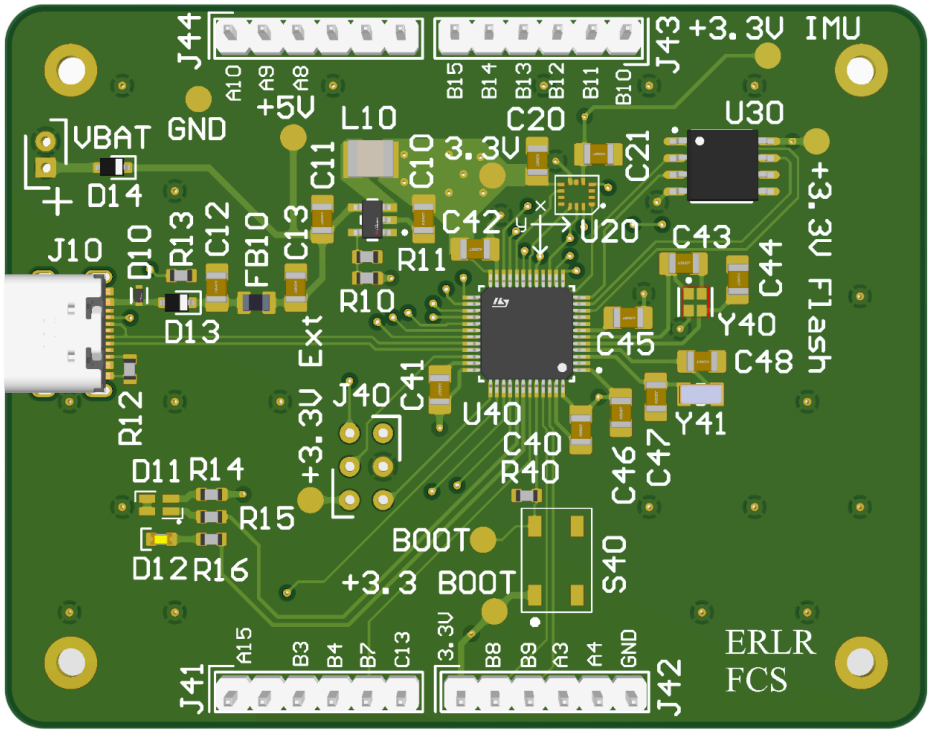
Ultra-Low Power STM32 Based Flight Characterization System

Eliott Lacomme Ruiz



GitHub

July - Sept ‘25

Acknowledgements

Thank you to my mentor, Professor Nicholas Marchuk, for guiding me through this amazing project and Northwestern University's Research Apprentice Training Grant for providing the funds for the project.

Introduction

This document serves as the final report on the complete process of building my custom STM32-based flight characterization system. It covers the ideas, mistakes, challenges, and future developments that emerged throughout the project.

My goal was to design a custom, ultra–low-power STM32-based printed circuit board (PCB) equipped with a high-precision sensor for characterizing dynamic systems. I had previously designed several two-layer PCBs for the Northwestern University Space Technology and Rocketry Society (NUSTARS), which provided a solid foundation for creating sensor modules and breakout boards used in earlier competitions. However, a recurring issue in past projects was slow data logging, caused by the limited speeds of sensor modules and Arduino-based microcontrollers. To address this, I decided to step outside my comfort zone and design a more advanced board using my own integrated circuits (ICs) and non-volatile flash memory.

To overcome these challenges, I developed a four-layer board that incorporated a custom IC peripheral, flash memory, and USB-C programming support. To push myself even further, I focused on using ultra–low-power components. This approach not only helped me gain deeper familiarity with working directly with microcontrollers, but also served as a stepping stone toward building more complex and thorough projects in the future.

Timeline

Before beginning this project, I needed a clear understanding of the amount of time I had available, since I was working against a strict deadline to finish before school started up again. This meant I had to plan carefully, particularly around when to order the PCB, so that I would still have enough time to assemble and test the circuit before the summer ended.

To stay organized, I created a Gantt chart, shown below.

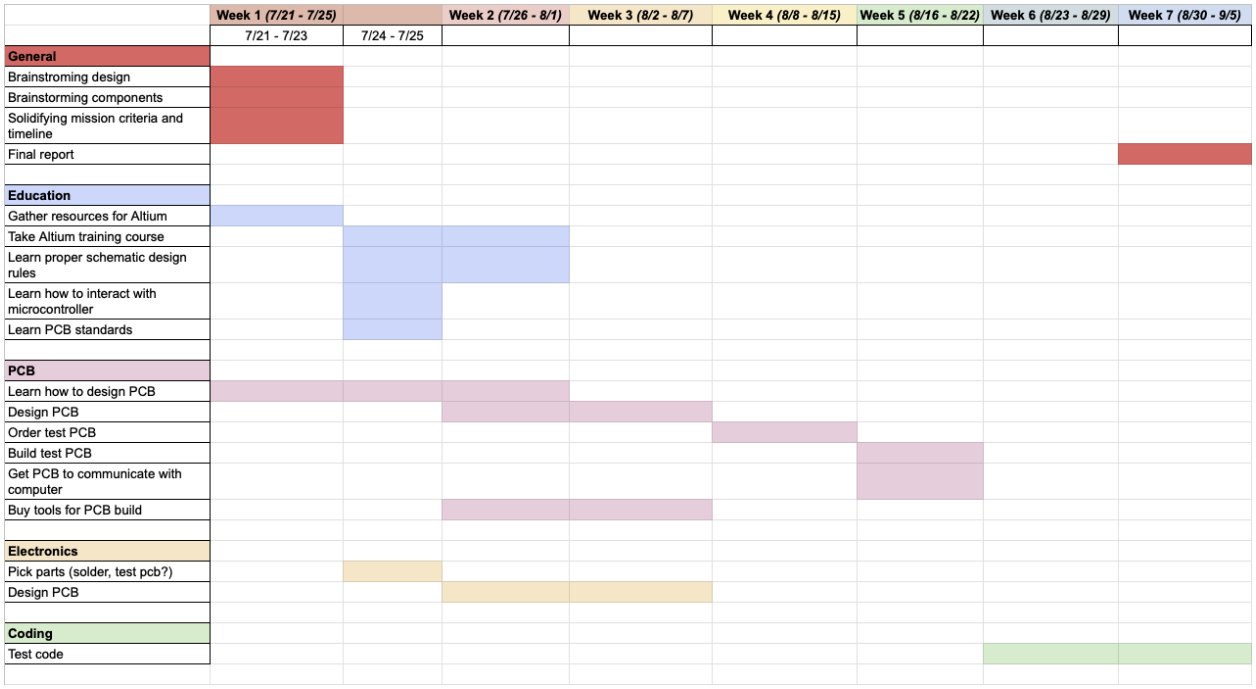


Figure 1: Gantt chart

As illustrated in the chart, I didn't have enough time to receive the PCB, build it, and then go through another full round of redesign and reordering if something went wrong. Because of this, I had to be especially cautious during the design phase to minimize the risk critical mistakes.

I chose to dedicate a significant portion of the first week to gathering resources online to give myself a solid foundation to start working on. I focused on learning how to use Altium Designer, how to choose a suitable microcontroller unit (MCU), and how to program it. This preparation helped me avoid early mistakes in the design phase, like misunderstanding how to establish communication with the microcontroller over USB-C.

Schematic Design and Component Selection

Mission Criteria

Before designing the schematic overview of the system, I first decided on the mission criteria:

- All components must be specced for low power consumption.
- The microcontroller must be programmed through USB C and have debugging capabilities.
- The microcontroller must be able to send telemetry data back through USB C.
- The circuit must have flash memory.
- The circuit must be able to be powered with a 1S LiPo battery.
- The circuit must have 1 peripheral.
- The circuit must communicate through SPI.
- The circuit must have LED indicator(s).

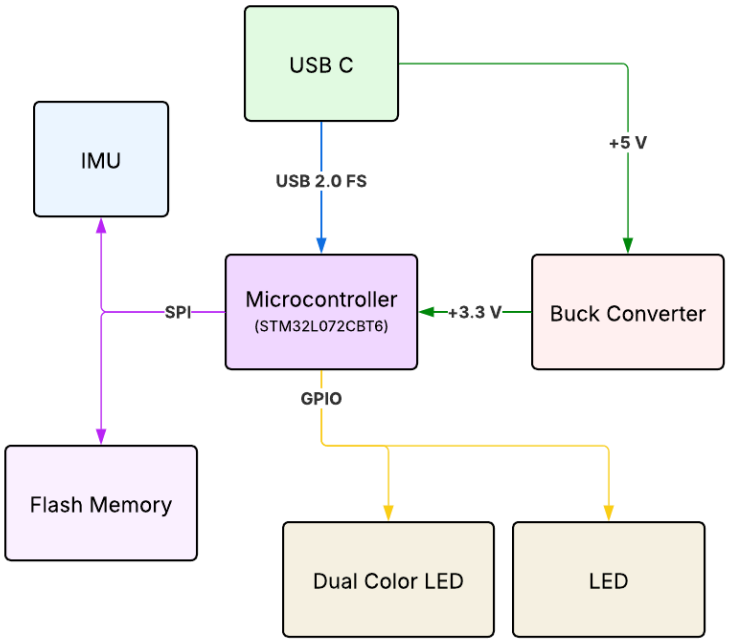


Figure 2: Flowchart of electronics design

Figure 2 shows a flowchart illustrating the design flow of the circuit based on the mission criteria. With these requirements in mind, I began to design the circuit in Altium Designer, splitting the schematic into 4 sections: Power, Peripherals, Data Logging, and Microcontroller.

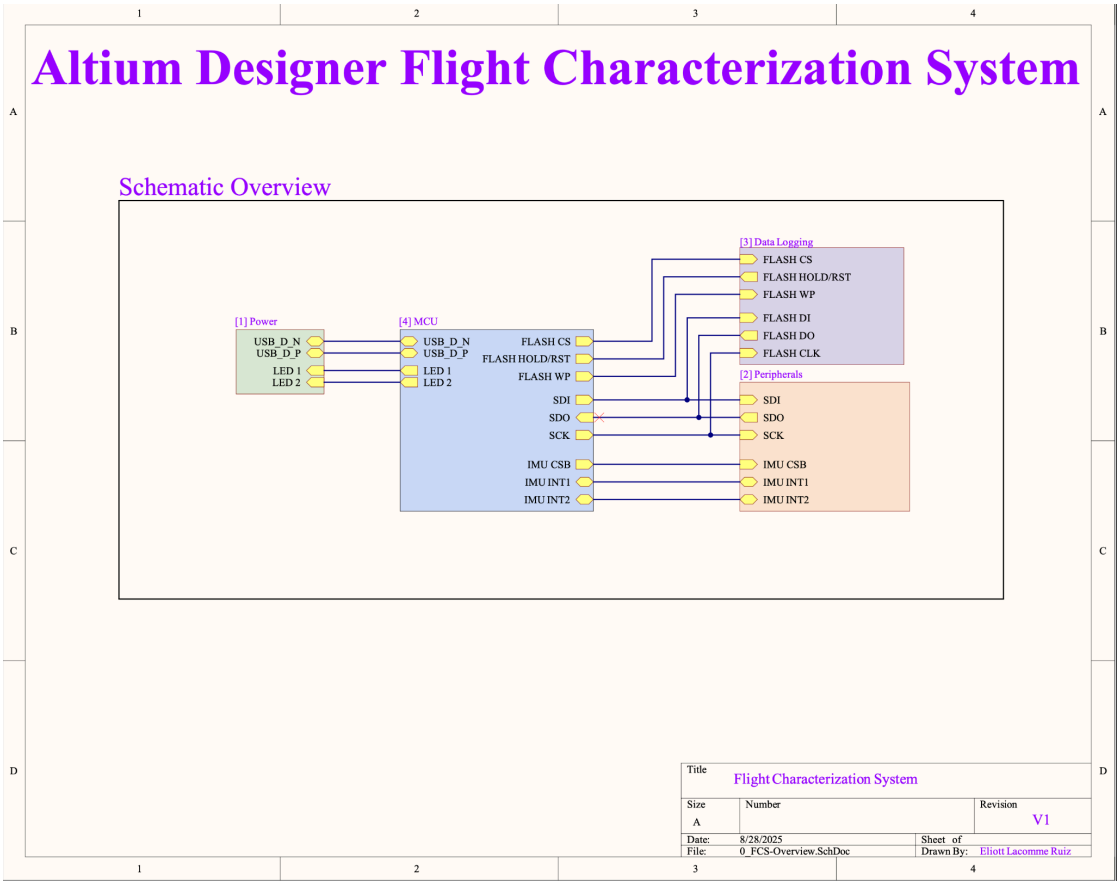


Figure 3: Schematic overview of the circuit

Figure 3 shows a schematic overview of the circuit in more detail.

(Sidenote: I won't be referring to exact component parts in this document, but they can be found in the BOM subsection under the PCB Construction section.)

Power

Power is one of the most critical aspects of the circuit. A poorly designed power system can introduce noise into the supply, which may distort measurements or even cause the entire circuit to fail.

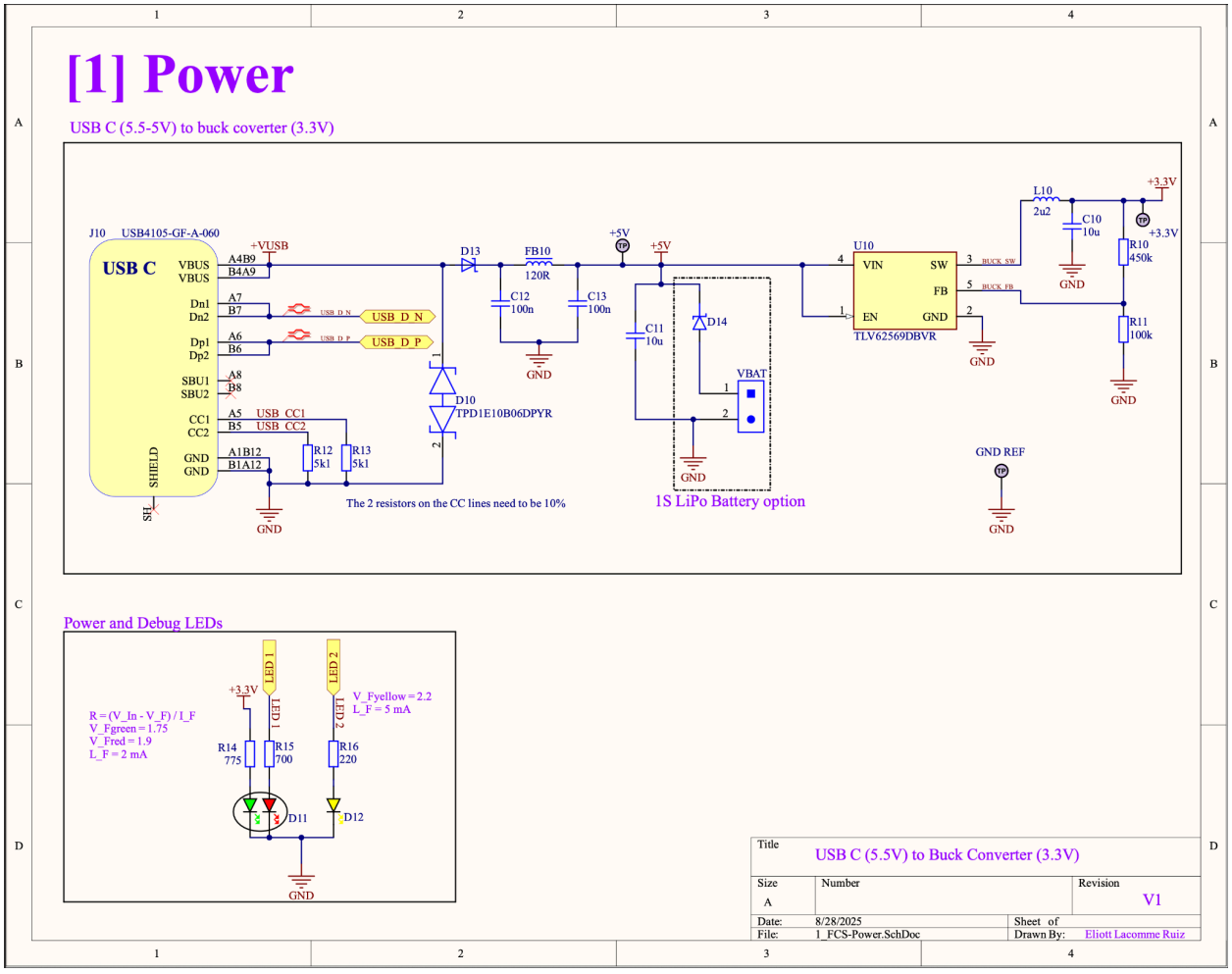


Figure 4: Power Schematic

As shown in Figure 4, the schematic includes a USB-C port, a TVS diode, a ferrite bead Pi filter, an option for a 1S LiPo battery, diodes, and a buck converter. In the following section, I will walk through each of these components and explain how I selected their values, starting from the left side of the circuit and moving toward the right.

USB C

As USB-C continues to become the universal standard, I wanted to incorporate it into my circuit. When selecting a USB-C connector, it's important to understand the differences between their various types and how they are configured. Microchip's AN1953 documentation provides an excellent overview of the history of USB and explains how USB-C ports are selected and initialized.

The most important feature of USB-C is its ability to carry both power and data. USB-C connectors can have anywhere from 6 to 32 contacts (pins), with more contacts allowing for higher data rates. The number of contacts required depends on the USB standard being implemented (e.g., USB 2.0, USB 3.0, USB 3.1). For this project, I selected a 16-pin USB-C connector, which provided more than enough pins to support both power and data transfer.

Equally important is the cable itself. Not all USB-C cables are capable of handling both power and data as some are power-only cables, so I made sure to use a cable that supported both.

Looking at the pin configuration, the CC (Configuration Channel) pins are critical. Depending on how resistors are connected to CC1 and CC2, the port can be configured as either a Downward-Facing Port (DFP) or an Upward-Facing Port (UFP). The DFP (Host) provides power, initializes communication, and receives data (e.g., laptops, desktops, chargers). The UFP (Device) consumes power and sends data (e.g., keyboards, mice, speakers).

In my case, I wanted the port to function as a DFP, so following the AN1953 documentation, I used two 5.1 kΩ resistors with 1% tolerance (10% would have been sufficient, but the tighter tolerance parts were the same price).

For data transmission, the D+ (USB_D_P) and D– (USB_D_N) lines are used as differential pairs to the microcontroller. Differential pairs have strict design rules, but the most important requirement is that the traces are routed to the same length (within a small tolerance) to avoid signal mismatch.

Finally, because USB-C connectors are symmetrical, I tied the redundant pin pairs together. This ensures proper operation regardless of which way the cable is inserted.

TVS Diode

A transient voltage suppression (TVS) diode is a protective component used to guard against electrostatic discharge (ESD) and other transient voltage spikes. During an ESD event, a short-duration but very high voltage spike can occur, which may damage sensitive components.

A TVS diode responds almost instantly in such cases. When the voltage exceeds its breakdown threshold, the diode switches into a low-impedance state (effectively acting like a short circuit), clamping the voltage to a safe level. This prevents the spike from reaching and damaging voltage-sensitive parts of the circuit.

TVS diodes are most effective when placed at points where the circuit could be exposed to external interference or human interaction like USB ports, external pin headers, or other connectors where touching or plugging/unplugging devices could generate surges.

From a specification standpoint, a TVS diode has a working voltage, which is the maximum continuous operating voltage it can handle before turning on. This value should be chosen slightly above the expected supply voltage to avoid accidental triggering from noise in the supply during normal operation. The clamping voltage defines the maximum voltage the diode will allow during an ESD discharge event.

For this design, I selected a TVS diode with a 5.5 V working voltage and a 10 V clamping voltage, which provided appropriate protection for my circuit.

Schottky Diodes

Schottky diodes are widely used to prevent accidental reverse current flow in circuits. A common example is in motor driver circuits. For example, when a motor is turned off, its inertia causes it to keep spinning briefly. In this state, the motor acts like a generator, sending current back into the circuit in the reverse direction, called flyback current. This reverse current can damage polarized or sensitive components if left unchecked.

The two most important characteristics of a Schottky diode are its forward current rating and forward voltage drop. The forward current specifies how much current the diode can safely conduct continuously. The forward voltage is the voltage drop across the diode when it is conducting. This drop is critical to consider in low-voltage systems, since too large a drop can pull the supply below the required operating threshold.

For this design, I selected a Schottky diode rated for 2 A forward current with a 330 mV forward voltage drop, which balanced efficiency with adequate current handling.

Filter

The filter formed by the two capacitors and a ferrite bead is known as a Pi filter (named because its shape resembles the Greek letter π). The purpose of this circuit is to reduce high-frequency noise on the USB power supply line.

Ferrite beads are a somewhat tricky subject. Their impedance varies with frequency, meaning their effectiveness depends on both the characteristics of the power supply and the noise frequencies present. I studied Analog Devices' AN-1368 application note, which does a good job of breaking down ferrite bead behavior, but I still found the topic fairly complex, and had trouble finding clarifying information online.

Since I wasn't ready to fully dive into all the nuances of bead selection, I leaned on a trusted source. I based my capacitor and ferrite bead choices on the recommendations of Phil's Lab, a well-regarded electronics creator on YouTube, whose suggested values matched the currents and voltages I was working with.

Battery Option

I included a battery option in the circuit to allow an external 1S LiPo battery to power the board for wireless operation. A single-cell LiPo has a voltage range of 4.35–2.7 V, which is well suited for the buck converter. To keep the battery healthy and avoid damage, however, it should never be discharged below about 3.2–3.0 V. Since my design does not include built-in over-discharge protection, I monitored the battery voltage manually during testing. Adding proper protection circuitry would be an important improvement for future iterations.

I also chose to bypass the pi-filter when using the battery, since a LiPo cell already provides a relatively clean and low-noise power source for the buck converter. To prevent potential damage in the case where both the USB port and the battery are connected simultaneously, I added a Schottky diode in series with the battery input. This ensures current cannot flow back into the battery, protecting it from reverse current.

It's worth noting that LiPo batteries require careful handling. Over-discharging, puncturing, or overcharging can cause them to swell, degrade rapidly, or even pose a fire risk. For that reason, following safe charging and storage practices is essential when working with LiPo-powered designs. I will not discuss these here, but there are plenty of resources available online, especially in the micro drone or car racing communities.

Buck Converter

A buck converter is an integrated circuit (IC) used to step down an input voltage to a lower, regulated output voltage. In addition to voltage conversion, it helps stabilize noisy supply rails. Inside the IC, a feedback network monitors the output and adjusts the duty cycle of the internal switch to maintain the desired voltage.

As shown in Figure 4, the output voltage is set using a voltage divider connected to the feedback pin. The equation provided in the datasheet relates the output voltage to the divider resistors and the feedback regulation voltage (V_FB):

$$V_{OUT} = V_{FB} \times \left(1 + \frac{R1}{R2}\right) = 0.6V \times \left(1 + \frac{R1}{R2}\right)$$

Where V_FB is the feedback regulation voltage (given in the datasheet), and R1 and R2 are resistors. In my design, R2 was the more constrained resistor, so I selected 100 kΩ for R2 and then solved the equation for R1.

The output capacitor and inductor combination was chosen from the datasheet's recommended tested values. Following these guidelines improves the likelihood that the converter will behave as intended without introducing instability.

It's also critical to size the inductor properly. The formula below can be used to solve for the inductor's inductance.

$$I_{L,MAX} = I_{OUT,MAX} + \frac{\Delta I_L}{2}$$

$$\Delta I_L = V_{OUT} \times \frac{1 - \frac{V_{OUT}}{V_{IN}}}{L \times f_{SW}}$$

where:

- $I_{OUT,MAX}$ is the maximum output current
- ΔI_L is the inductor current ripple
- f_{SW} is the switching frequency
- L is the inductor value

The inductor must be able to handle the maximum expected current draw of the system without saturating. While the datasheet often provides proven inductor recommendations, I double-checked that the chosen inductor could safely support my circuit's current requirements.

Debugging LEDs

To make debugging less frustrating, I added indicator LEDs to provide quick visual feedback when something goes wrong in the code or when certain events occur. For this design, I used three LEDs: red, yellow, and green. The red LED is tied directly to the power rail and serves as a power indicator, letting me know when the circuit is active. The yellow and green LEDs are connected to GPIO pins and controlled by the microcontroller, giving me flexible options to signal specific events, errors, or states during operation.

All of the LEDs are designed for low-power consumption, and their most important specifications are the forward voltage and forward current. The forward voltage is the minimum voltage required to turn the LED on, while the forward current determines how much current will flow through the LED once that voltage is applied, as well as how bright it will appear. Since LEDs cannot be driven directly from the supply without risking damage, a series resistor is required to limit the current. To properly limit the amount of current that passes through the LED, we can use the following formula:

$$R = (V\_in - V\_F) / I\_F$$

Where R is the resistance of the resistor, V_in is the input current, V_F is the forward voltage, and I_F is the forward current. This formula is derived from using Kirchoff's laws when an LED is in series with a resistor and a power supply. By choosing appropriate resistor values, I ensured that each LED operated within its safe range.

Microcontroller

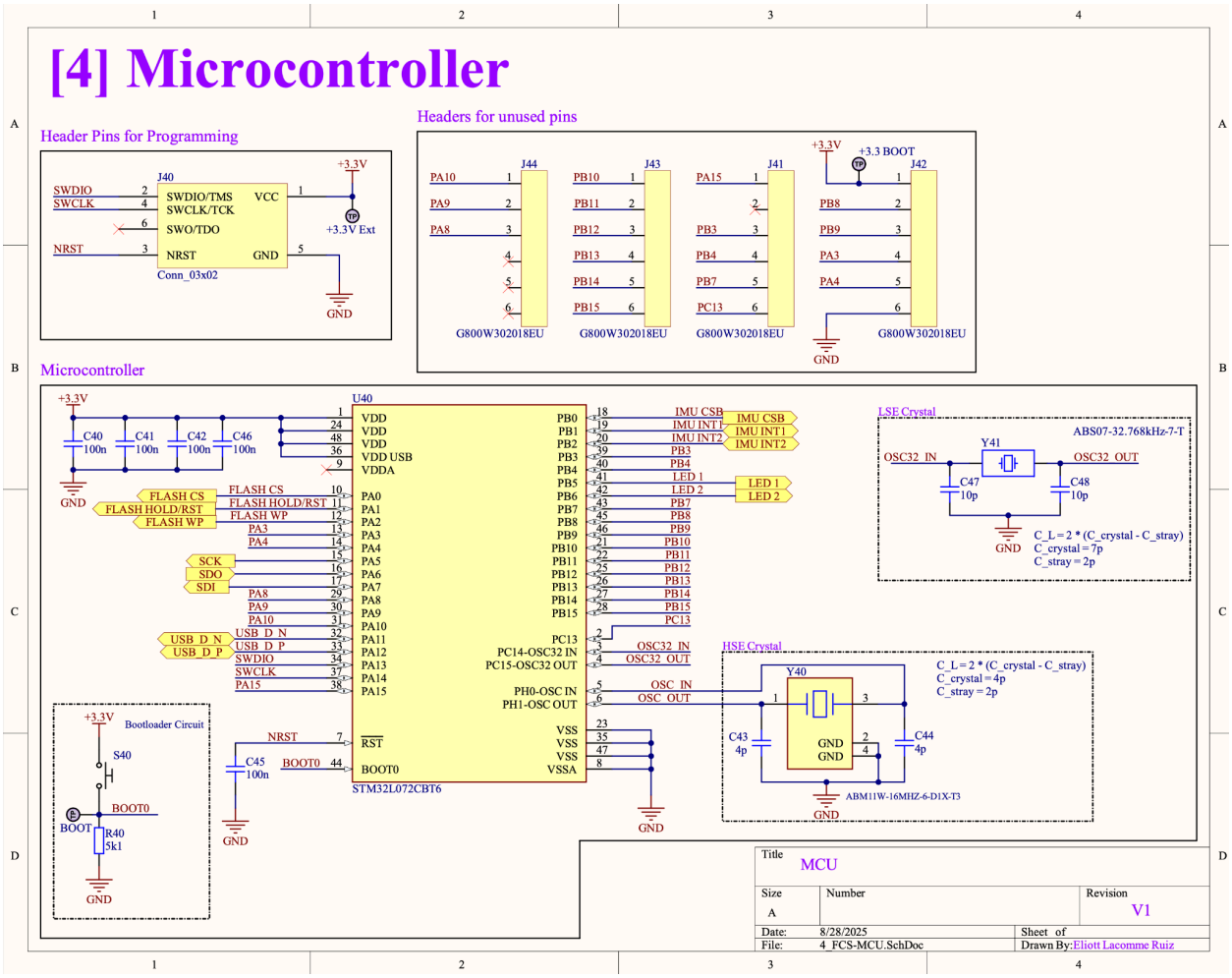


Figure 5: Schematic of microcontroller unit

Microcontroller Selection

Microcontrollers are the brain of a circuit. They handle communication with peripherals, interpret signals, and send data back to a host device. They are powerful yet versatile components, providing a platform for the user to program them to perform specific tasks.

Microcontrollers come in many different types, and I won't go into detail about all of them here. Instead, I'll focus on my thought process when selecting one for this project. Based on my mission criteria, I knew I wanted a low-power microcontroller, which immediately narrowed down the options. I also wanted something well-documented with strong community and manufacturer support. During my first week of research, I focused on the STM32 family, particularly the low-power STM32L series, which is highly regarded for efficiency. I also found STM32CubeIDE appealing because of its beginner-friendly environment and support for external debugging.

Another benefit of choosing the STM32 family was the availability of STM32 Nucleo boards, which serve as development platforms. I used the downtime between ordering my custom PCB and receiving it to practice programming on a Nucleo board from the same ultra–low-power family. This gave me valuable experience before working directly with my custom hardware.

After selecting a microcontroller family, I narrowed my search down even further. I needed a microcontroller with ample program memory, the ability to operate at the voltages provided by my buck converter, and plenty of GPIO pins to allow for breakout-board prototyping in the future. A reasonably high maximum clock frequency was also important, since I planned to pair it with an external high-speed oscillator (covered in the next subsection). Finally, since I was still learning surface-mount soldering, I wanted a package that wasn't too difficult to handle with a stencil.

Crystals

Clocks are used to provide a stable timing reference, and the same principle applies to crystal oscillators. A crystal oscillator is a small piece of material (commonly quartz) that vibrates at a precise frequency when voltage is applied. While most microcontrollers include built-in oscillators, these typically suffer from higher drift. For this reason, external crystal oscillators are often used when precise timing (MHz) is required.

In my design, I selected two crystal oscillators. One of them was a High-Speed External (HSE) 16 MHz crystal, which provides the system clock used by the MCU to process data, and the other was a Low-Speed External (LSE) 32.768 kHz crystal, primarily used for Real-Time Clock (RTC) functionality.

It is critical to select oscillators that are compatible with the chosen microcontroller and to properly size their load capacitors. To guide this process, I followed ST's AN2867 application note. Load capacitors are connected to each side of the crystal and help stabilize its oscillation frequency. Their value can be determined using the equation:

$$C_L = \frac{C_{L1} \times C_{L2}}{C_{L1} + C_{L2}} + C_s$$

Where C_L is the load capacitance, C_L1 and C_L2 are the external capacitors, and C_S is the stray capacitance. The stray capacitance is the capacitance of the traces and the pads on the board, and the load capacitance is specified by the crystal manufacturer. This is hard to measure, but when keeping the traces as short as possible, it is safe to assume around a 2-4 pF stray capacitance. For BOM consolidation, I set C_L1 = C_L2 and solved for the external capacitor.

To confirm that our crystal can start up properly, it is necessary to calculate the oscillator transconductance. To make an oscillator reach a stable phase, the oscillator must provide sufficient gain to compensate for any losses and be able to provide the energy for the oscillation buildup. More information on this can be found in the AN2867 application note. The gain

margin ratio is the ratio between the oscillator gain and the oscillation loop critical gain. This ratio must be greater than 5 to ensure that the oscillator will start properly. We can use the following equation:

$$gain_{margin} = g_m / g_{mcrit}$$

Where gain_margin is the ratio that should be larger than 5, g_m is the oscillator transconductance specified in the STM32 datasheet and g_mcrit is the minimal transconductance value required to maintain a stable oscillation.

To solve for g_mcrit, we can use the following equation:

$$g_{mcrit} = 4 \times ESR \times (2\pi F)^2 \times (C_0 + C_L)^2$$

where:

- ESR is the equivalent series resistance
- C_0 is the crystal shunt capacitance
- C_L is the crystal nominal load capacitance.
- F is the crystal nominal oscillation frequency

The ESR is given as a specification of the crystal oscillators. Below, in Figure 6, are my calculations for my selected 16MHz

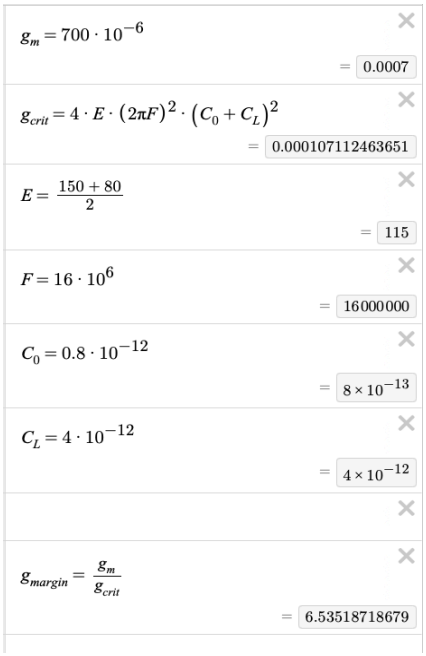


Figure 6: Calculation of the gain margin for 16MHz crystal oscillator

Similar steps can be taken to assure that the LSE works, but those calculations have been omitted.

Peripherals

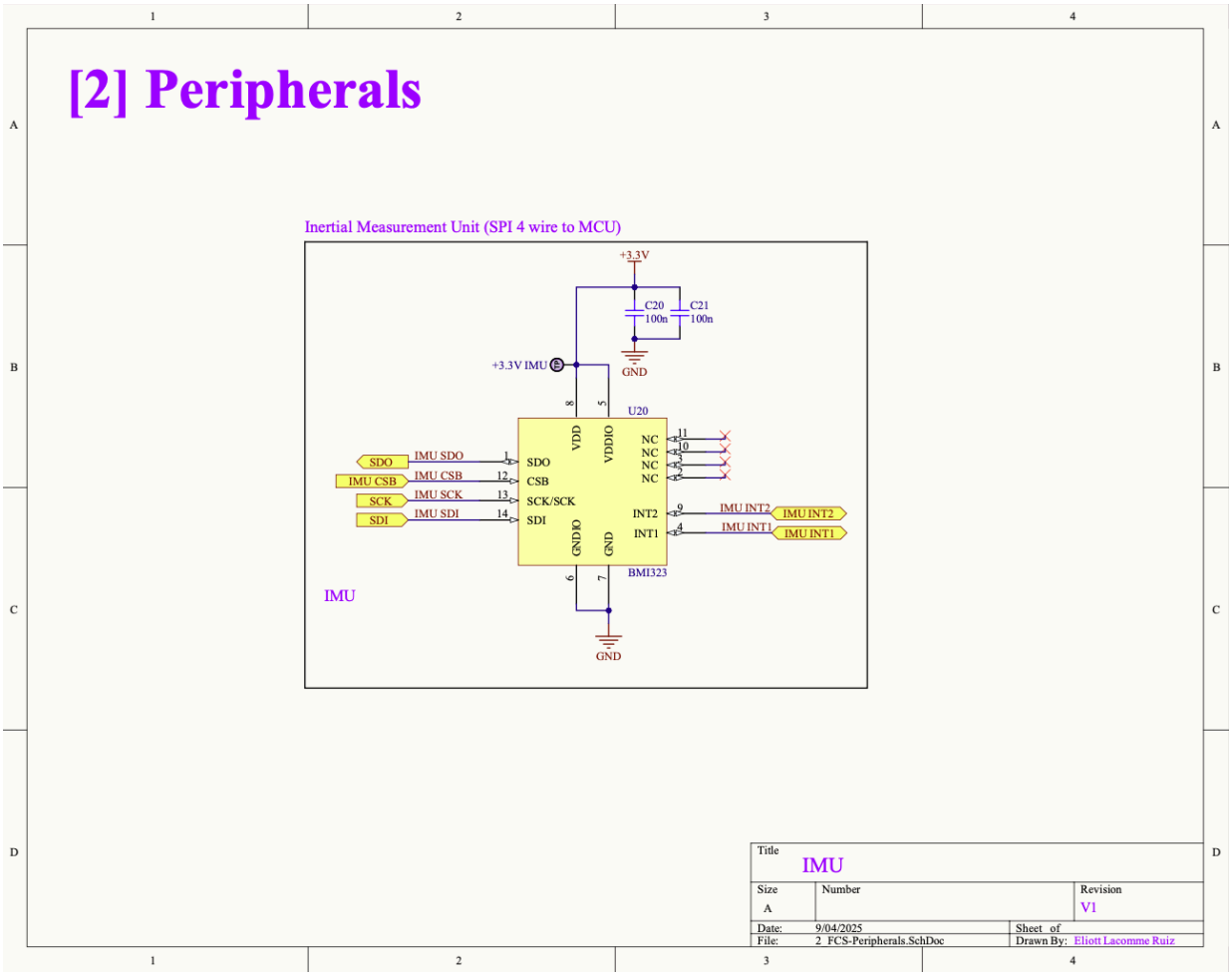


Figure 7: Schematic of the inertial measurement unit

Inertial Measurement Unit (IMU)

An inertial measurement unit (IMU) is an integrated circuit that measures acceleration and rotational velocity (gyroscopic data). IMUs form the core of nearly every motion-tracking device, from wearables and smartphones to automobiles and aircraft. Some models also include a magnetometer, which can improve accuracy by providing orientation data relative to Earth's magnetic field.

When selecting an IMU, it's important to first define what needs to be measured. Some devices only provide accelerometer data, others only gyroscope data, and many combine both (with the option to isolate outputs if needed). Another practical consideration is size as IMUs are extremely small packages, so careful planning is needed for soldering and assembly.

IMUs often use built-in filters and sensor fusion algorithms to combine data from their internal sensors, such as the accelerometer and gyroscope, into more accurate outputs. While the math behind sensor fusion is complex, these features allow the device to deliver stable and precise measurements.

For this project, I chose a sensor from Bosch, a manufacturer known for producing high-precision, reliable sensors that I had successfully used in previous projects. Specifically, I selected the BMI323, which offered ultra–low-power operation and supported communication over the Serial Peripheral Interface (SPI) protocol (explained further in the coding section). As with any peripheral, I referred closely to the manufacturer's datasheet for pinouts, specifications, and operating details.

Data Logging

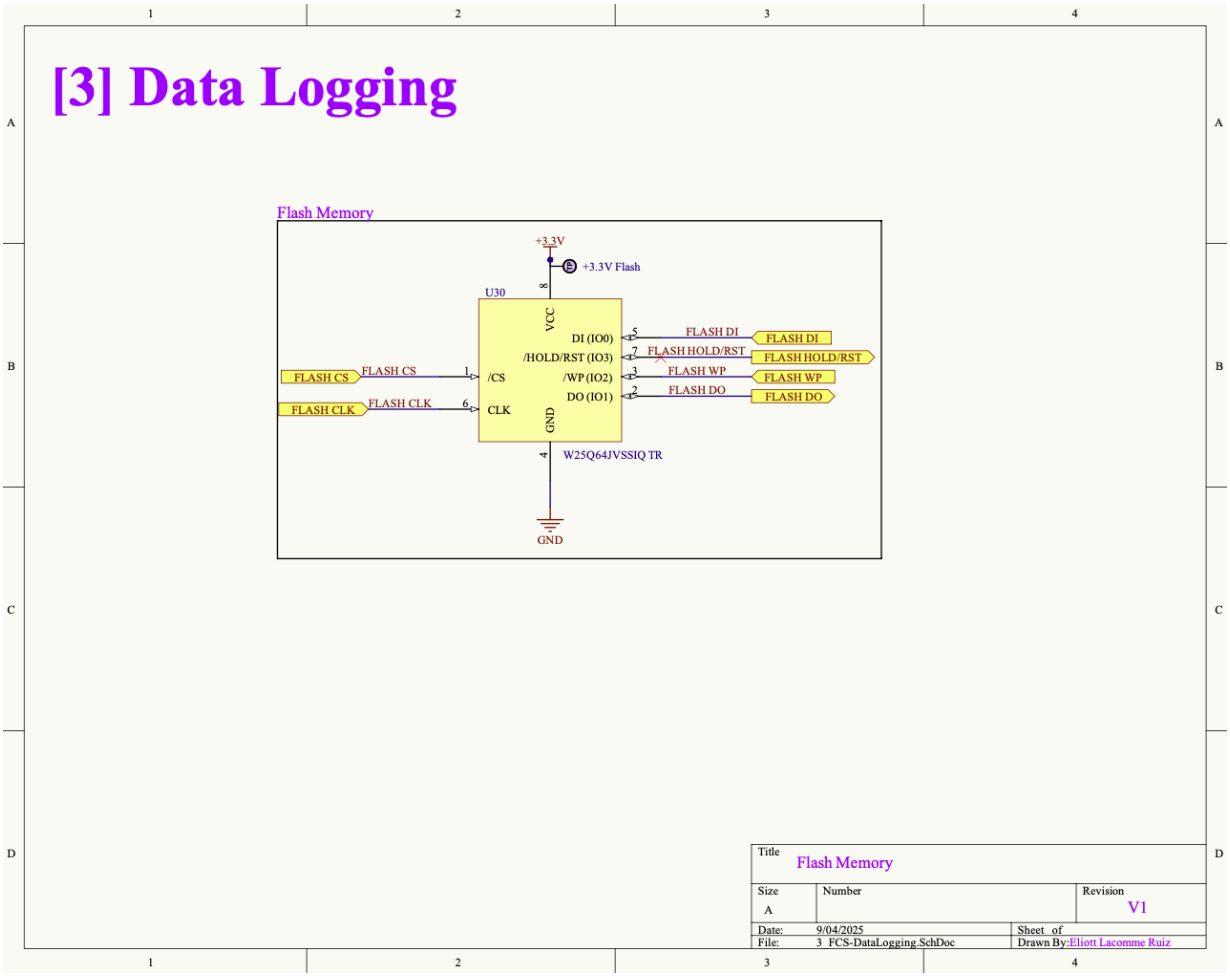


Figure 8: Schematic of the flash memory chip

Flash Memory

External flash memory provides additional storage space for both code and data beyond what is available inside a microcontroller. For this design, I used NOR flash memory, a type of non-volatile memory, meaning data is retained even after power is removed.

The most important characteristics of an external flash chip are its communication protocol, voltage ratings, and storage capacity. Depending on the programming language (C vs. C++) and the number of libraries used while coding, compiled code size can grow significantly. In such cases, the external flash can hold part of the program, which is then loaded into the MCU during startup.

Another critical application of flash memory is data storage. These compact chips are capable of storing large amounts of data at high speeds, which is particularly useful for extended logging sessions. In my system, the external flash provides a fast, reliable way to capture and store sensor data without being limited by the MCU's internal memory.

General Passive Components

Resistors and capacitors are the backbone of any circuit. For resistors, the key specifications are their power rating (how much heat they can safely dissipate) and their physical size. Capacitors, on the other hand, must be rated to withstand the maximum expected voltage in the circuit. A good rule of thumb is to choose capacitors with a voltage rating at least 1.5-2 times higher than the expected operating voltage to provide a safe margin.

Package size is another important consideration. Passive components are available in both imperial and metric sizing standards, and the two can be confusingly similar. In practice, anything smaller than 0603 (imperial) is, in my opinion, difficult to work with by hand. For this project, I stuck to 0603 as my minimum size, which proved to be a comfortable balance between compactness and manufacturability.

That said, I did have a classic "NASA Mars Climate Orbiter" moment, where I accidentally ordered 0603 metric resistors instead of 0603 imperial. The parts were comically tiny, and waiting for new parts nearly derailed the entire project. To put it into perspective, I took a photo under a digital microscope of one of the metric 0603 resistors next to an uncooked grain of rice.

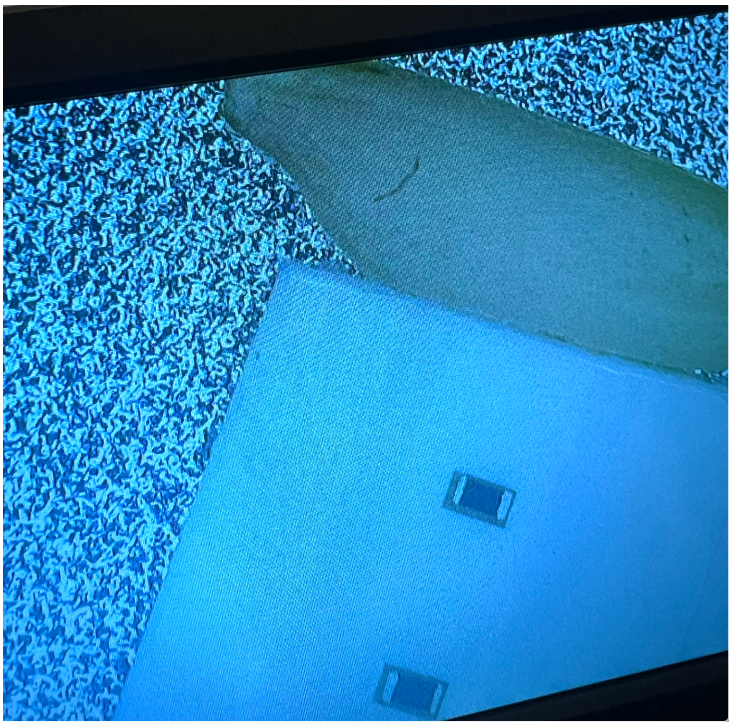


Figure 9: Image of 0603 (metric) resistor next to uncooked grain of rice

PCB Design

With the schematic finalized and all components selected, the next step was to bring the design into the PCB layout stage. I won't cover the details of how to use Altium specifically, but I will highlight the components that required special attention during routing and explain the reasoning behind my design choices. I'll also touch on the general PCB stack-up and the key concepts I found most important to consider.

The first and most critical step is choosing a PCB manufacturer. Two companies I've had positive experiences with are PCBWay and JLCPCB. Regardless of the manufacturer, it's absolutely essential to review their capability specifications and configure your PCB design rules accordingly before starting the layout. This ensures your design stays within manufacturable limits and saves a lot of time that would otherwise be lost in back-and-forth adjustments after submission.

For anyone learning PCB layout, I highly recommend reading David L. Jones' PCB Design Tutorial. Although it was written in 2004 and is somewhat dated, it's still widely regarded as one of the best foundational guides to PCB standards and practices.

Layer Stackup

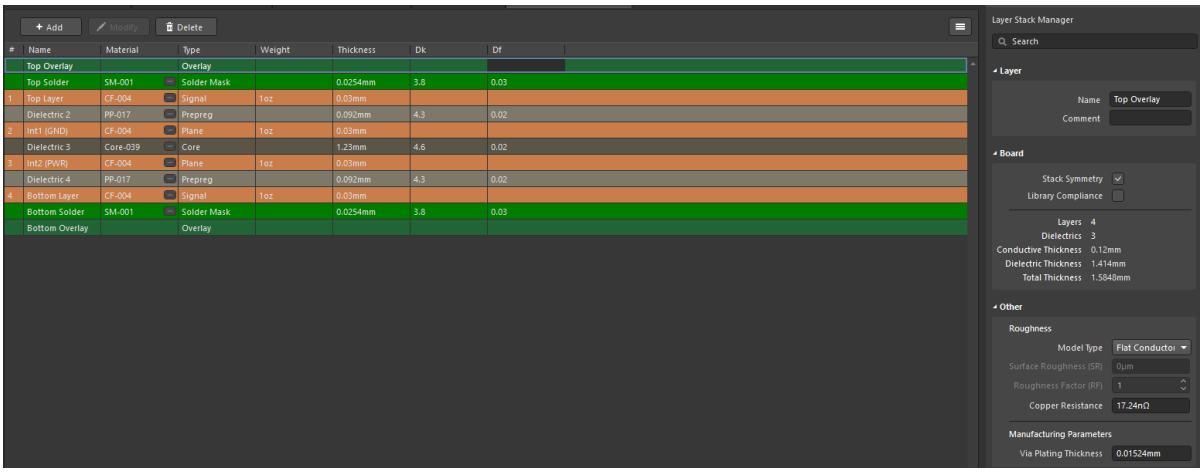


Figure 10: PCB stackup

To raise the difficulty of this project and create a more professional, industry-standard design, I decided to use a four-layer PCB. As shown in Figure 10, the stackup consists of two outer signal layers and two inner plane layers. The inner planes are dedicated to ground and +3.3 V, respectively.

Since all of the components are surface-mount devices (SMD), I wanted a solid ground plane underneath the communication traces. This is critical for maintaining signal integrity and providing well-defined return paths for high-speed signals. For the overall board structure, I selected a standard 1.6 mm thickness, with the specific layer thicknesses and dielectric constants defined by the manufacturer (JLCPCB), as shown below.

USB C Port

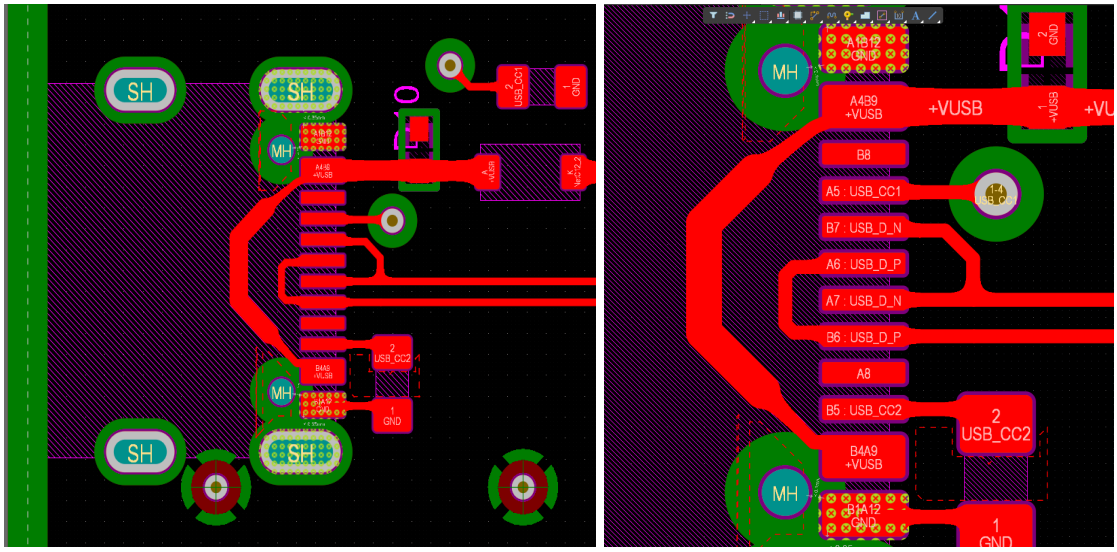


Figure 11: PCB layout of USB C port (left) and closeup (right)

The USB-C port has several layout requirements that must be met for it to function correctly. Because the connector is symmetrical, certain pins are duplicated and must be tied together, such as the USB_D_P and USB_D_N lines. These lines form a differential pair, meaning they carry equal and opposite signals. If one line has a +5 V signal, the other one has a -5 V signal. Differential pairs provide strong noise immunity because external interference tends to couple equally into both traces, while the opposing signals cancel out any induced noise or magnetic fields. The traces must be routed carefully. They should be the same length so that signals arrive simultaneously at the receiver, since even a 1–2 mm mismatch can cause timing errors and misalignment of data. The traces must also maintain the same width, as wider or narrower sections can create impedance mismatches that degrade the signal. Finally, the differential pair should ideally be routed close together and in parallel, which helps maintain consistent impedance and minimizes noise pickup.

Buck Converter

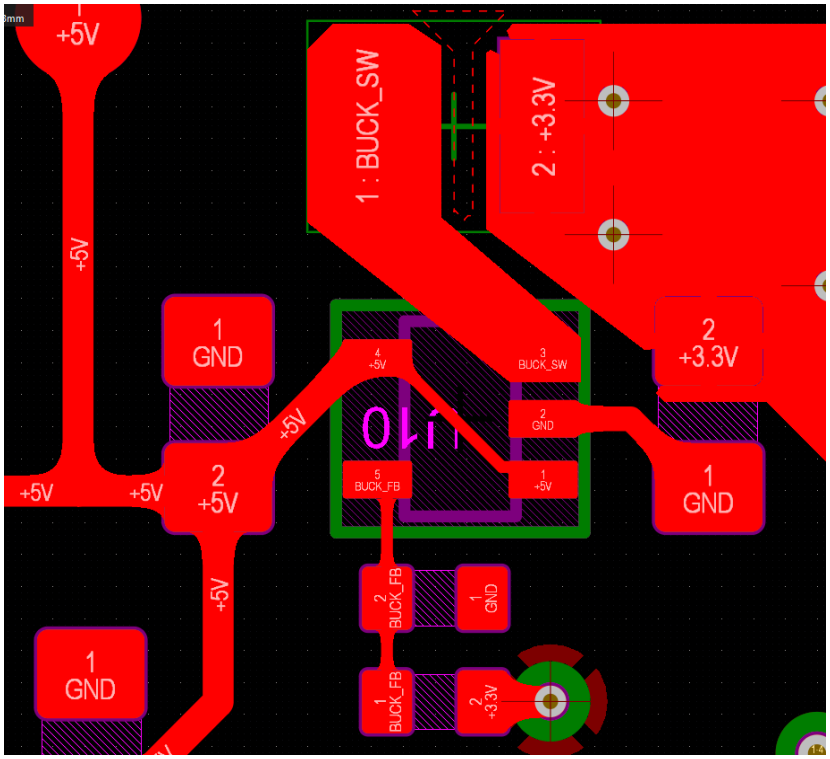


Figure 12: PCB layout of buck converter

Buck converters are considered one of the more complex ways of reducing voltage because of their sensitive high-speed feedback networks. By design, a buck converter is a switching regulator, meaning it regulates voltage by turning the input supply on and off at high frequencies, typically in the MHz range. Because of this switching method, the converter relies on its feedback pin (BUCK_FB in Figure 12) to maintain a stable duty cycle and ensure the output remains regulated. Manufacturers usually provide detailed layout guidelines, and while the specifics vary, they share common rules. These include rules such as maintaining a solid ground plane and keeping the current loops between components as small as possible. Minimizing loop area reduces trace impedance and allows the buck converter to operate more stably. In my layout, the capacitors, inductor, and resistors were placed to form these small loops, as shown in the figure.

I also used copper pours between traces to improve signal integrity and reduce overall impedance. On the right side of the layout, the +3.3 V output is tied into a group of vias that connect the top signal pour to the internal power plane. Using multiple vias is important, since each via has a finite current-carrying capacity. The more vias included, the less current each must conduct, which reduces impedance and heat buildup. Because the +3.3 V rail was my main source of power for the entire circuit, I wanted to ensure this connection could handle up to 1.5 A comfortably. A common rule of thumb is that a standard 1 oz plated via can carry around 1 A, so by placing seven vias I provided more than enough margin. While it's possible to calculate the exact current distribution and temperature rise per via, I chose to simply overspec the number of vias as a safe and reliable approach.

Decoupling Capacitors

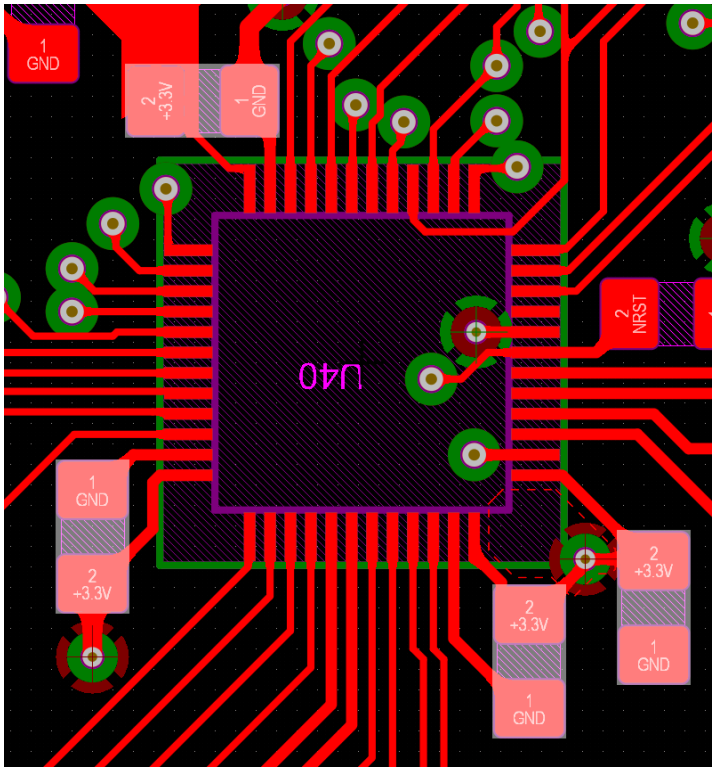


Figure 13: PCB layout of MCU with decoupling capacitors highlighted

Decoupling capacitors are small-value capacitors (0.1 µF in my design) placed as close as possible to the power pins of microcontrollers and sensors to keep the supply voltage clean and stable. When an MCU or sensor draws short bursts of current, such as when toggling GPIOs or transmitting data, the sudden demand can cause voltage dips or noise on the supply rail. The decoupling capacitor acts as a tiny local reservoir, instantly supplying current to smooth out these fluctuations and prevent disturbances from propagating through the rest of the circuit. Placement is just as important, and keeping the traces short and the loops small maximizes effectiveness, since it minimizes parasitic inductance and ensures the capacitor can respond quickly.

Vias

In Figure 13, the small green circles represent vias. Vias in a PCB are used to transition traces between layers, which makes routing more compact and allows direct access to internal power and ground planes. There are several types of vias, such as blind and buried vias, but most designs don’t require them and they are much more expensive to manufacture. In practice, the most common solution is a through-hole via, which passes through the entire PCB stackup. These vias can be configured to connect only to specific internal layers, leaving the others unconnected during fabrication. This approach is inexpensive and supported by standard manufacturers like JLCPCB.

Initially, I made the mistake of using blind vias to connect my signals to the power and ground planes. This choice drastically increased the cost of the PCB nearly tenfold and JLCPCB rejected the design since it was outside their manufacturing capabilities. This experience reinforced the importance of checking design rules and carefully reviewing a manufacturer's specifications, especially their via size limits and supported via types, before finalizing the layout.

Stitching Vias

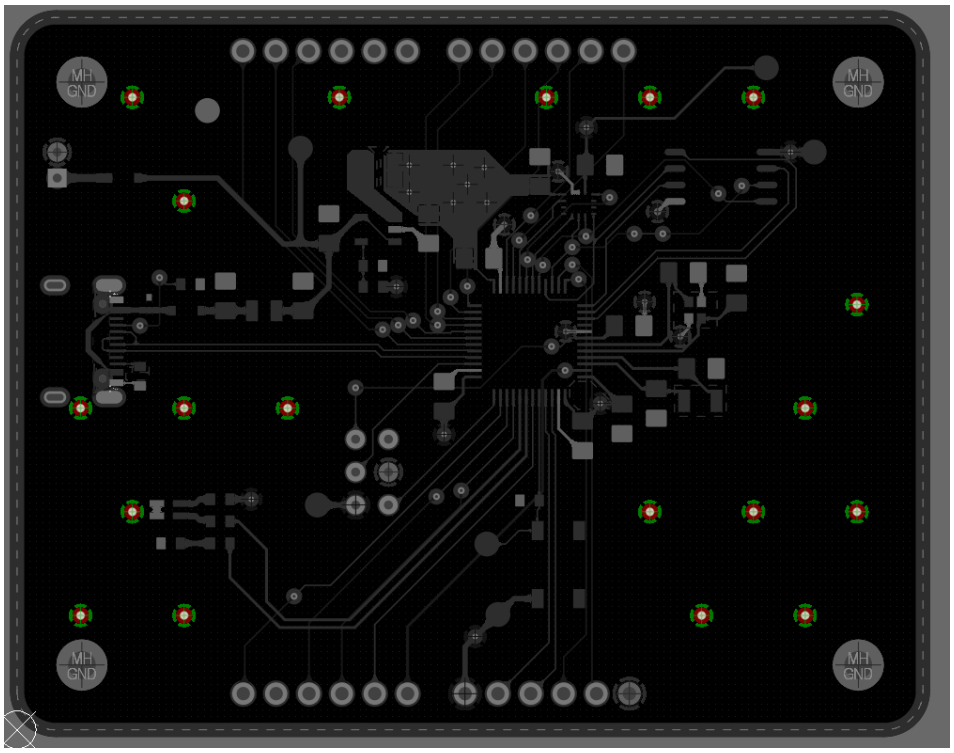


Figure 14: PCB layout of whole board with stitching vias highlighted

Stitching vias are used to connect large copper pours to the ground plane, effectively "stitching" the layers of the PCB together. It is common practice to add a copper pour in any areas of the board not occupied by traces, and stitching vias ensure that this copper is electrically tied to the selected inner plane. In my case, I tied the stitching vias to the inner ground plane. A useful way to visualize this placement is like a checkerboard pattern, with a via at the center of every dark square.

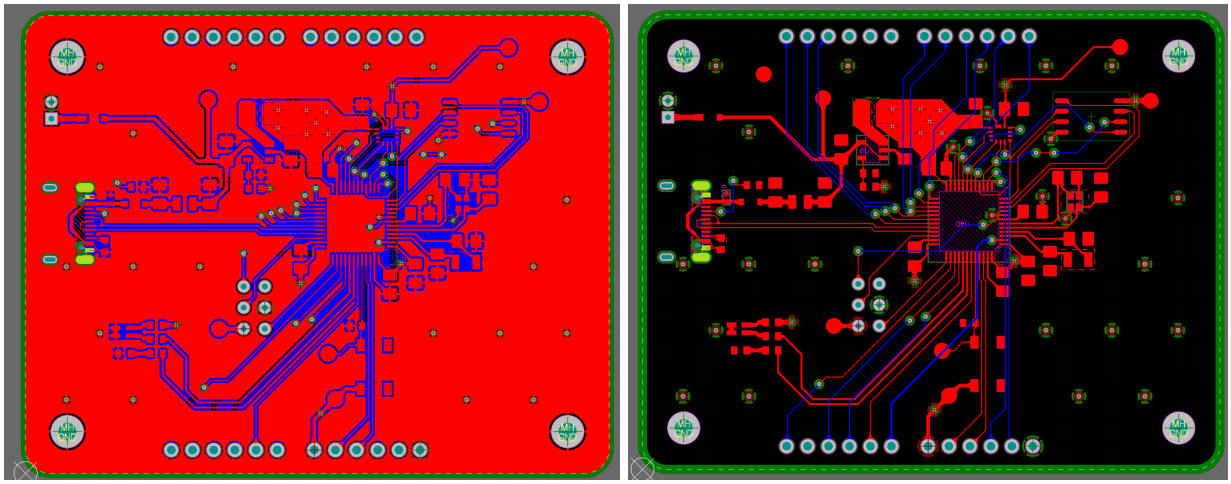


Figure 15: PCB layout of whole board with top copper pour (left) and without (right)

In addition to maintaining a solid ground reference, copper pours stitched with vias help reduce the amount of copper removed during manufacturing, as shown in Figure 15, where all of the black part of the PCB is the copper that must be removed from the plane. Adding the ground copper pour provides a continuous ground reference for signal return paths, improves thermal conductivity by allowing heat to spread into other layers, and reduces crosstalk between signals.

One note is that Altium's automatic stitching via placer can sometimes position them too close to board edges or under integrated circuits. Before finalizing the layout for manufacturing, it's important to review the placement carefully and adjust as needed.

Stencil

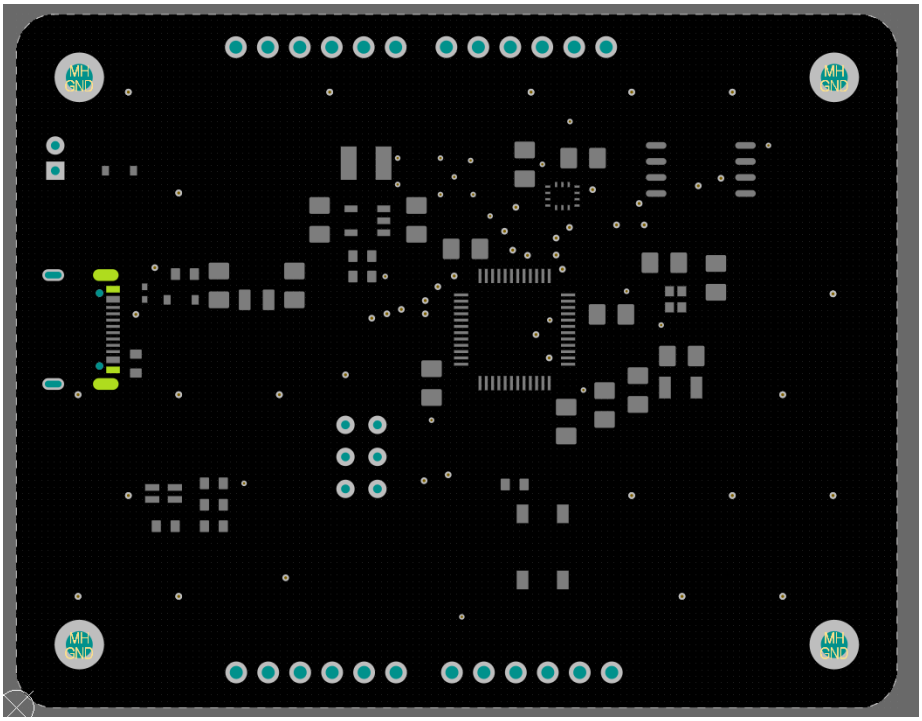


Figure 16: Stencil layout

For this project, I chose to assemble the PCB myself rather than have it professionally assembled by JLCPCB. I wanted the experience of working with stencils directly, since learning this process would be valuable for future projects.

A stencil is a thin sheet of metal with precisely cut openings that align with the exposed copper pads of the PCB (images in the PCB Construction section). Its purpose is to control the application and location of the solder paste. When the paste is spread across the stencil with a scraper, it fills only the openings and deposits the exact amount of solder needed for each pad.

It's important to ensure when designing the stencil openings that they match only the component pads and do not overlap with test pads or through-hole pads, since excess paste in those areas could interfere with assembly or testing. This can be done in software by removing the pads and through hole pads from the solder paste layer.

Test Points

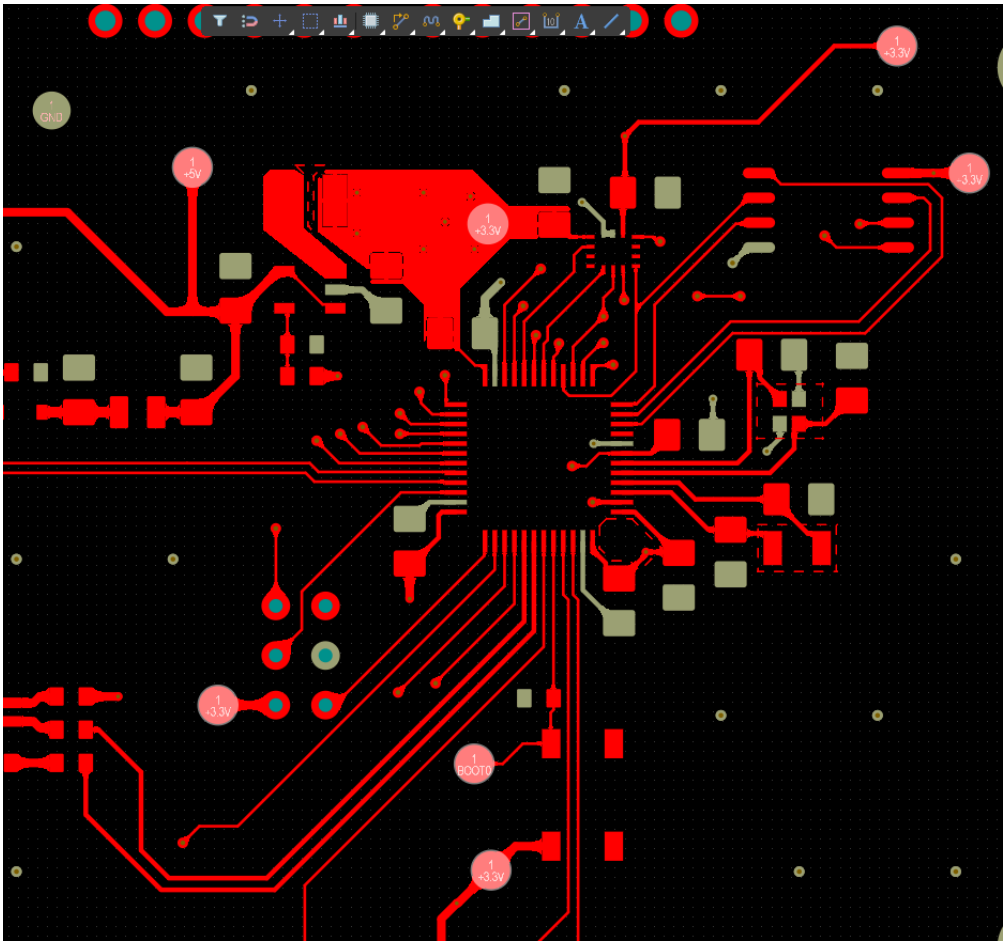


Figure 17: PCB top layer layout with test points highlighted

Test points are a simple but essential addition for debugging a circuit (Figure 17). They are essentially exposed copper pads without silkscreen or components placed on top, sized so that probes can make reliable contact. While they don't contribute directly to circuit function, they are invaluable for troubleshooting, since they provide easy access to measure voltages, signals, or continuity without having to touch delicate component leads.

In this design, I added test points for every main power input to the peripherals and flash memory, as well as for key pins on the MCU. These additions paid off during debugging, as they helped me identify an issue with the BOOT pin on the MCU and verify the output voltage of the buck converter. For reference, all test points in the schematic are labeled with the prefix "TP", as noted in the Schematic Design and Component Selection figures.

Full Design

Below are some images of the full design of the PCB, renders, and the actual PCB itself.

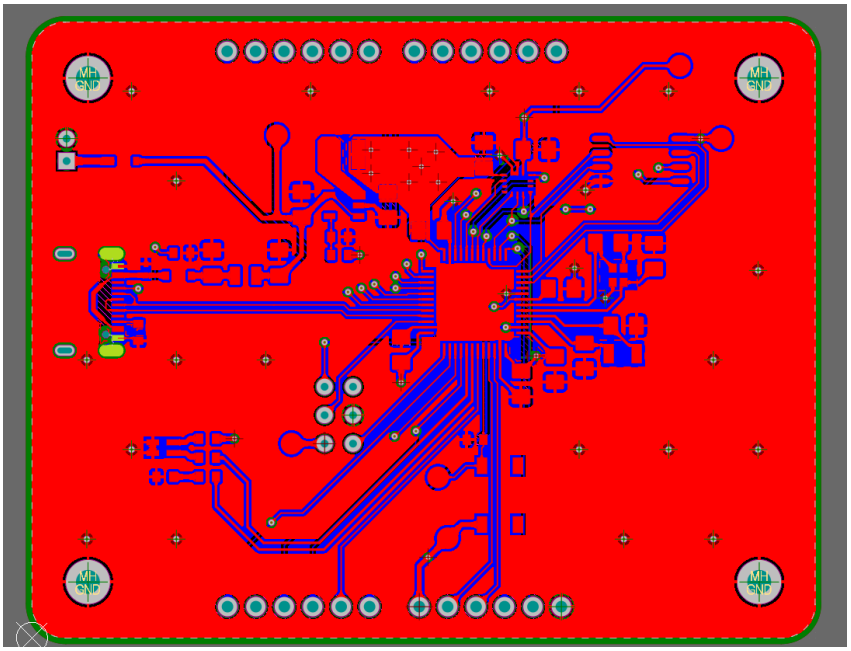


Figure 18: All PCB traces

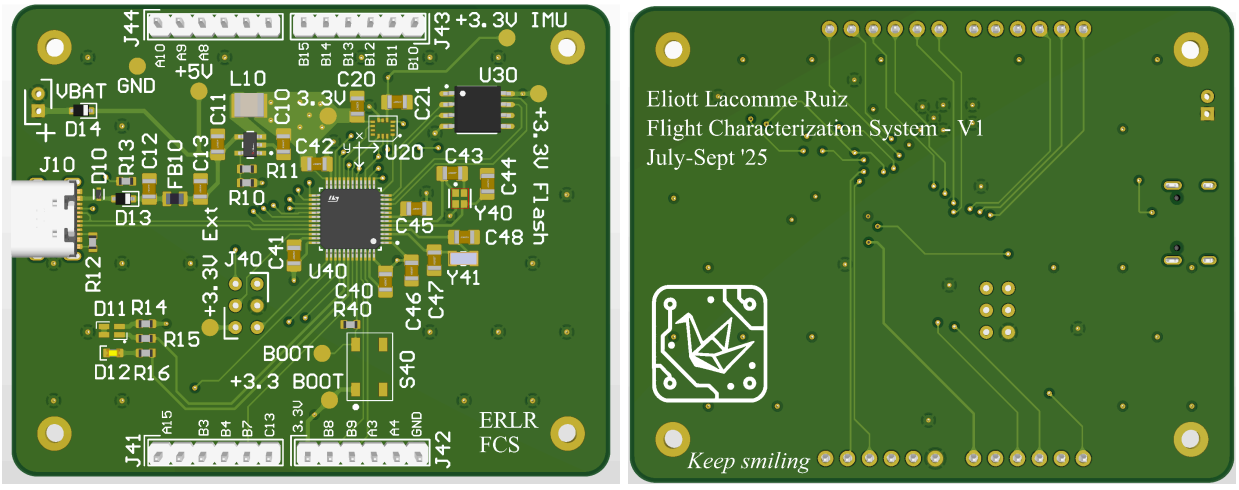


Figure 19: 3D PCB render of top (left) and bottom side (right)

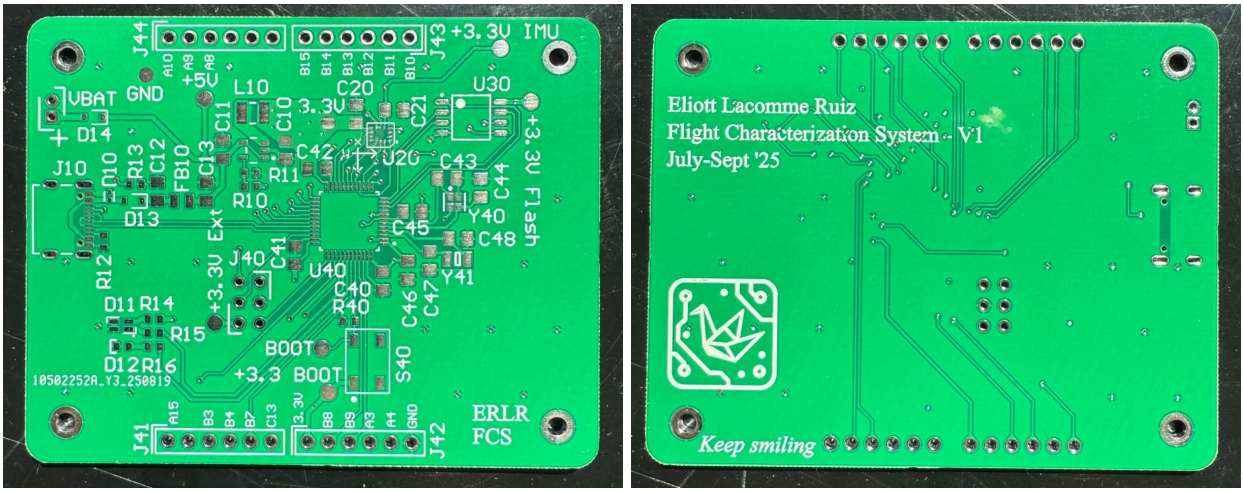


Figure 20: Image of real PCB

PCB Construction

PCB construction involves many steps that need to be carried out carefully, as everything from the material properties of the solder paste to the heating profile of the hot plate can affect the final outcome. Attention to detail in each stage is essential to ensure reliable assembly and functionality.

In the following section, I will walk through the exact steps I took when building my circuit, highlighting both the methods I used and the lessons I learned along the way.

Solder Paste

Technical Parameters

Heating Rate	Time Required for Reaching 110ºC	Constant Temperature 110 - 138ºC	Peak Temperature	210±10°C	Cooling Rate
1-3 ºC/sec	< 60-90 seconds	60-100 seconds	175 ±5ºC	30-60 seconds	< 4ºC / S

Storage, Operation and Storage Life

1. Once receiving the Sn64Bi35Ag1 mid temperature lead free solder paste, our clients had better put this product into a refrigerator for storage. We suggest that the solder cream should be kept in cold storage at a temperature range from 2 degrees Celsius to 8 degrees Celsius. The warranty period is 6 months from the date of production.

Figure 21: Solder paste heating profile

Unlike soldering through-hole components with wire solder from a spool, SMDs are more easily soldered using solder paste. Solder paste is a mixture of tiny solder balls suspended in flux, and it comes in several grades and material compositions that differ in their temperature characteristics, known as the heating profile. To achieve reliable solder joints, the paste must be heated according to a specific profile of time and temperature.

For this project, I chose unleaded solder paste, which has a relatively low melting point and is safer to handle than leaded solder. The paste I used required heating at specific temperature stages, as shown in Figure 21. To validate the profile, I ran a test on a practice board where one circuit was reflowed with the correct heating profile and another without. The difference was clear, as the joints produced with the correct profile were noticeably shinier, which from experience correlates with stronger, more conductive solder joints.

Solder paste is also classified by grade, typically ranging from T3 to T5. The number refers to the particle size of the solder balls, with T3 being the largest and T5 the smallest (and most expensive). For most hobbyist and prototyping applications, T4 offers the best compromise as it is fine enough to properly fill the narrow stencil cutouts and is not too expensive.

Proper storage of solder paste is just as important as selecting the right type. Paste should be kept refrigerated and upright, with a typical shelf life of 3–6 months depending on the manufacturer.

Refrigeration prevents the flux from drying out or separating from the solder. Before use, the paste should be removed from the refrigerator and allowed to warm to room temperature naturally.

Construction

Before starting PCB assembly, it was important to make sure every component was properly accounted for and was the correct package size. I printed out the bill of materials (BOM) and double-checked that I had each part ready in its labeled bag, then made a placement plan. Since I was assembling the board by hand, I decided to begin with the microcontroller and then move outward, placing components from the top down. This order made it easier to work with tweezers, as the smallest SMD components can easily shift if brushed or bumped. While solder paste does a good job of holding parts in place temporarily, any accidental contact can knock them off the pads. With the plan finalized, I set up the PCB for soldering and began assembly.

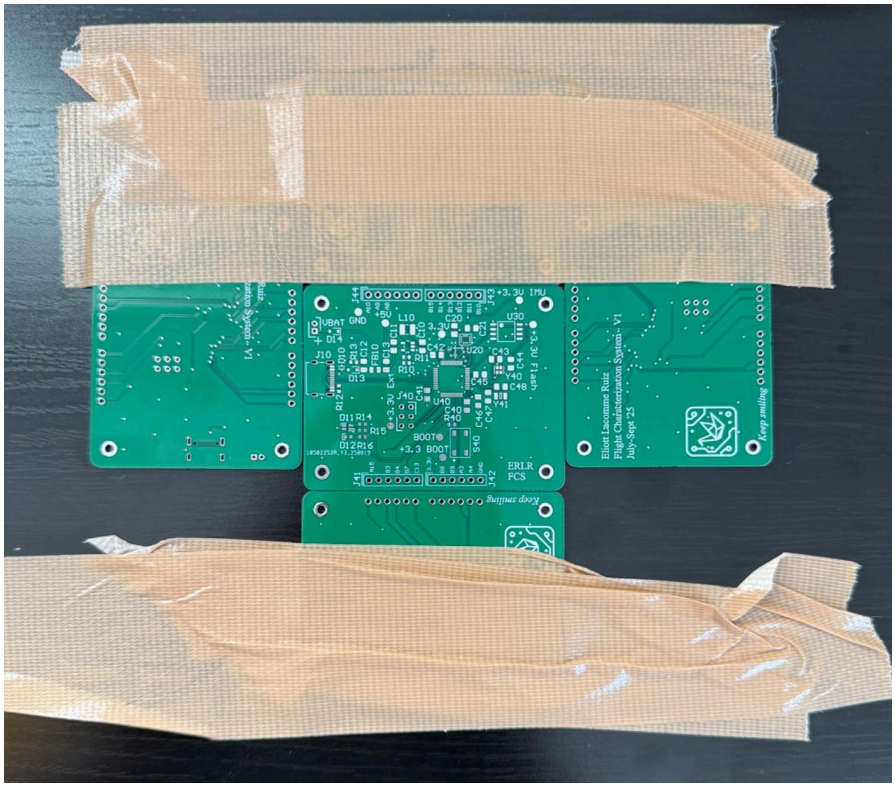


Figure 22: Wedged PCB between other PCBs

As seen in Figure 22, to prevent the PCB from moving, I wedged the PCB in between the other 4 PCBs that came with the circuit and taped them down. This formed a very solid foundation for the PCB.

Figure 23: Stencil lined up with the PCB

Figure 23 shows how I held the stencil in place after lining it up with the exposed pads on the PCB.

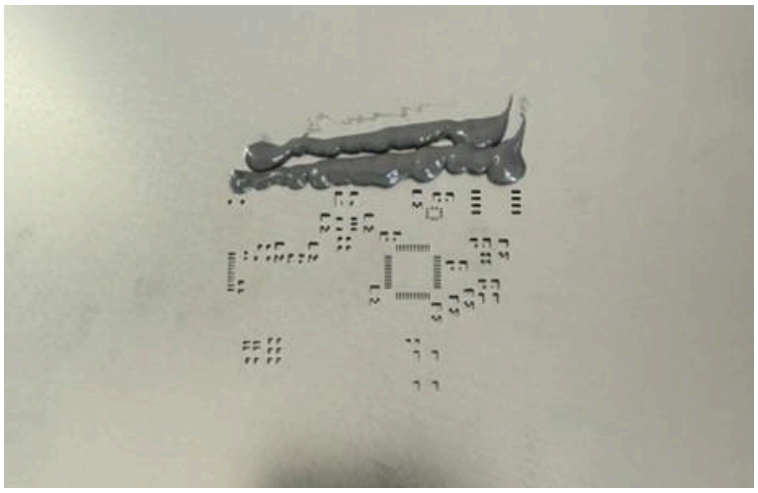


Figure 24: Stencil with solder paste

After securing the stencil with tape, I placed the solder paste above the holes on the stencil (Figure 24).

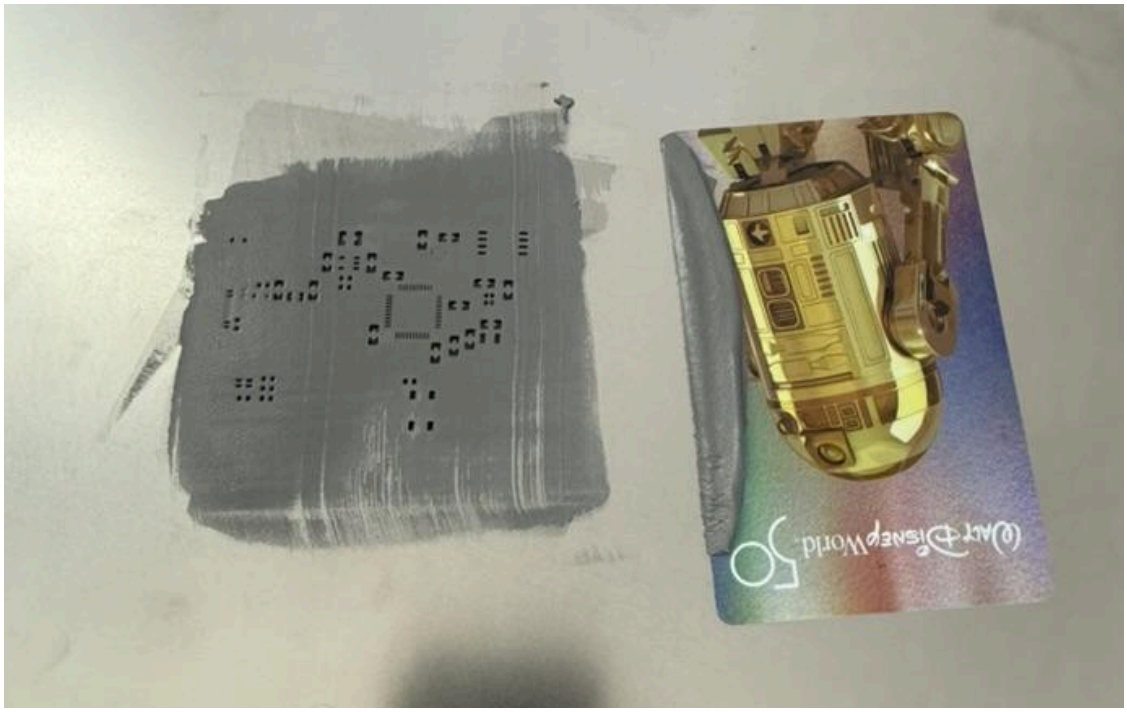


Figure 25: Scraping the solder paste on the stencil

The final step in preparing the board was to spread solder paste across the stencil using a plastic card or paint scraper. The thickness of the stencil makes sure that the correct amount of solder is deposited onto each pad. In my case, I applied a bit too much solder paste, which simply resulted in some waste but had no electrical consequences

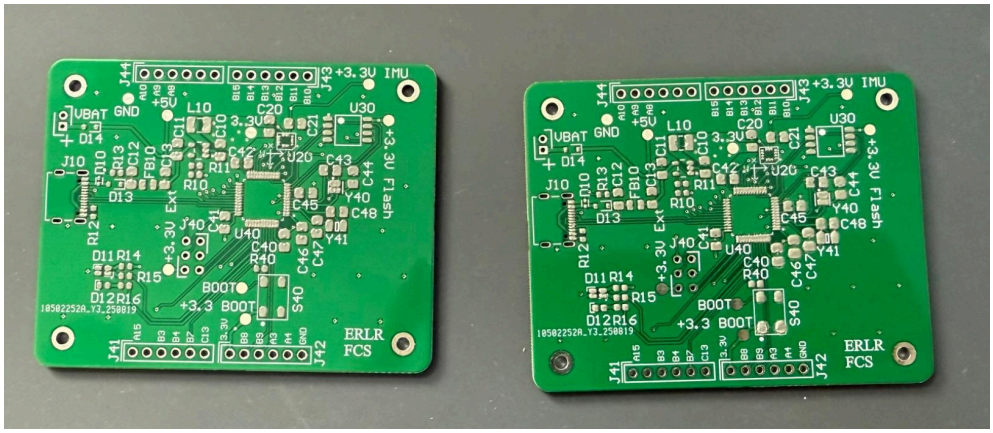


Figure 26: Two PCBs with solder paste on exposed pads

As shown in Figure 26, the solder paste is perfectly placed on the copper pads.

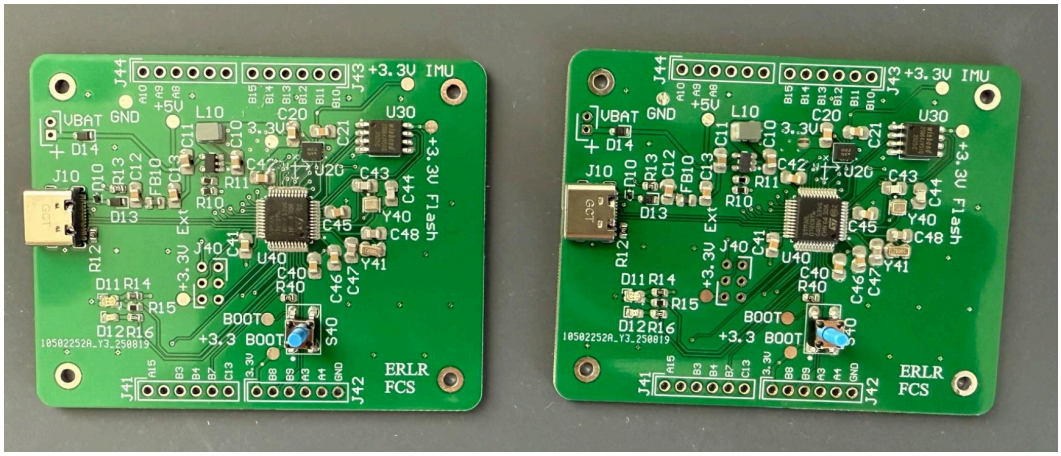


Figure 27: Components placed onto the two PCBs

The components do not have to be exactly straight on the pad, since as the solder heats up, it melts onto the pad and the surface tension of the melted solder pulls the components onto the pad.

I decided to make two PCBs just in case one of them failed in some way. This ended up being a lifesaver since the IMU on the left board (Figure 27) ended up not responding to the chip ID requests (more about this in the coding section). Now, the board was ready for heating on the hot plate.

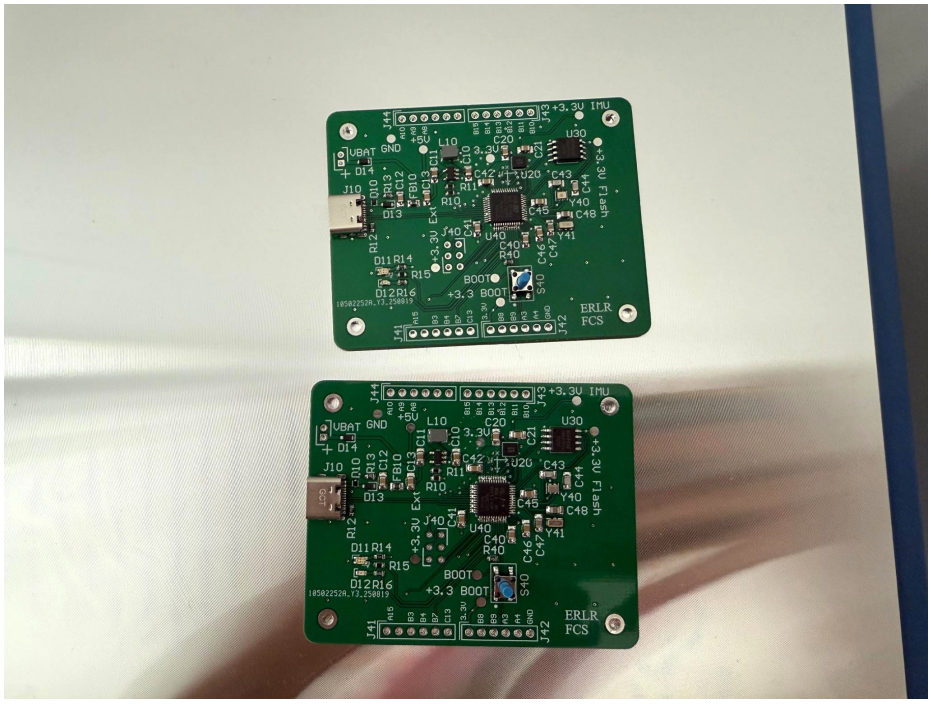


Figure 28: PCBs being heated on the hot plate

During heating, at around 170 °C, the solder melts and becomes shiny (Figure 28).

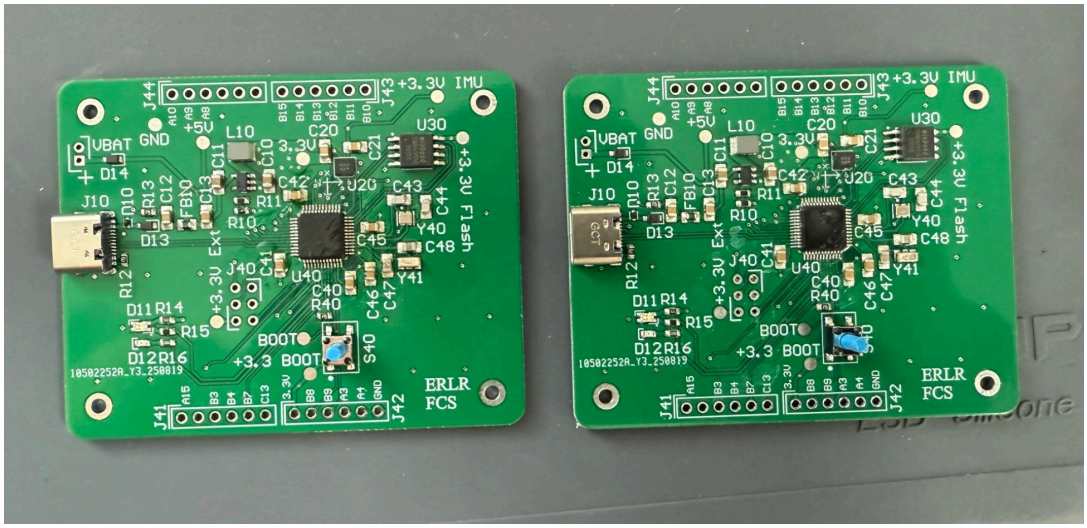


Figure 29: Completed PCBs

With the PCBs complete and cooled, I did encounter a few solder bridges on the microcontroller. To remove them, I applied a generous amount of flux and carefully used a desoldering wick (braided copper rope) along with my soldering iron to pull the excess solder away. It was important to work on just one or two pins at a time and move very slowly, since the MCU pins are extremely fragile and can bend or even break if the wick is dragged across them too aggressively. The process took a lot of patience, but I was ultimately able to clear the bridges successfully.

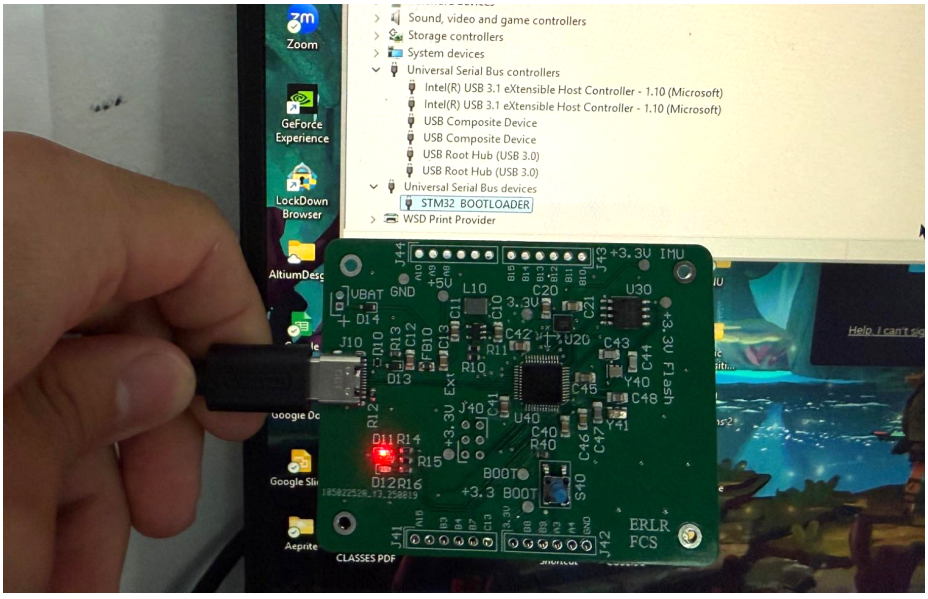


Figure 30: PCB being recognized by computer

Success! As seen in Figure 30, my computer was able to recognize the STM32 Bootloader, meaning the USB was properly configured, the board had the right amount of power, and there was proper connection to the power LED. Now onto the final step, coding.

Bill of Materials (BOM)

Line #	Name	Revision ID	Revision ID	Revision ID	Revision Status	Quantity	Manufacturer 1	Manufacturer Part Number 1	Manufacturer Lifecycle 1	Supplier 1	Supplier Part Number 1	Supplier Unit Price 1	Supplier Subtotal 1
	TLV62569DBVR	IC REG BUCK ADJ 2A SOT23-5	U10	CMP-04918-000387-1	Up to date	1	Texas Instruments	TLV62569DBVR	Volume Production	Mouser	595-TLV62569DBVR	0.25	0.25
	2u2	Inductor SMD 1210	L10			1	Vishay / Dale	IHLP1210BZEZ2R2M5A	Manual Solution	Mouser	70-IHLP1210BZZ2R2M5A	0.87	0.87
	10u	Capacitor SMD 0805	C10, C11			2	Murata	GRM21BR71A106KA73L	Volume Production	Mouser	81-GRM21BR71A106KA3L	0.12	0.24
	100n	Capacitor SMD 0805	C12, C13, C20, C21, C40, C41, C42, C45, C46			9	Walsin Technologies	0805B104J250CT	Volume Production	Mouser	791-0805B104J250CT	0.036	0.36
	TPD1E10B06DPYR	TVS DIODE 5.5V 14V 2X1SON	D10	CMP-0326-00004-4	Up to date	1	Texas Instruments	TPD1E10B06DPYR	Volume Production	Mouser	595-TPD1E10B06DPYR	0.41	0.41
	120R	Ferrite Bead SMD 0805	FB10			1	TDK	MPZ1608S121ATDH5	Volume Production	Mouser	810-MPZ1608S121ATDH5	0.1	0.1
	USB4105-GF-A-060	USB Connector Type C SMT 12 Pin	J10			1	Global Connector Technology	USB4105-GF-A-060	Volume Production	Mouser	640-USB4105-GF-A-060	0.78	0.78
	450k	Resistor SMD 0603	R10			1	Panasonic	ERJ-1GNF4533C	Unknown	Mouser	667-ERJ-1GNF4533C	0.1	0.1
	100k	Resistor SMD 0603	R11			1	Bourns	CR0201-JW-104GLF	Volume Production	Mouser	652-CR0201-JW-104GLF	0.1	0.1
	5k1	Resistor SMD 0603	R12, R13, R40			3	Panasonic	ERJ-1GNF5101C	Volume Production	Mouser	667-ERJ-1GNF5101C	0.1	0.3
	STM32L072CBT6	IC MCU 32BIT 128KB FLASH 48LQFP	U40	CMP-12105-000008-1	Up to date	1	STMicroelectronics	STM32L072CBT6	Volume Production	Mouser	511-STM32L072CBT6	4.36	4.36
	ABM11W-16MHZ-6-D1X-T3	Crystal 16MHz ±10ppm 6pF SMD-4 2mm x 1.6mm	Y40	CMP-27762-011071-1	Up to date	1	Abracon	ABM11W-16.0000MHZ-6-D1X-T3	Volume Production	Mouser	815-11W16-6D1XT	0.4	0.4
	4p	Capacitor SMD 0805	C43, C44			2	KEMET	CBR08C409B5GAC	Volume Production	Mouser	80-CBR08C409B5GAC	0.42	0.84
	10p	Capacitor SMD 0805	C47, C48			2	Samsung	CL21C100CBANNNC	Volume Production	Mouser	187-CL21C100CBANNNC	0.019	0.19
	APHBM2012LSURKCGKC	Ultra low forward current dual LED; Red and greed SMD; If = 2 mA	D11			1	Kingbright	APHBM2012LSURKCGKC	Volume Production	Mouser	604-APHBM2012LSURKCG	0.23	0.23
	YELLOW	LED 0603 YELLOW SMD	D12			1	Inolux	IN-S63AT5YG	Unknown	Mouser	743-IN-S63AT5YG	0.17	0.17
	D_SCHOTTKY_SOD_323	Diode Schottky SMD SOD-323	D13, D14			2	Pan Jit	SBA220CH-AU_R1_007A1		Mouser	241-SBA22CHAUR1007A1	0.36	0.72
	JST PH 2.0 2mm 2 pin Female	JST PH 2.0 2mm 2 pin Female	VBAT			1	Taiss	NA	Manual Solution	Amazon	NA	9.99	9.99
	775	Resistor SMD 0603	R14			1	Panasonic	ERJ-1GNF7680C	Unknown	Mouser	667-ERJ-1GNF7680C	0.1	0.1
	700	Resistor SMD 0603	R15			1	Vishay	CRCW0201698RFNED	Volume Production	Mouser	71-CRCW0201698RFNED	0.1	0.1
	220	Resistor SMD 0603	R16			1	Vishay	CRCW0201220RFNED	Volume Production	Mouser	71-CRCW0201220RFNED	0.1	0.1
	SPST 4.5 x 4.5mm SMD tactile switch	SPST 4.5 x 4.5mm SMD tactile switch	S40			1	ITT C&K	PTS647SK70SMTR2N LFS	Manual Solution	Mouser	611-PTS647SK70SMTR2N	0.26	0.26
	BMI323	IMU	U20			1	Bosch Tools	BMI323	Unknown	Mouser	262-BMI323	3.27	3.27
	W25Q64JVSSIQ TR	64M-bit, DTR, Winbond flash memory	U30			1	Winbond	W25Q64JVSSIQ TR	Unknown	Mouser	454-W25Q64JVSSIQYR	0.97	0.97
	ABS07-32.768KHz-7-T	32.768kHz crytal, smd, C_L=12.5pF, ESR = 80k, Crystal SMD 3215	Y41			1	Abracon	ABS07-32.768KHZ-7-T	Volume Production	Mouser	815-ABS07-32.768KTT	0.82	0.82

Figure 31: Bill of Materials

The bill of materials (BOM) was created in Altium Designer using its live tracking feature to monitor stock availability in real time. The figure above shows a screenshot of the Altium-generated BOM. For sourcing, I selected all of my components from Mouser, since I had worked with them before and found their service reliable. DigiKey is also an excellent option, but in general it's best to stick with a single distributor whenever possible to simplify logistics and reduce delivery times and fees.

Coding

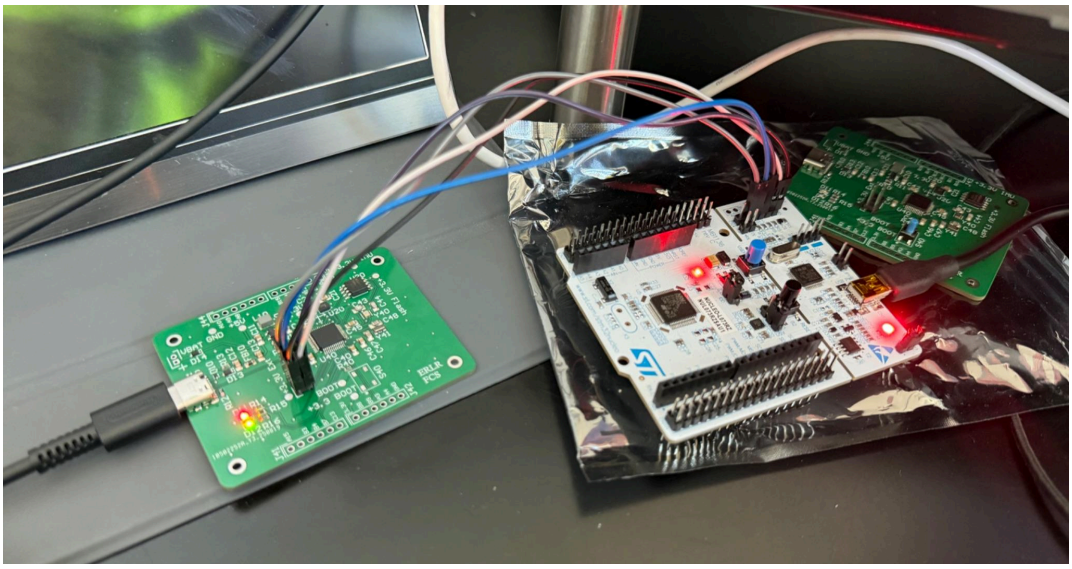


Figure 32: Custom board connected to the ST-LINK for programming and debugging

After successfully getting my computer to recognize the board, the final step of the project was verifying communication with both the IMU and the flash memory. From my first week of research, I had already decided to use ST's programming and debugging software, STM32CubeIDE. This is a powerful tool that simplifies microcontroller configuration and debugging. The only limitation is that it requires an external ST-LINK device to program the chip. During the downtime between ordering and receiving my PCB, I used an STM32 Nucleo board to practice programming, which gave me foundational experience with the STM32 environment and a chance to refresh my C programming skills. One convenient feature of the Nucleo board is that by removing two jumpers on the top half of the board, its built-in ST-LINK can be used to program external devices, which made it easy to adapt for my custom hardware (Figure 32).

Once the board was fully assembled, I wanted to familiarize myself with the coding standards typically used for embedded systems. To do this, I studied the Embedded C Coding Standard by Michael Barr, a widely recognized guide that defines best practices for writing safe, reliable, and maintainable C code in embedded applications. The book covers everything from naming conventions and commenting style to rules for avoiding undefined behavior and improving portability across compilers.

Reading this standard gave me a strong foundation for structuring my own firmware. It helped me appreciate the importance of consistency, clarity, and defensive coding in environments where debugging can be difficult and hardware resources are limited. By applying these guidelines, I was able to approach my firmware development with practices that align more closely with what is expected in professional engineering settings.

Now that I had a solid coding standard, I attempted to upload a simple blinking LED program to one of the GPIO-controlled LEDs. To connect the ST-LINK, I wired header pins from my board directly to the

programmer, as seen in Figure 32. However, when I uploaded the code, the program consistently crashed during the clock configuration stage. After using the testing points to probe voltages throughout the circuit, I discovered the issue. The BOOT pin on the MCU was accidentally being pulled high. My design included a button to pull the BOOT pin high when pressed, but I had overlooked the datasheet details for that button. The two leads that I wired to connect BOOT to ground were actually on the same side of the switch, which meant the BOOT pin was permanently tied to +3.3 V.

For context, the BOOT pin on STM32 microcontrollers determines the startup mode of the chip. When pulled low, the MCU boots from main flash memory (the normal user program). When pulled high, it enters the system memory bootloader or alternate startup modes. Because my BOOT pin was always high, the MCU never ran my user code and it kept trying to enter the wrong boot mode, which explained the crashes.

While removing the button with a heat gun, the pads on the PCB ended up peeling off. Electrically, this didn't cause any issues since the connections were still intact, but it was definitely a lesson for future projects. I'm not sure if the problem came from trying to remove the button before the solder was fully melted, or if my heat gun was simply too hot, around 350 °C, which may have been more than the pads could handle for an extended period of time.

While inspecting the board for other issues, I also discovered that I had accidentally rotated the HSE clock 90 degrees and soldered it incorrectly. I desoldered the part with the heat gun and reattached it in the correct orientation, this time without any pad damage.

After reconnecting the board to a power source, the test code immediately worked, confirming that I was finally able to program my own custom microcontroller board!

Next, I checked whether I could communicate with the IMU on the board. I had configured it to interface with the MCU using SPI (Serial Peripheral Interface), which is a common communication protocol for sensors and peripherals. SPI uses four main signals called MOSI (Master Out, Slave In), MISO (Master In, Slave Out), SCK (clock), and CS (chip select). The MCU acts as the master and controls the clock and chip select, while the IMU serves as the slave device.

When working with ICs over SPI, each chip has its own set of registers and command structure defined in its datasheet. Communication typically involves pulling the chip select (CS) line low to activate the device, then sending a register address over MOSI. The chip responds by sending back the contents of that register over MISO, which the MCU stores for processing. Every command is synchronized to the SPI clock, and depending on the device, the number of clock ticks required to complete a transaction may differ.

In practice, this means that while SPI is a standardized protocol, the exact way you talk to each peripheral, like what registers exist, how data is formatted, and which commands are valid, depends heavily on the specific IC. For the IMU, I followed the BMI323 datasheet closely to set up SPI communication and begin retrieving sensor data. The same can be said for flash memory.

To confirm that both the IMU and the flash memory were communicating properly with the MCU, I first requested their chip IDs. Each device has a unique ID value that identifies the chip type, and both peripherals responded correctly, which verified that the connections and protocols were functioning as expected.

To push the testing further, I wrote code to poll data from the IMU's gyroscope at 400 Hz and display it in real time over USB. This was an exciting step because it showed that not only could I establish communication, but I could also stream live sensor data from my custom hardware.

Due to time constraints, I wasn't able to develop the code much beyond this stage. However, the fact that I could reliably read chip IDs and poll sensor data demonstrated that I had full control over the ICs on the board. Looking ahead, I would like to expand the firmware to make it fully responsive to dynamic events and integrate more advanced data processing.

There is a lot more nuance in the code that I will not get into here, but feel free to look through my GitHub for more information.

Results

The theoretical maximum current draw of the entire circuit was about 40 mA, calculated by summing the maximum burst consumption of all components. The current theoretically spikes by roughly 25 mA when the flash memory is running at full capacity, but during typical operation with the IMU running at 400 Hz, the circuit settles between 10–15 mA. When connected to my power supply, the measured current draw was 11 mA during operation and about 1 mA in idle. At 5 V and 11 mA, this corresponds to a total power consumption of 55 mW at full speed IMU. The theoretical idle current is around 0.4 mA, though I didn't have the equipment necessary to measure values that low with confidence.

Voltage probing confirmed that the input supply remained steady at 5 V, while the buck converter output was about 3.4 V. Although slightly above the target of 3.3 V, this was still well within the safe operating range for all components in the system.

The project took a total of 45 days to fully complete, with another 5 days to write this final report. The total individual board cost came out to be $20.14 before shipping fees and taxes.

Overall, this project was a massive success. I was able to design a fully custom four-layer PCB, select and order all SMD parts, assemble the board by hand, and confirm through code that all components could be communicated with reliably. This met all of the mission criteria I had set for myself and gave me a strong foundation to build even more advanced projects in the future.

Future Developments

This project was a massive success, but there are still many ways it can be improved and expanded. All of my original mission criteria were met, and now the focus shifts towards adding more features in the design.

One of the biggest improvements would be to develop fully functional firmware capable of detecting events and logging data in response. For example, custom profiles could be implemented for rocket-related events such as launch and landing detection, allowing the board to act as a dedicated flight characterization system.

Another enhancement would be to test logging large amounts of data using DMA (Direct Memory Access). DMA allows peripherals to send or receive data directly to memory without continuous CPU intervention. This means that while the MCU is busy handling other tasks, the DMA controller can transfer sensor data in the background. Running the IMU at higher logging frequencies (on the order of ~100 Hz) with DMA would greatly increase throughput and reliability, while reducing CPU load. This would make the system more suitable for precise data analysis of dynamic systems.

Looking ahead, I would also like to expand the circuit's data logging capabilities by adding an SD card interface. This would enable high-capacity storage, ideal for long-duration testing campaigns where flash memory alone might not be sufficient. Alongside this, testing and integrating the Low Speed External crystal and implementing Real-Time Clock (RTC) would allow data logs to be timestamped precisely. This is particularly important for time-correlated measurements in dynamic systems, where events need to be tracked against absolute time.

Beyond firmware, I think it would be an excellent challenge to minimize the profile of the circuit. Reducing PCB size involves significant challenges, such as tighter routing, denser layouts, and potentially smaller component packages, but it could lower manufacturing costs and make the system more practical for smaller sized applications.

I designed the board in such a way that the MCU pins are already arranged in a breakout board configuration. This gives me flexibility to repurpose unused pins for additional features or sensors, making the current board a reusable prototyping platform for future projects.

Completing this project has been both challenging and rewarding, and I look forward to building on this foundation to tackle increasingly complex and meaningful engineering problems!

Resources

Altium Design Resources

- Altium STM32 Hardware Design - An Overview in Under 20 Minutes - Phil's Lab #38
- Microcontroller-Based Hardware Design With Altium Designer - #1 Introduction and Part Selection
- How to Make Custom ESP32 Board in Altium Designer | Full Tutorial - YouTube
- Microcontroller-Based Hardware Design With Altium Designer - #1 Introduction and Part Selection
- Symbol and Footprint Creation | Altium Designer - Phil's Lab #31 - YouTube
- How To Create Your Own Libraries in Altium Designer - YouTube
- Altium Designer Quick-Start Tutorial with Phil Salmony from Phil's Lab - YouTube
- Altium BOM Management Tool How to Create and Manage Your PCB Bill of Materials - YouTube
- Manual Footprint Creation with Altium 365 | Component Creation - YouTube
- Altium STM32 Hardware Design - An Overview in Under 20 Minutes - Phil's Lab #38 - YouTube
- Board Thickness in PCB Design
- Altium Designer Tutorial 9 : How to create Test Point Symbol and footprint in AD 21 Library.
- Off the Belt: How to Create a Bill of Materials Using the FORTE BOM Tool | Mouser Electronics
- Interactive Routing Shortcuts

PCB Design Resources

- Series Intro - Landing Model Rockets
- https://resources.altium.com/p/beginners-guide-esd-protection-circuit-design-pcbs
- Electrostatic Protection Must-Haves - Workbench Wednesdays - YouTube
- Switching Regulator PCB Design - Phil's Lab #60 - YouTube
- ESD Protection Basics - TVS Diode Selection & Routing - Phil's Lab #75 - YouTube
- Pi Filter Formulas and Simulation Deep Dive - YouTube
- Solder Mask Expansion and Minimum Sliver in Altium Designer - YouTube
- STM32 USB HID Custom Joystick/Gamepad - Phil's Lab #149 - YouTube
- Internal Power & Split Planes
- 3D Print your PCB Stencil Jig
- JLCPCB solder paste stencil order and use
- David L. Jones' PCB Design Tutorial

STM32 Resources (hardware and firmware)

- STM32: Things You Need to Know
- https://en.wikipedia.org/wiki/ARM_Cortex-M
- https://www.st.com/en/microcontrollers-microprocessors/stm32l0x2.html
- STM32 Tutorial: take your Microcontroller knowledge to the next level 🚀
- STM32 for Beginners | GPIO using Seven Segment LED Displays | Display Basics
- STM32 USB SD Card Mass Storage Device Tutorial - Phil's Lab #148 - YouTube
- https://stm32world.com/wiki/Main_Page
- https://deepbluembedded.com/stm32-hal-library-tutorial-examples/
- https://nefastor.com/microcontrollers/stm32/getting-started-with-stm32/creating-a-cube-project/
- https://www.freecodecamp.org/news/the-c-beginners-handbook/
- https://www.geeksforgeeks.org/c/c-programming-language/
- STM32 Programming Tutorial for Custom Hardware | SWD, PWM, USB, SPI - Phil's Lab #13

- STM32 Tutorial #1 - Overview, Families, Documentation and Development boards and tools (PLAYLIST)
- Embedded C Coding Standard by Michael Barr

JLCPCB Resources

- JLCPCB Parts
- PCB Manufacturing & Assembly Capabilities - JLCPCB
- Impedance Calculator
- Multi-Layer PCB Standard Laminated Structures
- JLCPCB Stencil Order Instructions