
Jakub Sułek

# Embedded Platform for Self-constructed Drone System

Metropolia University of Applied Sciences

Bachelor of Engineering

Electronics

Bachelor's Thesis

28 November 2024

## Abstract


| | |
|---|---|
| Author: | Jakub Sułek |
| Title: | Embedded Platform for Self-constructed Drone System |
| Number of Pages: | 123 pages |
| Date: | 28 November 2024 |
| Degree: | Bachelor of Engineering |
| Degree Programme: | Electronics |
| Professional Major: | Electronics |
| Supervisors: | Tuan Nguyen, Senior Lecturer |


---


This thesis explores the design and development of a modular embedded platform for a self-constructed drone system, combining theoretical understanding with practical implementation. The project focuses on creating custom hardware for the flight controller and operator interface, developing software libraries for essential modules, and validating their functionality through isolated testing.

Key components, including wireless communication modules, sensors, and motor controllers, were integrated into the design, with simulations conducted using eCalc XcopterCalc to estimate theoretical performance metrics such as hover time, thrust-to-weight ratio, and energy efficiency. While the full system integration remains incomplete, the validated software libraries and hardware designs provide a strong foundation for future work.

The work highlights the accessibility of modern resources for UAV development, demonstrating how affordable components, online tools, and systematic methodologies enable individuals with limited experience to construct sophisticated systems. This thesis contributes to the understanding of UAV systems by presenting a replicable and scalable approach to drone design, with potential applications in education, research, and hobbyist projects.



Keywords: Drones, Remote Control, Embedded, Software Development


---

# Contents

## List of Abbreviations

**ADC**: Analog-to-Digital Converter
**AI**: Artificial Intelligence
**ARM**: Advanced RISC Machine
**CAN**: Controller Area Network
**CE**: Chip Enable
**CSS**: Clock Security System
**CSN**: Chip Select Not
**DC**: Direct Current
**DMIPS**: Dhrystone Million Instructions Per Second
**DOF**: Degrees of Freedom
**DMP**: Digital Motion Processor
**EMC**: Electromagnetic Compatibility
**EMI**: Electromagnetic Interference
**ESC**: Electronic Speed Controller
**EXTI**: External Interrupt
**FPV**: First Person View
**GFSK**: Gaussian Frequency Shift Keying
**GPIO**: General Purpose Input/Output
**GPS**: Global Positioning System
**HSE**: High-Speed External
**I2C**: Inter-Integrated Circuit
**IMU**: Inertial Measurement Unit
**ISM**: Industrial, Scientific, and Medical
**LED**: Light Emitting Diode
**LiPo**: Lithium Polymer
**LNA**: Low Noise Amplifier
**MISO**: Master In Slave Out
**MOSI**: Master Out Slave In
**NDOF**: Nine Degrees of Freedom
**NVIC**: Nested Vectored Interrupt Controller
**OLED**: Organic Light Emitting Diode
**PA**: Power Amplifier
**PCB**: Printed Circuit Board

**PID**: Proportional Integral Derivative

**PWM**: Pulse Width Modulation

**RC**: Remote Control

**RF**: Radio Frequency

**RPM**: Revolutions Per Minute

**RTOS**: Real-Time Operating System

**SBC**: Single Board Computer

**SPI**: Serial Peripheral Interface

**SWCLK**: Serial Wire Clock

**SWD**: Serial Wire Debug

**SWDIO**: Serial Wire Data Input/Output

**TVS**: Transient Voltage Suppressor

**UART**: Universal Asynchronous Receiver/Transmitter

**UAV**: Unmanned Aerial Vehicle

**USART**: Universal Synchronous/Asynchronous Receiver/Transmitter

# 1 Introduction

The accessibility of knowledge and components for building complex systems like drones has grown exponentially over the past decade. What once required specialized equipment, proprietary hardware, and a deep technical understanding is now within reach for hobbyists, students, and professionals alike. Online resources, affordable development tools, and open-source platforms have transformed the learning curve, making it possible to design and construct sophisticated systems with limited prior experience.

This thesis is a personal exploration of this phenomenon, focusing on the design and development of an embedded platform for a self-constructed drone system. The project combines theoretical knowledge with practical implementation, covering aspects of hardware design, software development, and system integration. For me, this project is also an opportunity to learn how drones operate, integrate compatible hardware and software, and develop skills in embedded programming- areas I had limited experience in before starting this work.

The thesis is organized to present a clear progression of the project, beginning with the historical and technical foundations of drone technology, advancing through system design and implementation, and culminating in an analysis of performance metrics. This document serves as both a record of my learning journey and a potential resource for others pursuing similar projects.

## 1.1 Background

Drones represent a fascinating junction of advanced engineering and accessible technology, offering a platform for innovation across numerous disciplines. At the heart of this capability lies the integration of embedded systems, which enable drones to execute complex tasks such as precise navigation, stable flight control, and real-time communication. These systems have grown significantly in functionality and affordability, transforming drones

from specialized tools into widely accessible projects for enthusiasts and learners alike.

Embedded systems play a central role in drone operation, handling critical tasks such as motor control, sensor data processing, and communication. With advancements in microcontrollers, sensors, and wireless communication modules, these systems have become increasingly compact, affordable, and powerful. Together, this mixture of progressing changes has smoothed access to drone-building technology, making it possible for individuals with modest resources to construct their own UAVs.

Rather than focusing on a specific application, this project explores the fundamental principles of drone design and embedded systems integration. The primary objective is not to create a tool for any particular use case but to develop a deeper understanding of how these systems operate and interact. By approaching the drone as a learning platform, this work contributes to the foundational knowledge required for more specialized or application-driven designs in the future.

For someone like me, with limited prior experience in embedded systems, this shift represents an incredible opportunity. The availability of development tools, detailed documentation, and online communities provides an ideal environment to explore the theoretical and practical aspects of drone design. This project leverages these resources to develop a custom embedded platform tailored to a self-constructed drone, serving as a hands-on introduction to the field while contributing to the broader understanding of UAV systems.

## 1.2 Problem + Research Questions

Designing a functional drone requires integrating various components, such as sensors, microcontrollers, communication systems, and power supplies, into a cohesive embedded platform. The lack of a single resource guiding the development of a self-constructed drone for learning purposes presents a challenge for beginners.

This thesis addresses the following research questions:

- How can an embedded platform for a custom drone be developed to facilitate learning?
- What are the key challenges in integrating hardware and software components for a modular system?
- How can such a system be designed to allow scalability and future enhancements?

## 1.3 Proposed Solution

This thesis presents a modular embedded platform designed for a self-constructed drone system. The solution involves creating custom hardware for the flight controller and operator interface, alongside developing and testing software libraries for each module. While these modules have been evaluated in isolation, system-wide integration remains an open task, laying the groundwork for future enhancements. Emphasis is placed on accessibility, ensuring the design and development process can be replicated and expanded by others.

## 1.4 Contributions

This project contributes to the field by:

- **Custom Hardware Design:** Designing a flight controller and operator interface tailored to the project's needs.
- **Communication Systems:** Implementing robust wireless communication between the drone and controller.
- **Software Development and Optimization:** Creating libraries for modular software integration.
- **Scalability and Modularity:** Ensuring the system can support future expansions and enhancements.
- **Testing and Validation Framework:** Establishing methods for theoretical performance evaluation and testing of individual components.

# 2 Historical Evolution of Drones

The term "drone," often used interchangeably with "Unmanned Aerial Vehicle" (UAV) or "Unmanned Aircraft System" (UAS), refers to aircraft that operate without an onboard pilot. Historically rooted in military advancements, drones have since transformed into versatile tools utilized across civilian, commercial, and military domains. Their applications now range from agriculture and infrastructure inspection to combat and reconnaissance operations. [1.] This chapter explores the historical development of drones, focusing on their military origins, technological evolution, and their subsequent adoption into broader applications.

## 2.1 Early Development

The origins of drones trace back to the 19th century when the Austrian military deployed unmanned hot air balloons loaded with explosives against Venice in 1849. Although primitive by modern standards and subject to environmental factors, these devices represent the earliest form of pilotless aerial systems. During World War I, Charles Kettering introduced the "Kettering Bug" an aerial torpedo that incorporated a gyroscope for navigation, marking a significant milestone in drone innovation despite its limited deployment. In 1930s, the British Royal Navy developed the DH 82B Queen Bee, considered one of the first reusable and remotely controlled drones. Built primarily for target practice, the Queen Bee could be piloted remotely and survive multiple uses if undamaged. The term "drone" itself originated during this period when U.S. Navy Commander Delmer Fahrney named his own project in homage to the Queen Bee, associating the device with the male honeybee known for its buzzing sound. These innovations laid the foundation for modern UAVs. [2.]

## 2.2 UAVs in WWII and the Cold War

UAVs played an important role during World War II, especially the lethal types utilized by the Germans in the later years of the war. However, the first successful use of UAVs for reconnaissance purposes was not recorded until the

Vietnam War era. Equipped with cameras and other sensors, the Firefly UAVs undertook surveillance missions and collected data from hostile territories. The technological advancements during this period included improved navigation systems and payload capacities. [3.]

## 2.3 The Rise of Modern UAVs

The post-Cold War period marked a turning point in UAV development, as technological advancements in robotics, miniaturization, and automation expanded their capabilities. The Gulf War in the early 1990s demonstrated the tactical advantages of UAVs such as the Predator drone, which provided real-time intelligence and surveillance in hostile environments. [4.] These drones utilized advanced GPS navigation systems and real-time video transmission, significantly improving operational efficiency. [5.]

## 2.4 Modern Civilian and Commercial Uses of Drones

Drones have rapidly expanded beyond military operations to find utility in civilian and commercial domains. Their integration into everyday life has been driven by advancements in affordability, usability, and versatile applications. For example, drones are increasingly used in agriculture for crop monitoring and irrigation, significantly improving efficiency and resource utilization. [1.] Similarly, industries such as construction and logistics benefit from drones for site surveys and rapid delivery of goods. [2.]

Municipalities and public services have also leveraged drones for environmental monitoring and disaster management. For example, drones are now used to assess flood-affected areas and guide recovery efforts, reducing risks for human operators. [2.] Furthermore, their affordability and ease of use have made drones accessible to hobbyists, who use them increasingly for activities such as photography.

## 2.5 Modern Military and Security Applications of Drones

Drones remain integral to military and security operations worldwide. Modern UAVs, such as the MQ-9 Reaper, are equipped with sophisticated sensor arrays and precision-guided munitions, enabling high-stakes reconnaissance and combat missions. [6.]
In national security, drones serve as a critical component in intelligence gathering and real-time decision-making. Advanced UAVs equipped with artificial intelligence and autonomous capabilities are now being developed to enhance operational efficiency and reduce risks to human operators. [2.]

## 2.6 AI Integrations

In recent years, the development of UAVs has been further accelerated by advancements in artificial intelligence (AI) and machine learning. These technologies have enabled drones to operate autonomously and perform complex tasks while being more adaptable. AI-powered drones can now analyze vast amounts of data in real time, enabling them to navigate efficiently, recognize objects, and even predict maintenance needs before issues arise. Deep learning has given drones the ability to identify and classify objects in diverse settings, which is especially useful in fields like agriculture, logistics, and infrastructure management. At the same time, machine learning techniques, such as reinforcement learning, allow drones to adjust to changing environments and make decisions without constant human input. Together, these breakthroughs are transforming UAVs into intelligent systems capable of taking on roles that once required human oversight, opening up vast possibilities across both civilian and military domains. [7.]

## 2.7 Summary

From their military origins in the 19th century to their current role as multipurpose tools, drones have undergone a remarkable transformation. Early developments like the Kettering Bug and Queen Bee established the foundation for technological advancements that would revolutionize military and civilian

domains alike. Modern drones now play pivotal roles in agriculture, public services, and security, showcasing their versatility and potential for societal impact. As technology continues to evolve, drones are poised to become even more integral to our daily lives and global industries.

# 3 Quadcopters Unveiled: Principles, Designs, and Engineering

Quadcopters are a type of UAVs defined by their use of four rotors to achieve lift and stability. In quadcopters, “reactive moments are balanced due to the rotation of the support propellers in pairs in different directions or the inclination of the thrust vector of each propeller in a certain direction”. [8.]

Quadcopters are known for their ability to hover, perform vertical take-offs and landings, and maneuver with precision in confined spaces. This unique capability has made them one of the most versatile and widely adopted configurations in the UAV landscape. Over the years, quadcopters have evolved significantly, transitioning from simple experimental models to highly sophisticated systems used across various civilian, commercial, and military domains. Their popularity results from their adaptability, ease of use, and the ongoing advancements in materials, propulsion systems, and flight control algorithms. [9.]

Quadcopters are typically classified by their frame shapes and the number of rotors they employ, with frame shapes being designated by characters such as "I," "X," "+," and "V”. The accompanying numerical character designates the number of rotors, which, in case of quadcopters will naturally always be 4. Each frame shape is tailored to specific performance requirements and use cases. These designs influence factors such as stability, maneuverability, and suitability for various applications.

## 3.1 X4 Quadcopter

The X-shaped design, shown in Figure 1, is one of the most popular and versatile configurations, offering a balance between stability and maneuverability. It is widely used for applications such as aerial photography, videography, racing, and acrobatics. Variants like true X, square, hybrid X, and stretched X provide additional flexibility to meet specific operational needs. The symmetric rotor placement ensures balanced flight, making this design particularly popular among both professionals and hobbyists. [8.]

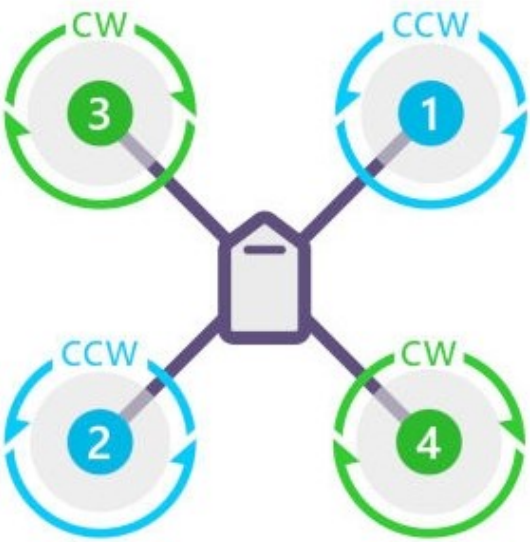



Figure 1: X4 Quadcopter diagram [10]

## 3.2 H4 Quadcopter

The H-shaped frame, shown in Figure 2, is characterized by its broader central body, which provides enhanced stability and space for mounting heavy payloads such as cameras or sensors. This design is especially favored in industrial and agricultural applications where precise hovering and data collection are critical. The larger frame also accommodates additional equipment for specialized tasks. [8.]

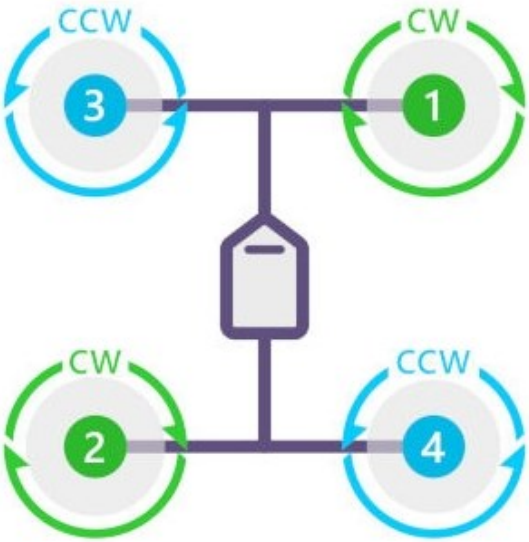



Figure 2: H4 quadcopter diagram [10]

## 3.3 +4 Quadcopter

The + configuration, visualized in Figure 3, aligns the rotors along the cardinal axes, optimizing aerodynamics for straight-line flight. This design is particularly effective for high-speed maneuvers and acrobatic tasks but may sacrifice some stability during complex turns compared to the X or H designs. It is best suited for applications where drag reduction and direct propulsion are prioritized. [8.]

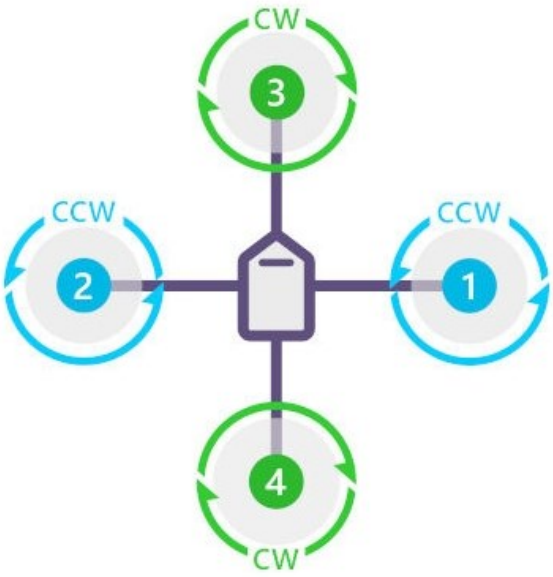



Figure 3: +4 quadcopter diagram [10]

## 3.4 Y4 Quadcopter

The Y4 design, as presented in Figure 4, features two coaxially mounted rotors on the rear arm, providing superior yaw control and increased lifting power. This setup enhances maneuverability and is often used in scenarios requiring precise navigation, such as search-and-rescue operations or inspections in confined spaces. [8.]

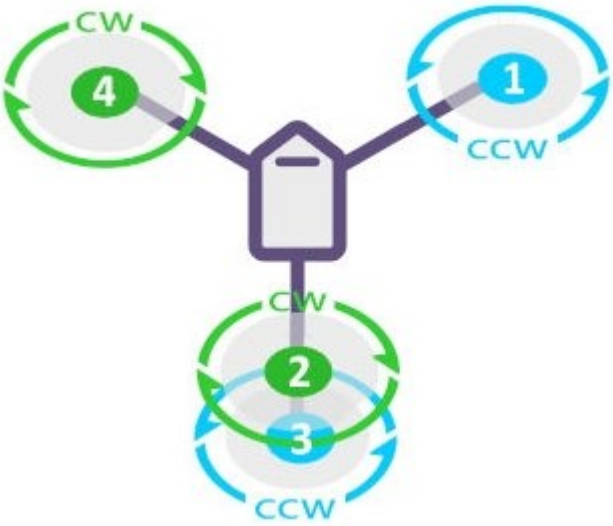



Figure 4: Y4 quadcopter diagram [10]

## 3.5 V-Tail or A-Tail Quadcopter

Known for its distinctive rear motors angled outward, the V-tail (Figure 5) or A-tail (Figure 6) configuration uses differential thrust to achieve enhanced yaw control. This design excels in applications requiring high-speed turns and agile maneuvering, such as drone racing or dynamic photography. The angled motor setup reduces drag and increases responsiveness during flight. [8.]

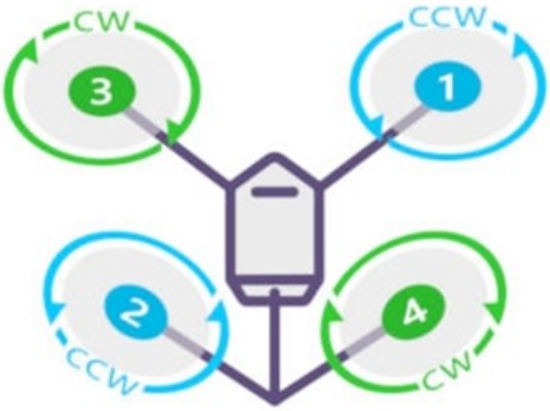



Figure 5: V-Tail quadcopter diagram [10]

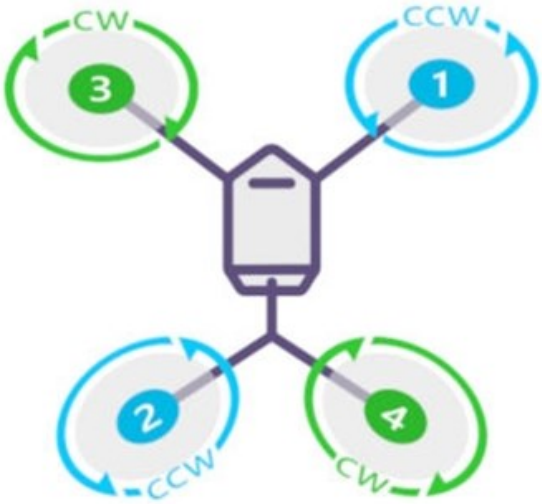



Figure 6: A-Tail quadcopter diagram [10]

## 4 Typical Flight Controllers

A flight controller is a critical component of UAVs, serving as the central processing unit that manages the aircraft's flight dynamics, navigation, and stability. It regulates motor speeds and interprets input instructions delivered by the operator, such as to ascent, descent, turn, or maintain a specific altitude, translating them into specific actions to be performed by the motors. The flight controller is also responsible for managing more autonomous operations, such as camera triggering, autopilot mode control, and others. [11.][12.]

## 4.1 Sensors

Sensors are essential for flight controllers to maintain stability and execute precise movements. Gyroscopes and accelerometers track angular velocity and linear acceleration, allowing the controller to monitor orientation and stabilize the drone. [13.] Barometers measure air pressure for altitude control, while GPS modules provide real-time location data for navigation and autonomous flight. Magnetometers assist with heading and navigation by detecting the Earth's magnetic field. By integrating data from these sensors, often through advanced sensor fusion algorithms, flight controllers ensure accurate and reliable operation in various conditions. [14.]

## 4.2 Motor Control

One of the most important functions of the flight controller is motor speed regulation. The flight controller uses data from the sensors to calculate the desired speed for each of the motors. The result of the calculation is sent to the Electronic Speed Controller (ESC), which, in turn, translates the speed into a signal for the motors. [15.] By doing so, the controller can manipulate the UAV's flight direction and stability. Continuous monitoring of the UAV's position allows it to make rapid motor speed corrections to maintain balance and adapt to changing flight conditions. [12.]

## 4.3 Communication

Communication systems are critical for ensuring seamless interaction between the flight controller, ground control stations, and auxiliary components such as cameras or payload delivery systems. Flight controllers typically use radio frequency (RF) modules to receive pilot commands via remote controllers. In autonomous drones, they also transmit telemetry data, such as battery status, GPS location, and altitude, back to the ground station in real time. [16.]

Modern flight controllers support additional communication protocols like Wi-Fi, Bluetooth, or 4G/5G, enabling advanced functionalities such as real-time video streaming, cloud data synchronization, and swarm coordination. [17.][18.]

### 4.4 Computation

The computational capability of a flight controller determines its ability to process data from sensors, execute control algorithms, and communicate efficiently. Many modern flight controllers use microcontrollers or single-board computers (SBCs) that run real-time operating systems (RTOS). These systems ensure low-latency responses, critical for motor adjustments and autonomous navigation. [19.]

## 5 Typical Controllers

Drone controllers serve as the primary interface between a pilot and the drone, enabling maneuvers, communication, and real-time feedback. Modern controllers integrate intuitive inputs, advanced communication systems, and user-friendly information displays to provide comprehensive operational experience. This chapter explores the theoretical underpinnings of controllers, focusing on the critical elements that make up their functionality: sticks, communication systems, information display, and buttons for menu navigation and various functions.

### 5.1 Sticks

The control sticks on a drone controller are the primary inputs for manipulating the drone’s movement. Typically arranged as two joysticks, these controls map to four fundamental movements:

- **Roll**: Tilts the drone left or right by varying the power sent to motors on opposite sides.
- **Pitch**: Moves the drone forward or backward by adjusting the motor speeds on the front and rear.

- **Yaw**: Rotates the drone clockwise or counterclockwise by modulating opposite motors' rotational directions.
- **Throttle**: Changes the drone's altitude by increasing or decreasing the overall motor power.

The position of each stick is converted into a corresponding signal and transmitted to the drone in real-time. Precise control is achieved through proportional adjustment- small movements result in fine adjustments, while larger stick deflections lead to aggressive maneuvers. [20.]

## 5.2 Communication

The communication systems utilized in the controller are closely aligned with the technologies outlined for the drone's communication. Both systems rely on radio frequency (RF) modules for transmitting and receiving commands, with additional support for modern protocols like Bluetooth, Wi-Fi, and 4G/5G where applicable. This shared technological foundation ensures interaction between the drone and controller.

## 5.3 Information Display

Drone controllers often incorporate screens or interfaces to provide real-time feedback to the pilot. This information is crucial for informed decision-making and enhances operational efficiency. Displays may feature:

- **Telemetry Data**: Battery levels, signal strength, GPS coordinates, and altitude are presented to keep the pilot informed.
- **First-Person View (FPV)**: Some controllers integrate FPV screens or allow smartphones/tablets to serve as displays, providing a live video feed from the drone.
- **System Warnings**: Alerts such as low battery or lost signal ensure timely intervention to prevent crashes or system failures.

Modern controllers, like DJI's Smart Controller, exemplify the integration of advanced displays with built-in telemetry and connectivity features, minimizing the need for additional devices. [20.]

### 5.4 Buttons and Menu

Buttons on the controller provide additional functionality, enabling quick adjustments and access to drone settings. Trim buttons correct drift for stable flight, while programmable buttons allow rapid execution of tasks like switching camera modes or activating return-to-home features.

Menu systems, often navigable via dedicated buttons, allow users to configure parameters, calibrate sensors, and manage communication settings. These menus, organized hierarchically, ensure ease of use while offering extensive customization options to suit diverse operational needs.

## 6 System Architecture

The system architecture describes the elemental needs of the X4 drone design within an abstract view and how they have been satisfied with chosen components and subsystems. The project consists of two devices, a remote-controlled vehicle in the form of a quadcopter, and a remote controller that allows for the operator to manipulate the behavior of the vehicle wirelessly.

Presented system architecture shown in the Figure 7, consists of four sets of propulsion mechanisms marked with the letter A, four sets of ESCs marked with the letter B, a flight controller PCB marked with the letter C, battery pack marked with the letter D, which are elements of the drone, as well as the remote controller marked with the letter E.

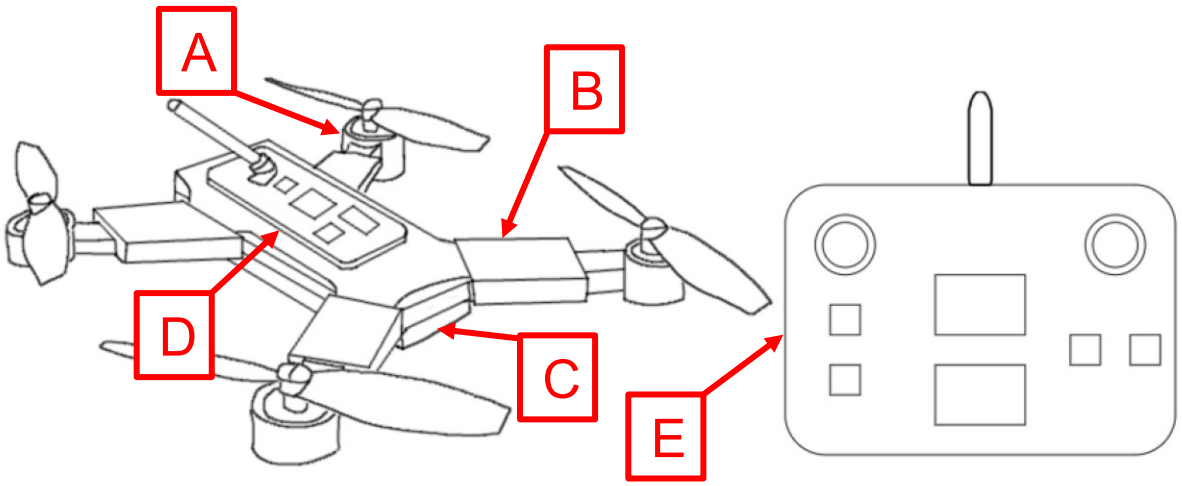



Figure 7: Diagram of the Drone-Controller system

The architecture of this drone is intentionally built around the fundamentals of stable flight, practical control, and ease of integration. Designed to reflect the peripheral needs of the chosen battery pack mass and dimensions, it prioritizes core functions of wireless communication, flight, awareness of the orientation in space and altitude, while allowing for an accessible, reliable platform to meet essential flight requirements. This design approach reflects a need for robustness and simplicity rather than cutting-edge performance, creating a system well-suited for manageable development and operation, and future expansion of features and capabilities.

## 6.1 Drone

The power storage has a core role for this architecture's dependencies. A six-cell LiPo battery pack in a 3S2P configuration provides the main power source, supporting both high-demand of the propulsion system and low-power usage by the supporting components with ample energy capacity. This battery pack configuration supplies power to flight controller PCB and each motor through ESC. LiPo batteries are advantageous here for their compact size relative to energy density, which keeps the overall design practical while affording the endurance necessary to sustain flight. The battery choice approach ensures

consistent power delivery of a medium sized drone allowing for margin of safety for inefficiencies of other components.

The propulsion system features four brushless DC motors driven by dedicated electronic speed controllers (ESCs). The configuration provides the lift needed and control necessary to sustain stable flight. The choice of these components is informed by their availability and the straightforward motor control they offer, ensuring consistent output even under varying load conditions. With this setup, thrust can be adjusted, offering balanced movement across axes, essential for a stable, responsive platform. While not designed for extreme efficiency, the motors and ESCs support adequate thrust and response, suitable for this larger, slightly heavier drone.

At the center of the architecture a consolidated PCB is designed that coordinates essential control and sensor functions. Integrated control board simplifies assembly and reduces weight by incorporating components directly, allowing them to interact with minimal wiring. Critical onboard elements include wireless communication, an inertial measurement unit (IMU), range and altitude sensors, and battery monitoring. The IMU, equipped with comprehensive orientation features, provides real-time data on the drone’s position and motion. Its ability to process multiple data types, such as acceleration and angular velocity, reduces the need for external computations, keeping the system streamlined. This module is central to maintaining stability, as it will continuously provide orientation data to the control system, enabling the drone to correct and stabilize its flight.

The inclusion of a range finder and a pressure sensor for altitude measurement adds redundancy and precision to the flight control. While the pressure sensor can track altitude changes during standard flight, the range finder is optimized for close-range measurements, ensuring accurate height data when the drone hovers close to the ground. This dual-system approach to altitude management provides flexibility and accuracy in various flight conditions, making it suitable for more informed decision making of the operator when there is need for careful maneuvers, even in inconsistent air pressure environments.

Wireless communication is another essential aspect, facilitated by a module that enables real-time control and data transfer. This component is chosen to allow remote control of the vehicle at distance of 100 meters, while providing scalable data rates, and reliability. The module is to provide coverage that will allow tests from a safe distance and data rate that could be modified to deliver a balance between power consumption and responsiveness. The features mentioned are decided on to give the system an added layer of operational flexibility.

In summary, this drone system architecture presented in Figure 8, provides a solid base by focusing on fundamental flight and control needs. The combination of propulsion, power, control, and sensor modules creates a balanced design that prioritizes simplicity and functional stability. By integrating essential components with minimal complexity, this architecture supports a reliable entry-level platform, ensuring that the drone meets core requirements while leaving room for further improvements and refinements in the future.

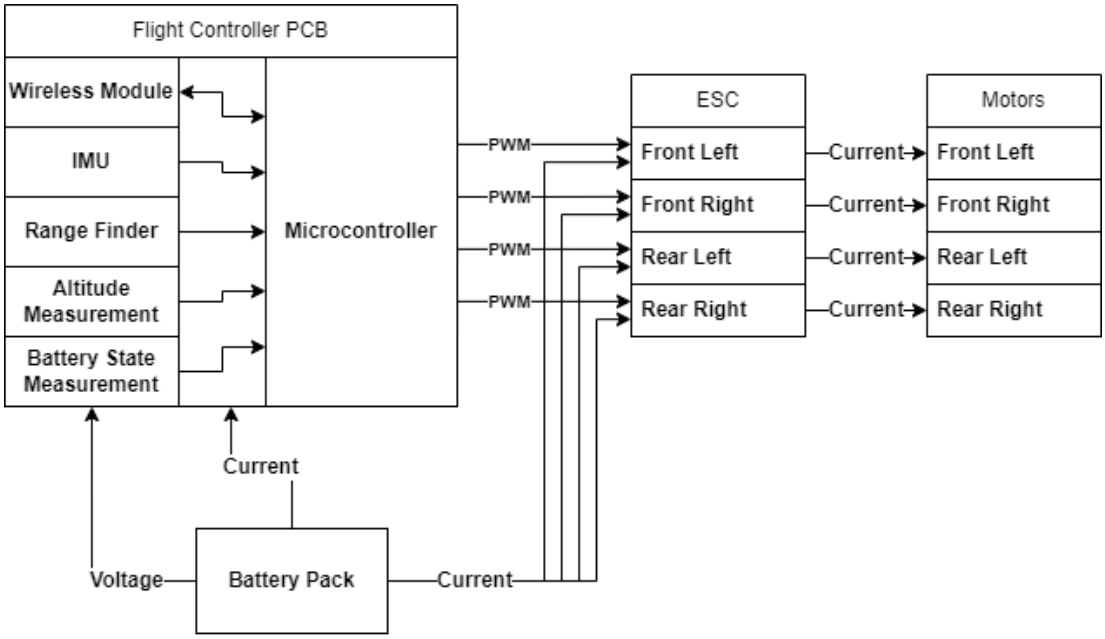



Figure 8: Vehicle architecture diagram

## 6.2 Controller

The controller architecture presented on Figure 9, is centered around a single microcontroller, serving as the core processing unit for managing

communication, user input, and system feedback. This microcontroller allows integration between the drone and the operator, utilizing components to provide responsive control over the drone's movements and allowing for clear indication of its status.

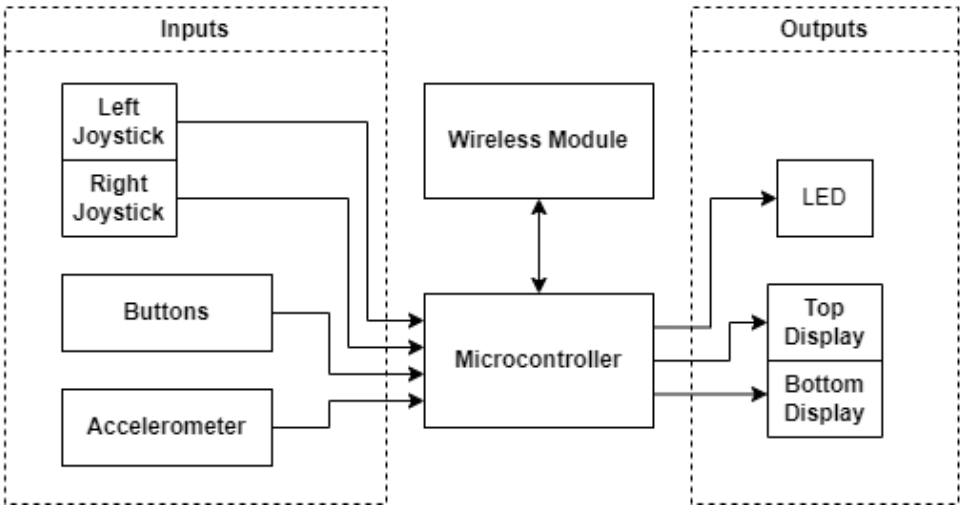



Figure 9: Controller system architecture diagram

A primary component of this setup is the wireless communication module, which enables real-time connection with the drone. The module allows for the transmission of control inputs and configuration information to be sent to the drone. The device will also have a role as a receiver of the sensor readings of the drone which will be presented to the operator.

Movement is controlled through two joysticks, each corresponding to specific axes of the drone's motion. The joysticks allow the operator to directly adjust the drone's altitude, direction, and speed with simple input. The microcontroller translates these inputs into corresponding commands for the drone, facilitating basic navigation.

For additional interaction, the controller includes four tactile switches, which navigate through system menus and allow the user to modify settings or modes. This setup keeps the interface simple and easy to navigate, letting the operator make adjustments without needing complex input sequences.

Two small OLED displays are included to give the user clear visual feedback. The first display serves as a primary interface for navigating through settings and adjusting parameters, while the second screen displays real-time status updates of the drone such as altitude, orientation and battery status. This separation of functions enhances usability, allowing the user to view crucial data at a glance without interrupting the control flow.

A LED indicator provides simple visual cues for various controller states, such as pairing status or alert conditions. This LED is programmed with different blinking patterns, giving the user quick, non-distracting indication of the controller's situation.

The accelerometer adds a unique dimension to the controller by allowing for tilt-based steering. This sensor detects the orientation of the controller and converts tilting motions into control signals, offering an alternative way to navigate the drone. By combining tilt-based control with traditional joystick input, this design gives users flexible control options that enhance maneuverability and provide a more dynamic, immersive experience.

Controller's architecture focuses on accessible and reliable operation, offering only the necessary features to meet essential control needs. Each component is chosen to facilitate a straightforward, practical interface that serves as a functional link between the user and the drone, emphasizing ease of use and responsiveness. The physical layout of the controller is presented in Figure 10.

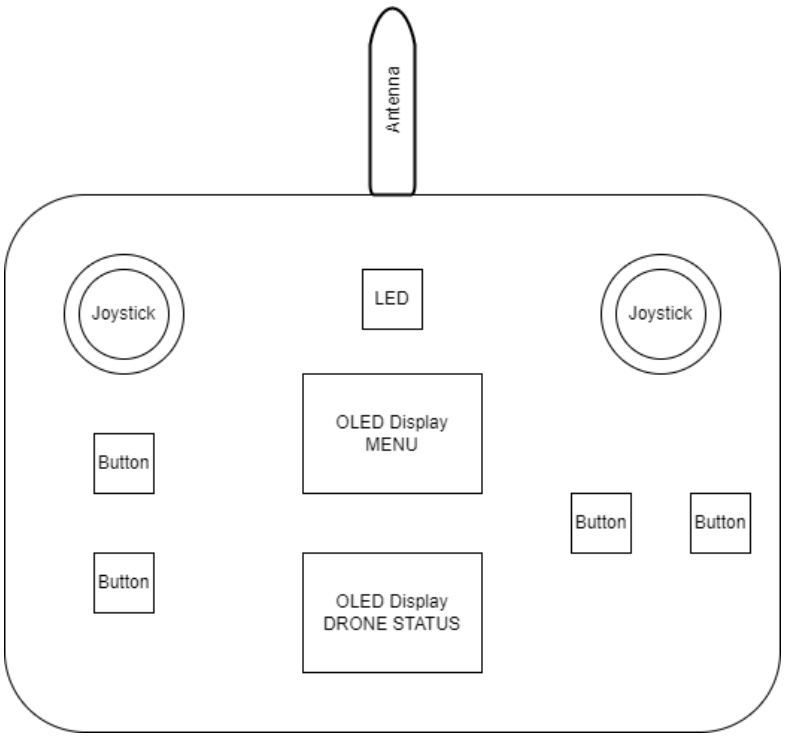



Figure 10: Sketch of the controller's user interface

# 7 Vehicle’s Sensors, Devices and Components

At heart of designing a drone is the decision-making regarding devices and components crucial for the fundamental needs of the vehicle. Fulfillment of vehicular needs must be balanced across issues such as mass, power consumption, integration and control. This section provides an in-depth look at each component included in the drone, focusing on how their specific features meet essential operational requirements, from powering the motors to capturing reliable sensor data.

Each component was chosen based on its ability to fulfill its role while remaining compatible with the drone’s power supply and control systems. For instance, sensors like the IMU and range finder are critical for stable navigation and require dependable power and consistent data transmission to effectively interface with the flight controller. These dependencies mean that each part’s power and data specifications, as outlined in its datasheet, had to be carefully matched with the system’s capacity to support smooth operation. The paramount focus is placed on the stability and relative margin of safety of the

mass constraints. It means that the design is making use of economy of scale in the sense that the larger and heavier the drone, the larger the tolerance for mass inefficiencies during the assembly. Considering this, the wheelbase of the vehicle's frame is nearly 450mm, considering the mass to be above 1500g which impacts resulting choices of the propulsion system elements and their dimensional needs.

This section explores each component's functionality, operational utilization dictated by the initial design requirements, and role within the system, explaining how they work together to provide the drone's basic capabilities. Each section highlights the interdependencies among components and the rationale behind each selection, building on the system architecture outlined in the previous chapter.

## 7.1 The Propulsion System

To achieve stable lift for a 1500 g drone and ensure reliable directional control without compromising altitude, a thrust-to-weight ratio of at least 2:1 was required [21]. Given these performance needs, the use of brushless electric motors was prioritized over brushed versions. Brushless motors provide several advantages, such as improved efficiency, reduced heat generation, and extended operational life due to their lack of physical brushes, which are susceptible to wearing over time. These benefits make brushless motors an effective choice for applications requiring consistent performance with lower maintenance demands. [22.]

The chosen motor, Flash Hobby D2836EVO 1450 KV motor shown in Figure 11, has a maximum power rating of 739.6 W, offering the necessary output to produce adequate lift and manage quick directional adjustments. With a maximum current draw of 48.69 A, this motor is compatible with the drone's LiPo battery.


Figure 11: Photo of the D2836EVO

The technical data provided by the manufacturer presented in Figure 12, provides confidence regarding thrust capability and outlines the needs imposed on the power supply.

Flash Hobby D2836EVO 1450KV

Max watts: 739.6W
Max Current: 48.69A
No Load current: 2.24A
Resistance: 33.5mΩ
Battery: 3-4S Li-Po
Configuration: 12N14P
48H Level Magnet
Suggested Prop: 9-inch prop

Weight: 88.6g(Including wires)
Motor Size: 28 x36mm
Shaft Size: 4.0*51.0mm
Stator Diameter: 22mm
Stator Height: 17mm
Motor Mount: 16*19mm(M3*4)

MOTOR TECHNICAL DATA:

| Item NO. | Volts(V) | Prop | Throttle(%) | CURRENT (A) | Power(W) | Thrust (g) | PRM | Efficiency (g/W) | Operating temperature(℃) |
|---|---|---|---|---|---|---|---|---|---|
| D2836-1450KV(EVO) | 12.25 | APC9x6 | 40% | 9.81 | 120.2 | 618 | 7540 | 5.1 | 43.5℃ |
| | 12.04 | | 50% | 13.06 | 157.2 | 743 | 8280 | 4.7 | |
| | 11.91 | | 60% | 16.21 | 193.1 | 879 | 8883 | 4.6 | |
| | 11.82 | | 70% | 19.69 | 232.7 | 995 | 9475 | 4.3 | |
| | 11.63 | | 80% | 24.92 | 289.8 | 1178 | 10239 | 4.1 | |
| | 11.60 | | 90% | 32.92 | 381.9 | 1429 | 11194 | 3.7 | |
| | 11.58 | | 100% | 37.35 | 432.5 | 1535 | 11602 | 3.5 | |
| | 15.97 | APC8x4 | 40% | 12.13 | 193.7 | 704 | 10860 | 3.6 | 44.2℃ |
| | 15.82 | | 50% | 16.29 | 257.7 | 847 | 11467 | 3.3 | |
| | 15.67 | | 60% | 19.77 | 309.8 | 945 | 12423 | 3.1 | |
| | 15.70 | | 70% | 25.24 | 396.3 | 1108 | 13360 | 2.8 | |
| | 15.48 | | 80% | 32.96 | 510.2 | 1292 | 14290 | 2.5 | |
| | 15.26 | | 90% | 43.38 | 662.0 | 1497 | 14668 | 2.3 | |
| | 15.19 | | 100% | 48.69 | 739.6 | 1582 | 16579 | 2.1 | |

Figure 12: Flash Hobby D2836EVO motor technical data [23]

The internal resistance of 33.5 mΩ allows for efficient power use and contributes to thermal stability. The 1450KV rating specifies the motor's RPM per volt, indicating it is optimized for moderate-speed, high-torque operation suitable for larger propellers, in this case, HQProp 9x5x3 carbon-reinforced nylon propellers shown in Figure 13. The 9-inch diameter and 5-inch pitch of these propellers are well-matched to the 1450 KV motor's output characteristics, providing a balance between lift and control while fitting within the dimensional assumptions. The tri-blade design of the propeller also increases surface area, improving grip in the air and enhancing thrust while maintaining control over rotational stability. Constructed from carbon-reinforced nylon, the propellers maintain structural integrity while adding minimal weight of 20 g each to the system, aligning with the thrust and efficiency needs outlined for the drone.


Figure 13: Photo of HQProp 9x5x3 on a scale

The ESCs selected for this drone are Hobbywing Skywalker 60 A RC brushless-compatible models shown in Figure 14**,** with continuous and peak current ratings of 60 A and 80 A, respectively. This rating provides a margin above the motor's maximum current draw of nearly 50 A, ensuring stable operation under typical load conditions without reaching ESC's upper limits. These ESCs are compatible with 3-6S LiPo input, matching the drone's 3S LiPo battery configuration, and include an integrated switch-mode BEC output at 5 V/3 A, which can supply power to low-power onboard electronics.

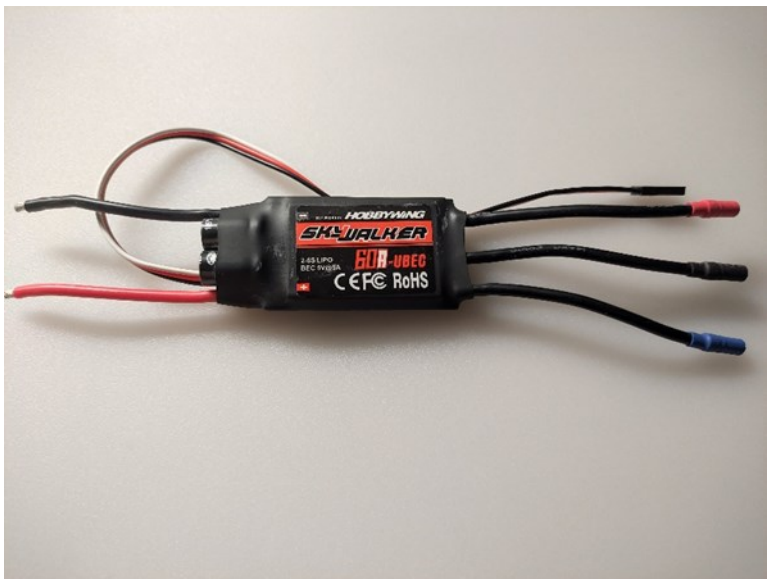



Figure 14: Photo of Hobbywing Skywalker ESC

The choice of a brushless ESC type aligns with the brushless motors selected for the propulsion system, allowing for efficient motor control and reduced wear compared to brushed ESCs. Each ESC weighs 63 g, a reasonable addition to the drone's overall mass. These ESCs were chosen not only for their compatibility with the motor specifications but also for their cost-effectiveness, balancing performance with budget constraints.

## 7.2 Power Supply

For the drone's power source, a 3S2P configuration of available at hand Melasta LiPo cells (SLPBB042126) was selected to provide a balance between capacity, discharge rate, and operational stability. This setup provides a nominal voltage of 11.1 V, suitable for the ESC and motor requirements (3S compatibility).

Lithium Polymer (LiPo) batteries are widely used in drone applications due to their high energy density, lightweight structure, and ability to deliver high discharge rates. Unlike traditional lithium-ion batteries, LiPo cells use a flexible polymer electrolyte, which allows for prismatic (flat) shapes that optimize space usage, ideal for compact designs where weight and form factors are critical. [24.] LiPo batteries offer a favorable balance of capacity and discharge

capabilities, essential for drones requiring consistent power supply to high-current components like motors and ESCs. However, LiPo chemistry requires careful handling to avoid over-discharge or overheating, as excessive stress can lead to swelling or thermal instability. For this reason, monitoring tools and safety margins are integrated into the drone's power system to leverage LiPo’s benefits while maintaining operational safety.

The 3S2P configuration doubles the capacity of individual cells, yielding a total of 13,200 mAh (13.2 Ah). Given a continuous discharge rate of 15 C, this pack can deliver up to 198 A continuously, sufficient to meet the motors’ combined peak demand of approximately 194.76 A. The pulse discharge rate is 20 C, which provides additional assurance of safety increasing the spike peak capability to 264 A. This setup ensures that, even under maximum load, the battery pack can supply adequate current without compromising voltage stability or risking excessive heat buildup. Each cell's low internal resistance (<1.5 mΩ) supports efficient energy transfer with minimal losses, helping maintain stable voltage levels during flight. [25.]

Using a 3S1P configuration with the same cells would significantly reduce both capacity and current supply, resulting in a lower maximum output of 99 A and a capacity of only 6,600 mAh. Such a setup would fall short of meeting the motors' peak demands, risking voltage drops, overheating, and reduced flight times. Thus, the 3S2P configuration was chosen to ensure sufficient power for both continuous and peak operation, supporting stable performance during dynamic flight and prolonged operational use.

## 7.3 Flight Controller PCB and Integrated Components

The Flight Controller PCB is designed to serve as the central control and integration platform for the drone, including the primary processing, communication, and sensor modules required for operation. The board's components and layout enable data flow and power distribution, ensuring that each subsystem performs its specific role in sync with the overall system.

### 7.3.1 Microcontroller (STM32F446RET6)

The computational needs of the flight controller PCB are satisfied by the STM32F446RET6 microcontroller, which coordinates decision-making, signal generation, and data management for the vehicle.
The STM32F446RET6 microcontroller, based on the ARM Cortex-M4 architecture, was selected for its balance of computational capability, cost, and availability, providing a 32-bit processing core with a maximum frequency of 180 MHz, capable of executing up to 225 DMIPS (Dhrystone Million Instructions per Second). This processing power allows the STM32F446RET6 to perform control tasks and signal processing efficiently, essential for the drone's real-time decision-making requirements. [26.]

The microcontroller handles critical tasks, such as interpreting data from the operator's controller and executing pre-scripted behaviors, which are essential for achieving stable flight and controlled maneuvering. The STM32F446RET6 generates PWM signals to control the ESCs, enabling precise adjustment of motor speeds and ensuring consistent thrust distribution. Additionally, the microcontroller monitors voltage levels from the battery cells, which is essential for power management and safe operation, allowing the system to respond to power drops or critical voltage thresholds.

The microcontroller is configured with multiple communication interfaces, including I2C, SPI, CAN bus, USART, and SWD, to support both current and future expansion needs as shown in Figure 15. Not all interfaces are connected in the initial design, but this range of ports provides versatility for additional

modules, such as servos or a GPS module, which could be added to enhance the system's functionality without requiring major hardware revisions.

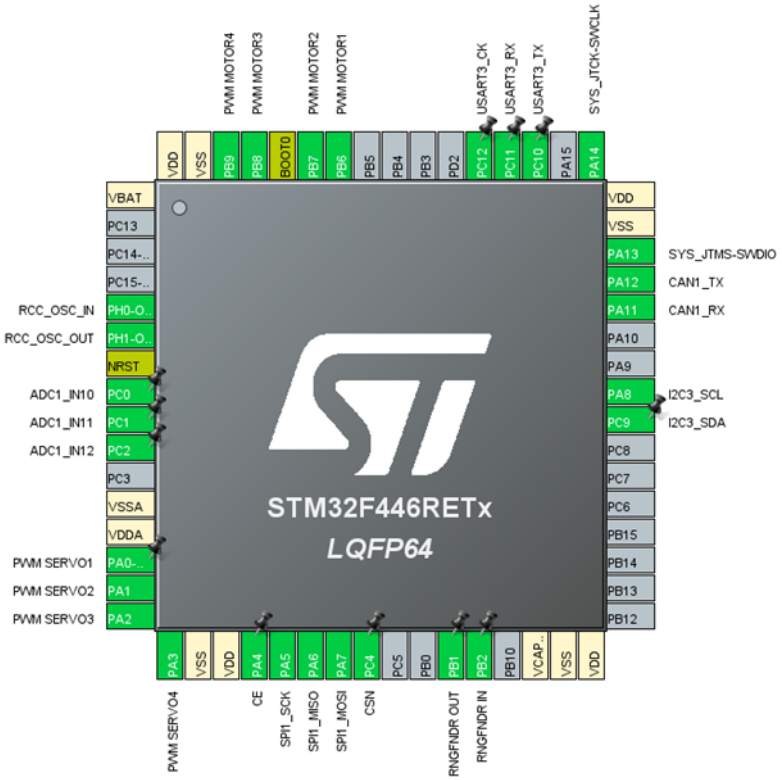



Figure 15: Picture of STM32F446RETx pin configuration in STM32CubeIDE

### 7.3.2 Wireless Communication (NRF24L01P + PA + LNA)

The NRF24L01P module, equipped with a power amplifier (PA) and low-noise amplifier (LNA), is used for communication between the drone and the operator's controller. Operating in the 2.4 GHz ISM band with GFSK modulation, this module can transmit at a data rate of up to 2 Mbps, which is sufficient for the exchange of control commands and status updates without requiring excessive bandwidth [27].

The choice of NRF24L01P was influenced by its affordability and widespread availability, making it suitable for entry-level UAV systems. It connects to the microcontroller via the SPI interface, ensuring a straightforward, low-latency communication pathway. The PA and LNA extend the effective communication

range beyond 100 meters, which is beneficial for increased safety precautions during test flights, enhancing operator control flexibility. Figure 16 shows the module with installed antenna.

Figure 16: Photo of an NRF24 module breakout board.

### 7.3.3 Inertial Measurement Unit (BNO055)

The BNO055 IMU is connected via I2C and integrates a 9-DOF sensor suite, combining an accelerometer, gyroscope, and magnetometer in a single package. The BNO055's onboard sensor fusion algorithms consolidate raw data into simplified orientation output, available as Euler angles or quaternion data. [28.] By handling sensor fusion internally, the BNO055 reduces the computational load on the STM32F446RET6, ensuring that processing resources remain available for control and communication tasks. Module has an approachable prototyping possibility through the use of Adafruit development board shown in Figure 17.

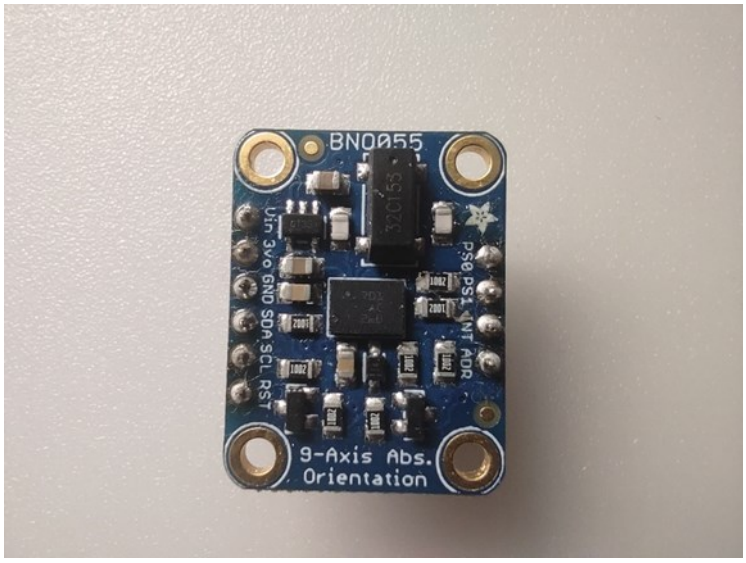



Figure 17: Photo of Adafruit BNO055 development board

This IMU was chosen for its comprehensive feature set relative to cost, as well as its capacity to deliver 100 Hz output data, which is sufficient for real-time orientation adjustments in flight. The sensor's integration of multiple sensing capabilities into a single unit also reduces PCB space requirements, allowing for a more compact board design.

### 7.3.4 Altitude Sensors (BMP180 and RCWL-1670)

The BMP180 barometric pressure sensor, also connected through I2C, is utilized for altitude measurement. The module development board shown in the Figure 18, is used for prototyping. With a pressure range of 300–1100 hPa and an altitude resolution of 0.17 meters, the BMP180 provides adequate accuracy for general altitude tracking at higher elevations [29]. The decision to use the BMP180 is based on its affordability and availability for UAV applications, where high-precision barometric data is not strictly required.

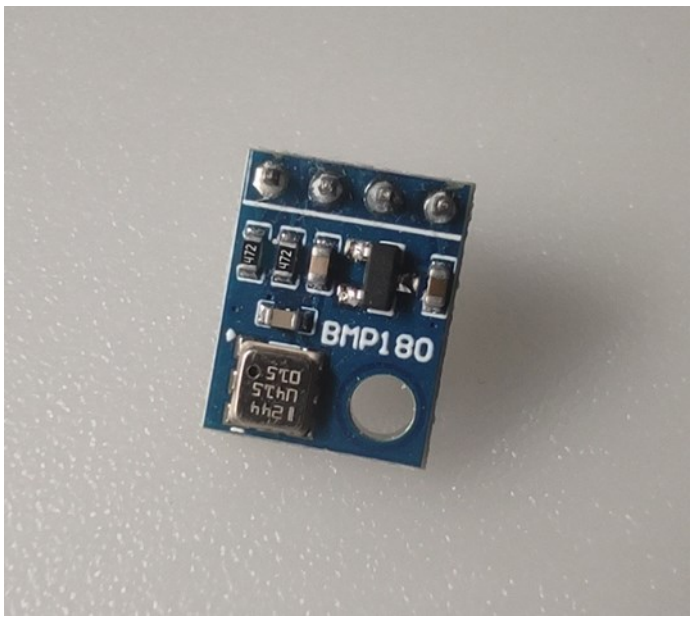



Figure 18: Photo of BMP180 development board

For near-ground altitude detection, the RCWL-1670 ultrasonic range finder is included, connected through GPIO. The standalone module is shown on Figure 19. This sensor measures distances from 2 cm up to 4 meters, providing more accurate altitude readings during low-flight or landing operations where barometric sensors may be less reliable [30]. This dual-sensor approach provides redundancy, with the RCWL-1670 enabling safer near-ground navigation and more controlled landings.


Figure 19: Photo of RCWL 1670 module

### 7.3.5 Future Expansion Potential

Although the current PCB design forgoes certain optional modules, such as a GPS, the availability of communication interfaces (for example USART) allows these components to be added in future revisions. This modularity ensures that the drone's capabilities can be expanded without requiring significant changes to the existing hardware, making the flight controller flexible for additional functions if future mission requirements evolve.

In summary, the Flight Controller PCB integrates key control, communication, and sensor components in a layout optimized for responsiveness, stability, and adaptability. Each element has been selected to fulfill a core role within the drone's control and navigation framework, while the board design enables flexibility for future growth and additional features as needed.

# 8 Controller's Sensors, Devices and Components

The controller's design centers around intuitive input and real-time feedback, allowing the operator to manage the drone's navigation, behavior, and status effectively. To achieve this, the controller integrates two analog joysticks, four tactile switches, two OLED displays, an indicator LED, and an MPU6050 motion sensor, all coordinated by an STM32G431KBT6 microcontroller. Wireless communication is handled by an NRF24 module, which interfaces directly with the drone's communication system, allowing seamless bidirectional data transfer. Each part was selected to balance usability, durability, and compatibility, supporting both control inputs and feedback to the operator. The modular selection of components allows for reliable input handling, wireless communication, and future system adaptability.

## 8.1 Microcontroller (STM32G431KBT6)

The STM32G431KBT6 microcontroller, based on the ARM Cortex-M4 core, serves as the controller's processing unit. The 32-bit processing core operating at a maximum clock speed of 170 MHz of this microcontroller is capable of

handling multiple analog and digital inputs, as well as managing real-time communication with the drone at up to 213 DMIPS.
Key features of the STM32G431KBT6 utilized by the controller include six GPIOs acting as inputs for tactile switches, multiple ADC channels for reading analog joystick inputs, and communication interfaces such as I2C, SPI, and Serial Wire Debug. These interfaces support seamless integration of both essential components and any additional modules that may be added in future revisions. This microcontroller's relatively low cost and availability make it suitable for this application, where essential functions are prioritized over high processing demands. [31.]

## 8.2 Wireless Communication (NRF24L01P + PA + LNA)

The NRF24L01P module with PA and LNA, identical to the module on the drone, handles wireless communication between the controller and the drone. The use of the same module on both ends ensures compatibility in data transmission protocols and allows for streamlined development of communication within the 2.4 GHz band. In the controller, the module's role is to transmit control commands and receive status feedback, maintaining crucial capabilities for real-time drone operation. Connected to the STM32G431KBT6 through the SPI interface, the design mimics the counterpart on the vehicle-side. The selection of this module for the controller was influenced by its availability, cost-efficiency, and capacity to support robust communication without requiring excessive bandwidth.

## 8.3 Analog Joysticks

The controller includes two analog joysticks, which serve as the primary input method for controlling the drone's movement. Each joystick consists of two potentiometers, one for the x-axis and one for the y-axi, that provide variable resistance in response to user movement. These analog inputs are read by the STM32G431KBT6's ADC channels, which convert the position of each joystick into digital values that the microcontroller uses to generate control signals for the drone's movement. This analog-based input method offers 12-bit precision,

allowing fine control over throttle, pitch, roll, and yaw while spring-loaded mechanism returns them to a neutral position automatically. These devices also include a tactile switch feature, providing digital button capability in addition to the analog readings.

## 8.4 Tactile Switches

Four tactile switches are included to provide additional input for navigating through menus or modifying settings on the controller. Each switch is connected to a GPIO pin on the STM32G431KBT6, allowing the microcontroller to detect button presses. These switches are used for non-directional commands, enabling the operator to access configuration menus, toggle modes, or adjust parameters. The tactile switches provide straightforward functionality, contributing to the controller's overall simplicity and ease of use.

## 8.5 OLED Displays

The controller includes two 0.96-inch OLED displays to provide clear feedback to the operator. OLED (Organic Light-Emitting Diode) technology allows each pixel to emit light independently, providing high-contrast visuals even in varying lighting conditions while minimizing power consumption. These displays connect via I2C to the STM32G431KBT6 and have two available addresses to avoid problems in integration. One display is dedicated to menu navigation, allowing the user to access various control options and configuration settings. The second display shows real-time drone status information, such as battery levels and communication status, helping the operator monitor critical data without interrupting control.

## 8.6 Indicator LED

A single indicator LED is incorporated to provide simple status cues, such as connection confirmation, low battery alerts, or operational states. The LED is controlled by a GPIO pin on the STM32G431KBT6 and can be programmed to stay lit, blink, or flash patterns depending on the status. This indicator serves as

an accessible and non-intrusive feedback tool, helping the operator remain informed without the need to reference the display frequently, along with acting as a debugging tool during development phase.

## 8.7 Accelerometer (MPU6050)

The MPU6050 motion sensor combines a three-axis accelerometer and a three-axis gyroscope, offering 6-DOF (six degrees of freedom) in a single compact package. This sensor operates on I2C, which simplifies integration with the STM32G431KBT6, and includes a DMP (Digital Motion Processor) for basic sensor fusion. [32.] While the MPU6050 offers a fraction of the capabilities found in BNO055 (such as lacking magnetometer data), it is suitable for tilt-based input, providing acceleration and rotational velocity data that can translate tilting motions into control commands. Its cost-effectiveness and lower computational demand make it a practical choice for testing and experimenting with motion-based control in the controller.

## 8.8 Summary

The controller’s components work in combination to provide input and real-time feedback, allowing the operator to control the drone remotely. With STM32G431KBT6 managing data from both the joysticks and motion sensor while coordinating OLED feedback and wireless communication, this setup offers a mix of control, simplicity, and flexibility. Each component was chosen to meet the controller’s primary functionality requirements, with attention to availability, cost, and compatibility to ensure that the controller supports reliable, effective drone operation in a streamlined format. The accelerometer provides an additional approach to the topic of control and allows an investigation of this feature.

# 9 Design of Printed Circuit Boards

The implementation of the drone system involves translating the design decisions from previous stages into functional printed circuit boards (PCBs) using KiCAD, an open-source electronic design tool. This process begins with schematic creation, where each component is represented by abstract symbols with various information assigned to them. When the schematic is prepared and every component has a footprint assigned, the PCB layout creation starts, which encompasses placement of components in their destination and connecting every logical connection (net) with the use of traces, polygons and vias. Emphasis is placed on the practical application of electronic design principles, such as EMC. This allows for the transition from conceptualization to a working system.

## 9.1 Drone

The overall schematic of the drone's flight controller, shown in Figure 20, serves as the foundation for its system integration. The schematic represents the relationships between the microcontroller, sensors, power management circuits, and communication modules, forming the design to meet the drone's requirements.

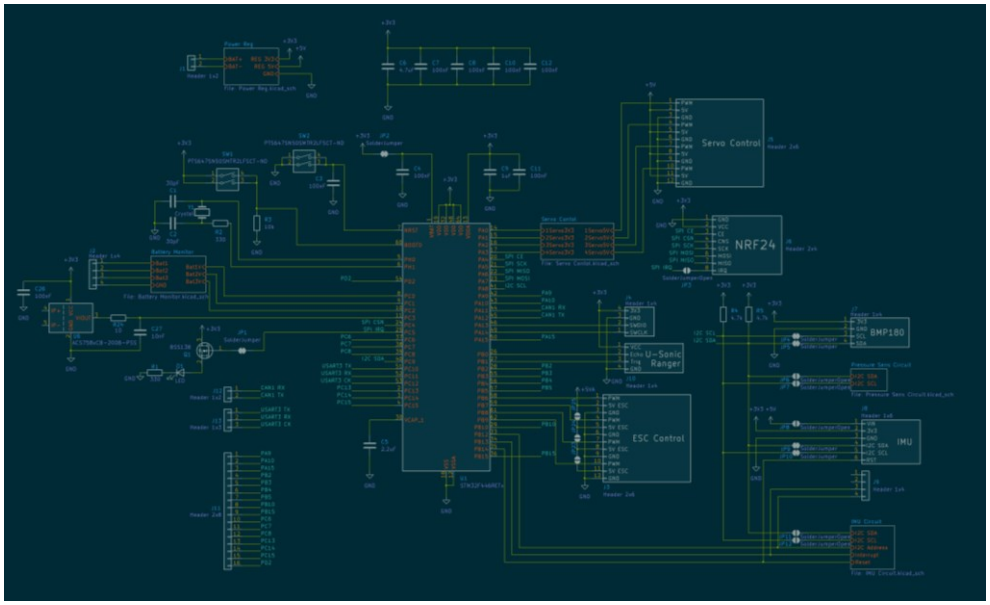



Figure 20: Schematic of the Drone Flight Controller

The designed prototype PCB layout is illustrated in Figure 21, with a two-layer structure. The choice of a two-layer design minimizes fabrication costs while supporting prototyping goals. Ground planes occupy most of both layers, ensuring consistent grounding and limiting electromagnetic interference (EMI). Vias throughout the board connect the ground planes to improve conductivity and reduce capacitive coupling effects. The board measures 120x51 mm, with four mounting holes to facilitate integration with the drone's frame.

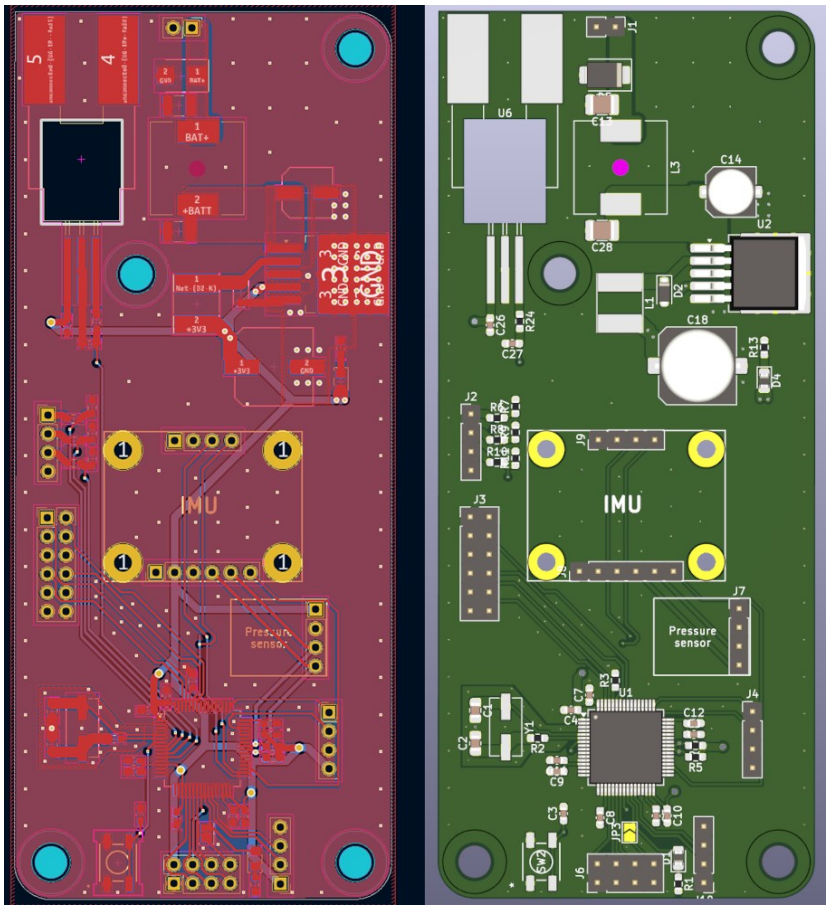



Figure 21: Board layout and 3D render side by side

The right side of Figure 21, shows the 3D render of the PCB. This visualization helps in ensuring that all components, including header connectors, align physically with the design intent and that sufficient spacing exists between components for accessibility during assembly.

The power input system is designed with safety and stability in mind. The circuit, shown in Figure 22, begins with a unidirectional TVS diode (SM6T15A),

which protects the system by clamping input voltages above 15 V to safe levels. This diode has a peak pulse power dissipation of 600 W and a leakage current of less than 1 µA at 12 V, ensuring minimal impact on nominal operation. [33.]

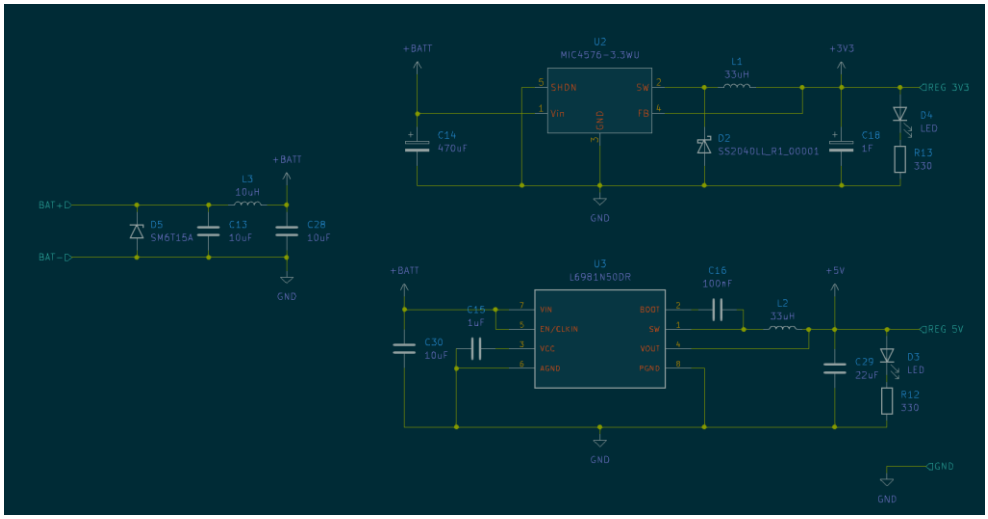



Figure 22: Power Input, Overvoltage Protection, and Voltage Regulation

Following the TVS diode, a Pi-filter configuration reduces noise and ripple in the input power. As shown in the same figure, this filter comprises two 10 µF ceramic capacitors and a 10 µH inductor. The X7R dielectric of the capacitors ensures stability across temperature variations, and the inductor is selected for its high saturation current rating (6.5 A), supporting stable operation even under dynamic load conditions.

Two voltage regulators are included in the schematic:

- **MIC4576-3.3WU** shown on the top right in Figure 22: A buck regulator providing 3.3 V output at up to 3 A with an efficiency of up to 92%. Supporting peripherals include a Schottky diode for flyback current control, an input/output smoothing capacitor, and an inductor rated above 3 A to ensure stable operation.
- **L6981N50DR** shown on the bottom right in Figure 22. A 5 V buck regulator capable of delivering up to 1.5 A. While excluded from the prototype (since no 5 V peripherals are present), it is included in the design for potential expansions requiring 5 V, such as servo motors.

Copper polygons and wide traces on the regulator layout, as highlighted in Figure 23, limit resistance and improve current handling capability. Numerous vias placed on the grounding pad of the regulator ensure improved thermal exchange with the copper ground plane, reducing heat buildup. The 3.3 V is being distributed to components through a main rail highlighted in Figure 24.

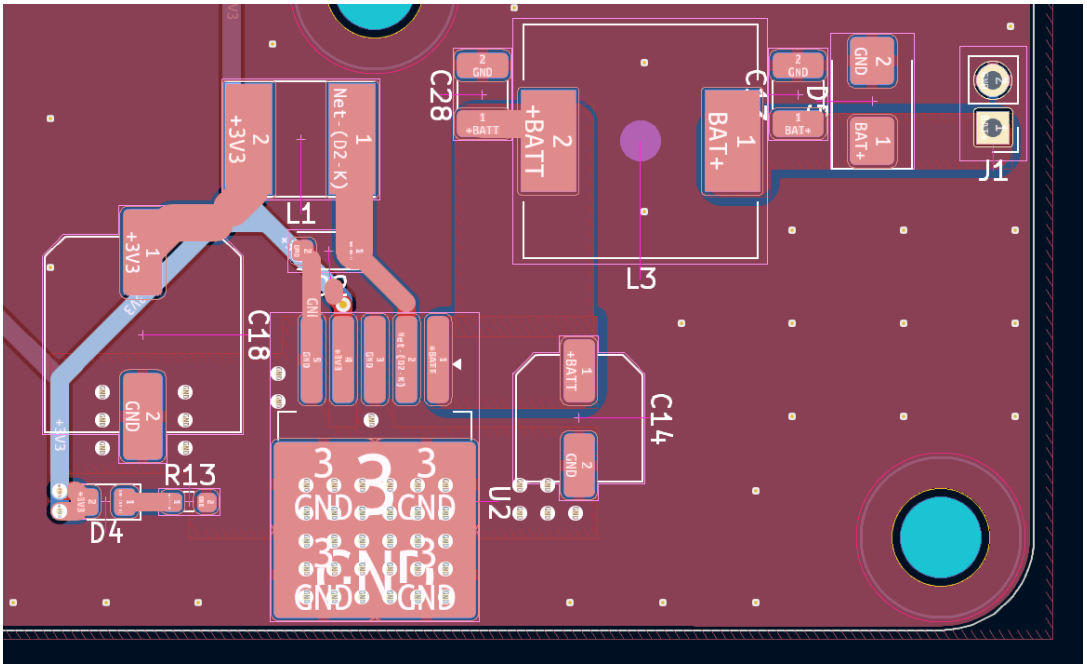



Figure 23: Circuit protection and voltage regulation layout

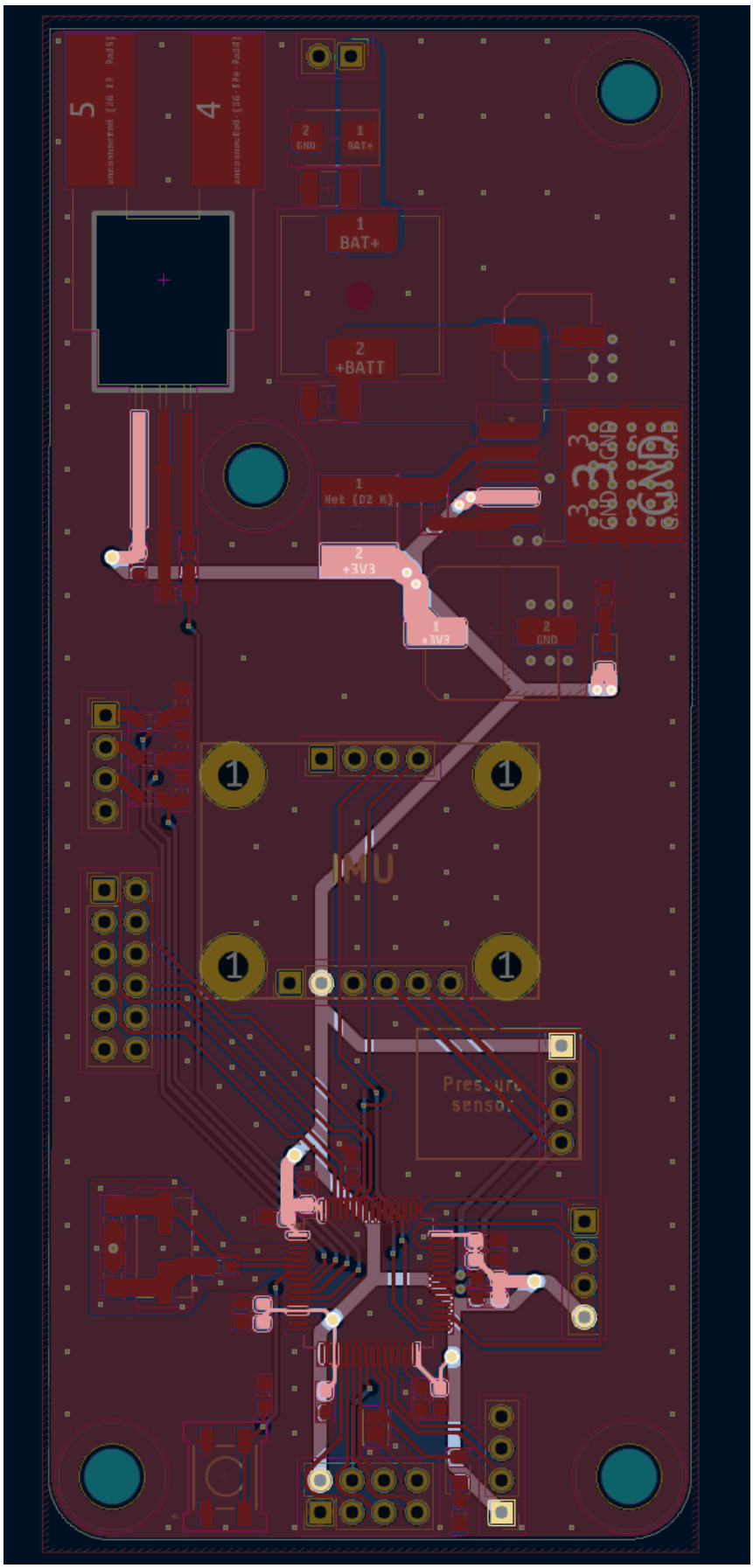



Figure 24: Highlight of 3.3 V rail on the board

Resistance at the longest path from the source through the rail is calculated using this formula:

$$R = \rho \cdot LT \cdot W \cdot [1 + \alpha \cdot (temp - 25)] \quad (1)$$

Where:
P = resistivity
α = temperature coefficient
L = length = 95 mm
W = trace width = 1 mm
T = trace height = 0.0348 mm
$\rho_{copper}$ = $1.7 \cdot 10^{-6}$ Ω-cm
$\alpha_{copper}$ = $3.9 \cdot 10^{-3}$ / °C

The resulting (1) resistance is 0.046 Ω is a relatively low value and suggests efficient current flow with minimal power loss.

The STM32F446RET6 microcontroller, its peripherals, and integration are shown in Figure 25. Supporting components include:

- Decoupling capacitors consist of a single 4.7 uF and 0.1 µF near each power input pin to stabilize voltage.
- Boot button for firmware flashing in case of problems with System Serial Wire.
- Reset button for system reinitialization.

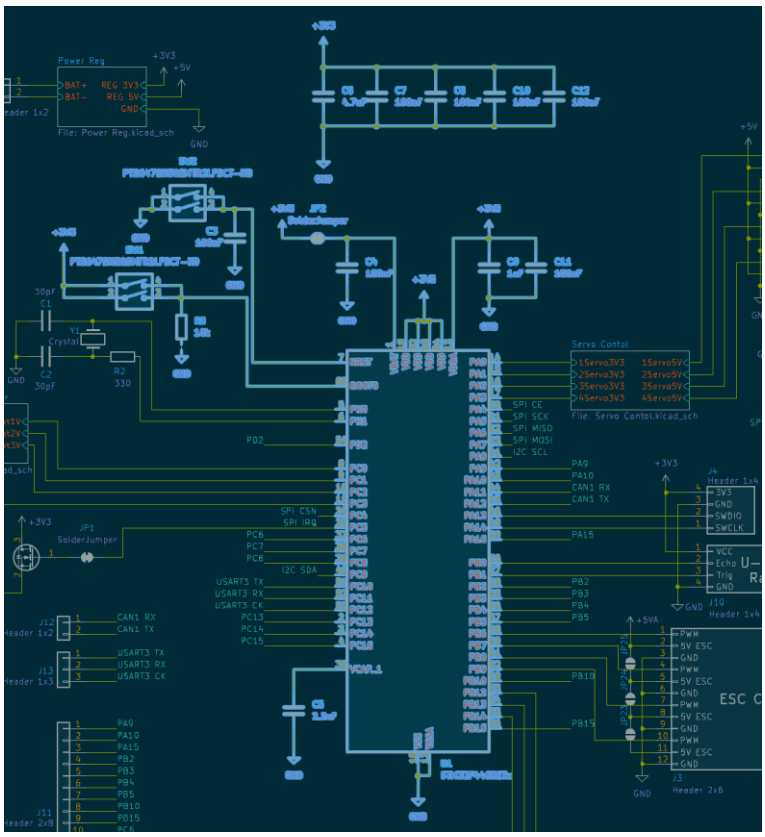



Figure 25: Microcontroller and peripherals highlighted on the schematic

As shown in the PCB layout in Figure 26, bypass capacitors are placed close to the power pins for optimal performance. The boot button is omitted in this prototype for simplicity, while the reset button is located on the bottom left corner for easy access.

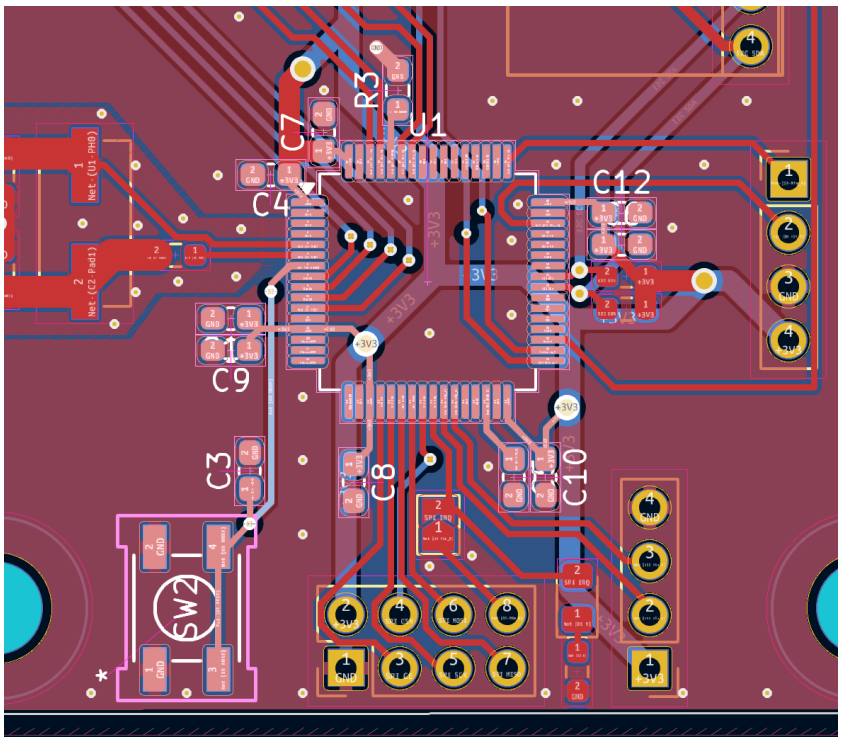



Figure 26: Layout of the microcontroller and its peripherals

The microcontroller's clock signal is provided by a 16 MHz ECS-160-20-3X-TR crystal oscillator, as shown in Figure 27. The circuit includes 30 pF load capacitors and a series resistor of 330 Ω to stabilize oscillation and reduce harmonics. The following design is abiding to guidelines provided by ST [34].

The load capacitance ($C_L$) for oscillator is calculated using the following formula:

$$C_L = \frac{C_{L1} \cdot C_{L2}}{C_{L1} + C_{L2}} + C_s \tag{2.1}$$

Where:

- $C_L$: Load capacitance specified by the crystal ($20\,\mathrm{pF}$) [35].
- $C_{L1}, C_{L2}$: External load capacitors (assumed equal)
- $C_s$: Stray capacitance, including PCB and pin capacitance ($5\,\mathrm{pF}$)

Rearranging the formula for $C_{L1}$ (equal to $C_{L2}$):

$$20 = \frac{C_{L1}^2}{2 \cdot C_{L1}} + 5 \tag{2.2}$$

Solving for $C_{L1}$:

$$C_{L1} = C_{L2} = 30\,\text{pF} \tag{2.3}$$

The resulting value (2) is $C_{L1} = C_{L2} = 30\,\text{pF}$. This ensures that the oscillator operates at the correct frequency as specified by the crystal manufacturer.

The external resistor ($R_{\text{Ext}}$) is calculated using the following formula:

$$R_{\text{Ext}} = \frac{1}{2\pi f C_{L2}} \tag{3.1}$$

Where:

- $R_{\text{Ext}}$: External series resistor (in $\Omega$)
- $f$: Crystal frequency ($16\,\text{MHz}$)
- $C_{L2}$: Load capacitor ($30\,\text{pF}$)

Substituting the values:

$$R_{\text{Ext}} = \frac{1}{2\pi \cdot 16 \times 10^{6} \cdot 30 \times 10^{-12}} \tag{3.2}$$

$$R_{\text{Ext}} \approx 332$$

The resulting value (3) is $R_{\text{Ext}} \approx 332$ . This resistor limits the current through the crystal to avoid excessive power dissipation and prevents overdriving the crystal. For practical implementation, the closest E12 series resistor of $330\,\Omega$ is used.

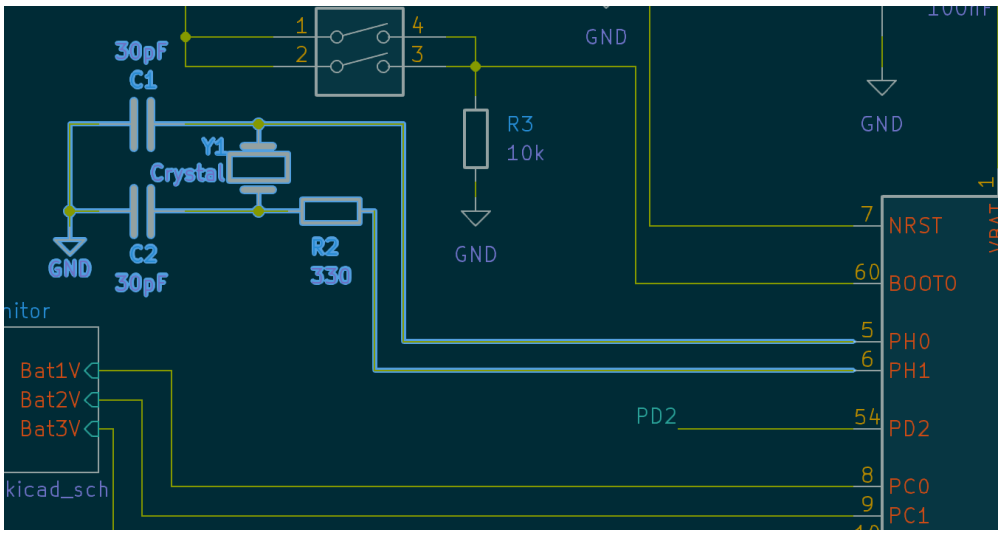



Figure 27: Oscillator crystal circuit

The PCB layout, depicted in Figure 28, highlights the oscillator and its dedicated ground island, which connects to the primary ground plane through vias. The crystal is placed on a ground-isolated island to mitigate EMI effects. [34.]

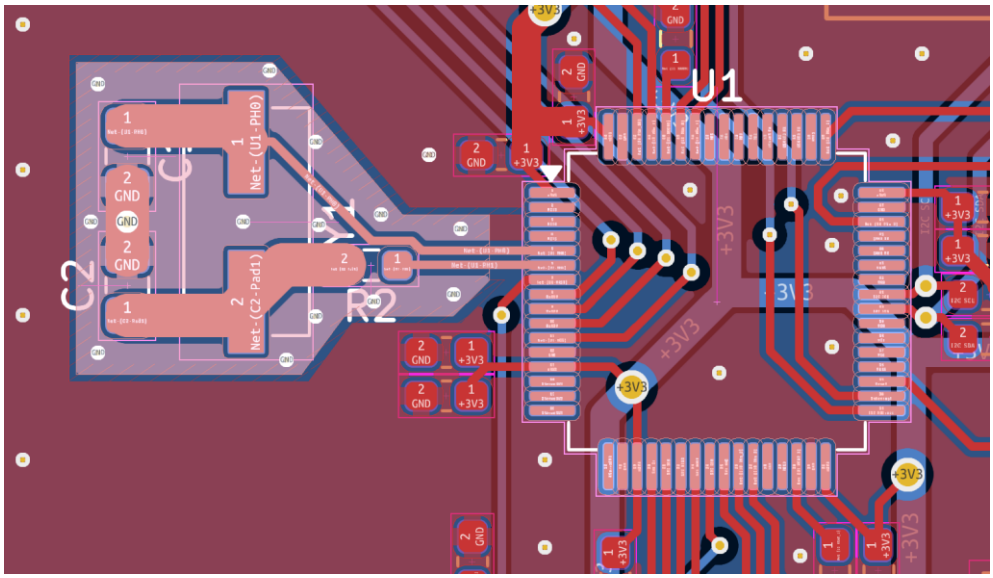



Figure 28: Oscillator crystal circuit layout

To observe the state of charge of the battery pack, a connector and signal processing circuit are implemented as shown on Figure 29.

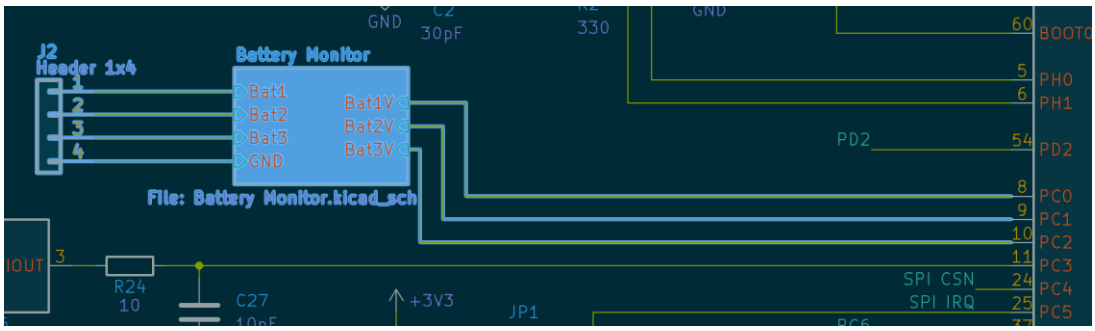



Figure 29: Schematic of battery pack readings input

Voltage dividers, as illustrated in Figure 30, scale the battery pack's 12.6 V maximum voltage (3S configuration) to ADC-compatible levels.

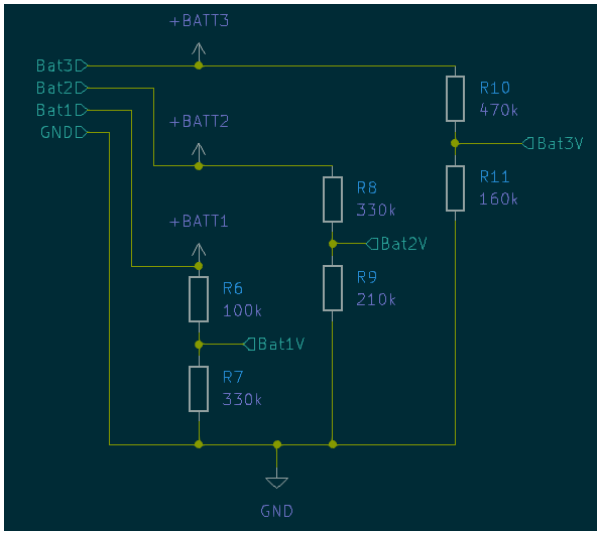



Figure 30: Voltage divider circuit for battery voltage monitoring

Each cell's voltage is divided according to the following formula:

$$V_{\text{out}} = \frac{R_2}{R_1 + R_2} \times V_{in} \tag{4}$$

The resulting values (4) at maximum charge:

- **Bat1**: Reads voltage of a single cell (4.2V) R1 = 100 kΩ, R2 = 330 kΩ, $V_{out} \approx 3.22$ V.
- **Bat2**: Reads cumulative voltage of Bat1 and Bat2 (8.4 V) R1 = 330 kΩ, R2 = 210 kΩ, $V_{out} \approx 3.27$ V.
- **Bat3**: Reads the total voltage of the pack (12.6 V) R1 = 470 kΩ, R2 = 160 kΩ, $V_{out} \approx 3.2$ V.

These dividers ensure that each cell's voltage is scaled appropriately for monitoring by the STM32F446RET6's ADC pins. The PCB layout, shown in Figure 31, illustrates the connection of divider outputs to the ADC pins.

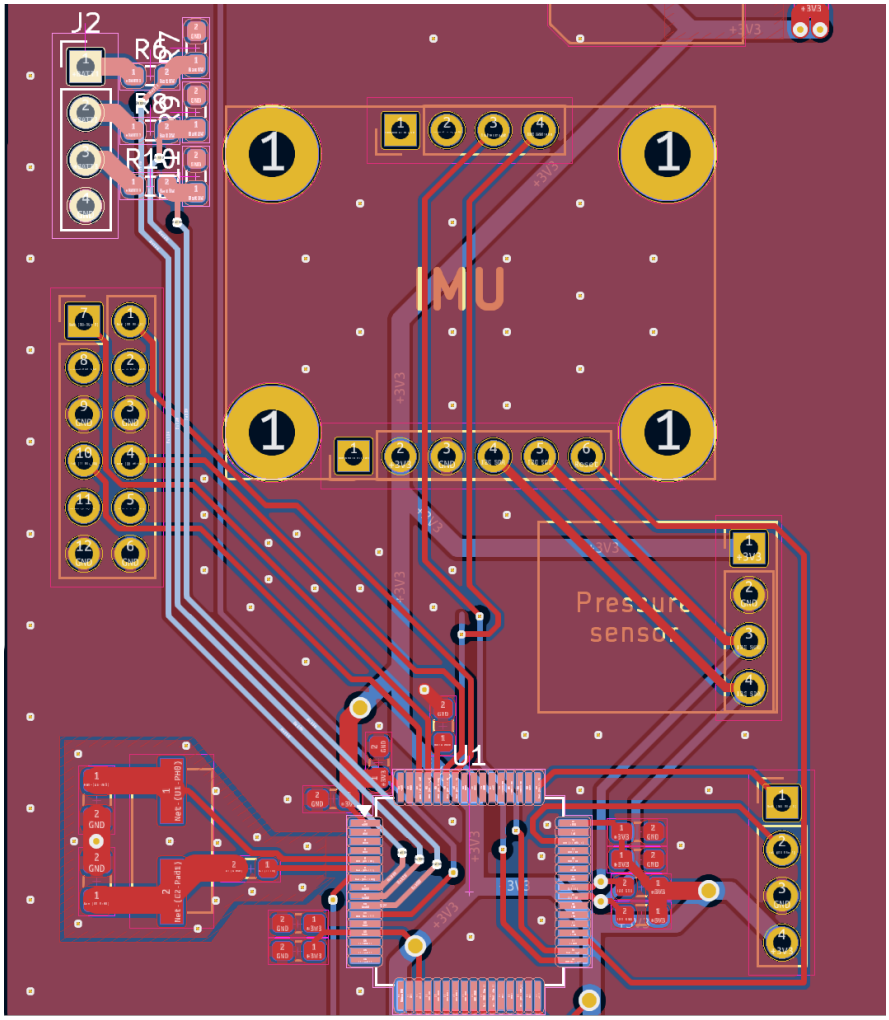



Figure 31: Layout of the battery pack monitoring circuit

The ACS758xCB-200B-PSS hall effect current sensor, as shown in Figure 32, measures current drawn by the motors. This sensor features a 120 kHz bandwidth suitable for detecting dynamic load changes. A low-pass RC filter (10 Ω and 10 nF) is included to suppress high-frequency noise and to smooth the signal before ADC conversion.

The cutoff frequency of the RC low-pass filter is calculated with the following formula:

$$f_c = \frac{1}{2\pi RC} \tag{5.1}$$

Where:

- $f_c$: Cutoff frequency (in Hz)
- $R$: Resistor value ($10\Omega$)
- $C$: Capacitor value ($10\,\mathrm{nF}$)

Substituting the values:

$$f_c = \frac{1}{2\pi \cdot 10\,\Omega \cdot 10 \times 10^{-9}\,\mathrm{F}} \tag{5.2}$$

The resulting value is:

$$f_c \approx 1.59\,\mathrm{MHz}$$

The cutoff frequency indicates (5) that the filter will attenuate signals above 1.59 MHz, allowing the Hall effect sensor's 120 kHz bandwidth to pass through while suppressing higher-frequency noise.

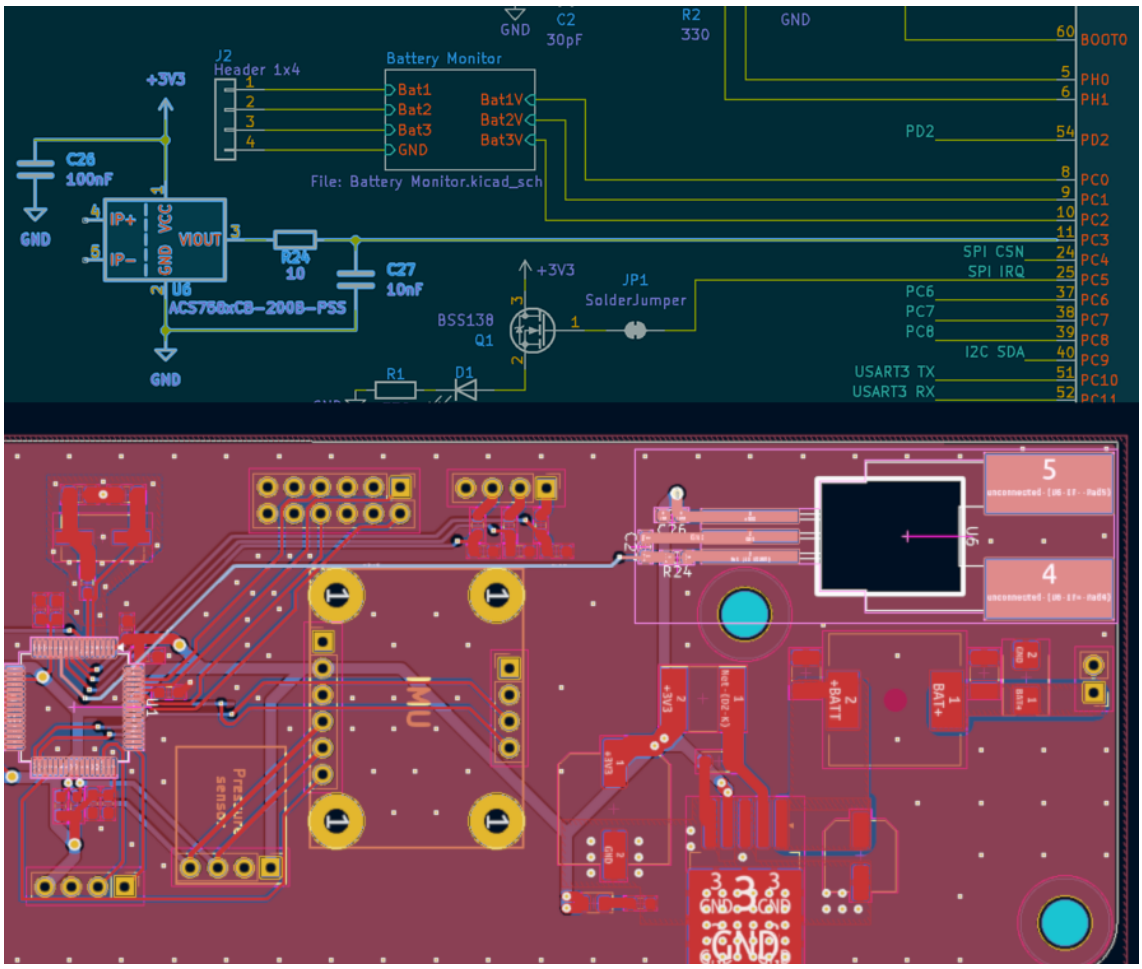



Figure 32: Hall effect sensor highlighted on schematic (top) and layout (bottom)

As depicted in Figure 33, the BNO055 IMU connects via I2C to the microcontroller. Pull-up resistors (4.7 kΩ) ensure proper I2C communication. The prototype PCB includes headers to accommodate an Adafruit breakout board for the BNO055, as shown in Figure 34.

The IMU's placement on the PCB ensures minimal interference from nearby inductive components, and its alignment simplifies centering adjustments.

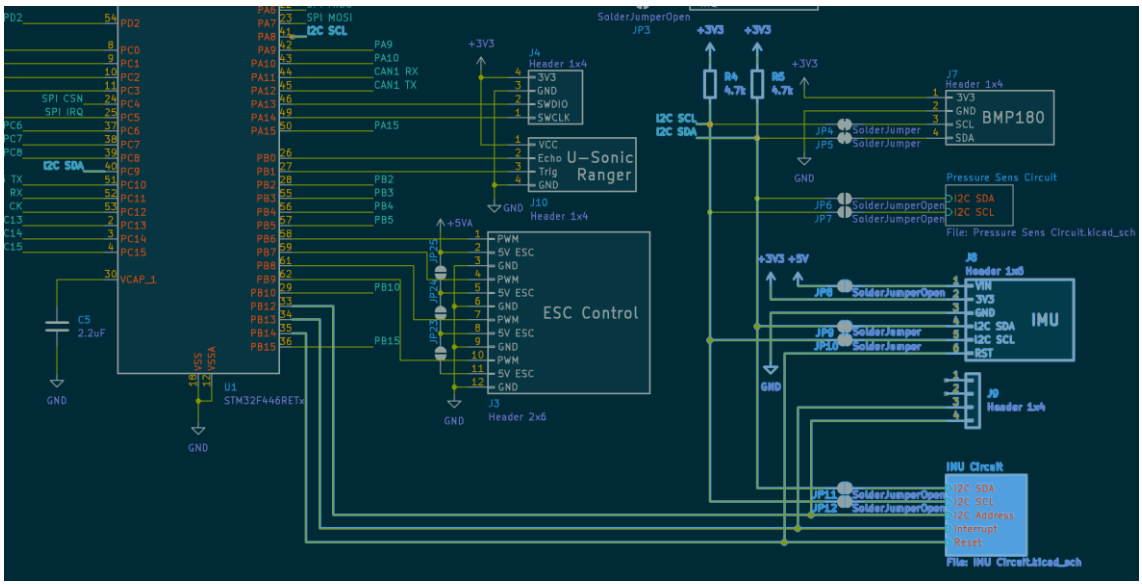



Figure 33: Schematic with IMU circuit highlighted

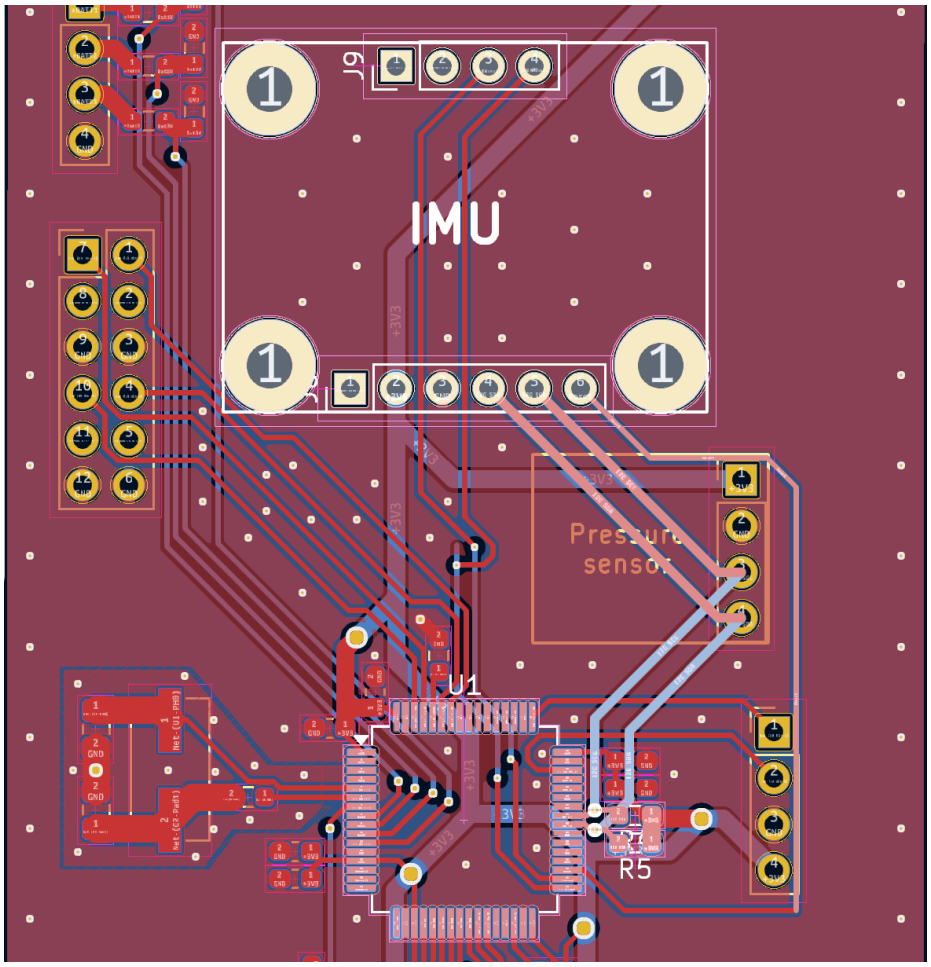



Figure 34: Layout with IMU placement and I2C interface highlighted

The hierarchical sheet visible in the bottom right corner of Figure 33, includes a complete BNO055 circuit schematic with all necessary peripherals like decoupling capacitors and oscillator crystal circuit as shown in Figure 35**.** This design is omitted in the prototype board to reduce costs and time of initial revision implementation.

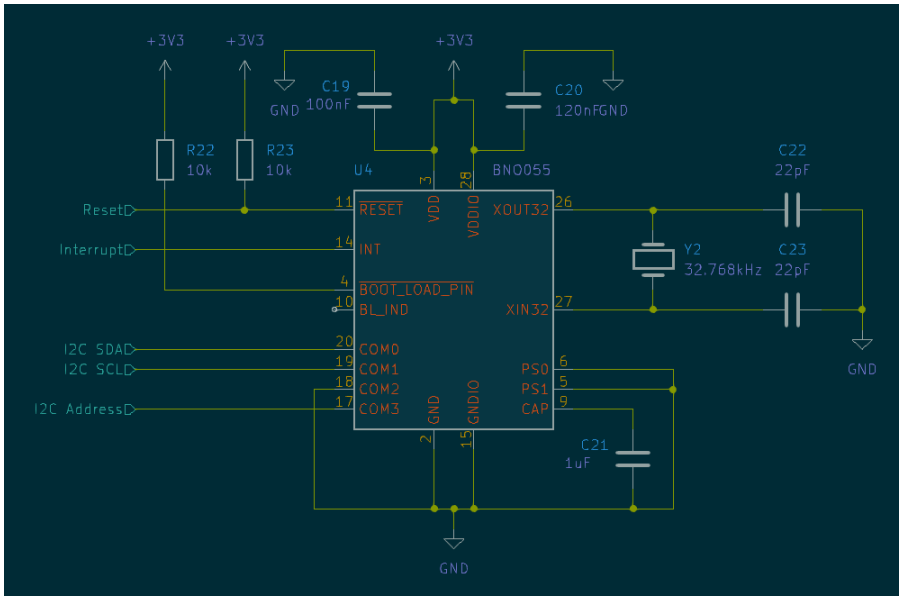



Figure 35: BNO055 absolute orientation sensor circuit schematic

The BMP180 pressure sensor, used for altitude measurement, is integrated via the I2C bus shared with the BNO055 IMU. The schematic in Figure 36, shows the BMP180 connected through a header connector (the top highlighted element) for a development board in this prototype, as well as the hierarchical sheet (lower highlighted element). The hierarchical sheet holds the BMP180 circuit with the recommended peripherals as shown in Figure 37. In the PCB layout shown in Figure 38, the BMP180 outline, and header connector are positioned near the microcontroller, module is sharing the I2C traces with the IMU. A development board is used for prototyping to reduce costs and assembly complexity.

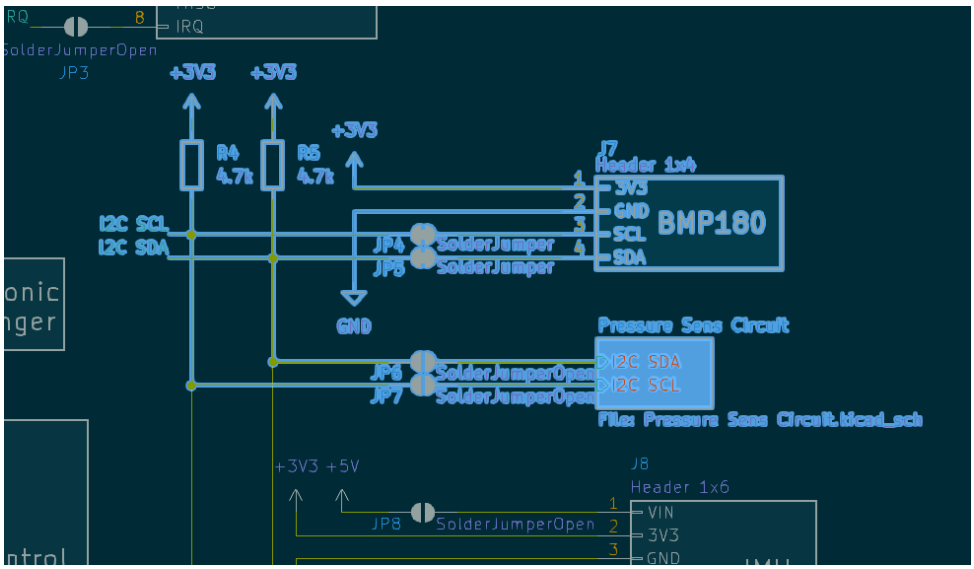



Figure 36: Schematic with BMP180 header connector and hierarchical sheet highlighted

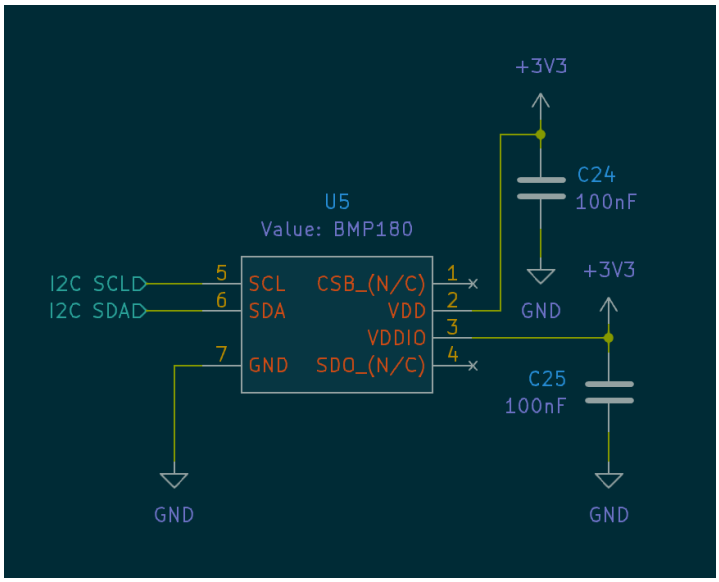



Figure 37: BMP180 pressure sensor circuit schematic

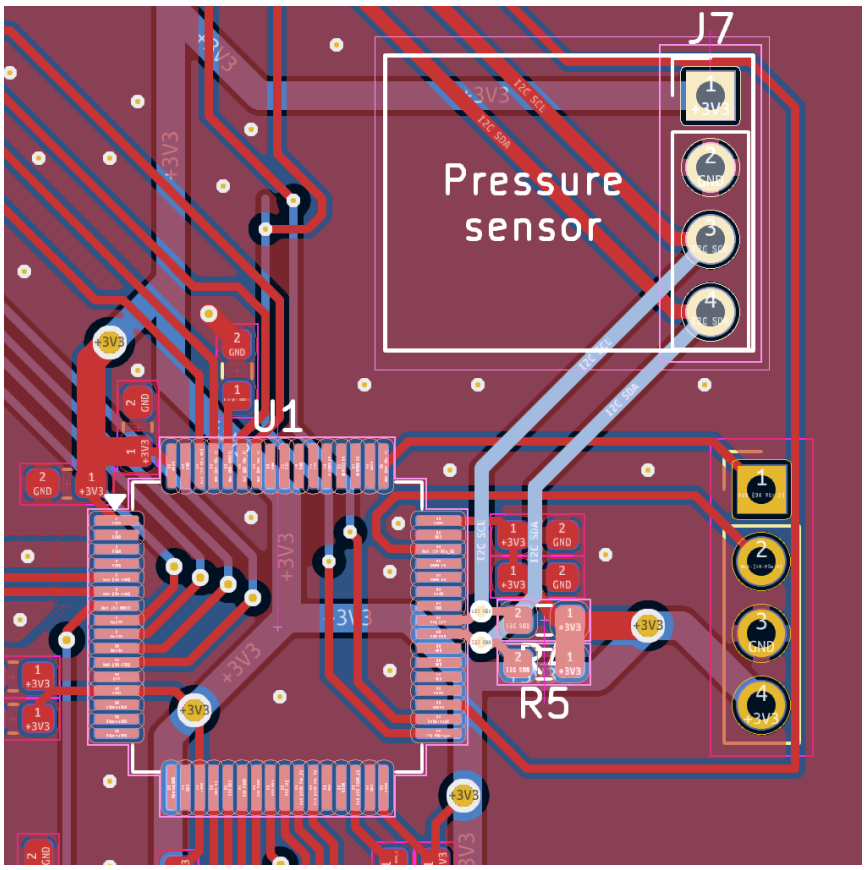



Figure 38: Layout highlighting the pressure sensor position along with the connections to the microcontroller

The RCWL 1670 ultrasonic rangefinder is utilized as an independent module placed on the underside of the drone, due to this, its presence on the flight controller PCB is indicated by a header connector and traces allowing the signal exchange between the module and the microcontroller shown on Figure 39.

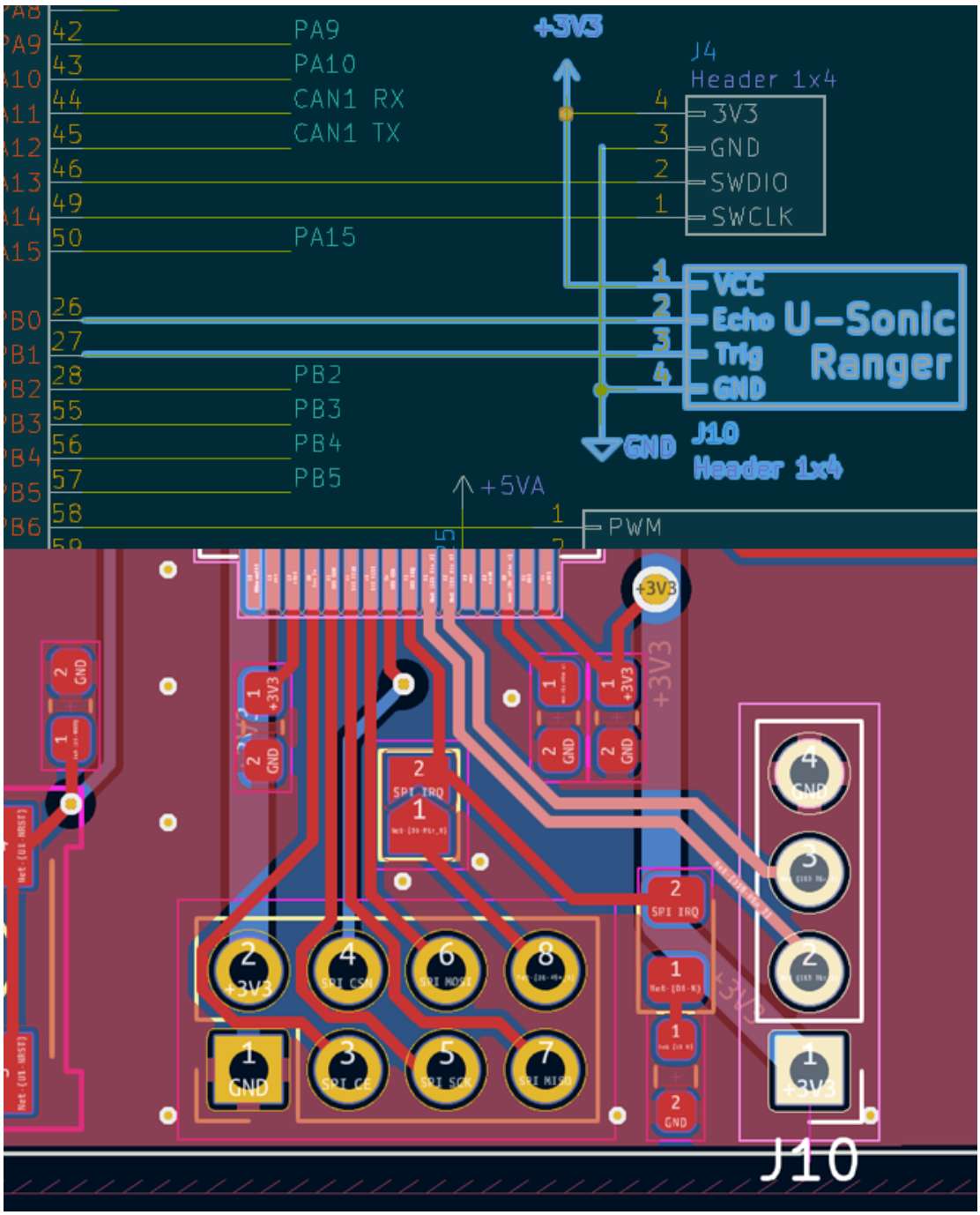



Figure 39: Schematic (top) and layout (bottom) with highlighted RCWL 1670 implementation

The ESCs connect to the microcontroller's PWM outputs through a 2x6 header as shown in Figure 40, each ESC receives PWM signals and ground through this connection. While the ESCs provide their own 5 V regulators, the 5 V pins are not routed to other parts of the PCB to avoid coupling noise or unintended power loops.

The PCB layout in Figure 40, highlights the header placement and direct routing of PWM signals to the corresponding microcontroller pins.

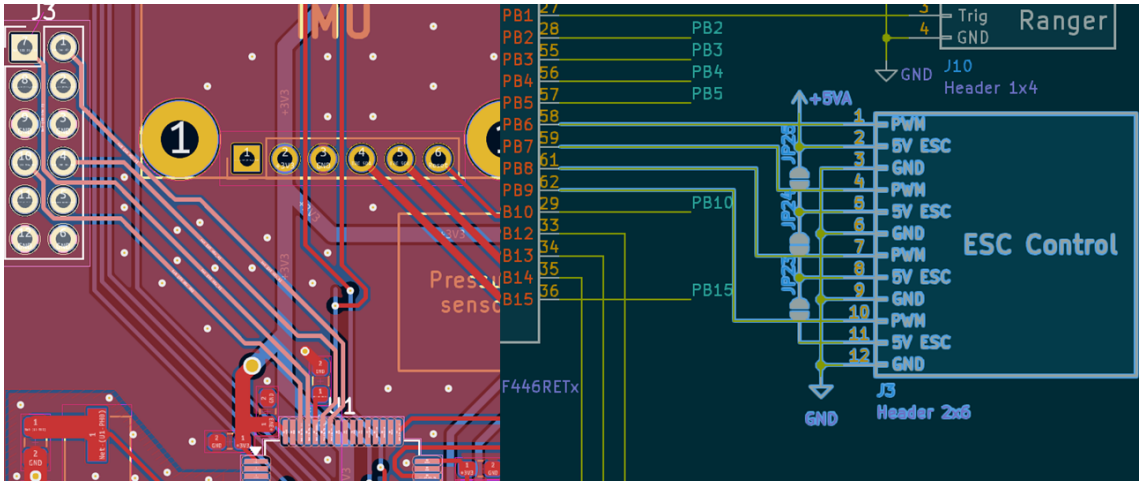



Figure 40: Schematic (left) and layout (right) with highlighted ESC connection circuit

The NRF24L01P module connects to the STM32F446RET6 via an SPI interface, as shown in the schematic in Figure 41. The 2x4 header provides connections for SPI signals (MOSI, MISO, SCK, CSN), module specific CE and IRQ, as well as additional power and ground pins. As seen on the layout part of Figure 41, the module is placed close to the microcontroller to ensure good communication through the SPI, simultaneously it is on the edge of the board, allowing the module to be suspended away from any possible interference that could have the source on the PCB.

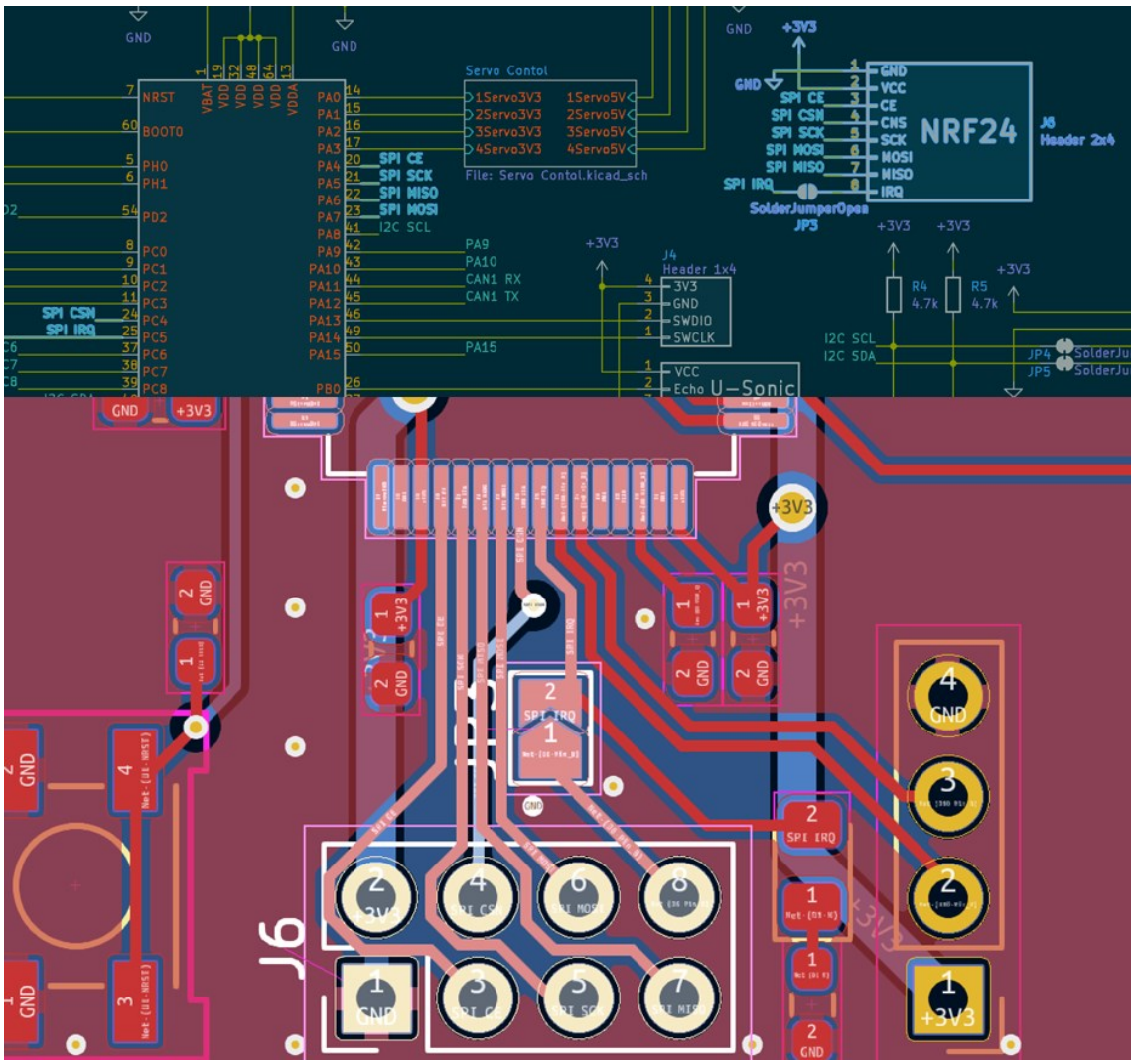



Figure 41: Schematic (top) and layout (bottom) of the NRF24 module connection

The Serial Wire Debug (SWD) interface, shown in the schematic in Figure 42, is connected via a 1x4 header. This interface provides access to the STM32F446RET6's debug and programming capabilities using ST-Link dongle and ST32CubeIDE software, with connections to the SWCLK and SWDIO pins. On PCB the SWD connector is placed close to the microcontroller to ensure good communication when flashing the code as seen on Figure 42.

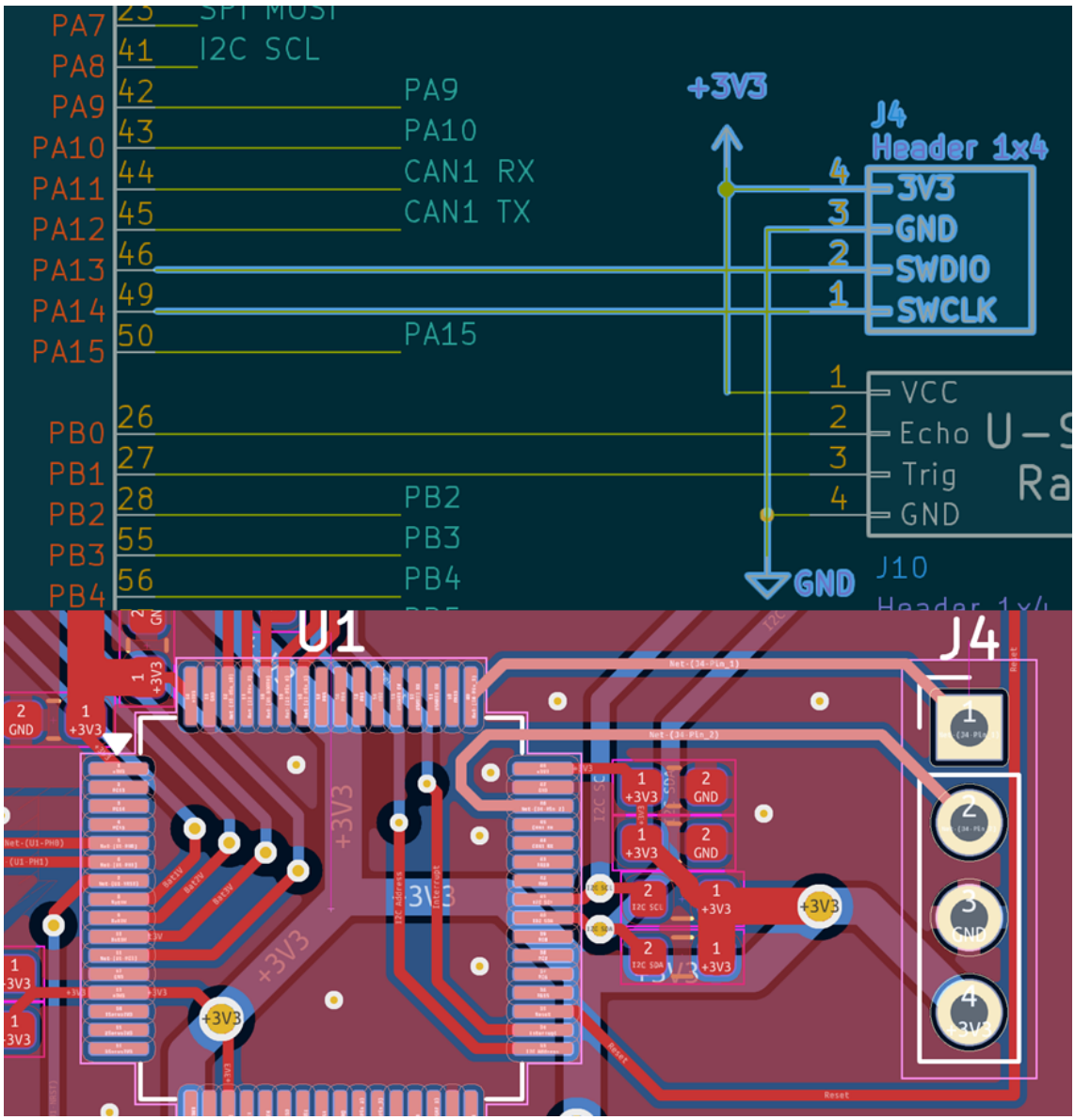



Figure 42: Schematic (top) and layout (bottom) of the SWD connector

Although not present in the prototype, the design includes provisions for servo control using a 2x4 header connected to the microcontroller's PWM pins, as shown in Figure 43. A hierarchical sheet within the schematic, illustrated in Figure 44, shows the signal processing for servos which comprises of level shifting circuits changing the 3.3 V PWM into 5 V PWM compatible with the servos.

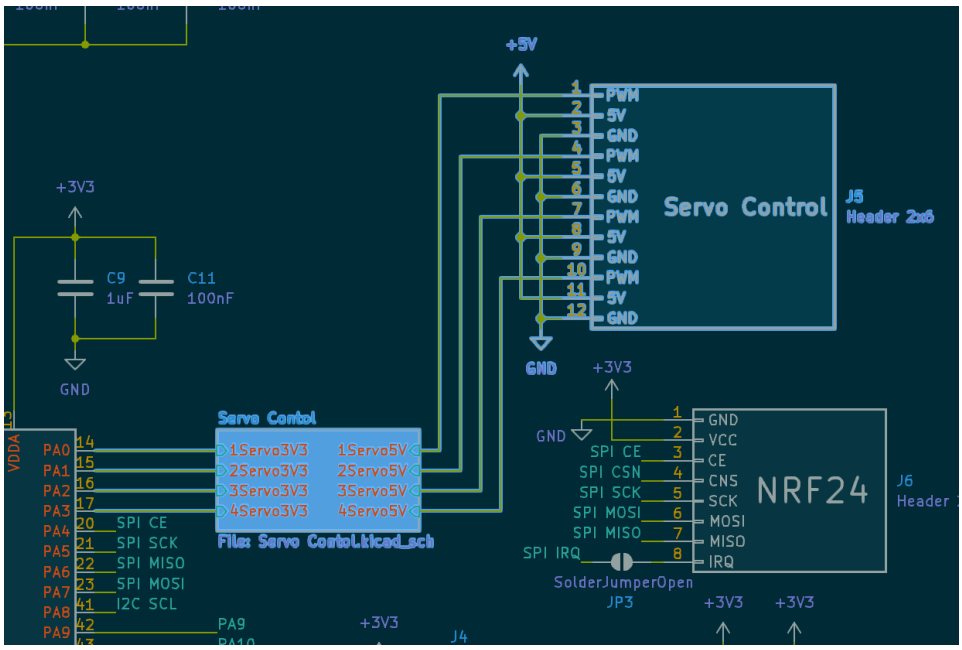



Figure 43: Schematic with highlighted servo control connector and signal control hierarchical sheet

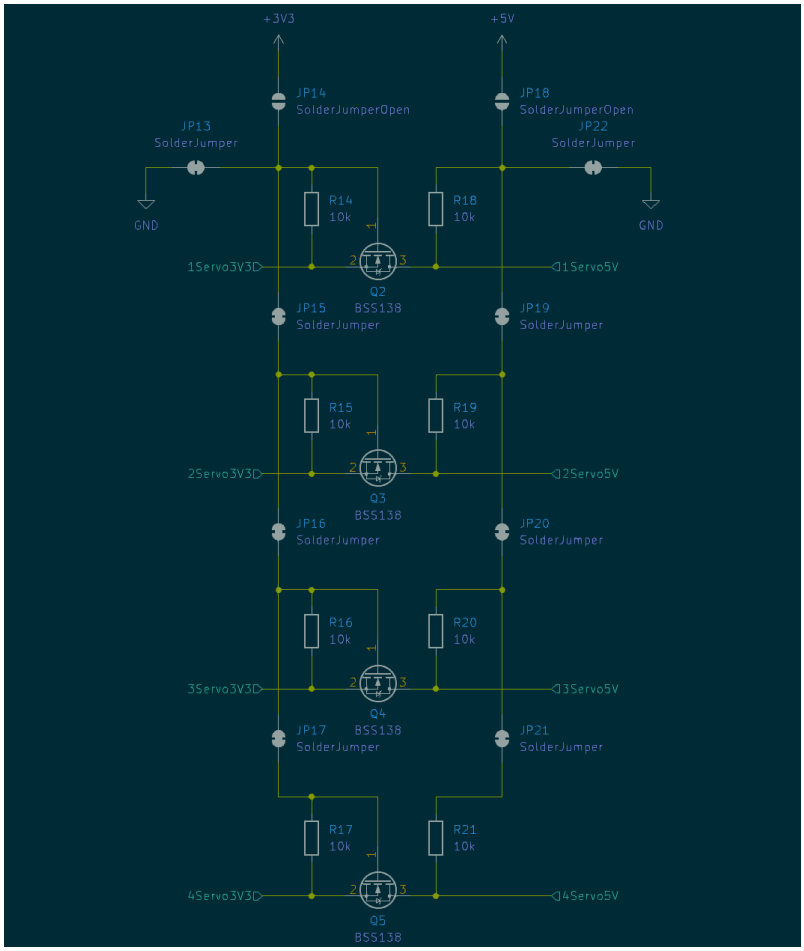



Figure 44: Schematic of signal processing of the servo control

The drone PCB design focuses on integrating essential systems and components to meet the functional requirements defined in earlier chapters. The design prioritizes modularity and compatibility while incorporating significant simplifications tailored to the first prototype revision. These adjustments, such as the use of headers for development boards and a two-layer PCB layout, balance cost and functionality to streamline testing and validation. This implementation establishes a foundational platform for the drone's operation, providing a practical baseline for iterative development and potential future enhancements.

## 9.2 Controller

The implementation of the drone controller focuses on the integration of components defined in earlier chapters. The controller was designed with objectives of simplicity, clarity, and user-friendly interaction. The positioning of input devices is determined through ergonomic assessments, which consider the comfortable and natural positions of human hands. Key circuits, interfaces, and components are detailed below, referencing the schematic and PCB layout for clarity.

The complete schematic of the controller is shown in Figure 45, providing an overview of the relationships between its components, including power management, the microcontroller, and user interface modules. Figure 46, illustrates the PCB layout and 3D render of the controller, with a board dimension of 120x92 mm. The layout closely follows the user interface configuration described in Figure 10, of the system architecture, ensuring ergonomic alignment between physical design and functional requirements.

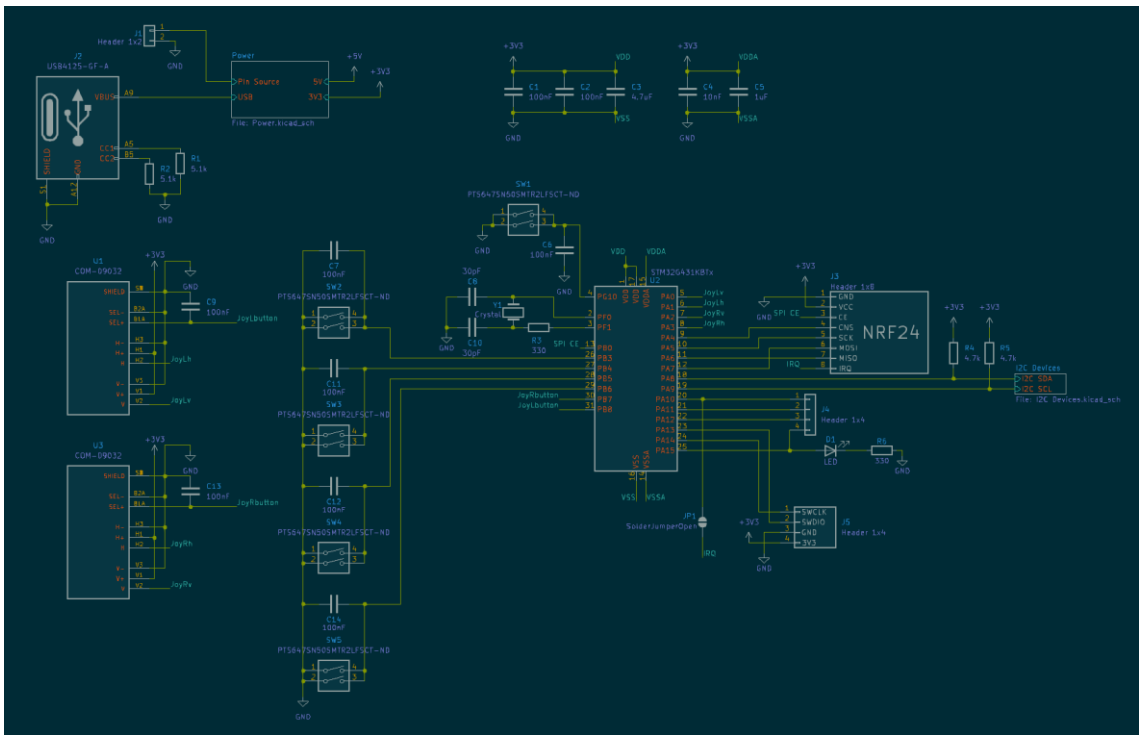



Figure 45: Schematic of the drone controller

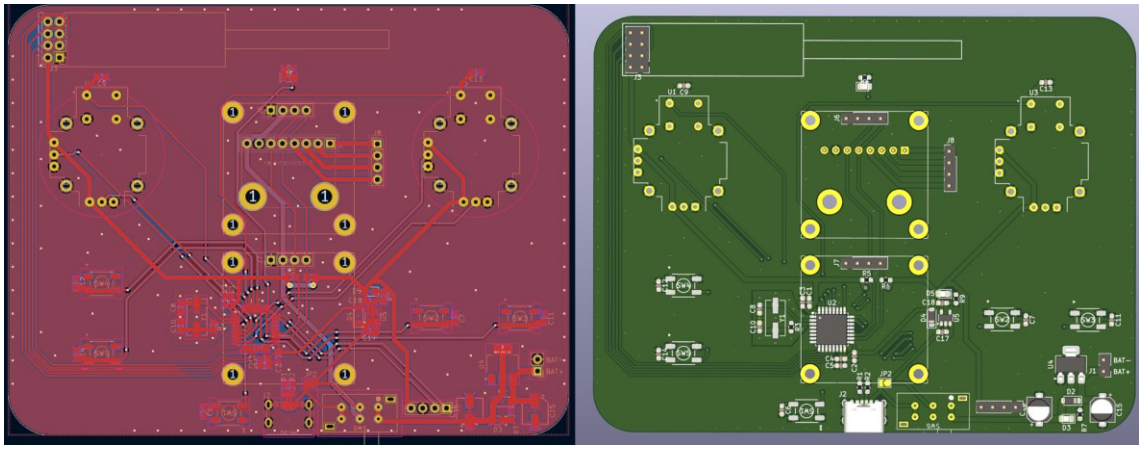


Figure 46: Board layout (left) and 3D render of the PCB (right)

The controller's power supply circuit is shown in Figure 47, incorporating a USB-C connector with 5.1 kΩ pull-down resistors on CC1 and CC2 lines for proper negotiation of 5 V power. An additional 1x2 header connector is included for an optional battery source.

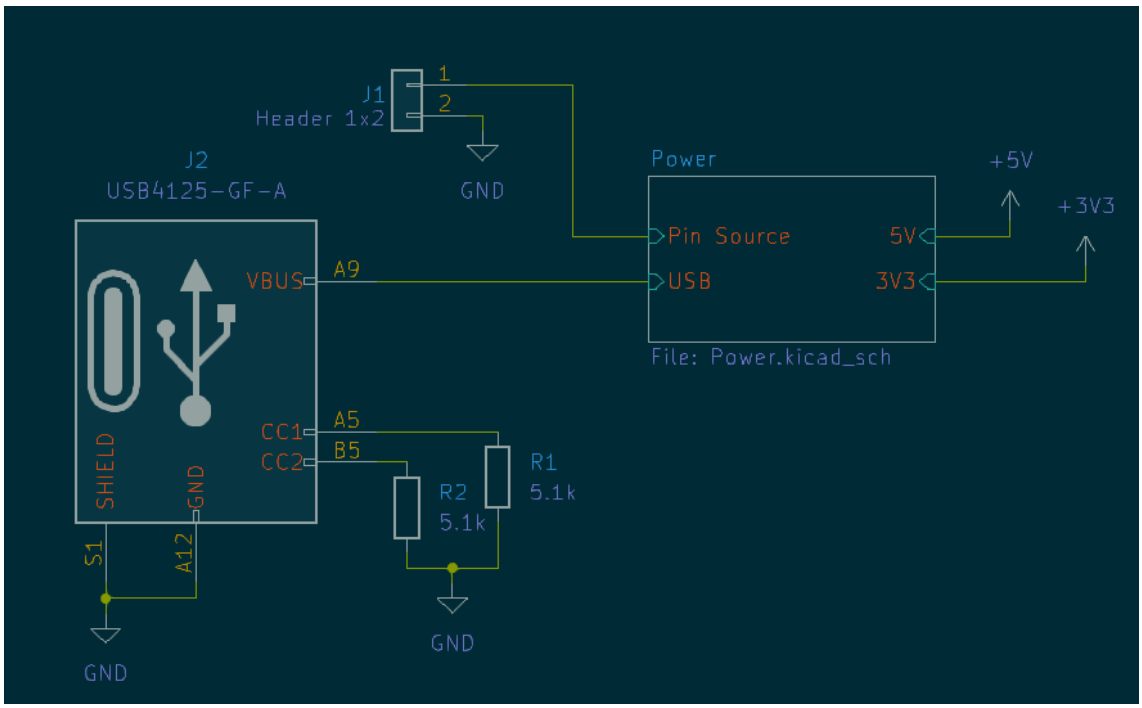



Figure 47: Schematic of controller's power source circuit

Two voltage regulators manage the 5 V and 3.3 V rails, as depicted in Figure 48, are as follows:

- The **AZ1117-5.0** regulator provides a 5 V output, with input/output smoothing capacitors and a Schottky diode (1N5819HW-7-F) placed across its input and output terminals for reverse voltage protection. An indicator LED on the output signals the presence of 5 V power.
- The **TLV75533PDBV** regulator steps down the 5 V to 3.3 V, with capacitors on both sides and a Schottky diode across its terminals for reverse voltage protection. A second indicator LED on the output confirms the 3.3 V rail is active.

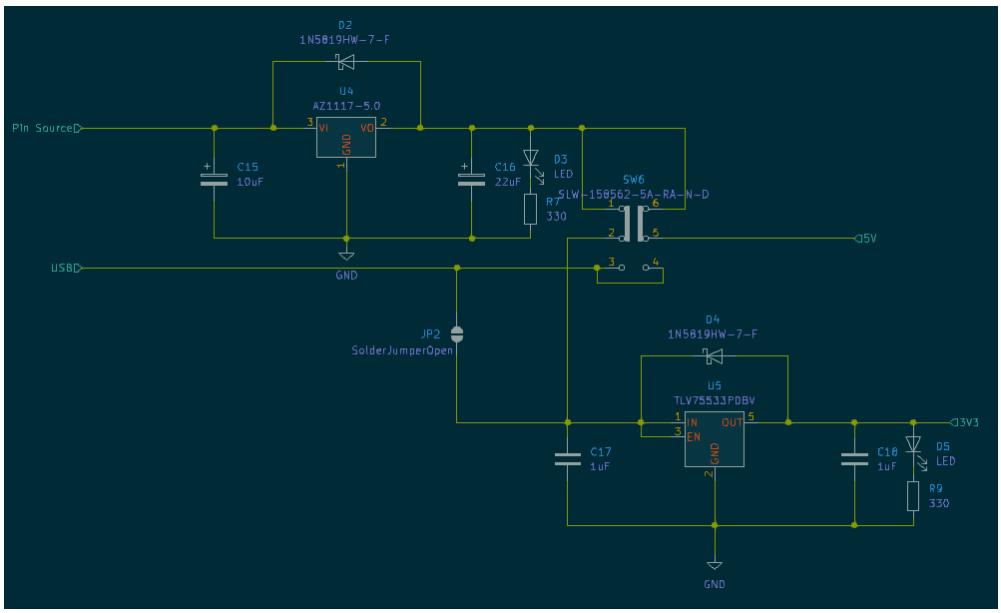



Figure 48: Schematic of voltage regulator circuits for the controller

A sliding switch between the regulators determines the source of power for the board. In one position, the USB power directly supplies 5 V rail. In the other, the 5 V rail is sourced from the AZ1117 regulator connected to a battery. This dual-mode setup ensures flexibility in power options for different use scenarios.

Figure 49, highlights the layout of these power components, including connectors, the switch, and the regulators. The 5 V and 3.3 V power distribution rails, shown in Figure 50, are 1 mm-wide traces on a 1 oz copper layer.

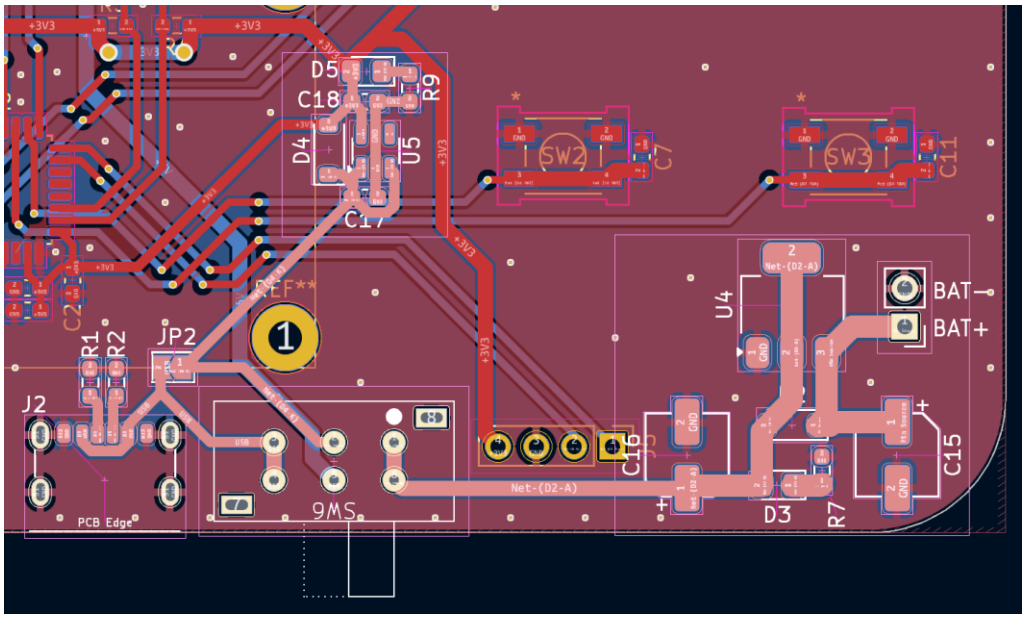



Figure 49: Board layout with power supply circuit highlighted

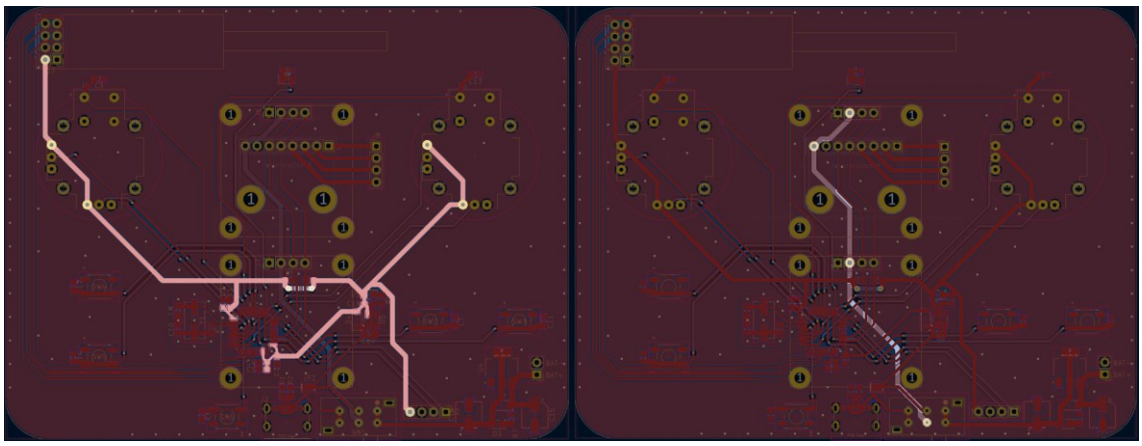


Figure 50: Board layout with 3.3 V rail (left) and 5 V rail (right) highlighted

Resistance at the longest path from the source through the rail is calculated using this formula:

$$R = \rho \cdot LT \cdot W \cdot [1 + \alpha \cdot (temp - 25)] \quad (6)$$

Where:

P = resistivity

α = temperature coefficient

L = length = 90 mm

W = trace width = 1 mm

T = trace height = 0.0348 mm

$\rho_{copper} = 1.7 \cdot 10^{-6}$ Ω-cm

$\alpha_{copper} = 3.9 \cdot 10^{-3}$ / °C

The resulting resistance (6) is 0.043 Ω is a relatively low value and suggests efficient current flow with minimal power loss.

The STM32G431KBT6 microcontroller, its supporting peripherals, and layout are illustrated in Figure 51. The decoupling capacitors are positioned close to the power pins to stabilize voltage and minimize noise. A reset button is included for system reinitialization during development.

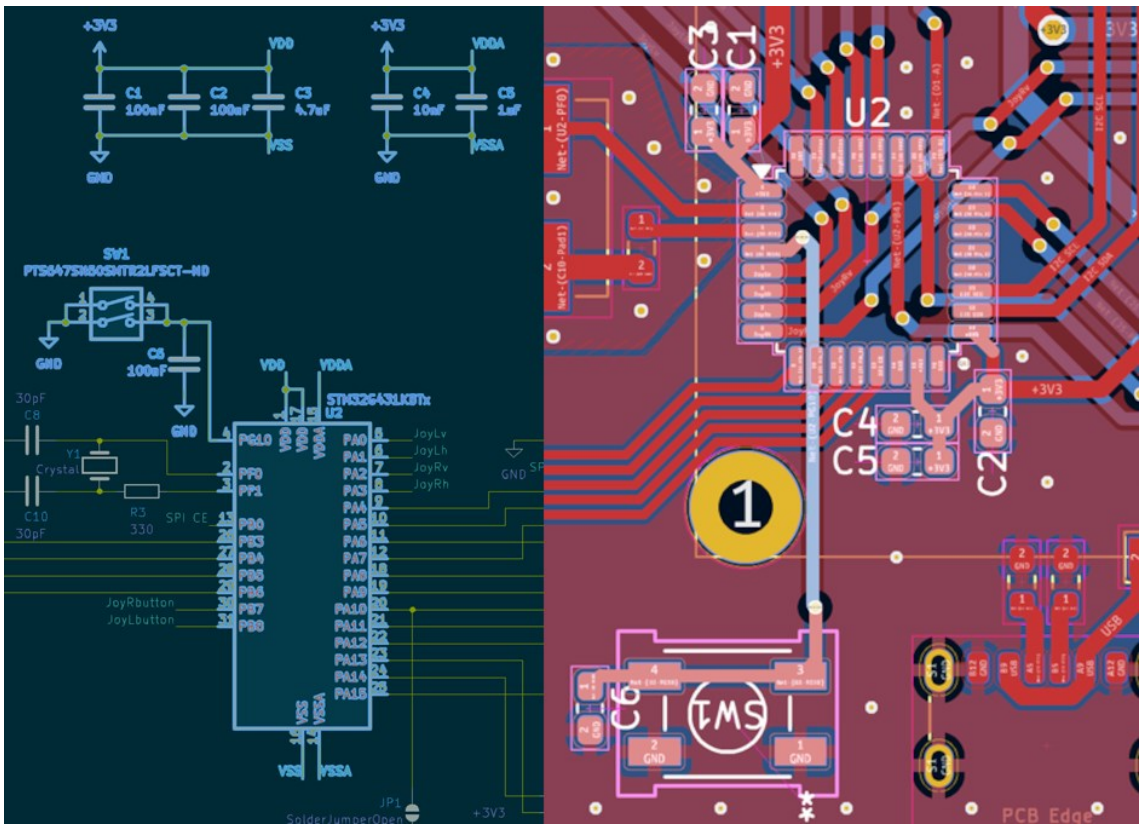



Figure 51: Schematic (left) and layout (right) with highlighted microcontroller circuit and its peripherals

The controller's oscillator circuit, detailed in Figure 52, replicates the design from the flight controller. It uses a 16 MHz crystal with load capacitors and a series resistor, allowing stable clock generation for the microcontroller. The PCB layout isolates the oscillator circuit to reduce EMI effects. [34.]

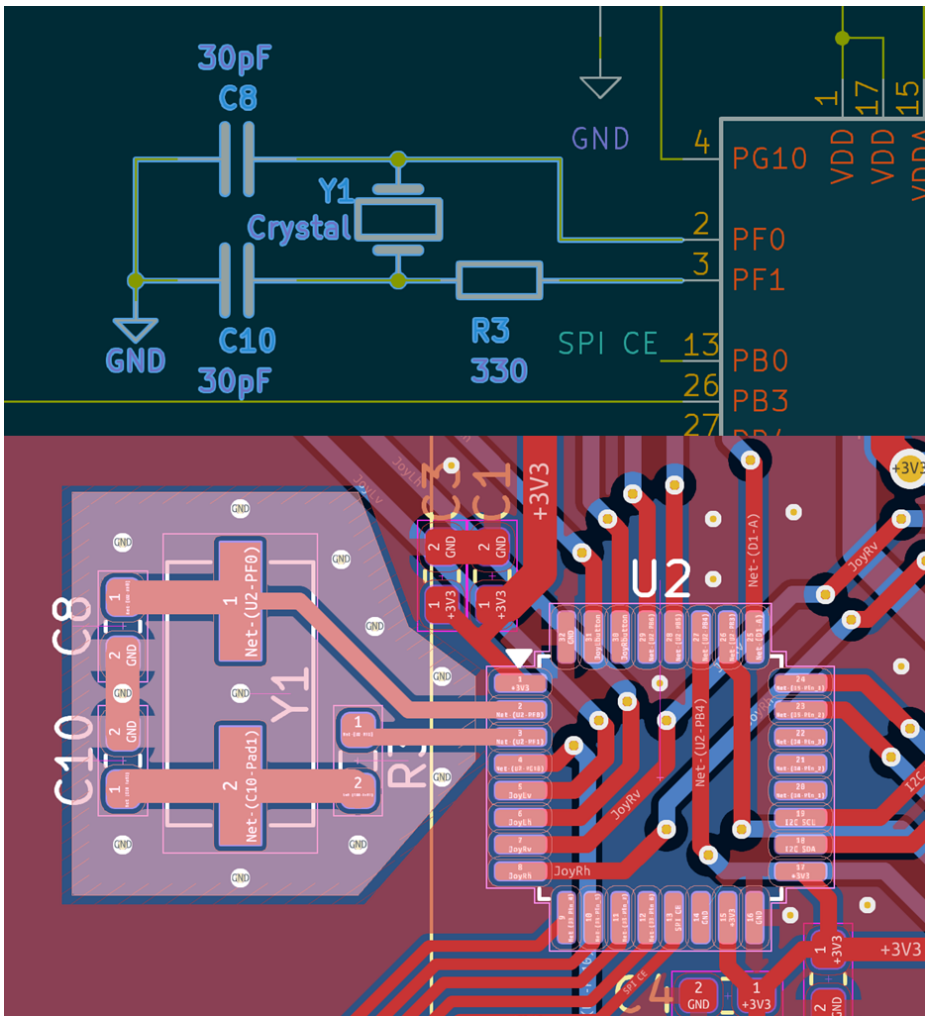



Figure 52: Oscillator schematic (top) and layout (bottom)

The analog joysticks and buttons are shown in Figure 53. Each button is equipped with a 100 nF capacitor to debounce the input signal, preventing false triggers caused by mechanical bouncing. The joystick axes are routed to the ADC pins of the microcontroller for precise position measurement.

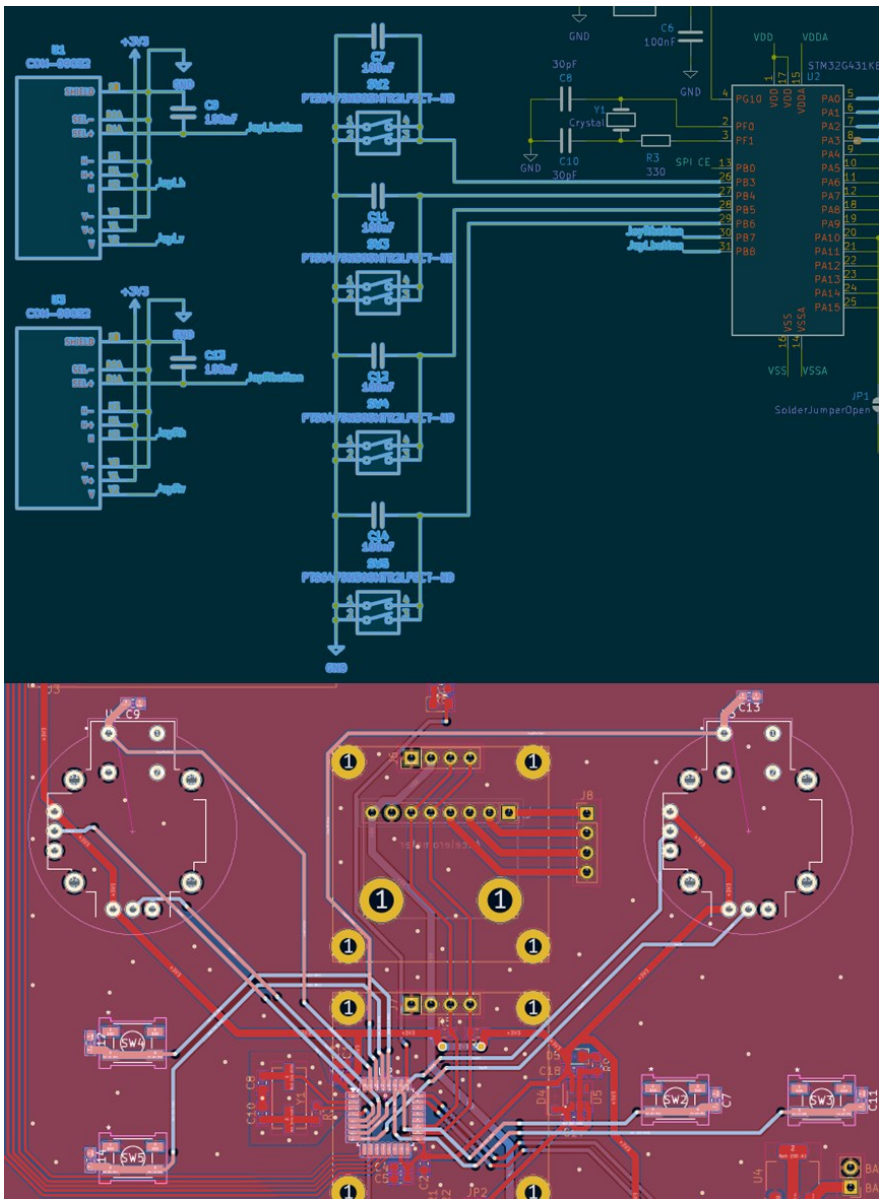


Figure 53: Schematic (top) and layout (bottom) with highlighted circuits of controller input devices

The I2C interface schematic and layout, shown in Figure 54, feature 4.7 kΩ pull-up resistors connected to the 3.3 V rail to ensure proper signal responsiveness. A hierarchical sheet within the schematic, detailed in Figure 55, includes headers for connecting the OLED displays and the MPU6050 module. These modules share the same I2C lines, simplifying routing and minimizing additional traces as shown in Figure 56.

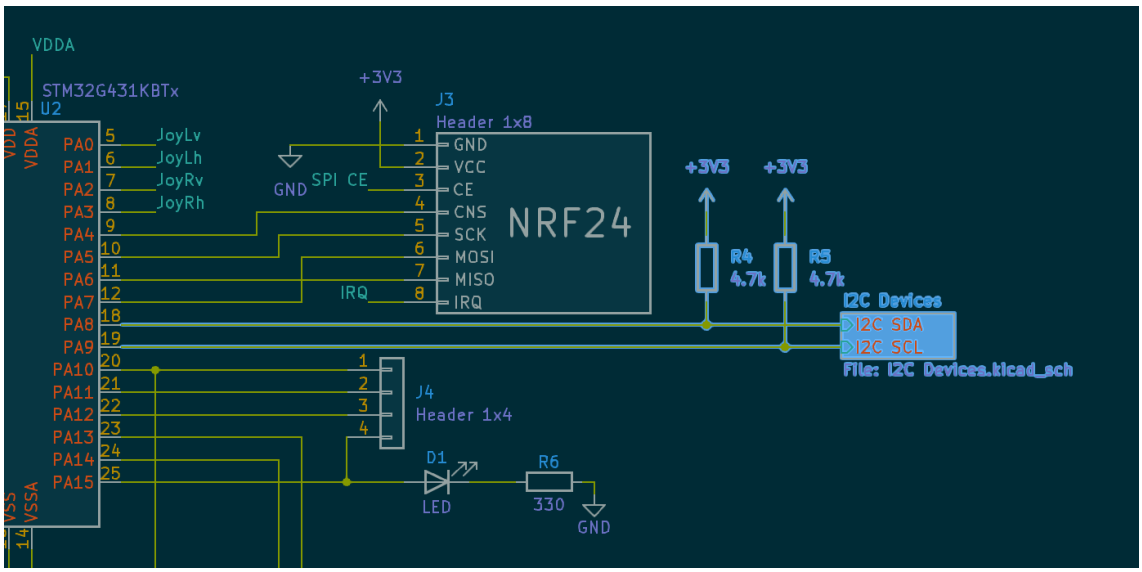



Figure 54: I2C circuit highlighted in the schematic

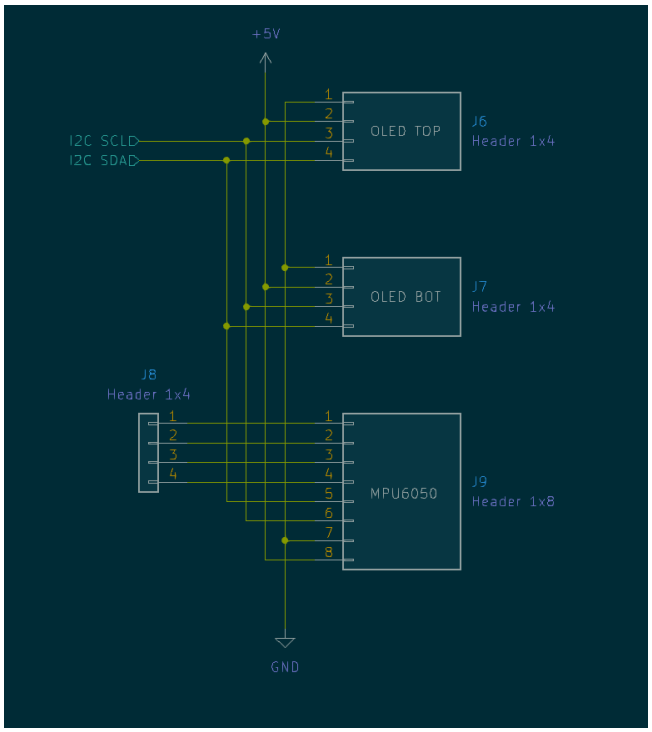



Figure 55: Schematic of the I2C distributed between connectors for OLED displays and the accelerometer

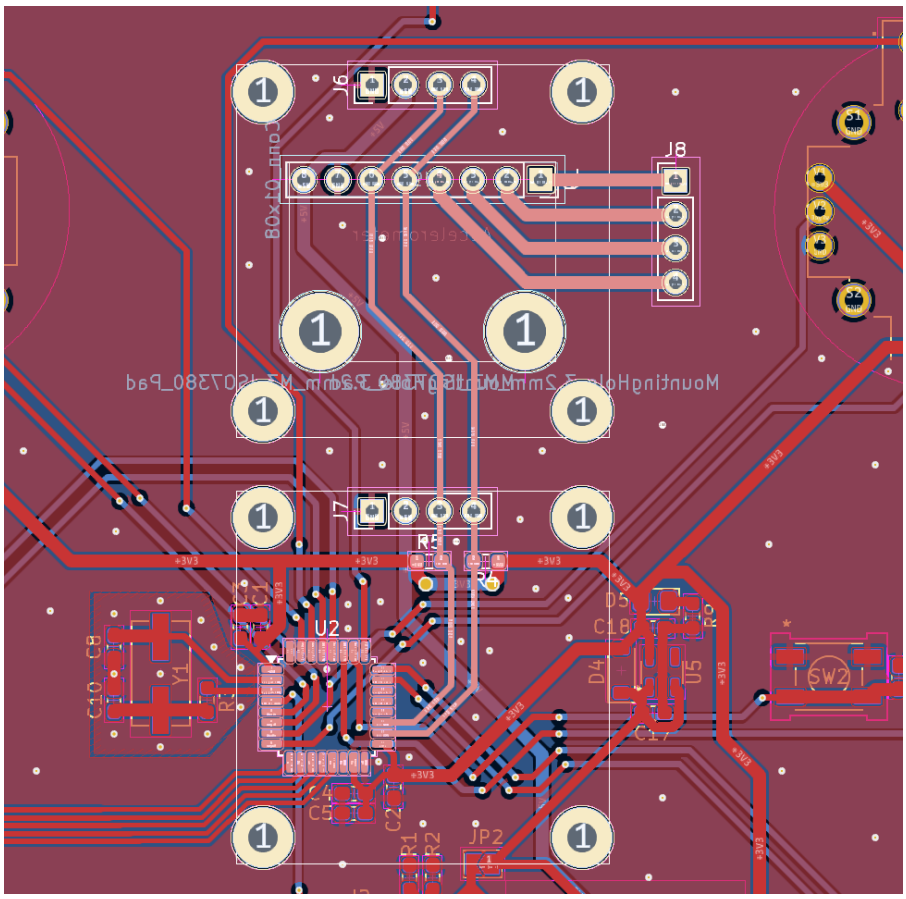



Figure 56: Board layout with highlighted I2C-dependent devices

The NRF24L01P wireless module connects to the microcontroller through a 2x4 header, as depicted in Figure 57. The layout positions the module away from potential interference sources, ensuring stable wireless communication.

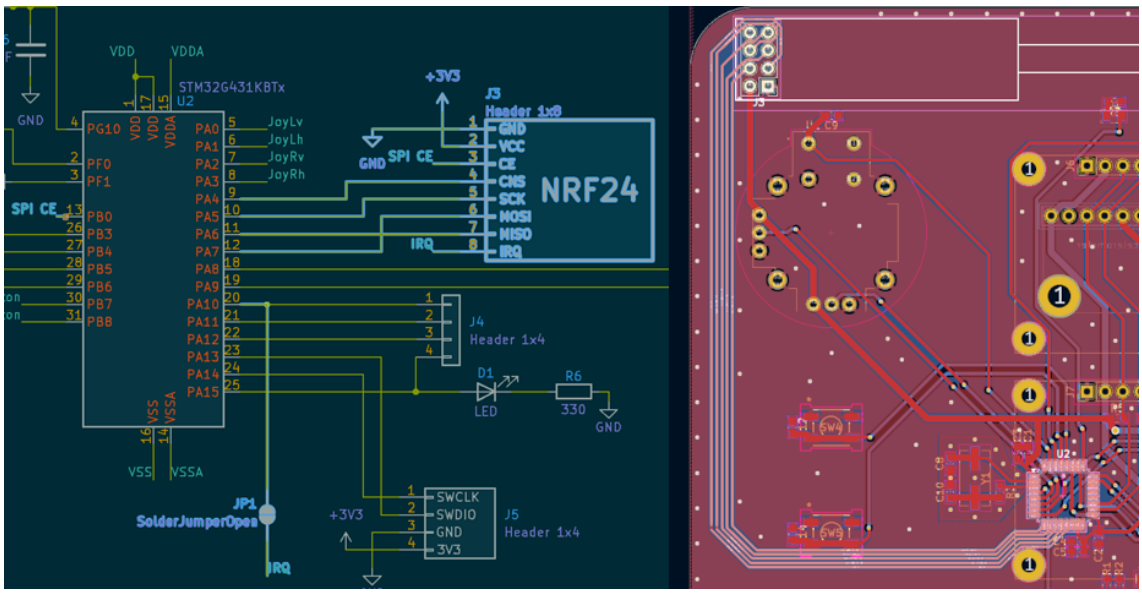



Figure 57: Schematic (left) and layout (right) with highlighted NRF24 routing

The indicator LED, located above the OLED displays, provides visual feedback about the controller's operational status. Figure 58, shows the LED's schematic and its position on the PCB layout.

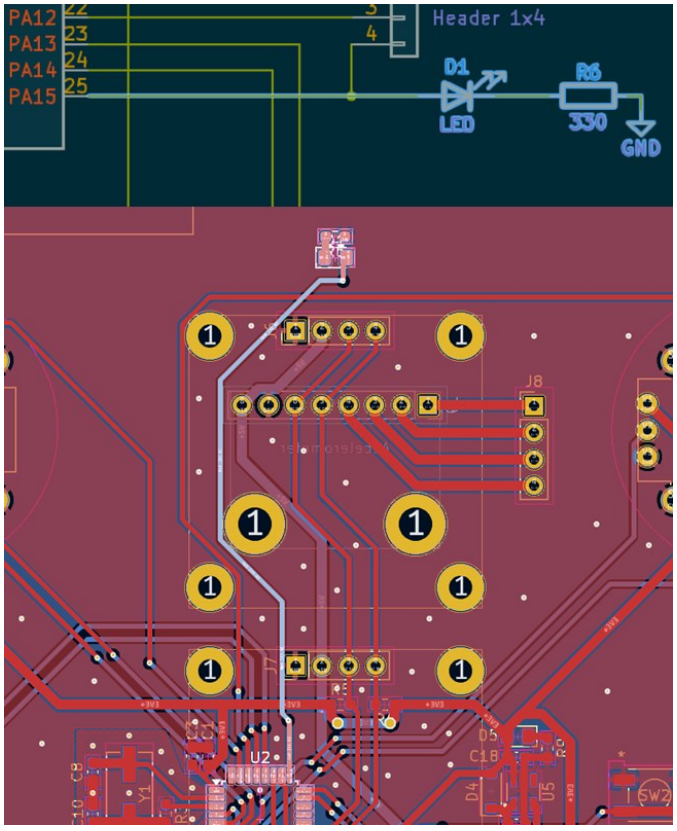



Figure 58: Schematic (top) and layout (bottom) of the indicator LED

The SWD (Serial Wire Debug) interface, shown in Figure 59, provides access to the microcontroller's programming and debugging capabilities. Due to configuration constraints, the connector is placed further from the microcontroller.

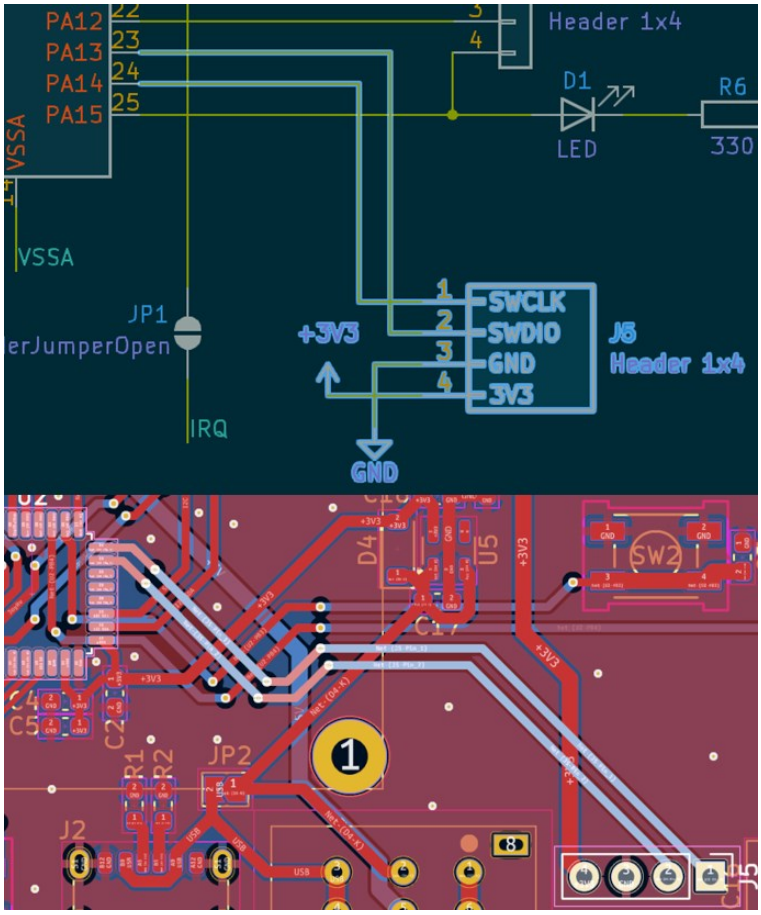



Figure 59: Schematic (top) and layout (bottom) with highlighted SWD routing

The controller's implementation detailed the integration of its components, circuits, and PCB layout to fulfill the functional requirements established in previous chapters. Each subsystem is designed to provide the necessary capabilities for effective communication, user interaction, and control of the drone. This implementation reflects a systematic approach to achieving the controller's core functions, providing a basic platform for further testing and development in the project.

# 10 Software Development of System's Libraries

Creating software for the drone-controller system operation involves implementing various communication interfaces, configuring microcontroller peripherals, and developing module-specific functions. The software development process was conducted using STM32CubeIDE, which provides a suite of tools tailored for embedded development, including peripheral configuration, code writing, and debugging. The different views in the IDE streamline tasks, making it easier to transition between configuration and implementation.

## 10.1 Drone

The software implementation for the drone encompasses the configuration of its microcontroller and the development of individual module libraries, each tailored to fulfill specific system requirements. These modules include wireless communication, orientation sensing, altitude measurement, motor control, and power monitoring. The process ensures that all components can operate in coordination while maintaining modularity to facilitate testing and future modifications.
The following subsections detail the implementation of individual modules, describing their functionality, configurations, and key software elements.

### 10.1.1 Microcontroller Configuration

The clock setup for the system uses the High-Speed External (HSE) oscillator running at 16 MHz as the primary source, as illustrated in Figure 60. The Clock Security System (CSS) monitors the clock source for failures and automatically switches to a backup clock if an issue is detected. This ensures that critical operations continue uninterrupted, making it particularly useful in a drone system where stability is essential.

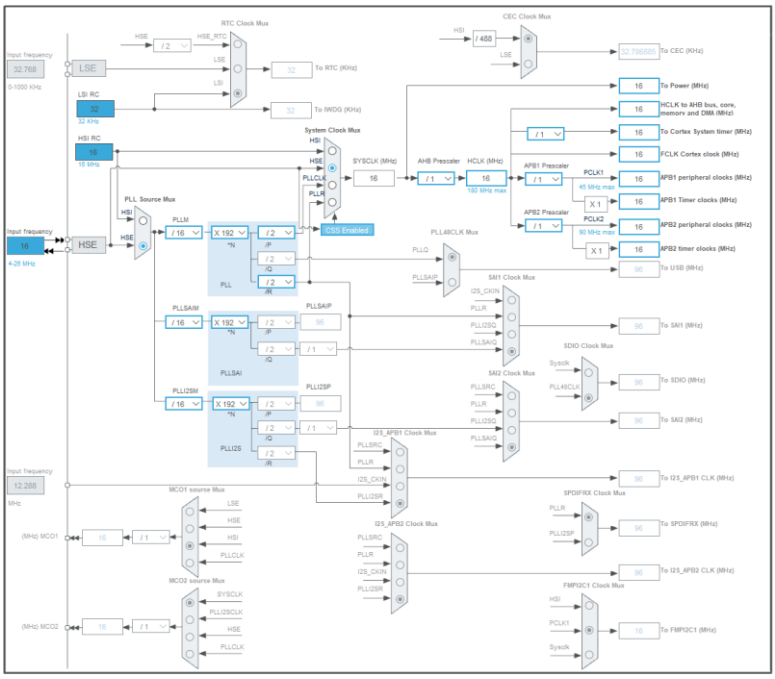


Figure 60: Clock configuration view in STM32CubeIDE

The clock signals are distributed across buses using prescalers, all configured to maintain a 16 MHz frequency. This simplifies timing calculations for peripherals like timers and communication interfaces, as all subsystems operate on a uniform clock speed.

To visualize the pin assignments for the microcontroller (Figure 61), CubeIDE highlights the utilized pins in green. Some pins are pre-assigned for potential future use, even if they remain unutilized in the current prototype. This ensures compatibility with future expansions and avoids conflicts with peripheral access. Unused pins, shown in gray, remain available for other applications.

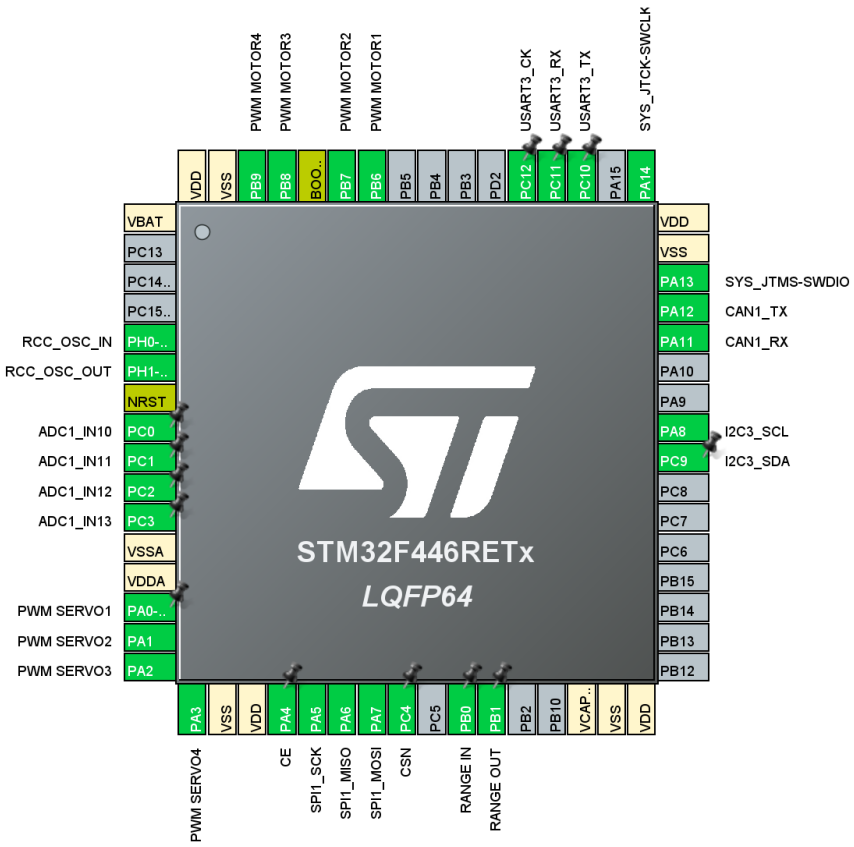



Figure 61: Pin configuration view in STM32CubeIDE

Interrupts are essential for responding to time-critical events in the system. Figure 62, demonstrates the NVIC (Nested Vectored Interrupt Controller) configuration menu, showing the setup for EXTI Line 0, which is prioritized at level 1. This line is connected to the Echo pin of the RCWL1670 module. The logic involves sending a pulse through the Trig pin, with the Echo pin triggering the interrupt upon receiving the reflected signal. The timing captured by the interrupt service routine is used for distance calculation.

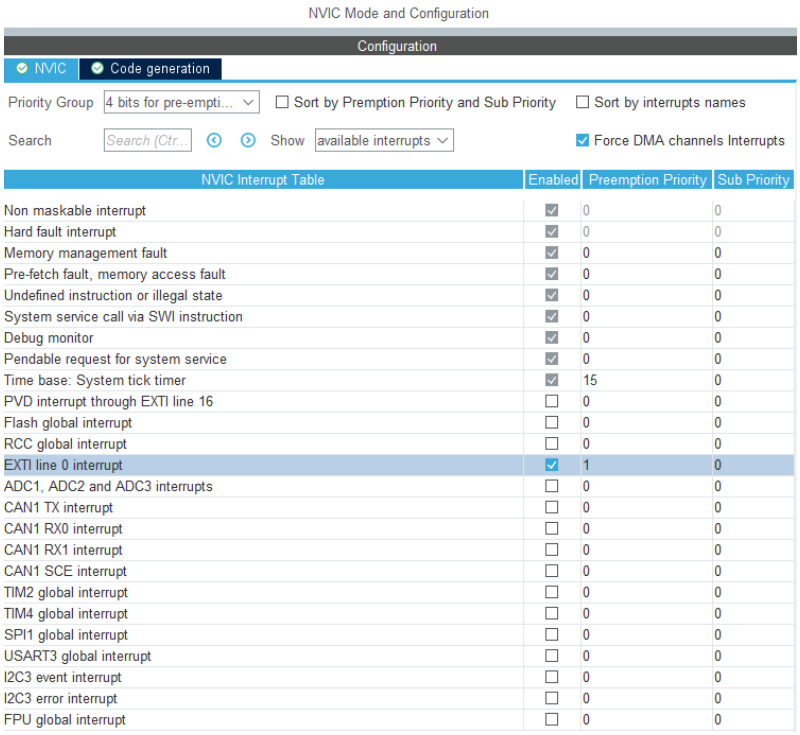



Figure 62: NVIC Mode and Configuration menu in STM32CubeIDE

I2C (Inter-Integrated Circuit) is a two-wire communication protocol used for interfacing with the IMU (BNO055) and barometric pressure sensor (BMP180). It operates in Fast Mode, configured at 400 kHz, as per standard utilized by the used modules. The primary address length is set to 7 bits. Figure 63, shows the I2C3 configuration menu.

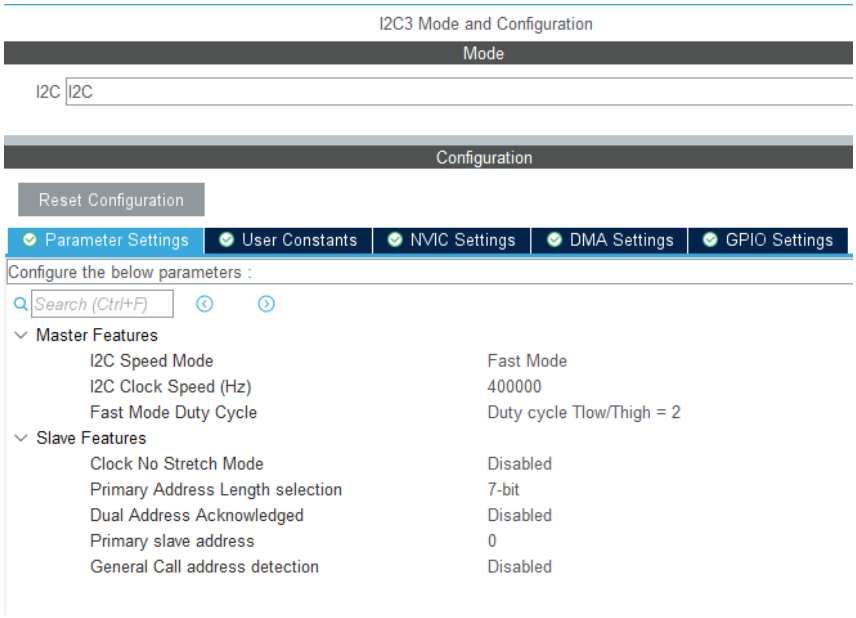



Figure 63: I2C Mode and Configuration menu in STM32CubeIDE

SPI (Serial Peripheral Interface) is a full-duplex communication protocol used to interface with the NRF24 module. Figure 64, shows the SPI1 configuration menu and the settings include:

- **Mode**: Master (the microcontroller initiates communication events).
- **Frame Format**: Motorola (a common configuration for SPI).
- **Data Size**: 8 bits per frame.
- **MSB First**: Data is transmitted with the most significant bit first.
- **Prescaler**: 2 (divides the clock speed to suit the NRF24 module's requirements).

This configuration ensures correct data exchange between the microcontroller and the NRF24 module.

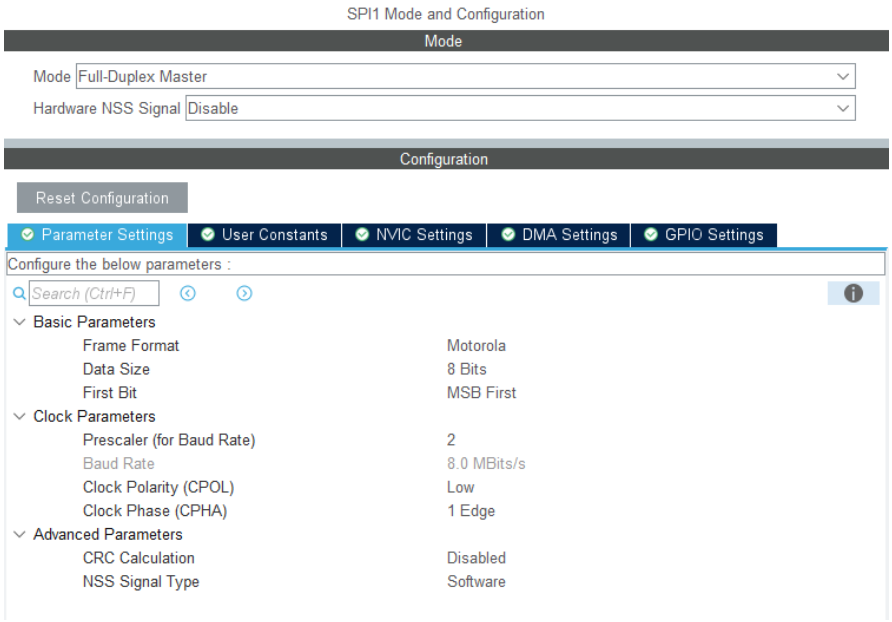



Figure 64: SPI Mode and Configuration menu in STM32CubeIDE

The ADC1 peripheral is set up to read analog signals from the battery voltage dividers and the current sensor. As detailed in Figure 65, the ADC operates with the following settings:

- Four active channels (IN10, IN11, IN12, IN13) mapped to battery and current measurements.
- A clock prescaler of PCLK divided by 2.
- 12-bit resolution, enabling values from 0 to 4095.

- Continuous conversion and scan modes for seamless multi-channel acquisition.
- Ranks are assigned to each channel for sequential scanning with a sampling time of 3 cycles per channel:
    - Rank 1: Channel IN10 for first battery in series.
    - Rank 2: Channel IN11 for first and second in series.
    - Rank 3: Channel IN12 for the whole power pack.

Rank 4: Channel IN13 for the current sensor measurement.

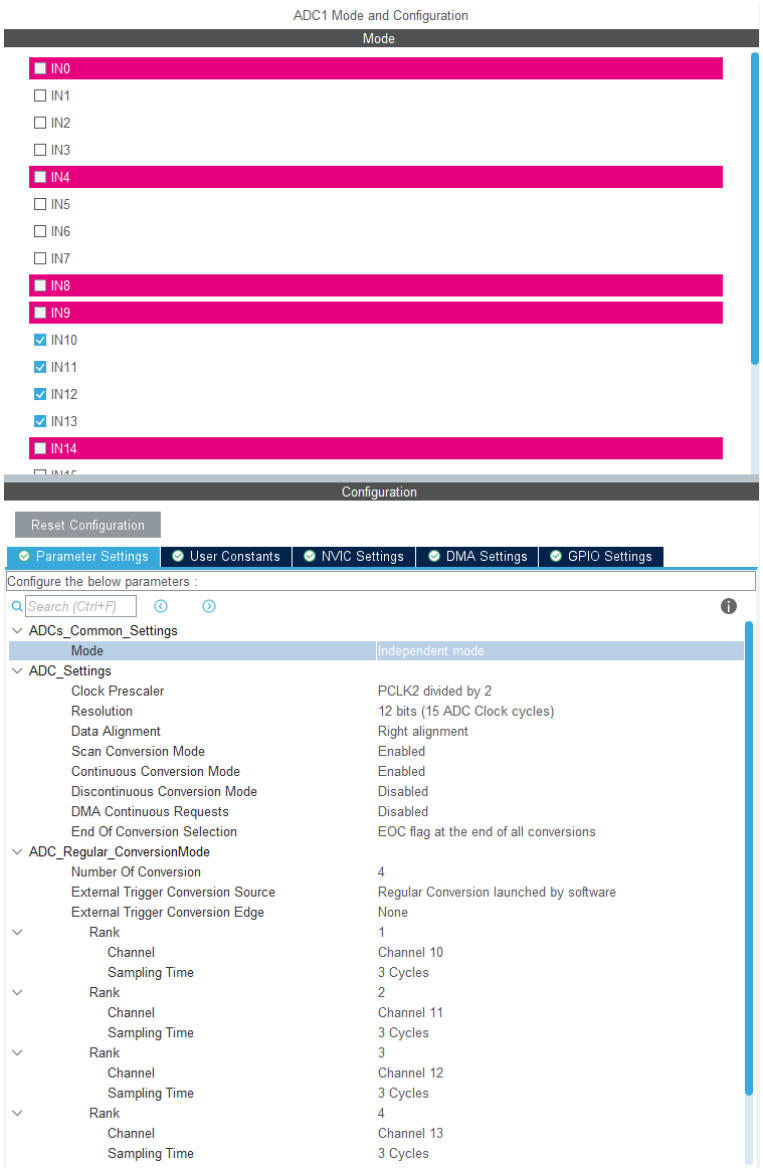



Figure 65: ADC Mode and Configuration menu in STM32CubeIDE

### 10.1.2 Wireless Communication (NRF24L01P)

The NRF24L01P module enables bidirectional communication between the drone and the controller. It utilizes five microcontroller pins, including three SPI lines (SCK, MOSI, MISO) and two GPIO lines (CSN and CE). The CSN pin activates communication, while the CE pin controls the module's radio operations [27]. The control state diagram governing the logic of the NRF24 module's operation is presented in Figure 66.

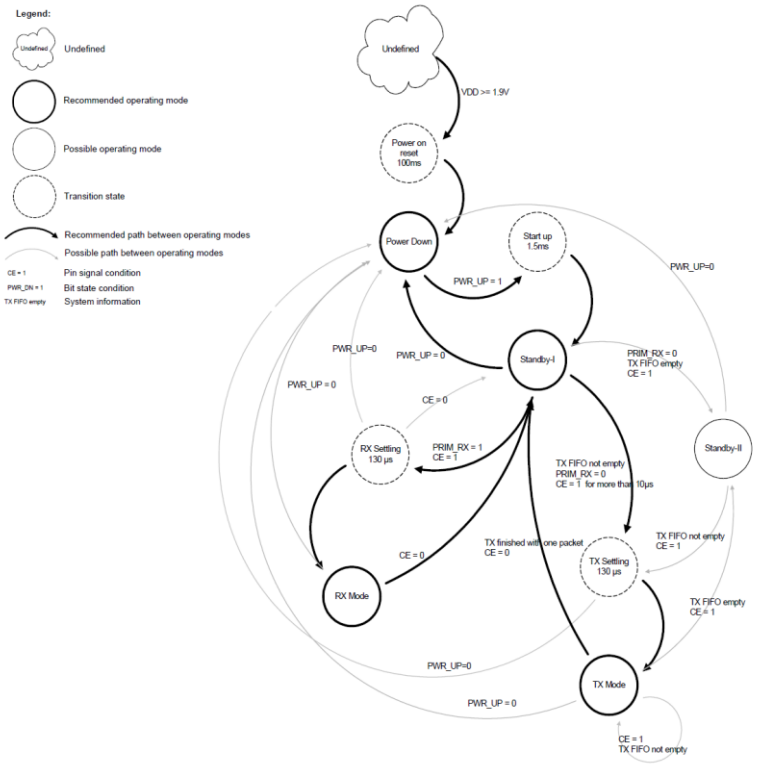



Figure 66: Control state diagram of the NRF24 module [27]

The software development focused on core functions:

- **Initialization**
- **Address Assignment**
- **Mode Selection**
- **Data Transmission (including packing data)**
- **Data Reception (including unpacking data)**

Initialization functions prepare modules for operation by setting them to a stable, default state. For the NRF24L01P, this involves clearing registers, configuring default parameters, and setting the module's initial operating mode. These steps ensure the module is ready for reliable communication within the drone-controller system. Listing 1, shows the implementation of the initialization function, outlining the required setup for beginning of operation.

```
void NRF24_Init(void) {
    HAL_Delay(5); // Ensure NRF24 is ready after power-up

    // Configure basic settings: auto-retransmit, RF power/data rate
    NRF24_WriteReg(SETUP_RETR, 0x5F); // Auto-retransmit delay 1500us, 15
retries
    NRF24_WriteReg(RF_SETUP, 0x03);  // 0dBm power, 1Mbps data rate

    // Activate FEATURE register
    NRF24_ReadReg(FEATURE); // Optional status check
    NRF24_CSN_L();
    NRF24_SPI_Send(ACTIVATE);
    NRF24_SPI_Send(0x73); // Activation magic number
    NRF24_CSN_H();
    NRF24_ReadReg(FEATURE); // Verify activation

    // Disable dynamic payload, enable auto-ack on all pipes, set RX pipes
    NRF24_WriteReg(DYNPD, 0x00);
    NRF24_WriteReg(EN_AA, 0x3F);
    NRF24_WriteReg(EN_RXADDR, 0x03);

    // Set payload sizes (32 bytes for pipes 0 and 1, others default)
    NRF24_WriteReg(RX_PW_P0, 0x20);
    NRF24_WriteReg(RX_PW_P1, 0x20);
    NRF24_WriteReg(RX_PW_P2, 0x20);
    NRF24_WriteReg(RX_PW_P3, 0x20);
    NRF24_WriteReg(RX_PW_P4, 0x20);
    NRF24_WriteReg(RX_PW_P5, 0x20);

    // Set address width, RF channel, and clear interrupt flags
    NRF24_WriteReg(SETUP_AW, 0x03); // 5-byte address width
    NRF24_WriteReg(RF_CH, 0x4C);    // RF channel 76 (2.476 GHz)
    NRF24_WriteReg(STATUS, 0x70);   // Clear IRQ flags

    // Flush RX and TX FIFOs
    NRF24_CMD(FLUSH_RX);
    NRF24_CMD(FLUSH_TX);

    // Configure and verify CONFIG register (initially RX mode, then TX
mode)
    NRF24_WriteReg(CONFIG, 0x0C);
    if (NRF24_ReadReg(CONFIG) != 0x0C) Error_Handler();
    NRF24_WriteReg(CONFIG, 0x0E);
    HAL_Delay(5);
    if (NRF24_ReadReg(CONFIG) != 0x0E) Error_Handler();

    // Reapply RF_SETUP to confirm settings
    NRF24_ReadReg(RF_SETUP);
    NRF24_WriteReg(RF_SETUP, 0x03);

    // Adjust payload sizes for outgoing and incoming data
    NRF24_WriteReg(RX_PW_P0, 0x10); // Pipe 0 (outgoing)
    NRF24_WriteReg(RX_PW_P1, 0x10); // Pipe 1 (incoming)
    NRF24_WriteReg(RX_PW_P2, 0x02); // Pipe 2
    NRF24_WriteReg(RX_PW_P3, 0x02); // Pipe 3
    NRF24_WriteReg(RX_PW_P4, 0x02); // Pipe 4
    NRF24_WriteReg(RX_PW_P5, 0x02); // Pipe 5
}
```

Listing 1: Initialization function of the NRF24

Listing 2, shows the address assignment functions for the NRF24L01P module. These functions configure the module's registers to define the addresses used for communication. For transmitting data, both RX_ADDR_P0 and TX_ADDR must be set to the same address to enable auto-acknowledgment. For receiving data, RX_ADDR_P0 is assigned an address to identify incoming packets.

The module supports receiving data from up to six unique addresses simultaneously, though this feature is not utilized in the design. Currently, the size of the payload is manually set during initialization. Future improvements will include dynamic payload sizing to enhance flexibility and responsiveness in communication sequence.

```
// Address assignment for data reception
void NRF24_RX_Address(void) {
    // Set RX address for Pipe 1
    NRF24_CSN_L();
    NRF24_SPI_Send(W_REGISTER | RX_ADDR_P1); // Write command for RX_ADDR_P1
    NRF24_SPI_Send(0x31); // Address byte 1
    NRF24_SPI_Send(0x4E); // Address byte 2
    NRF24_SPI_Send(0x6F); // Address byte 3
    NRF24_SPI_Send(0x64); // Address byte 4
    NRF24_SPI_Send(0x65); // Address byte 5
    NRF24_CSN_H();

    // Enable Pipe 1 for receiving data
    NRF24_WriteReg(EN_RXADDR, 0x02); // Enable RX_ADDR_P1
}

// Address assignment for data transmission
void NRF24_TX_Address(void) {
    // Set RX address for Pipe 0 (used for auto-acknowledgment)
    NRF24_CSN_L();
    NRF24_SPI_Send(W_REGISTER | RX_ADDR_P0); // Write command for RX_ADDR_P0
    NRF24_SPI_Send(0x31); // Address byte 1
    NRF24_SPI_Send(0x4E); // Address byte 2
    NRF24_SPI_Send(0x6F); // Address byte 3
    NRF24_SPI_Send(0x64); // Address byte 4
    NRF24_SPI_Send(0x65); // Address byte 5
    NRF24_CSN_H();

    // Set TX address (for sending data)
    NRF24_CSN_L();
    NRF24_SPI_Send(W_REGISTER | TX_ADDR); // Write command for TX_ADDR
    NRF24_SPI_Send(0x31); // Address byte 1
    NRF24_SPI_Send(0x4E); // Address byte 2
    NRF24_SPI_Send(0x6F); // Address byte 3
    NRF24_SPI_Send(0x64); // Address byte 4
    NRF24_SPI_Send(0x65); // Address byte 5
    NRF24_CSN_H();
}
```

Listing 2: Address assignment function of the NRF24

The mode selection function for the NRF24L01P module enables it to switch between transmitting and receiving states by configuring control registers. In transmit mode, the module prepares to send data, while in receive mode, it listens for incoming packets. Listing 3, illustrates how this functionality is implemented, allowing capability for bidirectional communication within the drone-controller system.

```
void NRF24_RX_Mode(void) {
    // Configure the NRF24 for RX (receive) mode
    NRF24_CE_L(); // Disable the CE pin to allow configuration
    NRF24_WriteReg(CONFIG, 0x0F); // Enable RX mode with CRC
    NRF24_WriteReg(STATUS, 0x70); // Clear any pending interrupts
    NRF24_WriteReg(EN_RXADDR, 0x02); // Enable Pipe 1 for receiving data
    NRF24_CE_H(); // Enable the CE pin to start listening
}

void NRF24_TX_Mode(void) {
    // Configure the NRF24 for TX (transmit) mode
    NRF24_CE_L(); // Disable the CE pin to allow configuration
    NRF24_WriteReg(CONFIG, 0x0E); // Enable TX mode with CRC
    NRF24_ReadReg(EN_RXADDR); // Optional: Read enabled RX addresses
(possibly for debugging)
    NRF24_WriteReg(EN_RXADDR, 0x03); // Enable Pipes 0 and 1 for TX and
auto-ack
}
```

Listing 3: Mode selection function of the NRF24

The data exchanged between the NRF modules of the drone-controller system is structured into a packed format containing multiple variables. This structure, presented in Listing 4, is used by the controller to send joystick readings and other settings to the drone. It includes:

- **Checkbyte**: A predefined value used to verify the integrity and correctness of the data block during transmission.
- **ADC Readings**: Positional data derived from the joystick inputs on the controller.
- **Configuration Byte**: A binary representation of settings adjusted through the controller's menu system.
- **Checksum**: A calculated value used to detect errors in the transmitted data block (detailed in a separate function mentioned later).

```
typedef struct {
    uint16_t checkbyte;
    uint16_t Joy1X;
    uint16_t Joy1Y;
    uint16_t Joy2X;
    uint16_t Joy2Y;
    uint16_t DroneConfig;
    uint16_t checksum;
} RemotePackage;
```

Listing 4: Example structure of controller data

The **checkbyte** is assigned a value of 43690 to create a distinct binary pattern of 1010101010101010, and along with the **checksum** ensures that transmitted data maintains consistency and can be validated during the communication process.
The structure transmitted from the drone to the controller will include data like orientation, altitude, battery status and current drawn.

The function responsible for packing the prepared structure into an array converts the structured data into a format suitable for transmission via the NRF24L01P module. This process organizes the whole structure into a sequential array, ensuring compatibility with the module's communication requirements. The packed array is then provided to the NRF module for transmission. Listing 5, demonstrates the implementation of this function, highlighting the steps taken to format the data correctly.

```
void PackIntoArray(RemotePackage *TheStructure, uint8_t array[]) {
     TheStructure->checksum = CalculateChecksum(*TheStructure);  //
Calculate and set the checksum

    uint8_t* data = (uint8_t*)TheStructure;
    for (uint32_t i = 0; i < sizeof(RemotePackage); i++) {
        array[i] = data[i];  // Pack the structure into the array
    }
}
```

Listing 5: Packing function turning structure into an array

The function responsible for sending data to the NRF24L01P module takes the prepared array and transmits it by interfacing with the module’s SPI communication. This function also enables the module's radio operations, initiating the transmission process. The module monitors for a confirmation

signal indicating successful payload delivery. This step ensures reliable communication between the controller and the drone. The implementation of this function is shown in Listing 6, where the sequence of actions is detailed.

```
void NRF24_Transmit(uint8_t array[], int array_size) {
    // Load the data to be transmitted into the TX payload
    NRF24_CSN_L();
    NRF24_SPI_Send(W_TX_PAYLOAD);
    for (int i = 0; i < array_size; i++) {
        NRF24_SPI_Send(array[i]); // Send each byte of the payload
    }
    NRF24_CSN_H();

    // Begin the transmission by setting the CE pin high
    NRF24_CE_H();

    // Wait for the transmission to complete or timeout
    uint8_t Check;
    uint32_t Timer = HAL_GetTick();
    do {
        NRF24_CSN_L();
        Check = NRF24_SPI_Send(NOP); // Check the status register
        NRF24_CSN_H();
    } while (Check == 0x0E && (HAL_GetTick() - Timer) < 100); // Timeout
after 100 ms
    NRF24_CE_L();

    // If the transmission failed, trigger the error handler
    if (Check == 0x0E) {
        Error_Handler();
    }

    // Clear the interrupt flags in the STATUS register
    NRF24_WriteReg(STATUS, 0x70);
}
```

Listing 6: Transmit function of the NRF24

The function handling data reception from the NRF24L01P module begins by checking the status of the module's FIFO to determine if new data is available. If data is present, it is read from the module and passed to the unpacking function for processing. This ensures that the data received is immediately validated and interpreted for further use. The implementation of this function is detailed in Listing 7, outlining the steps for efficient and reliable data reception.

```
void NRF24_Receive(RemotePackage *s, uint8_t size) {
    uint8_t data[size];  // Temporary array to hold received data
    uint8_t Check;

    NRF24_CE_H(); // Set CE high to enable receiving mode
    Check = NRF24_ReadReg(FIFO_STATUS); // Check FIFO status to see if data
is available

    if (Check != 0x10) { // If FIFO is not empty
        NRF24_CSN_L();
        NRF24_SPI_Send(R_RX_PAYLOAD); // Command to read RX payload

        // Read the received data byte by byte
        for (uint8_t i = 0; i < size; i++) {
            data[i] = NRF24_SPI_Send(NOP); // Send NOP and read data
        }

        NRF24_CSN_H();

        // Unpack the received data into the provided structure
        UnpackFromArray(data, s);
    }
}
```

Listing 7: Receive function of the NRF24

The unpacking function converts the received array back into its original structured format, restoring the individual variables for use in the system. During this process, the **checkbyte** and **checksum** are analyzed to verify the integrity and correctness of the data. This ensures that any errors during transmission are detected and handled appropriately. The implementation of this function is shown in Listing 8, where each step of the unpacking and validation process is detailed.

```
void UnpackFromArray(uint8_t array[], RemotePackage *s) {
    // Temporarily store the data to be checked
    RemotePackage temp;
    uint8_t* data = (uint8_t*)&temp;

    for (uint32_t i = 0; i < sizeof(temp); i++) {
        data[i] = array[i];  // Unpack the array into the temporary
structure
    }

    // Verify checksum before updating the actual structure
    if (VerifyChecksum(&temp) && temp.checkbyte == 43690) {
        *s = temp;  // Only update the original structure if the checksum is
correct
        correctPackets++;
    } else {
        // If checksum fails, you can log an error or handle it accordingly
     FailedPacket();  // Handle the error (e.g., discard the packet)
     correctPackets--;
    }
}
```

Listing 8: Unpacking from an array to structure and correction check function

### 10.1.3 IMU (BNO055)

The BNO055 is a system-in-package solution that simplifies the integration of orientation sensing into the embedded system. The BNO055 includes an onboard microcontroller running proprietary software to handle sensor fusion. This fusion combines data from the accelerometer, gyroscope, and magnetometer to provide ready-to-use orientation outputs. As a result, development did not include preparation of algorithms to process raw sensor data, significantly reducing the complexity of implementation.

The initialization function for the BNO055, shown in Listing 9, performs the standard setup required for module operation. This includes configuring communication settings, resetting the device, and verifying its readiness. At the end of the initialization sequence, the function sets the device to **NDOF mode** (Nine Degrees of Freedom), activating the onboard sensor fusion algorithm. This ensures the module is fully prepared to provide fused orientation data for the drone's future control algorithms.

```
void bno055_setup() {
    // Reset the BNO055 to ensure it starts in a known state
    bno055_reset();

    // Verify the chip ID to ensure the device is responding correctly
    uint8_t id = 0;
    bno055_readData(BNO055_CHIP_ID, &id, 1);
    if (id != BNO055_ID) {
        Error_Handler(); // Handle error if the device ID is incorrect
    }

    // Set the register page to 0 for configuration
    bno055_setPage(0);

    // Disable system-triggered operations to proceed with manual
configuration
    bno055_writeData(BNO055_SYS_TRIGGER, 0x0);

    // Enter configuration mode to allow modifying system settings
    bno055_setOperationModeConfig();
    bno055_delay(10); // Delay for settings to take effect

    // Set the device to NDOF (sensor fusion) mode for normal operation
    bno055_setOperationMode(BNO055_OPERATION_MODE_NDOF);
}
```

Listing 9: Initialization function of the BNO055

The BNO055 provides access to its processed sensor data through registers that store orientation outputs such as Euler angles or quaternions. These registers can be accessed using functions designed to retrieve the data, as shown in Listing 10. The retrieved data is intended for use in control algorithms, including PID controllers, which would adjust motor speed requests to maintain the drone's stability and orientation.

```
// Get accelerometer data as a vector (X, Y, Z components in m/s^2)
bno055_vector_t bno055_getVectorAccelerometer() {
    return bno055_getVector(BNO055_VECTOR_ACCELEROMETER);
}

// Get magnetometer data as a vector (X, Y, Z components in microteslas)
bno055_vector_t bno055_getVectorMagnetometer() {
    return bno055_getVector(BNO055_VECTOR_MAGNETOMETER);
}

// Get gyroscope data as a vector (X, Y, Z components in degrees/second)
bno055_vector_t bno055_getVectorGyroscope() {
    return bno055_getVector(BNO055_VECTOR_GYROSCOPE);
}

// Get orientation data as a vector of Euler angles (heading, roll, pitch in
degrees)
bno055_vector_t bno055_getVectorEuler() {
    return bno055_getVector(BNO055_VECTOR_EULER);
}

// Get linear acceleration data as a vector (X, Y, Z components in m/s^2,
gravity removed)
bno055_vector_t bno055_getVectorLinearAccel() {
    return bno055_getVector(BNO055_VECTOR_LINEARACCEL);
}

// Get gravity data as a vector (X, Y, Z components in m/s^2)
bno055_vector_t bno055_getVectorGravity() {
    return bno055_getVector(BNO055_VECTOR_GRAVITY);
}

// Get quaternion data as a vector (for representing orientation in 3D
space)
bno055_vector_t bno055_getVectorQuaternion() {
    return bno055_getVector(BNO055_VECTOR_QUATERNION);
}
```

Listing 10: Data retrieval function of the BNO055

### 10.1.4 Altitude Sensing

The BMP180 is a barometric pressure sensor that operates based on piezoresistive technology. It measures atmospheric pressure by detecting changes in resistance within a microfabricated membrane exposed to air pressure. The sensor converts this pressure reading into a digital value, which can then be used to calculate altitude using standard atmospheric models.

The initialization function for the BMP180, shown in Listing 11, begins by verifying the sensor's readiness and resetting it to ensure proper operation. It checks the device ID to confirm the module is functional and reads calibration data from the sensor's memory. This calibration data is validated and stored in the BMP180 structure for use in subsequent pressure and altitude calculations.

Additionally, the function configures the sensor's oversampling settings, which determine the precision and timing of measurements. The default sea level pressure is also initialized as a baseline for altitude calculations.

```
uint8_t bmp180_init(I2C_HandleTypeDef *hi2cx, bmp180_t *bmp180)
{
    bmp180->hi2cx = hi2cx;

    if (bmp180_is_ready(bmp180)) return 1; // Check if the device is ready

    uint8_t buffer[22];

    // Reset the sensor and verify the device ID
    bmp180_write(bmp180, SOFT, (uint8_t[]){0xB6}, 1);
    HAL_Delay(10);
    bmp180_read(bmp180, ID, &buffer[0], 1);
    if (buffer[0] != 0x55) return 2;

    // Read calibration data and validate it
    bmp180_read(bmp180, CALIB, buffer, 22);
    if (validate_calibration_data(buffer)) return 2;

    // Configure oversampling based on settings
    bmp180->oss = configure_oversampling(bmp180->oversampling_setting);

    // Save calibration data to structure fields
    save_calibration_data(buffer, bmp180);

    bmp180->sea_pressure = 101325; // Set default sea level pressure
    return 0;
}
```

Listing 11: Initialization function of the BMP180

The process for obtaining temperature and pressure readings from the BMP180 involves multiple steps as outlined in its datasheet. These steps include initiating measurements, reading raw data, and applying compensation algorithms using calibration coefficients stored in the sensor. Figure 67, illustrates this calculation process, which ensures the sensor provides accurate temperature and pressure values.

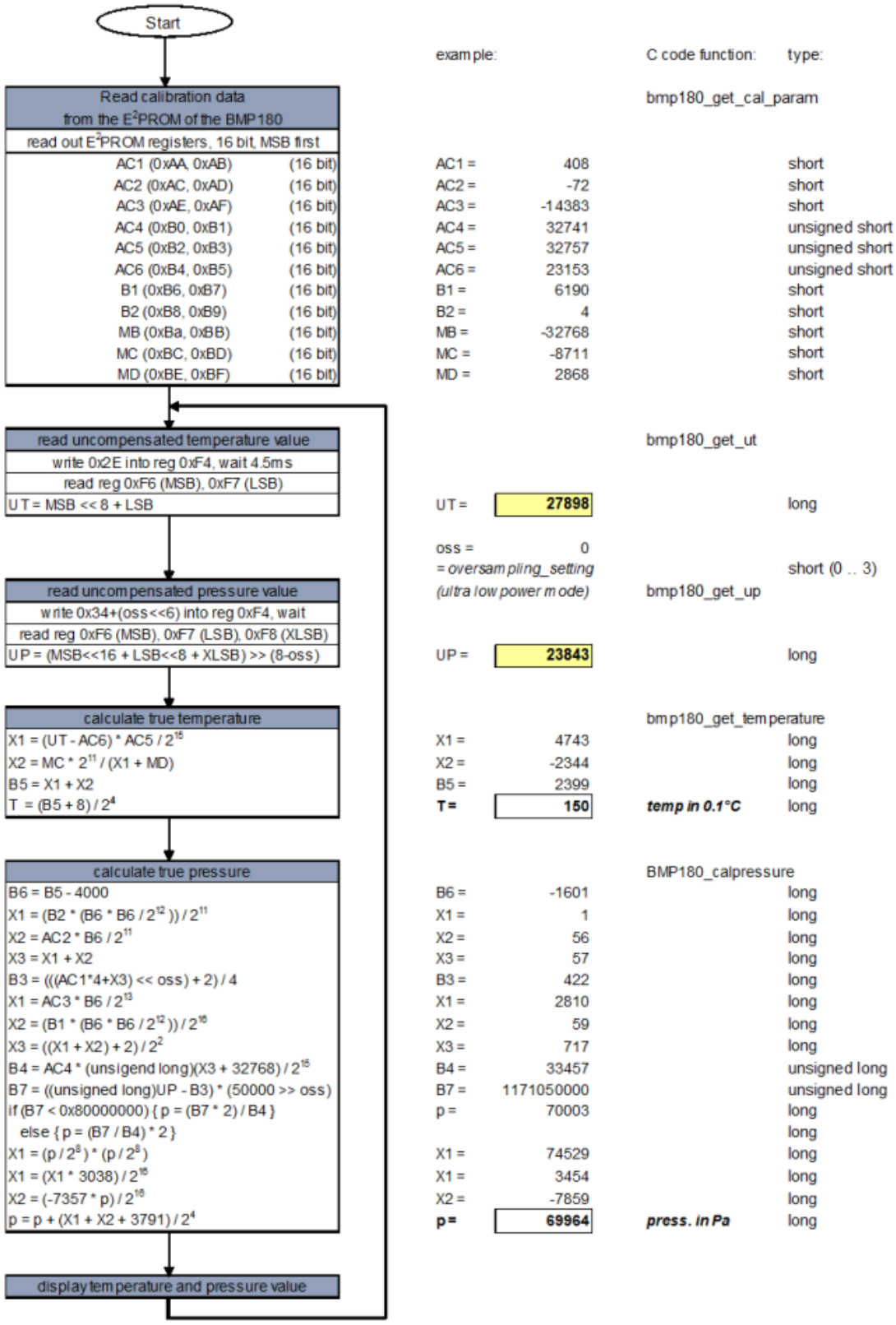



Figure 67: Calculation of pressure and temperature of the BMP180 [29]

The process of retrieving and calculating temperature, pressure, and altitude using the BMP180 is implemented in a unified function, as shown in Listing 12. This function retrieves raw temperature and pressure data from the sensor, applies the necessary calibration coefficients, and performs the compensation algorithms specified in the BMP180 datasheet [29].

```
void bmp180_get_measurements(bmp180_t *bmp180, float *temperature, int32_t
*pressure, float *altitude)
{
    int32_t X1, X2, X3, up, UT;
    uint32_t B7;

    // Read uncompensated temperature (UT)
    UT = _bmp180_read_ut(bmp180);

    // Calculate true temperature
    X1 = (UT - bmp180->AC6) * bmp180->AC5 / powerof2(15);
    X2 = (bmp180->MC * powerof2(11)) / (X1 + bmp180->MD);
    bmp180->B5 = X1 + X2;
    *temperature = (bmp180->B5 + 8) / powerof2(4) / 10.0; // Convert to °C

    // Read uncompensated pressure (UP)
    up = _bmp180_read_up(bmp180);

    // Calculate B6
    bmp180->B6 = bmp180->B5 - 4000;

    // Calculate intermediate values for pressure
    X1 = (bmp180->B2 * (bmp180->B6 * bmp180->B6 / powerof2(12))) /
powerof2(11);
    X2 = bmp180->AC2 * bmp180->B6 / powerof2(11);
    X3 = X1 + X2;
    bmp180->B3 = (((bmp180->AC1 * 4 + X3) << bmp180->oss) + 2) / 4;

    X1 = bmp180->AC3 * bmp180->B6 / powerof2(13);
    X2 = (bmp180->B1 * (bmp180->B6 * bmp180->B6 / powerof2(12))) /
powerof2(16);
    X3 = ((X1 + X2) + 2) / powerof2(2);
    bmp180->B4 = bmp180->AC4 * (uint32_t)(X3 + 32768) / powerof2(15);

    B7 = ((uint32_t)up - bmp180->B3) * (50000 >> bmp180->oss);

    // Calculate pressure
    if (B7 < 0x80000000) {
        *pressure = (B7 * 2) / bmp180->B4;
    } else {
        *pressure = (B7 / bmp180->B4) * 2;
    }
    X1 = (*pressure / powerof2(8)) * (*pressure / powerof2(8));
    X1 = (X1 * 3038) / powerof2(16);
    X2 = (-7357 * *pressure) / powerof2(16);
    *pressure = *pressure + (X1 + X2 + 3791) / powerof2(4);

    // Calculate altitude based on pressure
    *altitude = 44330.0 * (1 - pow(((float)*pressure / (float)bmp180-
>sea_pressure), 1 / 5.255));
}
```

Listing 12: Function including data retrieval and transfer functions of the BMP180

To measure distance with the RCWL-1670, the microcontroller sends a pulse to the module, which emits an ultrasonic signal and waits for the reflection. Upon detecting the reflection, the module generates a pulse for the microcontroller. This process enables near-ground altitude sensing or obstacle detection in the vehicle.

The initiation function, as shown in Listing 13, starts a measurement cycle by sending a 1 ms pulse to the module's trigger pin and activating a timer to count the elapsed time. The timer is then used to calculate the distance based on the sound wave’s time of flight.

```
void RCWL1670_StartMeasurement(void) {
    // Send pulse to RCWL-1670 module
    HAL_GPIO_WritePin(RANGE_OUT_GPIO_Port, RANGE_OUT_Pin, GPIO_PIN_SET); //
Set RANGE_OUT high
    HAL_Delay(1);                                                        //
Wait 1 ms
    HAL_GPIO_WritePin(RANGE_OUT_GPIO_Port, RANGE_OUT_Pin, GPIO_PIN_RESET);
// Set RANGE_OUT low

    // Reset and start TIM6
    __HAL_TIM_SET_COUNTER(&htim6, 0);   // Reset TIM6 counter to 0
    HAL_TIM_Base_Start(&htim6);        // Start TIM6 in the background

    // Clear previous measurement data
    measurement_ready = 0;
}
```

Listing 13: Pulse generation and time count initiation function for the RCWL-1670

An interrupt callback function is a mechanism that executes a predefined piece of code in response to a specific hardware event, such as a signal on a GPIO pin. In this case, the interrupt callback is triggered when the RCWL-1670 module generates a pulse on its Echo pin, indicating that the reflected ultrasonic signal has been received.

The callback function, shown in Listing 14, processes this event by capturing the elapsed time using the timer counter, which measures the time of flight of the ultrasonic pulse. The distance is then calculated in centimeters using the transfer function:

$$\text{Distance (cm)} = \frac{\text{Time elapsed (µs)}}{58} \tag{7}$$

The value 58 is derived from the speed of sound in air, which is approximately 343 m/s under standard atmospheric conditions (20 °C at sea level). Since the ultrasonic signal travels to the target and back, the effective speed is halved, and the unit conversions result in the constant 58 for distance in centimeters. After calculating the distance (7), the function stops the timer and marks the measurement as ready, allowing the system to use the result for further processing. This ensures precise and real-time response to the Echo signal.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    if (GPIO_Pin == RANGE_IN_Pin) {  // Check if triggered by RANGE_IN_Pin
        // Capture the elapsed time
        pulse_end_time = __HAL_TIM_GET_COUNTER(&htim6);

        // Stop the timer since the echo has been received
        HAL_TIM_Base_Stop(&htim6);

        // Calculate the distance using the transfer function
        // Distance (cm) = Time elapsed (µs) / 58
        RCWL1670_Distance = pulse_end_time / 58;

        // Mark the measurement as ready
        measurement_ready = 1;
    }
}
```

Listing 14: Callback function with transfer function for the RCWL-1670

### 10.1.5 ESC Control

Pulse Width Modulation (PWM) is generated by configuring a timer to produce pulses with a specific frequency and duty cycle. The frequency determines how often the pulse repeats, while the duty cycle defines the proportion of time the signal remains high within one cycle. In this case, TIM4 is used to generate PWM signals on four channels for controlling the ESCs.

Figure 68, shows the TIM4 Mode and Configuration menu, where the timer is set with a prescaler of 15 and a counter period of 19999. The prescaler divides the system clock (16 MHz) by 16, reducing the effective clock for the timer to:

$$\text{Timer Clock} = \frac{\text{System Clock}}{\text{Prescaler} + 1} = \frac{16\ \text{MHz}}{16} = 1\ \text{MHz} \tag{8}$$

The counter period of 19999 determines the total number of timer ticks in one cycle. With the timer clock (8) at 1 MHz, the period translates to:

$$\text{Cycle Duration} = \frac{\text{Counter Period} + 1}{\text{Timer Clock}} = \frac{20000}{1\ \text{MHz}} = 20\ \text{ms} \tag{9}$$

The result (9) corresponds to a PWM frequency of 50 Hz, which is used for driving the ESCs used in the project.

The ESCs are configured to respond to pulse widths between 800 ticks (minimum) and 2000 ticks (maximum). With the timer running at 1 MHz, these values correspond to pulse lengths of:

$$\text{Pulse Length (min)} = \frac{800}{1\ \text{MHz}} = 0.8\ \text{ms} \tag{10.1}$$

$$\text{Pulse Length (max)} = \frac{2000}{1\ \text{MHz}} = 2.0\ \text{ms} \tag{10.2}$$

These (10) pulse lengths represent the range of commands sent to the ESCs, where 0.8 ms corresponds to minimum throttle, and 2.0 ms corresponds to maximum throttle.

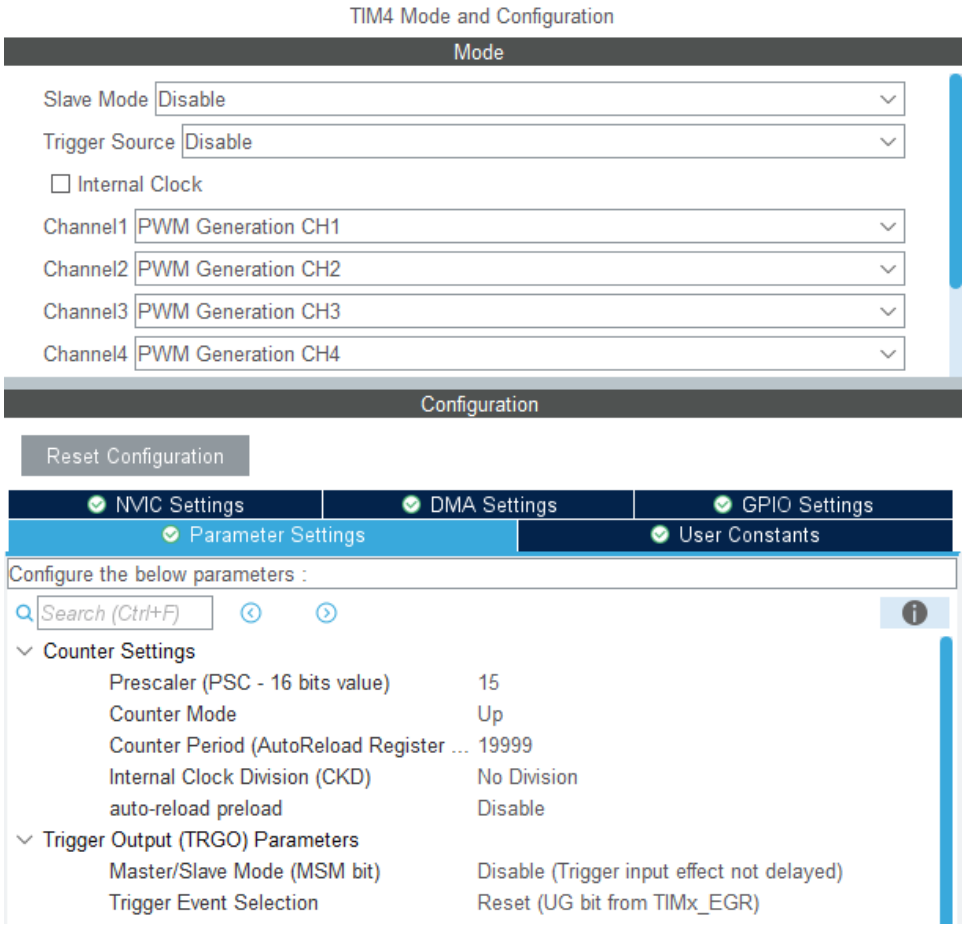



Figure 68: TIM4 Mode and Configuration in the STM32CubeIDE

The function responsible for updating motor speed requests and initialization of PWM output for the ESCs is presented in Listing 15. The function directly modifies the duty cycle of each PWM channel on TIM4 by updating the corresponding capture/compare registers, thereby controlling the motor speed. The initialization of the PWM for all four channels is achieved through calls to HAL_TIM_PWM_Start, which activate the PWM generation on each channel.

```
void UpdateMotorSpeed(int motor1, int motor2, int motor3, int motor4)
{
    TIM4->CCR1 = motor1;  // Update duty cycle for Channel 1
    TIM4->CCR2 = motor2;  // Update duty cycle for Channel 2
    TIM4->CCR3 = motor3;  // Update duty cycle for Channel 3
    TIM4->CCR4 = motor4;  // Update duty cycle for Channel 4
}

HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);
```

Listing 15: PWM initialization and control function for ESCs

### 10.1.6 Voltage and Current Monitoring

The battery voltage and current flow measurements are obtained using a single ADC sequence that reads all four channels in succession. This approach ensures simple and efficient data acquisition for monitoring the battery and current consumption.

Listing 16, demonstrates the function responsible for fetching ADC readings, where the microcontroller sequentially samples the four configured ADC channels. These readings correspond to the individual cell voltages of the battery pack and the current flow measured through the Hall effect sensor. This enables real-time monitoring of the power system, ensuring the drone operates within safe limits.

```
void BatteryAndCurrent_ADC(void) {
    HAL_ADC_Start(&hadc1);

    HAL_ADC_PollForConversion(&hadc1, 100);
    Vbat1 = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 100);
    Vbat2 = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 100);
    Vbat3 = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 100);
    CurrentRaw = HAL_ADC_GetValue(&hadc1);

    HAL_ADC_Stop(&hadc1);
}
```

Listing 16: ADC readings fetching function

The battery voltage values are calculated using the following transfer functions:

- **Voltage Divider Scaling Factors:**

1. Cell 1 ($V_{\text{bat1}}$):

$$\text{Scale Factor} = \frac{100+330}{330} = 1.303 \tag{11.1}$$

2. Cells 1+2 ($V_{\text{bat2}}$):

$$\text{Scale Factor} = \frac{330+210}{210} = 2.571 \tag{11.2}$$

3. Cells 1+2+3 ($V_{\text{bat3}}$):

$$\text{Scale Factor} = \frac{470+160}{160} = 3.938 \tag{11.3}$$

- **Final Transfer Functions:**

$$\begin{gathered} V_{\text{OUT(Q)}} = \frac{V_{\text{CC}}}{2} = 1.65\,\text{V} \\ V_{\text{Cell1}} = V_{\text{ADC1}} \cdot 1.303 \\ V_{\text{Cell2}} = (V_{\text{ADC2}} \cdot 2.571) - V_{\text{Cell1}} \\ V_{\text{Cell3}} = (V_{\text{ADC3}} \cdot 3.938) - V_{\text{ADC2}} \cdot 2.571 \\ V_{\text{BatteryPack}} = V_{\text{ADC3}} \cdot 3.938 \end{gathered} \tag{11.4}$$

Where:

$$V_{\text{ADCx}} = \frac{\text{ADC Value}}{4095} \cdot 3.3. \tag{11.5}$$

These transfer function formulas (11) have been implemented in the function shown in Listing 17.

```
void CalculateBatteryVoltages(uint16_t Vbat1, uint16_t Vbat2, uint16_t
Vbat3,
                              float *VCell1, float *VCell2, float *VCell3,
float *VBatteryPack) {
    // Constants
    const float VREF = 3.3;       // Reference voltage
    const float ADC_RES = 4095.0; // 12-bit ADC resolution

    // Scaling factors for each divider
    const float SCALE_CELL1 = 1.303;    // For Vbat1 (100k to 330k)
    const float SCALE_CELL1_2 = 2.571; // For Vbat2 (330k to 210k)
    const float SCALE_CELL1_2_3 = 3.938; // For Vbat3 (470k to 160k)

    // Convert ADC readings to actual voltages
    float VoltageBat1 = (Vbat1 / ADC_RES) * VREF * SCALE_CELL1;
    float VoltageBat1_2 = (Vbat2 / ADC_RES) * VREF * SCALE_CELL1_2;
    float VoltageBat1_2_3 = (Vbat3 / ADC_RES) * VREF * SCALE_CELL1_2_3;

    // Calculate individual cell voltages and total pack voltage
    *VCell1 = VoltageBat1;
    *VCell2 = VoltageBat1_2 - VoltageBat1;
    *VCell3 = VoltageBat1_2_3 - VoltageBat1_2;
    *VBatteryPack = VoltageBat1_2_3;
}
```

Listing 17: Transfer functions for battery cell voltage monitoring

The ACS758xCB-200B-PSS current sensor is used to measure the current flow to the drone's motors. This Hall-effect-based sensor provides an analog voltage output proportional to the current passing through it. Key specifications from the datasheet include:

- **Sensitivity**: $10\,\mathrm{mV/A}$ at $V_{CC} = 3.3\,\mathrm{V}$.
- **Quiescent Output Voltage**:

$$\mathrm{V_{OUT}\,(Q)} = \mathrm{V_{CC}}/2 = 1.65\,\mathrm{V} \tag{12.1}$$

The current is derived from the sensor's output voltage using the following transfer function:

Voltage from ADC Reading:

$$V_{\mathrm{OUT}} = \left(\frac{\mathrm{ADC\ Value}}{4095}\right) \cdot 3.3 \tag{12.2}$$

Current from Voltage:

$$I = \frac{V_{\text{OUT}} - 1.65}{0.01} \tag{12.3}$$

Combined Transfer Function:

$$I = \frac{\left(\frac{\text{ADC Value}}{4095} \cdot 3.3\right) - 1.65}{0.01} \tag{12.4}$$

Simplified Form:

$$I = 330 \cdot \frac{\text{ADC Value}}{4095} - 165 \tag{12.5}$$

This function (12) translates the raw ADC values into current readings, enabling real-time monitoring of the current supplied to the motors. Listing 18, shows the implementation of this transfer function in code, ensuring that the sensor's output is accurately converted for use in the drone's power management system.

```
float CalculateCurrent(uint16_t CurrentRaw, float *CurrentActual) {
    const float SCALE_FACTOR = 330.0;  // Scale factor for current
calculation
    const float OFFSET = 165.0;       // Offset for midpoint
    const float ADC_RES = 4095.0;     // 12-bit ADC resolution

    // Compute current in amperes
    *CurrentActual = (SCALE_FACTOR * CurrentRaw / ADC_RES) - OFFSET;
}
```

Listing 18: Transfer function of the current measurement

## 10.2 Controller

The software implementation for the controller focuses on configuring its microcontroller and developing module-specific libraries to manage user interaction and communication with the drone. Key functionalities include wireless communication, input handling from joysticks and buttons, and display management for providing real-time feedback and navigating menu systems. Each module is designed to operate independently while allowing integration within the overall control system.

### 10.2.1 Microcontroller Configuration

The clock configuration for the controller's microcontroller mirrors the principles established for the drone’s microcontroller to maintain consistency and simplicity. The configuration, illustrated in Figure 69, uses an HSE oscillator at 16 MHz as the clock source, with the Clock Security System (CSS) enabled to ensure stability by switching to a backup clock in case of failure. Prescalers are configured so that all peripheral clocks operate at 16 MHz, simplifying timing calculations and ensuring uniformity across the system.

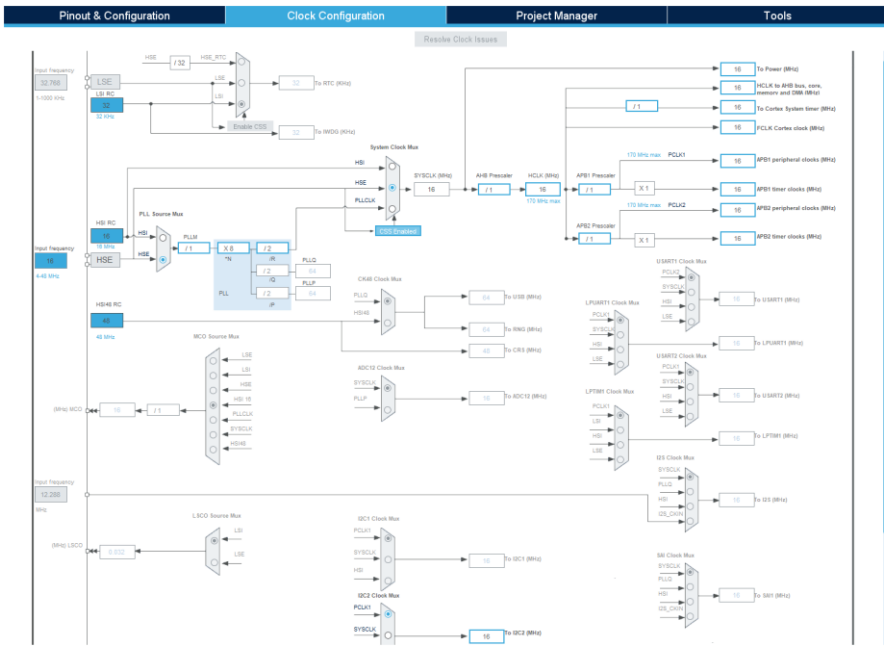



Figure 69: Clock configuration menu in STM32CubeIDE

The pin assignment for the controller's microcontroller, as visualized in Figure 70, demonstrates an organized layout where the majority of pins are utilized, leaving only three GPIOs unused. Notably, **PG10** is designated as the reset pin for the microcontroller.

The peripherals are grouped based on their functionality for logical organization. The top section primarily contains input pins configured with interrupts for the buttons, the left side is allocated to analog inputs for the joysticks, and the bottom section is dedicated exclusively to SPI communication with the NRF24 module. This arrangement optimizes the layout for efficient signal routing and peripheral management.

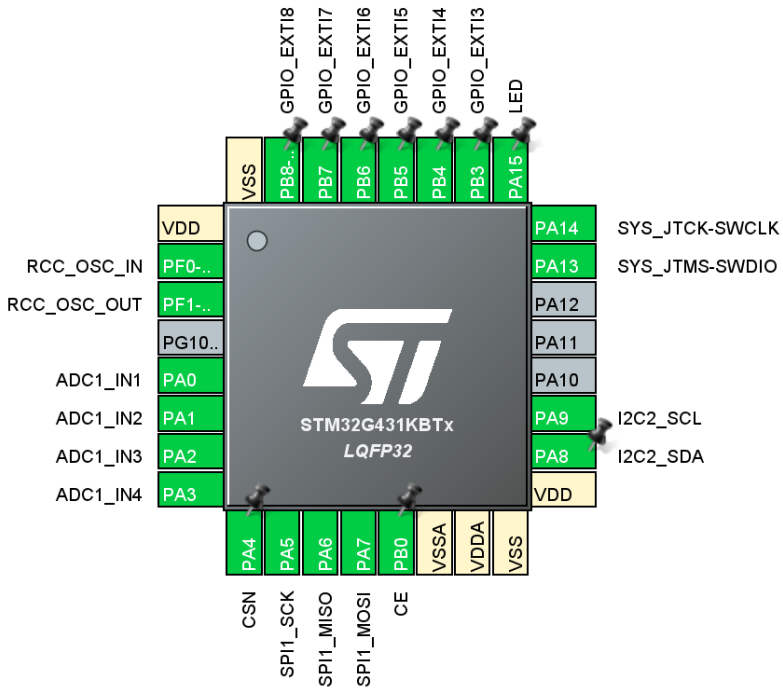
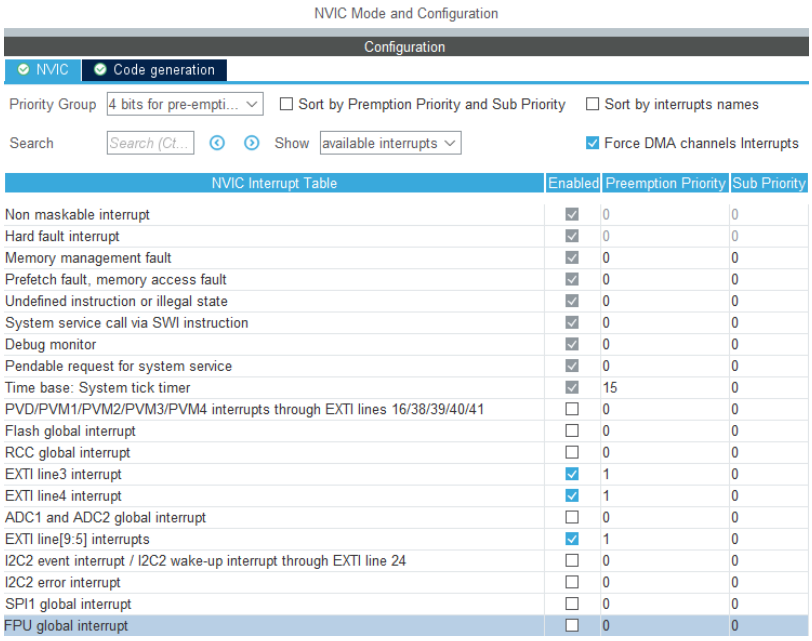



Figure 70: Pin configuration menu in STM32CubeIDE

The NVIC configuration, illustrated in Figure 71, includes the enabling of EXTI lines associated with the buttons. This setup ensures that button presses trigger interrupts, allowing the microcontroller to respond promptly and efficiently to user inputs. Proper interrupt configuration in the NVIC is essential for operation of the controller's input behavior.




| NVIC Interrupt Table | Enabled | Preemption Priority | Sub Priority |
|---|---|---|---|
| Non maskable interrupt | ☑ | 0 | 0 |
| Hard fault interrupt | ☑ | 0 | 0 |
| Memory management fault | ☑ | 0 | 0 |
| Prefetch fault, memory access fault | ☑ | 0 | 0 |
| Undefined instruction or illegal state | ☑ | 0 | 0 |
| System service call via SWI instruction | ☑ | 0 | 0 |
| Debug monitor | ☑ | 0 | 0 |
| Pendable request for system service | ☑ | 0 | 0 |
| Time base: System tick timer | ☑ | 15 | 0 |
| PVD/PVM1/PVM2/PVM3/PVM4 interrupts through EXTI lines 16/38/39/40/41 | ☐ | 0 | 0 |
| Flash global interrupt | ☐ | 0 | 0 |
| RCC global interrupt | ☐ | 0 | 0 |
| EXTI line3 interrupt | ☑ | 1 | 0 |
| EXTI line4 interrupt | ☑ | 1 | 0 |
| ADC1 and ADC2 global interrupt | ☐ | 0 | 0 |
| EXTI line[9:5] interrupts | ☑ | 1 | 0 |
| I2C2 event interrupt / I2C2 wake-up interrupt through EXTI line 24 | ☐ | 0 | 0 |
| I2C2 error interrupt | ☐ | 0 | 0 |
| SPI1 global interrupt | ☐ | 0 | 0 |
| FPU global interrupt | ☐ | 0 | 0 |

Figure 71: NVIC Mode and Configuration menu

The configurations for I2C and SPI in the controller are consistent with those used in the drone to maintain uniformity. The I2C interface is set to Fast Mode at 400 kHz with a 7-bit address length, ensuring efficient communication with peripherals such as the displays and the MPU6050 (not included in the first software implementation). The SPI interface is configured as Full-Duplex Master, using the Motorola frame format, an 8-bit data size, and MSB-first transmission, optimized for communication with the NRF24 module. These settings fulfil the fundamental requirements for the operation of interface-dependent modules.

Figure 72, shows the ADC mode and configuration menu. The four analog inputs for the joysticks are configured similarly to the drone's ADC settings. Each input is sequentially converted, ensuring simplicity of reading all sensors for processing joystick positions.

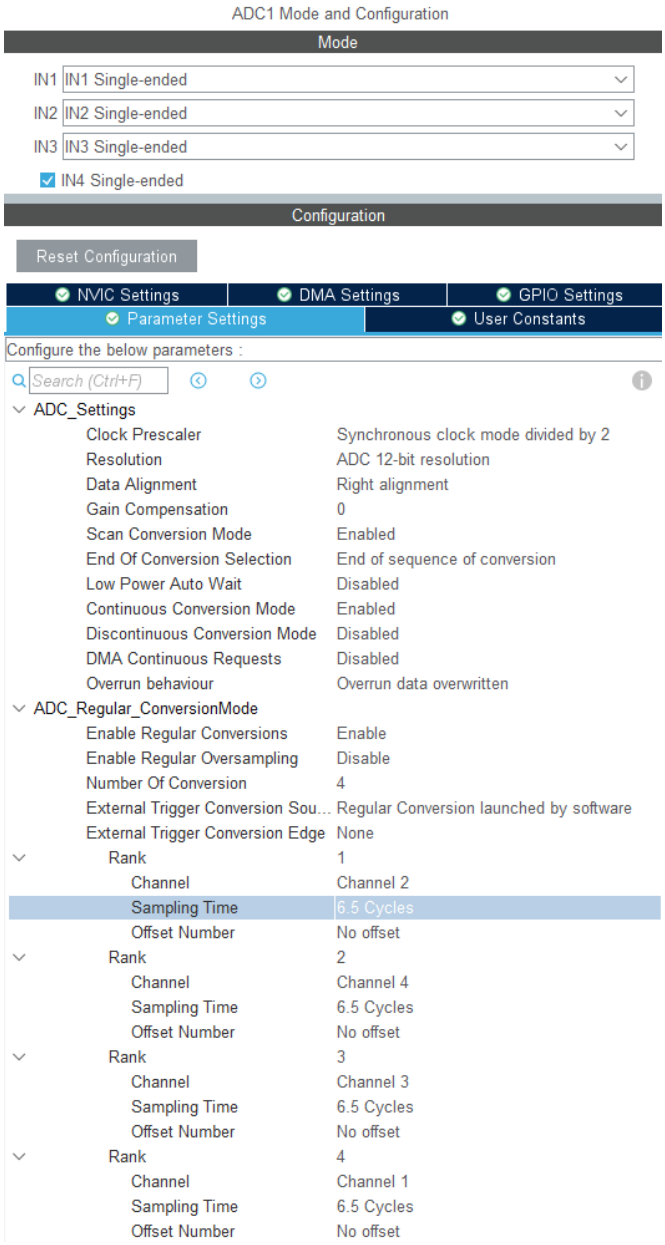



Figure 72: ADC1 Mode and Configuration menu in the STM32CubeIDE

### 10.2.2 Wireless Communication (NRF24L01P)

The wireless communication for the controller utilizes the same prepared NRF24 library as described in the drone section. This shared implementation ensures consistency in the operation of the communication system, simplifying development and testing while maintaining compatibility between the two modules.

### 10.2.3 Display Management

Each OLED display features a dedicated memory map, where commands and data define the pixel states for rendering text or graphics. The SSD1306 controller embedded in the display handles low-level operations, such as addressing pixels and managing display refreshes, allowing the microcontroller to focus on high-level data updates.

In the software implementation, the initialization function, shown in Listing 19, configures the SSD1306 controller by sending a sequence of commands over I2C. These commands define parameters such as display addressing mode, contrast, and scanning direction, ensuring proper functionality. Once initialized, the display memory is cleared, and the screen is updated to prepare it for rendering content like menu options or real-time status information.

```
void SSD1306_Init(SSD1306_t *oled, I2C_HandleTypeDef *hi2c) {
  oled->hi2cx = hi2c;
  oled->Address = SSD1306_I2C_ADDR;
  oled->Width = SSD1306_WIDTH;
  oled->Height = SSD1306_HEIGHT;
  oled->CurrentX = 0;
  oled->CurrentY = 0;
  oled->Inverted = 0;

  // Init OLED
  uint8_t init[] = {
    0xAE,         // Display OFF
    0xD5, 0x80,   // Set display clock divide ratio/oscillator frequency
    0xA8, 0x3F,   // Set multiplex ratio
    0xD3, 0x00,   // Set display offset
    0x40,         // Set start line
    0x8D, 0x14,   // Charge pump
    0x20, 0x00,   // Set memory addressing mode (horizontal)
    0xA1,         // Set segment remap
    0xC8,         // Set COM output scan direction
    0xDA, 0x12,   // Set COM pins hardware configuration
    0x81, 0xCF,   // Set contrast control
    0xD9, 0xF1,   // Set pre-charge period
    0xDB, 0x40,   // Set VCOMH deselect level
    0xA4,         // Set entire display ON
    0xA6,         // Normal display (non-inverted)
    0xAF          // Display ON
  };

  // Send initialization commands
  for (uint8_t i = 0; i < sizeof(init); i++) {
    SSD1306_WriteCommand(oled, init[i]);
  }

  // Clear screen
  SSD1306_Fill(oled, SSD1306_COLOR_BLACK);
  SSD1306_UpdateScreen(oled);
}
```

Listing 19: Initialization function of the SSD1306 OLED

The OLED displays are managed through a set of prototype functions, as shown in Listing 20, that handle the formatting and rendering of text and values. These functions provide flexibility in presenting data, accommodating various formats such as strings, integers, and floating-point numbers, either individually or in combination. This allows dynamic content, such as telemetry data or menu options, to be displayed effectively.

```
//Displays a simple string on the OLED
void SSD1306_text(const char *text) {
    SSD1306_WriteString(&oled, (char *)text, &Font_7x10,
SSD1306_COLOR_WHITE);
}

//Formats and displays an integer value
void SSD1306_INT(SSD1306_t *oled, uint16_t x){
     char buffer[64];
     snprintf(buffer, sizeof(buffer),"", x);
     SSD1306_WriteString(oled, buffer, &Font_7x10, SSD1306_COLOR_WHITE);
}

//Combines a string and an integer in a single line
void SSD1306_textINT(SSD1306_t *oled, const char *text, uint16_t x){
     char buffer[64];
     snprintf(buffer, sizeof(buffer), "%s %d", text, x);
     SSD1306_WriteString(oled, buffer, &Font_7x10, SSD1306_COLOR_WHITE);
}

//Formats and displays a string followed by a floating-point number
void SSD1306_textDBL(SSD1306_t *oled, const char *text, double x) {
  char buffer[64];
  snprintf(buffer, sizeof(buffer), "%s %0.0f", text, x);
  SSD1306_WriteString(oled, buffer, &Font_7x10, SSD1306_COLOR_WHITE);
}

//Formats and displays three floating-point numbers side by side, suitable
for multi-axis data like IMU readings
void SSD1306_DBLDBLDBL(SSD1306_t *oled, double x, double y, double z) {
    char buffer[64];
    snprintf(buffer, sizeof(buffer), "%4.0f %4.0f %4.0f", x, y, z);
    SSD1306_WriteString(oled, buffer, &Font_7x10, SSD1306_COLOR_WHITE);
}

//Combines three floating-point numbers with a descriptive string, offering
a detailed display option
void SSD1306_DBLDBLDBLtext(SSD1306_t *oled, double x, double y, double z,
const char *text) {
    char buffer[64];
    snprintf(buffer, sizeof(buffer), "%4.0f %4.0f %4.0f %s", x, y, z, text);
    SSD1306_WriteString(oled, buffer, &Font_7x10, SSD1306_COLOR_WHITE);
}
```

Listing 20: Prototype functions for displaying text on the SSD1306 OLED

### 10.2.4 Joystick Input

The function presented in Listing 21, cycles through all ADC channels configured in the controller, gathering raw analog readings corresponding to the joystick positions. These readings are processed sequentially and assigned to variables representing the X and Y axes of each joystick. The raw values, obtained directly from the ADC, are prepared for transmission to the drone via the NRF24 module, where they can be further processed for flight control and stability adjustments.

```
void UpdateJoystickValues(void) {

    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 100);
    Joy1X = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 100);
    Joy1Y = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 100);
    Joy2X = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 100);
    Joy2Y = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_Stop(&hadc1);

}
```

Listing 21: ADC data gathering function for the joystick position

### 10.2.5 Button Input

External Interrupts (EXTI) are used to detect and respond to specific events on GPIO pins, such as button presses. In this design, EXTI is configured to trigger interrupt requests when the state of a button changes. This enables the microcontroller to respond immediately, rather than continuously polling the button's status, which reduces processor overhead and improves responsiveness.

In the controller, debouncing (a process to eliminate false triggers caused by mechanical noise when buttons are pressed or released) is handled by capacitors on the PCB. These capacitors smooth out the voltage changes on the button pins, ensuring clean transitions that minimize spurious interrupts.

Listing 22, demonstrates the interrupt request handler (IRQ) functions that execute when specific buttons are pressed. Each IRQ handler first checks whether the interrupt flag for its corresponding pin is set. If so, the flag is cleared using __HAL_GPIO_EXTI_CLEAR_IT, and the appropriate handler function is executed.

```
void EXTI3_IRQHandler(void) {
    if (__HAL_GPIO_EXTI_GET_IT(GPIO_PIN_3) != RESET) {
        __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_3);  // Clear the interrupt flag
        HandleEscape();                        // Call the escape handler
    }
}

void EXTI4_IRQHandler(void) {
    if (__HAL_GPIO_EXTI_GET_IT(GPIO_PIN_4) != RESET) {
        __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_4);  // Clear the interrupt flag
        HandleEnter();                         // Call the enter handler
    }
}

void EXTI9_5_IRQHandler(void) {
    if (__HAL_GPIO_EXTI_GET_IT(GPIO_PIN_5) != RESET) {
        __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_5);  // Clear the interrupt flag
        HandleUp();                            // Call the up handler
    }
    if (__HAL_GPIO_EXTI_GET_IT(GPIO_PIN_6) != RESET) {
        __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_6);  // Clear the interrupt flag
        HandleDown();                          // Call the down handler
    }
}
```

Listing 22: Interrupt callback functions of the controller buttons

### 10.2.6 Menu Implementation

The controller features a prototype menu system that enables interaction through the OLED display and buttons. The menu is currently a proof of concept, consisting of placeholder options and a single functional feature. The functional option, "Initiate", triggers a configuration change by setting the global variable DroneConfig to 1. This action initiates the drone's ESC startup sequence, cycling the motor speeds as part of the initialization process. Other menu items serve as placeholders, leading to submenus that display their names but do not perform any additional actions.

Listing 23, presents the implementation of the RenderMenu function, which dynamically updates the OLED display based on the current menu state. If the user is in a submenu, the display shows a placeholder text along with the selected menu item's name. Otherwise, the main menu is rendered, highlighting the currently selected option with a > symbol.

```
// Global Variables
uint8_t currentMenuIndex = 0;  // Tracks the current menu index
uint8_t inSubMenu = 0;            // Tracks if we are in a submenu
uint16_t DroneConfig = 0;       // Configuration variable

// Menu Items
const char *mainMenu[] = {
    "Initiate",
    "Placeholder1",
    "Placeholder2"
};

// Render Menu Function
void RenderMenu(void) {
    SSD1306_Fill(&oled, SSD1306_COLOR_BLACK);

    if (inSubMenu) {
        // Show placeholder text in the submenu
        SSD1306_SetCursor(&oled, 0, 5);
        SSD1306_text(&oled, "Submenu:");

        SSD1306_SetCursor(&oled, 0, 17);
        SSD1306_text(&oled, mainMenu[currentMenuIndex]);
    } else {
        // Render Main Menu
        for (uint8_t i = 0; i < MAIN_MENU_COUNT; i++) {
            SSD1306_SetCursor(&oled, 0, 5 + (i * 12));
            if (i == currentMenuIndex) {
                SSD1306_text(&oled, ">"); // Highlight current menu item
            }
            SSD1306_text(&oled, mainMenu[i]);
        }
    }
    SSD1306_UpdateScreen(&oled);
}
```

Listing 23: Part of the menu implementation function group

The button press handling logic in the controller is implemented through a set of functions that manage menu navigation and trigger specific actions based on user input. Listing 24, demonstrates these functions, which respond to button presses by updating the current menu state or executing corresponding actions.

```
// Button Handlers
void HandleUp(void) {
    if (currentMenuIndex == 0) {
        currentMenuIndex = MAIN_MENU_COUNT - 1; // Wrap to the bottom
    } else {
        currentMenuIndex--;
    }
    RenderMenu();
}

void HandleDown(void) {
    currentMenuIndex = (currentMenuIndex + 1) % MAIN_MENU_COUNT; // Wrap to
the top
    RenderMenu();
}

void HandleEnter(void) {
    if (inSubMenu) {
        // Submenu Enter does nothing
    } else {
        if (currentMenuIndex == 0) {
            // Initiate action
            DroneConfig = 1;
            inSubMenu = 0; // Return to main menu after action
        } else {
            // Placeholder submenus
            inSubMenu = 1; // Enter submenu
        }
    }
    RenderMenu();
}

void HandleEscape(void) {
    if (inSubMenu) {
        inSubMenu = 0; // Return to main menu
        RenderMenu();
    }
    // Do nothing if already in the main menu
}
```

Listing 24: Functions corresponding to button presses

### 10.2.7 MPU6050 Accelerometer

The MPU6050 module is excluded from the initial software implementation, as the corresponding library has not yet been developed. However, the necessary hardware and communication configurations, such as the I2C interface setup, are already in place to facilitate its integration in future iterations. This ensures that the module can be added seamlessly when required, without the need for significant modifications to the existing system.

# 11 Tests and Results

The goal of this chapter is to evaluate the theoretical performance and preliminary software functionality of the drone system. With the physical prototype incomplete, testing focused on simulated calculations and standalone module validation. The drone’s performance metrics were estimated using the eCalc XcopterCalc platform, while software tests were conducted on Nucleo boards to verify core library implementations. The results provide valuable insights into the system’s expected performance and readiness for future integration and testing.

## 11.1 Drone Performance Metrics Calculation

To evaluate the performance of the drone in the absence of physical testing, the online platform eCalc XcopterCalc was utilized. This tool estimates critical flight metrics by simulating performance based on detailed component specifications and system parameters. Figure 73, illustrates the devices and parameters of the designed system that were used as inputs for the calculations, including motor specifications, propeller dimensions, battery configuration, and the total weight of the drone. Not all details perfectly align with the project composition due to the platform's limited selection of components. When an exact match is unavailable, the closest alternative with slightly inferior specifications is selected.

The objective of these calculations was to assess the feasibility of the design, validate the compatibility of components, and ensure that the drone could meet essential operational requirements, such as achieving stable hover, supporting payloads, and maintaining efficiency during flight. These calculations provide a theoretical benchmark, guiding further development and future physical testing.

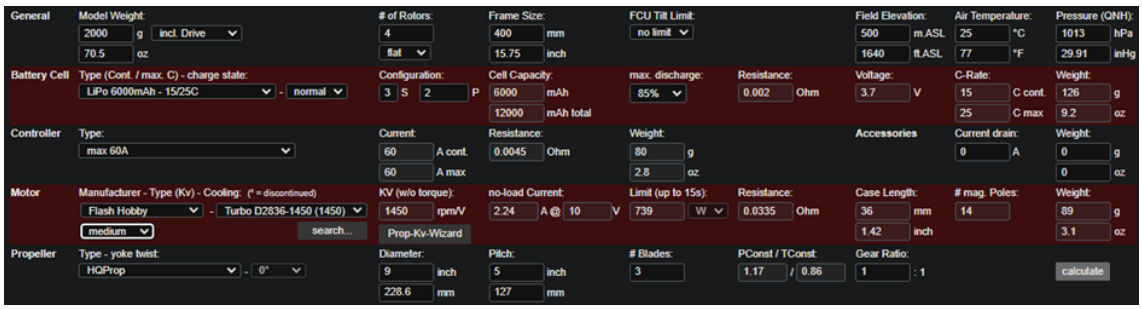



Figure 73: Input segment of the eCalc XcopterCalc

## 11.2 Software Tests

The software testing phase focused on validating the functionality of individual modules and ensuring compatibility between the microcontroller and peripheral components. These tests were conducted using Nucleo development boards as stand-ins for the drone and controller hardware, allowing for isolated module testing in a controlled environment.

The testing process followed a modular approach:

- **Peripheral Communication Validation:** Made sure that interfaces such as I2C, SPI, and ADC were properly configured and operating as intended. For instance, I2C-based devices were tested for correct data exchange using predefined initialization and communication functions.
- **Iterative Debugging:** Debugging tools in STM32CubeIDE, including variable inspection and breakpoint management, were used to iteratively refine code for each module. This ensured accurate data handling and processing across all peripherals.
- **Simulation of Operational Scenarios**: Key functionalities, such as PWM generation for motor control and ADC-based voltage monitoring, were simulated under conditions like expected real-world scenarios.
- **Functional Integrity:** Core library functions were tested individually to ensure their outputs matched the expected behavior. For example, sensor readings were compared to known values to verify accuracy.

While the focus was on overall software functionality, specific methods were applied to test different types of modules:

- **Communication Modules:** The NRF24 wireless library was tested by sending and receiving test payloads, confirming the integrity of the data and functionality of its operational modes.
- **Analog Inputs:** ADC configurations were tested with varying input signals to ensure the correct conversion of joystick positions and power system readings.
- **Graphical Output:** OLED display management functions were validated by rendering test patterns, text, and numeric data to confirm accurate visual representation.
- **Interrupt-Driven Inputs:** The button handlers, configured through EXTI interrupts, were verified by simulating button presses and monitoring the corresponding state transitions and actions.

## 11.3 Results

The performance metric results from the **eCalc XcopterCalc** platform, provide theoretical estimates of the drone's capabilities. The following results were obtained:

**Battery Metrics:**

- **Load:** 14.34 C
- **Voltage:** 10.07 V (under load)
- **Rated Voltage:** 11.10 V
- **Energy:** 133.2 Wh
- **Total Capacity:** 12,000 mAh (13,200 mAh in actual pack)
- **Used Capacity:** 10,200 mAh

These values indicate the energy demands and capacity utilization of the battery under load. The 14.34 C load reflects the high current demand during operation, while the rated and actual voltages highlight the voltage drop caused

by the load. The used capacity of 10,200 mAh represents the portion of the battery available for flight operations.

**Flight Times:**

- **Hover Flight Time:** 14.0 minutes
- **Mixed Flight Time:** 11.1 minutes
- **Minimum Flight Time:** 3.6 minutes

These values represent the expected flight durations under different scenarios. Hover flight time is calculated for steady flight with minimal maneuvering, while mixed flight time accounts for varying loads during typical operation. Minimum flight time indicates performance under maximum load conditions, such as aggressive maneuvers or high-speed flight.

**Motor Performance (Optimum Efficiency):**

- **Current:** 24.65 A
- **Voltage:** 10.40 V
- **Revolutions:** 13,776 rpm
- **Electric Power:** 256.3 W
- **Mechanical Power:** 213.7 W
- **Efficiency:** 83.4%

The motor performance at optimum efficiency reflects steady operation, balancing power consumption and output. At this point, the motor provides sufficient thrust with minimal energy loss.

**Motor Performance (Maximum Load):**

- **Current:** 43.03 A
- **Voltage:** 9.87 V
- **Revolutions:** 12,047 rpm
- **Electric Power:** 424.8 W
- **Mechanical Power:** 341.4 W
- **Efficiency:** 80.4%
- **Estimated Temperature:** 68°C

Under maximum load, the motors demonstrate significant power output, with a reduced efficiency due to thermal and electrical losses. The estimated temperature of 68 °C does not cause concern.

**Total Drive Metrics:**

- **Current @ Hover:** 43.79 A
- **Power (Input) @ Hover:** 486.1 W
- **Power (Output) @ Hover:** 379.3 W
- **Efficiency @ Hover:** 78.0%
- **Current @ Max:** 172.1 A
- **Power (Input) @ Max:** 1,910.3 W
- **Power (Output) @ Max:** 1,365.7 W
- **Efficiency @ Max:** 71.5%

These metrics summarize the overall power and efficiency characteristics of the propulsion system. The hover efficiency of 78.0% reflects effective energy use during steady flight, while the maximum efficiency of 71.5% highlights energy losses at peak performance.

**Flight Dynamics:**

- **Thrust-to-Weight Ratio:** 2.4:1
- **Maximum Speed:** 90 km/h
- **Rate of Climb:** 11.9 m/s
- **Additional Payload Capacity:** 2,201 g

The thrust-to-weight ratio indicates the system's capability to handle aggressive maneuvers and support payloads. The maximum speed and rate of climb showcase the drone’s dynamic performance capabilities, while the additional payload capacity provides flexibility for mission-specific equipment.

While physical integration testing one the designed PCBs was not possible at this stage, the software modules were validated on Nucleo development boards. Key outcomes from the testing include:

- **Library Validation:** All libraries, including those for wireless communication (NRF24), analog-to-digital conversions, and display management, performed as expected in isolation. Test payloads were successfully sent and received, sensor data was accurately read and displays rendered correctly.
- **Debugging and Iteration:** The debugging tools in STM32CubeIDE, combined with the SWD interface, allowed for effective troubleshooting and refinement of module functionality. Real-time inspection of variable states and peripheral configurations ensured that the code performed as intended.
- **Simulated Integration:** Though the modules were not tested as a complete system, simulated conditions on development boards confirmed their readiness for future integration.

## 11.4 Discussion – Design Choices

The project began with selecting the main components, a process influenced by compatibility with the chosen power source. The battery pack provides a total capacity of 13,200 mAh with a continuous discharge rate of 15 C, equating to a current discharge rating of 198 A. The current defined the specifications for the motors, Flash Hobby D2836 EVO 1450 KV maximum draw of 194.76 A (48.69 A each) meant over 3 A that would be available for the Flight Controller. The ESCs, rated at 60 A, provide sufficient overhead for current spikes, ensuring reliable operation under heavy load conditions. Propeller size was chosen to suit motor manufacturer’s recommendations, and a three-blade configuration was selected for lower noise profile. The initial propulsion testing using kitchen scale and 3D-printed structure as thrust tester showed promising results of over 900 g per motor. However, after initial tests, the tester was damaged, and the focus shifted to software development.

For the control system, STM32 microcontrollers were picked due to their comprehensive documentation, intuitive software tools, and robust debugging features. The initial choice of sensors included the MPU6050, but its

unmanageable drift during initial tests led to replacing it with the BNO055, which offered internally resolved sensor fusion and a magnetometer. The MPU6050 is now planned as an experimental feature for the controller, enabling motion-based input in the future.

## 11.5 Discussion – Challenges Encountered

The development of this embedded platform presented several challenges, particularly in programming and integration. One significant hurdle was establishing reliable communication between the NRF24 modules. Although initial experience with the NRF24 on Arduino was straightforward, adapting the module for use with STM32 proved more complex. Existing STM32 libraries for the NRF24 from online sources were insufficient, and after several unsuccessful attempts, it became clear that the issue lay in the software rather than the hardware. To overcome this, a Saleae Logic Analyzer was employed to observe the raw SPI communication when using the Arduino. Comparing the observed binary messages with the NRF24 datasheet allowed building a custom library from scratch, which provided a deeper understanding of the module's operation and limitations.

During initial communication tests, frequent communication failures surfaced, which led to the implementation of checkbytes and checksums for error detection and correction. Although this approach resolved the issue for now, future iterations may require optimizing the communication speed and power to meet the dynamic demands of the fully assembled system. Debugging and refining the software for individual modules required significant time, resulting in the delay of implementing PID controllers and sensor data integration.

## 11.6 Discussion – Future Work

The next steps involve completing physical integration and testing the prototype PCBs. Following successful testing, a revised PCB design will incorporate all components into a single board with improved ground planes and enhanced EMC performance. Developing and tuning the control system is a critical task.

Implementing PID controllers for stable hover and directional control will require iterative testing in a constrained environment, such as a suspension rig. The control loop will integrate real-time sensor data from the IMU and altitude sensors to adjust motor speeds dynamically.

Equipping the drone with payload-specific modules, such as cameras or environmental sensors, will expand its application scope. For instance, the drone's thrust-to-weight ratio and efficient power system make it suitable for aerial photography or air quality monitoring when paired with appropriate devices. Testing these applications will provide insights into real-world performance and further refine the design.

## 11.7 Discussion – Broader Perspective

Reflecting on the broader implications of this project, the future of drones appears promising. Despite geopolitical and economic challenges, component costs continue to decrease, making drone technology more accessible. Innovations in swarm algorithms and autonomous functionalities are poised to revolutionize commercial and military applications. However, increased regulation and licensing requirements may limit accessibility, shifting drones from a democratized technology to one dominated by governmental and corporate entities. Despite these challenges, the field remains a promising area for innovation and exploration.

# 12 Conclusion

This thesis explored the development of a modular embedded platform for a self-constructed drone system, fusing theoretical knowledge with practical application. The project served as an opportunity to understand the principles behind drone operation, embedded systems, and software development, reflecting the accessibility of modern resources and components.

Despite the challenges of integrating all modules into a fully functional system, significant progress was made in designing custom hardware, developing software libraries, and validating individual components. The use of simulation tools like eCalc XcopterCalc provided valuable insights into the drone's theoretical performance, demonstrating its capability to meet fundamental flight requirements. Metrics such as hover time, thrust-to-weight ratio, and efficiency indicate that the design aligns with expectations for a stable and versatile platform.

Software development and testing, conducted on Nucleo development boards, validated the functionality of critical modules in isolation. Libraries for wireless communication, sensor data processing, and display management performed as intended, laying a solid foundation for future integration.

This work represents an initial step in understanding and implementing embedded systems for drones. It underscores the value of modularity, accessibility, and iterative learning in tackling complex engineering projects. While the project is not yet fully realized, the knowledge gained and groundwork established provide a strong basis for future developments, whether to enhance the current system or pursue more specialized applications in the field of UAV technology.

## References


1. Custers B. Drones Here, There and Everywhere: Introduction and Overview. In: Custers B, editor. The Future of Drone Use: Opportunities and Threats. T.M.C. Asser Press; 2016. p. 3-20. DOI: 10.1007/978-94-6265-132-6_1.
2. Bálint M. History, Types, Application and Control of Drones. Safety and Security Sciences Review. 2022;4(3):2-4.
3. Fahlstrom PG, Gleason TJ. Ryan Firebee and UAV Applications in Vietnam. In: Introduction to UAV Systems. 4th ed. Hoboken: John Wiley & Sons; 2012. p. 4-7.
4. Military Saga. Gulf War and military intelligence [Internet]. 2024. Accessed 24.11.2024. Available from: https://militarysaga.com/gulf-war-and-military-intelligence/.
5. Smithsonian National Air and Space Museum. Predator drone: The innovation that transformed military combat [Internet]. 2018. Accessed 24.11.2024. Available from: https://airandspace.si.edu/stories/editorial/predator-drone-transformed-military-combat/.
6. Military Equipment HQ. Combat drone design considerations [Internet]. 2024. Accessed 24.11.2024. Available from: https://militaryequipmenthq.com/combat-drone-design-considerations/.
7. Caballero-Martin D, Lopez-Guede JM, Estevez J, Graña M. Artificial intelligence applied to drone control: A state of the art [Internet]. Drones. 2024;8(296):1-31. Accessed 24.11.2024. Available from: https://doi.org/10.3390/drones8070296.
8. Peksa J, Mamchur D. A review on the state of the art in copter drones and flight control systems. Sensors. 2024;24(3349):6-9. DOI: 10.3390/s24113349.
9. Seidu I, Olowu B, Olowu S. Advancements in quadcopter development through additive manufacturing: A comprehensive review. Int J Sci Res Sci Eng Technol. 2024;11(4):92-124. DOI: 10.32628/IJSRSET24114109.

10. T-Drones. Quadcopter drones: Everything you need to know [Internet]. Available from: https://www.t-drones.com/blog/quadcopter-drones.html. Accessed 24.11.2024.
11. Venkata Achuta Rao S, Srilatha P, Acharyulu GVRK, Suryanarayana. Introduction to drone flights: An eye witness for flying devices to the new destinations. In: Mohanty SN, Ravindra JVR, Narayana GS, Pattnaik CR, Sirajudeen YM, editors. Drone Technology: Future Trends and Practical Applications. Scrivener Publishing; Wiley; 2023. p. 21-51.
12. Highleap Electronic. Your ultimate guide to choosing the best drone PCB [Internet]. Accessed 24.11.2024. Available from: https://hilelectronic.com/drone-pcb/.
13. Drone Factor. UAV sensor systems: Essential tools for navigation and control [Internet]. 2018. Accessed 24.11.2024. Available from: https://dronefactor.co.uk/uav-sensor-systems/.
14. Fusion Engineering. Flight controller technology for UAVs [Internet]. 2021. Accessed 24.11.2024. Available from: https://fusion.engineering/wp-content/uploads/2021/05/Fusion-Engineering-Flight-Controller-Technology.pdf.
15. Gudde T. Flight controllers explained for everyone [Internet]. Fusion Engineering. 2021. Accessed 24.11.2024. Available from: https://fusion.engineering/flight-controllers-explained-for-everyone/.
16. Netline Technologies. How Drones are Being Controlled and the Way to Neutralise Them. [Internet]. 2024. Accessed 24.11.2024. Available from: https://netlinetech.com/how-drones-are-being-controlled-and-the-way-to-neutralise-them/.
17. Gur E. Drone Communication: Top Methods for UAV Connectivity. [Internet]. Elsight. 2024. Accessed 24.11.2024. Available from: https://www.elsight.com/blog/drone-connectivity-for-unmanned-aerial-vehicles-explained/
18. Hardesty G. Drones: The Wireless Technologies That Enable Operation and Control. [Internet. Data Alliance. 2023. Accessed 24.11.2024. Available from: https://www.data-alliance.net/blog/drones-the-wireless-technologies-that-enable-operation-and-control/.

19. Rico R, Rico-Azagra J, Sanz R, Sanz R. Hardware and RTOS Design of a Flight Controller For Professional Applications. IEEE Access. [Internet]. 2022;10:134870-134883. Accessed 24.11.2024. Available from: https://ieeexplore.ieee.org/document/9999639.
20. UAV Coach. Drone controller: Everything you need to know [Internet]. Accessed 24.11.2024. Available from: https://uavcoach.com/drone-controller/.
21. Omni Calculator. Drone Motor Calculator [Internet]. Release date: unknown. Accessed 24.11.2024. Available from: https://www.omnicalculator.com/other/drone-motor#drone-thrust-to-weight-ratio.
22. MPS. Brushless DC Motor Fundamentals: Application Note [Internet]. Release date: 2014. Accessed 24.11.2024. Available from: https://www.academia.edu/11475883/Brushless_DC_Motor_Fundamentals_Brushless_DC_Motor_Fundamentals_Application_Note.
23. Flash Hobby. D2836EVO Fixed Wing Motor [Internet]. Release date: unknown. Accessed 24.11.2024. Available from: https://www.flashhobby.com/d2836evo-fixed-wing-motor.html.
24. Galos J, Fredriksson C, Das R. Multifunctional sandwich panel design with lithium-ion polymer batteries [Internet]. Release date: 2020;23(8):3794-3813. Accessed 24.11.2024. Available from: https://doi.org/10.1177/1099636220946554.
25. Melasta. SLPBB042126 LiPo Battery Datasheet [Internet]. Release date: unknown. Accessed 24.11.2024. Available from: https://www.melasta.com/web/userfiles/download/SLPBB042126.pdf.
26. STMicroelectronics. STM32F446RE Datasheet [Internet]. Release date: January 2021. Accessed 24.11.2024. Available from: https://www.st.com/resource/en/datasheet/stm32f446re.pdf.
27. Nordic Semiconductor. nRF24L01+ Single Chip 2.4GHz Transceiver [Internet]. Release date: March 2008; v1.0. Accessed 24.11.2024. Available from: https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf.

28. Bosch Sensortec. BNO055 Intelligent 9-Axis Absolute Orientation Sensor [Internet]. Release date: November 2014; Document revision 1.2. Accessed 24.11.2024. Available from: https://cdn-shop.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf.
29. Bosch Sensortec. BMP180 Digital Pressure Sensor [Internet]. Release date: April 5, 2013. Accessed 24.11.2024. Available from: https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf.
30. Adafruit. Ultrasonic Distance Sensor - 3V or 5V - HC-SR04 Compatible - RCWL-1601 [Internet]. Release date: unknown. Accessed 24.11.2024. Available from: https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/4007_Web.pdf.
31. STMicroelectronics. STM32G431C6 Datasheet [Internet]. Release date: October 2021. Accessed 24.11.2024. Available from: https://www.st.com/resource/en/datasheet/stm32g431c6.pdf.
32. TDK InvenSense. MPU-6000 Datasheet [Internet]. Release date: 19.08.2013. Accessed 24.11.2024. Available from: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.
33. STMicroelectronics. SM6TxxA Datasheet [Internet]. Release date: September 2024. Accessed 24.11.2024. Available from: https://www.st.com/resource/en/datasheet/sm6t100a.pdf.
34. STMicroelectronics. Guidelines for Oscillator Design on STM8AF/AL/S and STM32 MCUs/MPUs [Internet]. Release date: November 2024. Accessed 24.11.2024. Available from: https://www.st.com/resource/en/application_note/an2867-guidelines-for-oscillator-design-on-stm8afals-and-stm32-mcusmpus-stmicroelectronics.pdf.
35. ECS Inc. CSM-3X Crystal Datasheet [Internet]. Release date: 2017. Accessed 24.11.2024. Available from: https://ecsxtal.com/store/pdf/CSM-3X.pdf