Get Started MaESP ESP32 in Arduino

Introduction

The Makerfabs MaESP ESP32 Kit intended to help learners getting started with MicroPython/Arduino on the ESP32. This Kits includes the most common electronic components and modules, for the starters to learn the basic knowledge of MicroPython/Arduino, usage of Thonny IDE and usage of Arduino IDE. 12 basic lessons help starters to learn the usage and program skills about MicroPython/Arduino.

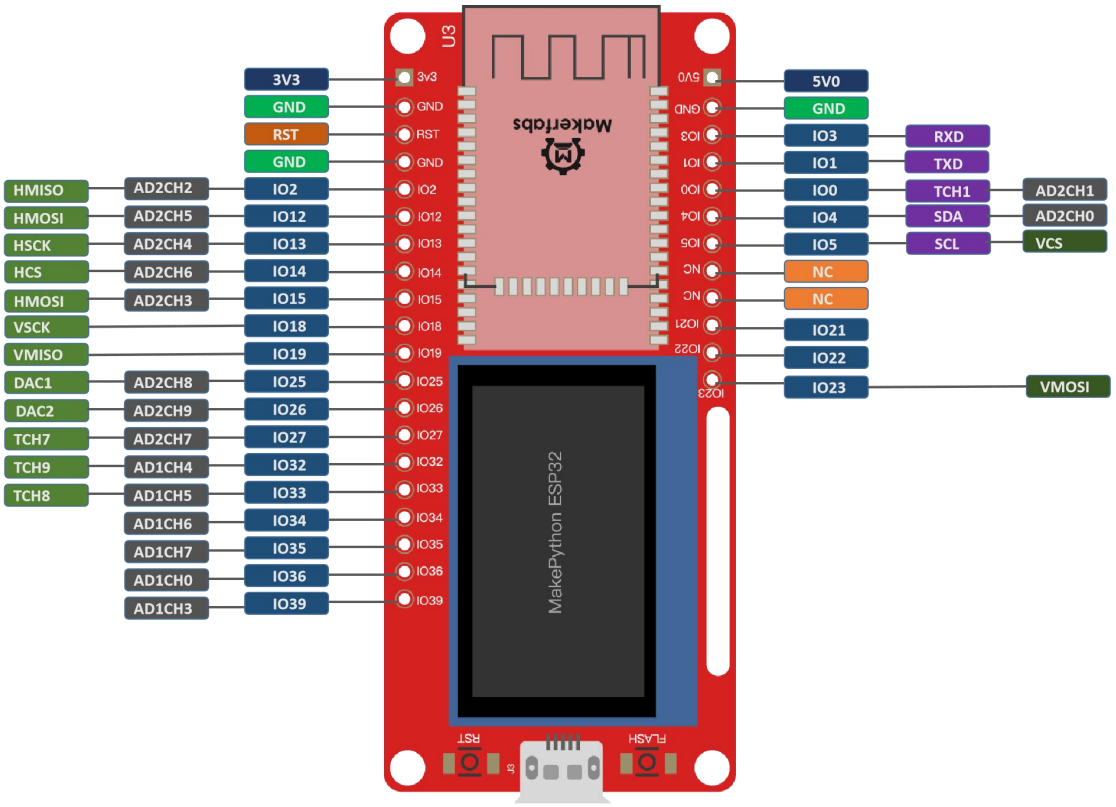
The total learning time for this kits& lessons is about 20 hours.

What is Arduino?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Product List:

Makerfabs MaESP ESP32 Pin figure:



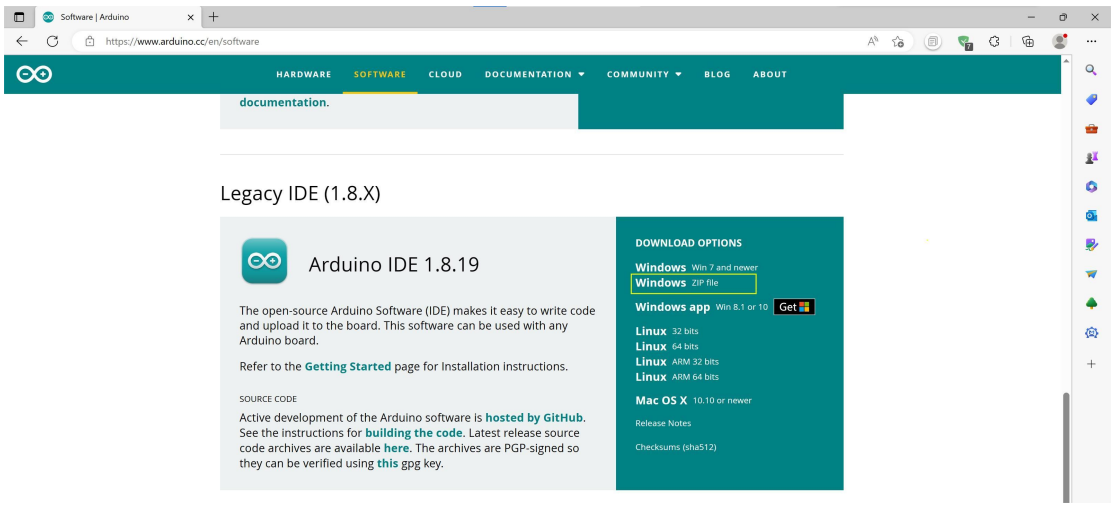
Directory

I. Arduino IDE Development Tool

Install Arduino IDE

For this tutorial, we use Arduino IDE, which is the most simple& easy way for starts to skip into Arduino. You can learn more about Arduino IDE on their GitHub repository or explore the Arduino IDE source code.

Click this link to download Arduino IDE **1.8.19** for Windows: https://www.arduino.cc/en/software.

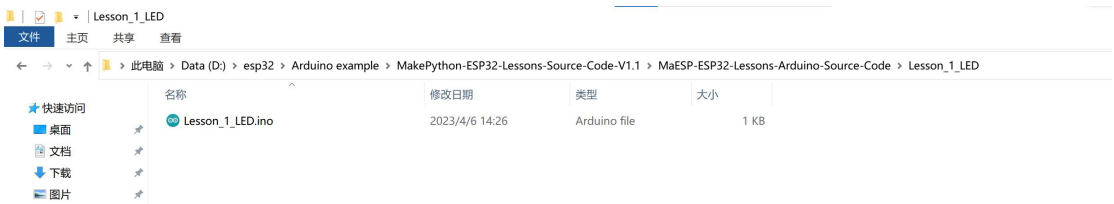


After downloading and unzipping, you should see the arduino.exe in your Downloads folder.

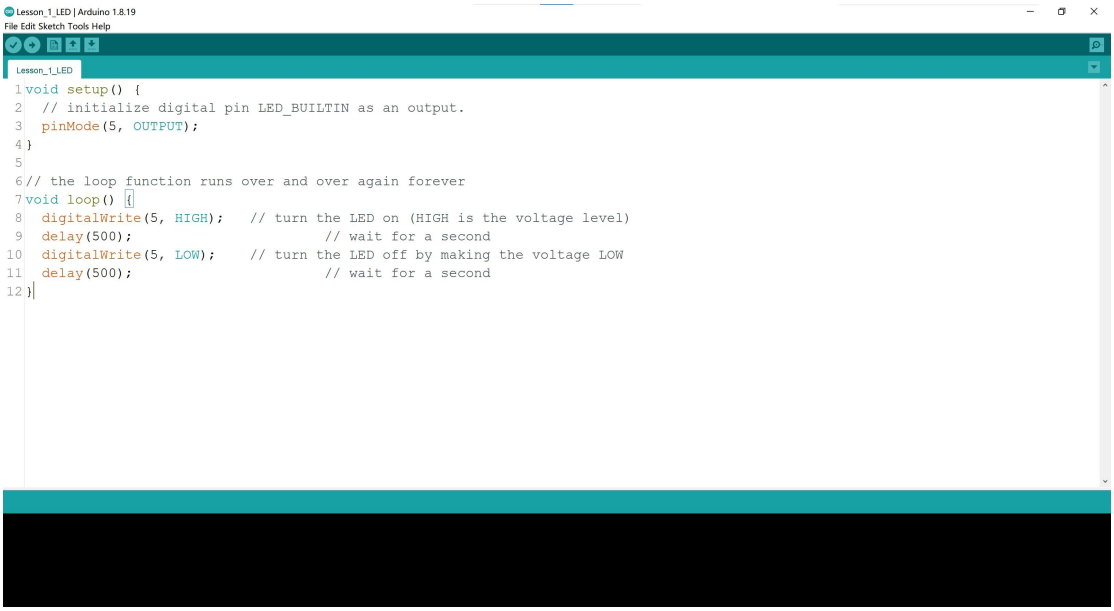
Download the Arduino demo code, and double-click that file to open the Arduino IDE. Running the Lesson 1 LED Arduino demo.

Code download link:

https://www.makerfabs.com/Makepython-esp32-starter-kit.html.



A Simple code for blinking an LED on ESP32 as follows:

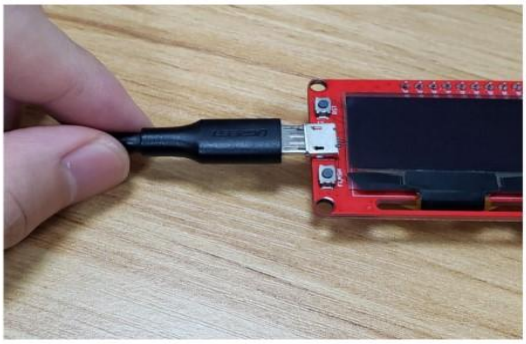


Get Start with Arduino IDE

Open Arduino IDE. Let's learn how to run C/C++ programs on Arduino IDE:

1. Establish a communication with the board

After unpacking the MaESP ESP32, connect it to the PC with a Type-C USB cable.



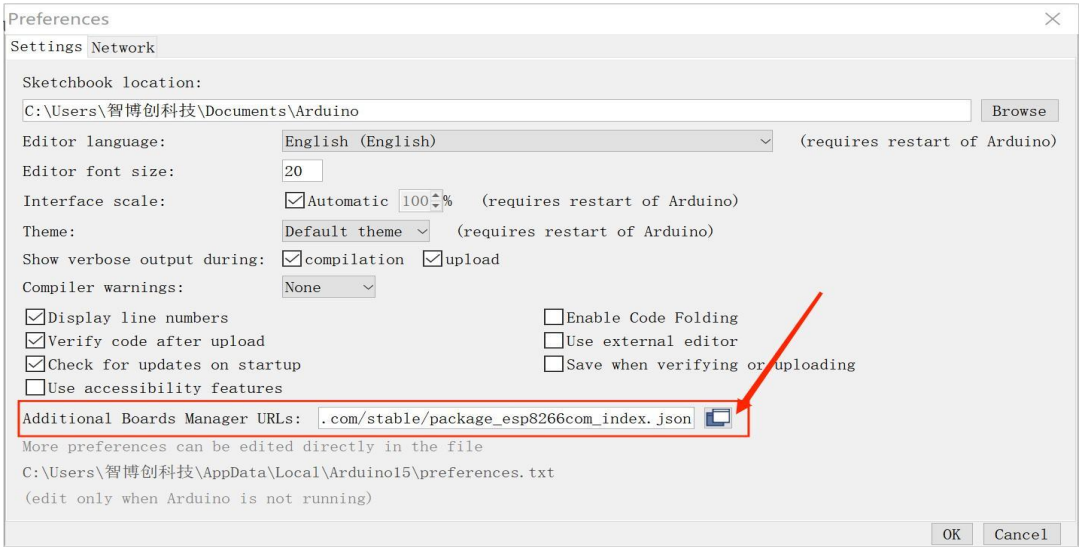
2. Installed the USB driver

• Go to ***Tools > Port：***

select the port your MaESP ESP32 connected (the like for downloading the USB driver is: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers)

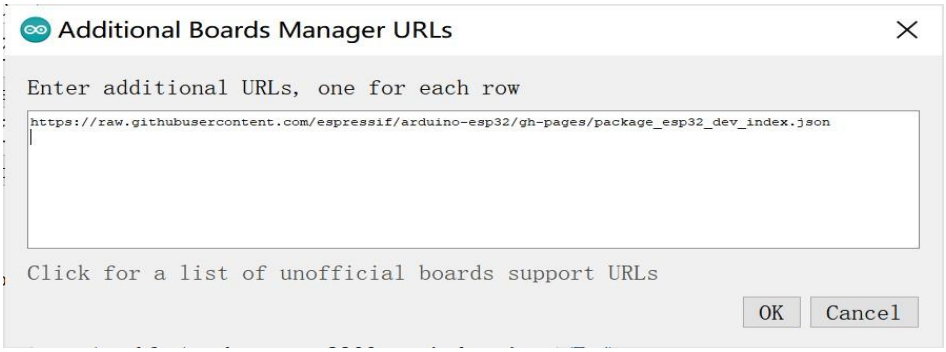
3. Open the Arduino IDE, add the ESP32 development environment.

• Click the ‘File’ ,choose the **Preferences**, to add the ESP32 manager URLs:



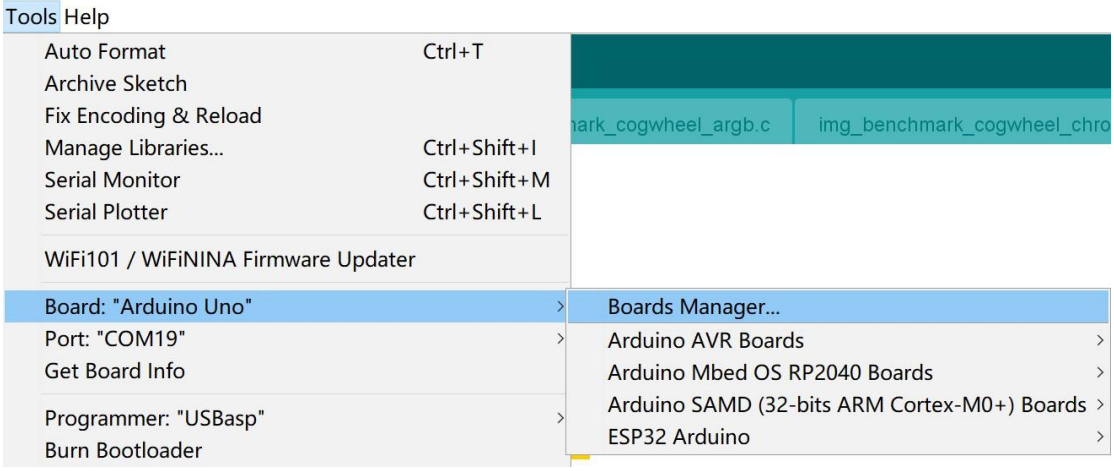
• Click the square box, copy the following URL in it.

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json

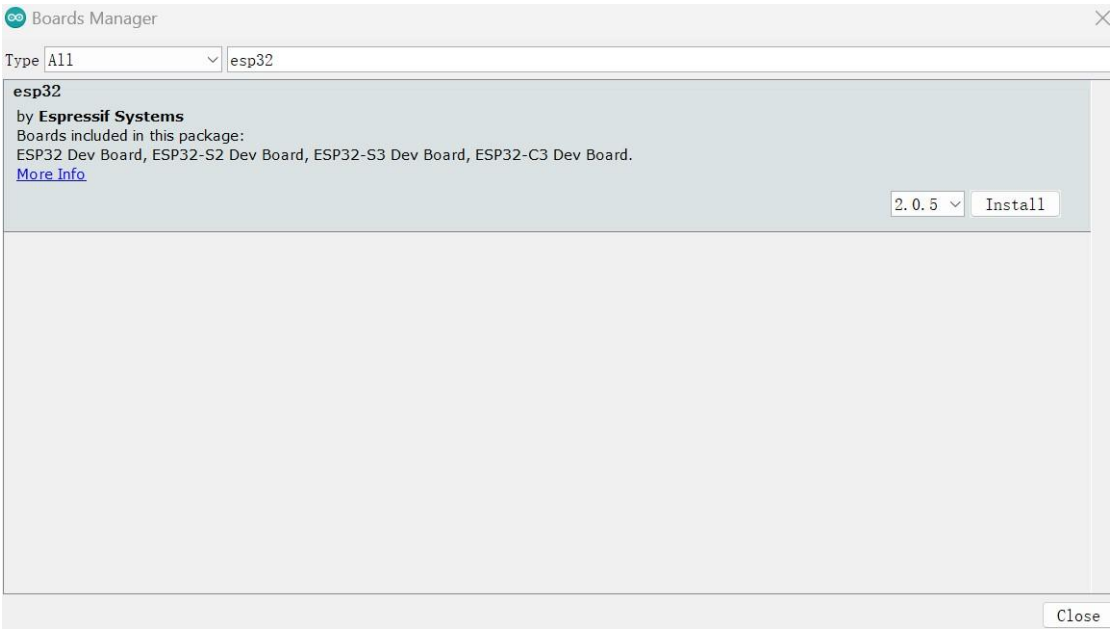


• Click ‘OK’ button to save the configuration.

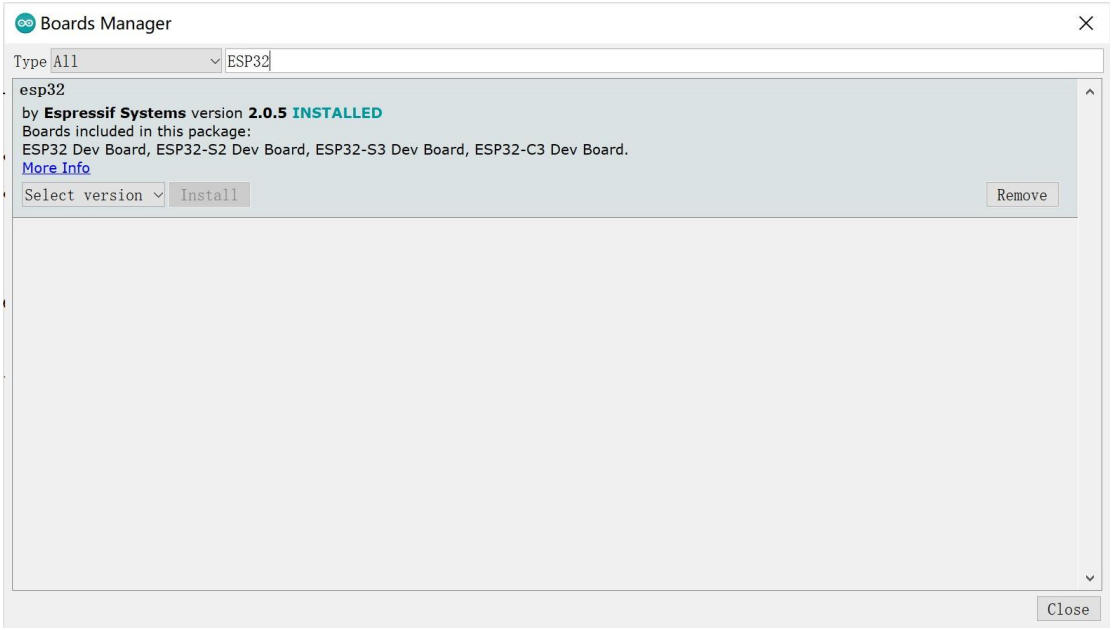
• Open the Boards Manager. Go to Tools > Board > Boards Manager.



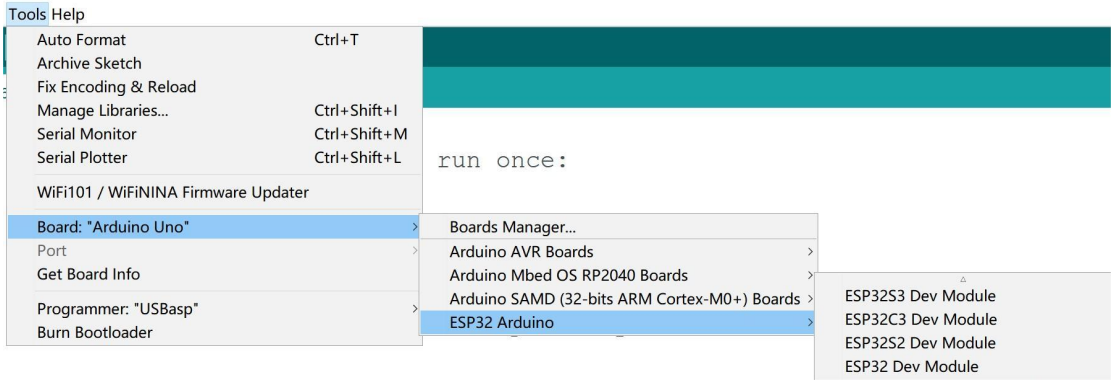
• Search for ESP32 and press install button for the “ESP32 by Espressif Systems“, and choose the **2.0.5** version to click the ‘Install’ .



• If it downloaded successfully, can see the board manager as follows.

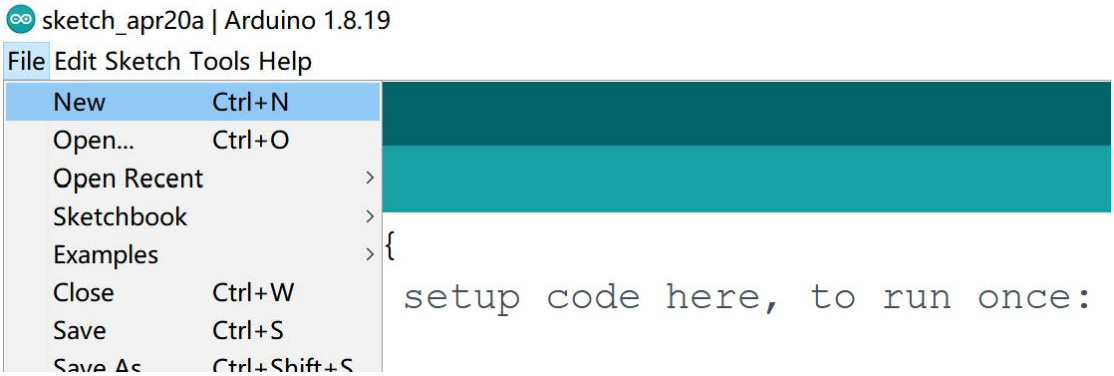


• Users can find the ESP32 board compiler environment in ‘Tools’.



4. Start the first Arduino Program.

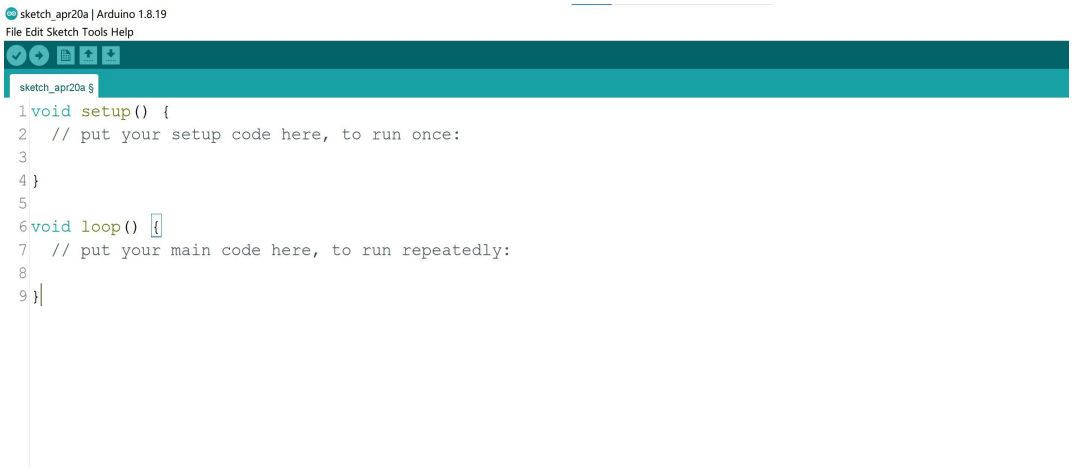
• Click the ‘new’ , the IDE will setup a new sketch in Arduino



• A new sketch display as follows

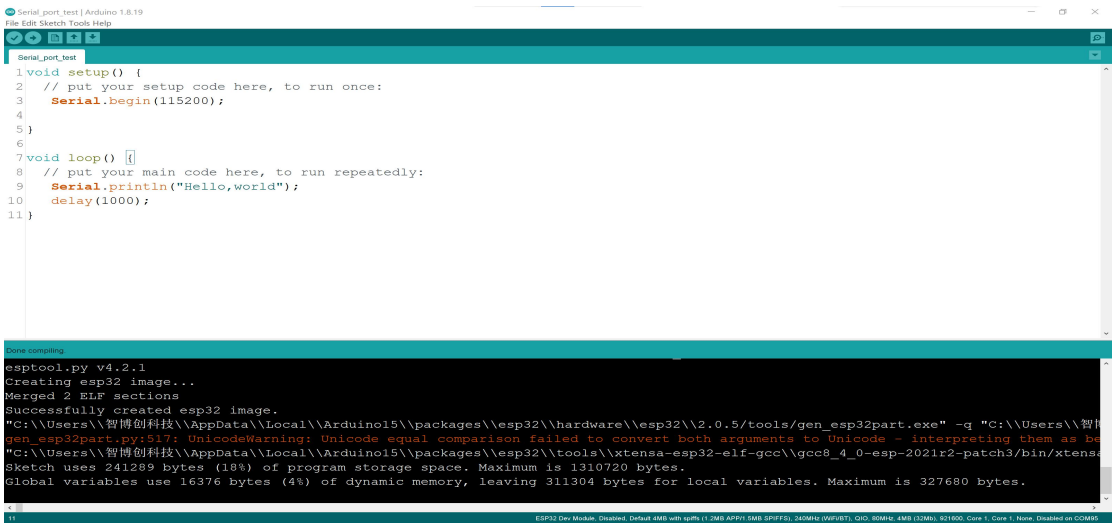
The ‘Void setup()’ function is used to initialize variables, pin modes, libraries, and other settings required for the Arduino to work correctly. This function runs only once when the board is powered on or reset.

The ‘Void loop()’ function is the main function of the Arduino code, which continuously executes the block of code placed inside it. The loop() function runs indefinitely until the board loses power or is reset. The code inside loop() typically reads sensor inputs or performs some other action based on the input received from the user.

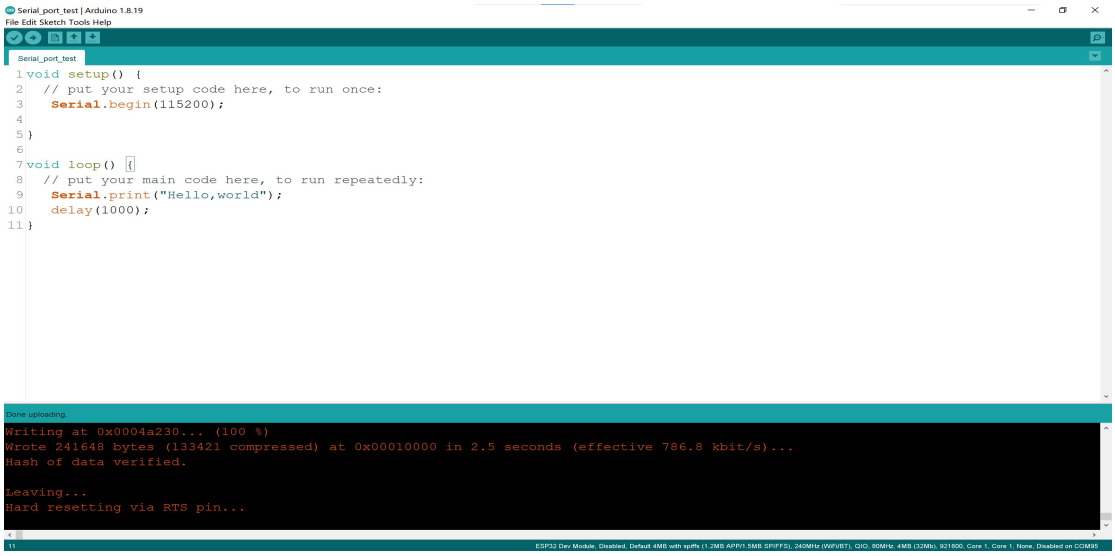


• How to verify and upload the code in MaESP ESP32, for example running a function of serial print the character string.

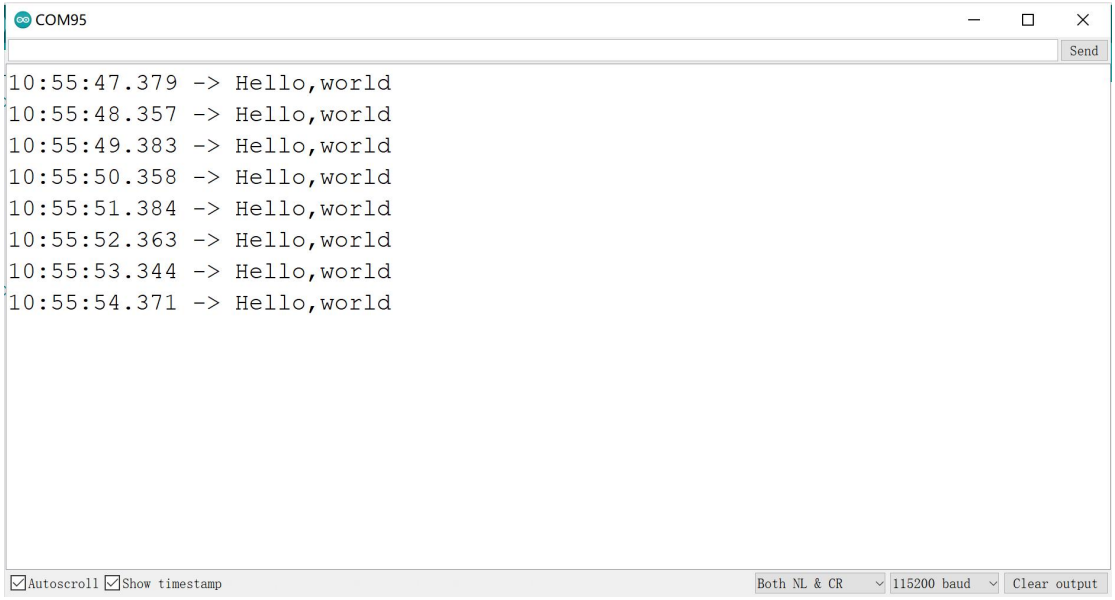
• Click the , the Arduino will verify the code, check the grammar error



• If the code checks well, users can click the upload-button to upload the code to the MaESP ESP32.

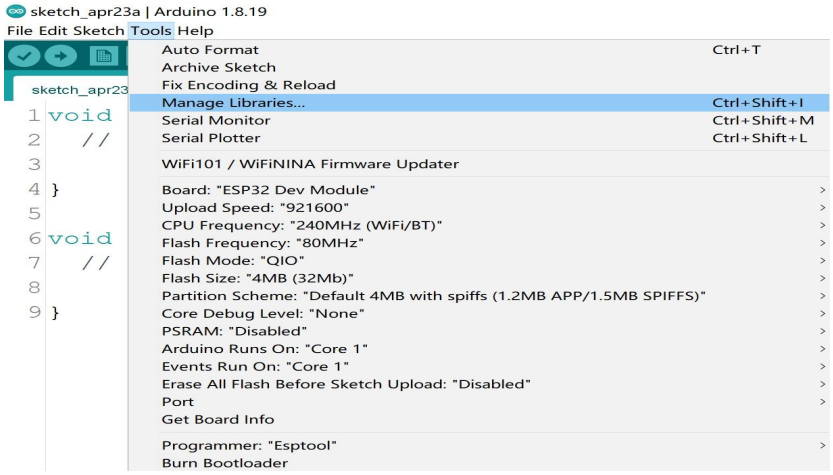


• Choose‘Tools’, open the serial monitor, can see the“Hello world”is printed in it.

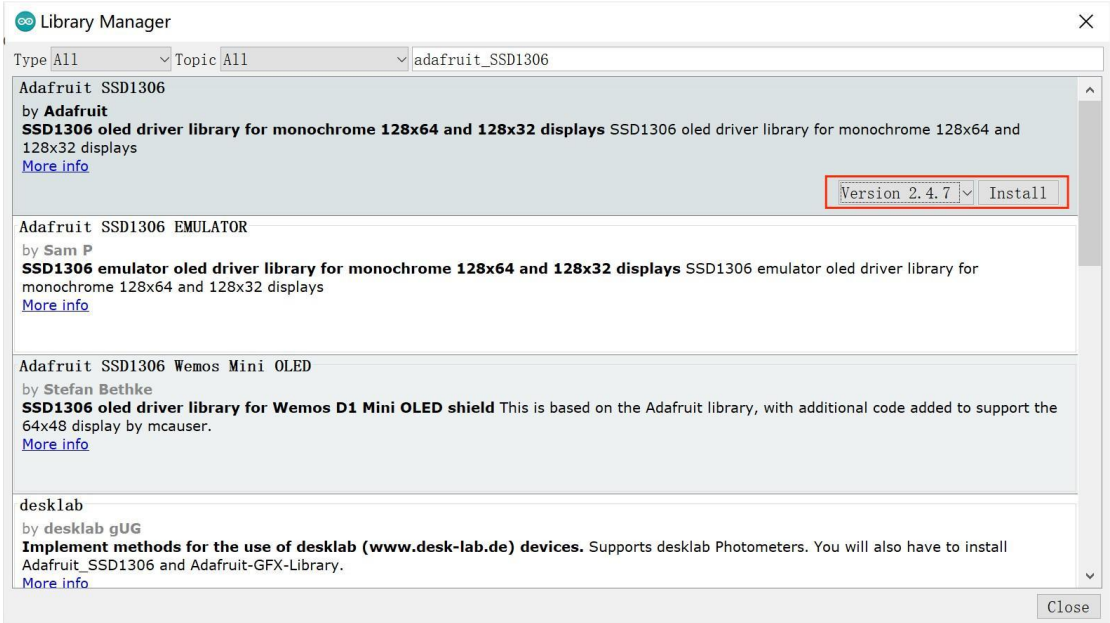


5. How to install Arduino library.

• Open the Arduino IDE and choose the tools in the menus, click the **manage libraris.**



• search the Arduino library and install, for example install the Adafruit_SSD1306.



• Select the library version and install it. When the library is installed, it will be displayed as follows.

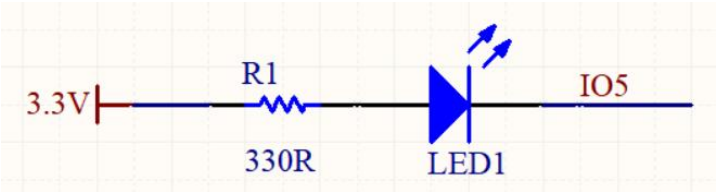
II. MaESP Lessons in Arduino

1. Lesson1: LED Control

This is the most basic lesson, in which we will learn how to control the digital output pins with Arduino, by control the lighting of an LED.

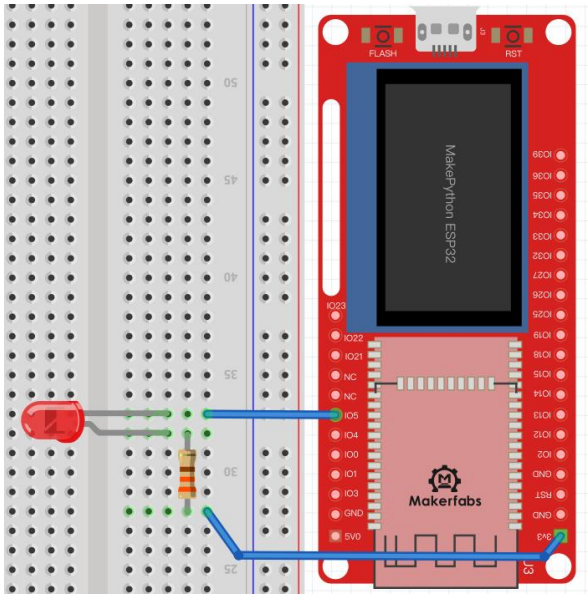
Material:

- 1*LED;
- Resistor: 1*330R;
- Jump wires;



Instructions: The short pin on the left is negative and the long pin on the right is positive.
The purpose of resistor is to limit the current, avoid burning LED lamp with too much current.

Wiring: Connect the long LED pin to the end of the 330R resistor, the other end of the resistance is connected to 3.3v, the short pin of the LED lamp is connected to the IO5 of ESP32. As follows:



Create a file：Open the Arduino IDE and click the “File>>New”, and Put the following code into the new sketch, check the demo code formatting and grammar. Then save the new sketch file in the computer path, and verify and upload the code.

Code and comments:

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(5, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(5, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(500);                                  // wait for a second
  digitalWrite(5, LOW);     // turn the LED off by making the voltage LOW
  delay(500);                                  // wait for a second
}
```

Grammar explanation:

- pinMode(5,OUTPUT);

Create an LED object and set it to output mode

- digitalWrite(5, HIGH);

Set the led pin value to high level. Since the other pin is connected to 3.3v, there is no voltage difference between the two pins.

- digitalWrite(5, LOW);

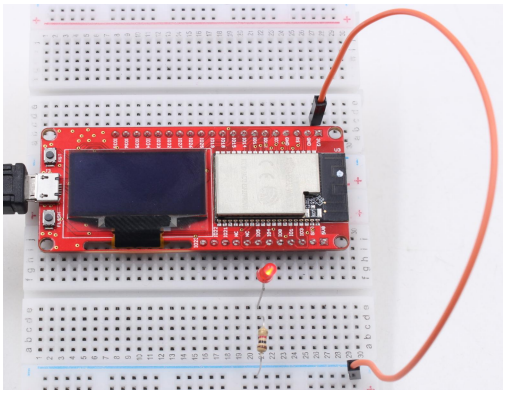
Set the pin value to low level, there is a voltage difference between two pins, there is a current through, the LED light is on.

- d*elay(500);*

Delay 0.5 seconds, in this time the control will be sleep and do nothing.

Results:

When verify and upload the code successfully, you will see the LED lights blinks with interval of 0.5 seconds. You can try changing it to other pins yourself. For example, change to ***pinMode(4,OUTPUT)*** , then connect the wire to IO4, and the light will flash, or the delay is changed to ***delay(1000),***The light flickers slowly, with interval of 1s. You can also try to increase the number of LED lights and keep them on and off.



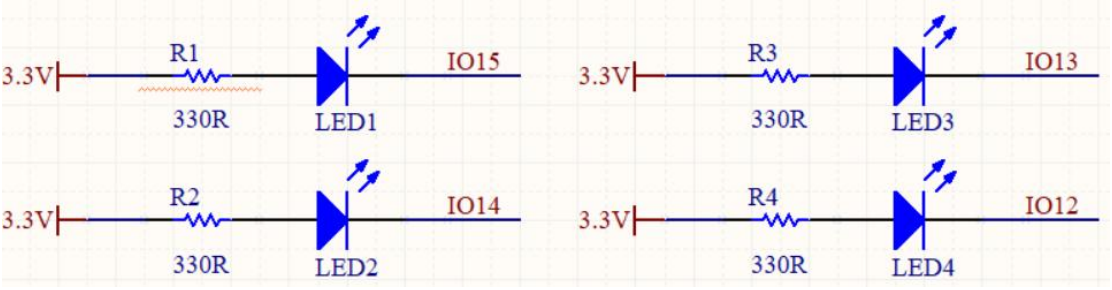
2. Lesson2: Running LED

In this experiment, multiple LED lights are controlled to deepen the understanding of board GPIO port, compilation environment and program framework.

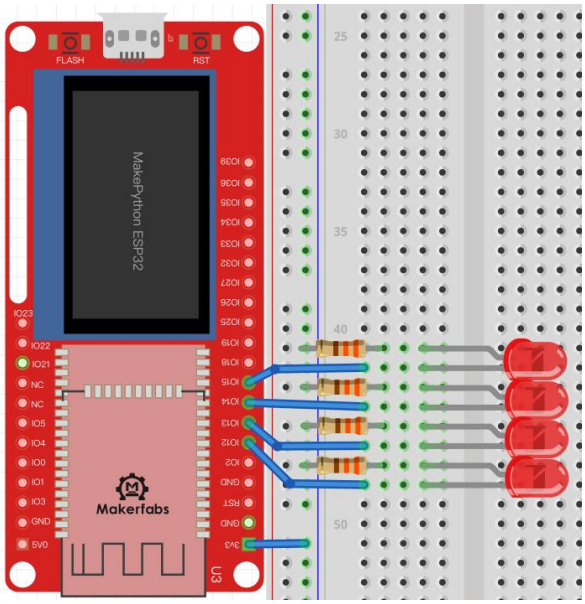
Material:

- 4*LEDs;
- resistance:4*330R;
- Jump wires;

Schematic:



Wiring: Connect the short LED pin to the IO15, IO14, IO13 and IO12 of ESP32, and connect the long pin to one end of the 330R resistor and the other end of the resistor to 3.3V:



Create a new **running_LED** file with the following code and comments:

```
const int ledPins[] = {15, 14, 13, 12}; //Create an array of Pin15,Pin14,Pin13,Pin12 leds
int n = 0;
void setup() {
  for (int i = 0; i < 4; i++) {
    pinMode(ledPins[i], OUTPUT);
  }
```

```
}
void loop() {
  n = (n + 1) % 4;       //The remainder sign % guarantees that n is between 0 and 3
  digitalWrite(ledPins[n], HIGH);    //The value of the NTH LED is high level,turn off
  delay(30);                 //Delay of 30 milliseconds
  digitalWrite(ledPins[n], LOW);     //The value of the NTH LED is low level,turn on
  delay(30);                 //Delay of 30 milliseconds
}
```

Grammar:

- const int ledPins[] = {15, 14, 13, 12};

Create an object for each LED lamp and control them separately, declaring a list form that is easy to understand

- n = (n+1)%4;

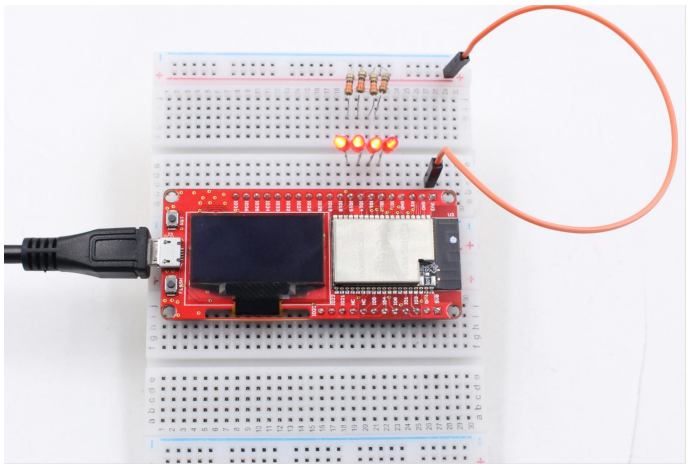
n represents the current LED and we can get the value of the next n after each loop execution (the residual symbol % ensures that the value of n is between 0 and 3)

- digitalWrite(ledPins[n],LOW/HIGH;)

The GPIO input high level can make the LED on, low level can make the LED off.

Results:

After running the code, you can see four LED lights go on and off in cycles. You can appropriately create more LED objects in the list to make the running lights look better. For example, change to const int ledPins[] = {15, 14, 13, 12,10,9},n = (n+1)%6 . You can see a row of lights dancing. You can even add more. Try it yourself.



---

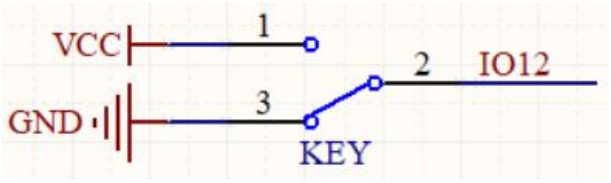
3. Lesson3: Button

The LED light in previous lesson is about the usage of output GPIO port. In the lesson, we will learn how to input the signals to the board, by learning the usage of input of GPIO port by pressing buttons. The MaESP ESP32 board will sense the button input, and thus to control the LED ON/OFF.

Material:

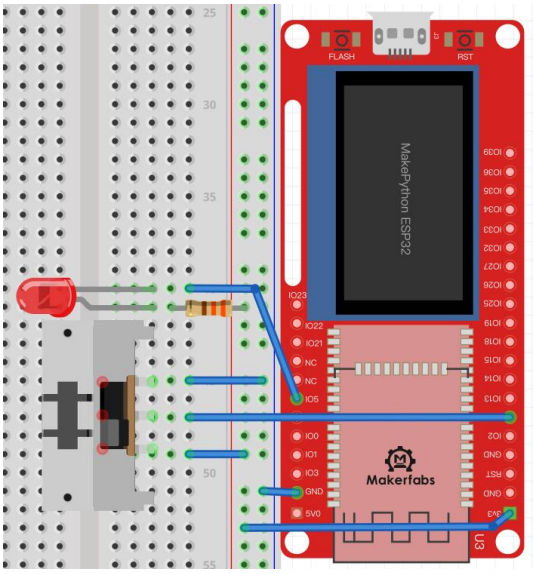
- 1*Button Module;
- 1* LED;

- resistance:1*330R;
- Jump wires;



Instructions: When the button is pressed, VCC is connected to OUT, and OUT pin outputs high level; when the key is released, GND is connected to OUT, and OUT pin outputs low level

Wiring: LED wiring is the same as Lesson1. Button VCC pin connects board 3V3, GND connects board GND, OUT pin connects IO12:



Create a new **Button** file with the following code and comments:

```
const int button = 12;
const int led = 5;
void setup() {
  pinMode(button, INPUT_PULLUP); //Set the key pin to input mode and turn on the internal pull-up resistor
  pinMode(led, OUTPUT); //Set the LED pin to output mode
}
void loop() {
  if (digitalRead(button) == LOW) { //Detects whether a key is pressed?
    digitalWrite(led, HIGH);
  } else {
    digitalWrite(led, LOW);
  }
}
```

Grammar explanation:

- const int button = 12;
- pinMode(button, INPUT_PULLUP);

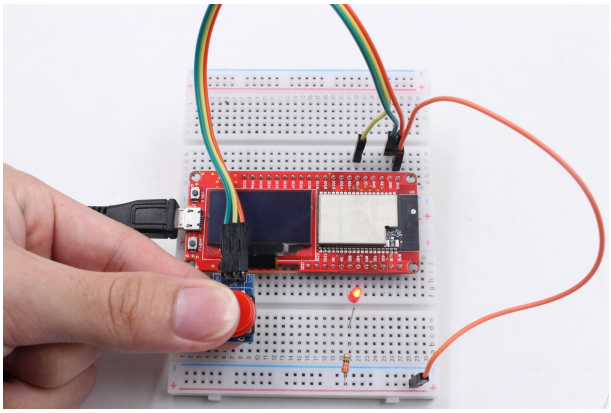
Create a key object and set it to input mode

- if...else:

Statement judge, If true, execute the statement after the if. Otherwise, execute the statement after the else

Results:

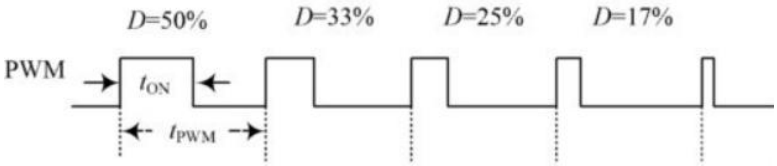
Press the button, the LED light is will be on, while release the button, the LED light is off;



---

4. Lesson4: PWM Control

Pulse width modulation (PWM) is a method of obtaining artificial analog output on digital pins. It does this by quickly switching pins from low to high. There are two related parameters: switching frequency and duty cycle. Duty cycle is defined as the time a pin is at a high level compared to the length of a single cycle (low level to high level time).The maximum duty cycle is that the guide foot is always at high level while the minimum duty cycle is always at low level



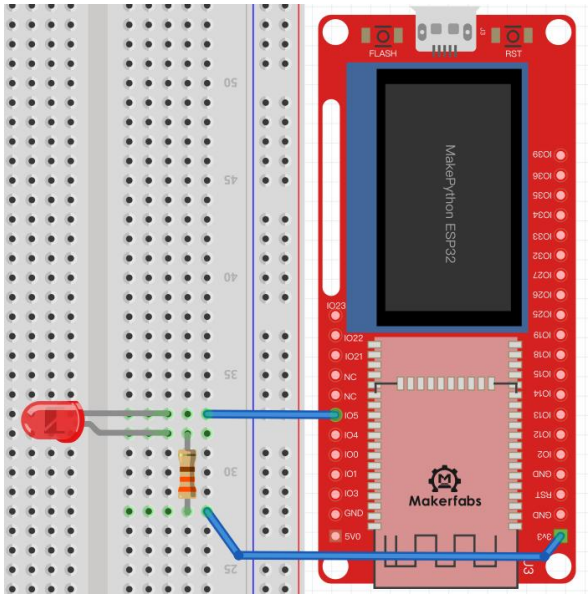
We will demonstrate how to use the ESP32 board to generate a PWM signal. For example, by changing duty cycle to reduce the brightness of LED, control steering gear rotation Angle.

Material:

- 1*Servomotor;
- 1*LED;
- 1*330R;
- Breadboard and jump wires;

Instructions: By adjusting the ratio of LED brightness to off in one cycle, the LED brightness can be adjusted. For example, the control cycle of PWM is 100ms, where 33ms is high level and 67ms is low level, and the duty cycle is 33/100=33%

Wiring: Connect the long LED pin to the end of the 330R resistor, the other end of the resistance is connected to 3.3v, the short pin of the LED lamp is connected to the IO5 of ESP32. As follows:



Create a new **PWM** file with the following code and comments:

```
const int ledPin = 5;       //    set the led pin number
int frequency = 1000;       // set frequency to 1 kHz
void setup() {
   pinMode(ledPin, OUTPUT);       // set the led pin in OUTPUT Mode
}

void loop() {
   for (int dutyCycle = 0; dutyCycle <= 1024; dutyCycle++) {      // dutyCycle cycles between 0 and 1024
      analogWrite(ledPin, dutyCycle);       // change the duty of led
      delay(2);
   }
}
```

Grammar explanation:

- const int ledPin = 5;   //   set the led pin number
- int frequency = 1000;   // set frequency to 5 kHz

Create the PWM object and set the default frequency

- analogWrite():

The larger the duty_cycle value of the led, the brighter the led.

Results:

The LED brightness goes from full bright to dark, then full bright again, and the cycle around, like a breath, which we called “breathing LED”

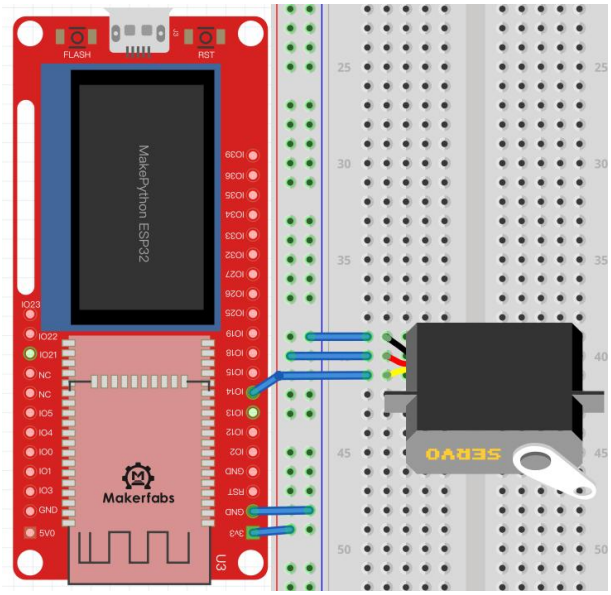
Servo motor is widely used in robot applications. It is an automatic control system composed of dc motor, reduction gear, sensor and control circuit, the rotation angle of the output shaft determined by the input

PWM signal.

Instructions: Orange wire: Signal , Red wire: Power , Brown wire: GND.

Users control the Servo by sending them a fixed frequency square wave signal (usually 50Hz for analog servers, but digital ones may also accept up to 300Hz). the sg90 is analog servo, we use the duty method to set the Angle.

Wiring: The orange line is connected to IO14, the red line is connected to 3.3v, and the brown line is connected to GND：



Create a new **Servo_Demo** file with the following code and comments:

```
#include <Arduino.h>    // Include the standard Arduino library header.

int servoPin = 14;      // Define the signal pin of the servo as digital pin 14.
int frequency = 50;     // Set the control frequency of the servo to 50Hz.

void setup() {              // Initialization function.
   pinMode(servoPin, OUTPUT);    // Set the servo signal pin as output mode.
}

void loop() {               // Looping function.
   for (int angle = 0; angle <= 180; angle++) {    // Increase the angle of the servo gradually from 0 degree to 180 degree.
      int pulseWidth = map(angle, 0, 180, 500, 2500);    // Map the current angle to a corresponding PWM pulse width value.
```

```
    digitalWrite(servoPin, HIGH);              // Turn on the servo signal.
    delayMicroseconds(pulseWidth);             // Wait for the specified pulse width.
    digitalWrite(servoPin, LOW);               // Turn off the servo signal.
    delay(20);                                 // Delay for 20ms to wait for the servo to rotate to the specified
angle.
  }
}
```

Grammar explanation:

- int servoPin = 14;
- int frequency = 50;

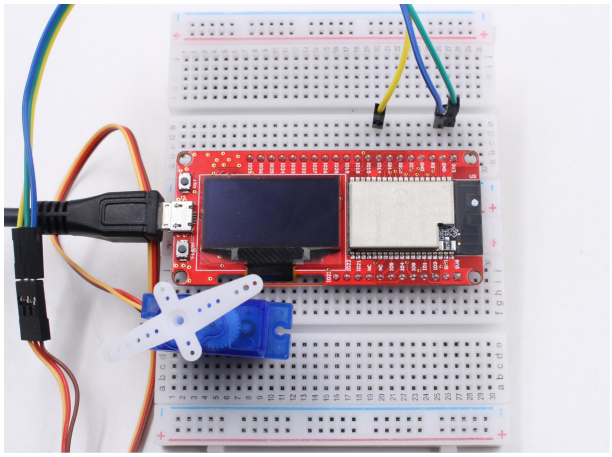
Create the PWM object and set the frequency to 50Hz

- int pulseWidth = map(angle, 0, 180, 500, 2500);:

Map the current angle to a corresponding PWM pulse width value.

Results:

The servo motor keeps turning.



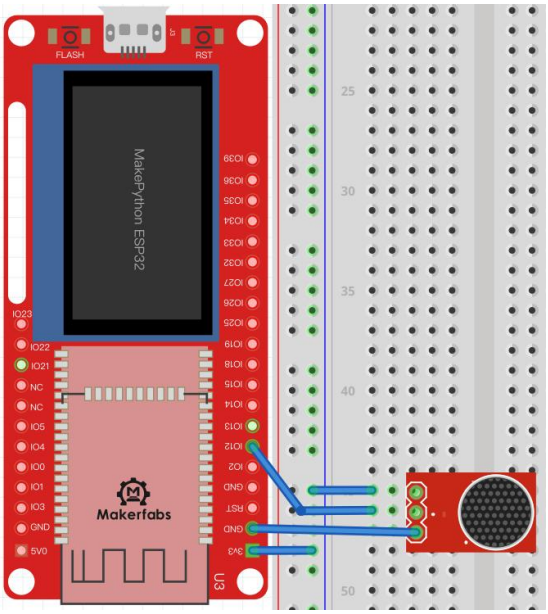
---

5. Lesson5: Voice Control

This experiment will use the buzzer and sound sensor module to do some experiments on sound.

Instructions: according to the driving mode is mainly divided into active buzzer and passive buzzer, active buzzer needs dc voltage can drive, passive buzzer needs a specific frequency of vibration signal to drive.The experiment use an active buzzer.

Wiring： VCC is connected to 3V3,GND is connected to GND, and I/O is connected to IO12:



Create a **Buzzer** file with code and comments:

```
const int pwmPin = 12;// Define the PWM output pin for the siren
int dutyCycle = 0;// Set the initial duty cycle to 0

void ambulenceSiren() {
ledcWriteTone(0, 400); // Set the tone frequency to 400Hz
ledcWrite(0, 512); // Set the duty cycle to 50%
delay(500); // Wait for 500 milliseconds
ledcWriteTone(0, 800); // Increase the tone frequency to 800Hz
delay(500); // Wait for 500 milliseconds

}

void setup() {
ledcSetup(0, 400, 10); // Configure PWM channel 0 with a frequency of 400Hz and a resolution of 10 bits
ledcAttachPin(pwmPin, 0); // Attach the PWM output to the specified pin
}

void loop() {
ambulenceSiren(); // Call the function to generate the siren sound repeatedly in a loop
}
```

Grammar explanation:

- void ambulenceSiren()

Define your own functions in ambulenceSiren, followed by the function identifier name and parentheses (), within which you can pass arguments and arguments.

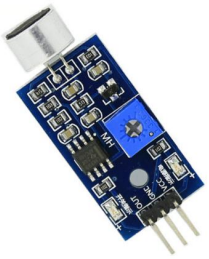
- ledcWriteTone(0, 400);

Set the frequency at 400Hz, different frequencies produce different sounds

Results:

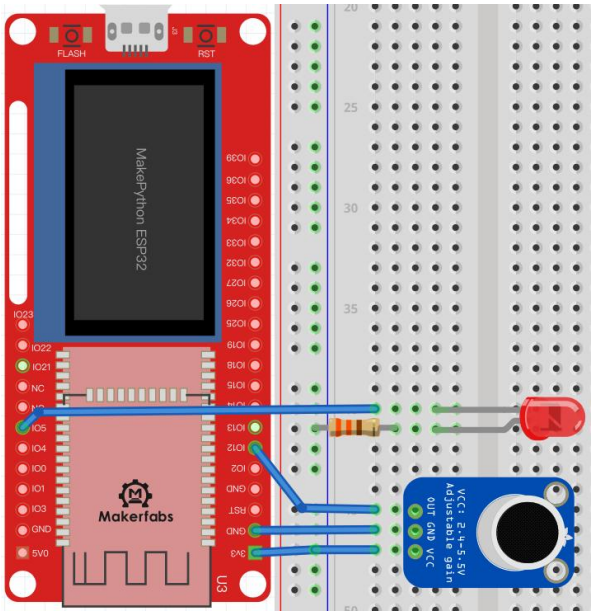
You can hear the sound of an ambulance whistle, but you can also modify **pwm.freq()** to simulate other sounds.

With the sound sensor in the kit, we can also make a “sound switch”, that controls an LED On/Off by single sound:



Instructions: the sound intensity of the surrounding environment can be detected. When the sound of the external environment fails to reach the set threshold, OUT outputs a high level; when the sound intensity of the environment reaches the set threshold, module OUT outputs a low level. The blue digital potentiometer on the module can be used to adjust the sensitivity.

Wiring: VCC is connected to 3V3, GND is grounded, OUT is connected to IO12, and LED is connected to IO5：



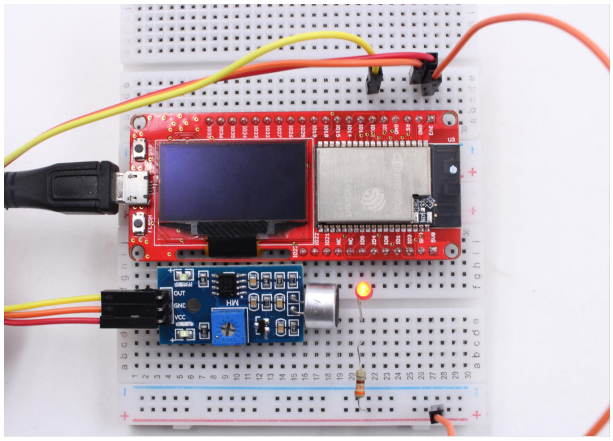
Create **Voice** files, code, and comments:

```
int voice = 12; //Define the 12 pin input for the variable "voice"
int LED = 5;    //Define the 5 pin output for the variable "LED"

void setup() {

pinMode(voice, INPUT);    //Set the "voice" pin as an input.
pinMode(LED, OUTPUT);      //Set the "LED" pin as an output.
```

```
}

void loop() {

        if(digitalRead(voice) == 1){       /*Checks if the digital signal received from the "voice" input pin is equal
to 1. */
          digitalWrite(LED, 0);    //Turns off the LED connected to the "LED" output pin.
          delay(1000);          // Waits for 1 second
    }
    else{      // If the digital signal received from the "voice" input pin is not equal to 1.
   digitalWrite(LED, 1);    // Turns on the LED connected to the "LED" output pin.
 }
}
```

Results:

When you clap your hands or snap your fingers, you'll notice that the light will be on for 1 second.



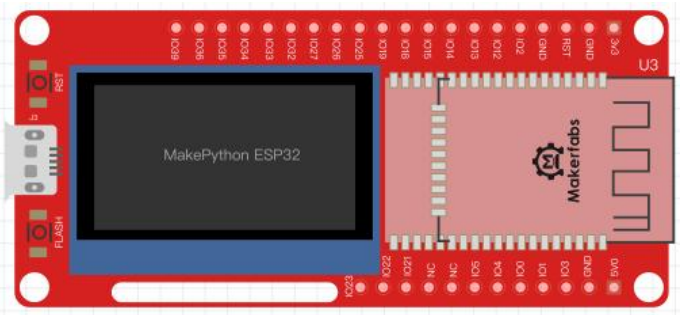
---

6. Lesson6: OLED Display

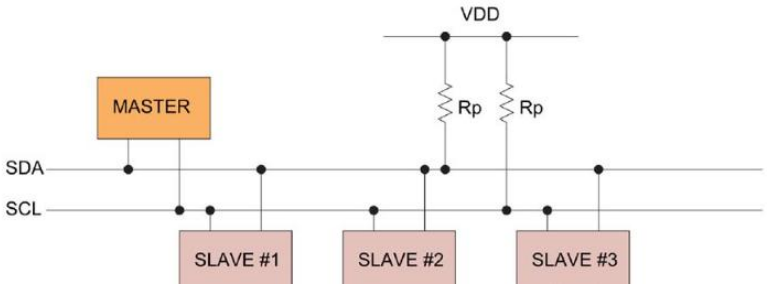
There an on-board OLED 1.3’ OLED module on MaESP board, with 128x64 pixel.One pixel of a monochrome screen is a light-emitting diode. OLED is "self-illumination", the pixel itself is the light source, so the contrast is very high. OLED screens have I2C and SPI communication protocols, which are completely incompatible due to different protocols. In our lesson, the OLED is configured to compatible with I2C protocol.

Material:

- MaESP ESP32:



I2C communication: I2C is widely used for controller communicating with onboard components such as sensors/ displays. Data transmission can be completed by only two signal lines, respectively clock line SCL and signal line SDA. There is only one main device Master and several Slave devices on the I2C line. In order to ensure that both buses are at high level when idle, SDA and SCL must be connected with the pull resistor. The classical value of the pull resistor is 10K.



Use the OLED Drivers.

In this demo，diver the OLED need a Arduino library，**Adafruit _SSD1306.** The beginner need to follow the tutorial(How to install Arduino library) to install the Adafruit_ssd1306 library before verifying the code.

Create a new **ssd1306Demo** file, code and comments:

```
#include <Wire.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)
#define SCREEN_ADDRESS 0x3c
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
#define I2C_SDA 4
#define I2C_SCL 5

void setup() {
  // put your setup code here, to run once:

    Serial.begin(115200);
    Wire.begin(I2C_SDA, I2C_SCL);
```

```
    if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
        Serial.println(F("SSD1306 allocation failed"));
        for (;;); // Don't proceed, loop forever
    }
    display.clearDisplay();
    display.display();
}

void loop() {
    display.clearDisplay();
    display.setCursor(30, 10);
    display.setTextColor(WHITE);
    display.setTextSize(1);
    display.println("Makerfabs");
    display.setCursor(30, 25);
    display.setTextColor(WHITE);
    display.setTextSize(1);
    display.println("time:");
    display.setCursor(30, 40);
    display.setTextColor(WHITE);
    display.setTextSize(1);
    display.println("2019-12-5");

    display.display();

    delay(500);

}
```

Grammar explanation:

- #define I2C_SDA 4
- #define I2C_SCL 5
- Wire.begin(I2C_SDA, I2C_SCL);

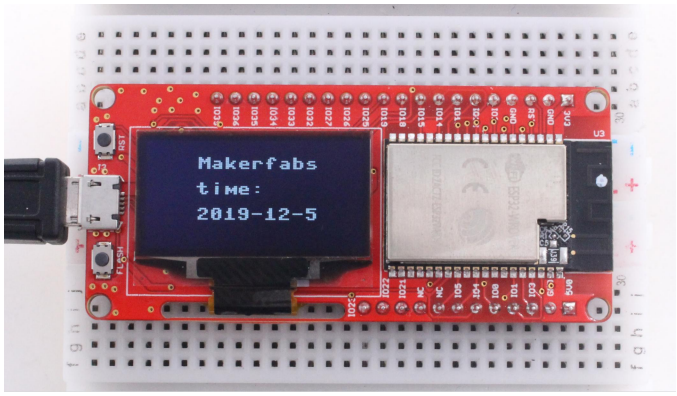
Initialize, configure the SCL and SDA pins

- display.clearDisplay();
- display.setCursor(30, 10);
- display.setTextColor(WHITE);
- display.setTextSize(1);

Something to display,Configure display coordinates, colors ,size and display content.

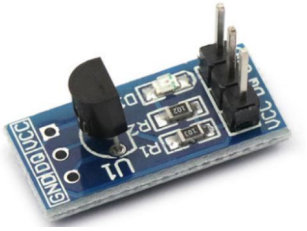
Results:

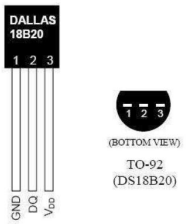
Save and click run, and you'll see Makerfabs/time:/2019-12-5; You can also make it display your own name, etc. Just use the display.println("") statement to write what you want to display.



---

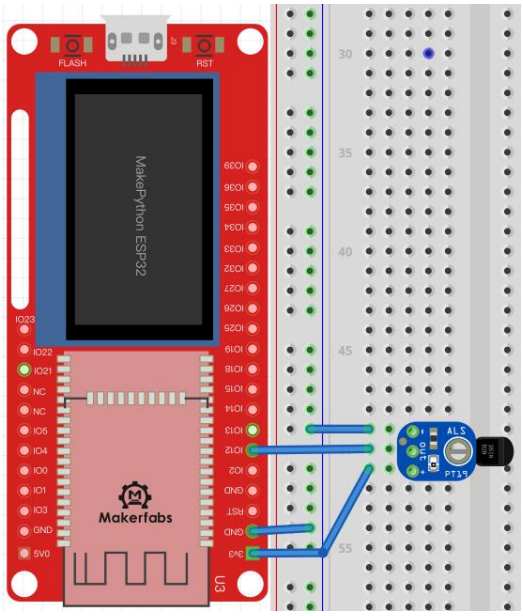
7. Lesson7: Temperature Monitor DS18B20





Instructions: 1:GND; 2:Data(need to connect the pull resistor a 10k resistor to 3.3v);3:VDD

Wiring: The GND pins are connected, VCC pin is connected to 3v3, and the DQ pin is connected to IO12.



**when verify the code, and the shell sent the error is no find the Adafruit_SSD1306.h, please downloaAdafruit_SSD1306 library first. Refer to lesson 6*

Create a new **DS18B20Demo** file, code and comments:

```
#include <Wire.h>      // include the Wire library for I2C communication
#include <Adafruit_SSD1306.h>      // include the Adafruit SSD1306 library for OLED display control
#include <OneWire.h>      // include the OneWire library for DallasTemperature sensor communication
#include <DallasTemperature.h>      // include the DallasTemperature library for temperature measurement

#define SCREEN_WIDTH 128 // define the OLED display width in pixels
#define SCREEN_HEIGHT 64 // define the OLED display height in pixels
#define OLED_RESET -1        // define the OLED reset pin number (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3c // define the OLED screen address
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);    /* initialize the
OLED display*/

#define I2C_SDA 4 // define the I2C SDA pin number
#define I2C_SCL 5 // define the I2C SCL pin number
OneWire oneWire(12); // initialize the OneWire object with data pin 12
DallasTemperature sensors(&oneWire); // initialize the DallasTemperature sensor object with the OneWire object

void setup() {
  Serial.begin (115200); // initialize serial communication at 115200 baud rate
  Wire.begin(I2C_SDA, I2C_SCL); // initialize I2C communication with the defined SDA and SCL pins

  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) { /* check if the OLED display was
successfully initialized*/
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // if not, loop forever
  }
}

void loop() {
  sensors.requestTemperatures(); // send a request to the DallasTemperature sensor to measure temperature
  float temperature = sensors.getTempCByIndex(0); // read and store the temperature in degrees Celsius

  display.clearDisplay(); // clear the OLED display
  display.setTextSize(1); // set the text size to 1
  display.setTextColor(WHITE); // set the text color to white
  display.setCursor(10, 16); // set the cursor position to (10, 16)
  display.println("temperatures:"); // display the text "temperatures:"
  display.setCursor(24, 40); // set the cursor position to (24, 40)
  display.println(String(temperature)); // display the measured temperature as a string
  display.display(); // update the OLED display with the new text
```

```
  delay(1000); // wait for 1000 milliseconds before updating the temperature again
}
```

Grammar explanation:

- sensors.requestTemperatures();

Start temperature reading

- float temperature = sensors.getTempCByIndex(0);

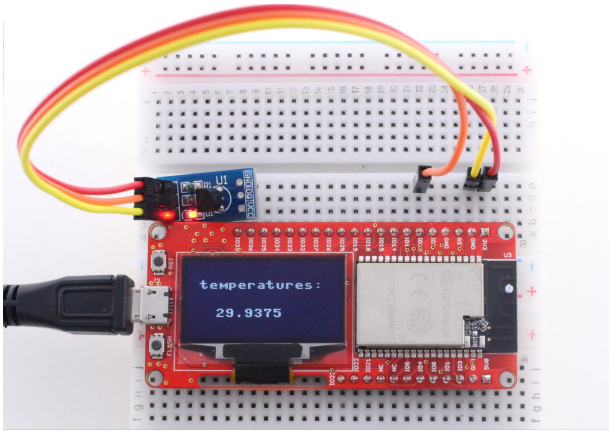
Read the collected temperature

- display.clearDisplay();

Empty the screen.

Results:

After running, the temperature collected on the display screen appears, which is refreshed every 750ms

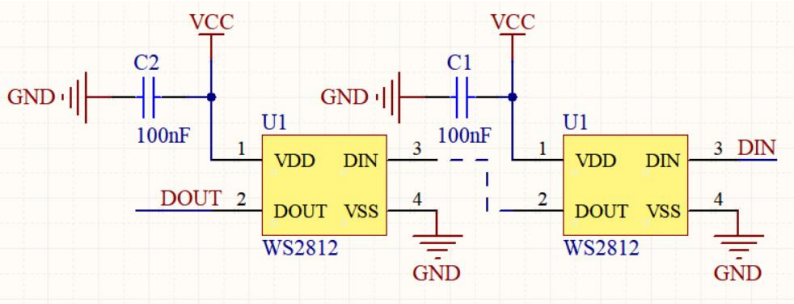


---

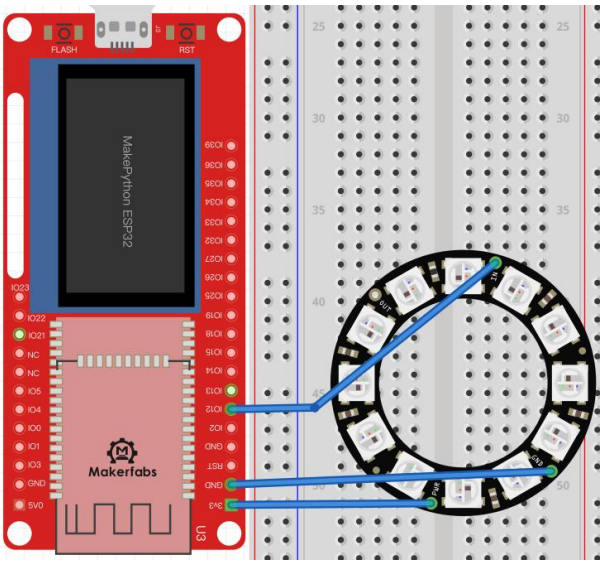
8. Lesson8: Digital LED WS2812

WS2812 is an integrated chip low-power RGB trichromatic lamp, chips and full-color legs wrapped together, only 4 pins, each one chip as a full-color "pixel", each pixel end, DIN client receive the data from the controller transfer to come over, every pixel of transmission, signal 24 bit to reduce, all the way.

This experiment will teach you how to make colorful LED light rings.



Wiring: VDD is connected to 3.3V/5V, VSS is connected to GND, DIN is connected to IO12 of the board:



**when verify the code, and the shell sent the error is no find the Adafruit_NeoPixel.h, please download Adafruit_NeoPixel library first. Refer to How to install Arduino Library.*

Create a new **ws2812Demo** file, code and comments:

```
#include <Adafruit_NeoPixel.h>

// Define the pin number that the LED strip is attached to and the number of LEDs in the strip
#define LED_PIN 12
#define NUM_LEDS 12

// Create an instance of the Adafruit_NeoPixel class with the specified number of LEDs and pin number
Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUM_LEDS, LED_PIN, NEO_GRB + NEO_KHZ800);

// Set up the LED strip by initializing it and showing the initial state (all LEDs off)
void setup() {
        strip.begin();
        strip.show();
}

// Continuously run the demo function in a loop
void loop() {
```

```
            demo();
}

// A custom function to create a circular effect and a rebound effect on the LED strip
void demo() {

// Circular effect: One pixel runs through all strip positions while the others are closed.
for (int i = 0; i < 4 * NUM_LEDS; i++) { // Loop through four times the total number of LEDs
for (int j = 0; j < NUM_LEDS; j++) { // Turn off all LEDs except for the current one being lit
            strip.setPixelColor(j, 0, 0, 0);
}
            strip.setPixelColor(i % NUM_LEDS, 255, 255, 255); // Set the color of the current LED
            strip.show(); // Show the updated LED state
            delay(25); // Wait for a short period of time to create a smooth animation
}

// Rebound effect: The light bounces back and forth across the strip, changing color with each bounce.
// The wait time determines the speed of the bouncing effect.
            for (int i = 0; i < 4 * NUM_LEDS; i++) { // Loop through four times the total number of LEDs
            for (int j = 0; j < NUM_LEDS; j++) { // Set all LEDs to a dark blue color
            strip.setPixelColor(j, 0, 0, 128);
                                    }
   if ((i / NUM_LEDS) % 2 == 0) { // If the current bounce is going right
        strip.setPixelColor(i % NUM_LEDS, 0, 0, 0); // Turn off the current LED
                                    }
        else { // Else, the current bounce is going left
        strip.setPixelColor(NUM_LEDS - 1 - (i % NUM_LEDS), 0, 0, 0); /* Turn off the mirror LED on the other
end of the strip*/
                    }
        strip.show(); // Show the updated LED state
        delay(60); // Wait for a longer period of time to create a slower bouncing effect
      }
}
```

Grammar explanation:

- `#include <Adafruit_NeoPixel.h>`

Include the necessary library for controlling NeoPixel LED strips

- `#define LED_PIN 12`
- `#define NUM_LEDS 12`

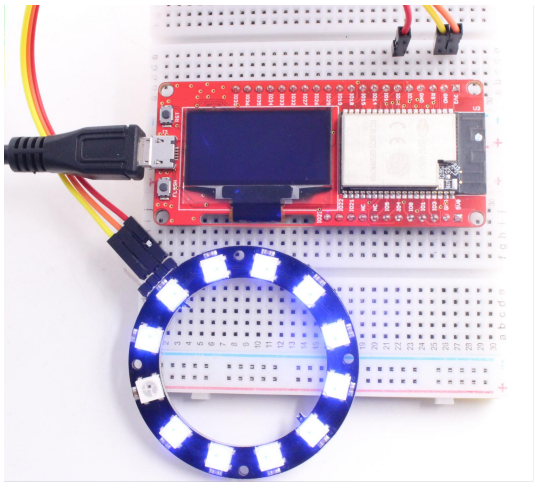
Create the NeoPixe object from pin12,set LED quantity

- `strip.show();`

Use the show() method to output the colors to the LED

Results:

Running the code, you can see the brightest white light in the loop, followed by the half-bright blue light bouncing back and forth. You can make colored lights by changing the **r**,**g**, and **b** values of np[i] to get different colors and brightness.

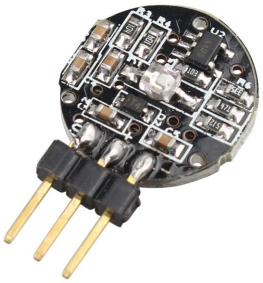


---

9. Lesson9: Pulse Sensor

The Pulse Sensor is a photoelectric reflex analog sensor for pulse and heart rate measurement. Wear it on finger, earlobe and so on, make use of the body tissue in the vascular pulse caused by different transmittance to measure pulse. The sensor filters and amplifies the photoelectric signal, and finally outputs the simulated voltage value. Then the heart rate value can be obtained by simple calculation.

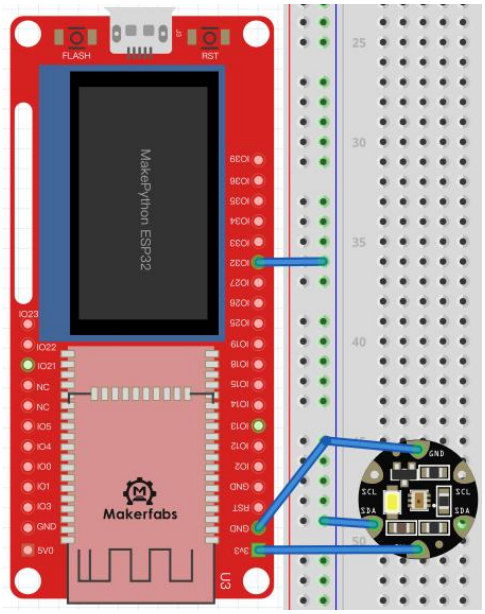
In this experiment, ADC will be learned through the Pulse Sensor module, and the results will be displayed on the OLED screen.



Instructions: On the left is the s-pin of the signal output, in the middle is the positive power supply VCC (3.3v ~5V), and on the right is the negative power supply GND.

On the ESP32 ADC functionality is available on Pins 32-39. Note that, when using the default configuration, input voltages on the ADC pin must be between 0.0v and 1.0v (anything above 1.0v will just read as 4095). Attenuation must be applied in order to increase this usable voltage range.

Wiring: The s-pin is connected to IO32, the middle pin is connected to 3V3, and the "-" pin is connected to GND:



**when verify the code, and the shell sent the error is no find the Adafruit_SSD1306.h, please downloaAdafruit_SSD1306 library first. Refer to lesson 6*

Create a new **pulse** file, code and comments:

```
#include <Wire.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)
#define SCREEN_ADDRESS 0x3c // Communication address for the OLED screen
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
#define I2C_SDA 4 // SDA pin for I2C communication
#define I2C_SCL 5 // SCL pin for I2C communication

const int analogInPin = 32; // Analog input pin for reading sensor data

void drawCircle(int x, int y, int r, int color, int fill=0); // Function to draw a circle on the OLED screen

void setup() {
    Serial.begin(115200); // Begin serial communication
    Wire.begin(I2C_SDA, I2C_SCL); // Begin I2C communication
    if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) { // Initialize the OLED screen
        Serial.println(F("SSD1306 allocation failed")); // If initialization fails, print an error message
        for (;;); // Don't proceed, loop forever
    }
    pinMode(analogInPin, INPUT); // Set analog input pin as input
    display.clearDisplay(); // Clear the OLED screen
```

```
    display.display(); // Display the cleared screen
}

void loop() {
  int adcValue = analogRead(analogInPin); // Read the analog value from the sensor
  float radius = ((float)adcValue / 4095.0) * 64.0; //Map the range of adc values to the range of circle radius (0-32)

  display.clearDisplay(); // Clear the OLED screen
  drawCircle(64, 32, (int)radius, WHITE, 1); // Draw a circle on the OLED screen with specified parameters
  display.display(); // Display the circle on the OLED screen

  delay(500); // Wait for 500 milliseconds
}

void drawCircle(int x, int y, int r, int color, int fill) {
  if (fill == 0) { // If fill parameter is 0, draw only the outline of the circle
    for (int i=x-r; i<=x+r; i++) {
      display.drawPixel(i, y-sqrt(r*r-(x-i)*(x-i)), color); // Draw the upper half of the circle
      display.drawPixel(i, y+sqrt(r*r-(x-i)*(x-i)), color); // Draw the lower half of the circle
    }
    for (int i=y-r; i<=y+r; i++) {
      display.drawPixel(x-sqrt(r*r-(y-i)*(y-i)), i, color); // Draw the left half of the circle
      display.drawPixel(x+sqrt(r*r-(y-i)*(y-i)), i, color); // Draw the right half of the circle
    }
  }
  else { // If fill parameter is 1, fill the circle with color
    for (int i=x-r; i<=x+r; i++) {
      int a = sqrt(r*r-(x-i)*(x-i));
      display.drawFastVLine(i, y-a, a*2, color); // Fill the vertical column of pixels within the circle
    }
    for (int i=y-r; i<=y+r; i++) {
      int a = sqrt(r*r-(y-i)*(y-i));
      display.drawFastHLine(x-a, i, a*2, color); // Fill the horizontal row of pixels within the circle
    }
  }
}
```

Grammar explanation:

- int adcValue = analogRead(analogInPin):

Set a integer variable to store the analog Pin value

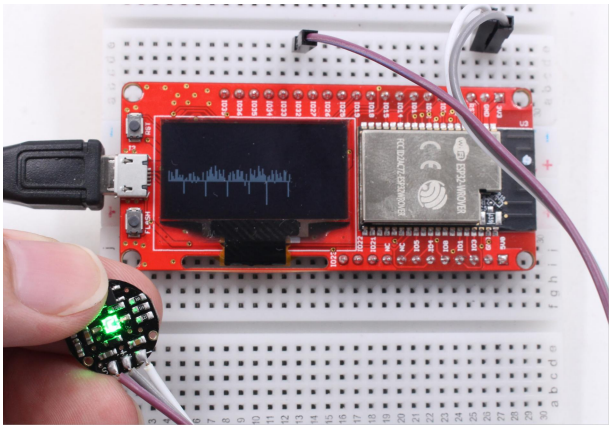
- float radius = ((float)adcValue / 4095.0) * 64.0;

Set a Floating variable to Map the range of adc values to the range of circle radius (0-32)

Results:

Touch the side printed with the heart with your finger, the screen will display your heart rate beat line, the module contact instability or vibration, will affect the measurement data, please save the static

measurement.

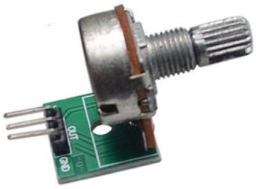


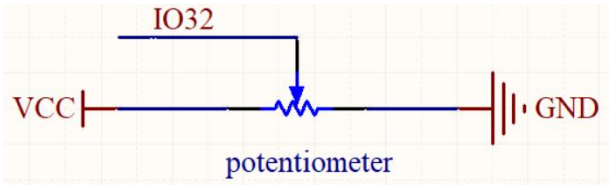
.

---

10. Lesson10: AnalogRead

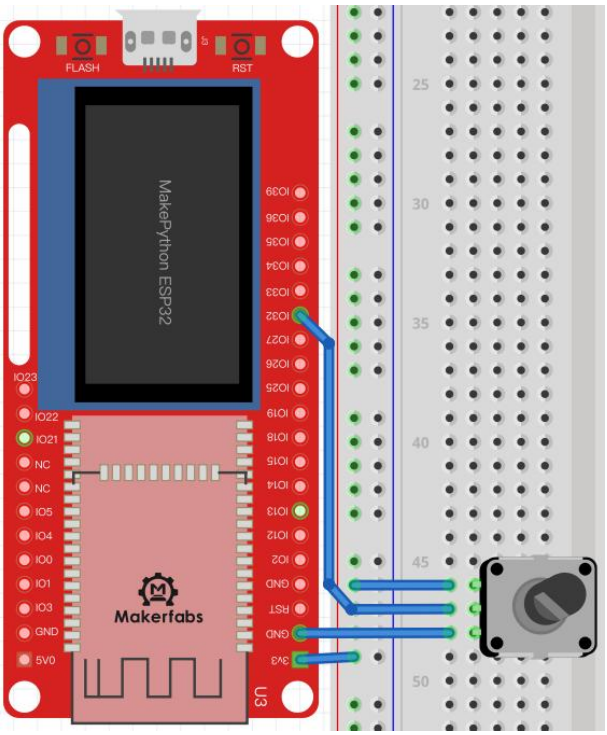
ADC is an Analog/Digital Converter that converts Analog signals into Digital. In the front control LED on, PWM inside, we know the difference between digital signal and analog signal. The signals we use in everyday life, such as light intensity, sound waves, and battery voltages, are all analog values. If we want to measure the analog signal (voltage, light intensity, sound wave) through the single-chip microcomputer and express it by digital signal, then we need ADC analog digital signal converter





Instructions: Potentiometer is an adjustable resistor with three leading ends and resistance values that can be adjusted according to a certain variation law. A potentiometer usually consists of a resistor body and a movable brush. When the brush moves along the resistance body, the resistance value or voltage in relation to the displacement is obtained at the output end.

Wiring : The pins on the left and right sides are connected to 3.3V and GND of the plate respectively, the middle pin is connected to IO32:



**when verify the code, and the shell sent the error is no find the Adafruit_SSD1306.h, please downloaAdafruit_SSD1306 library first. Refer to lesson 6*

Create a new **ssd1306_adc** file, code and comments:

```
#include <Wire.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)
#define SCREEN_ADDRESS 0x3c
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#define I2C_SDA 4
#define I2C_SCL 5

int counttime = 0;

void setup() {
  pinMode(32, INPUT);                    // set pin 32 as input
  Serial.begin (115200);                 // initialize serial communication at 115200 bits per second

  Wire.begin(I2C_SDA, I2C_SCL);      // initialize I2C communication using SDA and SCL pins
```

```
    if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {       // check if OLED display is
connected
    Serial.println(F("SSD1306 allocation failed"));     // print error message to serial monitor
        for(;;); // Don't proceed, loop forever
  }
}

void loop() {
  int adcValue;                          // declare variable to store ADC value
  adcValue = analogRead(32);          // read value from pin 32, 0-4095 across voltage range 0.0v - 1.0v
  adcValue = map(adcValue, 0, 4095, 0, 1023);       // convert ADC value to a 10-bit value
  display.drawLine(counttime, 40, counttime, adcValue-420, WHITE);     // draw the heart rate on OLED display
  display.display();                    // display the heart rate on OLED display
  Serial.println(adcValue);            // print the heart rate to serial monitor
  delay(1);                              // delay for 1 millisecond
  counttime++;                           // increment counttime

  if (counttime > 127) {                // if counttime is greater than 127
    counttime = 0;                       // reset counttime to 0
    display.clearDisplay();            // clear the OLED display
    display.display();                   // display the cleared OLED display
  }
}
```

Grammar explanation:

- adcValue = analogRead(32);

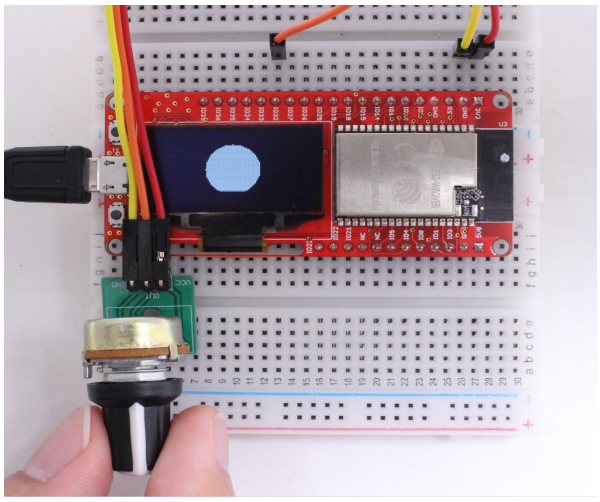
Set a adcValuer variable to store the analog Pin32 value

- adcValue = map(adcValue, 0, 4095, 0, 1023);

The adc sampling bit of esp32 is 12 bits, which is converted to 10-bit data.

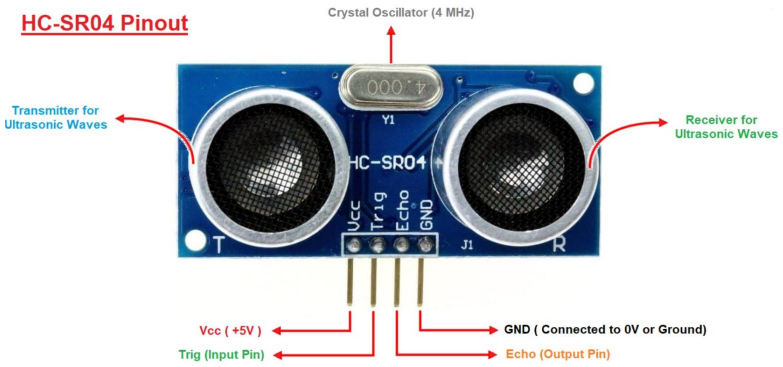
Results:

By rotating the potentiometer, the circle on the OLED display becomes larger or smaller.

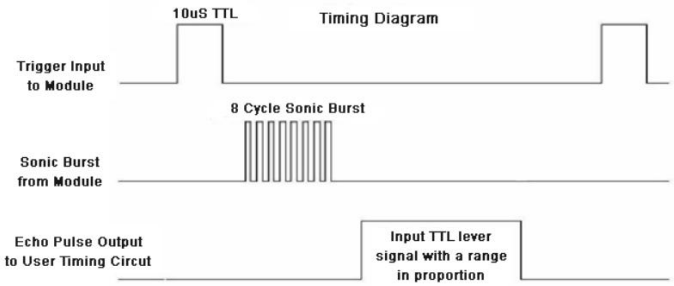


11. Lesson11: Ultrasonic Ranging

The ultrasonic module is for obstacle& distance measurement. It has stable performance, accurate measurement distance and high precision. The module includes ultrasonic transmitter, receiver and control circuit. Application: robot obstacle avoidance, object ranging, etc

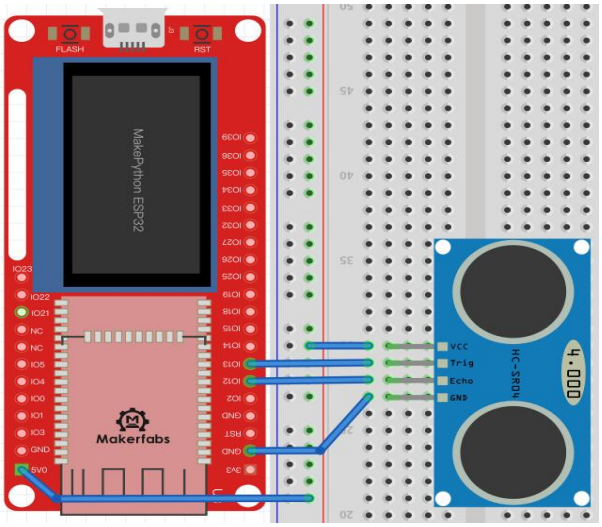


Sequence diagram:



Ranging principle: to module 1 at least 10 us high level, began after launch eight modules receive the high level of 40 KHZ sound waves, echo the feet will be changed from 0 to 1, MCU start timing, when the ultrasonic module receives the returned acoustic echo from 1 to 0, MCU stop timing, this time is the total time range, in a voice the speed of 340 m/SEC, in addition to 2 is distance.

Wiring: VCC is connected to 5V, GND is connected, Trig is connected to IO13, and Echo is connected to IO12:



**when verify the code, and the shell sent the error is no find the Adafruit_SSD1306.h, please downloaAdafruit_SSD1306 library first. Refer to lesson 6*

You can review how this module works. Step 1: you need to give the Trig pin a high level of 10us, step 2: calculate the time when the Echo pin is in the high level, and step 3: calculate the distance according to the time.

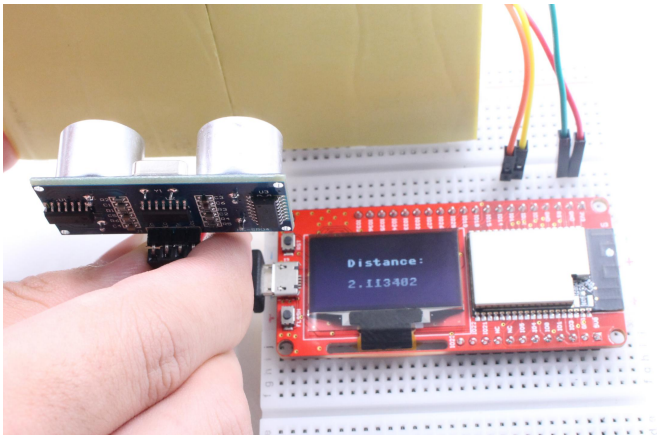
Create a new **Ultrasonic Ranging** file, code and comments:

```
#include <Wire.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)
#define SCREEN_ADDRESS 0x3c // I2C address of the SSD1306 OLED display

// Initialize the SSD1306 OLED display object with the specified screen width, height, I2C bus, and reset pin
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#define I2C_SDA 4 // Pin number for the I2C Serial Data (SDA) line
#define I2C_SCL 5 // Pin number for the I2C Serial Clock (SCL) line
#define TRIGGER_PIN 13 // Pin number for the ultrasonic sensor trigger
#define ECHO_PIN        12 // Pin number for the ultrasonic sensor echo

void setup() {
  Serial.begin(115200); // Initialize serial communication with a baud rate of 115200 bits per second
  Wire.begin(I2C_SDA, I2C_SCL); // Initialize the I2C bus with the specified SDA and SCL pins

  // If the SSD1306 OLED display fails to allocate memory, print an error message to the serial monitor and enter an infinite loop
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  pinMode(TRIGGER_PIN, OUTPUT); // Set the trigger pin as output
  pinMode(ECHO_PIN, INPUT); // Set the echo pin as input

  display.clearDisplay(); // Clear the OLED display
  display.display(); // Update the OLED display with the cleared buffer
}
```

```
void loop() {
  digitalWrite(TRIGGER_PIN, LOW); // Set the ultrasonic sensor trigger pin to low
  delayMicroseconds(2); // Wait for 2 microseconds
  digitalWrite(TRIGGER_PIN, HIGH); // Set the ultrasonic sensor trigger pin to high
  delayMicroseconds(10); // Wait for 10 microseconds
  digitalWrite(TRIGGER_PIN, LOW); // Set the ultrasonic sensor trigger pin to low again

  long duration = pulseIn(ECHO_PIN, HIGH); // Measure the time it takes for the echo signal to return
  float distance_cm = duration / 58; // Convert the duration to centimeters using the speed of sound

  Serial.print("Distance (cm): "); // Print a label to the serial monitor
  Serial.println(distance_cm); // Print the measured distance in centimeters to the serial monitor
  delay(500);
  display.clearDisplay(); // Clear the OLED display
  display.setCursor(30, 20); // Set the cursor position to (30, 20)
  display.setTextColor(WHITE); // Set the text color to white
  display.setTextSize(1); // Set the text size to 1x
  display.println("Distance:"); // Print a label to the OLED display
  display.setCursor(30, 40); // Set the cursor position to (30, 40)
  display.setTextSize(1); // Set the text size to 1x
  display.println(distance_cm); // Print the measured distance in centimeters to the OLED display
  display.display(); // Update the OLED display with the new buffer

  delay(500);
}
```

Results:

Point the module at objects in different directions and the OLED displays the distance.



12. Lesson12: WiFi

MaESP ESP32 devices have built-in Wifi, which enables us to access it via Wifi, or connect to your home Wifi network. In this lesson, users can remote control/upload MaESP, without the need for a USB cable, and also how to make the MaESP a web server, with socket communication.

1. WiFi connection

- **STA Mode:**

Each terminal connected to a wireless network can be called a site. Arduino need to call the WIFI library to setting the the MaESP ESP32 inter STA mode.

- Connect to Internet(STA):

```
#include <WiFi.h>

const char* ssid = "Makerfabs";        // Local network name
const char* password = "20160704"; // Local network password

void setup() {
Serial.begin(115200);        // Initialize serial communication at baud rate 115200
WiFi.mode(WIFI_STA);          // Set WiFi mode to station (client)
WiFi.begin(ssid, password);      // Connect to WiFi network using given SSID and password

while (WiFi.status() != WL_CONNECTED) {      // Wait until connection is established
delay(1000);
Serial.println("Connecting to WiFi...");
}

Serial.println("WiFi connected");                  // Print message once WiFi is connected
Serial.print("IP address: ");

Serial.println(WiFi.localIP());                    // Print the IP address obtained from DHCP server
}

void loop() {
// put your main code here, to run repeatedly:
}
```

Grammar explanation:

- const char* ssid = "Makerfabs";   // Local network name
- const char* password = "20160704"; // Local network password

Put the loacl wifi configration in the code.

- WiFi.mode(WIFI_STA);

Set the WiFi inter the STA mode.

- Serial.println(WiFi.localIP());

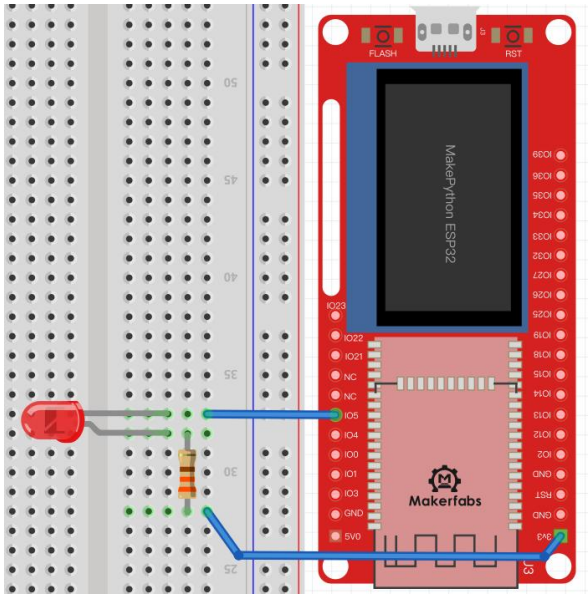
Print the WiFi ip address in the serial monitor.

2. Socket communication

Socket communication is the basis of Internet communication. It is the basic operation unit of network communication that supports TCP/IP protocol. To establish Socket communication requires a server side and a client side. This routine will use Python ESP32 as the client and computer browser as the server side. Both sides will use TCP protocol to transmit and receive data from each other.

Example_1: Socket Communication LED Remote control

In this example, we will make the MaESP as a server with LED, so the users can assess the LED control remotely with any computer, as they get the right IP address.



The **Socket_LED** source code:

```
#include <WiFi.h>                // Library for connecting to WiFi network
#include <WebServer.h>           // Library for creating a web server and handling HTTP requests

// Set up WiFi network credentials
const char* ssid = "Makerfabs";
const char* password = "20160704";

// Create a WebServer object, listening on port 80
WebServer server(80);

// Set LED pin number and initial state
int ledPin = 5;
bool ledState = LOW;
```

```
void setup() {
	Serial.begin(115200);// Start serial communication at 115200 baud rate

	// Set LED pin mode as output and initial state as low
	pinMode(ledPin, OUTPUT);
	digitalWrite(ledPin, ledState);

	// Connect to WiFi network and wait until connected
	WiFi.begin(ssid, password);
	while (WiFi.status() != WL_CONNECTED) {
	delay(1000);
	Serial.println("Connecting to WiFi...");
}

	// Print the IP address once connected to WiFi
	Serial.print("Connected to WiFi, IP address: ");
	Serial.println(WiFi.localIP());

	// Set up root webpage that displays ON/OFF buttons to control LED
	server.on("/", HTTP_GET, {
	String html = "<html><head><meta name="viewport" content="width=device-width,
initial-scale=1"></head>";
	html += "<body><h1>Makerfabs Web Server</h1><a href="/on"><button>ON</button></a> ";
	html += "<a href="/off"><button>OFF</button></a></body></html>";
	server.send(200, "text/html", html);
});

	/ / Handle ON button click by turning LED off and redirecting to root webpage
	server.on("/on", HTTP_GET, {
	ledState = LOW;
	digitalWrite(ledPin, ledState);
	server.sendHeader("Location", "/");
	server.send(303);
});

	// Handle OFF button click by turning LED on and redirecting to root webpage
	server.on("/off", HTTP_GET, {
	ledState = HIGH;
	digitalWrite(ledPin, ledState);
	server.sendHeader("Location", "/");
	server.send(303);
});
```

```
// Start the web server
    server.begin();
}

void loop() {
    // Handle incoming client requests
    server.handleClient();
}
```

Grammar explanation:

- WebServer server(80);

Create a WebServer object, listening on port 80

- server.on("/on", HTTP_GET, [](){):

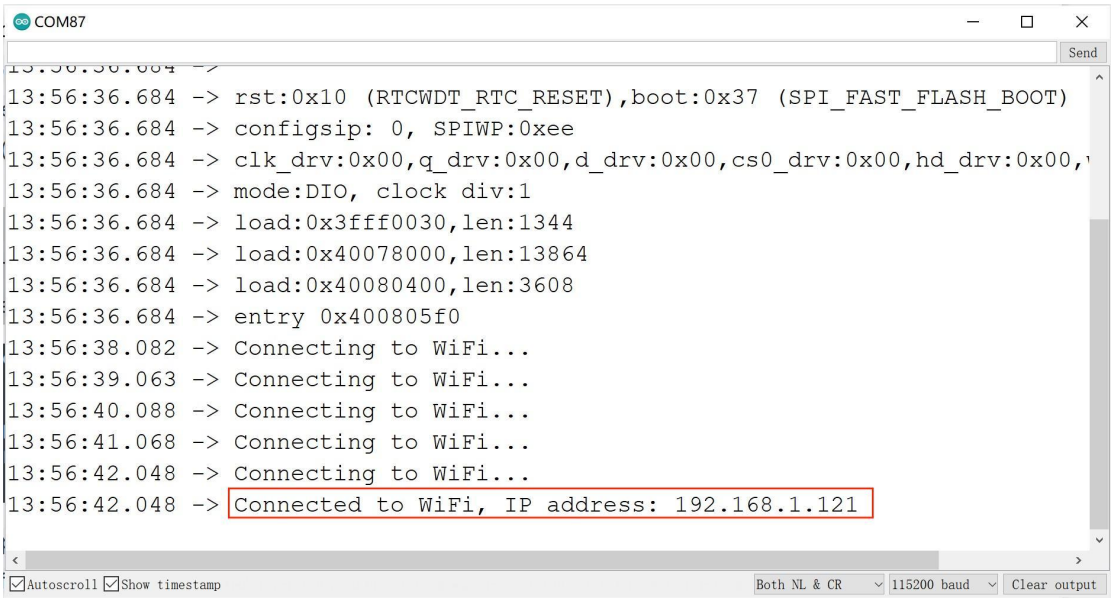
Create the socket object,AF_INET->IPV4,SOCK_STREAM->TCP

- server.begin();

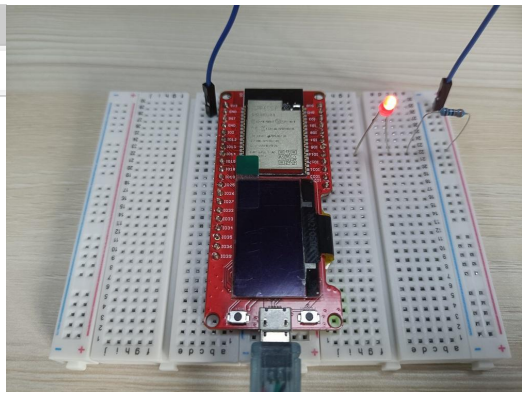
Start the web server

The experimental results:

Change SSID and Password to local network name and Password. The LED wiring is the same as Lesson1. When connecting to the router, you can see the IP address printed out of the serial port:

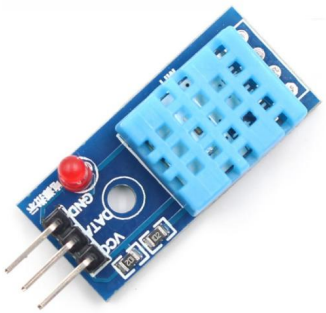


Enter the IP address printed from the serial port in any browser, with any computer/phone in the same WIFI, to control the LED on/off:



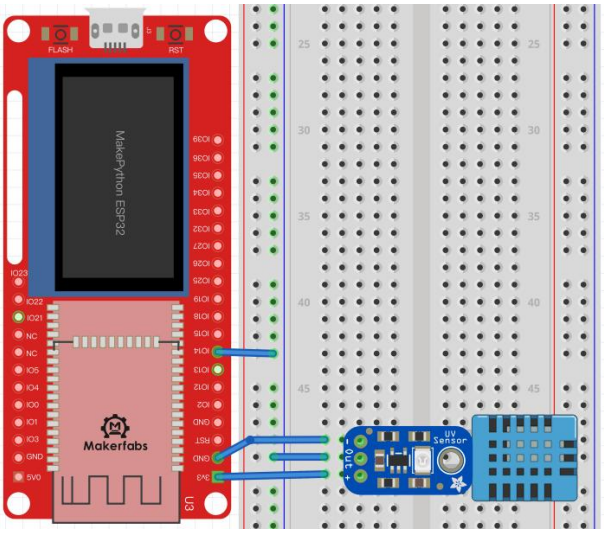
Example_2: Socket Communication Remote Monitoring

In addition to sending command and control LED lights, MaESP ESP32 can also receive measurement data. The DHT11 module is used in this experiment. With the Socket communication, the MaESP can be also a server to detect environment and report to the Internet for sharing. In this example, the MaESP be a temperature& humidity server, so any PC/Phone could assess it to get the related info.



Instructions: DHT11 digital temperature and humidity sensor is sensor containing calibrated digital signal output. It applies special digital module acquisition technology and temperature and humidity sensing technology. The sensor consists of a resistive moisture sensor and an NTC temperature sensor. Its precision humidity +-5%RH, temperature +-2℃, range humidity 20-90%RH, temperature 0~50℃.VCC power positive pole : 3V~ 5.5v,GND: power negative ground, DATA: serial DATA pin;

Wiring: VCC is connected to 3.3v, and DATA is connected to IO14:



Socket_DHT11 Sample code:

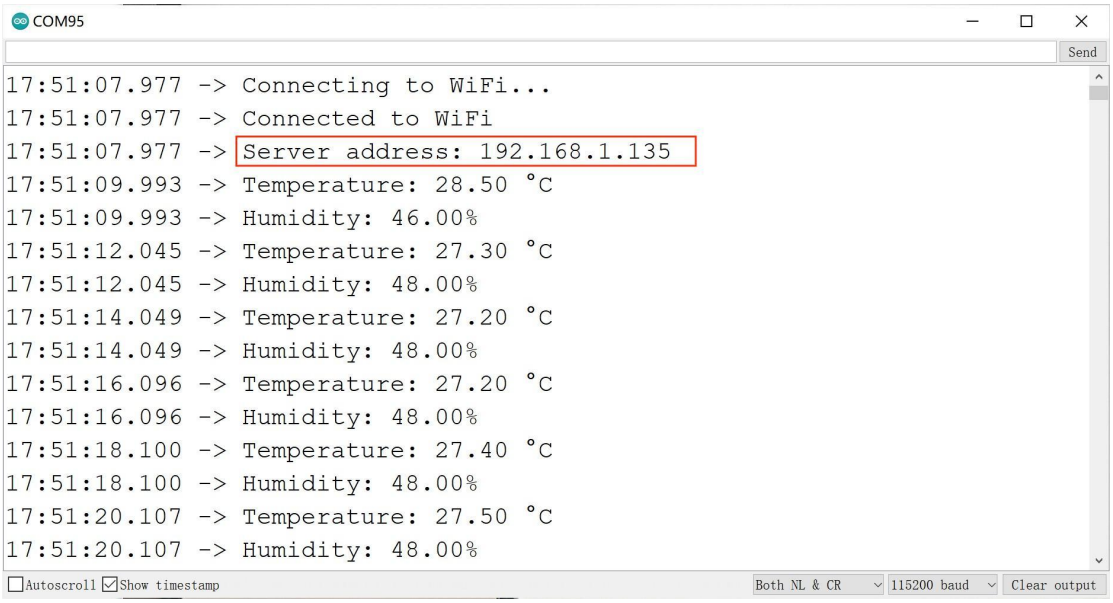
```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include "DHT.h"

const char* ssid = "Makerfabs";
const char* password = "20160704";

#define DHTPIN 14
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

WebServer server(80);

float temp, hum;
String webPage;

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

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
```

```
  Serial.println("Connected to WiFi");// Print a message indicating a successful connection to the WiFi network
// Define the behavior of the root URL ("/") when accessed with an HTTP GET request
  server.on("/", handleRoot);

  server.begin();
  Serial.print("Server address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  // Handle incoming client requests
  server.handleClient();
// Read the temperature and humidity values from the DHT sensor
  readSensor();
}
// Generate an HTML response with two progress bars showing the temperature and humidity readings
void handleRoot() {
  // Generate an HTML response with two progress bars showing the temperature and humidity readings
  String html = "<html><head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">";
  html += "<style>body{padding: 20px; margin: auto; width: 50%; text-align: center;}";
  html += ".progress{background-color: #F5F5F5;} .progress.vertical{position: relative;";
  html += "width: 25%; height: 60%; display: inline-block; margin: 20px;}";
  html += ".progress.vertical > .progress-bar{width: 100% !important;position: absolute;bottom: 0;}";
  html += ".progress-bar{background: linear-gradient(to top, #f5af19 0%, #f12711 100%);}";
  html += ".progress-bar-hum{background: linear-gradient(to top, #9CECFB 0%, #65C7F7 50%, #0052D4
100%);}";
  html += "p{position: absolute; font-size: 1.5rem; top: 50%; left: 50%; transform: translate(-50%, -50%); z-index:
5;}</style></head>";
  html += "<body><h1>Makerfabs DHT Sensor</h1><div class=\"progress vertical\">";
  html += "<p>" + String(temp) + " C<p>";
  html += "<div role=\"progressbar\" style=\"height: " + String((temp+6)/(40+6)*(100)) + "%;\"
class=\"progress-bar\"></div></div><div class=\"progress vertical\">";
  html += "<p>" + String(hum) + "%</p>";
  html += "<div role=\"progressbar\" style=\"height: " + String(hum) + "%;\" class=\"progress-bar
progress-bar-hum\"></div></div></body></html>";

  server.send(200, "text/html", html);// Send the HTML response with a status code of 200 (OK)
}

void readSensor() {
  temp = hum = 0;

  delay(2000);
```

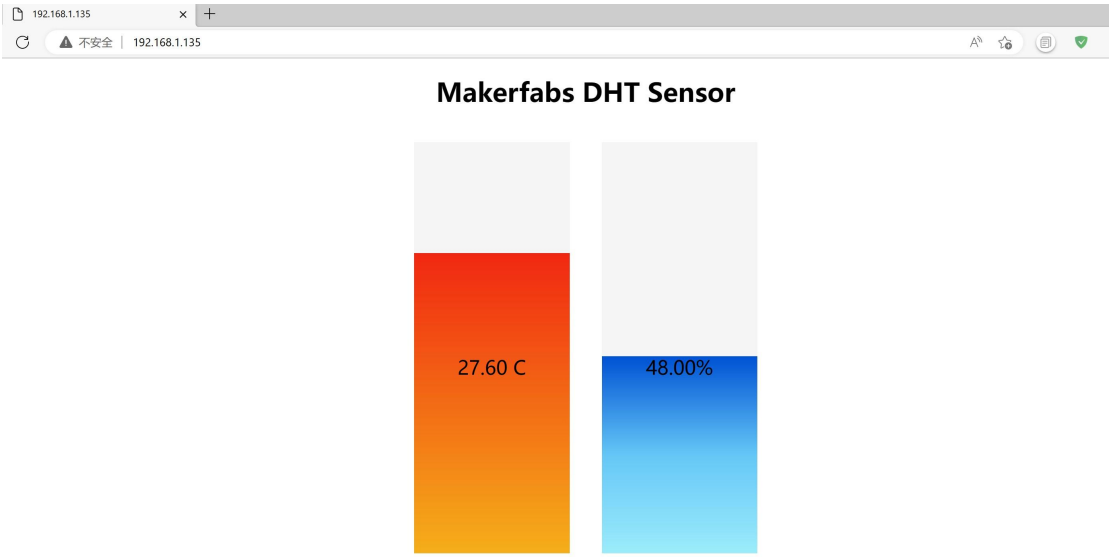
```
  dht.begin();

  float tempRead = dht.readTemperature();
  float humRead = dht.readHumidity();

  if (isnan(tempRead) || isnan(humRead)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
// Update the temperature and humidity variables
  temp = tempRead;
  hum = humRead;
// Print the temperature and humidity readings to the serial monitor
  Serial.println("Temperature: " + String(temp) + " ° C");
  Serial.println("Humidity: " + String(hum) + "%");
}
```

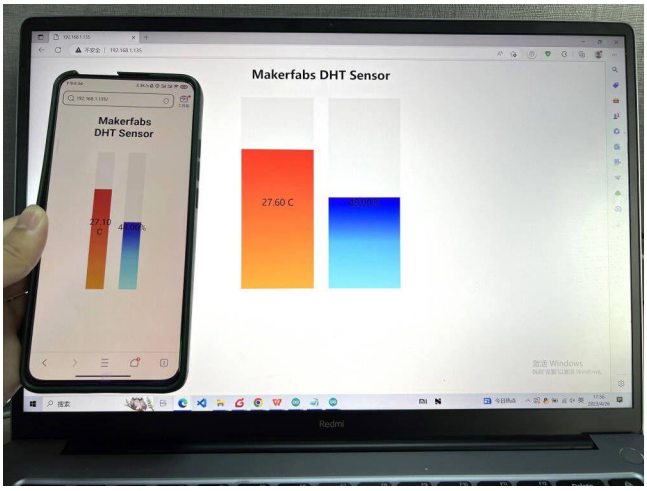
Results:

Modify **SSID** and **PASSWORD** in the code and restart. If the program runs and connects to the router, the serial port can be seen to print out the IP address:



Open your IP with any computer in the WIFI could access this temperature& humidity monitor now, by entering the IP address in any browser:


192.168.1.135
不安全 | 192.168.1.135
Makerfabs DHT Sensor
27.60 C
48.00%


Makerfabs DHT Sensor
Makerfabs
DHT Sensor
27.60 C

III. Troubleshooting

Here are some common troubleshooting issues when developing ESP32 using Arduino:

1.Connection issue: *If you cannot connect to the ESP32 board, you need to check if the USB cable is plugged in properly, if the correct serial port is selected, etc.*

2.Compilation issue*: If errors occur during compilation, you need to make sure that the library files are installed correctly;*

3.Burning issue: *If burning fails, you need to check whether the correct board and UART port are selected;*

4.Power supply issue: *If the ESP32 board does not work properly, it may be due to insufficient or unstable power supply. You can use a higher quality power supply/cable;*

5.Sensor issue: *If there are problems when using sensors, you need to check whether the sensor is connected correctly and authorized, and whether the code for reading data is correct, etc.*

About Makerfabs

Makerfabs is open hardware facilitator based on Shenzhen, China, We make open hardware projects and help our customers project prototyping& small batch production, includes PCBA/ Molding/ OEM. Contact service@makerfabs.com for more info.