ECE 445
SENIOR DESIGN LABORATORY
Design Document

Any-Screen to Touch-Screen Device

Team No. 1
Sakhiyuvio Farsya Yunalfian
(sfy2@illinois.edu)
Muthu Ganesh Arunachalam
(muthuga2@illinois.edu)

TA: Chi Zhang
Professor: Arne Fliflet
October 3, 2024

Table of Contents

1 Introduction

1.1 Problem

As touchscreens become an important method of user input, the demand for touch-enabled devices across industries has surged. However, retrofitting existing displays of any size, especially large ones, with touch capability is prohibitively expensive and technically challenging. Organizations like schools, businesses, and research departments use non-touch displays that lack the interactive functionality needed for modern applications, like collaborative learning, design work, or real-time marking up of digital documents. Upgrading these screens to support touch functionality would be a large cost and would require specialized hardware.

Although there are devices on the market that can overlay touch functionality on a non-touch screen, there are several drawbacks. Some of these devices only accept generic free-floating hand motions as input (instead of taps, clicks, and dragging), limiting user interaction with the screen. Other devices allow for direct interaction with the screen, but these devices can be inaccurate, resulting in user frustration. Several products are accurate and allow direct interaction with the screen, but use technologies like camera imaging which can cause faulty sensing if your hand is in the way; such sensing technologies can also limit the screen size on which the product can be used. Given the growing need for interactive interfaces and the lack of versatile products on the market, developing a cost-effective, scalable solution to this problem is crucial.

1.2 Solution

The proposed solution is a device that can convert any standard screen into a touchscreen using ultra-wideband (UWB) sensors to track a specially designed pen. This system relies on a network of UWB sensors placed at the corners of the display, which track the position of the pen in real time based on Time of Flight (ToF). The pen, equipped with a UWB tag and motion sensors (gyroscope), communicates its position to the sensors, allowing the device to detect movement across the screen. The pen communicates all sensor and location data via Bluetooth HID to the host device. This approach negates the need for expensive capacitive or resistive touch overlays, instead providing a wireless and modular solution that can be applied to any display size or type. By incorporating additional features such as buttons, the pen can offer more than basic touch functionality, enabling features like clicks, dragging, scrolling, and hotkeys. This solution not only provides an affordable way to retrofit non-touch displays but also expands the possibilities of user interaction in diverse settings and across multiple operating systems.

1.3 Visual Aid

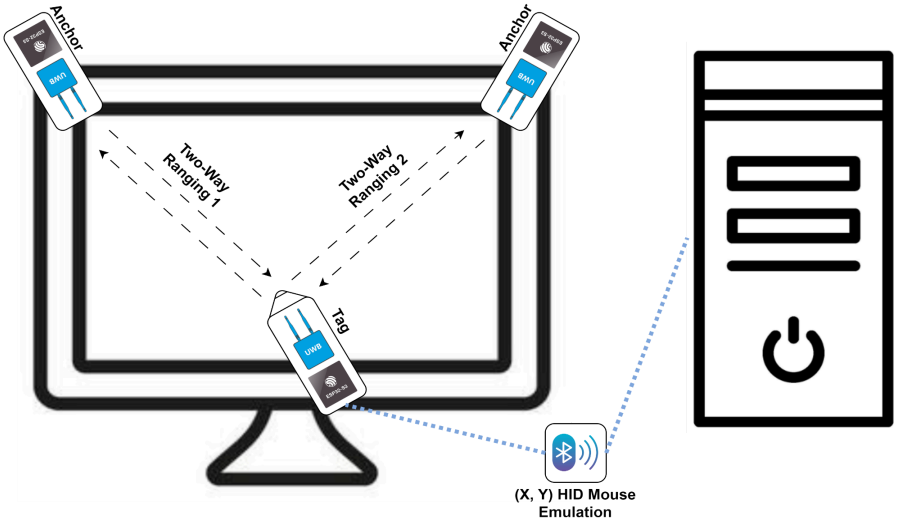

Figure 1: The Any-Screen to Touch-Screen System

1.4 High-level Requirements

1.4.1 Accuracy and Responsiveness

Our project's most crucial success indicator is touch emulation accuracy and responsiveness. Utilizing the Bluetooth module embedded in the ESP32-S3 microcontroller, we undoubtedly expect numerous sources of delay like propagation delay, lags, and queuing. We aim to quantitatively access this through continuous data logging through Python to monitor the time interval of **data transfe**r and keep it **under 250 milliseconds**. Aside from system responsiveness, another crucial requirement is for our location-tracking mechanism to be accurate. We aim to potentially reach a **5% margin of error regarding the Euclidean distance of where the pen is touching the screen** and what (x, y) coordinates are digitized and processed. We need to consider factors that may intensify the error, like sensor data fusion with the gyroscope to account for pen tilt, acceleration, and sensitivity.

1.4.2 System Integration and Compatibility

The next crucial high-level requirement is that our system be compatible with straightforward integration. Numerous compatibility aspects will be considered: **screen size and operating system**. Quantitatively, we will be **testing our system from small screens (5 inches) to large screens (20+ inches)**. We want to **keep our error rate consistent at 5% or less for any type of screen**. Lastly, we want to test our system integration and compatibility with **different operating**

systems like Windows, macOS, or Linux. The OS compatibility will be quantitatively measured through **success rate**, **aimed at 95%**, by **comparing location precision error rates** between different OSs.

1.4.3 Extended Functionality

The last high-level requirement is that the device must reliably support discrete touch-based actions such as **tapping, clicking, scrolling, and dragging**. Each input type should be detected and executed correctly at least **99%** of the time, with a clear distinction between tapping (single-touch actions) and dragging (continuous motion) gestures. Scrolling actions must be executed smoothly, with a minimal jitter rate of less than **2%** when moving across the screen. The system must ensure that scrolling and dragging actions are continuous, with no unintentional interruptions.

2 Design

2.1 Block Diagram

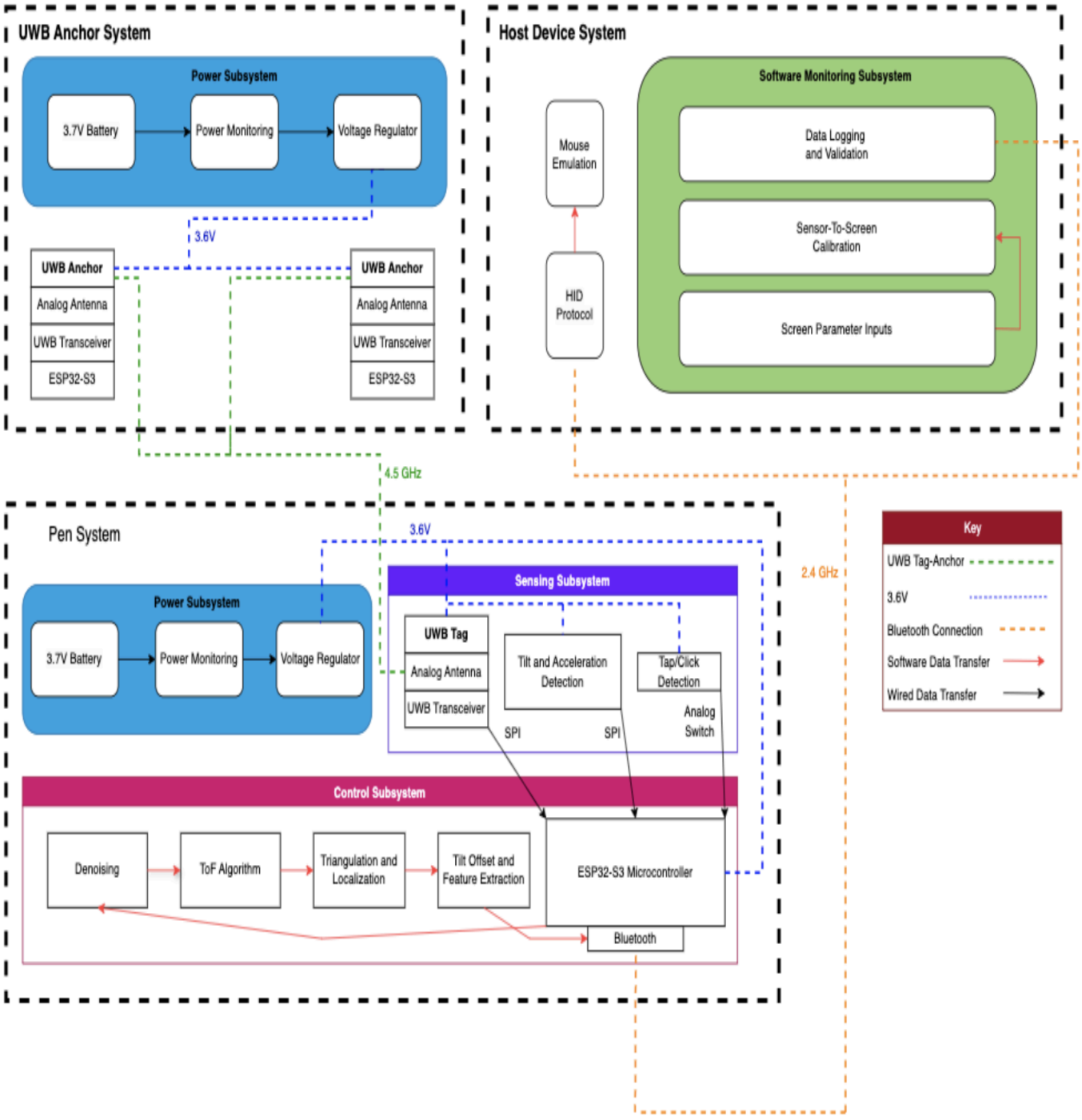

Figure 2: The Any-Screen to Touch-Screen Device Block Diagram

2.2 Physical Design

For our pen enclosure, we decided to do a 3D-printed DIY pen enclosure following a simple mechanical structure shown in Figure 3. We chose a plastic enclosure to avoid signal disruptions on the UWB communication if metallic enclosures were used instead. Instead of the buttons present on the body of the enclosure, we will replace it with mounting holes for crucial LEDs, switches, and buttons to be accessible from external environments for the purpose of debugging and system initialization.

Figure 3: Pen Plastic Enclosure

2.3 Power Subsystem

2.3.1 Hardware Design Overview

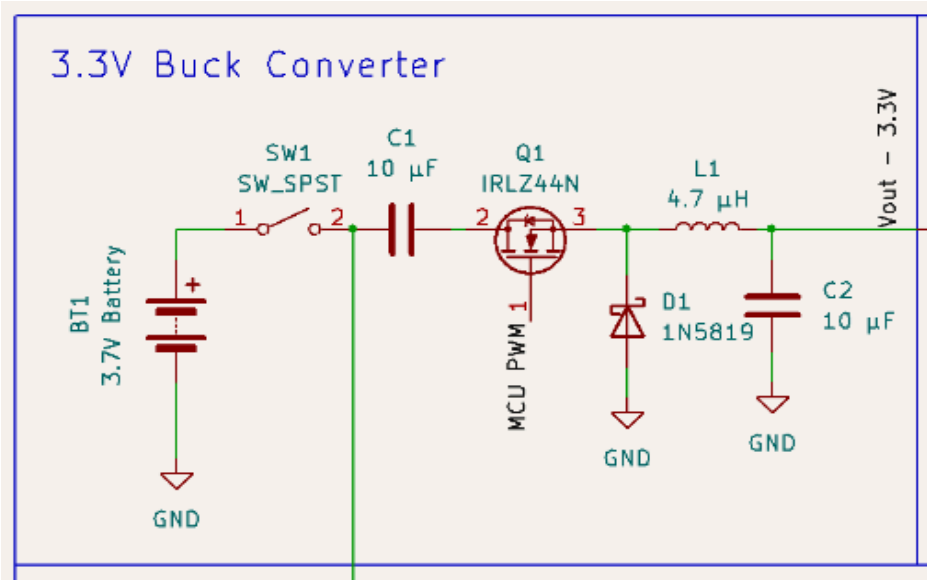

Figure 4: Battery to 3.3V Schematic Design

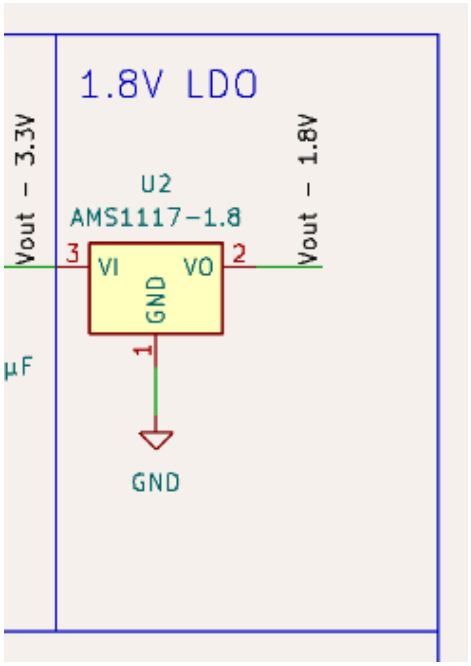

Figure 5: 3.3V to 1.8V Conversion Power Schematic Design

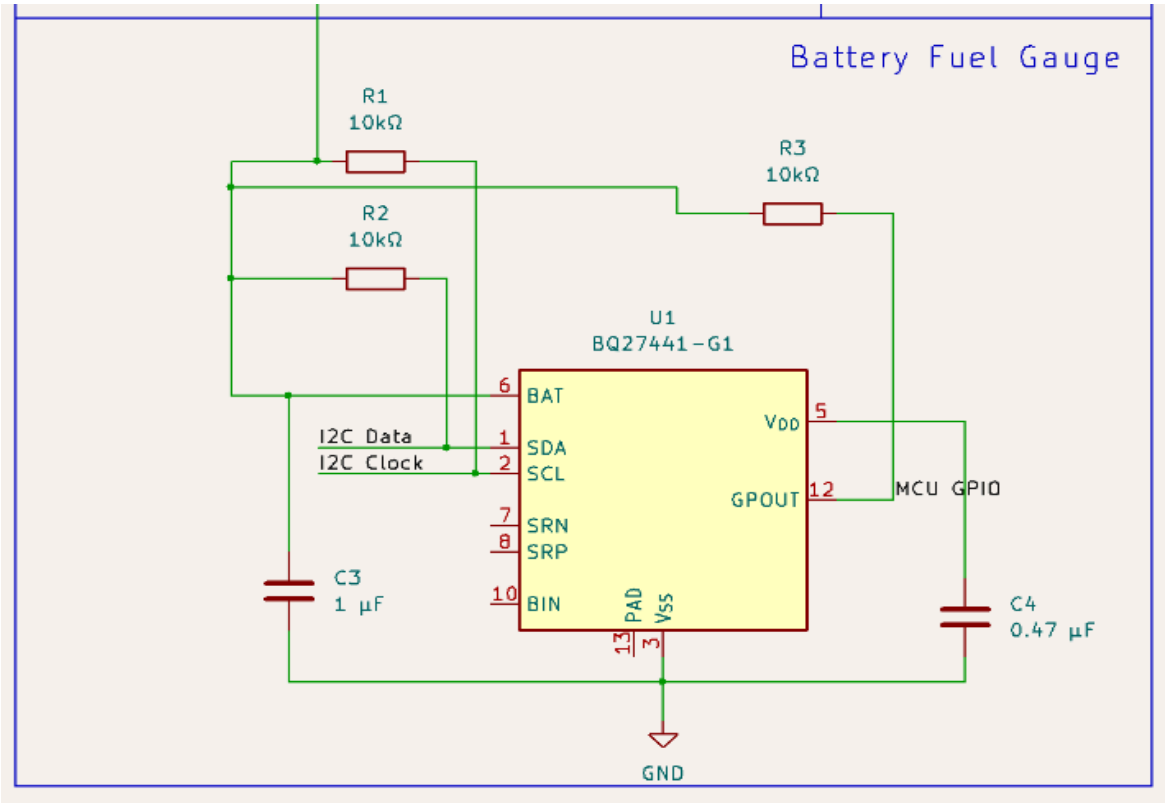

Figure 6: Fuel Gauge Schematic Design

The power subsystem utilizes a 3.7V battery, which is stepped down to 3.3V using a buck converter and then further regulated to 1.8V using a low dropout regulator. A battery fuel gauge monitors the battery's state of charge, voltage, current, and temperature, ensuring the device operates efficiently and alerts the user of battery conditions. The power subsystem is designed to interact seamlessly with other subsystems by supplying stable voltage and monitoring critical power metrics.

2.3.2 Functionality & Contribution

The primary function of the power subsystem is to convert the voltage from the 18650 Li-ion battery (nominally 3.7V) to two stable supply voltages: 3.3V and 1.8V. The subsystem achieves this through the following key components:

- 3.7V to 3.3V Buck Converter: This component efficiently steps down the battery voltage to 3.3V, which powers the main microcontroller (ESP32-S3-WROOM-1) and the UWB module (DWM1000).
- 3.3V to 1.8V Low Dropout Regulator (LDO) [1]: This converts the 3.3V to 1.8V to power low-voltage components, such as the LSM6DSL IMU.
- BQ27441-G1 Battery Fuel Gauge [11]: The fuel gauge monitors the battery's state, providing detailed information such as the remaining charge, battery voltage, and current. It communicates with the microcontroller via the I²C interface, allowing the system to log battery data and trigger alerts when necessary (e.g., low battery).

The power subsystem contributes to the reliable operation of the device and ensures reliable, stable, and efficient power for all other subsystems:

- Supplying stable voltages - The buck converter and LDO ensure that critical components receive the correct operating voltages, preventing under-voltage or over-voltage conditions that could harm the device or reduce performance.
- Monitoring battery health: The BQ27441-G1 fuel gauge allows the system to track battery health in real time, ensuring users are informed when the battery needs recharging, thereby preventing unexpected power loss.
- Optimizing energy efficiency: By using a buck converter for the 3.3V supply, the system efficiently manages the power drawn from the battery, extending operational time, especially for wireless communication components like the ESP32-S3 and UWB module.
- Supporting system shutdown protocols: The battery monitoring subsystem can trigger shutdowns or low-power modes in the microcontroller, preserving battery life when the battery reaches a critical low state.

2.3.3 Interfaces

Inputs:

- 3.7V Li-ion Battery (input to buck converter and BQ27441-G1):
  - Voltage range: 3.0V (discharged) to 4.2V (fully charged).
  - Input to buck converter: 3.7V.
  - Direct input to BQ27441-G1 for battery monitoring.
- 3.3V Output from Buck Converter:
  - Input to LDO for further regulation to 1.8V.
- Power Switch
  - Turns the device on and off

Outputs:

- 3.3V Output (from buck converter):
  - Powers the ESP32-S3 microcontroller and UWB module (DWM1000).
- 1.8V Output (from LDO):
  - Powers the LSM6DSL IMU.
- Battery Data Output (from BQ27441-G1):
  - Battery SoC, voltage, current, and temperature data sent via I²C to the ESP32-S3 for logging and monitoring.
- Power LED:
  - LED indicator to convey power status of device (ON, OFF, SLEEP)

2.3.4 Requirements and Verification

Requirements	Verification Procedure
1. Battery Fuel Gauge Accuracy: The BQ27441-G1 must measure the battery's State of Charge (SoC) with an accuracy of ±2% and alert the system when the battery falls below 10% charge.	• Connect the BQ27441-G1 fuel gauge to the ESP32-S3 via the I²C interface. • Fully charge the battery and begin discharging it while monitoring the reported SoC data from the BQ27441-G1. • Compare the reported SoC values with the actual measured charge using a separate power monitoring tool. • Verify that the SoC remains within ±2% of the actual value. • Ensure the system triggers a low battery alert when the SoC drops below 10%. • Document the accuracy over multiple charge/discharge cycles.
2. Power Supply Integrity: The	• Fully charge the 3.7V battery and power the device under

power subsystem must provide continuous power to the system under normal usage conditions for at least 6 hours from a fully charged battery.	normal usage conditions, including wireless communication, UWB sensor activity, and continuous touch emulation. ● Track the operational time using a stopwatch. ● Ensure the system remains operational for at least 6 hours before the battery reaches critical levels. ● Document the operational time and repeat the test to verify consistency.
3. Current Draw: The combined power draw of the system, including the microcontroller and sensors, should not exceed the buck converter's maximum current output.	● Set up the power subsystem and initiate peak operation, including ESP32-S3 Wi-Fi activity, UWB sensor communication, and IMU sensor activity. ● Use a multimeter to measure the total current consumption of the system. ● Verify that the total current draw does not exceed 1A at any point during peak operation. ● Ensure that the power supply provides stable voltage while supporting the peak load. ● Repeat the measurement under various operating conditions to ensure power stability.

Table 1: Power Subsystem Requirements & Verification Table

2.3.5 Design Decisions

We decided to use a buck converter for the 3.7V to 3.3V conversion since the buck converter will increase the current output while dropping the voltage. Buck converters are also more efficient than linear voltage regulators. However, for the 3.3V to 1.8V conversion, we chose an LDO to provide power to the IMU sensor, which incurs a much lighter load than devices needing 3.3V.

We wanted to prevent over-discharge of the battery which is why we included a battery fuel gauge to track the status of the battery. This allows us to alert the user and programmatically turn off the device when the voltage drops significantly. The included LED indicator is a great way for the user to see the status of the device without having to access a software interface.

Additionally, we decided not to integrate a battery recharging system since it would have introduced significant complexity to our device. Not including this system would also prevent issues like overheating and protect against the effects of overcharging. Additionally, given we became a two person team, we felt it was necessary to reduce complexity of development in certain areas in order to make our final product achievable; the focus of our project is on the sensor hardware and software and not on recharging capabilities.

2.4 Sensing Subsystem

2.4.1 Hardware Design Overview

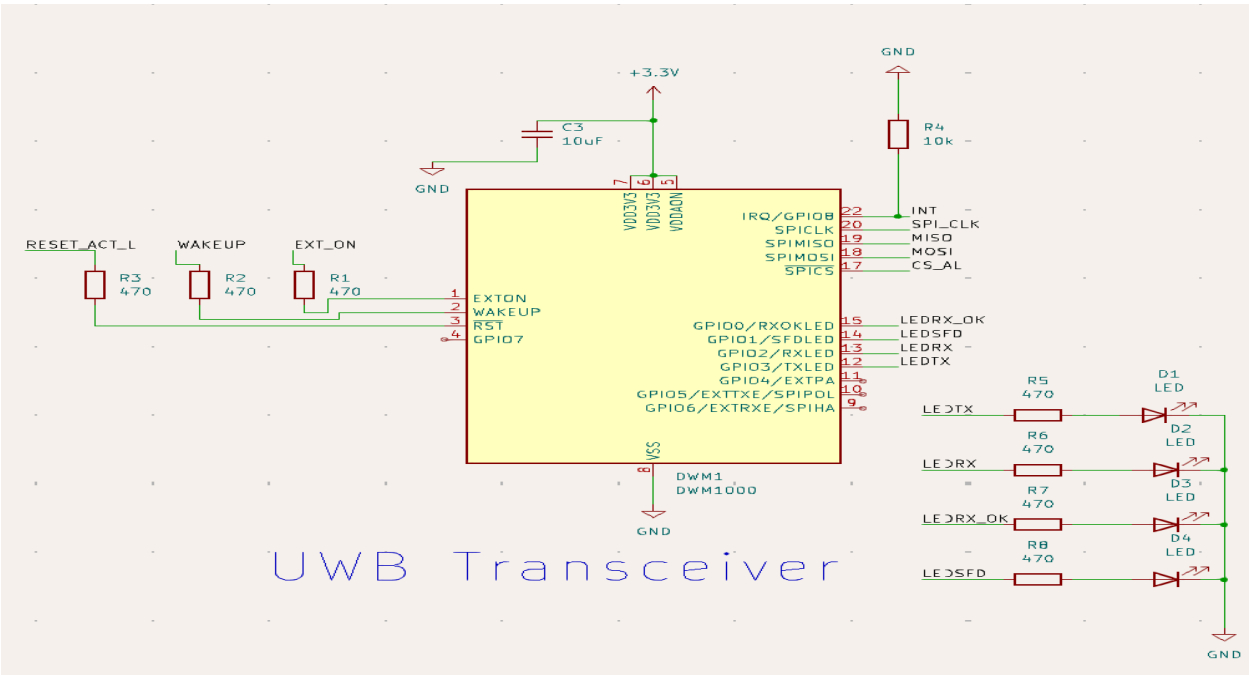

Figure 7: DWM1000 UWB Transceiver Schematic Design

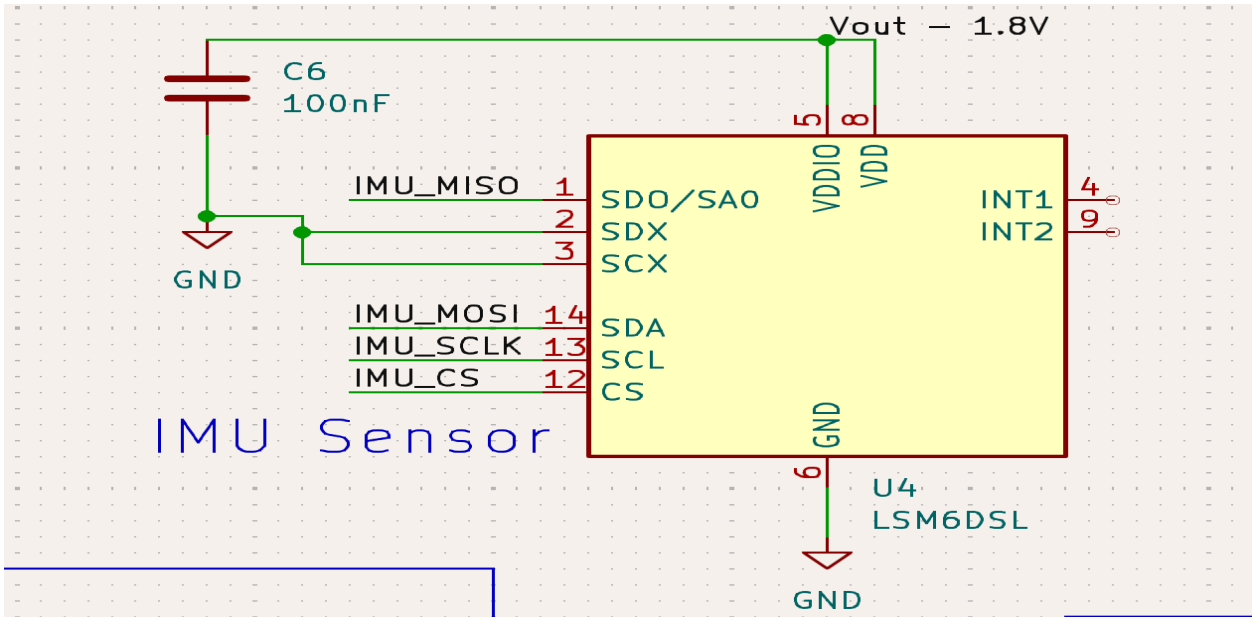

Figure 8: LSM6DSL IMU Sensor Schematic Design

In a technical sense, the two key sensing components are the UWB transceivers and the iMU sensor. We will be utilizing the DWM1000 for our UWB transceiver, modulating our carrier frequency to communicate with the microcontroller, the ESP32-S3-WROOM-1, at a frequency channel of around ~4.5 GHz. The communication protocol we will be using is the SPI communication protocol, which involves four crucial components which are the Chip Select (CS), Master-In Slave-Out (MISO), Master-Out Slave-In (MOSI), and the SPI Clock (CLK). The hierarchical structure of this communication protocol will be thoroughly processed in our firmware, utilizing resourceful libraries like the DW1000Ranging Arduino libraries for the complete abstraction of the communication design protocol.

We will be utilizing the LSM6DSL IMU sensor for our IMU sensor. The communication protocol to the microcontroller generally follows the same procedure as the SPI communication protocol. The main difference between this with the DWM1000 communication protocol is software interfaces. Specifically, we will be taking advantage of the Adafruit LSM6DS3 and STMicroelectronics software libraries designed to fulfill the abstraction requirements for Read/Write of the LSM6DSL IMU sensor, gaining complete data from the accelerometer and gyroscope embedded in the sensor.

The specific pinout layouts for both sensors are motivated and suggested from the datasheets of each sensor, which are LSM6DSL IMU Data Sheet [3] and DWM1000 Data Sheet [8].

2.4.2 Functionality & Contribution

At a high level, this subsystem captures important information on touch events between the pen and the screen using the concept of sensor fusion. Essentially, one UWB transceiver is embedded on the PCB that will contain the overall location data. The tilt and acceleration detection will use a gyroscope/IMU sensor, specifically the LSM6DSL IMU Sensor. This gyroscope will contribute to the exact 3-dimensional position of the pen as it touches the screen, taking into account the tilting of the pen when finalizing location-tracking data. Lastly, the tap/click detection will be done with a small button embedded at the tip of the pen. This will be debounced to make sure that data transfer from the location and motion sensors occurs only when there is direct contact between the pen and the screen; in this case, through the button being depressed.

The sensing subsystem will provide all pre-processed location data which is extremely crucial to create a distinctive judgment of the pen’s location concerning the screen for touch-screen emulation. The pre-processed data will go through a DSP pipeline specifically to decode and assign the microcontroller to communicate with the host device of touch coordinates and HID functionalities via Bluetooth.

2.4.3 Interfaces

- Inputs:
    1. 4.5 GHz RF signal communication between the UWB transceiver embedded on the pen PCB and the static UWB transceivers on the ends of the screen. Tx-Rx communication will use concepts like triangulation and time-of-flight (ToF) for the microcontroller to process the final coordinates of the pen.
    2. 3-dimensional angular velocity (rad/s) and linear acceleration (m/s^2) for tilting mechanism detection.
    3. The sensor power source comes at 3.6 V at around 150 mA.
- Outputs:
    1. Pre-processed data from the sensors through SPI communication with the ESP32-S3 microcontroller.
    2. The button's voltage (HIGH/LOW) to indicate contact with the screen.

2.4.4 Requirements and Verification

Requirements	**Verification Procedure**
1. The UWB sensors must provide useful location information within $\pm$ 1 centimeter of the true pen location.	● Assemble the UWB anchors and set them up at predetermined locations within a controlled environment. This should be on the top-edge locations of the host-device screen. ● Position the pen at a pre-configured location within the tracking area. ● Flash the ESP32-S3-WROOM-1 to allow the DWM1000 wireless communication. ● Connect to the microcontroller via Bluetooth and prepare the Python-based data-logging script. ● Depress the button of the pen on the specific location in a neutral configuration (no tilt, $\pm$ 1 mm). ● Measure the difference between the actual pen location and the post-processed sensor location.
2. The UWB sensors must be able to transfer and receive signals to one another at a 4.5 GHz frequency channel wirelessly.	● Configure DWM1000 firmware and specify the communication usage of Channel 3 at 4492.8 MHz or ~4.5 GHz. ● Assemble the UWB anchors and set them up at predetermined locations within a controlled environment. This should be on the top-edge locations of the host-device screen. ● Solder 4 light-emitting diodes (LED) each

	connected to 4 different pins of the DWM1000 ● Place 470 Ω between the pins and the LEDs. ● Flash the ESP32-S3-WROOM-1 to allow the DWM1000 wireless communication. ● The 4 LEDs will blink, providing information on the DWM1000's instances of transmitting data, receiving data, and synchronization between anchors.
3. The IMU sensor must provide data on angular velocities ($\pm$ 0.2 $\frac{rad}{s}$) and linear acceleration ($\pm$ 0.2 $\frac{m}{s}$)	● Position the pen at a pre-configured location within the tracking area. ● Flash the ESP32-S3-WROOM-1 to allow the IMU sensor communication. ● Connect to the microcontroller via Bluetooth and prepare the Python-based data-logging script. ● Set the pen to rotate at 1 $\frac{rad}{s}$, and operate both in clockwise and counterclockwise directions. ● Compare the gyroscope data from the LSM6DSL sensor with the actual angular velocity manually measured. ● Set the measurement threshold to ensure that data deviation falls under 0.2 $\frac{rad}{s}$. ● Repeat the procedure for the linear acceleration testing; now, instead of rotating, set the pen to accelerate at 1 $\frac{m}{s}$and ensure the deviation falls under 0.2 $\frac{m}{s}$.

Table 2: Sensor Subsystem Requirements & Verification Table

2.4.5 Design Decisions

We decided to go with the DWM1000 for the UWB transceiver and the LSM6DSL IMU sensor. Both sensors have compatibility with SPI and I2C communication protocols. However, after going through the datasheet, the ESP32-S3-WROOM-1 has more distinctive information on the SPI protocol. Additionally, the microcontroller has specialized input/output pins for HSPI and VSPI, which are hardware-based SPI communication. Specifically, they are designed for a fast communication between the master and slave devices. Although not compulsory, we also decided to utilize the output signals from the DWM1000 which are the RXOKLED, SFDLED, RXLED, and TXLED. These signals will be sent to power some LEDs and this part of the schematic is entirely used for debugging purposes to judge the transmission of data between the microcontroller and the UWB transceiver.

Another important design decision is to pick which frequency channel for the overall UWB wireless communication. Although the range of frequencies is between 3.5 to 6.5 GHz, the suggested frequency channel is around 4-5 GHz according to Qorvo UWB Tutorial [9]. This is to avoid over-attenuation from frequent collisions with walls and for the DWM1000 not to take too much of the microcontroller's computing power and resources due to the higher data rate at higher frequencies.

2.5 Control Subsystem

2.5.1 Hardware Design Overview

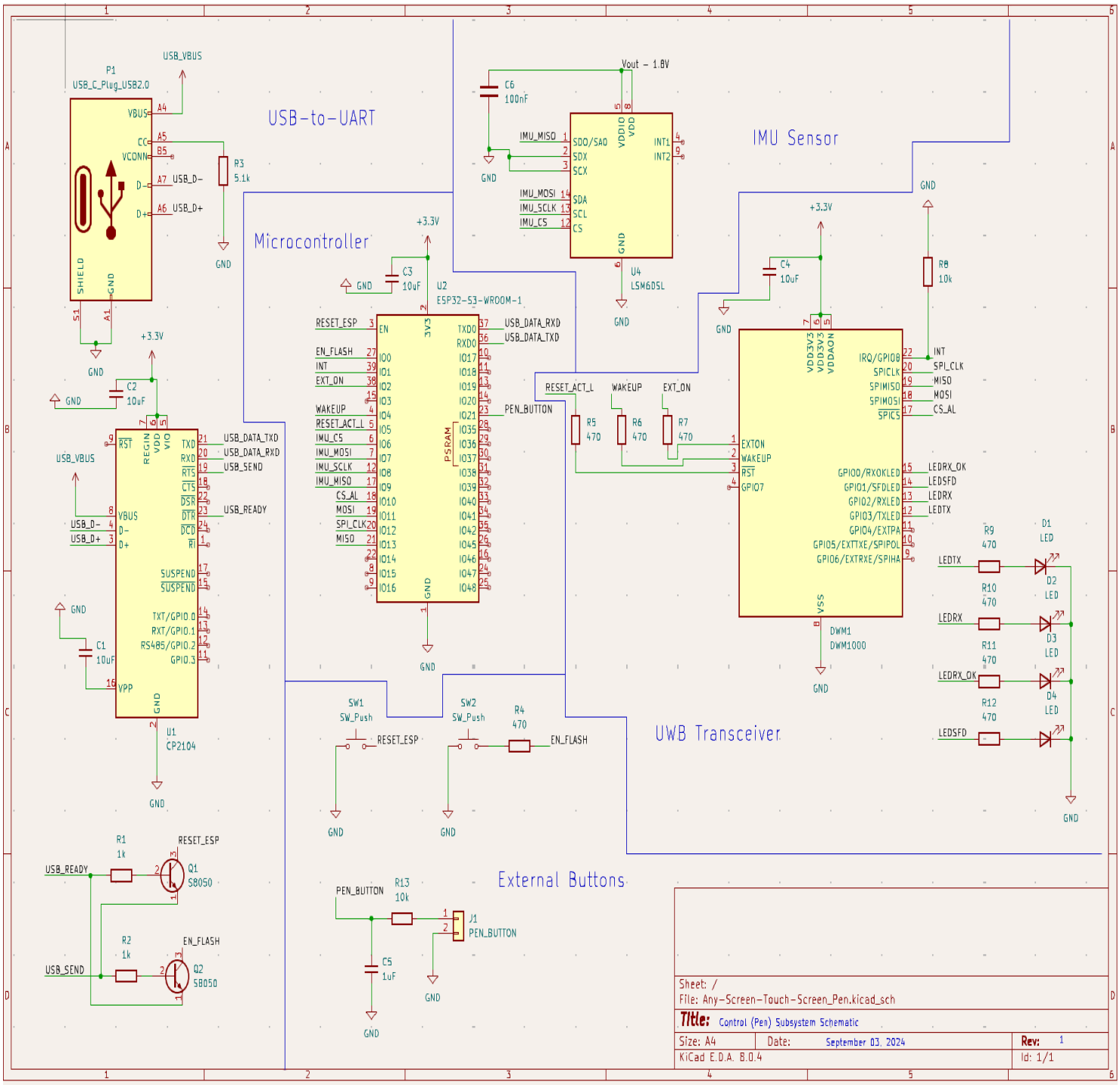

Figure 9: Pen (Control) Schematic Design

The general control design is embedded within the pen enclosure. The computing decisions are made by the microcontroller, the ESP32-S3-WROOM-1. Both data from the external sensors like the DWM1000 UWB transceivers and the LSM6DSL IMU sensor are extensively processed by our microcontroller. Due to the high-rate data transfer communication from the DWM1000, we are using the specialized General Purpose Input/Output (GPIO) pins of the microcontroller specifically for a fast SPI communication system as stated in the ESP32-S3-WROOM-1 data sheet [2]. Aside from the GPIO pins designed for SPI communication, other GPIO pins are utilized to send external signals to the DWM1000 for reset and interrupt messages; these pins are the WAKEUP, RESET_ACT_L, INT, and EXT_ON.
The microcontroller's state will be initially dictated by the two external buttons which are the RESET_ESP and EN_FLASH. As the name suggests, the sequence of pressing this button allows the ESP32-S3-WROOM-1 to enter boot mode, which loads the programs designed on an external host device. To send the programming data and instructions, we are utilizing a USB-to-UART bridge converter as our microcontroller is concurrently designed for UART data communication. We are going to embed a USB-C plug onto our PCB and numerous of the signals projected from the USB-to-UART bridge, the CP2104, will be used to automatically flash our microcontroller after the initial boot, specifically the USB_READY and USB_SEND signals.

Lastly, the pen button, when depressed, will provide the touch-screen emulation signal to a host device via the embedded Bluetooth module in real time; located in our firmware design by utilizing Arduino IDE BLE or Classic Bluetooth modular libraries.

2.5.2 Functionality & Contribution

The first crucial step is denoising; the UWB transceivers will first go through the denoising step for filtering and smoothing. This would involve software algorithms like the outlier rejection for pre-processing and simple moving average (SMA) smoothing for random noises. Subsequently, the control unit is in charge of feature extraction, to only process useful data especially when performing sensor fusion between the sensors. Lastly, the final coordinates estimation will be done through a triangulation algorithm of the three UWB transceivers and the tilt offset will be decoded through IMU's data. The control subsystem will depend hugely on real-time operation capabilities like timers and interrupts as well as utilizing FreeRTOS from ESP-IDF for real-time data processing.

This subsystem is the core of sensor data processing. The ESP32-S3 will receive digitized sensor signals that are sent via SPI protocol by the UWB transceivers and the IMU sensor. The UWB transceivers need to be accurate; therefore, the control subsystem includes a DSP pipeline embedded in the microcontroller to pre-process the signal data for reliable transfer to the host device by using Bluetooth HID protocol and specifications.

2.5.3 Interfaces

- Inputs:
    1. Pre-processed data from the sensors via SPI involving UWB and IMU signal data.
    2. The button's node voltage indicates contact with the screen.
    3. The sensor power source comes at 3.6 V at around 150 mA.
- Outputs:
    1. Post-processed sensor data, Tx transmission to host device via Bluetooth HID protocol.

2.5.4 Requirements and Verification

Requirements	Verification Procedure
1. Must be able to detect screen coordinates with an accuracy of 99%. The metric varies due to differing screen sizes, and the protocol of measuring accuracy will be done by calculating the Euclidean distance between actual and measured coordinates.	• Assemble the UWB anchors and set them up at predetermined locations within a controlled environment. This should be on the top-edge locations of the host-device screen. • Place a known grid of reference points (50 points) for the pen to be located on a device screen. • Flash the ESP32-S3-WROOM-1 to allow the DWM1000 wireless communication. • Connect to the microcontroller via Bluetooth and prepare the Python-based data-logging script. • Depress the button of the pen on the specific location in a neutral configuration (no tilt, $\pm$ 1 mm). • Operate the above procedure for all 50 points to gain a set of data points from the sensor. • Measure the difference between the actual pen location and the post-processed sensor location. Utilize the Euclidean distance measurement to calculate the error. • Measure the accuracy, $\frac{\#\ of\ Accurate\ Measurements}{50} \times\ 100\%$, where a measurement is 'accurate' if the measurement is $\pm$ 1 cm from the actual pen position. • Repeat the verification process with different devices having different screen sizes.
2. To ensure real-time feedback, complete the required sensor data processing with a total communication delay of 60 ms to the host device.	• Assemble the UWB anchors and set them up at predetermined locations within a controlled environment. This should be on the top-edge locations of the host-device screen. • Flash the ESP32-S3-WROOM-1 to allow the

	DWM1000 and LSM6DSL sensor communication. ● Connect to the microcontroller via Bluetooth and prepare the Python-based data-logging script. ● Depress the button of the pen on the specific location in a neutral configuration (no tilt, $\pm$ 1 mm). ● Utilize the *time* Python module to record the timestamp for the sensor data to propagate from the microcontroller to the host device. ● Set the measurement threshold to ensure that the total transmission delay falls under 60 ms.
3. The system must be able to only initiate processing needs when there is direct contact between the pen and the screen.	● Flash the ESP32-S3-WROOM-1 ● Connect to the microcontroller via Bluetooth and prepare the Python-based data-logging script. ● Print the cumulative data logged by the Python script; the terminal should return *None* as there is no instance of the pen being depressed on the screen. ● Contrarily, Depress the button of the pen on the specific location in a neutral configuration (no tilt, $\pm$ 1 mm); the terminal should print the data logged from the sensor and perform the touch-screen emulation on this instance.
4. Operate continuously with a peak power consumption of < ~ 450 mW.	● Assemble the pen with the complete power delivery subsystem. ● Solder the wire connection between the ESP32-S3-WROOM-1 power source (3.3 V), ground, and other compulsory electrical components or signals. ● Ensure the system is on, then utilizing a multimeter, verify that the powering nodes are ~3.3 V with a maximum current of ~136 mA. ● Perform this verification when the pen is idle and when it is actively sending data to the host device.

Table 3: Pen (Control) Subsystem Requirements & Verification Table

2.5.5 Design Decisions

Initially, we chose the generic ESP32-S3 module for our microcontroller and the general computing device. However, this requires us to create an external design for the Bluetooth antenna to communicate with other devices via Bluetooth properly. We decided that switching over to the ESP32-S3-WROOM-1 is a better choice as the microcontroller comes with a Bluetooth antenna embedded in the module. The specific reason we chose this microcontroller is due to its high processing capabilities, especially for real-time applications. Concurrently, it

comes with both a USB On-the-Go (OTG) module and Bluetooth, which gives us the ability to translate between one communication protocol or the other with the host device.

For flashing the microcontroller, we decided to take the approach of soldering a USB-C plug on our pen and anchor designs. The reason is that we have researched thoroughly and found out that a USB-to-UART serial communication for flashing the ESP32 series is recommended. Hence, we are utilizing the CP2104 as the bridge converter. The external buttons are mainly used for the initial boot of the microcontroller. The S8050 transistors are used to manipulate the outputting signal from the CP2104 to be used for automatic flushing of the microcontroller instead of continuously relying on the external buttons. This idea was inspired by a UWB-based project on indoor localization by makerfab [7].

2.6 Software Monitoring Subsystem

2.6.1 Software Design Overview

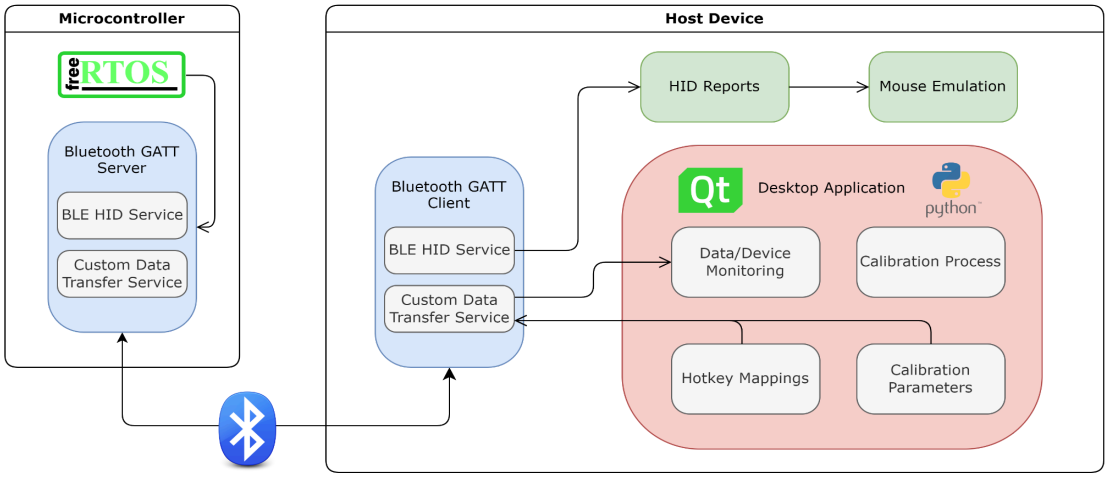

Figure 10: Software Monitoring Flow Chart

The software subsystem for the Any-Screen Touch-Screen device is designed to provide a cross-platform solution using the Qt framework, enabling communication between the ESP32-S3 microcontroller and the host computer. It processes user inputs, manages system calibration, and handles real-time data exchange between the device and the host via Bluetooth GATT and BLE HID over GATT. The software supports both touch input emulation and sensor data analysis (UWB and IMU) while offering a graphical interface for users to configure and interact with the system. The subsystem is designed for seamless operation across multiple operating systems, including Windows, macOS, and Linux.

2.6.2 Functionality & Contribution

The software subsystem plays a vital role in handling data and communication between the ESP32-S3 microcontroller and the desktop environment. It receives data via Bluetooth for touch events and for transferring UWB and IMU sensor logs. This data is processed to emulate mouse movements, clicks, and gestures, with a real-time data logging feature that captures sensor inputs for accuracy analysis and troubleshooting. A Qt-based GUI enables users to input screen parameters, such as resolution and size, for calibration. These calibration parameters are then transmitted to the microcontroller to ensure precise touch tracking.

The software also provides an interface for users to assign custom hotkeys and additional pen functionalities, such as scrolling, dragging, and clicking, to enhance interaction. A data logging feature captures location and IMU data from the microcontroller in real-time, which is stored for system calibration and long-term performance monitoring. Logs are transmitted through the BLE GATT Custom Data Transfer Service and can be analyzed to analyze the device's accuracy and performance.

Designed for cross-platform support, the software runs on Windows, macOS, and Linux, ensuring adaptability to different desktop environments and screen configurations. The software subsystem serves as the critical bridge between hardware components, such as UWB sensors and the ESP32-S3 microcontroller, and the end-user. It transforms raw sensor data into meaningful actions while maintaining system calibration through a user-friendly interface.

2.6.3 Bluetooth Communication

The software subsystem uses a dual Bluetooth GATT and BLE HID over GATT model for communication between the microcontroller and the host computer, facilitating both real-time touch input emulation and sensor data monitoring.

1. GATT Profile (Client-Server Model):
    - GATT Server: ESP32-S3 Microcontroller
    - GATT Client: Desktop application
2. GATT Services:
    - HID Service: Manages all HID-related communication for touch input (e.g., mouse movements, clicks).
    - Custom Data Transfer Service: Manages data logging and monitoring from the ESP32-S3, transmitting UWB and IMU sensor data to the desktop app.
3. GATT Characteristics:
    - HID Report Characteristic: Sends HID input reports (mouse movements, clicks) to the desktop application.
    - Protocol Mode Characteristic: Switches between Boot Mode and Report Mode for HID communication.

- Control Point Characteristic: Manages device control, such as resetting the HID device.
- Data Log Characteristic: Transmits sensor data and logs to the desktop application for analysis.
- Control Characteristic: Allows the desktop app to send control commands back to the ESP32-S3 (e.g., to start/stop data logging) or to communicate calibration parameters

2.6.4 Tech Stack

Tech Stack for Cross-Platform Desktop Application:

1. Frontend (User Interface and GUI):
   - Qt Framework (C++/Python): Used to develop the cross-platform desktop application.
   - QtWidgets/QML: For building the GUI, handling calibration, data logging, and touch functionalities.
2. Backend (Business Logic and Communication):
   - Qt Bluetooth Module: Provides support for BLE GATT communication, managing both HID events and data transfer services.
3. Bluetooth HID Communication:
   - Arduino BLE IDE: The firmware for the ESP32-S3 microcontroller will be developed using the Arduino BLE IDE instead of ESP-IDF. This simplifies the development process for handling Bluetooth HID and BLE GATT profiles. The firmware will:
     - Implement HID over GATT to transmit touch input data (e.g., mouse movements, clicks) from the UWB and IMU sensors.
     - Handle data transfer services to log sensor data and manage touch accuracy.
   - FreeRTOS will be leveraged to manage real-time data processing on the ESP32-S3, ensuring efficient sensor data handling with minimal delays.

Programming Languages and Technologies

- Frontend:
  - C++ or Python using the Qt framework for GUI development.
  - QtWidgets or QML for building user interfaces that handle calibration, logging, and user interaction.
- Backend:
  - C++ or Python for handling business logic, Bluetooth communication, and real-time data processing.

- Firmware:
    - C/C++ using the Arduino BLE IDE for developing the Bluetooth HID and BLE GATT profiles. FreeRTOS will be used on the microcontroller to manage real-time sensor data processing.

2.6.5 Interfaces

Inputs:

- Bluetooth HID data from the ESP32-S3 microcontroller (e.g., touch events such as clicks and cursor movements).
- Sensor data logs via the BLE GATT Custom Data Transfer Service (e.g., UWB location and IMU data).
- User input for screen resolution, size, and hotkey configurations through the Qt GUI.

Outputs:

- Calibration parameters sent to the microcontroller, ensuring the system is accurately calibrated based on the screen setup.
- HID input reports to the desktop application, simulating touch interactions and cursor movements.
- Data logs on touch interactions for accuracy analysis, ensuring long-term performance validation.
- Control commands sent to the ESP32-S3 to manage logging or reset functionality via the Custom Data Transfer Service.

2.6.6 Requirements and Verification

Requirements	Verification Procedure
1. The software must be able to handle incoming sensor data	• Set up the device on a display of known dimensions and screen size. • Calibrate the system using the GUI to input the screen size and resolution. • Flash the ESP32-S3 microcontroller with the Arduino BLE firmware to ensure it communicates location data to the software. • Use a known grid of reference points (for example, 16 points) on the screen for calibration. • Depress the pen on each reference point and log the actual vs processed location.

2. The software must complete data transmission and processing within 250 milliseconds to ensure responsive user interaction.	• Set up the device and perform system calibration. • Connect the ESP32-S3 to the desktop application via Bluetooth and log sensor data. • Use the Python **time** module to measure the transmission delay from the microcontroller to the host device. • Perform actions like mouse movements or taps and record timestamps at the moment the interaction is detected and the moment at which the interaction is emulated. • Ensure the time difference is under 250 milliseconds. • Repeat the procedure across different screen sizes to ensure consistent performance.
3. The software must support a graphical user interface (GUI) that allows users to input screen resolution, screen size, and hotkey mappings.	• Launch the GUI on the host device after installing the software on Windows, macOS, and Linux systems. • Verify that users can input screen resolution and screen size within the GUI. • Test the hotkey assignment feature by allowing users to assign specific actions (click, scroll, drag) to buttons on the pen. • Ensure the GUI correctly sends this data to the microcontroller and that assigned actions are executed correctly. • Test across different screen sizes and platforms to ensure consistent performance.
4. The software must log touch interaction data (e.g., location, clicks, and movements) for accuracy analysis.	• Set up the device and ensure the ESP32-S3 is connected via Bluetooth. • Perform touch interactions (taps, clicks, drags) on the screen while the software is running. • Verify that the software logs interaction data by reviewing the log files generated by the **Custom Data Transfer Service** via BLE GATT. • Analyze the logged data to ensure it includes time, location, and type of interaction. • Ensure the system allows for real-time data transmission and logs can be retrieved from the GUI for accuracy analysis.
6. The software must support a calibration process where the user maps the pen to multiple points on	• Launch the software on the host device and initiate the calibration process through the GUI. • The GUI should display multiple reference

the screen for precise touch accuracy.	points on the screen for calibration. ● Verify that users can map the pen to these points on the screen. ● Check that the software sends the calibration parameters to the ESP32-S3 microcontroller. ● Perform touch tests post-calibration and log the touch accuracy to ensure the system accurately tracks the pen location after calibration.

Table 4: Software Monitoring Subsystem Requirements & Verification Table

2.6.7 Design Decisions

One could expect the device to operate without any sort of GUI, but there are several reasons we decided to include one. First, it provides a method for the user to input custom parameters that affect the device's accuracy, the most important being the screen size and resolution. The user also needs a way to map custom hotkeys to the buttons on the pen. Additionally, it gives the user a way to monitor that device's accuracy over time and receive important alerts (eg. if the battery has low charge)

For the GUI, we decided to use the Qt framework as it would allow us to more easily build a cross-platform application compared to other frameworks like Flutter or Electron which would require more custom software to operate across multiple OSs. Using Qt allows us to easily develop on Windows, macOS, and Linux.

We've also decided to include the calibration process as it is important towards maintaining the accuracy of our device. Many UWB modules send inaccurate data right out of the box and performing this calibration step will allow us to modify the location data as necessary and account for variability in the accuracy of the DWM1000 UWB modules.

2.7 Tolerance Analysis

The most critical aspect of our device is the **UWB-based positioning and communication system**, which uses Time of Flight (ToF) to eventually locate the pen's position on the screen.

The accuracy of this positioning is essential, as even a small error in detecting the pen's position could lead to significant usability issues, such as incorrect touch points on the screen, poor gesture recognition, and overall frustration for the user.

The primary risk in this system is the error in the position calculation due to timing inaccuracies in the UWB sensors. To achieve accurate touch-screen emulation, the positioning error must remain below a threshold of 5% of the screen size. Given the short timing intervals involved in UWB signal transmission and reception, any noise, propagation delay, or timing drift could result in larger positioning errors.

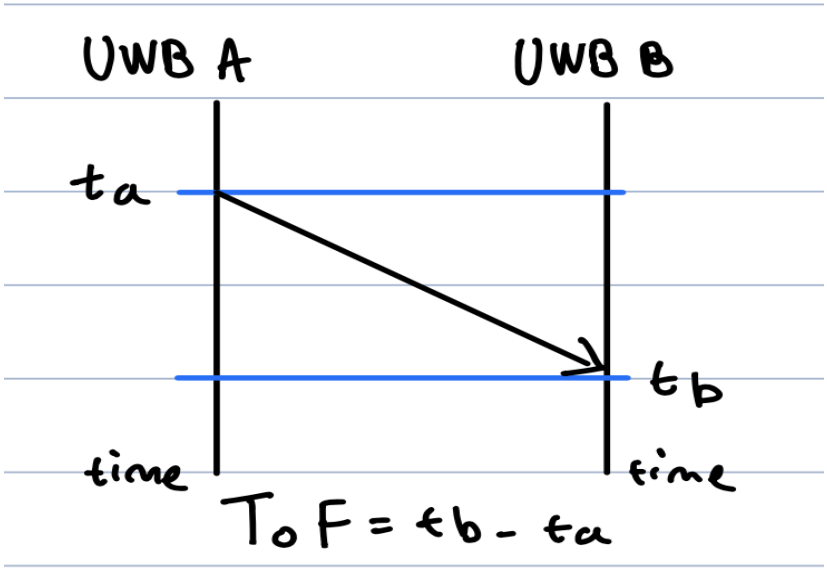

Figure 11: The SSR Time of Flight Algorithm

Figure 11 shows the simple Single-Sided Ranging (SSR) ToF. The ToF, which can be decoded to calculate the distance between the pen and the screen, will be given by this expression:

$$ToF = t_b - t_a$$

Although this looks like a simple technique, it holds a high potential for error when used for the UWB communication system: **the clock offsets and drifts of the UWB transceivers are not synchronized**.

A better way would be to use a Two-Way Ranging (TWR) ToF algorithm. Figure 12 shows the TWR in action between two UWB transceivers. Mathematically, the expression of the ToF becomes:

$$ToF = \frac{r_a - d_b}{2}$$

Note: r_a is the total time it takes for UWB a to transmit and receive back the signal from UWB b. d_b is the process time UWB b takes between receiving the initial signal from UWB a and sending it back.

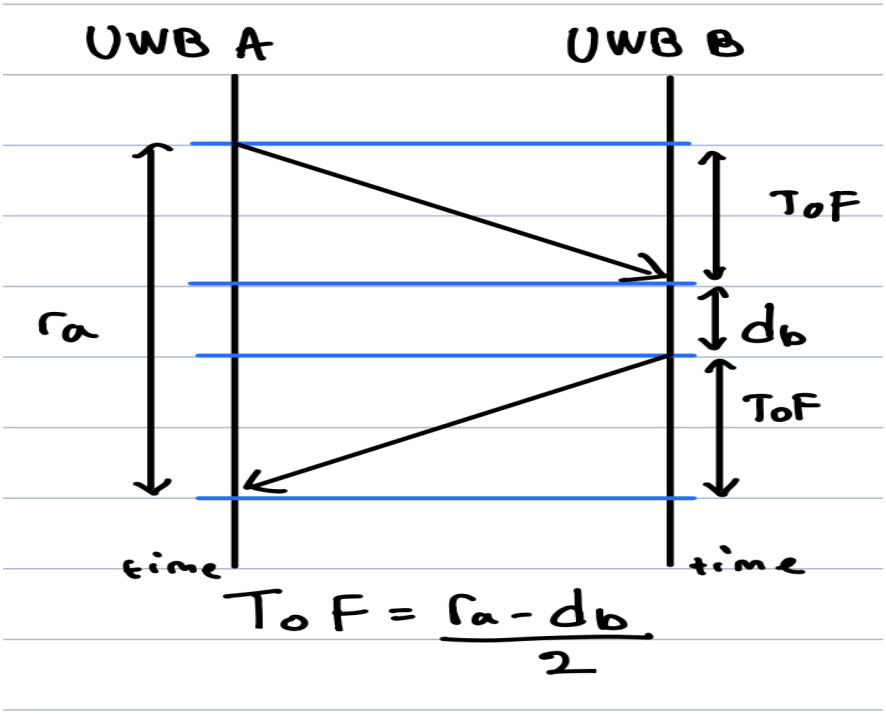

Figure 12: The TWR Time of Flight Algorithm

Now, the clock offsets and drifts will be completely local. This means the error of distance calculation no longer depends on strict synchronization. The error analysis for the TWR algorithm is the following:

Let e_a and e_b be the respective errors due to clock drifts, then:

$R_a = (1 + e_a)r_a$, $D_b = (1 + e_b)d_b$ where R_a and D_b are total time measurements accounting for errors. We then have:

$$ToF_{estimate} = \frac{R_a - D_b}{2}$$

$$ToF\ error = ToF_{estimate} - ToF = \frac{R_a - D_b}{2} - \frac{r_a - d_b}{2} = \frac{1}{2}(e_a r_a - e_b d_b)\ .$$

However, we know that $r_a = 2\,ToF + d_b$. Hence, we have:

$$distance\ error = c * (e_a ToF + \frac{d_b}{2}(e_b - e_b))\ ,$$ where c is the speed of light.

We have reduced the distance error to be completely independent of clock synchronization between the UWB transceivers. However, an issue remains. Notice that $e_a ToF$ is in the magnitude of nanoseconds due to the relatively high speed of light. However, $\frac{d_b}{2}$ is still in the **milliseconds ToF error magnitude, which makes our distance error relatively concerning**.

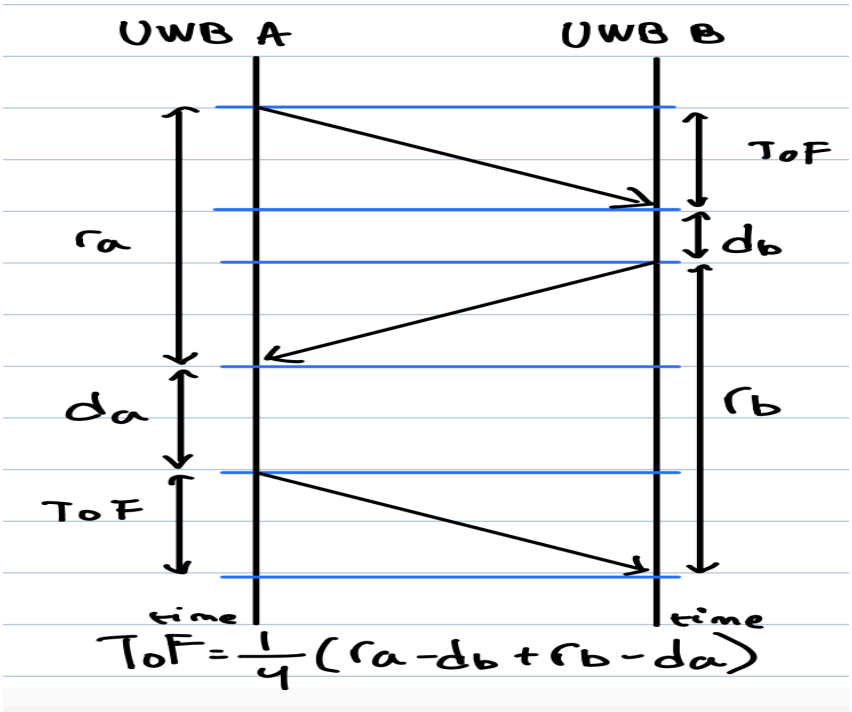

Figure 13: The ADS TWR Time of Flight Algorithm

Note: r_a is the total time it takes for UWB a to transmit and receive back the signal from UWB b. d_b is the process time UWB b takes between receiving the initial signal from UWB a and sending it back.

Further improvements need to be made to make our touch-screen emulation accurate. The Alternative Double-Sided TWR (ADS TWR) is an extension of the TWR to minimize the distance error by eliminating the $\frac{d_b}{2}$ error magnitude, as shown in Figure 13. The algorithm is as follows:

$$ToF = \frac{1}{2}(ToF_a + ToF_b) = \frac{1}{4}(r_a - d_b + r_b - d_a).$$

With this, the error analysis for our ADS TWR algorithm is the following:

$$r_a = 2T + d_b,\ r_b = 2ToF + d_a.\ \text{Then,}$$

$$r_a r_b = (2ToF + d_b)(2ToF + d_a) = 2ToF\,(2ToF + d_a + d_b) + d_a d_b$$

Then, $r_a r_b - d_a d_b = 2ToF\,(2ToF + d_a + d_b)$. Hence, the time of flight can be expressed as follows:

$ToF = \frac{1}{2}\frac{r_a r_b - d_a d_b}{2ToF + d_a + d_b}$. Substituting $r_a = 2ToF + d_b,\ r_b = 2ToF + d_a$ the denominator to omit ToF, our final expression will be:

$$ToF = \frac{1}{2}\frac{r_a r_b - d_a d_b}{2(r_a + d_a)} = \frac{1}{2}\frac{r_a r_b - d_a d_b}{2(r_b + d_b)}.\ \text{Accounting for error,}$$

$ToF_{estimate} = \frac{1}{2}\frac{R_a R_b - D_a D_b}{2(R_a + D_a)} = \frac{1}{2}\frac{R_a R_b - D_a D_b}{2(R_b + D_b)}$, where $R_a = (1 + e_a)r_a\ ,\ D_b = (1 + e_b)d_b$

Finally,

$$ToF_{estimate} = \frac{(1 + e_a)(1 + e_b)}{(1 + e_b)}\ \frac{1}{2}\frac{r_a r_b - d_a d_b}{2(r_b + d_b)} = (1 + e_a)\,ToF.$$

$$ToF\ error = ToF_{estimate} - ToF = (1 + e_a)\,ToF - ToF = e_a ToF.$$

Our distance error will then be:

$$distance\ error = c * ToF\ error = c * e_a ToF\ ,\ \text{where } c \text{ is speed of light.}$$

We reduced **the ToF error magnitude to nanoseconds** with the ADS TWR algorithm for UWB transceivers communication protocol.

Furthermore, to help the ADS TWR minimize the distance error, we plan to commit to these extra implementation strategies:

1. **Sensor Calibration**: Regularly calibrating the UWB sensors can help mitigate timing drift and errors, ensuring that the ToF calculations remain accurate over time. Furthermore, our calibration process involves taking sensor data at several predefined locations on the screen with the help of the GUI. We can use these data-to-location mappings to help with error correction and outlier detection in real-time.
2. **Error Correction Algorithms**: Implement filtering techniques or error correction algorithms to smooth out data points that are outside the expected range of variation, reducing the likelihood of large positioning errors
3. **Gyroscope Location Adjustment**: The UWB module may not be located at the tip of the pen, causing the UWB sensors to incorrectly sense where the pen's tip is located if the pen is held at an angle. Offsetting the location with the gyroscope data will allow accurate pen tip location.

3 Cost and Schedule

3.1 Cost Analysis

3.1.1 Parts/Materials

For this project, we are planning on making 2 prototypes for continuous debugging improvement and connectivity development. Table 5 specifies the total cost of the hardware parts and components we plan to utilize.

Component	Part Number	Unit Price	Quantity	Number of Prototypes	Total Cost	Sourcing from:
Microcontroller	ESP32-S3-WROOM-1	$3.48	4	2	$27.84	Electronic Services Shop
UWB Module	DWM1000	$23.62	4	2	$188.96	My.ECE ordering or free samples
IMU	LSM6DSLTR	$6.01	1	2	$12.02	Link
3.7V Batteries	Samsung 35E 18650	$2.75	5	2	$27.50	Link
Button	E-SWITCH TL1105AF100Q	$0.51	11	2	$11.22	ECE supply shop
Pen Enclosure	3D Printed	$7.00	1	2	$14.00	Grainger IDEA Lab

Anchor Enclosure	3D Printed	$10.00	2	2	$40.00	Grainger IDEA Lab
1.8V LDO	AMS1117-1.8	$0.31	1	2	$0.62	Link
Buck Converter	LM2596S-5.0	$4.73	4	2	$37.84	Link
Battery Gauge	BQ27441DRZT-G1A	$3.26	4	2	$26.08	Link
Battery Holder	BH-18650-W	$4.17	4	2	$33.36	Link
Power Switch	RSC141D1000-116	$1.73	4	2	$13.84	Link
10 kΩ resistor	RMCF0805JG10K0	$0.10	18	2	$3.60	ECE 445 Inventory/2070 Lab
10 uF capacitor	GRM21BR61H106ME43L	$0.33	20	2	$13.20	ECE 445 Inventory/2070 Lab
100 uF capacitor	50ZLH100MEFC8X11.5	$0.27	3	2	$1.62	ECE 445 Inventory/2070 Lab
0.47 uF capacitor	CL21A475KAQNNNE	$0.10	4	2	$0.80	ECE 445 Inventory/2070 Lab
1 uF capacitor	CL21B105KBFNNNG	$0.11	7	2	$1.54	ECE 445 Inventory/2070 Lab
BJT Transistor	S8050	$0.11	2	2	$0.44	Link
USB-to-UART Bridge	CP2104-F03-GM	$5.43	3	2	$32.58	ECE 445 Inventory/2070 Lab
USB-C (2.0) Plug	Molex 2171750001	$0.81	3	2	$4.86	Link
470 Ω resistor	MFR-50FRF52-470R	$0.18	24	2	$8.64	ECE 445 Inventory/2070 Lab
1 kΩ resistor	YAGEO CFR-25JR-52-1K	$0.10	6	2	$1.20	ECE 445 Inventory/2070 Lab
5.1 kΩ resistor	CFM14JT5K10	$0.10	3	2	$0.60	ECE 445 Inventory/2070 Lab
LED	XLMYK12W	$0.34	12	2	$8.16	ECE 445 Inventory/2070 Lab
				Final Cost	**$510.52**	

Table 5: Bill of Materials

3.1.2 Estimated Hours of Development

Both members of the group are Computer Engineering students. Based on the Grainger College of Engineering website on post-graduate success [5], the average starting salary for a Computer Engineering graduate from the University of Illinois at Urbana-Champaign is $118,752/yr, which is equivalent to $57.09/hr.

Category	**Estimated Hours**	
	Sakhi	Muthu
Circuit Design	15	15
Board Layout	10	10
Soldering	10	10
Firmware Development	25	15
Software Data Monitoring	5	15
Prototype & Debug	40	40
Documentation & Logistics	40	40
Total Hours	**145**	**145**

Table 6: Estimated Hours

With the hourly estimate configured in Table 6, we compute the estimated cost for the project labor as follows:

$$\$57.09/hr \times 145\,hrs = \$8,278.05\,per\ team\ member$$

3.1.3 External Materials and Resources

- ECE Supply & Electronic Services Shop
    - We are planning on purchasing existing resources that are available directly from the ECEB machine shop. The main component we are getting is the ESP32-S3-WROOM-1 microcontroller; the rest of the items are extra components for debugging purposes including resistors, wires, soldering equipment, etc.

- Senior Design Lab Resources
    - We are planning to utilize the soldering, testing, and debugging resources provided in the Senior Design Laboratory which include soldering iron, oscilloscope, multimeters, and power supplies. Concurrently, we will be taking advantage of the available electrical components which involve capacitors, resistors, transistors, buttons, and switches.

- External Resources
    - Aside from the generic electrical components provided, we will be purchasing the rest of the components which include the UWB transceivers (DWM 1000), IMU sensor (LSM6DSL), USB-to-UART bridge (CP2104), and other crucial components for the power subsystem from external resources that mainly come from Digikey and Amazon.

- Grainger IDEA Lab
    - We are planning on 3D printing our enclosures at the IDEA lab within Grainger Library. We have to submit 3D design files through an online portal and the lab will print the designs for us.

3.1.4 Total Estimated Cost

We conclude the cost analysis by summing up the cost estimation from both labor and materials cost breakdowns. Table 7 is the conclusion of the total estimated cost of our project.

Category	**Estimated Cost**
Material and Parts	$510.52
Total Labor Cost	$16,556.1
Total Estimated Cost	$17,066.62

Table 7: Total Estimated Cost

3.2 Schedule

- *Week of 9/30; Week 6*
    - Design Document, Schematic Review & Feedback, - Sakhi, Muthu
    - Cost Analysis and Breakdown, - Muthu
    - PCB Layout - Sakhi, Muthu

    - ESP32 Firmware Development (SPI Protocol) - Sakhi

- *Week of 10/7; Week 7*
    - Design Review Presentation, – Sakhi, Muthu
    - PCB Layout & Routing, – Muthu
    - Initial ESP32 Firmware Flash Test, – Sakhi, Muthu
    - ESP32 Firmware Development (DWM1000 TWR Algorithm), – Sakhi
    - Ordering Parts – Sakhi, Muthu

- *Week of 10/14; Week 8*
    - Ordering Parts, – Muthu
    - Ordering PCB, – Sakhi, Muthu
    - Sensor Development, – Sakhi, Muthu
    - ESP32 Firmware Development (DSP Triangulation) – Sakhi

- Week of 10/21; Week 9
    - Prototype I Assembly, – Sakhi, Muthu
    - Hardware Test, Debug, & Improve, – Sakhi, Muthu
    - ESP32 Software Development (Bluetooth HID Protocol), – Sakhi
    - Host Device Software Development (Data Logging) – Muthu

- Week of 10/28; Week 10
    - Prototype I: Hardware Debug & Review, – Sakhi, Muthu
    - Firmware Debug & Optimization, – Muthu
    - Order 2nd Wave PCB, – Sakhi, Muthu
    - Software Test on Touch Emulation – Sakhi

- Week of 11/4; Week 11
    - Prototype II: Hardware, Firmware, & Software Integration – Sakhi, Muthu
    - 3D Print Pen Enclosure,v– Muthu
    - Prepare for Mock Demo – Sakhi, Muthu

- Week of 11/11; Week 12
    - Prototype II: Debugging & Review, – Sakhi, Muthu
    - Prepare for Mock Demo – Sakhi, Muthu
- Week of 11/18; Week 13
    - Mock Demo, – Sakhi, Muthu
    - Prototype II: Debugging & Review, – Sakhi, Muthu
    - Prepare for the Final Demo & Presentation – Sakhi, Muthu

- Week of 11/25; Week 14
    - Fall Break

- Week of 12/2; Week 15
    - Final Debugging & Review, – Sakhi, Muthu
    - Work on Final Presentation & Paper, – Sakhi, Muthu
    - Final Demo – Sakhi, Muthu

- Week of 12/9; Week 16
    - Final Presentation – Sakhi, Muthu
    - Final Paper – Sakhi, Muthu
    - Lab Notebook and Checkout – Sakhi, Muthu

4 Ethics and Safety

RF Exposure and UWB Compliance: The UWB transceivers emit radiofrequency (RF) signals, which must comply with FCC regulations (Part 15 Subpart F) concerning RF exposure limits and interference control. We will ensure that our device operates within the permitted frequency bands (4.5 GHz) and maintains safe RF power levels to avoid harmful exposure to users and

prevent interference with other devices. The Code for Federal Regulations details an entire section on Ultra-Wideband operation:

Data Privacy: Our software subsystems compile data logs and take in the user's screen information and computer environment. Following the ACM Code of Ethics section 1.6-1.7, we must ensure that this data is handled with the highest regard for privacy and confidentiality. Any data collected during calibration or logging must not be misused or shared without explicit user consent.

To mitigate this risk, we will ensure users are informed about what data is collected, aligning with principles of transparency and accountability. We will also only collect data that is strictly necessary for location/mouse data logging and accuracy analysis.

Battery Safety: Our device uses 3.7V rechargeable batteries, which can pose risks like overheating or fire if mishandled. We will follow IEEE Standard 1725 for rechargeable battery safety, mitigating these risks by implementing a power monitoring system that will prevent excessive discharging of the battery

Campus and Lab Policies: We will adhere to the University of Illinois laboratory safety guidelines, which emphasize the proper handling of electronic components and the safe use of tools/materials. This includes wearing appropriate PPE and ensuring that workspaces/lockers are kept organized and free of hazards.

5 Citations

[1] Advanced Monolithic Systems. (n.d.). *AMS1117-1.8 low dropout voltage regulator datasheet*. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/6283/AMS1117-1.8.pdf

[2] Digi-Key Electronics. (n.d.). *ESP32-S3-WROOM-1-N8*. [Online]. Available: https://www.digikey.com/en/products/detail/espressif-systems/ESP32-S3-WROOM-1-N8/15200089

[3] Digi-Key Electronics. (n.d.). *LSM6DSLTR*. [Online]. Available: https://www.digikey.com/en/products/detail/stmicroelectronics/LSM6DSLTR/6192804

[4] Espressif Systems. (n.d.). *ESP32-S3 products*. [Online]. Available: https://www.espressif.com/en/products/socs/esp32-s3

[5] The Grainger College of Engineering. (n.d.). *Computer engineering major*. University of Illinois Urbana-Champaign. [Online]. Available: https://grainger.illinois.edu/academics/undergraduate/majors-and-minors/computer-engineering

[6] S. Hossain and B. A. Khawaja, "A survey on ultra-wideband communication technologies and applications," *IEEE Access*, vol. 7, pp. 29714-29730, 2019, [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2892430

[7] Makerfabs. (n.d.). *ESP32 UWB (Ultra-Wideband)*. [Online]. Available: https://www.makerfabs.com/esp32-uwb-ultra-wideband.html

[8] Qorvo. (n.d.). *DWM1000: Ultra-wideband module*. [Online]. Available: https://www.qorvo.com/products/p/DWM1000

[9] Qorvo. (n.d.). *Getting back to basics with ultra-wideband (UWB)*. [Online]. Available: https://www.qorvo.com/resources/d/qorvo-getting-back-to-basics-with-ultra-wideband-uwb-white-paper

[10] S. Raghavan, S. Kittipiyakul, and Y. Srikant, "Ultra-wideband channel modeling for wireless communication systems," *IEEE Transactions on Wireless Communications, 20(3)*, 1789-1801. [Online]. Available: https://doi.org/10.1109/TWC.2020.3041958

[11] RRC. (2023). *Real-world algorithms for IoT and data science*. [Online]. Available: https://rrc-uiuc.notion.site/Real-World-Algorithms-for-IoT-and-Data-Science-74d8f612f74a4c1689760dafa31ef93d

[12] Texas Instruments. (2018). *BQ27441-G1 fuel gauge datasheet* (SLUSBH2B). [Online]. Available: https://www.ti.com/lit/ds/symlink/bq27441-g1.pdf

[13] U.S. Government Publishing Office. (2022). *Title 47—Telecommunication, chapter I—Federal Communications Commission, subchapter A—General, part 15—Radio Frequency Devices, subpart F—Unlicensed National Information Infrastructure Devices*.

[Online]. Available:
https://www.ecfr.gov/current/title-47/chapter-I/subchapter-A/part-15/subpart-F