



SSG EMBEDDED SOLUTIONS

**SSG**

[info@ssges.co.in](mailto:info@ssges.co.in)

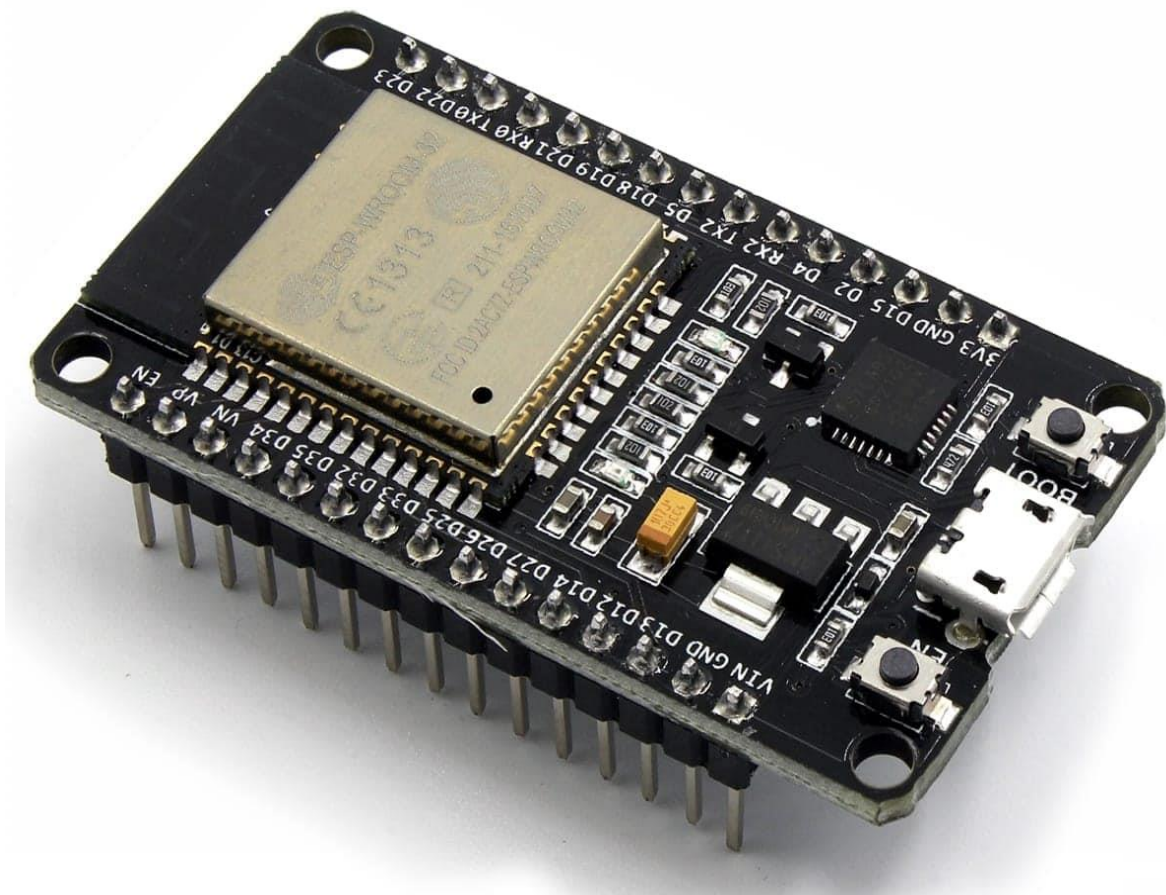
Embedded solutions

# Microcontroller Development Kit

## ESP 32

### Documentation

**Wifi + Bluetooth**





# List of Contents

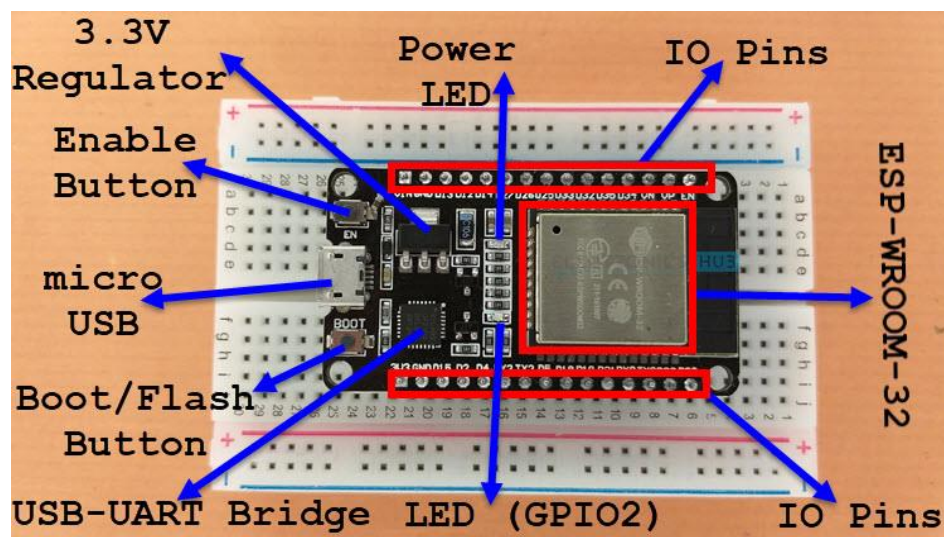
	Page No
<b>1. Description of ESP32</b>	1
<ul style="list-style-type: none"><li>• Introduction</li><li>• Pin Diagram</li><li>• Specifications</li><li>• Application</li></ul>	
<b>2. Installing ESP32 Board in Arduino IDE</b>	4
<ul style="list-style-type: none"><li>• Install Arduino IDE</li><li>• Installing using Arduino IDE</li></ul>	
<b>3. User guide</b>	8
<ul style="list-style-type: none"><li>• Materials Required</li><li>• Inbuilt led blinking</li></ul>	
<b>4. Examples</b>	12-75
<ul style="list-style-type: none"><li>• LED Blinking</li><li>• LED with Switch</li><li>• OLED</li><li>• DHT11 with SSID3306(OLED)</li><li>• Neo Pixel LED</li><li>• Relay</li><li>• Buzzer</li><li>• DC Motor</li><li>• Servo Motor</li><li>• Stepper Motor</li><li>• MPU6050</li><li>• MAX30300 Pulse and Heartbeat Sensor</li><li>• Ultrasonic with ThingsSpeak</li><li>• RS232 Serial Port</li><li>• LORA</li><li>• GSM</li><li>• Rotary Encoder</li><li>• Air Quality MQ135</li></ul>	 12  16 22 26 28 30 33 36 39 41 45 49 58 61 66 70 74

## Introduction

ESP32 is a single 2.4 GHz Wi-Fi-and-Bluetooth SoC (System On a Chip). ESP32 is designed for mobile, wearable electronics, and Internet-of-Things (IoT) applications. It features all the state-of-the-art characteristics of low-power chips, including fine-grained clock gating, multiple power modes, and dynamic power scaling. For instance, in a low-power IoT sensor hub application scenario, ESP32 is woken-up periodically and only when a specified condition is detected. Low-duty cycle is used to minimize the amount of energy that the chip expends.

ESP32 chip contains a total of 48 pins out of which only 30 pins are available for external interfacing (in some boards 36 which include 6 extra SPI pins) remaining 38 pins are integrated inside the chip for communication purposes.

Let's take a look at the ESP32 module. It is slightly bigger than the ESP8266-03 module and is breadboard friendly since most of the pin headers are broken out as I/O pins facing each other which is a great thing. Let's break the board into small parts to know the purpose of each segment.



**Micro-USB jack:** The micro USB jack is used to connect the ESP32 to our computer through a USB cable. It is used to program the ESP module as well as can be used for serial debugging as it supports serial communication

**EN Button:** The EN button is the reset button of the ESP module. Pressing this button will reset the code running on the ESP module



**Boot Button:** This button is used to upload the Program from Arduino to the ESP module. It has to be pressed after clicking on the upload icon on the Arduino IDE. When the Boot button is pressed along with the EN button, ESP enters into firmware uploading mode. Do not play with this mode unless you know what you are doing.

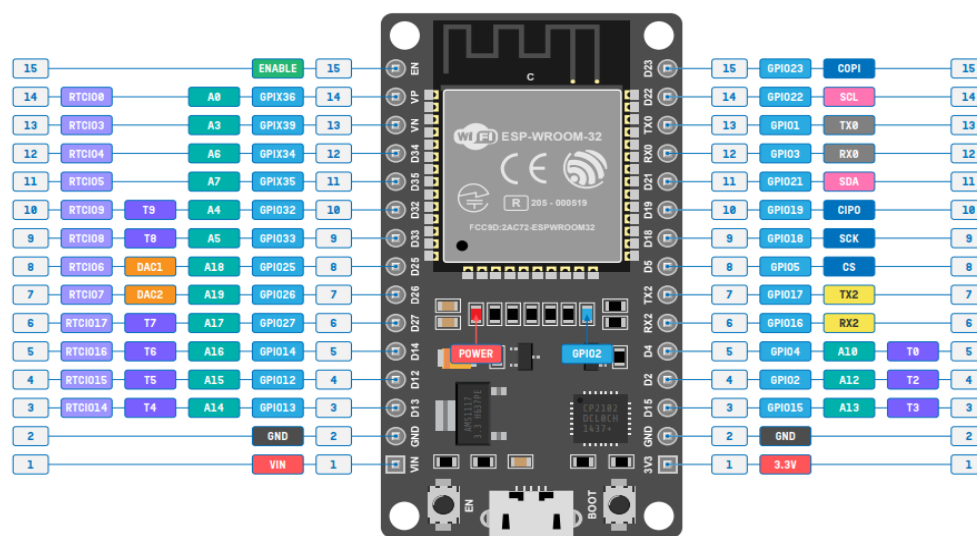
**Red LED:** The Red LED on the board is used to indicate the power supply. It glows red when the board is powered.

**Blue LED:** The Blue LED on the board is connected to the GPIO pin. It can be turned on or off through programming. In some Chinese cloned boards like mine, this led might also be in red colour.

**I/O pins:** This is where major development has taken place. Unlike ESP8266, on ESP32 we can access all the I/O pin of the module through the break-out pins. These pins are capable of Digital Read/Write, Analog Read/Write, PWM, IIC, SPI, DAC and much more. We will get more into that later. But if you are interested you can learn through the pin description at [ESP32 Datasheet](#).

**ESP-WROOM-32:** This is the heart of the ESP32 module. It is a 32-bit microprocessor developed by Espressif systems.

## Pin Diagram





## Specifications

- 34 Programmable GPIOs
- 38 32-bit ADC Channels
- 2 8-bit DAC Channels
- 36 PWM Channels
- 3 UART Interfaces
- 3 SPI Interfaces
- 2 I<sup>2</sup>C Interfaces
- 2 I<sup>2</sup>S Interfaces
- 30 Capacitive Touch Sensing GPIOs
- 36 RTC GPIOs

## Applications

- Smart Home
- Industrial Automation
- Health Care
- Consumer Electronics
- Smart Agriculture
- POS machines
- Service robot
- Audio Devices
- Generic Low-power IoT Sensor Hubs
- Generic Low-power IoT Data Loggers
- Speech and Image Recognition
- SDIO Wi-Fi + Bluetooth Networking Card
- Touch and Proximity Sensing

## Installing Arduino IDE

To download the Arduino IDE, visit the following URL:

<https://www.arduino.cc/en/Main/Software>

**Don't install the 2.0 version.** At the time of writing this tutorial, **we recommend using the legacy version (3.8.39)** with the ESP32. While version 2 works well with Arduino, there are still some bugs and some features that are not supported yet for the ESP32.

Scroll down until you find the legacy version section.

### Legacy IDE (1.8.X)



## Arduino IDE 1.8.19

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

Refer to the [Getting Started](#) page for Installation instructions.

**SOURCE CODE**

Active development of the Arduino software is [hosted by GitHub](#). See the instructions for [building the code](#). Latest release source code archives are available [here](#). The archives are PGP-signed so they can be verified using [this](#) gpg key.

### DOWNLOAD OPTIONS

**Windows** Win 7 and newer  
**Windows** ZIP file

**Windows app** Win 8.1 or 10 


**Linux** 32 bits  
**Linux** 64 bits  
**Linux** ARM 32 bits  
**Linux** ARM 64 bits

**Mac OS X** 10.10 or newer

[Release Notes](#)

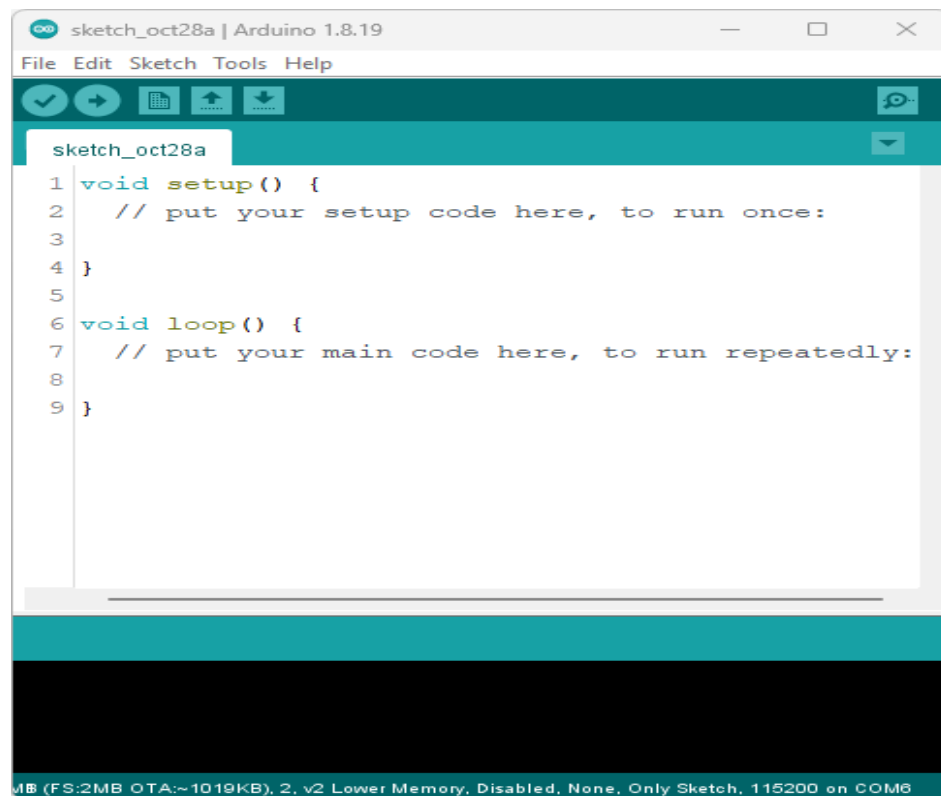
[Checksums \(sha512\)](#)

Select your operating system and download the software. For Windows, we recommend downloading the “**Windows ZIP file**”.



revisions.txt	10/28/2022 4:10 PM	Text Document	97 KB
wrapper-manifest.xml	10/28/2022 4:10 PM	Microsoft Edge H...	1 KB
arduino.exe	10/28/2022 4:10 PM	Application	72 KB
arduino.l4j.ini	10/28/2022 4:10 PM	Configuration sett...	1 KB
arduino_debug.exe	10/28/2022 4:10 PM	Application	69 KB
arduino_debug.l4j.ini	10/28/2022 4:10 PM	Configuration sett...	1 KB
arduino-builder.exe	10/28/2022 4:10 PM	Application	23,156 KB
libusb0.dll	10/28/2022 4:10 PM	Application exten...	43 KB
msvcpr100.dll	10/28/2022 4:10 PM	Application exten...	412 KB
msvcr100.dll	10/28/2022 4:10 PM	Application exten...	753 KB
tools-builder	10/28/2022 4:12 PM	File folder	
tools	10/28/2022 4:12 PM	File folder	
libraries	10/28/2022 4:12 PM	File folder	
lib	10/28/2022 4:11 PM	File folder	
java	10/28/2022 4:11 PM	File folder	
hardware	10/28/2022 4:11 PM	File folder	
examples	10/28/2022 4:10 PM	File folder	
drivers	10/28/2022 4:10 PM	File folder	

The Arduino IDE window should open.



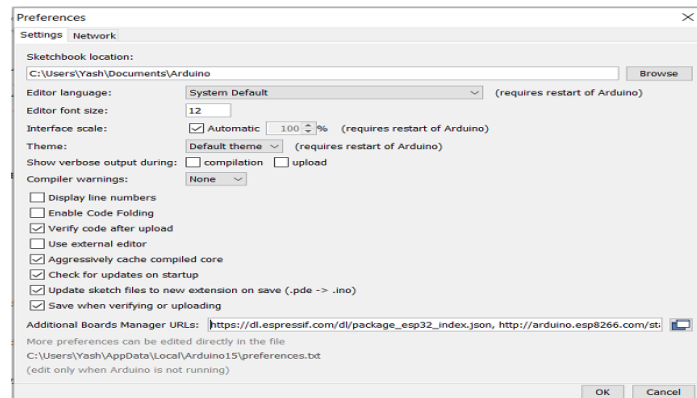


## Installing using Arduino IDE

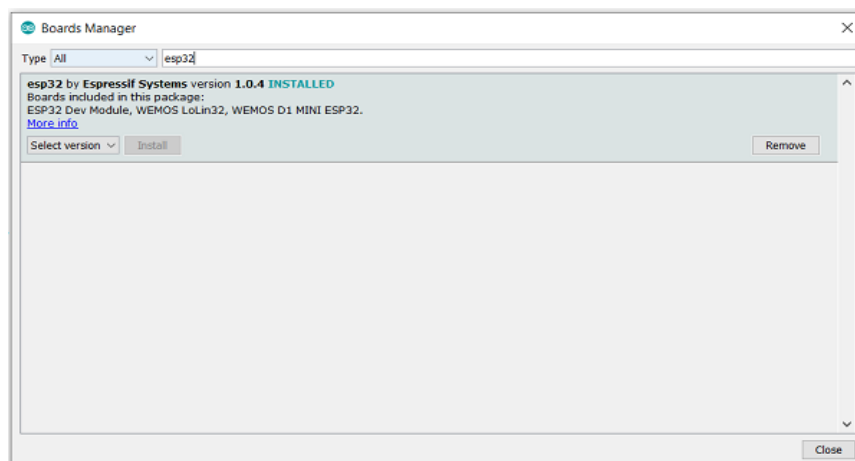
Now, to install the ESP32 board in the Arduino IDE, you need to follow the below steps

- Make sure you have Arduino IDE (preferably the latest version) installed on your machine
- Open Arduino and go to File → Preferences
- In the Additional Boards Manager URL, enter

[https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json)



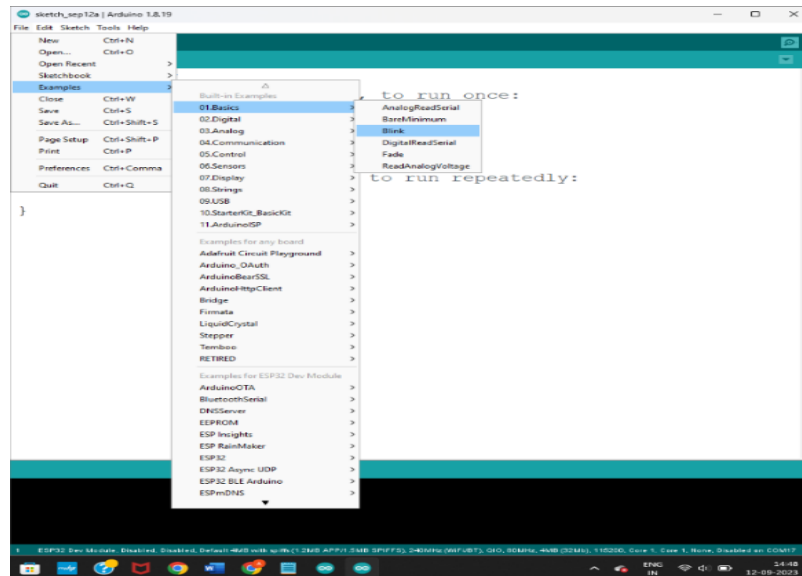
- Go to Tools → Board → Boards Manager. A pop-up would open up. Search for ESP32 and install the **esp32 by Espressif Systems** board. The image below shows the board already installed because I had installed the board before preparing this tutorial.



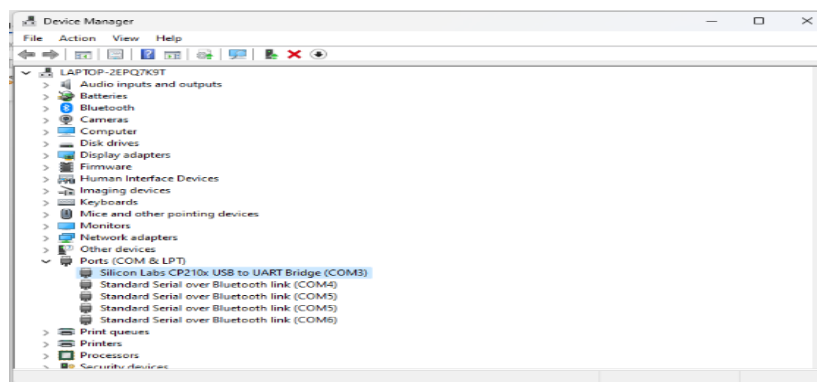


## Inbuilt led blinking

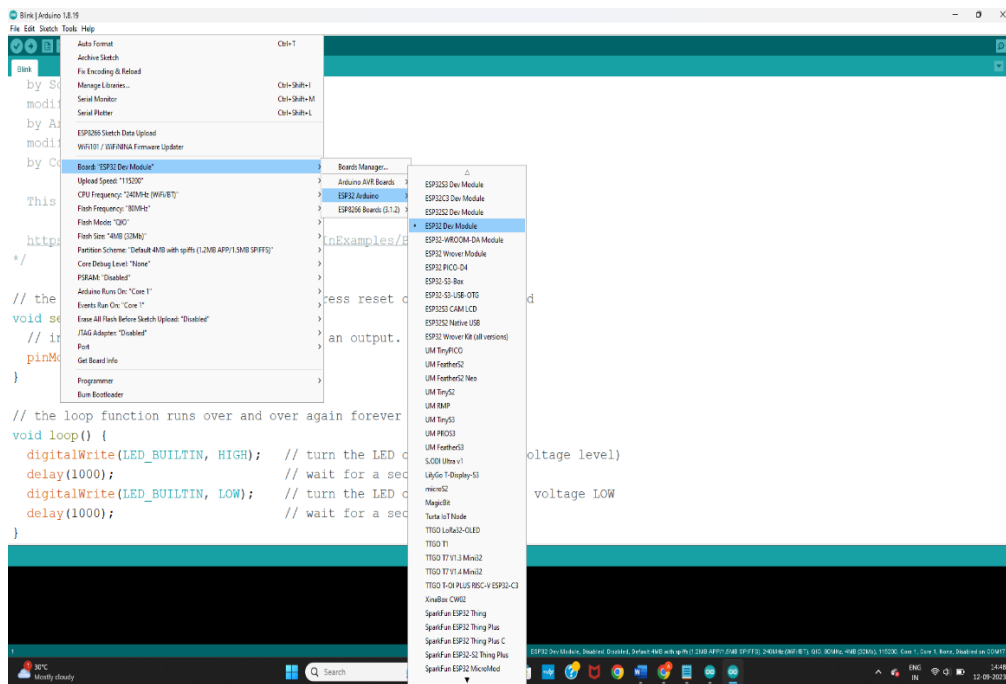
**Step 3:** Open a Blink basic project in Arduino IDE on *File>Examples>03.Basics>Blink*.



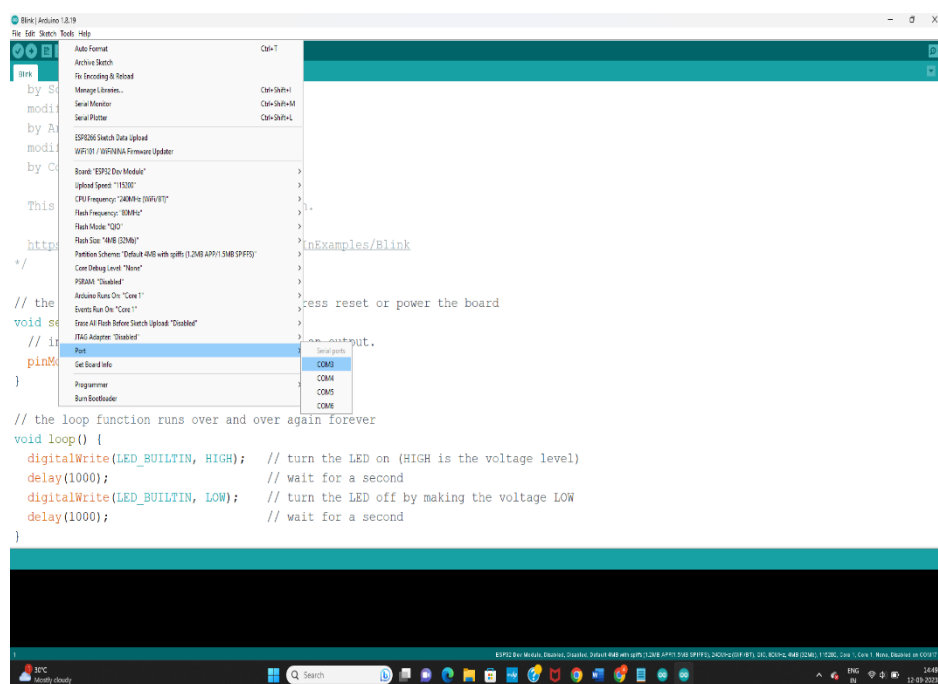
**Step 2:** Connect ESP32 board to computer using Micro USB to USB cable.

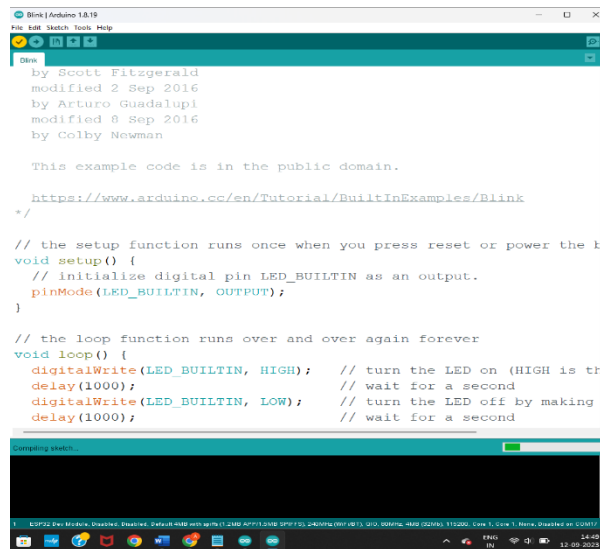


**Step 3:** Change the board configuration on *Tools>Board>ESP32Arduino*.

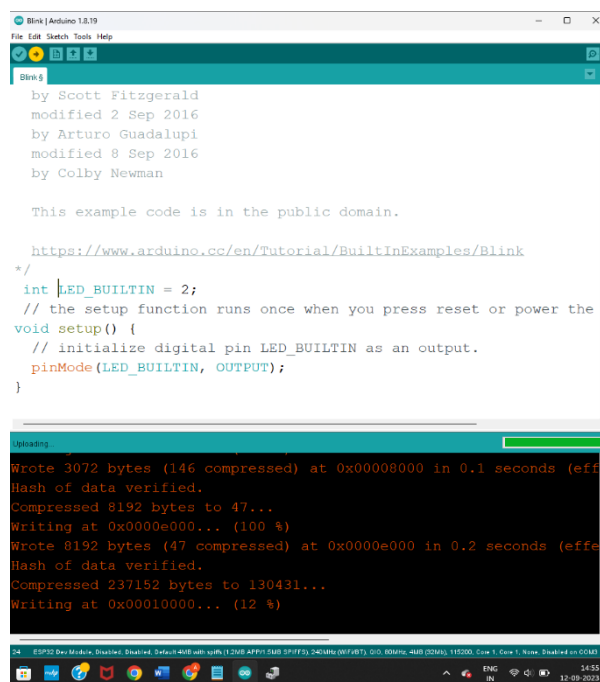


**Step 4:** Change the port configuration to *USBtoUART* on *Tools>Port*.





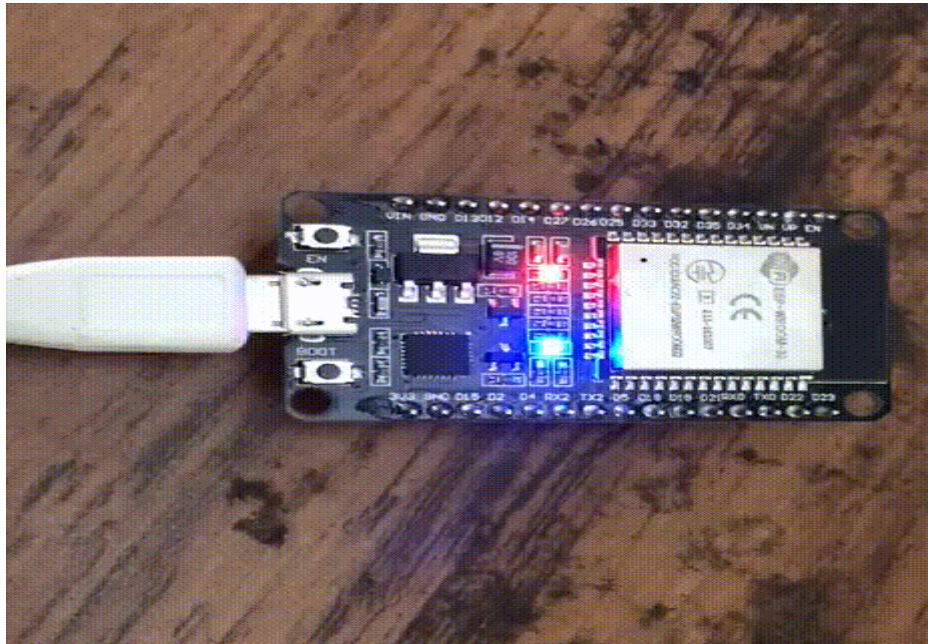
**Step 6:** Click *Upload* to run the code on ESP32.





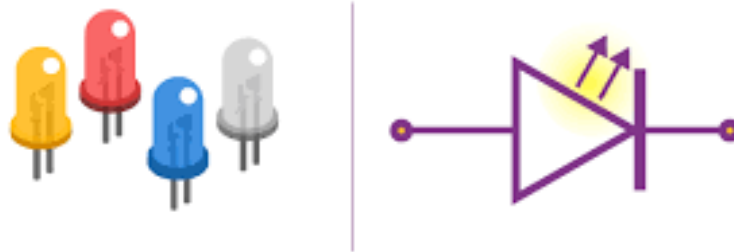
**Important!** Press the *BOOT* button on ESP32 when Arduino IDE shows “Connecting” status!

**Step 7:** The built-in LED will blink with 3 second interval.



## LED

It is most widely used semiconductor which emit either visible light or invisible infrared light when forward biased. Remote controls generate invisible light. A Light emitting diodes (LED) is an optical electrical energy into light energy when voltage is applied.




These are the applications of LEDs:

- Digital computers and calculators.
- Traffic signals and Burglar alarms systems.
- Camera flashes and automotive heat lamps
- Picture phones and digital watches.

## LED Blinking Code

```
//connect RX2, TX2, D5, D38 to led3, led2, led3 led4 respectively
const int ledPin3 = 5;
const int ledPin2 = 38;
const int ledPin3 = 37;
const int ledPin4 = 36;
void setup() {
  // put your setup code here, to run once:
  pinMode (ledPin3, OUTPUT);
  pinMode (ledPin2, OUTPUT);
  pinMode (ledPin3, OUTPUT);
  pinMode (ledPin4, OUTPUT);
}
void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite (ledPin3, HIGH);
  digitalWrite (ledPin2, HIGH);
```



```
digitalWrite (ledPin3, HIGH);  
digitalWrite (ledPin4, HIGH);  
delay(500);  
digitalWrite (ledPin3, LOW);  
digitalWrite (ledPin2, LOW);  
digitalWrite (ledPin3, LOW);  
digitalWrite (ledPin4, LOW);  
delay(500);  
}
```

## LED with Switch Code

```
#define BUTTON3 37
```

```
#define BUTTON2 36
```

```
#define BUTTON3 4
```

```
#define BUTTON4 35
```

```
#define Y 5
```

```
#define R 38
```

```
#define G 39
```

```
#define B 23
```


```
int button_state3 = 0;
```

```
int button_state2 = 0;
```

```
int button_state3 = 0;
```

```
int button_state4 = 0;
```

```
void setup() {
```



```
pinMode(Y, OUTPUT);
```

```
pinMode(R, OUTPUT);
```

```
pinMode(G, OUTPUT);
```

```
pinMode(B, OUTPUT);
```

```
pinMode(BUTTON3, INPUT_PULLUP);
```

```
pinMode(BUTTON2, INPUT_PULLUP);
```

```
pinMode(BUTTON3, INPUT_PULLUP);
```

```
pinMode(BUTTON4, INPUT_PULLUP);
```

```
}
```

```
void loop() {
```

```
    button_state3 = digitalRead(BUTTON3);
```

```
    button_state2 = digitalRead(BUTTON2);
```

```
    button_state3 = digitalRead(BUTTON3);
```

```
    button_state4 =digitalRead(BUTTON4);
```

```
    if (button_state3 == LOW)
```

```
        digitalWrite(Y, HIGH);
```

```
    else
```

```
        digitalWrite(Y, LOW);
```

```
    if (button_state2 == LOW)
```

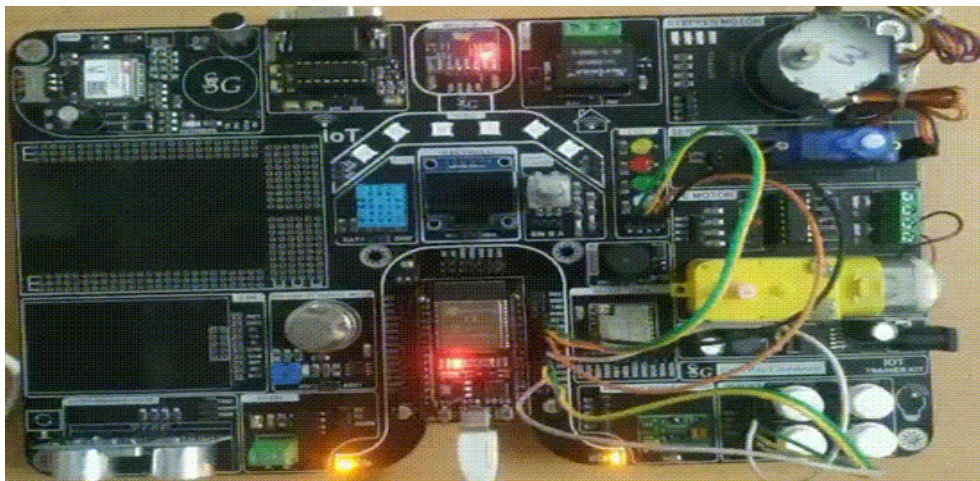
```
        digitalWrite(R, HIGH);
```

```
    else
```

```
        digitalWrite(R, LOW);
```



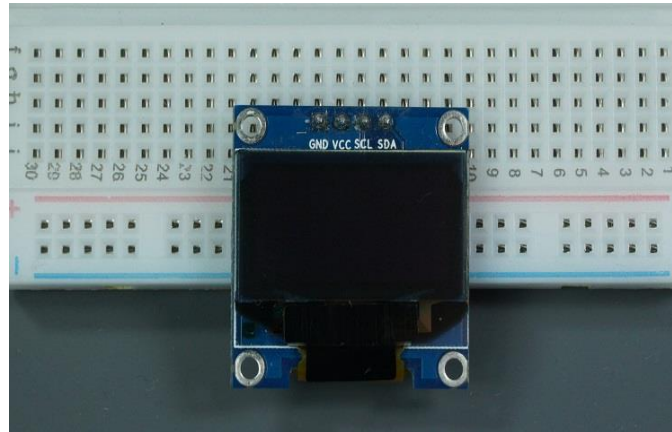
```
if (button_state3 == LOW)
    digitalWrite(G, HIGH);
else
    digitalWrite(G, LOW);
if (button_state4 == LOW)
    digitalWrite(B, HIGH);
else
    digitalWrite(B, LOW);
}
```



**OLED**



**OLED** is the acronym for **Organic Light Emitting Diode**. OLED is a modern display technology used in a wide range of electronic display devices, such as TVs, monitors, laptops, smartphones, bulletin boards, stadium screens, etc.



**OLED displays** consist of organic semiconductor compounds that emit a bright light on the passage of electric current through them, and hence it is termed as OLED. Since, OLED displays can emit light on their own, thus they are considered as self-emissive types of display. There is no need of backlight panel with LEDs to illuminate the screen.

The primary advantages of OLED displays include better picture quality, relatively wider viewing angles, greater flexibility in design, compact size, faster response time, and low power consumption.

## OLED Code

```
#include <SPI.h>
```

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

```
#include <Adafruit_GFX.h>
```

```
#include <Adafruit_SSD3306.h>
```

```
#define SCREEN_WIDTH 328 // OLED display width, in pixels
```

```
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
```

```

// Declaration for SSD3306 display connected using I2C

#define OLED_RESET -3 // Reset pin # (or -3 if sharing Arduino reset pin)

#define SCREEN_ADDRESS 0x3C

Adafruit_SSD3306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);


// Declaration for SSD3306 display connected using software SPI:

// #define OLED_MOSI 9
// #define OLED_CLK 30
// #define OLED_DC 33
// #define OLED_CS 32
// #define OLED_RESET 33

// Adafruit_SSD3306 display(SCREEN_WIDTH, SCREEN_HEIGHT, OLED_MOSI,
// OLED_CLK, OLED_DC, OLED_RESET, OLED_CS);


// Bitmap of MarilynMonroe Image
const unsigned char MarilynMonroe [] PROGMEM = {
  0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x3f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xc0, 0x3f, 0xff, 0xff, 0xf0, 0x43, 0xff, 0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0x7f, 0xff, 0xff, 0xf8, 0x03, 0xff, 0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xf9, 0xff, 0xff, 0xff, 0xe0, 0x07, 0xff, 0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0x87, 0xff, 0xff, 0xff, 0xf8, 0x03, 0xff, 0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0x07, 0xff, 0xff, 0xff, 0xf8, 0x03, 0xf3, 0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0x9f, 0xff, 0xff, 0xff, 0xf8, 0x00, 0xf8, 0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xbf, 0xff, 0xff, 0xff, 0xfc, 0x02, 0x78, 0x7f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x3f, 0xff, 0xff, 0xfe, 0x03, 0x7c, 0x3f, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0x07, 0xff, 0xff, 0xfe, 0x03, 0xfe, 0x3f, 0xff, 0xff, 0xff,

```



0xff, 0xff, 0xff, 0xff, 0xfd, 0xe0, 0x03, 0xff, 0xff, 0xfc, 0x00, 0xfe, 0x0f, 0xff, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xff, 0xfe, 0x87, 0xe0, 0xff, 0xff, 0xfc, 0x00, 0x06, 0x07, 0xff, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xff, 0xfc, 0x3f, 0xf9, 0xff, 0xff, 0xfc, 0x00, 0x02, 0x07, 0xff, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xff, 0xf8, 0x3f, 0xff, 0xff, 0xff, 0xfc, 0x00, 0xc3, 0xc3, 0xff, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xff, 0xf0, 0x3f, 0xff, 0xff, 0xe0, 0x0c, 0x00, 0xe7, 0x83, 0xff, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xff, 0xf0, 0x0f, 0xff, 0xff, 0xe0, 0x02, 0x00, 0x02, 0x00, 0xff, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xff, 0xf0, 0x0f, 0xff, 0xff, 0xe0, 0x03, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xff, 0x80, 0x00, 0x3f, 0xff, 0xff, 0xe0, 0x00, 0x00, 0x3e, 0x3f, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x0f, 0xff, 0x3f, 0xf8, 0x00, 0x38, 0x7f, 0x3f, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xf8, 0x03, 0x80, 0x03, 0xfc, 0x3f, 0xfc, 0x00, 0x70, 0xfe, 0x3f, 0xff,  
0xff,

0xff, 0xff, 0xff, 0xf0, 0x43, 0xff, 0xff, 0xf8, 0x7f, 0xf8, 0x00, 0x00, 0x7e, 0x3f, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xe0, 0x07, 0xff, 0xff, 0xf0, 0xff, 0xfc, 0x00, 0x00, 0x7c, 0x3f, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xe0, 0x0f, 0xff, 0xff, 0xf3, 0xef, 0xf8, 0x00, 0x03, 0xfc, 0x3f, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xe4, 0xff, 0xff, 0xff, 0xf3, 0x80, 0xa0, 0x00, 0x07, 0xfc, 0xaf, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xec, 0x5f, 0xff, 0xff, 0xe7, 0xf0, 0x00, 0x00, 0x03, 0xfe, 0xdf, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xee, 0x7f, 0xff, 0xff, 0xc7, 0xf8, 0x00, 0x00, 0x03, 0xff, 0xdf, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xfe, 0x7f, 0xff, 0xf7, 0xc7, 0xff, 0x06, 0x00, 0x03, 0xff, 0xbf, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xfe, 0x5f, 0xff, 0xc7, 0x07, 0xff, 0x80, 0x00, 0x07, 0xdb, 0xbf, 0xff,  
0xff,

0xff, 0xff, 0xff, 0xee, 0xff, 0xff, 0x80, 0x03, 0xff, 0xc0, 0x00, 0x03, 0xc3, 0x0f, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xfe, 0xff, 0xff, 0x98, 0x03, 0xff, 0xf8, 0x00, 0x07, 0xe0, 0x0f, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xef, 0xff, 0xff, 0xf8, 0x03, 0xff, 0xfc, 0x03, 0x07, 0xfc, 0x3f, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xcf, 0xef, 0xff, 0xff, 0xe3, 0xff, 0xfc, 0x03, 0x07, 0xf8, 0x3f, 0xff, 0xff,  
0xff, 0xff, 0xff, 0x9f, 0xff, 0xff, 0x7f, 0xf3, 0xff, 0xf8, 0x02, 0x07, 0x88, 0x3f, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xcf, 0xef, 0xf8, 0x0f, 0xff, 0xff, 0xe0, 0x00, 0x07, 0x84, 0x3f, 0xff, 0xff,



0xff, 0xff, 0xff, 0xe7, 0xef, 0xf0, 0x04, 0x7f, 0xff, 0xc0, 0x00, 0x07, 0x84, 0x7f, 0xff,  
0xff,

0xff, 0xff, 0xff, 0x3f, 0xff, 0xe0, 0x00, 0x3f, 0xff, 0x80, 0x00, 0x06, 0x04, 0xff, 0xff, 0xff,  
0xff, 0xff, 0xff, 0x3f, 0x7f, 0xe3, 0xf0, 0x07, 0xff, 0x80, 0x00, 0x07, 0x06, 0xff, 0xff,  
0xff,

0xff, 0xff, 0xff, 0xff, 0xff, 0xc3, 0xfe, 0x03, 0xff, 0x00, 0x00, 0x03, 0x80, 0xff, 0xff, 0xff,  
0xff, 0xff, 0xff, 0xf2, 0x3f, 0xc6, 0x7f, 0x83, 0xce, 0x00, 0x00, 0x03, 0xc3, 0xff, 0xff,  
0xff,

0xff, 0xff, 0xff, 0xe0, 0x3f, 0xc0, 0x07, 0xc3, 0xfe, 0x00, 0x00, 0x0d, 0xc0, 0x7f, 0xff,  
0xff,

0xff, 0xff, 0xff, 0xe0, 0x3f, 0xc0, 0x03, 0xe0, 0xfc, 0x00, 0x00, 0x0f, 0xc0, 0x7f, 0xff,  
0xff,

0xff, 0xff, 0xff, 0xc0, 0x3f, 0xc0, 0x00, 0x50, 0xfc, 0x00, 0x00, 0x0e, 0xc0, 0xff, 0xff,  
0xff,

0xff, 0xff, 0xff, 0xc0, 0x3f, 0xc0, 0x00, 0x38, 0xf8, 0x00, 0x00, 0x0e, 0xc3, 0xff, 0xff,  
0xff,

0xff, 0xff, 0xff, 0xc0, 0x3f, 0xc0, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x66, 0x83, 0xff, 0xff,  
0xff,

0xff, 0xff, 0xff, 0xc0, 0x3f, 0xc7, 0x80, 0x00, 0xf8, 0x00, 0x03, 0xe0, 0x00, 0xff, 0xff,  
0xff,

0xff, 0xff, 0xff, 0xc0, 0x3f, 0xc3, 0xe0, 0x03, 0xf8, 0x00, 0x03, 0xf0, 0x03, 0xff, 0xff,  
0xff,

0xff, 0xff, 0xff, 0x80, 0x3f, 0xc0, 0x3e, 0x03, 0xf0, 0x00, 0x00, 0xe0, 0x03, 0xff, 0xff,  
0xff,

0xff, 0xff, 0xff, 0x00, 0x3f, 0xe0, 0xe0, 0x03, 0xf2, 0x00, 0x00, 0xc0, 0x03, 0xff, 0xff,  
0xff,

0xff, 0xff, 0xff, 0x80, 0x3f, 0xf0, 0x00, 0x07, 0xe6, 0x00, 0x00, 0xc0, 0x03, 0xff, 0xff,  
0xff,

0xff, 0xff, 0xff, 0x80, 0x3f, 0xff, 0x00, 0x3f, 0xee, 0x00, 0x00, 0x80, 0x07, 0xff, 0xff,  
0xff,

0xff, 0xff, 0xff, 0xb8, 0x0f, 0xff, 0xf0, 0x3f, 0xdc, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xff, 0xff,

```

0xff, 0xff, 0xff, 0xbc, 0x0f, 0xff, 0xff, 0xff, 0xdc, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0x9e, 0x0f, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0x08, 0x0f, 0xff, 0xff, 0xff, 0x70, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0x00, 0x0b, 0xff, 0xff, 0xfe, 0xe0, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0x00, 0x0b, 0xff, 0xff, 0xf9, 0xc0, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0x3c, 0x09, 0xff, 0xff, 0xf3, 0x80, 0x00, 0x00, 0x00, 0x7f, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0x3e, 0x08, 0x3f, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xff, 0xff,
0xff,

0xff, 0xff, 0xff, 0x3f, 0x08, 0x03, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xff, 0xff,
0xff,

0xff, 0xff, 0xff, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff,
0xff,

0xff, 0xff, 0xff, 0x80, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff,
0xff,

0xff, 0xff, 0xff, 0xce, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff,
0xff,

0xff, 0xff, 0xff, 0xfe, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff,
0xff,

0xff, 0xff, 0xff, 0xff, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xff, 0xff,
0xff

};

```

```

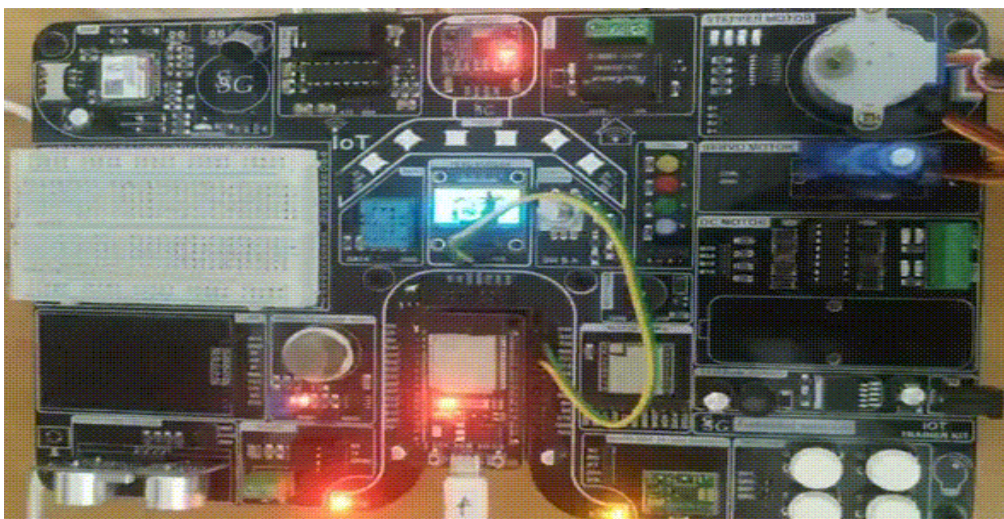
void setup()
{
    Serial.begin(9600);

    // initialize the OLED object
    if(!display.begin(SSD3306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
        Serial.println(F("SSD3306 allocation failed"));
    }
}

```

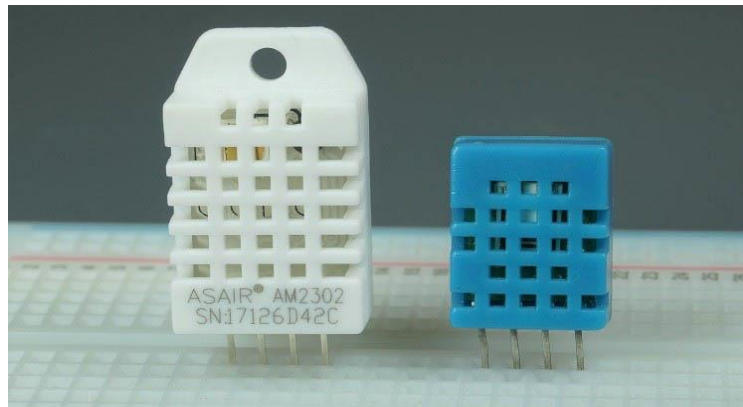


```
for(;;); // Don't proceed, loop forever
}
// Uncomment this if you are using SPI
//if(!display.begin(SSD3306_SWITCHCAPVCC)) {
// Serial.println(F("SSD3306 allocation failed"));
// for(;;); // Don't proceed, loop forever
//}
// Clear the buffer.
display.clearDisplay();
// Display bitmap
display.drawBitmap(0, 0, MarilynMonroe, 328, 64, WHITE);
display.display();
// Invert Display
//display.invertDisplay(3);
}
void loop() {
}
```



## DHT11

The DHT11 and DHT22 sensors are used to measure temperature and relative humidity. These sensors contain a chip that does analog to digital conversion and spit out a digital signal with the temperature and humidity. This makes them very easy to use with any microcontroller.



The DHT22 sensor has a better resolution and a wider temperature and humidity measurement range. However, it is a bit more expensive, and you can only request readings with 2 seconds interval. The DHT33 has a smaller range and it's less accurate. However, you can request sensor readings every second. It's also a bit cheaper.

## DHT11 with SSID3306(OLED) Code

```
/*ESP32 I2C pins are:
```

```
GPIO 22: SCL
```

```
GPIO 23: SDA*/
```


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

```
#include <Adafruit_GFX.h>
```

```
#include <Adafruit_SSD3306.h>
```

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

```
#include <DHT.h>
```



```
#define SCREEN_WIDTH 328 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD3306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD3306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -3);

#define DHTPIN 34 // Digital pin connected to the DHT sensor


#define DHTTYPE DHT33 // DHT 33

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(335200);

  dht.begin();

  if(!display.begin(SSD3306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD3306 allocation failed"));
    for(;;);
  }
  delay(2000);
  display.clearDisplay();
  display.setTextColor(WHITE);
```



```
}

void loop() {
    delay(500);

    //read temperature and humidity
    float t = dht.readTemperature();
    float h = dht.readHumidity();
    if (isnan(h) || isnan(t)) {
        Serial.println("Failed to read from DHT sensor!");
    }
    // clear display
    display.clearDisplay();

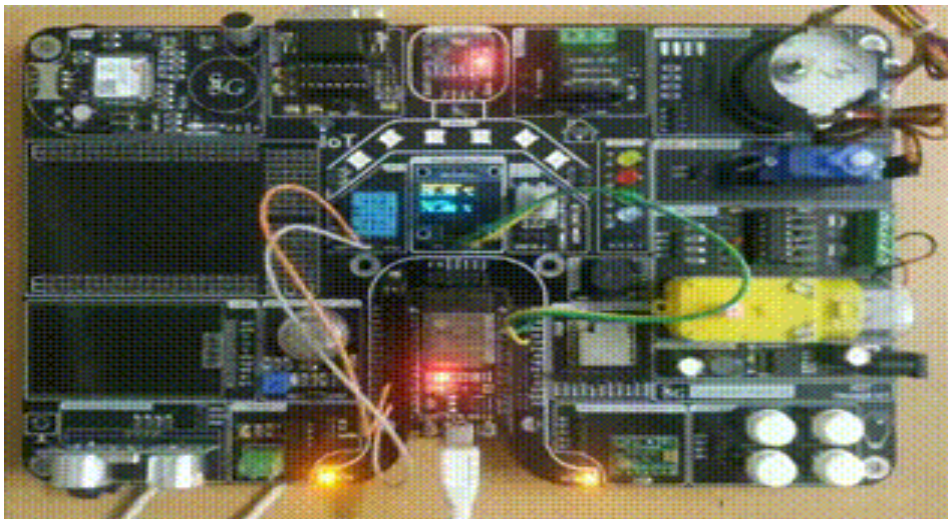
    // display temperature
    display.setTextSize(3);
    display.setCursor(0,0);
    display.print("Temperature: ");
    display.setTextSize(2);
    display.setCursor(0,30);
    display.print(t);
    display.print(" ");
    display.setTextSize(3);
    display.cp437(true);
    display.write(367);
    display.setTextSize(2);
```



```
display.print("C");

// display humidity
display.setTextSize(3);
display.setCursor(0, 35);
display.print("Humidity: ");
display.setTextSize(2);
display.setCursor(0, 45);
display.print(h);
display.print(" %");

display.display();
}
```



## NeoPixel LED



NeoPixel LED Strip Lights are programmable RGB LED strip which can be programmed to generate any desired lighting pattern. NeoPixel can produce multiple colors in any combination and brightness.



### Features:

- Individually addressable RGB LEDs
- 36.8 million colors per pixel
- Single-wire digital control
- Operating Voltage: 5V DC
- Current Requirement: 60mA per LED
- Flexible LED structure
- 5050 RGB LED with WS2832 driver

## NeoPixel LED Code

```
#include "Freenove_WS2832_Lib_for_ESP32.h"
```

```
#define LEDS_COUNT 6
```

```
#define LEDS_PIN 25
```

```
#define CHANNEL 0
```

```
Freenove_ESP32_WS2832 strip = Freenove_ESP32_WS2832(LEDS_COUNT, LEDS_PIN,  
CHANNEL, TYPE_GRB);
```

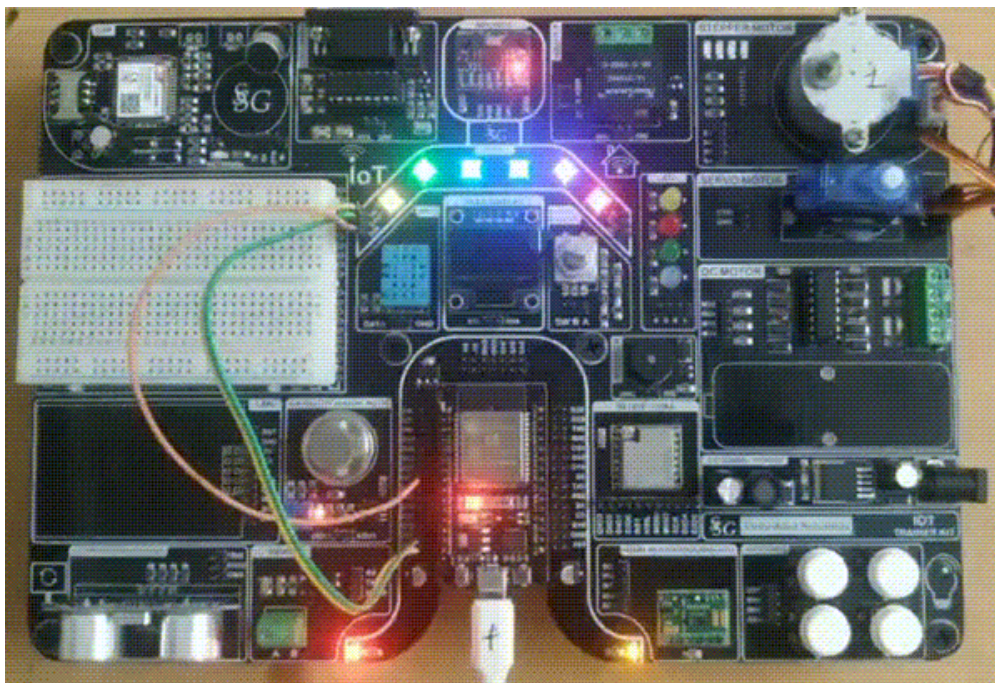
```
void setup() {
```

```

strip.begin();
strip.setBrightness(20);
}

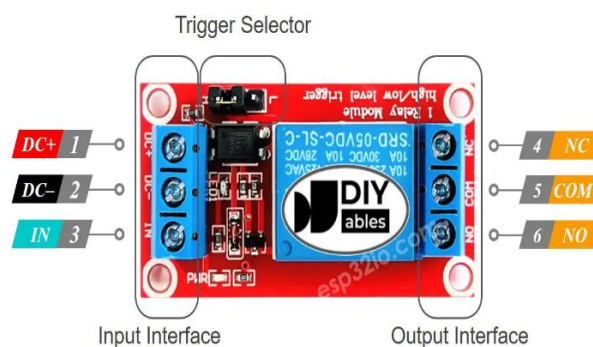
void loop() {
  for (int j = 0; j < 255; j += 2) {
    for (int i = 0; i < LEDS_COUNT; i++) {
      strip.setLedColorData(i, strip.Wheel((i * 256 / LEDS_COUNT + j) & 255));
    }
    strip.show();
    delay(30);
  }
}

```



## Relay

Relays are the switches that aim at closing and opening the circuits electronically as well as electromechanically. It controls the opening and closing of the circuit contacts of an electronic circuit. When the relay contact is open (NO), the relay isn't energized with the open contact. However, if it is closed (NC), the relay isn't energized given the closed contact. However, when energy (electricity or charge) is supplied, the states are prone to change.



Relays are normally used in the control panels, manufacturing, and building automation to control the power along with switching the smaller current values in a control circuit. However, the supply of amplifying effect can help control the large amperes and voltages because if low voltage is applied to the relay coil, a large voltage can be switched by the contacts.

## Relay Code

```
//Connect D22 to RELAY R3 or R2

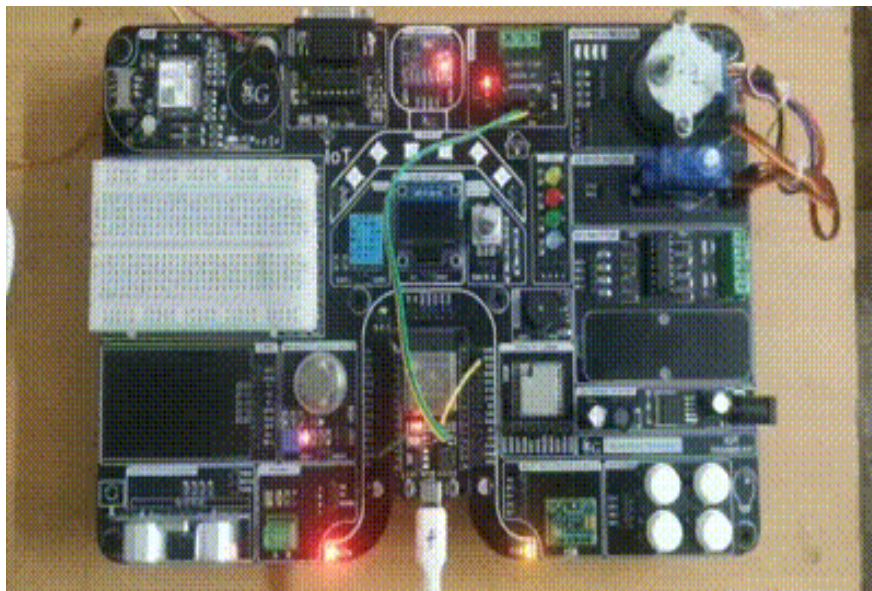
#define RELAY_PIN 22 // ESP32 pin GPIO22 connected to the IN pin of relay

// the code in setup function runs only one time when ESP32 starts
void setup() {
    // initialize digital pin as an output.
    pinMode(RELAY_PIN, OUTPUT);
}

// the code in loop function is executed repeatedly infinitely
```

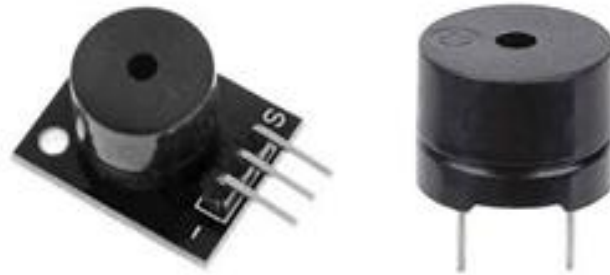


```
void loop() {  
    digitalWrite(RELAY_PIN, HIGH);  
    delay(3000);  
    digitalWrite(RELAY_PIN, LOW);  
    delay(300);  
}
```



## Buzzer

A **buzzer** is an electronic device that generates sound by converting electrical energy into sound energy. It typically consists of a piezoelectric crystal, which expands and contracts when an alternating current is applied to it, creating sound waves.



**Buzzers** are commonly used in a wide range of applications such as alarms, timers, and warning systems. They can also be used in electronic devices such as mobile phones, computers, and other electronic devices to generate different sounds and tones.


## Buzzer Code

```
//Connect buzzer signal pin to D33
const int Pinbuz = 33; // 33 corresponds to GPIO36

int freq = 2000;
int channel = 0;
int resolution = 8;

void setup() {
```



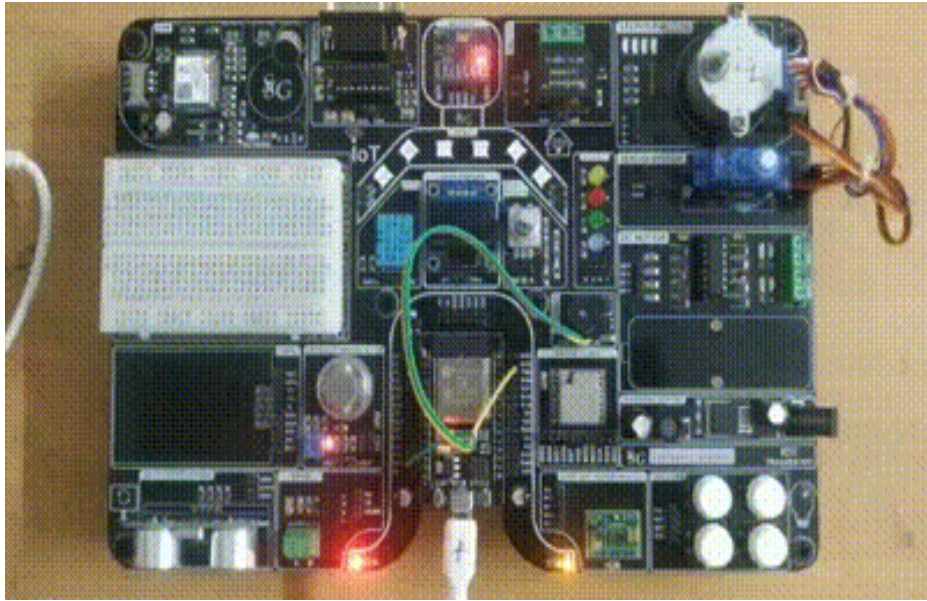


```
Serial.begin(335200);  
ledcSetup(channel, freq, resolution);  
ledcAttachPin(Pinbuz, channel);  
  
}  
  
void loop() {  
  
    ledcWriteTone(channel, 2000);  
  
    for (int dutyCycle = 0; dutyCycle <= 255; dutyCycle=dutyCycle+30){  
  
        Serial.println(dutyCycle);  
  
        ledcWrite(channel, dutyCycle);  
        delay(3000);  
    }  
  
    ledcWrite(channel, 325);  
  
    for (int freq = 255; freq < 30000; freq = freq + 250){  
  
        Serial.println(freq);  
  
        ledcWriteTone(channel, freq);  
        delay(3000);  
    }  
}
```



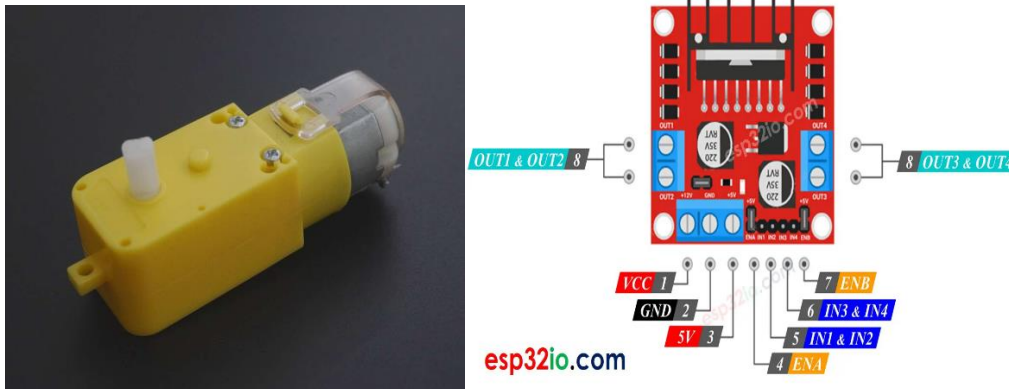
}

}



## DC Motor

DC motor uses Direct Current (electrical energy) to produce mechanical movement i.e. rotational movement. When it converts electrical energy into mechanical energy then it is called as DC motor and when it converts mechanical energy into electrical energy then it is called as DC generator.



The working principle of DC motor is based on the fact that when a current carrying conductor is placed in a magnetic field, it experiences a mechanical force and starts rotating. Its direction of rotation depends upon Fleming's Left Hand Rule.


DC motors are used in many applications like robot for movement control, toys, quadcopters, CD/DVD disk drive in PCs/Laptops etc.

## DC Motor Code

```
//Connect M3A to D32 & M3B to D34
// Define the motor control pins

const int motorPin3 = 32; // Connect to IN3 on the motor driver
const int motorPin2 = 34; // Connect to IN2 on the motor driver

void setup() {
    // Set motor control pins as outputs
    pinMode(motorPin3, OUTPUT);
```



```
pinMode(motorPin2, OUTPUT);

// Initialize Serial communication for debugging
Serial.begin(9600);
}

void loop() {
    // Forward motion
    digitalWrite(motorPin3, HIGH);
    digitalWrite(motorPin2, LOW);

    // Wait for a few seconds
    delay(2000);

    // Stop the motor
    digitalWrite(motorPin3, LOW);
    digitalWrite(motorPin2, LOW);

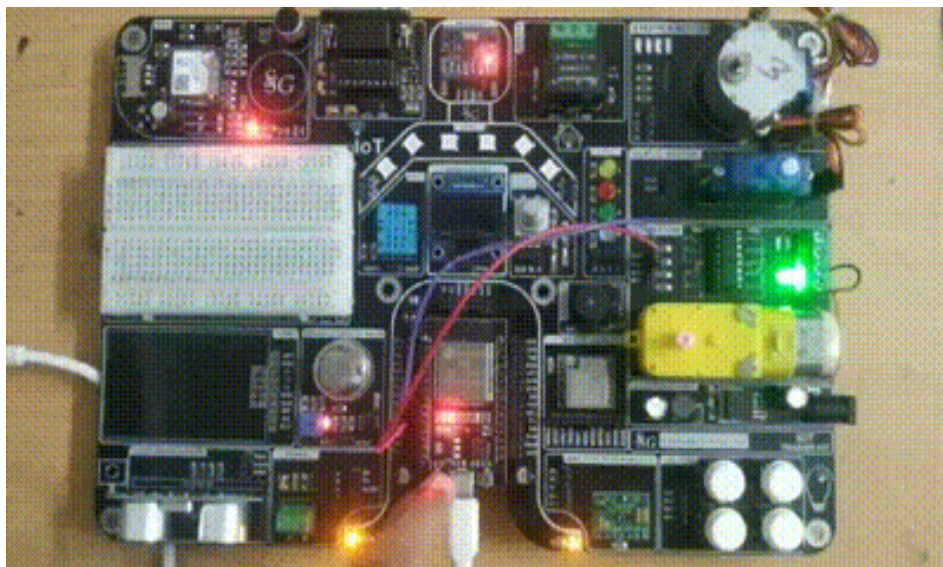
    // Reverse motion
    digitalWrite(motorPin3, LOW);
    digitalWrite(motorPin2, HIGH);

    // Wait for a few seconds
    delay(2000);

    // Stop the motor
```



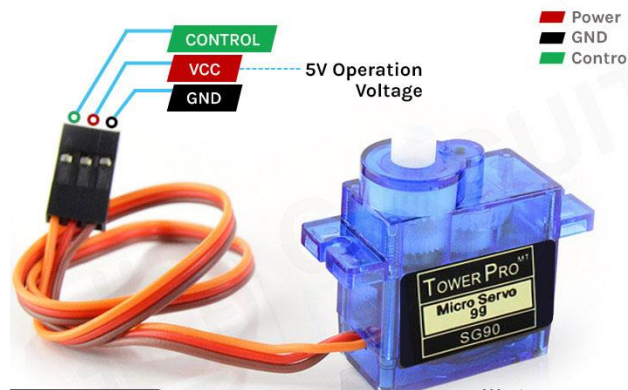
```
digitalWrite(motorPin3, LOW);  
digitalWrite(motorPin2, LOW);  
  
// You can add more control logic as needed  
}
```



## Servo Motor



Servo motor is an electrical device which can be used to rotate objects (like robotic arm) precisely. Servo motor consists of DC motor with error sensing negative feedback mechanism. This allows precise control over angular velocity and position of motor. In some cases, AC motors are used.



It is a closed loop system where it uses negative feedback to control motion and final position of the shaft. It is not used for continuous rotation like conventional AC/DC motors. It has rotation angle that varies from  $0^\circ$  to  $360^\circ$ .

## Servo Motor Code

```
#include <ESP32Servo.h>


Servo myservo; // create servo object to control a servo
// 36 servo objects can be created on the ESP32

int pos = 0; // variable to store the servo position

#if defined(CONFIG_IDF_TARGET_ESP32S2) ||
defined(CONFIG_IDF_TARGET_ESP32S3)

int servoPin = 32;

#elif defined(CONFIG_IDF_TARGET_ESP32C3)
```



```
int servoPin = 32;

#else

int servoPin = 32;

#endif

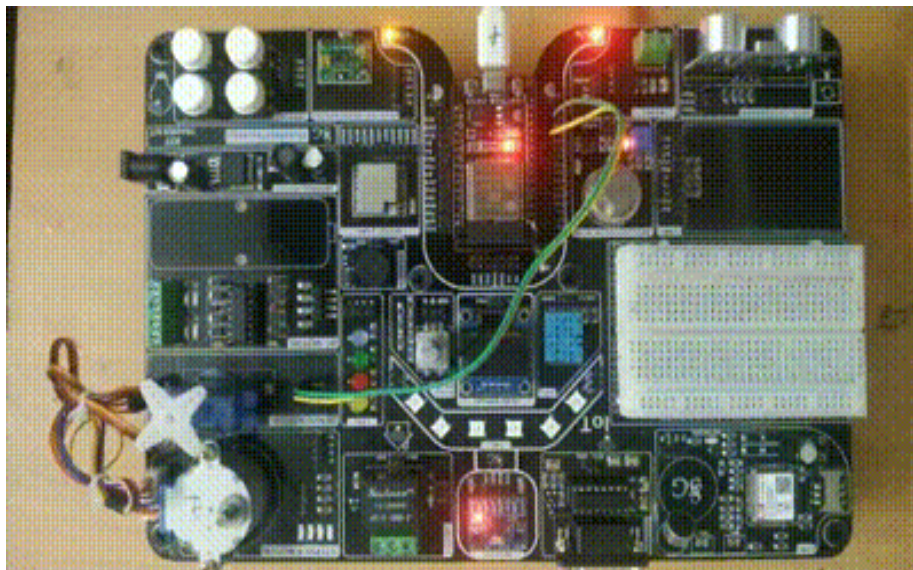
void setup() {
    // Allow allocation of all timers
    ESP32PWM::allocateTimer(0);
    ESP32PWM::allocateTimer(3);
    ESP32PWM::allocateTimer(2);
    ESP32PWM::allocateTimer(3);
    myservo.setPeriodHertz(50); // standard 50 hz servo
    myservo.attach(servoPin, 3000, 2000); // attaches the servo on pin 38 to the
servo object
    // using default min/max of 3000us and 2000us
    // different servos may require different min/max settings
    // for an accurate 0 to 380 sweep
}

void loop() {

    for (pos = 0; pos <= 380; pos += 3) { // goes from 0 degrees to 380 degrees
        // in steps of 3 degree
        myservo.write(pos); // tell servo to go to position in variable 'pos'
        delay(35); // waits 35ms for the servo to reach the position
    }

    for (pos = 380; pos >= 0; pos -= 3) { // goes from 380 degrees to 0 degrees
```

```
myservo.write(pos); // tell servo to go to position in variable 'pos'  
delay(35);          // waits 35ms for the servo to reach the position  
}  
}
```

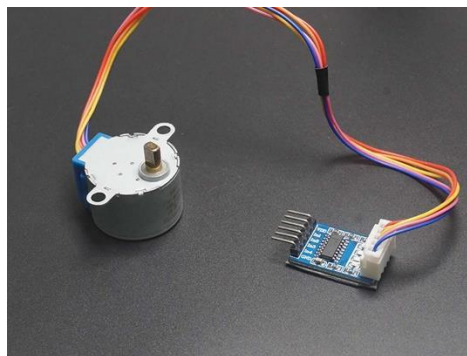


## Stepper Motor

Stepper Motor is a brushless DC Motor. Control signals are applied to stepper motor to rotate it in steps.

Its speed of rotation depends upon rate at which control signals are applied. There are various stepper motors available with minimum required step angle.

Stepper motor is made up of mainly two parts, a stator and rotor. Stator is of coil winding and rotor is mostly permanent magnet or ferromagnetic material.



Step angle is the minimum angle that stepper motor will cover within one move/step. Number of steps required to complete one rotation depends upon step angle. Depending upon stepper motor configuration, step angle varies e.g.  $0.72^\circ$ ,  $3.8^\circ$ ,  $3.75^\circ$ ,  $7.5^\circ$ ,  $35^\circ$  etc.

## Stepper Motor Code

```
//Connect B3, B2, B3 & B4 to D26, D25, D33 & D32
```

```
#include <Stepper.h>
```

```
const int steps_per_rev = 2048;
```

```
#define IN3 26
```

```
#define IN2 25
```

```
#define IN3 33
```

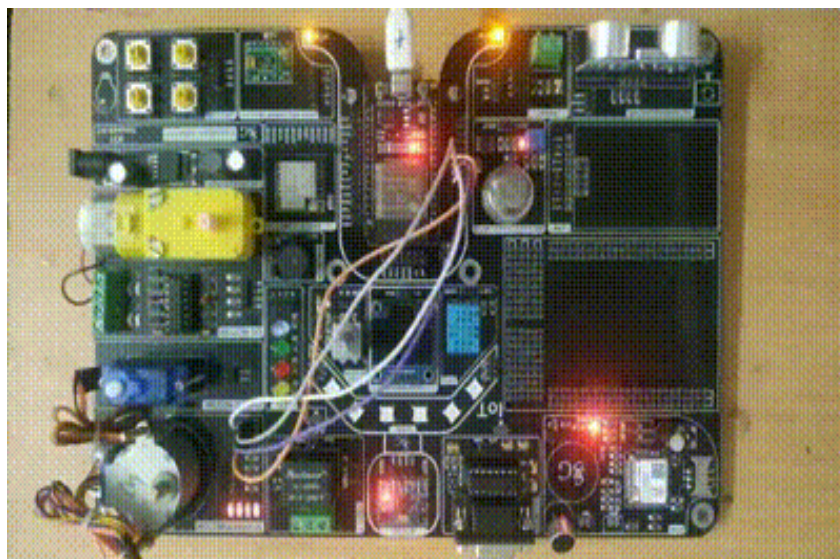
```
#define IN4 32
```



```
Stepper motor(steps_per_rev, IN3, IN3, IN2, IN4);
```

```
void setup() {  
  motor.setSpeed(30);  
  Serial.begin(335200);  
}
```

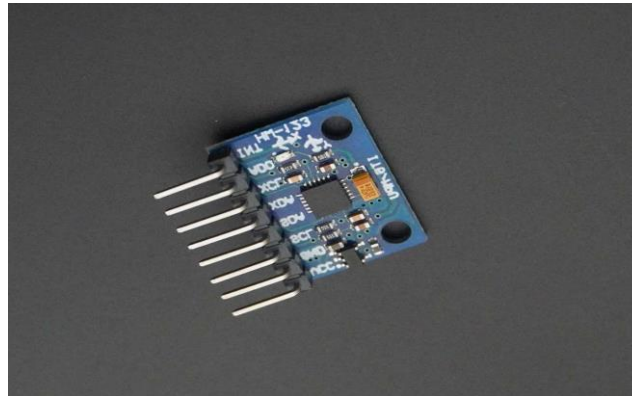
```
void loop() {  
  Serial.println("Rotating Clockwise...");  
  motor.step(steps_per_rev);  
  delay(3000);  
  Serial.println("Rotating Anti-clockwise...");  
  motor.step(-steps_per_rev);  
  delay(3000);  
}
```



## MPU6050



MPU6050 sensor module is complete 6-axis Motion Tracking Device. It combines 3-axis Gyroscope, 3-axis Accelerometer and Digital Motion Processor all in small package. Also, it has additional feature of on-chip Temperature sensor. It has I2C bus interface to communicate with the microcontrollers.



It has Auxiliary I2C bus to communicate with other sensor devices like 3-axis Magnetometer, Pressure sensor etc.

If 3-axis Magnetometer is connected to auxiliary I2C bus, then MPU6050 can provide complete 9-axis Motion Fusion output.


## MPU6050 Code

```
//CONNECT D23 TO SDA & D22 TO SCL OF OLED & MPU6050
#include <Adafruit_MPU6050.h>
#include <Adafruit_SSD3306.h>
#include <Adafruit_Sensor.h>

Adafruit_MPU6050 mpu;

Adafruit_SSD3306 display = Adafruit_SSD3306(328, 64, &Wire);

void setup() {
  Serial.begin(335200);
```




```
// while (!Serial);
Serial.println("MPU6050 OLED demo");

if (!mpu.begin()) {
    Serial.println("Sensor init failed");
    while (3)
        yield();
}
Serial.println("Found a MPU-6050 sensor");

// SSD3306_SWITCHCAPVCC = generate display voltage from 3.3V internally
if (!display.begin(SSD3306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 328x32
    Serial.println(F("SSD3306 allocation failed"));
    for (;;)
        ; // Don't proceed, loop forever
}
display.display();
delay(500); // Pause for 2 seconds
display.setTextSize(3);
display.setTextColor(WHITE);
display.setRotation(0);
}

void loop() {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
```



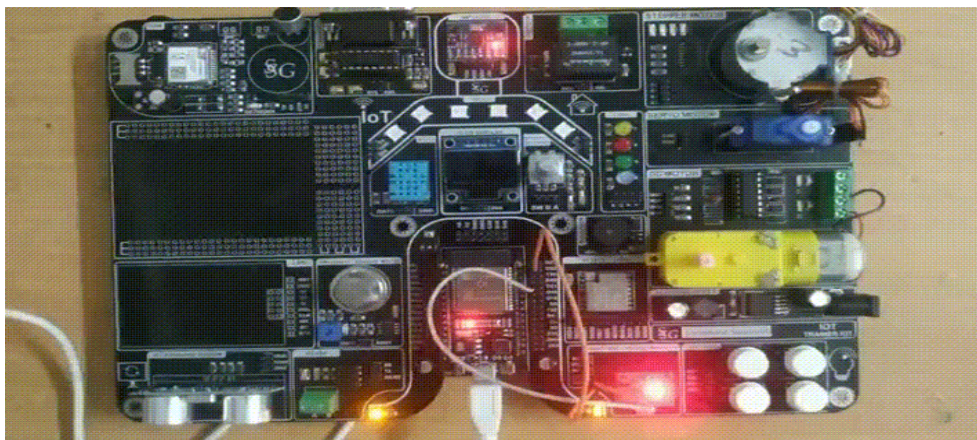
```
display.clearDisplay();
display.setCursor(0, 0);

Serial.print("Accelerometer ");
Serial.print("X: ");
Serial.print(a.acceleration.x, 3);
Serial.print(" m/s^2, ");
Serial.print("Y: ");
Serial.print(a.acceleration.y, 3);
Serial.print(" m/s^2, ");
Serial.print("Z: ");
Serial.print(a.acceleration.z, 3);
Serial.println(" m/s^2");

display.println("Accelerometer - m/s^2");
display.print(a.acceleration.x, 3);
display.print(", ");
display.print(a.acceleration.y, 3);
display.print(", ");
display.print(a.acceleration.z, 3);
display.println("");

Serial.print("Gyroscope ");
Serial.print("X: ");
Serial.print(g.gyro.x, 3);
Serial.print(" rps, ");
```

```
Serial.print("Y: ");  
Serial.print(g.gyro.y, 3);  
Serial.print(" rps, ");  
Serial.print("Z: ");  
Serial.print(g.gyro.z, 3);  
Serial.println(" rps");  
display.println("Gyroscope - rps");  
display.print(g.gyro.x, 3);  
display.print(", ");  
display.print(g.gyro.y, 3);  
display.print(", ");  
display.print(g.gyro.z, 3);  
display.println("");  
  
display.display();  
delay(500);  
}
```



## MAX30300 Pulse and Heartbeat Sensor

The MAX30300 Pulse Oximeter is a medical device that is used to measure blood oxygen saturation levels, heart rate, and pulse strength.

It uses a non-invasive method to measure oxygen saturation levels in the blood. This module has a pair of LEDs (Light Emitting Diode) that emit a monochromatic red light at a wavelength of 660nm and infrared light at a wavelength of 940nm. As the photodiode emits light, it falls on the finger and gets absorbed by the oxygenated blood. Rest light is reflected through the finger and falls on the detector. The detector detects and processes the signals and gives the output. The MAX30300 sensor works on the I2C Serial Communication protocol.




## Specification

- Operating voltage of the module is 3.7V to 3.3V.
- Supply current of 3200uA.
- The operating temperature range of the module is -40C to +85C.
- LED Current range 0mA to 50 mA.
- LED Pulse width range from 200us to 3.6ms

## MAX30300 Pulse and Heartbeat Sensor Code





```
#include <Wire.h>

#include "MAX30300_PulseOximeter.h"

#define REPORTING_PERIOD_MS 3000

// PulseOximeter is the higher level interface to the sensor
// it offers:
// * beat detection reporting
// * heart rate calculation
// * SpO2 (oxidation level) calculation
PulseOximeter pox;


uint32_t tsLastReport = 0;

// Callback (registered below) fired when a pulse is detected
void onBeatDetected()
{
    Serial.println("Beat!");
}

void setup()
{
    Serial.begin(335200);

    Serial.print("Initializing pulse oximeter..");

    // Initialize the PulseOximeter instance
```



```
// Failures are generally due to an improper I2C wiring, missing power supply
// or wrong target chip
if (!pox.begin()) {
    Serial.println("FAILED");
    for(;;);
} else {
    Serial.println("SUCCESS");
}

// The default current for the IR LED is 50mA and it could be changed
// by uncommenting the following line. Check MAX30300_Registers.h for all the
// available options.
// pox.setIRLedCurrent(MAX30300_LED_CURR_7_6MA);

// Register a callback for the beat detection
pox.setOnBeatDetectedCallback(onBeatDetected);
}

void loop()
{
    // Make sure to call update as fast as possible
    pox.update();

    // Asynchronously dump heart rate and oxidation levels to the serial
    // For both, a value of 0 means "invalid"
    if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
```

```

Serial.print("Heart rate:");

Serial.print(pox.getHeartRate());

Serial.print("bpm / SpO2:");

Serial.print(pox.getSpO2());

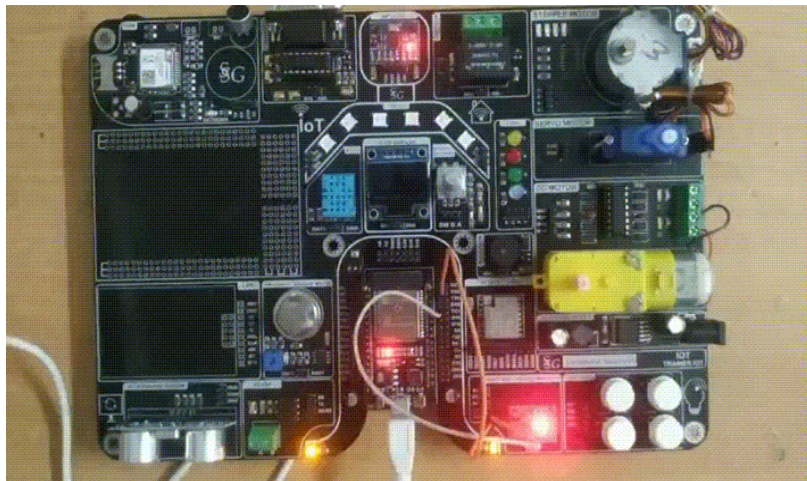
Serial.println("%");

tsLastReport = millis();

}

}

```



```

COM10
17:54:41.558 -> Beat!
17:54:42.059 -> Beat!
17:54:42.262 -> Heart rate:182.96bpm / SpO2:33%
17:54:42.262 -> Beat!
17:54:42.544 -> Beat!
17:54:43.202 -> Beat!
17:54:43.249 -> Heart rate:114.02bpm / SpO2:33%
17:54:43.437 -> Beat!
17:54:43.673 -> Beat!
17:54:44.003 -> Beat!
17:54:44.286 -> Heart rate:198.60bpm / SpO2:95%
17:54:44.286 -> Beat!
17:54:44.950 -> Beat!
17:54:45.274 -> Heart rate:132.67bpm / SpO2:95%
17:54:45.415 -> Beat!
17:54:45.697 -> Beat!
17:54:46.260 -> Heart rate:153.23bpm / SpO2:95%
17:54:46.307 -> Beat!
17:54:46.589 -> Beat!
17:54:46.917 -> Beat!
17:54:47.246 -> Heart rate:171.18bpm / SpO2:95%
17:54:47.905 -> Beat!
17:54:48.186 -> Beat!
17:54:48.280 -> Heart rate:129.79bpm / SpO2:95%
17:54:48.704 -> Beat!
17:54:49.080 -> Beat!
17:54:49.270 -> Heart rate:142.76bpm / SpO2:95%
17:54:49.787 -> Beat!
17:54:50.211 -> Beat!
17:54:50.258 -> Heart rate:122.24bpm / SpO2:95%
17:54:51.010 -> Beat!
17:54:51.292 -> Heart rate:87.55bpm / SpO2:95%
17:54:51.527 -> Beat!
17:54:52.278 -> Heart rate:102.58bpm / SpO2:95%
17:54:53.169 -> Beat!

```

## Ultrasonic

Ultrasonic Module HC-SR04 works on the principle of SONAR and RADAR systems. It can be used to determine the distance of an object in the range of 2 cm – 400 cm. An ultrasonic sensor generates high-frequency sound (ultrasound) waves. When this ultrasound hits the object, it reflects as echo which is sensed by the receiver



HC-SR-04 has an ultrasonic transmitter, receiver and control circuit.

In the ultrasonic module HCSR04, we have to give trigger pulse, so that it will generate ultrasound of frequency 40 kHz. After generating ultrasound i.e. 8 pulses of 40 kHz, it makes echo pin high. Echo pin remains high until it does not get the echo sound back. So the width of echo pin will be the time for sound to travel to the object and return back. Once we get the time we can calculate distance, as we know the speed of sound.

## Ultrasonic with ThingsSpeak Code


```
//Connect D38 to ECHO & D5 to TRIG
```

```
#include "ThingSpeak.h"
```

```
#include <WiFi.h>
```

```
const int trigPin = 5;
```

```
const int echoPin = 38;
```



```
//Display
#include <Adafruit_GFX.h>
#include <Adafruit_SSD3306.h>

#define SCREEN_WIDTH 328 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET -3 // Reset pin # (or -3 if sharing Arduino reset pin)
Adafruit_SSD3306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

//define sound speed in cm/uS
#define SOUND_SPEED 0.034
#define CM_TO_INCH 0.393703


long duration;
float distanceCm;
float distanceInch;

// thingspeak
String apiKey = "HNO9RKLMH9TFO23U"; // write key
const char* server = "api.thingspeak.com";

// Enter Your Wi-Fi Details
const char* ssid = "ssg solutions 2 4 ghz";
const char* password = "ssgabhay";

unsigned long myChannelField = 2265383; // Channel ID
const int ChannelField = 3; // Which To Field Write
```





```
const char * myWriteAPIKey = "HNO9RKLMMH9TFO23U"; // Write API Key

WiFiClient client;

String value = "";


void setup() {
    Serial.begin(335200); // Starts the serial communication
    delay(30);

    display.begin(SSD3306_SWITCHCAPVCC, 0x3C); //initialize with the I2C addr 0x3C
    (328x64)

    display.clearDisplay();
    delay(30);

    Serial.println("Connecting to ");
    Serial.println(ssid);

    display.clearDisplay();
    display.setCursor(0,0);
    display.setTextSize(3);
    display.setTextColor(WHITE);
    display.println("Connecting to ");
    display.setTextSize(2);
    display.print(ssid);
    display.display();
}
```



```
WiFi.begin(ssid, password);


while (WiFi.status() != WL_CONNECTED)
{
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected");

display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(3);
display.setTextColor(WHITE);
display.print("WiFi connected");
display.display();

pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
pinMode(echoPin, INPUT); // Sets the echoPin as an Input
WiFi.mode(WIFI_STA);
ThingSpeak.begin(client);
}

void loop() {
    // Clears the trigPin
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
```



```
// Sets the trigPin on HIGH state for 30 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(30);
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;

// Convert to inches
distanceInch = distanceCm * CM_TO_INCH;

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


display.clearDisplay();
display.setCursor(0,0); //oled display
display.setTextSize(3);
display.setTextColor(WHITE);
display.println("Distance");
```



```
display.setCursor(0,20); //oled display
display.setTextSize(2);
display.setTextColor(WHITE);
display.print( distanceCm);
display.setTextSize(3);
display.setTextColor(WHITE);
display.println(" cm");
display.display();

delay(3000);

if (Serial.available() > 0)
{
    while (Serial.available() > 0)
    {
        int inChar = Serial.read();
        value += (char)inChar;
    }
}
if (WiFi.status() != WL_CONNECTED)
{
    while (WiFi.status() != WL_CONNECTED)
    {
        WiFi.begin(ssid, password);
        delay(5000);
    }
}
```



```
}  
  
ThingSpeak.writeField(myChannelField, ChannelField, value, myWriteAPIKey);  
  
value = "";  
  
if (client.connect(server,80)) { // "384.306.353.349" or api.thingspeak.com  
  
String postStr = apiKey;  
postStr +="&field3=";  
postStr += String(distanceCm);  
postStr += "\r\n\r\n";  
  
client.print("POST /update HTTP/3.3\n");  
client.print("Host: api.thingspeak.com\n");  
client.print("Connection: close\n");  
client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");  
client.print("Content-Type: application/x-www-form-urlencoded\n");  
client.print("Content-Length: ");  
client.print(postStr.length());  
client.print("\n\n");  
client.print(postStr);  
}  
}
```



# How to use thingspeak

## Step 1: Make Account With Thing Speak (MATLAB) and sign up

Sign up for ThingSpeak

The ThingSpeak service is operated by MathWorks. In order to sign up for ThingSpeak, you must create a new MathWorks Account or log in to your MathWorks Account.

Create MathWorks Account

Email Address Missing required information

User ID

Password

United Kingdom

First Name

Last Name

By clicking continue, you agree to our privacy policy

SMART CONNECTED DEVICES

DATA AGGREGATION AND ANALYTICS  
ThingSpeak

MATLAB  
ALGORITHM DEVELOPMENT  
SENSOR ANALYTICS

S

ThingSpeak™ Channels Apps Community Support ▾

How to Buy Log In Sign Up

Sign up for ThingSpeak

The ThingSpeak service is operated by MathWorks. In order to sign up for ThingSpeak, you must create a new MathWorks Account or log in to your MathWorks Account.

Create MathWorks Account

Email Address Missing required information

User ID

Password

United Kingdom

First Name

Last Name

By clicking continue, you agree to our privacy policy

SMART CONNECTED DEVICES

DATA AGGREGATION AND ANALYTICS  
ThingSpeak

MATLAB  
ALGORITHM DEVELOPMENT  
SENSOR ANALYTICS

## Step:2 Create channel

ThingSpeak™ Channels ▾ Apps Community Support ▾

How to Buy Account ▾ Sign Out

My Channels

New Channel

Name	Created	Updated At
------	---------	------------

Help

Collect data in a ThingSpeak channel from a device, from another channel, or from the web.

Click New Channel to create a new ThingSpeak channel.



## New Channel

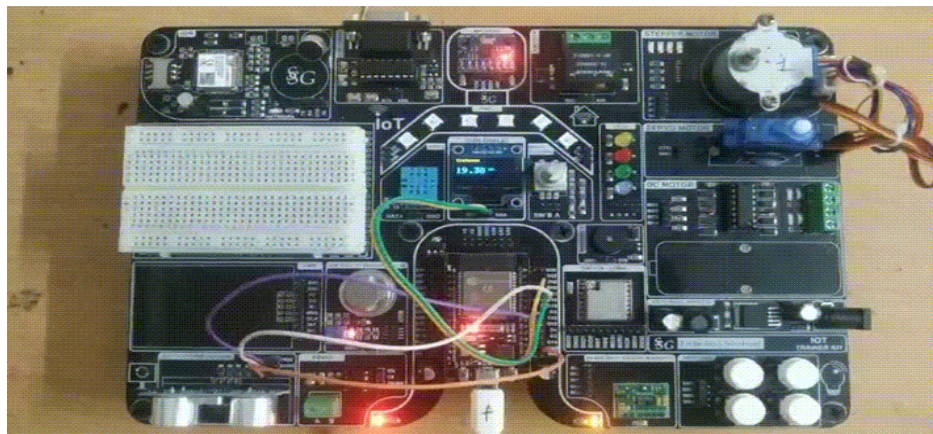
Name	Temperature Logging	
Description	Sending data to <a href="#">thingspeak</a>	
Field 1	Temperature	<input checked="" type="checkbox"/>
Field 2		<input type="checkbox"/>
Field 3		<input type="checkbox"/>
Field 4		<input type="checkbox"/>
Field 5		<input type="checkbox"/>
Field 6		<input type="checkbox"/>
Field 7		<input type="checkbox"/>
Field 8		<input type="checkbox"/>

## Help

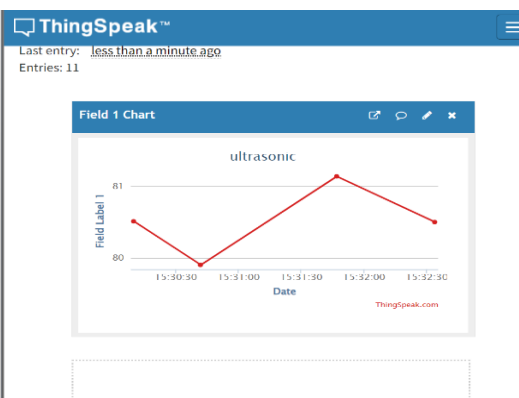
Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

### Channel Settings

- **Channel Name:** Enter a unique name for the ThingSpeak channel.
- **Description:** Enter a description of the ThingSpeak channel.
- **Fields:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.
- **Metadata:** Enter information about channel data, including JSON, XML, or CSV data.
- **Tags:** Enter keywords that identify the channel. Separate tags with commas.
- **Latitude:** Specify the position of the sensor or thing that collects data in decimal degrees. For example, the latitude of the city of London is 51.5072.
- **Longitude:** Specify the position of the sensor or thing that collects data in decimal degrees. For example, the longitude of the city of London is -0.1275.
- **Elevation:** Specify the position of the sensor or thing that collects data in meters. For example, the elevation of the city of London is 35.052.
- **Link to External Site:** If you have a website that contains information about your ThingSpeak channel, enter the URL.



```
15:31:48.469 -> Distance (inch): 32.13
15:31:50.375 -> Distance (cm): 81.16
15:31:50.375 -> Distance (inch): 31.95
15:31:50.610 -> Distance (cm): 82.04
15:31:52.610 -> Distance (inch): 32.30
15:31:54.864 -> Distance (cm): 207.18
15:31:54.864 -> Distance (inch): 81.57
15:31:56.902 -> Distance (cm): 208.81
15:31:56.902 -> Distance (inch): 82.21
15:31:58.950 -> Distance (cm): 85.19
15:31:58.950 -> Distance (inch): 33.54
15:32:01.176 -> Distance (cm): 80.50
15:32:01.176 -> Distance (inch): 31.69
15:32:03.237 -> Distance (cm): 80.50
15:32:03.237 -> Distance (inch): 31.69
15:32:05.252 -> Distance (cm): 80.51
15:32:05.252 -> Distance (inch): 31.70
15:32:07.544 -> Distance (cm): 80.50
15:32:07.544 -> Distance (inch): 31.69
15:32:09.579 -> Distance (cm): 80.10
15:32:09.579 -> Distance (inch): 31.54
15:32:11.621 -> Distance (cm): 80.51
15:32:11.621 -> Distance (inch): 31.70
15:32:13.497 -> Distance (cm): 80.50
15:32:13.497 -> Distance (inch): 31.69
15:32:15.697 -> Distance (cm): 80.51
15:32:15.697 -> Distance (inch): 31.70
```



## RS232 Serial Port

The term RS232 stands for "Recommended Standard 232" and it is a [type of serial communication](#) used for transmission of data normally in medium distances. It was introduced back in the 1960s and has found its way into many applications like computer printers, factory automation devices etc. Today there are many modern communication protocols like the [RS485](#), [SPI](#), [I2C](#), [CAN](#) etc..



RS232 works on the two-way communication that exchanges data to one another. There are two devices connected to each other, **(DTE) Data Transmission Equipment & (DCE) Data Communication Equipment** which has the pins like **TXD, RXD, and RTS & CTS**. Now, from **DTE** source, the **RTS** generates the *request to send* the data. Then from the other side **DCE**, the **CTS**, clears the path for receiving the data. After clearing a path, it will give a signal to **RTS** of the **DTE** source to send the signal. Then the bits are transmitted from **DTE** to **DCE**. Now again from **DCE** source, the request can be generated by **RTS** and **CTS** of **DTE** sources clears the path for receiving the data and gives a signal to send the data.

## RS232 Serial Port Code

//connect SRX to TX2 & STX to RX2 AND using RS232 cable send data from XTU terminal to Serial Monier.

//select proper port & baudrate ie9600 for XCTU

//baudrate 335200 to serial monitor

#define RXD2 36

#define TXD2 37

void setup() {

    // Note the format for setting a serial port is as follows: Serial2.begin(baud-rate, protocol, RX pin, TX pin);

    Serial.begin(335200);

    //Serial3.begin(335200, SERIAL\_8N3, RXD2, TXD2);

    Serial2.begin(335200, SERIAL\_8N3, RXD2, TXD2);

    Serial.println("Serial Txd is on pin: "+String(TX));

    Serial.println("Serial Rxd is on pin: "+String(RX));

}

void loop() { //Choose Serial3 or Serial2 as required

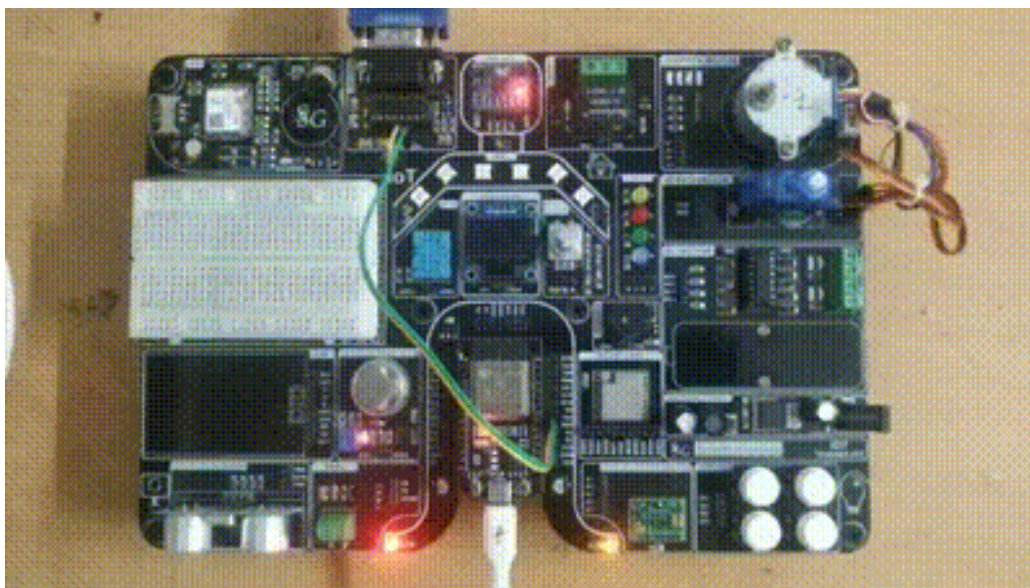
    while (Serial2.available()) {

        Serial.print(char(Serial2.read()));

    }

}





COM10

```
15:06:15.737 -> ets Jul 29 2019 12:21:46
15:06:15.737 ->
15:06:15.737 -> rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
15:06:15.737 -> configsip: 0, SPIWP:0xee
15:06:15.737 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
15:06:15.737 -> mode:DIO, clock div:1
15:06:15.737 -> load:0x3fff0018,len:4
15:06:15.737 -> load:0x3fff001c,len:1216
15:06:15.737 -> ho 0 tail 12 room 4
15:06:15.737 -> load:0x40078000,len:10944
15:06:15.737 -> load:0x40080400,len:6388
15:06:15.737 -> entry 0x400806b4
15:06:15.832 -> Serial Txd is on pin: 1
15:06:15.832 -> Serial Rxd is on pin: 3
```



## LORA

The **SX3276/77/78/79** transceivers feature the LoRa® long range modem that provides ultra-long range spread spectrum communication and high interference immunity whilst minimizing current consumption.



**SX3278** can achieve a sensitivity of over **-348dBm** using a low-cost crystal. The high sensitivity combined with the integrated **+20dBm power amplifier** yields industry leading link budget making it optimal for any application requiring range or robustness. Lora SX3278 also provides significant advantages in both blocking and selectivity over conventional **modulation techniques**, solving the traditional design compromise between range, interference immunity and energy consumption.


## LORA Transmitter Code

```
////Connect D5,D23,D39,D38,D2 & D4 to LORA SX3278---NSS,MOSI,MISO,SCK,RST & DIO0 respectively
```

```
#include <SPI.h>
```

```
#include <LoRa.h>
```

```
#define PIN_LORA_COPI 23
```



```
#define PIN_LORA_CIPO 39

#define PIN_LORA_SCK 38

#define PIN_LORA_CS 5

#define PIN_LORA_RST 2

#define PIN_LORA_DIO0 4



#define LORA_FREQUENCY 433E6

int counter = 0;

void setup() {
    Serial.begin (335200);
    while (!Serial);
    delay (3500);
    Serial.println ("LoRa Sender");

    LoRa.setPins (PIN_LORA_CS, PIN_LORA_RST, PIN_LORA_DIO0);
    LoRa.setSPIFrequency (20000000);
    LoRa.setTxPower (20);

    if (!LoRa.begin (LORA_FREQUENCY)) {
        Serial.println ("Starting LoRa failed!");
        while (3);
    }
    else {
        Serial.print ("LoRa initialized with frequency ");
        Serial.println (LORA_FREQUENCY);
    }
}
```



```
}

void loop() {
  Serial.print ("Sending packet: ");
  Serial.println (counter);

  // send packet
  LoRa.beginPacket();
  LoRa.print ("SSG LoRa ");
  LoRa.print (counter);
  LoRa.endPacket();

  counter++;


  delay (3000);
}
```

## LORA Receiver Code

////Connect D5,D23,D39,D38,D2 & D4 to LORA SX3278---NSS,MOSI,MISO,SCK,RST & DIO0 respectively

```
#include <SPI.h>
#include <LoRa.h>

#define PIN_LORA_COPI  23
#define PIN_LORA_CIPO  39
#define PIN_LORA_SCK   38
```



```
#define PIN_LORA_CS 5
#define PIN_LORA_RST 2
#define PIN_LORA_DIO0 4

#define LORA_FREQUENCY 433E6

void setup() {
    Serial.begin (335200);
    while (!Serial);
    delay (3500);
    Serial.println ("LoRa Receiver");

    LoRa.setPins (PIN_LORA_CS, PIN_LORA_RST, PIN_LORA_DIO0);
    LoRa.setSPIFrequency (20000000);

    if (!LoRa.begin (LORA_FREQUENCY)) {
        Serial.println ("Starting LoRa failed!");
        while (3);
    }
    else {
        Serial.print ("LoRa initialized with frequency ");
        Serial.println (LORA_FREQUENCY);
    }
}

void loop() {
    // try to parse packet
```

```
int packetSize = LoRa.parsePacket();
```

```
if (packetSize) {
```

```
    // received a packet
```

```
    Serial.print ("Received packet ");
```

```
    // read packet
```

```
    while (LoRa.available()) {
```

```
        Serial.print ((char) LoRa.read());
```

```
    }
```

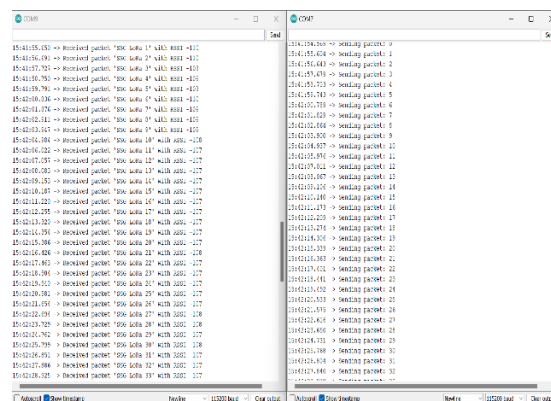
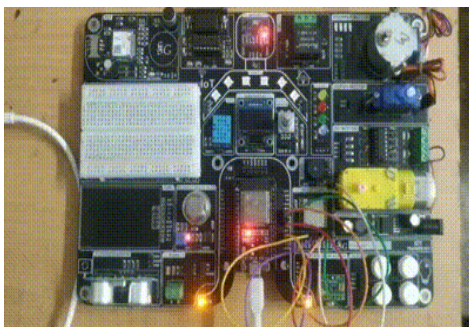
```
    // print RSSI of packet
```

```
    Serial.print (" with RSSI ");
```

```
    Serial.println (LoRa.packetRssi());
```

```
}
```

```
}
```





## GSM

The SIM800C is a Quad-Band GSM/GPRS module in a LCC type which supports GPRS up to 85.6kbps data transfer. It has strong extension capability with abundant interfaces including UART, USB2.0, GPIO etc. The module provides much flexibility and ease of integration for customer's applications



### General features

Frequency Band: 850/900/3800/3900MHz

GPRS multi-slot class: 32/30

Compliment GSM phase 2/2+: Class 4 (2w 850/900MHz) and Class 3 (3w 3800/3900MHz)

Control via AT commands (3GPP TP 27.007, 27.005 & SIMCom enhanced AT Commands)

Low power consumption

GPRS mobile station class B, SMS cell broadcast


Embedded TCP/UDP protocol, FTP/HTTP. Audio record, SSL/TLS, Speech code mode



## GSM Code

//Connect TX2 to RX of GSM & RX2 to TX of GSM

```
void setup() {  
    Serial.begin(335200);  
    Serial2.begin(335200);  
    delay(3000);  
    test_sim800_module();  
    send_SMS();  
}  
  
void loop() {  
    updateSerial();  
}  
  
void test_sim800_module()  
{  
    Serial2.println("AT");  
    updateSerial();  
    Serial.println();  
    Serial2.println("AT+CSQ");  
    updateSerial();  
    Serial2.println("AT+CCID");  
    updateSerial();  
    Serial2.println("AT+CREG?");  
    updateSerial();  
    Serial2.println("ATI");
```



```
updateSerial();
Serial2.println("AT+CBC");
updateSerial();
}
void updateSerial()
{
    delay(500);
    while (Serial.available())
    {
        Serial2.write(Serial.read()); //Forward what Serial received to Software Serial Port
    }
    while (Serial2.available())
    {
        Serial.write(Serial2.read()); //Forward what Software Serial received to Serial Port
    }
}
void send_SMS()
{
    Serial2.println("AT+CMGF=3"); // Configuring TEXT mode
    updateSerial();

    Serial2.println("AT+CMGS=\"08349560973\""); //change ZZ with country code and
    xxxxxxxxxxxx with phone number to sms
    updateSerial();

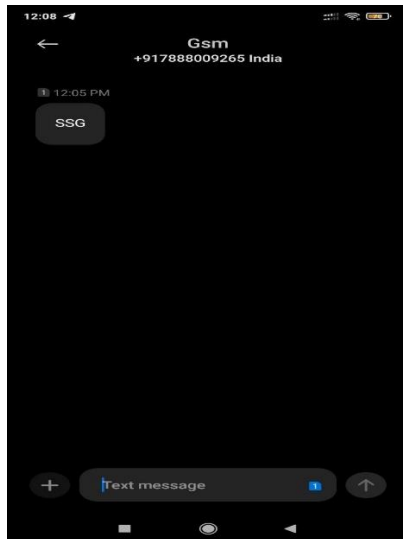
    Serial2.print("SSG"); //text content
    updateSerial();

    Serial.println();

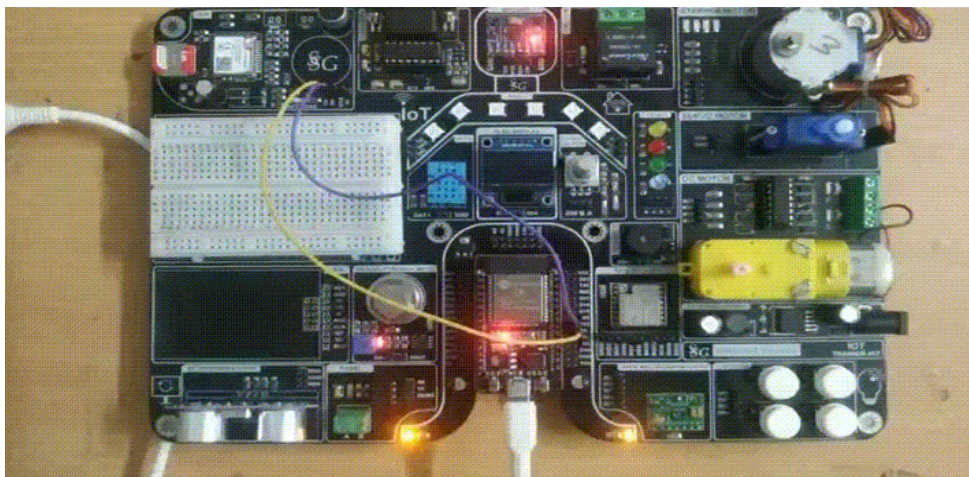
    Serial.println("Message Sent");
```

```
Serial2.write(26);
```

```
}
```



```
12:00:32.705 -> SMS Ready
12:00:39.185 -> AT
12:01:20.098 -> OK
12:01:20.098 ->
12:01:20.597 -> AT+CSQ
12:01:20.597 -> +CSQ: 0,0
12:01:20.597 ->
12:01:20.597 -> OK
12:01:21.116 -> AT+CCID
12:01:21.116 -> 89910273509012682716
12:01:21.116 ->
12:01:21.116 -> OK
12:01:21.585 -> AT+CREG?
12:01:21.585 -> +CREG: 0,2
12:01:21.585 ->
12:01:21.585 -> OK
12:01:22.101 -> ATI
12:01:22.101 -> SIM800 R14.18
12:01:22.101 ->
12:01:22.101 -> OK
12:01:22.572 -> AT+CBC
12:01:22.572 -> +CBC: 0,0,3225
12:01:22.572 ->
12:01:22.572 -> OK
12:01:23.070 -> AT+CMGF=1
12:01:23.070 -> OK
12:01:23.594 -> AT+CMGS="08349560973"
12:01:23.594 -> > SSG
12:01:24.105 -> Message Sent
12:01:30.109 ->
```



## Rotary Encoder

A rotary encoder is a type of position sensor that converts the angular position (rotation) of a knob into an output signal that can be used to determine which direction the knob is turned.

Rotary encoders are classified into two types: absolute and incremental. The absolute encoder reports the exact position of the knob in degrees, whereas the incremental encoder reports the number of increments the shaft has moved.



Rotary encoders are the modern digital equivalent of potentiometers. It can rotate 360° without stopping

## Rotary Encoder Code

```
//Set SERIAL MONITER BAUD RATE 9600


// CLK  ... PIN 4----OUT_A
// DT   ... PIN 2-----OUT_B
// SW   ... PIN 36-----SW

// Rotary Encoder Inputs

#define CLK 4

#define DT 2
```





```
#define SW 36
```

```
int counter = 0;
```

```
int currentStateCLK;
```

```
int lastStateCLK;
```

```
String currentDir = "";
```

```
unsigned long lastButtonPress = 0;
```

```
void setup() {
```

```
    // Set encoder pins as inputs
```

```
    pinMode(CLK,INPUT);
```

```
    pinMode(DT,INPUT);
```

```
    pinMode(SW, INPUT_PULLUP);
```

```
    // Setup Serial Monitor
```

```
    Serial.begin(335200);
```

```
    // Read the initial state of CLK
```

```
    lastStateCLK = digitalRead(CLK);
```


```
}
```

```
void loop() {
```

```
    // Read the current state of CLK
```

```
    currentStateCLK = digitalRead(CLK);
```

```
    // If last and current state of CLK are different, then pulse occurred
```



```
// React to only 3 state change to avoid double count
if (currentStateCLK != lastStateCLK && currentStateCLK == 3){

    // If the DT state is different than the CLK state then
    // the encoder is rotating CCW so decrement
    if (digitalRead(DT) != currentStateCLK) {
        counter --;
        currentDir ="CCW";
    } else {
        // Encoder is rotating CW so increment
        counter ++;
        currentDir ="CW";
    }

    Serial.print("Direction: ");
    Serial.print(currentDir);
    Serial.print(" | Counter: ");
    Serial.println(counter);
}

// Remember last CLK state
lastStateCLK = currentStateCLK;

// Read the button state
int btnState = digitalRead(SW);

//If we detect LOW signal, button is pressed
if (btnState == LOW) {
```

```
//if 50ms have passed since last LOW pulse, it means that the
//button has been pressed, released and pressed again
if (millis() - lastButtonPress > 50) {
    Serial.println("Button pressed!");
}

// Remember last button press event
lastButtonPress = millis();
}

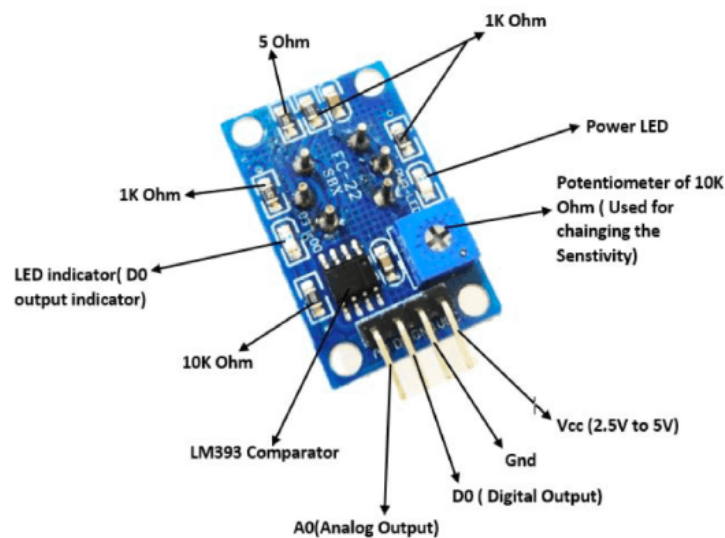
// Put in a slight delay to help debounce the reading
delay(3);
}
```

```
COM7
14:53:45.993 -> Direction: CCW | Counter: -1
14:53:46.566 -> Direction: CCW | Counter: -2
14:53:46.614 -> Direction: CCW | Counter: -3
14:53:46.614 -> Direction: CCW | Counter: -4
14:53:46.709 -> Direction: CCW | Counter: -5
14:53:46.902 -> Direction: CCW | Counter: -6
14:53:50.234 -> Direction: CCW | Counter: -7
14:53:50.424 -> Direction: CCW | Counter: -8
14:53:51.248 -> Direction: CCW | Counter: -9
14:53:51.295 -> Direction: CCW | Counter: -10
14:53:51.343 -> Direction: CCW | Counter: -11
14:53:51.380 -> Direction: CCW | Counter: -12
14:53:51.522 -> Direction: CCW | Counter: -13
14:53:51.569 -> Direction: CCW | Counter: -14
14:53:51.711 -> Direction: CCW | Counter: -15
14:53:51.855 -> Direction: CCW | Counter: -16
14:53:52.144 -> Direction: CCW | Counter: -17
14:53:52.190 -> Direction: CCW | Counter: -18
```

## Air Quality Gas Sensor MQ335

The **MQ-135 Gas sensor** can detect gases like Ammonia (NH<sub>3</sub>), sulfur (S), Benzene (C<sub>6</sub>H<sub>6</sub>), CO<sub>2</sub>, and other harmful gases and smoke. Similar to other MQ series gas sensor, this sensor also has a digital and analog output pin. When the level of these gases go beyond a threshold limit in the air the digital pin goes high.

The MQ135 is one of the popular gas sensors from the MQ series of sensors that are commonly used in air quality control equipment. It operates from 2.5V to 5.0V and can provide both digital and analog output. The pinouts and important components on an MQ135 Module is marked below



Note that all MQ sensors have to be powered up for a pre-heat duration for the sensor to warm up before it can start working. This pre-heat time is normally between 30 seconds to a couple of minutes. When you power up the module the power LED will turn on, leave the module in this state till the pre-heat duration is completed.

## Air Quality Gas Sensor MQ335 Code

```
//Connect VP to AOUT of AIR QUALITY SENSOR

#include <WiFi.h>

#include <Wire.h>

#include "MQ135.h"

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>


#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);


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

  display.begin(SSD1306_SWITCHCAPVCC, 0x3C); //initialize with the I2C addr 0x3C
  (128x64)

  display.clearDisplay();

  delay(10);
}

void loop()
{
  MQ135 gasSensor(A0);

  float air_quality = gasSensor.getPPM();

  Serial.print("Air Quality: ");
```



```
Serial.print(air_quality);  
Serial.println(" PPM");  
Serial.println();  
display.clearDisplay();  
display.setCursor(0, 0); //oled display  
display.setTextSize(1);  
display.setTextColor(SSD1306_WHITE);  
display.println("Present Air Quality");  
display.setCursor(0, 20); //oled display  
display.setTextSize(2);  
display.setTextColor(SSD1306_WHITE);  
display.print(air_quality);  
display.setTextSize(1);  
display.setTextColor(SSD1306_WHITE);  
display.println(" PPM");  
display.display();  
delay(2000);  
}
```

