

EE3080 DIP Guide

UAVIONICS

Version 2.1
(10 February 2021)

Notes:

- This guide can get updated along the way. Please keep an eye on newer versions.
- Codes for this project can be found here:
<https://github.com/NelsenEW/EEE-DIP-UAVONICS>

Table of Content

Overview	3
DIP Overview.....	3
Overview on the Hardware.....	3
i. Quadcopter	3
ii. ESP32 Development Board with Camera.....	4
iii. IMU Sensor Module	5
iv. Other Sensor Modules	5
End Goal and Examples of Application	6
Experiments with the Hardware	7
Playing with the Quadcopter	7
Experiment with ESP32-CAM.....	7
Uploading Codes to the ESP32-CAM Board	8
Important Notes for Working with ESP32-CAM	10
Experiment with GY-91 IMU Sensor	10
Explanation for This Example.....	12
Experiment with NH-Flie PCB	13
Schematic_ESP32 NH	13
ESP32 NH Description	13
Uploading codes to NH-Flie PCB	14
Experimenting Camera Module with NH-Flie	14
Experimenting built-in MPU-6050 on NH-Flie	15
Setting Up the ESP32 Toolchain	16
Installing the Necessary Software.....	16
Installing ESP-IDF.....	16
Testing the ESP-IDF Installation	19
ESP-IDF Example Projects.....	19
ESP-IDF SDK Configuration.....	20
Building the Project.....	22
Uploading to the ESP32	24
Working with the Drone PCB and esp-drone-nh Project	26
Testing the Workspace	27

Overview

DIP Overview

UAVONICS DIP is administered in 2 phases.

The first phase emphasizes on cultivating direct individualized hands-on experience specific to electronics and coding. To achieve this, students will engage in building a flying camera, driven by [ESP32 Microcontrollers](#). The camera will have a small form factor (smaller than 10cm x 10cm) maneuvered by user from a smart phone.

The second phase emphasizes on collaborative team work to achieve a common objective of crafting novel solution to a real-world problem scenario to be unveiled during the project. The whole project brings forth a holistic experience of hands-on practicality; harnessing creativity to innovate and improvise in problem-solving.

In short, the objective of the first phase is building a camera-equipped quadcopter, while the aim of the second phase is to utilize the quadcopter for real-world applications.

Overview on the Hardware

We will use various hardware for this project. The main ones are:

1. Quadcopter drone
2. ESP32 development module with camera
3. IMU sensor module
4. Other sensor modules (if needed)

i. Quadcopter

The quadcopter used for this project is small (also known as MAV; Micro Aerial Vehicle) for safety and legal reasons. Some specifications of the quadcopter:

- **Model:** Eachine E010 Nano Drone



Figure 1. Eachine E010 Nano Drone.

- **Specifications:**
 - Motor: 4x brushed motors
 - Flight time: 5-7 minutes (onboard LED blinks if battery goes low)

- Charge time: approx. 40 minutes
- Remote control range: 30 m
- Dimension/weight: 85 x 85 x 30 mm
- Net weight: 25 gr

Due to the small size of the drone, it has **short flight time** and **small payload** (we cannot add much weight to the drone), which are the limiting factors for this project.

ii. ESP32 Development Board with Camera

The ESP32 is a widely popular family of development boards. It has a powerful *dual-core* microcontroller and Wi-Fi and Bluetooth capability built into the board. It is mostly used for IoT (Internet of Things) applications.

A microcontroller is a device with multiple input/output pins that can be programmed to perform lots of different tasks. You can think of it as a single-chip, tiny computer (which is obviously not as powerful as a “real” computer) that can interface with sensors and actuators (motors, etc.). Arduino boards, if you are familiar with them, are also development boards featuring microcontrollers on them.



Figure 2. Various members of the ESP32 family.

From <https://randomnerdtutorials.com/getting-started-with-esp32/>.

For this project, we also need a camera. The ESP32-CAM is a member of the ESP32 development board family with built-in camera. We can either use this development board or create our own design based on this board’s design.

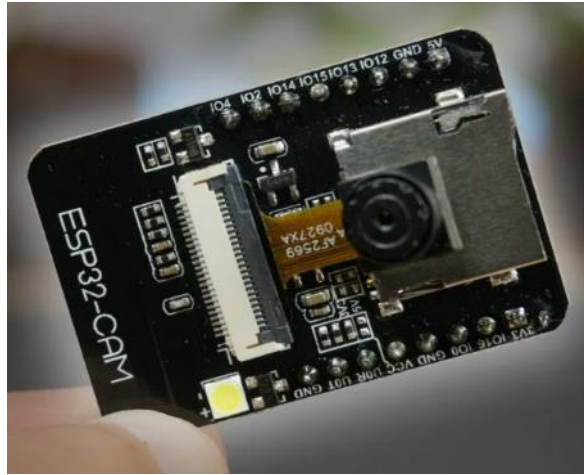


Figure 3. ESP32-CAM development board.

From <https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/>.

iii. IMU Sensor Module

To stabilize the drone, we need an Inertial Measurement Unit (IMU) sensor. This can be a 6-axis accelerometer & gyroscope¹ or a 9-axis accelerometer, gyroscope, and magnetometer. Although the additional magnetometer (which measures magnetic field strength) can be handy, 6-axis IMU sensors are usually enough.

The drone that we use already has built-in IMU sensor to determine how to control each of its 4 motors to achieve the desired maneuver. However, we cannot access the data from this sensor. Instead, we use our own IMU sensor module. An example of IMU sensor module is the GY-91 module based on the MPU9250 9-axis IMU chip. This module also has a built-in barometer (BMP280).

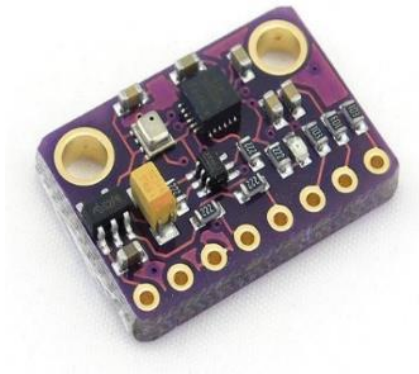


Figure 4. GY-91 9-axis IMU sensor module.

iv. Other Sensor Modules

We may also need additional sensor modules to augment the capability of the drone. For example, a ToF (time of flight) sensor module can be handy to measure the height of the drone above ground

¹ Accelerometer measures linear acceleration, gyroscope measures angular velocity (rate of rotation). Both the accelerometer and gyroscope have 3 axis of measurement each (hence "6-axis accelerometer & gyroscope").

accurately, which is hard to get from IMU sensor alone. Keep in mind of the **payload limitation** of the drone, though.

End Goal and Examples of Application

The goal is to build a quadcopter with camera and sensors so that it is versatile enough to perform useful tasks. Some example of useful tasks:

- Flying camera
- Face tracking
- Waypoint following (flying in a pre-determined path and altitude)
- etc.

We have put several demo videos on the GitHub repository under the `demonstration` folder: <https://github.com/NelsenEW/EEE-DIP-UAVONICS/tree/main/demonstartion>. These are the example applications that you might look into developing.²

² These examples are done with different drone(s), so it is fine if your drone has different stability characteristics.

Experiments with the Hardware

This section is for experimenting with each individual hardware used for this project and getting some sense of how they work. Try to do these experiments in Week 1 to allow enough time for further developments.

Playing with the Quadcopter

You are given an unmodified Eachine E010 drone. Fly and play around with the drone to get a sense of how it flies. Before taking off, **calibrate the drone** by connecting the remote control to the drone then holding both joysticks to the bottom left corner. Refer to the drone user manual for the controls.

A few things about the flight characteristic of the drone that you might notice upon flying it:

- The drone does not have advanced stabilization feature; it needs constant adjustment from the pilot to counter drifts (i.e. the drone does not hover in place and drifts if you let go of the remote controller).
- The drone quickly loses thrust as the battery discharges; overtime you need to apply more power to make it hover at the same height.

It might seem hard to fly the drone at first, but after some time you should be familiar with the handling and be able to fly it stably. The aim is to **make our camera-equipped drone as stable as this drone**.

Experiment with ESP32-CAM

This experiment is for you to get started with the ESP32-CAM. To program the ESP32-CAM board, we will use Arduino IDE. Install Arduino IDE on your computer if you have not done so: <https://www.arduino.cc/en/software/>.

After that, install the ESP32 add-on so you can program the ESP32-CAM board (which is not an Arduino board) using the Arduino IDE. Guide to do so:

- <https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/> (for Windows)
- <https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-mac-and-linux-instructions/> (for Mac OS and Linux)

Finally, there are some libraries that you will need to download on Arduino IDE for the example code to run. All the required libraries are listed in `packages.txt` file on the GitHub page (<https://github.com/NelsenEW/EEE-DIP-UAVONICS/blob/main/packages.txt>). As we progress, the list of libraries that you need to install might grow, so keep an eye on this list. Guide to install libraries on Arduino IDE: <https://www.arduino.cc/en/guide/libraries>.

We will now run example code available in the Arduino IDE to familiarize with the process of uploading program to the board and see the capability of the board. The example code (`CameraWebServer`) streams the video data taken by the onboard camera to a webpage.

Follow the guide here (serves as a nice getting-started guide): <https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/>.

In the link above, an FTDI programmer is used to program the board. However, you will be using an Arduino board to replace the FTDI programmer. See **“Uploading Codes to the ESP32-CAM Board”** below.

Uploading Codes to the ESP32-CAM Board

The board does not have onboard USB connector, hence an external programmer is needed to upload programs to it. We will use an Arduino Nano as the programmer. Make the following connection:

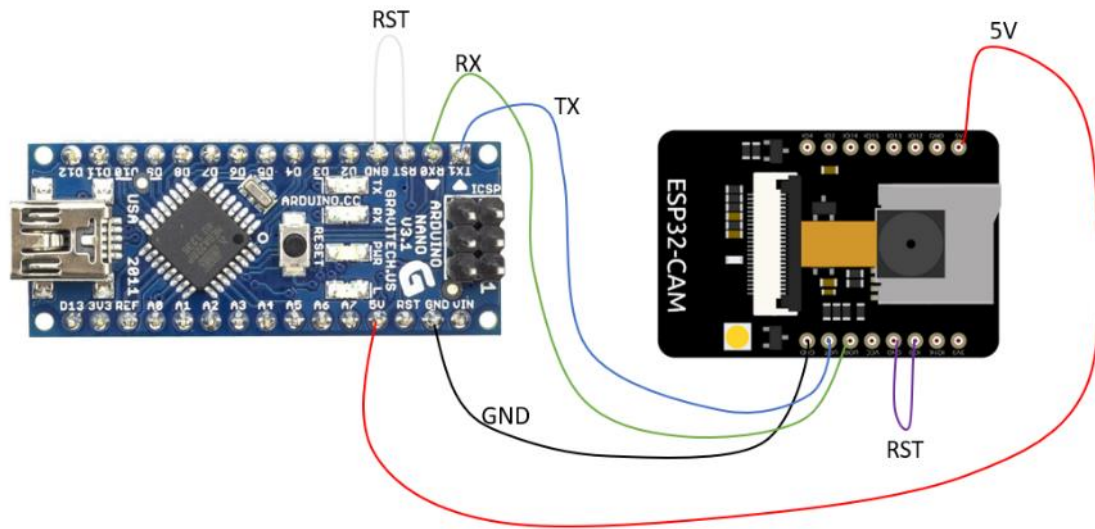


Figure 5. Configuration to upload program to the ESP32-CAM board.

Note that the RX of Arduino Nano connects to RX of the ESP32-CAM³, and TX to TX. After making the necessary connection, plug the Arduino Nano to your computer using the board's USB port. Reset the ESP32-CAM so it enters programming mode. Resetting the ESP32-CAM can be done in the following ways:

- Pressing the RESET button (normal way).
- If the RESET button is unreachable, the board can be power cycled: unplug and re-plug the GND cable going from the Arduino Nano to the ESP32 board (black cable), **not the 5V/red cable**⁴.

³ If you are familiar with UART: you might think that this is a typo; it is not. Usually TX connects to RX and vice versa. For this case however, RX connects to RX and TX to TX because the ESP32-CAM “borrows” the Arduino Nano’s onboard USB-TTL converter chip responsible for translating the USB protocol your computer uses into UART protocol that the ESP32-CAM understands. This is also why you need to connect the Arduino Nano’s RST pin to GND; to keep the Arduino Nano’s own microcontroller on reset so that it does not disturb the communication between your computer and the ESP32-CAM.

⁴ The purpose of “unplugging and re-plugging” the GND cable is to cut and restore power to the ESP32-CAM board, effectively resetting it. Normally, doing so on the 5V cable instead of the GND should also work. However, if you have other things connected to the ESP32-CAM board (which will be the case as we move further),

On the Arduino IDE under the “Tools” menu, set the **board to “AI Thinker ESP32-CAM”** and the **port to the correct one** (one way to know is to unplug the Arduino Nano and see which one disappears from the list). See the following screenshot:

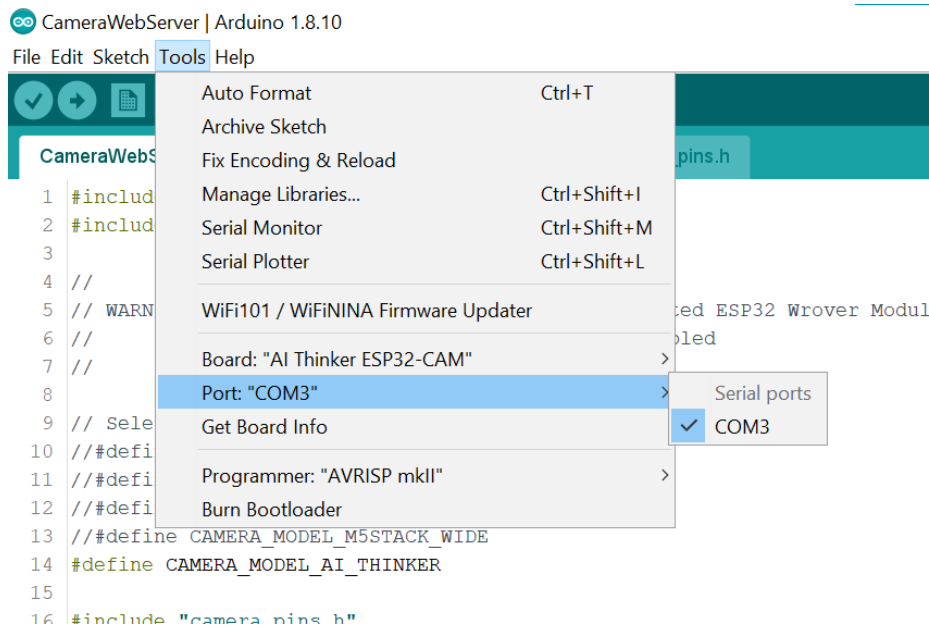


Figure 6. Choose the correct Board and Port under the Tools menu.

Finally, press Upload (the right arrow logo below the “Edit” menu).

Remarks:

- **This will be the method that you use every time you upload codes to the ESP32-CAM board.**
- If the upload fails, try to reset the board again and reupload.
- After uploading the code, do not forget to **disconnect IO0 from GND (purple cable) and reset the board** to bring the ESP32-CAM out of programming mode. The connection from IO0 to GND is to bring the board to programming mode. Hence, forgetting to remove it will make the board stay in programming mode (does not run the uploaded code). In that case, this message will be printed on the Arduino IDE serial monitor:

```
rst:0x1 (POWERON_RESET),boot:0x3 (DOWNLOAD_BOOT(UART0/UART1/SDIO_REI_REO_V2))
waiting for download
```

- When you need to upload another code to the board, reconnect IO0 to GND and reset the board to bring it back to programming mode.

unplugging the 5V cable does not always remove power from the ESP32-CAM. The board can still be “accidentally” powered via the GPIO pins due to the ESP32-CAM’s internal circuitry.

TL;DR: unplugging the 5V cable does not always remove power from the board, unplugging GND does.

Important Notes for Working with ESP32-CAM

- A website listing errors that might occur when using the ESP32-CAM board and how to fix them:
<https://randomnerdtutorials.com/esp32-cam-troubleshooting-guide/>.
 - Using Arduino IDE, the syntax and language used for programming the ESP32-CAM board are similar to that of Arduino boards (e.g. `digitalWrite()` also works for ESP32-CAM, as well as many other Arduino functions). This is helpful if you are familiar with Arduino boards.
 - Like other microcontrollers, ESP32-CAM has multiple input/output pins (also referred to as GPIO) which can be programmed independently. Some GPIOs also have special functions, such as for communicating with other devices (including a computer with proper hardware).
- ESP32-CAM pinout:

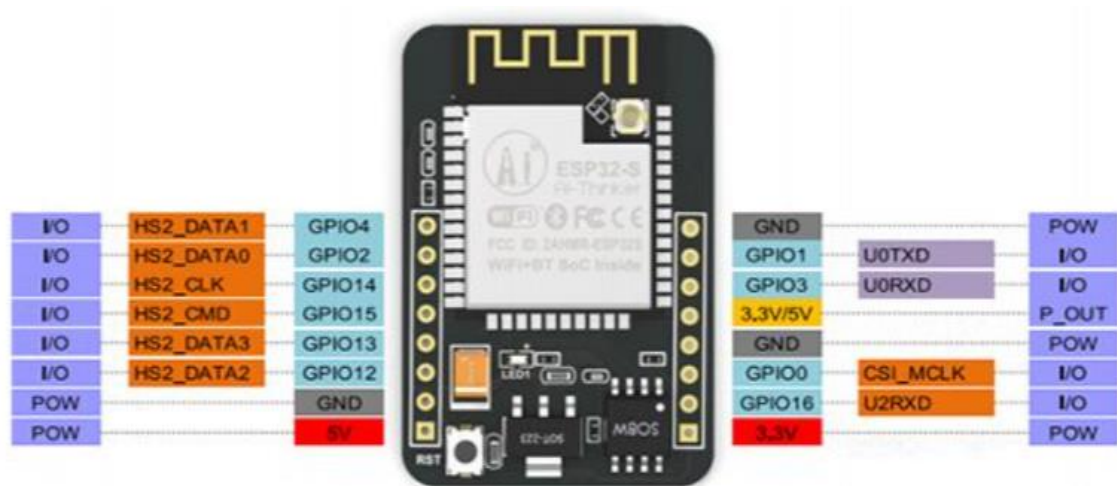


Figure 7. ESP32-CAM pinout.

From <https://www.seeedstudio.com/ESP32-CAM-Development-Board-with-camera-p-3153.html>.

- It is not easy to connect to enterprise network (NTUSECURE, NTUWL, etc.). Connect the ESP32-CAM board to a private router or to mobile hotspot (e.g. from your phone) for the `CameraWebServer` example and for further developments.
- The `CameraWebServer` example also has facial recognition feature (for demonstration purpose). The algorithm runs directly on the ESP32-CAM board and is very slow (approx. 2 FPS). Hence, if you want to run computationally heavy tasks (e.g. face recognition), **you have to send the image data from the ESP32-CAM to a computer** (e.g. your laptop), then the computer can perform the task and send the result back to the drone.
- The ESP32-CAM board has a dual-core microcontroller. You can make use of both cores to split the workload.

Experiment with GY-91 IMU Sensor

In this experiment, the ESP32-CAM reads data from the IMU sensor and prints the reading on Arduino IDE's serial monitor. You can download the code for this experiment from this project's GitHub repository under `examples` folder: `mpu9250_test.ino`.

Note: some example codes on the GitHub page also include header files (file extension: `.h`) and/or other C++ codes to compile correctly (you can see what header files the code needs by looking at the `#include<>` directives inside the code). We have put these include files under the `include` folder

on the GitHub repository. Move or copy these files to the same folder as the code before you compile it.

The ESP32-CAM communicates with the GY-91 module via a communication protocol known as Inter-Integrated Circuit (I²C). This protocol allows multiple devices (up to 112 for standard addressing scheme) to communicate with each other using only two wires. Arduino provides functions to perform communication using I²C, but you can check this out if you want to know more about I²C: <https://www.robot-electronics.co.uk/i2c-tutorial>.

For this experiment, make the following setup:

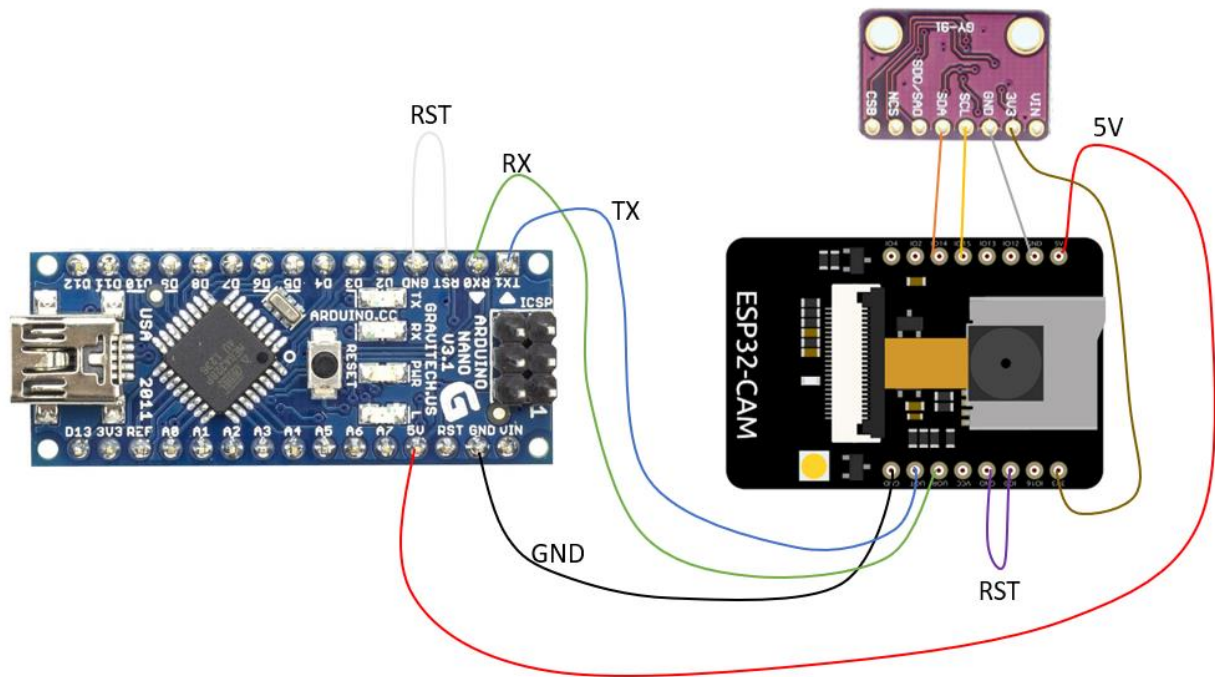


Figure 8. Hardware connection for experiment with GY-91 IMU sensor.

Notice that the setup above is for both programming the ESP32-CAM and reading data from the GY-91 module. You can see that only 4 wires go to the GY-91 module; 2 for power, 2 for I²C communication⁵. The GPIOs of the ESP32-CAM which we use for the I²C communication can be specified from the code (and can be changed).

After uploading the code to the board, do not forget to remove the connection from GPIO0 to GND (purple cable) and reset the ESP32-CAM. **Do not** remove other wires or unplug the Arduino Nano from your computer. Then, open Serial Monitor on the Arduino IDE and set the baud rate of the monitor to 115200. You should see some output on the serial monitor.

If the code works successfully, try to move the sensor and observe the reading, and read through the code to get a glimpse of how it works. If it does not run, check that:

- You got all connections correct. The GY-91 module's red LED will light up if it receives power (does not necessarily mean that the I²C connection is correct).

⁵ The 2 wires are SDA for serial data and SCL for serial clock. The clock's purpose is to synchronize all devices on the bus; recall EE2004.

- You have disconnected GPIO0 from GND (purple cable) **and reset the ESP32-CAM** after uploading.
- You have selected the correct port on the Arduino IDE Tools menu.
- You have set the correct baud rate on the serial monitor.

Explanation for This Example

The ESP32-CAM queries the readings from the GY-91 module via the I²C lines (as can be seen in the hardware connection). The readings from the sensor module are raw and need to be processed by the ESP32-CAM:

- The readings sent via I²C are not in proper units of measurement. They need to be converted into proper units (e.g. m/s² for acceleration). The code already handles this.⁶
- Converting the readings alone might not be enough. Sometimes we would also like to apply some algorithm on the reading, for example to smooth out the reading using a filter algorithm, or to estimate the sensor's position and attitude (roll, pitch, and yaw).

After processing the readings, they are sent to the computer to be displayed on the Arduino IDE Serial Monitor. The computer understands USB protocol, while the ESP32-CAM does not. Thus, in order to send data to the computer, we have to convert the protocol that the ESP32-CAM supports into USB protocol that the computer knows.

The protocol on the ESP32-CAM that we use to send the data to the computer is called UART (Universal Asynchronous Receiver Transmitter). Similar to I²C, it only uses two wires to send and receive data. However, unlike I²C:

- UART only supports communication between 2 devices (one-to-one).
- The 2 wires used are RX for receiving and TX for transmitting (no clock signal). Since it has dedicated wires for receiving and transmitting, UART supports transmitting and receiving data at the same time (referred to as “full-duplex”) provided that both the communicating devices support that capability. In contrast, I²C is half-duplex (cannot send and receive at the same time).
- Due to the absence of clock signal, the two devices must therefore agree beforehand on how fast the communication should be performed. This is why you need to set the Serial Monitor's baud rate to 115200; you are telling the computer to communicate at 115200 bits/sec. This same speed is also programmed to the ESP32-CAM in the code.

The ESP32-CAM does not have USB-UART converter onboard, neither does the computer. The Arduino Nano, however, has one. The connection we make from the ESP32-CAM to the Arduino Nano is to allow the ESP32-CAM to “borrow” the Arduino Nano's USB-UART converter (also referred to as USB-TTL) so that it can communicate with the computer. This is why we do not remove the Arduino Nano after programming the ESP32-CAM⁷; we still need it.

⁶ In case you want to know how to convert the reading into proper units, refer to the datasheet of the sensor used on the module (MPU9250).

⁷ By the way, when we program the ESP32-CAM, the board receives the new code via UART, too. This is why we need the Arduino Nano for programming the board.

Experiment with NH-Flie PCB

Schematic_ESP32 NH

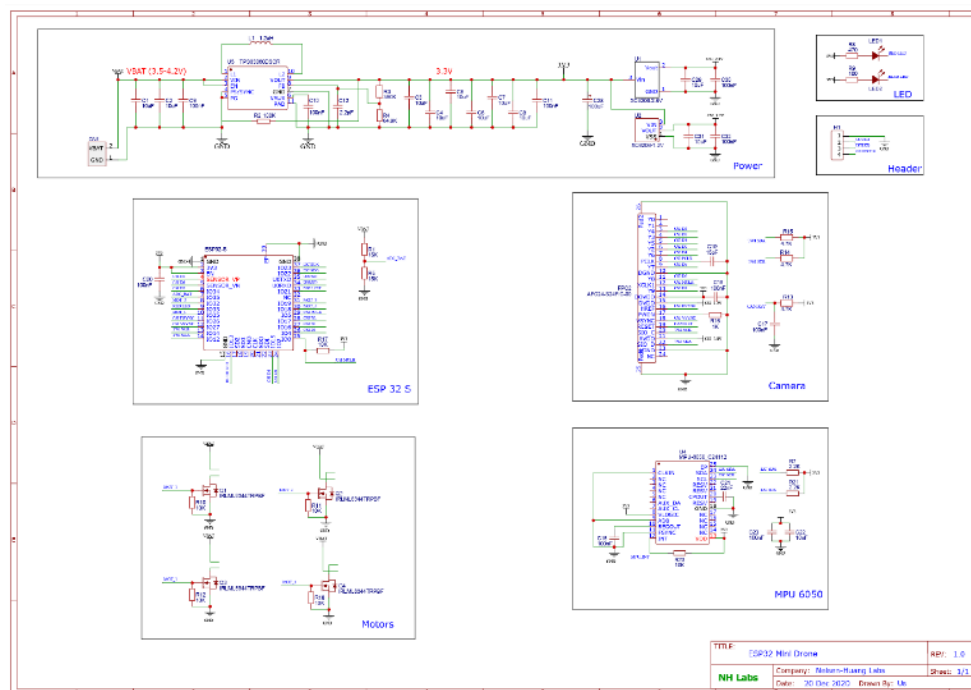


Figure 9. ESP32 NH Board Schematic

ESP32 NH Description

The NH-Flie board behaves similarly to ESP32 CAM but it has MPU-6050 module embedded on it. There are also four output pins located on the long side of the PCB. (*Refer to the datasheet to have a better understanding of the PCB*)

We will upload the example codes onto this board and obtain readings from the MPU-6050.

Important Notes:

Blue LED indicates the board is in BOOT mode, Red LED indicates the board is in RUNNING mode.



Figure 10. NH-Flie PCB Front



Figure 11. NH-Flie PCB Back

Uploading codes to NH-Flie PCB

The PCB board does not have onboard USB connector, an external programmer is required to upload program to it. In this case, we will use Arduino Nano as the programmer.

Make the following connection:

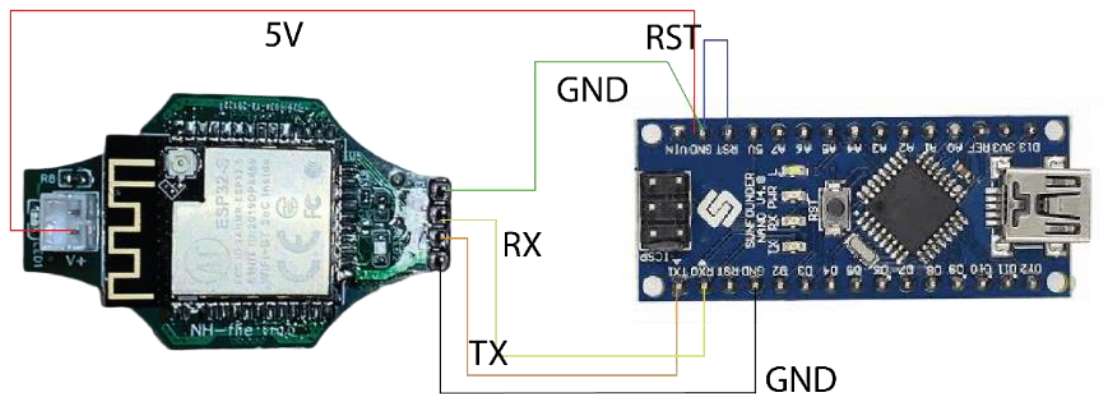


Figure 12. Configuration to upload program to NH-Flie

Important Notes:

- When the blue LED will light up when the board enters programming mode. Do note that it is unable to upload codes onto the board while the red LED is on.
- After uploading codes onto NH-Flie, remove the GND wire labelled in GREEN color.

Experimenting Camera Module with NH-Flie

Configuration

Sample code named *0.NH_CameraWebServer.ino* has been provided in Github. In this case, we will still be using AI-THINKER for the board. Connect the camera module to slot mounted on the board.

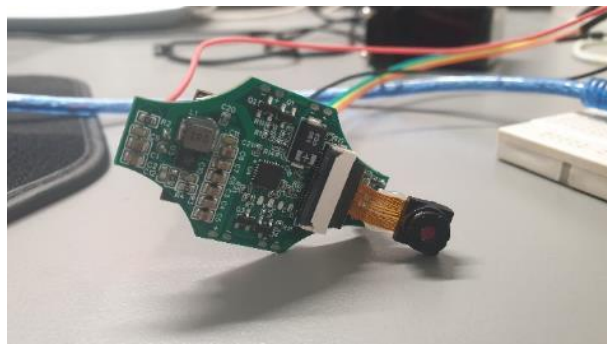


Figure 13. Connecting the camera module

Remember to change your Network Credentials including SSID and Password within the codes to connect your NH-Flie to local network.


```
#include "camera_pins.h"

const char* ssid = "*****";
const char* password = "*****";
```

Figure 14. Changing Network Credential in the codes

Accessing Server

“Wifi connected” message will be printed in the Serial Monitor if NH-Flie has successfully connected to the network.

Connect your device to the same local network and browse to <http://192.168.X.X> (IP address printed out in the Serial Monitor) to access the web server.

Experimenting built-in MPU-6050 on NH-Flie

The NH-Flie board has a built-in Inertia Measurement Unit (IMU) sensor named MPU-6050.

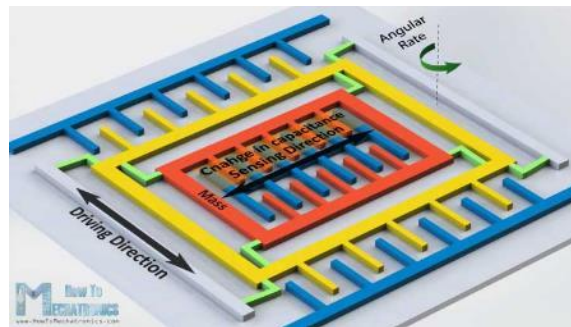


Figure 15. Mechanism of MPU-6050 ¹

MPU-6050 is a Micro Electro-Mechanical Systems (MEMS)¹ and it has a 3-axis gyroscope, 3- axis Accelerometer and a Digital motion processor in it. This helps us to measure acceleration, velocity, orientation, displacement, and many other motions related parameter of the drone.

Obtaining Readings

Sample code named 1.MPU6050_test.ino has been provided in Github. The function of this code is to obtain readings from the sensors and print it on the Serial Monitor.

Important Notes:

- Set the Baud Rate to 115200 to view the readings in Serial Monitor.

¹ To understand how MEMS works, visit this website <https://howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyroscope-magnetometer-arduino/>

Setting Up the ESP32 Toolchain

Installing the Necessary Software

In order to develop and upload programs to the ESP32 on the PCB, we need to set up the toolchain. We are going to use **ESP-IDF** (Espressif IoT Development Framework), which is provided directly by the company behind the ESP32 family. We are not going to use Arduino IDE as it is not powerful enough for the task.

Before setting up the toolchain, install these on your computer if you have not already:

1. **Git** (not to be mistaken with **Github**), a version control software. You do *not* need to install Github on your desktop. Link: <https://git-scm.com/download>
2. **Python 3.6 or later**⁸. You can install Python 3 using Anaconda, although you do not have to. Link: <https://www.python.org/downloads/release/python-379/>
3. **Microsoft Visual Studio Code (VS Code)**, not to be mistaken with **Visual Studio**. This is a powerful code editor that we will use. Link: <https://code.visualstudio.com/download>

Additional setup for Mac user using terminal (in order):

4. **Homebrew**, MacOS package manager.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

5. **CMake**, build process integration for cross environment.

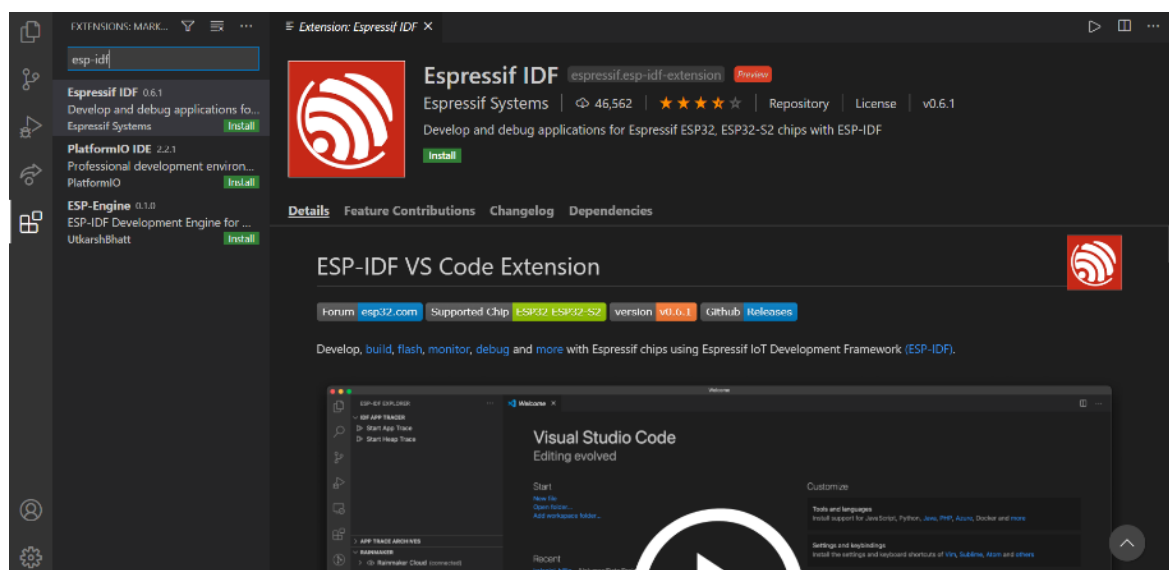
```
brew install cmake
```

6. **Ninja**, a small build system for speed.

```
brew install ninja
```

Installing ESP-IDF

After installing all the above software, launch VS Code and install the **ESP-IDF extension**.



⁸ We have tested with Python 3.7.9. Newer versions (e.g. Python 3.9.1) should also work although we have not tested yet.

Figure 16. ESP-IDF extension on VS Code extension marketplace.

After you finish installing the extension, you should see this window on VS code:

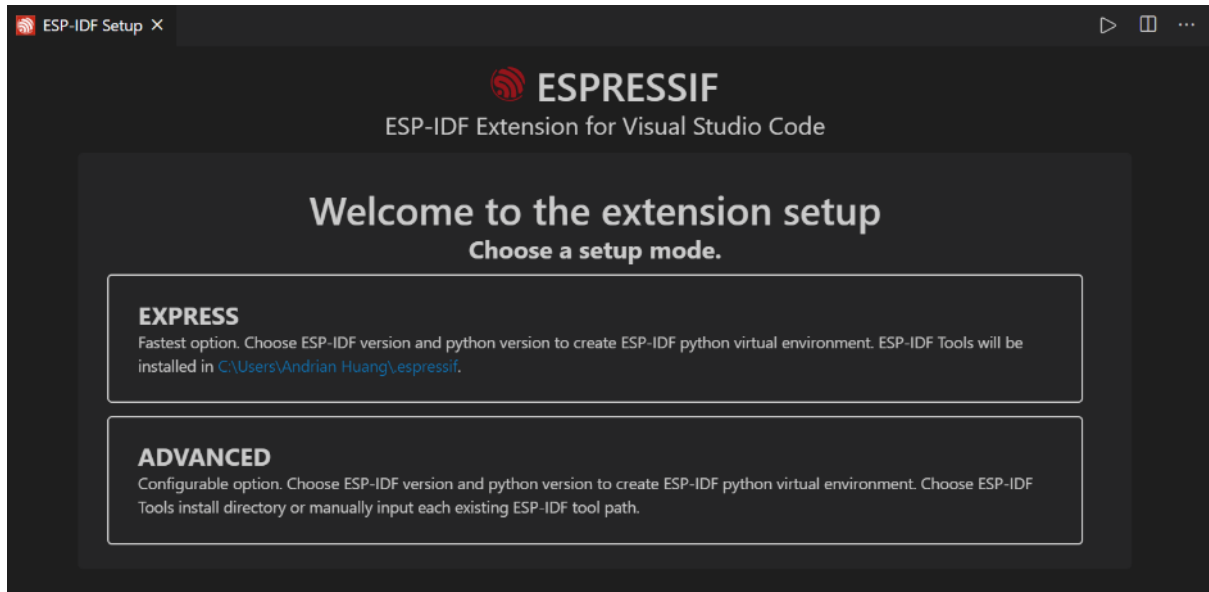


Figure 17. ESP-IDF setup screen after installing the extension.

Next, For **Mac** user choose “Express”, for **Windows** user choose “Advanced”.

The ESP-IDF container directory **does not support directory with whitespaces in its full path**. As such, choose a folder which full path does not contain spaces.

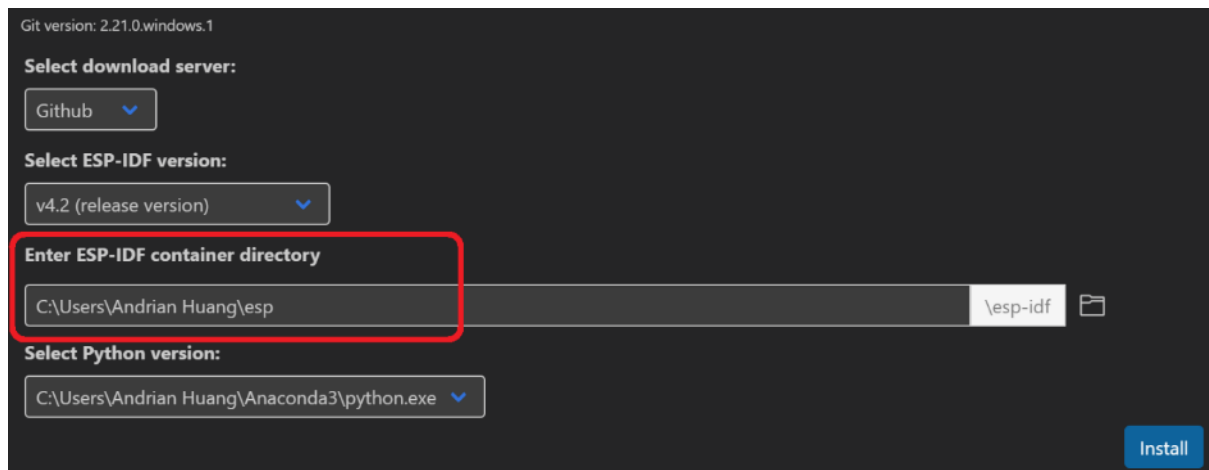


Figure 18. Choose the ESP-IDF container directory to be a path without whitespaces. The chosen directory in the example above **will NOT work!**

For example, on Windows computer you may create a folder called “espressif” on C: drive and choose this folder to be the directory for ESP-IDF container. For the other dropdown lists:

- Download server: we tested with Github but this choice should not matter.
- ESP-IDF version: “v4.2 (release version)”.
- Python version: choose your Python installation.

See the screenshot below for a correct example. After choosing the correct directory, click “Install” and wait for ESP-IDF to be installed.

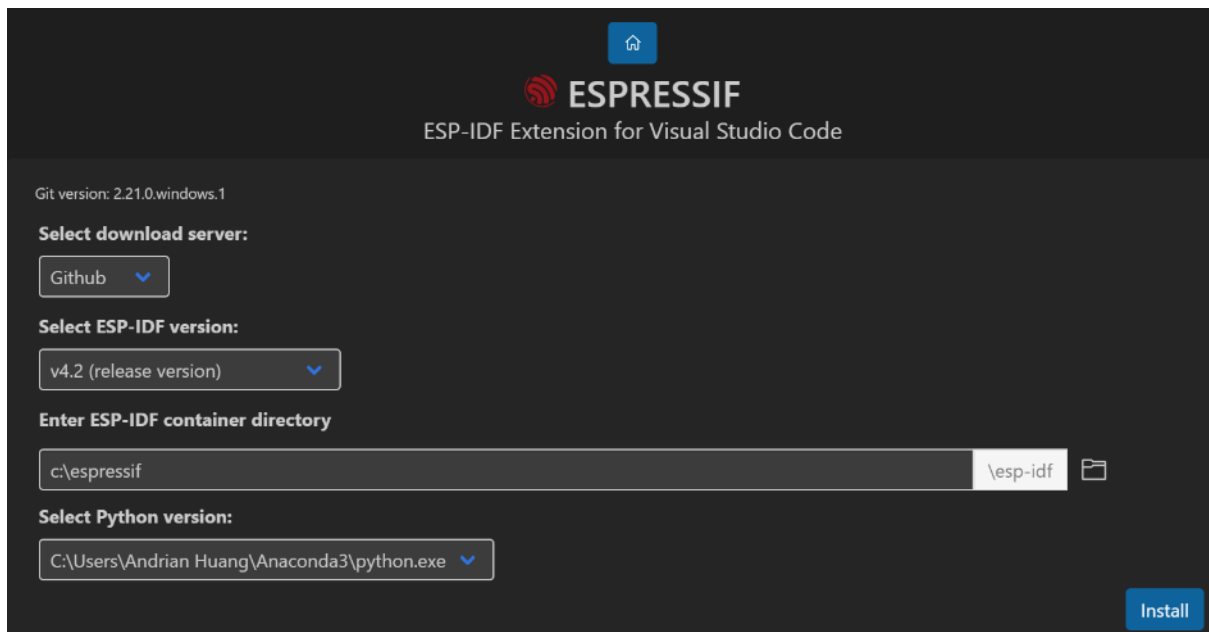


Figure 19. Example of a working setup.

After ESP-IDF installed successfully, you should see this screen (if you chose “Advanced”):

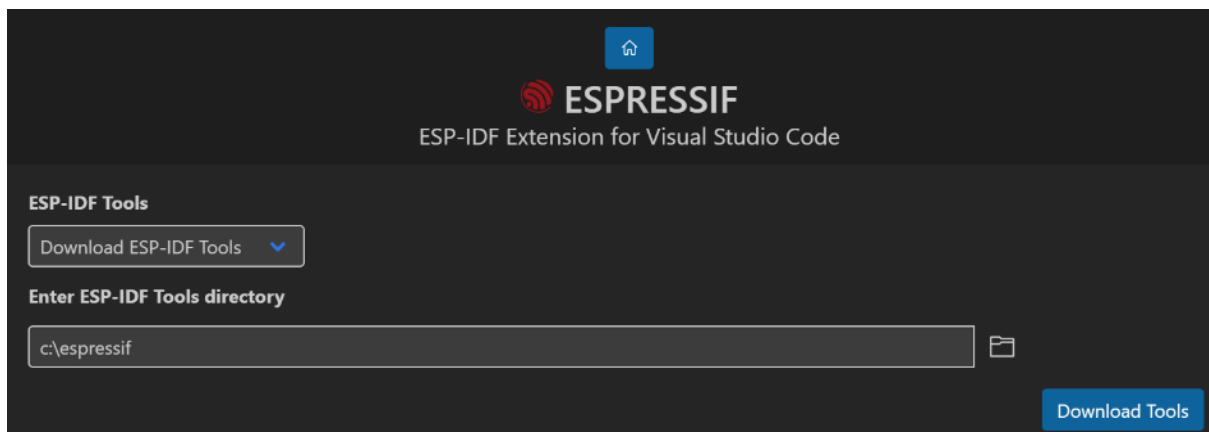


Figure 20. ESP-IDF Tools download page. This setup works (there are no whitespaces for the ESP-IDF Tools directory).

Here, choose “Download ESP-IDF Tools” (should be the default). Similar to before, the ESP-IDF Tools directory **must not include any whitespaces**. By default, ESP-IDF Tools is put under the folder “.espressif” (a hidden folder). You can leave this as it is as long as the full path has no spaces. See the above screenshot for a correct example.

Then, click “Download Tools” and wait for the installation to finish. After everything has been installed, you should see the following window:

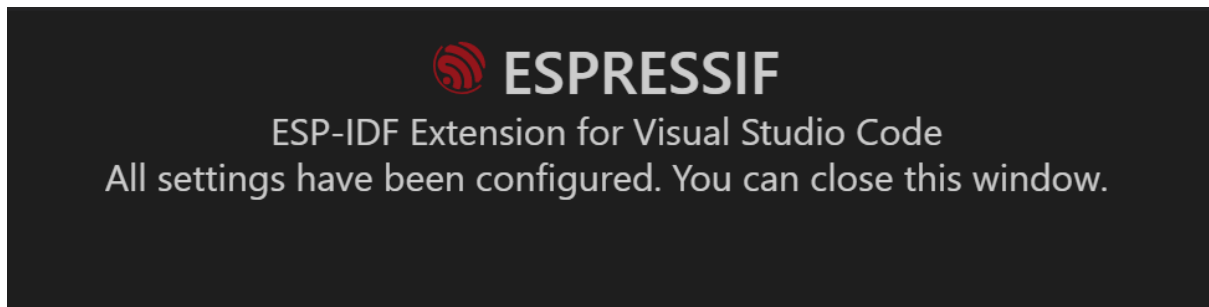


Figure 21. ESP-IDF installation window after finishing the installation process.

Next, create another folder where you will put all your ESP-IDF projects in. You can place this folder anywhere and name it whatever you like, as long as the full path **does not contain whitespaces**. An example is **C:/espressif/workspaces**. Another example is shown below:

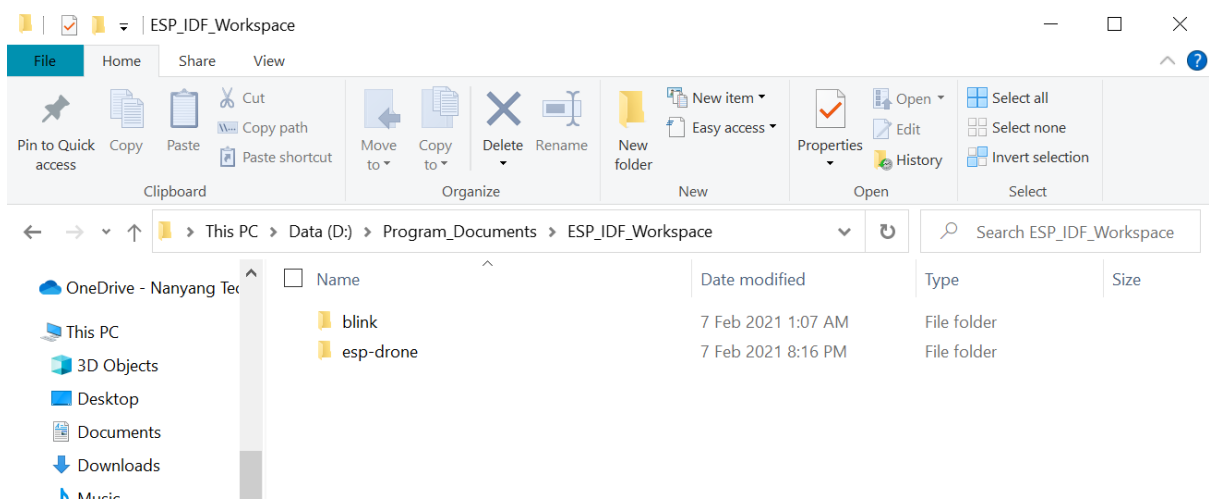


Figure 22. Example of a workspace folder with two projects inside it (the full path here is **D:/Program_Documents/ESP_IDF_Workspace**). Notice that **there are no spaces in the full path**.

Testing the ESP-IDF Installation

To confirm that the ESP-IDF installation works, we will compile a built-in example project in ESP-IDF.

ESP-IDF Example Projects

To open the example project, press “F1” and type “esp-idf examples”, then choose “ESP-IDF: Show Examples Projects” -> “Use current ESP-IDF”.

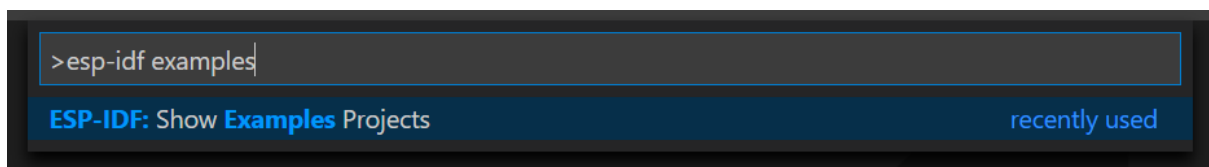


Figure 23. Accessing ESP-IDF example projects.

We will try a simple example: the blinking LED. Find the project `blink` under `get_started` and click “Create project using example blink”.

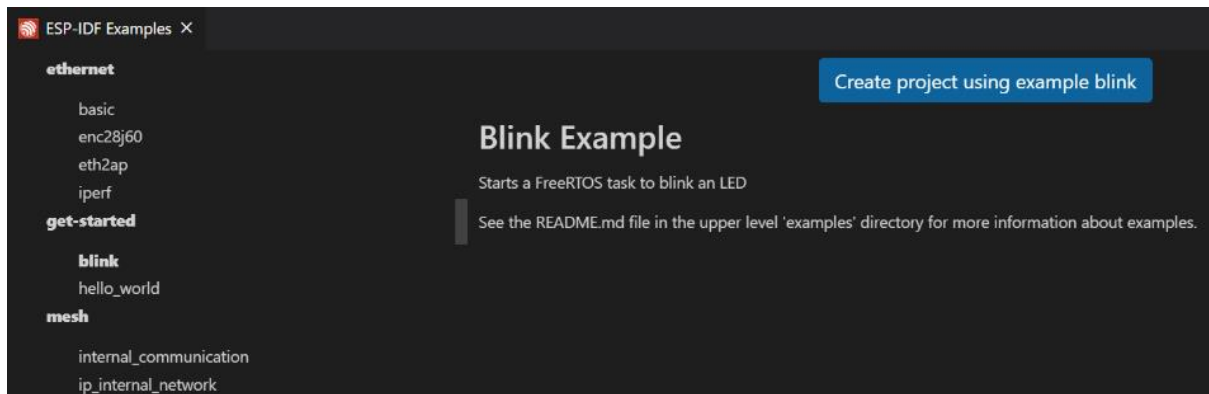


Figure 24. Opening the `blink` example.

You will be asked to choose the folder in which you want to put the new example project in. Choose the folder that you have created previously. You should now see the new example project open automatically.

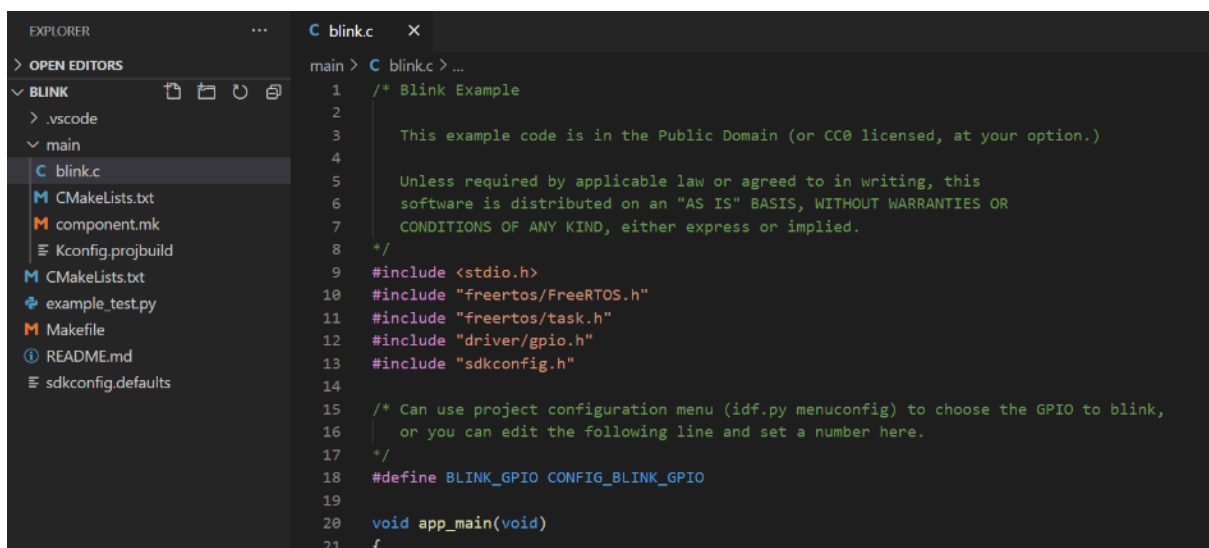


Figure 25. Example project `blink`.

The code here has been written, so we only need to set the configuration and build the project.

ESP-IDF SDK Configuration

Every ESP-IDF project has its own SDK configuration to set many aspects of the ESP32 for the project (including pins that are used in the code). For the `blink` example, the pin which the LED is attached to is configured in the SDK configuration.

To access the SDK configuration editor, click the gear icon on the bottom left of the screen. You should then see the following window:

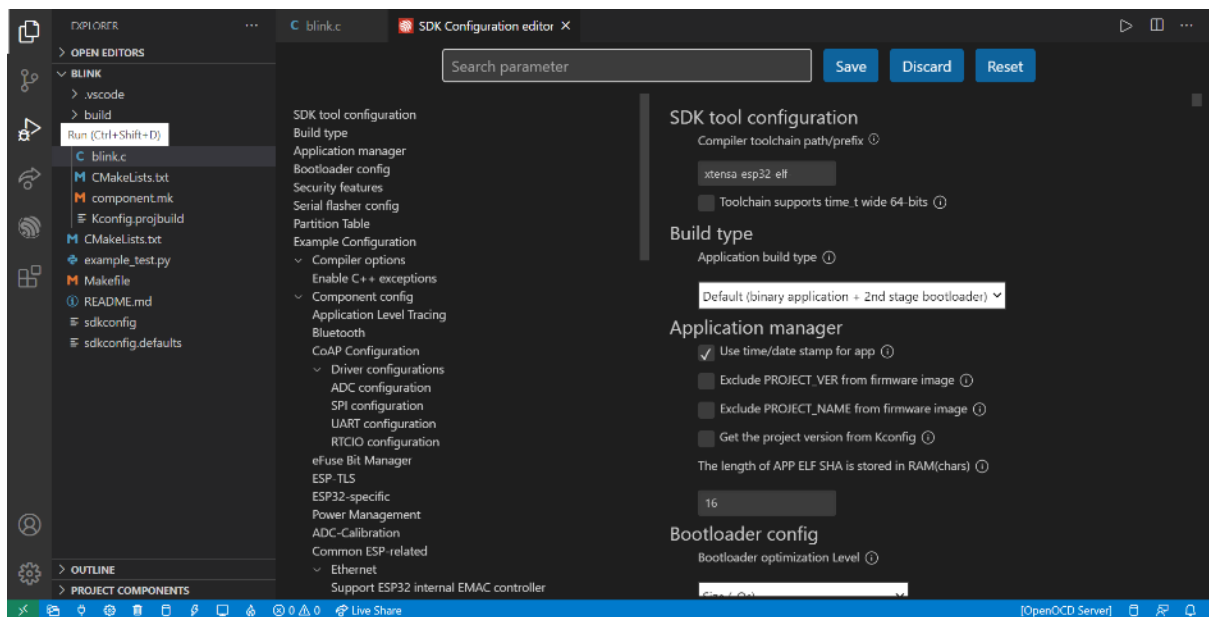


Figure 26. ESP-IDF SDK Configuration editor window.

Scroll down to find “Blink GPIO number” in the configuration. Change it to 33 since the red LED on our drone PCB is on that GPIO pin (you can also use GPIO 13 for the blue LED if you want to). After that, click “Save”. You should see a new `sdkconfig.old` file appear on the left⁹.

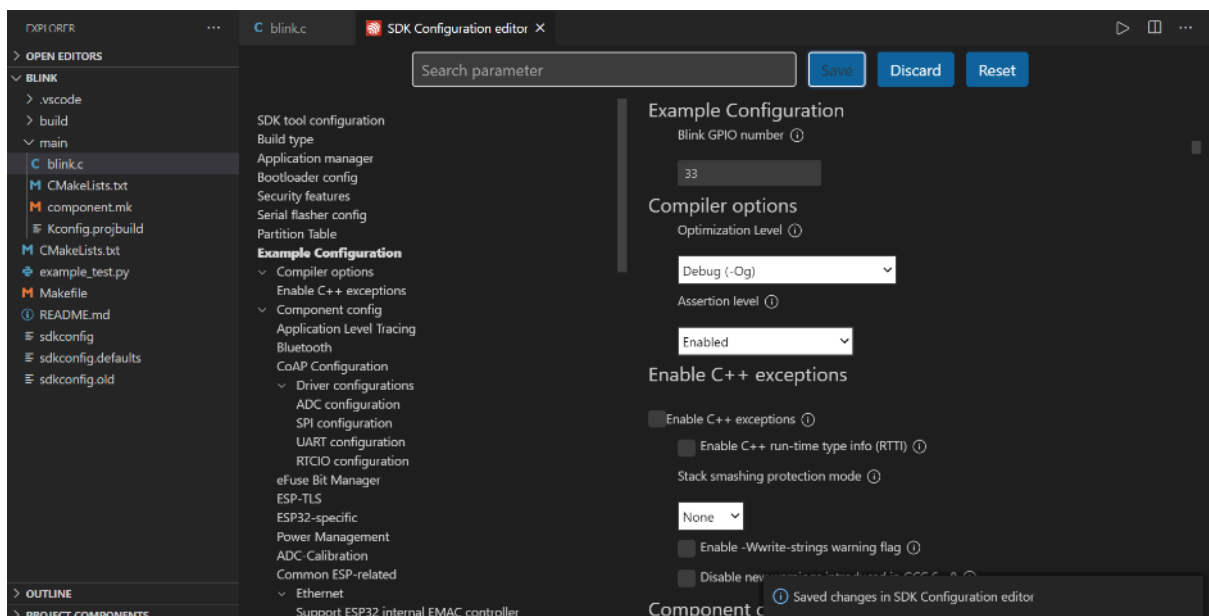


Figure 27. Configure the GPIO pin number to blink to either 33 or 13 in the SDK Configuration editor.

⁹ This `sdkconfig.old` file is actually the former `sdkconfig` file. The `sdkconfig` file contains all the configuration that will be used when compiling the code. When we modify the blink GPIO number and save the configuration, a new `sdkconfig` file is generated, hence the old one is automatically renamed into `sdkconfig.old`.

Building the Project

After modifying and saving the configuration, we can build (compile) the project. To do so, click the cylinder icon on the bottom left of the screen (to the right of the trash can icon). Wait for the build process to finish. A successful build should look like the following:

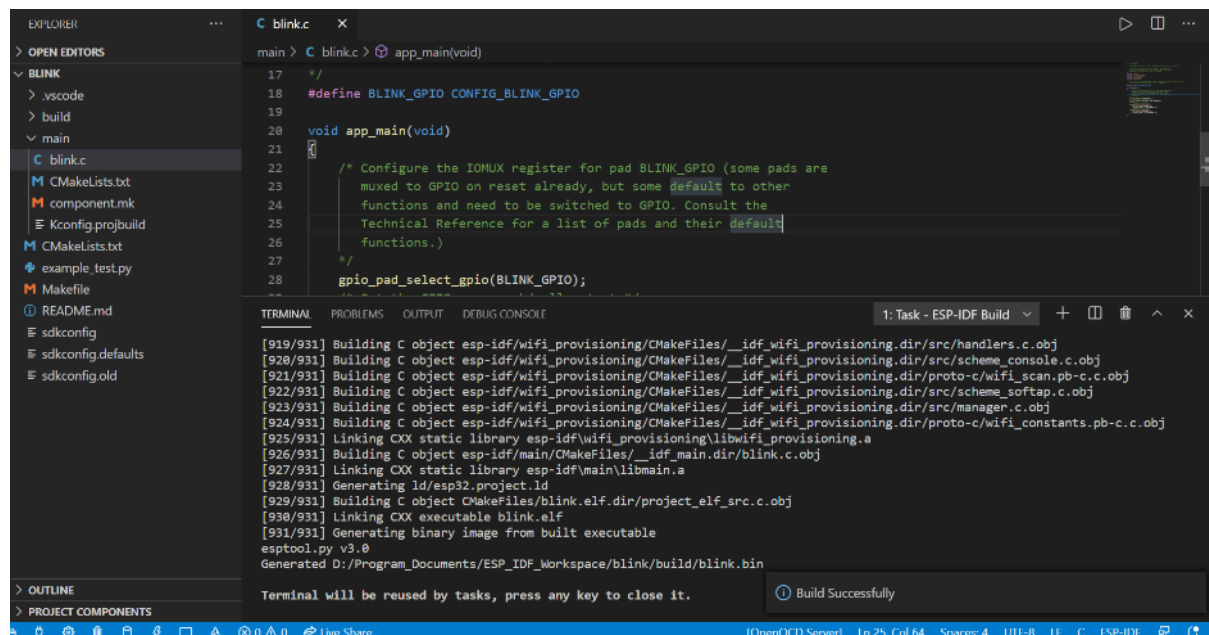


Figure 28. Example of a successful build.

If your build is successful, great! You have installed ESP-IDF correctly. You can proceed to uploading the program to the drone PCB (see **Uploading to the ESP32** below).

While building, you may encounter an error similar to:

```
Invalid escape sequence \o
```

If this happens, do the following modifications:

1. Find the Python script `idf.py` inside `esp-idf/tools` directory. For example, if you chose `C:/espressif` as the directory to install ESP-IDF into, find the Python script inside `C:/espressif/esp-idf/tools`.

Now, inside the Python script, change the code at line 52

```
os.environ["PYTHON"] = sys.executable

into

import pathlib
os.environ["PYTHON"] = pathlib.Path(sys.executable).as_posix()
```

Save the modified script (do not rename or save as).

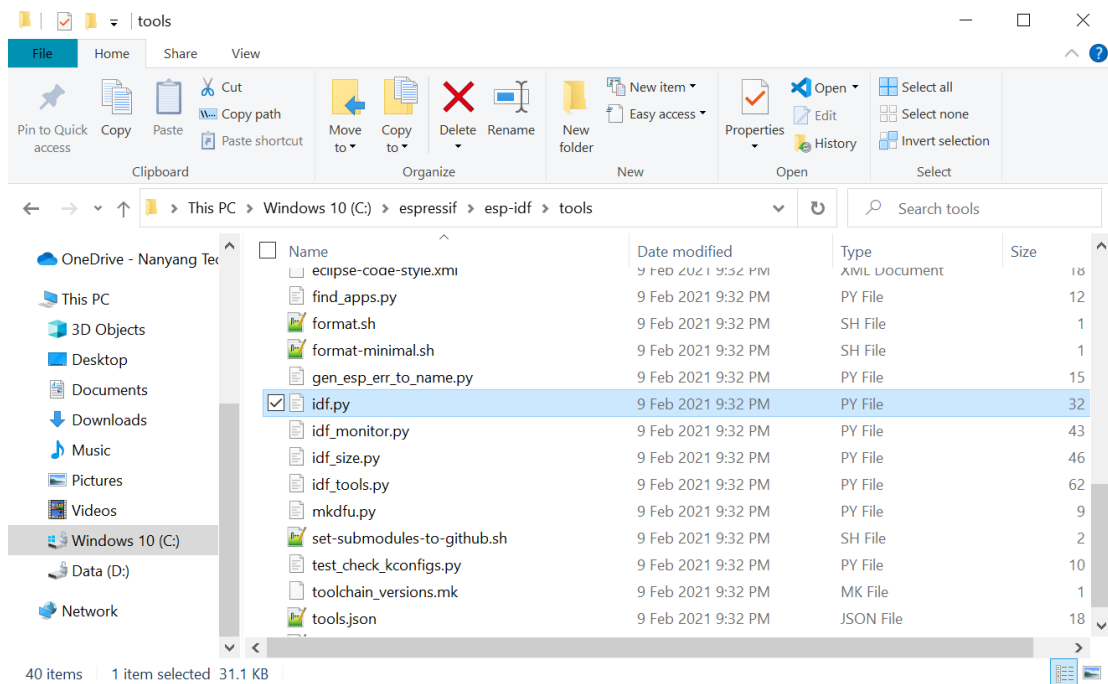


Figure 29. Example of where to find `idf.py` (if you installed ESP-IDF in `C:/espressif`).

2. Open the ESP-IDF extension setting window. To do so, click the gear icon to the right of “Espressif IDF” under Extensions. Then, click “Extension Settings”.

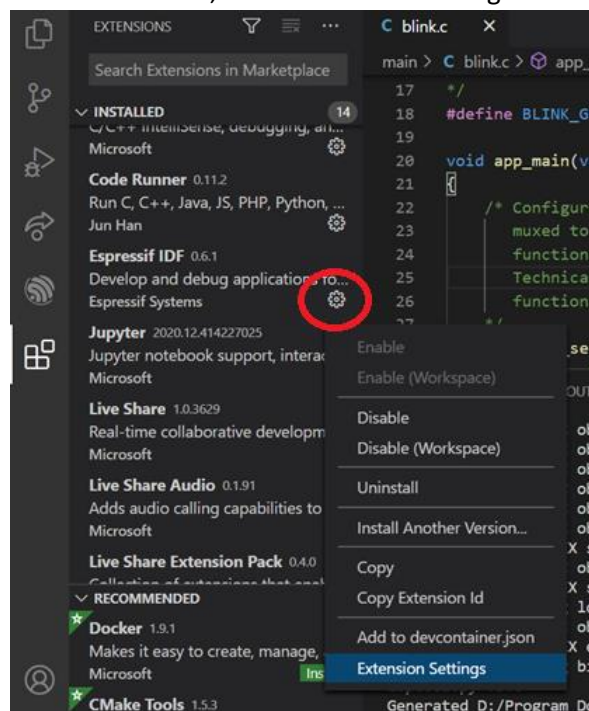


Figure 30. Accessing extension settings.

Then, find “Idf: Python Bin Path Win” and replace all backslashes (\) with forward slashes (/).

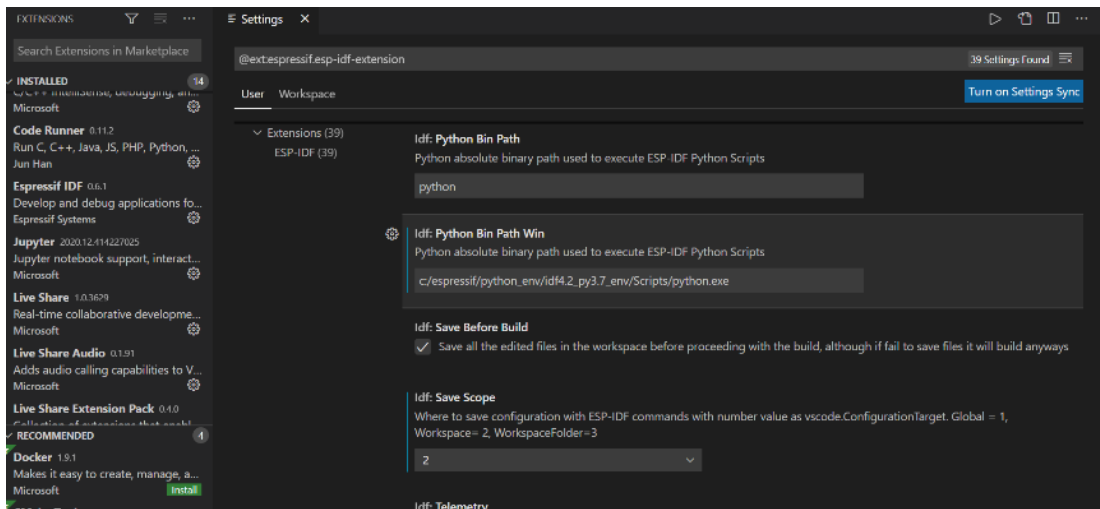


Figure 31. Python Bin Path Win under Extension Settings.

Uploading to the ESP32

Note: before you upload the example blink program to the drone PCB, unplug the camera from the PCB if you still have it plugged in (the camera will not be used now).

To upload the program that we have built to the ESP32 board:

- Plug the ESP32 board to your computer via the Arduino board (the same way as how we have been doing so far, see Figure 12 on page 14). Make sure that the ESP32 has entered bootloader mode (on the drone PCB, this is indicated by the blue LED being lit dimly).
- On VS Code, choose the correct port that the board is connected to by clicking the plug icon on the bottom left of the screen (to the left of the gear icon). A selection list will then appear on the top of the screen:

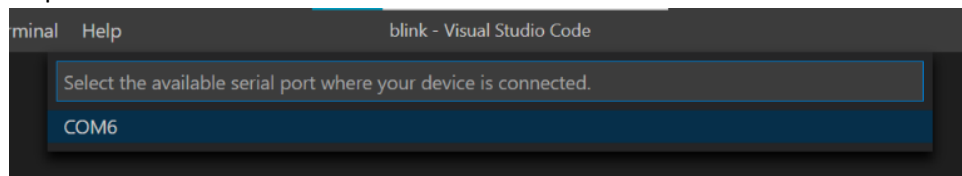


Figure 32. Selecting the port before uploading. One way to check which port is correct is to unplug the drone PCB (the Arduino) from the computer and see which port disappears.

- Finally, upload to the board by clicking the flash icon on the bottom left of the screen (to the left of the monitor icon). A selection list will then appear on the top of the screen. Choose “UART”.

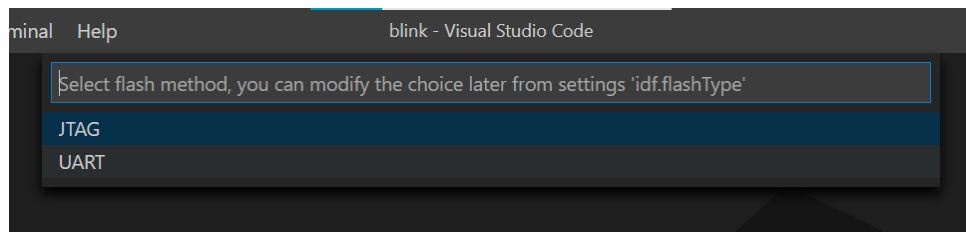
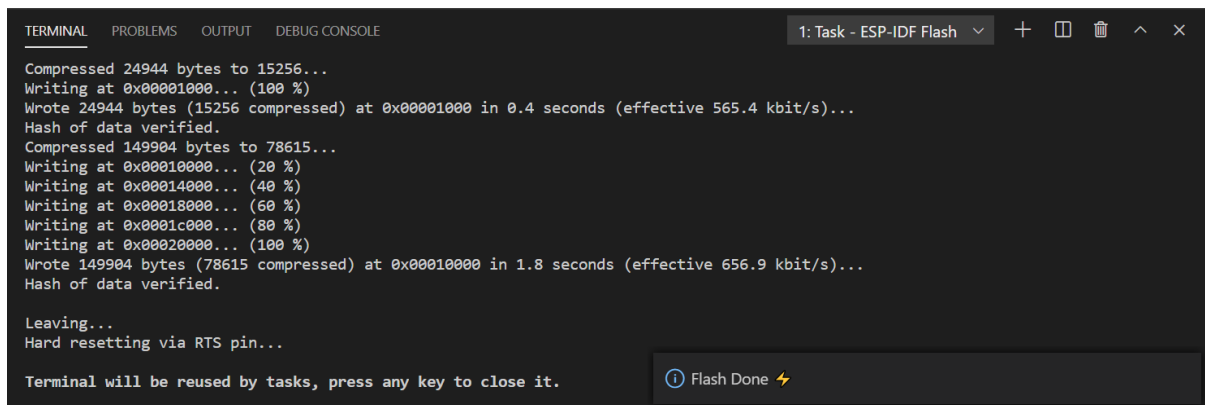


Figure 33. Uploading the program to the ESP32. Choose “UART” (which is the serial communication protocol we use to upload the program to the board).

A successful upload should look like this:



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  1: Task - ESP-IDF Flash
Compressed 24944 bytes to 15256...
Writing at 0x00001000... (100 %)
Wrote 24944 bytes (15256 compressed) at 0x00001000 in 0.4 seconds (effective 565.4 kbit/s)...
Hash of data verified.
Compressed 149904 bytes to 78615...
Writing at 0x00001000... (20 %)
Writing at 0x00001400... (40 %)
Writing at 0x00001800... (60 %)
Writing at 0x00001c00... (80 %)
Writing at 0x00002000... (100 %)
Wrote 149904 bytes (78615 compressed) at 0x00001000 in 1.8 seconds (effective 656.9 kbit/s)...
Hash of data verified.

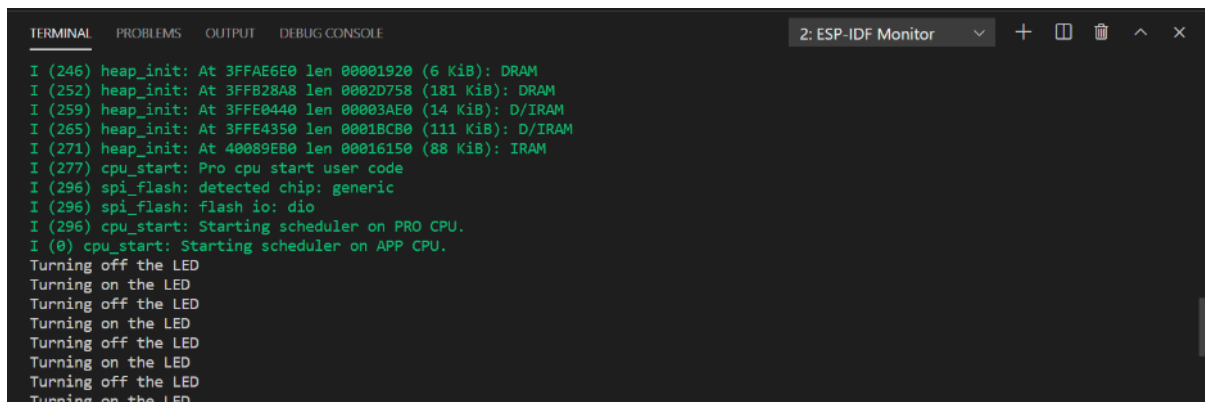
Leaving...
Hard resetting via RTS pin...

Terminal will be reused by tasks, press any key to close it.  Flash Done ⚡
```

Figure 34. Terminal output after a successful upload to the ESP32 board.

Now, you can unplug the cable going from Arduino GND to D0 of the drone PCB. Keep all the other cables connected, including the TX and RX wires. Finally, reset the PCB (e.g. by unplugging the USB cable and plugging again) to bring the ESP32 out of bootloader mode.

You can now see the LED blinking! Additionally, you can open the serial monitor by clicking the monitor icon on the bottom left of the screen to see what the ESP32 sends via serial communication. For this blink example, you will see the following:



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  2: ESP-IDF Monitor
I (246) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (252) heap_init: At 3FFB28A8 len 0002D758 (181 KiB): DRAM
I (259) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (265) heap_init: At 3FFE4350 len 00018CB0 (111 KiB): D/IRAM
I (271) heap_init: At 400809E0 len 00016150 (88 KiB): IRAM
I (277) cpu_start: Pro cpu start user code
I (296) spi_flash: detected chip: generic
I (296) spi_flash: flash io: dio
I (296) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
Turning off the LED
Turning on the LED
Turning off the LED
Turning on the LED
Turning off the LED
Turning on the LED
Turning off the LED
Turning on the LED
Turning off the LED
Turning on the LED
```

Figure 35. Serial monitor output for the blink example project.

Working with the Drone PCB and esp-drone-nh Project

In the following weeks, we will use the open source `esp-drone` project for development with the drone PCB. We have put the repository for this project on Github:

<https://github.com/NelsenEW/EEE-DIP-UAVONICS/tree/main/esp-drone-nh>

You can put this inside the folder that you have previously created.

To use this project, two project link scripts must be modified.

1. Go to “C:\espressif\esp-idf\components\esp32\ld” and find the script `esp32.project.ld.in`.

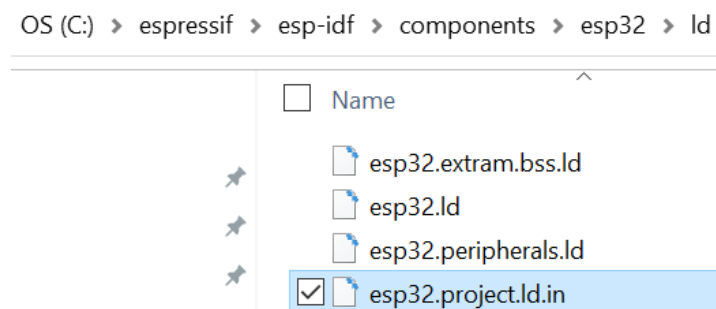


Figure 36. ESP32 project link script.

Right click on the script and click “Open with Code” to open the script. Then, add the following lines before `} >default_rodata_seg`:

```
/* Parameters and log system data */
_param_start = .;
KEEP*(.param)
KEEP*(.param.*)
_param_stop = .;
. = ALIGN(4);
_log_start = .;
KEEP*(.log)
KEEP*(.log.*)
_log_stop = .;
. = ALIGN(4);
```

The newly added lines (highlighted in green) should look as follow:

```
311     _thread_local_end = ABSOLUTE(.);
312     . = ALIGN(4);
313     /* Parameters and log system data */
314     _param_start = .;
315     KEEP*(.param)
316     KEEP*(.param.*)
317     _param_stop = .;
318     . = ALIGN(4);
319     _log_start = .;
320     KEEP*(.log)
321     KEEP*(.log.*)
322     _log_stop = .;
323     . = ALIGN(4);
324 } >default_rodata_seg
325
326 .flash.text :
```

Figure 37. ESP32 project link script modification.

2. Similarly, go to "C:\espressif\esp-idf\components\esp32s2\ld" and find the script esp32s2.project.ld.in.

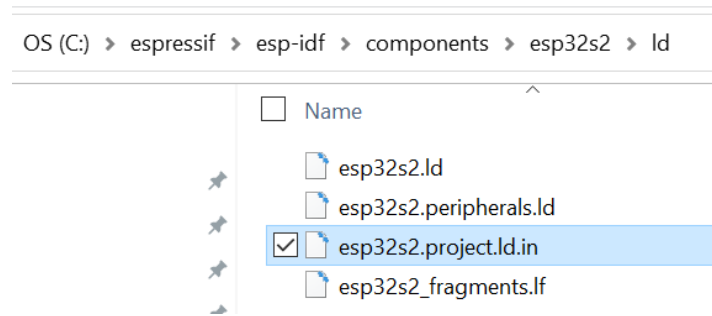


Figure 38. ESP32S2 Project link script.

Add the same lines to the script at the same location:

```
296     _rodata_reserved_end = ABSOLUTE(.);
297     . = ALIGN(4);
298     /* Parameters and log system data */
299     _param_start = .;
300     KEEP(*(.param))
301     KEEP(*(.param.*))
302     _param_stop = .;
303     . = ALIGN(4);
304     _log_start = .;
305     KEEP(*(.log))
306     KEEP(*(.log.*))
307     _log_stop = .;
308     . = ALIGN(4);
309 } >default_rodata_seg
310
311 .flash.text :
```

Figure 39. ESP32S2 Project link script modification.

Testing the Workspace

Once you have downloaded the esp-drone-nh project, open VS Code and click "Open Workspace..." on the menu bar. Select the file "workspace.code-workspace" inside the esp-drone-nh directory. The result should look like the following:

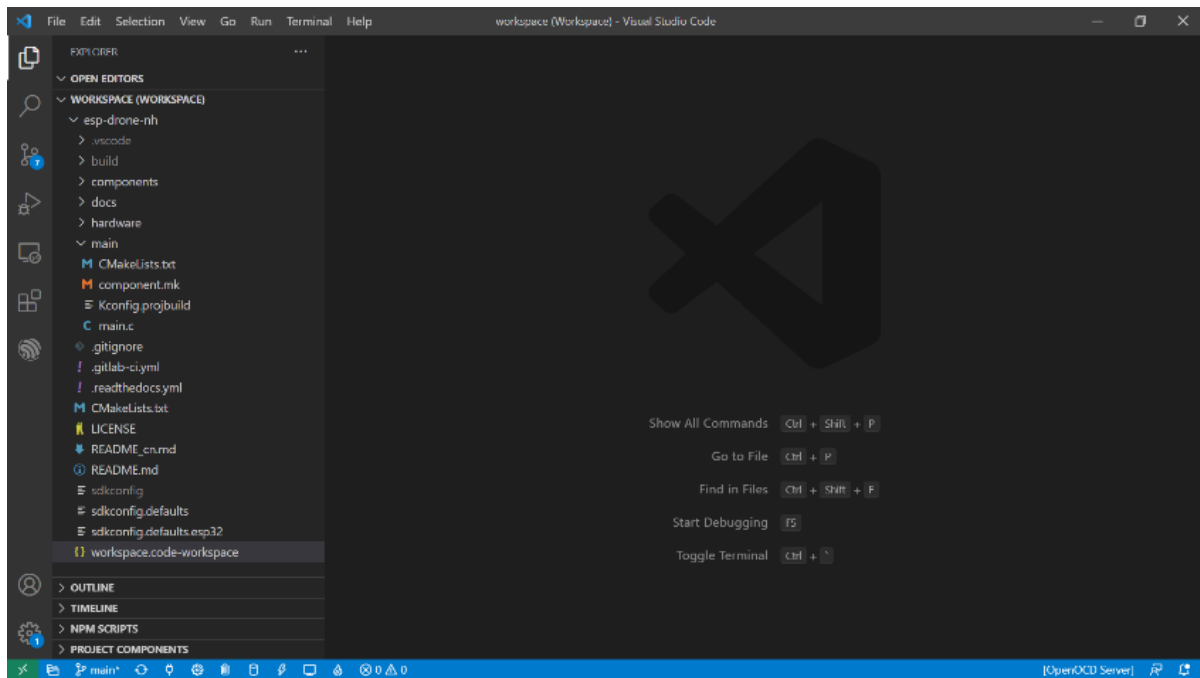


Figure 40. Visual Studio Code with `esp-drone-nh` workspace.

To test the project, we can try building the workspace by clicking the cylinder icon on the bottom left of the screen (to the right of the trash can icon). Wait for the build process to finish. A successful build should look like the following:

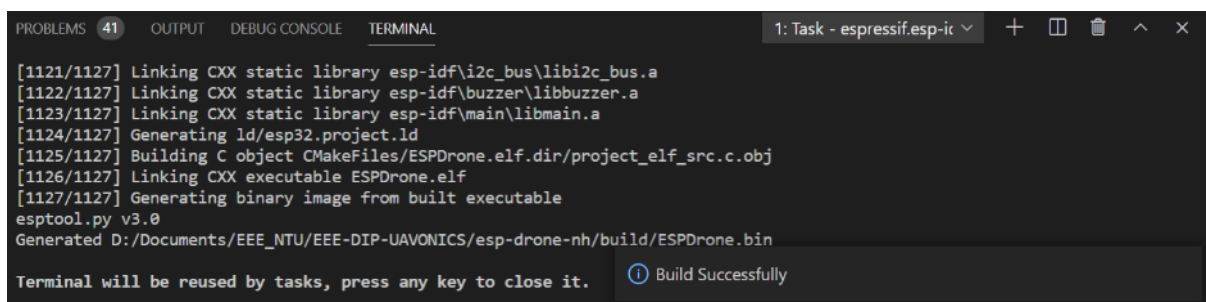


Figure 41. `esp-drone-nh` successfully built.

For now, **do not upload** this code to the drone PCB as we have not set the SDK configuration yet. This is just for confirming that the project can be built successfully on your computer.

For future developments with the `esp-drone` project, you can refer to the `esp-drone` documentation:

<https://github.com/NelsenEW/EEE-DIP-UAVONICS/blob/main/esp-drone-nh/docs/esp-drone-docs.pdf>