LEADING THE WAY
KHALĪFAH • AMĀNAH • IQRA' • RAHMATAN LIL-ĀLAMĪN
LEADING THE WORLD

AN INTERNATIONAL AWARD-WINNING INSTITUTION FOR SUSTAINABILITY

IIUM ROBOTEAM

ROBOTIC WORKSHOP 11.0 MODULE

PART 1:

MICROCONTROLLERS & ARDUINO IDE BASICS

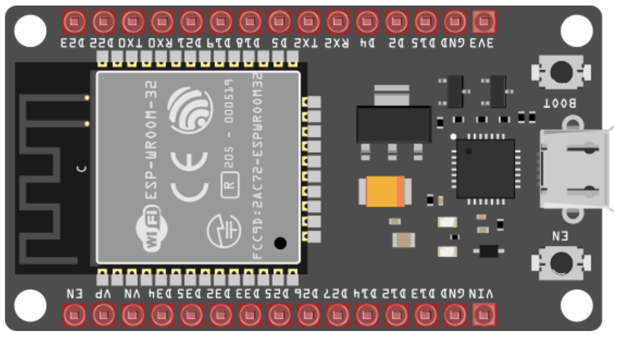
BASIC MODULE

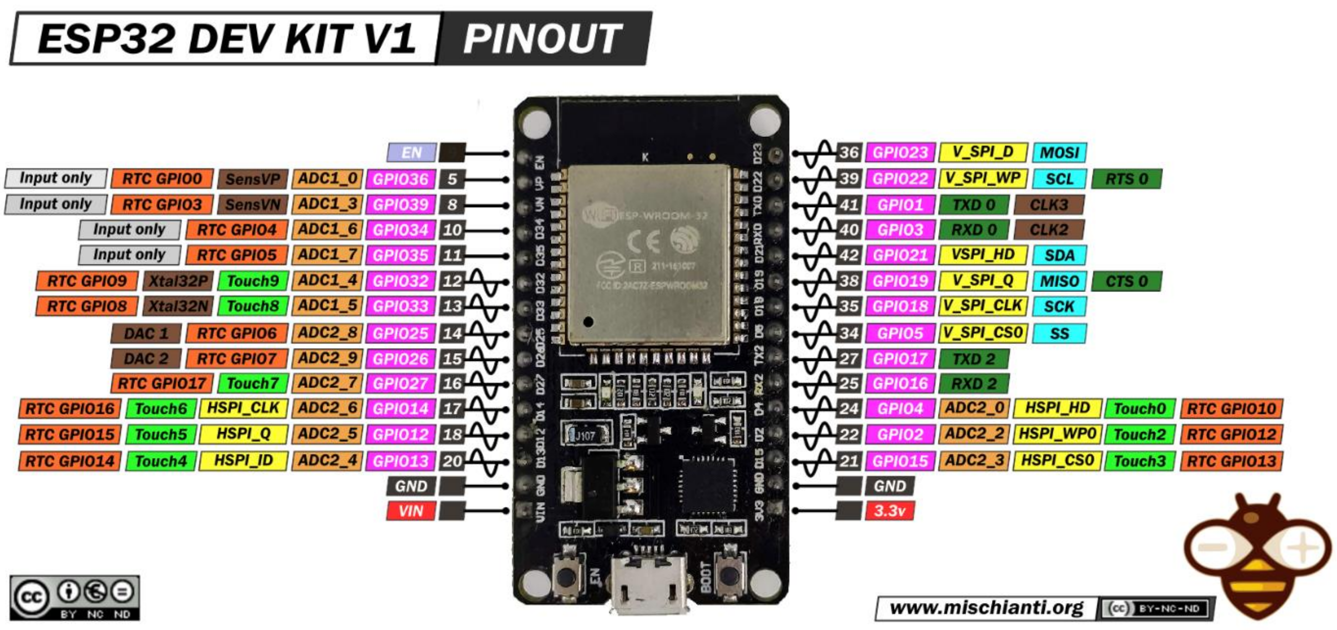
1. Introduction to ESP 32

1.1 ESP 32

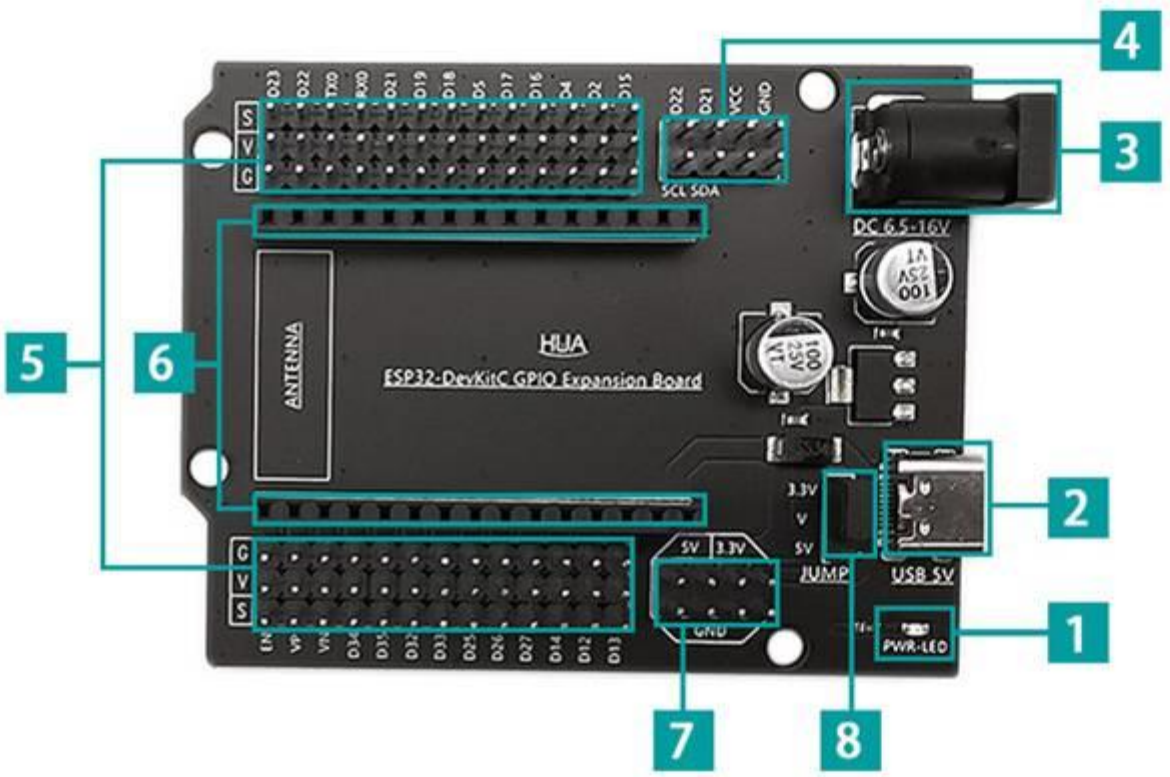
The ESP32 is a low-cost microcontroller with built-in Bluetooth and WiFi capabilities. The ESP32 is dual-core, meaning it has two processors that can run simultaneously and perform tasks independently, making it a powerful device for IoT applications. It is also equipped with various peripherals, such as multiple analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and pulse-width modulators (PWMs), which enable it to interact with a variety of sensors and actuators. In addition, the ESP32 has a rich set of development tools and a strong community, making it a popular choice for IoT projects.

The DOIT ESP32 DEV KIT V1 :




ESP32 DEV KIT V1 PINOUT
EN
Input only | RTC GPIO0 | SensVP | ADC1_0 | GPIO36 | 5
Input only | RTC GPIO3 | SensVN | ADC1_3 | GPIO39 | 8
Input only | RTC GPIO4 | ADC1_6 | GPIO34 | 10
Input only | RTC GPIO5 | ADC1_7 | GPIO35 | 11
RTC GPIO9 | Xtal32P | Touch9 | ADC1_4 | GPIO32 | 12
RTC GPIO8 | Xtal32N | Touch8 | ADC1_5 | GPIO33 | 13
DAC 1 | RTC GPIO6 | ADC2_8 | GPIO25 | 14
DAC 2 | RTC GPIO7 | ADC2_9 | GPIO26 | 15
RTC GPIO17 | Touch7 | ADC2_7 | GPIO27 | 16
RTC GPIO16 | Touch6 | HSPI_CLK | ADC2_6 | GPIO14 | 17
RTC GPIO15 | Touch5 | HSPI_Q | ADC2_5 | GPIO12 | 18
RTC GPIO14 | Touch4 | HSPI_ID | ADC2_4 | GPIO13 | 20
GND
VIN
36 | GPIO23 | V_SPI_D | MOSI
39 | GPIO22 | V_SPI_WP | SCL | RTS 0
41 | GPIO1 | TXD 0 | CLK3
40 | GPIO3 | RXD 0 | CLK2
42 | GPIO21 | VSPI_HD | SDA
38 | GPIO19 | V_SPI_Q | MISO | CTS 0
35 | GPIO18 | V_SPI_CLK | SCK
34 | GPIO5 | V_SPI_CS0 | SS
27 | GPIO17 | TXD 2
25 | GPIO16 | RXD 2
24 | GPIO4 | ADC2_0 | HSPI_HD | Touch0 | RTC GPIO10
22 | GPIO2 | ADC2_2 | HSPI_WP0 | Touch2 | RTC GPIO12
21 | GPIO15 | ADC2_3 | HSPI_CS0 | Touch3 | RTC GPIO13
GND
3.3v
www.mischianti.org

1.2 ESP32 GPIO Expansion Board



1. POWER INDICATOR	5. IO EXTENSION PIN
2. USB 5V CONNECTOR	6. ESP32 BOARD'S SOCKET
3. DC 6.5V – 16V POWER CONNECTOR	7. 5V/3.3V VOLTAGE PIN
4. I2C EXTENSION PIN （2 GOUP）	8. JUMP – SELECT V EITHER 3.3V OR 5V

1.3 Jumper

MALE TO MALE

MALE TO FEMALE

FEMALE TO FEMALE

1.4 Installation of Extension

Tutorial Website:
https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

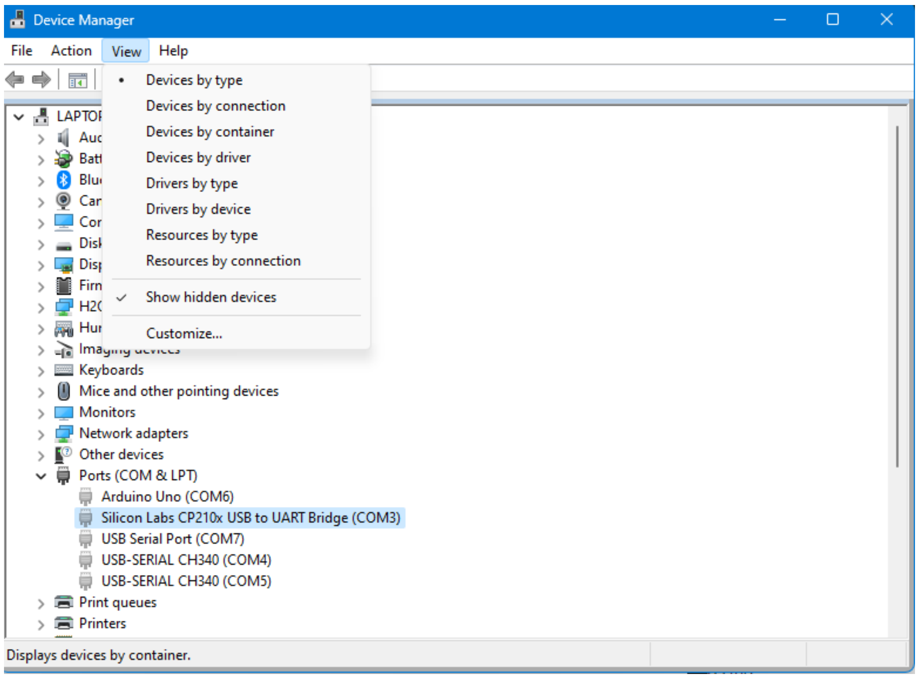
ESP 32 Additional Boards Manager URLs:
Link : *https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json*

Alternative link:
https://dl.espressif.com/dl/package_esp32_index.json

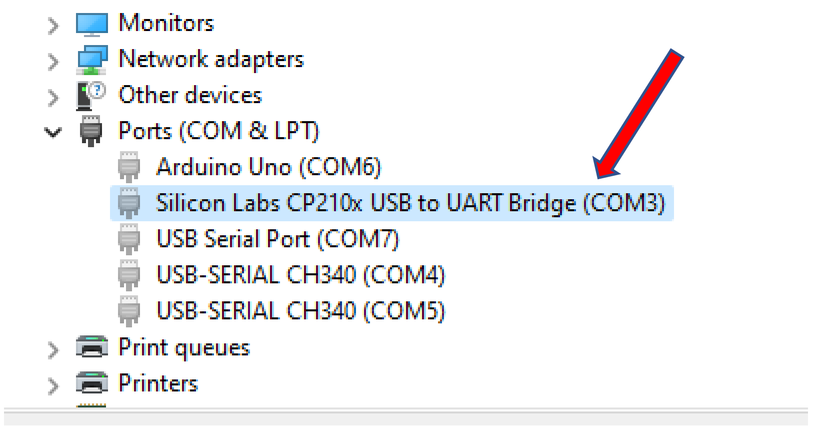
1.5 Com Port Driver: **Silicon Labs CP210x Driver** *(if applicable)*

How to check your serial port:

1. Plug-in ESP 32 using micro USB cable
2. Open Device Manager
3. View - Show Hidden Devices



4. Check Ports (COM & LPT)

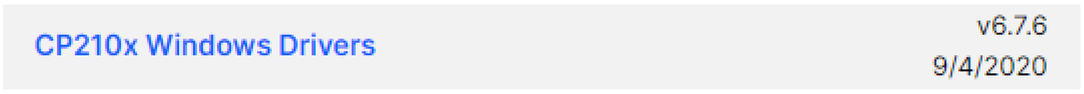


- Monitors
- Network adapters
- Other devices
- Ports (COM & LPT)
  - Arduino Uno (COM6)
  - Silicon Labs CP210x USB to UART Bridge (COM3)
  - USB Serial Port (COM7)
  - USB-SERIAL CH340 (COM4)
  - USB-SERIAL CH340 (COM5)
- Print queues
- Printers

5. Install the driver using the link below if **Silicon Labs CP210x** doesn't appear in your device.

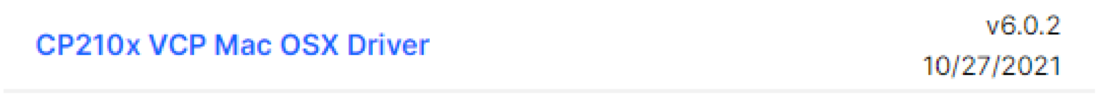
Link: *https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads*

For Windows:



CP210x Windows Drivers v6.7.6 9/4/2020

For Mac OS:

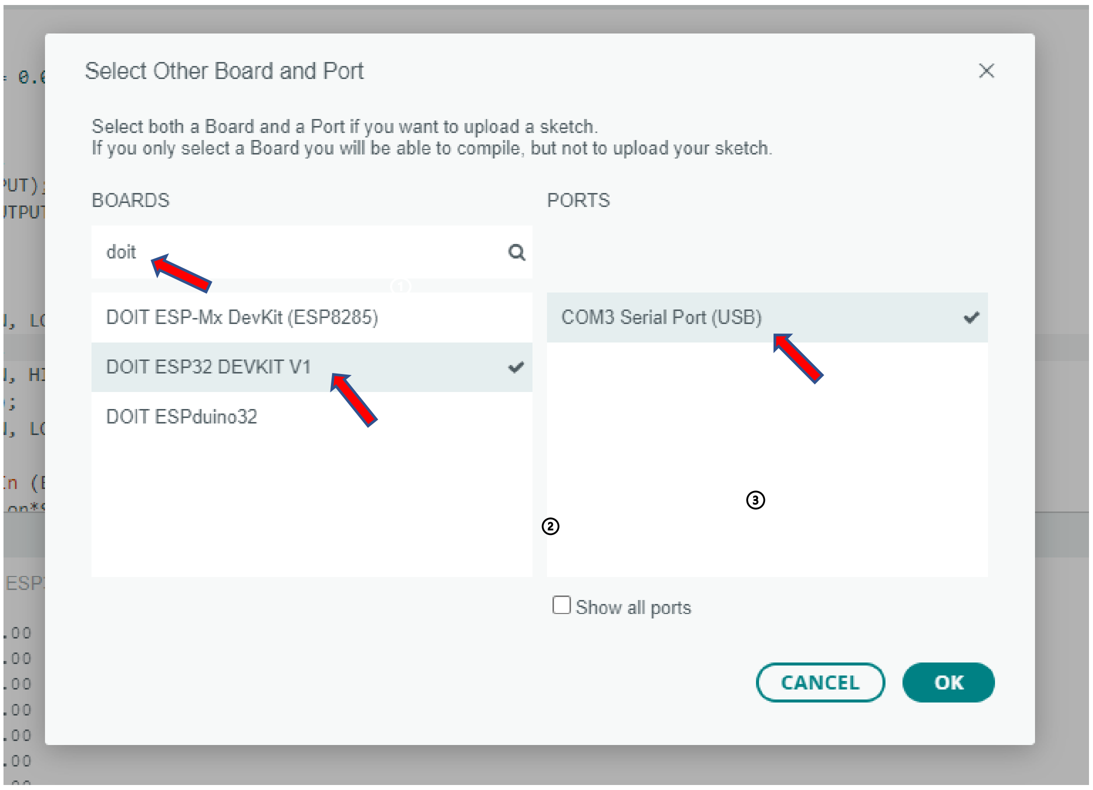


CP210x VCP Mac OSX Driver v6.0.2 10/27/2021

1.6 Selecting board in IDE

1. Plug-in ESP 32 using micro USB
2. Choose **DOIT ESP32 DEVKIT V1** and it's respective **COM PORT**

1.7 Upload to board



Select Other Board and Port

Select both a Board and a Port if you want to upload a sketch.
If you only select a Board you will be able to compile, but not to upload your sketch.

BOARDS

doit

- DOIT ESP-Mx DevKit (ESP8285)
- DOIT ESP32 DEVKIT V1 ✓
- DOIT ESPduino32

PORTS

- COM3 Serial Port (USB) ✓

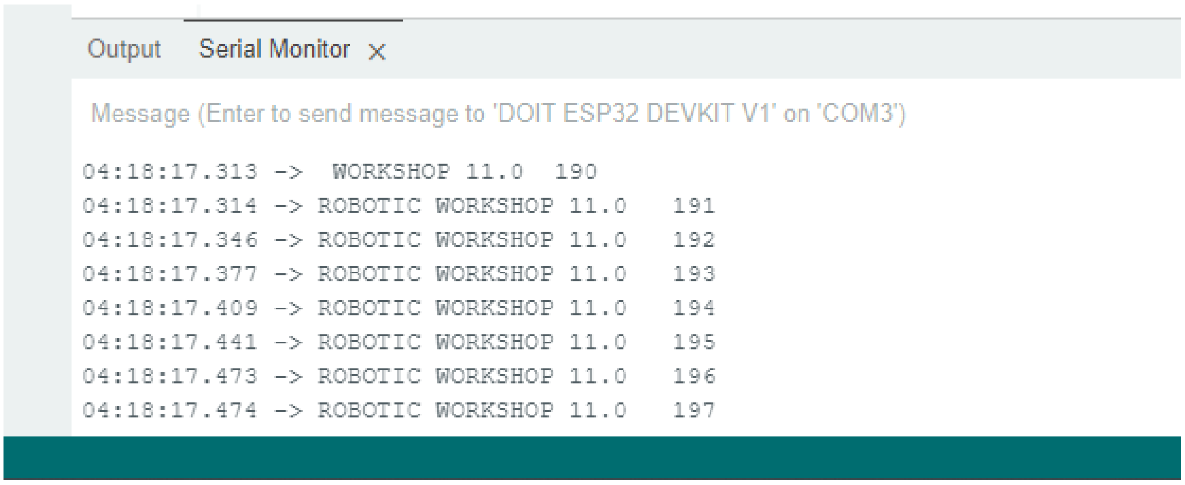
☐ Show all ports

2. Serial Monitor

2.1 Serial Print

```
//SERIAL PRINT

int count;

void setup() {
  // put your setup code here, to run once:
  Serial.begin (9600) ;     // open serial port to sent data
                            // back to the computer at 9600 bps
  Serial.println("IIUM ROBOTEAM\n");
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.print("ROBOTIC WORKSHOP 11.0\t");
  Serial.println(count);
  count++;
}
```

Example of expected output in Serial Monitor:



Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM3')

```
04:18:17.313 ->  WORKSHOP 11.0  190
04:18:17.314 -> ROBOTIC WORKSHOP 11.0   191
04:18:17.346 -> ROBOTIC WORKSHOP 11.0   192
04:18:17.377 -> ROBOTIC WORKSHOP 11.0   193
04:18:17.409 -> ROBOTIC WORKSHOP 11.0   194
04:18:17.441 -> ROBOTIC WORKSHOP 11.0   195
04:18:17.473 -> ROBOTIC WORKSHOP 11.0   196
04:18:17.474 -> ROBOTIC WORKSHOP 11.0   197
```

2.2 Reading input from Serial Monitor – char

This example will show output in serial monitor when character a is entered.

```
//READ SERIAL CHAR
void setup() {
   Serial.begin (115200);  // open serial port to send data
                           // back to the computer at 115200
bps
}

void loop() {
  // if there is input from serial monitor
  if (Serial.available()) {
    //read input character from serial monitor
    char inputChar = Serial.read();
    Serial.println(inputChar);
    if (inputChar == 'a')
       Serial.println ("Character a has been entered");
  }
}
```

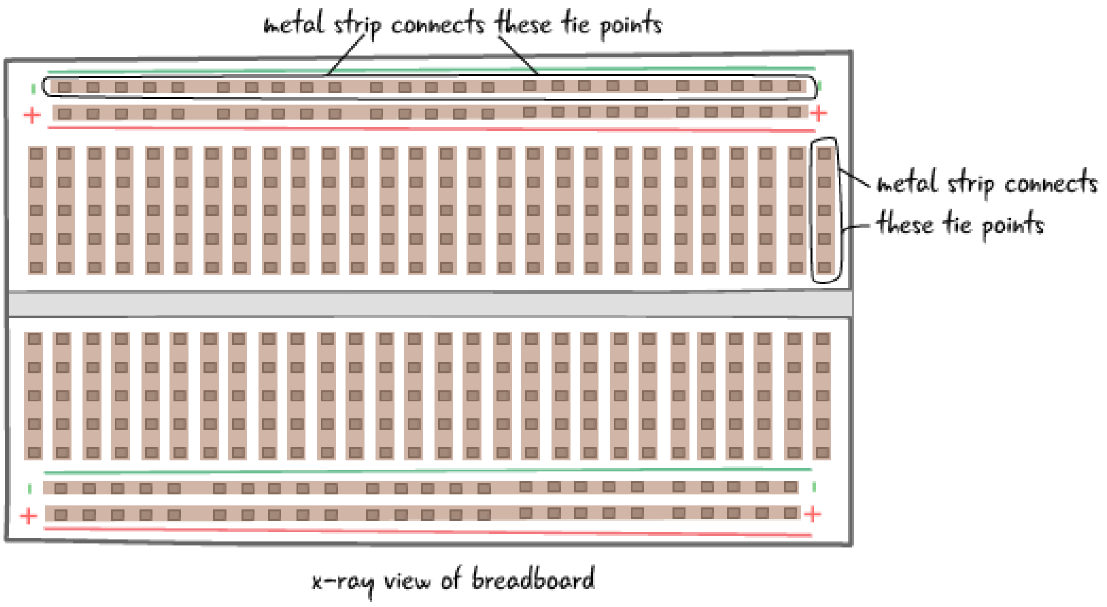
2.3 Reading input from Serial Monitor – int

This example will show output in serial monitor when number 2 is entered.

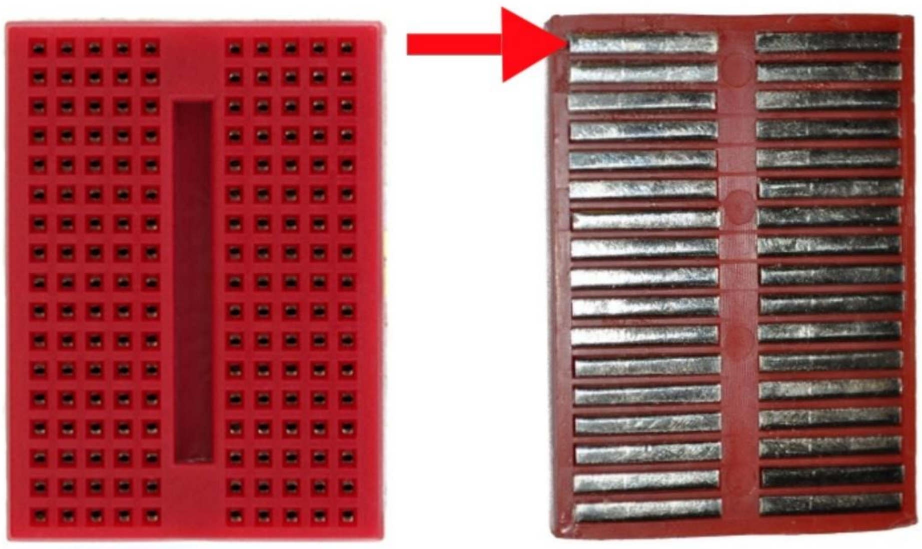
```
// READ SERIAL INT
void setup() {
  Serial.begin (115200); //open serial port to send data
                         //back to the computer at 115200 bps
}

void loop() {
  // if there is input form serial monitor
  if (Serial.available() ){
    // read input character from serial monitor
    int inputInt = Serial.parseInt();
    Serial.println (inputInt);
    if (inputInt == 2)
       Serial.println ("Integer 2 has been entered");
  }
}
```

3. Breadboard

Anatomy of a regular breadboard:

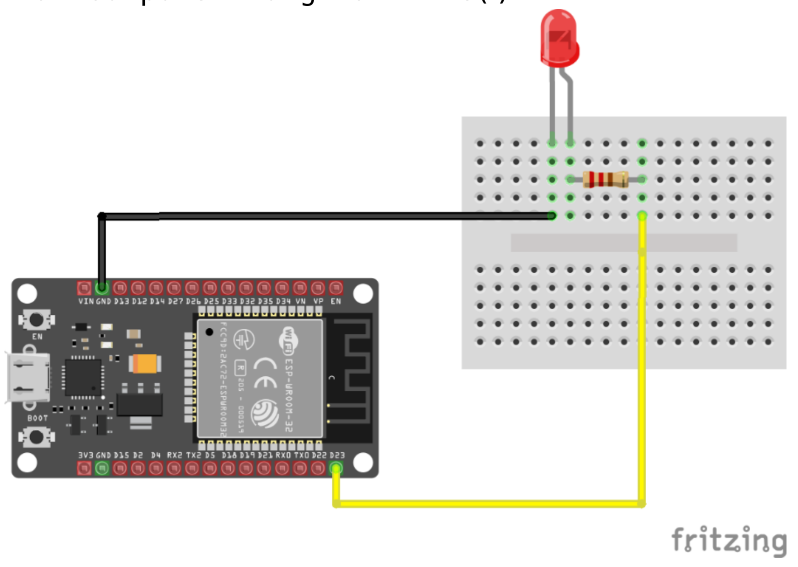

metal strip connects these tie points
metal strip connects these tie points
x-ray view of breadboard

A mini breadboard with it’s adhesive removed:



4. GPIO

4.1 Using digital outputs – digitalWrite()



This example will blink the led every one second :

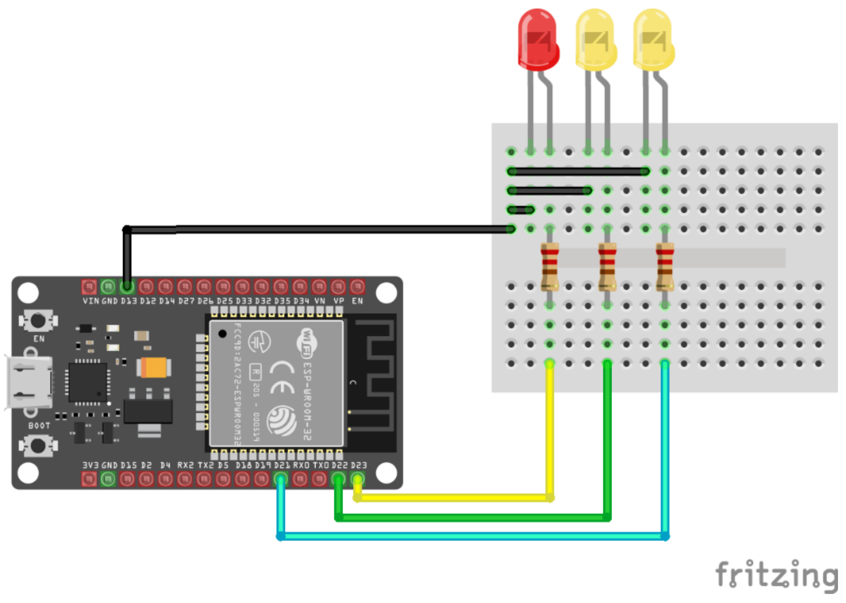
```
// LED BLINK
// LED connected to digital pin 23
const int ledPin = 23;

// the setup function runs once when you press reset,
// power the board or open serial  monitor
void setup () {
   // initialize LED_PIN as an output
   pinMode (ledPin, OUTPUT );
}

// the loop function runs over and over again
void loop() {
   digitalWrite (ledPin, HIGH);     //turn LED on (HIGH)
   delay(1000);                     //wait for a second
   digitalWrite (ledPin, LOW);      //turn the LED off (LOW)
   delay(1000);                     //wait for a second
}
```

Activity 1: Blink 3 leds using delay

Next let’s try to blink 3 leds with the following conditions:

The red led every 1 second, green every 3 seconds and blue every 5 seconds.

Example circuit diagram:


VIN GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23
EN
BOOT
ESP-WROOM-32
fritzing

4.2 millis()

This example will print current millis.

```
// millis()

unsigned long currentMillis;
void setup() {
  // open Serial port
  Serial.begin (115200);
}

void loop() {
  // store current time
  currentMillis = millis();
  //print current time
  Serial.println (currentMillis);
}
```

4.3 Replacing delay() with millis()

```
// Replacing delay() with millis()
// LED BLINK without delay

const int ledPin = 23;

unsigned long previousMillis;
int ledState;

void setup () {
   // initialize LED_PIN as an output
   pinMode (ledPin, OUTPUT);
}

void loop() {
   if (millis() - previousMillis > 1000) {
      ledState = !ledState;
      digitalWrite (ledPin, ledState);
      // store the last time you blink the LED
      previousMillis = millis();
   }
}
```

Aktivity 2: Blink 3 leds using millis()

Next let’s try to blink 3 leds with the following conditions but using the millis() instead of delay():

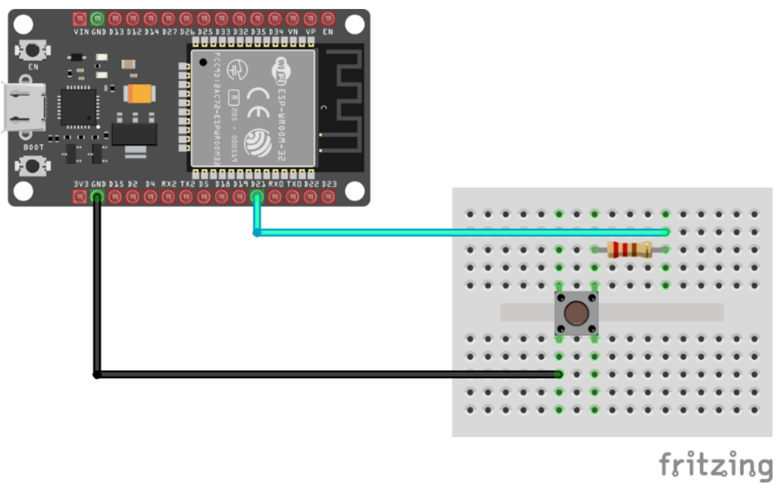
The red led every 1 second, green every 3 seconds and blue every 5 seconds.

Answer :

```
// Replacing delay() with millis()
// LED BLINK without delay

int ledPin = 23;
int ledPin2 = 22;
int ledPin3 = 21;

unsigned long timer[3];

void setup() {
  // initialize LED_PIN as an output
  pinMode(ledPin, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
}

void loop() {
  if (millis() - timer[1] > 1000) {
    digitalWrite(ledPin, HIGH);
    delay(50);
    digitalWrite(ledPin, LOW);
  }
  if (millis() - timer[1] > 1500) {
    digitalWrite(ledPin2, LOW);
    timer[1] = millis() - 500;
  }
  if (millis() - timer[2] > 3000) {
    digitalWrite(ledPin2, HIGH);
  }
  if (millis() - timer[2] > 3500) {
    digitalWrite(ledPin2, LOW);
    timer[2] = millis() - 500;
  }
  if (millis() - timer[3] > 5000) {
    digitalWrite(ledPin3, HIGH);
  }
  if (millis() - timer[3] > 5500) {
    digitalWrite(ledPin3, LOW);
    timer[3] = millis() - 500;
  }
}
```

4.4 Digital Read – Push Button

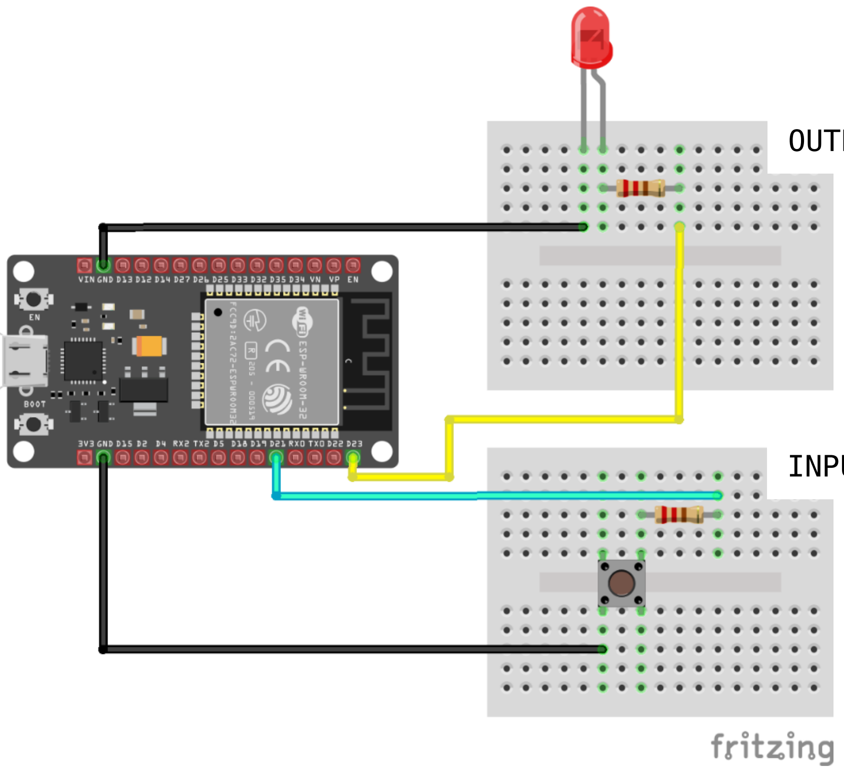
This example will show output high when the push button closed



```
//PUSH BUTTON
// push button connected to digital pin 21
const int buttonPin = 21;

void setup() {
   Serial.begin (115200);
   // initialize  BUTTON_PIN as input
    pinMode (buttonPin, INPUT_PULLUP);
}
void loop(){
    // read button state either HIGH ot LOW (1 or 0)
    int buttonState = digitalRead (buttonPin);

    // print button state
    Serial.println (buttonState);
}
```

4.5 Using digital inputs and outputs together

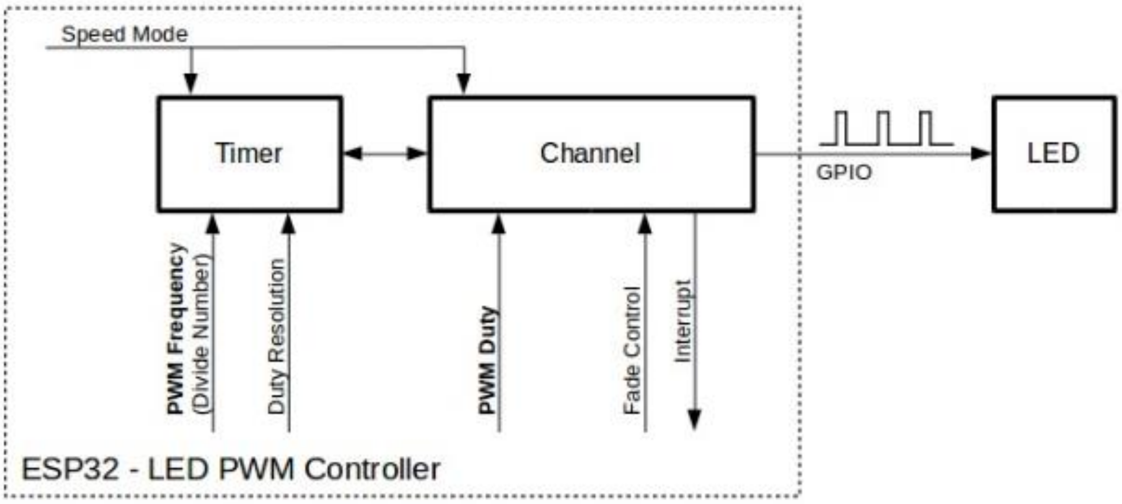
This example will turn on LED when the push button is closed



```
// LED & PUSH BUTTON
// LED connected to digital pin 23
const int ledPin = 23;
// push button connected to digital pin 21
const int buttonPin = 21;

void setup(){
  Serial.begin (115200);
   // initialize LED_PIN as output
   pinMode (ledPin, OUTPUT);
   // initialize BUTTON_PIN as input
   pinMode (buttonPin, INPUT_PULLUP);
}

void loop(){
   // read button state either HIGH or LOW (1 or )
   int buttonState = digitalRead (buttonPin);

   if (!buttonState)
       digitalWrite (ledPin, HIGH);
   else
       digitalWrite (ledPin, LOW);
   Serial.println (buttonState);
}
```

5. Analog Input and Output

The ESP32 has 16 PWM channels that can be assigned to any output pins.

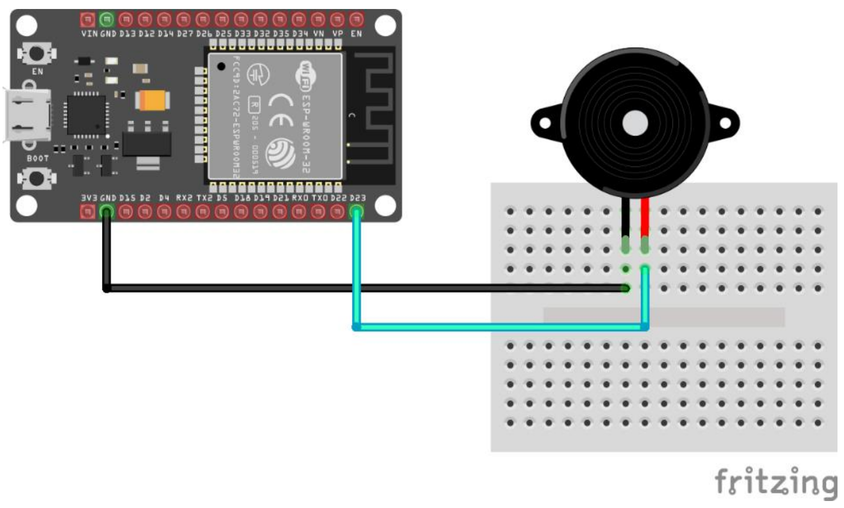


Key Settings of LED PWM Controller's API

We will utilise the ESP32's Led Control (LEDC) functions to manipulate the pwm values.

5.1 Analog Write – Buzzer

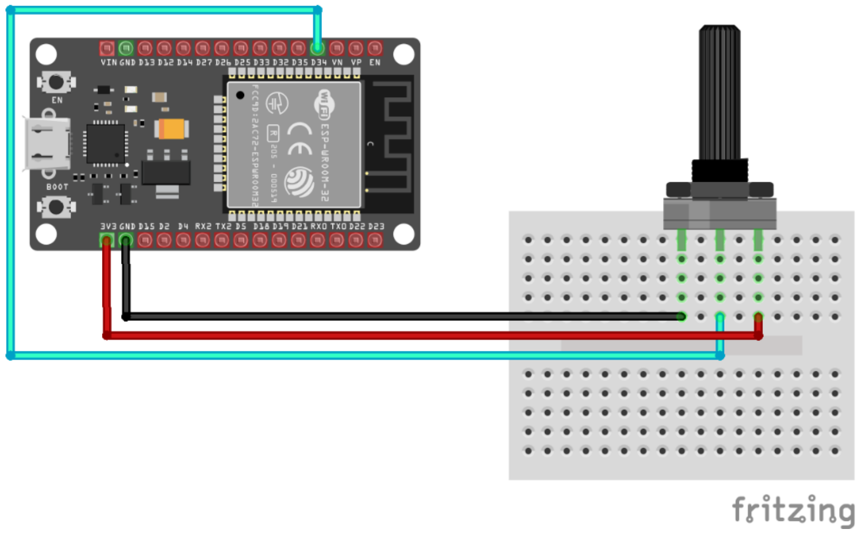
This example will turn on the buzzer



```
const int ledPin = 23;

// PWM channel 0 parameter
const int freq = 5000; // 5000 Hz
const int ledChannel = 0;
const int resolution = 8; // 8-bit resolution

void setup(){
    // Configure the channel 0
    ledcSetup(ledChannel, freq, resolution);

    // Attach the channel 0 on the pin
    ledcAttachPin(ledPin, ledChannel);
}

void loop(){
    // Increase the brightness of the led in the loop
    for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
        ledcWrite(ledChannel, dutyCycle);
        delay(15);
    }
}
```

5.2 Analog Read – Potentiometer

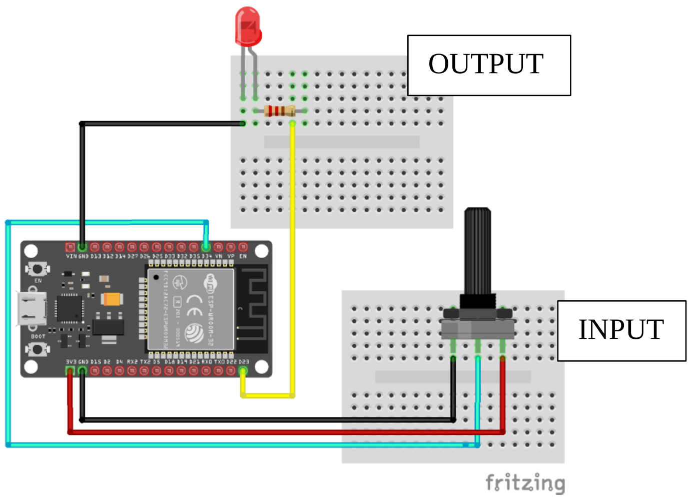
This example will be using potentiometer to control resistant


VIN GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23
EN
BOOT
ESP-WROOM-32
fritzing

```
const int potPin = 13;

// variable for storing the potentiometer value
int potValue = 0;

void setup() {
  Serial.begin(115200);
  delay(1000);
}

void loop() {
  // Reading potentiometer value
  potValue = analogRead(potPin);
  Serial.println(potValue);
  delay(500);
}
```

5.3 Potentiometer + LED

This example will be using potentiometer to control LED brightness



```
const int ledPin = 23;

// PWM channel 0 parameter
const int freq = 5000; // 5000 Hz
const int ledChannel = 0;
const int resolution = 8; // 8-bit resolution

// Potentiometer is connected to GPIO 34 (Analog ADC1_CH6)
input only pin
const int potPin = 34;

// variable for storing the potentiometer value
int potValue = 0;
int pwmVal = 0;

void setup() {
  Serial.begin(115200);
  delay(1000);

  // Configure the channel 0
  ledcSetup(ledChannel, freq, resolution);

  // Attach the channel 0 on the pins
  ledcAttachPin(ledPin, ledChannel);
}
void loop() {
  // Reading potentiometer value
  potValue = analogRead(potPin);
  pwmVal= map(potValue,0,4095,0,255);
  Serial.println(potValue);
  ledcWrite(ledChannel, pwmVal);
  delay(500);
}
```

6. Sensor

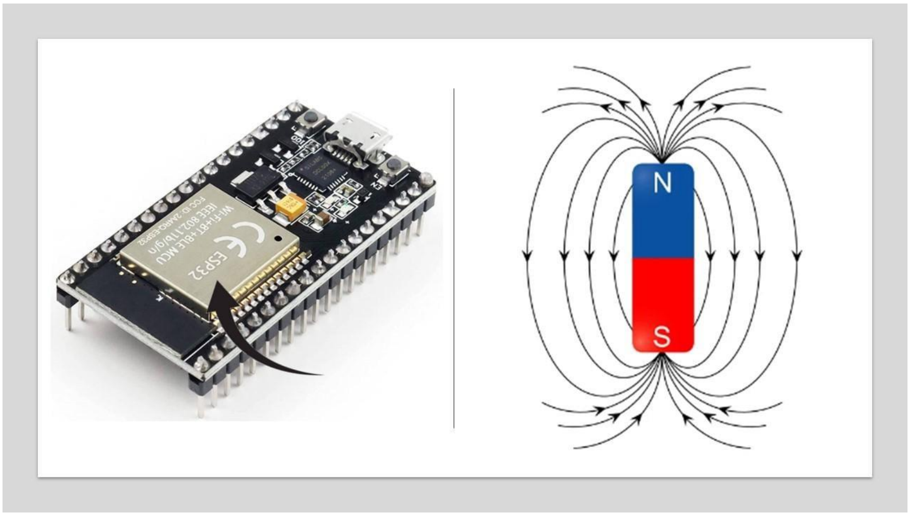
6.1 Introduction to sensors

A sensor is a device that detects and measures a physical property or phenomenon and records, indicates, or otherwise responds to it. Sensors are used in a variety of applications and industries, including robotics, manufacturing, and automation.

Sensors can be used to gather information about the environment, such as the location and movement of objects, as well as the conditions in the Robolab itself, such as temperature and humidity. This information can be used by the robotic system to make decisions, control processes, and ensure the safety of personnel and equipment.

6.2 Hall Sensor

A Hall Sensor is a device that uses the Hall Effect to sense a magnetic field. The ESP32 comes with a hall sensor built-in.



Here is an example sketch to read the value of the sensor:

```
int val = 0;
void setup() {
  Serial.begin(9600);
    }

void loop() {
  // put your main code here, to run repeatedly:
  val = hallRead();
  // print the results to the serial monitor:
  //Serial.print("sensor = ");
  Serial.println(val);//to graph
}
```

6.3 Touch Sensor

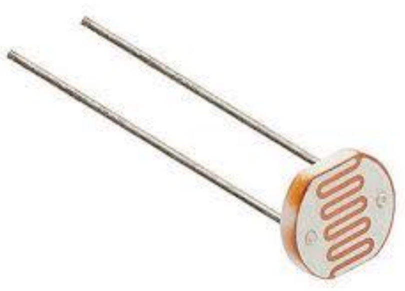
A touch switch is simply a conductive plate that you can activate by touching it. Because this is a capacitive touch switch, touching it changes the capacitance. This difference is measured and used to activate the switch.

On the ESP32, there are ten touch switch inputs that share their function with ten GPIO pins. We're using touch switch 0, which corresponds to GPIO 4.

```
// ESP32 Touch Test
// Just test touch pin - Touch0 is T0 which is on GPIO 4.

void setup()
{
  Serial.begin(115200);
  delay(1000); // give me time to bring up serial monitor
  Serial.println("ESP32 Touch Test");
}

void loop()
{
  Serial.println(touchRead(T0));  // get value using T0
  delay(1000);
}
```

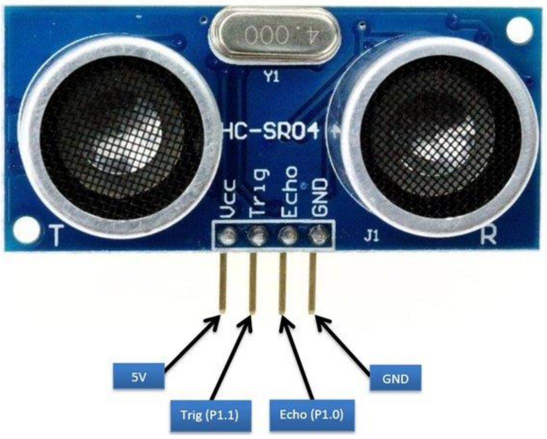
6.4 Photoresistors – Light Dependent Resistors (LDR)

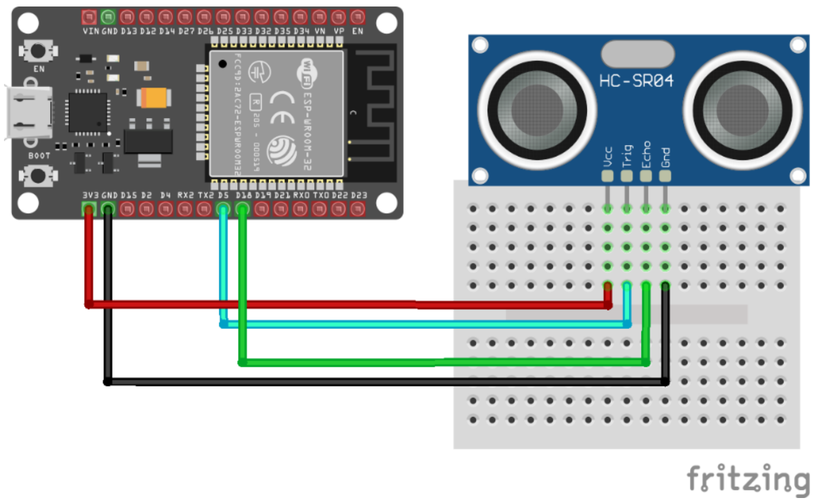
The resistance of the LDR decreases with an increase in light intensity. This property allows us to use them for making light sensing circuits.



6.5 Proximity Sensor - Ultrasonic Sensor (HC-SR04)

Ultrasonic sensors work by emitting sound waves at a frequency too high for humans to hear. They then wait for the sound to be reflected back, calculating distance based on the time required.




VIN GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RXD TXD D22 D23
EN
BOOT
ESP-WROOM-32
HC-SR04
Vcc
Trig
Echo
Gnd
fritzing

This is the example to read distance using ultrasonic sensor

```
const int trigPin = 5;
const int echoPin = 18;

//define sound speed in cm/uS
#define SOUND_SPEED 0.034

long duration;
float distanceCm;

void setup() {
  Serial.begin(115200); // Starts the serial communication
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
}

void loop() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Reads the echoPin, returns the sound wave travel time in
microseconds
  duration = pulseIn(echoPin, HIGH);

  // Calculate the distance
  distanceCm = duration * SOUND_SPEED/2;

  // Prints the distance in the Serial Monitor
  Serial.print("Distance (cm): ");
  Serial.println(distanceCm);

  delay(1000);
}
```

6.6 Accelerometer Sensor – MPU6050

MPU6050 is a Micro Electro-mechanical system (MEMS), it consists of three-axis accelerometer and three-axis gyroscope. It helps us to measure velocity, orientation, acceleration, displacement and other motion like features.

7. Wireless Control

7.1 Bluetooth

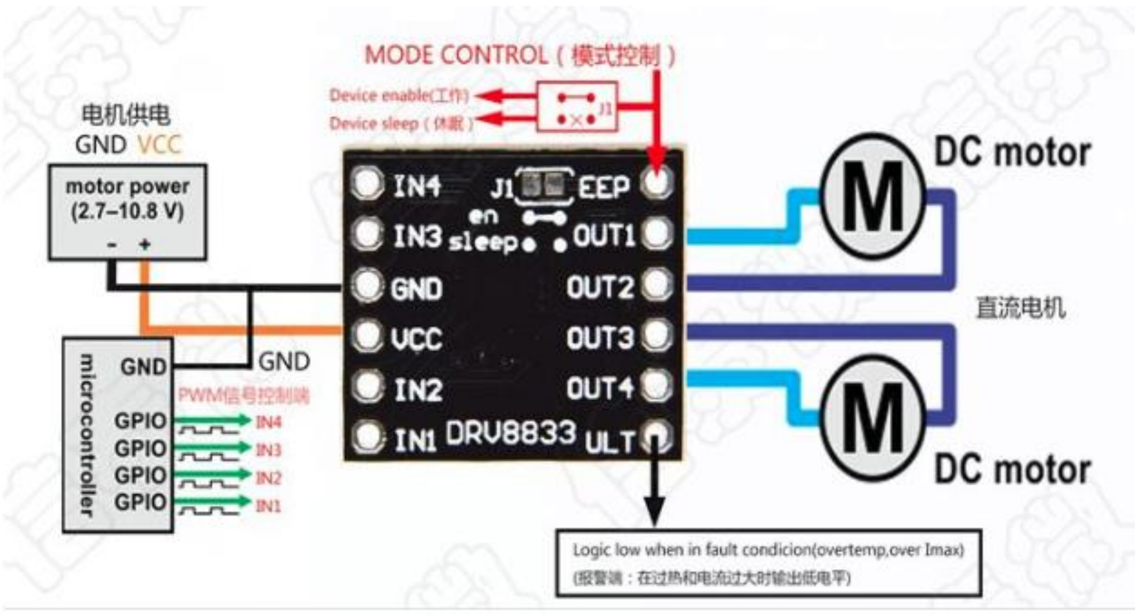
```
#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) ||
!defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to
and enable it
#endif

BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32test"); //Bluetooth device name
  Serial.println("The device started, now you can pair it with
bluetooth!");
}

void loop() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
}
```

8. Motor Driver

A motor driver showcases itself as an interface between the motor and the microcontroller. The reason is that the microcontroller and the motor work on different ranges of voltages. The engine will use up a higher current level than the microcontroller.

SPECIFICATIONS

- Driver Chip: DRV8833
- Power supply voltage (VM): 2.7-10.8V
- Output current: Iout=1.0A(average) / 4A (peak)
- Standby control to save power
- CW/CCW/short brake/stop motor control modes
- Built-in thermal shutdown circuit and low voltage detecting circuit
- Filtering capacitors on both supply lines



This is example to use motor driver

```
int motL1 = 5;   //4 pins for motor drive
int motL2 = 18;
int motR1 = 19;
int motR2 = 21;

void setup() {
  // put your setup code here, to run once:
  ledcSetup(0, 20000, 8);  //pwm setup
  ledcSetup(1, 20000, 8);
  ledcSetup(2, 20000, 8);
  ledcSetup(3, 20000, 8);
  ledcAttachPin(motL1, 0);  //pinmode of motors
  ledcAttachPin(motL2, 1);
  ledcAttachPin(motR1, 2);
  ledcAttachPin(motR2, 3);
}

void loop() {
  // put your main code here, to run repeatedly:
   //motor drive
  ledcWrite(0, 150);
  ledcWrite(1, 0);
  ledcWrite(2, 150);
  ledcWrite(3, 0);

}
```