Tektork IoT Kit
Sensors Wiring and Coding

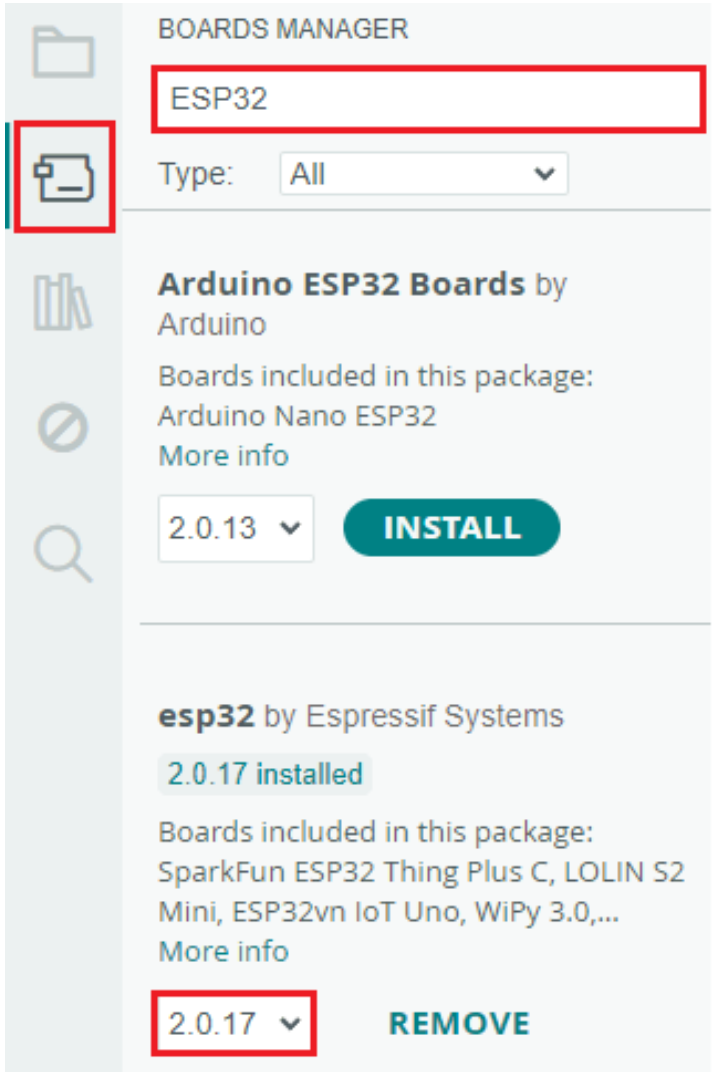
Tektork Private Limited, Coimbatore

Hall Effect Sensor Code

```
int val = 0;  // Variable to store sensor reading
void setup() {
  Serial.begin(115200);                  // Start serial communication at 115200 baud
  Serial.println("ESP32 Hall Effect Sensor Test");  // Print startup message
}
void loop() {
  val = hallRead();   // Read raw value from the built-in Hall effect sensor
  Serial.print("Hall Value: ");
  Serial.print(val);  // Print the sensor value

  // Check if the absolute value is beyond threshold (|val| > 50)
  // Larger values indicate stronger magnetic field detection
  if (abs(val) > 50) {
    Serial.println(" - Magnetic field detected!");
  } else {
    Serial.println(" - No magnetic field");
  }
    delay(1000);   // Wait 1 second before next reading
}
```

Note: The hallRead() function may not be supported in newer versions of the ESP32 Arduino core (version 3.X and above). If you encounter issues, you may need to downgrade to version 2 of the ESP32 board package

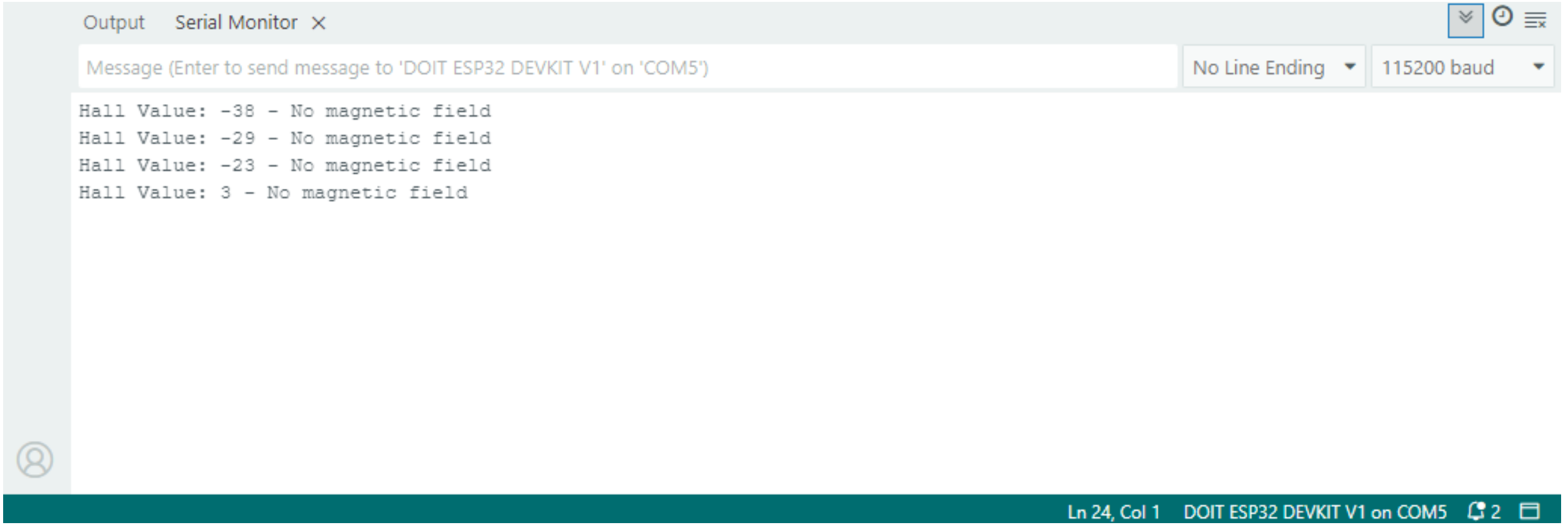
Arduino IDE ESP32 Core Version (Board Manager version)



If you installed the ESP32 board support via Arduino IDE (using *Boards Manager*), you can downgrade to a previous release:

- Open **Arduino IDE → Tools → Board → Boards Manager**.
- Search for **ESP32**.
- Select the version dropdown and choose the desired **older version** (e.g., 2.0.11, 2.0.9, or 1.0.6 which many libraries still support).
- Click **Install**.
- The IDE will replace the current version with the selected one.

Hall Effect Sensor - Output



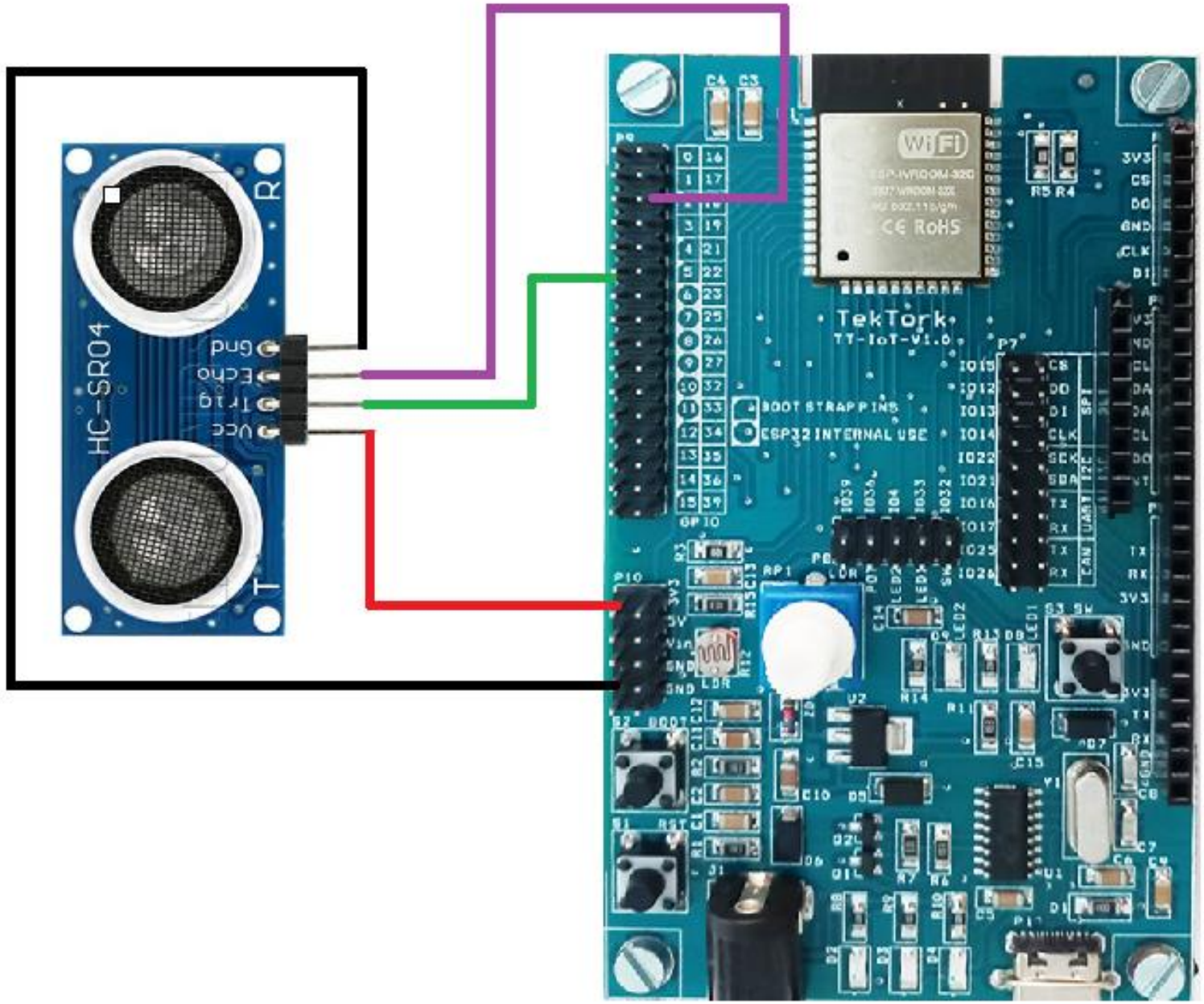
Ultrasonic Sensor

Ultrasonic Sensor Pinout

Ultrasonic Sensor Pin to ESP32

Ultrasonic HC-SR04	ESP32
VCC	3.3 V
TRIGGER	GPIO 5
ECHO	GPIO 18
GND	GND

Wiring Diagram of the Ultrasonic Sensor Connected to ESP 32



Ultrasonic Sensor Pin to ESP32

Ultrasonic HC-SR04	ESP32
VCC	3.3 V
TRIGGER	GPIO 5
ECHO	GPIO 18
GND	GND

Ultrasonic Sensor Specification

Parameter	Value / Description
Operating Voltage	5V DC
Operating Current	15 mA (typical)
Frequency	40 kHz
Max Range	4–5 meters
Min Range	2 cm
Accuracy	±3 mm
Trigger Input Signal	10 µs TTL pulse
Echo Output Signal	TTL pulse width proportional to distance
Measuring Angle	~15 degrees
Dimensions	45 mm × 20 mm × 15 mm
Interface	4 pins: VCC, Trig, Echo, GND

Ultrasonic Sensor Code

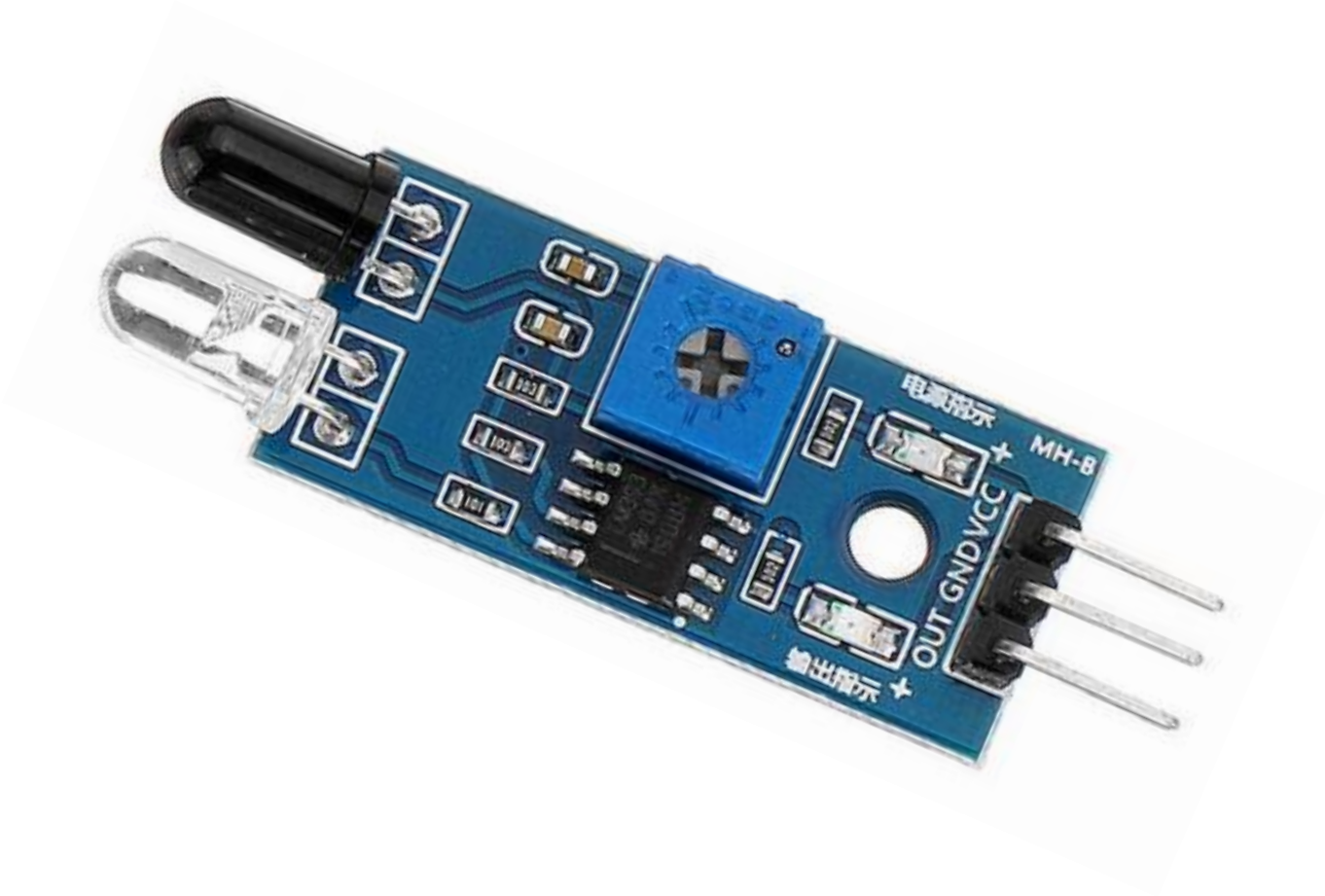
```
// Define ultrasonic sensor pins
#define TRIG_PIN 5    // GPIO pin used for Trigger
#define ECHO_PIN 18   // GPIO pin used for Echo

long duration;        // Variable to store echo pulse duration (µs)
float distanceCm;     // Variable to store calculated distance

void setup() {
  Serial.begin(115200);       // Start serial communication at 115200 baud
  pinMode(TRIG_PIN, OUTPUT); // Set trigger pin as output
  pinMode(ECHO_PIN, INPUT);  // Set echo pin as input
}
void loop() {
  // Step 1: Clear the trigger pin
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  // Step 2: Send a 10 µs HIGH pulse on trigger pin
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
```

Ultrasonic Sensor Code Cond..

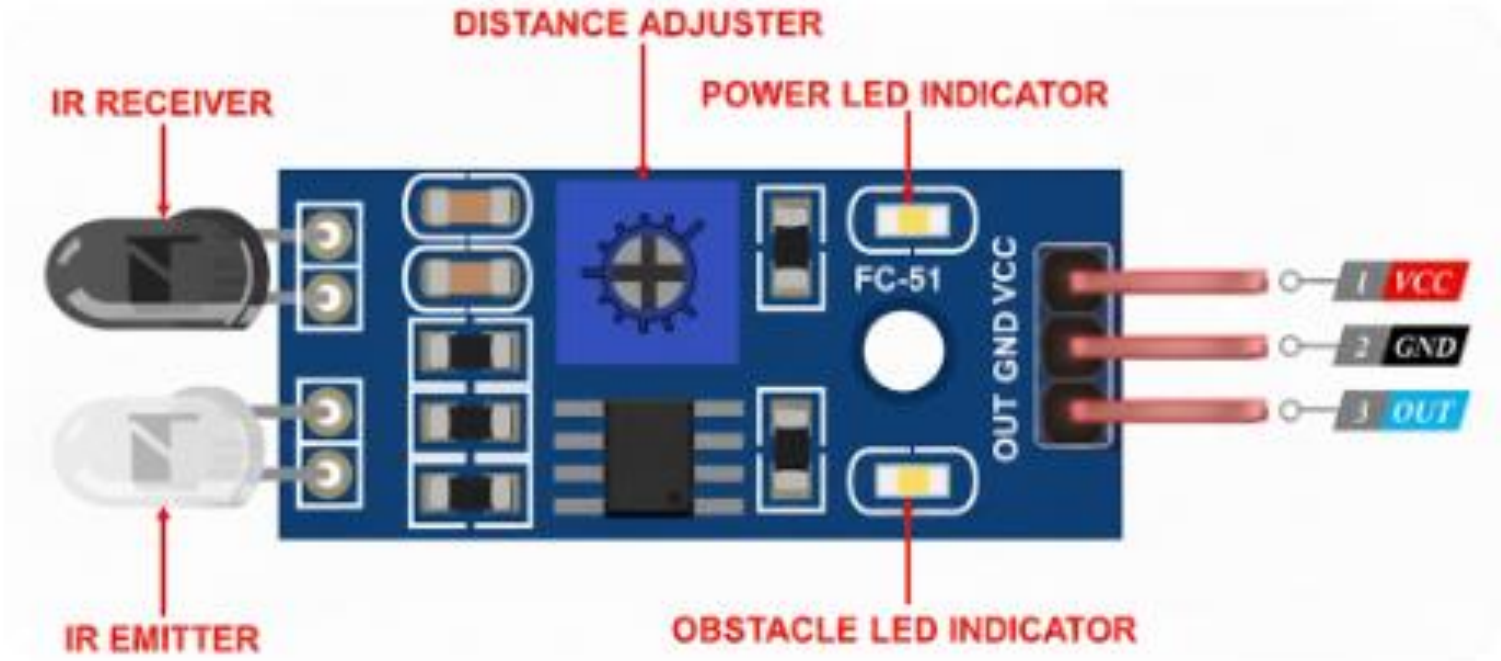
```
  // Step 3: Measure time taken for echo to return
  // pulseIn() waits for the echo pin to go HIGH and then LOW
  duration = pulseIn(ECHO_PIN, HIGH);

  // Step 4: Convert duration to distance (cm)
  // Formula: distance = (time * speed of sound) / 2
  // Speed of sound = 343 m/s = 0.0343 cm/µs
  distanceCm = (duration * 0.0343) / 2;

  // Step 5: Print distance to Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distanceCm);
  Serial.println(" cm");

  // Small delay before next measurement
  delay(500);
}
```

Ultrasonic Sensor Output

IR Sensor



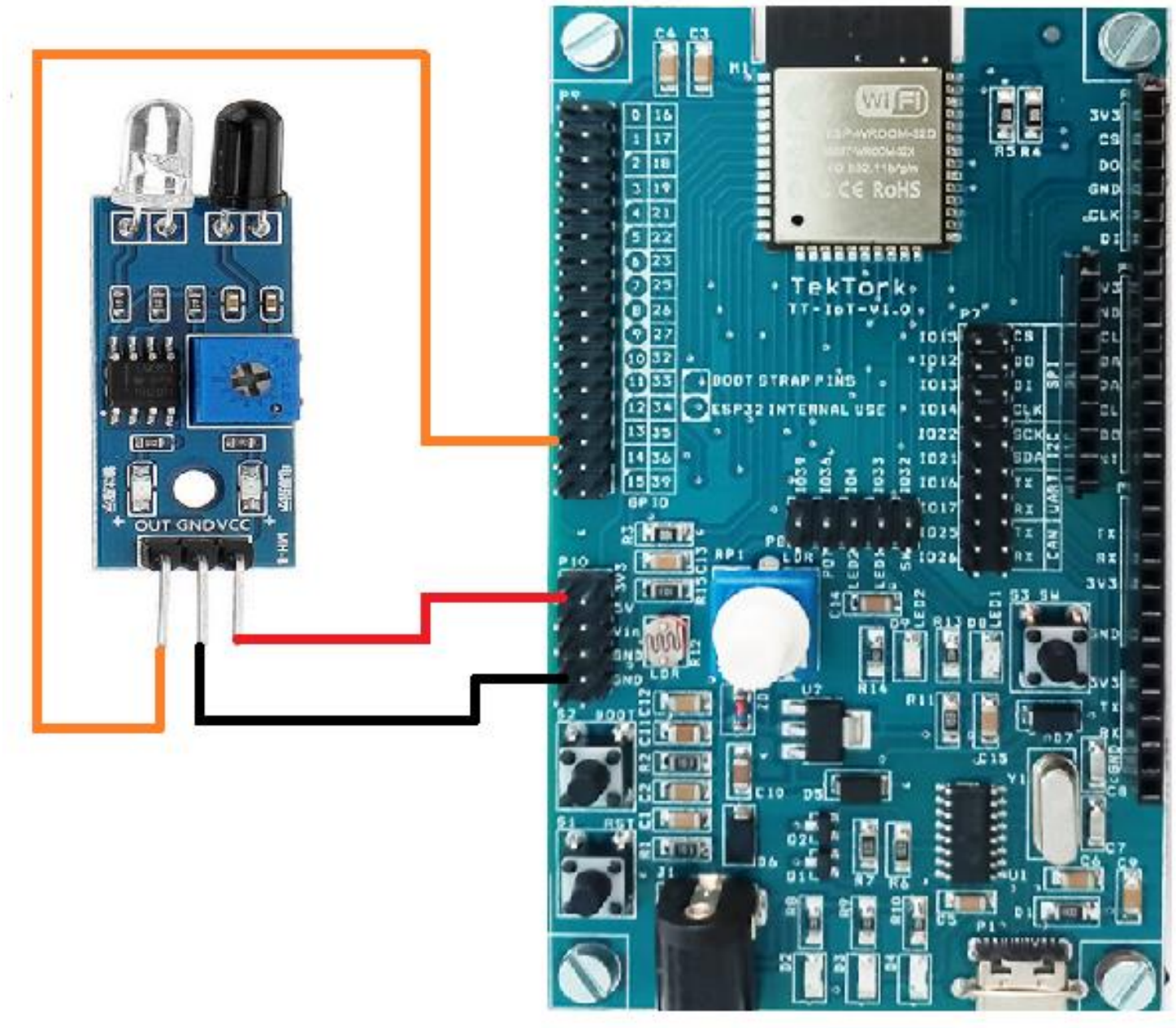
IR Sensor



IR Sensor Pin to ESP32

IR Sensor Pin	ESP32 Pin
VCC	VIN/5V/3.3V
GND	GND
OUT	GPIO13

Wiring Diagram of the IR Sensor Connected to ESP 32



Ultrasonic Sensor Pin to ESP32

IR Sensor Pin	ESP32 Pin
VCC	VIN/5V/3.3V
GND	GND
OUT	GPIO13

IR Sensor Specification

Parameter	Typical Value / Description
Sensor Type	Infrared Reflective / IR Transmitter & Receiver
Operating Voltage	3.3V – 5V DC (compatible with ESP32, Arduino)
Output Type	Digital (0/1) or Analog (some modules)
Detection Range	2 cm – 30 cm (depends on module and surface)
Signal Output	High/Low (digital), Analog voltage proportional to reflection
Current Consumption	20 – 25 mA (IR LED ON)
Operating Temperature	-10 °C to 70 °C
Response Time	~10 ms
Dimensions	Varies (~3–5 cm × 1–2 cm module)

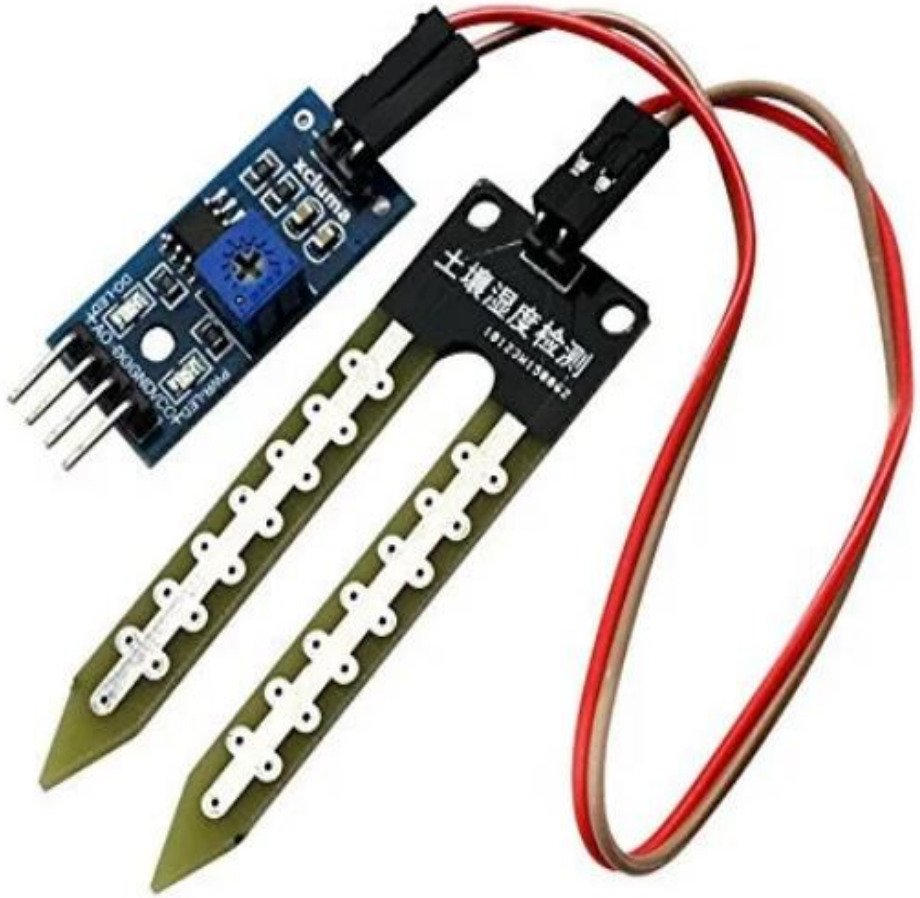
IR Sensor Code

```
#define IR_SENSOR_PIN 13  // Define ESP32 GPIO13 as the digital input pin connected to the IR sensor output
void setup() {
  Serial.begin(115200);           // Initialize serial communication at 115200 baud
  pinMode(IR_SENSOR_PIN, INPUT);  // Configure GPIO13 as input because sensor sends digital HIGH/LOW signals
  Serial.println("IR Obstacle Sensor Test");  // Print header message for reference
 }
void loop() {
  int sensorValue = digitalRead(IR_SENSOR_PIN);  // Read digital state from IR sensor pin (HIGH or LOW)
  // Typical IR obstacle sensor logic:
  // - LOW signal means obstacle detected (sensor output active low)
  // - HIGH signal means no obstacle detected
  if (sensorValue == LOW) {
    Serial.println("Obstacle detected!"); // Print when an obstacle is present
  } else {
    Serial.println("No obstacle");       // Print when no obstacle is detected
  }
  delay(1000);  // Delay 1 second before next reading
}
```

IR Sensor Output

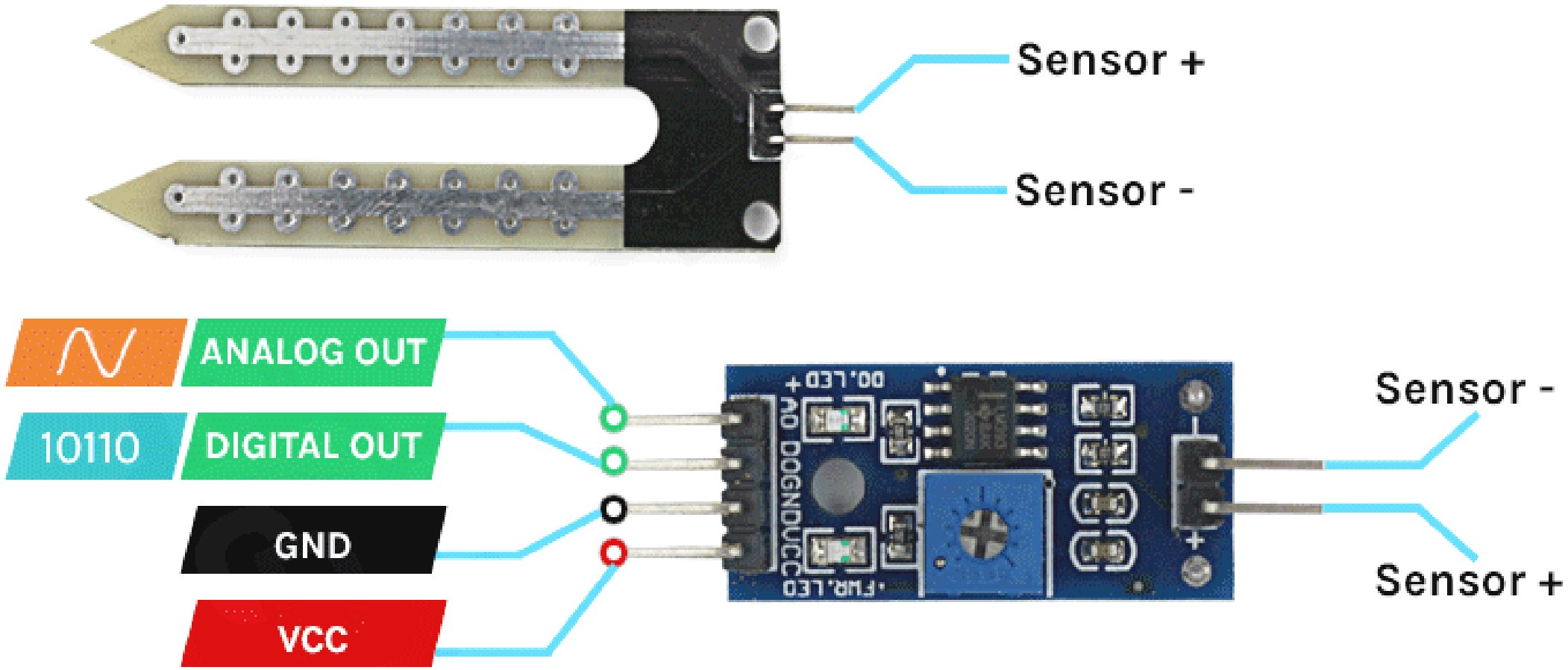
Soil Moisture Sensor

Resistive Soil Moisture Sensor

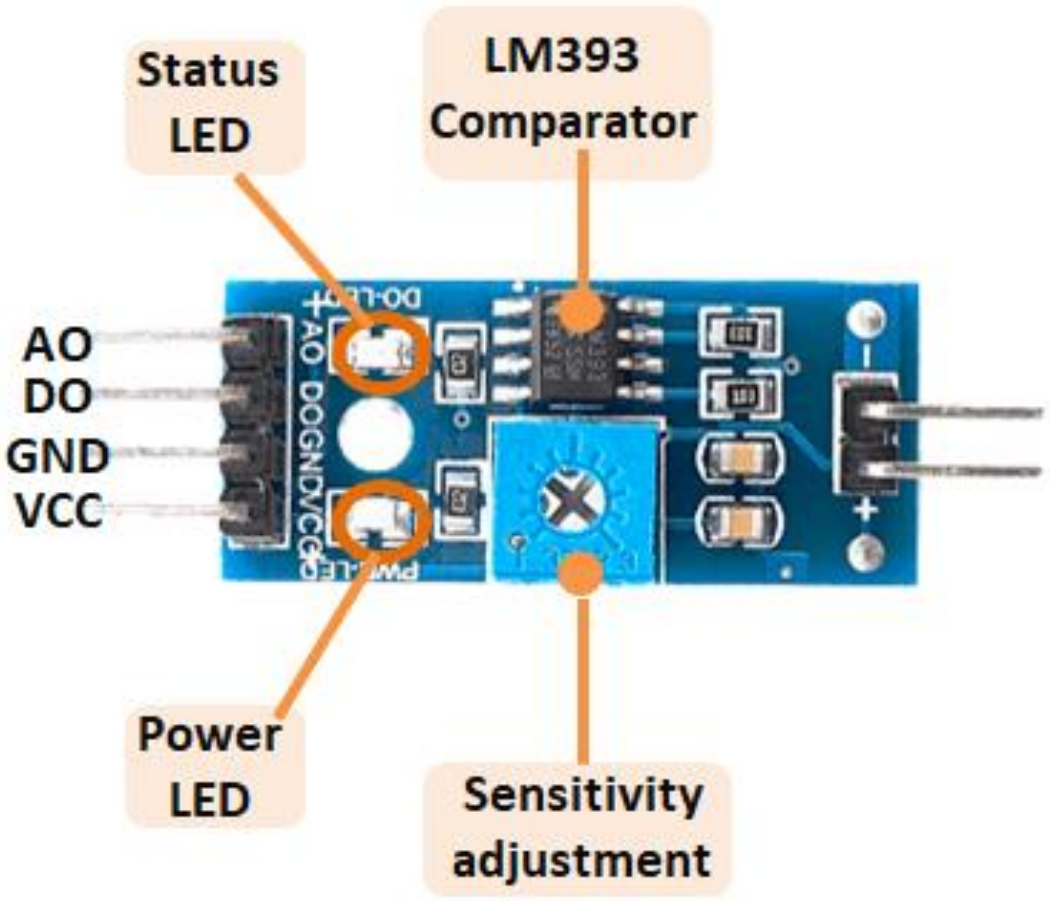


Capacitive Soil Moisture Sensor

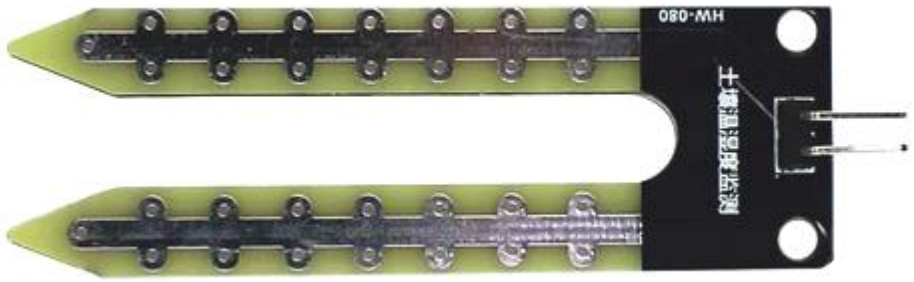
Resistive Soil Moisture Sensor Pinout



Soil Moisture Sensor Module



Soil Moisture Sensor Probe



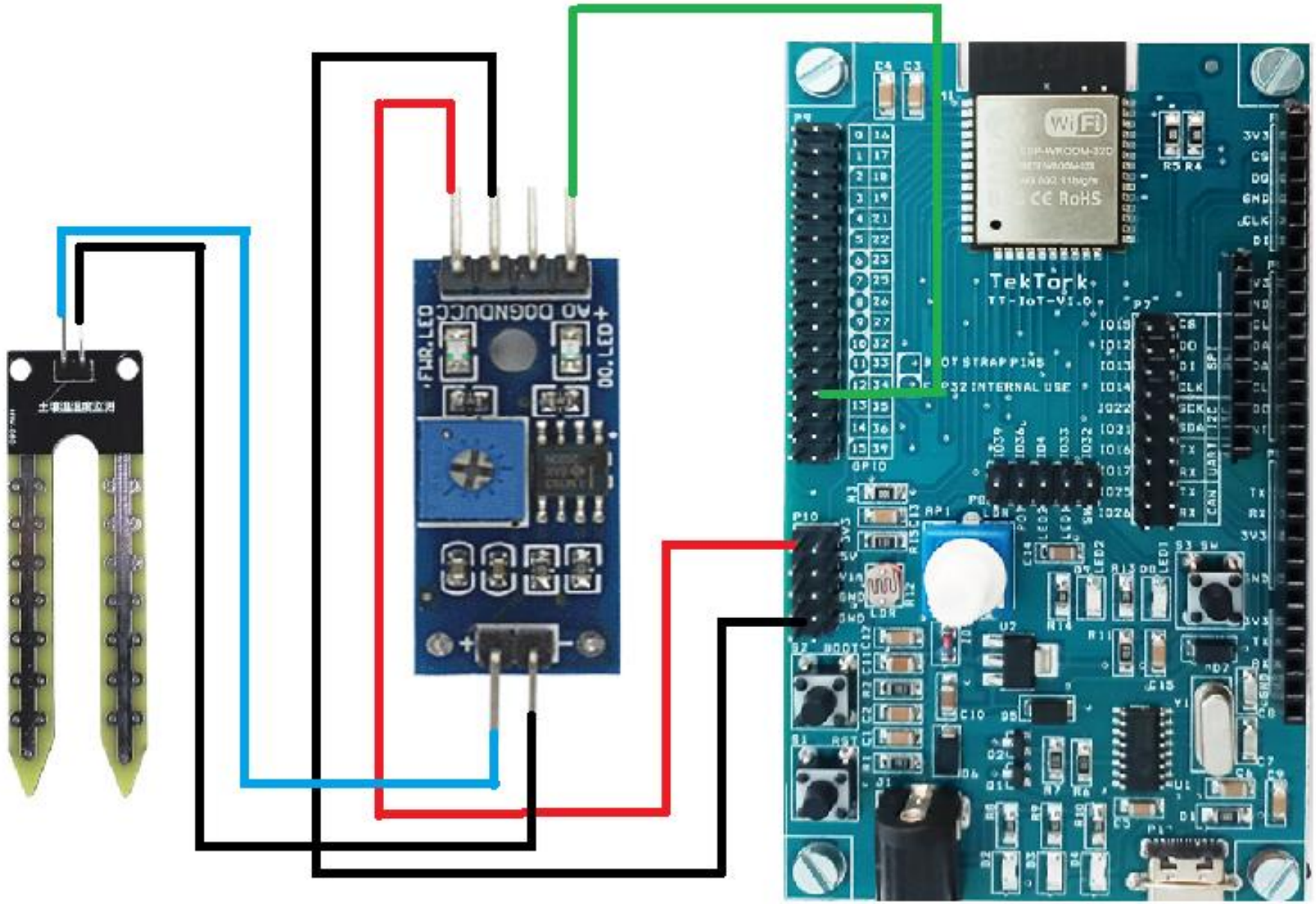
Soil Moisture Sensor Pin to ESP32

Soil Moisture Sensor Pin	ESP32 Pin
A0	GPIO34
D0	GPIO13
GND	GND
VCC	VIN/5V/3.3V

Soil Moisture Sensor Specification

Parameter	Specification
Operating Voltage	3.3V – 5V DC (works with ESP32 & Arduino)
Operating Current	~15 mA
Output Types	Analog (0–5V), Digital (0V or 5V)
ADC Resolution	10-bit (0–1023 for Arduino, 12-bit / 0–4095 for ESP32)
Comparator IC	LM393 (used for digital output threshold detection)
Adjustability	Potentiometer (sets digital threshold level)
PCB Size	~3.2 cm × 1.4 cm
Probe Material	Conducting plates, often with immersion gold plating for corrosion resistance

Wiring Diagram of the Soilmoisture Sensor Connected to ESP 32



Soil Moisture Sensor Pin to ESP32

Soil Sensor Pin	ESP32 Pin
A0	GPIO34
D0	GPIO13
GND	GND
VCC	VIN/5V/3.3V

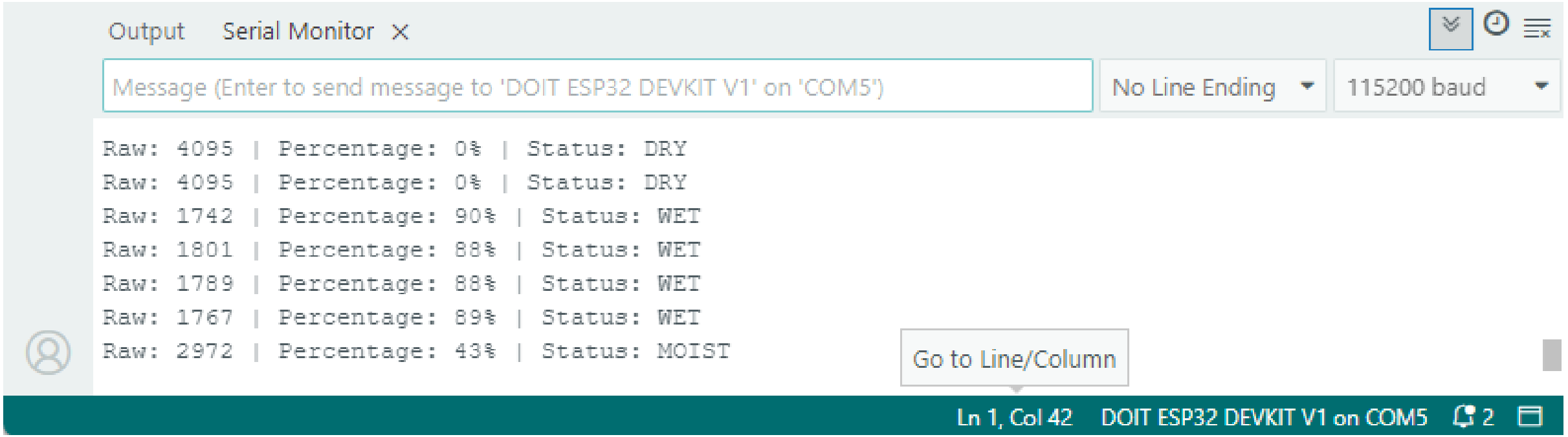
Soil Moisture Sensor Code

```
const int moisturePin = 34;      // GPIO34 (ADC1_CH0) is used for analog input
const int dryValue = 4095;       // ADC value when sensor is completely dry (in air)
const int wetValue = 1500;      // ADC value when sensor is fully wet (dipped in water)

void setup() {
  Serial.begin(115200);                    // Start Serial communication at 115200 baud
  analogSetAttenuation(ADC_11db);          // Configure ADC range for 0-3.3V input
  delay(1000);                             // Small delay for stabilization
  Serial.println("ESP32 Soil Moisture Sensor"); // Print header text to Serial Monitor

}

void loop() {
  // Read raw analog value from the soil moisture sensor (range: 0-4095 for 12-bit
ADC)
  int rawValue = analogRead(moisturePin);
  // Map the raw sensor value into percentage (0% dry → 100% wet)
  // Since higher ADC value means drier soil, we invert the range using dryValue→0 and
wetValue→100
  int moisturePercent = map(rawValue, dryValue, wetValue, 0, 100);
```

Soil Moisture Sensor Code Cond..

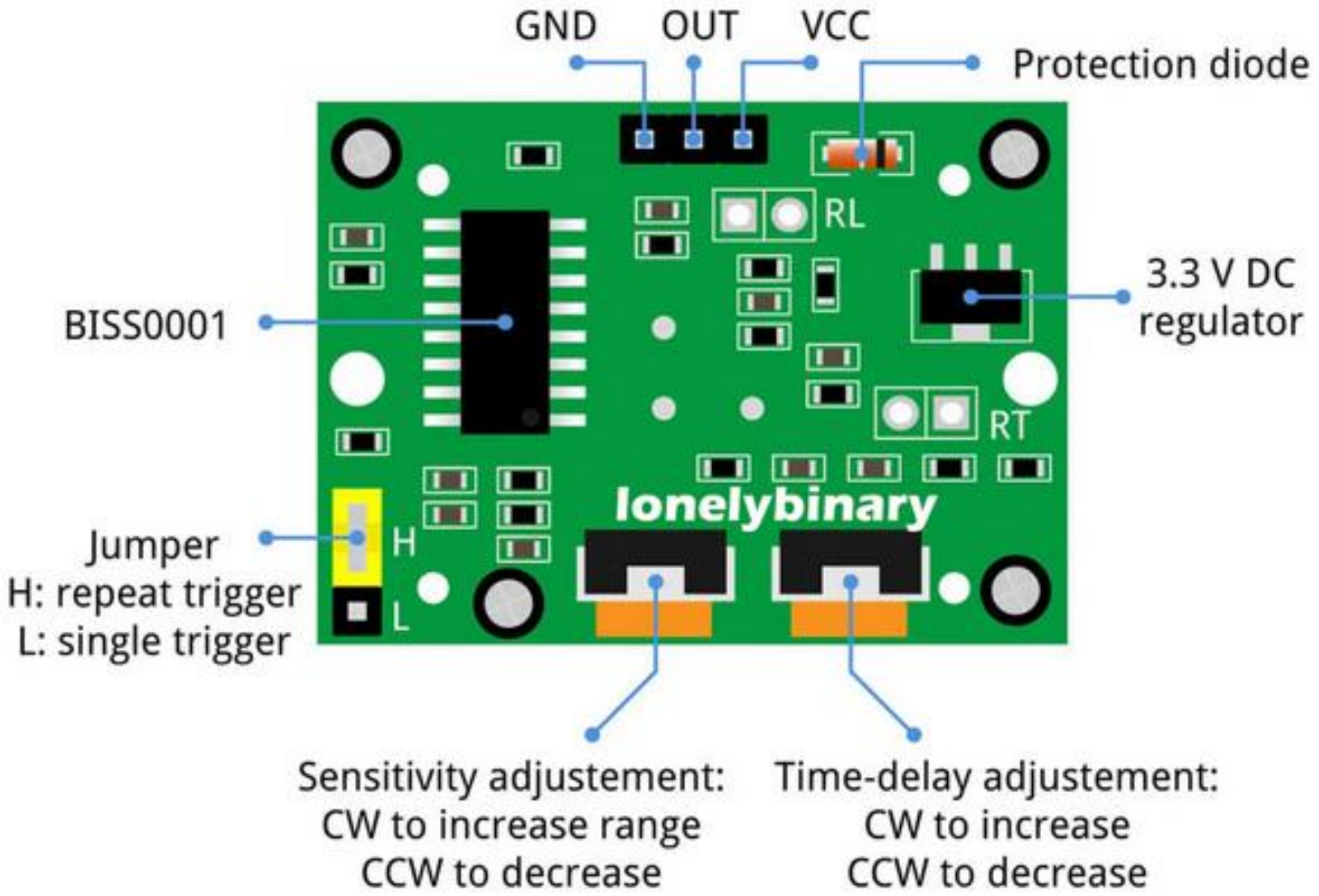
```
// Keep percentage within valid range (0-100%)
  moisturePercent = constrain(moisturePercent, 0, 100);

  // Decide soil status based on percentage
  String status;
  if (moisturePercent > 60) {
    status = "WET";        // Above 60% is considered wet
  } else if (moisturePercent > 30) {
    status = "MOIST";      // Between 30-60% is moderately moist
  } else {
    status = "DRY";        // Below 30% is considered dry
  }
    // Print sensor readings to Serial Monitor
  Serial.print("Raw: ");
  Serial.print(rawValue);
  Serial.print(" | Percentage: ");
  Serial.print(moisturePercent);
  Serial.print("% | Status: ");
  Serial.println(status);
  delay(1000); // Wait for 1 seconds before the next reading
}
```

Output

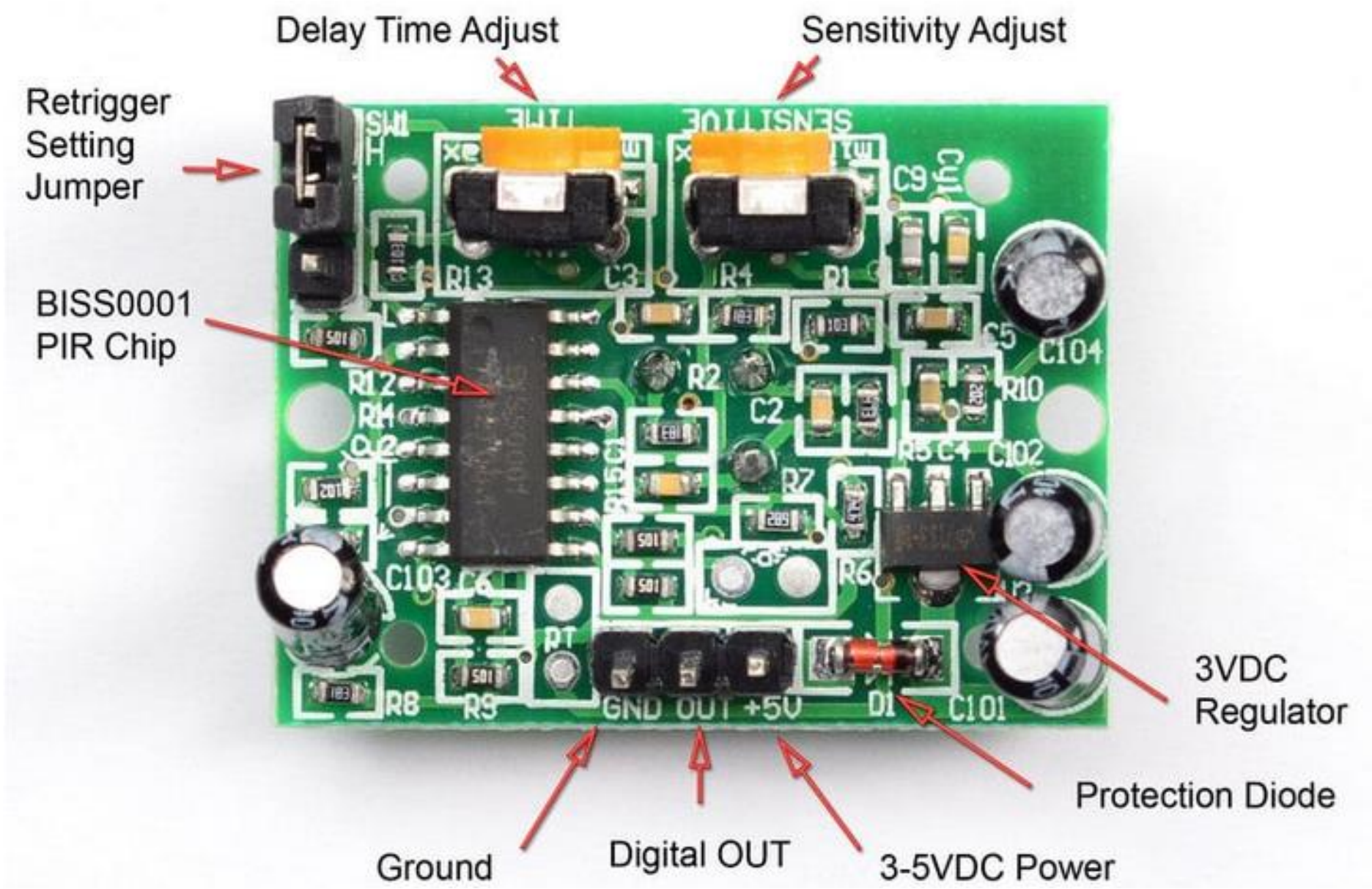

Output
Serial Monitor
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')
No Line Ending
115200 baud
Raw: 4095 | Percentage: 0% | Status: DRY
Raw: 4095 | Percentage: 0% | Status: DRY
Raw: 1742 | Percentage: 90% | Status: WET
Raw: 1801 | Percentage: 88% | Status: WET
Raw: 1789 | Percentage: 88% | Status: WET
Raw: 1767 | Percentage: 89% | Status: WET
Raw: 2972 | Percentage: 43% | Status: MOIST
Go to Line/Column
Ln 1, Col 42
DOIT ESP32 DEVKIT V1 on COM5
2

PIR Sensor

PIR Sensor


GND
OUT
VCC
Protection diode
RL
3.3 V DC
regulator
BISS0001
RT
lonelybinary
Jumper
H: repeat trigger
L: single trigger
H
L
Sensitivity adjustement:
CW to increase range
CCW to decrease
Time-delay adjustement:
CW to increase
CCW to decrease

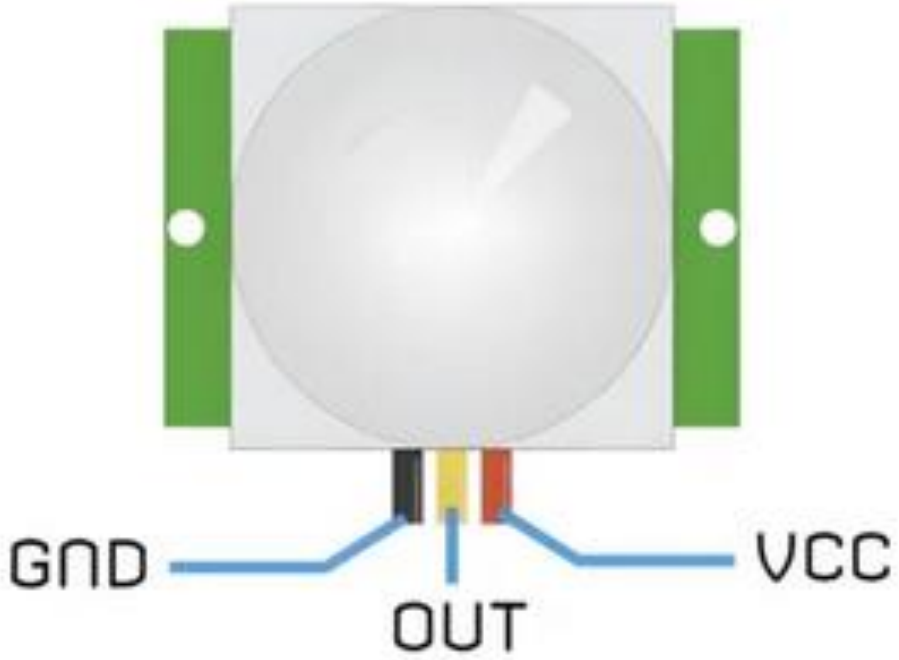
PIR Sensor


Delay Time Adjust
Sensitivity Adjust
Retrigger
Setting
Jumper
BISS0001
PIR Chip
3VDC
Regulator
Protection Diode
Ground
Digital OUT
3-5VDC Power
GND OUT +5V

PIR Sensor

Specification	Typical Range / Details
Operating Voltage	**4.5V to 20V** (5V recommended)
Output Signal	**3.3V digital output** (High when motion detected)
Current Consumption	~65mA (active), **<50µA** (idle)
Detection Range	Up to **7 meters**
Detection Angle	Approximately **110°**
Delay Time	Adjustable from **~0.3 seconds up to 5 minutes**
Sensitivity	Adjustable via potentiometer
Operating Temperature	**-15°C to +70°C**
Trigger Modes	**Repeatable (H)** and **Non-Repeatable (L)**, set by jumper

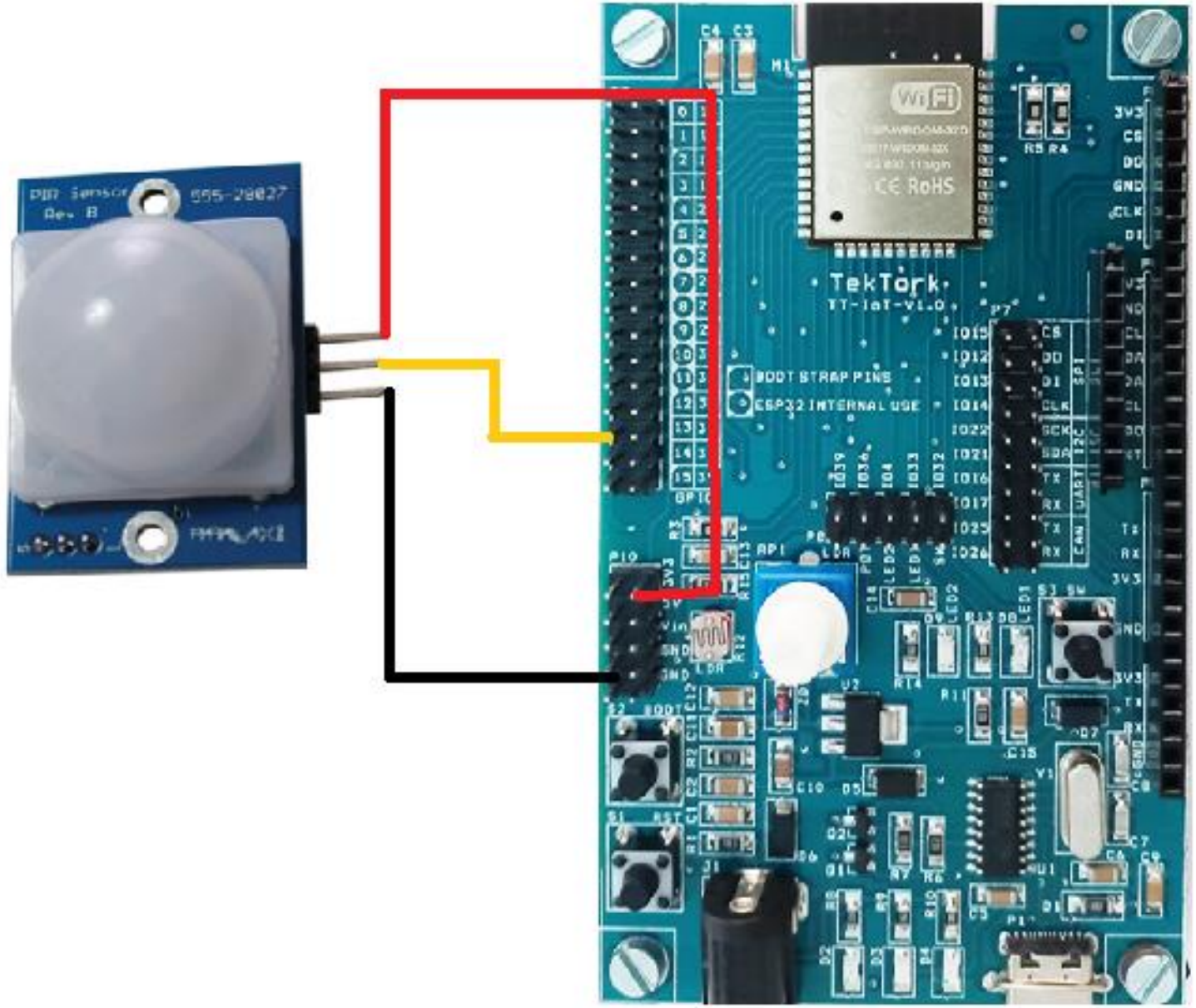
PIR Sensor Pinout



PIR Sensor Pin to ESP32

PIR Pin	ESP32 Pin
VCC	3.3V (or 5V)
OUT	Any GPIO (e.g., GPIO 13)
GND	GND

Wiring Diagram of the PIR Sensor Connected to ESP 32



PIR Sensor Pin to ESP32

PIR Pin	ESP32 Pin
VCC	3.3V (or 5V)
OUT	Any GPIO (e.g., GPIO 13)
GND	GND

PIR Sensor Code

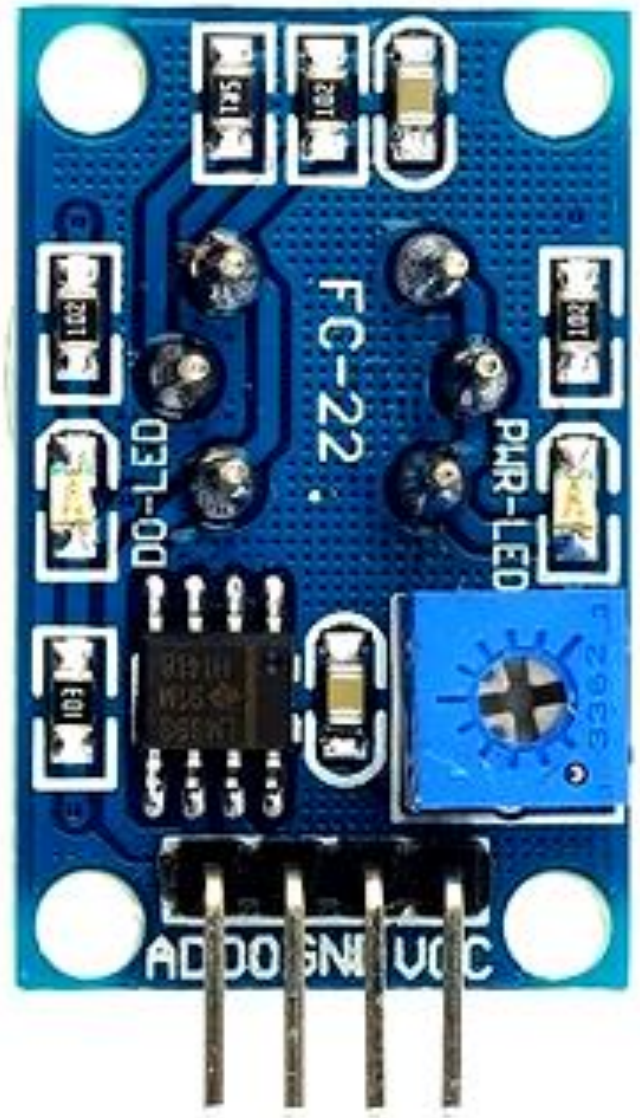
```
// Define the digital pin where the PIR sensor output is connected
const int PIR_SENSOR_OUTPUT_PIN = 13;  /* PIR sensor O/P pin */

int warm_up; // Variable to keep track of warm-up status (flag)

void setup() {
  // Configure the PIR sensor pin as input
  pinMode(PIR_SENSOR_OUTPUT_PIN, INPUT);

  // Start serial communication at 115200 baud rate
  Serial.begin(115200);

  // Notify that sensor is warming up
  Serial.println("Waiting For Power On Warm Up");

  // Wait 20 seconds to let the PIR sensor stabilize (PIR sensors need a warm-up time
for accuracy)
  delay(20000);

  // Notify that PIR sensor is ready after warm-up
  Serial.println("Ready!");
}
```

```
void loop() {
  int sensor_output;  // Variable to store sensor reading
  // Read the current output state from the PIR sensor (HIGH = motion, LOW = no
motion)
  sensor_output = digitalRead(PIR_SENSOR_OUTPUT_PIN);
    if (sensor_output == LOW)  // No motion detected (sensor output is LOW)
  {
    // If warm-up flag is set, print a message once during transition to "no motion"
    if (warm_up == 1)
    {
      Serial.print("Warming Up\n\n");  // Special case message
      warm_up = 0;   // Reset flag
      delay(2000);   // Short delay after transition
    }
    Serial.print("No object in sight\n\n");  // Print when no motion detected
    delay(1000);  // Small delay before next check
  }
    else  // Motion detected (sensor output is HIGH)
  {
    Serial.print("Object detected\n\n");  // Print motion detection message
    warm_up = 1;  // Set flag for transition handling
    delay(1000);  // Small delay before next check
  }
}
```

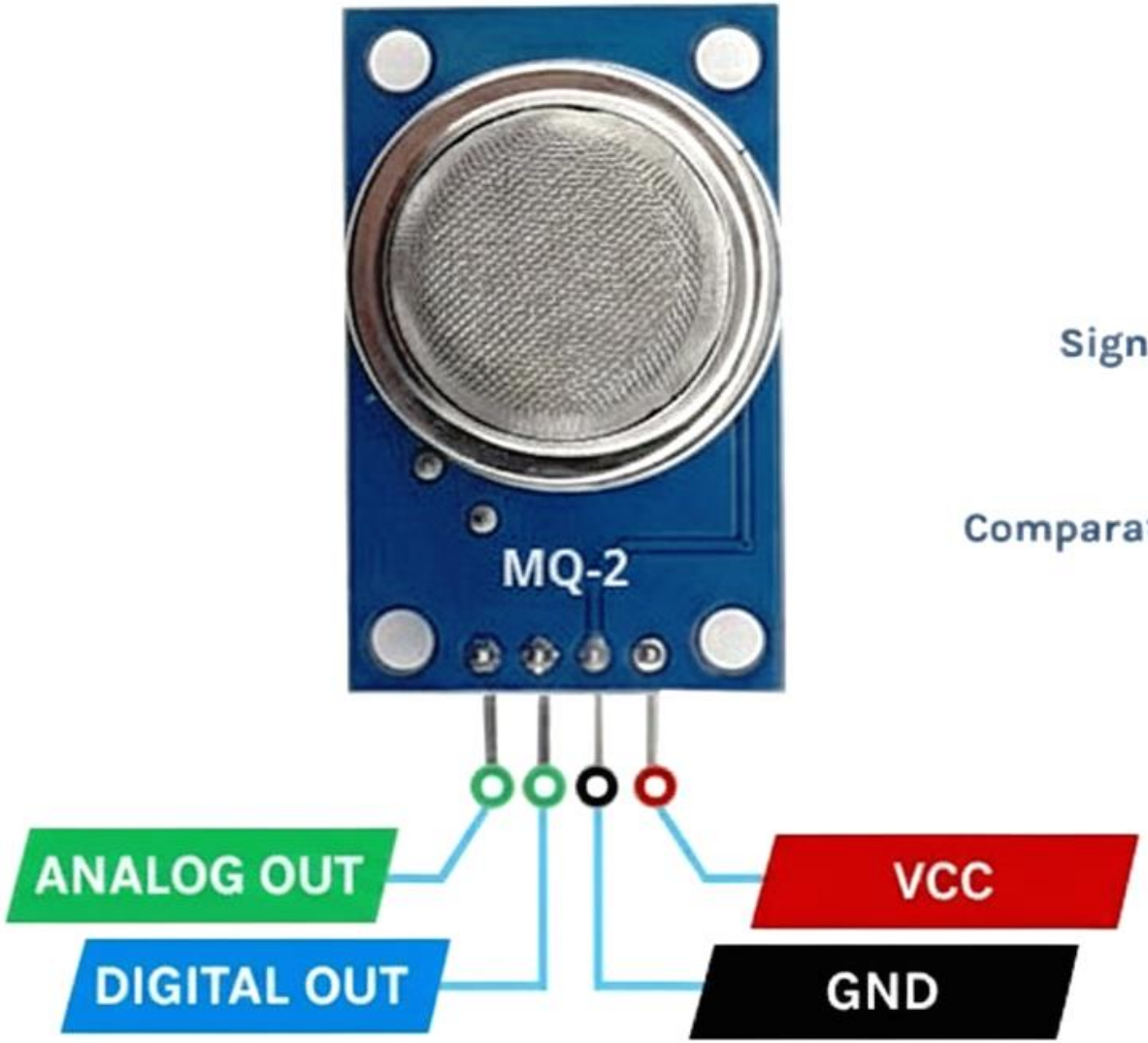
PIR Sensor

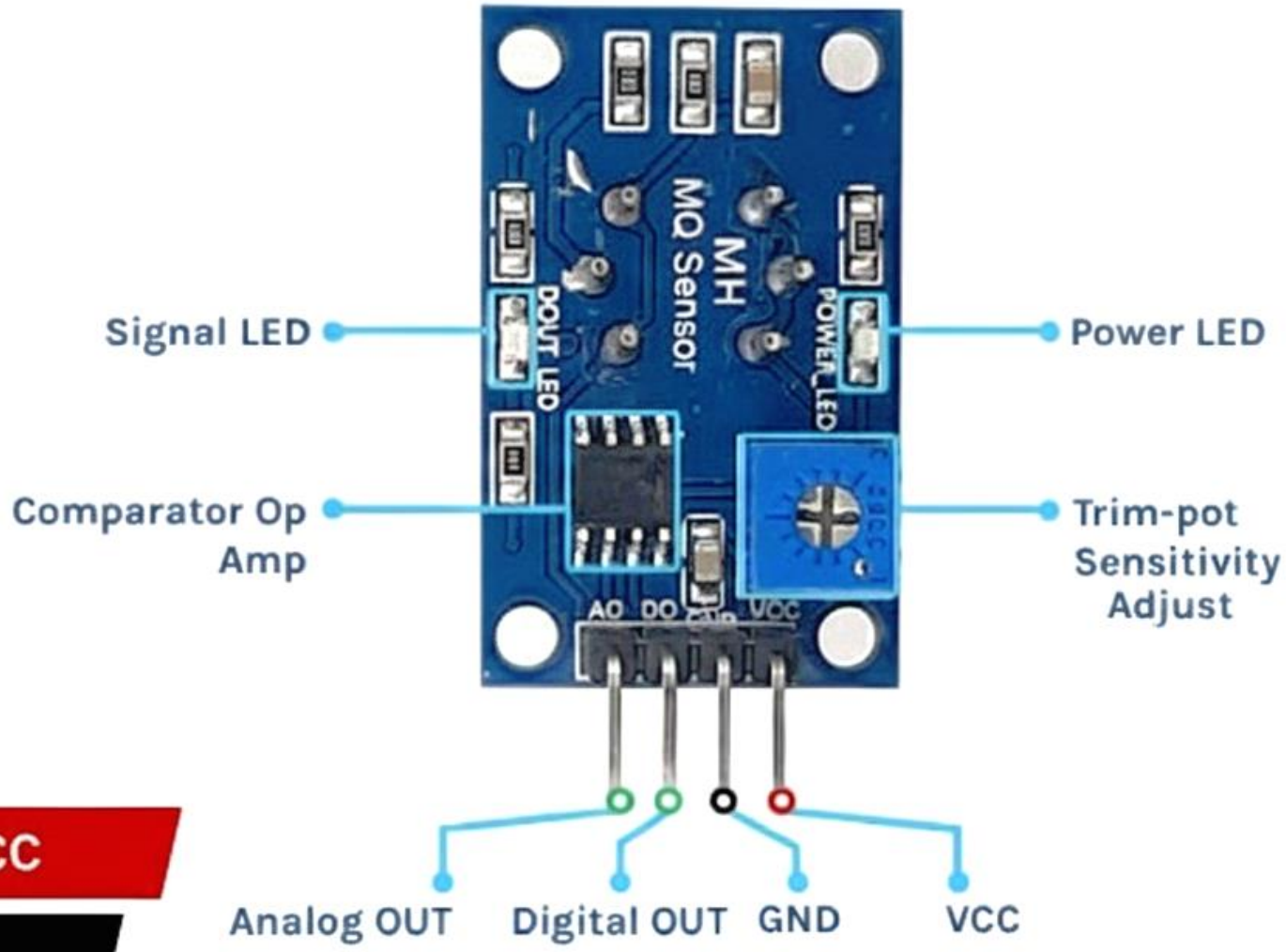

Output
Serial Monitor
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')
No Line Ending
115200 baud
No object in sight
No object in sight
No object in sight
Object detected
Warming Up
Ln 14, Col 38
DOIT ESP32 DEVKIT V1 on COM5
2

MQ2 Gas Sensor

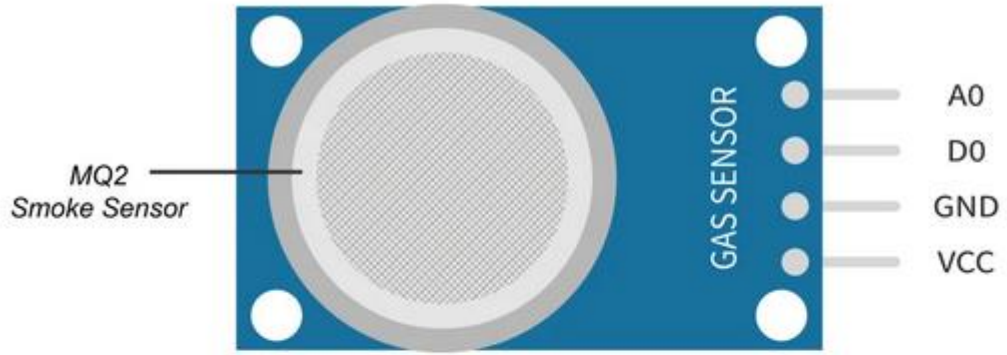


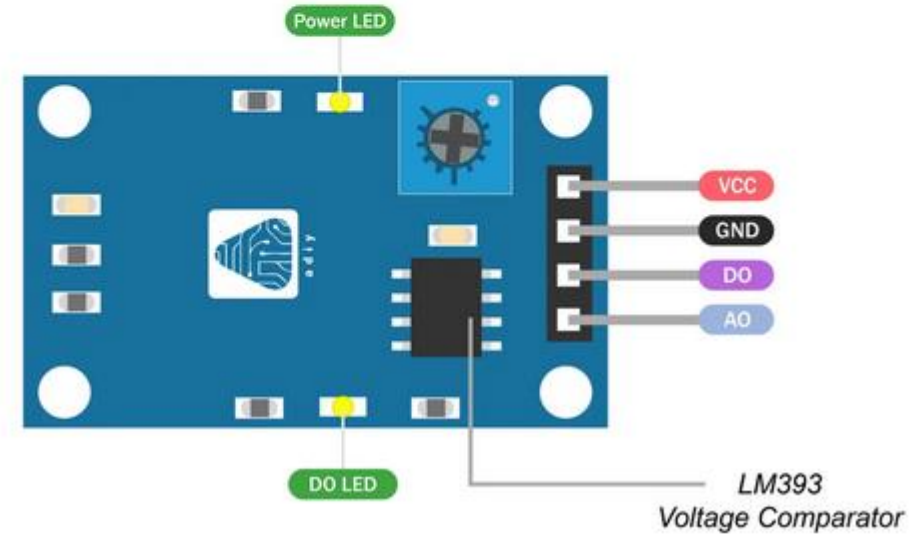
Gas Sensor MQ2





Gas Sensor MQ2 Pin to ESP32





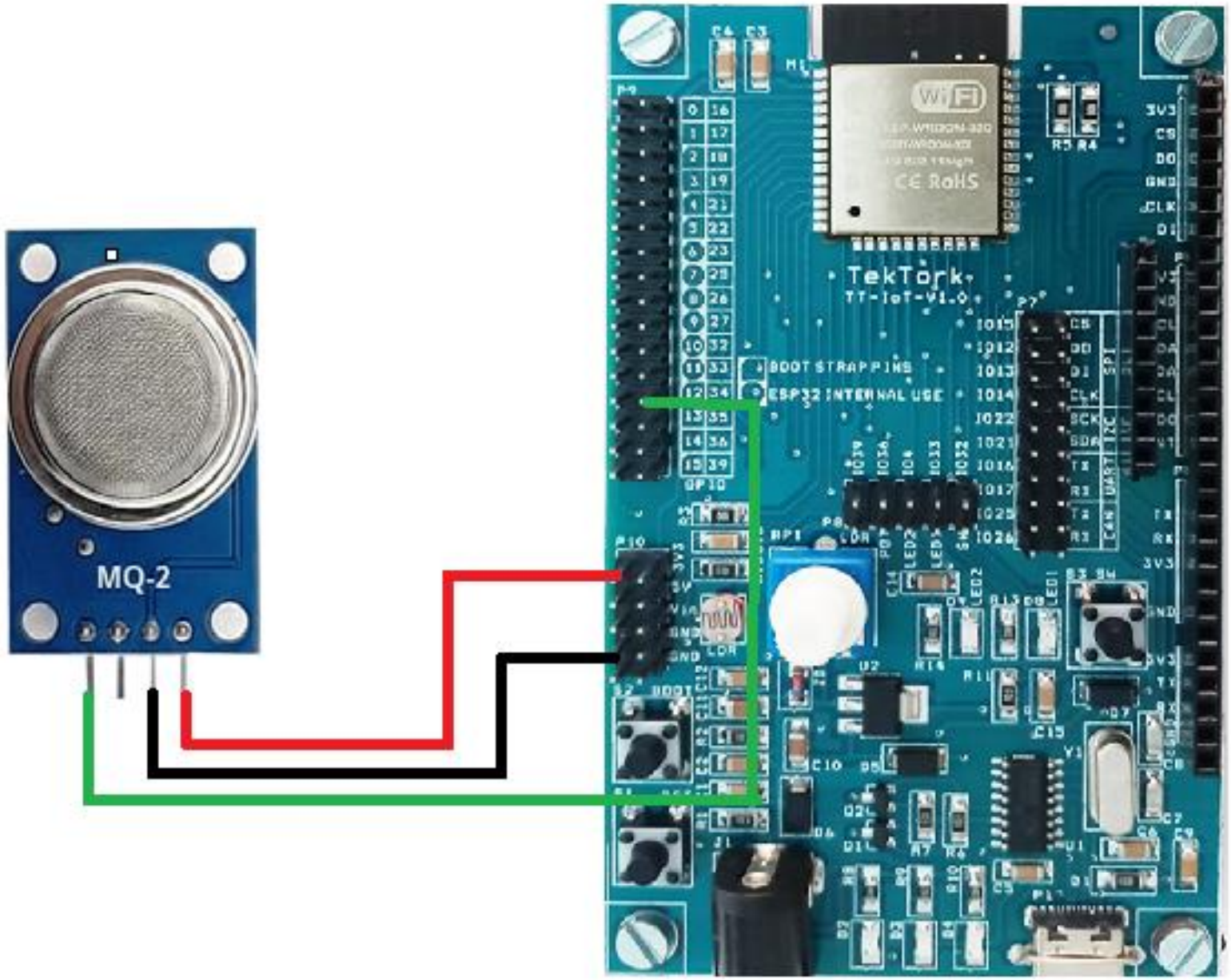
MQ2 Sensor Pin to ESP32

MQ2 Gas Sensor Pin	ESP32 Pin
A0	Analog pins
D0	Digital pins
GND	GND
VCC	3.3V - 5V

Gas Sensor MQ2 Specification

Parameter	Specification
Sensor Type	Semiconductor (SnO_2 sensitive material)
Target Gases	LPG, Propane, Methane, Hydrogen, Smoke, Alcohol, Butane
Operating Voltage	2.5V-5V DC
Heater Voltage (VH)	5V ± 0.2V (for internal heating element)
Load Resistance (RL)	Adjustable (typically 10kΩ)
Heater Resistance (RH)	~33Ω ± 5Ω (at room temperature)
Heater Consumption	≤ 900mW
Preheat Time	20–30 seconds (initial), 24–48 hours recommended for stable calibration
Sensing Resistance (RS)	1 kΩ – 10 kΩ in 200–10000 ppm H_2
Detection Range	200 ppm – 10,000 ppm (varies with gas type)
Output Types	- Analog Output (AO): 0–5V proportional to gas concentration - Digital Output (DO): 0 or 1 (via onboard LM393 comparator & potentiometer)
Response Time	< 10 seconds
Recovery Time	< 30 seconds
Operating Temperature	-20°C to +50°C
Operating Humidity	≤ 95% RH non-condensing

Wiring Diagram of the Gas(MQ2) Sensor Connected to ESP 32 –Analog Input



MQ2 Sensor Pin to ESP32

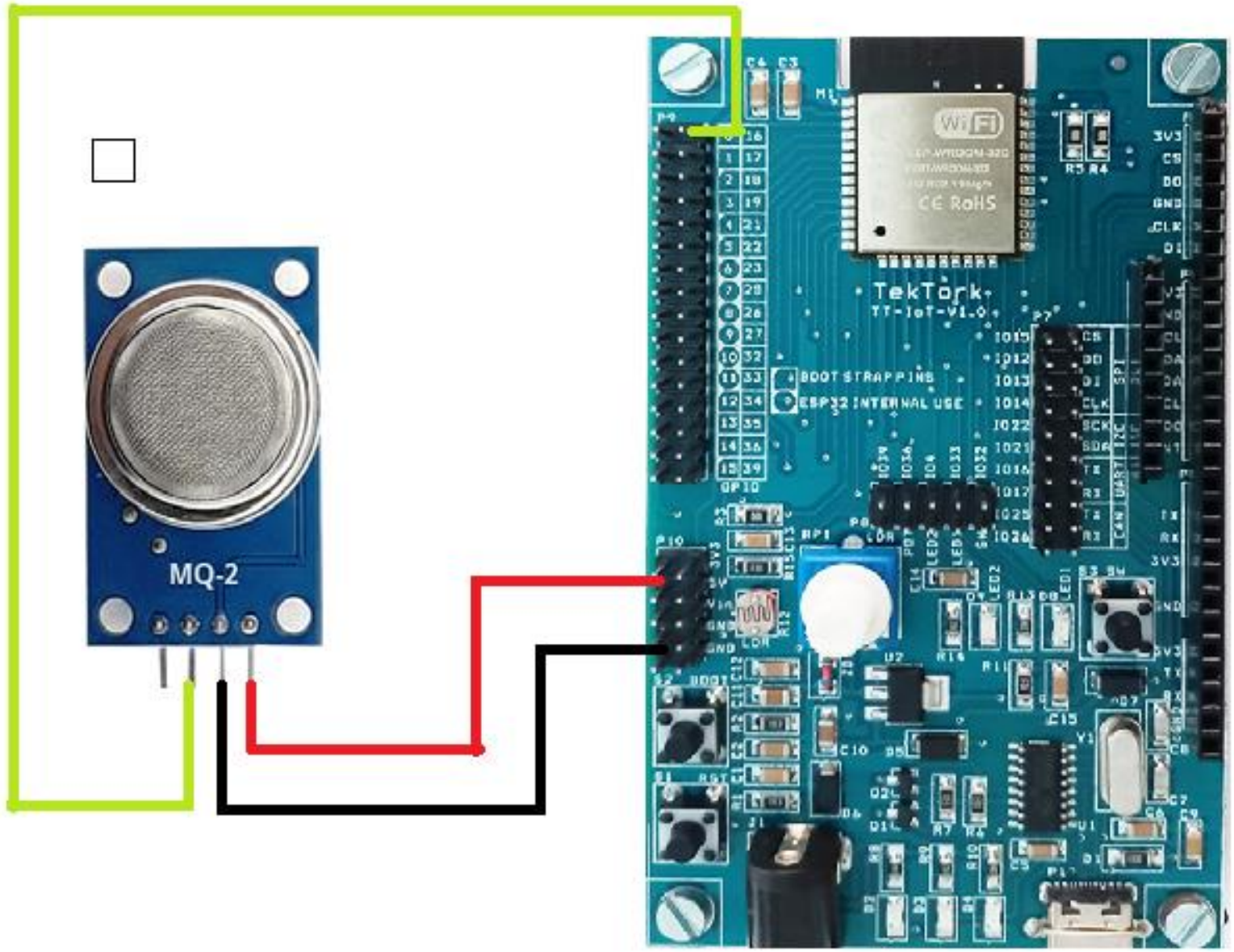
MQ2 Gas Sensor Pin	ESP32 Pin
A0	Analog pins
GND	GND
VCC	3.3V - 5V

Gas Sensor MQ2 Code- Analog

```
#define AO_PIN 34 //Define ESP32 GPIO34 as the pin connected to MQ-2 AO(Analog Output)
void setup() {
  Serial.begin(115200);  // Start Serial Monitor at 9600 baud for debugging/monitoring
 // Configure ESP32 ADC range
 // ADC_11db attenuation allows measurement up to ~3.3V safely (full sensor voltage
range)
  analogSetAttenuation(ADC_11db);
  // MQ-2 sensor requires a short preheating time for the internal heater element
  // This ensures more stable readings before measuring gas concentration
  delay(20000);  // Wait 20 seconds for sensor warm-up
}
void loop() {
  // 1) Read the raw analog value from MQ-2 sensor
  // ESP32 ADC gives values from 0 (0V) to 4095 (~3.3V with 12-bit resolution)
  int gasValue = analogRead(AO_PIN);
  // 2) Print raw sensor value to Serial Monitor
  Serial.print("MQ-2 Sensor AO Value: ");
  Serial.println(gasValue);
  // 3) Delay before the next reading to avoid flooding Serial Monitor
  delay(1000);  // Read AO value every 1 second
}
```

MQ2 – Analog Output

Wiring Diagram of the Gas(MQ2) Sensor Connected to ESP 32 –Digital Input



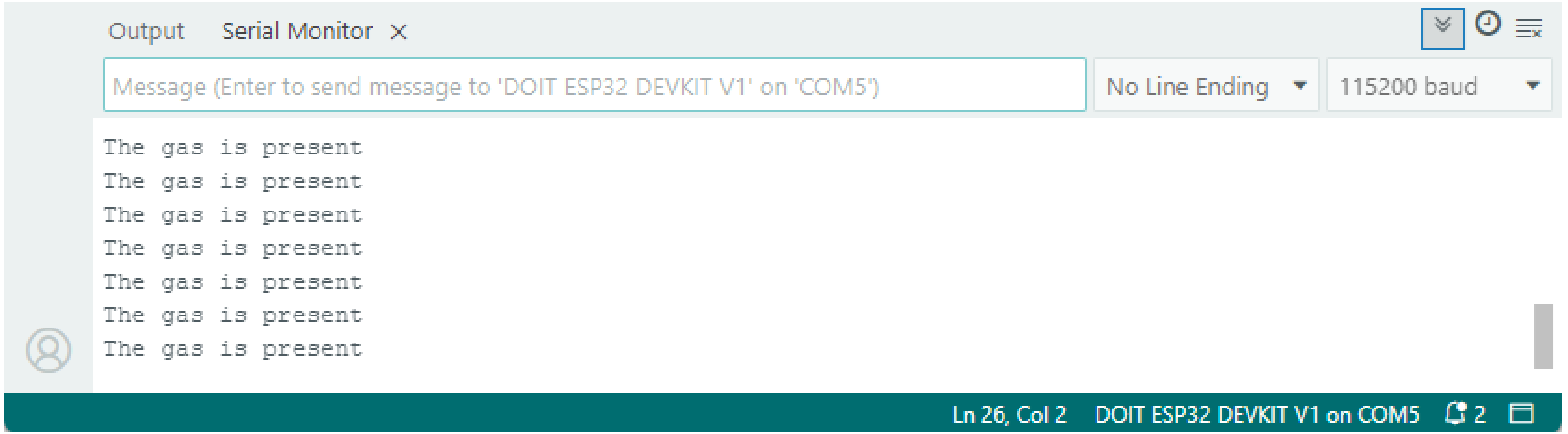
MQ2 Sensor Pin to ESP32

MQ2 Gas Sensor Pin	ESP32 Pin
D0	Digital pins
GND	GND
VCC	3.3V - 5V

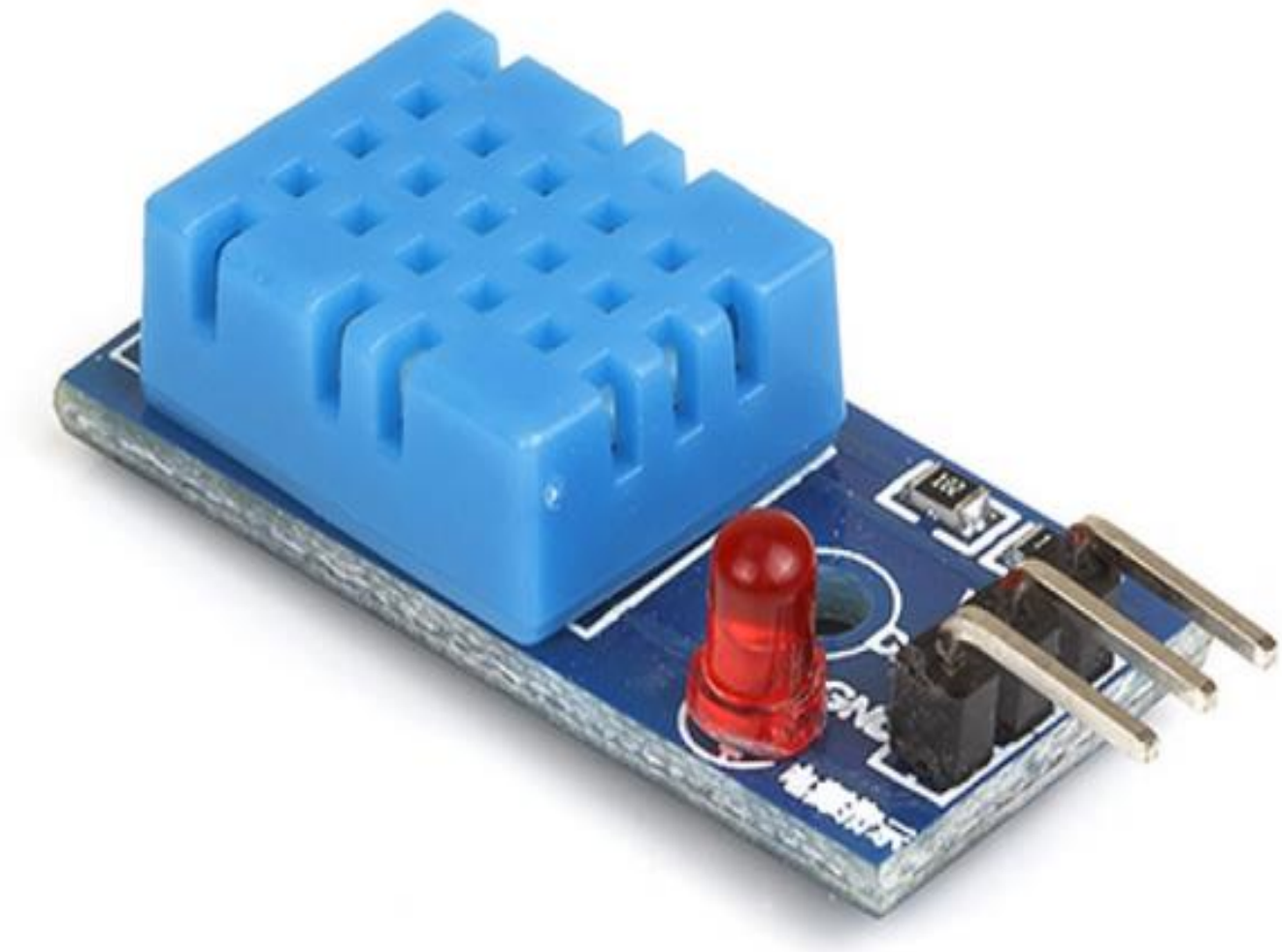
Gas Sensor MQ2 Code - Digital

```
#define DO_PIN 16 // Define GPIO16 on ESP32 as input pin connected to MQ-2
"DO"(digital output)
void setup() {
  Serial.begin(115200); // Start Serial Monitor communication at 115200 baud
  pinMode(DO_PIN, INPUT); //Configure DO pin as input(reads HIGH or LOW from the MQ-2)
  // MQ-2 requires heater warm-up for stable readings
  Serial.println("Warming up the MQ2 sensor");
  delay(20000);  // Wait 20 seconds before starting detection
}
void loop() {
  // Read digital output from MQ-2 module
  // DO pin gives HIGH when gas concentration is below threshold
  // DO pin gives LOW when gas concentration exceeds threshold
  int gasState = digitalRead(DO_PIN);
  // Interpret and print the sensor state
  if (gasState == HIGH)
    Serial.println("The gas is NOT present");    // Safe condition
  else
    Serial.println("The gas is present");       // Gas detected above threshold
  // 3) No extra delay here (loop refreshes as fast as possible)
  // You may add delay(500-1000) if you want slower serial updates
}
```

MQ2 – Digital Output



DHT 11



DHT11 Pinout and Specification

DHT11 4 Pin to ESP32

DHT11	ESP32
VCC	3.3 V
GND	GND
NC	NC
OUT	GPIO 13

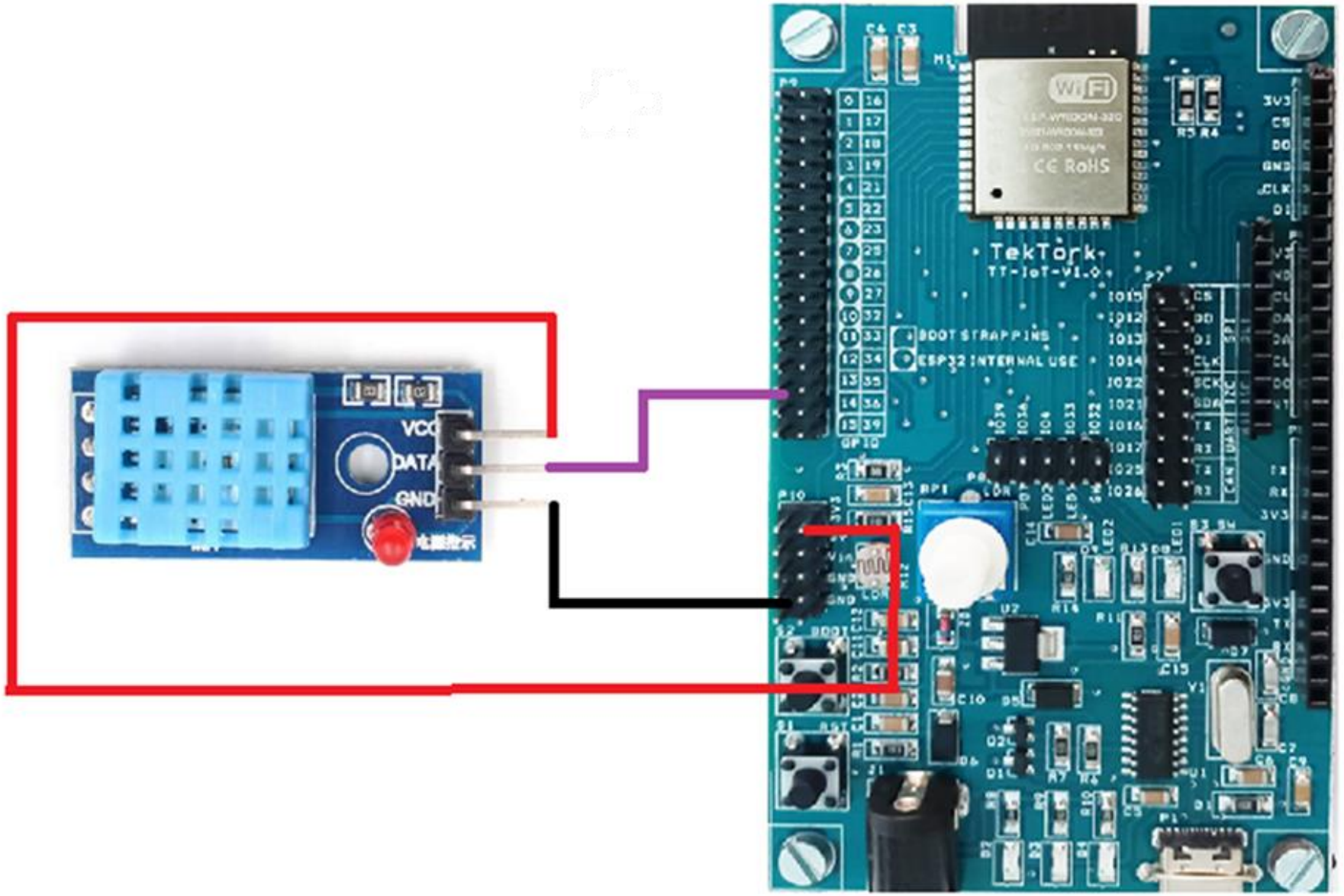
DHT11 3 Pin to ESP32

DHT11	ESP32
VCC	3.3 V
GND	GND
OUT	GPIO 13

DHT11 Specification

Parameter	Value
Sensor Type	Digital temperature & humidity sensor
Operating Voltage	3.3V – 5.5V
Max Current	2.5 mA (during data conversion)
Temperature Range	0 °C to 50 °C
Temperature Accuracy	±2 °C
Humidity Range	20% – 90% RH (non-condensing)
Humidity Accuracy	±5% RH
Resolution (Temp)	1 °C
Resolution (Humidity)	1 %RH
Sampling Rate	1 Hz (1 reading per second)
Output Signal	Digital, single-wire protocol
Dimensions (module)	~15.5mm × 12mm × 5.5mm

Wiring Diagram of the DHT11 Sensor Connected to ESP 32

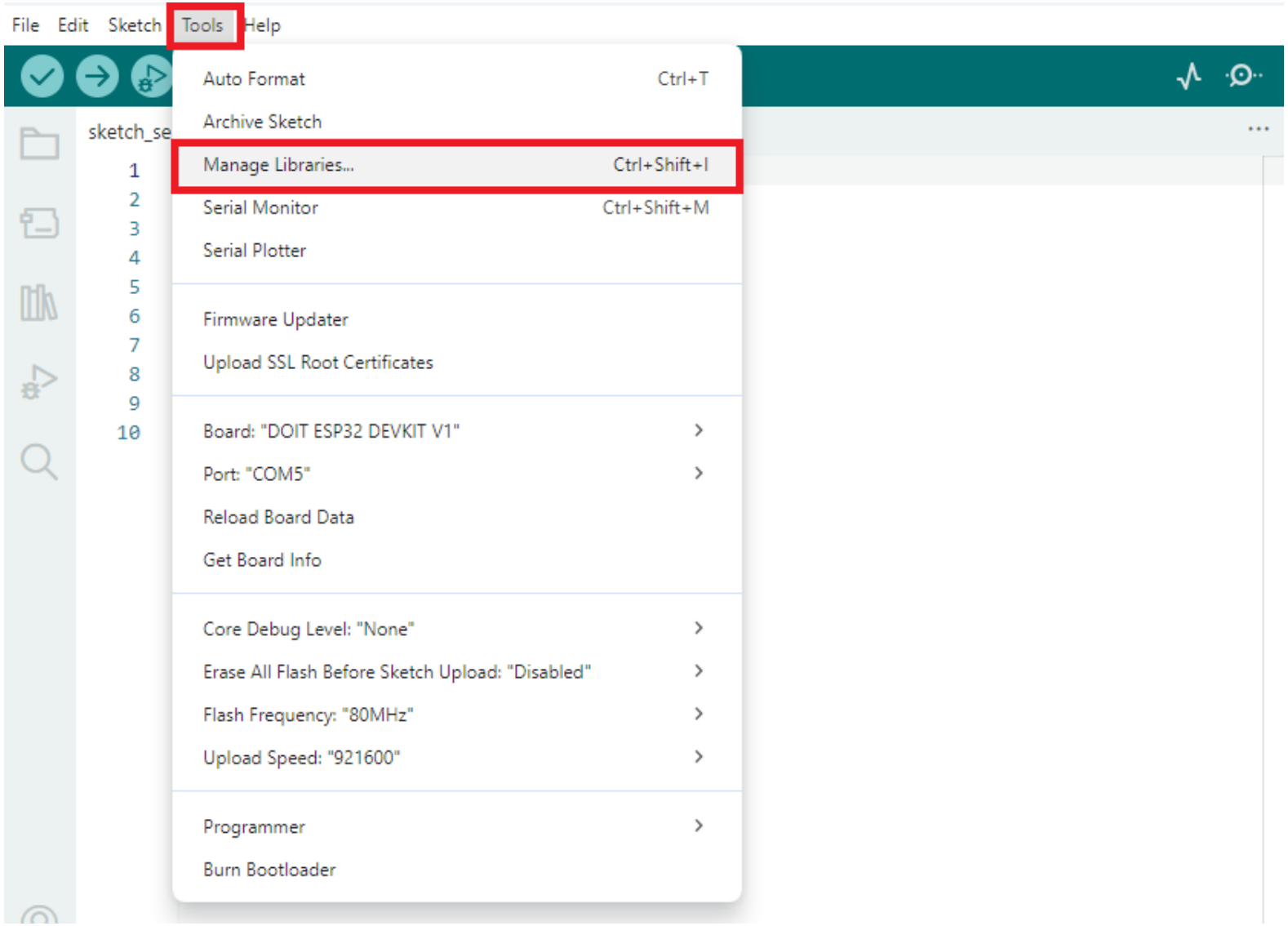


DHT11 3 Pin to ESP32

DHT11	ESP32
VCC	3.3 V
GND	GND
OUT	GPIO 13

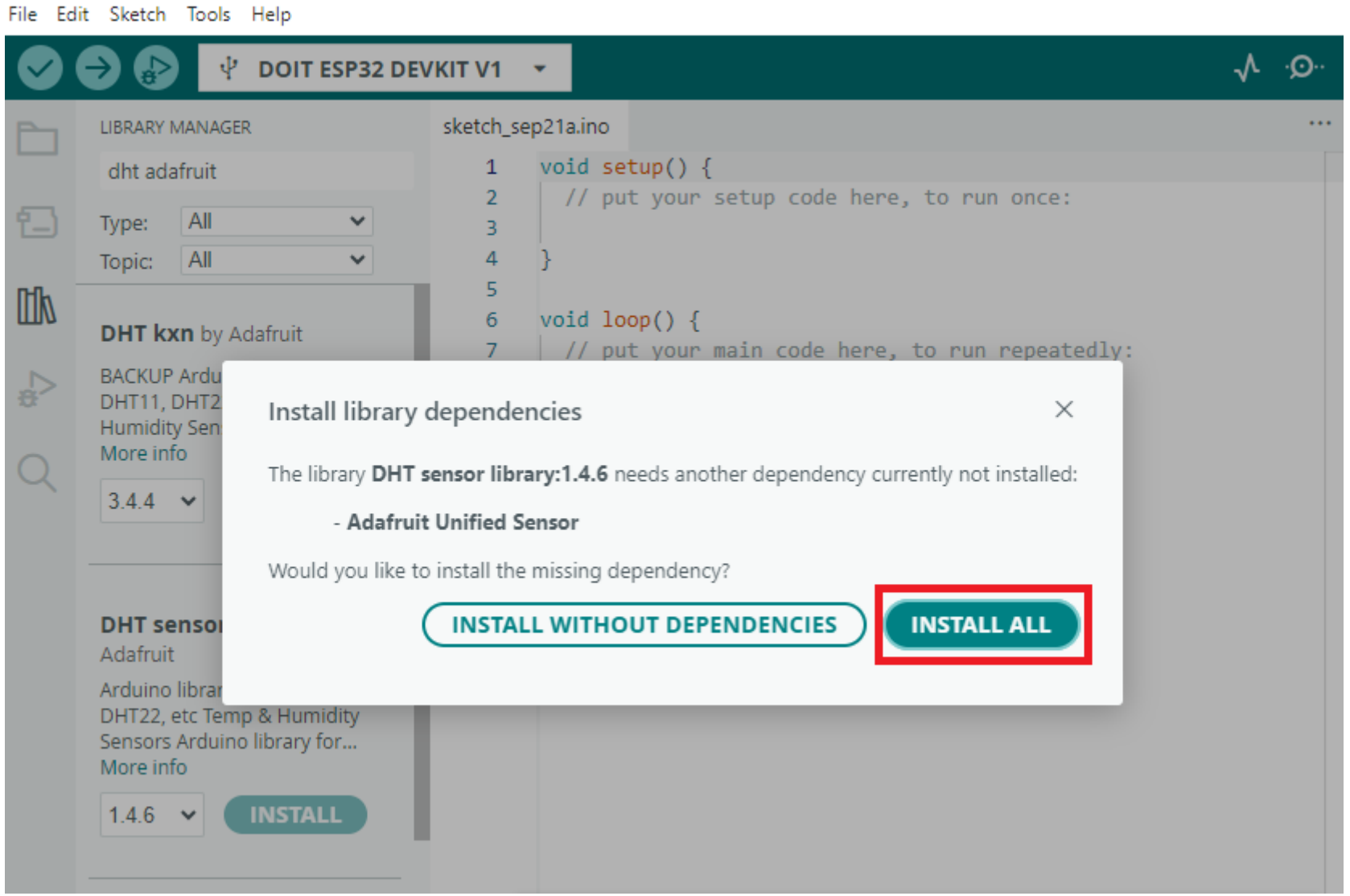
Library Installation

- In the Arduino IDE, go to the Tools menu, then select Manage Libraries... to open the Library Manager.

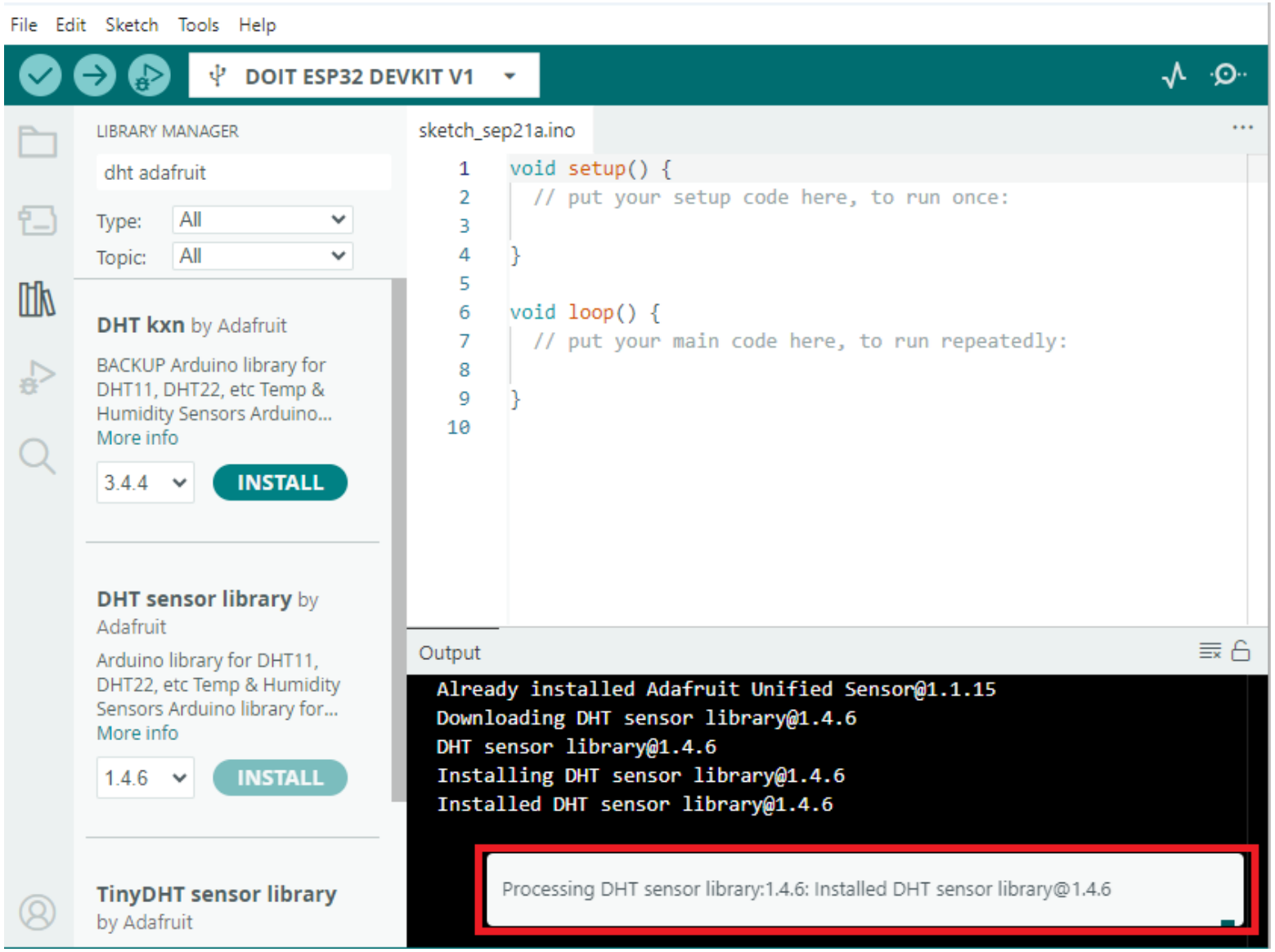


- In the Library Manager search box, type DHT11 Adafruit, then select the **Adafruit DHT Sensor Library** and click Install

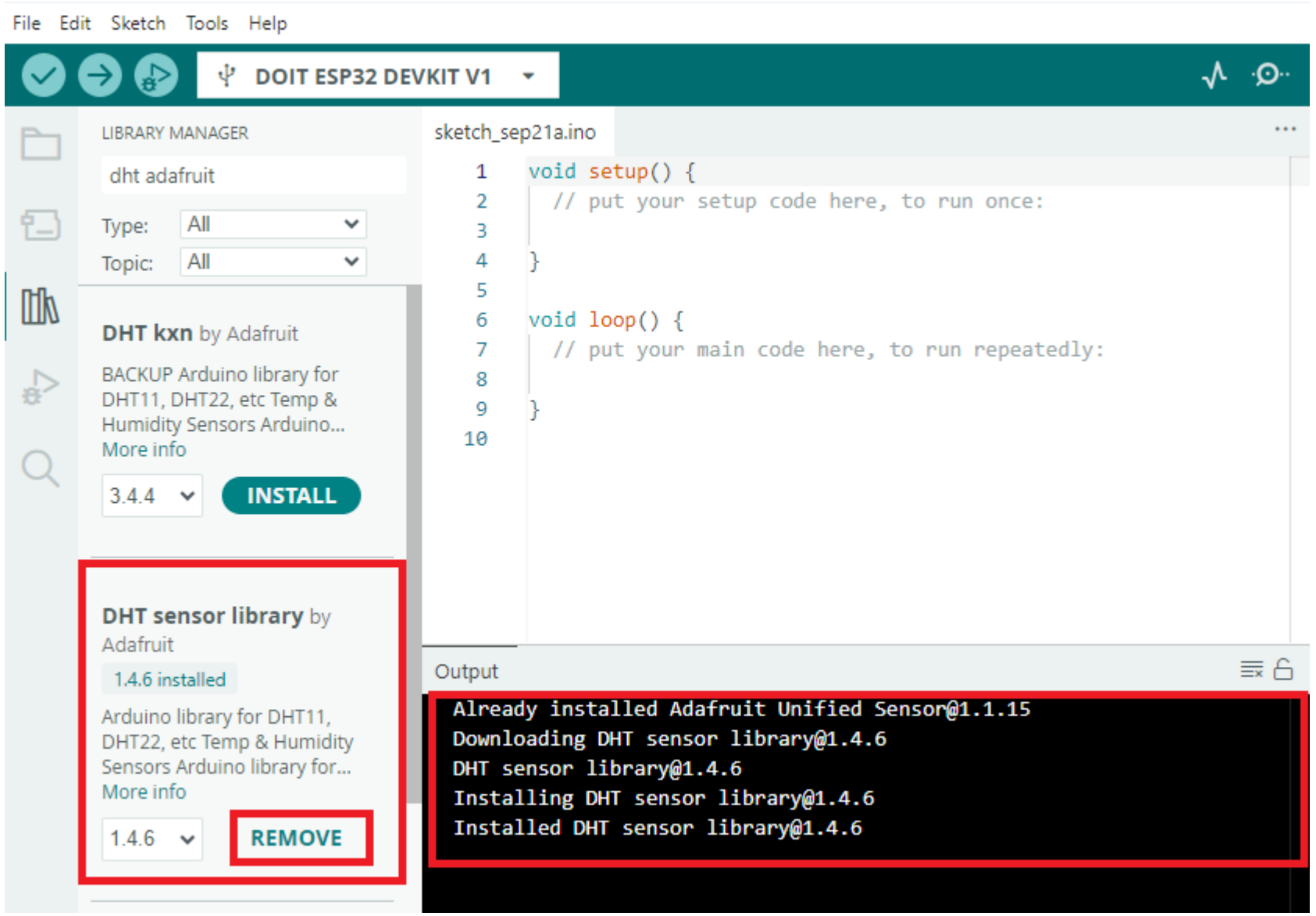
- After clicking the Install button, the library dependency installation window appears. Select the Install All option to install all required dependencies.



- The DHT11 library installation process is in progress. Wait until the installation completes, once finished, you can include the library in your Arduino sketch and start using the DHT11 sensor



- After the installation is complete, the Install button will change to Remove, and the output window will display the installation details.



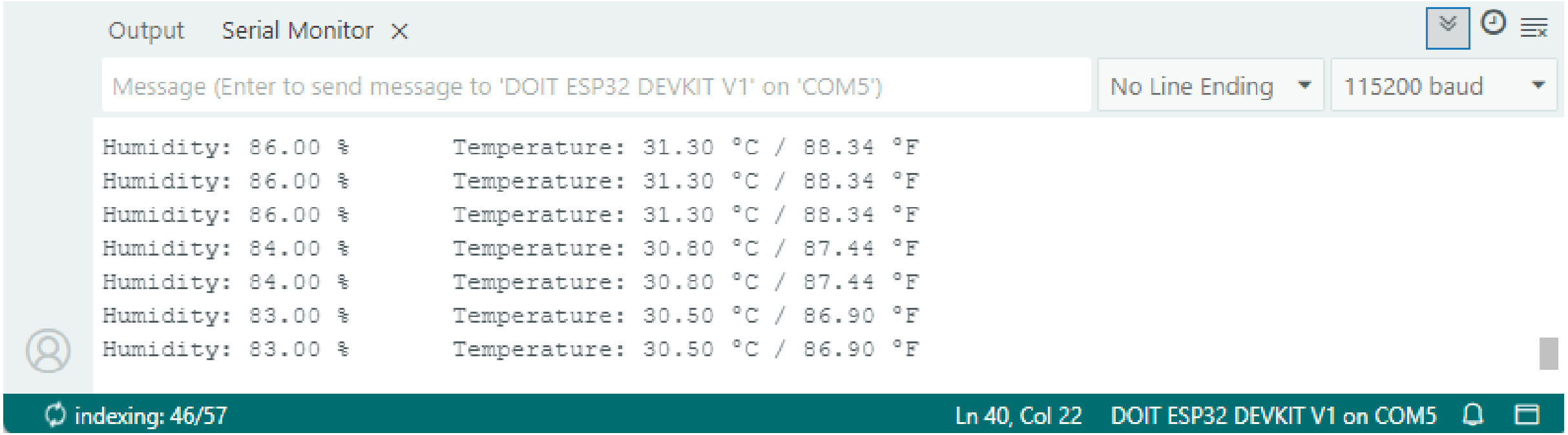
DHT11 Code

```
#include "DHT.h"                // Include the DHT sensor library
#define DHTPIN 27               // GPIO pin where DHT11 data pin is connected
#define DHTTYPE DHT11           // Define the sensor type (here: DHT11)

DHT dht(DHTPIN, DHTTYPE);// Create a DHT object using the defined pin and sensor type

void setup() {
  Serial.begin(115200);     // Initialize serial communication at 115200 baud
  dht.begin();              // Initialize the DHT11 sensor
}
void loop() {
  // Reading temperature and humidity takes about 250ms
  float humidity = dht.readHumidity();            // Read relative humidity (%)
  float temperatureC = dht.readTemperature();     // Read temperature in Celsius
  float temperatureF = dht.readTemperature(true); // Read temperature in Fahrenheit
  // Check if any of the readings failed (NaN = Not a Number)
  if (isnan(humidity) || isnan(temperatureC)) {
    Serial.println("Failed to read from DHT sensor!"); // Print error if sensor not
responding
    return; // Exit loop iteration and try again in next cycle
  }
```

DHT11 Code Cond…

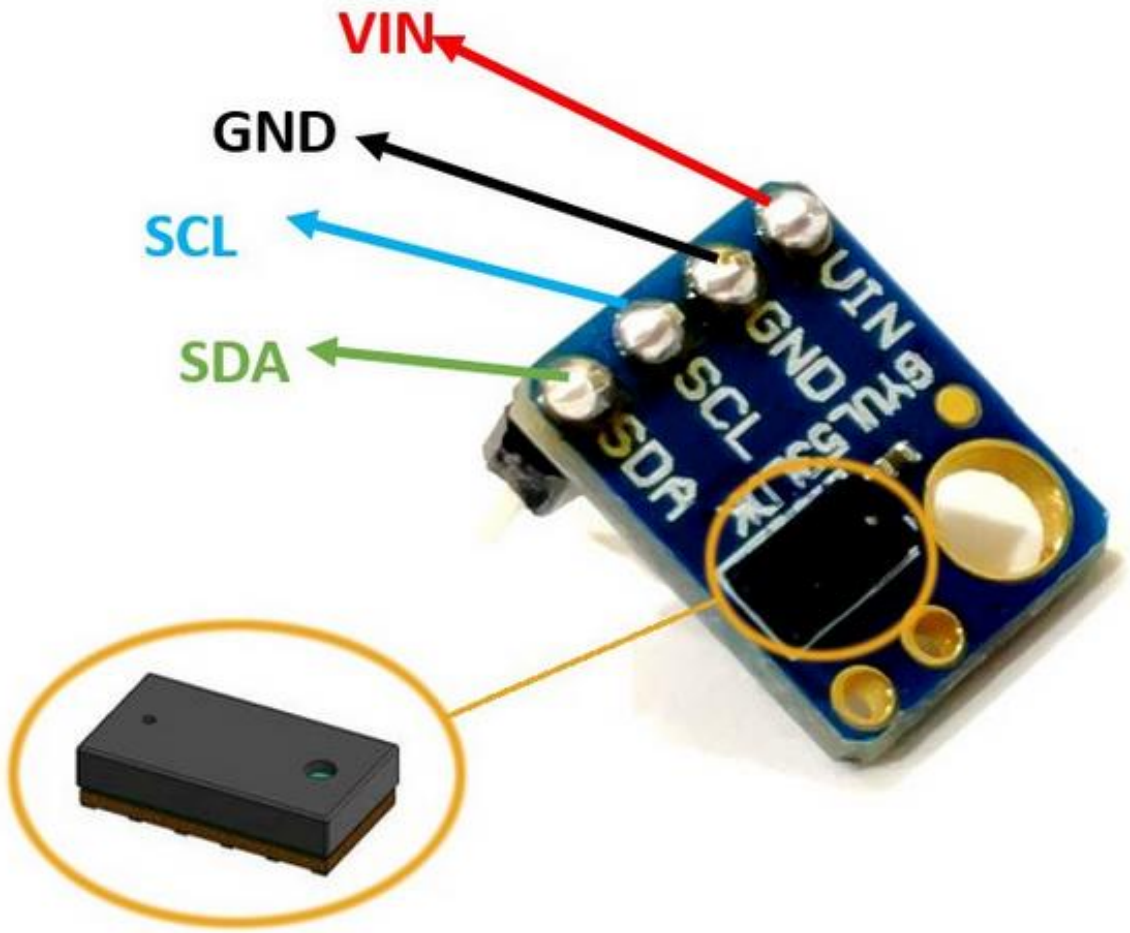
```
// Print the measured values to Serial Monitor
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print(" %\t");                    // Print with percentage symbol
  Serial.print("Temperature: ");
  Serial.print(temperatureC);
  Serial.print(" °C / ");
  Serial.print(temperatureF);
  Serial.println(" °F");
  delay(1000);   // DHT11 sensor requires a 1-second delay between readings
}
```

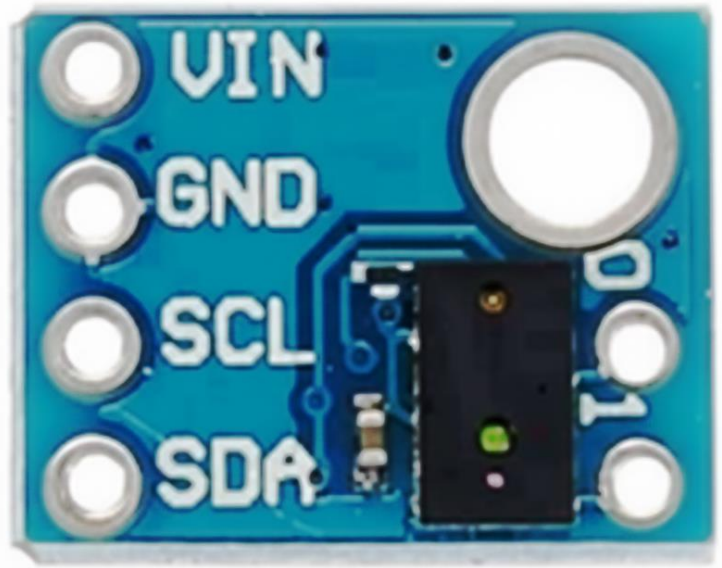
DHT11 Output



VL53LOX Sensor

VL53LOX Sensor





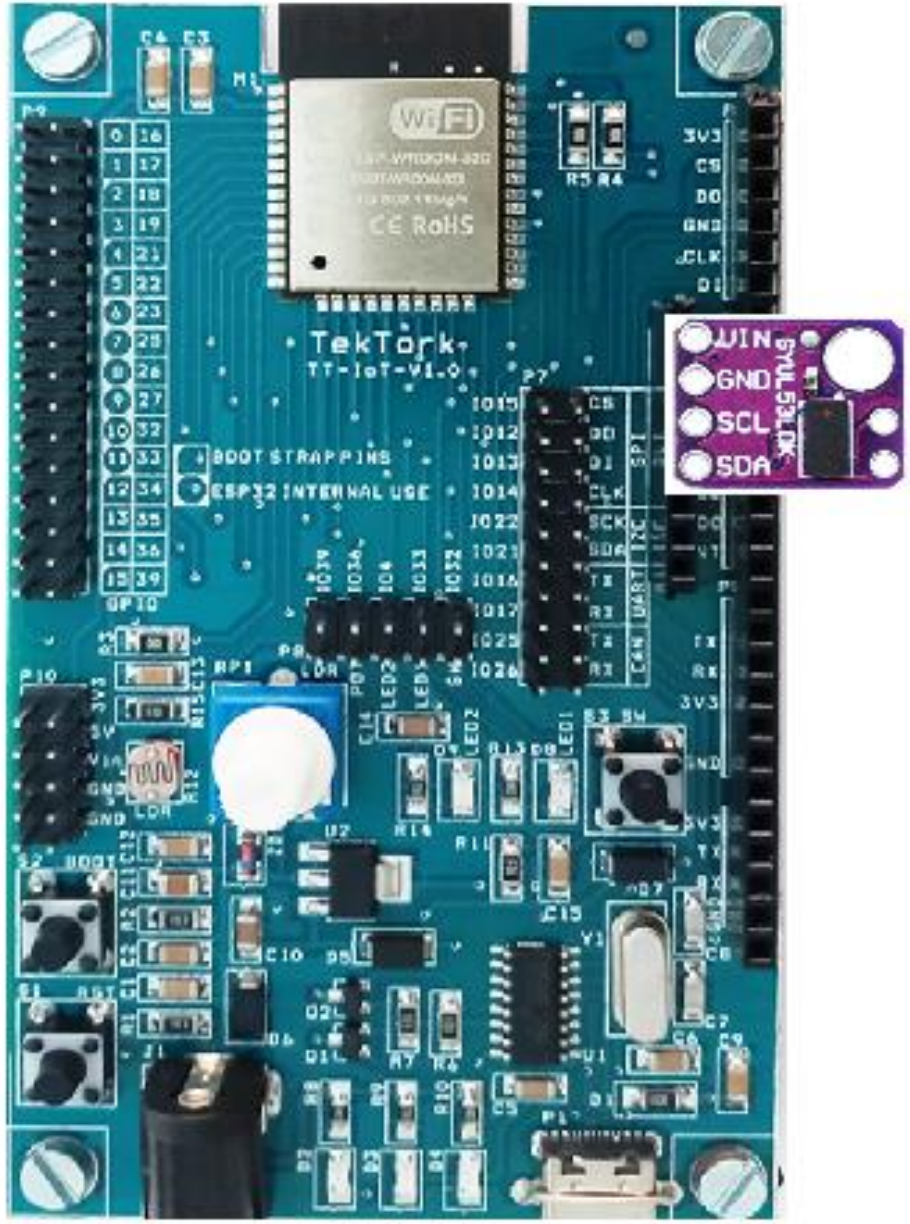
VL53L0X Pin to ESP32

VL53L0X Pin	ESP32 Pin
VCC	3.3V or 5V
GND	GND
SDA	GPIO 21 (default I2C data)
SCL	GPIO 22 (default I2C clock)

VL53LOX Sensor

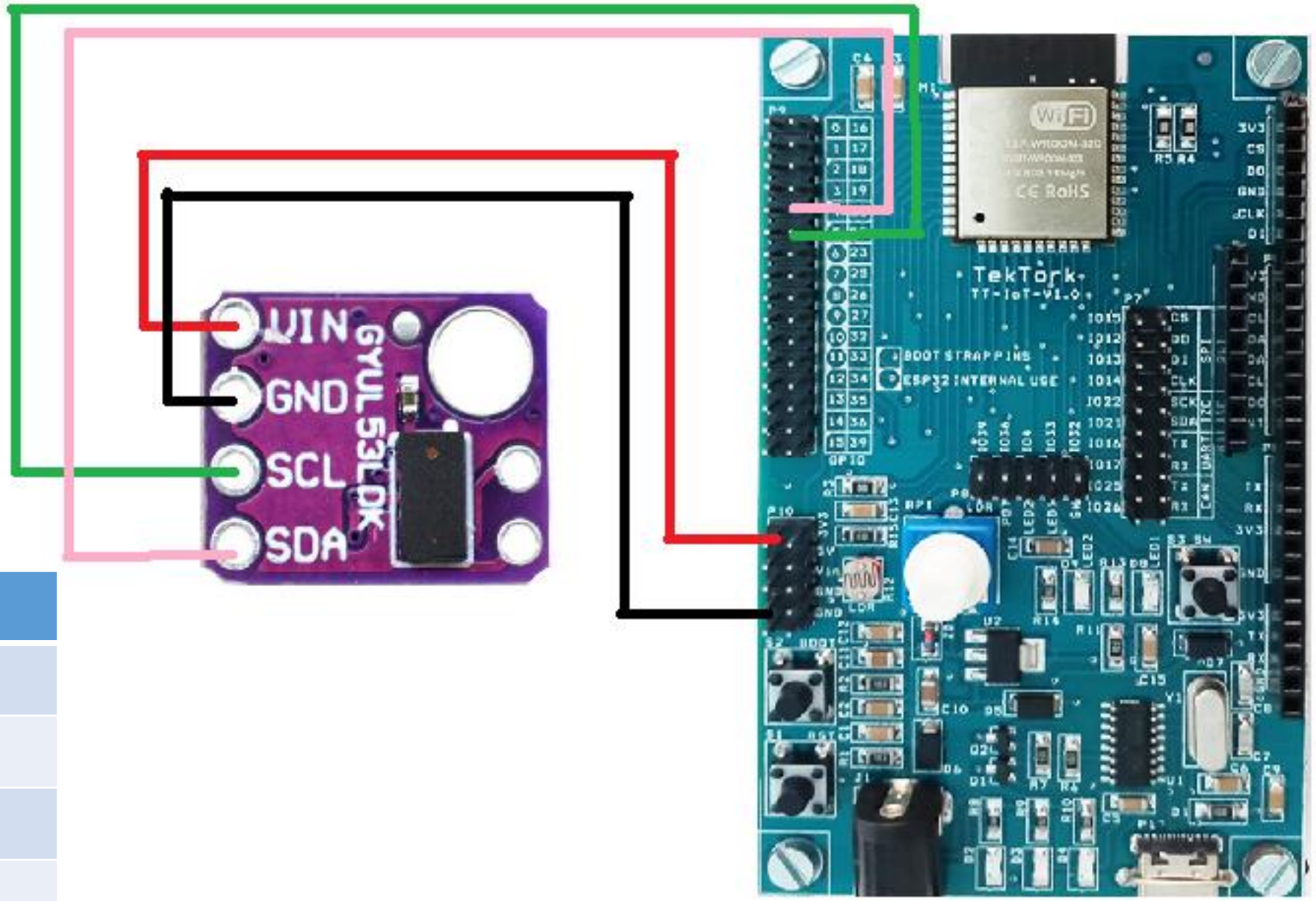
Specification	Details
Technology	Time-of-Flight (ToF), laser ranging
Measuring Range	Default: ~30-50 mm to 1000-1200 mm Long Range Mode: Up to 2 meters (on reflective surfaces)
Accuracy	Ranging accuracy from ±3% to over ±10%, depending on conditions
Resolution	1 mm
Field of View	~35 degrees (narrow cone)
Interface	I^2C (I2C)
I^2C Address	Default 7-bit address: 0x29 (or 0101001b)
Operating Voltage	2.6 V to 5.5 V (for breakout boards with regulators)
Supply Current	Typical: ~20 mA (peaks up to 40 mA)
Laser Wavelength	940 nm infrared VCSEL

Wiring Diagram of the VL53L0X Sensor Connected to ESP 32



Note: Insert the jumper into the GPIO21 and GPIO22 pin headers (P7) of the IoT kit. These are the default I²C Data (SDA) and I²C Clock (SCL) pins, respectively

Wiring Diagram of the VL53L0X Sensor Connected to ESP 32

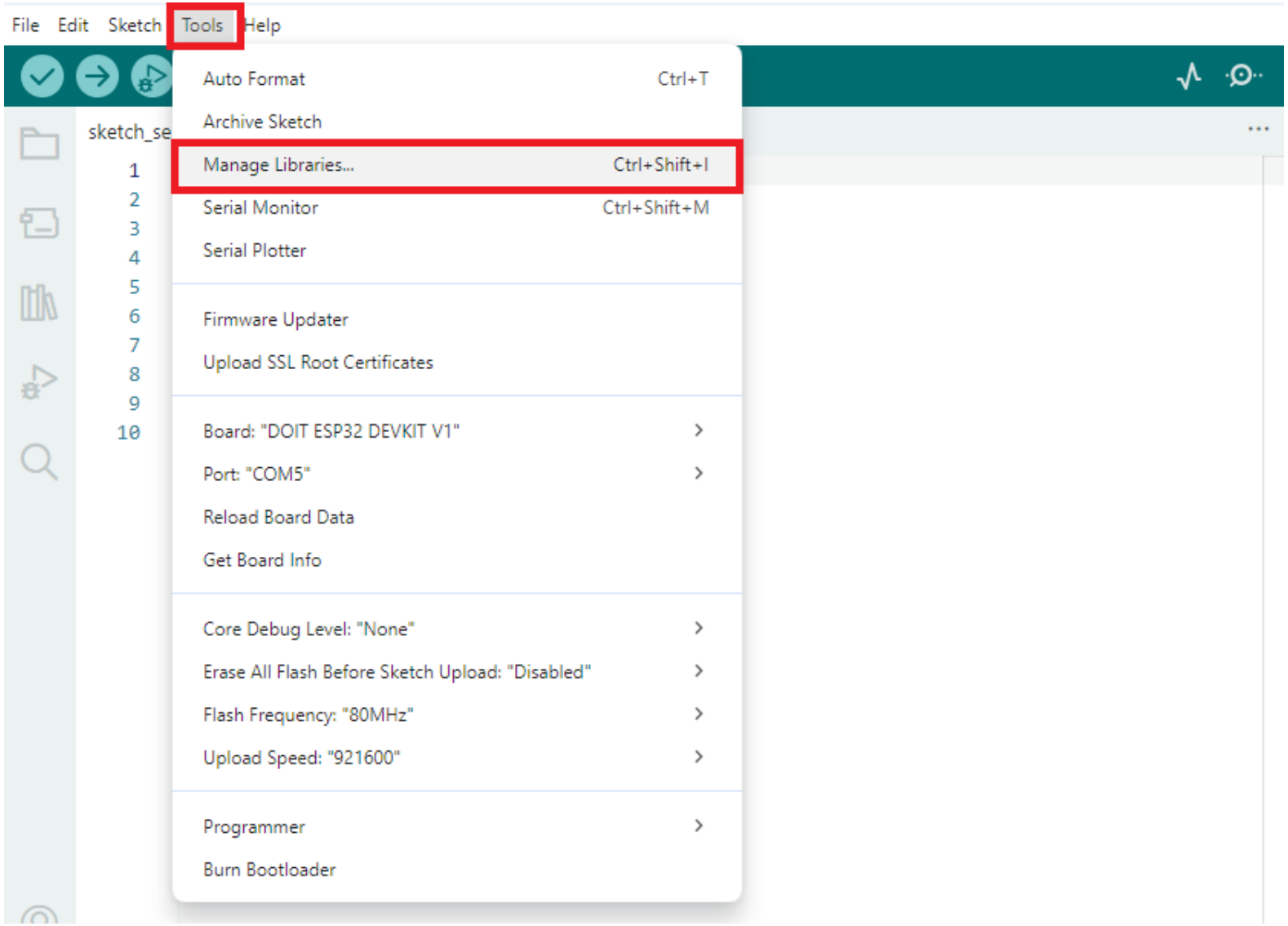


VL53L0X Pin to ESP32

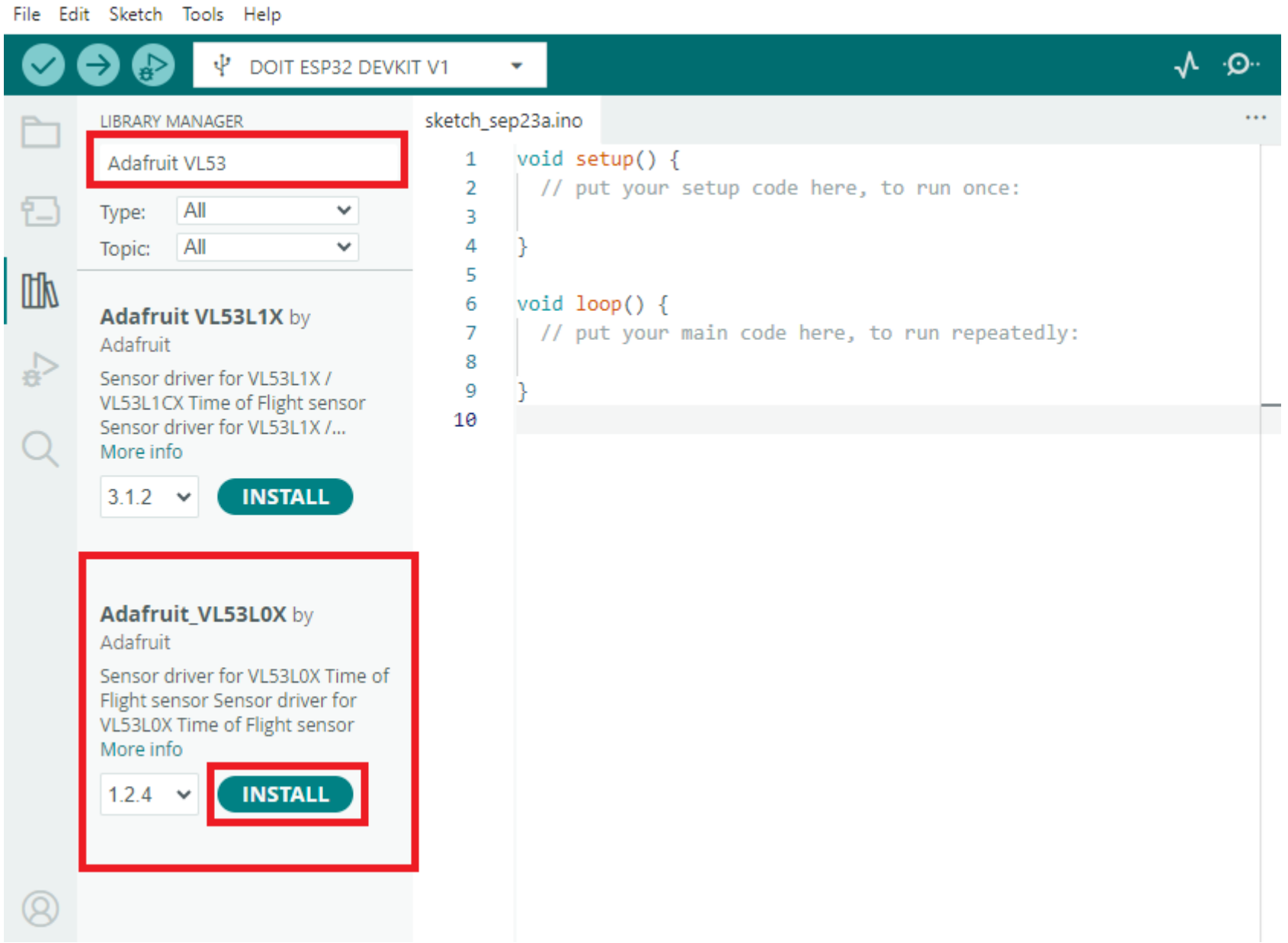
VL53L0X Pin	ESP32 Pin
VCC	3.3V or 5V
GND	GND
SDA	GPIO 21
SCL	GPIO 22

Library Installation

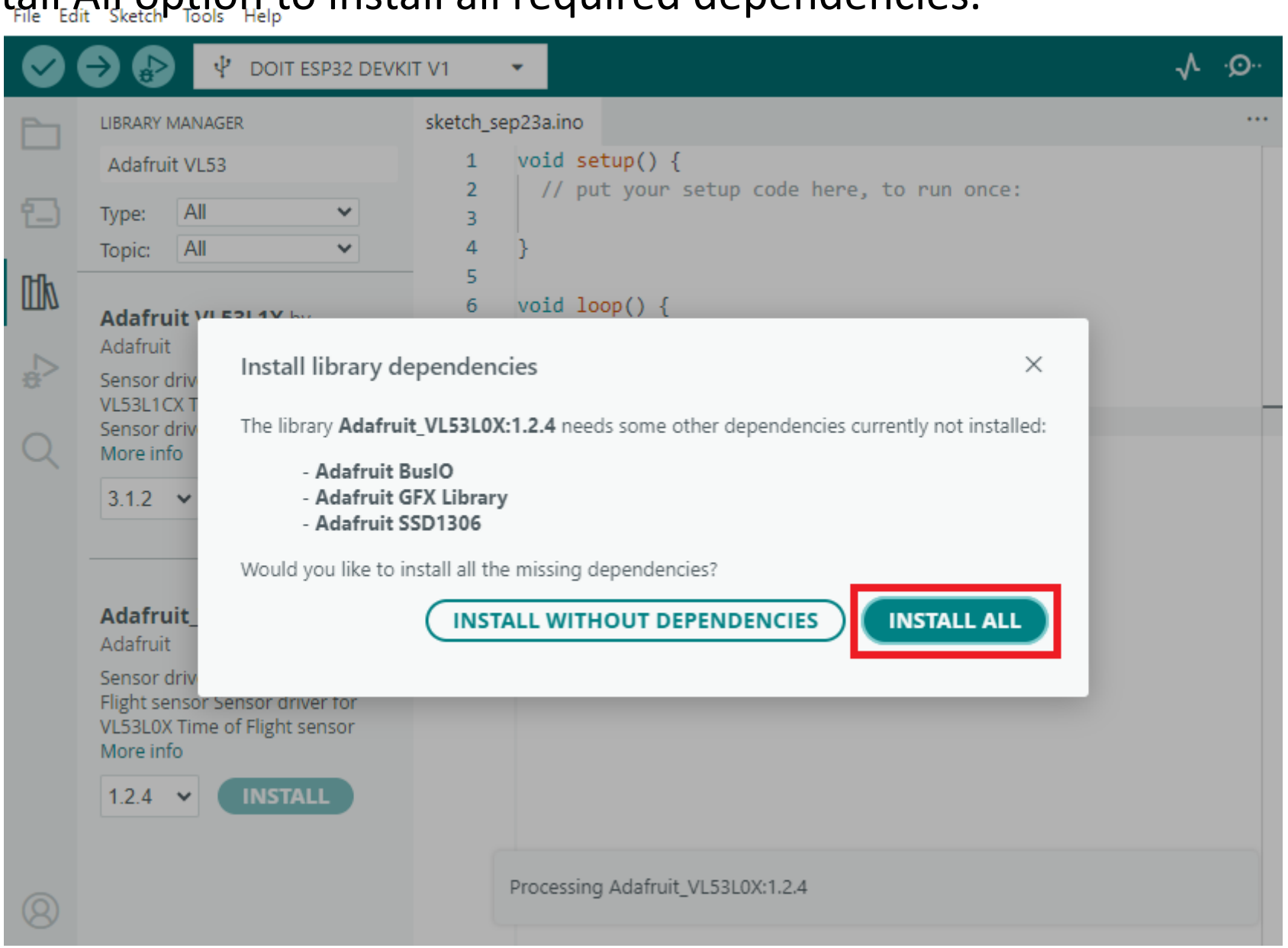
- In the Arduino IDE, go to the Tools menu, then select Manage Libraries... to open the Library Manager.



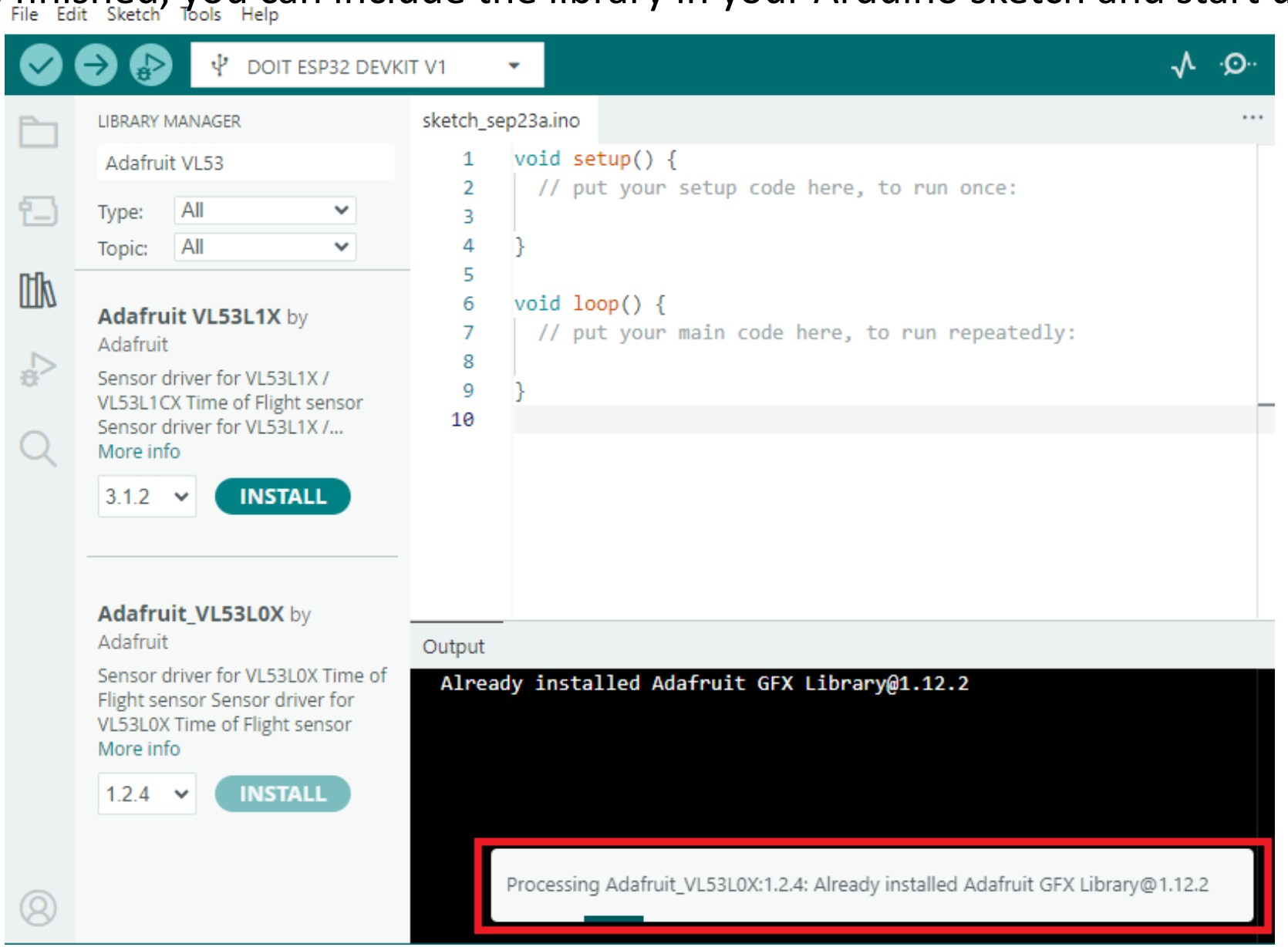
- In the Library Manager search box, type **AdafruitVL53**, then select the **Adafruit VL53L0XLibrary** and click Install

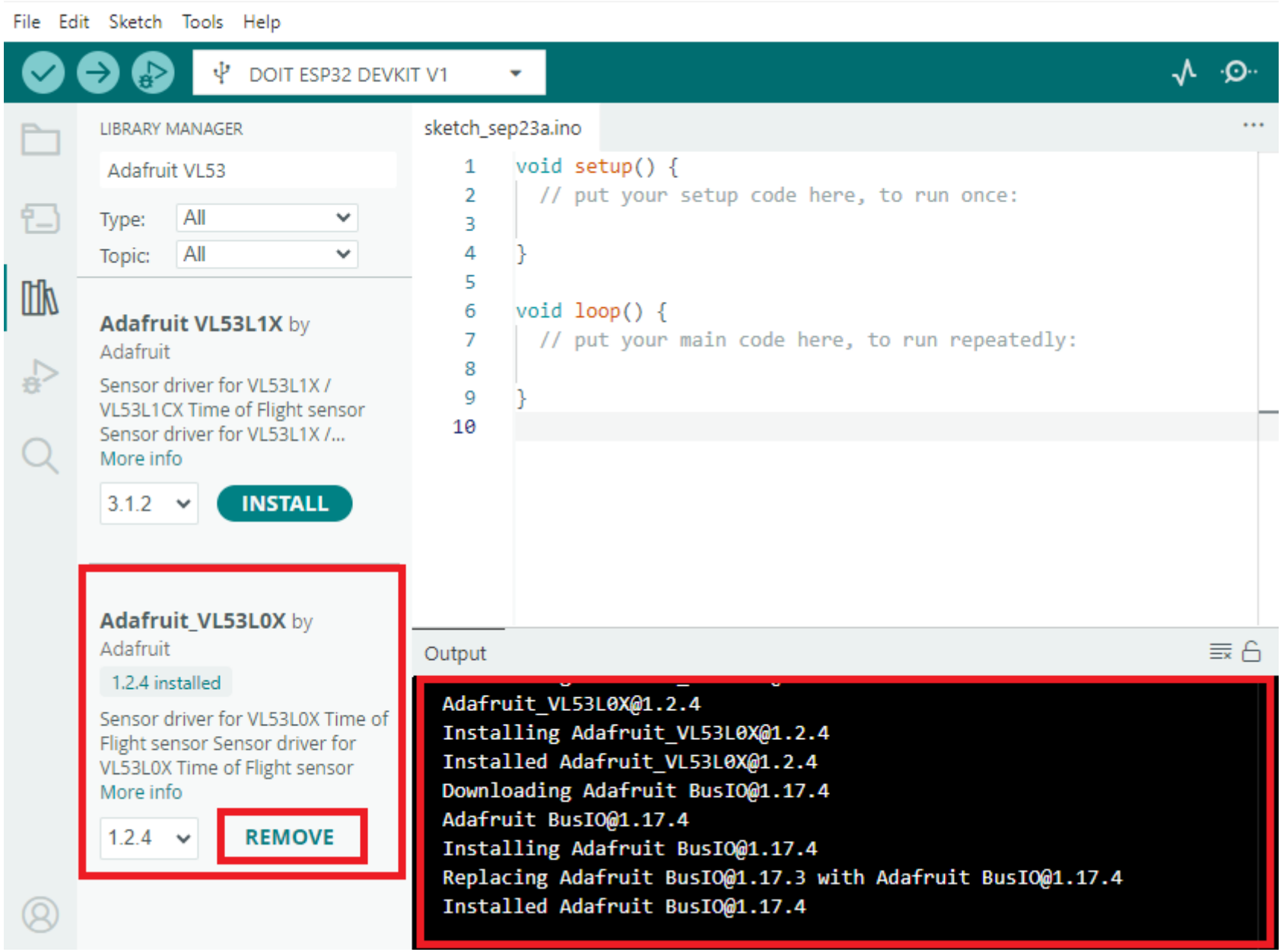


- After clicking the Install button, the library dependency installation window appears. Select the Install All option to install all required dependencies.



- The **Adafruit VL53L0X** library installation process is in progress. Wait until the installation completes; once finished, you can include the library in your Arduino sketch and start using the VL53L0X sensor



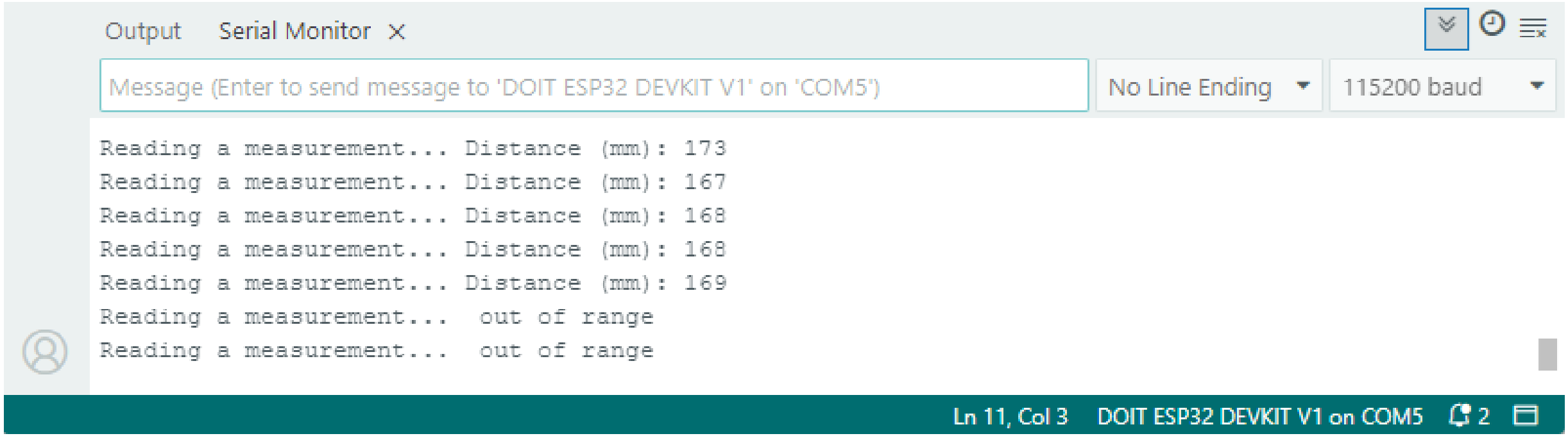

File Edit Sketch Tools Help
DOIT ESP32 DEVKIT V1
LIBRARY MANAGER
Adafruit VL53
Type: All
Topic: All
Adafruit VL53L1X by
Adafruit
Sensor driver for VL53L1X / VL53L1CX Time of Flight sensor Sensor driver for VL53L1X /...
More info
3.1.2
INSTALL
Adafruit_VL53L0X by
Adafruit
1.2.4 installed
Sensor driver for VL53L0X Time of Flight sensor Sensor driver for VL53L0X Time of Flight sensor
More info
1.2.4
REMOVE
sketch_sep23a.ino
1 void setup() {
2 // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7 // put your main code here, to run repeatedly:
8
9 }
10
Output
Adafruit_VL53L0X@1.2.4
Installing Adafruit_VL53L0X@1.2.4
Installed Adafruit_VL53L0X@1.2.4
Downloading Adafruit BusIO@1.17.4
Adafruit BusIO@1.17.4
Installing Adafruit BusIO@1.17.4
Replacing Adafruit BusIO@1.17.3 with Adafruit BusIO@1.17.4
Installed Adafruit BusIO@1.17.4

VL53LOX Sensor Code

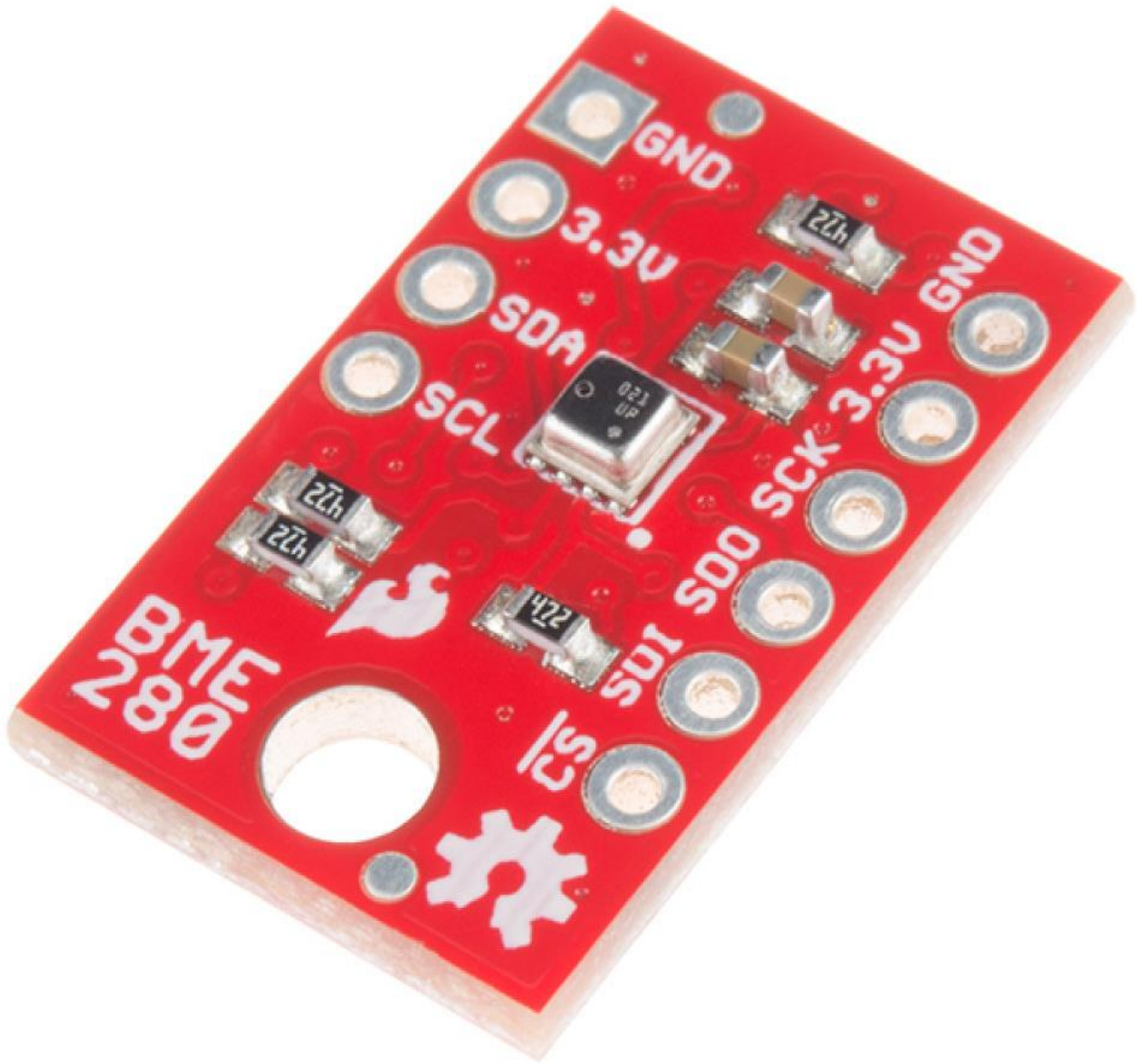
```
#include "Adafruit_VL53L0X.h" // Include the Adafruit VL53L0X library to use the
sensor functions
// Create an object of the VL53L0X class to communicate with the ToF sensor
Adafruit_VL53L0X lox = Adafruit_VL53L0X();

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

  // Wait until the serial port opens (needed for boards like ATmega32u4 / SAMD21 with
native USB)
  while (! Serial) {
    delay(1);
  }
    Serial.println("Adafruit VL53L0X test"); // Print a message to confirm the test
has started
    // Initialize the VL53L0X sensor
  if (!lox.begin()) {  // Check if the sensor is detected and started correctly
    Serial.println(F("Failed to boot VL53L0X"));  // Print error message if sensor
init fails
    while(1);  // Infinite loop to stop program if sensor is not found
  }
```

```
  // Print confirmation that the sensor is ready
  Serial.println(F("VL53L0X API Simple Ranging example\n\n"));
}

void loop() {
  VL53L0X_RangingMeasurementData_t measure;  // Create a data structure to store
measurement results

  Serial.print("Reading a measurement... ");  // Print status message

  // Perform a ranging test; 'false' disables debug output, 'true' enables debug
  lox.rangingTest(&measure, false);

  // Check if a valid reading is available
  if (measure.RangeStatus != 4) {  // Status code 4 means phase failure (invalid data)
    Serial.print("Distance (mm): ");  // Print label
    Serial.println(measure.RangeMilliMeter);  // Print measured distance in millimeters
  } else {
    Serial.println(" out of range ");  // If reading is invalid, print "out of range"
  }

  delay(100);  // Small delay between measurements (100ms -> ~10Hz reading rate)
}
```

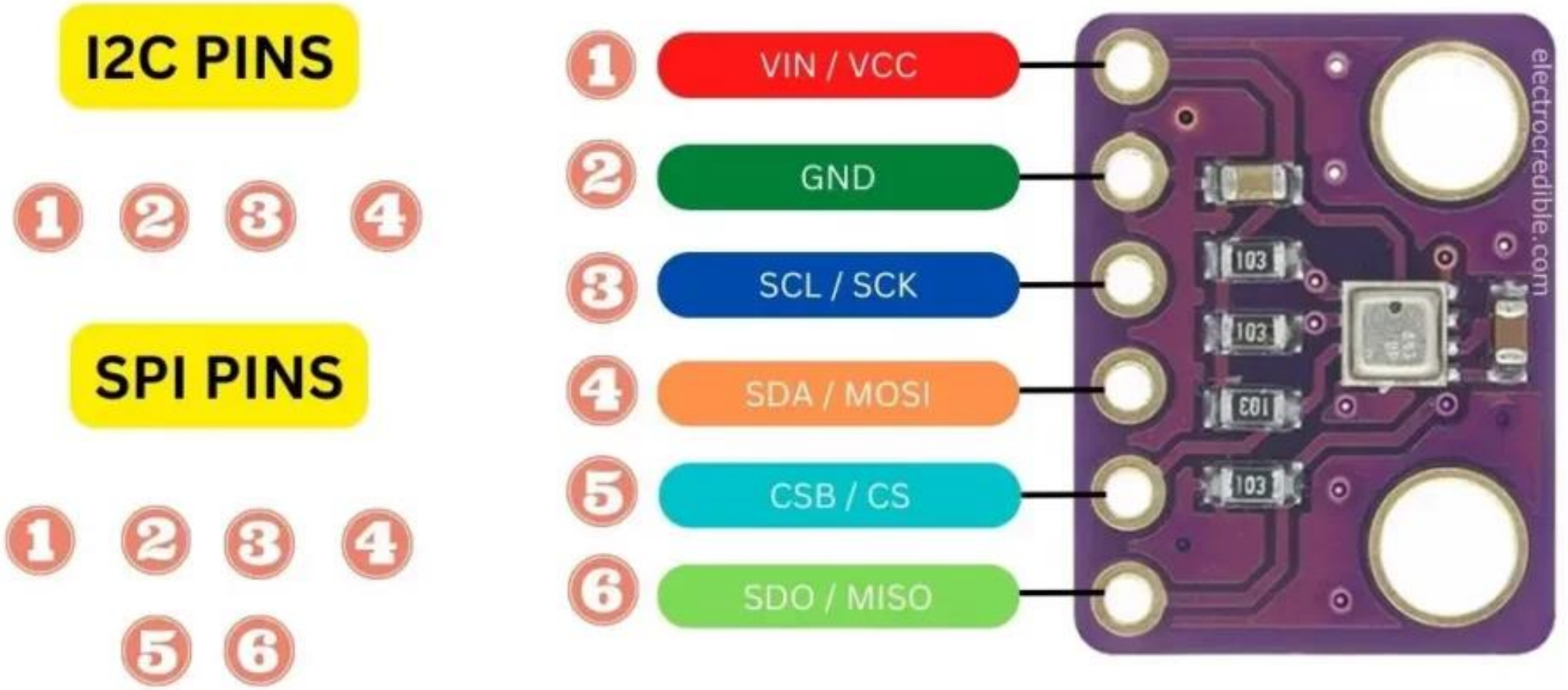
VL53LOX Sensor Output



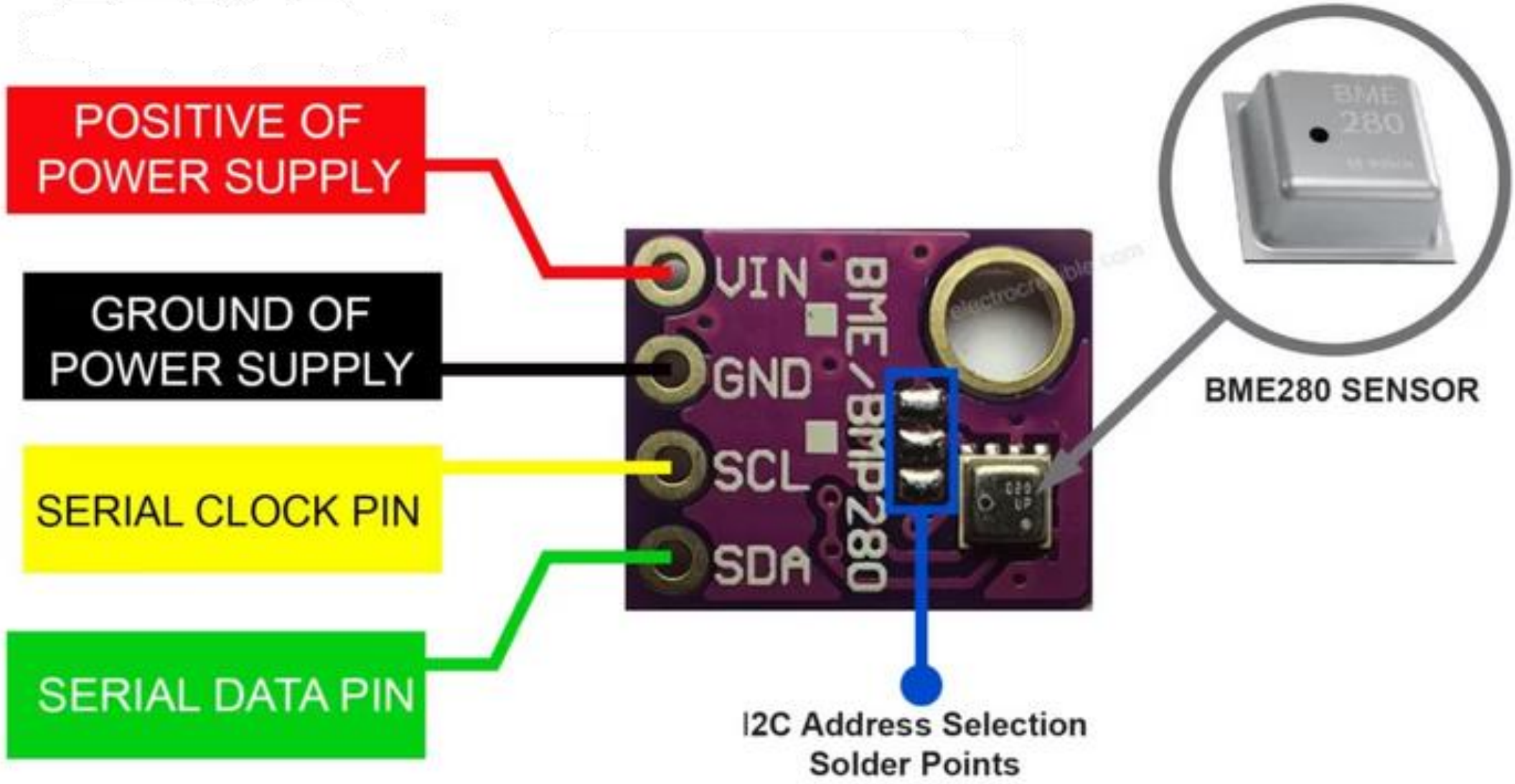
BMP280 Sensor



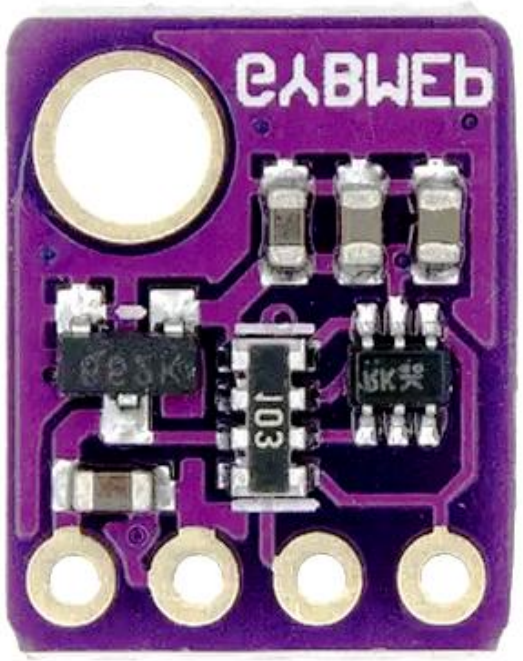
BMP280


I2C PINS
1 2 3 4
SPI PINS
1 2 3 4
5 6
1 VIN / VCC
2 GND
3 SCL / SCK
4 SDA / MOSI
5 CSB / CS
6 SDO / MISO
103
103
103
103
electrocredible.com

BMP280 Sensor



BMP280 Pinout

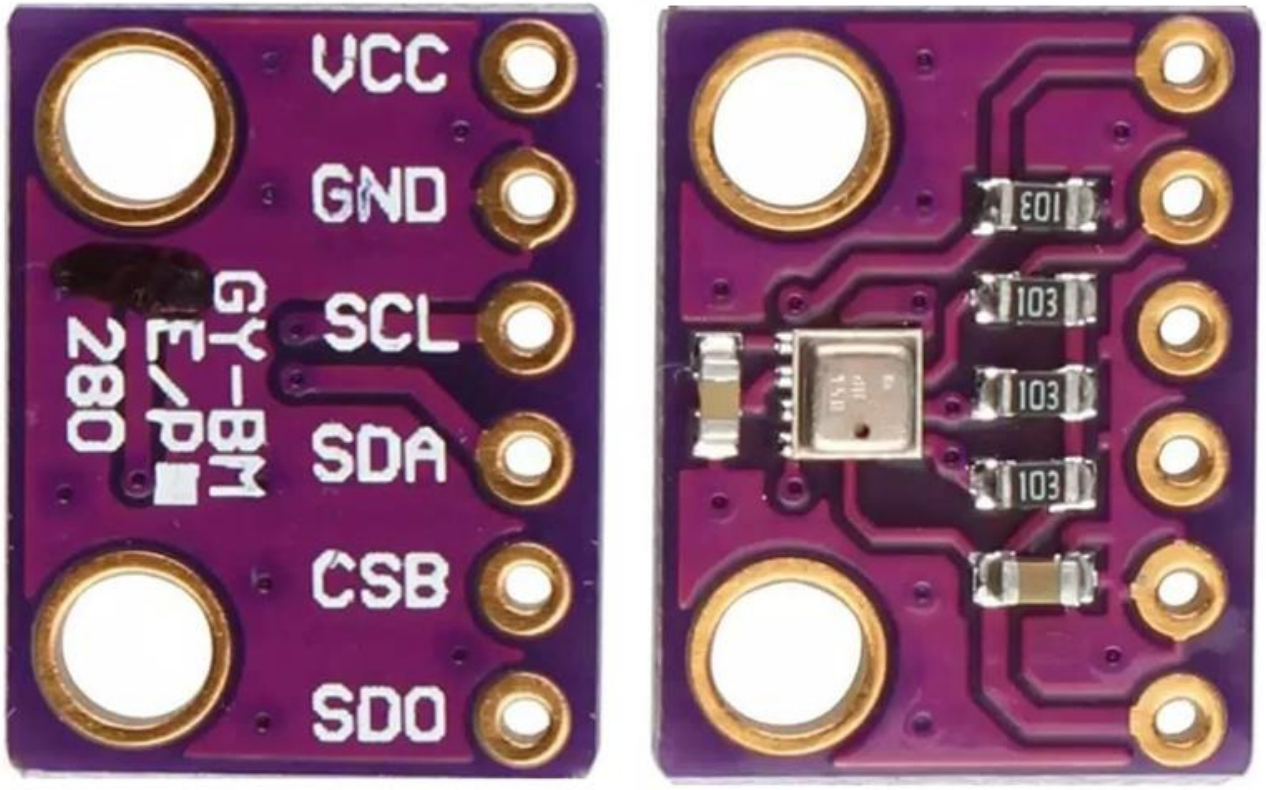


BMP280 Sensor Pin to ESP32

I2C Communication

BME280 Pin	ESP32 Pin
VIN	3.3V
GND	GND
SCL	GPIO 22
SDA	GPIO 21

BME280 Pinout



BMP280 Sensor Pin to ESP32

SPI Communication

BME280 Pin	ESP32 Pin
VIN	3.3V
GND	GND
SCK	GPIO 18
SDO (MISO)	GPIO 19
SDI (MOSI)	GPIO 23
CS	GPIO 5

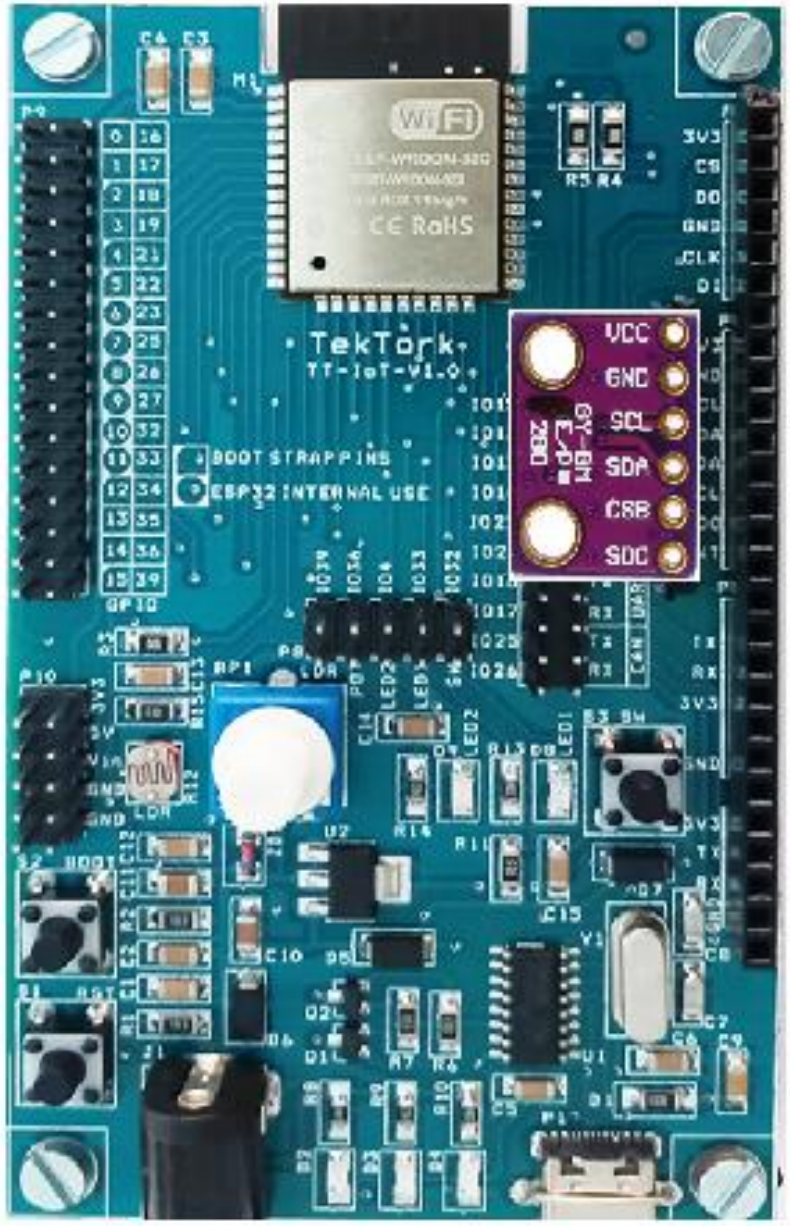
BME280 Specification

Parameter	Specification
Sensor Type	Temperature, Humidity, Pressure
Operating Voltage	1.71V – 3.6V (typical 3.3V)
Logic Level (I/O)	1.2V – 3.6V (5V tolerant if breakout has regulator/level shifters)
Current Consumption	< 1 mA (active mode), ~0.1 µA (sleep mode)
Interfaces	I^2C (up to 3.4 MHz) & SPI (up to 10 MHz)
Temperature Range	-40°C to +85°C
Temperature Accuracy	±1.0°C
Temperature Resolution	0.01°C
Humidity Range	0% – 100% RH
Humidity Accuracy	±3% RH (typical, 20–80% RH)
Humidity Resolution	0.008% RH
Pressure Range	300 hPa – 1100 hPa (approx. altitude -500 m to +9000 m)
Pressure Accuracy	±1 hPa (typical)
Pressure Resolution	0.18 Pa
Output Data Rate (ODR)	0.5 Hz – 157 Hz
Package Size (bare IC)	2.5 mm × 2.5 mm × 0.93 mm (LGA)

Wiring Diagram of the BME280 Sensor Connected to ESP 32 –Analog Input

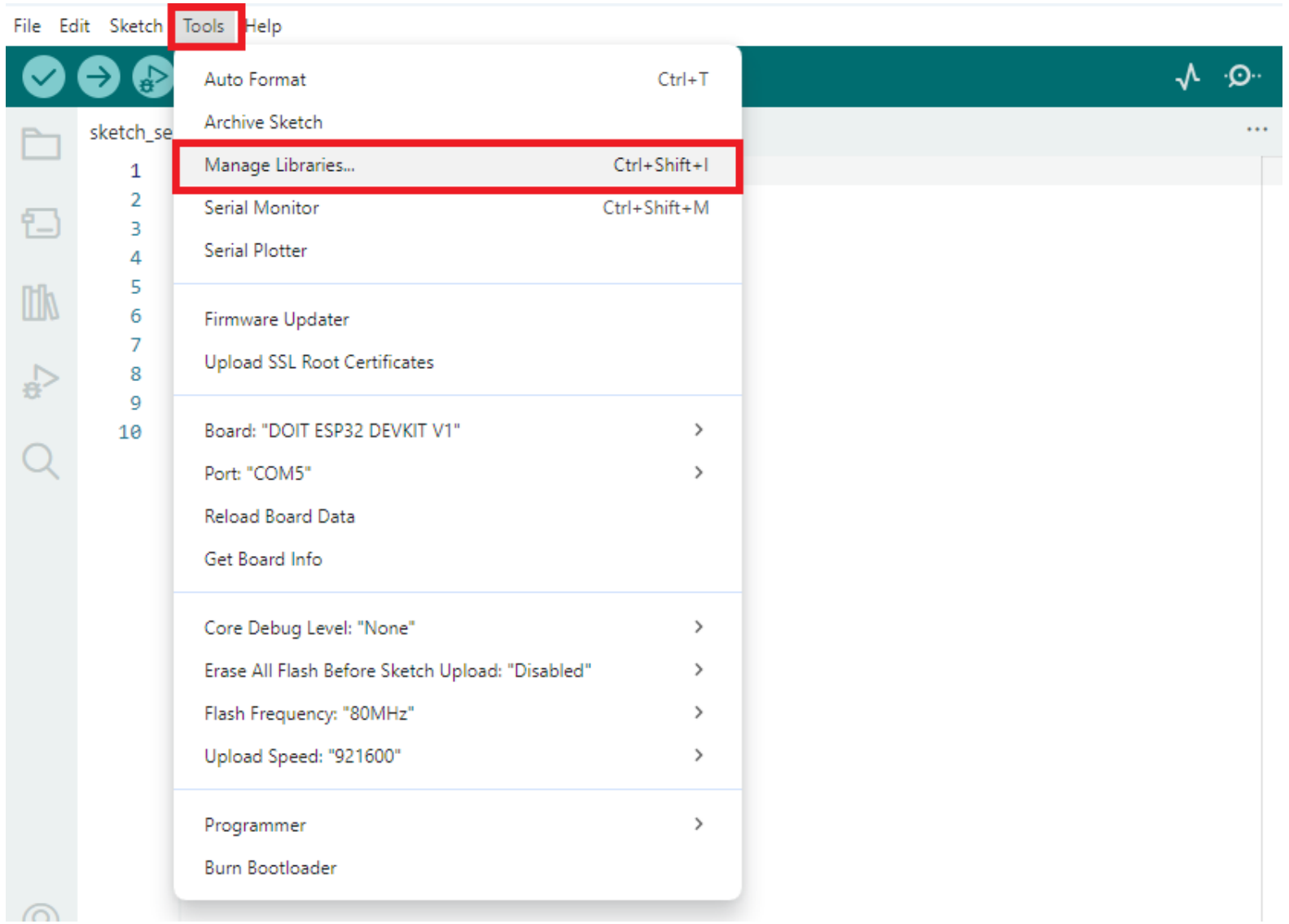
BMP280 Sensor Pin to ESP32

BMP280 Pin	ESP32 Pin
VIN	3.3V
GND	GND
SCK	GPIO 18
SDO (MISO)	GPIO 19
SDI (MOSI)	GPIO 23
CS	GPIO 5

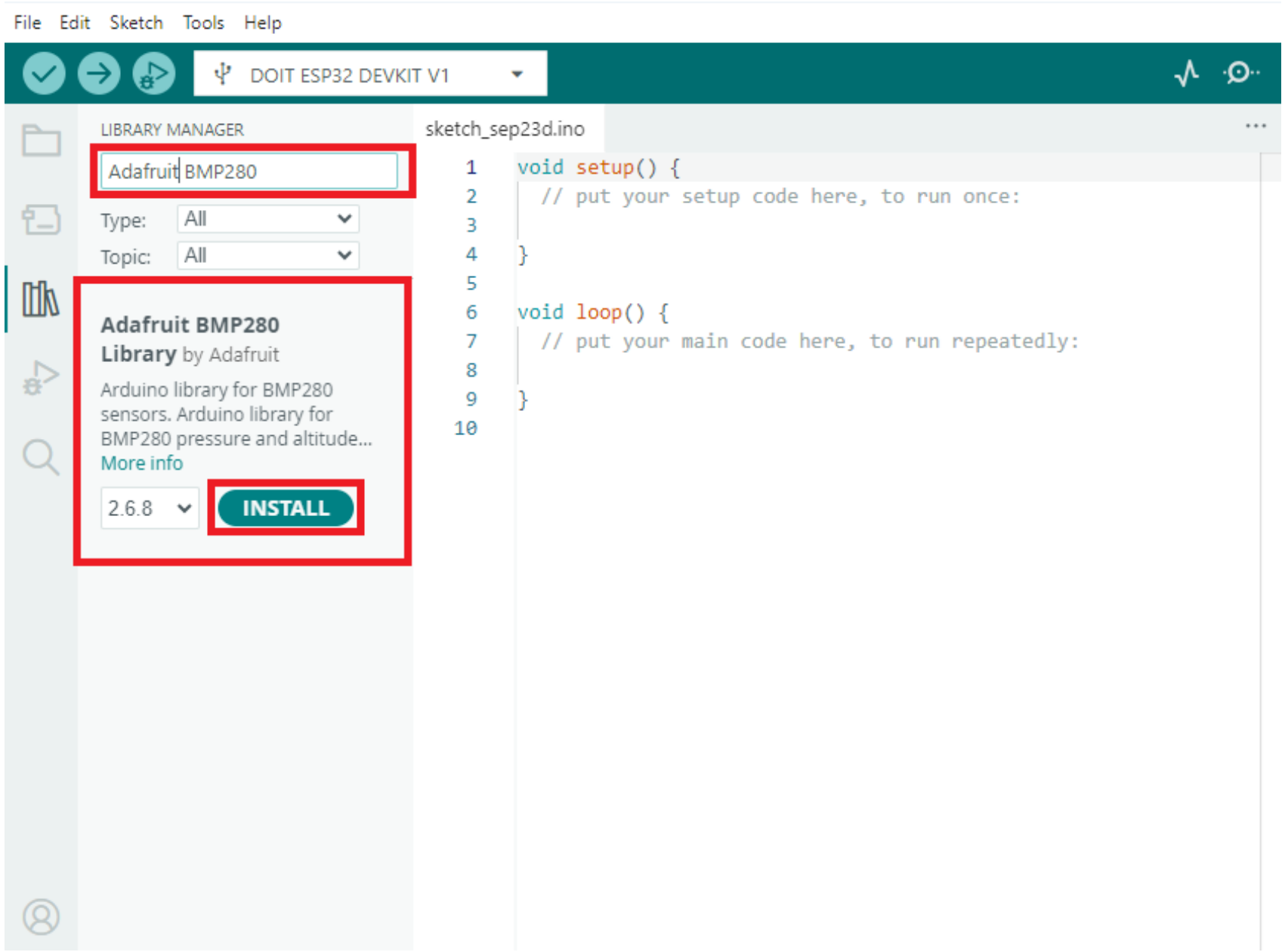


Library Installation

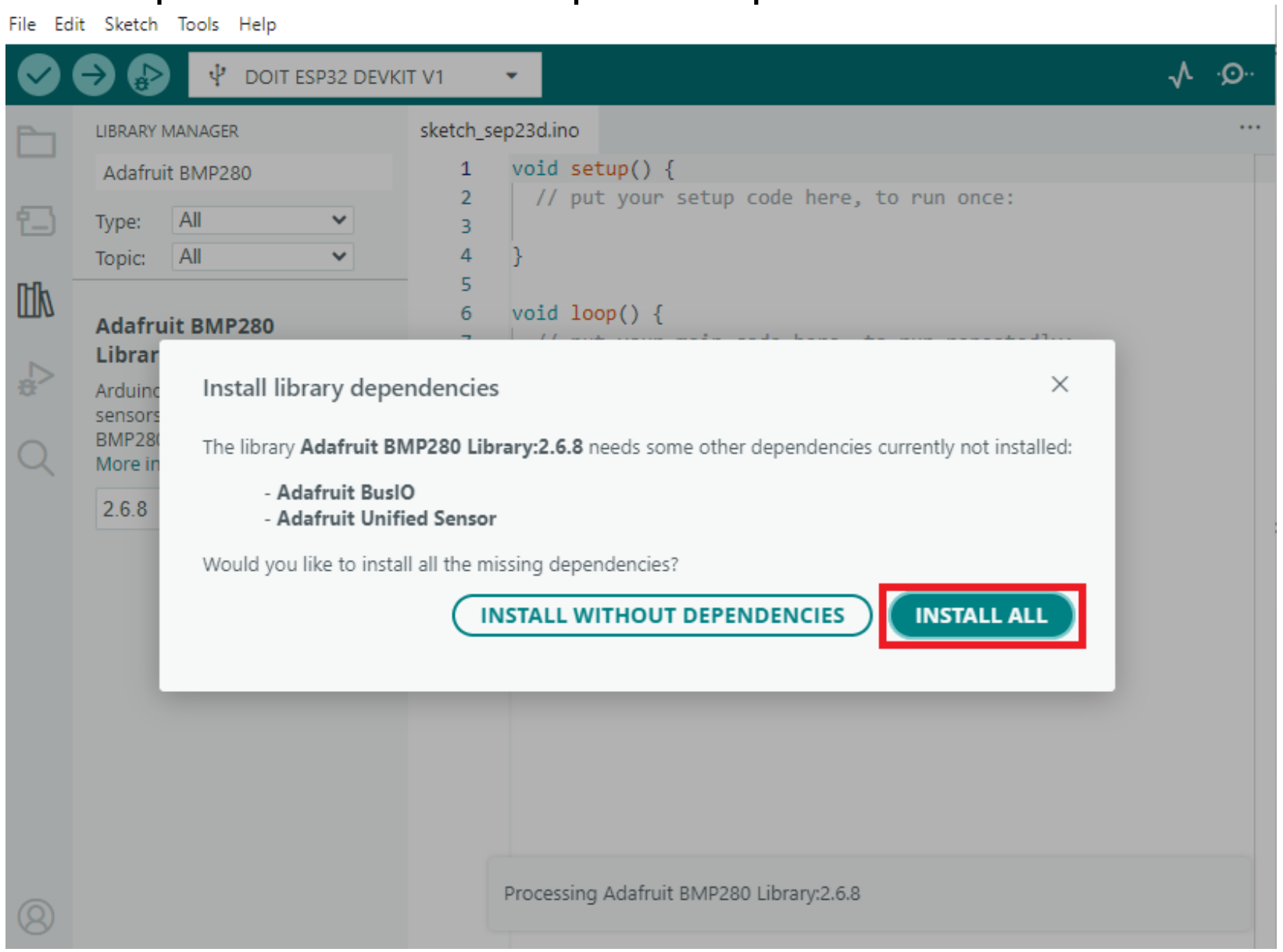
- In the Arduino IDE, go to the Tools menu, then select Manage Libraries... to open the Library Manager.



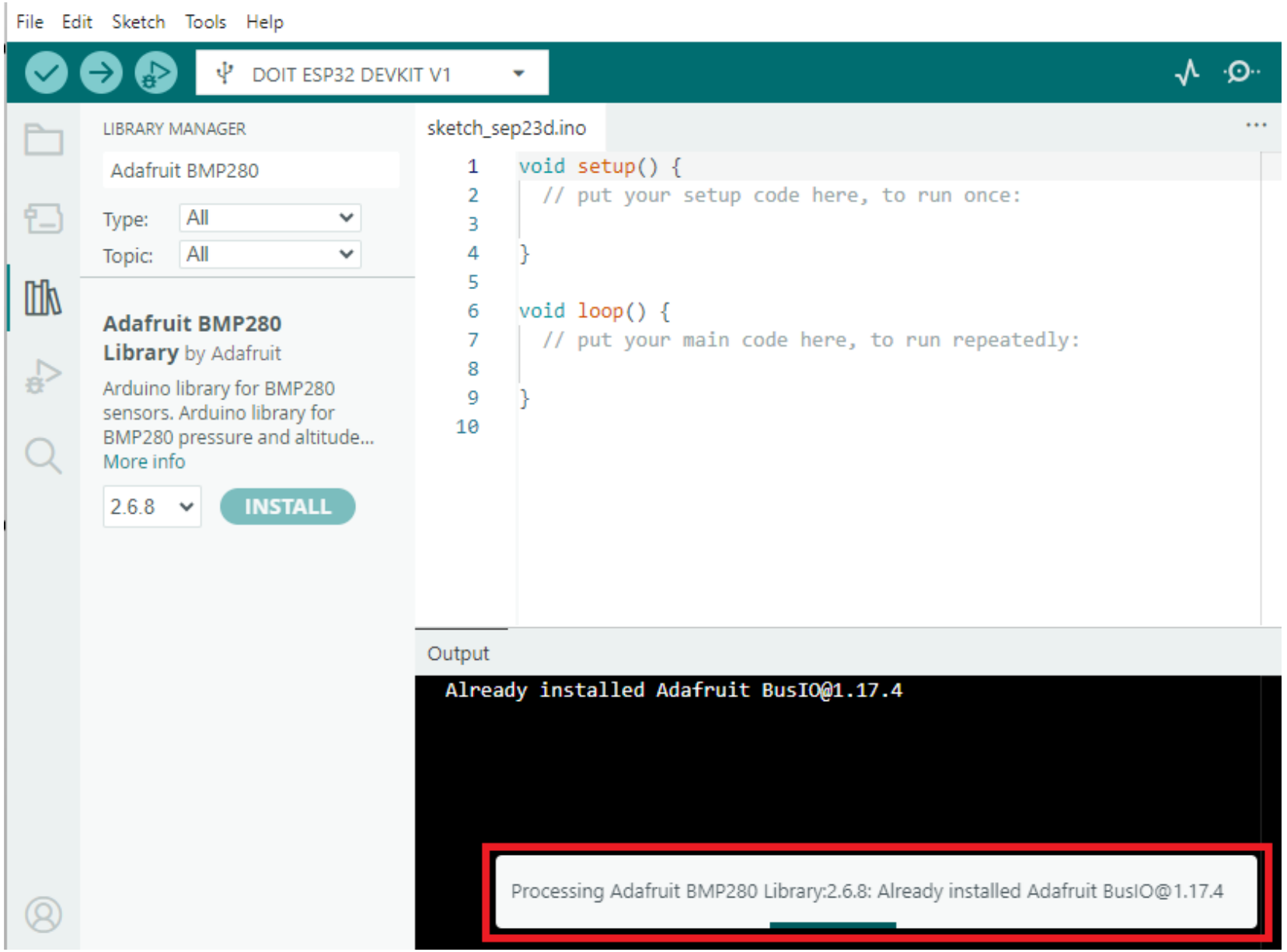
- In the Library Manager search box, type Adafruit BMP280, then select it and click Install



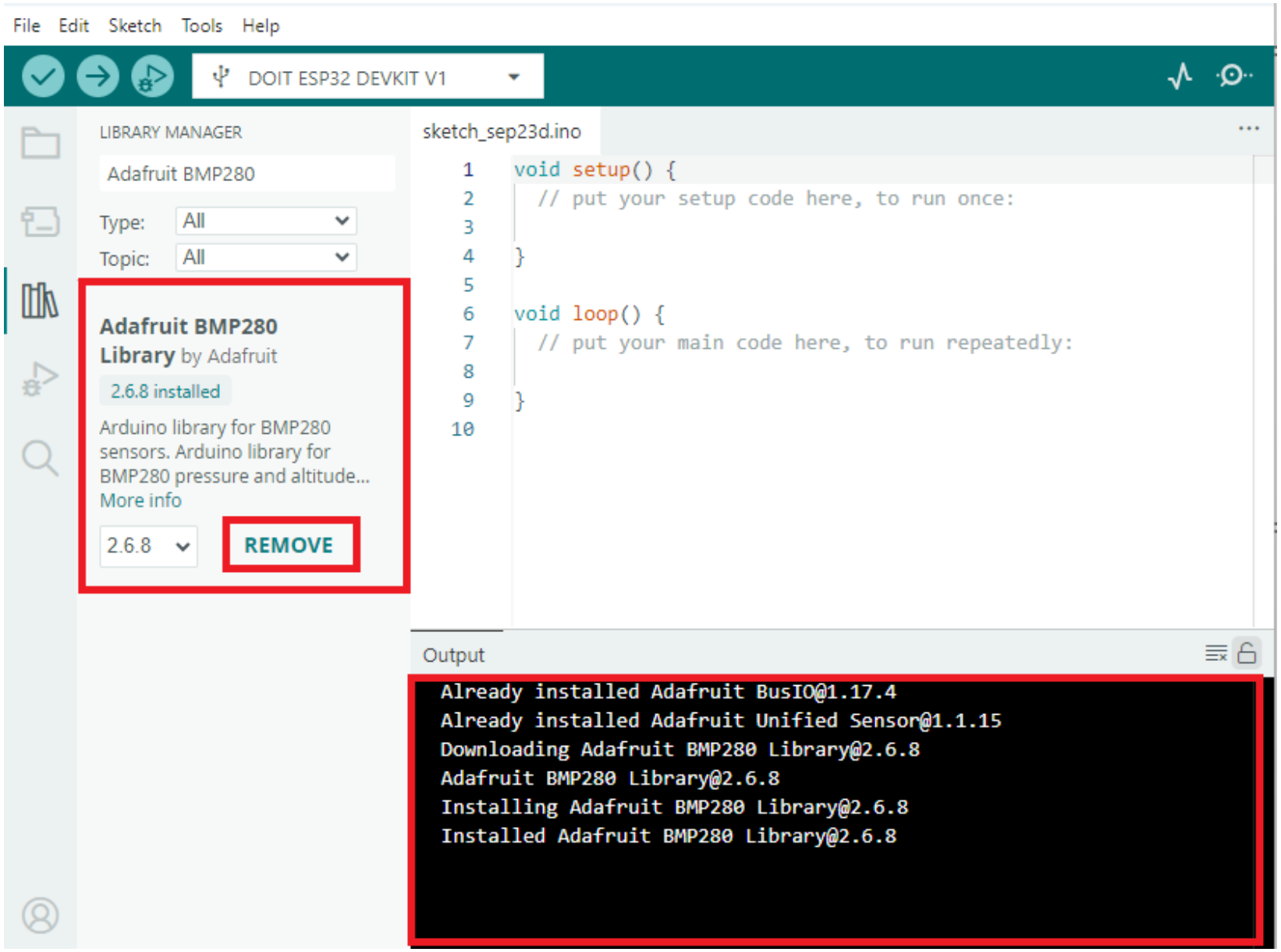
- After clicking the Install button, the library dependency installation window appears. Select the Install All option to install all required dependencies.



- The BMP280 library installation process is in progress. Wait until the installation completes; once finished, you can include the library in your Arduino sketch and start using the BMP280 sensor



- After the installation is complete, the Install button will change to Remove, and the output window will display the installation details.



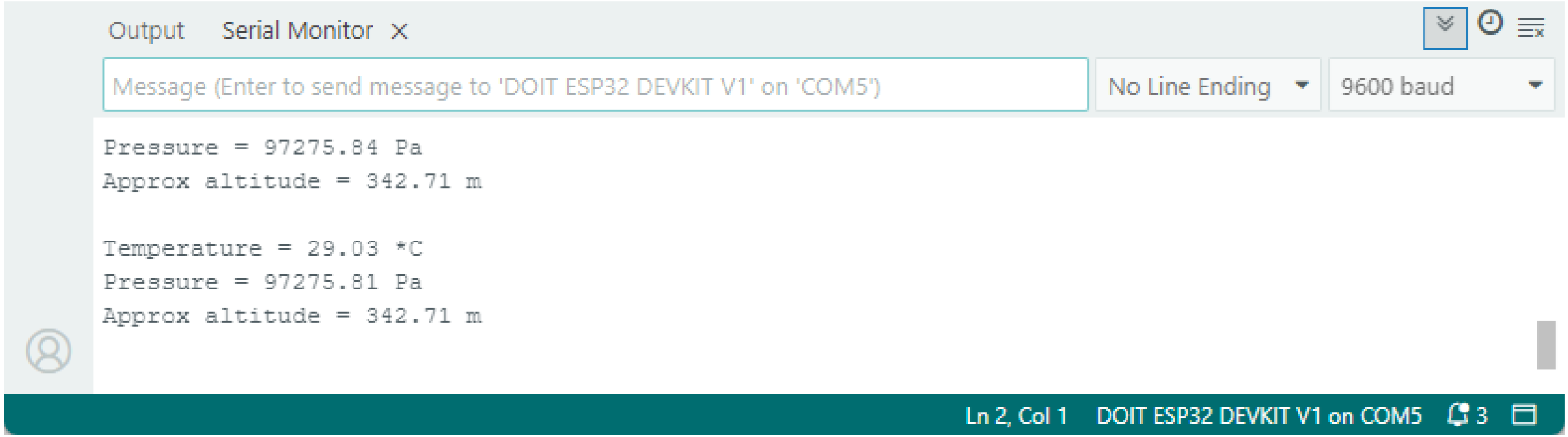
BME280 Code

```
#include <Adafruit_BMP280.h>   // Include the Adafruit BMP280 sensor library
// Pin definitions if using SPI connection (not used for I2C by default)
#define BMP_SCK  (13)
#define BMP_MISO (12)
#define BMP_MOSI (11)
#define BMP_CS   (10)

// Create BMP280 object using I2C interface
Adafruit_BMP280 bmp;
// Alternative constructors for SPI connection (commented out)
// Adafruit_BMP280 bmp(BMP_CS); // hardware SPI with defined CS
// Adafruit_BMP280 bmp(BMP_CS, BMP_MOSI, BMP_MISO, BMP_SCK); // software SPI

void setup() {
  Serial.begin(9600);                    // Start Serial Monitor at 9600 baud rate
  Serial.println(F("BMP280 Forced Mode Test."));
  // Initialize BMP280 with I2C address 0x76
  if (!bmp.begin(0x76)) {
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring or "
                     "try a different address!"));
    while (1) delay(10);                 // Stay here forever if sensor not found
  }
```

```
  /* Configure BMP280 sensor settings */
  bmp.setSampling(
    Adafruit_BMP280::MODE_FORCED,     // Forced mode (sensor measures only when commanded)
    Adafruit_BMP280::SAMPLING_X2,     // Temperature oversampling = x2
    Adafruit_BMP280::SAMPLING_X16,    // Pressure oversampling = x16
    Adafruit_BMP280::FILTER_X16,      // IIR filter strength = 16
    Adafruit_BMP280::STANDBY_MS_500   // Standby time (not much used in forced mode)
  );
}
void loop() {
  // In forced mode, we must trigger a measurement manually
  // takeForcedMeasurement() wakes sensor, blocks until data is ready
  if (bmp.takeForcedMeasurement()) {
    // After measurement, we can read sensor values
    Serial.print(F("Temperature = "));
    Serial.print(bmp.readTemperature());  // Read temperature (in Celsius)
    Serial.println(" *C");
    Serial.print(F("Pressure = "));
    Serial.print(bmp.readPressure());     // Read atmospheric pressure (in Pascals)
    Serial.println(" Pa");
    Serial.print(F("Approx altitude = "));
    Serial.print(
      bmp.readAltitude(1013.25)           // Estimate altitude using sea level pressure (hPa)
    );
```

```
    Serial.println(" m");
    Serial.println();
    delay(2000); // Wait 2 seconds before taking another forced measurement
  } else {
    Serial.println("Forced measurement failed!");  // Error if measurement did not complete
  }
}
```

BME280 Output



Thank You