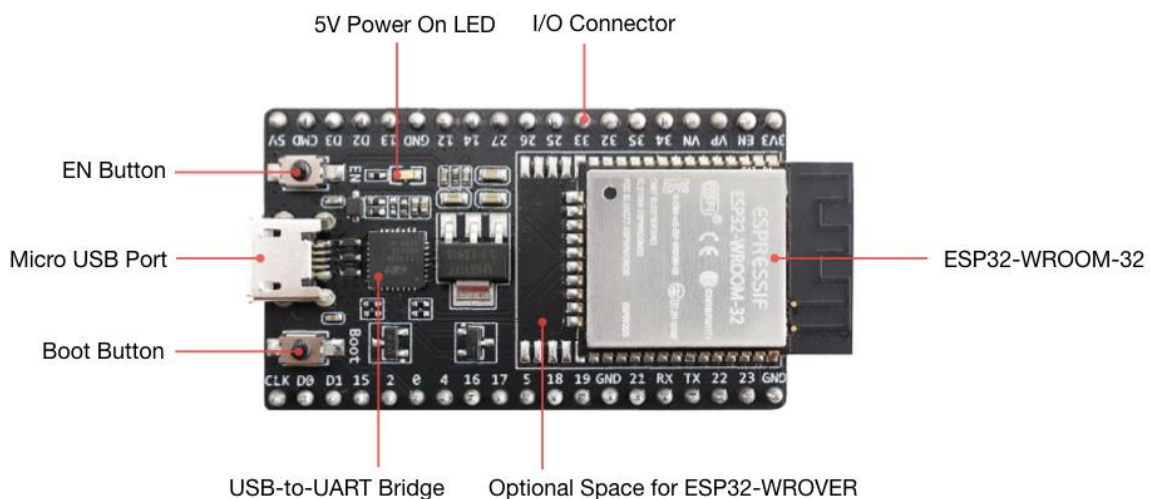


ESP32

ESP32 is a series of low-cost, low-power system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth. At the core of this module is the ESP32-D0WDQ6 chip*. The chip embedded is designed to be scalable and adaptive. There are two CPU cores that can be individually controlled, and the clock frequency is adjustable from 80 MHz to 240 MHz. The user may also power off the CPU and make use of the low-power co-processor to constantly monitor the peripherals for changes or crossing of thresholds. ESP32 integrates a rich set of peripherals, ranging from capacitive touch sensors, Hall sensors, SD card interface, Ethernet, high-speed SPI, UART, I2S and I2C.

- The ESP32 is dual core.
- It has Wi-Fi and Bluetooth built-in.
- It runs 32-bit programs.
- The clock frequency can go up to 240MHz and it has a 512 kB RAM.
- This particular board has 30 or 36 pins, 15 in each row.
- It also has wide variety of peripherals available, like: capacitive touch, ADCs, DACs, UART, SPI, I2C and much more.
- It comes with built-in hall effect sensor and built-in temperature sensor.



Key Component	Description
ESP32-WROOM-32	A module with ESP32 at its core. For more information
EN	Reset button.
Boot	Download button. Holding down Boot and then pressing EN initiates Firmware Download mode for downloading firmware through the serial port.
USB-to-UART Bridge	Single USB-UART bridge chip provides transfer rates of up to 3 Mbps.

Micro USB Port	USB interface. Power supply for the board as well as the communication interface between a computer and the ESP32-WROOM-32 module.
5V Power On LED	Turns on when the USB or an external 5V power supply is connected to the board.
I/O	Most of the pins on the ESP module are broken out to the pin headers on the board. You can program ESP32 to enable multiple functions such as PWM, ADC, DAC, I2C, I2S, SPI, etc.

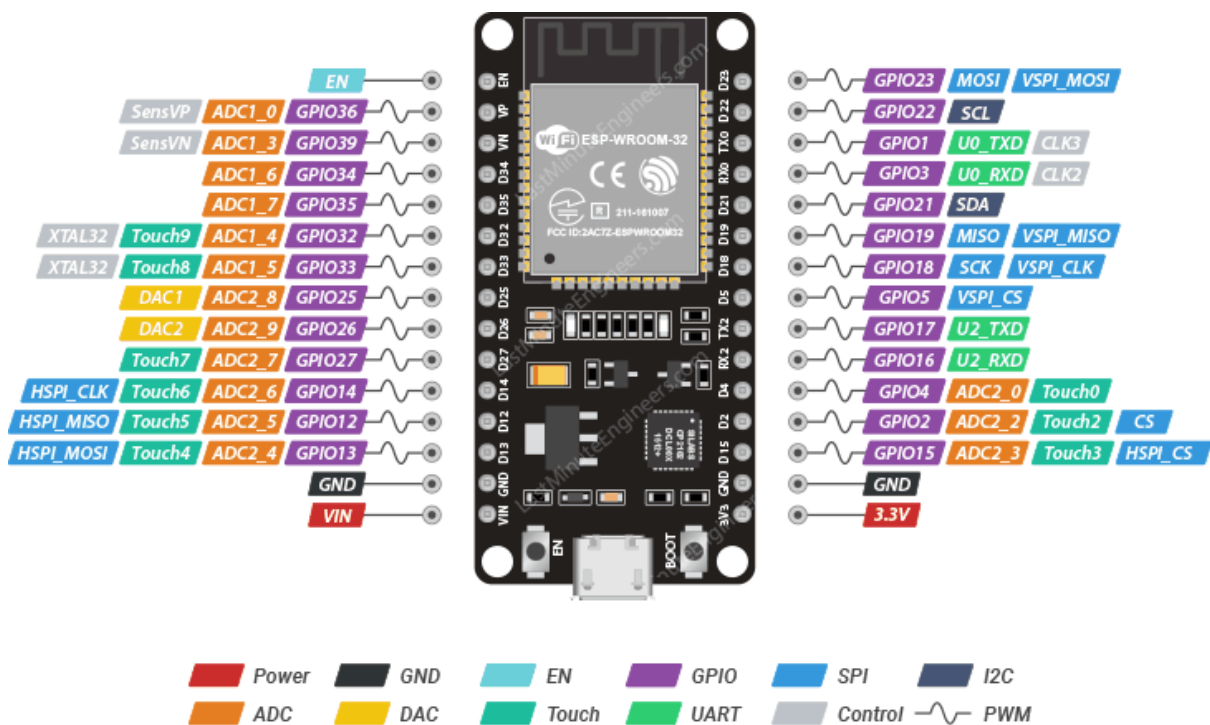
Note: The pins D0, D1, D2, D3, CMD and CLK are used internally for communication between ESP32 and SPI flash memory. They are grouped on both sides near the USB connector. Avoid using these pins, as it may disrupt access to the SPI flash memory / SPI RAM.

Features of the ESP32 include the following:

- **Processors:**
 - CPU: Xtensa dual-core (or single-core) 32-bit LX6 microprocessor, operating at 160 or 240 MHz and performing at up to 600 DMIPS
 - Ultra-low power (ULP) co-processor
- **Memory:** 320 KiB RAM, 448 KiB ROM
- **Wireless connectivity:**
 - Wi-Fi: 802.11 b/g/n
 - Bluetooth: v4.2 BR/EDR and BLE (shares the radio with Wi-Fi)
- **Peripheral interfaces:**
 - 34 × programmable GPIOs
 - 12-bit SAR ADC up to 18 channels
 - 2 × 8-bit DACs
 - 10 × touch sensors (capacitive sensing GPIOs)
 - 4 × SPI
 - 2 × I²S interfaces
 - 2 × I²C interfaces
 - 3 × UART
 - SD/SDIO/CE-ATA/MMC/eMMC host controller
 - SDIO/SPI slave controller
 - Ethernet MAC interface with dedicated DMA and planned IEEE 1588 Precision Time Protocol support[4]
 - CAN bus 2.0
 - Infrared remote controller (TX/RX, up to 8 channels)
 - Motor PWM
 - LED PWM (up to 16 channels)
 - Hall effect sensor
 - Ultra-low power analog pre-amplifier
- **Security:**

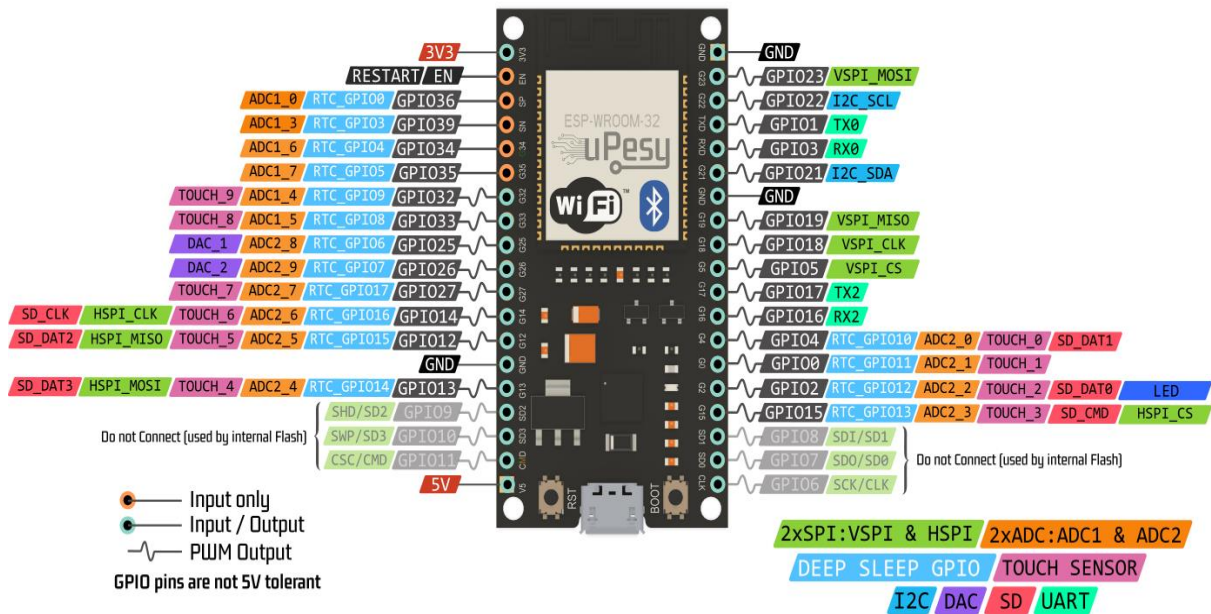
- IEEE 802.11 standard security features all supported, including WPA, WPA2, WPA3 (depending on version)[5] and WLAN Authentication and Privacy Infrastructure (WAPI)
 - Secure boot
 - Flash encryption
 - 1024-bit OTP, up to 768-bit for customers
 - Cryptographic hardware acceleration: AES, SHA-2, RSA, elliptic curve cryptography (ECC), random number generator (RNG)
- Power management:**
 - Internal low-dropout regulator
 - Individual power domain for RTC
 - 5 μ A deep sleep current
 - Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

Pin Layout:

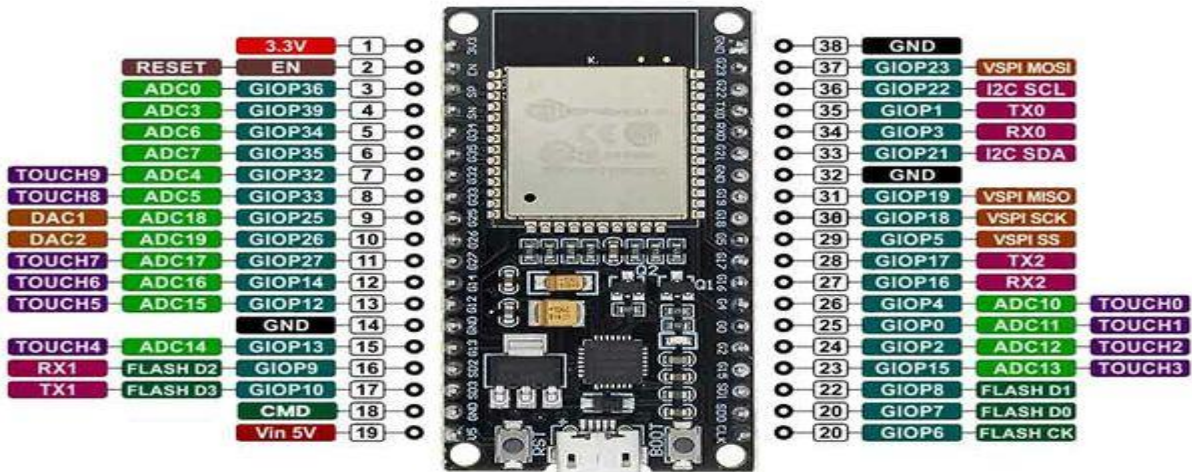


ESP32 Dev. Board Pinout

ESP32 Wroom DevKit Full Pinout



PINOUT ESP32 38 PINES ESP WROOM 32



For more details : <https://www.upesy.com/blogs/tutorials/esp32-pinout-reference-gpio-pins-ultimate-guide>

https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf

Install the Arduino Desktop IDE:

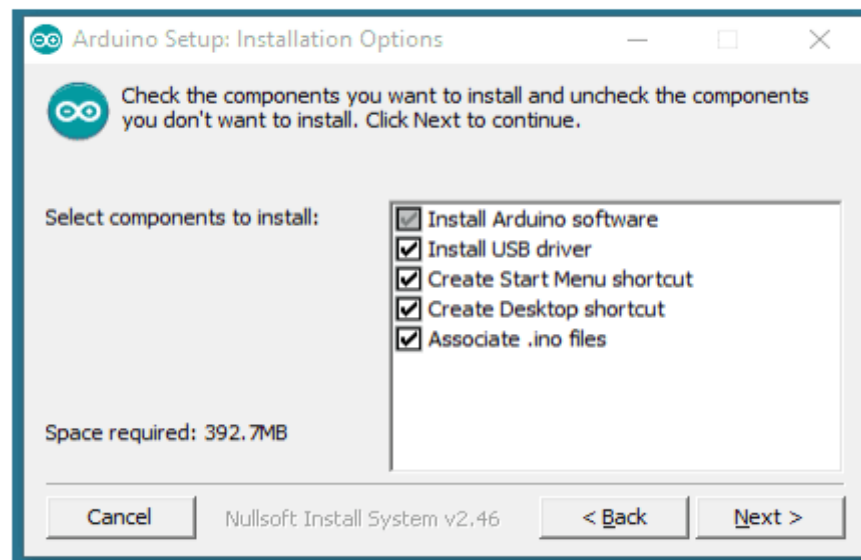
Arduino IDE Installation (Windows):

Step 1: Download the Arduino Software (IDE):

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

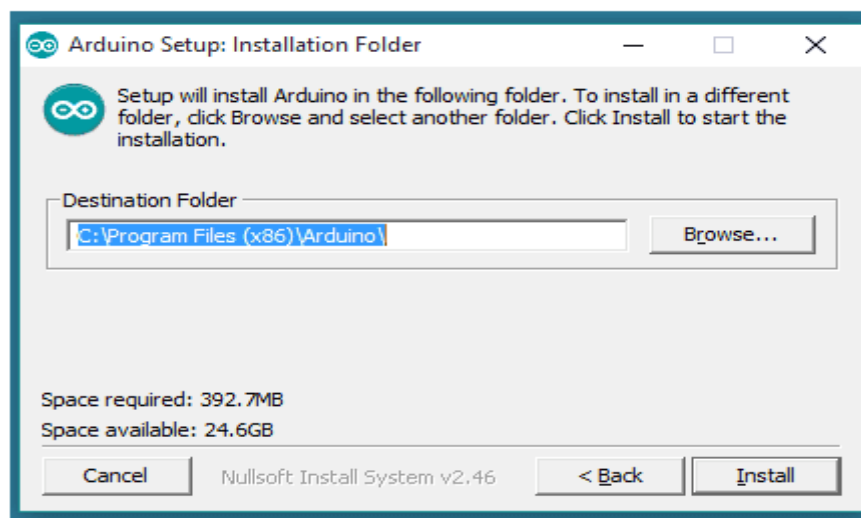
Step 2:

When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system.



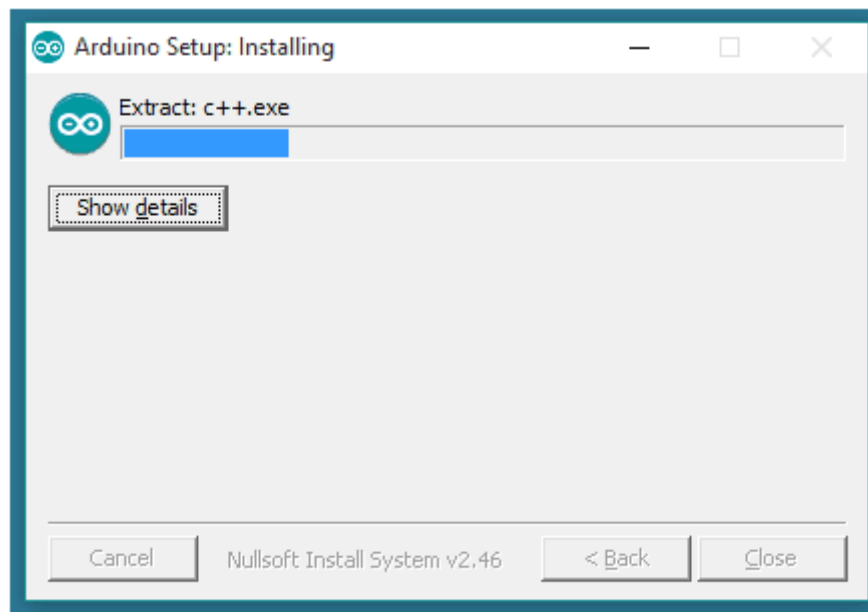
Choose the components to install.

Step 3:



Choose the installation directory.

Step 4:



Installation in progress.

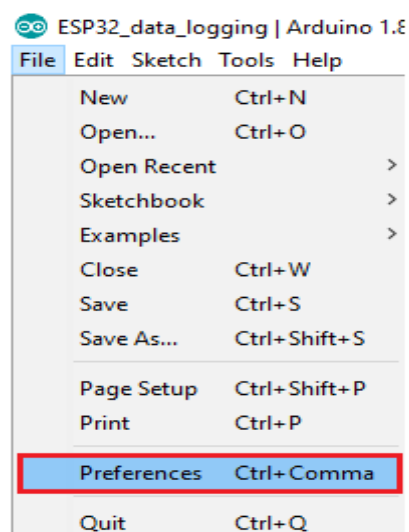
The process will extract and install all the required files to execute properly the Arduino Software (IDE)

Installing the ESP32 Board in Arduino IDE (Windows, Mac OS X, Linux)

There's an add-on for the Arduino IDE that allows you to program the ESP32 using the Arduino IDE and its programming language.

To install the ESP32 board in your Arduino IDE, follow these Steps:

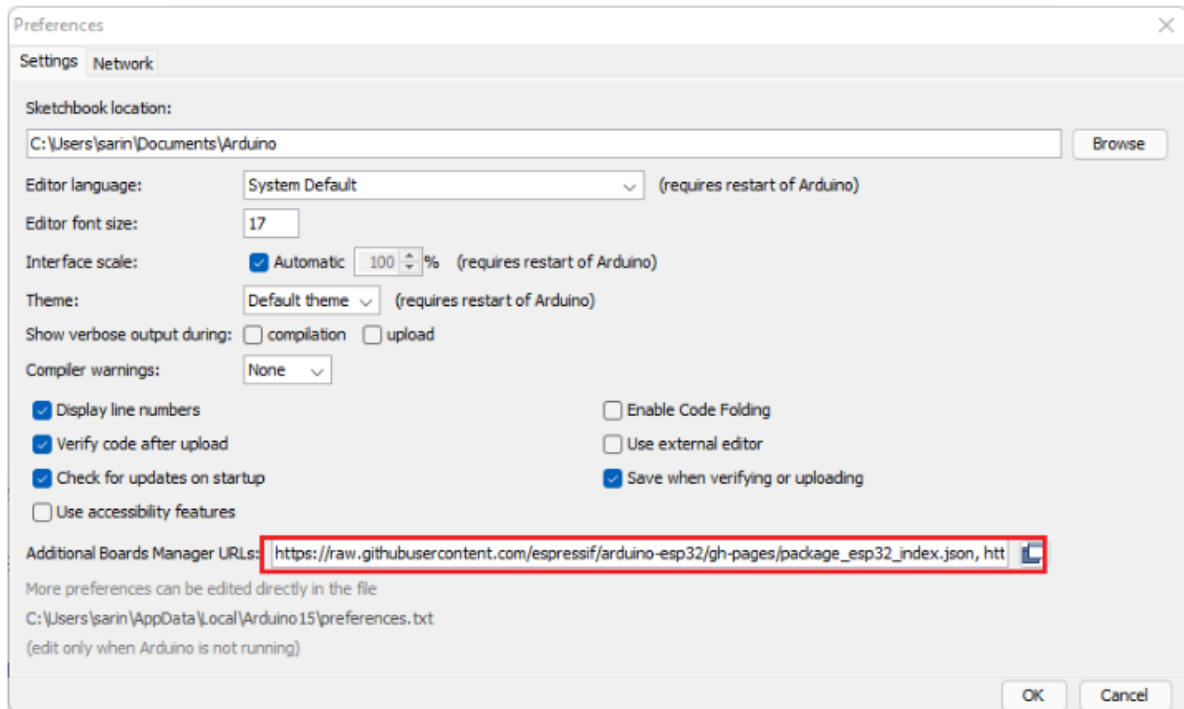
Step 1: In your Arduino IDE, go to File> Preferences



Step 2: Enter the following into the “Additional Board Manager URLs” field:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

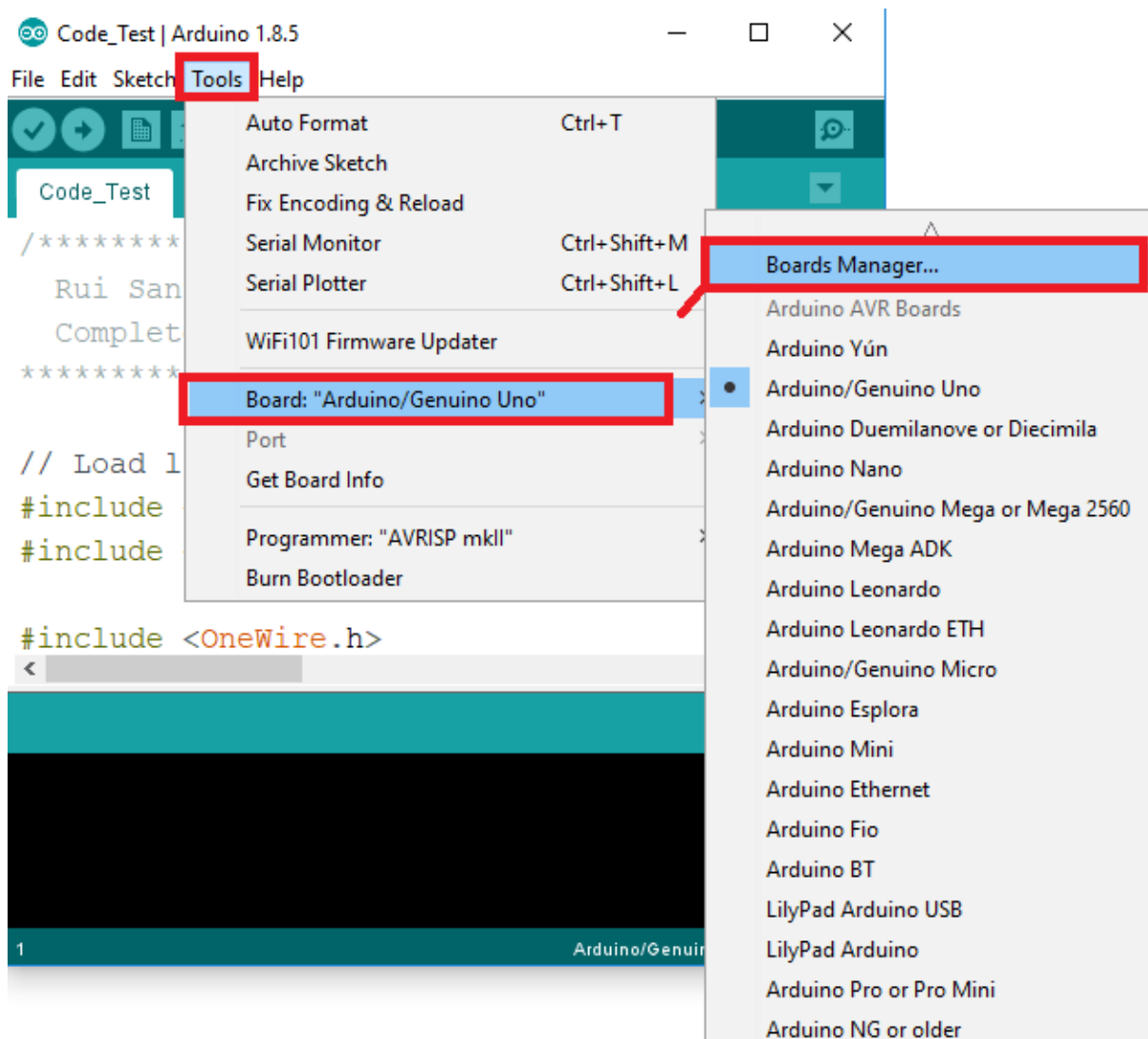
Then, click the “OK” button:



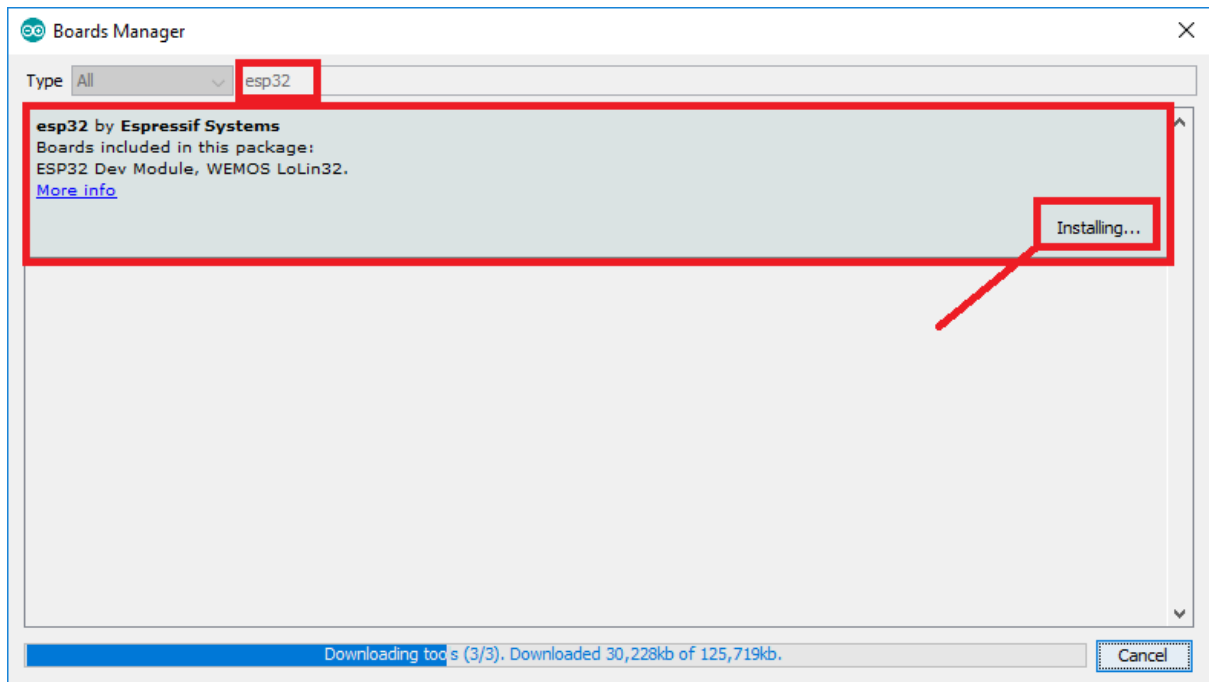
Note: if you already have the ESP8266 boards URL, you can separate the URLs with a comma as follows:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json,
[http://arduino.esp8266.com/stable/package_esp8266com_index.j
son](http://arduino.esp8266.com/stable/package_esp8266com_index.json)

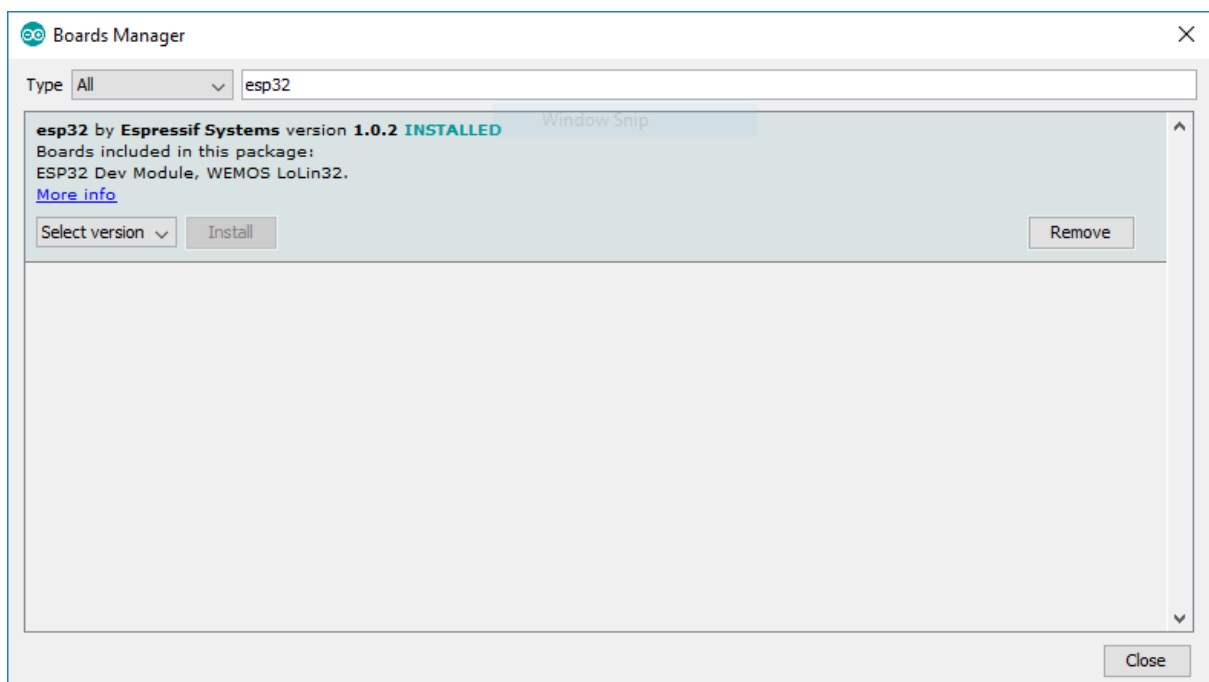
Step 3: Open the Boards Manager. **Go to Tools > Board > Boards Manager...**



Step 4: Search for ESP32 and press install button for the “ESP32 by Espressif Systems”:



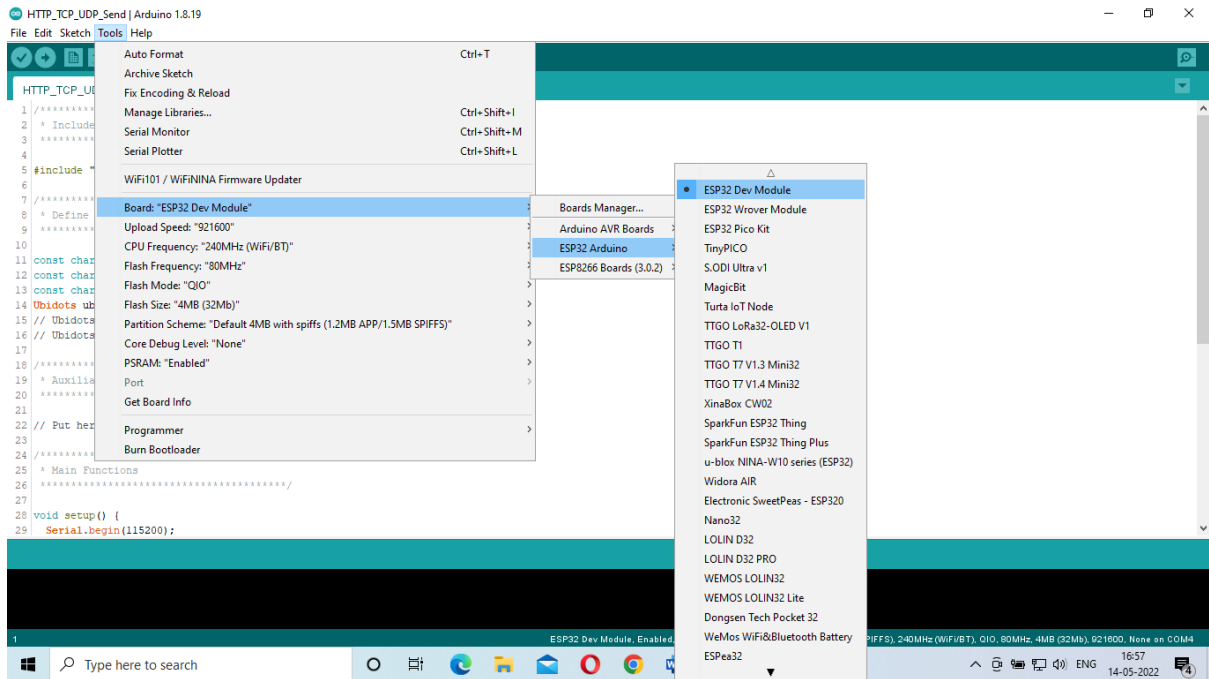
Step 5: That's it. It should be installed after a few seconds.



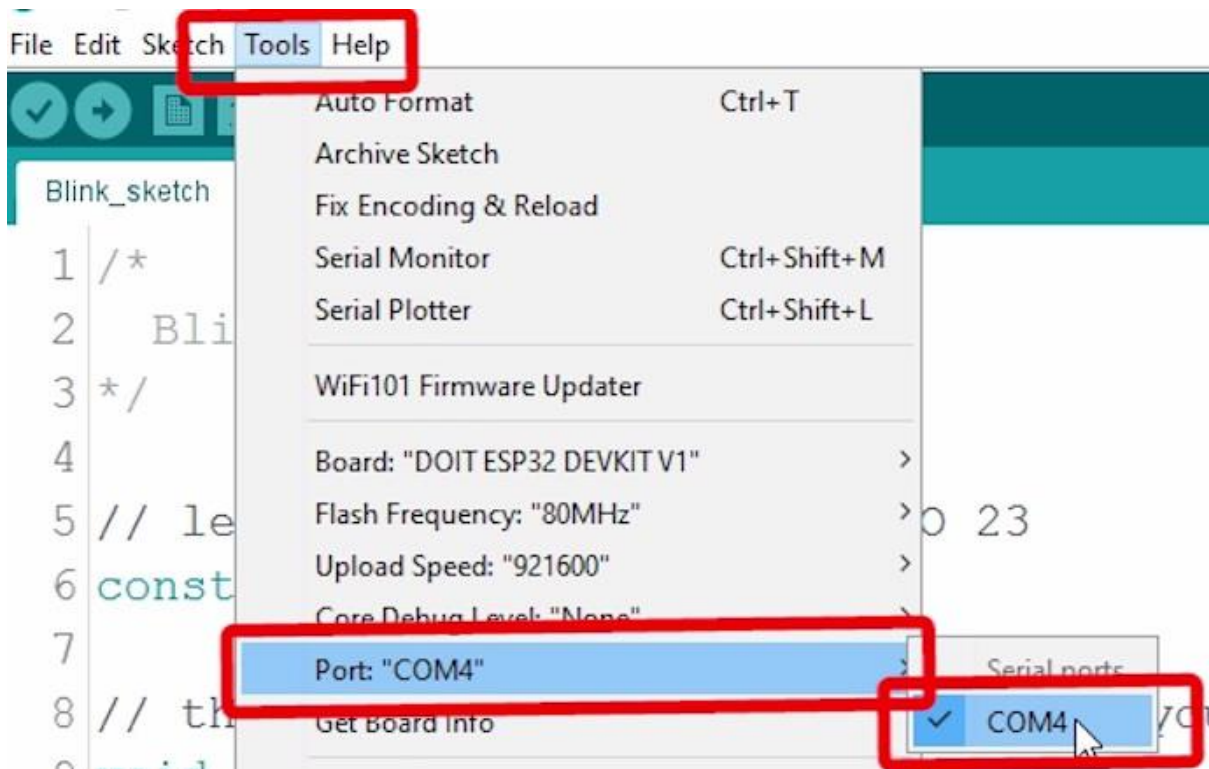
Testing the Installation:

Plug the ESP32 board to your computer. With your Arduino IDE open, follow these steps:

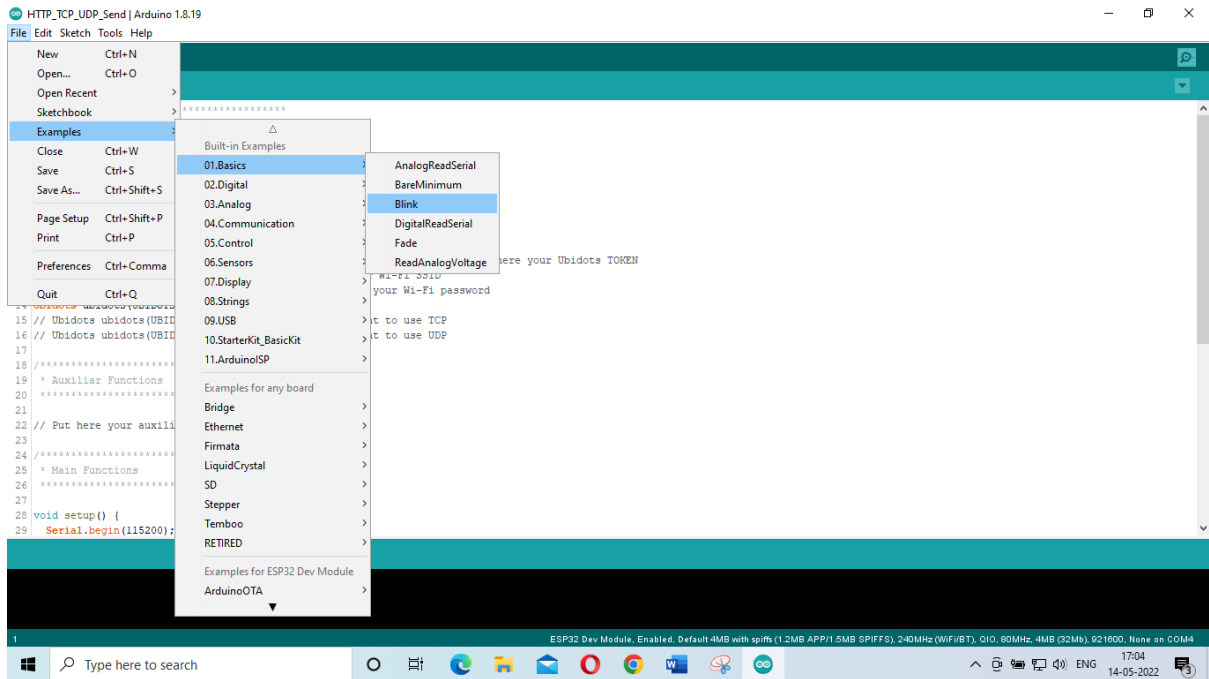
Step 1: Select your Board in Tools > Board menu (in my case it's the "ESP32 Dev Module")



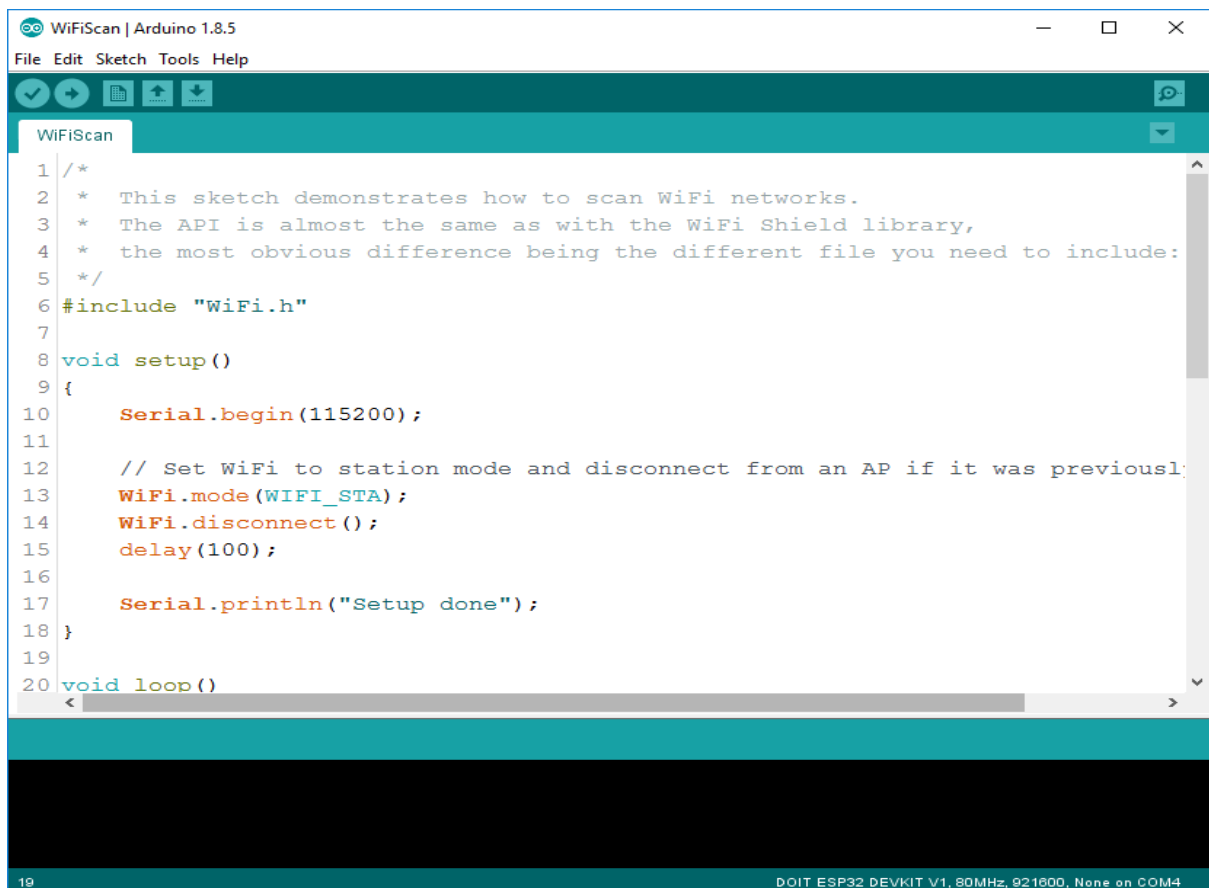
Step 2: Select the Port (if you don't see the COM Port in your Arduino IDE, you need to install the CP210x USB to UART Bridge VCP Drivers):



Step 3: Open the following example under **File > Examples > Basics > Blink**



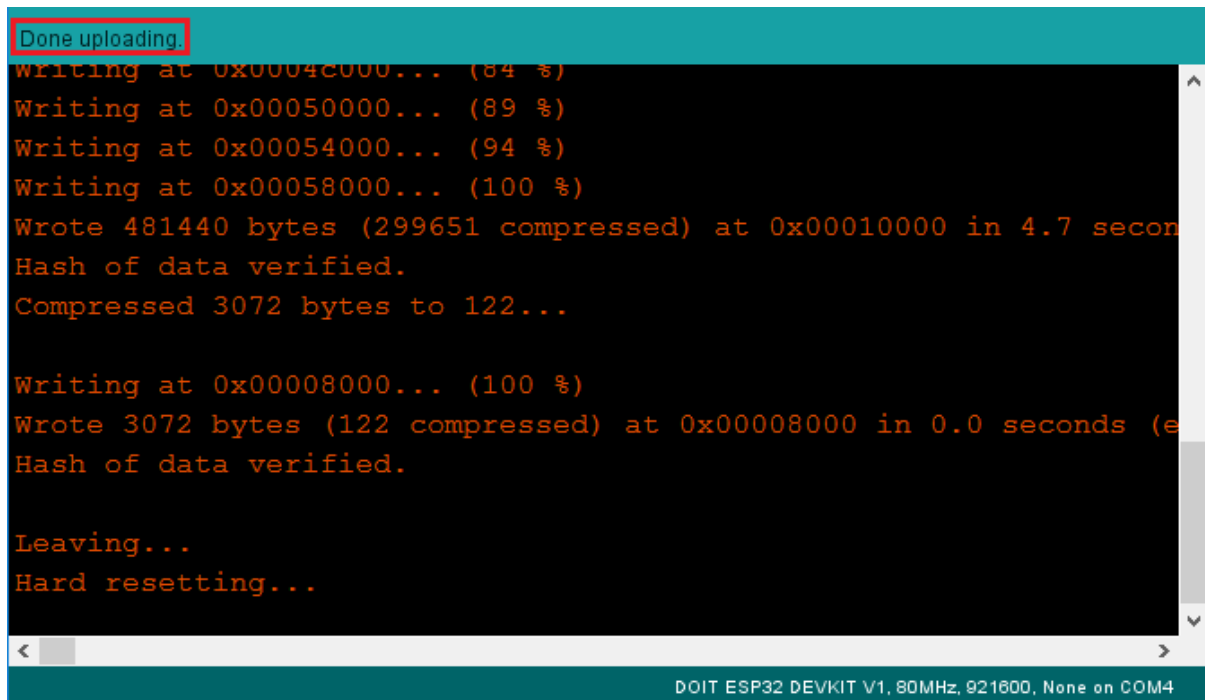
Step 4: A new sketch opens in your Arduino IDE:



Step 5: Press the Upload button in the Arduino IDE. Wait a few seconds while the code compiles and uploads to your board.



Step 6: If everything went as expected, you should see a “Done uploading.” message.

A screenshot of the Arduino IDE terminal window. The terminal has a black background with orange text. At the top, a red rectangular box highlights the text "Done uploading.". Below this, the terminal shows the progress of writing code to memory: "Writing at 0x0004c000... (84 %)", "Writing at 0x00050000... (89 %)", "Writing at 0x00054000... (94 %)", and "Writing at 0x00058000... (100 %)". It then reports "Wrote 481440 bytes (299651 compressed) at 0x00010000 in 4.7 seconds" and "Hash of data verified.". Next, it shows "Compressed 3072 bytes to 122...". This is followed by another write operation: "Writing at 0x00008000... (100 %)", "Wrote 3072 bytes (122 compressed) at 0x00008000 in 0.0 seconds (e", and "Hash of data verified.". The final lines are "Leaving..." and "Hard resetting...". At the bottom of the terminal window, a status bar reads "DOIT ESP32 DEVKIT V1, 80MHz, 921600, None on COM4".

```
Done uploading.
Writing at 0x0004c000... (84 %)
Writing at 0x00050000... (89 %)
Writing at 0x00054000... (94 %)
Writing at 0x00058000... (100 %)
Wrote 481440 bytes (299651 compressed) at 0x00010000 in 4.7 seconds
Hash of data verified.
Compressed 3072 bytes to 122...

Writing at 0x00008000... (100 %)
Wrote 3072 bytes (122 compressed) at 0x00008000 in 0.0 seconds (e
Hash of data verified.

Leaving...
Hard resetting...
```

More details : <https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/>

Experiment -1

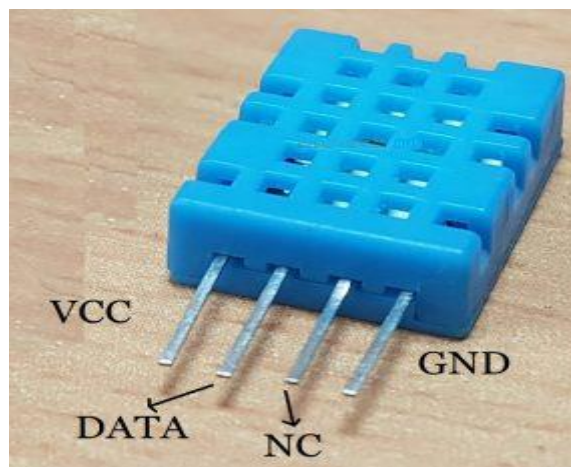
Aim: To measure the Temperature, Humidity and Heat index using ESP32 Development Board and display the values in serial monitor.

APPARATUS:

S No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	DHT11 or DHT22 temperature and humidity sensor	1
3	Jumper wires	3
4	Micro USB cable	1

The DHT11 and DHT22 sensors are used to measure temperature and relative humidity.

- Temperature range: 0 to 50°C +/- 2°C
- Relative humidity range: 20 to 90% +/-5%
- Temperature resolution: 1°C
- Humidity resolution: 1%
- Operating voltage: 3 to 5.5 V DC
- Current supply: 0.5 to 2.5 mA
- Sampling period: 1 second



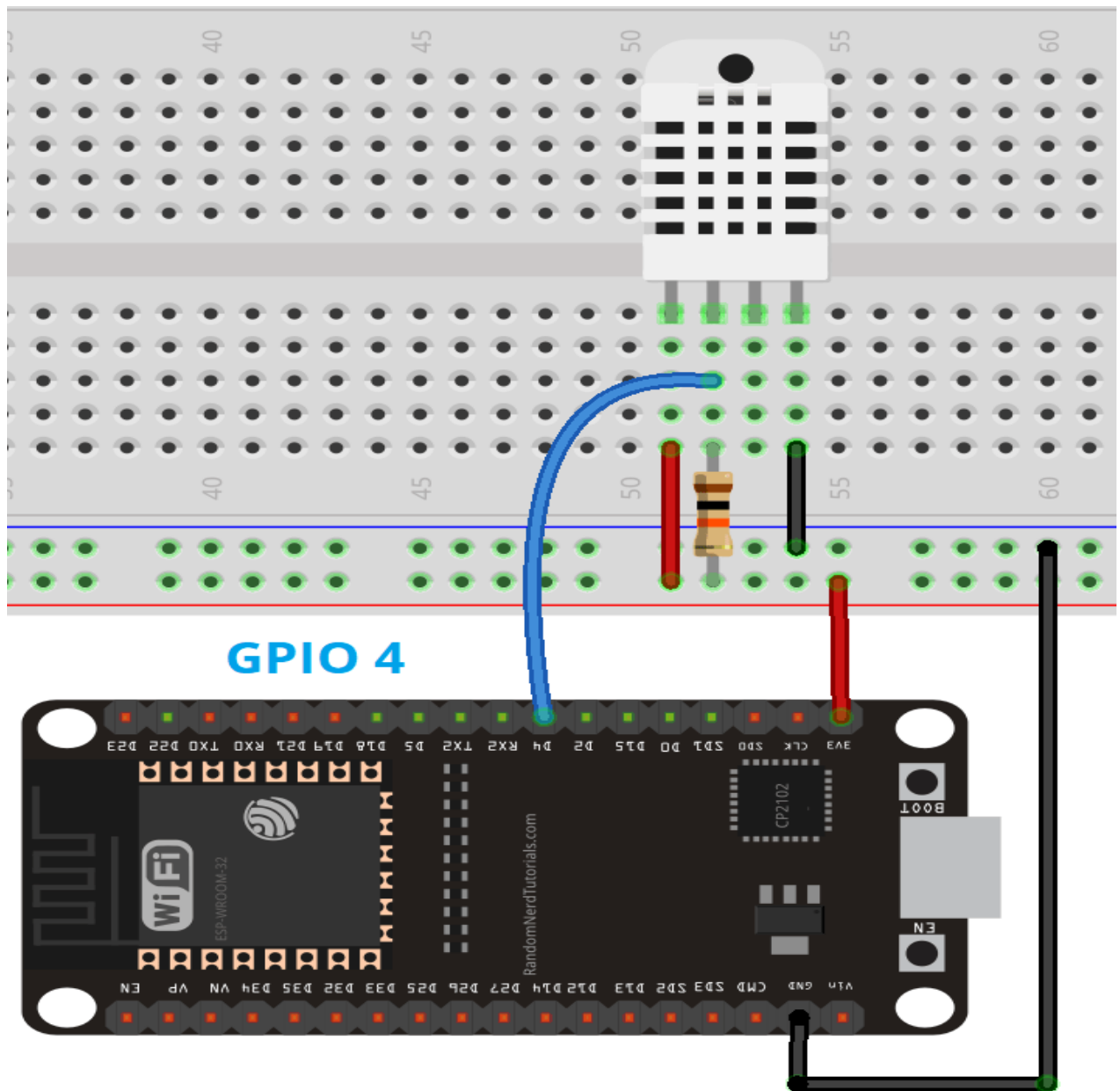


Figure.1 Circuit Diagram for measurement of Temperature and Humidity.

PROCEDURE:

1. Make the connection as per the circuit diagram
2. Open the Arduino IDE in computer and write the program. Save the new sketch in your working directory
 Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings.**
3. Compile the program and upload it to the ESP32 Development Board.
4. After uploading the code, open the Serial Monitor at a baud rate of 115200.

CODE:

```
#include "DHT.h"

#define DHTPIN 4 // Digital pin connected to the DHT sensor
// Feather HUZZAH ESP8266 note: use pins 3, 4, 5, 12, 13 or 14 --
// Pin 15 can work but DHT must be disconnected during program upload.

// Uncomment whatever type you're using!
#define DHTTYPE DHT11 // DHT 11
// #define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
// #define DHTTYPE DHT21 // DHT 21 (AM2301)

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor

// Initialize DHT sensor.
// Note that older versions of this library took an optional third parameter to
// tweak the timings for faster processors. This parameter is no longer needed
// as the current DHT reading algorithm adjusts itself to work on faster procs.
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  Serial.println(F("DHTxx test!"));

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
```

```

float t = dht.readTemperature();
// Read temperature as Fahrenheit (isFahrenheit = true)
float f = dht.readTemperature(true);

// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
}

```

```

// Compute heat index in Fahrenheit (the default)
float hif = dht.computeHeatIndex(f, h);
// Compute heat index in Celsius (isFahreheit = false)
float hic = dht.computeHeatIndex(t, h, false);

```

```

Serial.print(F("Humidity: "));
Serial.print(h);
Serial.print(F("% Temperature: "));
Serial.print(t);
Serial.print(F("°C "));
Serial.print(f);
Serial.print(F("°F Heat index: "));
Serial.print(hic);
Serial.print(F("°C "));
Serial.print(hif);
Serial.println(F("°F"));
}

```

```

// Compute heat index in Fahrenheit (the default)
float hif = dht.computeHeatIndex(f, h);
// Compute heat index in Celsius (isFahreheit = false)
float hic = dht.computeHeatIndex(t, h, false);

```

```

Serial.print(F("Humidity: "));
Serial.print(h);
Serial.print(F("% Temperature: "));
Serial.print(t);
Serial.print(F("°C "));
Serial.print(f);
Serial.print(F("°F Heat index: "));
Serial.print(hic);
Serial.print(F("°C "));
Serial.print(hif);
Serial.println(F("°F"));

```

}

Result:

COM7

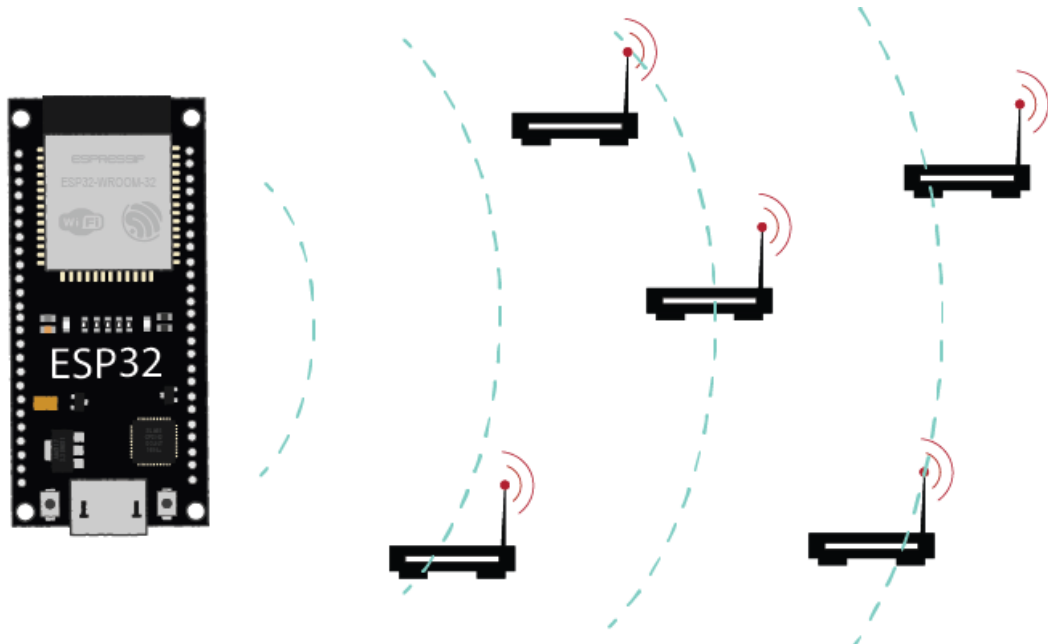
Send

Humidity: 85.40%	Temperature: 20.40°C	68.72°F	Heat index: 20.73°C	69.31°F
Humidity: 85.40%	Temperature: 20.50°C	68.90°F	Heat index: 20.84°C	69.50°F
Humidity: 85.40%	Temperature: 20.40°C	68.72°F	Heat index: 20.73°C	69.31°F
Humidity: 85.40%	Temperature: 20.40°C	68.72°F	Heat index: 20.73°C	69.31°F
Humidity: 85.40%	Temperature: 20.50°C	68.90°F	Heat index: 20.84°C	69.50°F
Humidity: 85.40%	Temperature: 20.50°C	68.90°F	Heat index: 20.84°C	69.50°F
Humidity: 85.30%	Temperature: 20.40°C	68.72°F	Heat index: 20.72°C	69.30°F
Humidity: 85.30%	Temperature: 20.50°C	68.90°F	Heat index: 20.83°C	69.50°F
Humidity: 85.30%	Temperature: 20.50°C	68.90°F	Heat index: 20.83°C	69.50°F
Humidity: 85.30%	Temperature: 20.50°C	68.90°F	Heat index: 20.83°C	69.50°F
Humidity: 85.30%	Temperature: 20.50°C	68.90°F	Heat index: 20.83°C	69.50°F
Humidity: 85.20%	Temperature: 20.40°C	68.72°F	Heat index: 20.72°C	69.30°F
Humidity: 85.20%	Temperature: 20.50°C	68.90°F	Heat index: 20.83°C	69.49°F
Humidity: 85.20%	Temperature: 20.50°C	68.90°F	Heat index: 20.83°C	69.49°F

☒ Autoscroll ☐ Show timestamp Newline 9600 baud Clear output

Experiment -2

2 a: Scan Wi-Fi networks and get Wi-Fi strength using ESP32 Development board.



Code:

```
#include "WiFi.h"
```

```
void setup()
```

```
{
```

```
  Serial.begin(115200);
```

```
  // Set WiFi to station mode and disconnect from an AP if it was previously connected
```

```
  WiFi.mode(WIFI_STA);
```

```
  WiFi.disconnect();
```

```
  delay(100);
```

```
  Serial.println("Setup done");
```

```
}
```

```
void loop()
```

```
{
```

```
  Serial.println("scan start");
```

```
  // WiFi.scanNetworks will return the number of networks found
```

```
  int n = WiFi.scanNetworks();
```

```
  Serial.println("scan done");
```

```
  if (n == 0) {
```

```

    Serial.println("no networks found");
} else {
    Serial.print(n);
    Serial.println(" networks found");
    for (int i = 0; i < n; ++i) {
        // Print SSID and RSSI for each network found
        Serial.print(i + 1);
        Serial.print(": ");
        Serial.print(WiFi.SSID(i));
        Serial.print(" ");
        Serial.print(WiFi.RSSI(i));
        Serial.print(")");
        Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)? " ":"*");
        delay(10);
    }
}
Serial.println("");

// Wait a bit before scanning again
delay(5000);
}

```

2 b: Configure/Set Your ESP32 Development board as an access point

When you set your ESP32 board as an access point, you can be connected using any device with Wi-Fi capabilities without connecting to your router. When you set the ESP32 as an access point, you create its own Wi-Fi network, and nearby Wi-Fi devices (stations) can connect to it, like your smartphone or computer. So, you don't need to be connected to a router to control it.



CODE:

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

```
char ssid[]="SSID Name(your choice)";  
char password[]="Password(your choice)";
```

```
void setup()
```

```
{  
  Serial.begin(115200);
```

```
  Serial.print("Starting Access-Point...");  
  WiFi.begin(ssid,password);
```

```
}
```

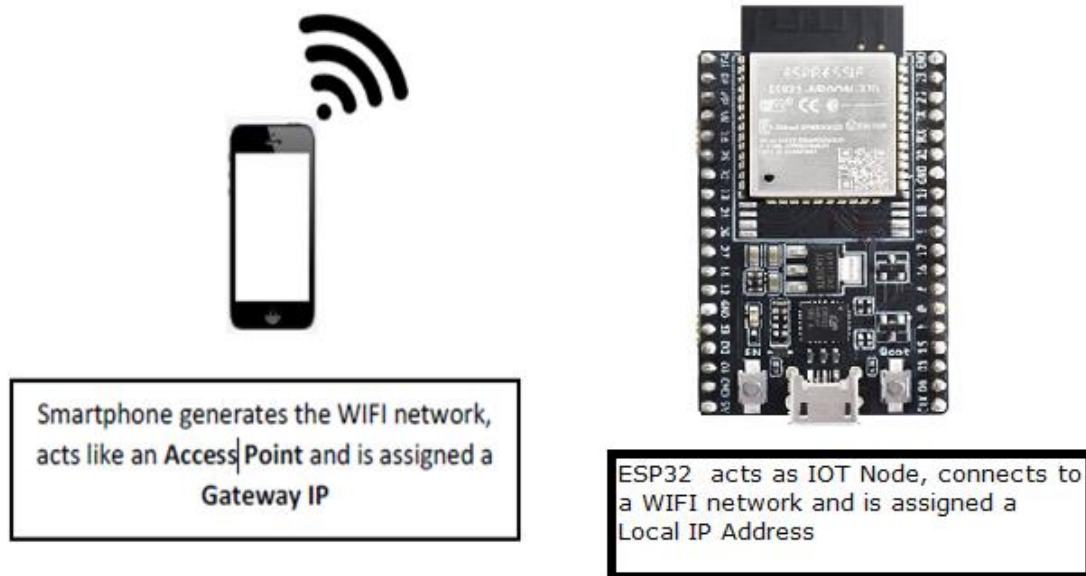
```
void loop()
```

```
{  
  // put your main code here, to run repeatedly:
```

```
}
```


2 c: Establish connection between IoT Node and Access point and Display of local IP and Gateway IP

The goal of this experiment is to understand the configuration of IoT Node to connect to an Access Point. Any device when connected to a network is assigned an IP address, which is a unique number for each device on a network. We shall also see how to identify the IP network of our IoT Node. The Access Point also has an IP address, known as the Gateway IP. These concepts are fundamentals to understand the architecture of an IoT Network.



CODE:

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

```
char ssid[]="REPLACE_WITH_YOUR_SSID";  
char password[]="REPLACE_WITH_YOUR_PASSWORD";
```

```
IPAddress ip;  
IPAddress gateway;
```

```
void setup()  
{  
  Serial.begin(115200); //Initialize Serial Port  
  //attempt to connect to wifi  
  Serial.print("Attempting to connect to Network named: ");  
  // print the network name (SSID);
```

```
Serial.println(ssid);

//Connect to WiFi
WiFi.begin(ssid, password);

//Wait untill wifi is connected
while ( WiFi.status() != WL_CONNECTED)
{
  // print dots while we wait to connect
  Serial.print(".");
  delay(300);
}

//If you are connected print the IP Address
Serial.println("\nYou're connected to the network");

//wait untill you get an IP address
while (WiFi.localIP() == INADDR_NONE) {
  // print dots while we wait for an ip addresss
  Serial.print(".");
  delay(300);
}

ip=WiFi.localIP();
Serial.println(ip);
gateway=WiFi.gatewayIP();
Serial.println("GATEWAY IP:");
Serial.println(gateway);
}

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

}
```