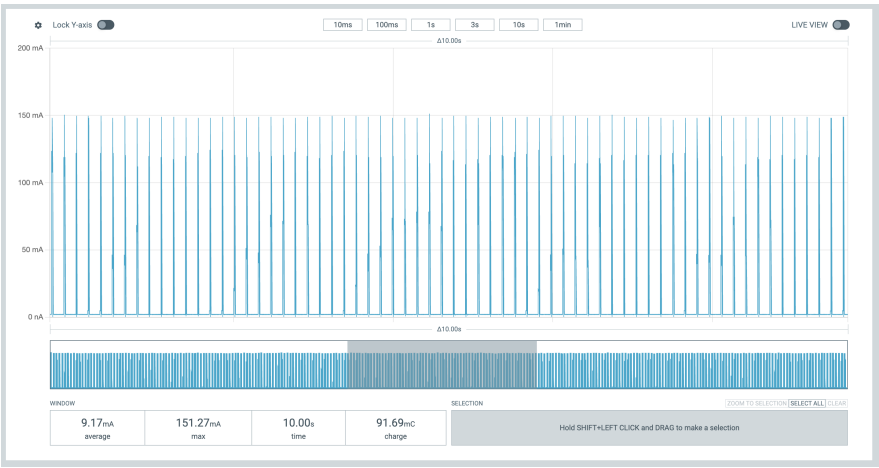


Figure 12: Current consumption with optimized sleep