Design and Development of a Custom Quadcopter Flight Controller

Jinalka Herath

Table of Contents

1. Introduction 3
2. System Architecture 4
3. Hardware Design 6
3.1 Microcontroller Selection Rationale 6
3.1.1 Comparison: STM32F722 vs STM32F405, F411, F427 8
3.2 Sensor integration 10
3.2.1 MPU 6050 VS MPU 6000 12
3.3 Barometer 13
3.4 Flash Memory 15
3.5 GPS 17
3.6 ESC_CONNECTORS 18
4. Power and Actuation Layout 19
5. Orientation Estimation Logic 22
6. PID or chosen control strategy 24
7. Firmware Structure 27
..... 27
7.2 Key Functions and Modules 29
7.3 Timing & Scheduling 31
7.4 Pin Configuration and Peripheral Mapping 32
8. Limitations and Improvements 36
9. Appendix 37

1. Introduction

The purpose of this project is to design and develop a custom flight controller for a quadcopter, using a 32-bit STM32F722RET6 microcontroller without relying on Arduino or Arduino-based libraries. The project is focused on building a high-performance embedded system that supports real-time control, sensor data processing, and efficient communication, using industry-standard tools and techniques.

The goal is to:

- Achieve stable and responsive quadcopter flight through sensor feedback and motor control.
- Design from scratch a complete system including hardware schematics, firmware, and control algorithms.
- Avoid dependency on Arduino-based platforms to ensure optimized, low-level control with greater flexibility and professional applicability.

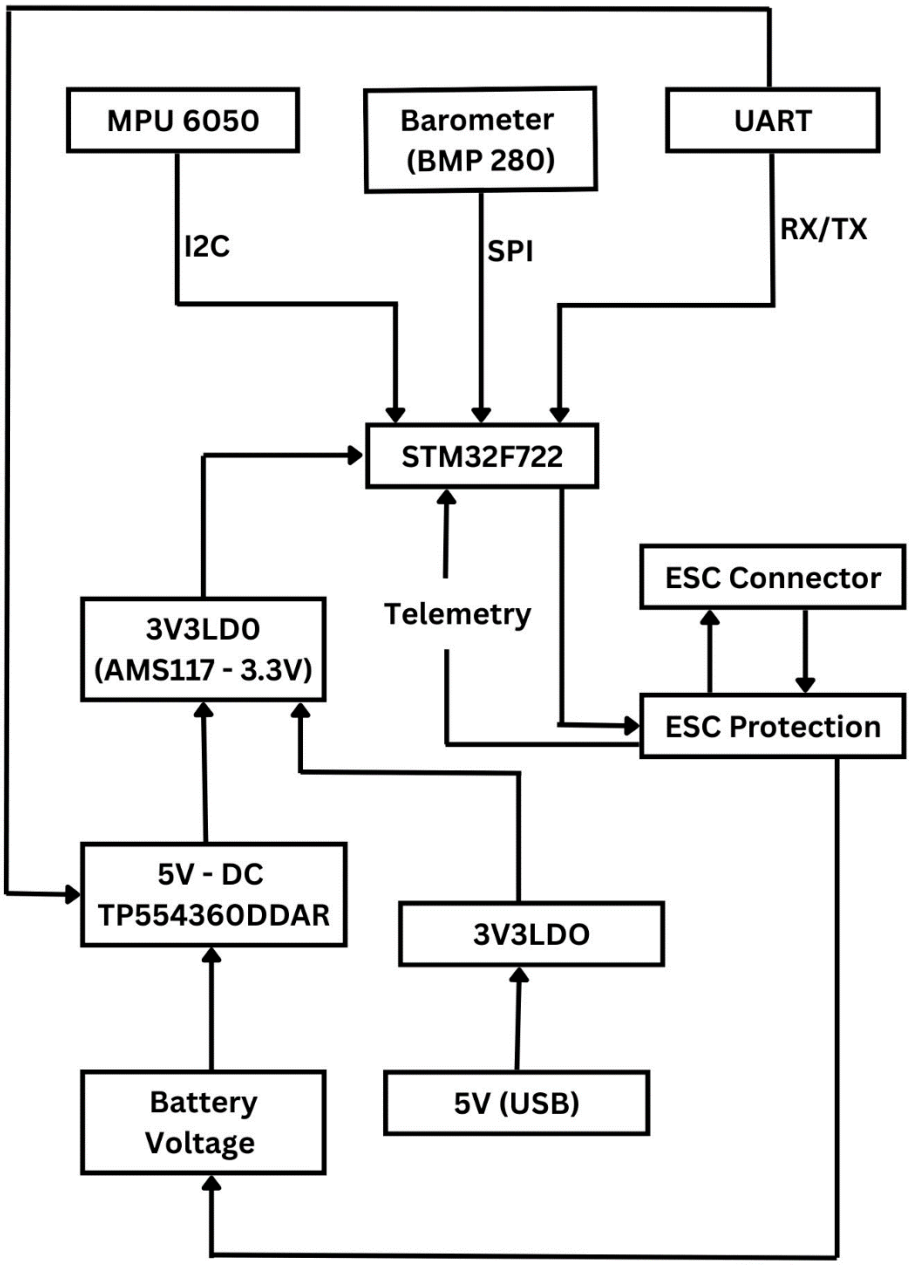
Why STM32F722RET6 is more efficient?

The STM32F722RET6 was selected based on the following advantages:

- **High Performance**: 32-bit ARM Cortex-M7 core with FPU running at 216 MHz, suitable for real-time control and floating-point PID calculations.
- **Advanced Timer Support**: 18 timers including advanced control timers (TIM1 and TIM8), ideal for generating precise PWM signals for ESC control.
- **Ample Memory**: 512 KB Flash and 256 KB SRAM for complex firmware and logging.
- **Rich Peripheral Set**: Multiple USARTs, I2C, SPI, and USB OTG HS/FS support enable flexible sensor and communication integration.
- **Professional Development Ecosystem**: Supported by STM32CubeIDE, ST-Link debuggers, and extensive STM HAL/LL drivers for scalable development.

This MCU strikes a balance between computational power, peripheral availability, and community support, making it highly suitable for flight controller development.

2. System Architecture



The given system architecture is use to represent the flight control and power management design of a drone, using the STM32F722 microcontroller as the core of the system. This microcontroller is responsible for handling sensor data, executing flight control algorithms, and communicating with various subsystems such as telemetry modules and motor controllers. The flight controller system integrates multiple components, including sensors, voltage regulators, and protection circuits, all working together to enable stable and safe flight.

The STM32F722 is a powerful ARM Cortex-M7 based microcontroller chosen for its high-speed processing capabilities, which is essential for real-time control in drone applications. It receives data from two key sensors: the MPU6050 and the BMP280. The MPU6050 is a 6-axis Inertial Measurement Unit (IMU) that includes a 3-axis gyroscope and a 3-axis accelerometer. Imu provides real-time orientation and motion data, which is critical for the flight controller in order to maintain stability and respond to movement. The BMP280 barometer measures atmospheric pressure in order to estimate altitude, which assists in height control and altitude hold features. These sensors send their data via I2C or SPI communication lines to the STM32F722, where sensor fusion algorithms process the data to determine the drone's precise attitude and altitude.

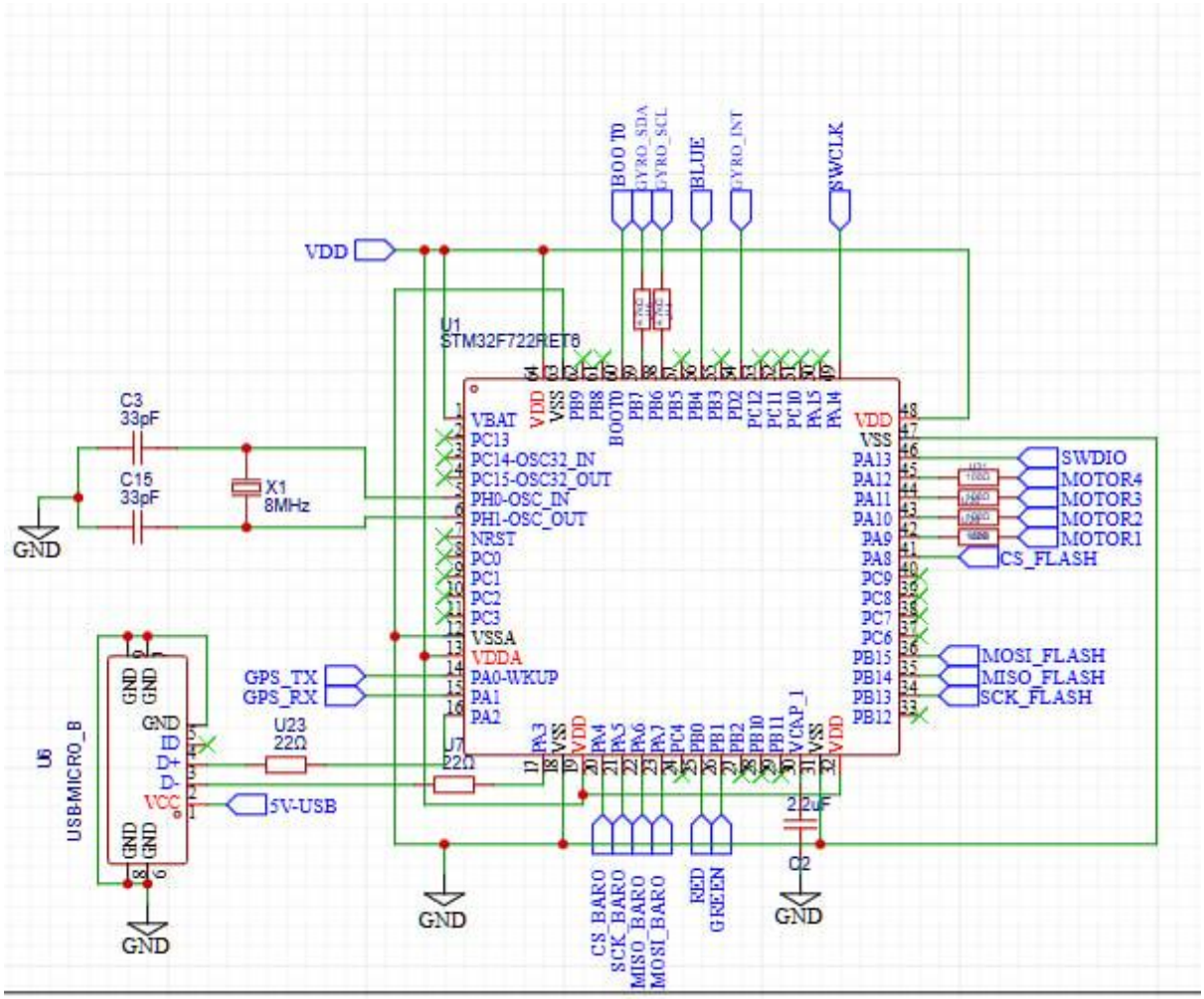
The UART interface provides serial communication between the flight controller and external modules, such as GPS. Data received through UART is used for commands, parameter updates, or GPS coordinates, while outgoing data includes battery status, and real-time sensor feedback.

To control the motors, the STM32F722 sends Pulse Width Modulated (PWM) signals to the ESC connectors, which are protected by an ESD (Electrostatic Discharge) protection circuit. This circuit safeguards the microcontroller against sudden voltage spikes and static discharges which occurs when switching high-speed motor lines. The ESCs convert these signals into appropriate voltages and currents to drive the drone’s brushless motors, controlling their speed and direction based on flight controller inputs.

Power management of the drone system is handled through a well-structured voltage regulation system. The battery (typically a LiPo pack) serves as the main power source. The battery voltage is first stepped down by a 5V-DC buck converter (TP554360DDAR), which provides a regulated 5V output. From this 5V rail, multiple LDO (Low Dropout) voltage regulators are used to supply clean and stable 3.3V power to sensitive components. The AMS1117-3.3V LDO powers components like sensors and the STM32F722. Another 3.3V LDO is connected to a USB 5V input, offering a secondary power source for debugging or firmware flashing.

The data flow of the drone system begins with sensor readings (IMU and barometer) which is sent to the STM32F722. Control algorithms inside the microcontroller are used to analyze this data and generate commands in order to adjust motor speeds via the ESCs. Simultaneously, telemetry data and communication over UART are handled for external monitoring. Control flow starts from power distribution where the battery feeds the DC converter, which supplies voltage to regulators, and these in turn power the microcontroller and sensors. Every signal and power line is organized for real-time control, safety, and efficient operation. This architecture supports reliable and stable drone flight.

3. Hardware Design



3.1 Microcontroller Selection Rationale

The STM32F722RET6 was selected as the core microcontroller for the flight controller due to its powerful performance, broad peripheral support, and energy-efficient design, all of which are essential for real-time drone applications. Built on the ARM Cortex-M7 core, the STM32F722 operates at a frequency of up to 216 MHz, providing the processing speed required for high-speed flight control computations. System is built-in Floating Point Unit (FPU) significantly accelerates mathematical operations, enabling accurate and fast execution of flight algorithms such as sensor fusion, PID control, and navigation logic. These capabilities are particularly crucial for ensuring precise stability and responsiveness during dynamic aerial maneuvers.

In addition to its performance, the microcontroller features 512 KB of Flash memory and 256 KB of SRAM, which offers sufficient space for storing the embedded firmware, handling multiple flight modes and logging sensor inputs.MCU memory capacity is vital for reliable operation, especially when integrating multiple sensors and executing real-time data processing on the fly. The STM32F722's memory also ensures scalability for future firmware

updates or advanced features such as blackbox flight data logging or waypoint-based autonomous missions.

The STM32F722RET6 is equipped with a variety of communication peripherals, making it highly suitable for integrating with diverse sensors and modules in the drone system. These include SPI interfaces for high-speed communication with components like the BMP280 barometer and external flash memory; I2C buses for sensors like the MPU6050 IMU; and USART/UART ports to support modules such as GPS and telemetry radios. Multiple PWM-capable timers allow precise motor control through the ESCs, and ADC channels which are available for monitoring analog signals, such as battery voltage levels. The presence of DMA channels are used to enhance the efficiency of the system by allowing peripherals to move data directly to and from memory without interrupting operation of the CPU.

Another advantage of the STM32F722 chip is its inbuilt debugging and boot options. The inclusion of Serial Wire Debug (SWD**)** which enables in-circuit programming and real-time debugging that simplifies development and testing. The BOOT0 pin configuration allows flexible selection between boot modes from system memory, Flash, or external sources which makes it easier to implement bootloaders or recovery firmware mechanisms. These features are especially very important for drone applications that requires firmware updates or diagnostics in the field.

Despite its powerful capabilities, the STM32F722 remains power-efficient, which is crucial for battery-operated drones. The microcontroller supports multiple low-power operating modes that enables energy-saving during flight hold or failsafe states. Its 64-pin LQFP package is also compact and easy to integrate into custom PCB designs for small and lightweight flight controllers.

Finally, one of the main reasons for selecting the STM32F722RET6 is its user friendly development ecosystem and strong community support. Tools like STM32CubeMX and STM32CubeIDE allow easy configuration of clocks, peripherals, and pin assignments. The stm microcontroller is compatible with a wide range of open-source libraries, including those for FreeRTOS and HAL drivers, which significantly accelerate development time. Additionally, its widespread use in the drone and robotics community ensures the availability of documentation, forums, and shared codebases for troubleshooting and enhancements. All these factors together make the STM32F722RET6 an ideal choice for this drone flight controller design.

3.1.1 Comparison: STM32F722 vs STM32F405, F411, F427

Parameter	STM32F722RET6	STM32F405RG	STM32F411RE	STM32F427VIT6
Core Architecture	Cortex-M7 (ARM v7-M)	Cortex-M4	Cortex-M4	Cortex-M4
Max Clock Speed	**216 MHz**	168 MHz	100 MHz	180 MHz
Flash Memory	**512 KB**	1 MB	512 KB	2 MB
RAM (SRAM)	**256 KB**	192 KB	128 KB	256 KB
FPU (Floating Point Unit)	**Single-precision HW**	Single-precision HW	Single-precision HW	Single-precision HW
Instruction Cache / TCM	**I/D cache + TCM**	No	No	Only I-cache
CoreMark Score *(Performance)*	**1082**	469	273	602
SPI / I2C / UART Ports	Rich peripheral set	Less compared to F7	Limited	Rich peripheral set
DMA Channels	16 DMA streams (2x Ctrl)	12 DMA streams	8 DMA streams	16 DMA streams
Timers	14	17	13	17
ADC Resolution	12-bit	12-bit	12-bit	12-bit
Price Range (as of 2024)	$7–9 USD	$5–7 USD	$4–6 USD	$9–12 USD

When comparing the STM32F722RET6 to other widely used microcontrollers from the STM32F4 series specifically the STM32F405**,** STM32F411**,** and STM32F427 becomes evident that the F7 series offers substantial advantages in terms of performance, memory architecture, and suitability for high-demand applications like drone flight control systems. The most significant improvement lies in the core architecture. While the STM32F4 series is based on the ARM Cortex-M4 core, the STM32F722RET6 uses more advanced Cortex-M7

core, which operates at a higher frequency of 216 MHz and includes advances instruction pipelines and a hardware floating-point unit (FPU)**.** This makes the STM32F722 significantly faster in executing complex control algorithms, sensor fusion, and real-time signal processing, with CoreMark performance scores reaching over 1000, nearly double that of the STM32F405 or F427.

Memory capacity and architecture also play a critical role where the STM32F722RET6 features 256 KB of SRAM, which is on par or better than many STM32F4 variants, and includes I/D (Instruction/Data) caches and Tightly Coupled Memory (TCM)**.** These architectural enhancements are use to reduce the latency and boost execution speed, particularly important for drones that require high-frequency sensor sampling and motor control. In comparision, the STM32F411 has only 128 KB of RAM and lacks instruction caching, making it less suitable for more advanced, multi-threaded flight control applications.

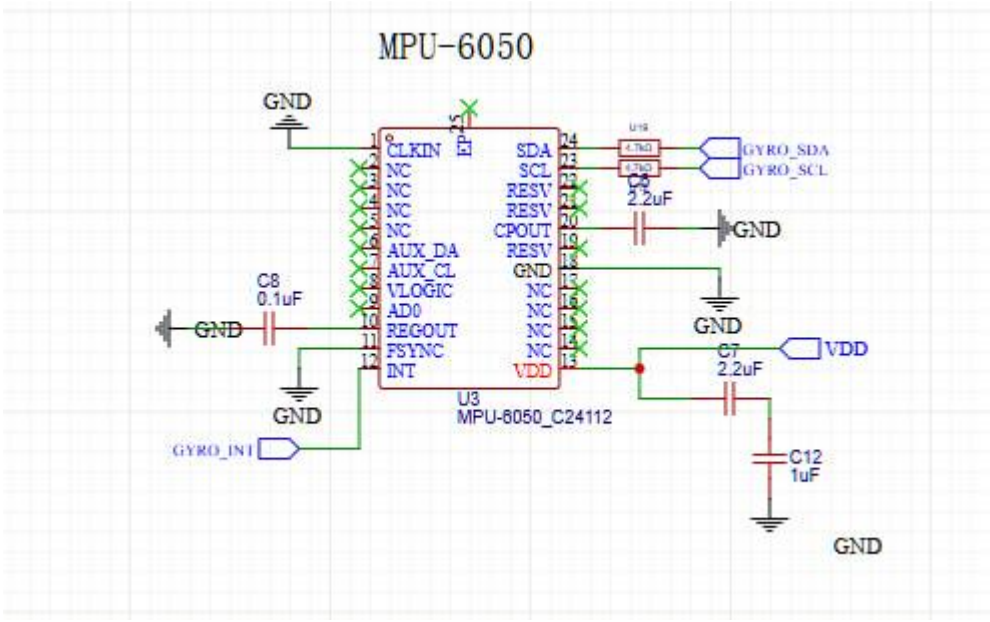
Furthermore, the STM32F722 boasts 16 DMA channels distributed across two controllers**,** allowing multiple high-speed data transfers to occur without burdening the CPU. This is especially beneficial for drones, where real-time communication with sensors like IMUs, barometers, GPS modules, and telemetry systems is essential. The F4 series, while offering decent DMA capabilities (e.g., 12 streams in F405), still falls behind in bandwidth and flexibility.

Another advantage of the STM32F722 is its energy-efficient performance, thats ideal for battery-powered systems. Despite running at a higher frequency, it offers advanced power modes that allow the flight controller to reduce power consumption during idle or standby phases, such as during loiter or failsafe conditions. This extends the drone's overall flight time and improves efficiency without compromising responsiveness.

In addition, the STM32F722 includes enhanced debugging and boot options, such as Serial Wire Debug (SWD) support and flexible BOOT pin configurations, which aid in real-time debugging, firmware upgrades, and fail-safe programming. It also includes an MPU (Memory Protection Unit) to safeguard critical memory regions, a feature lacking in the older F4 devices.

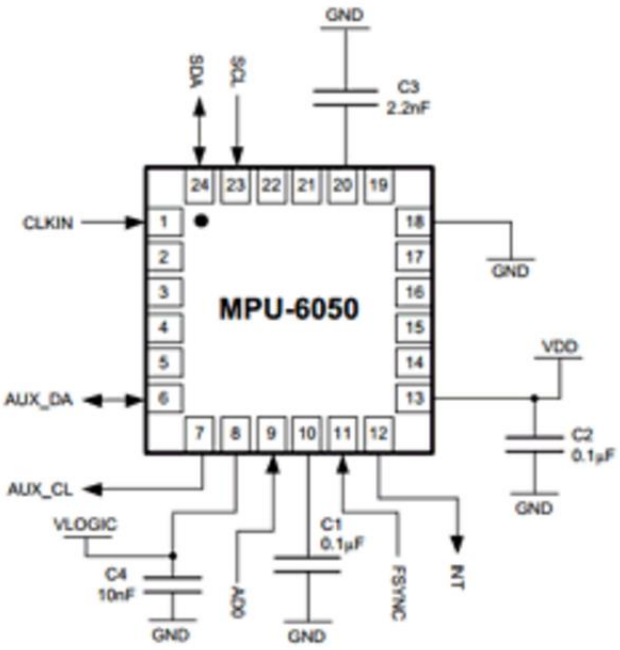
Finally, the STM32F722RET6 is fully supported by modern drone firmware stacks like Betaflight**,** iNav**,** and ArduPilot**,** many of which have started to deprecate older STM32F4 variants due to performance limitations. The F722 enables features such as high-rate sensor loops, fast ESC protocols like DShot, onboard logging, and OSD integration—all of which are critical for modern drones. Considering all these points, the STM32F722RET6 stands out as the most suitable and future-proof choice for a high-performance flight controller, offering superior speed, flexibility, and scalability compared to the STM32F4 family.

3.2 Sensor integration



The MPU6050 was selected as the primary motion sensing of the flight controller designed due to its versatility, compact design, and reliability in drone and robotics applications. Imu integrates a 3-axis gyroscope and a 3-axis accelerometer into a single chip, offering a full **6** degrees of freedom (6DOF) measurement that is essential for gaining the orientation and movement data. One of the key primary reasons for choosing the MPU6050 is its built-in Digital Motion Processor (DMP)**,** which can handle complex motion algorithms internally, reducing the computational load on the main microcontroller (STM32F722). Additionally, the MPU6050 supports both I2C communication, which is energy-efficient (a crucial feature in battery-powered drones), and provides high accuracy with programmable filters and

temperature compensation. Its cost-effectiveness, makes it a popular and practical choice in UAV systems.



As per the datasheet, the MPU6050 features important pins such **as** VDD**,** GND**,** SDA**,** SCL**,** INT, and AD0, among others. In the schematic provided the sensor is powered by a regulated 3.3V supply that is generated using an AMS1117-3.3V regulator. The ground (GND) is connected to the system's common ground. Communication between mpu and the microcontroller is established through the I2C protocol, using the GYRO_SCL and GYRO_SDA lines, which connect to the STM32's I2C peripheral pins. These lines are appropriately pulled up to 3.3V with resistors, ensuring clean signal transmission.

The AD0 pin, determines the I2C address of the device, which is connected to ground in this schematic, assigning the default address of 0x68 to the MPU6050. The INT pin is connected to a GPIO pin on the STM32F722 capable of handling external interrupts. This setup allows the MPU6050 to notify the microcontroller when new sensor data is available, thereby enabling an interrupt-driven data acquisition approach that improves processing efficiency and responsiveness. Other optional pins such as FSYNC**,** REGOUT, and CPOUT are not used in this pin configuration, aligning with a minimalist setup that focuses on essential operations.

To ensure electrical stability, decoupling capacitors (typically 0.1 µF and 2.2 µF) are placed close to the MPU6050's power pins, helping to filter out voltage spikes or noise from the power line. The physical placement of the sensor near the STM32 on the PCB is a design choice to minimize trace length, which helps maintain the integrity of high-speed I2C signals.

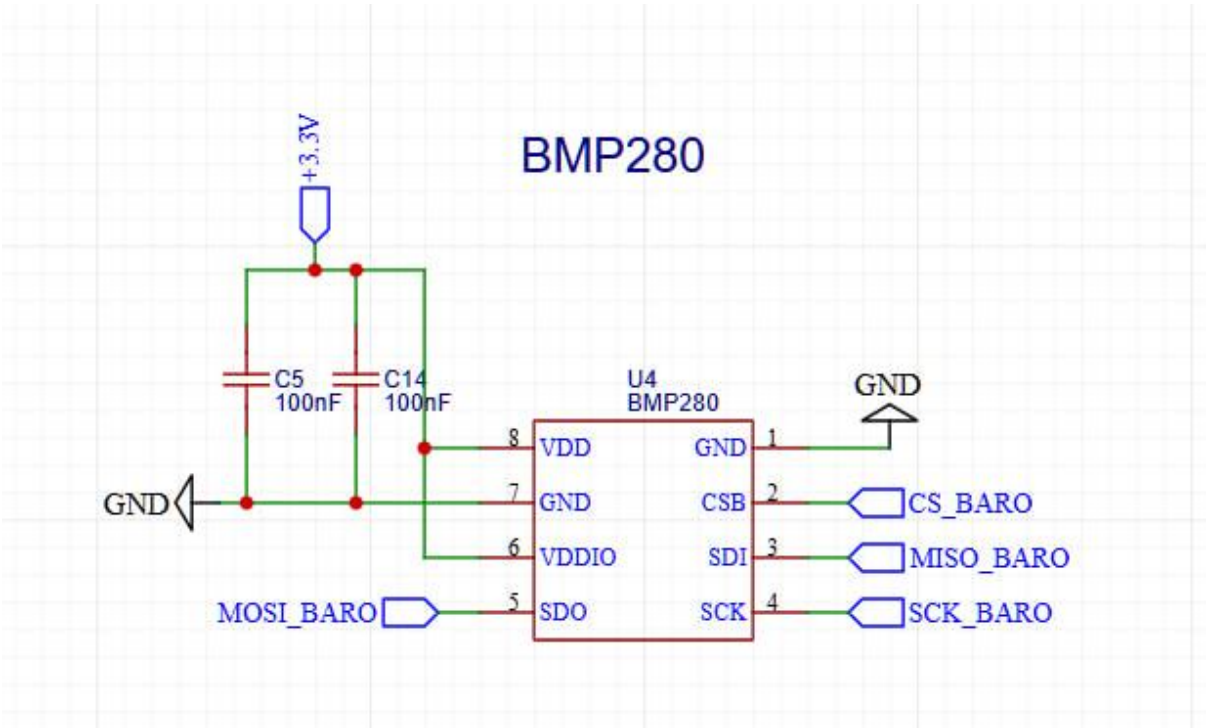
During actual flight operations, the MPU6050 plays a major role in acquiring real-time acceleration and angular velocity data. IMU information is sent to the STM32F722, where sensor fusion algorithms such as complementary filters or Kalman filters are used to calculate the drone's orientation (pitch, roll, yaw). The data is essential for real-time flight stabilization, auto-leveling, and maneuver corrections. By integrating the MPU6050 in this way, the system achieves a compact, power-efficient, and highly responsive motion sensing solution ideal for drone control applications.

3.2.1 MPU 6050 VS MPU 6000

Feature	MPU6050	MPU6000	Which is Better for This Project
Sensor Type	6-axis (3-axis gyro + 3-axis accel)	6-axis (3-axis gyro + 3-axis accel)	Same
Communication Interface	I²C only	I²C & SPI	**MPU6050** (simpler integration)
Digital Motion Processor (DMP)	Yes	Yes	Same
Noise Performance	Standard	Better (lower noise)	MPU6000 (but marginal benefit here)
Power Consumption	Lower typical current (~3.9 mA)	Slightly higher (~5.0 mA)	**MPU6050**
Package	QFN-24	QFN-24	Same
Temperature Range	-40°C to +85°C	-40°C to +105°C	MPU6000 (if used in harsh environments)
Voltage Range	2.375V – 3.46V	2.375V – 3.46V	Same
Cost	Lower cost	Higher cost	**MPU6050**
Availability	Widely available	Less common	**MPU6050**
Application Suitability	Drones, robotics, wearable electronics	Industrial & harsh environments	**MPU6050** (for drone use)

While the MPU6000 offers slightly better noise performance and industrial-grade ruggedness, the MPU6050 is more suitable for your drone project due to its simpler I²C interface**,** lower power consumption**,** cost-efficiency**,** and adequate performance for consumer-level and research-grade flight control systems.

3.3 Barometer



The BMP280 barometric pressure sensor was chosen for this drone system due to its high accuracy, low power consumption, and compact footprint, making it the most ideal for UAV applications where real-time altitude estimation is critical. Manufactured by Bosch Sensortec, the BMP280 is a digital pressure sensor that provides precise measurements of atmospheric pressure, which can be translated into altitude information using compensation algorithms. This sensor has the capability of delivering absolute pressure measurements with an accuracy of ±1 hPa and relative accuracy of ±0.12 hPa, enabling an altitude resolution of around ±1 meter. This level of precision is sufficient for most drone applications that require stable hover, altitude hold, and smooth vertical navigation. Furthermore, the sensor is highly energy-efficient, consuming just a few microamps in normal mode, which is advantageous for battery-powered systems like drones. Its small 2.0 × 2.5 mm package also allows easy placement on a compact PCB.

One of the key reasons for selecting the BMP280 was its flexibility in communication interfaces it supports both I2C and SPI. In this design, SPI mode is utilized to ensure a fast and dedicated data link to the microcontroller, reducing bus contention that can occur if multiple sensors share I2C. According to the schematic, the BMP280 is connected to the STM32F722 using SPI, with dedicated lines for chip select (CS_BARO), clock (SCK_BARO), data output (MISO_BARO), and data input (MOSI_BARO). The chip select pin (CSB) is controlled via a GPIO pin from the STM32F722, allowing the microcontroller to

enable or disable communication with the sensor as needed. This setup allows the sensor to operate at high data rates, by ensuring timely and accurate altitude data acquisition during flight.

The BMP280 also receives power from a regulated 3.3V supply, ensuring a stable voltage suitable for its operation. Proper power decoupling is also implemented, with a capacitor placed near the sensor's power line to minimize noise and voltage fluctuations. This is critical in drone environments where power lines may experience noise due to motor activity and switching regulators. Ground connections are also clearly routed to the main system ground to ensure consistent referencing and prevent ground loops.

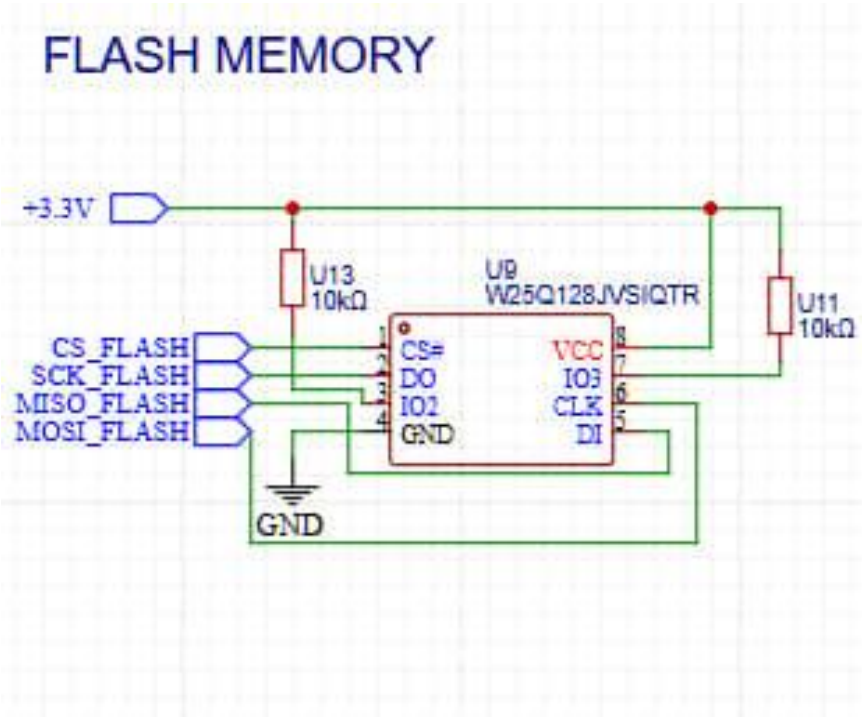
The barometer, specifically the BMP280 used in this drone's flight controller, plays a critical role in determining and maintaining the drone's altitude. It functions by measuring the atmospheric pressure, which changes with height pressure decreases as altitude increases. By comparing the real-time pressure to a reference value, usually recorded at the time of takeoff, the drone's flight controller can calculate its current altitude. This information is very essential for precise vertical positioning during flight.

One of the primary applications of the barometer is altitude estimation. Unlike GPS, which can be slow and inaccurate in vertical measurements, the barometer offers fast, responsive, and more accurate altitude data, particularly useful during low-altitude flight or indoor navigation where GPS may not be available. This altitude data of the barometer is used to enable altitude hold, a feature that allows the drone to maintain a fixed height automatically. When the pilot releases the throttle, the barometer helps the flight controller keep the drone hovering at a constant altitude by making real-time motor adjustments based on slight changes in pressure.

The barometer assists in ensuring smooth takeoffs and landings. By monitoring altitude continuously, the drone can ascend or descend gradually, allowing for more controlled and safe operations. During flight, especially under varying environmental conditions like wind or turbulence, the drone may experience unintended changes in height. The barometer detects even small fluctuations and helps the flight controller make corrections to stabilize vertical movement, maintaining a consistent flight path.

Furthermore, in autonomous missions, such as waypoint navigation or terrain-following tasks, accurate altitude information is vital. The barometer allows the drone to maintain specific heights above the ground, helping it avoid obstacles or follow terrain contours effectively. When used alongside other sensors like the IMU (MPU6050) and GPS, the barometer contributes to sensor fusion, combining different data sources to provide a more accurate and stable overall flight experience. While the IMU tracks orientation and acceleration, and the GPS provides horizontal location data, the barometer adds reliable vertical positioning, creating a comprehensive navigation system.

3.4 Flash Memory



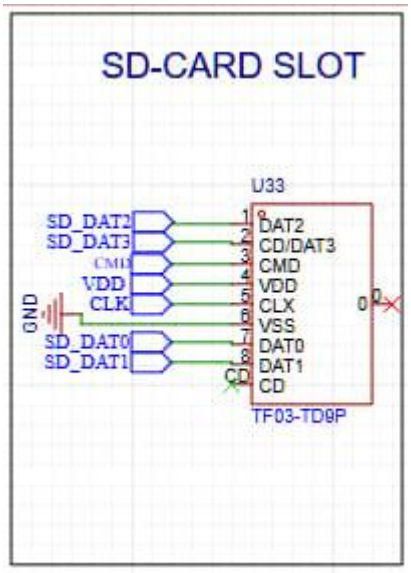
The W25Q128J flash memory was carefully selected for integration into the flight controller due to its high storage capacity, speed, and low power consumption attributes that are critical for drone applications such as blackbox data logging, configuration storage, and firmware updates. This chip offers a storage capacity of 128 Mbit (16MB)**,** making it the most ideal for recording large volumes of flight data such as sensor logs, GPS coordinates, and system diagnostics. The flash memory operates over a standard SPI interface that ensures compatibility with the STM32F722 microcontroller used in the design. Sensor supports SPI clock frequencies up to 80 MHz in standard mode, enabling fast and efficient read/write operations, crucial for real-time logging without introducing latency or data loss.

The flash memory is connected using the SPI protocol lines as follows: the CS (chip select) pin is connected to the `CS_FLASH` line, allowing the STM32 to initiate and manage communication; the DO (data output) pin is routed to `MISO_FLASH`, while the DI (data input) pin connects to `MOSI_FLASH`. The SCK (serial clock) pin is driven by the `SCK_FLASH` line from the microcontroller. These four primary SPI signals manage all data exchange between the microcontroller and the flash memory. Power is supplied through the VCC pin, which is connected to a 3.3V rail provided by the AMS1117 voltage regulator, and the GND pin is tied to the board's common ground. Pins IO2 and IO3 are unused in this implementation as the design employs only standard SPI mode, not the dual or quad I/O modes, which require additional wiring and more complex firmware handling.

According to the datasheet, the W25Q128J supports flexible erase operations such as sector erase (4KB)**,** block erase (32KB or 64KB)**,** and full chip erase. It is rated for 100,000 program/erase cycles**,** with 20+ years of data retention, making it highly reliable for long-

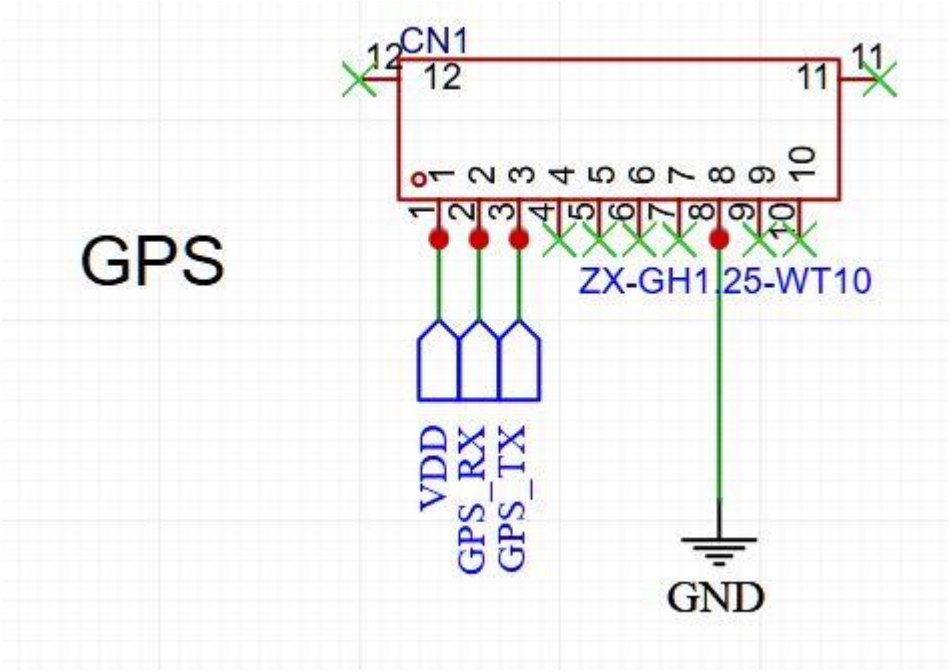
term use in mission-critical embedded systems. The memory is also optimized for power efficiency, supporting deep power-down and standby modes to conserve energy when not actively used this is particularly valuable in drones where power management is essential for flight duration.

The decision to use this flash memory was based on its reliability, performance, and compact footprint. It integrates smoothly with the SPI peripheral of the STM32F722, does not require complex circuitry, and meets the performance needs for continuous high-speed data logging. Its compatibility with the 3.3V logic level also simplifies power rail design and reduces the need for level shifters. In summary, the W25Q128J provides an ideal balance between capacity, speed, and system integration, making it a perfect choice for advanced flight controllers used in UAV systems.



Using both an SD card (via SDIO) and the W25Q128JVSIQ SPI Flash in your flight controller offers a powerful combination of high-capacity storage and low-latency, reliable data access, each complementing the other's limitations. The SD card provides gigabytes of storage, ideal for logging extensive flight data (e.g., sensor readings, GPS coordinates, or diagnostics) over long durations, thanks to its removable and scalable nature. However, SD cards have slower access times, higher power consumption, and are prone to wear from frequent writes. In contrast, the SPI Flash excels at fast, deterministic access for critical tasks like storing configuration parameters, firmware backups, or real-time control data, thanks to its low-latency SPI interface and 100,000+ write-cycle endurance. By offloading frequent, small writes (e.g., PID tuning values) to the SPI Flash and reserving the SD card for bulk logging, you optimize performance and longevity. Additionally, the SPI Flash ensures instant availability of boot-time settings, while the SD card handles post-flight analysis. This dual-storage approach balances speed, reliability, and capacity essential for robust flight control systems.

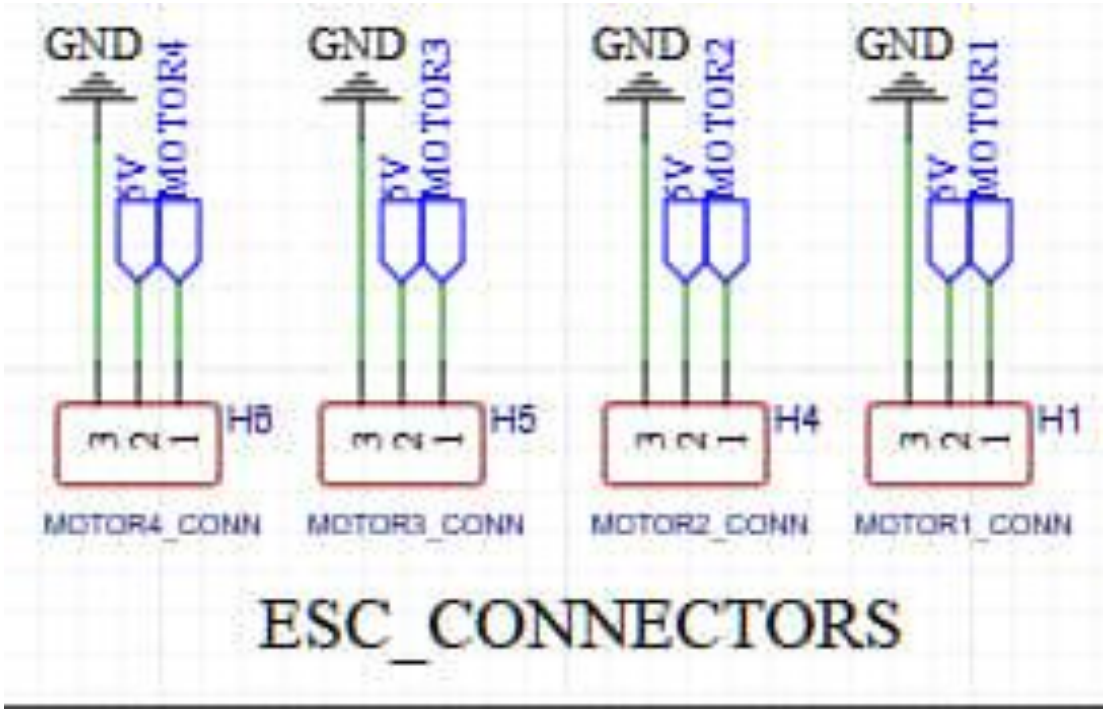
3.5 GPS



The GPS interface on the flight controller PCB is designed to enable reliable communication with an external GPS module through a UART connection. This interface consists of two key signal lines `GPS_TX` and `GPS_RX` which are connected to one of the UART peripherals of the STM32F722 microcontroller. The `GPS_RX` line is used to receive continuous data streams from the GPS module, typically in the form of standard NMEA sentences containing information such as latitude, longitude, altitude, number of satellites, and UTC time. This data is essential for real-time position tracking, especially during autonomous navigation or logging tasks. The `GPS_TX` line, while not always used, allows the microcontroller to send configuration commands back to the GPS module, such as setting the baud rate or enabling specific types of data output.

Physically, these UART lines are routed to a dedicated connector labeled ZX-GH1.25-WT10GPS, which facilitates easy attachment of commonly used GPS modules like those from the u-blox NEO-M8 series. The schematic also includes protective components such as ESD diodes to safeguard the communication lines from voltage spikes or electrostatic discharge, enhancing the reliability of the interface in real-world conditions. Additionally, nearby filtering capacitors help suppress electrical noise, ensuring stable signal integrity even in the high-interference environment typical of drones. This UART-based GPS integration provides a flexible and standardized solution for incorporating global positioning functionality into the flight controller, forming a foundational element in any GPS-aided or autonomous drone navigation system.

3.6 ESC_CONNECTORS



The ESC_CONNECTORS section of the schematic is a crucial part of the flight controller, serving as the interface between the microcontroller (STM32F722) and the drone's Electronic Speed Controllers (ESCs), which in turn drive the brushless DC motors. This design supports four motors labeled MOTOR1 through MOTOR4 each connected through a dedicated 3-pin header (H3 to H6) as in the schematic. Each connector provides three essential connections: a PWM signal line from the MCU, a VDD line for power, and a GND line for grounding. The PWM (Pulse Width Modulation) signal is generated by the STM32F722's hardware timers, allowing precise motor speed control based on control algorithms. These signals may also be configured to use DShot, a digital motor control protocol that offers improved resolution and error detection compared to traditional PWM.

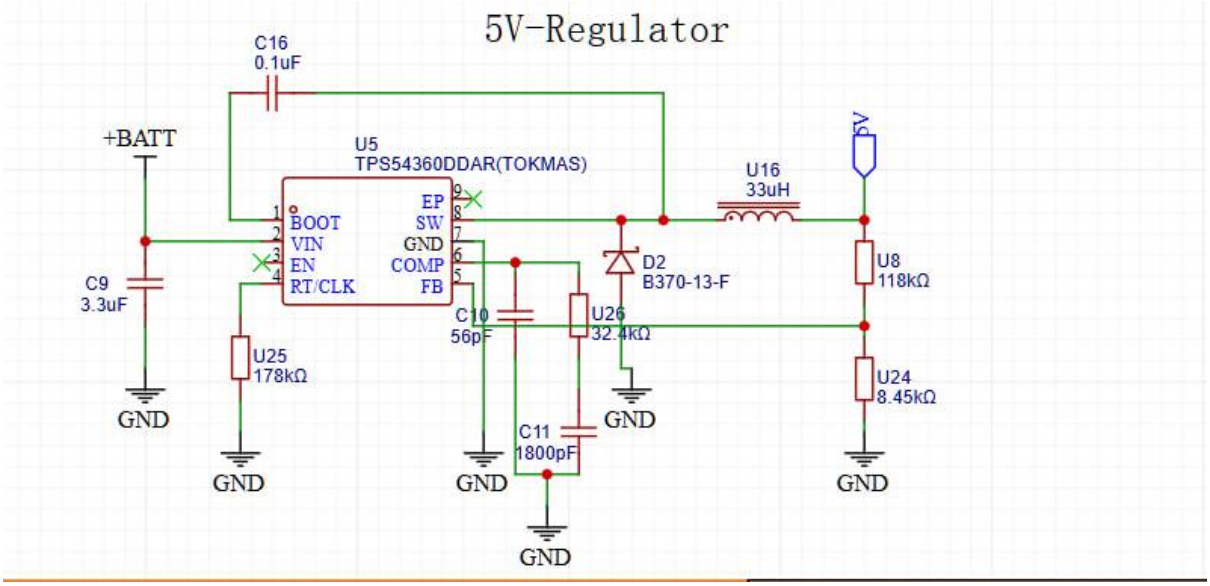
The VDD line on each connector supplies 5V power, typically drawn from the output of the TPS54360 buck converter. This voltage is used primarily to power the logic circuitry within the ESCs, especially in opto-isolated types that do not draw power from the battery directly. The GND pin ensures a shared electrical reference between the microcontroller and ESCs, which is critical for maintaining signal integrity and preventing unstable motor behavior. The separation of each motor connection into individual headers provides modularity, allowing for easier assembly, replacement, and troubleshooting of the propulsion system.

From a design perspective, these connectors are directly mapped to the PWM-capable GPIO pins of the STM32, likely driven by advanced control timers (such as TIM1 or TIM8), which are well-suited for high-frequency and high-resolution PWM generation. The connectors form the final actuation link in the drone's control loop, translating processed sensor data and navigation commands into real-time motor movements. The layout also takes into

consideration EMI (Electromagnetic Interference) concerns by maintaining clean signal paths and ensuring proper power filtering. Overall, the ESC_CONNECTORS module plays a vital role in ensuring smooth, stable, and responsive flight performance by reliably interfacing the microcontroller with the propulsion system.

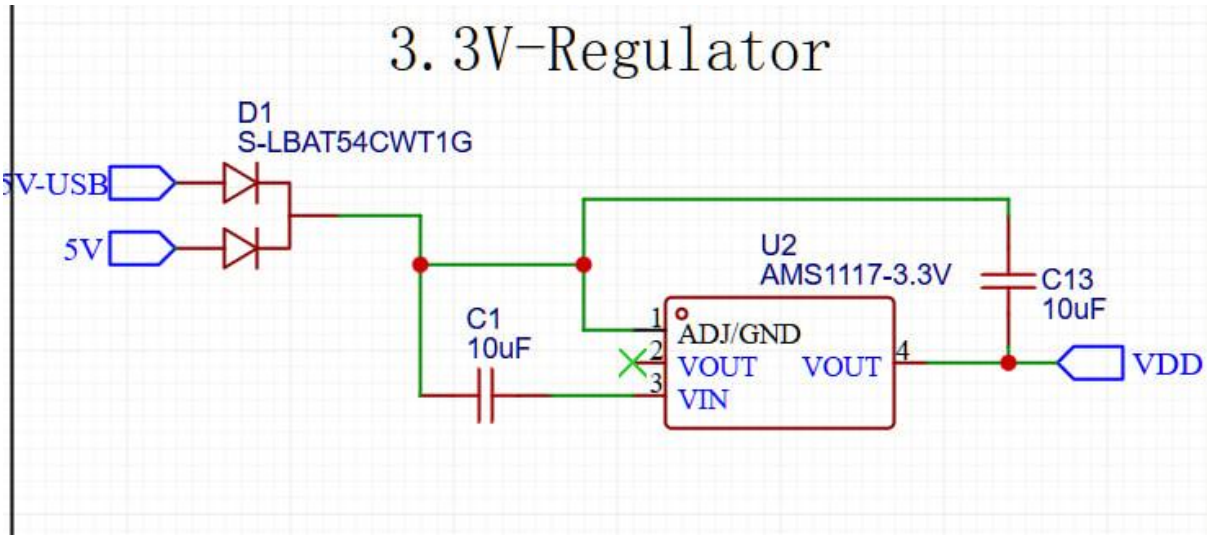
4. Power and Actuation Layout

The power and actuation system of the flight controller has been designed to ensure efficient voltage regulation, stable power delivery, and reliable motor control for drone operations. The primary power input is supplied through the +VBAT line, which is typically connected to a 2-cell (7.4V) to 4-cell (14.8V) LiPo battery. This wide voltage input range supports various drone configurations depending on size and payload requirements. To convert this variable battery voltage into stable usable levels, the board incorporates a two-stage voltage regulation system. The first stage uses a high-efficiency TPS54360DDAR buck converter to step down the battery voltage to a regulated 5V rail. This converter is capable of delivering up to 3A of current and is accompanied by filtering components such as a 33µH inductor and multiple capacitors to smooth out the voltage and reduce ripple.



The 5V regulator in the flight controller is built around the TPS54360DDAR, a high-efficiency buck (step-down) converter designed to convert the higher battery voltage from the +VBAT line (ranging from 7.4V to 14.8V) into a stable 5V output. This regulator is critical for powering the mid-voltage components on the board, such as the ESC control logic, USB interface, microSD card, and also serves as the input to the 3.3V regulator. The TPS54360 features an integrated high-side MOSFET and can supply up to 3A of continuous current, making it suitable for drone applications where consistent power delivery is essential. The design includes key external components such as an inductor (33µH**)** and multiple capacitors

to filter the switching noise and maintain a clean DC output. A feedback network consisting of resistors ensures that the output voltage is regulated precisely at 5V. Additionally, the TPS54360 includes built-in protections such as overcurrent, thermal shutdown, and undervoltage lockout**,** which enhance the reliability and safety of the power system. Its switching nature makes it much more efficient than linear regulators, especially when stepping down from higher voltages like those in LiPo batteries. This ensures that less power is wasted as heat, which is vital in compact embedded systems like drones where heat dissipation and battery life are both concerns.



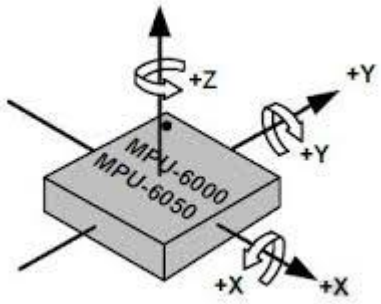
The 3.3V regulator used in the flight controller is an AMS1117-3.3V, a widely used low dropout (LDO) voltage regulator. Its primary role is to convert the 5V rail generated by the main buck converter (TPS54360) down to a stable 3.3V output, which is required by most of the low-power digital components on the board. This includes the STM32F722 microcontroller**,** MPU6050 IMU sensor**,** BMP280 barometric sensor**,** and the W25Q128J SPI flash memory. As an LDO regulator, the AMS1117 is preferred in this context for its simplicity and low noise characteristics, which are beneficial when powering sensitive analog and digital circuits. It can supply up to 1A of current, making it suitable for small embedded systems. The schematic shows the typical configuration with an input capacitor and an output capacitor to improve transient response and ensure voltage stability. This 3.3V line acts as the core power supply for all logic-level operations and communication interfaces like I2C and SPI. Although LDOs are less efficient than switching regulators, their clean output and compact footprint make them ideal for regulating down from 5V in low- to moderate-power applications such as flight controllers.

Motor control and actuation are handled through four ESC connectors labeled MOTOR1 to MOTOR4, each consisting of three pins: a PWM signal line (or DShot), VDD, and GND. These connectors link directly to the STM32's motor output pins and provide a clear path for controlling the Electronic Speed Controllers, which in turn drive the brushless motors. The inclusion of both traditional PWM and optional digital DShot support allows for precise motor speed control, enabling responsive and stable drone maneuvering. Each motor

connection is routed through a dedicated 3-pin header to allow modular and organized wiring, which is essential for physical stability and ease of maintenance.

The power system is further safeguarded by protection mechanisms. ESD protection diodes are implemented near the USB interface to prevent voltage spikes from damaging components. Reverse polarity protection is provided using diodes such as the B370-13-F, ensuring that incorrect battery connections do not harm the circuitry. Additionally, ferrite beads and low-pass filter capacitors are used across power rails to suppress electromagnetic interference and ensure noise-free power delivery. Together, this power and actuation layout ensures that the drone operates efficiently, reliably, and safely under varying flight conditions.

5. Orientation Estimation Logic



The MPU6050 is a 6-axis Inertial Measurement Unit (IMU) that integrates a 3-axis accelerometer and a 3-axis gyroscope, making it a crucial component for estimating the orientation of a drone during flight. In this flight controller system, the MPU6050 is interfaced via the I2C1 peripheral, specifically connected through pins PB6 (SCL) and PB7 (SDA) as shown in the CubeMX pin configuration. The primary role of this sensor is to provide real-time data about the drone's angular velocity and linear acceleration along the X, Y, and Z axes. The implementation in the code involves initializing the MPU6050 and continuously reading 14 bytes of data from the sensor, which includes acceleration and gyroscopic values. These raw values are then converted into physical units' acceleration into meters per second squared (m/s^2) and gyroscope readings into radians per second (rad/s). The accelerometer helps determine the tilt of the drone (roll and pitch angles), while the gyroscope captures the rate of rotation, offering short-term angular motion data. Although the code does not explicitly implement a full sensor fusion algorithm like a complementary or Kalman filter, the system is structured in a way that allows combining accelerometer data with gyroscope data .This combination provides a more reliable estimation of the drone's orientation. These orientation estimates are essential inputs to the PID controllers for roll, pitch, and yaw stabilization, ultimately allowing the drone to maintain a balanced and stable flight path.

$$\text{Angle estimate} = \alpha \cdot (\text{angle\_estimate} + \text{gyro\_rate} \cdot dt) + (1 - \alpha) \cdot \text{accel\_angle}$$

Equation represents a complementary filter, which is a common and effective method used in embedded systems like flight controllers for estimating orientation. This approach combines the strengths of both the gyroscope and the accelerometer to produce a stable and responsive angle estimate. The gyroscope provides smooth and fast short-term angular velocity data, which is integrated over time to estimate changes in orientation. However, gyroscopes tend to drift over time, making them unreliable for long-term accuracy. On the other hand, the accelerometer offers a more stable and absolute measure of the tilt angle (accel_angle) by detecting the direction of gravity, but it can be noisy and affected by vibrations and dynamic movements. The complementary filter balances these two data sources using the parameter alpha, where a high value of alpha (typically around 0.95 to 0.98) means more trust is placed on the gyro for responsiveness, while a lower value increases reliance on the accelerometer

for stability. In the context of your drone project, although the exact equation may not appear directly in the code, the logic follows this principle by using both accelerometer and gyroscope readings from the MPU6050 to maintain accurate and stable orientation estimation for roll and pitch angles. This estimated orientation is then fed into the PID control loop to help the drone stay balanced and responsive during flight.

The BMP280 barometric pressure sensor plays a crucial role in estimating the drone's altitude, which is an integral part of the overall orientation and stability control system. Although it does not measure angular orientation like the IMU, it contributes directly to vertical positioning essential for maintaining a consistent flight height and enabling altitude hold features. The BMP280 is connected to the STM32F722RET6 via the I2C1 interface, using the same bus as the MPU6050 IMU (typically on pins PB6 for SCL and PB7 for SDA as shown in the CubeMX configuration). During system initialization in main.c, the function BMP280_Init() is called to initialize the sensor through the HAL I2C interface. Once initialized, the sensor provides atmospheric pressure and temperature data, which is processed to calculate the drone's altitude. This conversion typically uses the barometric formula:

$$\text{altitude} = 44330 \times \left(1 - \left(\frac{P}{P_0}\right)^{0.1903}\right)$$

Where P is the current measured pressure and P0 is the standard sea-level pressure. This calculated altitude is then passed into the flight control system, particularly the altitude PID controller (alt_pid), which adjusts the motor outputs to stabilize and maintain the drone at the desired height. While simple in design, this barometric method provides effective altitude estimation for indoor and outdoor environments, though it may be sensitive to sudden pressure changes or turbulence. When integrated with the IMU data, it enables the drone to maintain a stable position not only in the horizontal plane but also along the vertical axis, which is essential for controlled and safe flight.

6. PID or chosen control strategy

The control strategy employed in this STM32F722RET6-based drone flight controller is built around the widely used PID (Proportional-Integral-Derivative) algorithm. This control mechanism is essential for stabilizing and maneuvering the drone across its primary axes—roll, pitch, yaw, and altitude. Each of these axes is governed by an individual PID controller, namely roll_pid, pitch_pid, yaw_pid, and `alt_pid`, all of which are declared and initialized within the main.c file. These controllers use feedback data obtained from the MPU6050 inertial measurement unit (IMU) connected via the I2C1 interface to estimate the drone's current orientation in real time.

The core principle of the PID algorithm is to minimize the error between a desired setpoint and the current measured value. This is achieved by calculating a corrective output based on three components: the proportional (P), integral (I), and derivative (D) terms. The general form of the PID control equation is:

$$\text{Output}(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{de(t)}{dt}$$

Where:

- e(t) is the error at time (difference between desired and actual values),
- K_p is the proportional gain, which determines the strength of the correction based on the current error,
- K_i is the integral gain, which addresses the accumulation of past errors to eliminate steady-state offset,
- K_d is the derivative gain, which predicts future error trends based on the rate of change and adds a damping effect.

In the implemented code, this formula is translated into the `PID_Update ()` function, which takes the current error and time delta as inputs. The proportional component is calculated by directly multiplying the error with K_p. The integral component is updated each cycle by accumulating the product of the error and the elapsed time (`dt`). To prevent the integral term from growing excessively the code includes output limiting using predefined `output_min` and `output_max` thresholds. The derivative term is derived from the rate of change of the error, computed as the difference between the current and previous error divided by `dt`. The final control output is then the sum of all three components and is also bounded within the min/max range to ensure motor signal safety.

This output is then used to modulate the PWM signals sent to the motors using functions such as `PWM_SetMotorPulse()`. For example, if the drone is tilting too far to the left (negative roll error), the PID controller for roll will increase the motor speed on the left side and decrease it on the right side to restore balance. This process is applied independently to all three rotational axes and to the vertical axis for altitude control, ensuring stable and responsive flight.

This architecture allows each axis to be controlled individually with its own tuned gains, offering a flexible and modular approach to flight stability. The simplicity and computational efficiency of the PID algorithm make it particularly well-suited for real-time embedded systems like the STM32F7, which may not always include floating-point units. While basic in form, this control approach can be expanded in the future with adaptive PID tuning, sensor fusion techniques (like Kalman filtering), or additional flight modes (such as Loiter or Position Hold) as needed. Thus, the PID-based control strategy forms the foundation of a reliable and extensible flight control system.

The drone's stabilization and navigation are achieved through a modular PID (Proportional-Integral-Derivative) control strategy, which forms the core of the control algorithm implemented in the firmware. Each flight axis roll**,** pitch**,** yaw**,** and altitude is regulated using a separate instance of the `PID_Controller` structure, as declared in the `main.c` file:

PID_Controller roll_pid, pitch_pid, yaw_pid, alt_pid;

This structure, defined in `pid.h`, holds the PID gains (`kp`, `ki`, `kd`), the error integral term, previous error value, and boundaries for output saturation to ensure control stability. During initialization, each controller is configured with tailored gains using the `PID_Init()` function. For example, in `main.c`, the roll axis controller is initialized as follows:

```
PID_Init(&alt_pid, 1.2f, 0.1f, 0.5f, -300, 300);
```

Here, 1.2 is the proportional gain, 0.5 the integral gain, and 0.01 the derivative gain, while the minimum and maximum bounds limit the controller output to ±500 to avoid excessive motor commands.

The real-time control logic is executed within the PID_Update () function in pid.c, which calculates the output based on the error between the desired setpoint and the current measured value from the IMU (in degrees or radians). The function updates the integral term with respect to the elapsed time (`dt`), calculates the derivative using the difference between the current and previous error, and computes the control output using the PID formula:

```
float output = pid->kp * error + pid->ki * pid->integral + pid->kd * derivative;
```

This output is then clamped to ensure it stays within the safe range for motor pulse-width modulation (PWM). Additionally, to prevent integral wind-up a common issue in control systems the code limits the integral term to the same output bounds before applying it. This is

critical for maintaining long-term stability and avoiding oscillations or overshoot in attitude corrections.

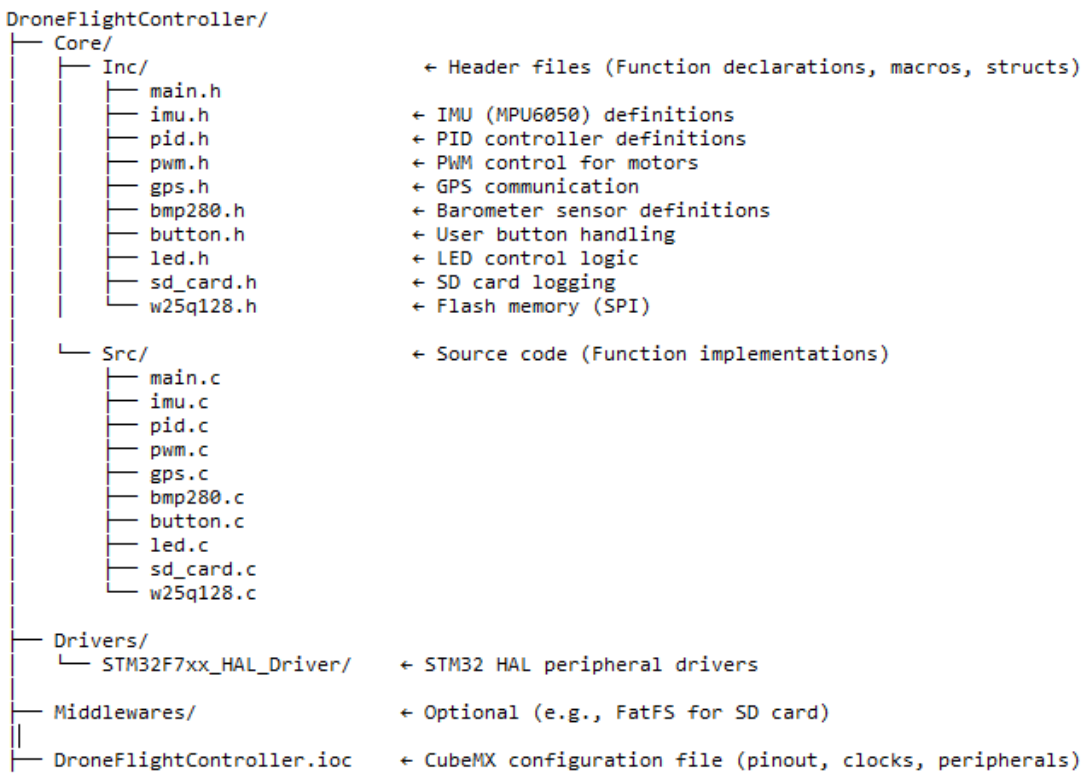
The PID outputs for each axis are then combined in the motor mixing logic within the main control loop. For example, the pulse signal to each motor is calculated by summing the throttle value with the respective PID corrections:

```
// Motor mixing with altitude control
motor_pulse[0] = SAFE_MOTOR_PULSE_MIN + roll_output - pitch_output + yaw_output + alt_output;
```

This kind of differential control allows the drone to adjust orientation in all directions simultaneously by speeding up or slowing down individual motors appropriately.

The design of this PID-based strategy demonstrates several key strengths. First, the code is written in a modular fashion, separating sensor data acquisition, control computation, and actuator output into cleanly defined functions and files (imu.c, pid.c, pwm.c). This modularity improves readability and maintainability. Second, the use of floating-point arithmetic ensures smooth and precise control, which is vital in aerial systems where minute adjustments can lead to major differences in motion. Additionally, anti-windup clamping and output limits reflect good control engineering practices, ensuring the controller remains stable even under varying conditions. Overall, the PID implementation is both robust and adaptable, providing a solid foundation for further enhancements such as adaptive tuning, sensor fusion, or advanced flight modes.

7. Firmware Structure



The code base layout of the STM32F722RET6-based drone flight controller is systematically organized to ensure modularity, clarity, and ease of maintenance. At the heart of the firmware structure lies the Core directory, which is subdivided into two primary folders: Inc and Src. The Core/Inc/ folder houses all the header files (.h) responsible for declaring function prototypes, defining data structures, macros, and shared constants. These headers act as the interface between different code modules and ensure structured communication across the project. For instance, imu.h defines the functions and structures required for IMU interactions, pid.h declares the data structure and method for the PID controller, and pwm.h provides the interface for PWM motor control. Similarly, other headers like gps.h, led.h, and button.h provide declarations for their respective modules, while main.h aggregates global variables and common declarations shared across files.

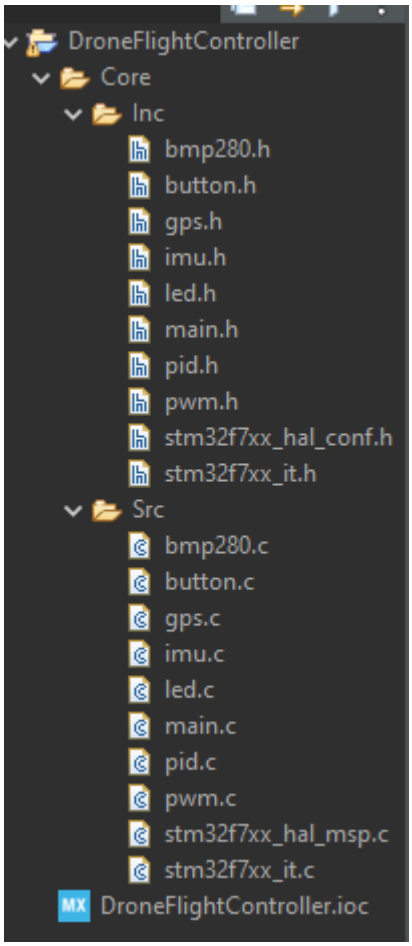
The implementation of these functionalities is carried out in the Core/Src/ folder, which contains the corresponding C source files (.c). Each .c file is matched with a header file and contains the logic needed for specific modules such as sensor data acquisition, control algorithms, peripheral communication, and real-time feedback. For example, imu.c implements I2C communication routines to read raw accelerometer and gyroscope data from

the MPU6050 sensor, while pid.c contains the control logic that computes stabilization outputs using PID algorithms. The pwm.c file adjusts motor speeds by writing PWM signals to the motor channels, and gps.c parses GPS data received via UART. Meanwhile, main.c serves as the entry point of the firmware, responsible for system initialization, peripheral configuration, and execution of the primary control loop that coordinates sensor readings, control calculations, and actuator updates. Supporting modules such as led.c and button.c handle user interface feedback and manual inputs respectively.

In addition to these core source and header files, several configuration and metadata files support the project's infrastructure. The .ioc file, generated using STM32CubeMX, encapsulates the complete configuration of the microcontroller’s peripherals, clock settings, and pin mappings. Changes made in CubeMX are reflected automatically in the generated initialization code. Other project-specific files such as .project, .cproject, and .mxproject are used by STM32CubeIDE to maintain build settings, project structure, and metadata. The firmware also includes system-level files which contains the interrupt vector table and startup routines written in assembly, which initializes the system clocks and core settings.

This modular and well-structured layout follows best practices in embedded system design. By keeping functionality in separate files and adhering to clear interfaces, the firmware becomes highly scalable new features like additional sensors or communication protocols can be integrated with minimal interference to the existing structure. It also improves maintainability, as developers can focus on individual components without risking unintended effects on unrelated parts of the code. Overall, this structured codebase design enables efficient development, testing, and future expansion of the drone flight controller firmware.

7.2 Key Functions and Modules



The firmware of the STM32F722RET6-based drone flight controller is structured in a modular and efficient manner, allowing for clarity, maintainability, and scalability. Each major functionality of the drone is encapsulated within dedicated source and header files, which collectively handle the core flight operations. At the heart of the system lies the main.c file, which serves as the central execution unit. It is responsible for initializing the hardware abstraction layer (HAL), setting up the system clock, and calling initialization routines for all peripheral modules such as GPIO, I2C, UART, SPI, and timers. Once initialization is complete, main.c enters an infinite control loop, where it sequentially reads sensor data, processes PID control algorithms, updates motor speeds through PWM signals, and monitors user inputs or system status. This polling-based loop enables deterministic execution suitable for bare-metal control without relying on an operating system.

One of the key modules is the IMU interface, implemented in imu.c and imu.h, which manages communication with the MPU6050 sensor via the I2C1 interface. It configures the sensor, reads raw accelerometer and gyroscope data, and converts them into usable physical quantities. These real-time orientation values are essential for maintaining stable flight. Complementing this is the PID controller module (pid.c and pid.h), which implements a standard PID control formula that calculates the required output to correct orientation and altitude errors. It takes into account proportional, integral, and derivative terms, and includes logic to prevent integral windup. The outputs from the PID module are sent to the PWM module, defined in pwm.c and pwm.h, which utilizes Timer 1 (TIM1) to generate precise

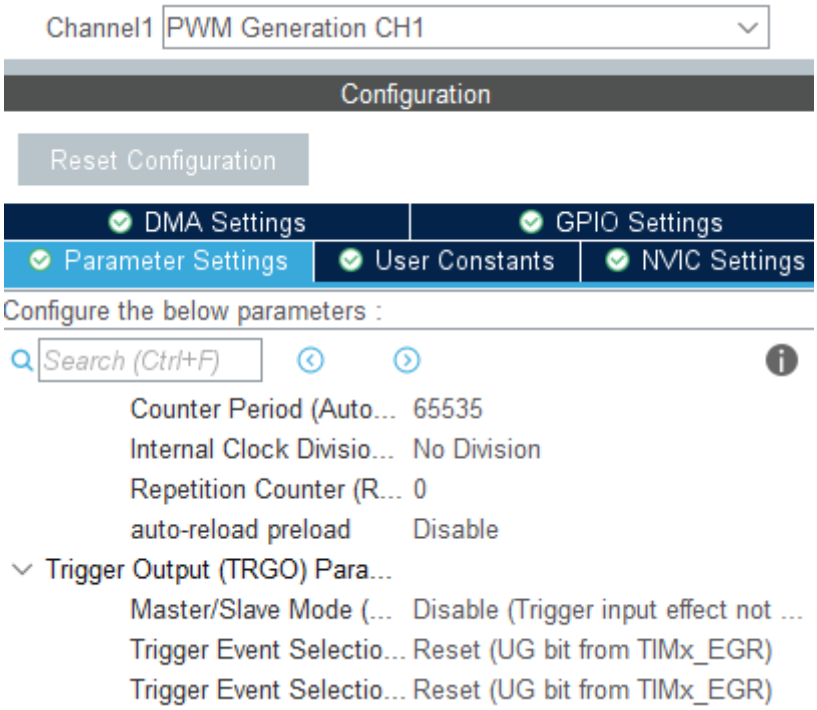
PWM signals on motor control pins. These PWM signals directly control the speed of each motor via Electronic Speed Controllers (ESCs), allowing the drone to respond to PID corrections in real-time.

Navigation and telemetry are supported by the GPS module, implemented through gps.c and gps.h, which communicates with a GPS receiver over USART6. This module reads NMEA data, extracts latitude and longitude, and provides position feedback that may be used for higher-level flight control or mission planning. Altitude estimation is handled by the bmp280.c and bmp280.h module, which interfaces with a BMP280 barometric pressure sensor over I2C1. This module calculates altitude based on pressure variations and supplies altitude data to the PID controller responsible for maintaining vertical stability.

User interaction is facilitated by the LED and button modules. The LED module uses Timer 4 (TIM4) to control status LEDs through PWM, signaling different states such as armed, disarmed, or error. The button module reads inputs via GPIO and allows the user to toggle flight modes or initiate specific commands. For data storage, two modules are employed: the SD card module (sd_card.c) and the Flash memory module (w25q128.c). The SD card interface operates via SDIO and is primarily used for flight logging or configuration file storage, while the Flash memory module communicates over SPI1 and can store persistent parameters such as calibration data or saved missions.

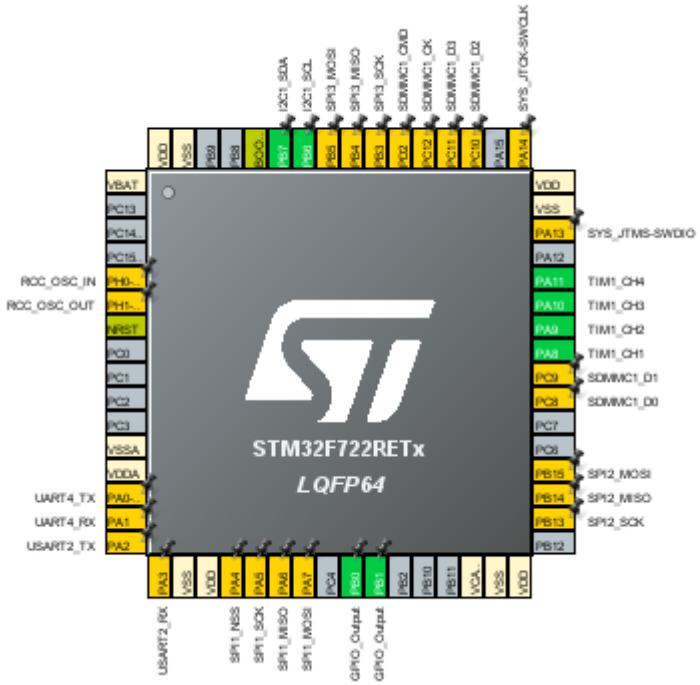
All modules are designed to work independently yet communicate through clearly defined interfaces, enabling a smooth flow of information throughout the system. Sensor modules feed real-time data into the control logic, which then uses this data to compute necessary corrections. The motor control module applies these corrections, while user interface and storage modules operate in parallel to enhance system reliability and usability. This modular and tightly integrated firmware structure ensures that the drone can perform stable, responsive, and efficient flight operations, while remaining open to future upgrades or additional features.

7.3 Timing & Scheduling



The firmware of the STM32F722RET6-based drone flight controller manages timing and scheduling using a straightforward polling-based main loop without employing a real-time operating system (RTOS). Within the infinite loop in the main.c file, the system sequentially performs critical tasks such as reading sensor data from the IMU, barometer, and GPS; computing PID control outputs for stabilizing roll, pitch, yaw, and altitude; updating motor PWM signals accordingly; and handling data logging and user interface elements like buttons and LEDs. Precise timing is achieved through the use of hardware timers—TIM1 generates accurate PWM signals for motor control, while TIM4 modulates LED brightness. Additionally, the SysTick timer provides millisecond-level system ticks that facilitate delay management and time tracking through HAL library functions like HAL_Delay() and HAL_GetTick(). Interrupts are employed for time-sensitive peripheral communications such as I2C data exchange with sensors and USART reception from the GPS module, allowing the system to respond promptly to incoming data without blocking the main control loop. The design has multitasking capabilities of an RTOS, the cooperative scheduling model is well-suited for the flight controller's predictable task timing and real-time responsiveness requirements, ensuring stable flight performance within the embedded system constraints.

7.4 Pin Configuration and Peripheral Mapping



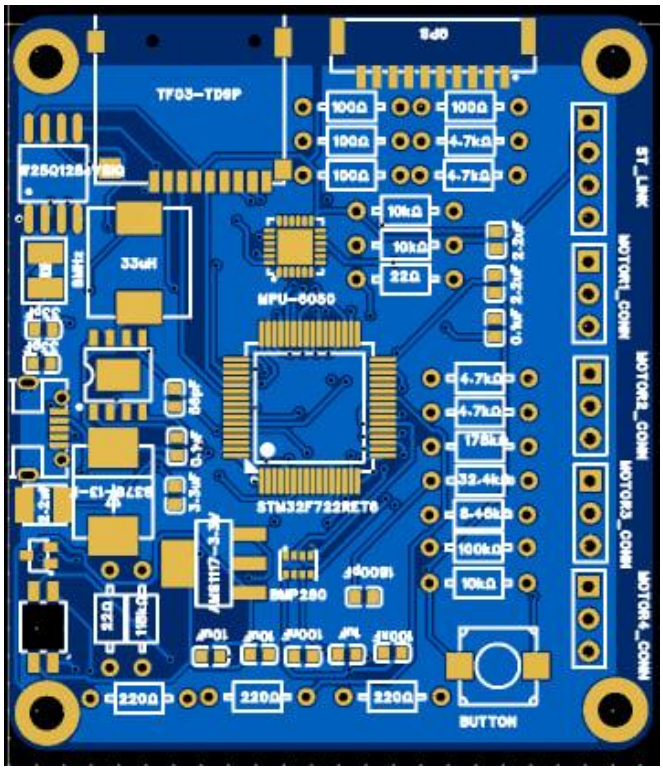
The STM32F722RET6 microcontroller's pin configuration for the drone flight controller is set up using STM32CubeMX, which facilitates assigning specific pins to hardware peripherals based on their function and the project's requirements. For sensor communication, the I2C1 interface is mapped to pins PB6 and PB7, serving as the clock (SCL) and data (SDA) lines respectively. This bus connects key sensors like the MPU6050 IMU and the BMP280 barometer, enabling reliable, low-overhead communication. The GPS module communicates over USART6, with PC6 and PC7 assigned as the transmit (TX) and receive (RX) pins. This UART setup allows the microcontroller to receive positioning data in NMEA format from the GPS device.

For external flash memory storage, SPI1 is utilized with pins PA5 (clock), PA6 (MISO), and PA7 (MOSI), enabling high-speed synchronous data transfer to the W25Q128 flash chip, which supports data logging and configuration storage. The motor speed control relies on the TIM1 peripheral configured to generate PWM signals on pins PA8 through PA11, corresponding to channels 1 to 4. These PWM signals are used to control the four brushless motors via their electronic speed controllers (ESCs), allowing precise regulation of thrust. Similarly, TIM4 uses pins PD12 to PD15 to manage onboard status LEDs, employing PWM to adjust brightness or create signaling patterns such as blinking or fading, providing important visual feedback about the system status.

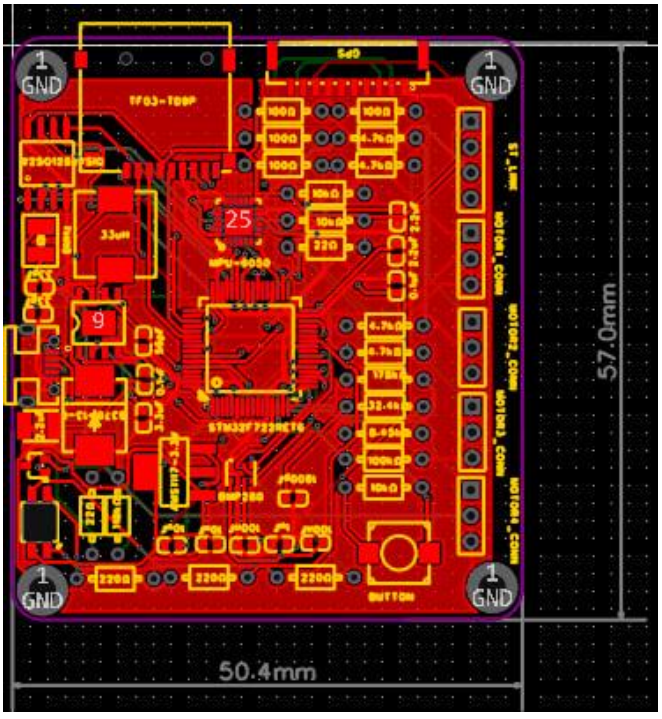
Additionally, general-purpose input/output pins PA0 to PA3 are configured for auxiliary inputs like buttons or switches, enabling user interaction such as arming or mode switching. This careful assignment of pins leverages the STM32's hardware features, maximizing efficiency and signal integrity. By using dedicated peripheral pins for each function, the firmware benefits from reliable timing, reduced CPU load, and modular expandability for future sensor or interface additions. Moreover, adhering to standard pin mappings simplifies development with STM32 HAL drivers and CubeMX-generated code, facilitating easier debugging and maintenance.

Peripheral	Pins	Function
I2C1	PB6 (SCL), PB7 (SDA)	IMU, Barometer communication
USART6	PC6 (TX), PC7 (RX)	GPS communication
SPI1	PA5 (SCK), PA6 (MISO), PA7 (MOSI)	Flash memory access
TIM1	PA8, PA9, PA10, PA11	PWM for motor ESCs
TIM4	PD12, PD13, PD14, PD15	LED control (PWM)
GPIO	PA0, PA1, PA2, PA3	Buttons, analog inputs

8. Flight controller



2D - PCB



Flight controller-Schematic

The PCB layout of the flight controller is designed using a four-layer stack-up that ensures high performance, efficient power distribution, and minimal electromagnetic interference (EMI). The four layers are thoughtfully designated with the top layer handling component placement and high-speed signal routing, inner layer 1 allocated for power distribution, inner layer 2 for signal routing, and the bottom layer serving as a solid ground plane. This structured approach ensures clear separation between power, signal, and ground domains, leading to improved signal integrity and system reliability.

On the top layer, all surface-mounted components are carefully positioned to minimize trace lengths and interference. The STM32F722 microcontroller sits at the core, with closely routed connections to essential peripherals like the MPU6050 (I^2C), BMP280 (SPI), flash memory, and the SD card interface. PWM outputs to ESCs are also routed directly from the MCU to the corresponding connectors (MOTOR1–MOTOR4), avoiding unnecessary vias. The UART lines for GPS (GPS_TX, GPS_RX) and SWD debug lines are clearly separated from noisy power sections to ensure clean data transfer. This layer prioritizes signal integrity by minimizing crossings and keeping differential pairs like USB closely matched.

The inner layer 1 functions as the dedicated power plane, distributing +VBAT, +5V, and +3.3V across the board using wide copper pours to reduce voltage drop and resistance. The power traces are routed using a star topology wherever possible to isolate sensitive sections from high-power transients. Local decoupling capacitors connect this power layer to the ground plane through via stitching, which enhances filtering and stabilizes the voltage rails for all major ICs including the STM32, flash, sensors, and the SD card.

The inner layer 2 is used exclusively for signal routing, providing a clean and interference-free path for communication lines like SPI, SDIO, and GPIOs. By separating this from the power layer, the design ensures that signal lines do not pick up noise from power transients. All high-speed or timing-sensitive signals are routed with attention to impedance control, and return paths are carefully maintained by stitching the signal traces with ground vias connecting to the bottom layer.

Finally, the bottom layer serves as a continuous ground plane, which is a best practice in multilayer PCB design. It acts as a shield for all signals above it and provides low-impedance return paths for signal currents. This improves overall EMI performance and reduces the risk of ground loops. The ground plane also contributes to better thermal dissipation, particularly for power-intensive components like the TPS54360 buck regulator and AMS1117 voltage regulator. Throughout the board, stitching vias connect the top and internal ground points to the bottom layer, ensuring a consistent and robust grounding system.

Overall, the routing is clean, efficient, and follows professional standards. High-current paths are wide and direct, sensitive analog and digital lines are isolated, and the board is well-balanced both electrically and thermally. The use of four layers allows optimal space utilization and clean signal integrity, making the board robust for high-reliability applications like autonomous drone flight control. The design’s modularity, ease of debugging via dedicated connectors (GPS, USB, ST-Link), and adherence to EMC guidelines make this PCB layout both technically sound and practically effective.

8. Limitations and Improvements

The current flight controller implementation has several limitations that affect its performance and reliability. The sensor fusion system relies on a complementary filter for attitude estimation, which is less accurate than more sophisticated approaches like Kalman filters. This can lead to noticeable drift in angle estimation during aggressive maneuvers, potentially causing stability issues. While a Madgwick filter has been implemented as an improvement, representing a temporary simplification in the system's design.

The GPS navigation system currently employs a basic return-to-home function that only considers bearing and distance without incorporating proper path planning algorithms. This simplified approach may result in inefficient flight paths and potential overshooting of the home position. Additionally, the system lacks obstacle avoidance capabilities and terrain following functionality, which limits its operational safety in complex environments. These simplifications were made to accelerate initial development but represent areas needing improvement for more robust autonomous operation.

Several significant enhancements could substantially improve the flight controller's capabilities. Implementing a proper Kalman filter for sensor fusion would enable more accurate state estimation by optimally combining IMU data with GPS for position estimation and barometer readings for altitude hold. This would be particularly beneficial during aggressive maneuvers where current estimation methods show limitations.

The navigation system could be substantially enhanced through several key developments. Implementing waypoint navigation with spline-based paths would enable smoother, more efficient routes between points. Adding obstacle avoidance using distance sensors and terrain following capabilities through combined barometer and rangefinder data would greatly improve operational safety and enable true autonomous mission capability. These upgrades would transform the system from simple position holding to comprehensive autonomous navigation.

The control system could be made more robust through adaptive algorithms. Implementing gain scheduling based on battery voltage would maintain consistent performance as power levels change. Online PID tuning methods like Ziegler-Nichols or relay auto-tuning could automatically optimize controller parameters during flight. More advanced approaches like model predictive control could further improve performance, especially for aggressive maneuvers. These improvements would help maintain consistent flight characteristics across varying conditions and payload configurations.

9. Appendix

1. **GitHub Repository** (Source codes + Schematics)

   https://github.com/JinalkaHerath/Flight_Controller-STM32.git