# The Useless Box
# ISTEM 2025

Assistant Professor Timothy S Leishman

College of Technology

Idaho State University

ISTEM 2025

May 19, 2025

# The Useless Box
# ISTEM 2025

**Contents**

**Lab Safety** **6**

**Servo Project** **6**

Servo Introduction . . . . . 6

Servo Specifications . . . . . 7

Initial Servo Test Steps: . . . . . 9

555 Timer . . . . . 9

**The Useless Box Project** **11**

Introduction . . . . . 11

Useless Box Video . . . . . 11

DigiKey Parts List . . . . . 11

Amazon Parts List . . . . . 11

3D Printer Files . . . . . 11

Useless Box Schematic . . . . . 11

Circuit Description . . . . . 13

Circuit Calculations . . . . . 13

PCB Assembly . . . . . 14

Useless Box Assembly . . . . . 19

**ESP32 Getting Started** **21**

Additional ESP32 Course Recommendations: . . . . . 21

ESP32 Introduction . . . . . 21

ESP32 (WROOM-32) Dev Kit Specifications: . . . . . 23

ESP32 Functional Block Diagram . . . . . 23

Data Sheet for ESP-WROOM-32 (IC) . . . . . 23

**Arduino IDE and the ESP32** **24**

Arduino IDE Installation . . . . . 24

Arduino IDE setup for ESP32 . . . . . 24

CP210X USB to UART Driver Install . . . . . 27

**Arduino IDE ESP32 Board and Comm Port Verification** **29**

**ESP32 Testing** **30**

My First Sketch - Blink . . . . . 30

External LED and GPIO Testing . . . . . 32

GPIO Test Code . . . . . 33

Pulse Width Modulation (PWM) . . . . . 34

LED PWM . . . . . 35

RGB LED . . . . . 36

Stop Light . . . . . 36

Color Spectrum Change . . . . . 37

Digital Read - Input Button . . . . . 38

Analog Read with Serial Monitor . . . . . 39

Map Function . . . . . 42

GPIO Interrupt . . . . . 44

Touch . . . . . 46

Touch Interrupt . . . . . 48

Photo-Resistor Sensor . . . . . 49

DHT11 Temperature and Humdity Sensor . . . . . 50

$I^2C$ (Inter-Integrated Circuit) . . . . . 52

MPU6050 3 Axis Accelerometer Gyroscope . . . . . 53

OLED . . . . . 54

Adafruit SSD1306 Test Sketch . . . 54
Hello World . . . 55
Hexadecimal . . . 56
Converting Decimal to Hexadecimal . . . 56
Serial Monitor Read . . . 57
Serial Monitor Read and Display on OLED . . . 58
MPU6050, LED, and OLED . . . 58
Servo Motor . . . 59
Servo Motor Sweep . . . 59
ESP32 Servo Library . . . 60
Servo Motor Knob . . . 61
Brushed DC Motor . . . 62
WIFI . . . 63
WIFI Scanner . . . 63
WIFI Connect . . . 63
WIFI Simple Time . . . 64
WIFI Additional Information . . . 65
Bluetooth Classic . . . 65
Serial to Serial Bluetooth . . . 65
Bluetooth OLED . . . 66
Bluetooth OLED RGB . . . 67
Bluetooth OLED Temperature and Humidity . . . 68
Functions . . . 69
What are Functions? . . . 69
Bluetooth OLED RGB Temperature and Humidity with Functions . . . 69
BLE Bluetooth Low Energy . . . 69
BLE GATT . . . 70

BLE Write . . . . . 70
MIT App Inventor . . . . . 71
Bluepad32 . . . . . 71

## Lab Safety

Working in an electronics science lab is exciting, but it's important to follow safety rules to keep everyone safe. Here are some key guidelines to remember every time you step into the lab!

1. Safety Glasses: Always wear safety glasses while in the lab. Safety glasses will protect eyes from flying objects.
2. No loose jewelry or open-toed shoes.
3. No food or drink in the lab!
4. 4. Be Aware of Your Surroundings
    - Stay focused: Don't horse around or get distracted in the lab. Accidents can happen when you're not paying attention.
    - Know the emergency exits: Always know how to get out of the lab quickly in case of an emergency.
    - Be aware of where the fire extinguisher, eye wash station, and first aid kit are located.

## Servo Project

### Servo Introduction

**What is a Servo Motor?** A servo motor is a special kind of motor that can move to a specific position, like turning to a certain angle or lifting an object to just the right height. It doesn't just spin around like regular motors but follows specific commands to stop exactly where you want it.

**How Does a Servo Work?**

- Inside a servo motor, there's a small motor, gears, and a circuit.
- It works by receiving a signal (a command) to move to a specific angle (for example, 90 degrees).
- The gears inside help to slow down or speed up the motor, so it moves accurately.

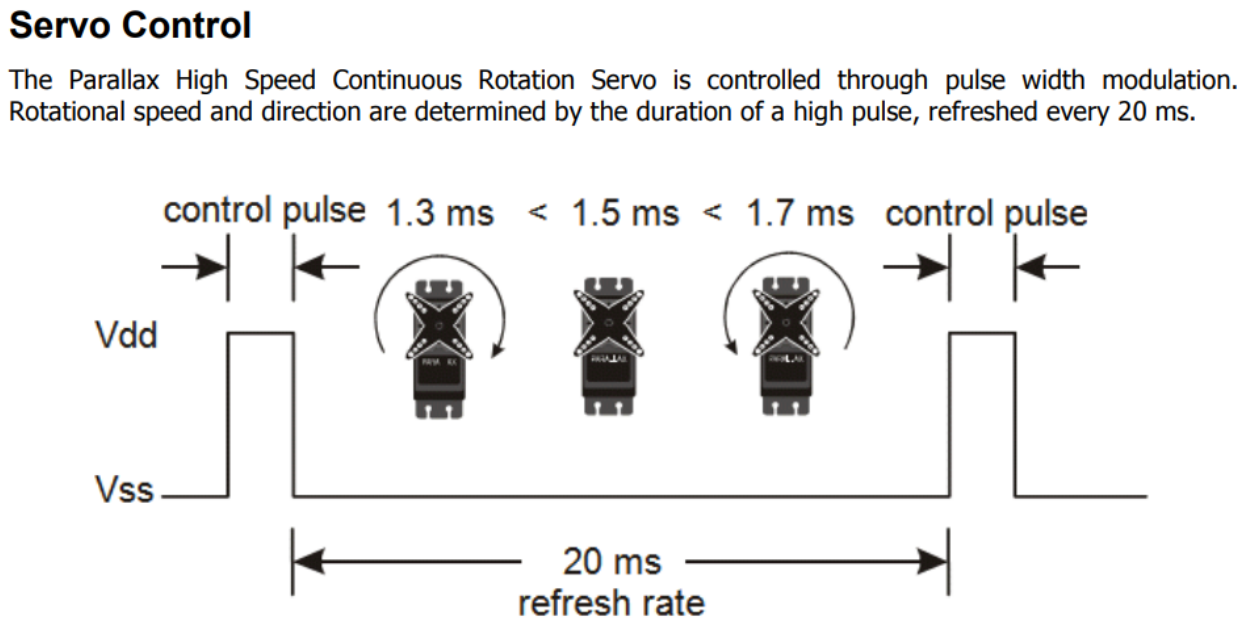
- It uses feedback to know exactly where it is, so it can adjust itself if needed.

**Where Do You Find Servo Motors?**

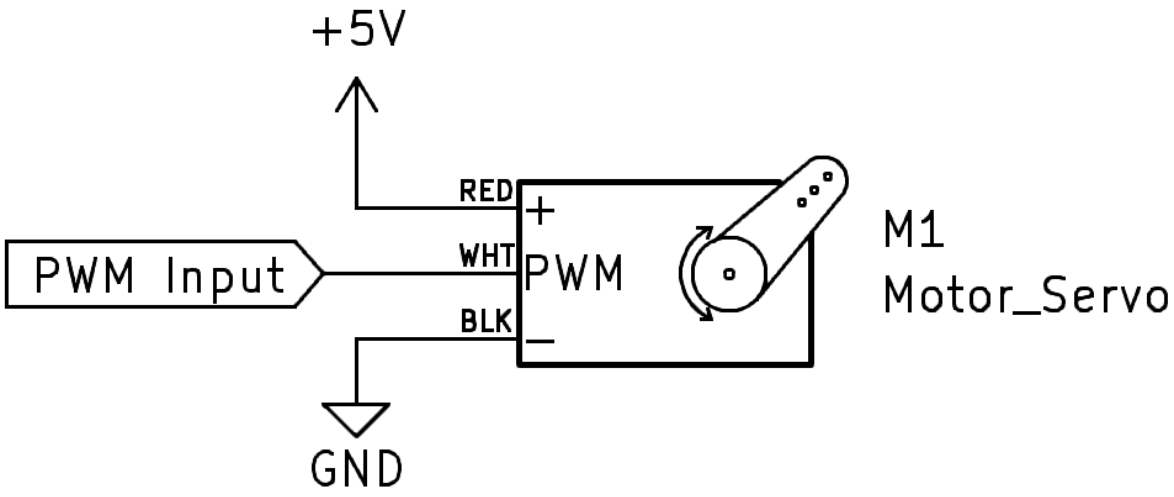
- Robots: Servos help robots move their arms, legs, or head with great precision.
- RC Cars/Airplanes: They control the wheels, flaps, or other parts of remote-controlled vehicles.
- Manufacturing Machines: In factories, servos are used in machines that need precise movements for making products.

**Servo Specifications**

The servo is controlled using a standardized signal. Parallax manufactures servos. The Parallax High-Speed Continuous Rotation Servo (900-00025) data sheet details the standard servo control signals.




*Servo Control Signal*



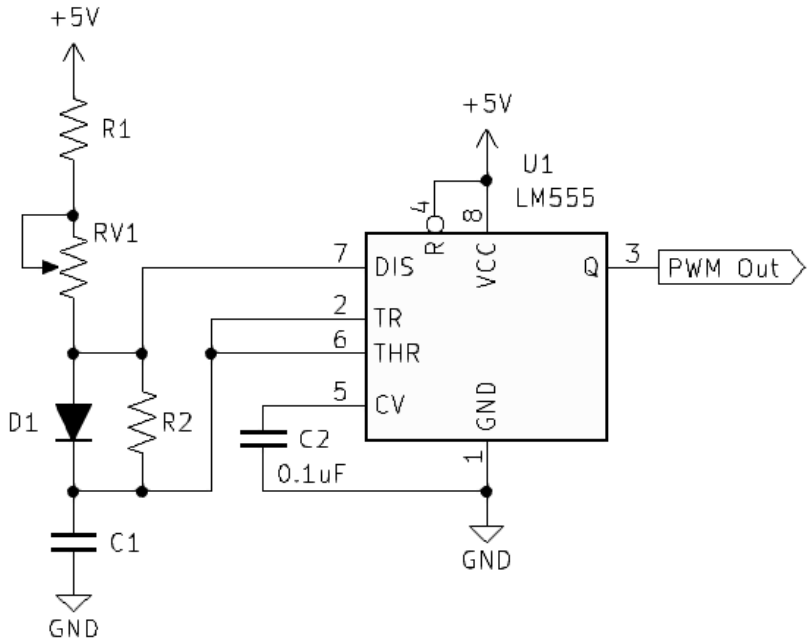

*Basic Servo Schematic*

Download Parallax High-Speed Continuous Rotation Servo (900-00025) Data Sheet

**Initial Servo Test Steps:**

1. Use an Oscilloscope to measure and set a DC power supply output to 5VDC.
2. Setup a Function Generator to produce a 0 to 5v, pulse waveform with a positive pulse of 1.5ms and a 20ms off time ($\approx 50hz$). Use an Oscilloscope to verify the signal.
3. Once the 5VDC and the 0 to 5V pulse waveform has been verified, the servo can be connected as follows.
    - Red to 5VDC
    - Black (sometimes Brown) to Reference
    - White (sometimes yellow) to Function Generator (0 to 5V pulse waveform)
    - Incrementally adjust the pulse width of the waveform from 1.5ms to 2.0ms and back down to 1.0ms and observe the Servo motion.

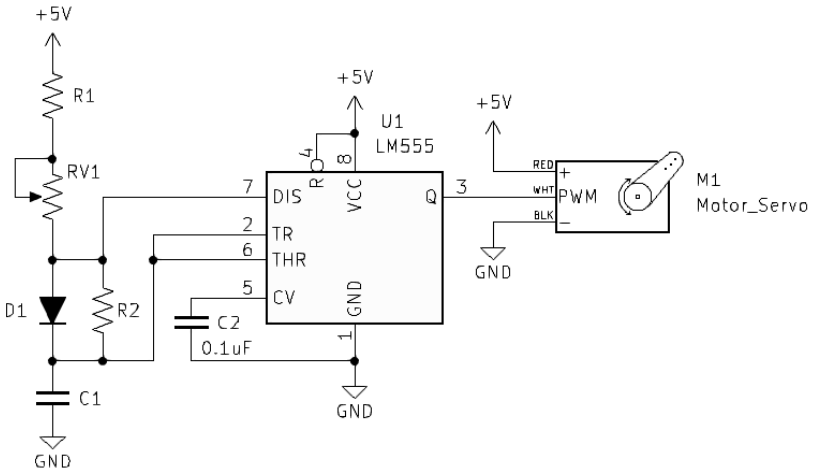
**555 Timer**




*555 Timer Test Circuit*

4. Verify the PWM Out waveform using an Oscilloscope.

    - $Time_{On} \approx 0.693(R1 + RV1)(C1)$
    - $Time_{Off} \approx 0.693(R2)(C1)$
    - Calculate circuit C1, R1, RV1, and R2.
    - Build the circuit, adjust RV1 and observe the pulse width will vary from 1.3ms to 1.7ms.
    - Verify that the pulse space is approximately 20ms.
    - Verify that the waveform voltage is 0 to 5V.




*555 Timer Servo Circuit*

5. Test the final circuit with the servo motor.

# The Useless Box Project

## Introduction

The Useless Box Project is an engaging and interactive electronics build that combines mechanical motion with custom circuitry. This project features a servo motor, which drives the box's signature self-switching mechanism, housed in a 3D-printed enclosure. It is powered by a USB-C rechargeable battery, providing a modern and convenient power source. At its core, the project utilizes a custom-designed PCB, integrating essential components such as a 555 timer for control logic, ensuring smooth and reliable operation.

This project serves as an excellent educational tool, introducing students to electronics, CAD design, PCB development, and embedded systems in a fun and interactive way. By building the Useless Box, students gain hands-on experience with electronic circuit design, and mechanical assembly, making it a valuable addition to any STEM curriculum or robotics program.

## Useless Box Video

- Youtube Useless Box Video

## DigiKey Parts List

- DigiKey Parts List

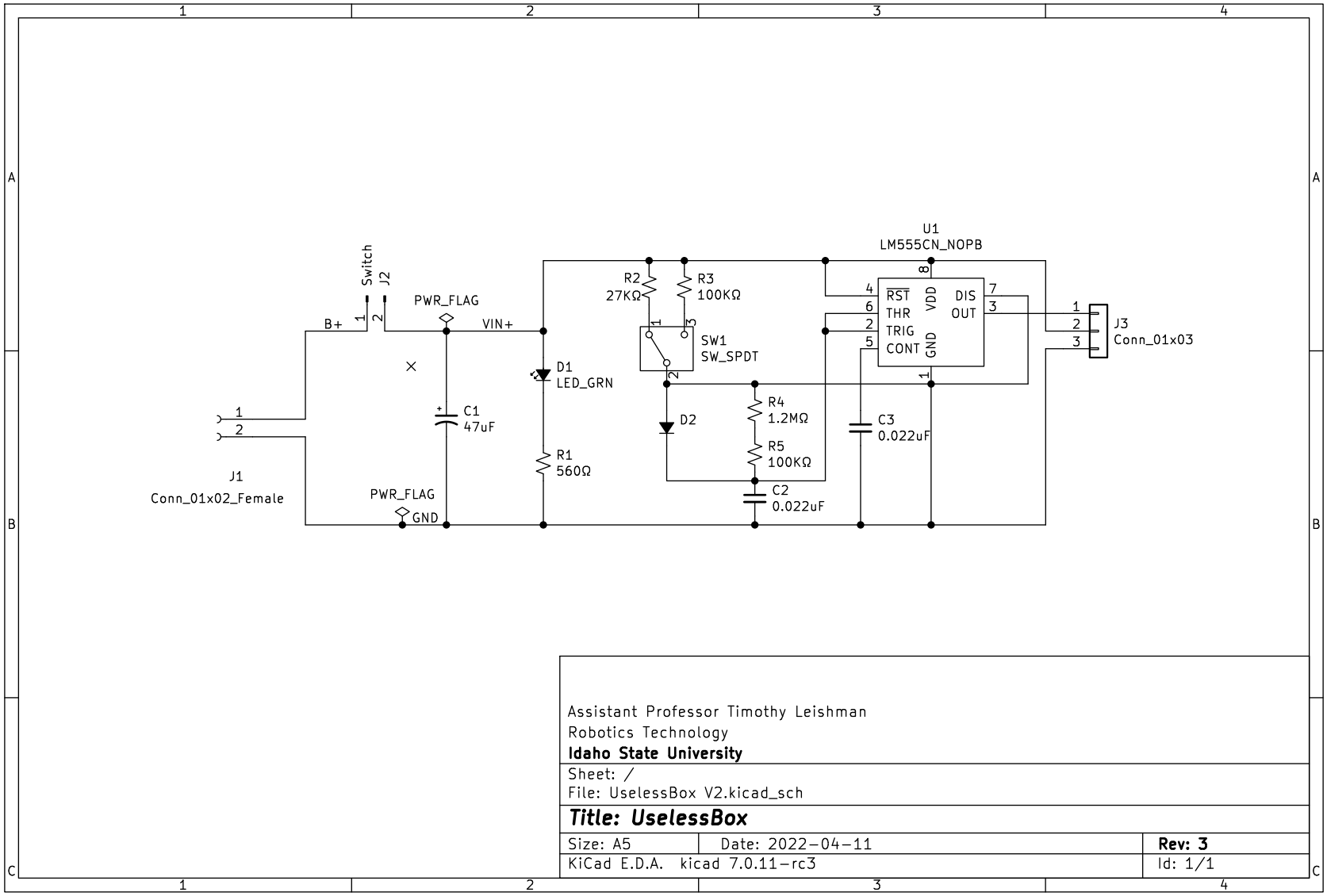
## Amazon Parts List

- Amazon Parts List

## 3D Printer Files

- Useless Box 3D Printer Files

## Useless Box Schematic


U1
LM555CN_NOPB
Switch
J2
PWR_FLAG
B+
VIN+
R2
27KΩ
R3
100KΩ
SW1
SW_SPDT
D1
LED_GRN
C1
47uF
R1
560Ω
D2
R4
1.2MΩ
R5
100KΩ
C2
0.022uF
C3
0.022uF
RST
THR
TRIG
CONT
VDD
GND
DIS
OUT
J3
Conn_01x03
J1
Conn_01x02_Female
PWR_FLAG
GND
Assistant Professor Timothy Leishman
Robotics Technology
Idaho State University
Sheet: /
File: UselessBox V2.kicad_sch
Title: UselessBox
Size: A5
Date: 2022-04-11
Rev: 3
KiCad E.D.A. kicad 7.0.11-rc3
Id: 1/1

**Circuit Description**

- J1 - Connects 1S LiPo battery voltage to circuit $\approx 3.7V_{DC}$
- J2 - Power switch connection
- C1 - Filter capacitor
- D1 - Green LED (Light Emitting Diode) power on indication
- R1 - Current limiting resistor for LED
- R2 - Time On resistor for 555 timer PWM (Pulse Width Modulation)
- R3 - Time On resistor for 555 timer PWM
- D2 - Diode used for current direction control
- C2 - Capacitor used for both Time On and Time Off control for 555 timer PWM
- R4 and R5 - Time Off resistors for 555 timer PWM (Time Off should be $\approx$ 20mS)
- C3 - Filter capacitor
- U1 - 555 timer used to create the astable multivibrator signal
- J3 - Astable multivibrator signal connection to the servo motor

**Circuit Calculations**

- $R1 = \frac{(Vin - V_{D1})}{I_{D1}}$

  $R1 = \frac{(3.7V - 2V)}{3mA}$

  $R1 \approx 566.67\Omega$ or $560\Omega$

- Time On Position 1

  $TimeOn_{P1} \approx 0.693(R2)(C2)$

$$TimeOn_{P1} \approx 0.693(27K\Omega)(0.022\mu F)$$

$$TimeOn_{P1} \approx 0.412mS \text{ or } 412\mu S$$

- Time On Position 2

$$TimeOn_{P2} \approx 0.693(R3)(C2)$$

$$TimeOn_{P2} \approx 0.693(100K\Omega)(0.022\mu F)$$

$$TimeOn_{P2} \approx 1.525mS$$

- Time Off

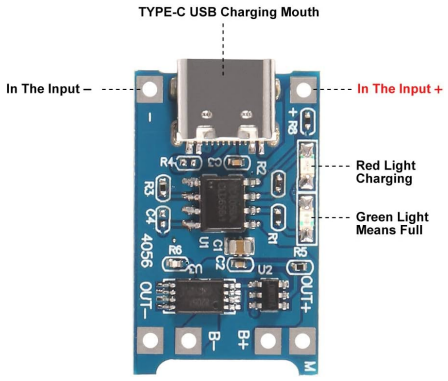
$$TimeOff \approx 0.693(R4+R5)(C2)$$

$$TimeOff \approx 0.693(1.3M\Omega)(0.022\mu F)$$

$$TimeOff \approx 19.82mS$$

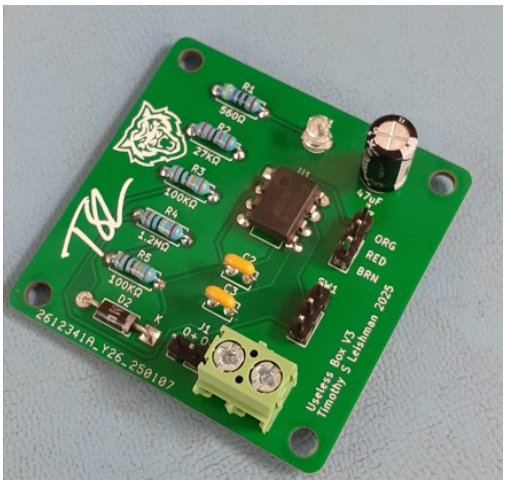
**PCB Assembly**

- C1 $47\mu F$
- C2 and C3, $0.022\mu F$
- D1 3mm LED
- D2 1N914B, suitable substitute 1N4001 or equivalent.
- R1 560Ω, suitable substitute 470Ω
- R2 $27K\Omega$, suitable substitute $22K\Omega$
- R3 $100K\Omega$
- R4 $1.2M\Omega$, suitable substitute $1M\Omega$ (Make R5 $220K\Omega$)
- R5 $100K\Omega$ suitable substitute $220K\Omega$ (Make R4 $1M\Omega$)



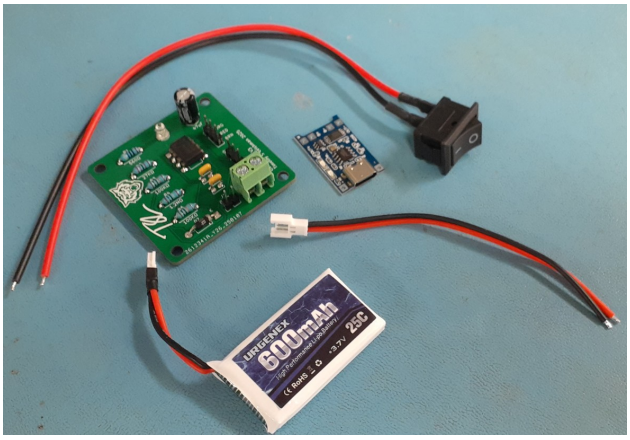

*USBC Power Management 4056*


*Lithium polymer battery*

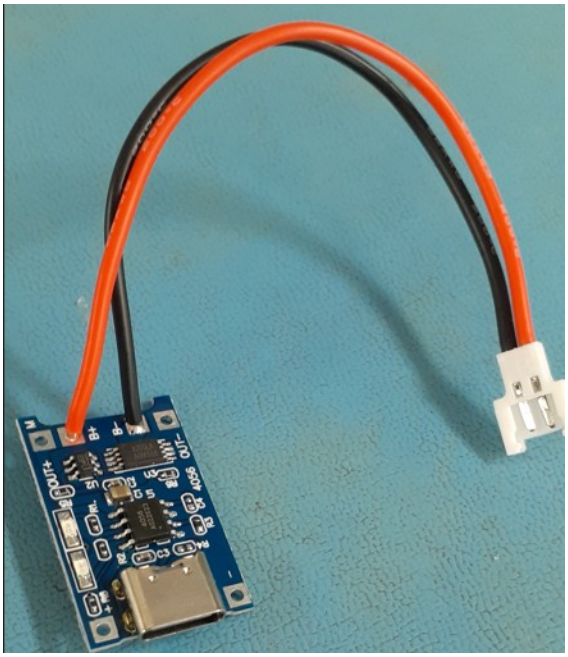



*Assembled PCB*

☐ Solder parts to the PCB, paying special attention to the correct alignment of the OpAmp, the 47uF capacitor, the LED, and the diode.



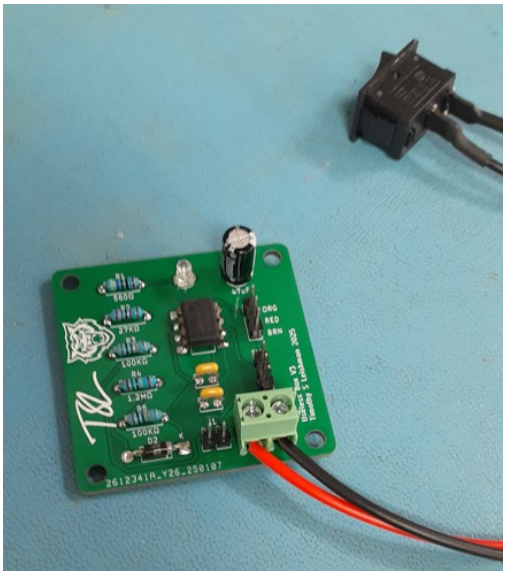

*PCB wiring part 1*



*PCB wiring part 2*

☐ Solder the male battery connector wire's black lead to B- and the red lead to B+ of the 4056 USBC Power Management IC.

☐ (Not shown in image) Solder a two-pin Dupont connector Red lead to OUT+ and black lead to the OUT - of the 4056 USBC Power Management IC.

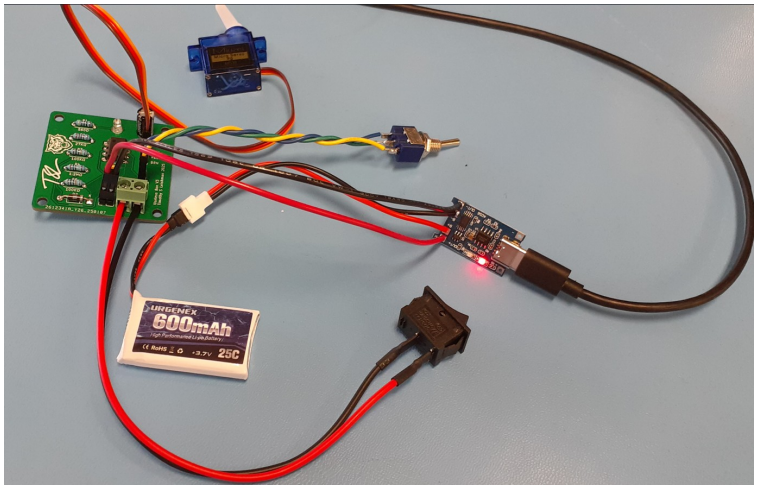



*PCB wiring part 3*

☐ Connect the red and black switch leads to the screw terminal block.

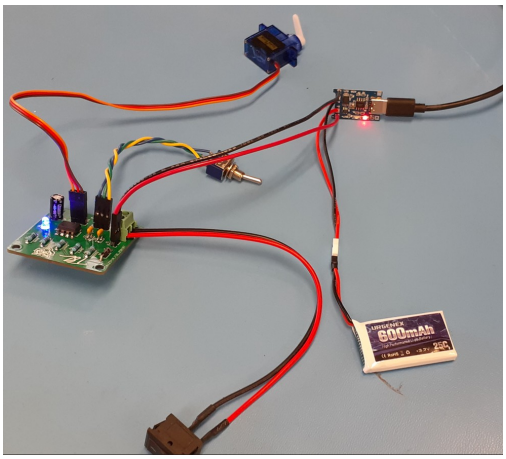
*PCB wiring part 4*

☐ Solder the toggle switch to a 3-pin Dupont header wire, make sure to connect the center pin of the toggle to the center pin of the Dupont connector.




*PCB wiring part 5*

☐ Connect the Battery last! Make sure the On/Off rocker switch is in the off or O position. Connect all the wires to the PCB paying attention to the correct orientations.



*PCB wiring part 6*

☐ With all the wiring connected turn the rocker switch to the On position.

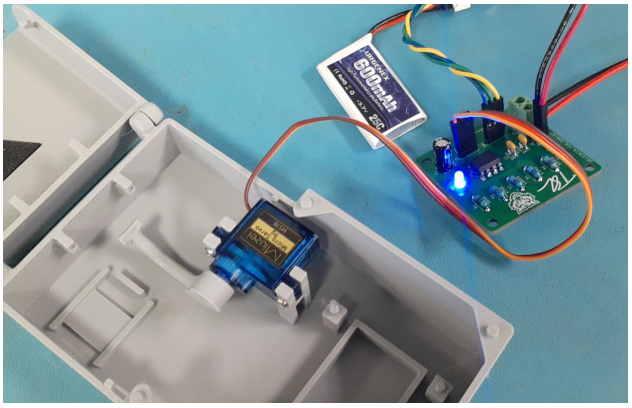
☐ Verify the PCB LED turns on.

☐ Verify the servo changes position when the toggle switch is switched.

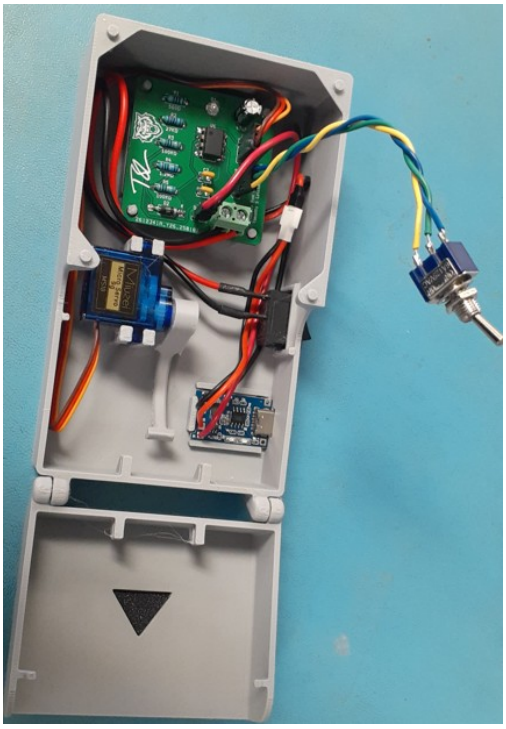
☐ Remove the USBC cable and verify the PCB will operate from battery power.

**Useless Box Assembly**

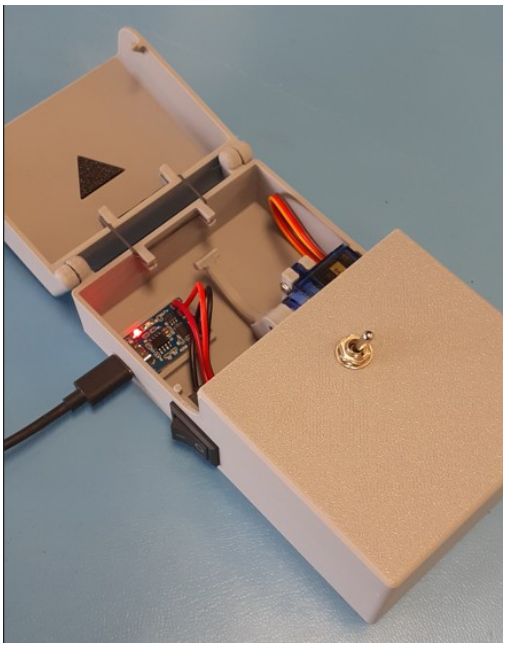
*Silicon Adhesive Bengal Head and Arm*



*Servo Attach and Test*



*Silicon Adhesive USBC power management and PCB boards, wire routing*



*Silicon Adhesive Top, attach rubber band, and test USBC*

# ESP32 Getting Started

**Additional ESP32 Course Recommendations:**

☐ Udemy Course: *ESP32 for Arduino Makers* by Dr. Peter Dalmaris

☐ Udemy Course: *ESP32 By Example* by Dr. Peter Dalmaris and James Solderitsch

## ESP32 Introduction

### What is a ESP32?

The ESP32 is a low-cost, power-efficient microcontroller with built-in Wi-Fi and Bluetooth capabilities, developed by Espressif Systems. It features a dual-core or single-core processor, making it suitable for IoT applications, smart home devices, wearables, and embedded systems. With a rich set of peripherals, including GPIO, ADC, DAC, PWM, and SPI, the ESP32 supports various sensors and actuators, allowing for versatile project development. Its low power consumption and deep sleep modes make it ideal for battery-powered applications, while its strong community support and compatibility with Arduino, MicroPython, and ESP-IDF simplify development.

### ESP32 Key Features (In Simple Terms):

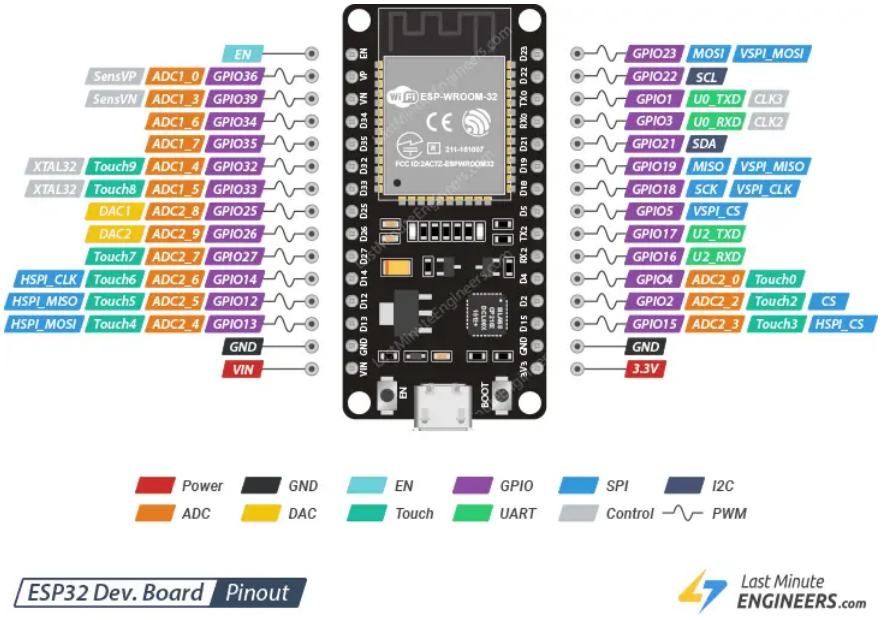
- Fast Processor – Can handle multiple tasks at once, making it great for smart devices.
- Built-in Wi-Fi & Bluetooth – Connects to the internet and other devices wirelessly.
- Low Power Usage – Can run on batteries for a long time with special sleep modes.
- Lots of Connection Pins – Can connect to buttons, sensors, motors, and screens.
- Works Quickly – Runs at high speed (up to 240 MHz) for smooth performance.
- Secure – Has built-in protection to keep data safe.
- Memory for Storing Code – Can store and run programs directly on the chip.

- Multitasking – Can run several things at once, like reading a sensor while controlling a motor.
- Easy to Program – Supports popular coding platforms like Arduino and Python.
- Flexible Power Options – Works with different power sources, including batteries.
- Touch Support – Can detect touch without extra hardware, useful for touch buttons.
- Works with Many Devices – Can connect to different sensors, LEDs, motors, and displays.

**Where Do You Find ESP32s used?**

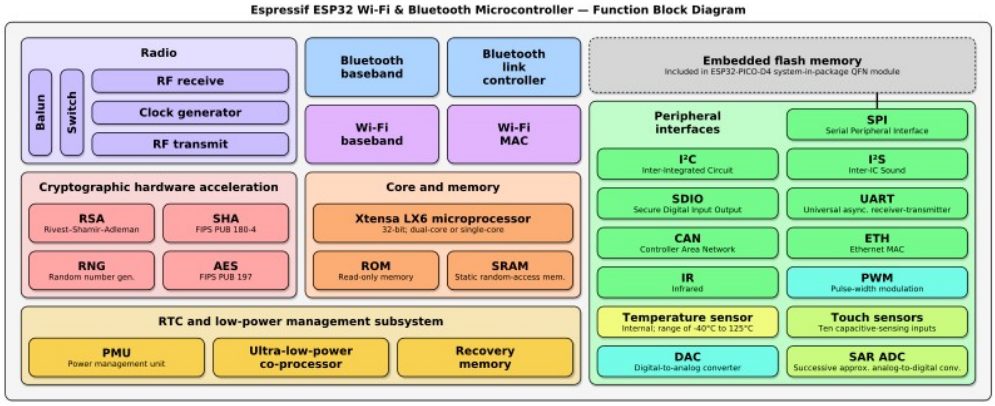
ESP32 microcontrollers are widely used in IoT devices, smart home automation, wearable technology, industrial automation, and robotics due to their wireless capabilities and low power consumption. They are commonly found in smart lights, thermostats, security cameras, and voice assistants, enabling remote control and automation. In industrial settings, ESP32s monitor sensors, control machinery, and manage data transmission. They are also used in DIY electronics projects, drones, weather stations, and health monitoring devices. Their versatility makes them ideal for applications requiring wireless communication, sensor integration, and real-time processing, making them a popular choice for both hobbyists and commercial products.

**ESP32 (WROOM-32) Dev Kit Specifications:**




*https://lastminuteengineers.com/esp32-pinout-reference/*

**ESP32 Functional Block Diagram**




*https://commons.wikimedia.org/wiki/File:Espressif_ESP32 _Chip_Function_Block_Diagram.svg*

**Data Sheet for ESP-WROOM-32 (IC)**

ESP32-WROOM-32 Datasheet

https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

## Arduino IDE and the ESP32

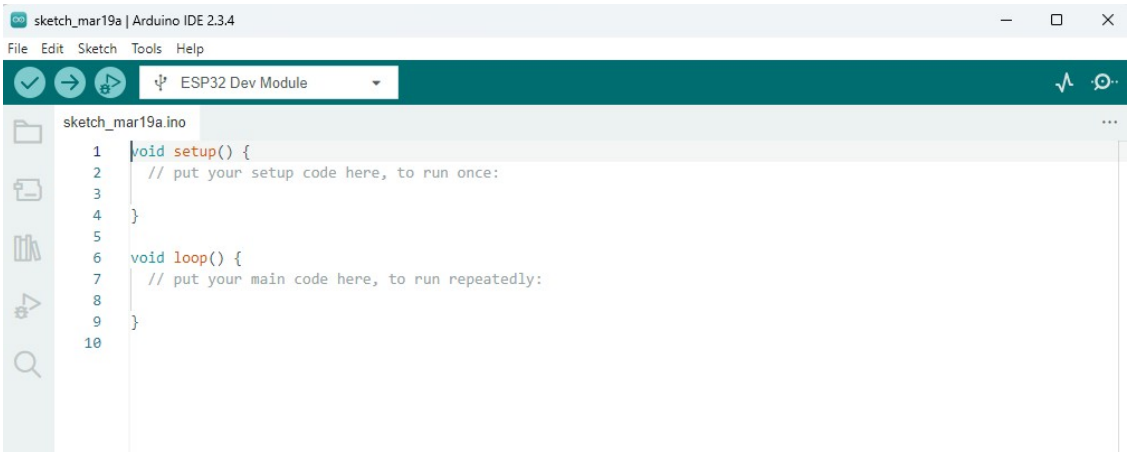
### Arduino IDE Installation

☐ Download the Arduino IDE here: Arduino IDE Software Download Website

https://www.arduino.cc/en/software

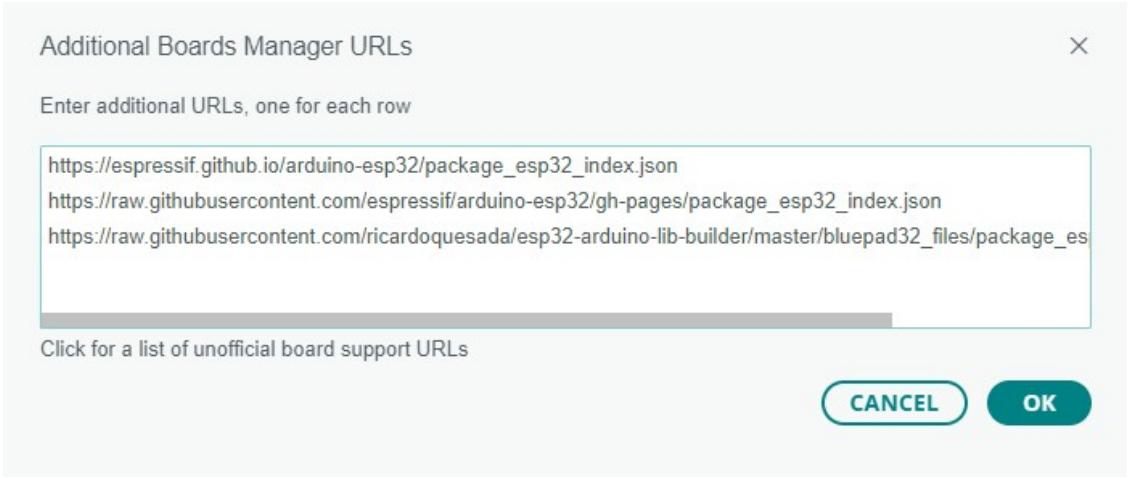
### Arduino IDE setup for ESP32

- FYI https://github.com/espressif/arduino-esp32




*Arduino IDE*

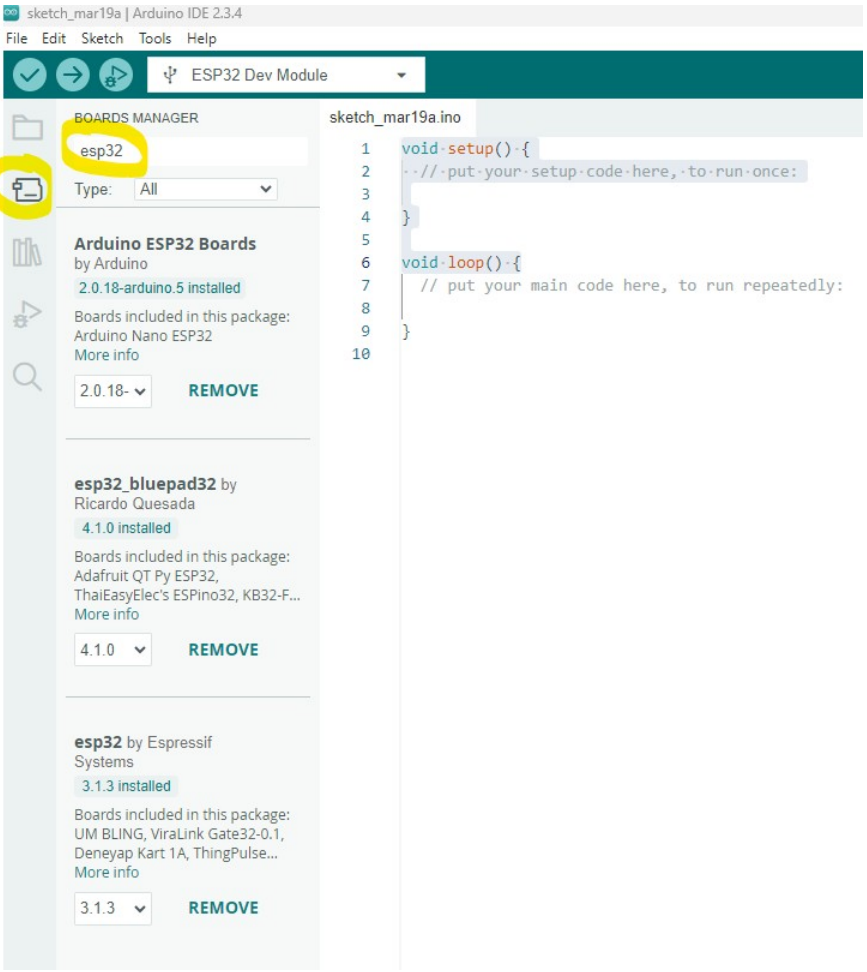
☐ Open the Arduino IDE




*Copy and paste the below links into the Additional Boards Manager URLs*

Go to File, Preferences, Additional boards manager URLs: Copy the following URLs and add them to the "Additional Boards URLs" in the Arduino IDE:

- ☐ https://espressif.github.io/arduino-esp32/package_esp32_index.json
- ☐ https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
- ☐ https://raw.githubusercontent.com/ricardoquesada/esp32-arduino-lib-builder/master/bluepad32_files/package_esp32_bluepad32_index.json




*Install ESP Boards*

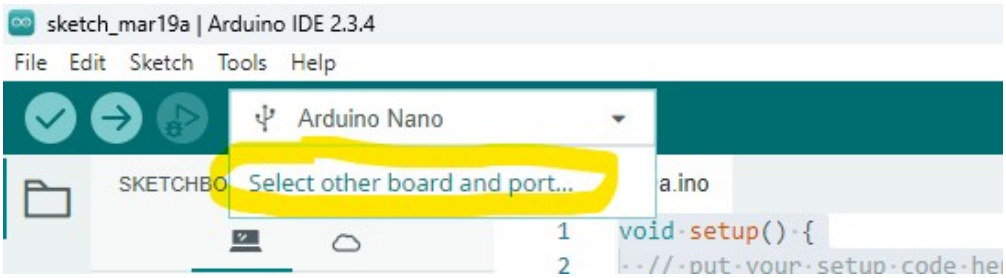
Install the following board libraries.

☐ Select the board manager icon on the left

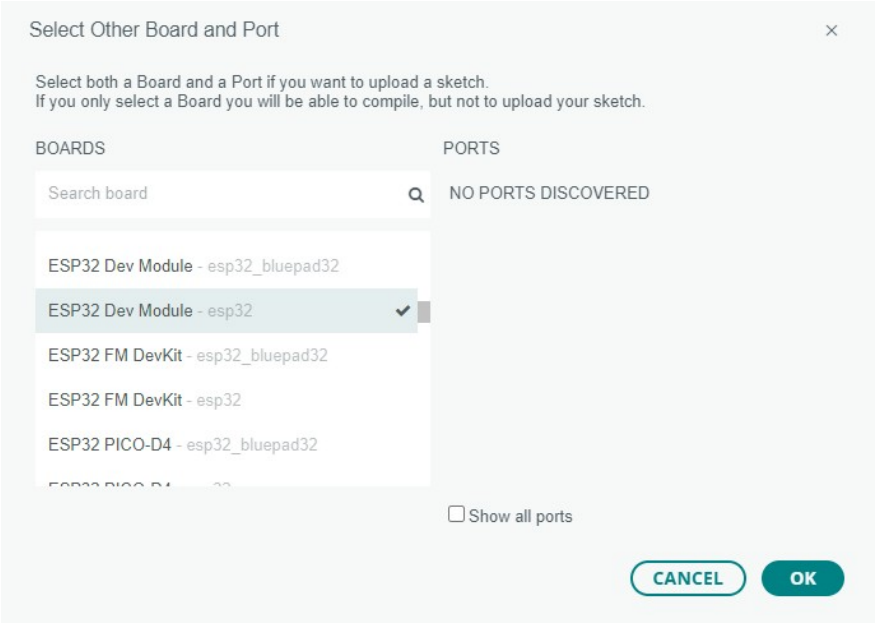
☐ Install the Arduino ESP32 Boards

☐ Install the esp32 bluepad32

☐ Install the esp32




*Select Boards*




*Verify ESP32 Dev Module - esp32 and esp32 bluepad32*

☐ Go to "Select other board and port..." Verify that the ESP32 Dev Module - esp32 and the ESP32 Dev Module -esp32_bluepad32 boards are available.

### CP210X USB to UART Driver Install

#### Step 1: Download the CP210x Universal Windows Driver

1. Open your web browser and go to the official **Silicon Labs** website:
   https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads
2. Select the **Universal Windows Driver** section.
3. Click on the **"Download for Windows 10/11"** link to get the latest driver package.

#### Step 2: Extract the Downloaded File

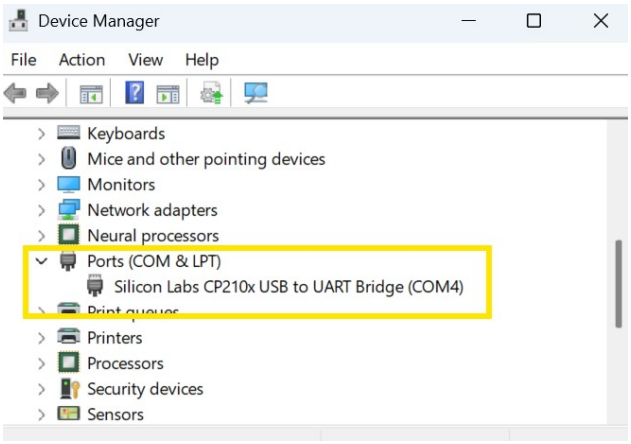
1. Locate the downloaded ZIP file (usually in your **Downloads** folder).
2. **Right-click** the file and select **"Extract All..."**.
3. Choose a destination folder and click **"Extract"**.

#### Step 3: Install the Driver

1. Open the extracted folder.
2. Locate the **"silabser.inf"** file.
3. **Right-click** it and select **"Install"**.
4. A prompt will appear—click **"Yes"** to install the driver.
5. Once completed, you should see a success message.

#### Step 4: Verify Driver Installation

1. **Connect your ESP32 Dev Kit** using a USB cable.
2. Open **Device Manager** (Press **Win + X**, then click **Device Manager**).
3. Expand the **"Ports (COM & LPT)"** section.




*Driver Verification*

4. Look for **"Silicon Labs CP210x USB to UART Bridge (COMx)"**.
    - If it appears, the driver installed successfully!
    - If there's a **yellow warning sign**, the driver may not have installed correctly.

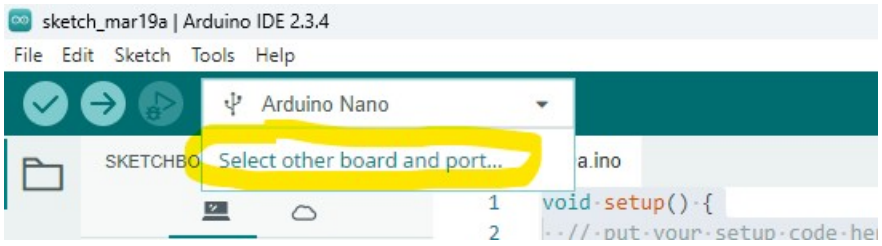
**Step 5: Identify Your COM Port**

1. Note the **COM Port Number** (e.g., COM3, COM5).
2. Use this **COM port** in **Arduino IDE, PlatformIO, or ESP-IDF** to communicate with your ESP32.

**Troubleshooting Tips**

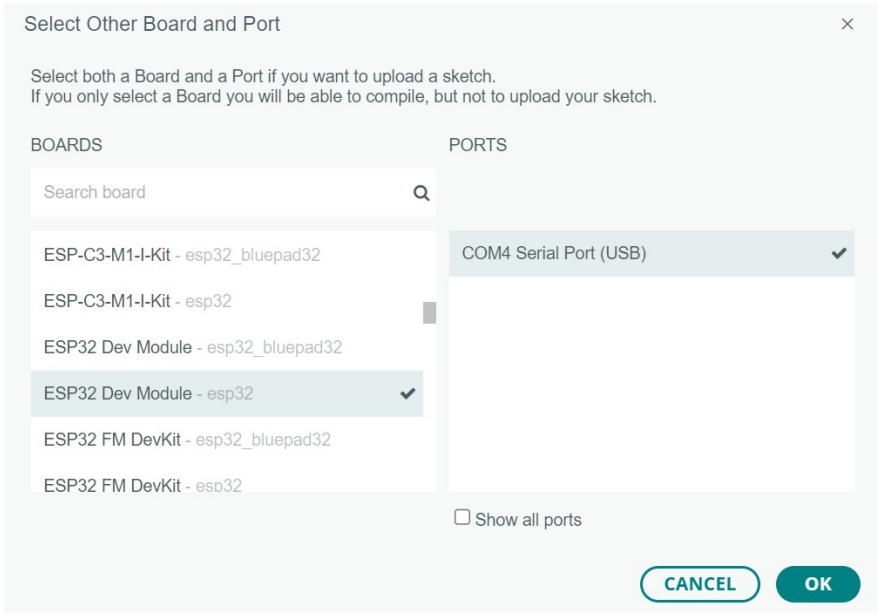
- If the driver doesn't install properly, **restart your PC** and try again.
- If your ESP32 is not detected, try using a **different USB cable** (some cables only provide power but no data transfer).
- If you see **"Unknown Device"** in Device Manager, right-click it, select **"Update Driver"**, then choose **"Browse my computer for drivers"**, and point to the extracted driver folder.

**Arduino IDE ESP32 Board and Comm Port Verification**



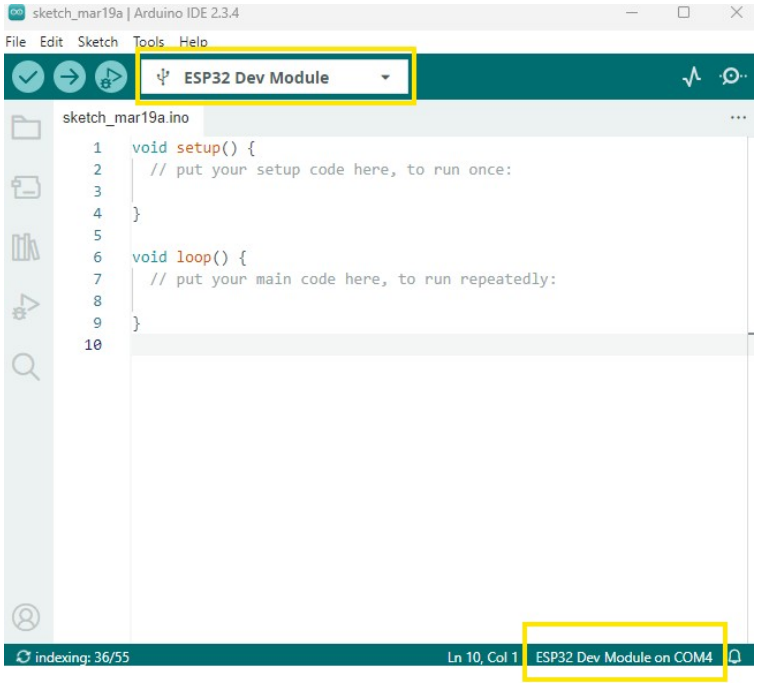

*Select other board and port...*

☐ Open the Arduino IDE, Select other board and port from the drop down.




*ESP32 Dev Module - esp32 with COMM Serial Port (USB)*

☐ Select the ESP32 Dev Module - esp32 and verify COMX Serial Port (USB) is connected and select OK.

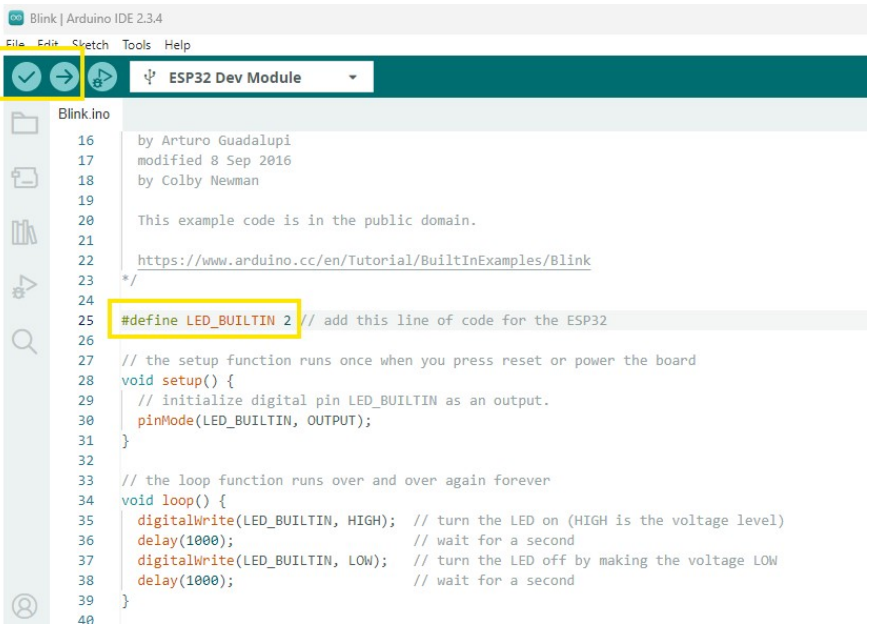



*Arduino IDE ESP32 Connection*

## ESP32 Testing

### My First Sketch - Blink

☐ Go to File, Examples, 01.Basics, and select Blink




*My First Sketch*

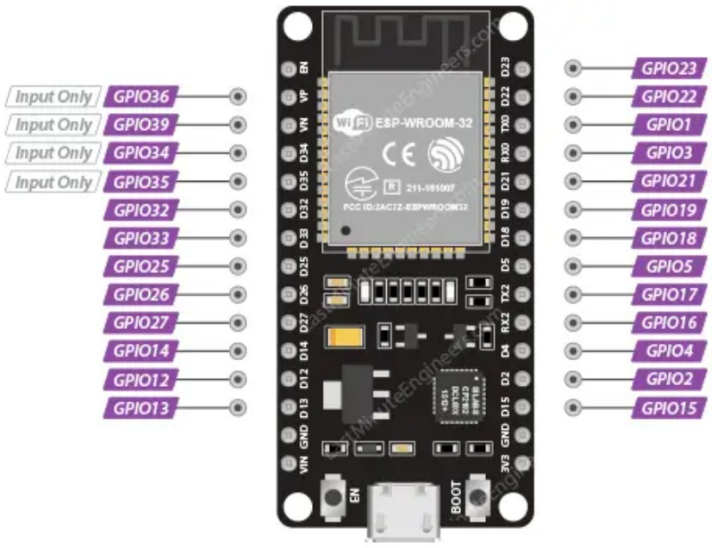
☐ Add the following line of code just above the "void setup": #define LED_BUILTIN 2

☐ Check the code by clicking the check mark located just below the File drop-down. No error messages should occur.

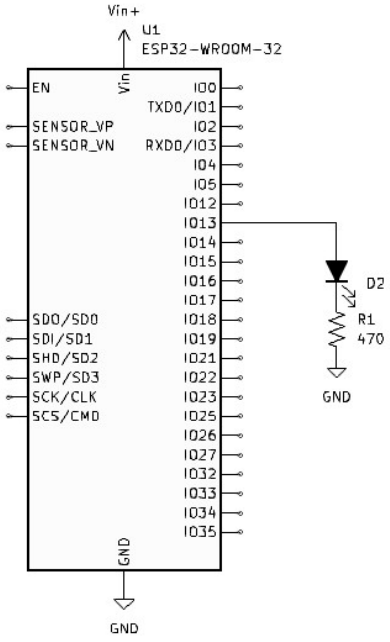
☐ Push or upload the code to the ESP32 by selecting the right arrow located next to the check mark.

☐ Verify that the blue on-board led flashes on and off.

**External LED and GPIO Testing**




*https://lastminuteengineers.com/wp-content/uploads/iot/ESP32-GPIO-Pins.png*




*External LED Schematic*

☐ Connect the VIN pin of the ESP32 dev board to the red (+) of the breadboard.

☐ Connect the GND pin of the ESP32 dev board to the black or blue (-) of the breadboard.

☐ Connect the anode (long wire) of the LED GPIO13.

☐ Connect the cathode (short wire) of the LED to a 470Ω resistor.

☐ Connect the other side of the 470Ω resistor to the black or blue (-) rail of the breadboard.

***GPIO Test Code***

1_Blink.ino

```
#define LED_BUILTIN 2
#define LED_External 13 //Add Variable LED_External and assign it to GPIO13

// the setup function runs once when you press reset or power the board
void setup() {

  pinMode(LED_BUILTIN, OUTPUT);   // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_External, OUTPUT);  // initialize digital pin LED_External as an output.

  digitalWrite(LED_BUILTIN, LOW);   // initialize low or off.
  digitalWrite(LED_External, LOW);  // initialize low or off.

}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_External, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_External, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

*ESP32 Blink External*

☐ Upload the code and verify the operation.

☐ (Challenge) Modify the code and wiring to operate the LED on the GPIO12 output. Repeat on 14, 27, 26...(GPIO 34, 35, 36, and 39 are input-only pins, these pins will not work as outputs)

**Pulse Width Modulation (PWM)**

PWM, or Pulse Width Modulation, is a clever way to control the amount of power sent to electronic devices by rapidly turning the power on and off. Instead of just giving a device full power or none at all, PWM switches the power on and off so quickly that the device responds as if it’s getting a steady level of power somewhere in between. The key here is the "duty cycle"—this means how much of the time the signal is on during each cycle. For example, a 50% duty cycle means the signal is on half the time, which would make a motor spin at half speed or an LED glow at half brightness.

This technique is super useful in electronics because it allows for precise control of motors, lights, and even sound, all without wasting energy like other methods might. PWM is widely used in robotics, drones, and anything that needs smooth, adjustable control.

**LED PWM**

2_LED_PWM.ino

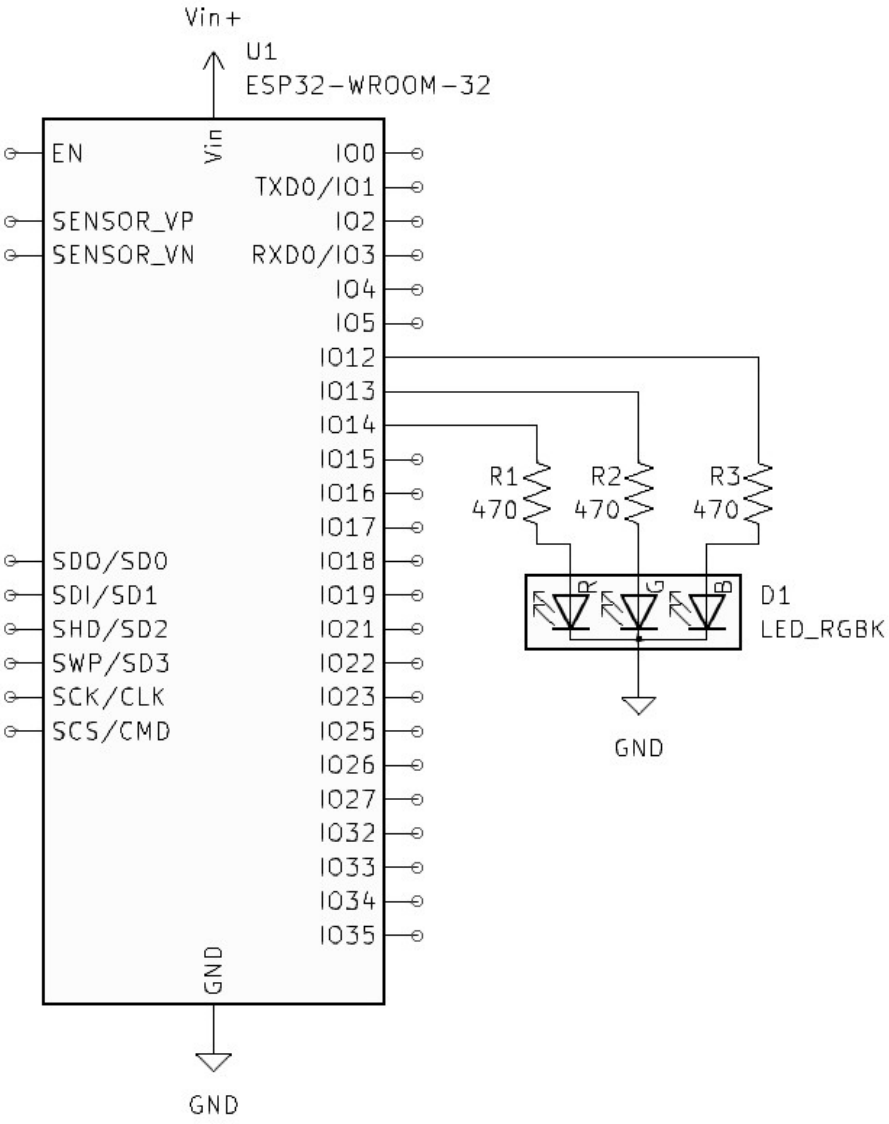
```
#define LED_BUILTIN 2
#define LED_External 13 //Add Variable LED_External and assign it to GPIO13

int DutyCycle = 0; //LED brightness variable, brightnes equals duty cycle
int FadeAmount = 1; // steps of fade
int Delay = 15; //variable called Delay

// the setup function runs once when you press reset or power the board
void setup() {

  ledcAttach(LED_External, 8000, 8); // light the external LED using a 8Khz 8-bit PWM
}

// the loop function runs over and over again forever
void loop() {

  ledcWrite(LED_External, DutyCycle);

  DutyCycle = DutyCycle+FadeAmount;

  // reverse the direction of the fading when the DutyCycle gets to 90%:
 if (DutyCycle >= 90) {
   while(DutyCycle>=0) {
   ledcWrite(LED_External, DutyCycle);
   delay(Delay); // wait for 7 milliseconds to see the dimming effect
   DutyCycle = DutyCycle-FadeAmount;} //decrement DutyCycle
  }
  // wait for 7 milliseconds to see the dimming effect
  delay(Delay);

}
```

*ESP32 LED Fade Pulse Width Modulation*

☐ (Challenge) Measure the PWM waveform using an Oscilloscope.

**RGB LED**

***Stop Light***




*ESP32 RGB Schematic*

3_LED_RGB_StopLight.ino

```
#define LED_BUILTIN 2
#define LED_Green 13 //Add Variable LED_External and assign it to GPIO13
#define LED_Red 14
#define LED_Blue 12

//int DutyCycle = 0; //LED brightness variable, brightnes equals duty cycle
//int FadeAmount = 1; // steps of fade
int Delay = 2000; //variable called Delay

// the setup function runs once when you press reset or power the board
void setup() {

  ledcAttach(LED_Red, 8000, 8); // light the external LED using a 8Khz 8-bit PWM
  ledcAttach(LED_Green, 8000, 8);
  ledcAttach(LED_Blue, 8000, 8);

}

// the loop function runs over and over again forever
void loop() {

  ledcWrite(LED_Red, 100);
  ledcWrite(LED_Green, 0);
  ledcWrite(LED_Blue, 0);

  delay(Delay);

  ledcWrite(LED_Red, 0);
  ledcWrite(LED_Green, 75);
  ledcWrite(LED_Blue, 0);

  delay(Delay);

  ledcWrite(LED_Red, 100);
  ledcWrite(LED_Green, 40);
  ledcWrite(LED_Blue, 0);

  delay(Delay);


}
```

*ESP32 RGB Stop Light Code*

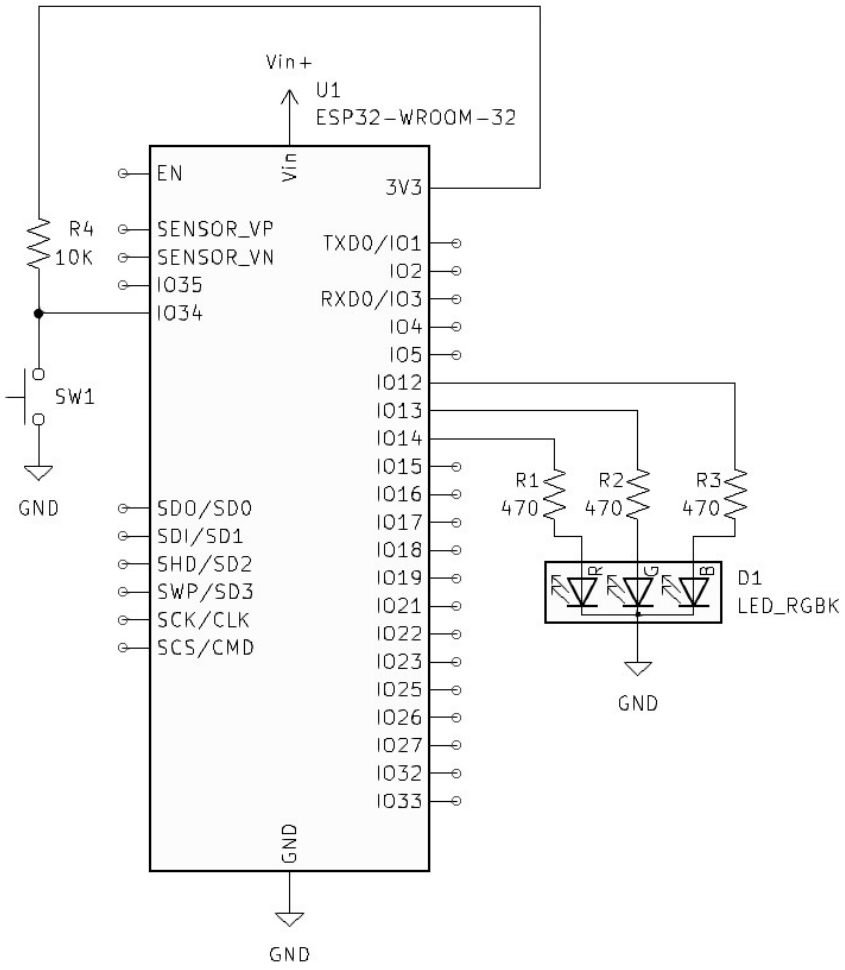
***Color Spectrum Change***

☐ Go to File, Examples, ESP32, AnalogOut, and select ledcWrite_RGB

☐ Modify the code so that uint8_t ledR = 14, uint8_t ledG = 13, and uint8_t ledB = 12

☐ Upload the Sketch and verify operation.

**Digital Read - Input Button**




*ESP32 Button Press Schematic*

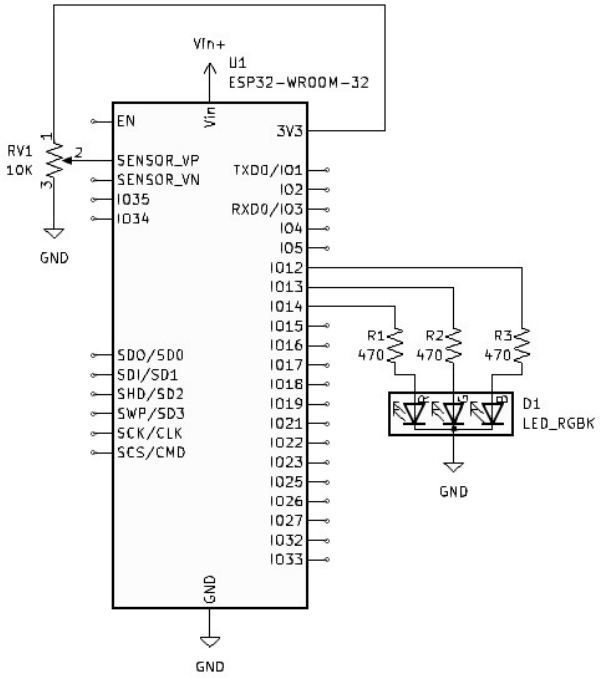
☐ Go to File, Examples, 20. Digital, and select Button.

☐ Modify the code so that the buttonPin aligns with the *ESP32 Button Press Schematic*, assign the ledPin to operate the Red/R1 output of the RGB LED.

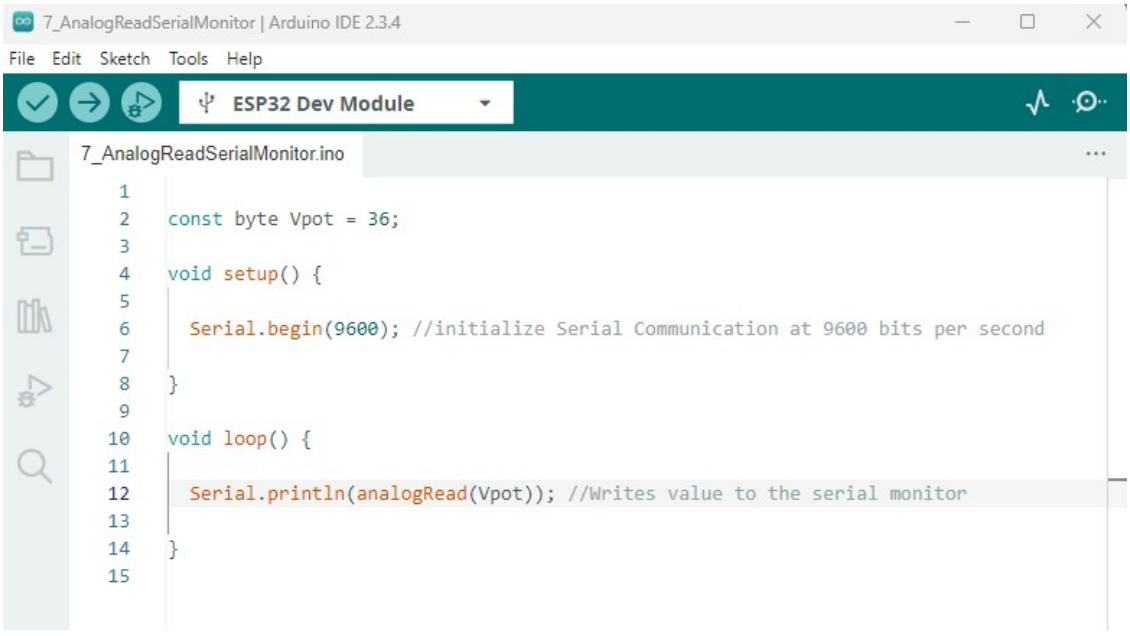
☐ Upload and verify that the Red LED is lit and turns off when the button is pressed.

☐ (Challenge) Modify the code so the Green LED will light when the button is pressed and turn off when the button is not pressed.

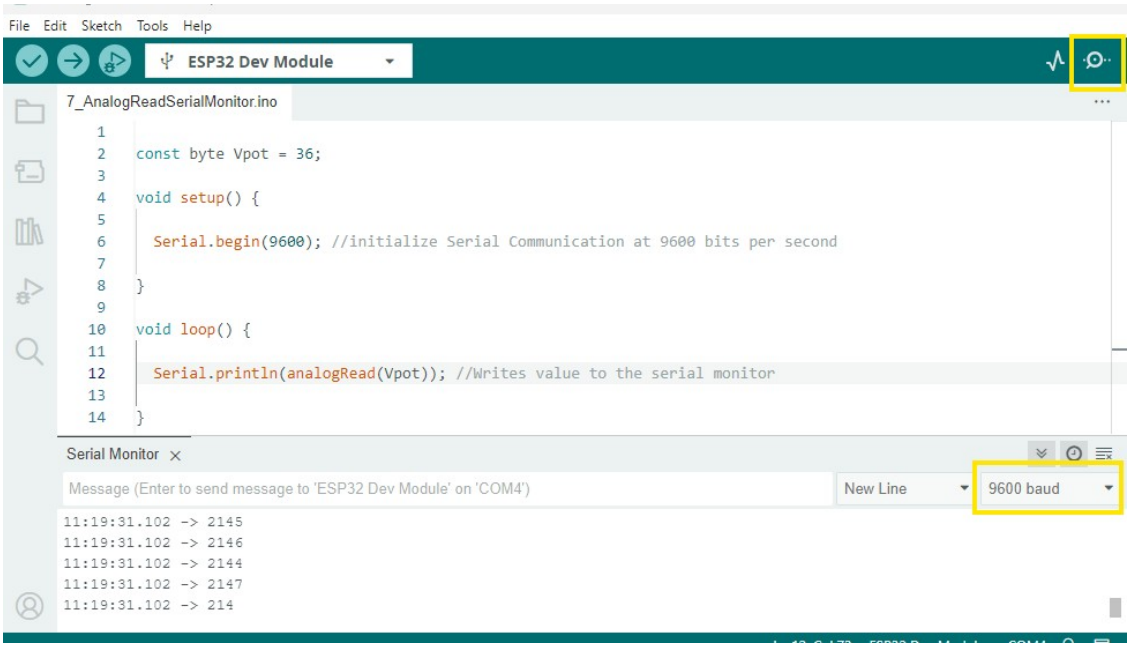
**Analog Read with Serial Monitor**




*Analog Read Schematic*




*Analog Read Sketch*




*Serial Monitor*

☐ Upload the Sketch.

☐ Enable the Serial Monitor view by selecting the upper right icon.

☐ Verify operation by adjusting the potentiometer and viewing the serial monitor displayed data (0 to 4095 range).

☐ (Challenge) When variable Vpot is less than 1365 make the Red LED light, when Vpot is between 1365 and 2730 make the Green LED light, and when Vpot greater than 2730 make the Blue LED light. And display which LED should be illuminated in the serial monitor. Try to do this on your own! (If you get stuck, see my code on the next page.)

8_AnalogReadSerialMonitor.ino

```
#define LED_Green 13
#define LED_Red 14
#define LED_Blue 12
const byte Vpot = 36;

void setup() {

  Serial.begin(9600); //initialize Serial Communication at 9600 bits per second
  pinMode(LED_Green, OUTPUT);
  pinMode(LED_Red, OUTPUT);
  pinMode(LED_Blue, OUTPUT);
  digitalWrite(LED_Green, LOW);
  digitalWrite(LED_Red, LOW);
  digitalWrite(LED_Blue, LOW);
}
void loop() {

  if (analogRead(Vpot)< 1365){
    Serial.print(analogRead(Vpot)); //Writes value to the serial monitor
    Serial.println(" Red LED ON");
    digitalWrite(LED_Green, LOW);
    digitalWrite(LED_Red, HIGH);
    digitalWrite(LED_Blue, LOW);
    delay(1000);
  }
    else if (1365<analogRead(Vpot) && analogRead(Vpot)< 2730){
    Serial.print(analogRead(Vpot));
    Serial.println(" Green LED ON");
    digitalWrite(LED_Green, HIGH);
    digitalWrite(LED_Red, LOW);
    digitalWrite(LED_Blue, LOW);
    delay(1000);
  }
    else if (analogRead(Vpot)> 2730){
    Serial.print(analogRead(Vpot));
    Serial.println(" Blue LED ON");
    digitalWrite(LED_Green, LOW);
    digitalWrite(LED_Red, LOW);
    digitalWrite(LED_Blue, HIGH);
    delay(1000);
  }
    else {
    Serial.println(analogRead(Vpot));
    Serial.println("Error");
    digitalWrite(LED_Green, LOW);
    digitalWrite(LED_Red, LOW);
    digitalWrite(LED_Blue, LOW);
    delay(1000);
  }
}
```

*Challenge Code*

### *Map Function*

In the previous section, we used an analog input which our ADC converted to a digital value representing a range of 0 to 4095. It may be helpful to convert this range to another range, for example 0 to 100 which then could be used as a duty cycle for the previous PWM code. The Map Function allows for this type of range conversion.

More info on the Map function here:

https://docs.arduino.cc/language-reference/en/functions/math/map/

9_AnalogReadSerialMapFunction.ino

```
#define LED_Green 13
#define LED_Red 14
#define LED_Blue 12
const byte Vpot = 36;
int DutyCycle;

void setup() {

  Serial.begin(9600); //initialize Serial Communication at 9600 bits per second
  pinMode(LED_Green, OUTPUT);
  pinMode(LED_Red, OUTPUT);
  pinMode(LED_Blue, OUTPUT);
  digitalWrite(LED_Green, LOW);
  digitalWrite(LED_Red, LOW);
  digitalWrite(LED_Blue, LOW);
}
void loop() {

  DutyCycle=analogRead(Vpot);
  DutyCycle=map(DutyCycle,0, 4095, 0, 100); //Map function changes the analogRead range from 0 - 4095 to 0 - 100

  Serial.print(DutyCycle);
  Serial.println("%DC");
  delay(500);

}
```

*Analog Map Function Sketch*

☐ Using the Serial Monitor, Verify that the potentiometer range is now 0 to 100.

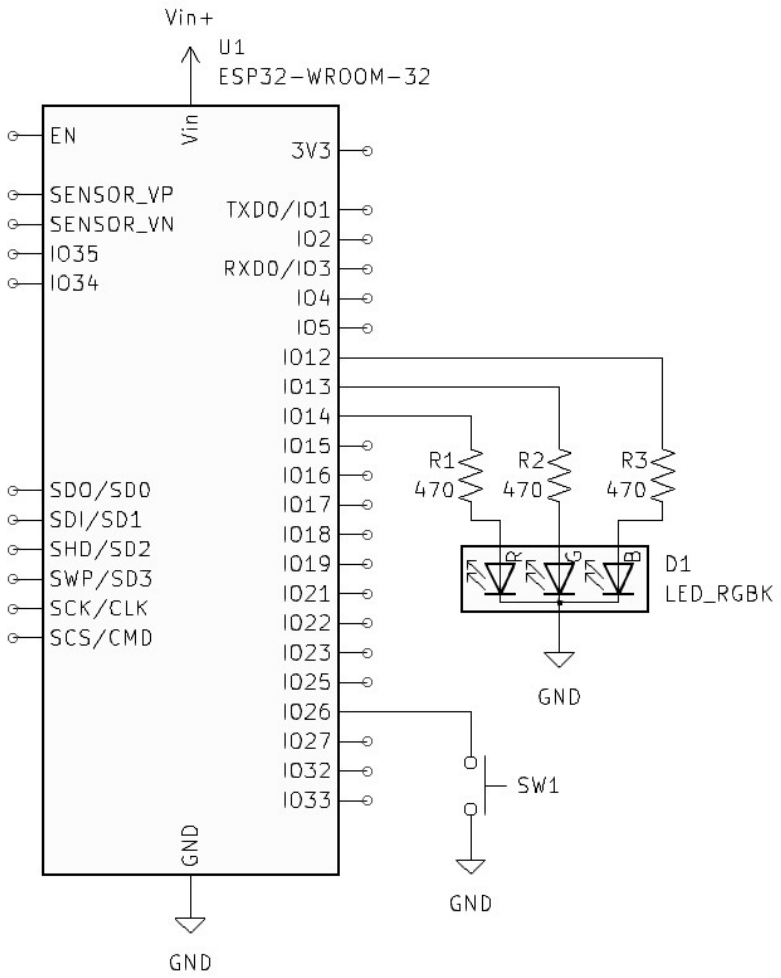
☐ (Challenge) Modify the Sketch using PWM, so that the Red LED brightness can be adjusted using the potentiometer. Try to do this on your own! (If you get stuck, see my code on the next page.)

10_AnalogReadSerialMapFunction.ino

```
int DutyCycle;

void setup() {

  Serial.begin(9600); //initialize Serial Communication at 9600 bits per second
  pinMode(LED_Green, OUTPUT);
  pinMode(LED_Red, OUTPUT);
  pinMode(LED_Blue, OUTPUT);
  digitalWrite(LED_Green, LOW);
  digitalWrite(LED_Red, LOW);
  digitalWrite(LED_Blue, LOW);

  ledcAttach(LED_Red,8000, 8);
}
void loop() {

  DutyCycle=analogRead(Vpot);
  DutyCycle=map(DutyCycle,0, 4095, 0, 100); //Map function changes the analogRead range from 0 - 4095 to 0 - 100

  Serial.print(DutyCycle);
  Serial.println("%DC");
  ledcWrite(LED_Red, DutyCycle);
  delay(500);
}
```

*Potentiometer Controlled LED Brightness Sketch*

**GPIO Interrupt**




*GPIO Interrupt Schematic*

Imagine you have a doorbell at your house. There are two ways to check if someone is ringing it:

1. **Polling** – You constantly walk to the door every few seconds and check if someone is pressing the bell. Even if no one is there, you keep checking, wasting a lot of time and energy.

2. **Interrupts** – You relax inside your house, doing other important things, until the doorbell rings and alerts you. Then, you immediately get up to see who's there.

In a microcontroller, polling means continuously checking a sensor or button in a loop, which wastes processing power and can slow down other tasks. Interrupts, on the other hand, let the microcontroller focus on other tasks and only respond when something important happens, like a button press or sensor trigger.

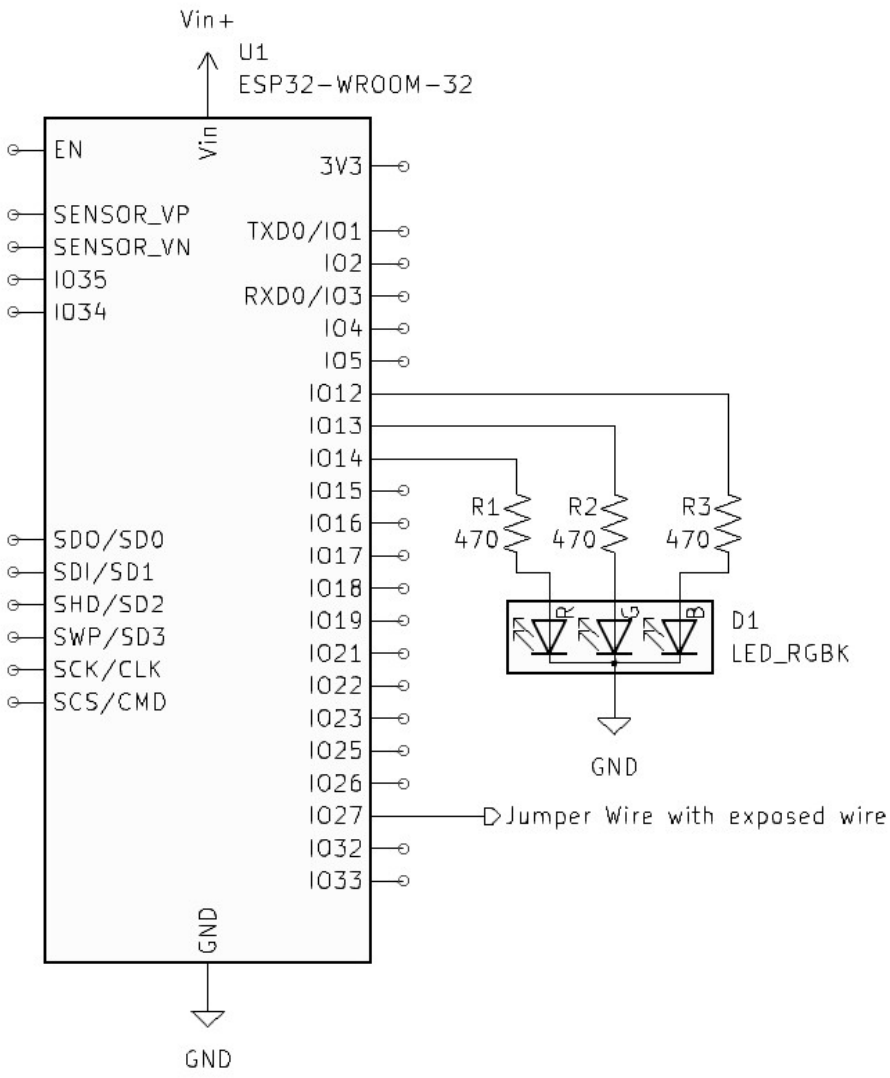
Using interrupts makes microcontrollers more efficient, responsive, and better at handling multiple tasks at once—just like how you'd rather hear a doorbell than keep checking the door every few seconds!

Review the following GPIO Interrupt Sketch Code: GPIO Interrupt Sketch Code

https://github.com/leistimo/ISTEM2025/blob/main/ESP32%20Sketches/11_GPIO_Interrupt/11_GPIO_Interrupt.ino

- ☐ (Challenge) Modify the previous circuit and GPIO Interrupt Sketch Code to add a second interrupt switch that will light the RGB led Green.

**Touch**




*ESP32 Touch Schematic*

| Touch Pins | | | | | |
|---|---|---|---|---|---|
| Touch0 (T0) | GPIO4 | Touch4 (T4) | GPIO13 | Touch7 (T7) | GPIO27 |
| Touch2 (T2) | GPIO2 | Touch5 (T5) | GPIO12 | Touch8 (T8) | GPIO33 |
| Touch3 (T3) | GPIO15 | Touch6 (T6) | GPIO14 | Touch9 (T9) | GPIO32 |

12_Touch.ino

```
1
2  const byte LED_GPIO = 14;  // Marked volatile so it can be read inside the ISR
3
4  void setup() {
5
6    Serial.begin(115200);
7    pinMode(LED_GPIO, OUTPUT);
8    Serial.println("ESP32 Touch Test");
9    delay(1000);
10 }
11
12 void loop() {
13
14   Serial.println(touchRead(T7)); //get value and print value of T7 (GPIO27)
15
16   if (touchRead(T7)<40)
17     digitalWrite(LED_GPIO, HIGH);
18   else
19     digitalWrite(LED_GPIO, LOW);
20
21   delay(500);
22
23 }
```

*ESP32 Touch Sketch*

**Touch Interrupt**

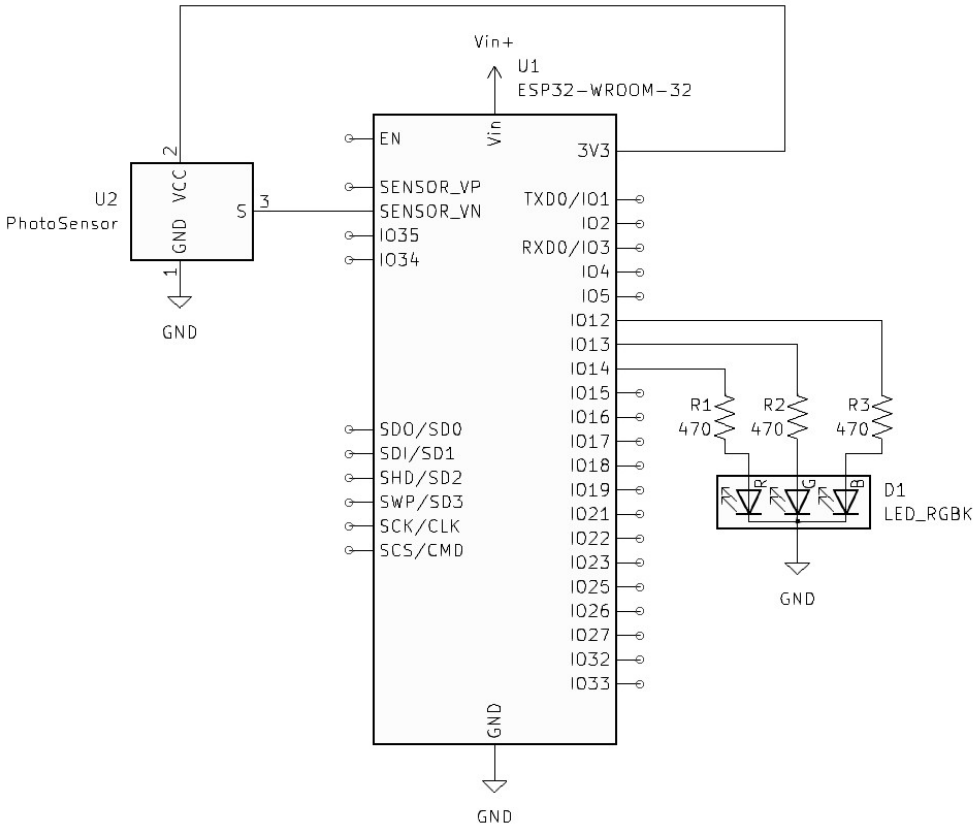
13_Touch_Interrupt.ino

```
const byte LED_Red = 14;
int threshold = 40;

bool touch7detected = false;

void gotTouch(){
  touch7detected = true;
}

void setup() {

  Serial.begin(115200);
  pinMode(LED_Red, OUTPUT);
  Serial.println("ESP32 Touch with interrupt");
  delay(1000);
  touchAttachInterrupt(T7, gotTouch, threshold);
}

void loop() {

  if (touch7detected)
  {
    digitalWrite(LED_Red, HIGH);
    Serial.println("Touch detected (T7)");
    touch7detected =false;
    delay(500);
    digitalWrite(LED_Red, LOW);
  }

}
```

*ESP32 Touch Interrupt Sketch*

☐ (Challenge) Add two additional touch interrupts. One that will turn on the RGB Blue when touched and another that will turn on the RGB Green LED.

**Photo-Resistor Sensor**




*ESP32 Photo-resistor Schematic*

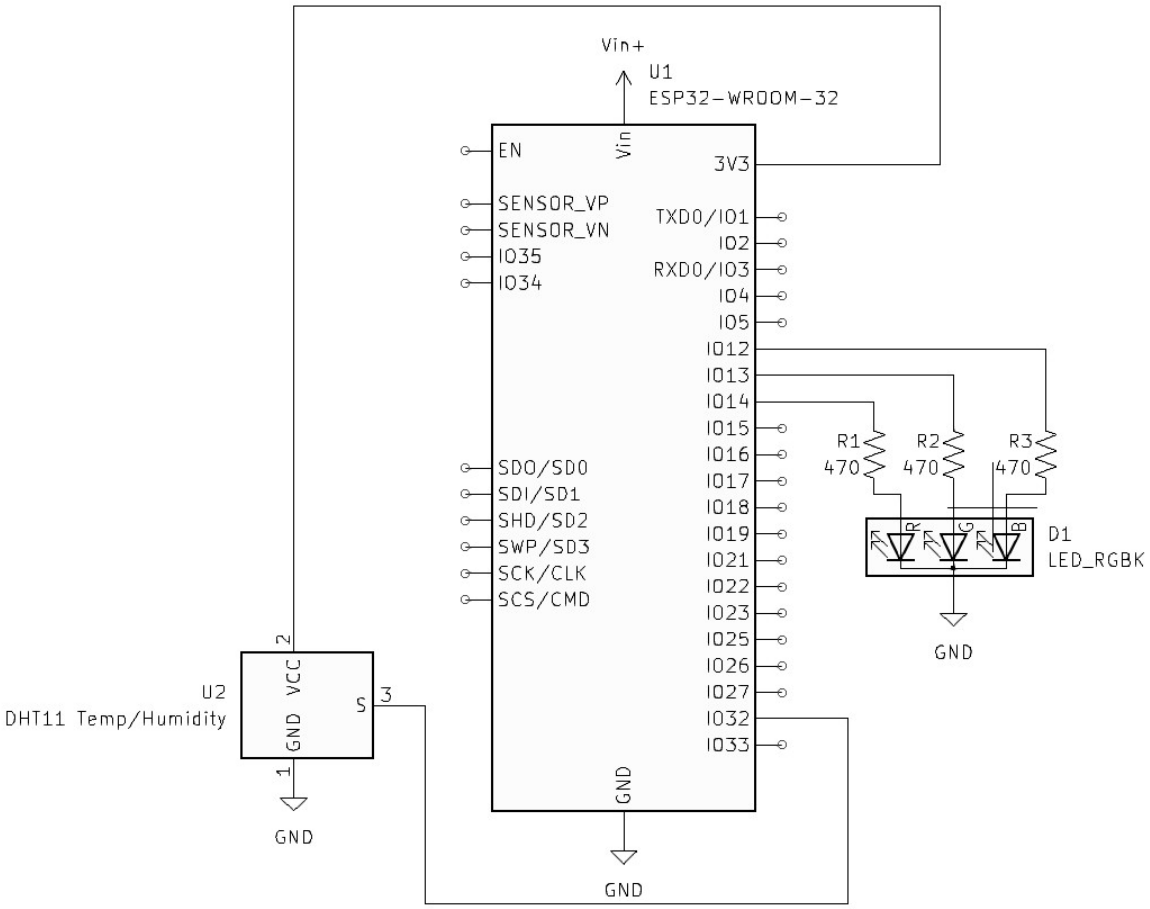
15_PhotoResistorSensor.ino

```
1   const byte Photo_Resistor = 39;
2
3   void setup() {
4     Serial.begin(115200);
5     //analogSetWidth(9); // Default resolution is 12 bits. (Choose 9-12)
6   }
7
8   void loop() {
9     Serial.println(analogRead(Photo_Resistor));
10    delay(500);
11    }
12
```

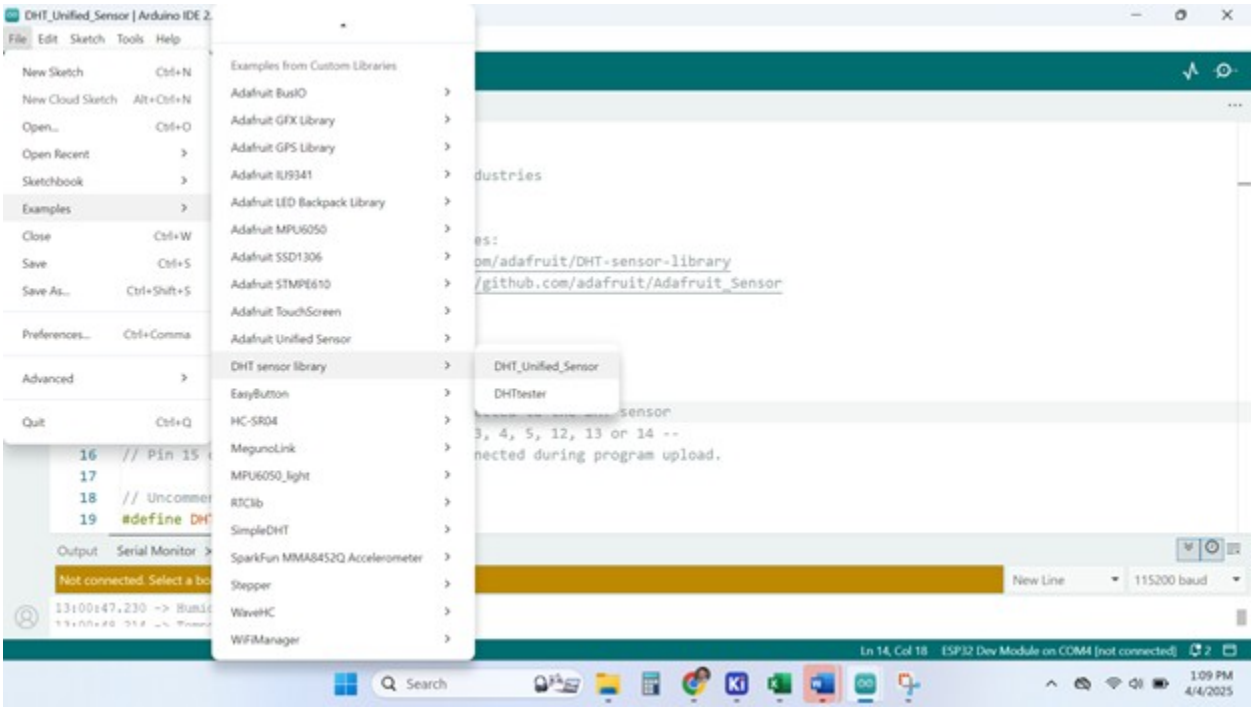
*ESP32 Photo-resistor Sketch*

☐ (Challenge - Night Light) Modify the sketch to turn on the RGB LED to produce a white light.

**DHT11 Temperature and Humdity Sensor**




*DHT11 Schematic*




*DHT Unified Sensor Example Sketch*

☐ Modify the code:

- ☐ #define DHTPIN 32
- ☐ #define DHTTYPE DHT11 (comment out DHT22)

☐ (Challenge) To ensure optimal protection against Electrostatic Discharge (ESD), it's important to maintain relative humidity (RH) within a controlled range. A general guideline is to keep RH between 30% and 70%, which helps reduce static charge accumulation and promotes effective charge dissipation. For best results, the ideal humidity range is typically between 40% and 60%, as this provides a balanced environment that minimizes the risk of ESD without introducing moisture-related issues.

Maintaining this humidity range is particularly critical in environments where sensitive electronic components are handled or stored. Lower humidity levels can increase the likelihood of static buildup, while excessively high humidity may cause condensation and

potential corrosion. By keeping humidity within the optimal window, facilities can better protect electronics from ESD-related damage.

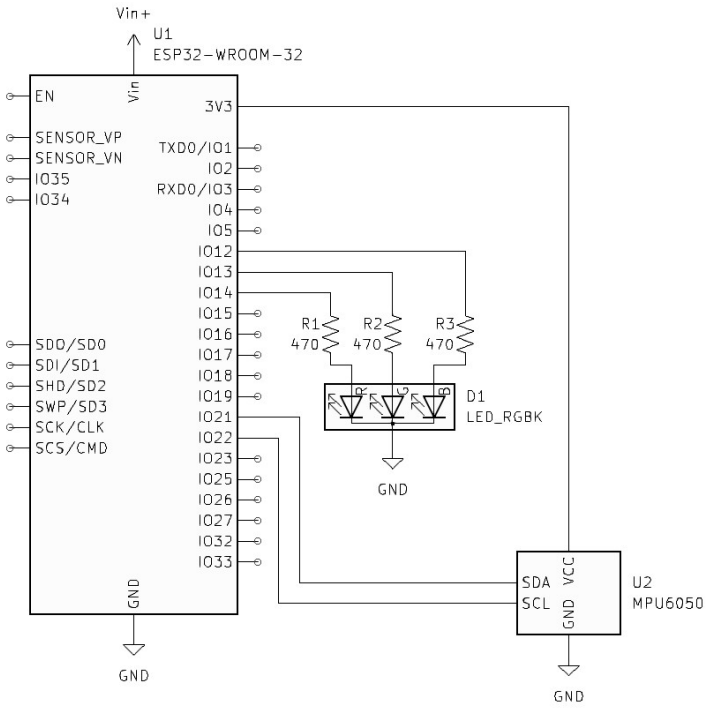
- ☐ Modify the sketch to light the Red LED when the humidity is below 30%.
- ☐ Modify the sketch to light the Blue LED when the humidity is above 70%.

**$I^2C$ (Inter-Integrated Circuit)**

$I^2C$, or Inter-Integrated Circuit, is a simple way for electronic devices to talk to each other using just two wires. Imagine a group of friends passing notes in class — instead of everyone shouting, they quietly send messages one at a time. In I²C, the two "note-passing" wires are called SDA (Serial Data Line) and SCL (Serial Clock Line). The SDA line carries the actual data, while the SCL line keeps the rhythm, like a metronome, making sure every device knows exactly when to send or read a message.

This system is commonly used in electronics to connect things like sensors, displays, and memory chips to a microcontroller (the "brain" of a device). One device acts as the "boss" (called the master), and the rest are "helpers" (slaves or peripherals). Since I²C only uses two wires, it’s very efficient for small projects or tight spaces, making it popular in robotics, wearables, and everyday gadgets.

**MPU6050 3 Axis Accelerometer Gyroscope**




*MPU6050 Schematic*

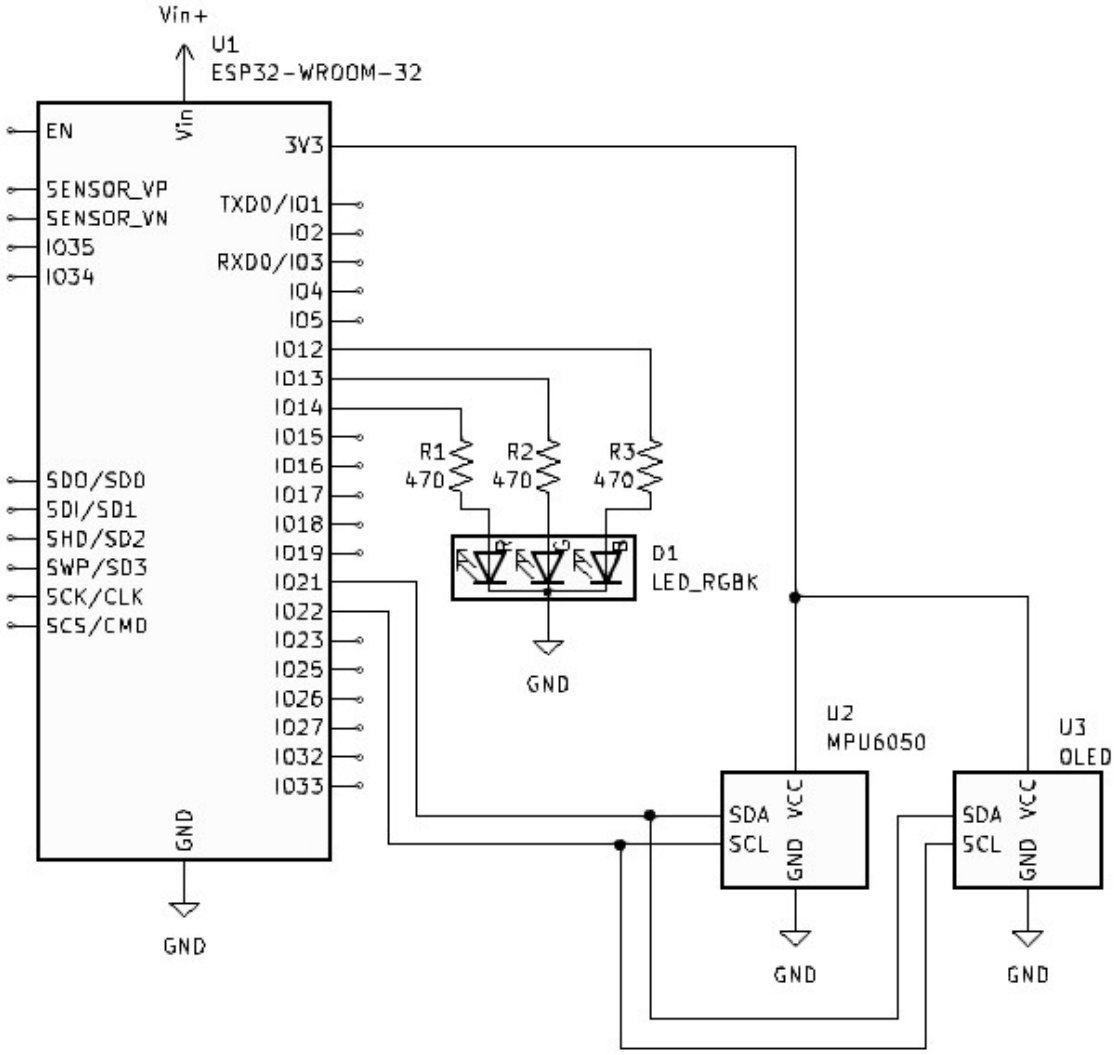
☐ Arduino Sketch: File, Examples, Adafruit MPU6050, Plotter.

☐ Open the serial monitor and verify that x and y values change when the breadboard is tipped left, right, forward, and backward.

☐ (Challenge) Modify the sketch so that when the board is tipped right the red led lights, when tipped left the green led lights, and when the board is tipped forward or backward the blue led lights.

**OLED**

***Adafruit SSD1306 Test Sketch***




*OLED Schematic*

☐ Load OLED Test Sketch: File, Examples, Adafruit SSD1306, ssd1306_128x32_12c.

☐ Upload and verify that the OLED is operating.

### *Hello World*

22_OLED_HelloWorld.ino

```
#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_Sensor.h>

#define SCREEN_WIDTH 128  // OLED display width, in pixels
#define SCREEN_HEIGHT 32  // OLED display height, in pixels

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire);

void setup() {
  Serial.begin(115200);
  Wire.begin();  // Initialize I2C communication
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {  // Address 0x3C for 128x32
    Serial.println(F("SSD1306 allocation failed"));
    for (;;)
      ;  // Don't proceed, loop forever
  }
  display.display();  // Display AdaFruit Logo
  delay(2000);        // Pause for 2 seconds
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setRotation(0);
  display.setCursor(0, 0);

  display.println("Message: Hello World");//loading data prior to display
  display.display();             //Update the OLED display
}

void loop() {

}
```

*Hello World Sketch*

☐ Analyze and recreate the "Hello World" Sketch. File, New Sketch.

### *Hexadecimal*

The **hexadecimal number system**, or **hex** for short, is a way of representing numbers using **base 16** instead of the usual base 10 we use in everyday life. In hex, we count from 0 to 15, but since we run out of single-digit numbers after 9, we use the letters **A to F** to represent 10 to 15. So it goes:

$$0, 1, 2, \ldots, 9, A, B, C, D, E, F.$$

After `F` comes `10` in hex, which is actually 16 in decimal.

This system is very useful in electronics and computer science because it aligns neatly with **binary**—the language of computers. One hexadecimal digit represents exactly **four binary bits**, making it much easier to read and write long binary numbers. For example, the binary number

$$1111\ 0000$$

can be written simply as `F0` in hex. Hex is essentially a shorthand for binary, helping engineers and programmers work more efficiently when dealing with memory addresses, digital color codes, and microcontroller instructions.

### *Converting Decimal to Hexadecimal*

Let's convert the decimal number **156** to hexadecimal step by step.

#### *Step 1: Divide by 16*

Divide the decimal number by 16 and find the **quotient** and **remainder**:

$$156 \div 16 = 9 \text{ remainder } 12$$

Since hexadecimal digits go from 0–15, we convert the remainder of 12 into its hexadecimal equivalent:

$$12 = C$$

#### *Step 2: Divide the Quotient Again*

Take the quotient from the previous division and divide by 16 again:

$$9 \div 16 = 0 \text{ remainder } 9$$

***Step 3: Read the Remainders from Bottom to Top***

We now read the remainder in reverse order (bottom to top):

$$156_{10} = 9C_{16}$$

***Final Answer***

$$\boxed{156_{10} = 9C_{16}}$$

☐ (Challenge) Convert the decimal number 255 to Hexadecimal.

$$255_{10} = \_\_\_\_\_{}_{16}$$

***Serial Monitor Read***

☐ File, Examples, 04. Communications, Read ASCII String

☐ Modify the following code to work with our circuit:

- ☐ const int redPin = ?;
  const int greenPin = ?;
  const int bluePin = ?;
- ☐ set the serial communications to operate at 115200, Serial.begin(115200);

☐ upload code and open the serial monitor

☐ In the serial monitor message box, enter 255 0 0 and press enter. Verify the Red LED is illuminated.

- ☐ enter 0 255 0 and verify the Green LED is illuminated.
- ☐ enter 0 0 255 and verify the Blue LED is illuminated.

☐ Modify the sketch code (serial print) to make the serial monitor more reader/user-friendly. Example:


*Serial Monitor reader/user-friendly*

***Serial Monitor Read and Display on OLED***

☐ (Challenge) In addition to the Serial Monitor, display the reader/user-friendly data on the OLED (use the Hello World Sketch as reference).

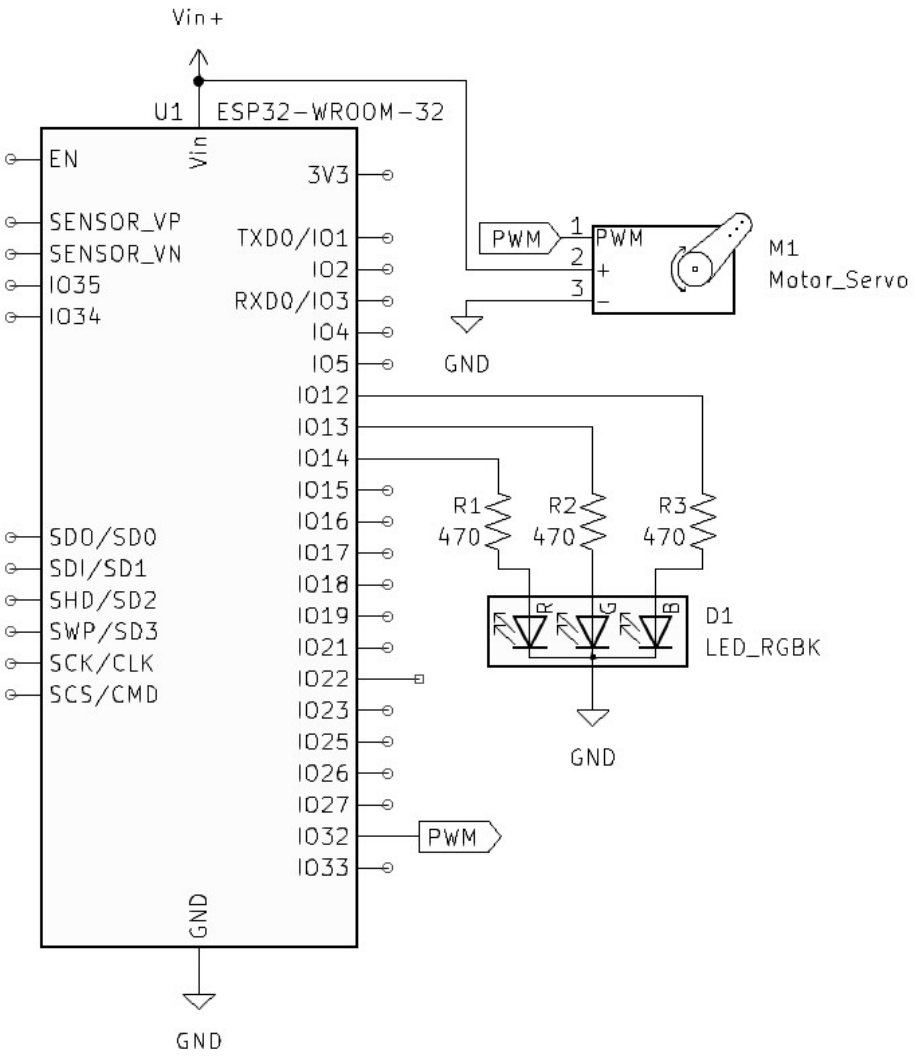
***MPU6050, LED, and OLED***

☐ (Challenge) Modify the previous Sketch to add the MPU6050 (use the MPU6050 LED Sketch as reference).

- ☐ On the OLED, display "Bank Right" and the x value when the x value is greater than 1 and the red led is on.
- ☐ On the OLED, display "Bank Left" and the x value when the x value is less than 1 and green led is on.
- ☐ On the OLED, display "Nose Up" when the y value is less than 1 and "Nose Down" when the y value is greater than 1.
- ☐ When the MPU is flat and all LEDs are off, display on the OLED, println "Straight" println "Level".

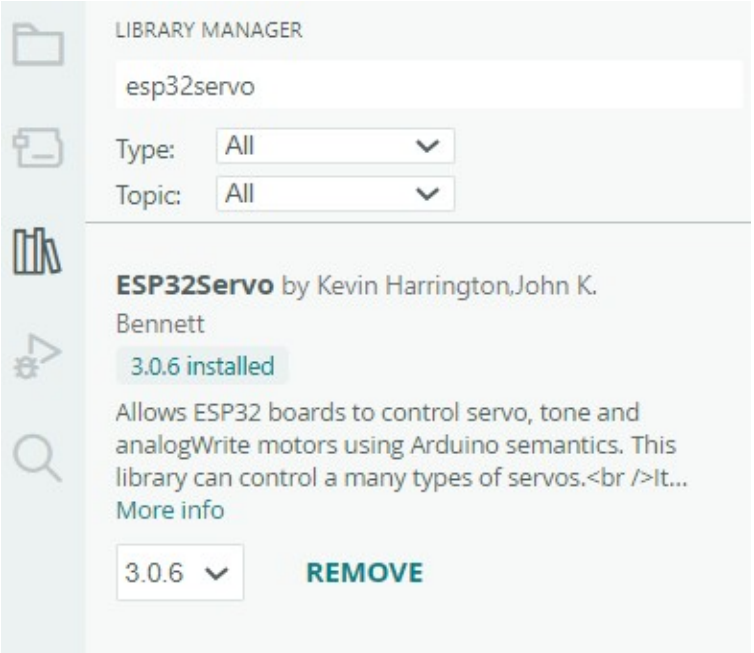
## Servo Motor

### *Servo Motor Sweep*




*ESP32 Servo Sweep Schematic*

***ESP32 Servo Library***




*ESP32 Servo library Download*

☐ Download the library ESP32Servo by Kevin Harrington

☐ In the Arduino IDE open the Servo example sketch : File. Examples, Servo, Sweep.

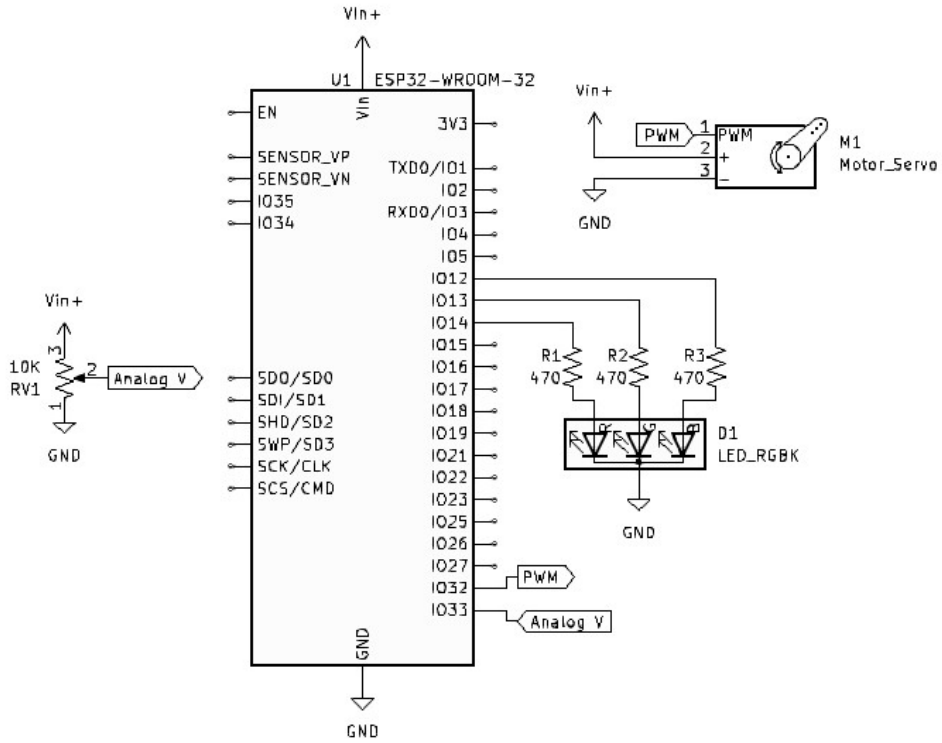
☐ Modify the code:

- ☐ #include ~~<Servo.h>~~ replace with # ESP32Servo
- ☐ myservo.attach~~(9)~~ replace with myservo.attach(32)

☐ Upload and test.

☐ (Challenge) Measure the PWM signal using an Oscilloscope.

***Servo Motor Knob***




*ESP32 Servo Knob Schematic*

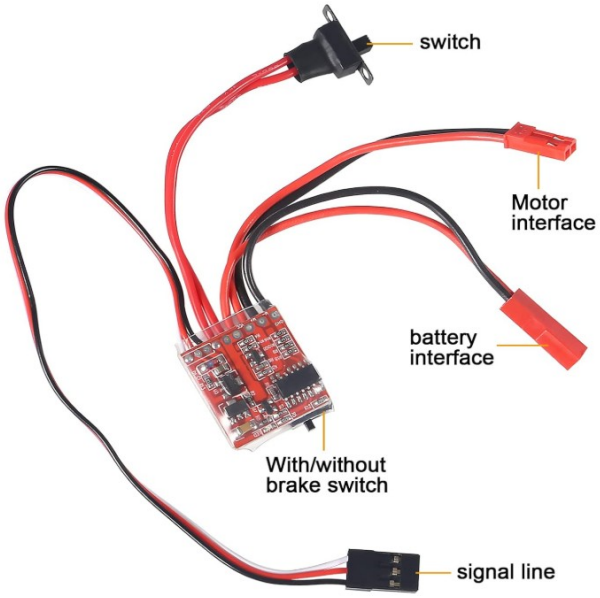
☐ In the Arduino IDE open the Servo example sketch : File. Examples, Servo, Knob.

☐ Modify the code:

- ☐ #include ~~<Servo.h>~~ replace with # ESP32Servo
- ☐ int potpin = ~~A0~~ replace with int potpin = A5
- ☐ myservo.attach~~(9)~~ replace with myservo.attach(32)
- ☐ val = map(val, 0, ~~1023~~, 0, 180) replace with val = map(val, 0, 4095, 0, 180)

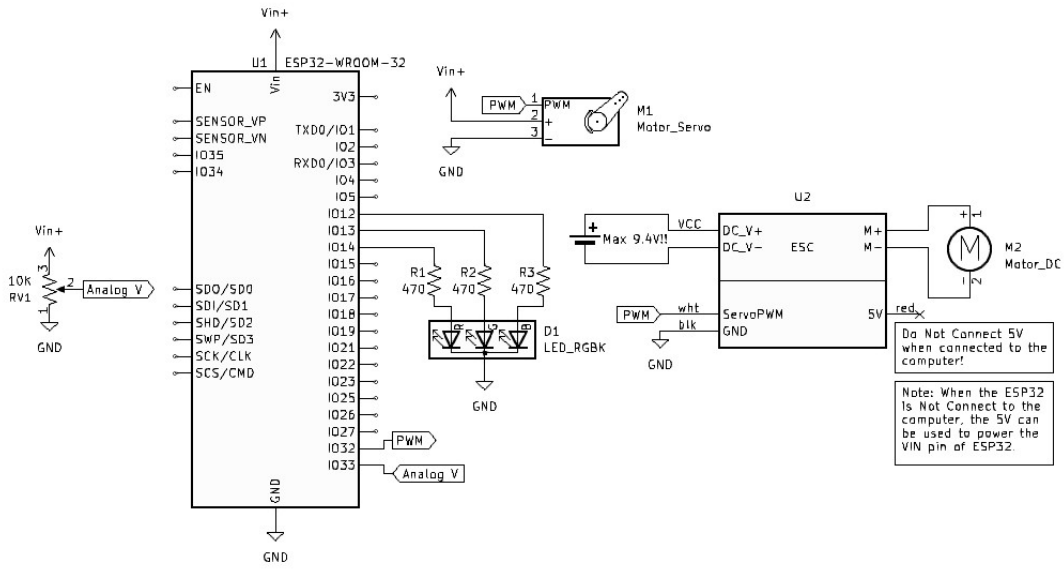
☐ Upload and test, adjust knob and verify servo operation.

**Brushed DC Motor**




*ESC Image*

- Switch - on/off
- Motor interface - connects to the motor
- battery interface - (max 9.4VDC OR 2S battery)
- signal line
  - white - Servo Signal input
  - red - 5v output (**do not connect** when the ESP32 is connected to the computer!) Note: When the ESP32 is not connected to the computer, this 5V output can be connected to VIN of the ESP32 to provide power.)
  - black - connect to ground GND.




*DC Motor ESC Schematic*

☐ The DC motor ESC operates using the same Servo PWM signal introduced previously.

☐ Connect the ESC and DC motor as seen in the above schematic. (Do not connect the 5V red wire from the signal line while connected to the computer!)

☐ Using the previous Servo Knob code, verify motor operation.

## WIFI

### *WIFI Scanner*

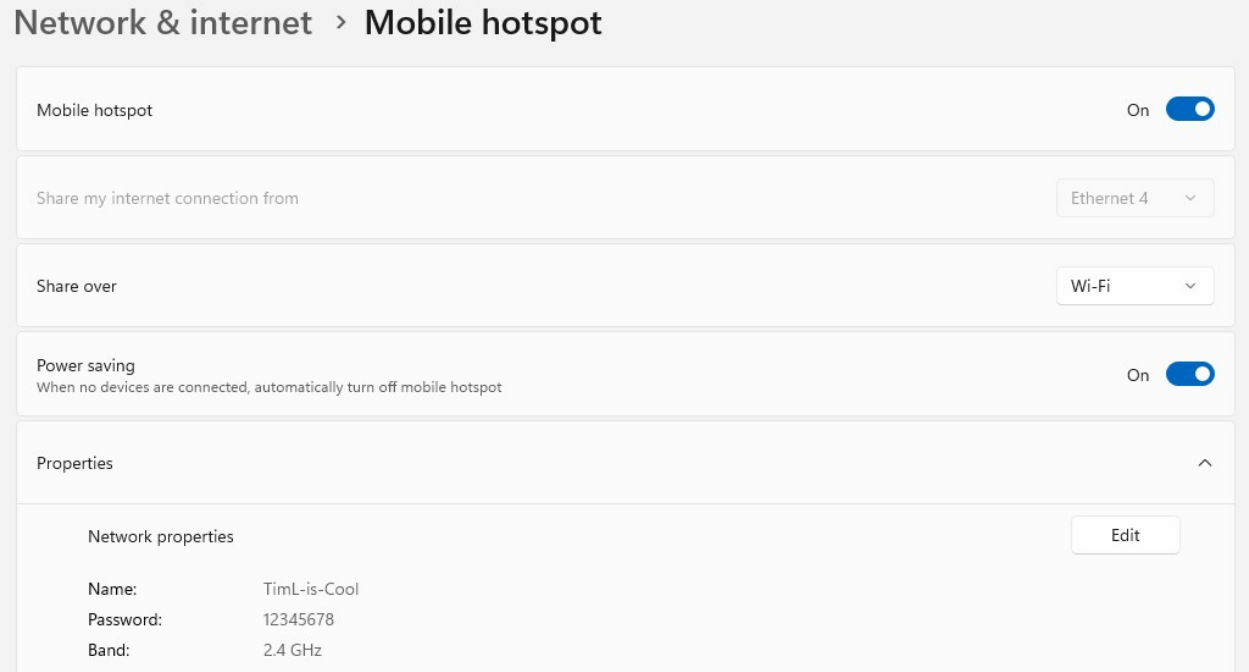
☐ In the Arduino IDE, open a new example Sketch: File, Example, WIFI, WIFI Scan.

☐ Upload the Sketch, open the Serial Monitor and observe the WIFI networks.

### *WIFI Connect*

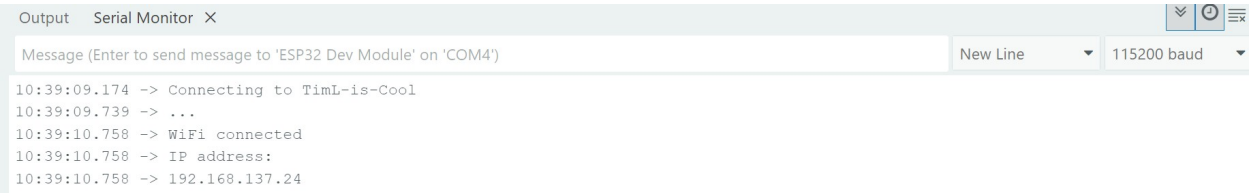
☐ Download the WIFI Connect Code here: WIFI Connect Test Code

☐ Turn on your WIFI Hot Spot on your PC




*WIFI Hot Spot*

☐ Update the ESP32 WIFI Connect code with your SSID and Password




*WIFI Connect Verification*

☐ Upload the code to the ESP32 and verify the ESP32 connects to WIFI using the Serial Monitor.

***WIFI Simple Time***

☐ In the Arduino IDE, Load the Example Sketch: File, Examples, ESP32, Time, Simple Time. Upload the Sketch and observe the Serial Monitor.

☐ (Challenge) Modify the code to display local time.

- ☐ (Challenge) Modify the code to display on the OLED.

  Turn off the Mobile Hot Spot and observe the ESP32 and OLED.

***WIFI Additional Information***

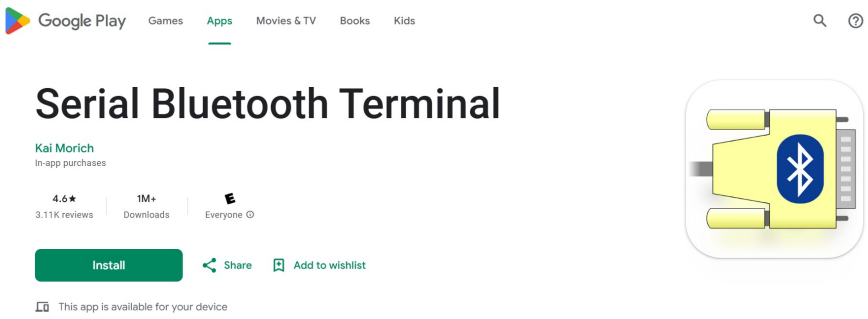
https://dronebotworkshop.com/wifimanager/

https://packetsender.com/download

**Bluetooth Classic**

***Serial to Serial Bluetooth***

- ☐ Start a new sketch: File, Examples, BluetoothSerial, SerialToSerialBT.
- ☐ Rename the Bluetooth device here:
  - ☐ String device_name = ~~"ESP32-BT-Slave"~~;
  - ☐ String device_name = "TimL-is-Cool";
- ☐ Upload the sketch
- ☐ Open the Serial Monitor in the Arduino IDE




*Serial Bluetooth Terminal Application*

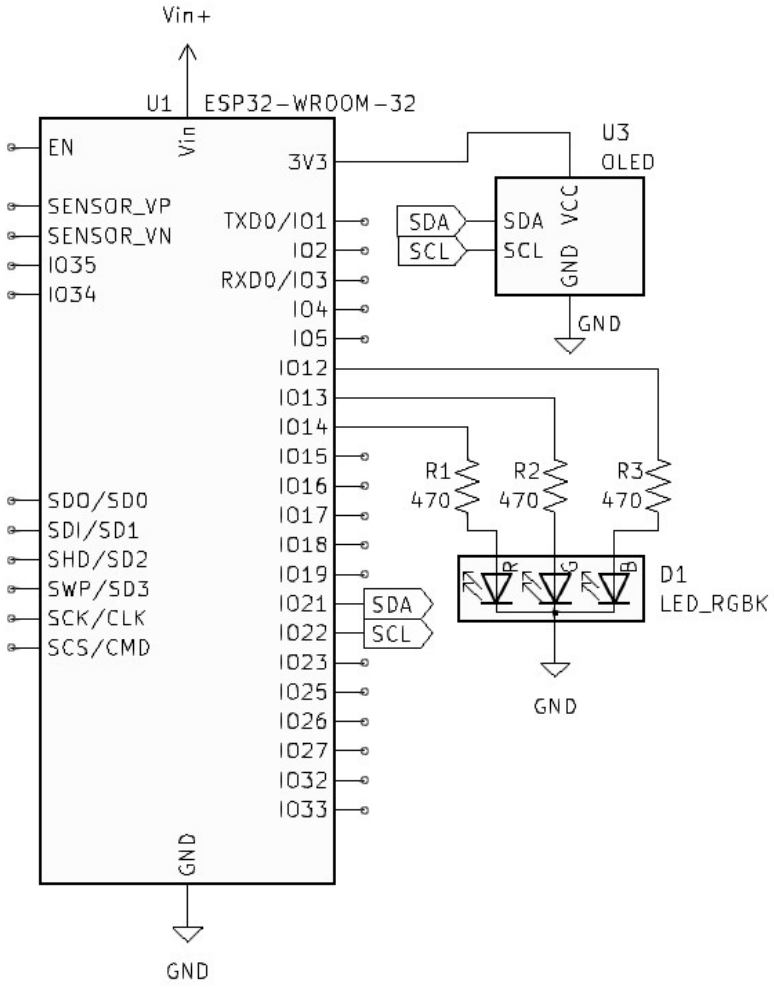
- ☐ Download the Serial Bluetooth Terminal application on your phone
- ☐ Pair your phone with the ESP32 via Bluetooth.

☐ Open the Serial Bluetooth Terminal application on your phone and connect the app to Devices, Bluetooth Classic, and connect your ESP32 ("TimL-is-Cool").

☐ In the Terminal type and send the message "From my phone" and observe the Serial Monitor in the Arduino IDE.

☐ In the Serial Monitor type and send the message "From my computer" and observe the Terminal Monitor.

***Bluetooth OLED***




*Bluetooth OLED Schematic*

***Bluetooth OLED RGB***

- ☐ (Challenge) Modify the previous Serial to Serial Bluetooth code to display both the Serial Monitor Messages and the Bluetooth Messages on the OLED.
- ☐ (Challenge) Modify the previous Serial to Serial Bluetooth OLED code to:
  - ☐ Use Bluetooth from Phone to turn on LEDs when the following messages are received ($RGB combination: Capial letters = on and lower case letters = off):

    $Rgb = Red LED on

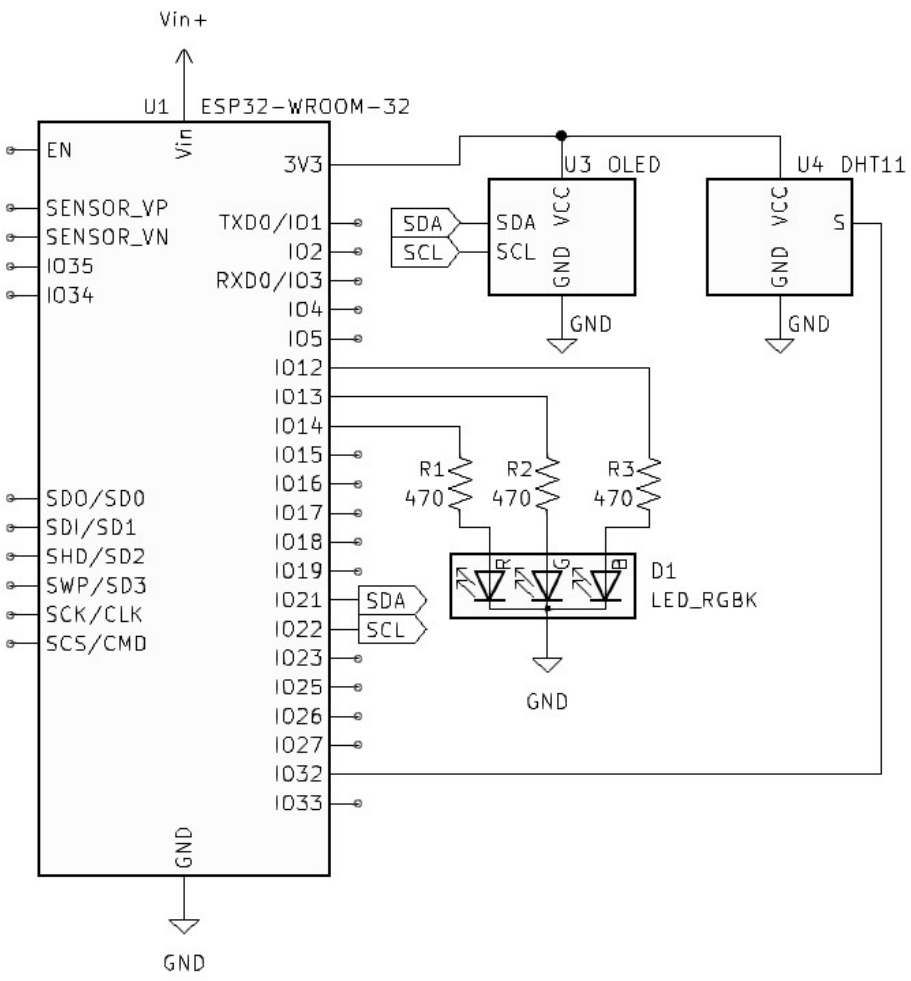
    $rGb = Green LED on

    $rgB = Blue LED on

    $RGB = All LEDs on

    $rgb = All LEDs off

***Bluetooth OLED Temperature and Humidity***




*Bluetooth OLED RGB DHT11 Schematic*

☐ (Challenge) Modify the previous schematic and sketch code using the DHT11 Temperature and Humidity Sensor to acquire Temperature and Humidity data via Bluetooth.

## Functions

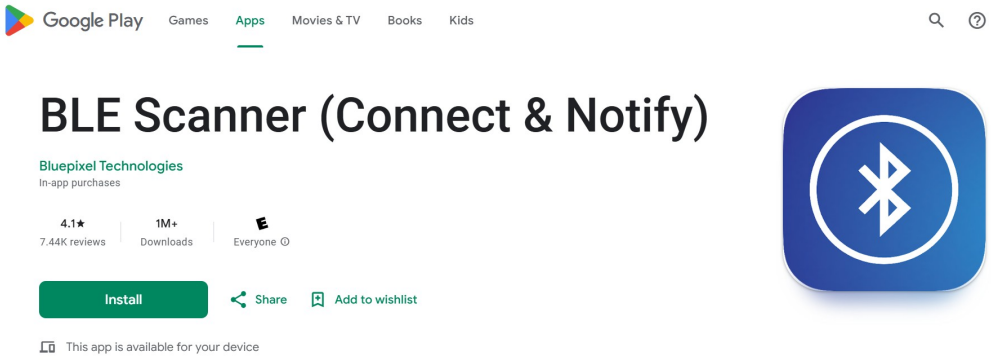
### *What are Functions?*

In ESP32 Arduino IDE programming, functions are essential building blocks that allow you to organize and reuse your code efficiently. A function is a named block of code designed to perform a specific task—such as reading a sensor, turning on an LED, or sending data over Wi-Fi. You can define your own custom functions, in addition to using built-in ones provided by the Arduino framework. For example, the special functions setup() and loop() are always present in an Arduino sketch: setup() runs once at startup, while loop() runs repeatedly. By breaking your program into functions, you make it more readable, easier to debug, and more modular—especially helpful when working on complex ESP32 projects involving multiple inputs, outputs, or communication protocols.

### *Bluetooth OLED RGB Temperature and Humidity with Functions*

- ☐ (Challenge) In the previous Bluetooth section, DHT11 Temperature and Humidity integration, my code in the Loop became cumbersome to read. The challenge is to modify the code by writing custom functions that will clean up and make the loop code easier to read.

## BLE Bluetooth Low Energy




*BLE Application*

### *BLE GATT*

Bluetooth Low Energy (BLE) uses a structure called GATT—short for Generic Attribute Profile—to organize and manage how data is exchanged between devices. GATT defines the way two BLE devices communicate: one acts as a server (which holds the data), and the other as a client (which requests or receives the data).

Data in GATT is structured in a hierarchy of services and characteristics. A service is like a category (e.g., heart rate), and inside each service are characteristics, which are individual pieces of data (like the current heart rate value). Characteristics can be read, written, or notified, depending on their settings. This structure makes GATT flexible and efficient, and it's used in many BLE applications like fitness trackers, smart locks, and environmental sensors.

More on BLE GATT here: Bluetooth GATT: How to Design Custom Services & Characteristics


*Online UUID Generator*

### *BLE Write*

- ☐ In the Arduino IDE, open a new Sketch: File, Examples, BLE, Write.
- ☐ Modify the following code:
  - ☐ #define SERVICE_UUID "~~4fafc201-1fb5-459e-8fcc-c5c9c331914b~~" Replace the

UUID with a unique code generated by the Online UUID Generator.

- ☐ #define CHARACTERISTIC_UUID "~~beb5483e-36e1-4688-b7f5-ea07361b26a8~~" Replace the UUID with a unique code generated by the Online UUID Generator.
- ☐ BLEDevice::init("~~MyESP32~~"); Rename the BLEDevice
- ☐ Upload the Sketch and use the BLE Scanner App to connect to the ESP32. Once connected use the Custom Service to Write and send a text message to the ESP32, verify the message is displayed in the Serial Monitor.

☐ (Challenge) Modify the Sketch to allow for two separate BLE Write services.

☐ (Challenge) Modify the Circuit and the Sketch use the OLED to display the BLE messages sent from your phone.

**MIT App Inventor**

Design a MIT App Inventor application to turn on and off the ESP32 on-board led.

☐ Follow the instructions in the video:

https://www.youtube.com/watch?v=w5LgLsCumFI&t=32s

☐ (Challenge) Modify the code to control an external LED and or other function from your phone.

**Bluepad32**

https://github.com/ricardoquesada/bluepad32

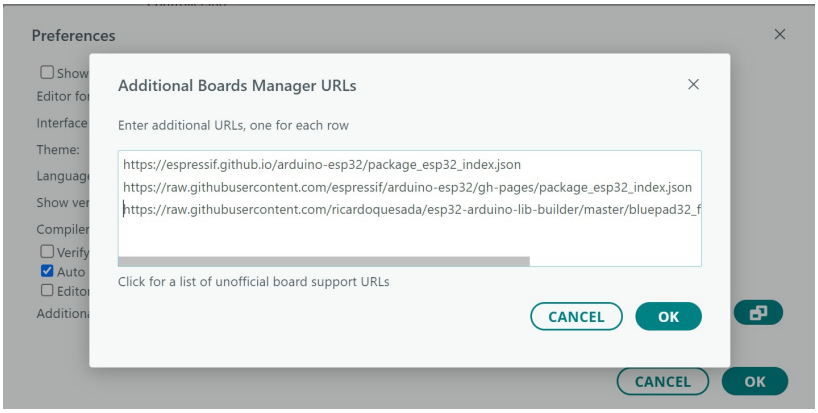
https://www.youtube.com/watch?v=0jnY-XXiD8Q

Amazon: Megadream Android Gamepad Bluetooth Controller

AlliExpress: X3/T3 Wireless Gamepad

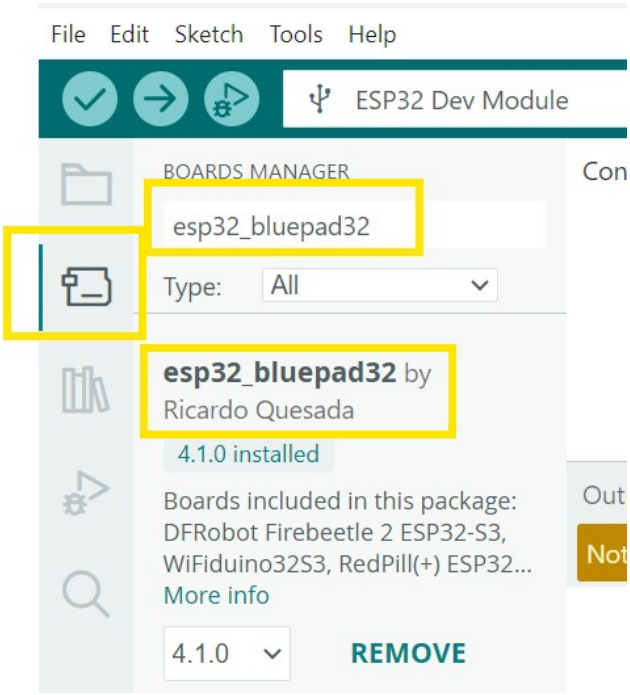
☐ In the Arduino IDE, goto File, Preferences, Additional Boards Managers URL: and add the following URL.

https://raw.githubusercontent.com/ricardoquesada/esp32-arduino-lib-builder/master/bluepad32_files/package_esp32_bluepad32_index.json



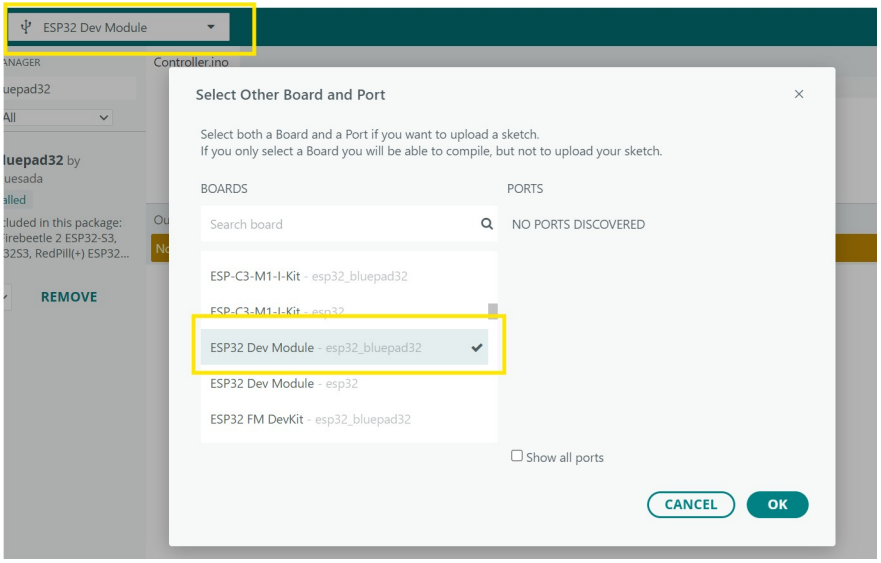

*bluepad32 Additional Boards Managers URL add*

☐ Go to the boards manager icon on the left and search and install the esp32_bluepad32.



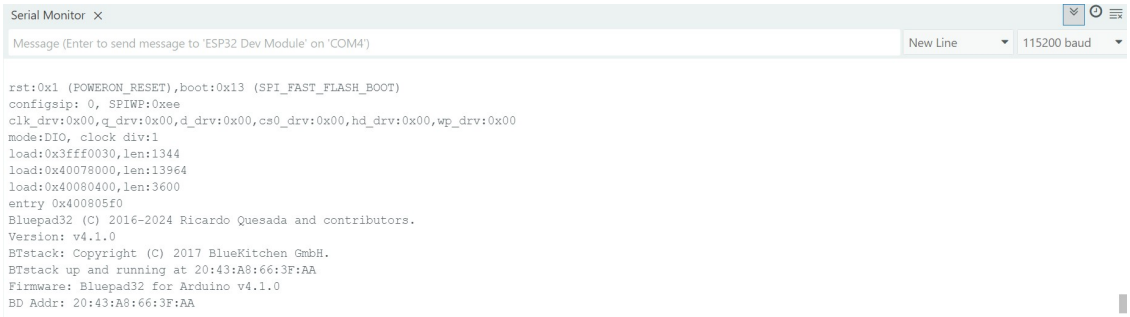

*bluepad32 Additional Boards Manager Install*

☐ Select Other Boards and Ports, find and select the ESP32_Dev Module - esp32_bluepad32




*ESP32 Dev Module - esp32_bluepad32*

☐ In the Arduino IDE, goto File, Examples, Bluepad32_ESP32, Controller. Upload the example sketch and open the Serial Monitor and press the EN button on the ESP32.




*Controller Serial Monitor, EN button push*

☐ Using the X3/T3 Wireless Gamepad, (first time pairing) press and hold the X button, while holding X, press the HOME button. After the controller has been paired to the ESP32 for the first time, the controller should only need to be turned on (HOME button press) to pair.

☐ (Challenge) Identify all the variables associated with each button/control on the controller and modify the code to turn on an LED when a specific button is pressed.

☐ (Challenge) Modify the code so that a joystick can control a Servo Motor and ESC-controlled DC Motor.