VoidLink

Electrical and Computer Engineering
Capstone Project

September 2024 – April 2025

Team S7 - RF DOOM

Lukas Kotuza-Janisch, 400238647, kotuzajl@mcmaster.ca
Stefan Praniauskas, 400315373, praniaus@mcmaster.ca
Milena Thibault, 400315606, thibaulm@mcmaster.ca
Boran Seckin, 400305852, seckinb@mcmaster.ca

Introduction

Modern communication technologies such as cellular networks, Wi-Fi, and GPS have become critical, but depend on centralized infrastructures. In environments like dense forests, remote deserts, or underground caves, where such infrastructure is inaccessible or nonexistent, traditional communication methods fail. This project addresses that gap. VoidLink is a decentralized mesh communication system for off-grid environments, using LoRa to enable low-cost, portable nodes that relay messages and location data without any external infrastructure. VoidLink offers a self-sustaining alternative to satellite phones for adventurers, geologists, and emergency responders.

VoidLink can be applied in a range of scenarios, from outdoor recreation and scientific fieldwork to disaster response; VoidLink is designed for when off-grid communication is critical. Its low cost, decentralized design makes it accessible for individuals and communities without infrastructure access.

VoidLink has the potential to support communication when infrastructure fails, enhance safety in remote environments, and provide a reliable alternative in situations where traditional systems are unavailable or too costly.

Technical Breakdown

VoidLink is divided into four main modules—User Interface (UI), Networking, and Hardware, with hardware responsibilities split into two key areas: overall schematic/layout and power simulations/RF design.

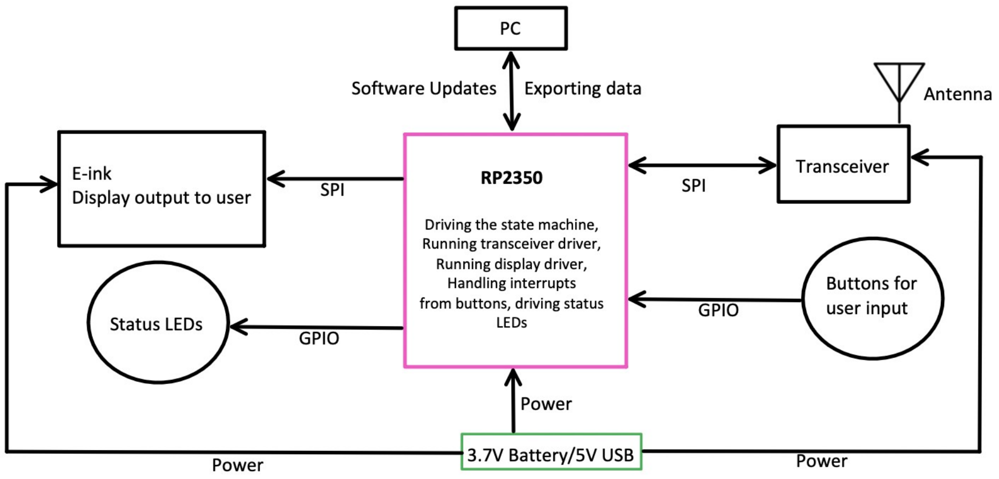


Figure 1: System Block Diagram

All four modules work together to make VoidLink run (see Figure 1: System Block Diagram). The UI lets users see messages, check nearby nodes, and control the device. It shows info from the network, which sends and receives messages using LoRa. The hardware layout connects everything on a custom PCB, wiring up the buttons, screen, and chip. The power and RF part makes sure the device runs on both USB and battery, and that the antenna works well for sending signals.

Module I: User Interface (UI), Lukas Kotuza-Janisch, 25% of Overall Project:
The UI module manages interaction between the user and the device through an E-Ink display and six programmable buttons. It provides menus for viewing nearby nodes, received messages, device status, and settings. The UI integrates outputs from the networking module to display relevant information and sends user actions back into the system for processing. It also involved mechanical considerations, such as case design and 3D printing for fit and functionality.

Module II: Network, Boran Seckin, 25% of Overall Project:
This module implements VoidLink's custom communication protocol using a LoRa transceiver. It supports message routing, acknowledgements, and mesh networking through a layered architecture. Messages are encoded, transmitted, and relayed across nodes with minimal overhead, enabling decentralized communication over long ranges without infrastructure.

Module III: Hardware System Schematic and Layout, Stefan Praniauskas, 25% of Overall Project:
This section was responsible for designing the complete schematic and PCB layout, including MCU implementation, component selection, GPIO assignment, debounced push button inputs, and display integration.

Module IV: Hardware Power Simulation and RF Design, Milena Thibault, 25% of Overall Project:
This part of the hardware section was responsible for ensuring the power circuitry would work as expected and integrating the transceiver implementation. It involved simulating the power circuitry using PSpice, managing transitions between USB and battery power, and designing the RF section with impedance-matched traces for the SX1262 transceiver. The RF layout was optimized for signal integrity and thermal performance.

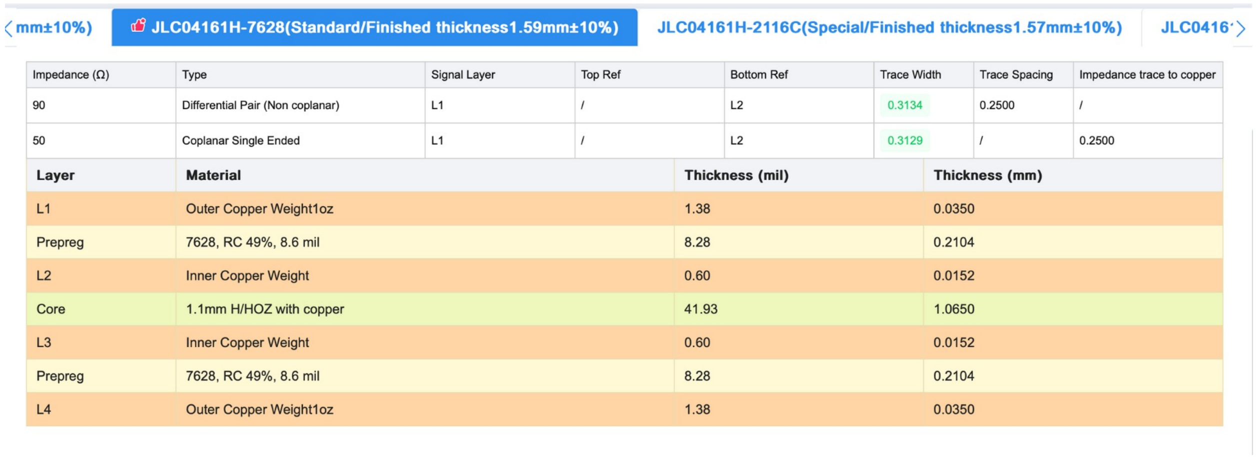
Hardware Module (Stefan Praniauskas, Milena Thibault, 50% of Overall Project)

While hardware responsibilities were initially split between system schematic/layout and power simulations/RF design, the technical discussion is merged in the report due to their interconnection. Combining the hardware discussion offers a clearer view of the system's overall integration.

Revision 1

Impedance Matching (Stefan Praniauskas, 2.5%)

February to March (2 Hours)



mm±10%) | JLC04161H-7628(Standard/Finished thickness1.59mm±10%) | JLC04161H-2116C(Special/Finished thickness1.57mm±10%) | JLC0416

Impedance (Ω)	Type	Signal Layer	Top Ref	Bottom Ref	Trace Width	Trace Spacing	Impedance trace to copper
90	Differential Pair (Non coplanar)	L1	/	L2	0.3134	0.2500	/
50	Coplanar Single Ended	L1	/	L2	0.3129	/	0.2500

Layer	Material	Thickness (mil)	Thickness (mm)
L1	Outer Copper Weight1oz	1.38	0.0350
Prepreg	7628, RC 49%, 8.6 mil	8.28	0.2104
L2	Inner Copper Weight	0.60	0.0152
Core	1.1mm H/HOZ with copper	41.93	1.0650
L3	Inner Copper Weight	0.60	0.0152
Prepreg	7628, RC 49%, 8.6 mil	8.28	0.2104
L4	Outer Copper Weight1oz	1.38	0.0350

Figure 2: JLCPCB Impedance Calculator Results [1]

There are two signals that require impedance matching. Our manufacturer, JLCPCB, provided their own impedance matching calculator which was used to determine the required trace widths. The first signal is the USB differential pair which requires a standard 90-ohm matched impedance [2]. A trace spacing of 0.25mm was used. The second is the RF signal, which requires a standard 50-ohm matched impedance [3]. This signal is coplanar due to the ground pour in the RF area, in which a trace to copper spacing of 0.25mm was selected.

Power Circuitry (Stefan Praniauskas, %5)

December to March (48 Hours)

The power circuitry of the system is designed to ensure uninterrupted power delivery from any available power source, whether from the battery, USB, or both. The system operates primarily on a 3.7V lithium-ion battery (Adafruit, P/N: 1578), which serves as the main power source when USB is not connected. When a USB connection is present, the device is powered directly from the 5V USB supply while simultaneously charging the battery. This design was developed with load-sharing principles [4] to seamlessly transition between power sources, ensuring reliable operation and continuous power to the RP3050 without brownout or reset conditions.

The power circuitry ensures that the input voltage remains within the acceptable range for system operation. To regulate the voltage for system components, the TPS62811QWRWYRQ1 buck converter is used to provide a stable 3.3V supply [5]. This converter operates within an input voltage range of 1.8V to 6.5V, allowing it to efficiently step down the voltage from either the battery or USB power source [5].

The battery charging IC (MCP738312) is used to manage battery charging, ensuring a steady charge while preventing overcharging [6]. This IC regulates the charging process to maintain battery health and efficiency when the USB power source is available [6].

To maintain battery health and efficiency when the USB power source is available, this configuration uses two back-to-back PMOS transistors (DMP1045U-7) with their sources tied together to manage the transition between USB and battery power [4]:

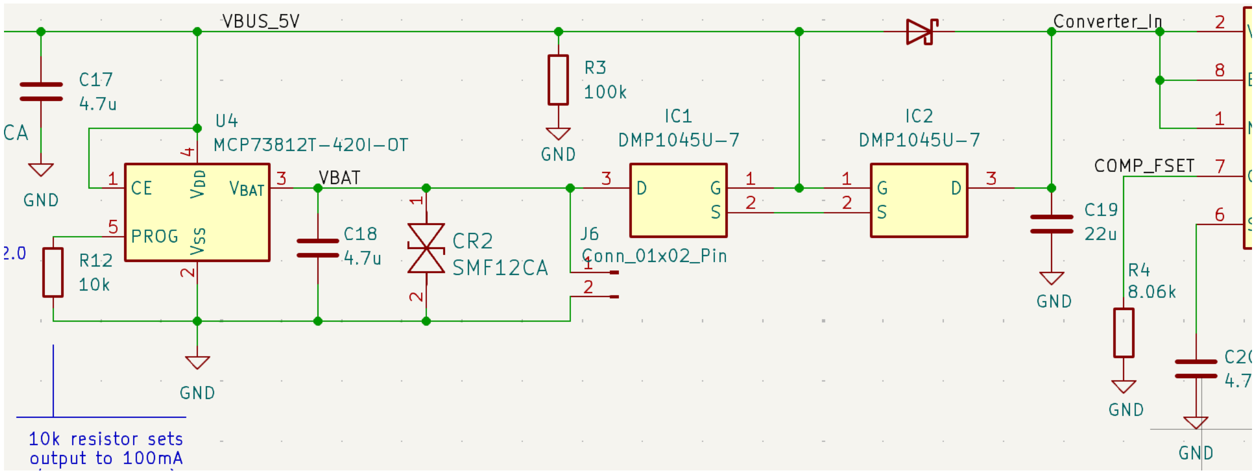


Figure 3: Battery Charging & Power Source Selecting Schematic [4]

When USB Power is Present (VUSB is Connected):

- PMOS1 (IC1) Behavior: VUSB is applied to the gate of PMOS1, driving VGS high, which causes PMOS1 to enter cutoff and not conduct. This disconnects VBATT from the circuit.
- PMOS2 (IC2) Behavior: VUSB is applied to the gate of PMOS2, which causes VGS to be high, also turning PMOS2 off.
- MCP73812T-420I-OT receives power and can charge the battery when it is connected.
- Result: The circuit is powered by VUSB through the Schottky diode, allowing current from VUSB to flow to the load and charge the battery simultaneously.

When USB Power is Absent (USB is Disconnected):

- PMOS1 (IC1) Behavior: With the gate of PMOS1 at a lower voltage (~0V), VGS becomes negative, turning PMOS1 on and allowing VBATT to supply power to the circuit.
- PMOS2 (IC2) Behavior: The gate of PMOS2 is at 0V, and since PMOS2's **source** is connected to VBATT (via PMOS1), PMOS2 conducts. The **drain** of PMOS2 mirrors the voltage at its source (VBATT). With VGSVGS negative, PMOS2 turns **on**, allowing VBATT to supply power to the load while preventing reverse current from flowing back to VUSB.
- Result: The circuit is powered by VBATT.

This back-to-back PMOS configuration ensures efficient power switching, preventing leakage currents and reverse conduction [4]. PMOS1 controls the switch to battery power when USB power is unavailable, while PMOS2 prevents reverse current. The Schottky diode allows VUSB to power the load while blocking feedback from VBATT. This setup ensures USB power is prioritized when available and seamlessly switches to battery power when USB is disconnected [4].

Additionally, TVS (Transient Voltage Suppression) diodes are placed at the two critical connection points, at the battery input and USB interface. These diodes protect the system against voltage spikes and electrostatic discharge (ESD) events.

This power management approach provides an efficient and simple solution, ensuring continuous power supply while protecting the device against power fluctuations and transient disturbances.

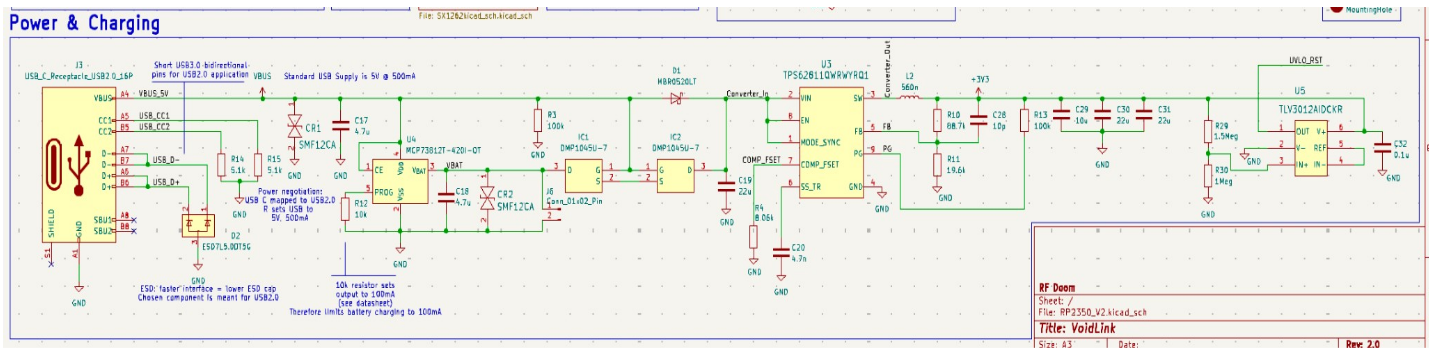


Figure 4: Full Power & Charging Schematic [4] [5] [6] [7]

Battery

A Lithium-Ion Polymer Battery (3.7V, 500mAh, P/N: 1578) from Adafruit was used to power the system. The battery is connected via a polarized 2-pin JST-PH connector, ensuring correct polarity each time it is re-plugged. Based on real-world measurements, the maximum battery life of revision 2 was 14 hours. This was a 65% improvement from revision 1, which had a maximum of 8.5 hours. See Revision 2 section for more details.

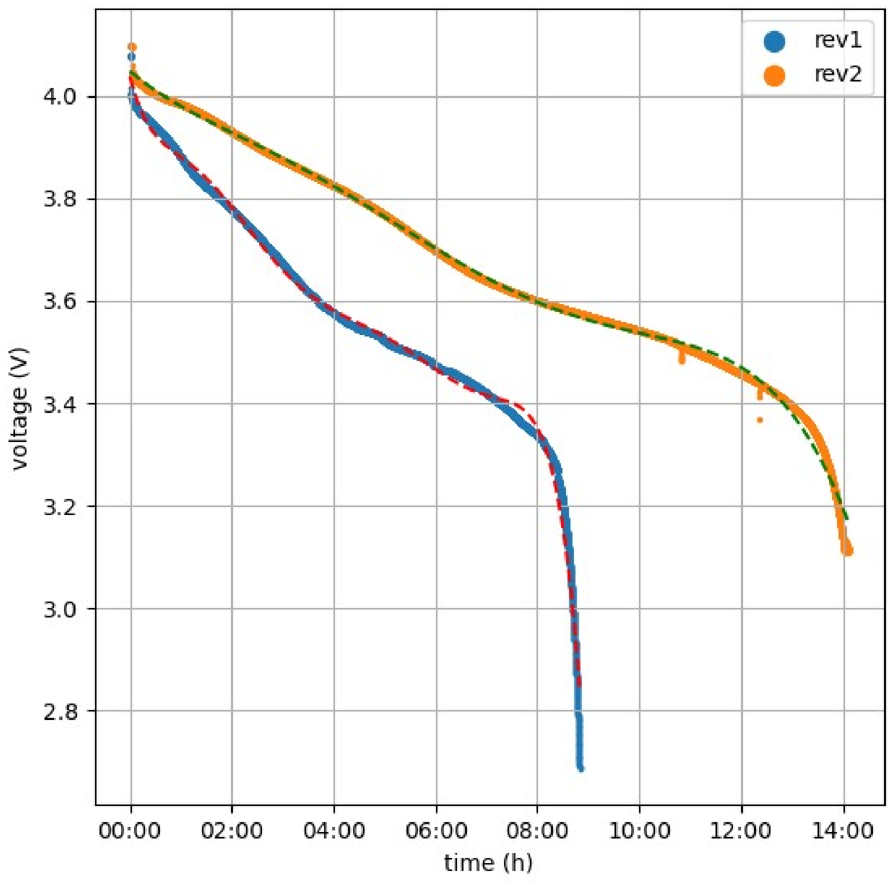


Figure 5: VoidLink Battery Discharge Test

Power Simulations (Milena Thibault, %5)

January to February (35 hours)

The power circuitry was simulated with PSpice to validate functionality and performance:

The TLV3012B comparator was configured with a resistor divider to assert reset when the input voltage drops below 3.13V, matching the RP3050's minimum operating requirement [7] [8]. This causes the RP3050 to enter reset at 3.1V, preventing undervoltage operation. When both power sources are disconnected, the voltage drops to ~0.5V, and the system remains in reset as intended.

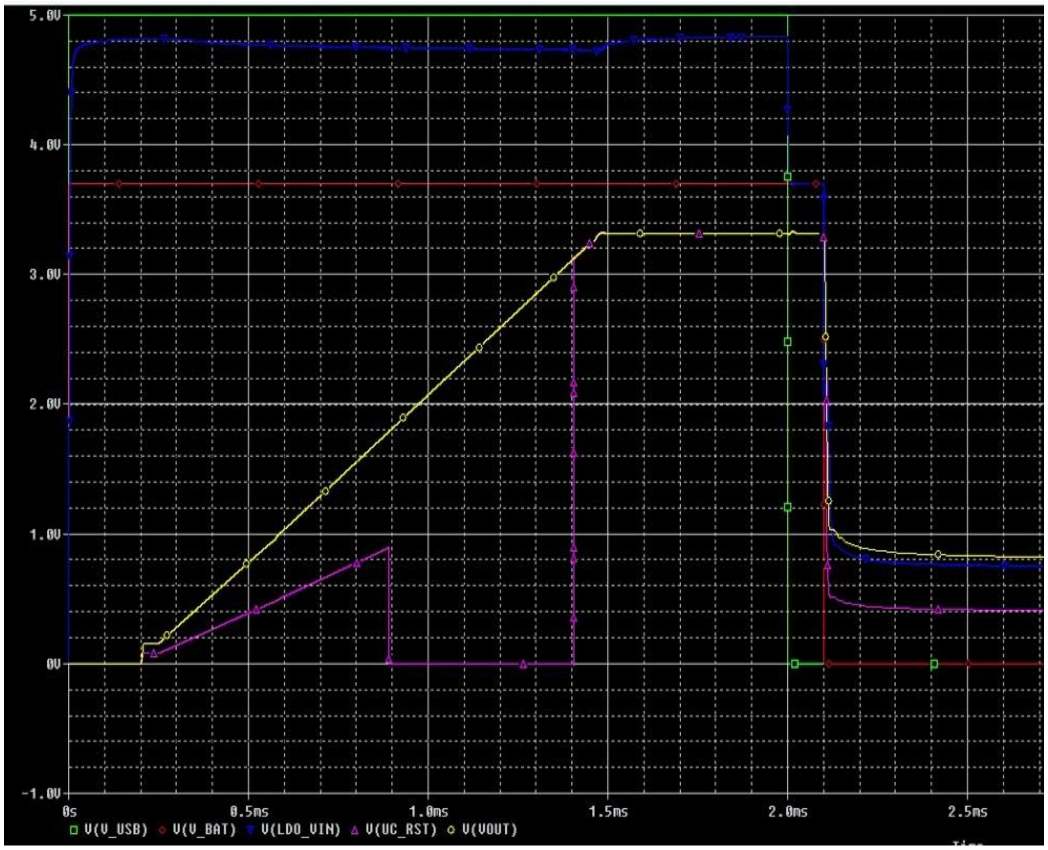


Figure 6: PSpice Power Simulation 1, USB and Battery Connected -> Disconnect USB -> Disconnect Battery

VUSB: plugged in, disconnected at 2.0ms.

VBATT: plugged in, supplying 3.7V until disconnected at 2.1ms.

LDO_VIN: Drops from 5V to 3.7V when USB is disconnected, then gradually drifts to 0.9V once both power sources are removed.

MCU_RESET: Roughly follows Vout (system 3.3V) but drops to 0V when Vout is between 0.9V and 3.1V. Since the RP3050 requires a minimum input voltage of 3.13V, it enters reset at 3.1V, preventing under voltage conditions. Once both power sources are disconnected, it drifts to 0.5V. This behavior is functioning as expected.

VOUT (system 3v3): Takes just under 1.5ms to ramp up to a stable 3.3V and experiences only a minor voltage dip (0.05V) when USB is unplugged, maintaining a steady 3.3V supply. Once both power sources are removed, it gradually drifts to 0.9V.

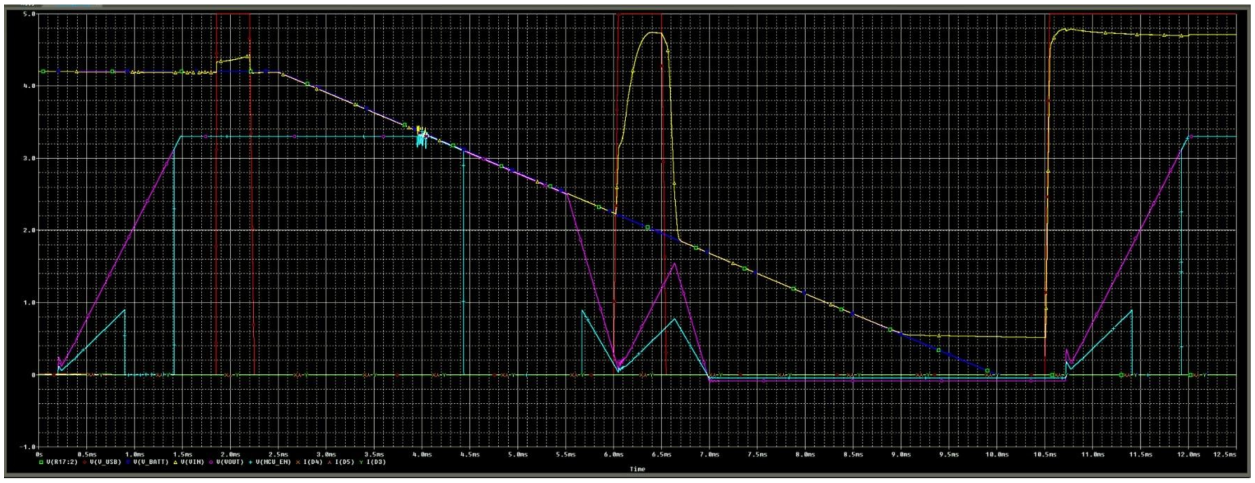


Figure 7: PSpice Power Simulation 2, USB and Battery Connected -> Disconnect USB -> Disconnect Battery

VUSB: plugged in and unplugged at 1.8ms -2.2ms, 6.0ms-6.6ms and plugged in at 10.5ms.

VBATT: Remains at its maximum of 4.2V until it begins descending at 2.5ms, until it reaches 0 at 10ms.

MCU_RESET: Roughly follows Vout (system 3.3V) but drops to 0V when Vout is between 0.9V and 3.1V. Since the RP3050 requires a minimum input voltage of 3.13V, it enters reset at 3.1V, preventing under voltage conditions. This behavior is functioning as expected.

VIN_LDO: VIN_LDO follows the highest available voltage, whether from USB or the battery, whichever is greater. This ensures that when USB is plugged in, it correctly powers the system.

VOUT (system 3v3): VOUT takes just under 1.5ms to ramp up to a stable 3.3V while powered by the battery and remains steady even when USB is connected between 1.8ms and 2.2ms. It continues to stay stable after USB is disconnected, as the battery maintains the supply, until the battery voltage drops to approximately 3.2V. At this point, VOUT starts to decline more rapidly until USB is reconnected. When USB is plugged in again at 1.8ms, VOUT begins ramping up, but since USB is not connected long enough, the voltage starts dropping again once USB is unplugged at 6.6ms. Finally, when VUSB is restored at 10.5ms, VOUT ramps up again, taking another 1.5ms to reach a stable 3.3V.

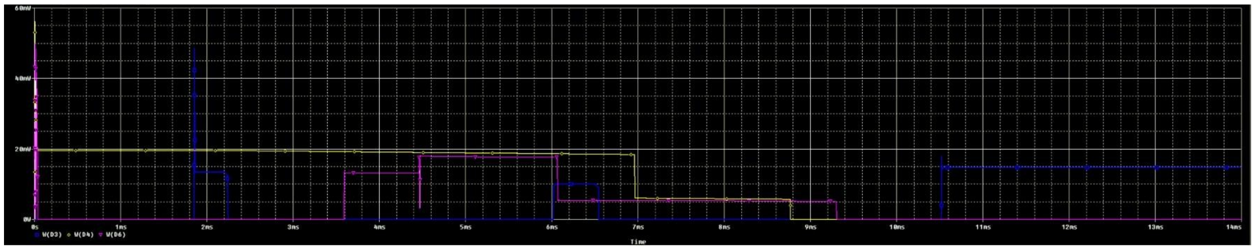


Figure 8: PSpice Power LED Simulation

USB LED Current: Turns on when the USB is plugged in, as expected.

BATTERY LOW LED: Turns on at peak current when the battery voltage drops below ~3.1V, as expected. (A slight current draw before 3.1V was corrected in the final revision).

BATTERY ON LED: Turns off when the battery voltage drops to 1.8V.

MCU Schematic (Stefan Praniauskas, 5%)

December to January (24 Hours)

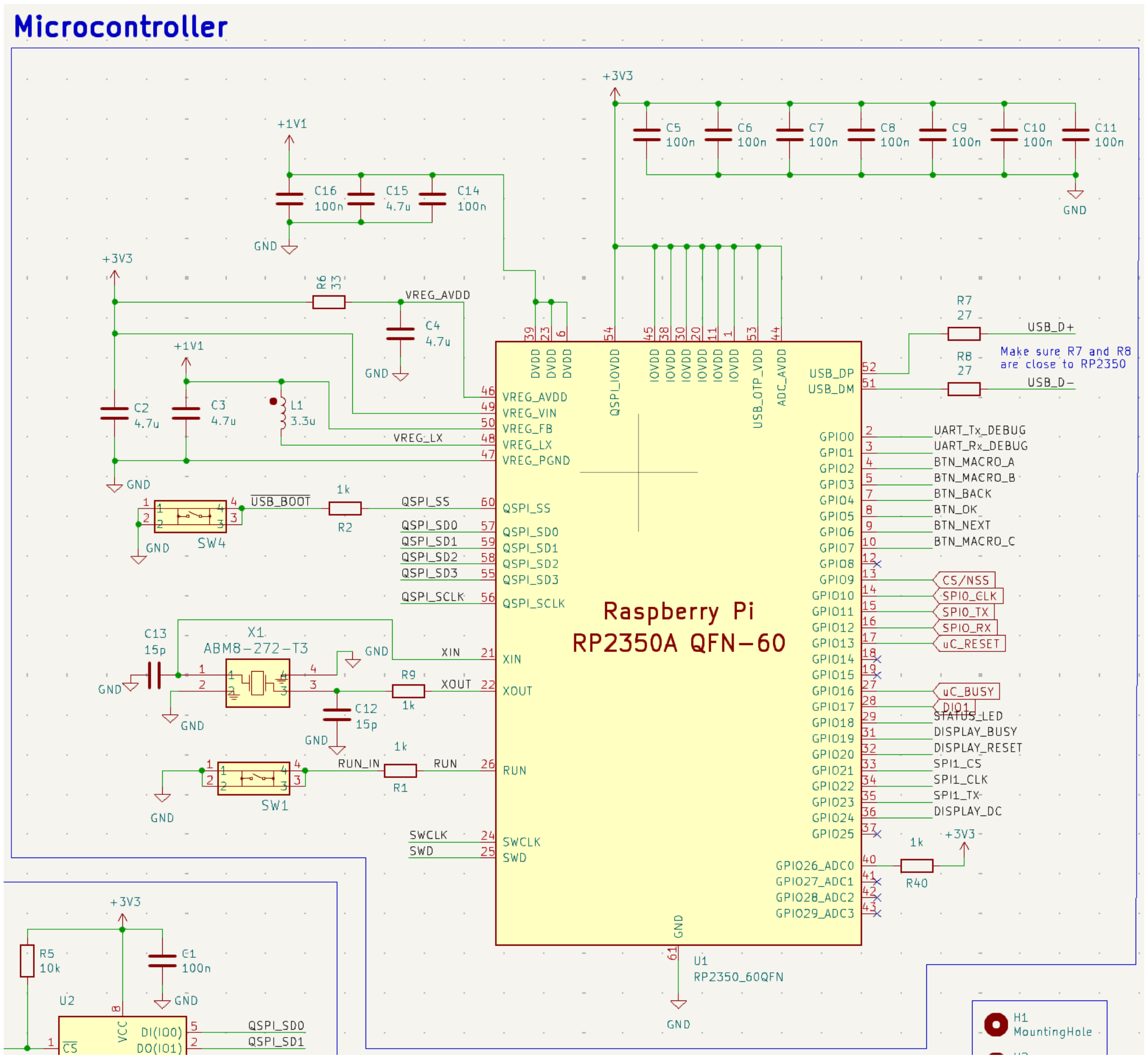
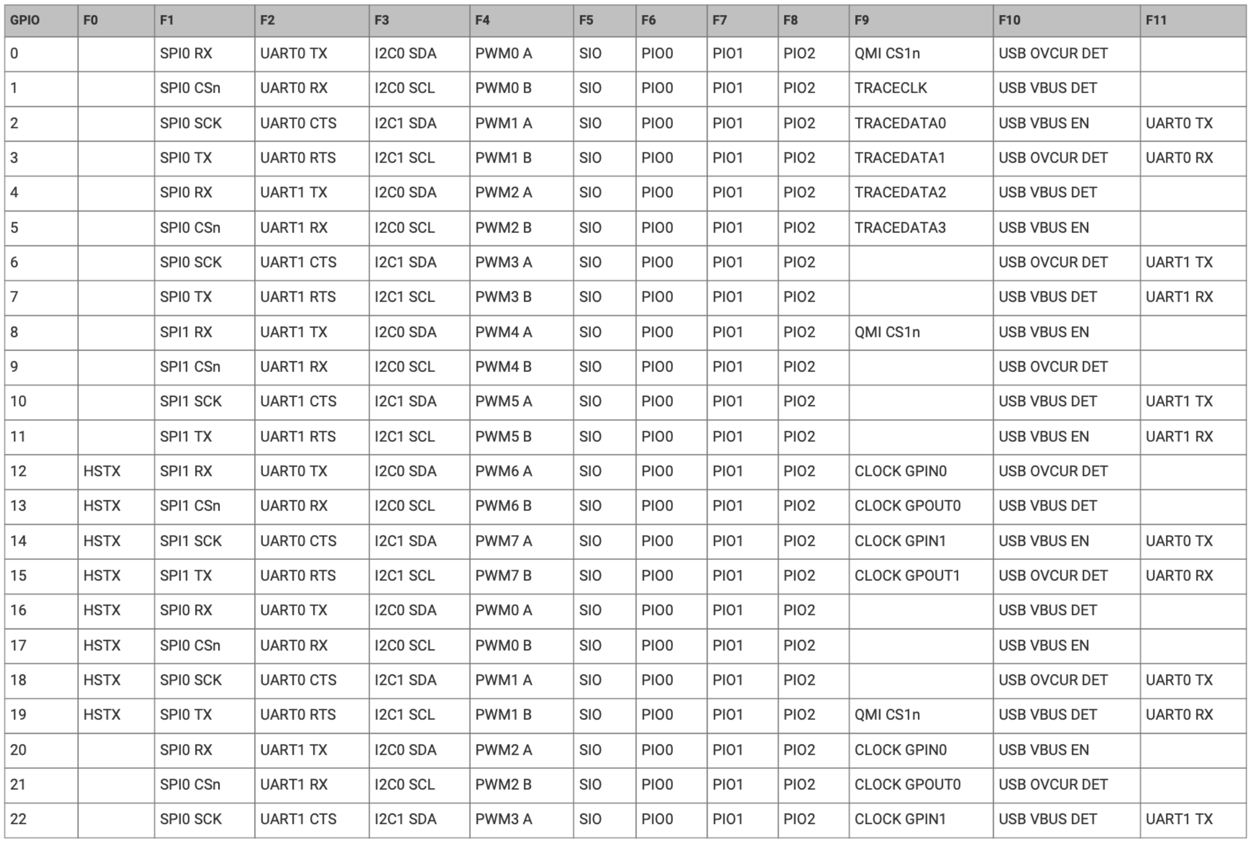


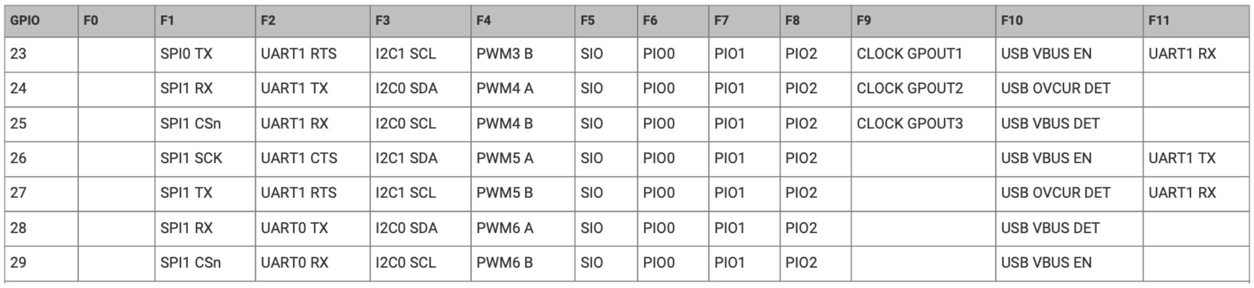
Figure 9: MCU Schematic Implementation [2]

Table 1: RP2350 GPIO Functions 1 [8]



GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
0		SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB OVCUR DET	
1		SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1	PIO2	TRACECLK	USB VBUS DET	
2		SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1	PIO2	TRACEDATA0	USB VBUS EN	UART0 TX
3		SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1	PIO2	TRACEDATA1	USB OVCUR DET	UART0 RX
4		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	TRACEDATA2	USB VBUS DET	
5		SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1	PIO2	TRACEDATA3	USB VBUS EN	
6		SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART1 TX
7		SPI0 TX	UART1 RTS	I2C1 SCL	PWM3 B	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART1 RX
8		SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB VBUS EN	
9		SPI1 CSn	UART1 RX	I2C0 SCL	PWM4 B	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	
10		SPI1 SCK	UART1 CTS	I2C1 SDA	PWM5 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART1 TX
11		SPI1 TX	UART1 RTS	I2C1 SCL	PWM5 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	UART1 RX
12	HSTX	SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN0	USB OVCUR DET	
13	HSTX	SPI1 CSn	UART0 RX	I2C0 SCL	PWM6 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB VBUS DET	
14	HSTX	SPI1 SCK	UART0 CTS	I2C1 SDA	PWM7 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN1	USB VBUS EN	UART0 TX
15	HSTX	SPI1 TX	UART0 RTS	I2C1 SCL	PWM7 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT1	USB OVCUR DET	UART0 RX
16	HSTX	SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	
17	HSTX	SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	
18	HSTX	SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART0 TX
19	HSTX	SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB VBUS DET	UART0 RX
20		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN0	USB VBUS EN	
21		SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB OVCUR DET	
22		SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN1	USB VBUS DET	UART1 TX

Table 2: RP2350 GPIO Functions 2 [8]



GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
23		SPI0 TX	UART1 RTS	I2C1 SCL	PWM3 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT1	USB VBUS EN	UART1 RX
24		SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT2	USB OVCUR DET	
25		SPI1 CSn	UART1 RX	I2C0 SCL	PWM4 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT3	USB VBUS DET	
26		SPI1 SCK	UART1 CTS	I2C1 SDA	PWM5 A	SIO	PIO0	PIO1	PIO2		USB VBUS EN	UART1 TX
27		SPI1 TX	UART1 RTS	I2C1 SCL	PWM5 B	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART1 RX
28		SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	
29		SPI1 CSn	UART0 RX	I2C0 SCL	PWM6 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	

Pinouts to the SX1262 transceiver and display had to be selected carefully to ensure the GPIO pins had the correct functionality; GPIO0 can do SPI0 RX, but not SPI0 TX or Chip select, for example [8].

MCU XTAL

A 12MHz crystal oscillator was selected as the clock for the RP2350 MCU as recommended by the RP2350 datasheet. Initially, we wanted to use a CMOS oscillator for the reduced temperature sensitivity and ease of layout, but supply chain limitations necessitated the use of a more common XTAL. The added 15p capacitors ensure predictable capacitance on the clock signal nets.

MCU On-Chip Voltage Regulator

In addition to the 3V3 power pins, the RP2350 also requires 1V1 to supply the DVDD pins. This 1V1 is generated by RP2350 with the addition of some external components. The main component is a polarized inductor (L1) which is specified in the datasheet (AOTA-B201610S3R3-101-T).

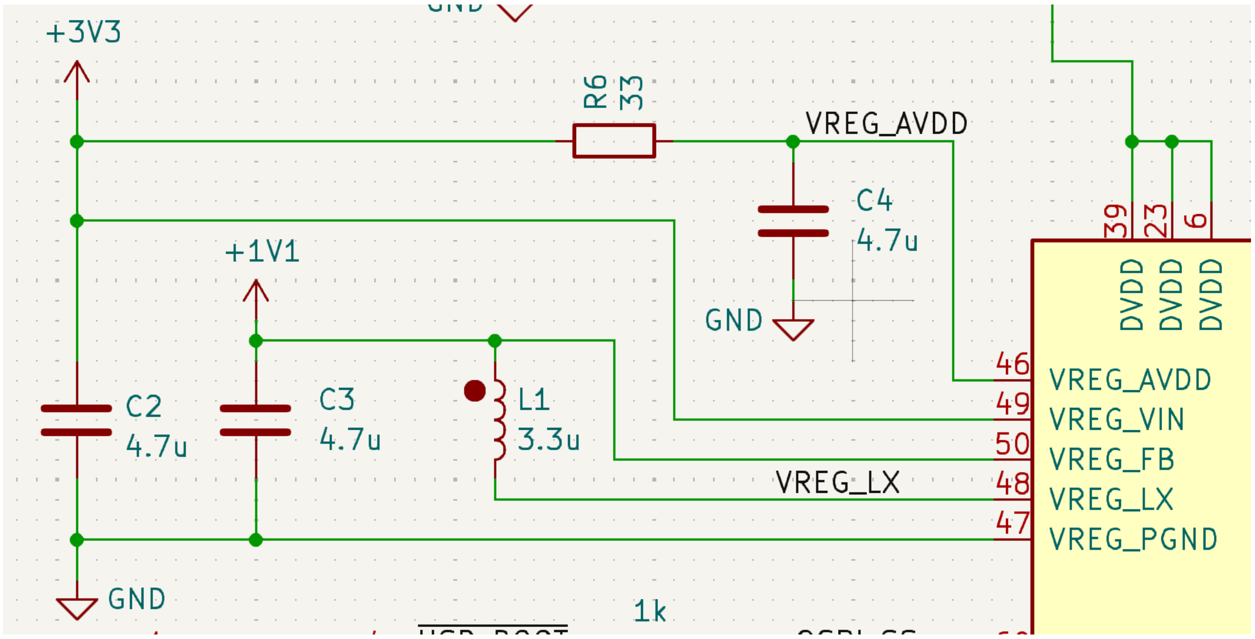


Figure 10: RP2350 VREG Schematic Implementation [2]

I/O (Stefan Praniauskas, 2.5%)

January (12 Hours)

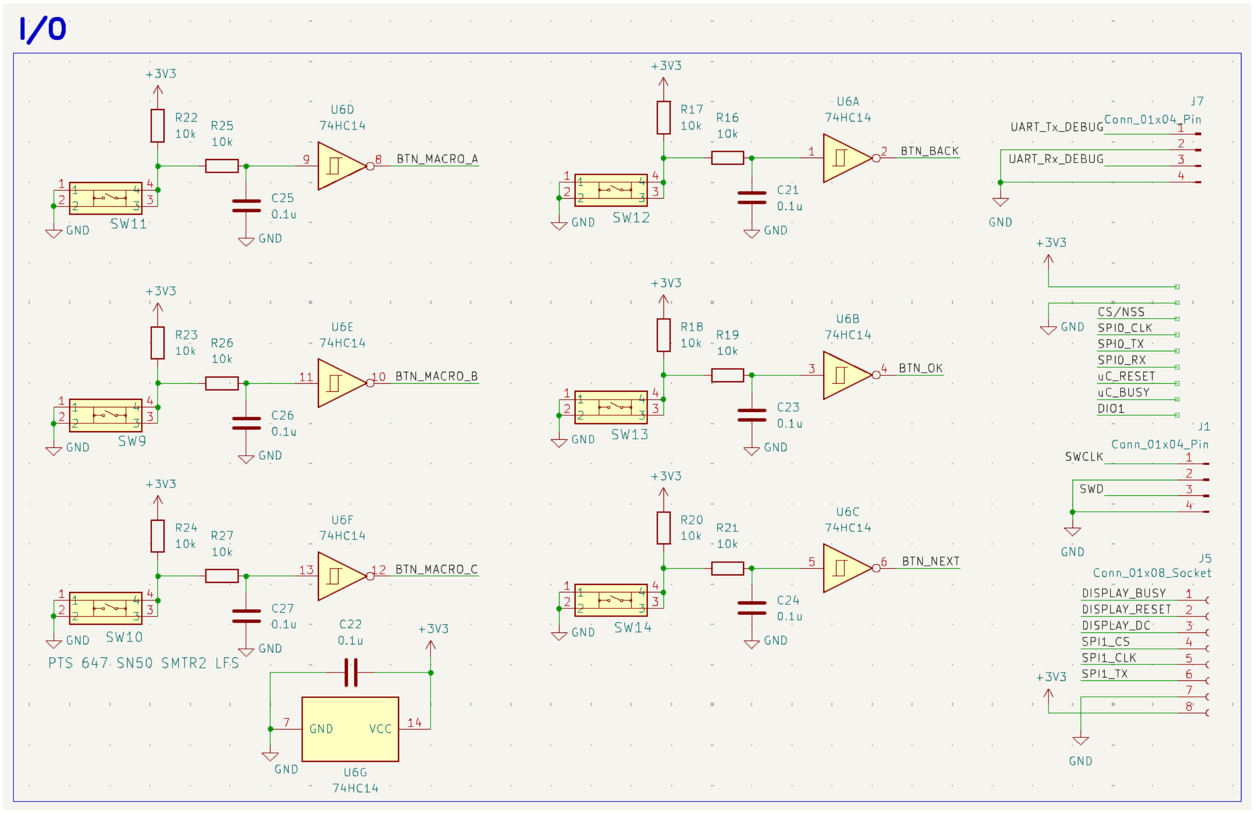


Figure 11: Debounced Macro Pushbutton Schematic Implementation [9]

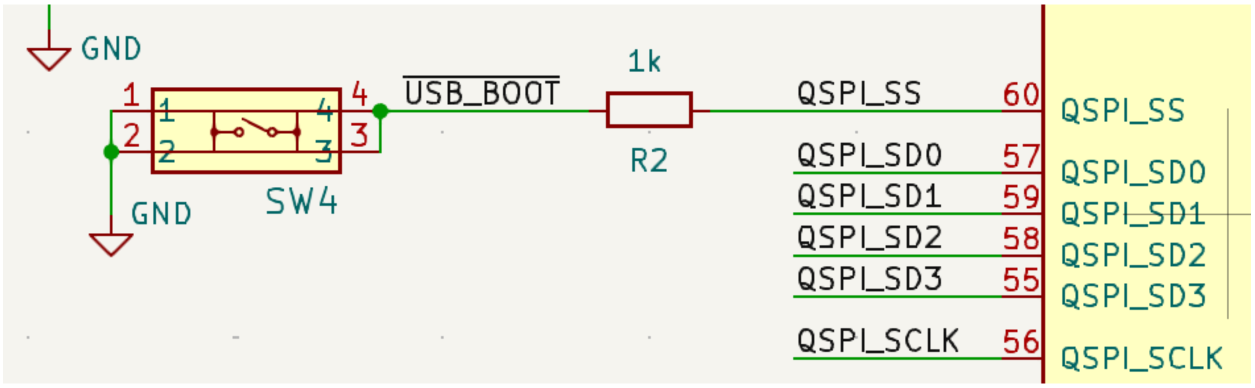


Figure 12: QSPI Select Pushbutton Input

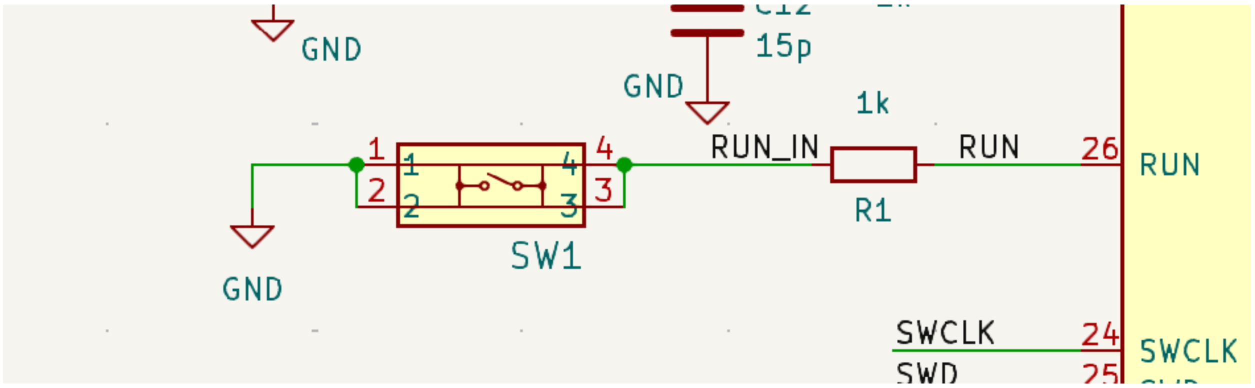


Figure 13: RUN (Reset) Pushbutton Input

The user IO consists of 8 pushbuttons where 6 are allocated to programmable macro functions and 2 are allocated for the boot and run pins. As per the RP2350 datasheet the boot and run pushbuttons both output to a series resistor to limit current flow into the pins. They are both active low, where the pull up resistors are included within the RP2350 IC. No debouncing was required for these pins because they simply read a static signal for restarting and boot mode purposes.

The 6 macro push buttons feature more advanced debouncing hardware because they are used for user input to the system. The pushbutton is configured as active low, so pull up resistors are included. There is also a simple RC low pass filter used to smoothen the input signal which is then inputted to the 74HC14 Schmitt Trigger IC, outputting a clean high or low voltage.

PCB Layout of Revision 1 (Stefan, 10%)

January to February (48 Hours)

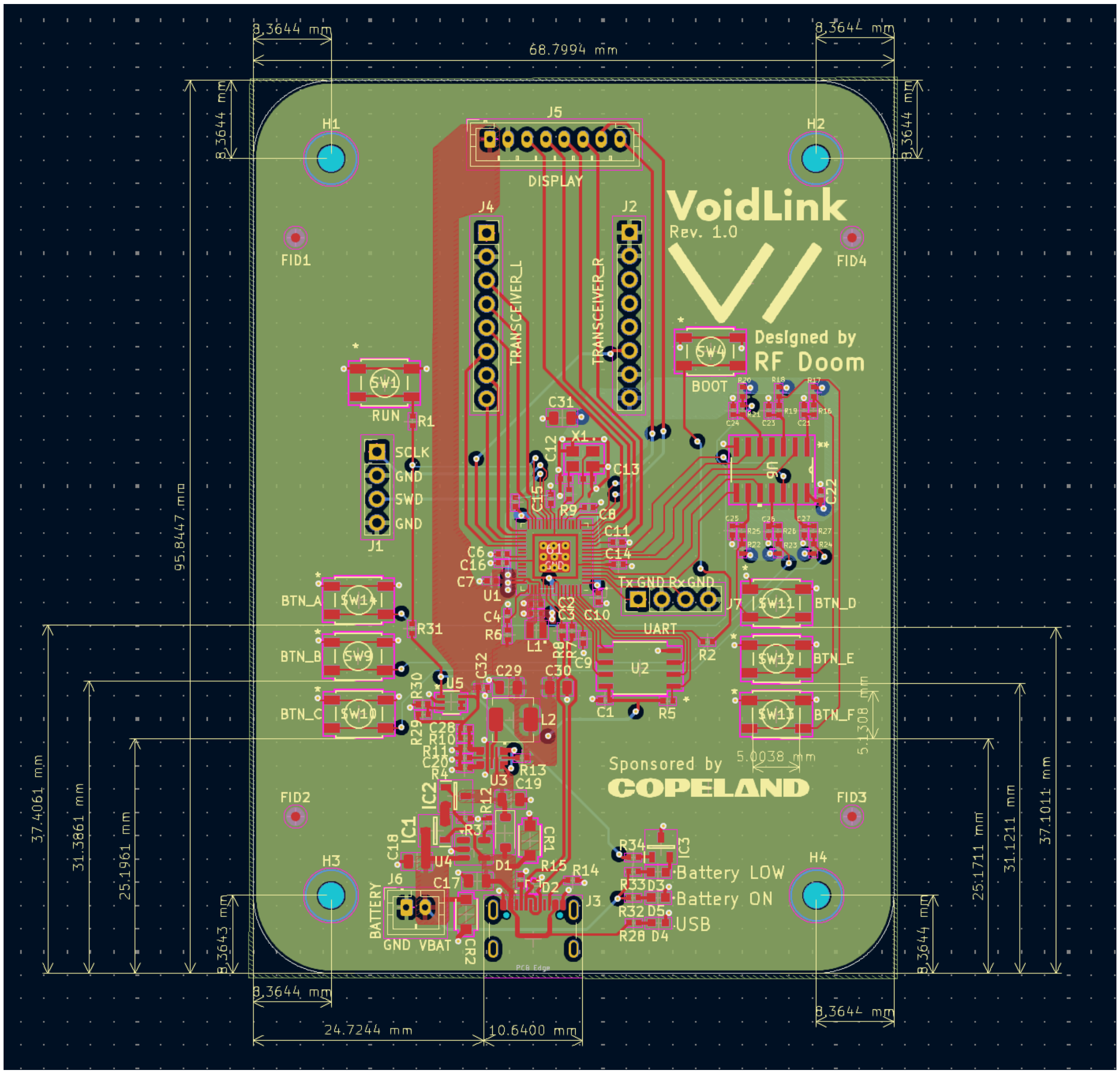


Figure 14: PCB Revision 1 Layout

The first revision of the board was a 4-layer, 69mm × 96mm design created in KiCad. KiCad was chosen due to its minimal learning curve, extensive online resources, and because it is free and open source. KiCad's simplicity was the main benefit that was considered in its selection because of the project's time constraint. Furthermore, KiCad provided all the necessary tools required to implement the PCB layout. Ideally, if time was not a constraint, we would have considered more sophisticated PCB layout software such as OrCad due to its wide industry use and integration with PSpice.

With external pinouts to attach to the SX1262 breakout board, display, and buttons. A general trace width of 0.3mm was used for signal traces. For the 3V3 power distribution network, power pours were implemented to reduce power resistance. Large pours were specifically used in [9]higher current areas to minimize heat generation. This approach helps maintain a stable voltage across the system, reduces losses, and ensures better thermal performance [2].

Board Stackup

The four layer board stackup that was selected for both revisions was (signal/power) - (ground) - (ground) - (signal/power). The double internal ground planes provide a strong ground reference for signals on the top and bottom layers. Compared to the more traditional alternative of an internal ground and power plane, the double internal ground provides a strong reference for signals routed on the top and bottom layers. The two ground layers were also forecasted to simplify the transition from the routing in the MCU area to the transceiver area which would require ground pours on all four layers (more on this in the rev2 section) [3]. Furthermore, this stackup provides a clear current return path, reducing overall loop area and thus improving EMI. This double internal ground plane is a more modern approach to high performance four-layer boards as it has been more recently popularized by Rick Hartley [10]. The high-performance nature of this stackup makes the board easily adaptable to future additions. The main drawback of not including a dedicated power plane is the increased routing complexity required for the power net. In the case of VoidLink, this was not an issue as careful routing decisions were sufficient to accommodate all power and signal nets on the top and bottom layers. It is also worthy to note that in rev1, the two internal ground planes were uninterrupted due to the simplicity of the MCU routing.

MCU XTAL

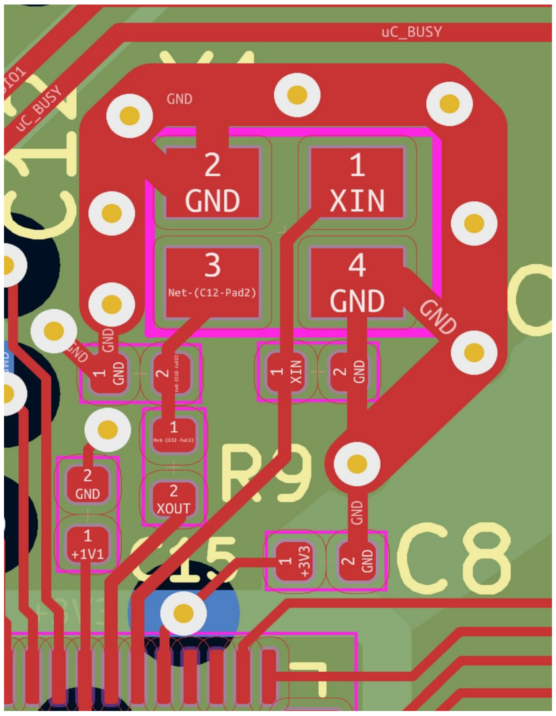


Figure 15: MCU XTAL Implementation

The MCU XTAL was placed close to the MCU IC to reduce the length of the clock traces, reducing parasitic resistance on this high-speed net [2]. The XTAL also has a thick ground trace routed around it with frequent VIAs for a strong connection [2]. This ground trace provides ground shielding to protect the high speed XTAL traces from coupling with other signals. To further minimize parasitic capacitance on the clock nets, they were routed using minimum width traces [2].

Power

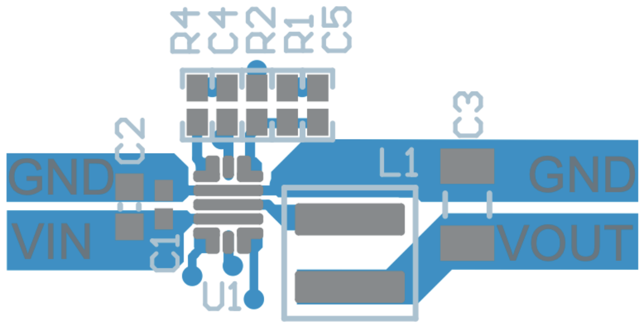


Figure 16: DC/DC Converter Layout from TPS62811 Datasheet [5]

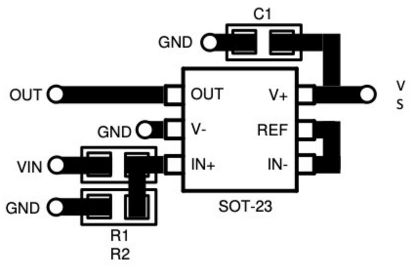


Figure 17: UVLO Layout Reference from TLV3012 Datasheet [7]

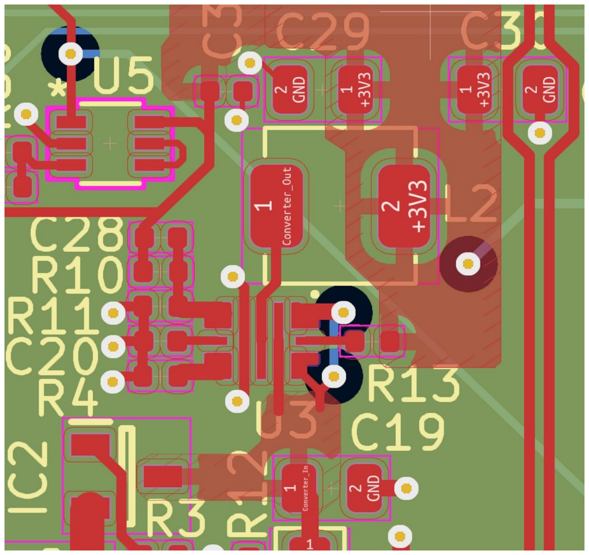


Figure 18: TPS62811 & TLV3012 Layout Implementation

The power section layout is based on various datasheet references according to the ICs used. The floorplan was designed such that area was minimized between ICs to shorten trace length, improving parasitic resistance. Parasitic inductance is generally addressed in the high current regions by following the reference design for the DC/DC converter [5] and using wide traces/power pours with clear current return paths.

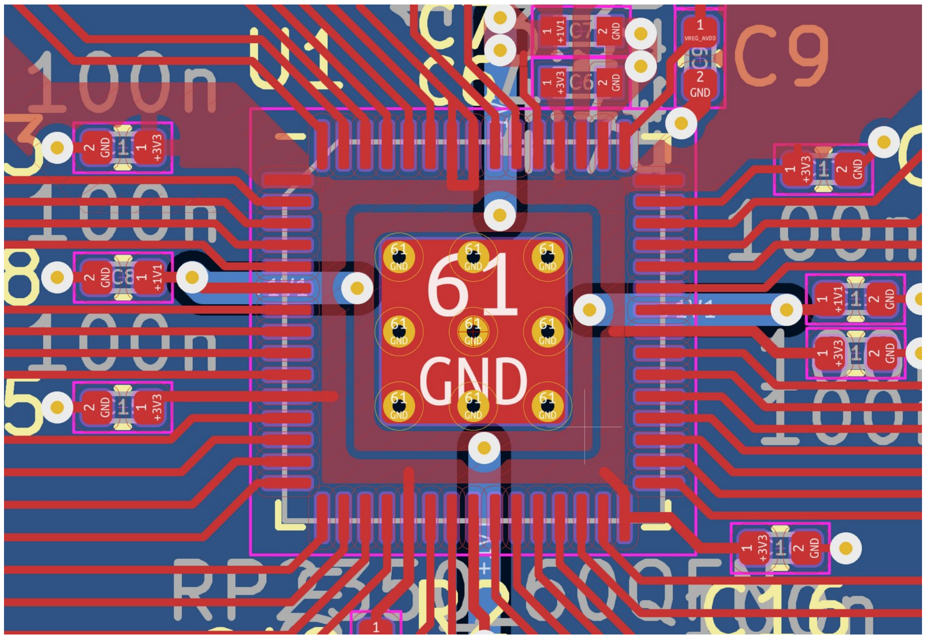


Figure 19: RP2350 Power Delivery Layout Reference from RP2350 Minimal Reference Design Files

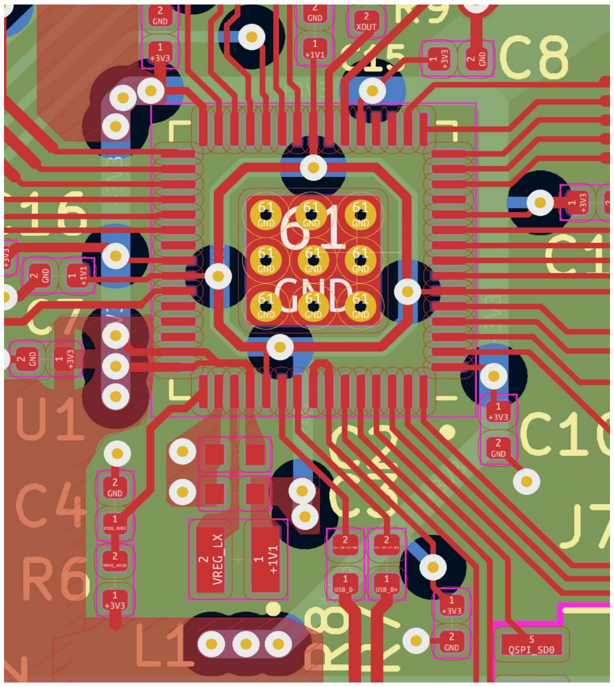


Figure 20: RP2350 Power Delivery Implementation, Front Copper Focus

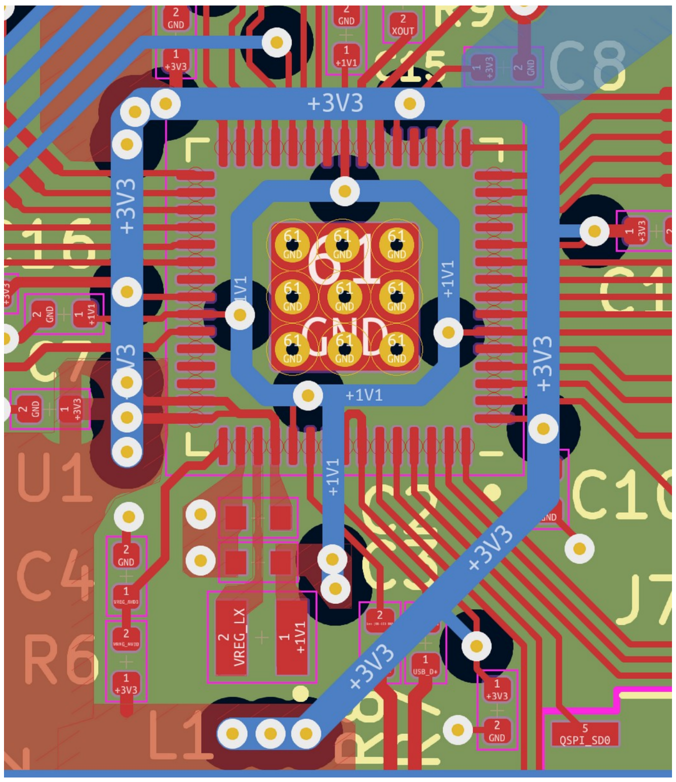


Figure 21: RP2350 Power Delivery Implementation, Back Copper Focus

Due to manufacturing constraints of our design, we were not able to implement the reference routing where both 3V3 and 1V1 were routed under the MCU IC. Instead, a custom design was implemented, inspired by the low impedance power delivery of the reference design [2]. Whenever transitioning layers on a power net, multiple vias were used for a low impedance connection. Furthermore, power pours were used when possible and power traces were routing in a loop to further minimize parasitic impedance in the power nets.

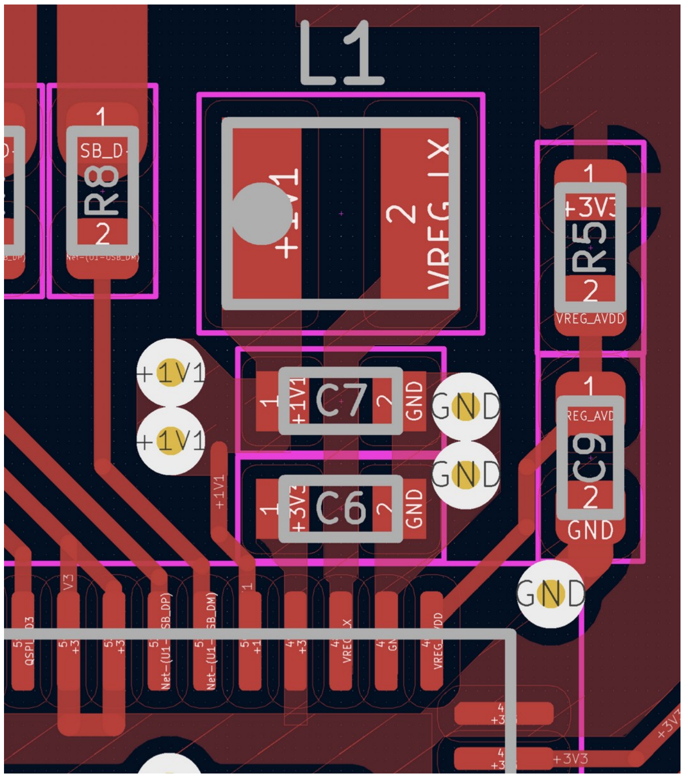


Figure 22: VREG Layout Reference From RP2350 Hardware Design Document [2]

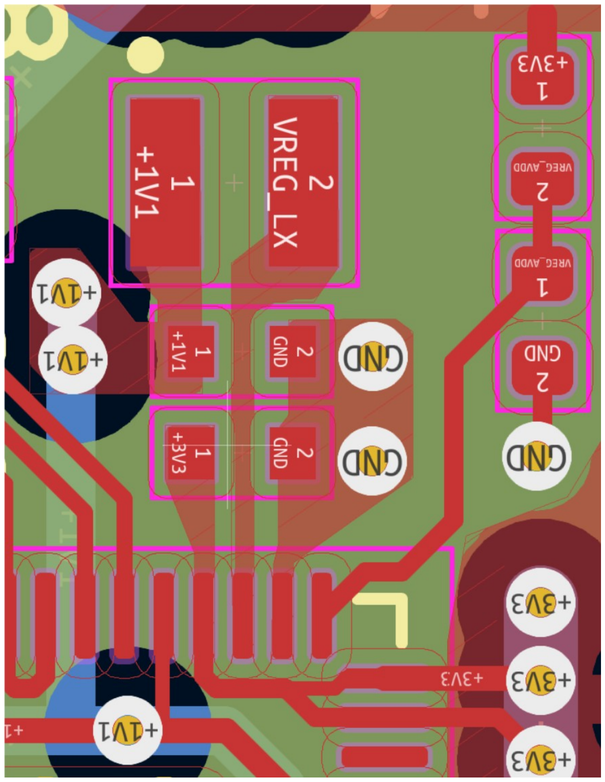


Figure 23: VREG Layout Implementation

The RP2350 Hardware Design document highly recommended implementing the exact reference layout for the VREG region [2]. This was due to its highly optimized nature and the necessity of this supporting circuitry in generating the internally used 1V1 power net which is crucial for the digital function of the MCU [8]. Once again, manufacturing constraints and DRC limitations in KiCad prevented the reference design from being copied exactly, but an implementation was created that was sufficient to operate the MCU.

Revision 2

PCB Layout for Revision 2—Transceiver Integration (Milena Thibault 10%)

March (50 hours)

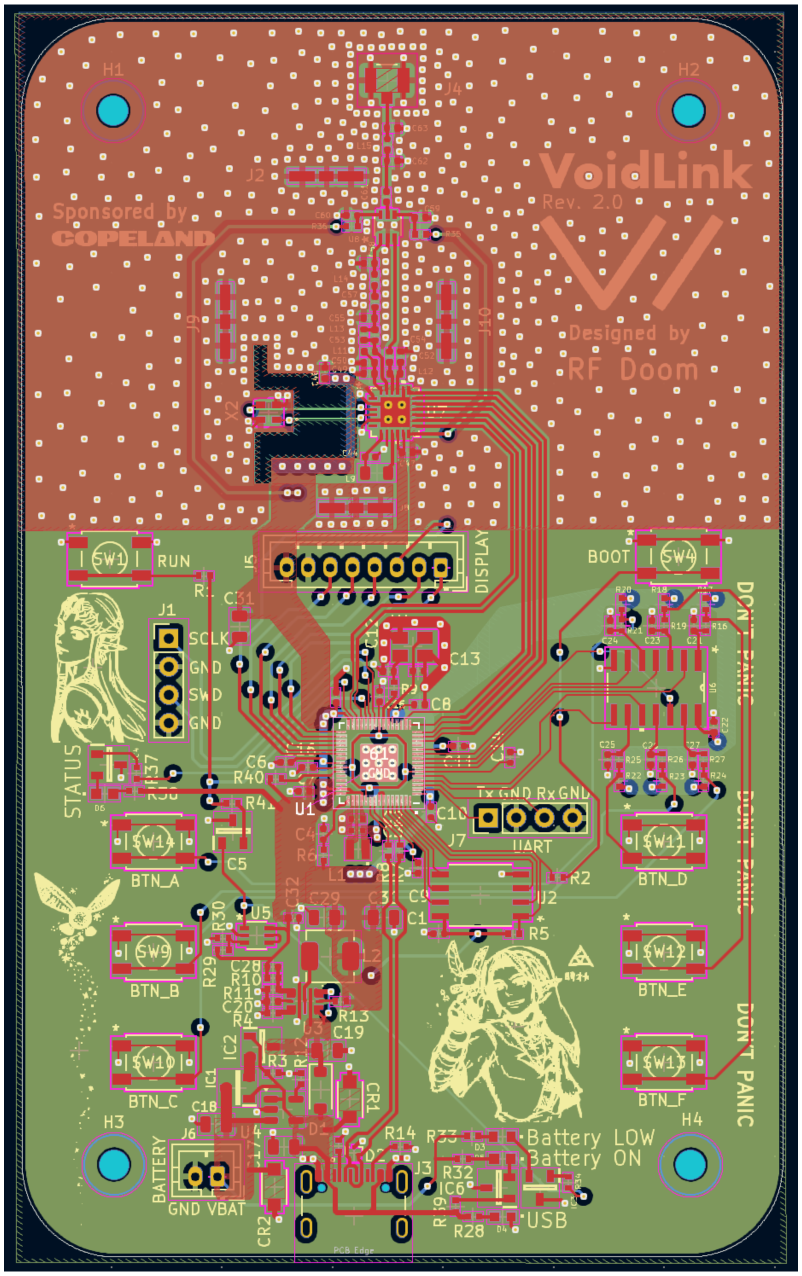


Figure 24: PCB Revision 2 Layout

The SX1262 transceiver was directly integrated into the board, replacing the previous pinout designed for the SX1262 breakout board. The overall height was increased to facilitate the

SX1262 layout. Furthermore, additional design/implementation adjustments were included to improve functionality, ease of use, and optimize performance.

For the integration of the transceiver, due to the sensitive nature of RF layout, the reference design from Semtech was followed closely to maximize signal integrity and transceiver performance [3].

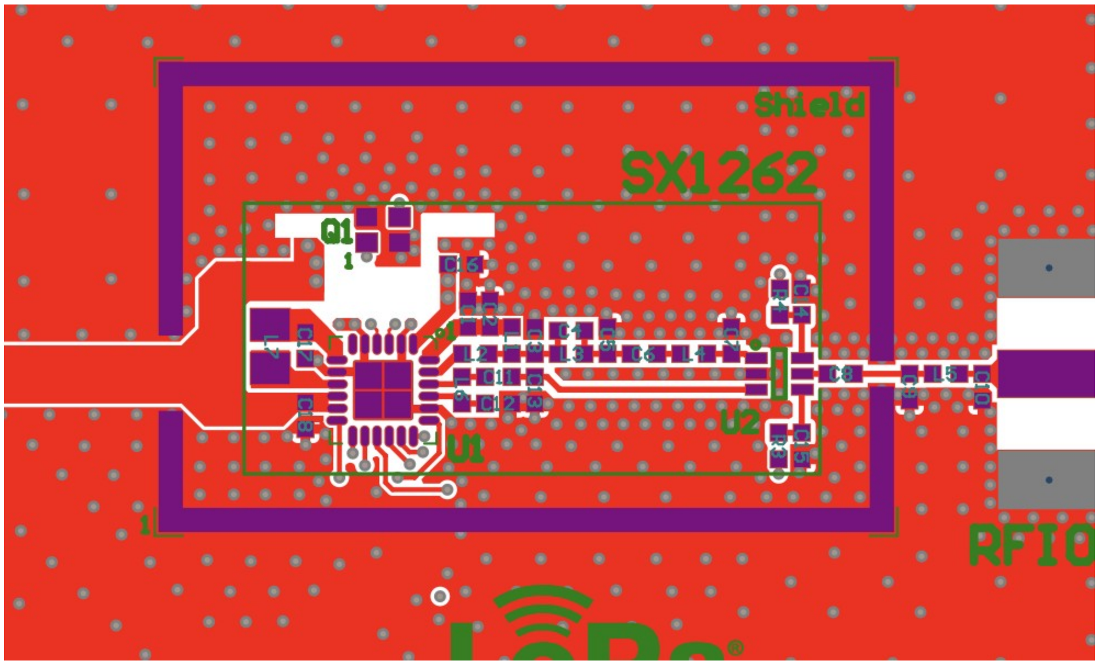


Figure 25: SX1262 Reference Layout Design from Semtech (Layer 1) [3]

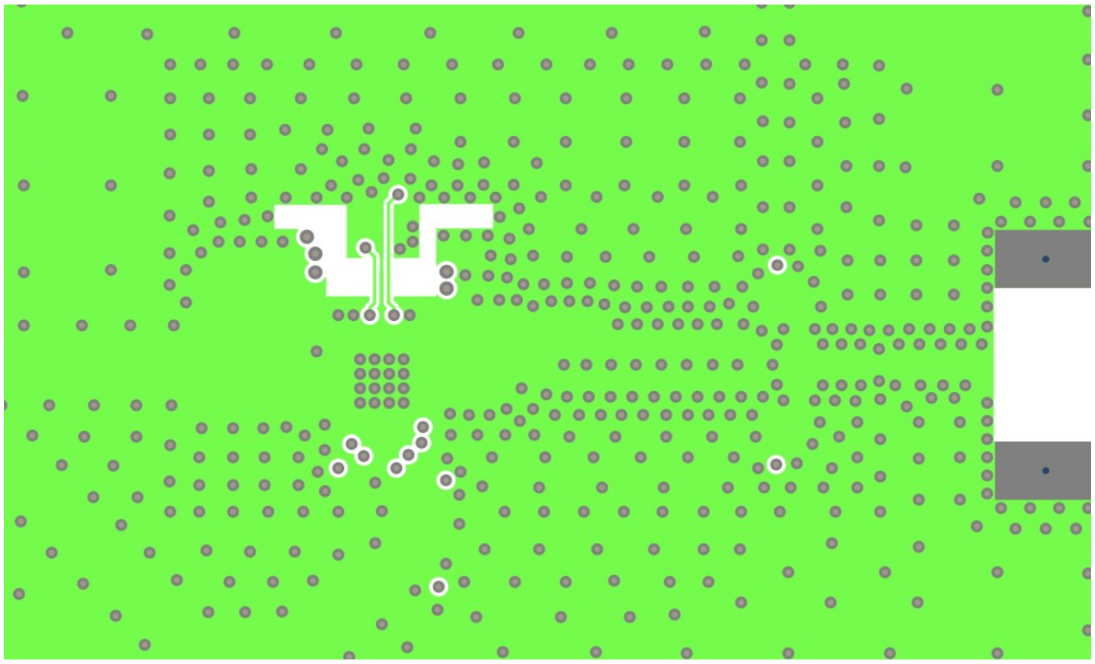


Figure 26: Reference Layout Design from Semtech (Layer 2) [3]

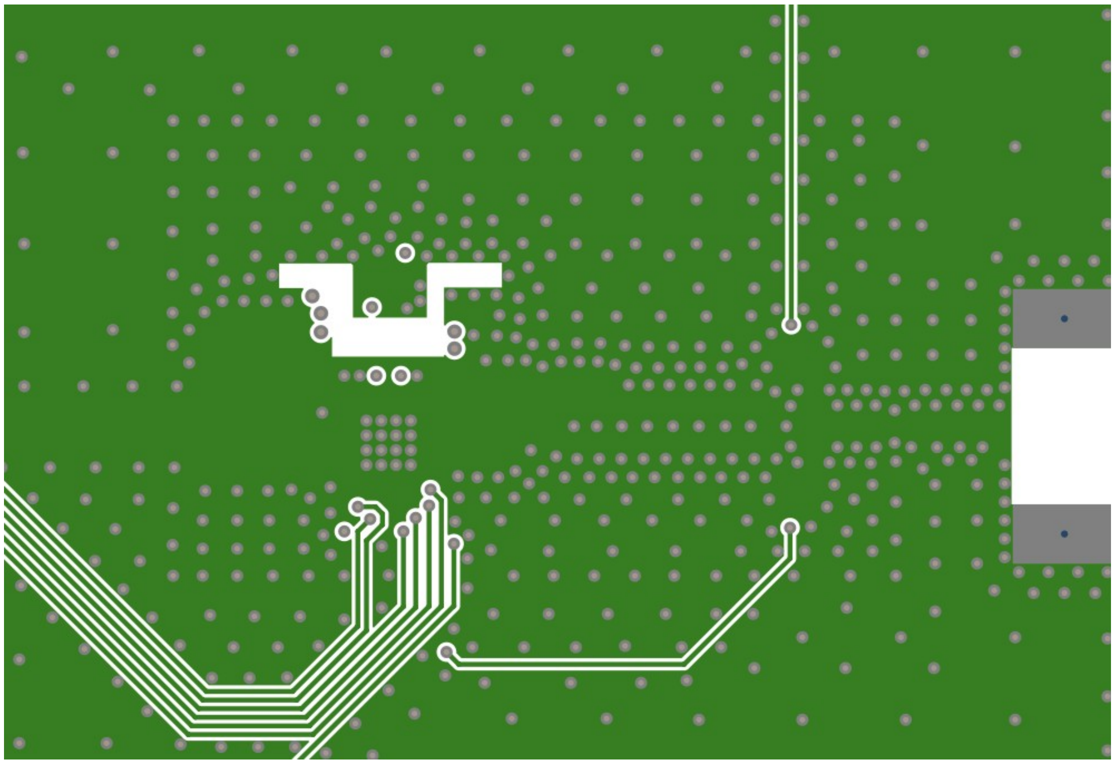


Figure 27: SX1262 Reference Layout Design from Semtech (Layer 3) [3]

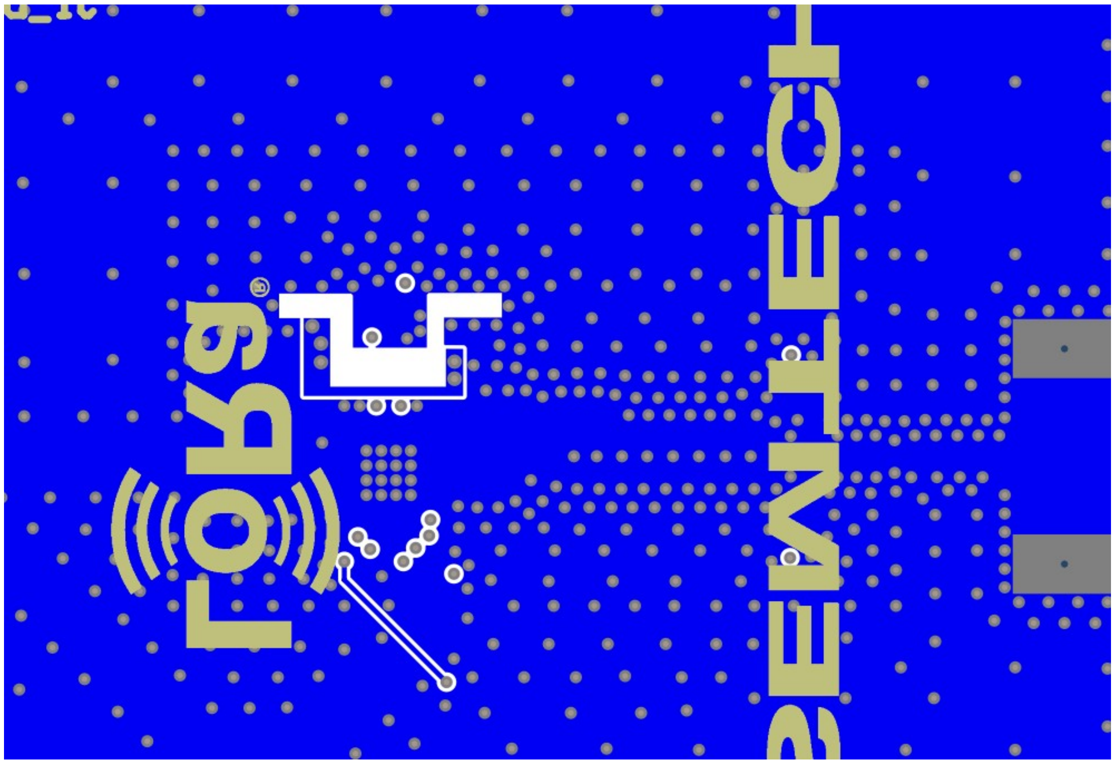


Figure 28: SX1262 Reference Layout Design from Semtech (Layer 4) [3]

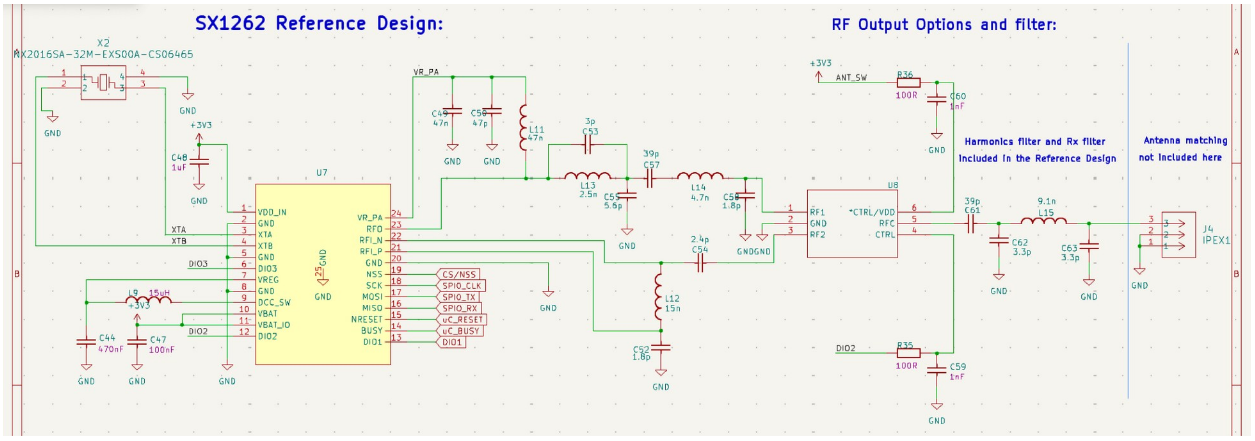

SX1262 Reference Design:
RF Output Options and filter:
Harmonics filter and Rx filter included in the Reference Design
Antenna matching not included here

Figure 29: SX1262 Transceiver Schematic Implementation Adapted from Reference Design [3]

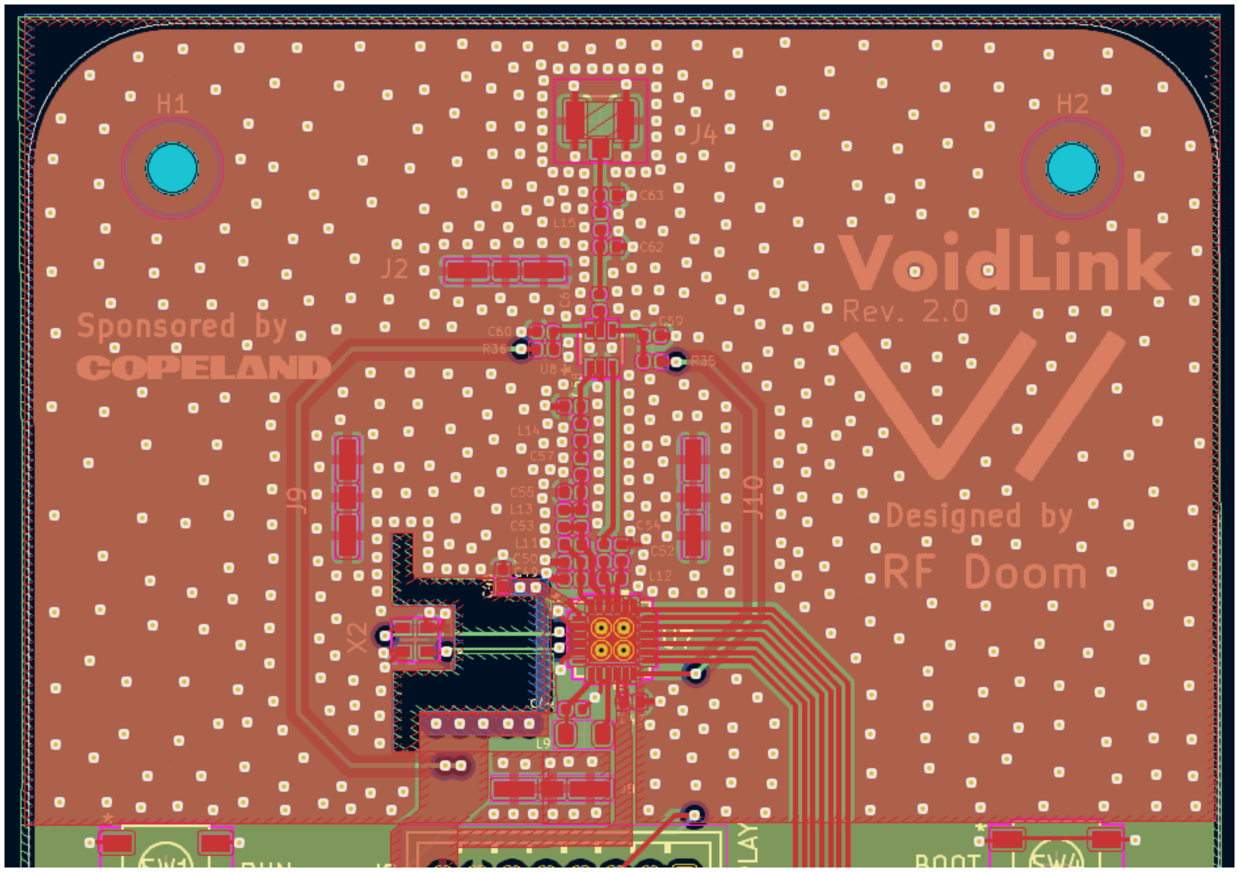


Figure 30: SX1262 Transceiver Layout Implementation

RF circuit layout requires careful impedance matching, grounding, and shielding to maximize signal integrity and minimize interference [3]. The RF traces were impedance-matched to a single-ended coplanar 50 ohms, with two (mostly) uninterrupted internal ground planes and frequent vias to ensure a low-inductance and low-impedance ground connection [3]. An RF shield was added to reduce external interference, while decoupling capacitors were placed close to ICs to suppress noise on the 3V3 power net [3]. The RF signal line was kept as short as possible to minimize loss and signal degradation [3]. To prevent the heat dissipation of the transceiver IC from affecting the XTAL, it has a surrounding cutout, and its clock nets are routed on layer 2 (because it is an internal layer) [3]. The antenna is connected via a Hirose U.FL (IPEX variant) connector located at the end of the board, ensuring minimal signal loss and a reliable RF connection [3].

The SX1262's CMOS oscillator was replaced with a passive crystal which contributed to the 65% battery life improvement.

Design Fixes from Revision 1 (Milena Thibault 5%)

March (15 hours)

In the previous design, the MCP73812T-420I-OT battery charging IC's charge enable pin was left disconnected due to a slightly different IC being implemented compared to the reference design—this prevented the battery from charging [6]. In this revision, it has been properly connected to the 3v3 high voltage, resolving the issue. See [4]Figure 3: Battery Charging & Power Source Selecting Schematic [4] for further details.

In the first revision, the display and power connections were flipped, and the pinouts to the SX1262 transceiver were incorrect. Luckily, none of these errors affected the actual functionality of the board, and after some manual soldering, the board operated as expected.

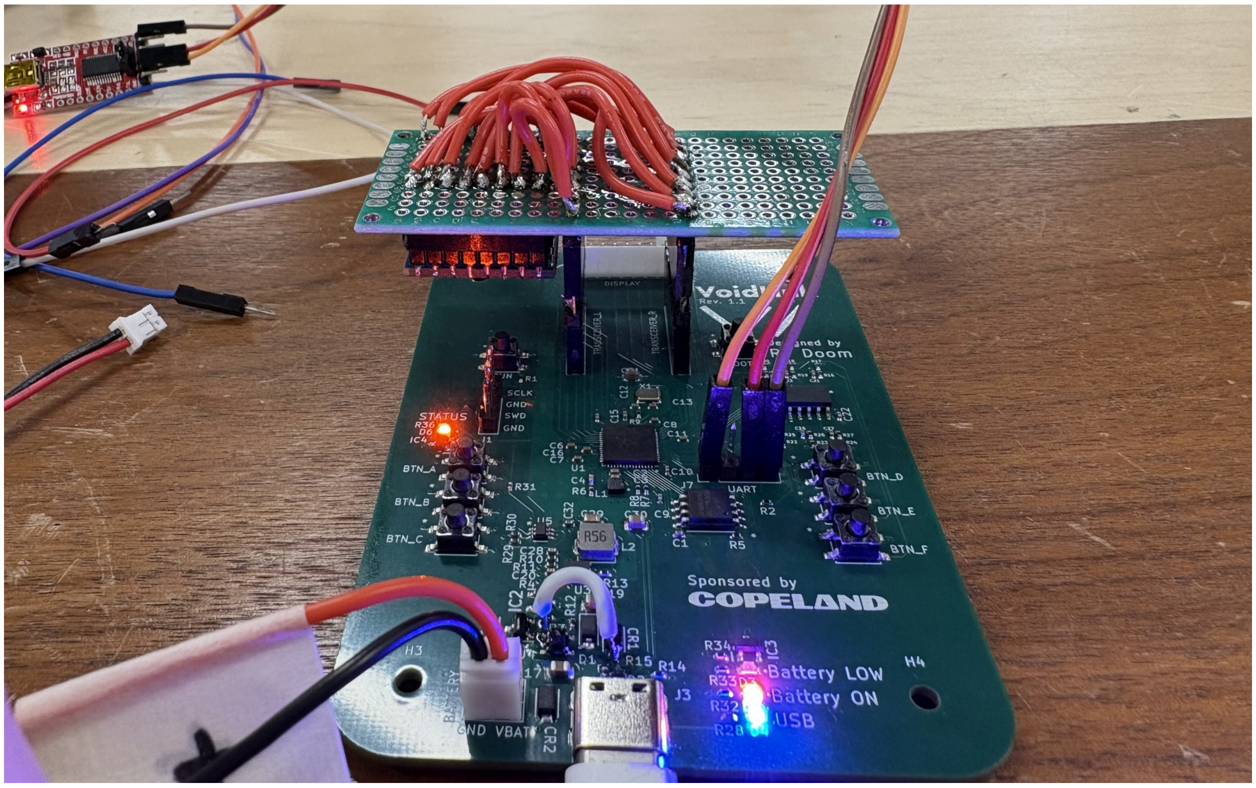


Figure 31: PCB Revision 1 with Manual Soldering Fixes

These issues were all corrected in the second revision, ensuring that all pins were ordered correctly and mapped to the MCU with the correct functionality while also optimizing the layout by avoid overlapping traces [8].

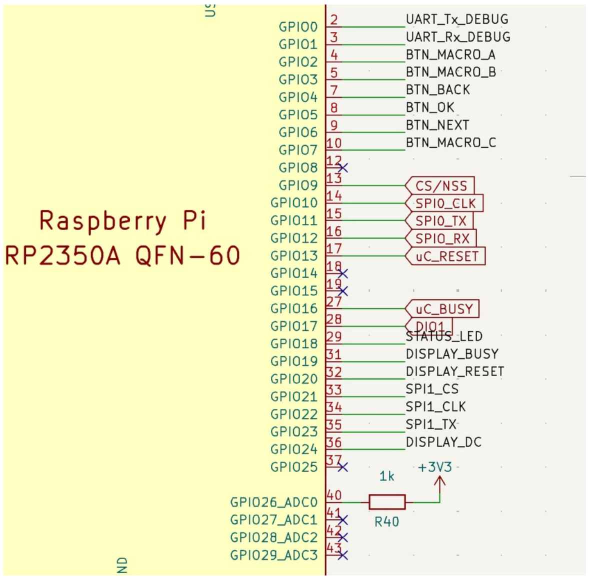


Figure 32: Revision 2 Updated MCU Pinout Schematic

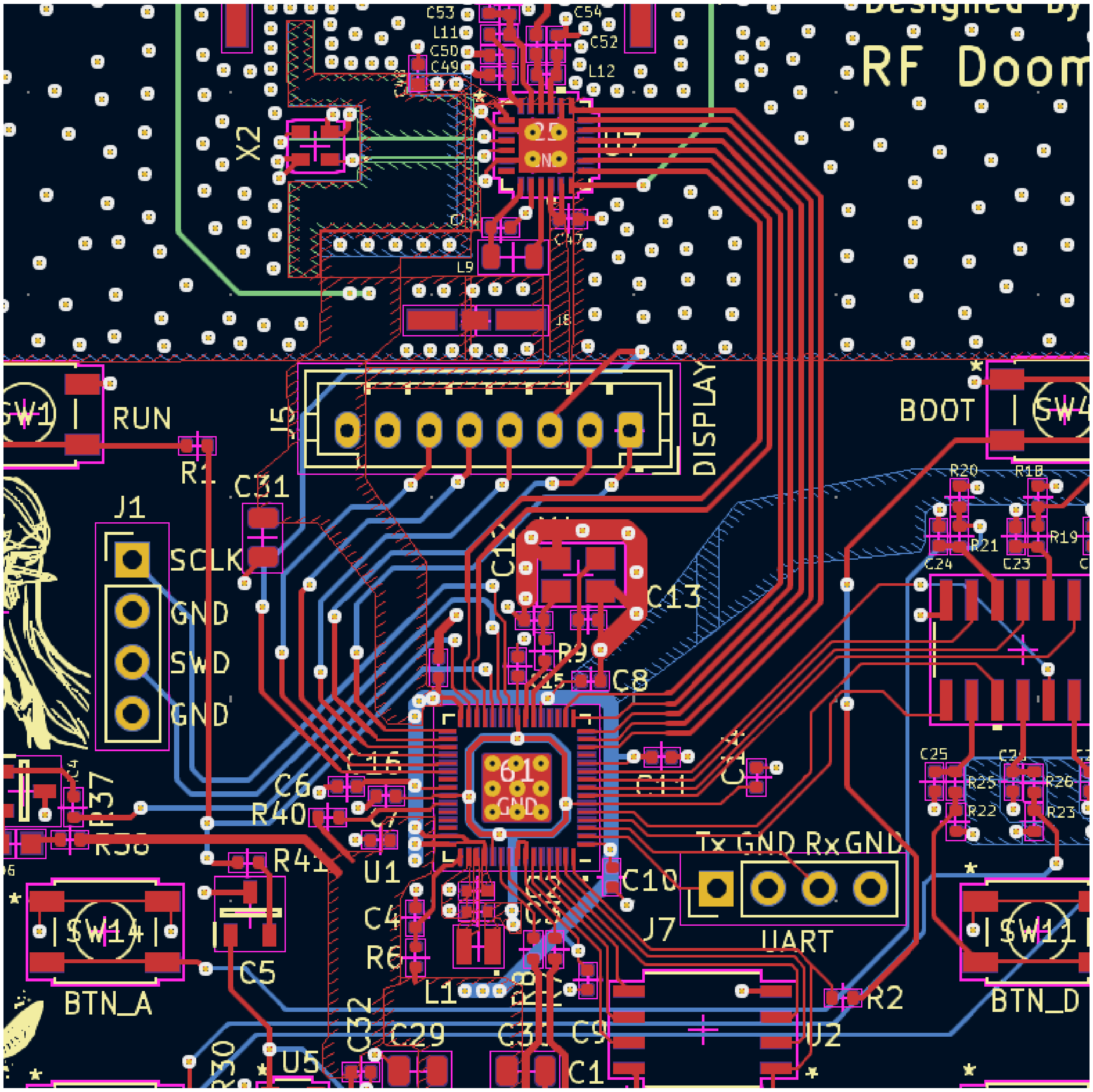


Figure 33: Revision 2 Updated MCU Pinout Layout

The display pinouts were corrected, and the SX1262 breakout board was fully integrated, eliminating the need for pin corrections.

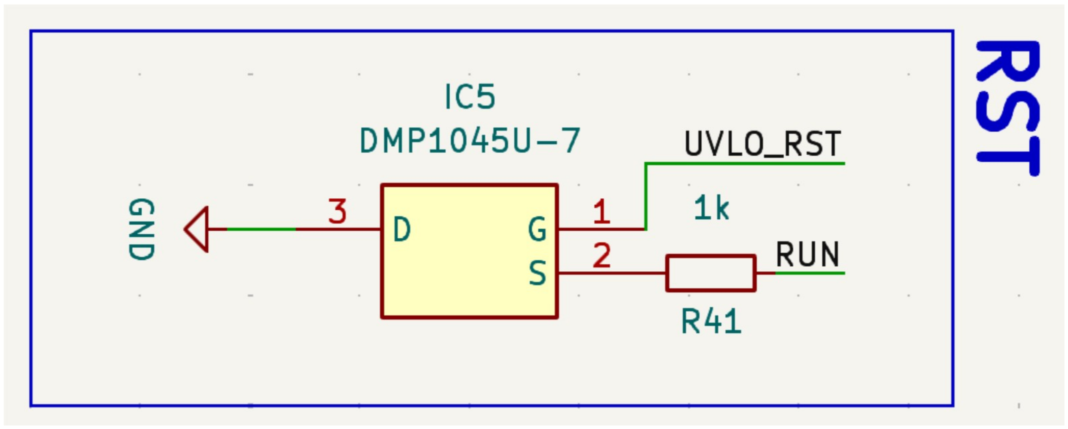


Figure 34: PMOS Switching Circuit to Increase Drive Strength into MCU Reset Pin

The UVLO circuit failed to properly reset the MCU. This turned out to be a confusing issue to debug. When supplying the board with voltages below our set low voltage threshold, it was confirmed through physical measurements that the TLV3012 UVLO IC output would output a low voltage close to ground, which was the expected behavior [7]. Despite the IC outputting a low voltage, which powered the Battery LOW LED, it would not trigger the RP2350's reset function [11].

This hinted that the issue might be the drive strength of the TLV3012 output. To address this, the TLV3012 output was connected to the gate of a low V_ON PMOS (the same one that was used in the power section). With the drain connected directly to the ground net and the source connected to the RUN pin through a series termination resistor, when the TLV3012 outputs a logic low due to a low battery input, it will turn the PMOS on, connecting the RUN pin to ground through its conductive channel. This change improves the drive strength of the signal as it is being supplied by a low resistance connection to the ground net. Likewise, when the battery is above the set threshold and the TLV3012 output follows the input voltage (considered a logic high), the PMOS will turn off, causing the IC's internal pull up resistor to set the pin to a logic high. Upon testing the rev2, this change fixed the issue and the MCU was able to successfully be reset by the TLV3012 UVLO IC.

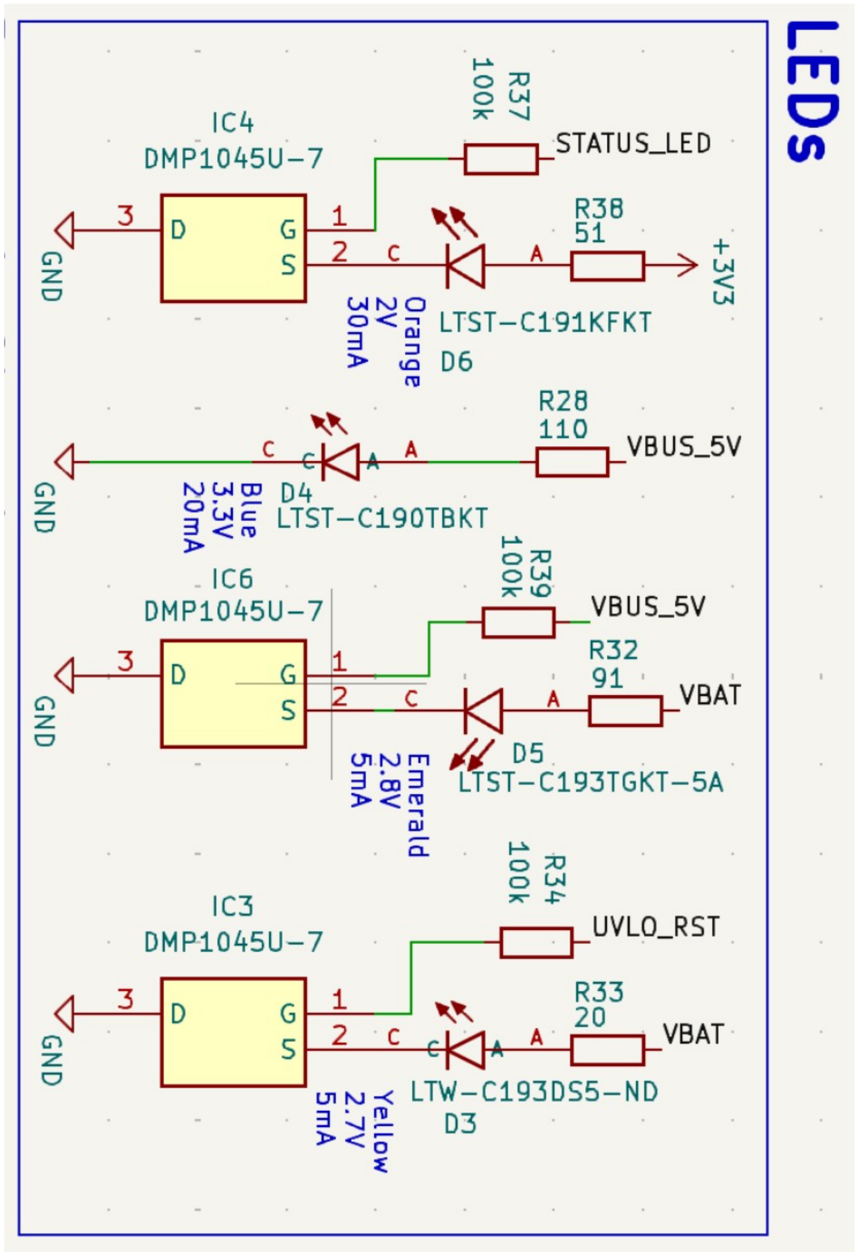


Figure 35: Revision 2 Updated LED Logic & Resistor Values

The LED resistor values were all slightly increased, reducing their current draw to improve battery life at the expense of the excessive brightness of rev1, also contributing to the 65% battery life improvement. Due to the fix implemented in the charging IC, the VBAT net is now powered when the USB is plugged in. This caused the Battery ON LED to always be on whenever the USB was plugged in, even if the physical battery was disconnected. To solve this, a low V_ON PMOS was used to implement new logic. With the gate tied to VBUS_5V (the USB power net which includes a pull down resistor), and the LED being powered by VBAT through the PMOS's conductive channel, the LED will only turn on when the USB is disconnected (causing the pull down resistor to pull the net down to ground) the PMOS will conduct and allow the VBAT to power the Battery ON LED. Likewise, when the USB is plugged in and VBUS_5V is high, the PMOS does not conduct, thus preventing the Battery ON LED from turning on.

Rev 1 of the PCB was ordered on March 1 and received on March 10. After testing, some non-critical issues were identified, and the full RF section and transceiver were not yet integrated. Rev 2 was a quick turnaround—ordered March 18 and received March 31—which fixed those minor issues and included the full integration of the RF and transceiver components.

Network Module (Boran, 25% of Overall Project)

The network module is responsible for developing the communication protocol, its underlying data structures, and the logic required to receive and transmit packets. It ensures reliable data transfer, fair scheduling of tasks and quality of service.

This module also provides many public functions and interfaces to access the underlying data and take actions on behalf of the device. The user interface module uses these features to show information to the user and handle button input.

Layers (2.5%)

September to October (24 hours)

VoidLink uses the SX1262 transceiver as a LoRa modem to transmit data over the physical medium (PHY), which acts as layer 0. The transceiver accepts a payload and encapsulates it in a LoRa frame as shown in Figure 36: LoRa Frame [11]. This frame includes important features, like a cyclic-redundancy check (CRC) to ensure the correctness of the data and a preamble to notify the arrival of the data [11].

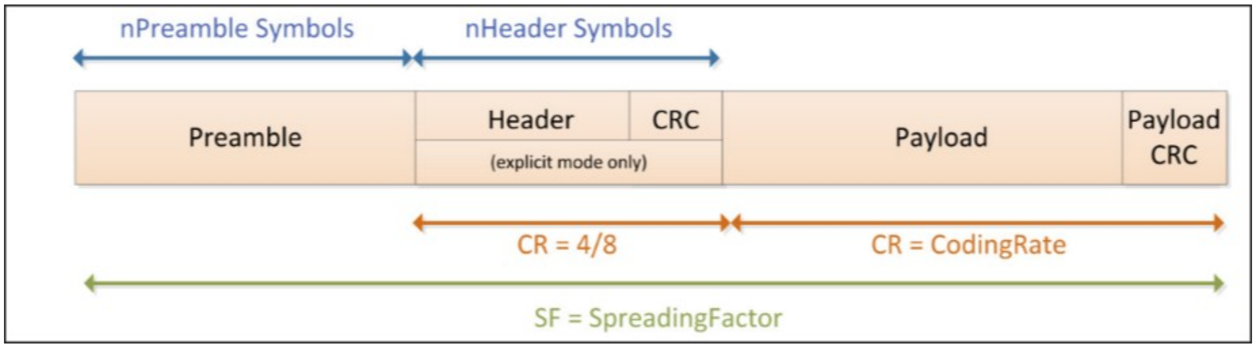


Figure 36: LoRa Frame [11]

The network module builds on top of this to transmit meaningful information between nodes. This layer-by-layer design is inspired by the Open Systems Interconnection (OSI) [12] model and the Meshtastic Broadcasting Algorithm [13]. In both cases, each layer is responsible for handling one part of the network, which gets utilized by the layer above it.

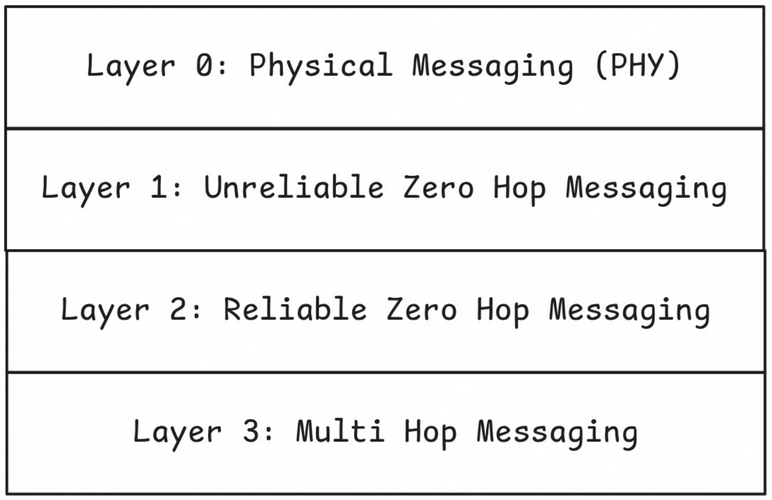

Layer 0: Physical Messaging (PHY)
Layer 1: Unreliable Zero Hop Messaging
Layer 2: Reliable Zero Hop Messaging
Layer 3: Multi Hop Messaging

Figure 37: VoidLink Network Protocol Layers

Layer 1 uses the VoidLink protocol discussed below to unreliably transfer information between nodes. It encodes the data efficiently and correctly into a byte stream before handing it over to layer 0. This layer does not track if the packet reached the desired destination, therefore, it is labelled unreliable.

At this layer, the network checks the source and the destination fields of the packet to decide if it should proceed to process it. It uses multiple parts of the packet to verify that the packet is unique and not already received. Finally, it records the source node and the transmission quality into a table of “neighbours”. This table is used to show nearby nodes and the connection quality to the user.

Layer 2 introduces acknowledgements to the network to achieve reliable messaging. Each packet has an “acknowledgement required” flag, which conveys to the receiver if the sender requires confirmation. If the receiver gets a packet with the flag set, it proceeds to respond with a special “ACK” message that includes the ID of the original message. The inclusion of the original message ID disambiguates which message is being acknowledged.

Each node keeps a record of its messages with a pending acknowledgment. If the node does not receive the acknowledgment before a timeout value, it retransmits the packet for up to a specified number of times.

Layer 3 adds the key feature of message forwarding to the network. When a node receives a message destined to a different node (or to a broadcast address), it retransmits the message on behalf of the original sender. This allows sending messages to nodes that are out of reach by utilizing intermediate nodes as relays, which forms the “mesh” network.

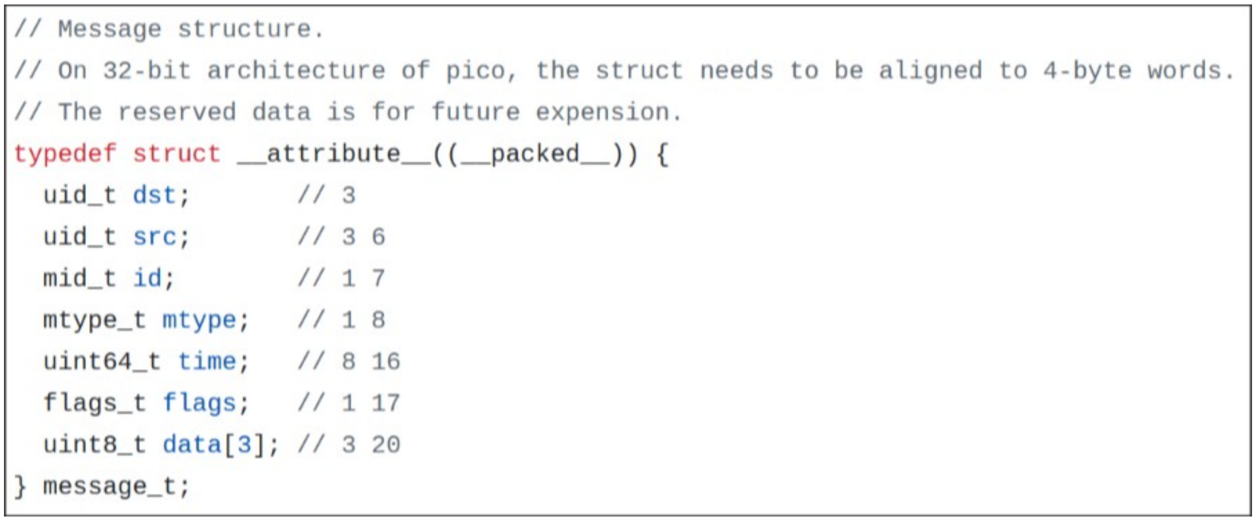
The drawback of carelessly relaying all foreign messages (also known as flooding) is that it creates unnecessary and harmful traffic in the network. In the best case, these extra packets only waste resources. However, in many cases, these packets cause infinite loops in the network, which escalates into a self-induced denial-of-service attack.

While there are multiple algorithms for controlling the flooding behaviour, the VoidLink protocol aims to be simple and lightweight first. For this reason, each packet includes a “hop-limit” field to indicate how many more times it can be retransmitted, which gets decremented on each hop. On top of this, as discussed in layer 1, nodes keep a history of recent messages and promptly reject already seen ones.

Protocol (10%)

December to April (96 hours)

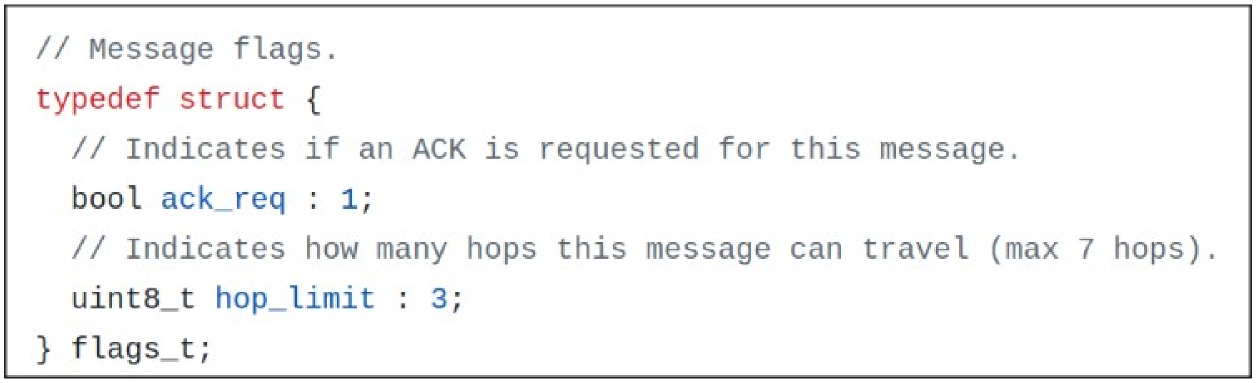
The information is encoded using a proprietary protocol developed specifically for this project. It aims to pack as much information as possible while keeping the size of each packet to a bare minimum. This is achieved by many careful considerations and iterative prototyping.



```c
// Message structure.
// On 32-bit architecture of pico, the struct needs to be aligned to 4-byte words.
// The reserved data is for future expension.
typedef struct __attribute__((__packed__)) {
  uid_t dst;        // 3
  uid_t src;        // 3 6
  mid_t id;         // 1 7
  mtype_t mtype;    // 1 8
  uint64_t time;    // 8 16
  flags_t flags;    // 1 17
  uint8_t data[3]; // 3 20
} message_t;
```

Figure 38: Message Structure

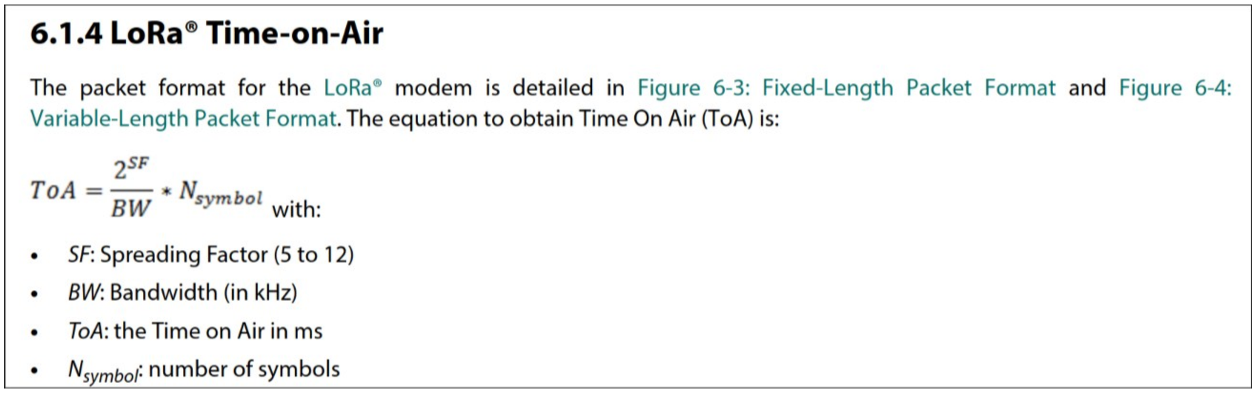
Each message is stored as a 20-byte-long C struct. The size and the ordering of each field are chosen so that they respect the alignment of the 32-bit processor and fit into four 4-byte words.



```
// Message flags.
typedef struct {
  // Indicates if an ACK is requested for this message.
  bool ack_req : 1;
  // Indicates how many hops this message can travel (max 7 hops).
  uint8_t hop_limit : 3;
} flags_t;
```

Figure 39: Flags Structure

Inner fields of the message were also condensed to not waste space. For example, the flags_t structure uses bit fields to represent multiple pieces of data in one byte.



6.1.4 LoRa® Time-on-Air

The packet format for the LoRa® modem is detailed in Figure 6-3: Fixed-Length Packet Format and Figure 6-4: Variable-Length Packet Format. The equation to obtain Time On Air (ToA) is:

$$ToA = \frac{2^{SF}}{BW} * N_{symbol}$$ with:

- *SF*: Spreading Factor (5 to 12)
- *BW*: Bandwidth (in kHz)
- *ToA*: the Time on Air in ms
- N_{symbol}: number of symbols

Figure 40: Time-on-Air Calculations from SX1262 Datasheet [11]

Using the time-on-air formula given by the datasheet and the modulation parameters chosen by our team, the bandwidth of the protocol was calculated [11].

- In the slowest mode, the node can achieve 0.18 kbps.
- In the fastest mode, the node can achieve 10.94 kbps.

Given the 20-byte size of each packet, the network can approximately transmit 1 to 68 packets every second in each mode. We expect every node to transmit on average 1 packet every minute, therefore, even the slowest speed is an order of magnitude faster than our requirements.

State Machine (10%)

November to February (96 hours)

After going through an initialization period, the microcontroller enters a main loop where the main state machine of the device dictates the actions to take in each iteration. This main state is tied closely to the state of the transceiver.

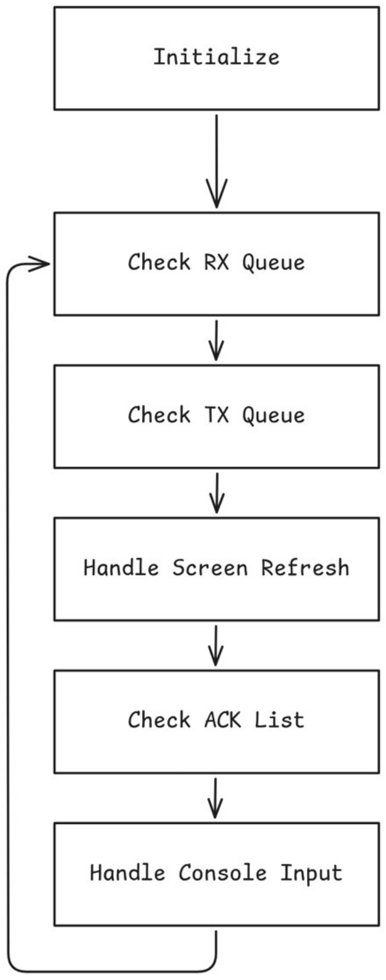


Figure 41: Main State Machine Diagram

1. Check if there are previously received but unprocessed messages and process the oldest one.
2. Check if there are messages to be transmitted and send the oldest one.
3. If there are no messages to send, continue receiving.
4. Check if there are screen updates and trigger a refresh.
5. Check if there are expired acknowledgments and retransmit their original messages.
6. Check if there are incoming console commands and handle them accordingly.

Interrupts (2.5%)

October to November (48 hours)

Since the microcontroller is not running an operating system, there are no schedulers or threads to multitask. Instead, all asynchronous tasks are handled via interrupts. Utilizing interrupts is crucial to not block the execution of the main loop while waiting for events to happen. For example, when transmitting a packet, the transceiver requires a certain amount of time to ramp up its power amplifier and transmit each bit [11]. Once the transmit command is issued, the microcontroller does not wait for transmission to be completed but expects an interrupt to be sent by the transceiver when it is done.

1. When a transmission is complete, or an error occurs.
2. When a preamble is detected on the air.
3. When a packet is received.
4. When a physical button is pressed.
5. When a character is sent over the USB serial port.
6. When a character is sent over the debug serial port.

```
void handle_irq_callback(uint gpio, uint32_t events) {
  if (gpio == PIN_DIO1) {
    handle_dio1_callback(gpio, events);
  } else if (gpio == PIN_BUTTON_NEXT || gpio == PIN_BUTTON_OK || gpio == PIN_BUTTON_BACK ||
             gpio == PIN_BUTTON_PREV || gpio == PIN_BUTTON_HOME || gpio == PIN_BUTTON_SLEEP) {
    handle_button_callback(gpio, events);
  }
}
```

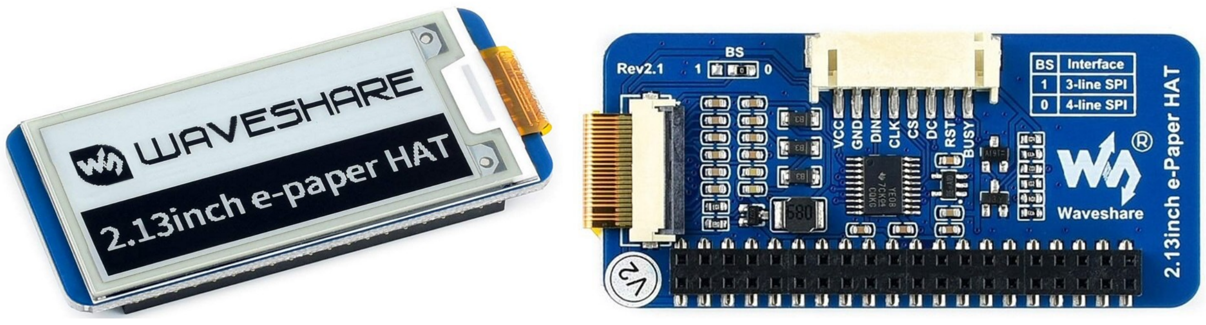
Figure 42: Interrupt Callback Functions

User Interface Module (Lukas Kotuza-Janisch, 25% of Overall Project)

The user interface module of VoidLink provides an intuitive and reliable way for users to interact with the device in off-grid or low-power environments. It features a low-energy E-Ink display paired with six tactile push buttons, enabling clear visual feedback and straightforward navigation through various menu screens and options. This module integrates all other modules to provide the users with the ability to monitor and control network status, send and view messages, and change settings. To complement the electronics, a custom-designed and 3D-printed case securely houses the components, offering durability and ease of use in various conditions.

Display

As mentioned above, we selected to use an E-Ink display, specifically the Waveshare 2.13-inch e-Paper HAT. This 2 coloured display is a compact, energy-efficient display designed for Raspberry Pi' and other microcontroller platforms. It has an onboard SPI control interface and features a 250×122-pixel resolution and utilizes Microencapsulated Electrophoretic Display (MED) technology, which reflects ambient light to display content, eliminating the need for a backlight. This results in a paper-like viewing experience with a wide viewing angle exceeding 170°. A full refresh takes 2 seconds, and the display supports partial refresh, allowing for quicker updates and reduced power consumption, taking only 0.3 seconds [14].

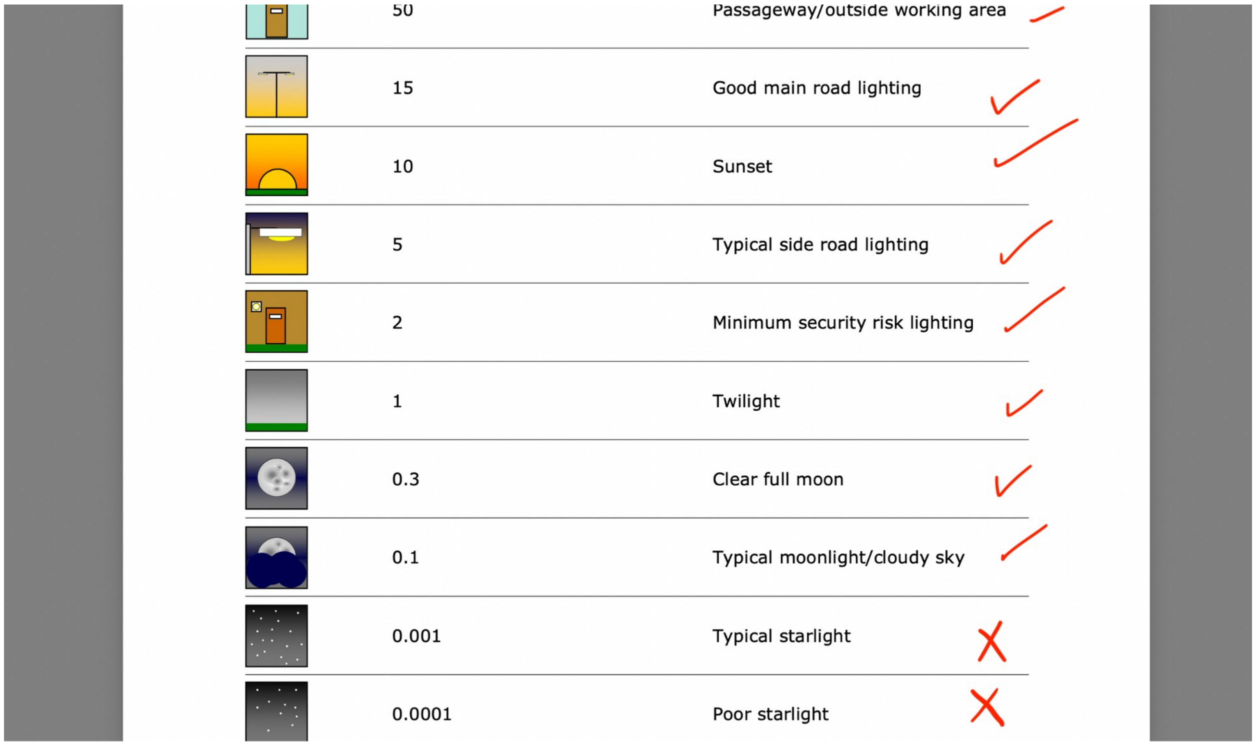


BS	Interface
1	3-line SPI
0	4-line SPI

Figure 43: Waveshare 2.13 E-Ink Display

Our primary reason for choosing this display over OLED or LCD was its power efficiency, given the nature of the display it consumes less than 0.01uA in deep sleep, which is the mode the display is in most of the time [14]. Its ability to retain images without continuous power make it suitable for battery-powered and always-on applications, only using tangible power when refreshing. We went specifically with the 2.13inch e-Paper HAT due to it having the quickest refresh rates as well as coming at a reasonable price point.

One area of interest regarding the display was its operational effectiveness in different lighting and temperature conditions. The full intent of VoidLink is it can be used outside while off grid, for avid backcountry campers or explorers. The datasheet claims the display is operational down to 0 degrees Celsius and up to 50 degrees Celsius. There was no specific range given for luminance, just that ambient light is required [14]. We did our own testing once we received the display to verify margins. In our testing we found the display refresh rate became slower at around 5 degrees Celsius and lower, the screen still operated properly but required more delay between button presses to complete the refreshes. Overall, we were content with this temperature range as the display would operate efficiently in most situations. Additionally, I did some luminance testing to see at what point ghost images become an issue. To run the tests I used my phone's front facing camera and a highly rated app called light meter to measure illuminance in terms of lux (1 lux is one lumen per square meter). The display operated at 100% efficiency down to 0.02 lux, which for comparison is less light than given off by the moon on an average night [15].



50	Passageway/outside working area		
15	Good main road lighting	✓	
10	Sunset	✓	
5	Typical side road lighting	✓	
2	Minimum security risk lighting	✓	
1	Twilight	✓	
0.3	Clear full moon	✓	
0.1	Typical moonlight/cloudy sky	✓	
0.001	Typical starlight	✗	
0.0001	Poor starlight	✗	

Figure 44: Depicts Lux measurements of various outdoor conditions

Screens - 10%

September to April (96 hours)

From our earliest discussions we knew that we would have a home/menu screen, a settings screen and some type of messaging screen. Over time these depictions changed quite a bit. Initially we were thinking of usings a raspberry pi zero instead of the pico 2 and an LCD instead of an E-Ink display, this would allow me to use micropython and be able to use different graphical user interface (gui) libraries to make actual buttons, labels, images, graphs and interactive features. The initial concepts were based on these ideas.

Screen Flow Conceptualization

- Not touch screen
- Show battery percentage/level
- Show signal strength

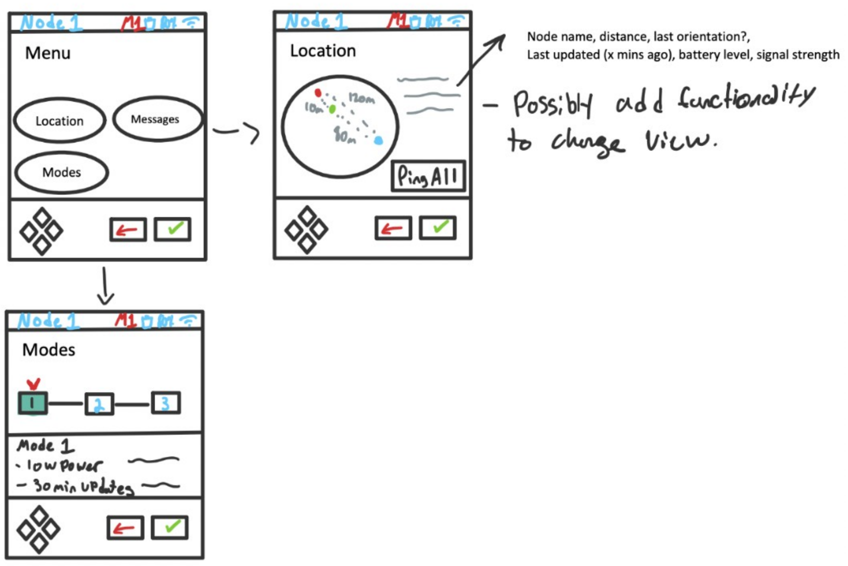
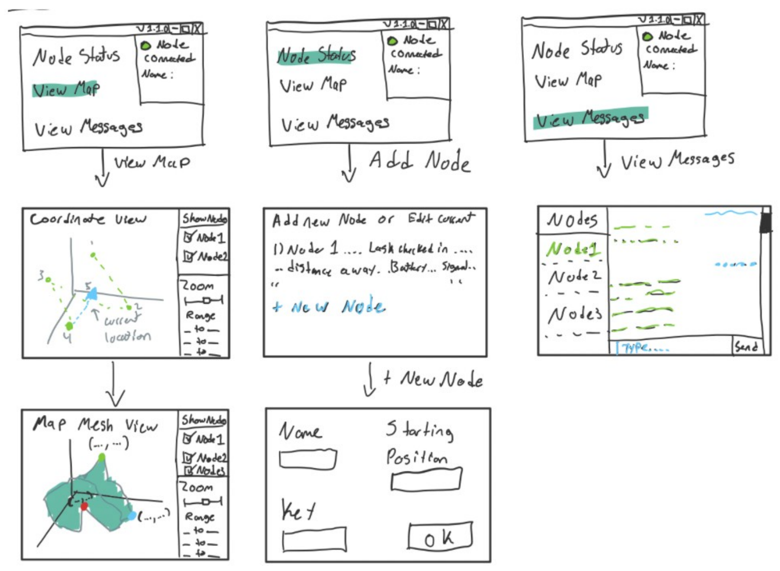

Node 1
Menu
Location
Messages
Modes
Location
Ping All
Node name, distance, last orientation?,
Last updated (x mins ago), battery level, signal strength
- Possibly add functionality to change view.
Modes
Mode 1
- low Power
- 30 min UPdates

Figure 45: Early Device Concepts



- Identifying where there is connectivity to other modules (Pyserial, psutil, etc.)
- Necessary features vs wanted features

Figure 46: Screen Flow concepts with different pages

In the end a lot of the same ideas are still present, but the screens morphed into what would work better with the new hardware change. The idea of viewing nearby nodes, messages, and node statistics all remained a crucial focal point. Unfortunately, the idea of a map or graph to plot the distances between nodes was unable to be brought into fruition, this is discussed more in a later section.

In November the decision was made to pivot our hardware to the raspberry pi Pico 2 and an E-Ink display. For the same reasons as stated above, power efficiency and a smaller form factor.

Switching to the E-Ink meant drawing pixel by pixel and relying on less dynamic and colourful screens. Touchscreens, graphing and keyboards were now out of contention. The change to Pico 2 was even more impactful. We went from using a minicomputer with an operating system to a small microcontroller. This meant a move from micropython to C as we now knew everything needed to run in sync with one another: Display code, Pico 2 code, transceiver code, and the network code. With these new constraints there was a strong focus on ease of use and making sure each screen stands out differently from the others. A new U.I. concept was created to simplify the design and bring it in line with what was possible given the hardware.

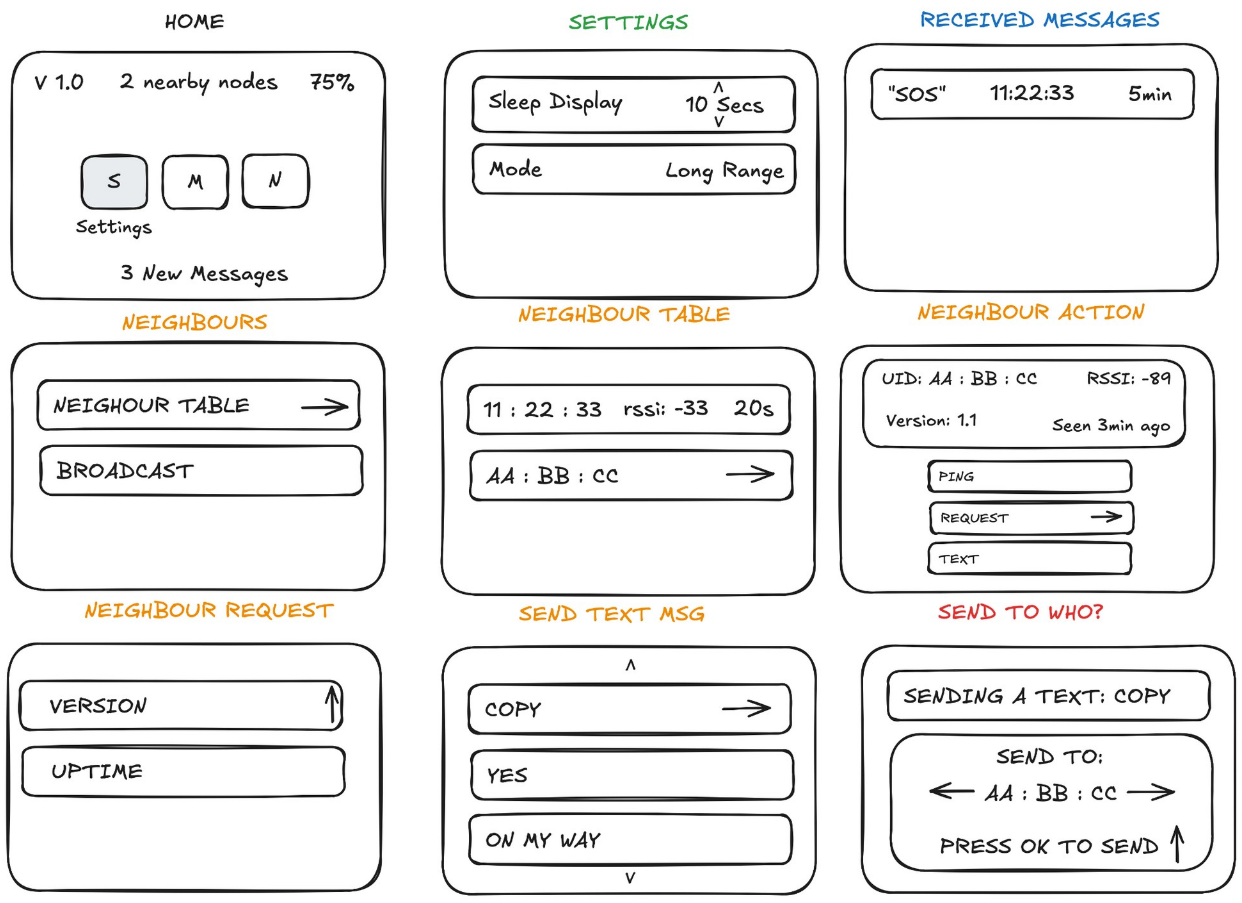


Figure 47: Display Screen Concept Designs

We came up with 4 main screens: home, settings, received messages and neighbours. Each of these screens also branch off into more detailed screens providing more information and actions. A lot of these decisions were made with the network side in mind. Having a neighbour table, calculating up-time getting firmware version and having set text messages. Likewise, there were some aspects included to fully utilize the hardware side as well, such as displaying the RSSI values between communicating nodes, battery level and changing the transceiver mode.

Home

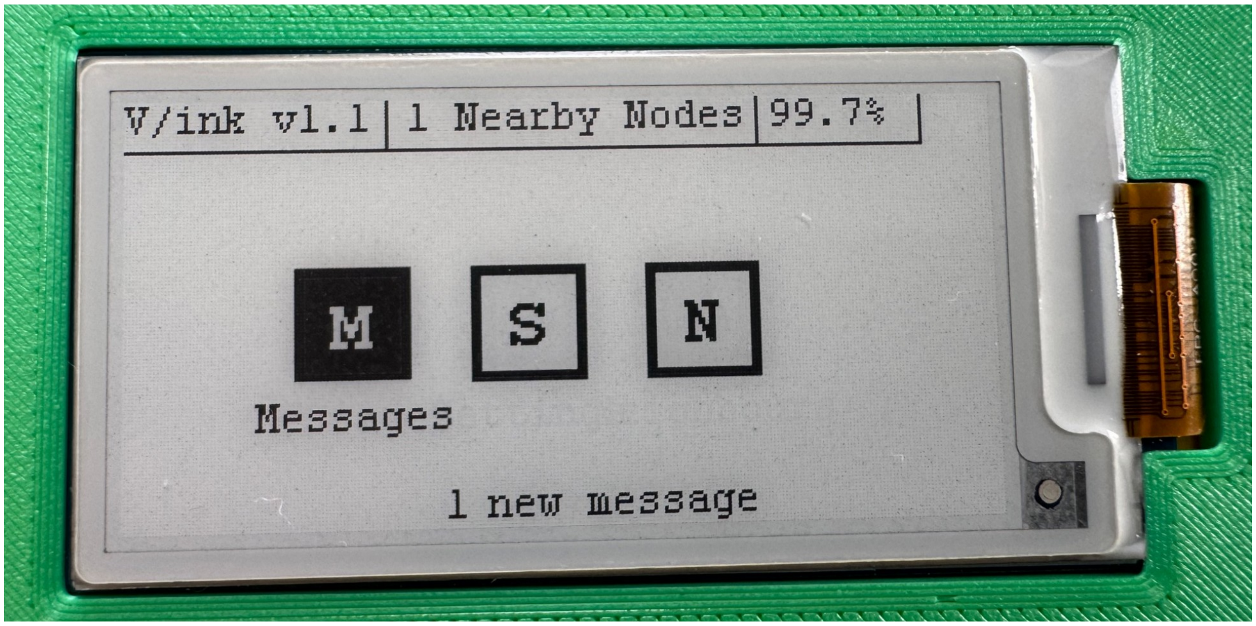
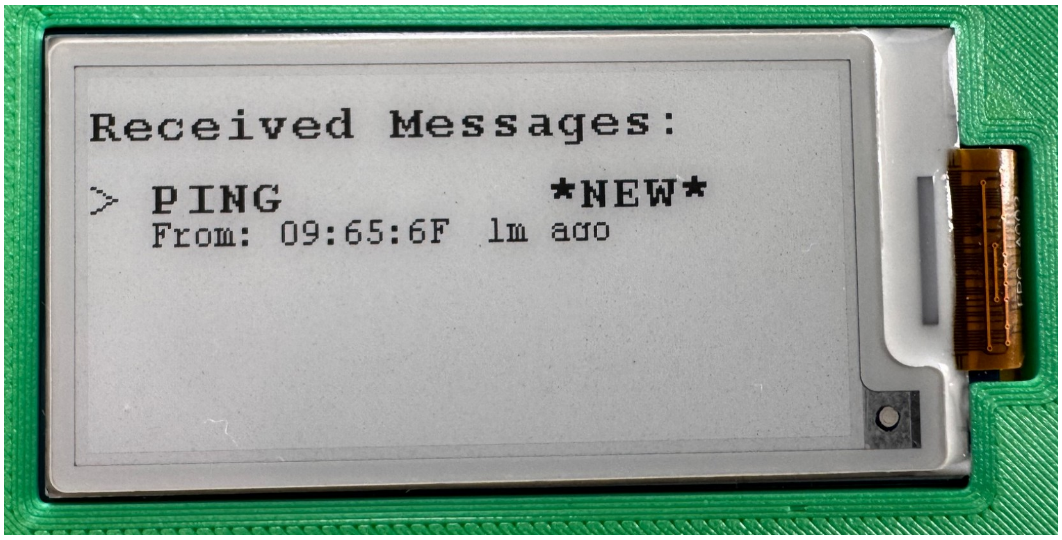


Figure 48: Home Screen

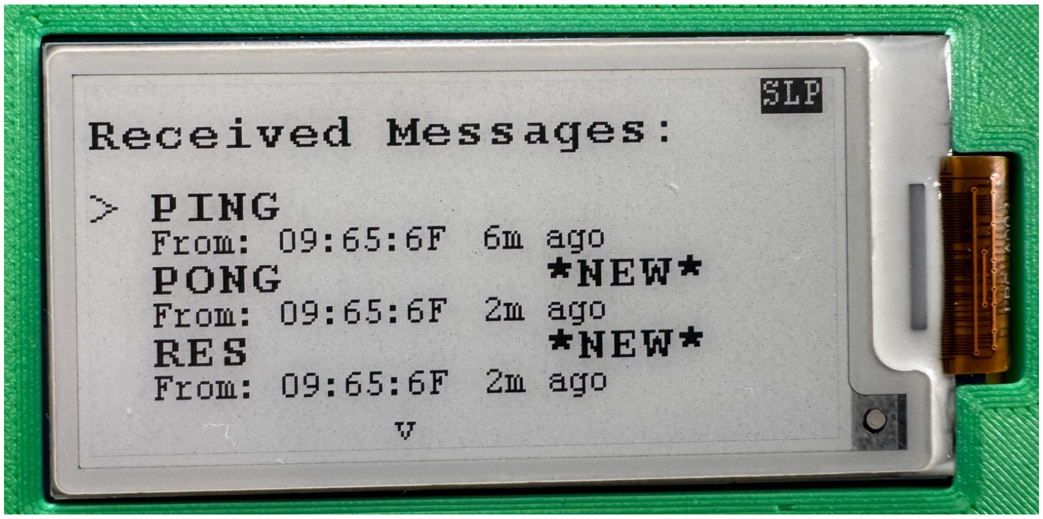
The home screen is important to get right as all navigation essentially goes through this screen. It also acts as a status screen, showing important details about the device, such as version, battery level and new messages. To show what screen is selected the icon's colour is inverted, the icon becomes black while the letter in the middle is white. Additionally, the selected icon's screen name is displayed under the icon to make it clear what is selected. There's some device status along the top header of the screen to show the firmware version that is running on the device, how many nearby nodes the device has communicated with and the current battery percentage. Additionally at the bottom of the screen (footer), a new message notification will be displayed if there's been a newly received message that hasn't yet been viewed.

Received Messages



```
Received Messages:

> PING              *NEW*
  From: 09:65:6F  1m ago
```

Figure 49: Ping Message received

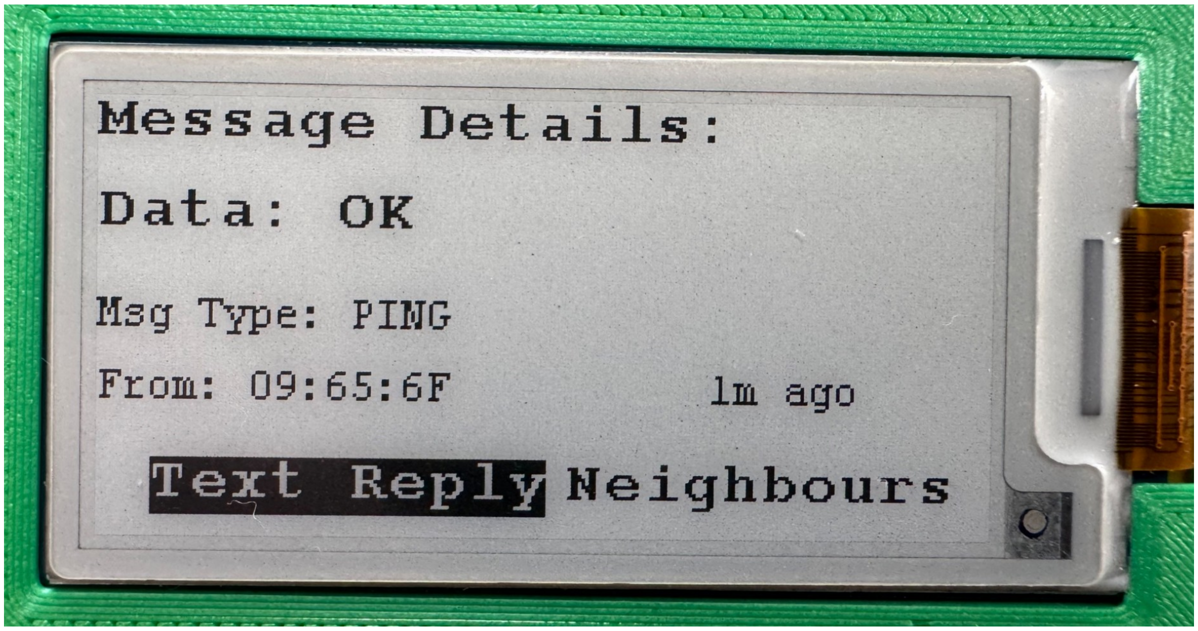


```
                          SLP
Received Messages:

> PING
  From: 09:65:6F  6m ago
  PONG              *NEW*
  From: 09:65:6F  2m ago
  RES               *NEW*
  From: 09:65:6F  2m ago
              v
```

Figure 50: Showing multiple newly received messages

This screen shows the list of all received messages 3 messages at a time. Up and down icons will appear if there is a page above or below the section currently being viewed. The message type is depicted in the biggest font as its the key differentiator between all the messages. Secondly it also shows what node the message came from and how long ago it was received. The length of the list is dynamic and will grow to the size of saved messages which is defined in the network side, once this number is reached, since it's a cyclical list it will loop back to the first message and overwrite it. This makes it important to have a way of knowing what messages are new and unread. Which is why displaying how long ago the message was received is important. Additionally, there is a *NEW* message notification beside newly received and unread messages. This notification clears if the user’s cursor is on the message.

Received Message Details



Message Details:

Data: OK

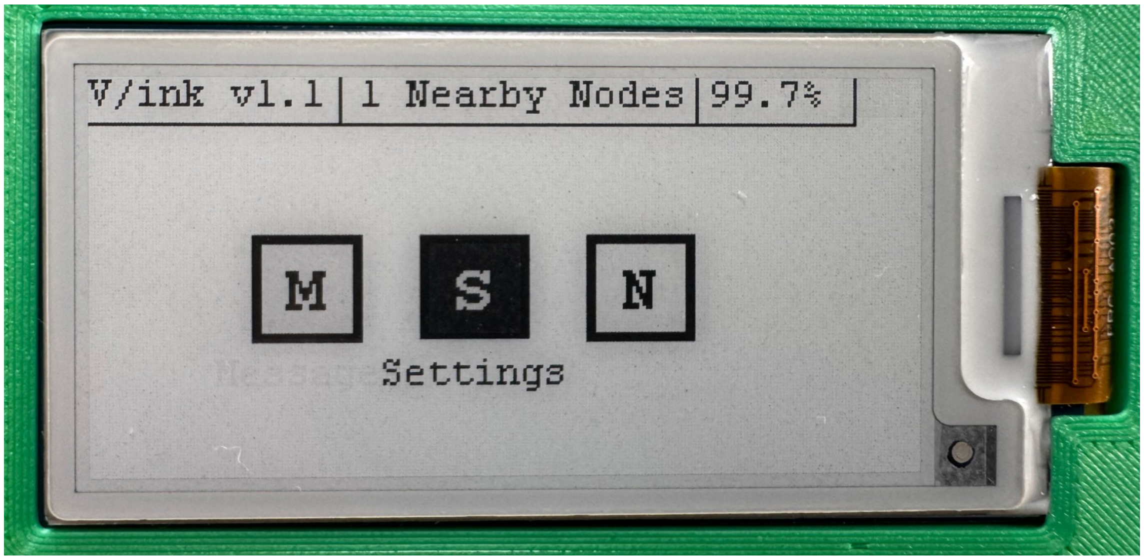
Msg Type: PING

From: 09:65:6F 1m ago

Figure 51: Message details screen

This screen expands the message to be viewed more clearly in isolation. It shows all the details of the received transmission, here the data packets, neighbour ID, message type, how long ago the message was received are shown as well as also providing the option to reply.

Settings



V/ink v1.1 | 1 Nearby Nodes | 99.7%

Figure 52: Home screen with settings selected

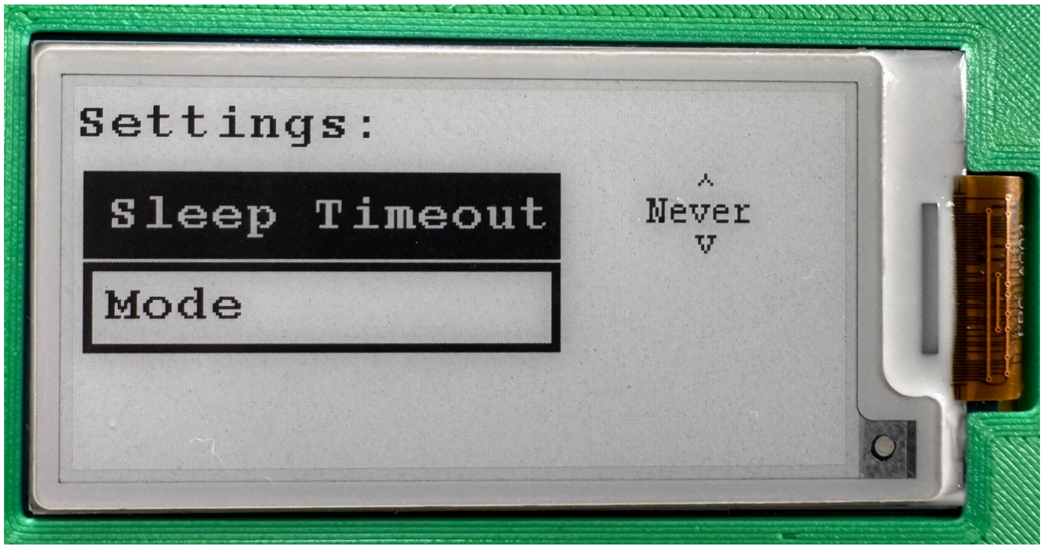


Figure 53: Sleep timeout setting

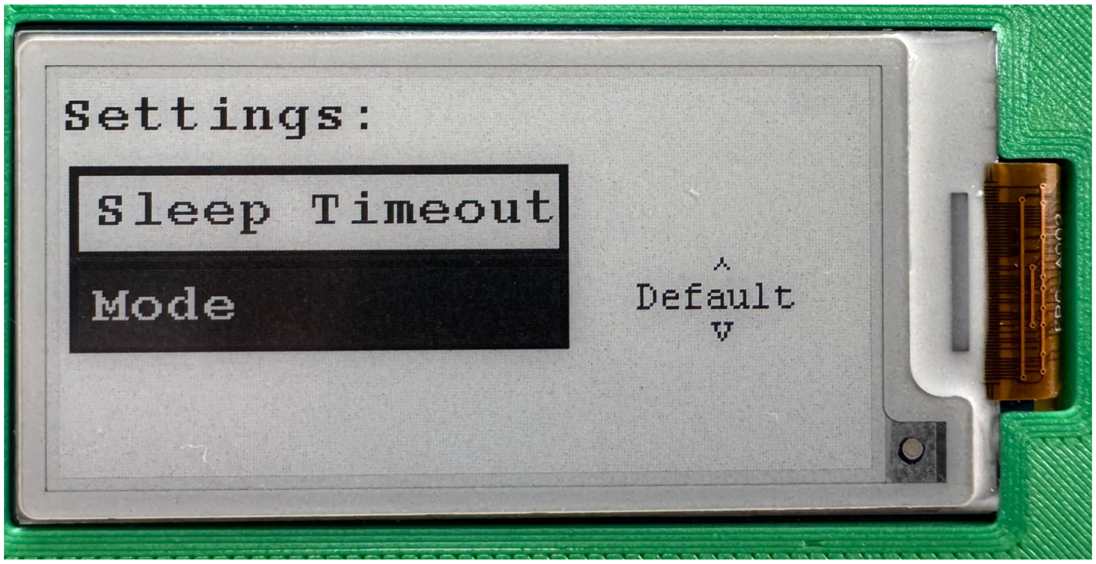


Figure 54: Operation mode setting

In settings there are 2 configurable options, display timeout and mode. For display timeout the options are 5 seconds, 10 seconds, 30 seconds, 1 minute, 2 minutes and never. This is the amount of time of inactivity required before the display goes to sleep. A button interrupt will restart the timer. Mode is in reference to the transmitting mode, there are 3 options, default, fast and long range. A key to note is nodes must be using the same node to send and receive to each other.

Neighbours

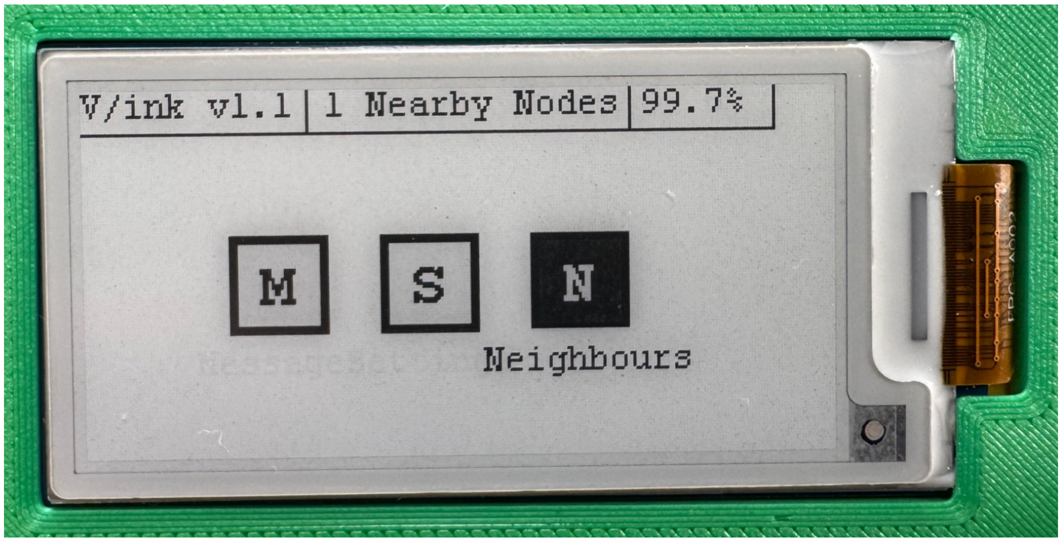


Figure 55: Home screen with neighbours selected

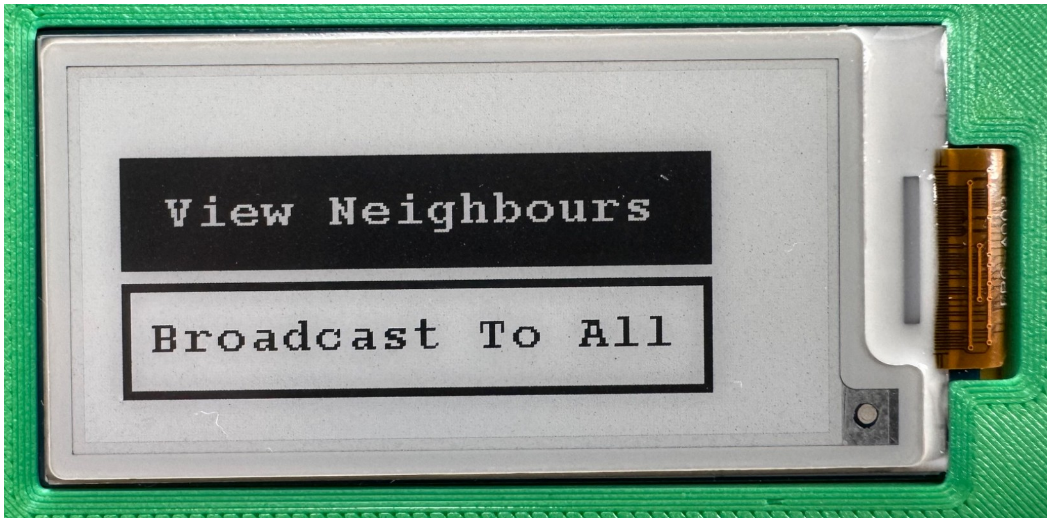
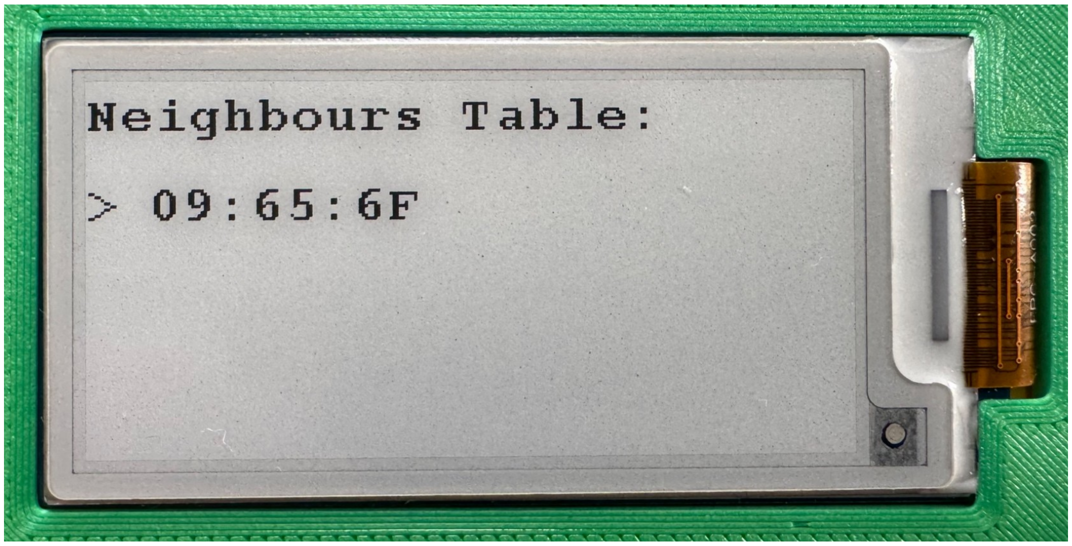


Figure 56: Neighbours screen with neighbours table selected

This is an intermediate screen to choose if you'd like to view the neighbours table or broadcast a message out to everyone.

Neighbours Table

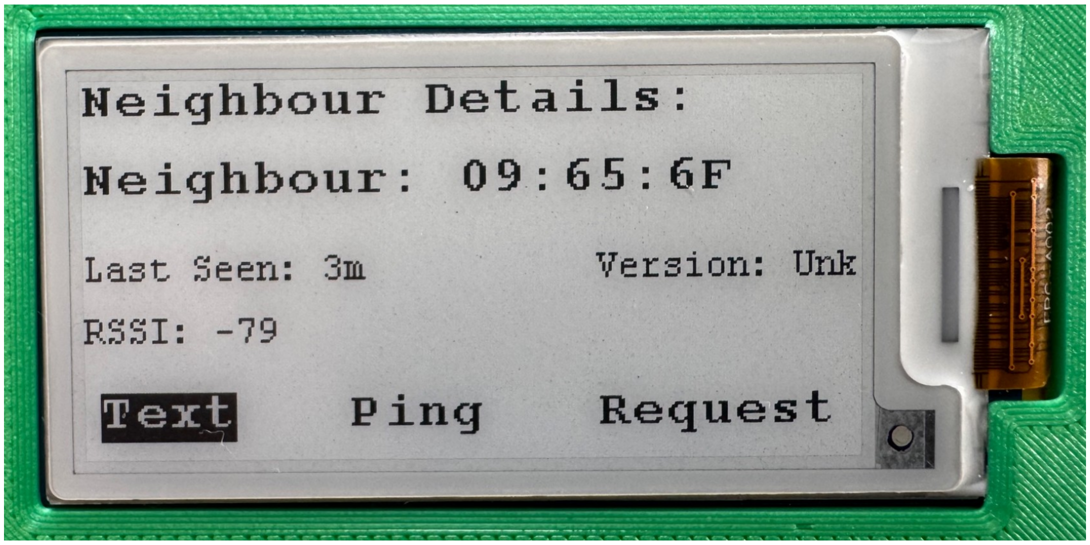

Neighbours Table:

> 09:65:6F

Figure 57: Neighbours table screen with 1 neighbour present

In a similar way to the received messages screen this screen shows a list of nearby neighbours by neighbour ID that the current device has sent to and received from.

Neighbour Details

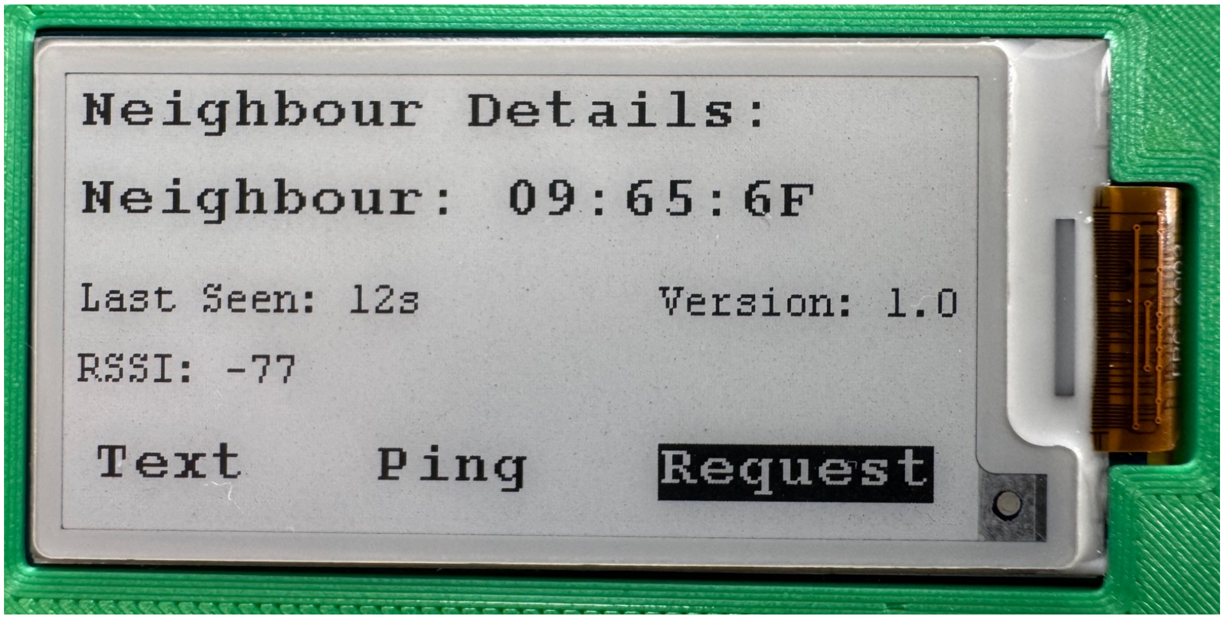

Neighbour Details:

Neighbour: 09:65:6F

Last Seen: 3m Version: Unk

RSSI: -79

Figure 58: Neighbour details screen with text message selected


Neighbour Details:

Neighbour: 09:65:6F

Last Seen: 12s Version: 1.0

RSSI: -77

Figure 59: Updated neighbour details screen showing version and updated last seen

This screen shows more information about the selected neighbour. Information such as up-time, version, the RSSI value when last in range and how long ago the neighbour was last seen. You can see in the first image we don't know what firmware version this neighbour is running on, by sending a request for their version we then see they're on version 1.0 and the last seen also updates. Additionally, here there is a list of actions the user can perform with the other neighbour such as sending a ping, text, or request (up-time, version).

Message Types

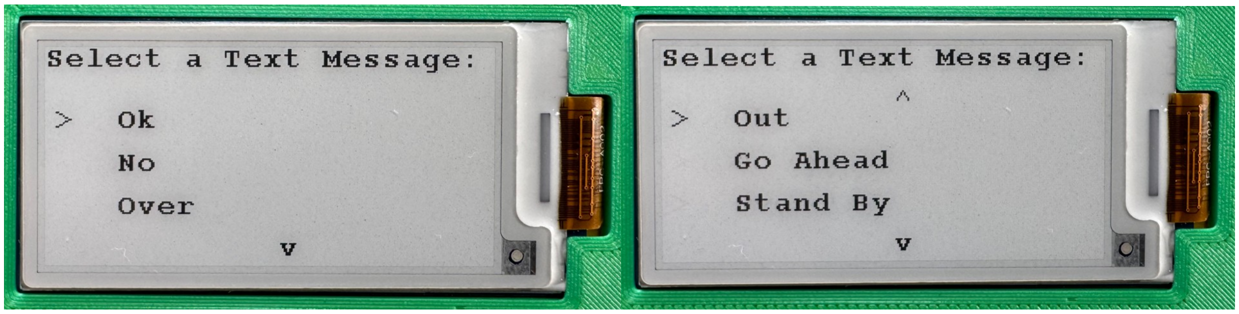


Figure 60: Send a text message screen

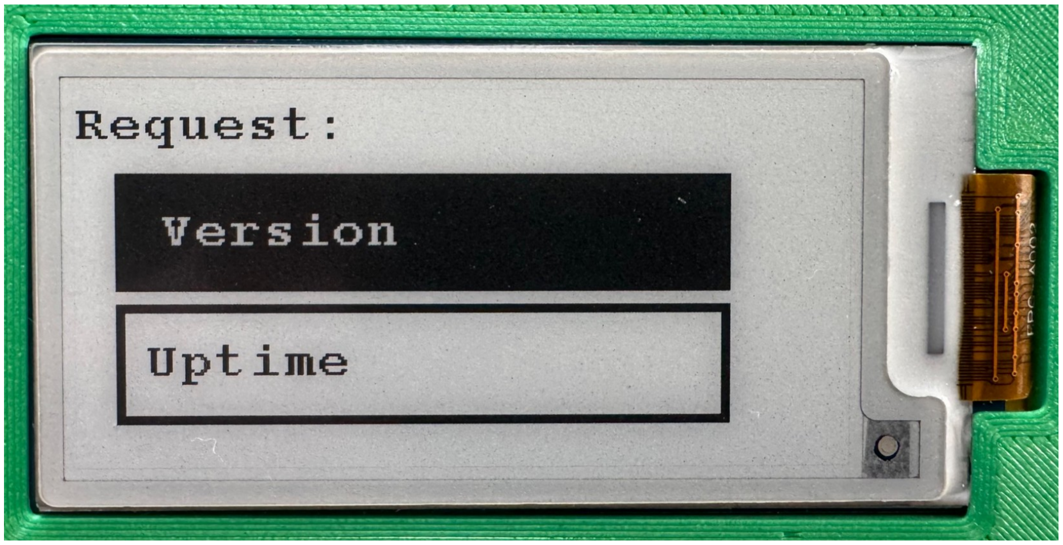


Figure 61: Neighbour request screen with version selected

Send To and Broadcast

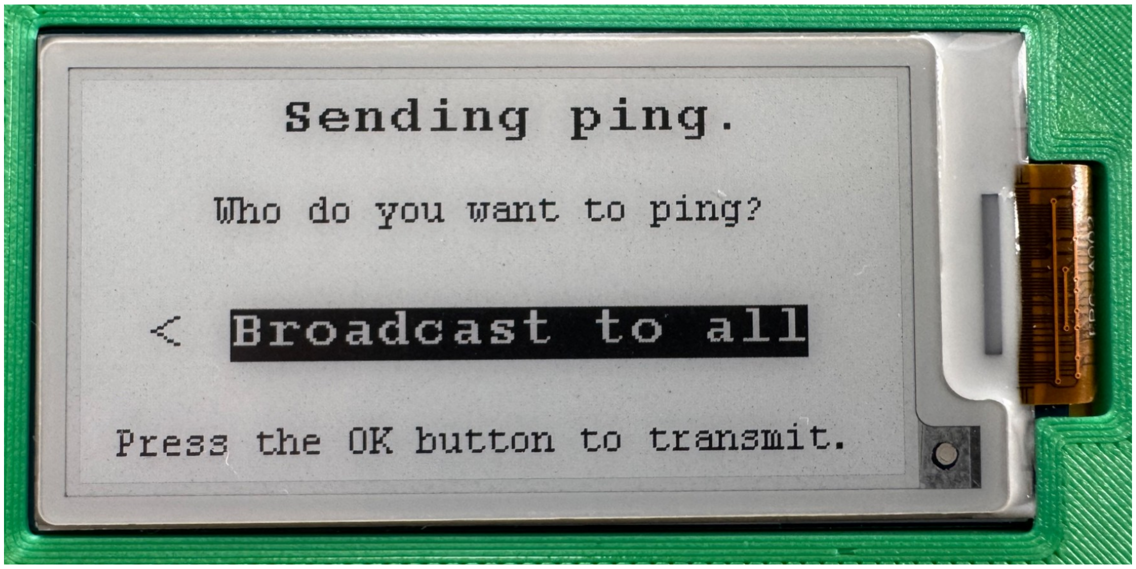


Figure 62: Sending a ping to everyone

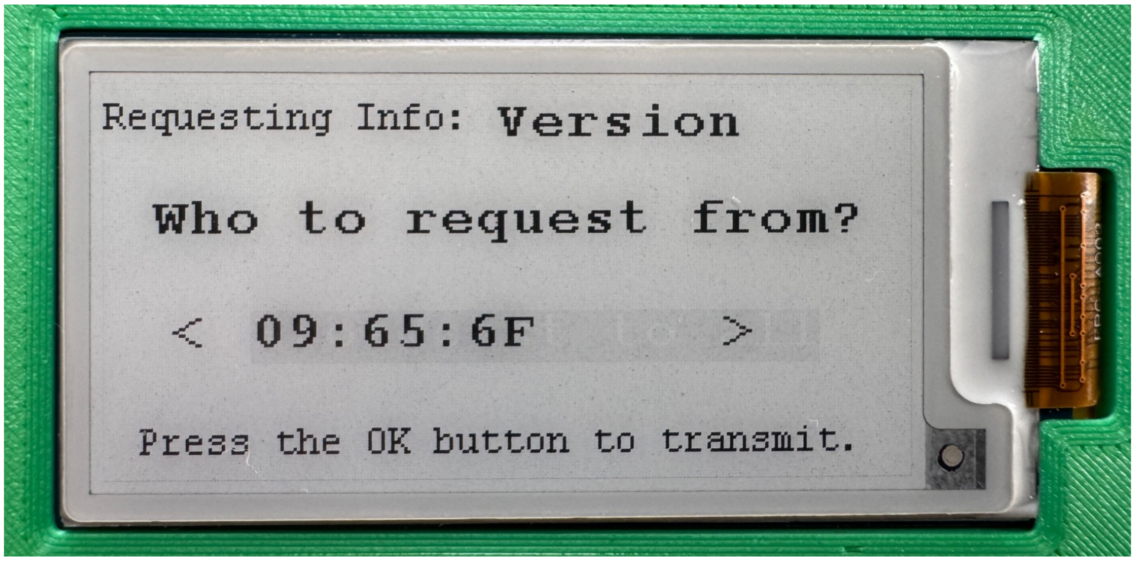


Figure 63: Requesting firmware version from specific node

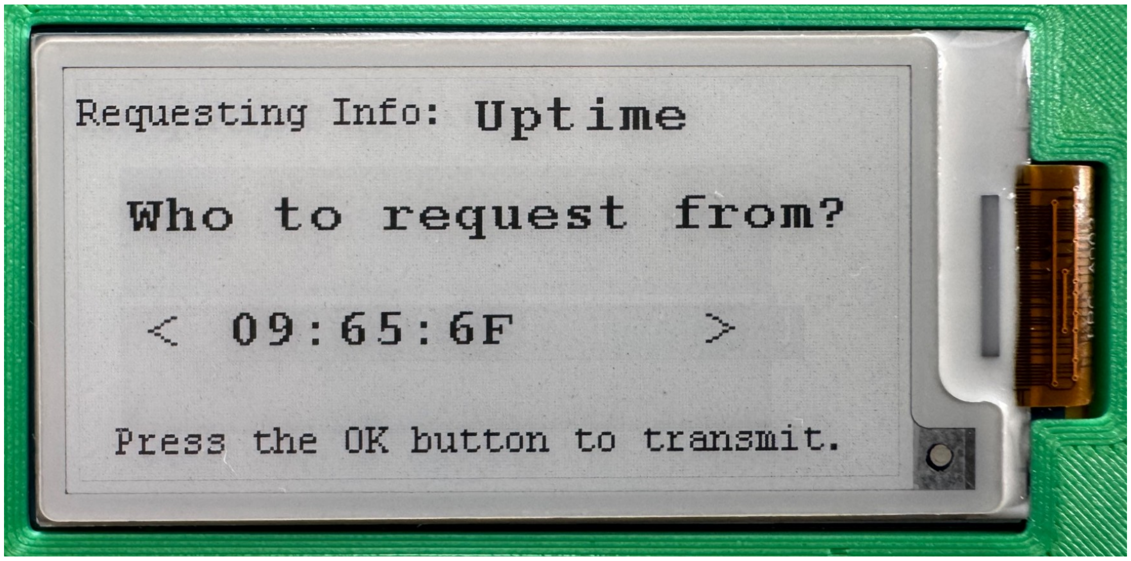
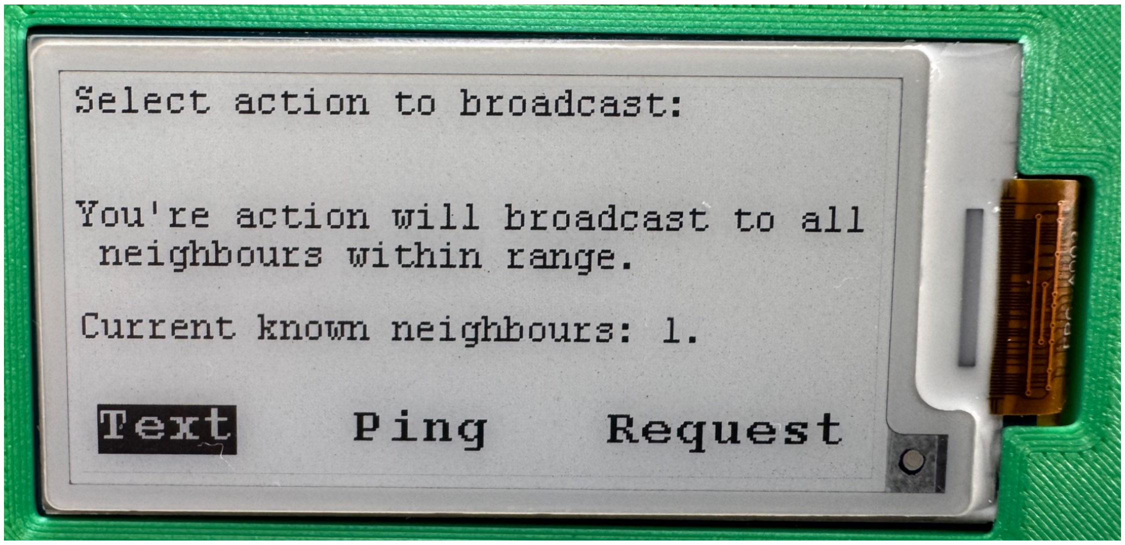


Figure 64: Requesting uptime from specific node

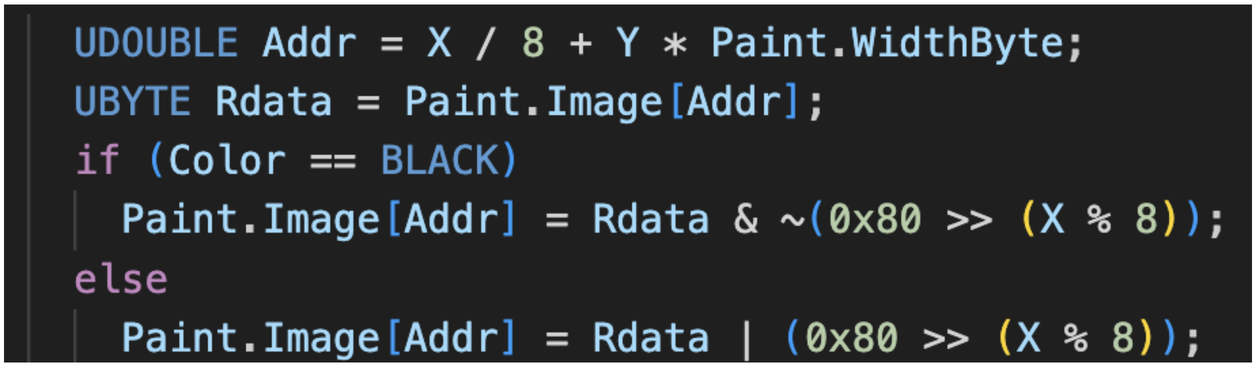


```
Select action to broadcast:

You're action will broadcast to all
 neighbours within range.

Current known neighbours: 1.

Text      Ping      Request
```

Figure 65: Broadcast screen

This screen acts as a final confirmation of who you want to transmit to, whether it be to a specific neighbour or to broadcast to all within range, using the cycle buttons the user can cycle through who to send to. The screen changes slightly based on the message type the user selects. There is also a specific broadcast screen if the user selects broadcast from the neighbours screen.

To achieve these screens you have to quite literally set each pixel to either black or white. Thankfully the company we purchased the display from also provided a driver file to help draw basic things like text and shapes through converting to and setting each pixel. For a given screen a new image instance is created and stored in the image buffer, each drawing is then saved to this instance then along with x and y coordinate and colour choice the pixels are drawn in accordingly. The draw pixel function also takes into account rotation (changeable setting), mirroring (changeable setting), and pixel scale (bit depth, font dependant), and modifies the image buffer accordingly [16].



```
UDOUBLE Addr = X / 8 + Y * Paint.WidthByte;
UBYTE Rdata = Paint.Image[Addr];
if (Color == BLACK)
    Paint.Image[Addr] = Rdata & ~(0x80 >> (X % 8));
else
    Paint.Image[Addr] = Rdata | (0x80 >> (X % 8));
```

Figure 66: Modifying the image buffer

Each byte in the buffer holds 8 pixels. It sets or clears the appropriate bit depending on the color, where black is 0 and white is 1, given we choose a monochrome display these are our only 2 cases.

Given the monochrome and static aspect of the E-Ink, there was a strong emphasis to make each page feel different and specific. This is the reason some pages use a cursor versus using inverted colours to show selections. The idea was to alternate the selection type page by page for more uniqueness and enhanced user experience. Likewise this is why a transmission animation was added upon sending a message, to give the user satisfying visual feedback.

User Input and Output – 6.25%

October to April (60 hours)

There are a few ways user input and output are achieved for VoidLink. The main inputs being the push buttons. There are 6 push buttons all with a specific purpose. 2 buttons are dedicated to cycling through selectable options on the screen. There are also 2 buttons dedicated to confirming (enter button) and going back (cancel). Then there is a button to go directly back to the home screen without having to press the back button several times. The final button is a sleep/wake button, if the display is awake it will sleep the display and if the display is asleep it will wake up the display. All other push buttons will also wake the display and reset the sleep timer.

As stated in the hardware modules there are LEDs to show when the device is powered off of USB C, one to show if the device is powered off of battery and one to show when the battery is low/dead. There is also one other LED we call the status LED. We use this in 2 ways, firstly it flashes on a reboot so we know when the code is being re-run and secondly it is used to show that there are new messages waiting.

Finally through the use of cursors, send animation and page arrows the user will always know what is clearly selected and what action was and will be performed if requested.

Features - 5%

November to April (48 hours)

Apart from the screens and handling button interrupts there are a few other key features that were needed to achieve VoidLink.

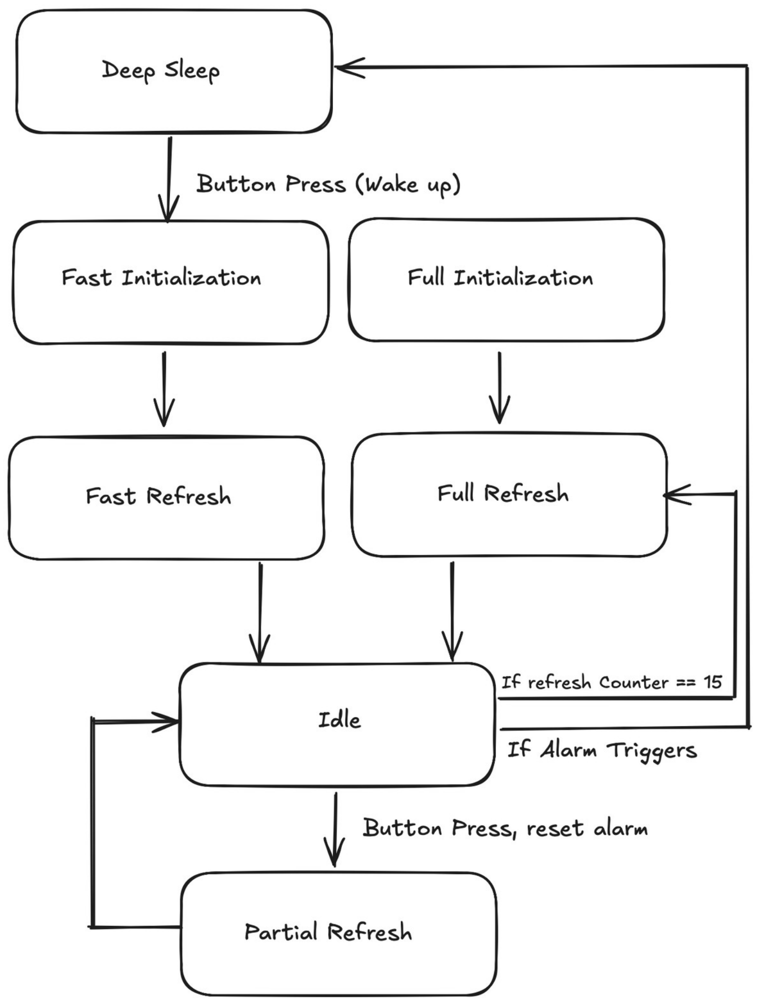
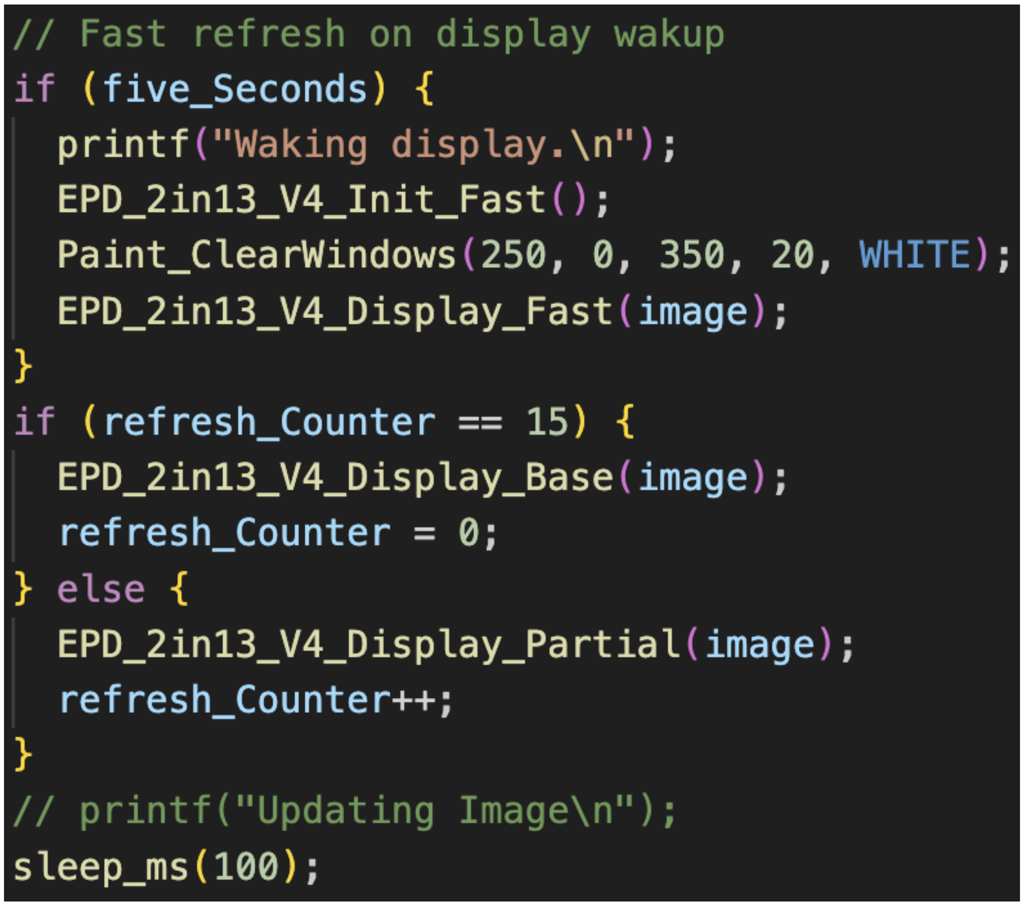


Figure 67: Screen refresh state diagram

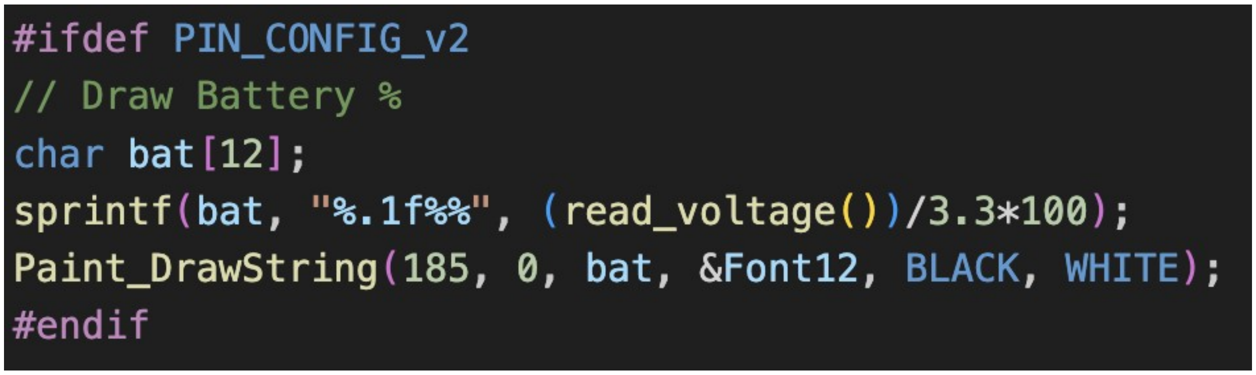


```c
// Fast refresh on display wakup
if (five_Seconds) {
  printf("Waking display.\n");
  EPD_2in13_V4_Init_Fast();
  Paint_ClearWindows(250, 0, 350, 20, WHITE);
  EPD_2in13_V4_Display_Fast(image);
}
if (refresh_Counter == 15) {
  EPD_2in13_V4_Display_Base(image);
  refresh_Counter = 0;
} else {
  EPD_2in13_V4_Display_Partial(image);
  refresh_Counter++;
}
// printf("Updating Image\n");
sleep_ms(100);
```

Figure 68: Screen refresh code

The first main feature is screen refreshing. This of course is the key point of the E-Ink display and we leveraged Waveshare's library to easily call each type. The trick is knowing when you want to do a certain type of refresh, for VoidLink we have 3 types. Full refresh, fast refresh and partial refresh, each of these serve a specific purpose and have different tradeoffs associated with themselves. Full refresh takes about 2 seconds to complete, it flashes the screen several times but it is the best option for clearing images and preventing ghost images. We utilize this refresh when changing screens to wipe the screen but also signal the user that a page change has been made. Additionally we have a counter that will do a full refresh every 15 button presses to ensure proper refreshing, as it is recommended to do so by Waveshare and partial refreshing should only be done a certain amount of times before full refreshing again. Fast refreshing takes around 1 second and flashes the screen once. We use this type when waking up from a deep sleep, it gives a clear visual to the user that the display has woken up while also not taking too long to perform since there isn't much to clear on a wake up. We also use fast in this case because using a partial refresh after initializing from a wake up will lead to definite ghost images. Lastly we have the partial refresh type, this is the main reason we went with this specific Waveshare display. The partial refresh only takes 0.3 seconds which makes the user feel as if there is no latency when using the buttons to cycle through options. We use this refresh within screens, it is perfect when moving a cursor around the screen or updating small information such as the new message notification or the sleep notification.

The new message notification keeps track of how many saved messages there are and which of the saved messages has the cursor not cycled to yet. Once the cursor cycles over that message on the next refresh the *NEW* notification will disappear and the new message notification on the home screen will decrement by 1.



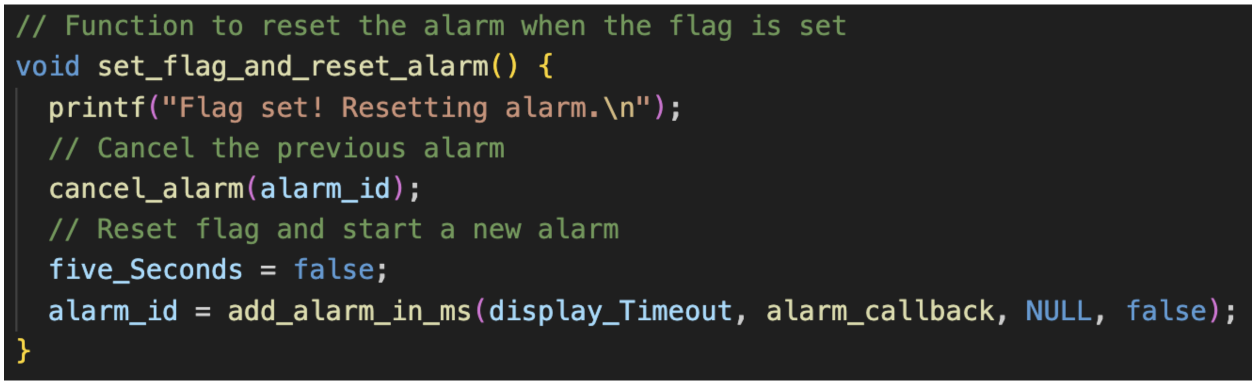
```
#ifdef PIN_CONFIG_v2
// Draw Battery %
char bat[12];
sprintf(bat, "%.1f%%", (read_voltage())/3.3*100);
Paint_DrawString(185, 0, bat, &Font12, BLACK, WHITE);
#endif
```

Figure 69: Displaying battery by reading ADC and converting voltage to percentage

On the home screen we also show battery percentage. We read from the ADC connected to the power rail and return a voltage value then convert to percentage to display to the user in more simply standardized terms.

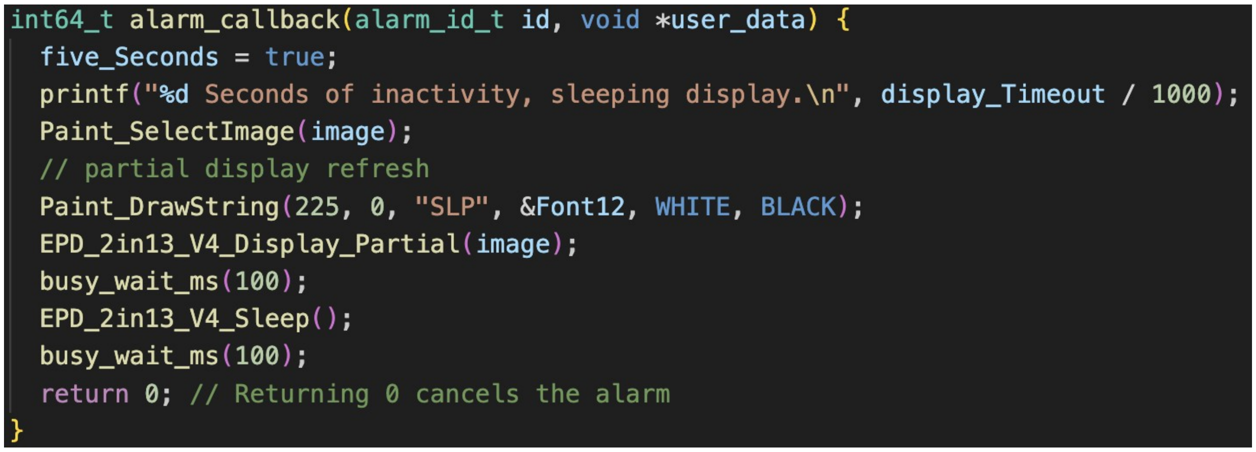
On the home screen there is also the number of nearby nodes/neighbours, this is achieved by displaying the count of saved neighbours.

Another feature is dynamic page scrolling. On the received messages and neighbours table page the number of neighbours and messages are constantly changing, the user needs to be able to easily navigate through different pages. As shown above, page markers will appear only if there are messages on the next page or on the previous page. The user can cycle forwards and backwards across items. If the user is at the end of the page and tries to cycle further the cursor will stay on the same page but jumps down/up to the last message on the page, acting as a cyclical cursor. Additionally if no message or neighbours are present the screen will show that there are none and will prevent the cursor from incrementing.



```
// Function to reset the alarm when the flag is set
void set_flag_and_reset_alarm() {
  printf("Flag set! Resetting alarm.\n");
  // Cancel the previous alarm
  cancel_alarm(alarm_id);
  // Reset flag and start a new alarm
  five_Seconds = false;
  alarm_id = add_alarm_in_ms(display_Timeout, alarm_callback, NULL, false);
}
```

Figure 70: Cancel current alarms and set new alarm



```c
int64_t alarm_callback(alarm_id_t id, void *user_data) {
  five_Seconds = true;
  printf("%d Seconds of inactivity, sleeping display.\n", display_Timeout / 1000);
  Paint_SelectImage(image);
  // partial display refresh
  Paint_DrawString(225, 0, "SLP", &Font12, WHITE, BLACK);
  EPD_2in13_V4_Display_Partial(image);
  busy_wait_ms(100);
  EPD_2in13_V4_Sleep();
  busy_wait_ms(100);
  return 0; // Returning 0 cancels the alarm
}
```

Figure 71: timer function to sleep the display if alarm triggered

A very important feature is the ability to sleep the display, this is critical to ensure long battery life, and only need the screen powered when refreshing. There is a timer alarm that triggers after a set amount of time, which is user defined in the settings (5s, 10s, 30s, 1min, 2min, never). If the timer triggers it calls a function that displays the sleep notification at the top right of the screen and then sets the state of the display to deep sleep. This way the display draws less than 1uA. If a button is pressed then the timer resets and if the display is sleeping it wakes up and re-initializes the display. We also have a button dedicated to sleeping and waking up the display if the user chooses, it is useful if the user prefers a longer timeout but wants the ability to sleep the display when they want.

Mechanical Case – 3.75%

February 18th to April 7th (36 hours excluding printing time)

Given the unique formfactor and peripherals it was apparent a custom case would need to be designed and fabricated. The clear option was to model a case in CAD software and 3D print it out of PLA plastic. The case was modeled using FreeCad and dimensions were based on the PCB specifications and the display datasheet with additional tolerances. There are 4 separate model parts: front, back, button cap, button topper.

The front was constructed from 8 sketches, 4 pads, 5 pockets and 4 fillets. With an additional 7 shapestrings and pockets to imprint our logo and button icons.

The back was made with 11 sketches, 6 pads, 4 pockets, 5 fillets and 1 hole with an additional 4 shapestrings and pockets to imprint our names.

The button cap was made from 4 sketches, 3 pads and 1 pocket.

The button topper consisted of 2 sketches 1 pad 1 pocket.

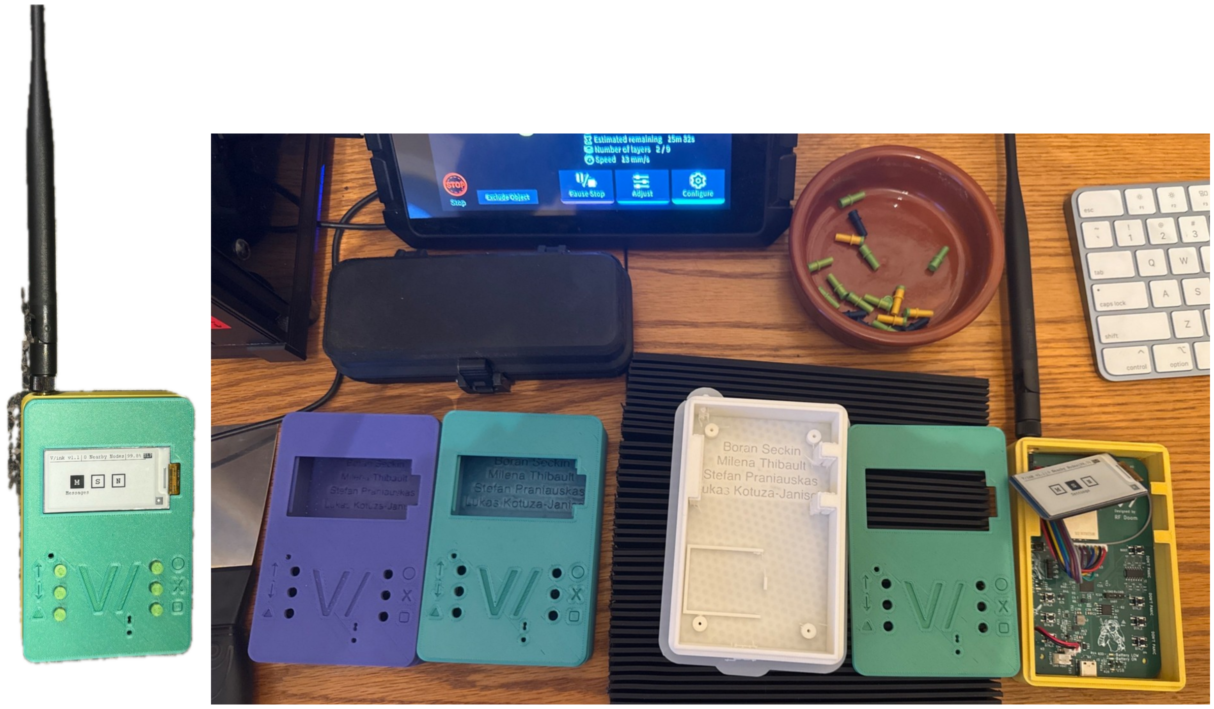


Figure 72: Final case design

Key features of the front piece were the buttonholes, the LED holes and the display slot and tabs to keep the display fitted tightly. The print time for this piece is around 1 hour and 53 minutes.

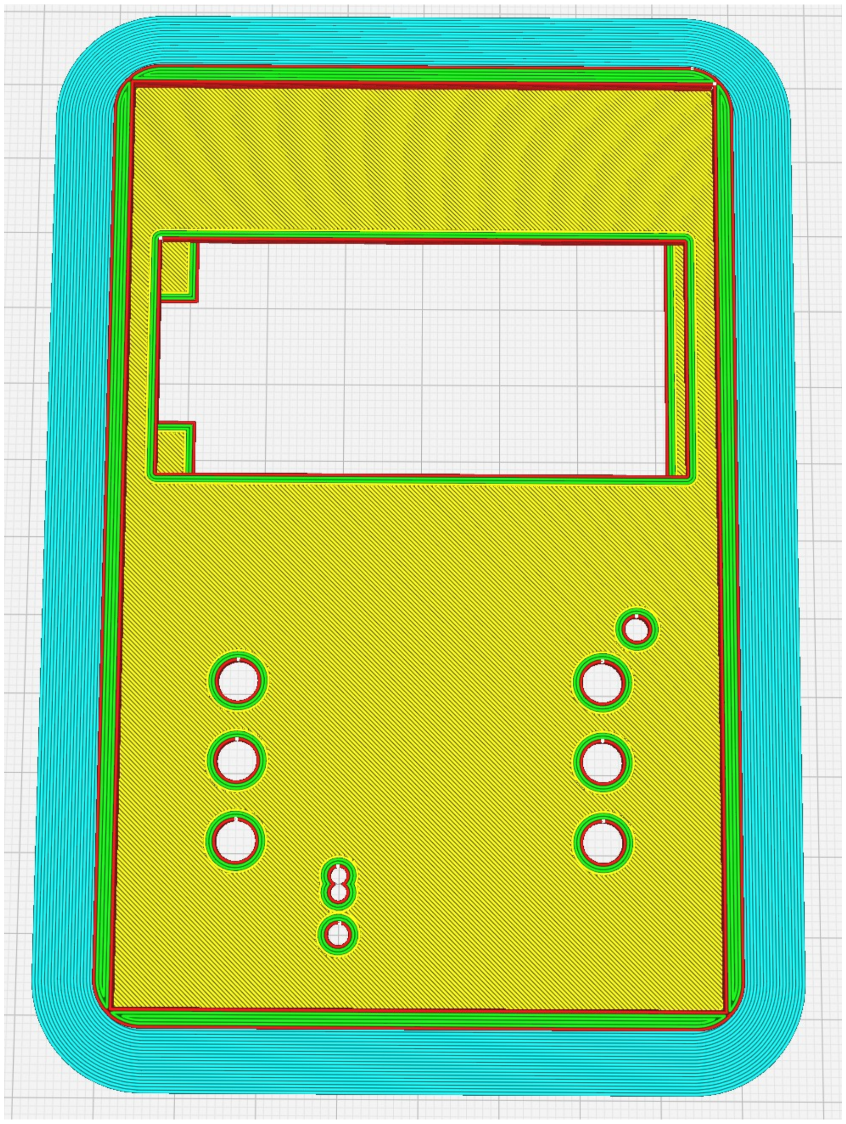


Figure 73: Final front piece design

Key features of the back are the antenna socket which allows for the cable to be firmly bolted in while still allowing the user to change antenna sizes if desired. There is also a display lock which when the front and back connects keeps the display firmly locked in and prevents the display from falling into the case. There's also a USB C slot which aligns with the port on the PCB. There are also 4 standoffs added to the inside of the case, the reason for this is to keep the PCB a mounting point using M2 screws (the holes in the standoffs are threaded for M2 screws to be used). The other reason for the standoffs are to keep the pins coming out of the bottom of the PCB from touching the bottom of the case and allowing space to keep the battery. The battery container keeps the battery securely positioned so it doesn't shift around in the case. The print time for the back piece is approximately 8 hours and 3 minutes.

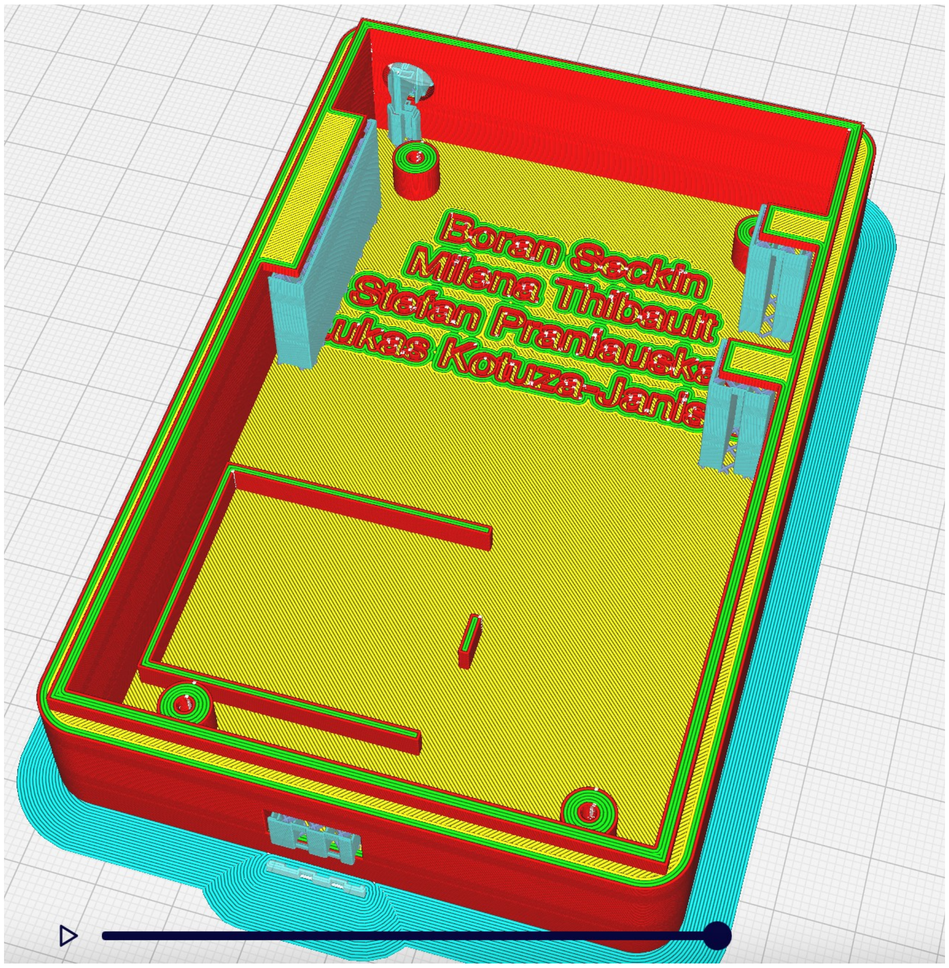


Figure 74: Final back piece design

The button caps are essentially for using the device in the case. The buttons on the PCB are quite deep in the case and are not the farthest extending PCB component height wise either. A button cap was devised to fit over top of the PCB button and extend out to the front piece of the case. The button cap also has a lip to keep it locked in the case and prevents it from falling out of the case. One button takes around 7 minutes to print, each case has 6 buttons, which gives a total print time of 42 minutes.

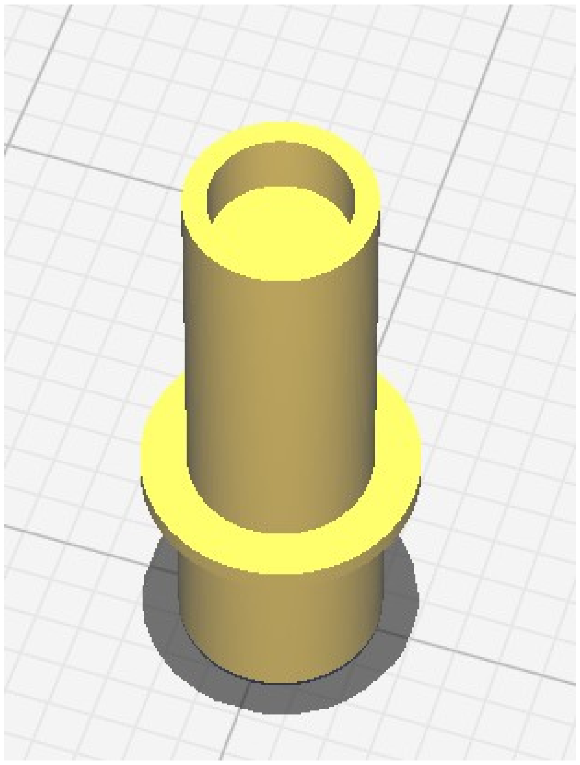


Figure 75: Final button cap design

The button topper was created to make case assembly easier, keeping the button caps locked in the front piece until they're overtop and resting on the PCB buttons. The print time per topper is 1 minute, given 6 takes around 6 minutes.

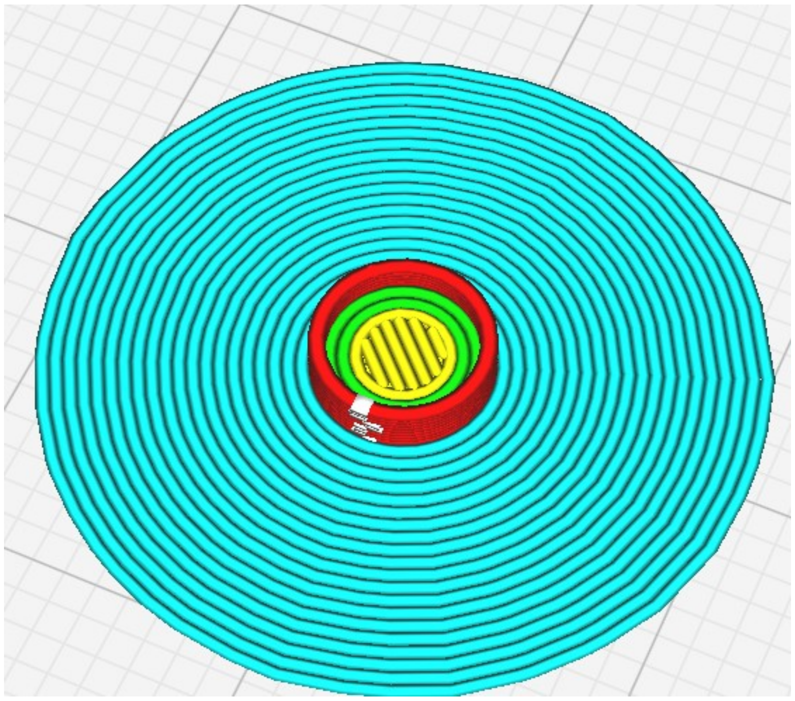


Figure 76: Final button topper design

The front and back piece are designed to snap together, with the button piece having an extruding inner lip and the front piece having an outer lip so that the 2 overlap and friction keeps the case securely shut.

Progress and Results

Module I (UI): Lukas Kotuza-Janisch, 100% Complete

The user interface module was tested incrementally at every step. Each page was constructed and tested in a shell setting with test data to ensure proper formatting and behaviour. The cursor logic for each screen was also tested separately in isolation before integrating everything. Most of the work in this module was additive, and integration with all the other modules.

I learned a great deal of software and product development. It was a rapidly changing environment with new ideas and challenges every day. Constant revisions to the case designs and formatting changes to the UI based on code revisions. It was interesting and fun to work on embedded programming and work with different code base libraries to create our own function and own use.

Timeline of Milestone Tests

Initial button concept testing in Python (November 7, 2024):
https://drive.google.com/file/d/1UbAMeTVh9cbh4JfzX5qPsCf-qu29V9xj/view?usp=sharing

Running demo display code (November 27, 2024):
https://drive.google.com/file/d/158PEagqyLY4dJVuVOuRc1IMLiWVLe4cA/view?usp=sharing

First display test showing Hello World (January 14, 2025):
https://drive.google.com/file/d/15ptvwqtOv9Uiql36titealmaGMxLqi70/view?usp=sharing

Display, button and interrupt testing (January 14, 2025):
https://drive.google.com/file/d/1KRI7_pIn6xeAwb7DluhU_YXcxw4bweaS/view?usp=sharing

First transmit with display (January 20, 2025):

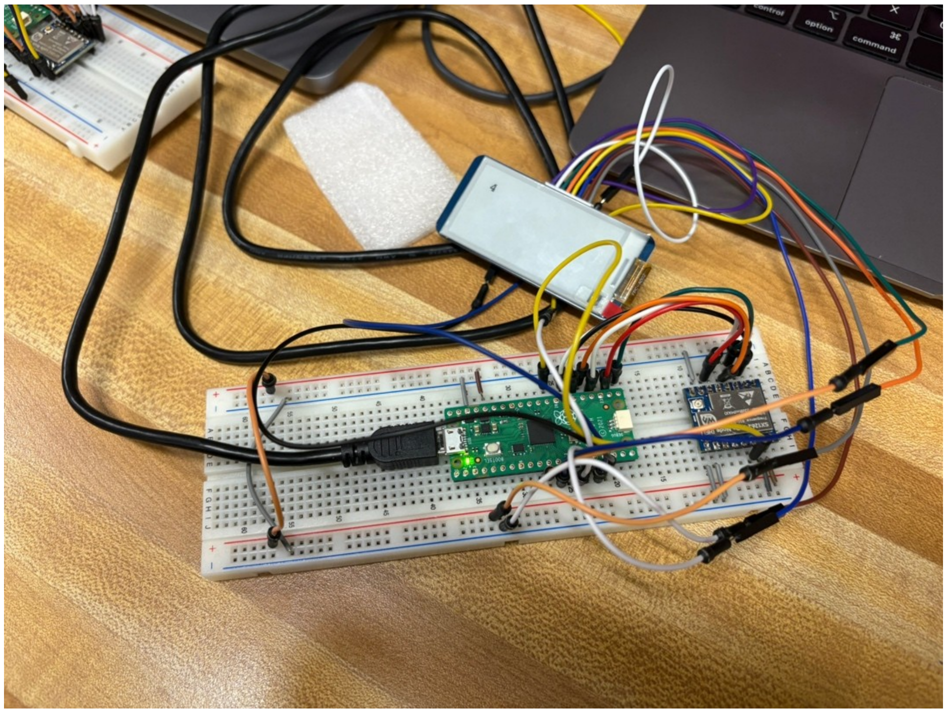


Figure 77: First Data Ever Sent ("4" Shown on the E-Ink Display)

First Time Sending a Text (January 20, 2025)

Display Sleep Testing (January 22, 2025)

First Range test (February 11, 2025)

First case concept (February 18th, 2025):

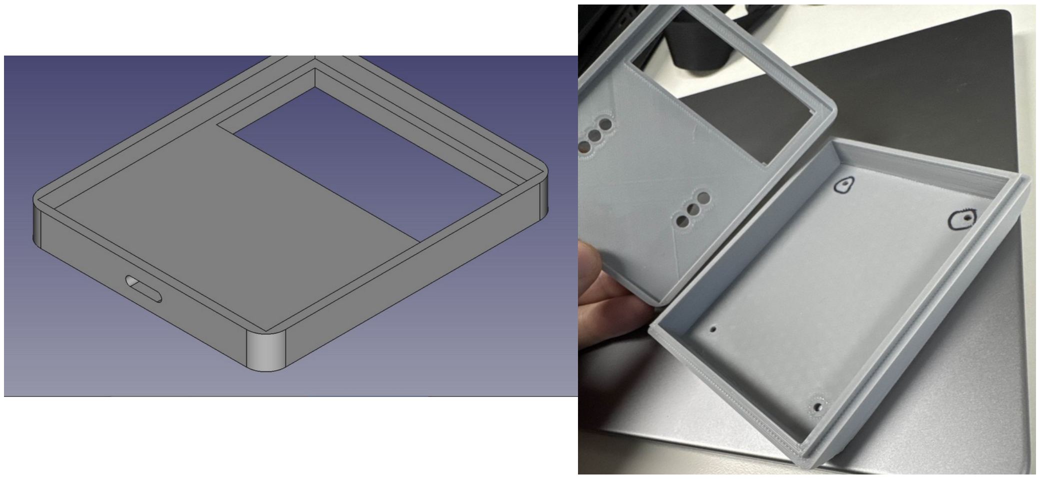
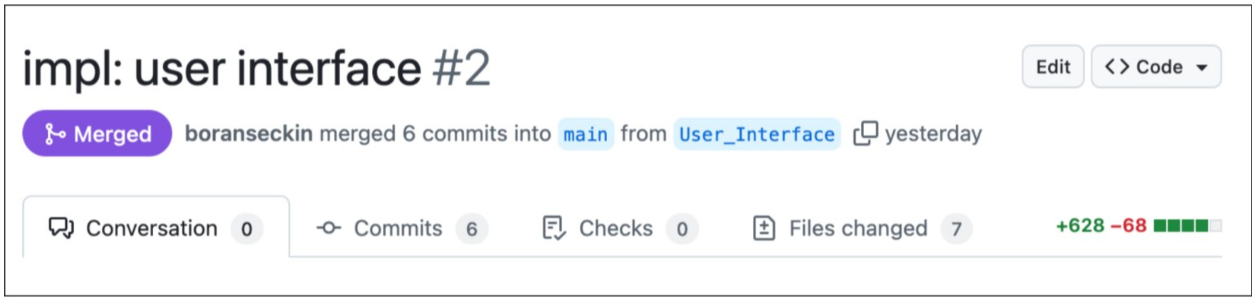


Figure 78: First 3D Printed Case Concept

Navigating screens with test data (February 27, 2025)

UI and Network code integration (March 18th, 2025):


impl: user interface #2

Merged boranseckin merged 6 commits into main from User_Interface yesterday

Figure 79: Merging of UI and Network Code on GitHub

Challenges

There were a few challenges and learning opportunities throughout the project. There was a bit of a learning curve given the nature of the E-Ink. Drawing pixel by pixel and ensuring drawings don't cover different drawings and overlap. Also, the concept of drawing and writing to the buffer but then also refreshing to see the drawings before clearing the image buffer.

Another challenge came with integrating the user interface code with the networking code. The merge itself went relatively smoothly however upon testing nothing seemed to work, and the Pico 2 was getting stuck. After some time digging, we discovered it wasn't an issue with how we integrated the code together but rather it was to do with the SDK version we were using. It turns out there was a new SDK version that got pushed, and this version had re-written the timing section that we use to trigger alarms to sleep the display, the new version removed a section that used busy waits which we needed for our code to work. After adding it back in, everything was nominal once again.

There was also the new and unique challenge of working with an E-Ink display, something that I have never done before. The thinking for the user interface is a lot different with these displays given its less dynamic and monochrome aspects. It was difficult to make each page feel different and provide information in meaningful ways. Specifically, I spent a lot of time figuring out how to get multiple messages/neighbours to display on a page and then have the next set display on the next page all while keeping track of where the cursor is and then clearing the old messages/neighbours to make way for drawing in the next set.

Future Work

One of the aspects we wanted from the beginning was location tracking, we had early initial concepts depicting what it might look like.

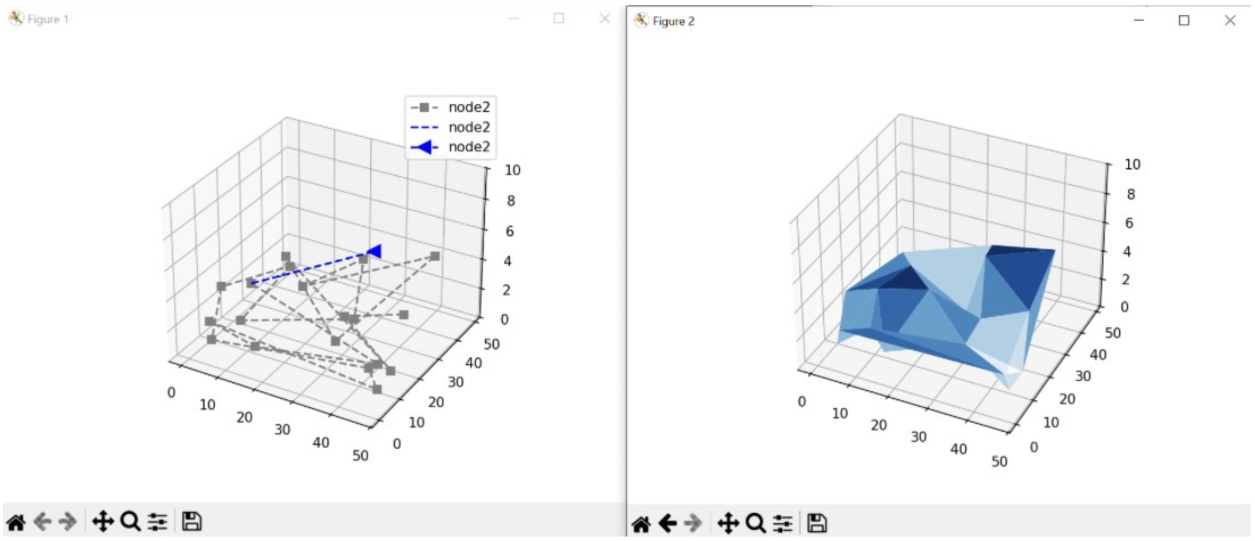


Figure 80: Distance/mesh network concept visual

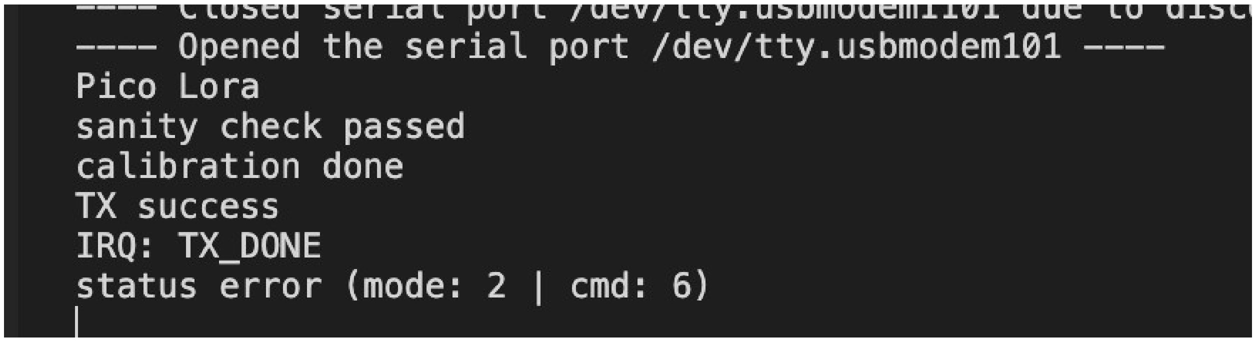
This is something we weren't fully able to implement apart from ping times and RSSI values. Something for a future version would be to take these values and be able to visualize the current mesh network and locations of other neighbours via a visual. Or even have the neighbour connected to a laptop to do some stronger processing to show possible weak signal zones through a heat map.

We also tossed around the idea of adding speakers and microphones to process audio or even being able to process other types of data as well, and what that might look like to the user. This is something I would have liked to explore.

Module II (Network): Boran Seckin, 100% Complete

The network module was built layer by layer and tested thoroughly at each step.

Layer 0: Initial communication with the transceiver is established, and the transceiver was configured correctly to transmit arbitrary data. (November 18, 2024)

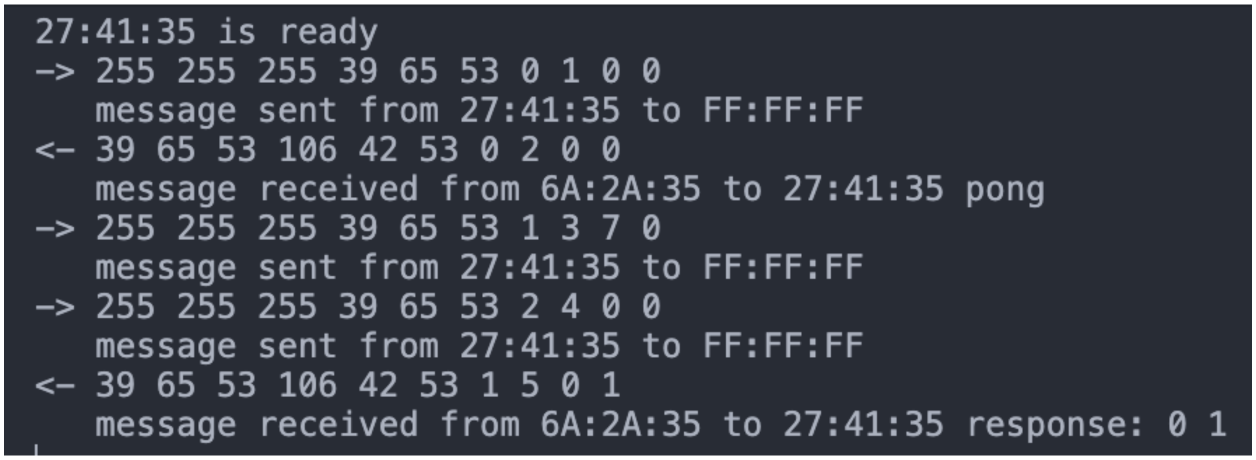


```
---- Closed serial port /dev/tty.usbmodem101 due to disc
---- Opened the serial port /dev/tty.usbmodem101 ----
Pico Lora
sanity check passed
calibration done
TX success
IRQ: TX_DONE
status error (mode: 2 | cmd: 6)
```

Figure 81: Console Output from Initial Communication

Layer 0 continuation: First message exchange between two nodes was performed. (January 14, 2025)

Layer 1: Initial testing of the protocol, where each node encodes the source and destination identifiers as well as the message type and its corresponding data was conducted. (January 27, 2025)

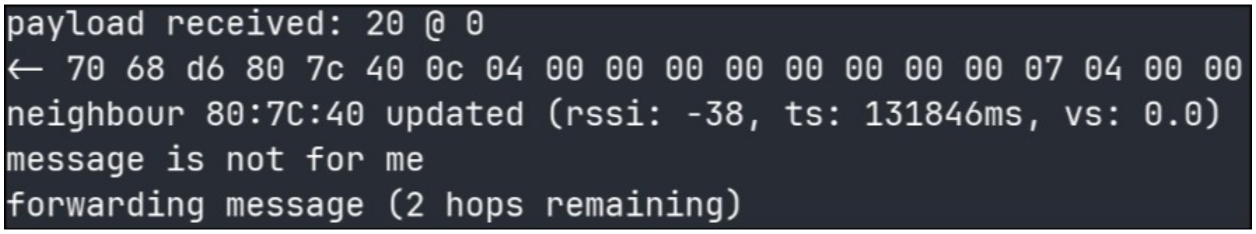


```
27:41:35 is ready
-> 255 255 255 39 65 53 0 1 0 0
   message sent from 27:41:35 to FF:FF:FF
<- 39 65 53 106 42 53 0 2 0 0
   message received from 6A:2A:35 to 27:41:35 pong
-> 255 255 255 39 65 53 1 3 7 0
   message sent from 27:41:35 to FF:FF:FF
-> 255 255 255 39 65 53 2 4 0 0
   message sent from 27:41:35 to FF:FF:FF
<- 39 65 53 106 42 53 1 5 0 1
   message received from 6A:2A:35 to 27:41:35 response: 0 1
```

Figure 82: Console Output from Protocol Testing

Layer 2: Video of ping transmission and pong reception from the debug console. (February 20, 2025)

Layer 3: Testing of sending multi-hop messages to an intermediate node, which proceeds to forward it. (March 25, 2025)



```
payload received: 20 @ 0
← 70 68 d6 80 7c 40 0c 04 00 00 00 00 00 00 00 00 07 04 00 00
neighbour 80:7C:40 updated (rssi: -38, ts: 131846ms, vs: 0.0)
message is not for me
forwarding message (2 hops remaining)
```

Figure 83: Console Output from Multi-Hop Testing

During this project, I gained valuable experience in embedded development. Working with a microcontroller pushed me to optimize the use of limited resources. Additionally, I engaged in an iterative process to engineer a network protocol, which helped me grasp the complexities of encoding data effectively. The final version of the protocol underwent 11 revisions before I felt it was ready for completion.

Challenges

While developing the code, it became apparent that transmitting and receiving packages must be done asynchronously to handle networks with heavier traffic. To achieve this, we implemented a queue system for both input and output and served each queue over time.

One of the challenges of creating the network was synchronizing nodes to a reference time frame. Without Internet or GPS access, it is not possible to use the same time reference for all the nodes. Moreover, due to the asynchronous nature of our code, each packet experiences an uncertain amount of time waiting to be served. For this reason, we developed a method to share time between two nodes using only relative time data. By recording time spent processing packets at each step, nodes can accurately calculate the propagation time of a packet. This duration is crucial in measuring the distance between two nodes. The process is outlined in Figure 84: Two-Way Timing Synchronization.

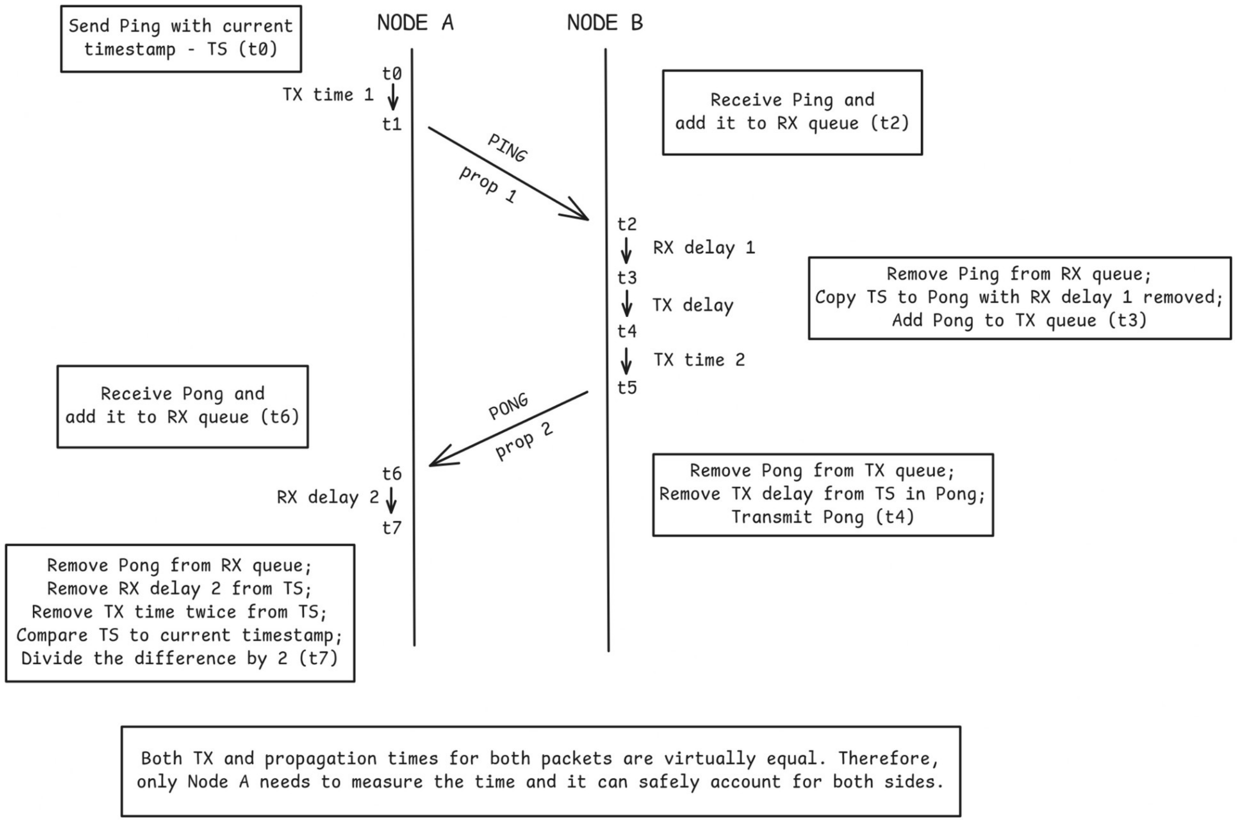

Send Ping with current timestamp - TS (t0)
NODE A
NODE B
t0
TX time 1
t1
PING
prop 1
Receive Ping and add it to RX queue (t2)
t2
RX delay 1
t3
TX delay
t4
TX time 2
t5
Remove Ping from RX queue; Copy TS to Pong with RX delay 1 removed; Add Pong to TX queue (t3)
Receive Pong and add it to RX queue (t6)
PONG
prop 2
t6
RX delay 2
t7
Remove Pong from TX queue; Remove TX delay from TS in Pong; Transmit Pong (t4)
Remove Pong from RX queue; Remove RX delay 2 from TS; Remove TX time twice from TS; Compare TS to current timestamp; Divide the difference by 2 (t7)
Both TX and propagation times for both packets are virtually equal. Therefore, only Node A needs to measure the time and it can safely account for both sides.

Figure 84: Two-Way Timing Synchronization

Future Work

The protocol was created with future expansion in mind. It includes reserved fields in the packet for potential future use, and its flexible design permits arbitrary enhancements to the messages.

At present, the network module has successfully implemented all the planned features of the project. However, some potential quality-of-life enhancements for this module include an improved flooding algorithm to minimize unnecessary traffic, and an encryption layer to enable secure data transmission.

Hardware: Stefan Praniauskas, Milena Thibault, 100% Complete

The final VoidLink hardware design was a major success, resulting in a compact, fully integrated 4-layer PCB that performed exactly as intended. Key features include the integrated SX1262 LoRa transceiver, impedance-matched RF traces, MCU integration, power switching circuitry, USB charging with battery management, onboard LEDs for status indication, support for an E-Ink display, and six user-programmable buttons. The layout also included decoupling capacitors, EMI-conscious grounding, and mechanical integration for the custom 3D-printed case.

Module III (Hardware A): Stefan Praniauskas, 100% Complete

Results and Testing

The full schematic and board layout were completed across two design revisions, both of which were fully functional. The first revision included the initial implementation of the MCU and power circuitry, as well as user I/O (display output, pushbuttons, USB 2.0 data lines, and power + status LEDs). The second revision was especially successful, integrating all components (mainly the transceiver) in a compact manner, and resolving issues found in the first revision. The final PCB fit perfectly into the custom case, with precise alignment for the mounting holes, buttons, display, LEDs, and antenna. All hardware features operated as intended, with no signs of overheating, shorts, or electrical crosstalk during extended testing. Power delivery was stable, signal integrity was maintained, and the board performed reliably under all tested conditions. A fully functional demo of the revision 1 PCB can be seen in this video, when the maximum range test was performed. A minor issue involving occasional resets during USB-to-battery switching was traced to non-ideal decoupling capacitor placement, but it had no impact on overall functionality.

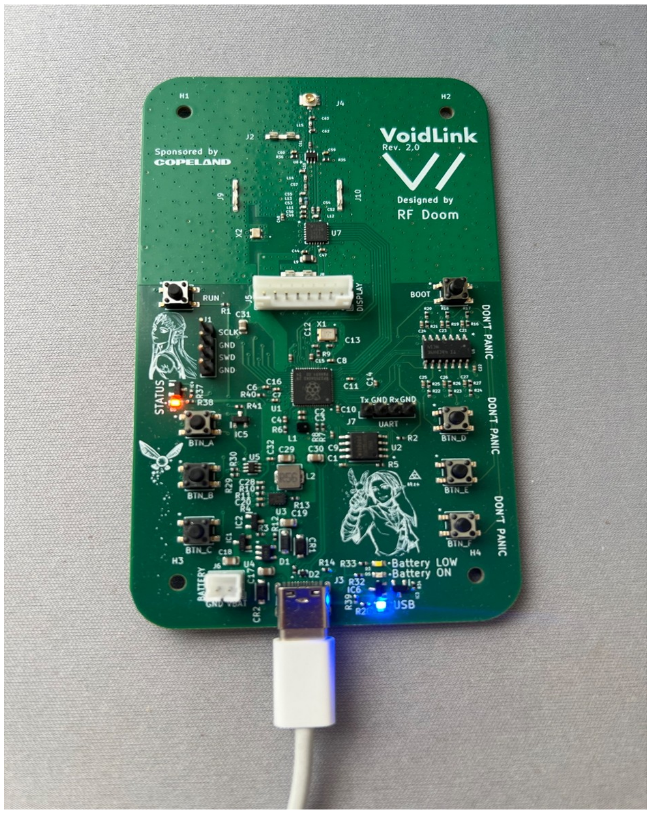
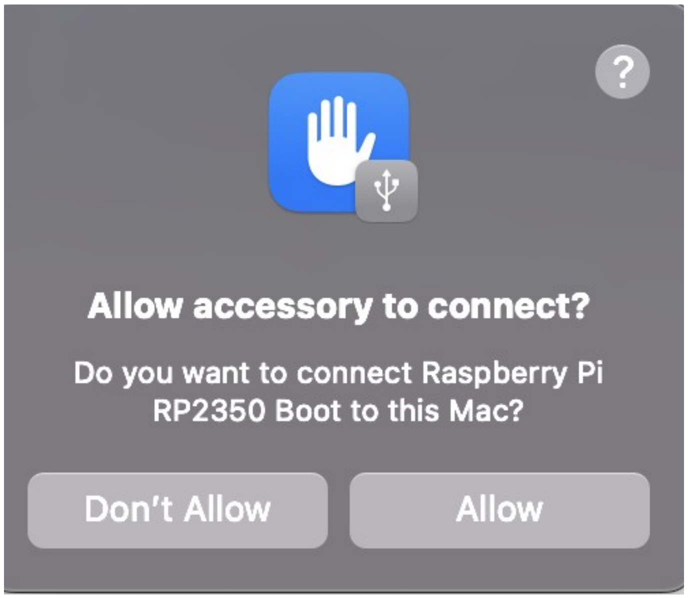


Figure 85: Revision 2 PCB Powered ON Through USB, USB LED is ON



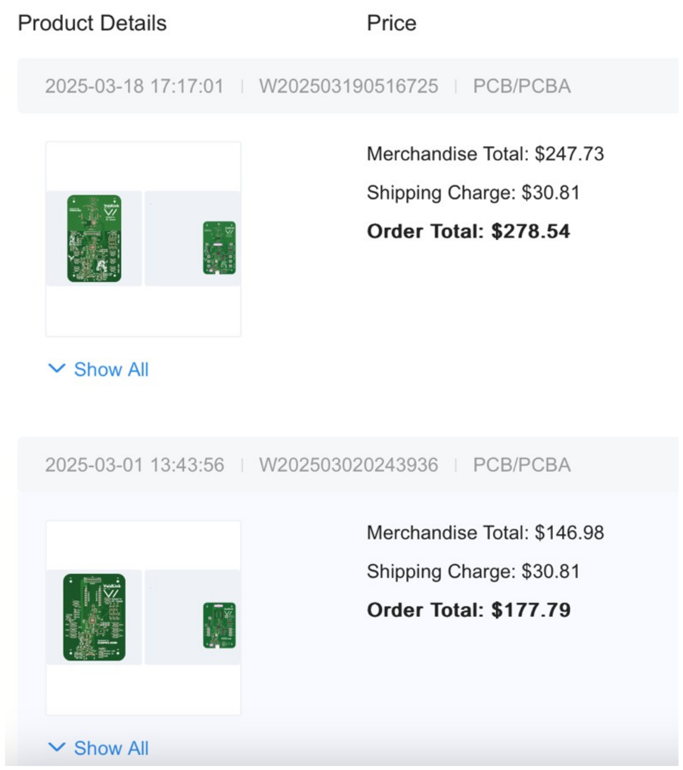
Allow accessory to connect?

Do you want to connect Raspberry Pi RP2350 Boot to this Mac?

Figure 86: Revision 2 PCB Detected on Computer

Work Contributions and Timeline

Starting in October through December, once our project goal had been clearly formulated, I began researching how we would implement our first hardware revision. This included significant research regarding the initial component selection of the MCU, and display, hardware scope definition, ECAD tool selection, and PCB manufacturer selection (roughly 56 hours). In late December, I began implementing the schematic for the first revision based on my learnings (24 hours). Then in January, I began implementing the layout for the first revision (48 hours). In early March, we taped out the first revision of the PCB through our manufacturer, JLCPCB (12 hours). The first revision met our functionality needs, resulting in 100% completion of our initial goals. After helping with the hardware validation, development for the second revision began. From March to April, I acted mainly as a consultant/assistant for the transceiver implementation (24 hours). This was done mainly by performing research on how to implement RF design and performing layout reviews on our designs. In late March, I taped out the second revision and once again (12 hours) helped with hardware validation. The second revision was also successful, as all the noted problems of the first revision were solved and the transceiver was successfully integrated, resulting in 100% completion.


Product Details | Price

2025-03-18 17:17:01 | W202503190516725 | PCB/PCBA

Merchandise Total: $247.73
Shipping Charge: $30.81
Order Total: $278.54

2025-03-01 13:43:56 | W202503020243936 | PCB/PCBA

Merchandise Total: $146.98
Shipping Charge: $30.81
Order Total: $177.79

Figure 87: Revisions 1 & 2 Tapeout Dates (Note: Prices in USD)

Challenges

The final success of the hardware module was not without its challenges along the way. The most crucial challenge was first finding ICs that matched our functional requirements and had sufficient implementation documentation. This quickly ruled out the possibility of using a microprocessor, as these are generally unavailable to DIY projects due to the lack of application documentation and unavailability of small quantity orders. Fortunately, the Raspberry PI foundation had recently released their RP2350 microcontroller, which included clear implementation documentation with DIY projects in mind, making this powerful and versatile IC

very accessible. The same was true for the transceiver. Once again, the SX1262 from Semtech was selected due to its impressive functionality as well as sufficient documentation (although certainly more advanced and more industry focused). With these big-ticket items out of the way, the project became possible and all that remained was its actual implementation.
One other critical challenge worth mentioning was the final design verification during tapeout. After meticulously verifying the PCB schematic and layout for revision 2, the design files were submitted to JLCPCB, where then, one of their engineers modified the orientation of the transceiver crystal (this was due to a confusing footprint that was provided for this crystal). Luckily, their error was caught through a final careful visual inspection before confirming the production design files and was later resolved. This last-minute error would have rendered the transceiver, and by extension the whole PCB, unusable. This highlights the responsibility of an engineer to first understand, then diligently review their designs to ensure correctness.

Lessons Learned

Becoming familiarized with the full hardware design cycle was highly beneficial. Going through the stages of initial product conception, design target outlining, component selection, design tool selection, schematic design, layout, verification, product tapeout, PCB validation, and product iteration, allowed me to gain the skills necessary to take an idea and turn it into a fully functional physical product. More specifically, I gained the new technical skill of PCB design including schematic formulation, and layout. I was also exposed to working with a professional third party through our PCB manufacturer, JLCPCB, with whom I troubleshooted various technical issues during tapeout and across a 12hr time difference (the JLCPCB office is located in Hong Kong). Moreover, I gained experience in PCB validation, especially through debugging the unexpected behaviour of the undervoltage IC (TLV3012), in revision 1. Using voltage sources and an oscilloscope, we took measurements that we were able to interpret using our understanding of the datasheet to understand the issue and successfully fix it in revision 2. This iterative design flow is crucial to become comfortable with, as this is how hardware design functions in industry.

Future Work

Although all the goals of both PCB revisions were met, there are some future items that could be completed given more time. It would be beneficial for our learning to implement the same PCB design fully in the Cadence OrCad X design environment. Although this would not affect the functionality of the final product (as KiCad contained all the required tools to create a fully functional PCB), it would be a good learning exercise, and it would improve our ECAD tool knowledge. One nice feature that may be considered for a future third revision could be to replace the pushbuttons with button pads, making the PCB more professional and improving the user experience.

Module IV (Hardware B): Milena Thibault, 100% Complete

Results and Testing

The power system was designed and tested to support seamless operation from both USB and battery sources. We ran the device through all power scenarios to confirm this, and it performed reliably in nearly every case. However, when USB was disconnected, a brief power dip caused the MCU to reset. While simulations predicted smooth transitions, real-world testing revealed a brief power dip when USB was disconnected, causing the MCU to reset. This minor issue, caused by non-ideal decoupling capacitor placement in layout, will be addressed in future revisions but did not impact overall performance.

On the RF side, integrating the transceiver directly onto the PCB was a major success. The system consistently transmitted and received messages between nodes, confirming that the RF circuit, shielding, impedance matching, and layout were fully functional. In **this video**, message transmission between nodes was clearly visible, and waveforms were successfully detected using an external antenna setup. We also performed range testing—while the overall range was about the same as before, signal strength (RSSI) appeared slightly better and more stable, though we don’t have measured data to confirm this.

Replacing the SX1262’s CMOS oscillator with a passive crystal and reducing status LED currents significantly improved power efficiency, increasing battery life by approximately 65%. However, signal strength remained as inconsistent as it had been with the earlier breakout board setup, which limited the functionality of planned distance and heading features.

Not only did our transceiver integration work as intended—it also improved overall system performance by extending battery life and potentially enhancing RSSI stability.

Contributions and Timeline

From January to February, I spent over 35 hours running power simulations in PSpice, focusing on validating the system’s power architecture under various conditions. In March, I contributed approximately 50 hours toward laying out the second PCB revision, and an additional 15 hours implementing fixes based on issues discovered in revision 1. I was also involved in the early transceiver bring-up using the initial breakout board, helping to validate basic communication and guide integration into the custom design. All this work—spanning simulation, debugging, layout, and validation—was completed in full (100% completion) and contributed directly to delivering a functional and reliable hardware system.

Challenges

Developing the hardware came with challenges: simulating the power circuitry in PSpice was particularly demanding, one 30ms transient simulation took over six hours due to the complexity of the models of the LDO and UVLO ICs, as well as the MOSFET models. I also had to learn how to use PSpice from scratch, including how to import and configure complex component models, which added to the difficulty. RF layout presented an even steeper learning curve, especially since I had no prior experience with RF design and only a basic background in general PCB layout. RF design requires careful grounding, shielding, and impedance matching, and is much more involved than non-RF PCB layout. Another unexpected challenge was dealing with supply chain constraints, which required several last-minute component substitutions. This emphasized the importance of choosing parts with stable availability and ample stock.

Lessons Learned

Through this project, I gained hands-on experience in end-to-end hardware development, spanning schematic design, PCB layout, simulation, assembly, and testing. Working with simulation tools such as PSpice deepened my understanding of power system behavior and transient analysis, while RF schematic design and layout introduced new concepts such as grounding, shielding, and impedance matching. I also learned to troubleshoot real-world issues, including probing transceiver pins to diagnose sleep mode lockups and identifying unexpected resets caused by power dips. Additionally, navigating supply chain challenges highlighted the importance of selecting components with stable availability. Altogether, this experience provided a strong foundation in hardware design and reinforced the value of planning, iteration, and adaptability in engineering.

Future Work

The project was completed to 100%. However, for future work, the system could benefit from a more advanced transceiver with improved RSSI stability for more reliable distance estimation. Additionally, integrating a dual-antenna setup could enable orientation detection between nodes without requiring the user to manually rotate the device.

Overall, for the team's first custom boards, we’re very happy with how well the system came together. There were plenty of challenges and a lot of learning throughout the process, so seeing a fully functional, integrated system in the end was very rewarding.

Mac Video Link

Conclusion/Summary

VoidLink successfully delivers a low-cost, off-grid mesh communication system. All core modules, hardware, networking, and UI were fully implemented and worked reliably. The hardware operated without faults, RF communication was functional, the network performed well under load and the UI provided an intuitive user experience. Key challenges like power dips, RF layout, and simulation complexity were overcome, and valuable lessons were learned in hardware design, troubleshooting, and supply-aware development. While some advanced features like distance plotting and encryption were not completed, the system met all core goals and is a solid foundation for future development.

References

[1] JLCPCB, "JLCPCB Impedance Calculator," [Online]. Available: https://jlcpcb.com/pcb-impedance-calculator/. [Accessed 2025].

[2] Raspberry Pi Ltd., "Hardware Design with RP2350," February 2025. [Online]. Available: https://datasheets.raspberrypi.com/rp2350/hardware-design-with-rp2350.pdf. [Accessed 2025].

[3] Semtech Corporation, "SX1262 Reference Desig," January 2019. [Online]. Available: https://cdn-reichelt.de/documents/datenblatt/A200/SX1262REFERENCE.pdf. [Accessed 2025].

[4] MicroType, "Lithium-Ion Battery Charger Circuit with Load Sharing," 12 December 2022. [Online]. Available: https://www.microtype.io/lithium-ion-battery-charger-circuit-load-sharing/. [Accessed 2025].

[5] Texas Instruments, "TPS62811-Q1 Step-Down Converter Datasheet," October 2022. [Online]. Available: https://www.ti.com/lit/ds/symlink/tps62811-q1.pdf. [Accessed 2025].

[6] Microchip Technology Inc., "MCP73831/2 Li-Ion/Li-Polymer Charge Management Controller Datasheet," 2007. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/DataSheets/22036b.pdf. [Accessed 2025].

[7] Texas Instruments, "TLV3012B Nanopower Voltage Comparator With Reference Datasheet," June 2022. [Online]. Available: https://www.ti.com/lit/ds/symlink/tlv3012b.pdf. [Accessed 2025].

[8] Raspberry Pi Ltd., "RP2350 Datasheet," February 2025. [Online]. Available: https://datasheets.raspberrypi.com/rp2350/rp2350-datasheet.pdf. [Accessed 2025].

[9] Texas Instruments, "SNx4HC14 Hex Inverters with Schmitt-Trigger Inputs," December 1982. [Online]. Available: https://www.ti.com/lit/ds/symlink/sn74hc14.pdf?ts=1745176067553. [Accessed 2025].

[10] J. Warner, "PCB Stack-up Design Best Practices with Rick Hartley," Altium, 24 July 2018. [Online]. Available: https://resources.altium.com/p/pc-board-stack-up-best-practices-with-rick-hartley. [Accessed 2025].

[11] Semtech Corporation, "SX1261/2 Datasheet," December 2024. [Online]. Available: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/RQ000008nKCH/hp2iKwMDKWl34g1D3LBf_zC7TGBRIo2ff5LMnS8r19s. [Accessed 2025].

[12] "What is the OSI Model?," Cloudflare, [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/.

[13] "Mesh Broadcast Algorithm," Meshtastic, [Online]. Available: https://meshtastic.org/docs/overview/mesh-algo/.

[14] Waveshare, "2.13inch_e-Paper_V4_Specification," 2025. [Online]. Available: https://files.waveshare.com/upload/4/4e/2.13inch_e-Paper_V4_Specification.pdf. [Accessed 2025].

[15] use IP, "lux light level chart," 2025. [Online]. Available: https://www.use-ip.co.uk/datasheets/lux_light_level_chart.pdf. [Accessed 2025].

[16] Waveshare, "Pico_ePaper_Code," 2025. [Online]. Available: https://github.com/wavesharetеam/Pico_ePaper_Code. [Accessed 2025].

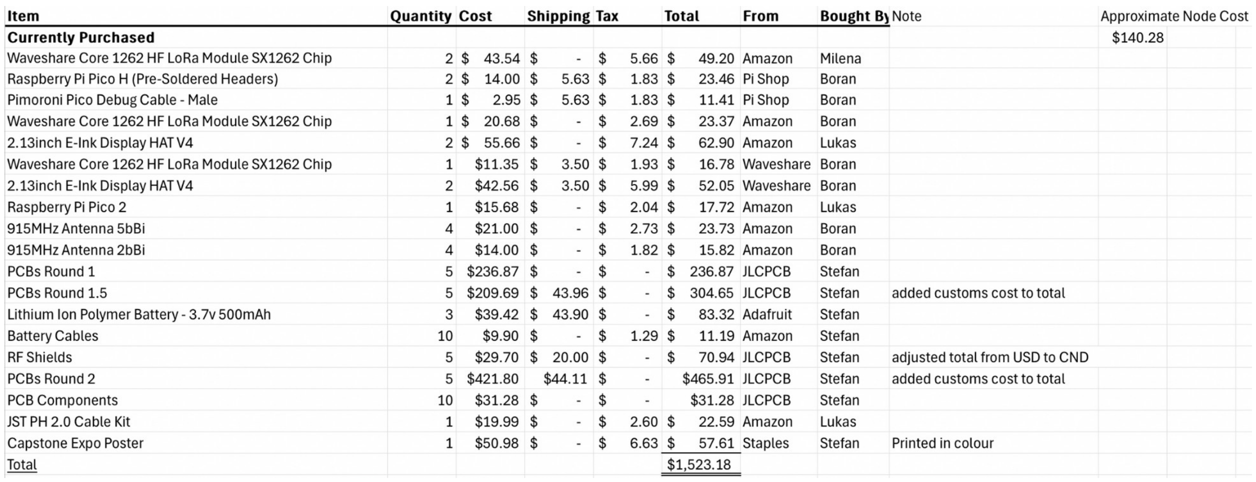
[17] E-Ink, "How it works," E-Ink, April 2025. [Online]. Available: https://www.eink.com/tech/detail/How_it_works. [Accessed 2025].

[18] Waveshare, "2.13inch e-Paper HAT Manual," Waveshare, April 2025. [Online]. Available: https://www.waveshare.com/wiki/2.13inch_e-Paper_HAT_Manual#accordion29. [Accessed 2025].

[19] Semtech, "sx126x_driver," 2025. [Online]. Available: https://github.com/Lora-net/sx126x_driver/. [Accessed 2025].

Appendix

Bill of Materials



Item	Quantity	Cost	Shipping	Tax	Total	From	Bought By	Note	Approximate Node Cost
Currently Purchased									$140.28
Waveshare Core 1262 HF LoRa Module SX1262 Chip	2	$ 43.54	$ -	$ 5.66	$ 49.20	Amazon	Milena		
Raspberry Pi Pico H (Pre-Soldered Headers)	2	$ 14.00	$ 5.63	$ 1.83	$ 23.46	Pi Shop	Boran		
Pimoroni Pico Debug Cable - Male	1	$ 2.95	$ 5.63	$ 1.83	$ 11.41	Pi Shop	Boran		
Waveshare Core 1262 HF LoRa Module SX1262 Chip	1	$ 20.68	$ -	$ 2.69	$ 23.37	Amazon	Boran		
2.13inch E-Ink Display HAT V4	2	$ 55.66	$ -	$ 7.24	$ 62.90	Amazon	Lukas		
Waveshare Core 1262 HF LoRa Module SX1262 Chip	1	$11.35	$ 3.50	$ 1.93	$ 16.78	Waveshare	Boran		
2.13inch E-Ink Display HAT V4	2	$42.56	$ 3.50	$ 5.99	$ 52.05	Waveshare	Boran		
Raspberry Pi Pico 2	1	$15.68	$ -	$ 2.04	$ 17.72	Amazon	Lukas		
915MHz Antenna 5bBi	4	$21.00	$ -	$ 2.73	$ 23.73	Amazon	Boran		
915MHz Antenna 2bBi	4	$14.00	$ -	$ 1.82	$ 15.82	Amazon	Boran		
PCBs Round 1	5	$236.87	$ -	$ -	$ 236.87	JLCPCB	Stefan		
PCBs Round 1.5	5	$209.69	$ 43.96	$ -	$ 304.65	JLCPCB	Stefan	added customs cost to total	
Lithium Ion Polymer Battery - 3.7v 500mAh	3	$39.42	$ 43.90	$ -	$ 83.32	Adafruit	Stefan		
Battery Cables	10	$9.90	$ -	$ 1.29	$ 11.19	Amazon	Stefan		
RF Shields	5	$29.70	$ 20.00	$ -	$ 70.94	JLCPCB	Stefan	adjusted total from USD to CND	
PCBs Round 2	5	$421.80	$44.11	$ -	$465.91	JLCPCB	Stefan	added customs cost to total	
PCB Components	10	$31.28	$ -	$ -	$31.28	JLCPCB	Stefan		
JST PH 2.0 Cable Kit	1	$19.99	$ -	$ 2.60	$ 22.59	Amazon	Lukas		
Capstone Expo Poster	1	$50.98	$ -	$ 6.63	$ 57.61	Staples	Stefan	Printed in colour	
Total					$1,523.18				

Figure 88: Bill of Materials