Let's create a **full, step-by-step guide to building a basic real-world quadcopter drone using ESP32**. I'll include **pin connections, electronics setup, code structure, and explanations**. This will be **beginner-friendly**, yet close to what real flight controllers do.

Step 1 Decide Your Drone Type

We'll make a **basic quadcopter (X configuration)**:

- 4 brushless motors
- 4 ESCs
- 1 LiPo battery
- ESP32 DevKit as the flight controller
- MPU6050 for orientation

(**MPU-6050 3-Axis Accelerometer and Gyro Sensor**)

Advantages of this type: stable, easy to control, minimal hardware.

Step 2 List All Components

Electronics:

Component	Purpose
ESP32 DevKit	Flight controller
MPU6050	Gyro + accelerometer for orientation
4 BLDC motors	Propulsion
4 ESCs	Motor speed control
4 propellers	Matched to motors
LiPo battery (3S/4S)	Power source
Optional: buzzer / LEDs	Status / debugging

Mechanical:

Component	Purpose
Quadcopter frame	X configuration

Component	Purpose
Motor mounts / screws	Fix motors
Battery strap	Secure battery

Tools:

- Soldering iron
- Multimeter
- Screwdriver

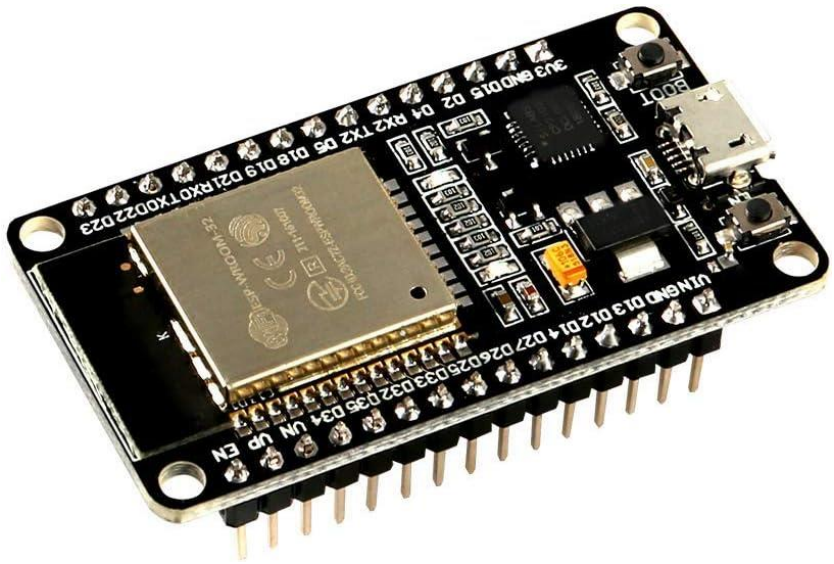
Step 3 Assemble the Frame

1. Mount the **4 motors** at the ends of the quadcopter frame.

2. Attach **propellers** only after testing motors (safety!).

3. Mount **ESP32** in the center of the frame (vibration-resistant).

ESP-WROOM-32



4. Mount the **battery** with straps.

 Tip: Keep **wires short** and avoid loose cables near motors.

Step 4 Wire Electronics

A) ESC to Motors

- Each ESC has **3 motor wires** → motor
- ESC power wires → LiPo battery
- ESC ground → common GND

B) ESC signal to ESP32

- ESC signal wire → ESP32 GPIO (PWM capable)
- Common GND between ESCs and ESP32

Motor Position	ESC Signal Pin	ESP32 GPIO
Front Right	Motor1_ESC	32
Front Left	Motor2_ESC	33
Rear Left	Motor3_ESC	25
Rear Right	Motor4_ESC	26

C) MPU6050 (I²C) Connections

MPU6050 Pin	ESP32 Pin	Notes
VCC	3.3V	Power
GND	GND	Common ground
SDA	21	Data line for I²C
SCL	22	Clock line for I²C

D) Optional Sensors

- BMP280 → I²C (same bus as MPU6050, use different address)
- GPS Neo6M → UART pins
- Buzzer → GPIO for status

Step 5 Test Components Individually

1. **ESC + motor test**
   - Power ESCs with LiPo battery
   - Send PWM signal from ESP32 (1000–2000 µs)
   - Verify motors spin correctly
2. **MPU6050 test**
   - Use Arduino code to read **ax, ay, az, gx, gy, gz**
   - Verify that tilt changes readings

Step 6 ESP32 Flight Controller Code

Libraries Needed

- `Wire.h` → for I²C
- `Servo.h` → for PWM to ESCs
- `MPU6050.h` → for sensor

Note: For real hardware, `Servo.h` works with ESP32 ESCs. If not, you can use **`ledcWrite()`**.

Basic Structure

```
#include <Wire.h>
#include <MPU6050.h>
#include <Servo.h>

MPU6050 mpu;

// Motors
Servo motor1, motor2, motor3, motor4;

// PID constants
float Kp=0.6, Ki=0.0, Kd=0.05;
```

```
// Target orientation
float targetRoll=0, targetPitch=0;

// PID variables
float lastRollError=0, rollIntegral=0;
float lastPitchError=0, pitchIntegral=0;

void setup() {
  Serial.begin(115200);
  Wire.begin(21,22); // MPU6050 I2C

  // Initialize sensor
  mpu.initialize();
  if (mpu.testConnection()) Serial.println("MPU6050 connected!");

  // Attach ESCs
  motor1.attach(32);
  motor2.attach(33);
  motor3.attach(25);
  motor4.attach(26);

  // Initialize motors
  int throttle=1000;
  motor1.writeMicroseconds(throttle);
  motor2.writeMicroseconds(throttle);
  motor3.writeMicroseconds(throttle);
  motor4.writeMicroseconds(throttle);
}

void loop() {
  // Read MPU6050
  int16_t ax,ay,az,gx,gy,gz;
  mpu.getMotion6(&ax,&ay,&az,&gx,&gy,&gz);

  float roll=gx/131.0;   // degrees/sec
  float pitch=gy/131.0;

  // PID Roll
  float rollError = targetRoll - roll;
  rollIntegral += rollError;
  float rollDerivative = rollError - lastRollError;
  float rollOutput = Kp*rollError + Ki*rollIntegral + Kd*rollDerivative;
  lastRollError=rollError;

  // PID Pitch
  float pitchError = targetPitch - pitch;
  pitchIntegral += pitchError;
  float pitchDerivative = pitchError - lastPitchError;
  float pitchOutput = Kp*pitchError + Ki*pitchIntegral + Kd*pitchDerivative;
  lastPitchError=pitchError;

  // Motor adjustments
  int throttle=1200;
  motor1.writeMicroseconds(throttle + rollOutput - pitchOutput);
  motor2.writeMicroseconds(throttle - rollOutput - pitchOutput);
  motor3.writeMicroseconds(throttle - rollOutput + pitchOutput);
```

```
  motor4.writeMicroseconds(throttle + rollOutput + pitchOutput);

  // Debug
  Serial.print("Roll:"); Serial.print(roll);
  Serial.print(" Pitch:"); Serial.println(pitch);

  delay(100);
}
```

Step 7 Calibrate Sensors and Motors

1. **MPU6050**: Ensure zero offsets
2. **ESCs**: Calibrate throttle range (1000–2000 µs)
3. Verify **motor direction** (swap two wires if wrong)

Step 8 First Flight Test

1. Remove propellers → dry run, test motor signals
2. Install propellers → use **prop guards**
3. Start with **low throttle**
4. Gradually adjust **Kp/Ki/Kd** for stable hover

Step 9 Optional Features

- Altitude hold (BMP280)
- GPS waypoint navigation
- WiFi/Bluetooth remote control
- Telemetry logging

Step 10 Safety Tips

- Always test **without props first**
- Use **low battery throttle** on first flights

- Keep **common ground** between ESP32, ESC, and battery
- Incrementally tune PID

For building a **real quadcopter drone with ESP32**, you want a board that:

✔ has **enough GPIOs** for motors + sensors
✔ supports **PWM output**
✔ provides standard **power pins (3.3V, GND)**
✔ is easy to **program with Arduino/PlatformIO**

□ Best ESP32 Boards for a Quadcopter Controller

□ 1. ESP32 DevKit V1 (WROOM / WROVER)

- Most commonly used for maker projects
- Easy to program via USB
- Plenty of GPIOs for sensors + motors
- Supports PWM for ESCs

Why choose this
✔ Simple USB programming
✔ Works with most Arduino libraries
✔ Lots of online examples

□ This is the *best choice* for a beginner drone controller.

□ 2. ESP32 WROVER Module on Development Board

Same as above but includes:

- Extra **PSRAM memory**
- Good for sensor fusion or logging

Use case

✔ Larger programs
✔ More data buffering
✔ Advanced filtering algorithms