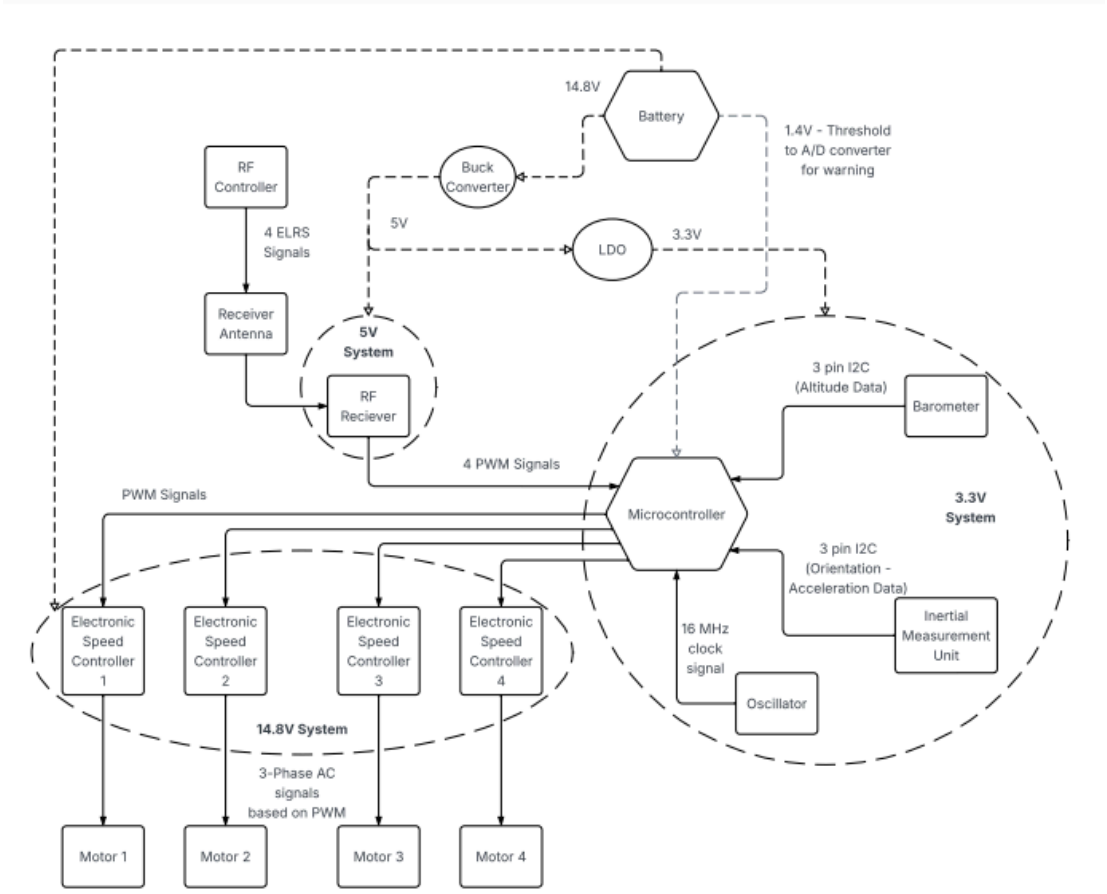
Connor Prior **Email: cprior3@horizon.csueastbay.edu (Team Leader)**
Nahom Solomon **Email: nsolomon@horizon.csueastbay.edu**
Christopher Ruiz-Guerra **Email: cruizguerra@horizon.csueastbay.edu**
Seth Iris Canonigo **Email: scanonigo@horizon.csueastbay.edu**

Modular Aerial Realtime Instrument (M.A.R.I)



Introduction:

Our Senior Design Project is a quadcopter drone that is controlled by a radio controller that will be able to fly and perform basic aerial movements. A radio controller will send four radio signals to our receiver on our drone, which will send PWM signals to our microcontroller.

These signals will control the throttle (upward/downward movement), yaw (rotation), pitch (forward/backward movement), and roll (side to side movement). The microcontroller will alter these signals to keep the drone level with the help of an IMU(Inertial Measurement Unit) and a barometer, which will detect the orientation of the drone so it can correct its movement to stay level in the air, and also stop its vertical movement when it reaches a certain height. Once the microcontroller computes the correct PWM signal to send to each motor, those signals will be sent to four Electronic Speed Controllers (ESCs), which will turn those PWM signals into 3-phase power to the four motors. These motors will turn the propellers that lift our drone into the air and control the drone's movements.

Basic Movement:

The quadcopter flies using 4 propellers with motors and movement is generally controlled by increasing/decreasing the thrust generated by these propellers to change the throttle, pitch, roll and yaw of the drone. These propellers must move in different directions to be able to tilt and rotate it without it spinning out of control while keeping the net torque on the drone at zero. The throttle is increased by providing thrust from all four propellers equally, while pitch, roll, and yaw are varied as shown in the figure below.

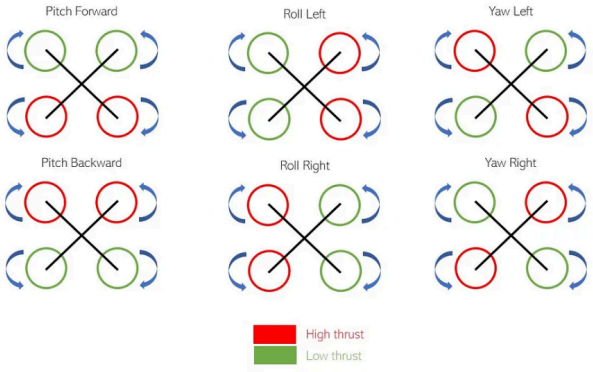


Figure from Hanewitch, Tim

Subsystem Descriptions:

Microcontroller Unit:

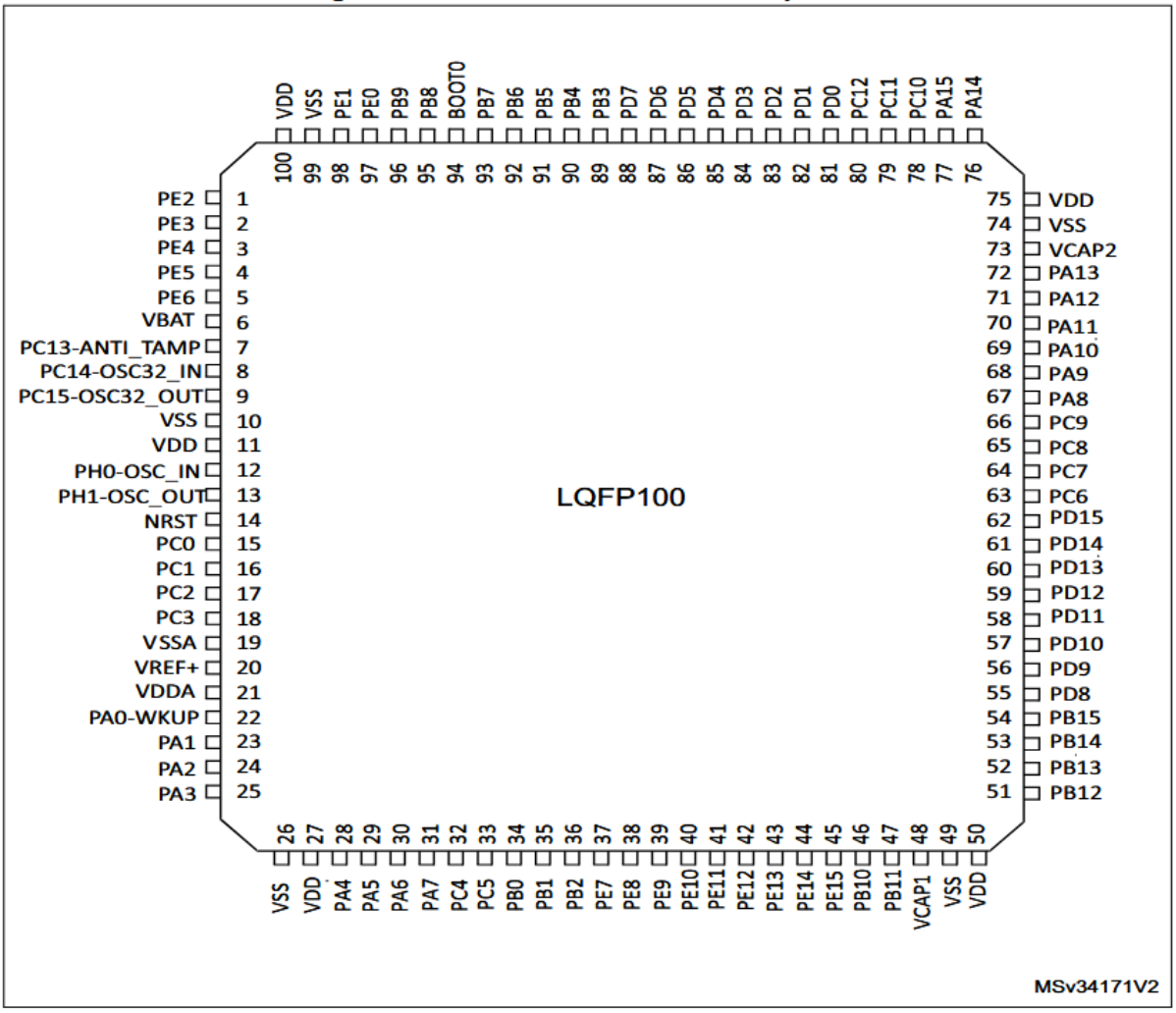
The Microcontroller Unit (MCU) is the “brain” of MARI as it handles the data processing and communication between the other subsystems. It continuously gathers data from sensors such as the gyroscope, the accelerometer, and the barometer, then runs real-time control loop algorithms that calculate how the motors should respond given the data from the sensors. Using the timers, the MCU generates PWM to control the electronic speed controllers (ESCs).

The advanced timer, TIM1, will generate the PWM for the ESCs which will go directly through the respective GPIO pins via the Alternate Function Multiplexer, specifically AF1.

Data from the IMU and barometer will be directly stored to the RAM via the Direct Memory Controller, specifically DMA1, through the APB1 bus. This is done to save the main CPU from constantly handling interrupts for every therefore massively reducing overhead time. They will be connected to two separate I2C buses and ground. For the ELRS RF Receiver, we will be using four 32-bit timers, two channels each.

All pins are connected through their specific ports and AF as per stated in the datasheet.

Figure 11. STM32F756Vx LQFP100 pinout



Component Specs:

Supply Voltage: 3.3 V

Total Max Supply Current: 320 mA

Max Supply Current per VDD line: 100 mA

SRAM: 1 MByte

Flash Memory: 320 KBytes

Power:

The drone's power system is supplied by a CNHL 6200 mAh 14.8 V (4S) 30C LiPo battery with an XT90 plug. The battery's 30C C-rating corresponds to a continuous discharge capability of 186 A, with its 60C burst rating allowing up to 372 A for brief periods, as defined by the battery's capacity and design. The XT90 connects to a bullet plug adapter, which provides the connection to the PCB. From the PCB, power is routed along two paths. The first path delivers battery voltage to the Electronic Speed Controllers (ESCs) via XT60 with bullet plugs, which convert the DC input into three-phase AC (3P-AC) to drive the motors. The second path uses two regulators to step the battery voltage down for the electronics: a buck converter generates a 5 V rail for the RF receiver with high efficiency, avoiding the ~70% power loss that a linear regulator would waste as heat, and a low-dropout regulator (LDO) provides a stable 3.3 V rail with reduced noise for the microcontroller, barometer, internal measurement unit (IMU), and oscillator.

A voltage-monitoring circuit is integrated into the PCB, dividing the battery voltage and feeding it into the MCU's analog-to-digital converter (ADC). The MCU interprets this divided voltage and classifies battery health into three alert levels:

- **Good (Green):** 1.59–1.33 V
- **Warning (Yellow):** 1.33–1.25 V
- **Critical (Red):** < 1.25 V

When the battery enters the Warning or Critical ranges (corresponding to low LiPo cell voltage), the MCU activates an LED alert to notify the user before unsafe discharge thresholds are reached. This setup balances efficiency with low noise, ensuring reliable power delivery across

all subsystems and supporting an estimated flight time of approximately 10 - 20 minutes depending on flight conditions.

Component Specs:

Voltage range: 14.8 V nominal (16.8 V max, 13.2 V min under load)

Discharge Rate: 30C Continuous, 60C Burst

Continuous current: 186 A

Burst current: 372 A for ≤ 15 s

Battery Cell Range: 4S LiPo

Sensors:

IMU(Inertial Measurement Unit) - BNO055

The purpose of the BNO055 inertial measurement unit is to determine the angular position of the drone, track the acceleration of the drone, and determine the relative position in respect to a magnetic field (mainly Earth's). These values will be stored on the BNO055's register and then sent to our STM32F microcontroller at 100 kHz in I^2C to regulate the signals being sent to the motors.

The gyroscope on the chip will determine the angular orientation and acceleration of the drone. The orientation of the drone will be determined and the pitch, roll, and yaw data will be stored/sent in the form of Euler angles. The gyroscope will also be able to send angular rate data to determine the direction and speed the drone is rotating. These measurements are the most important for stabilizing the drone.

The accelerometer will track the drone's movement across 3 axes to determine how fast the drone is going in whatever direction it is moving. It will send this data to the microcontroller, which will regulate the speed of the motors.

The IMU also has a built-in Kalman Filter in the form of Fusion Mode. This will make all the measurements done by the IMU more accurate by using the data from multiple sensors to adjust the data from each individual sensor. This will reduce the error from each individual sensor.

Component Specs:

Voltage Supplied: 3.3V

Current Supplied: 12.3mA

Barometer - BMP390

The purpose of the barometer is to track altitude data using barometric pressure sensors so we can control how high the drone can fly. It will store pressure values in Pascals in registers on the chip and send that data to the microcontroller so we can calculate the height of the drone relative to the ground. If the measurement from the barometer exceeds a certain value, the drone will stop accelerating upward.

Component Specs:

Voltage Supplied: 3.3V

Current Supplied: 3.2uA

Actuators:

ELRS RF Receiver

The M.A.R.I drone will receive PWM signals from the Radiomaster ER6 Express Long Range system (ELRS) receiver (RX) that connects to our microcontroller. The ER6 has six different output channels, but we use four of them, which our microcontroller processes with constraints to output speeds to different motors for throttle, pitch, yaw, and roll. The ELRS Protocol encodes our gimbal positions into packets that are received from our RX. Due to the amount of noise our drone experiences the farther it is, A Chirp Spread Spectrum (CSS) sweeps between 2-2.4 GHz for our receiver to demodulate and reconstruct the signals via dechirping which gets processed to be a PWM signal at a packet rate of 50 Hz. Our receiver will operate at 5V at 60mA.

Component Specs:

Voltage Supplied: 5V

Current Supplied: 60mA

Electronic Speed Controller (ESC)

The ESC converts our DC power supply into AC by utilizing MOSFETs to create three-phase power through the motor. We will be using a 14.8V battery and ESC's that are rated for 2-40A to vary the speed of rotation. The microcontroller sends a PWM signal to the ESC to control the speed of our Brushless motor from our battery. This is important as the ESC role in our project is to control maneuvering and thrust while in flight.

Component Specs:

Voltage Supplied: 14.8v

Current Supplied: 10-40A

Brushless Motors

Our MARI drone will have brushless motors as they offer higher efficiency from the reduced friction from brushed motors. We connect our motors to the ESC that sends power in three phases which drives rotation in our motor. We plan on utilizing the Emax ECO II Series 1300KV brushless motor to generate over 1000 grams of force per motor at 315 watts. We will be supplying 14.8v to the Brushless Motors between 2 - 40A which the ESC will handle in addition to the rotational speed of our motors.

Component Specs:

Voltage Supplied: 14.8v

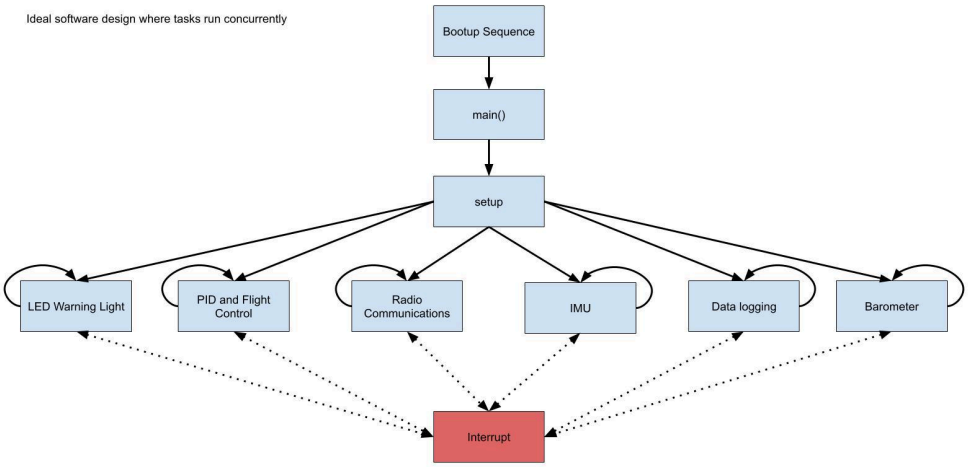
Current Supplied: 10-40A

Software (Operating System):

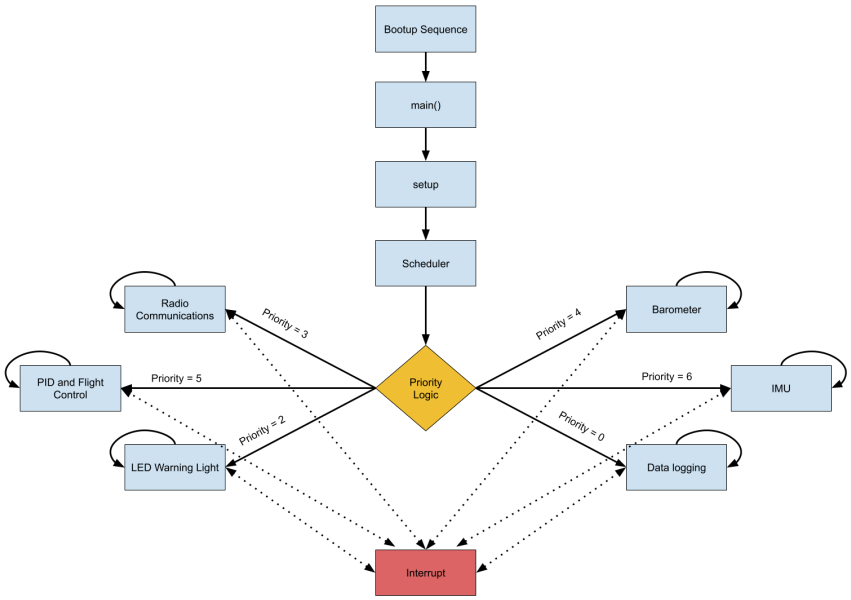
The main purpose of the software is to stabilize the drone by regulating the signals sent out to the ESCs/motors so the drone keeps an orientation that keeps it in the air. To achieve this, it takes the data from the RF receiver, IMU, and barometer. The data from the RF receiver allows the user to control the drone, while the data from the IMU and the barometer keeps the drone stable.

The RTOS ensures this by separating all of these functions as separate tasks efficiently. There is also an additional task that will collect and log data from the IMU and barometer to the flash memory. There is also the occasional check for the voltage in the LED Warning Light task.

Ideally, these tasks would run in parallel as shown in the flowchart below



However, since this is a single-core processor, it can only run one task at a time. Therefore these tasks will be switched in a very fast and efficient way through the task scheduler that they will appear to run concurrently.



Bootup:

Programmer: ST-Link SWJ-DP (JTAG or SWD)

System Bootloader Peripheral: USART1

Tools: STM32CubeIDE

When the MCU on the drone first powers up or resets, it begins by checking the BOOT pins/option bytes to decide where to start execution. In the normal case (BOOT0 = 0), it fetches the initial stack pointer and reset vector from the start of Flash memory and begins running the user bootloader, which then verifies and hands off to the main RTOS and flight software. If BOOT0 = 1, the chip instead enters the built-in System Bootloader stored in ROM, which allows programming or recovery over a USART interface. In either path, the MCU initializes its clocks, regulators, and peripherals, then proceeds into the application code, ready to run the flight stack.

After the initial bootup, the drone will enter one of three operating modes: RUN, STOP, and STANDBY. STOP mode is where most parts of the MCU are powered down, however it retains all SRAM and registers. It only “wakes up” to RUN from an external interrupt such as the RF Controller. STANDBY is a near total “dormancy” mode where nearly all parts are powered down, including the registers and SRAM. It can only “wake up” from a WKUP pin, then afterwards the software will reinitialize all registers.

All code needed for the bootup sequence is automatically generated by STM32CubeIDE and stored into flash memory.

Operating System (MIKU)

The drone software runs on a custom RTOS called the Multi-Integrated Kernel Unit (MIKU) where separate tasks handle sensing, control, and safety. Hardware events from the

IMU, barometer, RF receiver, and power monitor trigger interrupts that wake the appropriate tasks via semaphores. These tasks communicate through shared data structures such as CurrentDroneStates, Altitude, RC_Commands, and Flight Mode, which together describe the drone's current state and how it should behave.

The IMU generates periodic interrupts that wake the IMU Task. This task simply records the current state of the drone based on IMU data and writes it into the CurrentDroneStates structure. That state includes roll, pitch, and yaw angles, their angular rates, and acceleration/direction data. The IMU Task does not perform control itself; it just ensures that the rest of the system always has fresh motion data.

The barometer triggers the Baro Task through its interrupt. The Baro Task reads the barometric pressure, computes the current altitude, and stores this in the Altitude data structure. It also checks whether the measured altitude is higher than the configured safety altitude. If it is, the Baro Task forces the altitude data so that the controller will drive the drone back toward that safe altitude, ensuring the drone never stays above the allowed height.

Radio commands from the pilot are handled by the RF hardware and RC Task. The RF interrupt wakes the RC Task, which decodes the packet and updates the RC_Commands structure with the latest throttle, pitch, roll, yaw, and auxiliary switch positions. The Flight Mode flag determines whether the system is in manual or automatic flight. In manual flight, the PID Task is allowed to use RC_Commands directly to move the drone as the pilot intends. In automatic flight, the PID Task ignores RC_Commands and relies only on internal logic, such as target or safe altitude, to control the drone.

The radio receiver input will take the input from the user who will control throttle/pitch/roll/yaw using the radio controller, which will be translated to individual signals given the following equations:

Motor1: Front Left = throttle - roll - pitch + yaw

Motor2: Front Right = throttle + roll - pitch - yaw

Motor3 Rear Right = throttle + roll + pitch + yaw

Motor4: Rear Left = throttles - roll + pitch - yaw

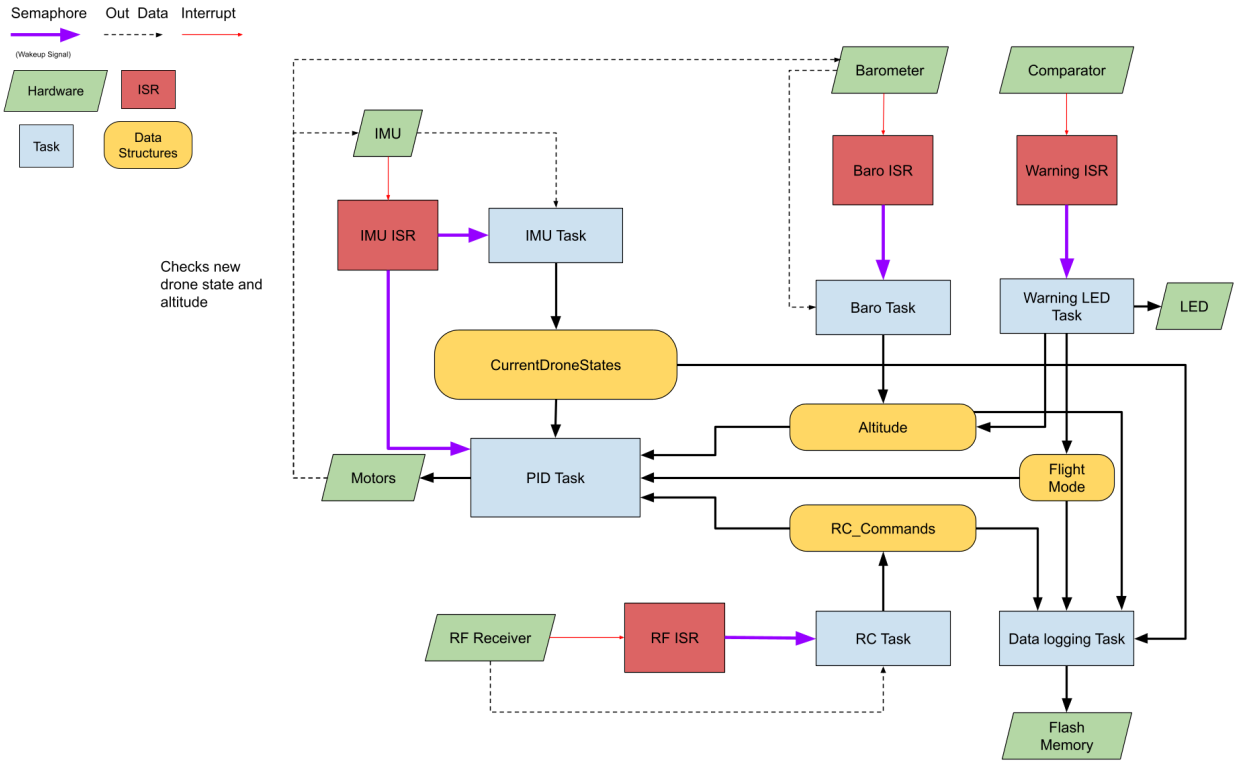
To adjust for error, we will be implementing PID (Proportional Integral Derivative) equations to adjust the roll, pitch, and yaw data from the IMU.

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

Within the RTOS, these equations are implemented inside the PID Task. At each control tick, the PID Task reads CurrentDroneStates, Altitude, RC_Commands (if Flight Mode is manual), and the Flight Mode flag. It computes the error between the desired attitude/altitude and the measured values and then applies PID control. Here, u(t) is the adjustment we are making to the commands sent to the motors, e(t) is the amount of error between the intended input from the user and the actual data from the sensor, and Kp, Ki, and Kd are tunable hyperparameters. This PID control signal is then added to the throttle/pitch/roll/yaw commands in the motor equations above. Using those adjusted values, the PID Task generates the PWM signals to the motor ESCs, increasing or decreasing their duty cycles to self-stabilize the drone while still tracking the pilot's intent when manual control is allowed.

The Warning LED Task is responsible for reacting to low-power conditions. When the voltage-monitoring circuit detects that the battery is low, it triggers an interrupt that wakes the Warning LED Task. This task turns on the appropriate LED pattern and also forces Flight Mode to automatic flight while setting the safe altitude to 0 (ground level). With these changes, the PID Task begins commanding a descent to altitude 0 and will not accept any new RF controls from RC_Commands. This guarantees that once a critical power condition is detected, the drone performs a safe, automatic landing under RTOS control.

Full Software Flow:



Citations:

Hanewich, Tim. “How I Developed the Scout Flight Controller, Part 1: Quadcopter Flight Dynamics.” *Medium*, Medium, 31 July 2023, timhanewich.medium.com/how-i-developed-the-scout-flight-controller-part-1-quadcopter-flight-dynamics-400af73d21db.