Department of Electronics & Communication Engineering

Internet of Things Lab

VI Semester (22ECL651)

Staffs in-charge:
Dr. Pavithra G
Dr. Swapnil S N

Lab Instructors: Mrs. Gayatri & Mrs. Veena

HOD: Dr. Shobha K R

Name of the Student	:	
Semester /Section	:	
USN	:	
Batch	:	

Dayananda Sagar College of Engineering
Shavige Malleshwara Hills, Kumaraswamy Layout,
Banashankari, Bangalore-560111, Karnataka
Tel : +91 80 26662226 26661104 Extn : 2731 Fax : +90 80 2666 0789
Web - http://www.dayanandasagar.edu Email : hod-ece@dayanandasagar.edu
(An Autonomous Institute Affiliated to VTU, Approved by AICTE & ISO 9001:2008 Certified)

About the College

The Dayananda Sagar College of Engineering was established in 1979 by Sri R. Dayananda Sagar. Dayananda Sagar College of Engineering operates under the aegis of the **Mahatma Gandhi Vidya Peetha Trust** is approved by All India Council for Technical Education (AICTE), Govt. of India and affiliated to Visvesvaraya Technological University. It has the widest choice of engineering branches having 20 Under Graduate courses & 6 Post Graduate courses. In addition, it has 20 Research Centres in different branches of Engineering catering to research scholars for obtaining Ph.D. under VTU. Various courses are accredited by NBA.

The Institute is spread over 23 acres of land with large infrastructure supported by laboratories with state-of-the-art, Equipment & Machines. The Central Library with modern facilities and the Digital Library provides the knowledge base for the students.

The campus is WI-FI equipped with large bandwidth internet facility. The College has good faculty strength with highest professional integrity and committed to the academics with transparency in their actions. Each faculty takes the responsibility of mentoring a certain number of students through personal attention paving the way for the students' professional growth. The faculty are research oriented having number of sponsored R & D projects from different agencies such as Department of Science & Technology, Defence R & D organizations, Indian Space Research Organization, AICTE etc.

About the Department

Department of Electronics and Communication Engineering is one of the oldest and biggest department in the DSI group with 50 faculty members and 1200+ students. Most of our faculty have pursued Ph.D. and others are in the process. The ECE department provides high-quality, innovative technical education at the undergraduate, graduate, and research levels. At present, the department offers an UG course (BE) with an intake of 240, a PG course in VLSI Design and Embedded Systems with an intake of 18 students and Ph.D. The department has got an excellent infrastructure with sophisticated labs, class rooms and R & D center. The department faculty and support personnel are dedicated towards helping students achieve success in today's world by offering them the knowledge and skills required. The department places a strong emphasis on leading-edge research through its research centre and research forum. These factors together enable our graduating students to create a big impact on the workforce from day one.

In addition to offering innovative courses, the department is more focused on improving students' educational experiences by offering value-added courses, skill-development programs, technical challenges and hackathons at state, national, and international arenas. Through its professional societies and technical clubs, the department takes great satisfaction in exposing its students to current industry developments and upcoming technologies. The dedicated staff at the ECE department help students procure jobs in prestigious companies both domestically and abroad.

Vision of the College:

To impart quality technical education with a focus on Research and Innovation emphasizing on Development of Sustainable and Inclusive Technology for the benefit of society.

Mission of the College:

- ❖ To provide an environment that enhances creativity and Innovation in pursuit of Excellence.
- ❖ To nurture teamwork in order to transform individuals as responsible leaders and entrepreneurs.
- ❖ To train the students to the changing technical scenario and make them to understand the importance of Sustainable and Inclusive technologies.

Vision of the Department

To achieve continuous improvement in quality technical education for global competence with focuson industry, societal needs, research and professional success.

Mission of the Department

- ❖ Offering quality education in Electronics and Communication Engineering with effective teaching learning process in multidisciplinary environment.
- ❖ Training the students to take-up projects in emerging technologies and work with team spirit.
- ❖ To imbibe professional ethics, development of skills and research culture for better placement opportunities.

Program Educational Objectives [PEOs]

The Graduate students must be able to:
PEO-1: Ready to apply the state-of-art technology in industry and meeting the societal need with Knowledge of Electronics and Communication Engineering due to strong academic culture.
PEO-2: Competent in technical and soft skills to be employed with capability of working in Multidisciplinary domains.
PEO-3: Professionals, capable of pursuing higher studies in technical, research management programs.

Programme Specific Outcomes [PSOs]

PSO-1: Design, develop and integrate electronic circuits and systems using current practices and Standards.
PSO-2: Apply knowledge of hardware and software in designing Embedded and Communication Systems.

Programme Outcomes [POs]

Engineering Graduates will be able to:

1. **Engineering Knowledge:** Apply the Knowledge of Mathematics, Science, Engineering Fundamentals, and an Engineering specialization to the solution of complex Engineering problems.
2. **Problem Analysis:** Identify, Formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of Mathematics, natural sciences and engineering sciences.
3. **Design/Development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental conditions.
4. **Conduct investigations on complex problems:** Use research-based knowledge and research methods including design of Experiments, analysis and interpretation of data, and synthesis of Information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate technique, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess society, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Lifelong learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Internet of Things Lab *(Standalone Lab)*

Course Code: 22ECL651 **CIE Marks:** 50 **Credits:** 1
L : T : P : 0 : 0 : 1 **Lab Duration**: 02 Hrs **Total:** 12 Lab sessions

COURSE OBJECTIVES:

1.	Develop skills in interfacing sensors and actuators with ESP32 for real-time data acquisition.
2.	Explore IoT communication protocols and networking concepts using ESP32.
3.	Implement IoT-based web servers and cloud integrations for remote monitoring and control.
4.	Design and develop smart IoT applications with automation, security, and efficient data handling.

COURSE OUTCOMES: At the end of the course, the student will be able to

CO1	Attain knowledge of IoT and its significance in today's world.
CO2	Able to understand different components, and design IoT applications.
CO3	Assimilate the IoT concepts and the use of various IoT components for effective networking.
CO4	Acquire technical knowhow of IoT, sensors and integration with networks.

List of Experiments

1. **Connect an LED to an ESP32 and create a program that switches the LED on for 1 second every 2 seconds.**
2. **Measure the temperature, humidity and heat index using ESP32 development board and display the values on serial monitor.**
3. **Calculate the distance to an object with the help of an ultrasonic sensor and display it on serial monitor.**
4. **Measure soil moisture value using ESP32 development board and display the sensor values on serial monitor.**
5. **Motion detection using IR sensor and turn on buzzer upon detection.**
6. **Measure BPM value using pulse sensor and display on serial monitor.**
7. **Configure/Set your ESP32 development broad as an access point.**
8. **Scan Wi-Fi networks and get Wi-Fi strength using ESP32 development board.**
9. **Establish connection between IoT node and access point and display of local IP and gateway IP.**
10. **Design and implementation of HTTP based IoT web server to control the status of LED.**
11. **Design and implementation of HTTP based IoT web server to display sensor value.**
12. **Design and implementation of MQTT based controlling and monitoring using Ubidots MQTT server.**

DEPARTMENT of ELECTRONICS AND COMMUNICATION ENGINEERING

INTERNET OF THINGS LABORATORY
Index Sheet

Dayananda Sagar College of Engineering
Dept. of Electronics & Communication Engineering

Name of the Laboratory	**:**	Internet of Things Lab (Standalone Lab)
Semester	**:**	VI
No. of students /batch	**:**	22
No. of equipment's	**:**	22
Major Equipment's	**:**	ESP32 Development kit, Computer, Arduino Software
Operating System	**:**	Windows 10 and higher
Simulation Tools	**:**	Arduino
Area of the lab inSq.mts	**:**	102 Sqmts
Lab in-charges	**:**	Dr. Pavithra G Dr. Swapnil S N
Instructor name	**:**	Mrs. Gayatri Mrs. Veena
HoD	**:**	Dr. Shobha K R

DAYANANDA SAGAR COLLEGE OF ENGINEERING
DEPARTMENT OF ELCTRONICS & COMMUNICATION ENGINEERING

DO's

- All the students should come to LAB on time with proper dress code and identity cards.
- Keep your belongings in the bag rack of laboratory.
- Students have to enter their name, USN, time-in/out and signature in the log register maintained in the laboratory.
- All the students should submit their records before the commencement of Laboratory experiments.
- Students should come to the lab well prepared for the experiments which are to be performed in that particular session.
- Students are asked to do the experiments on their own and should not waste their precious time by talking, roaming and sitting idle in the labs.
- Observation book and record book should be complete in all respects and it should be corrected by the staff member.
- Before leaving the laboratory, students should arrange their chairs and leave in orderly manner after completion of their scheduled time.
- Prior permission to be taken, if for some reasons, they cannot attend lab.
- Immediately report any sparks/ accidents/ injuries/ any other untoward incident to the faculty /instructor.
- In case of an emergency or accident, follow the safety procedure.
- Switch OFF the power supply after completion of experiment.

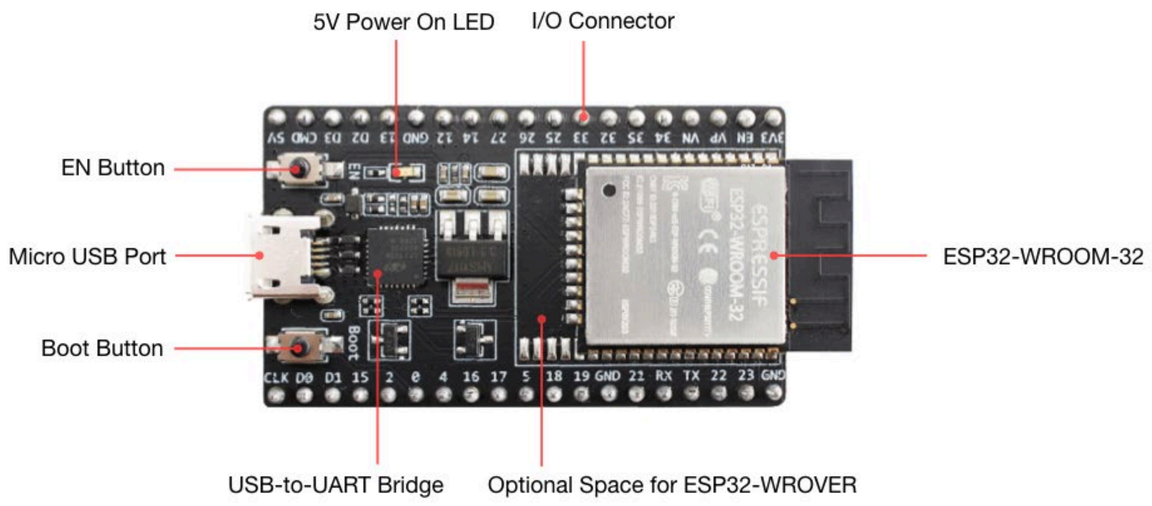
DONT's

- The use of mobile/ any other personal electronic gadgets is prohibited in the laboratory.
- Do not make noise in the Laboratory & do not sit on experiment table.
- Do not make loose connections and avoid overlapping of wires
- Don't switch on power supply without prior permission from the concerned staff.
- Never leave the experiments while in progress.
- Do not leave the Laboratory without the signature of the concerned staff in observation book.

About 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.

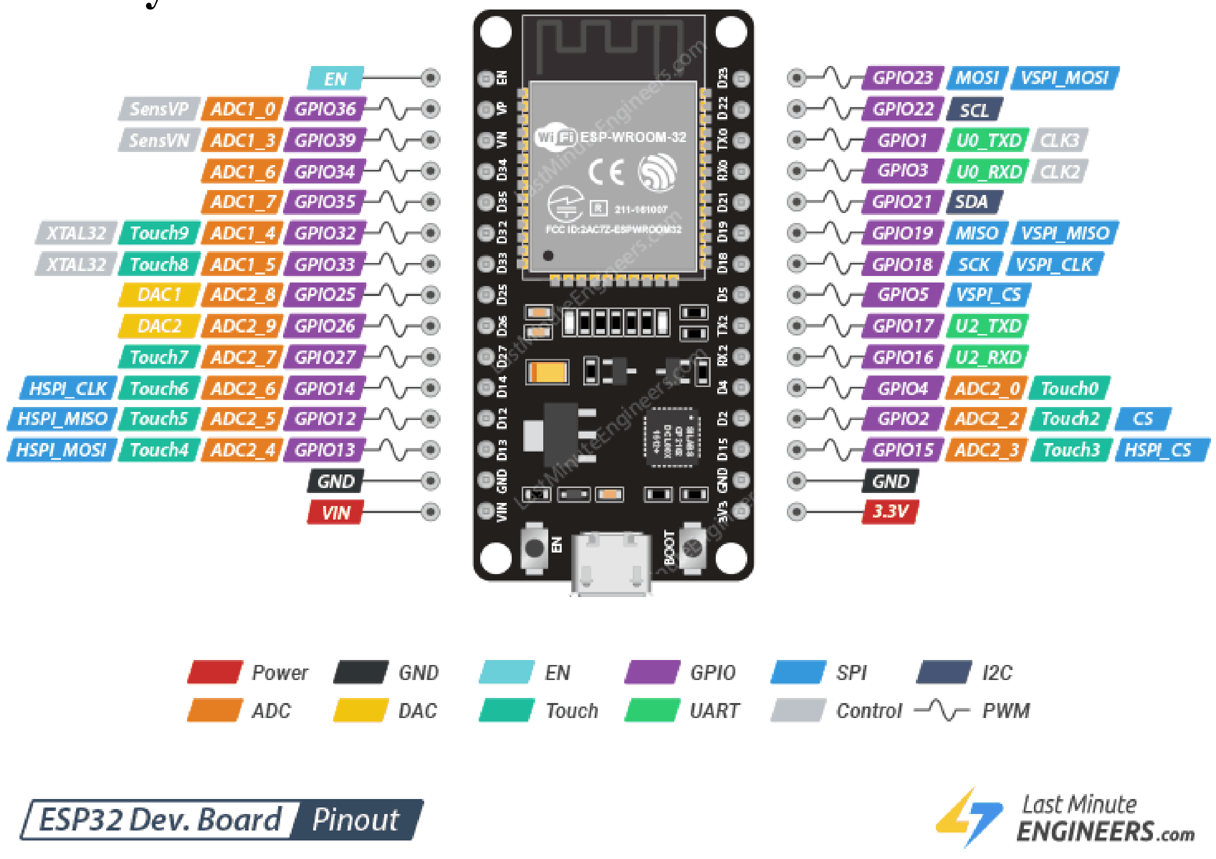
Note: The pins D0, D1, D2, D3, CMD and CLK are used internally for communication between ESP32 and SPI flash memory. They are grouped on both sides near the USB connector. Avoid using these pins, as it may disrupt access to the SPI flash memory / SPI RAM.

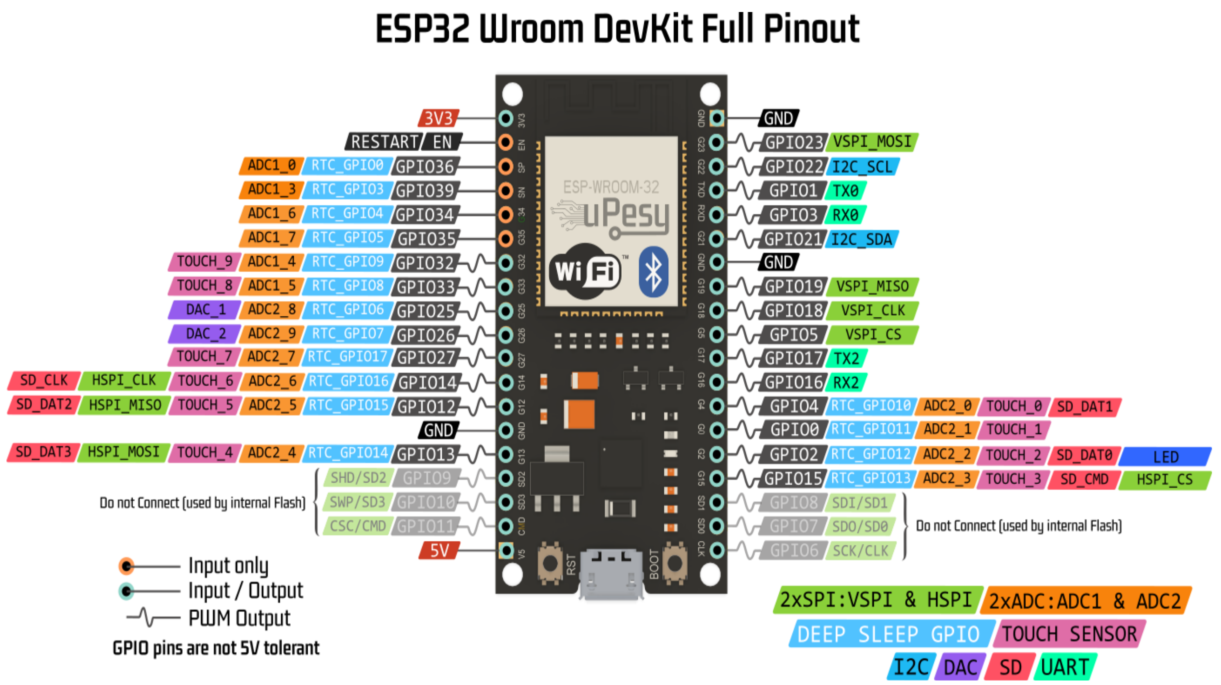
Features of the ESP32 include the following:

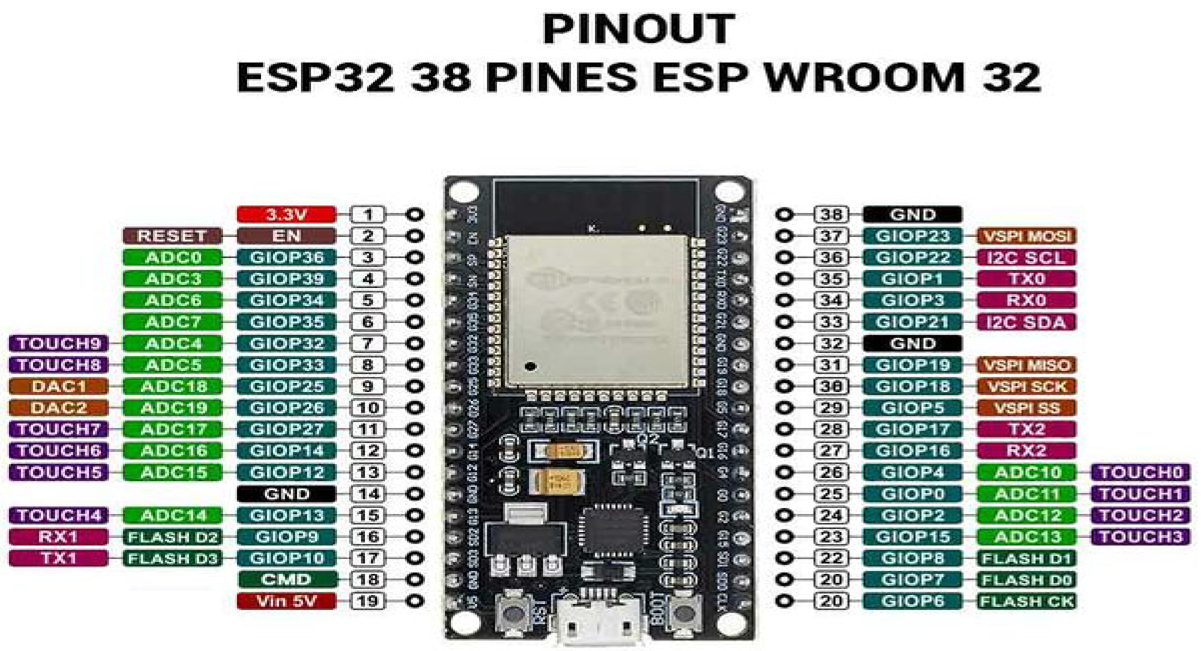
- **Processors:**
  - CPU: Xtensa dual-core (or single-core) 32-bit LX6 microprocessor, operating at 160 or 240 MHz and performing at up to 600 DMIPS
  - Ultra-low power (ULP) co-processor
- **Memory:** 320 KiB RAM, 448 KiB ROM
- **Wireless connectivity:**
  - Wi-Fi: 802.11 b/g/n
  - Bluetooth: v4.2 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 × I^2S interfaces
  - 2 × I^2C 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 µA deep sleep current
    - Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

Pin Layout:




ESP32 Wroom DevKit Full Pinout
3V3
RESTART / EN
ADC1_0 / RTC_GPIO0 / GPIO36
ADC1_3 / RTC_GPIO3 / GPIO39
ADC1_6 / RTC_GPIO4 / GPIO34
ADC1_7 / RTC_GPIO5 / GPIO35
TOUCH_9 / ADC1_4 / RTC_GPIO9 / GPIO32
TOUCH_8 / ADC1_5 / RTC_GPIO8 / GPIO33
DAC_1 / ADC2_8 / RTC_GPIO6 / GPIO25
DAC_2 / ADC2_9 / RTC_GPIO7 / GPIO26
TOUCH_7 / ADC2_7 / RTC_GPIO17 / GPIO27
SD_CLK / HSPI_CLK / TOUCH_6 / ADC2_6 / RTC_GPIO16 / GPIO14
SD_DAT2 / HSPI_MISO / TOUCH_5 / ADC2_5 / RTC_GPIO15 / GPIO12
GND
SD_DAT3 / HSPI_MOSI / TOUCH_4 / ADC2_4 / RTC_GPIO14 / GPIO13
SHD/SD2 / GPIO9
SWP/SD3 / GPIO10
CSC/CMD / GPIO11
Do not Connect (used by internal Flash)
5V
GND
GPIO23 / VSPI_MOSI
GPIO22 / I2C_SCL
GPIO1 / TX0
GPIO3 / RX0
GPIO21 / I2C_SDA
GND
GPIO19 / VSPI_MISO
GPIO18 / VSPI_CLK
GPIO5 / VSPI_CS
GPIO17 / TX2
GPIO16 / RX2
GPIO4 / RTC_GPIO10 / ADC2_0 / TOUCH_0 / SD_DAT1
GPIO0 / RTC_GPIO11 / ADC2_1 / TOUCH_1
GPIO2 / RTC_GPIO12 / ADC2_2 / TOUCH_2 / SD_DAT0 / LED
GPIO15 / RTC_GPIO13 / ADC2_3 / TOUCH_3 / SD_CMD / HSPI_CS
GPIO8 / SDI/SD1
GPIO7 / SDO/SD0
GPIO6 / SCK/CLK
Do not Connect (used by internal Flash)
ESP-WROOM-32
uPesy
RST
BOOT
Input only
Input / Output
PWM Output
GPIO pins are not 5V tolerant
2xSPI:VSPI & HSPI
2xADC:ADC1 & ADC2
DEEP SLEEP GPIO
TOUCH SENSOR
I2C
DAC
SD
UART



For more details : [**https://www.upesy.com/blogs/tutorials/esp32-pinout-reference-gpio-pins-ultimate-guide**](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**](https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf)

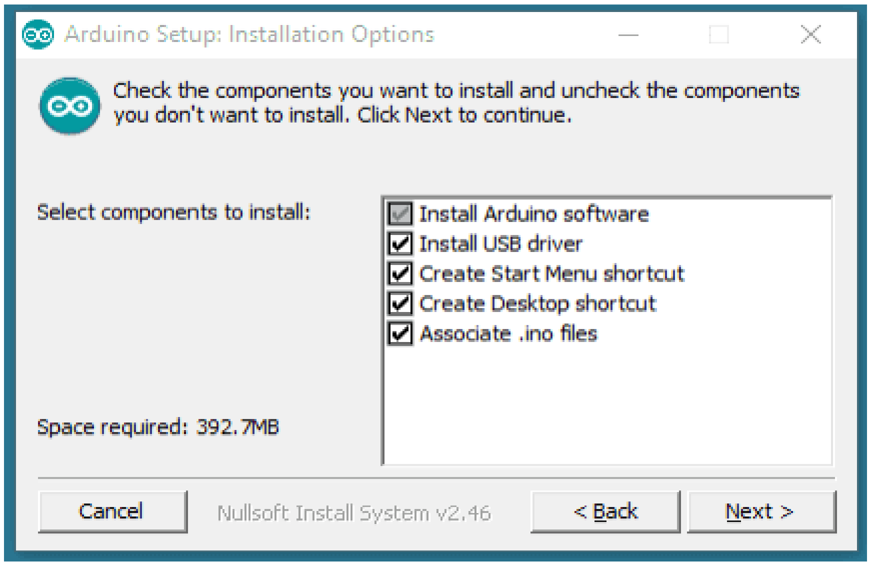
Install the Arduino Desktop IDE:
Arduino IDE Installation (Windows):

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.


Arduino Setup: Installation Options

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

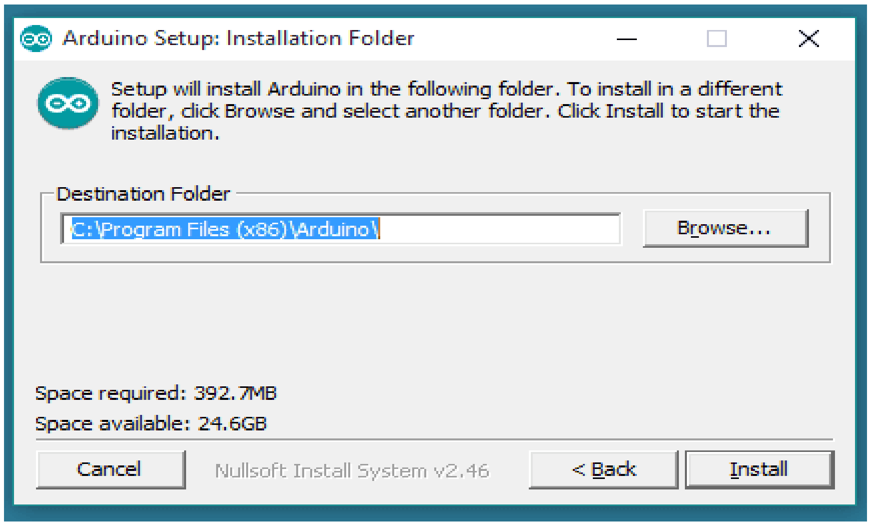
Select components to install:

- [✓] Install Arduino software
- [✓] Install USB driver
- [✓] Create Start Menu shortcut
- [✓] Create Desktop shortcut
- [✓] Associate .ino files

Space required: 392.7MB

Choose the components to install.

Step 3:


Arduino Setup: Installation Folder

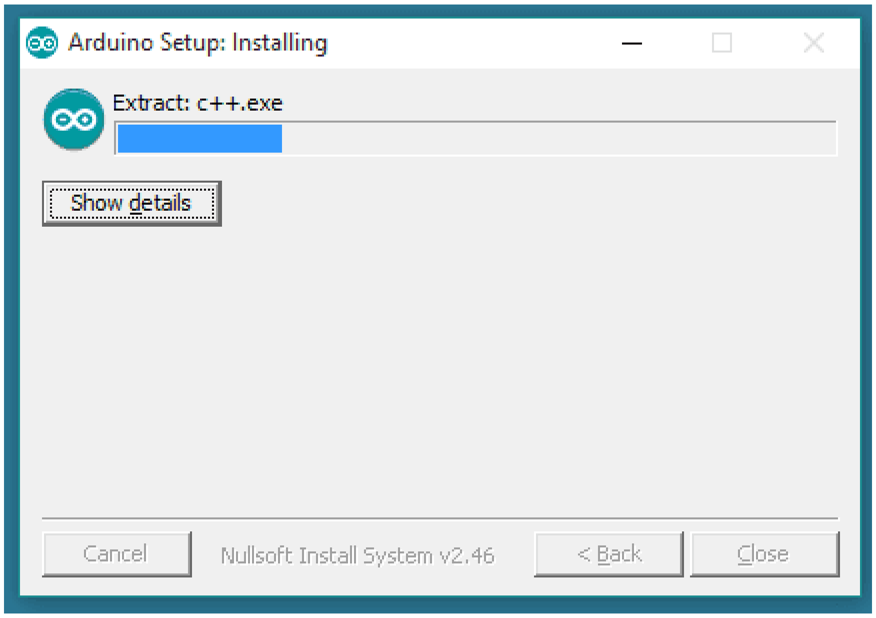
Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files (x86)\Arduino\

Space required: 392.7MB
Space available: 24.6GB

Choose the installation directory.

Step 4:



Installation in progress.

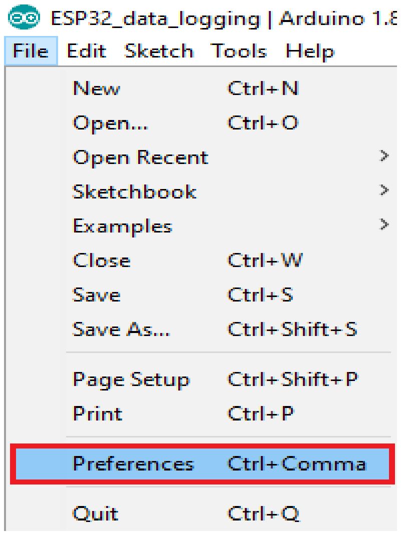
The process will extract and install all the required files to execute properly the Arduino Software (IDE)

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.

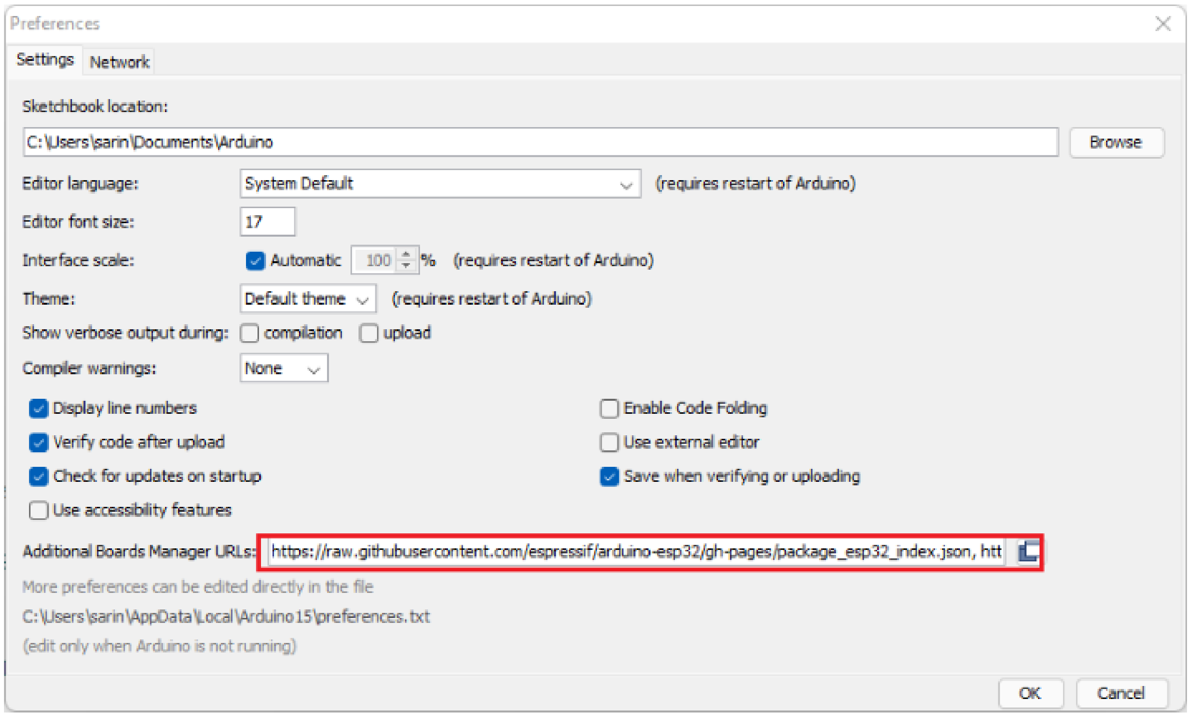
To install the ESP32 board in your Arduino IDE, follow these Steps:

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:



Preferences

Settings | Network

Sketchbook location: C:\Users\sarin\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 17

Interface scale: ☑ Automatic 100 % (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

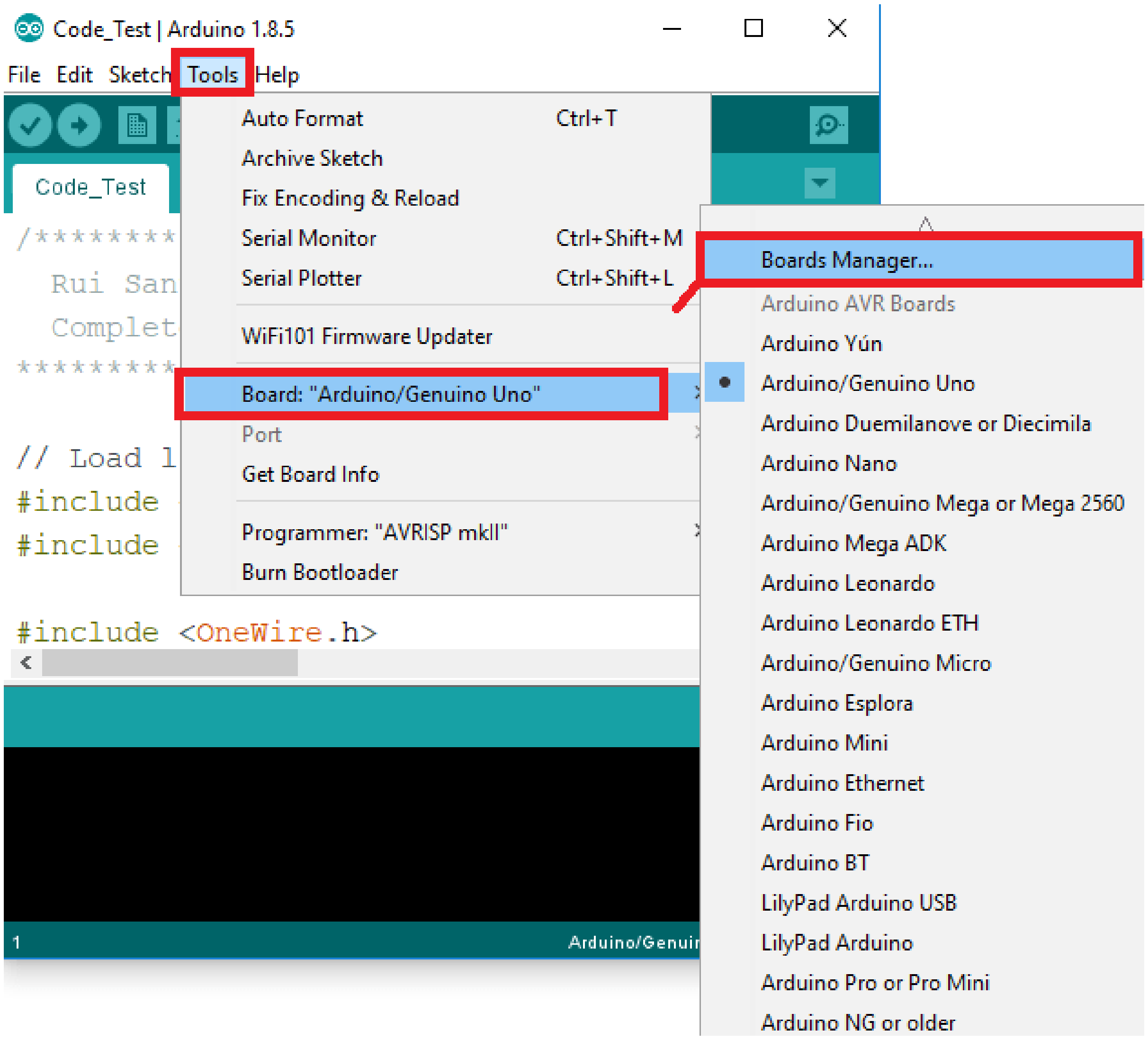
☑ Display line numbers
☑ Verify code after upload
☑ Check for updates on startup
☐ Use accessibility features
☐ Enable Code Folding
☐ Use external editor
☑ Save when verifying or uploading

Additional Boards Manager URLs: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json, htt

More preferences can be edited directly in the file
C:\Users\sarin\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

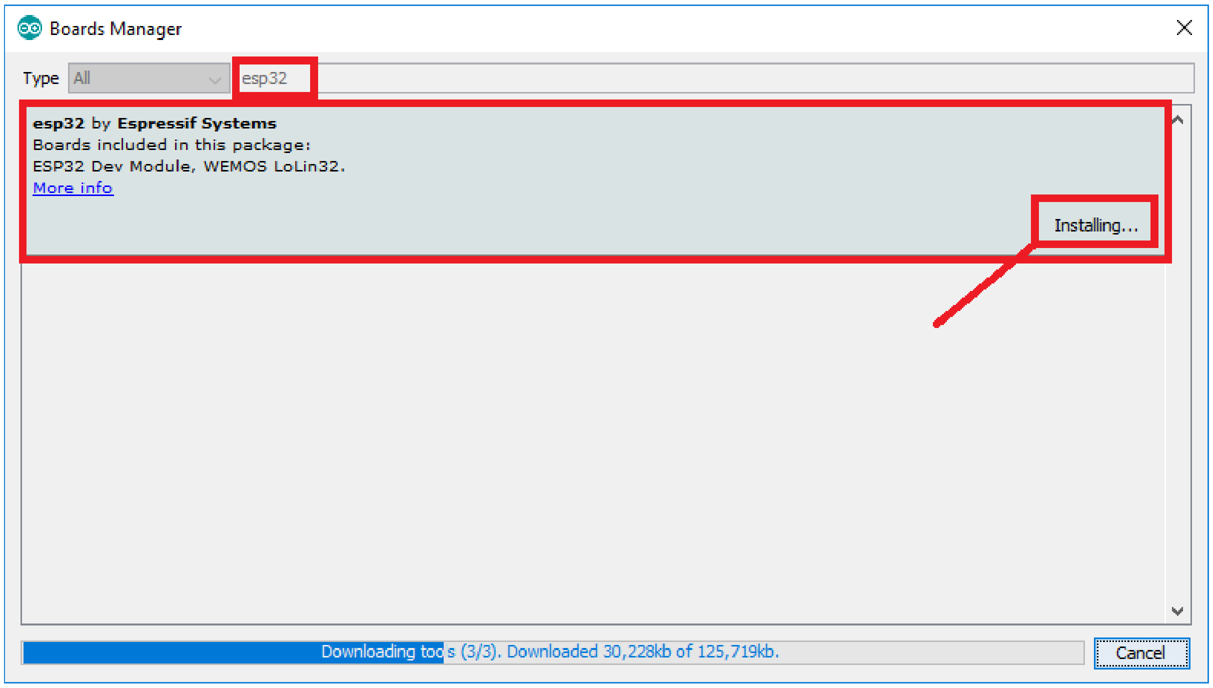
Note: if you already have the ESP8266 boards URL, you can separate the URLs with a comma as follows:
`https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json, http://arduino.esp8266.com/stable/package_esp8266com_index.json`

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



```
/********
  Rui San
  Complet
*********

// Load l
#include
#include

#include <OneWire.h>
```

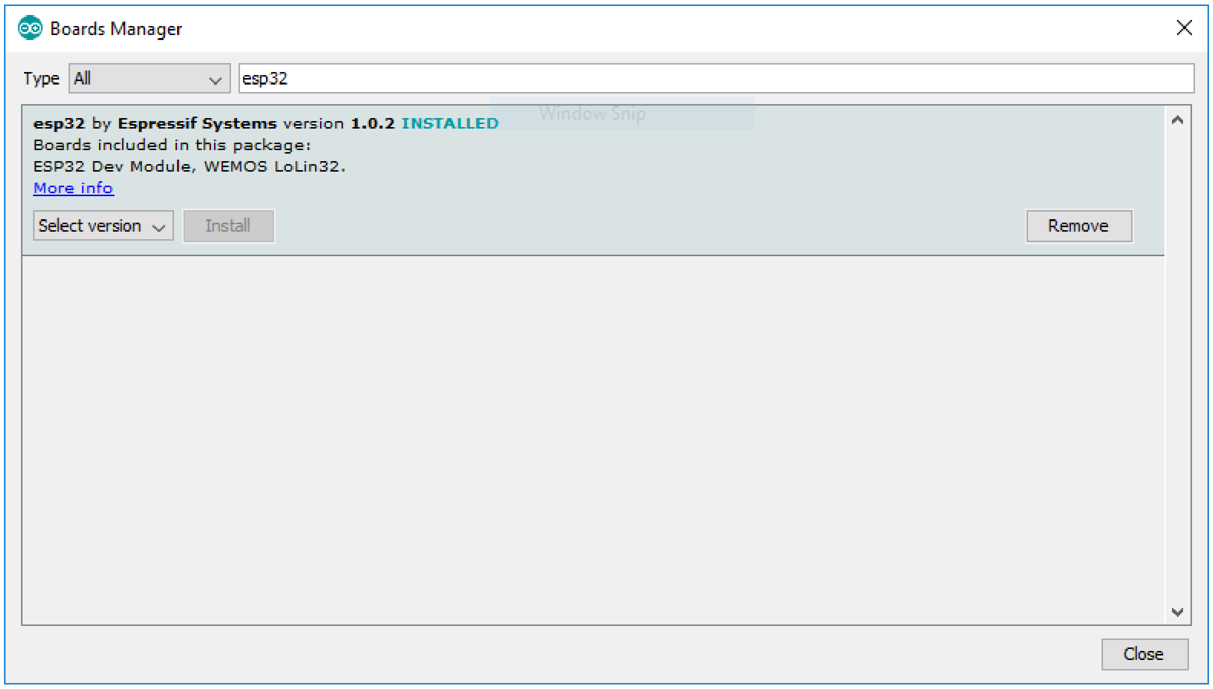
Step 4: Search for ESP32 and press install button for the "**ESP32 by Espressif Systems**":



esp32 by **Espressif Systems**
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32.
More info

Downloading tools (3/3). Downloaded 30,228kb of 125,719kb.

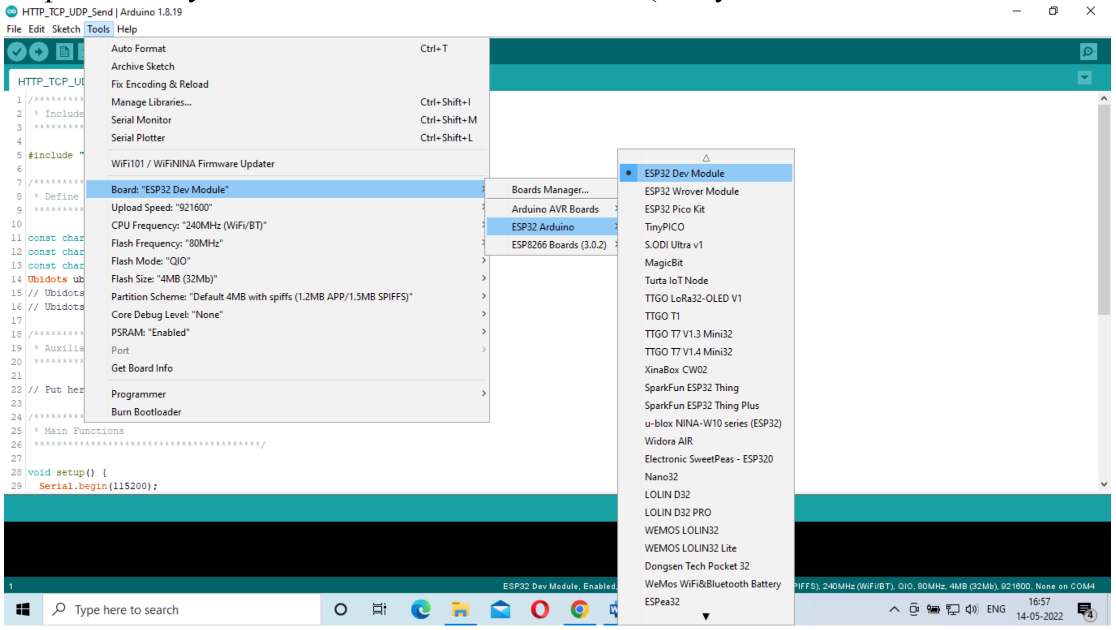
Step 5: That's it. It should be installed after a few seconds.



esp32 by **Espressif Systems** version **1.0.2** **INSTALLED**
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32.
More info

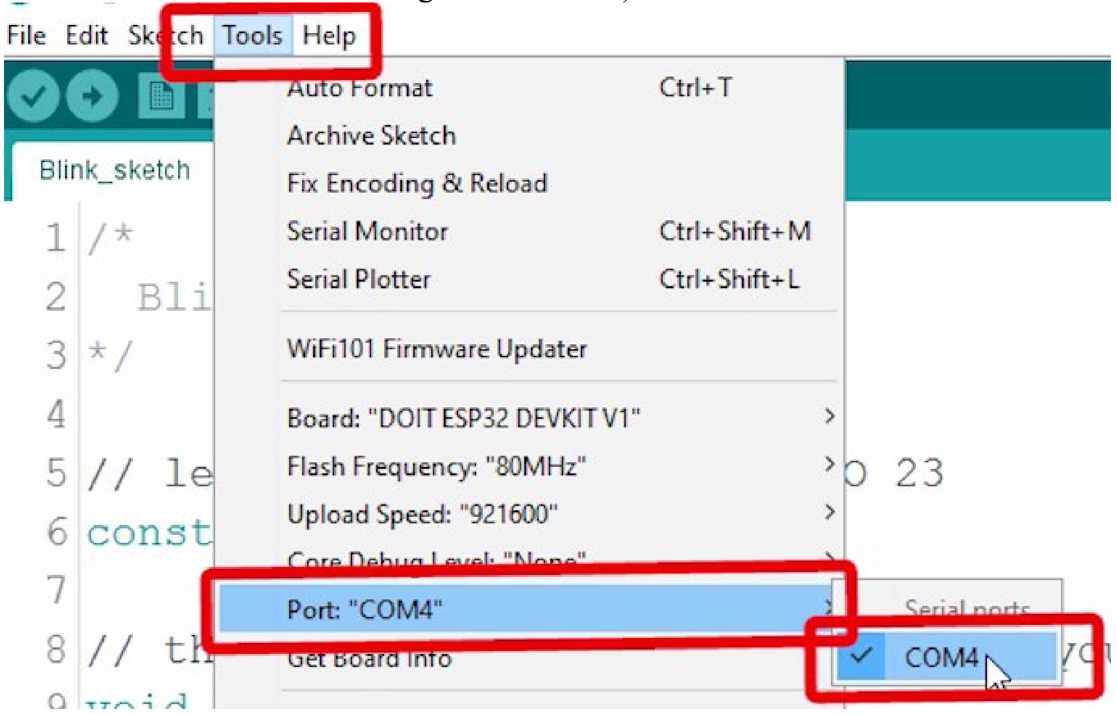
Testing the Installation:

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")



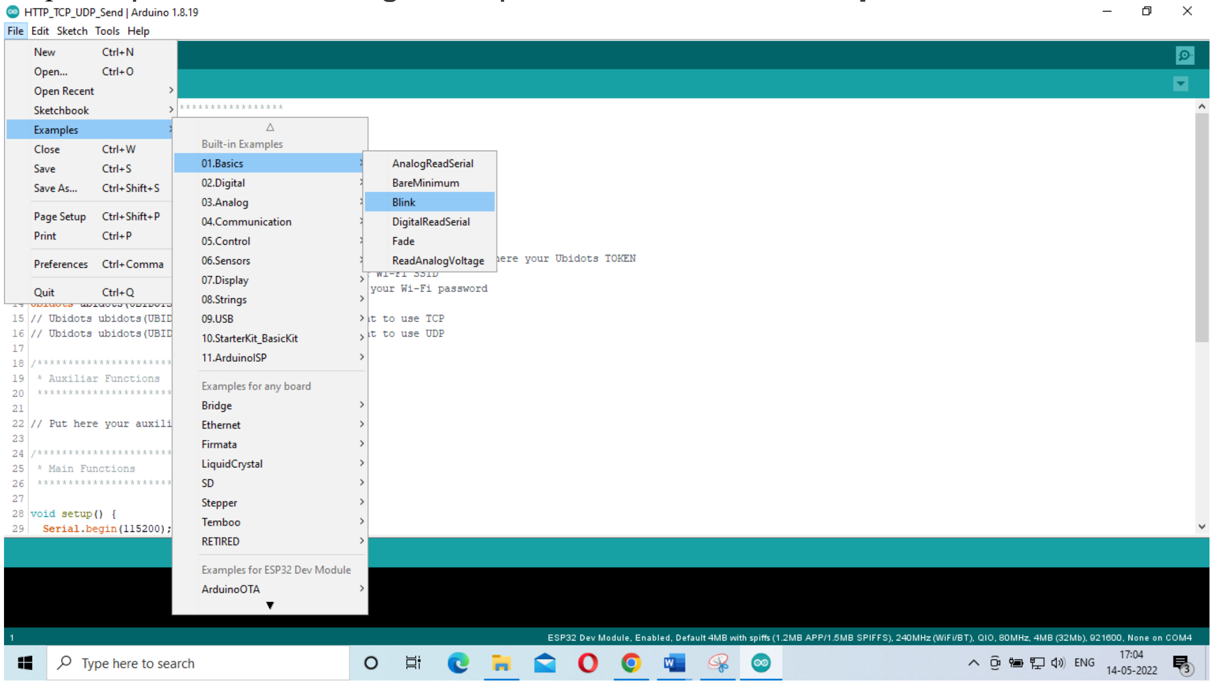
Board: "ESP32 Dev Module"
Upload Speed: "921600"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "4MB (32Mb)"
Partition Scheme: "Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)"
Core Debug Level: "None"
PSRAM: "Enabled"

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):



Board: "DOIT ESP32 DEVKIT V1"
Flash Frequency: "80MHz"
Upload Speed: "921600"
Core Debug Level: "None"
Port: "COM4"

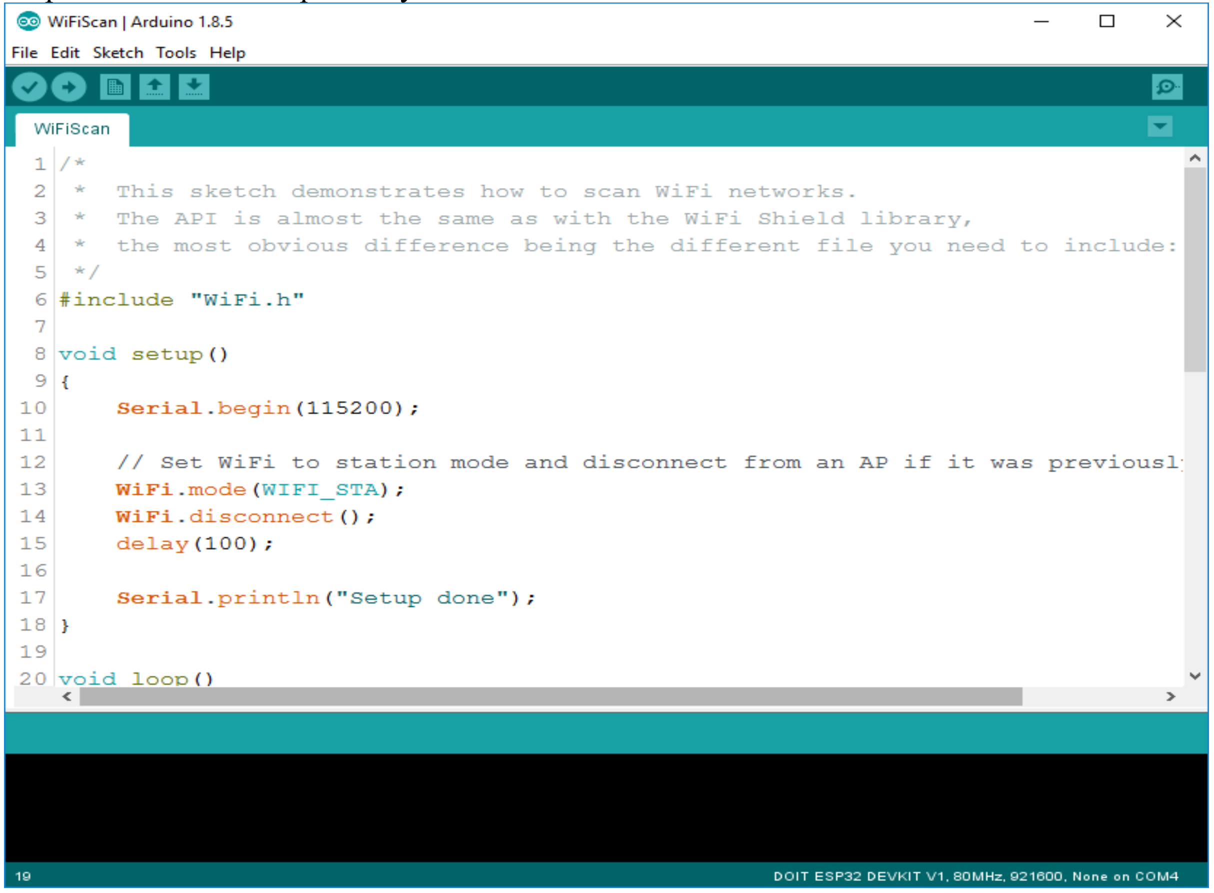
Step 3: Open the following example under **File** > **Examples >Basics> Blink**



```
// Ubidots ubidots(UBID
// Ubidots ubidots(UBID

/*********************
 * Auxiliar Functions
 *********************

// Put here your auxili

/*********************
 * Main Functions
 *********************

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

ESP32 Dev Module, Enabled, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, None on COM4

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

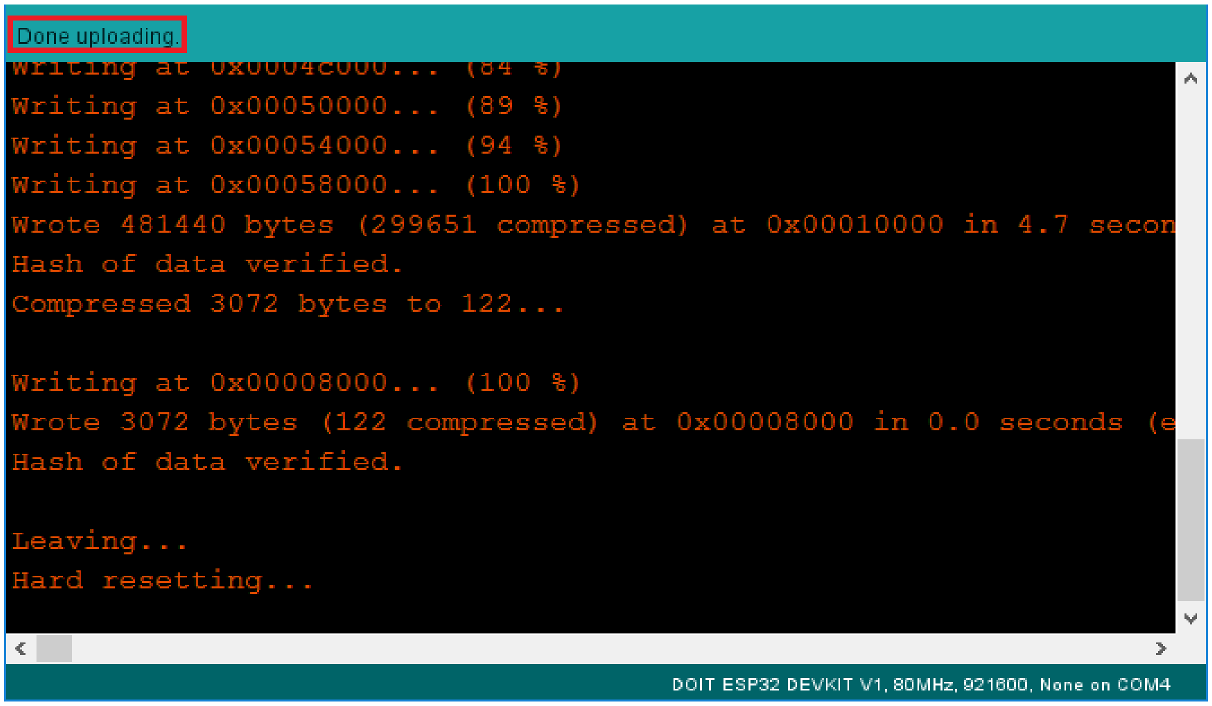


```
/*
 *  This sketch demonstrates how to scan WiFi networks.
 *  The API is almost the same as with the WiFi Shield library,
 *  the most obvious difference being the different file you need to include:
 */
#include "WiFi.h"

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

    // Set WiFi to station mode and disconnect from an AP if it was previousl
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);

    Serial.println("Setup done");
}

void loop()
```

19   DOIT ESP32 DEVKIT V1, 80MHz, 921600, None on COM4

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.



```
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 secon
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...
```

DOIT ESP32 DEVKIT V1, 80MHz, 921600, None on COM4

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

EXPERIMENT NO. 1

Aim: Connect an LED to an ESP32 and create a program that switches the LED on for 1 second every 2 seconds.

Description: Connect an LED to an ESP32 and create a program that switches the LED on for 1 and OFF every 2 seconds. Also, connect a push button or digital sensor (such as an IR sensor or LDR) to the ESP32 and develop a program that activates the LED when the button is pressed, or the sensor is triggered.

Apparatus:

Sl No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	LED	1
3	Micro USB cable	1

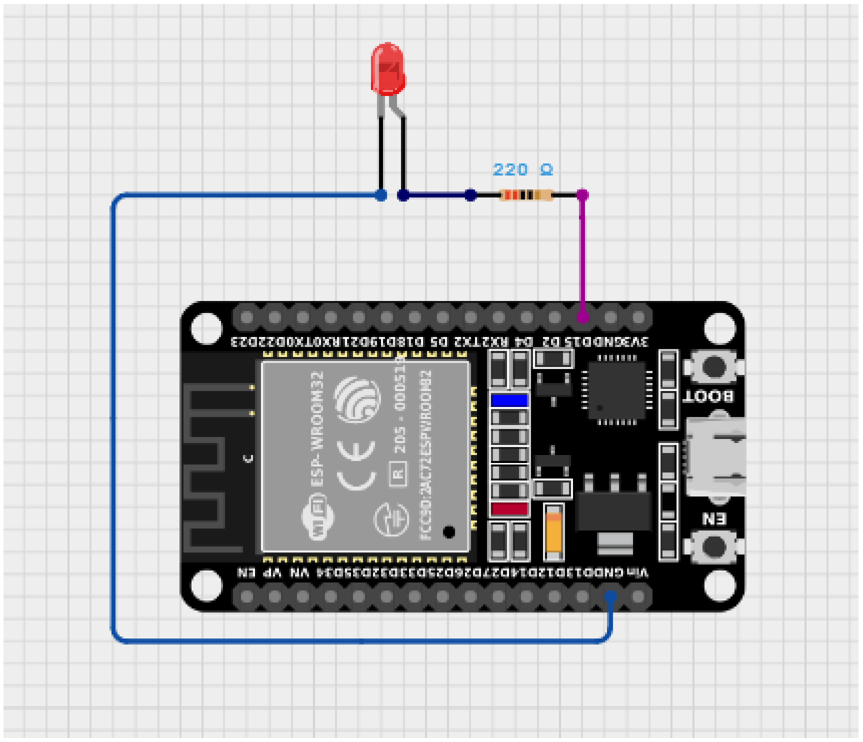


Figure: Connect an LED to an ESP32

Procedure:

1. Plug the ESP32 development board to your PC
2. Make the connection as per the circuit diagram
3. 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**.

4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a "Done uploading" message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

Code:

```
#define LED_PIN 15
void setup() {
pinMode(LED_PIN, OUTPUT);
}
void loop() {
// Blink LED every 2 seconds
digitalWrite(LED_PIN, HIGH);
delay(1000);
digitalWrite(LED_PIN, LOW);
delay(1000);
}
```

Work Sheet

Signature of Staff In-charge with date

EXPERIMENT NO. 2

Aim: Measure the temperature, humidity and heat index using ESP32 development board and display the values on serial monitor.

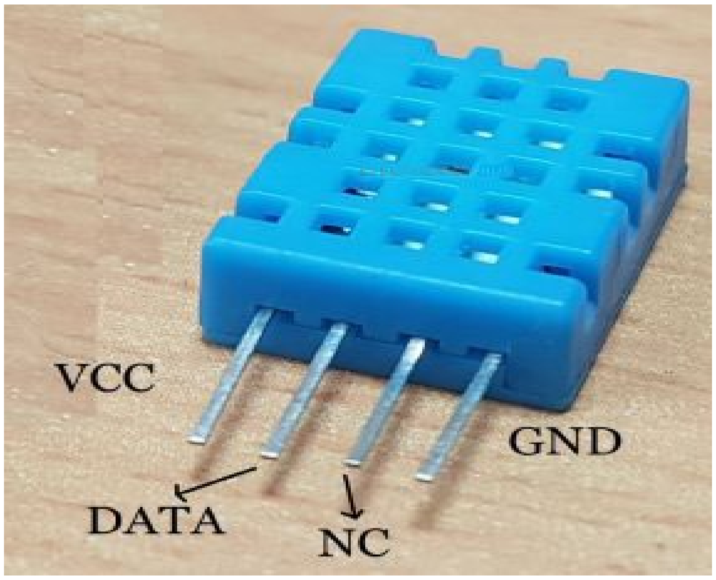
Apparatus:

Sl 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

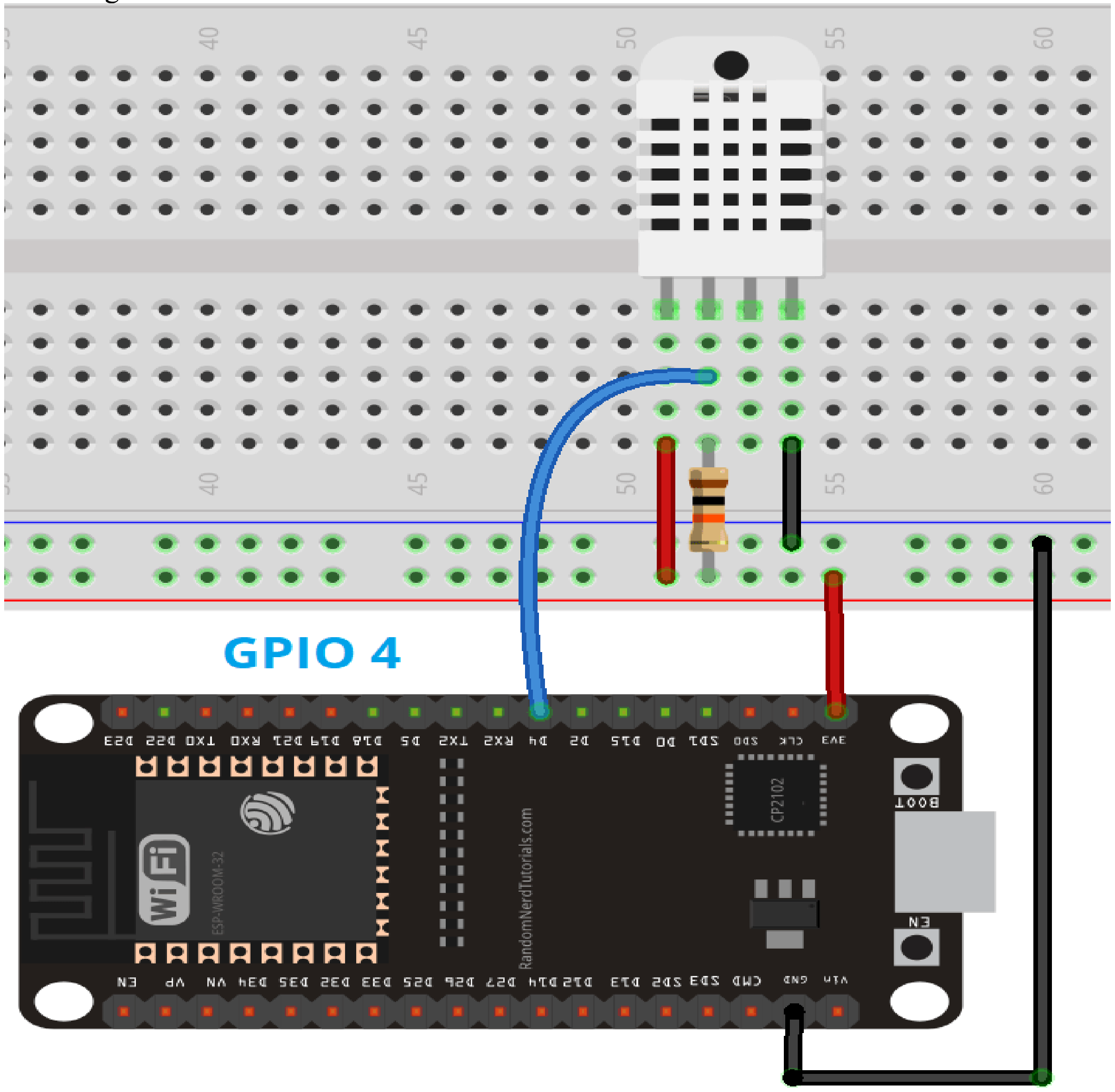
The DHT11 and DHT22 sensors are used to measure temperature and relative humidity.

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

DHT11 sensor:



Circuit diagram:



Procedure:

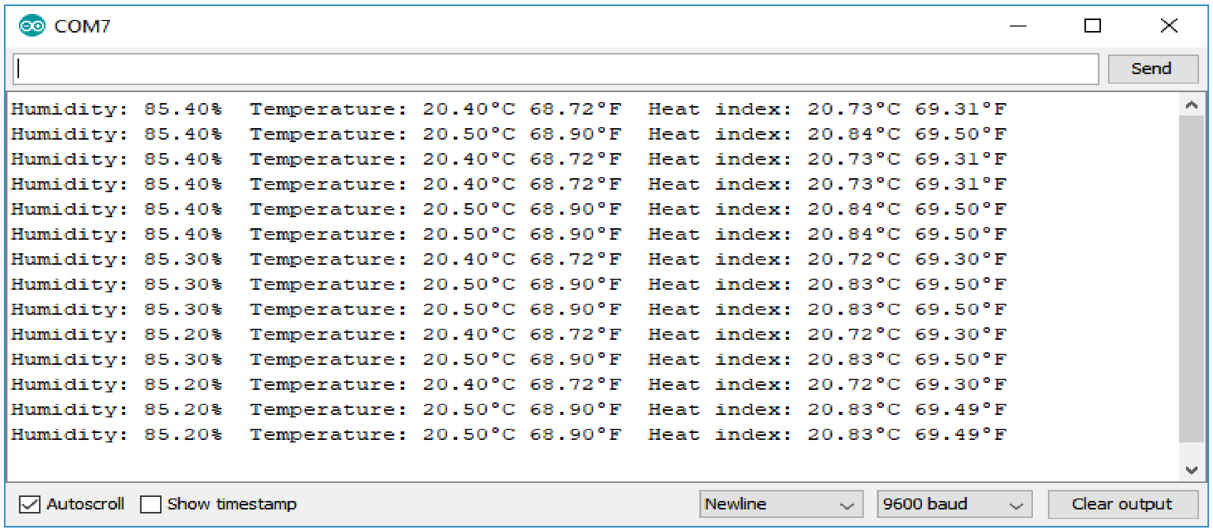
6. Plug the ESP32 development board to your PC
7. Make the connection as per the circuit diagram
8. Open the Arduino IDE in computer and write the program. Save the new sketch in your working directory

   Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings**.
9. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a "Done uploading" message. (You need to hold the ESP32 on-board Boot button while uploading).
10. 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("\xC2\xB0 C, "));
  Serial.print(f);
  Serial.print(F("\xC2\xB0 F,  Heat index: "));
  Serial.print(hic);
  Serial.print(F("\xC2\xB0 C, "));
  Serial.print(hif);
  Serial.println(F("\xC2\xB0 F."));
}
```

Result:



```
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: 85.20%  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: 85.20%  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
```

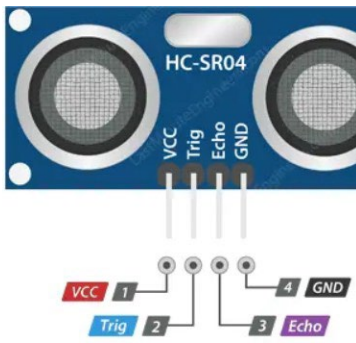
<u>Work Sheet</u>

Signature of Staff In-charge with date

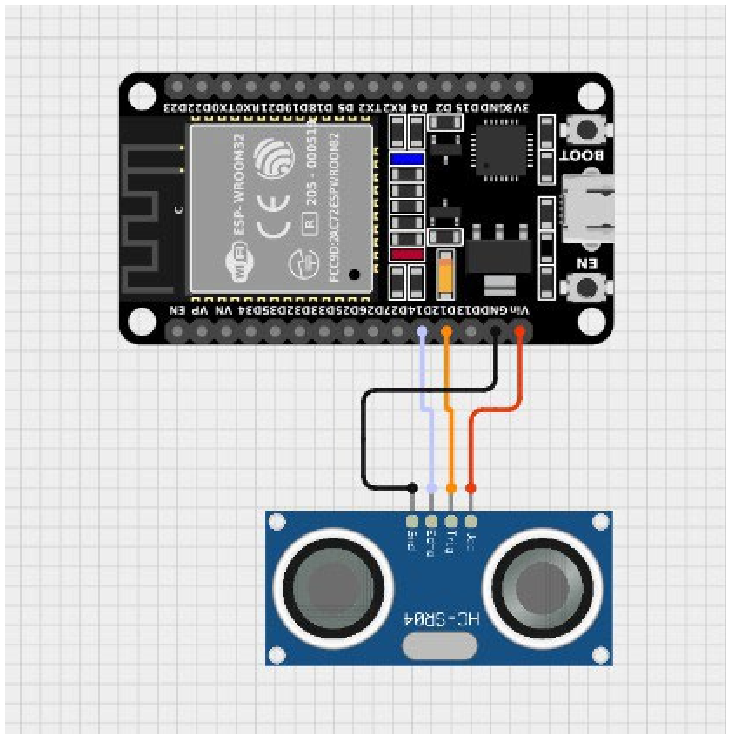
EXPERIMENT NO. 3

Aim: Calculate the distance to an object with the help of an ultrasonic sensor and display it on serial monitor.

Ultrasonic sensor:



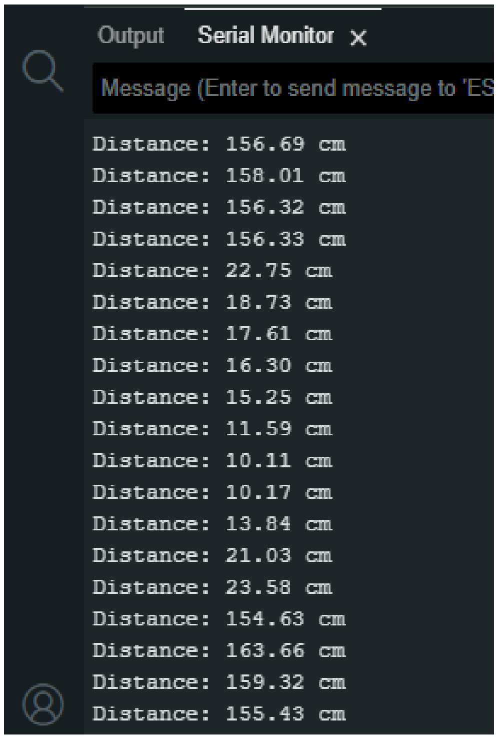
Circuit diagram:



Code:

```
#define TRIG_PIN 12
#define ECHO_PIN 14
void setup() {
Serial.begin(9600);
pinMode(TRIG_PIN, OUTPUT);
pinMode(ECHO_PIN, INPUT);
}
void loop() {
digitalWrite(TRIG_PIN, LOW);
delayMicroseconds(2);
digitalWrite(TRIG_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW);
long duration = pulseIn(ECHO_PIN, HIGH);
float distance = duration * 0.034 / 2;
Serial.print("Distance: ");
Serial.print(distance);
Serial.println(" cm");
delay(500);
}
```

Output on serial monitor:



```
Distance: 156.69 cm
Distance: 158.01 cm
Distance: 156.32 cm
Distance: 156.33 cm
Distance: 22.75 cm
Distance: 18.73 cm
Distance: 17.61 cm
Distance: 16.30 cm
Distance: 15.25 cm
Distance: 11.59 cm
Distance: 10.11 cm
Distance: 10.17 cm
Distance: 13.84 cm
Distance: 21.03 cm
Distance: 23.58 cm
Distance: 154.63 cm
Distance: 163.66 cm
Distance: 159.32 cm
Distance: 155.43 cm
```

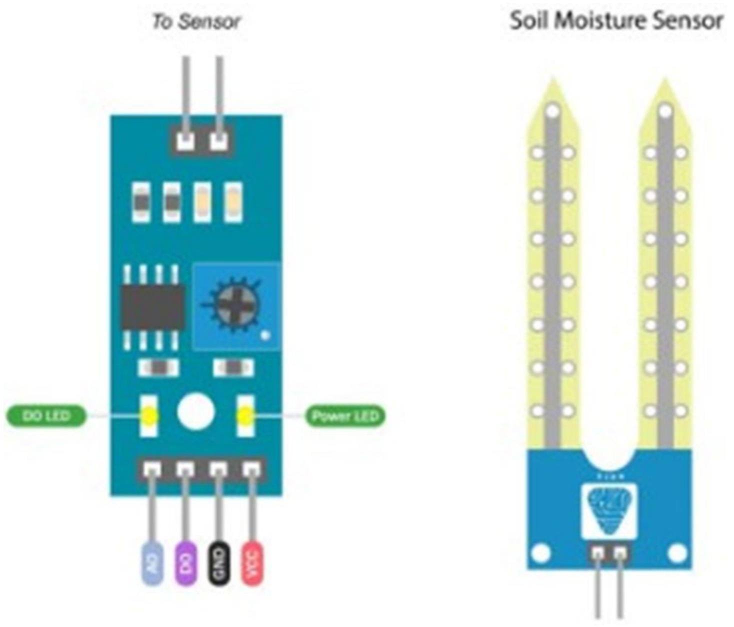
Work Sheet

Signature of Staff In-charge with date

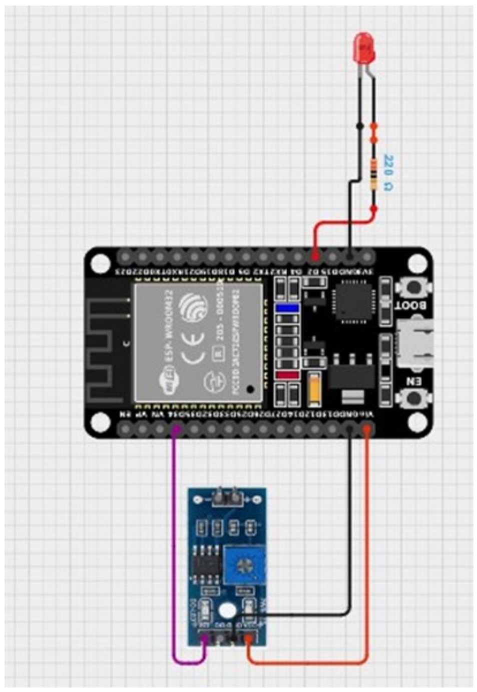
EXPERIMENT NO. 4

Aim: Measure soil moisture value using ESP32 development board and display the sensor values on serial monitor.

Soil moisture sensor:



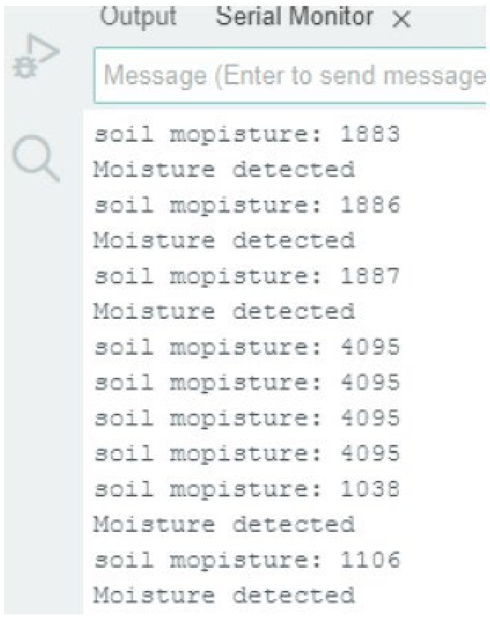
Circuit diagram:



Code:

```
#define soil_PIN 34
 #define LED_PIN 2
int THRESHOLD=2500;
void setup() {
pinMode(soil_PIN, OUTPUT);
Serial.begin(115200);
}

void loop() {
int soilValue = analogRead(soil_PIN);
Serial.print("soil mopisture: ");
Serial.println(soilValue);

if (soilValue < THRESHOLD)
{
digitalWrite(soil_PIN, HIGH);
Serial.println("Moisture detected");
}
else
{
digitalWrite(soil_PIN, LOW);
}
delay(1000);
}
```

Output on serial monitor:



```
soil mopisture: 1883
Moisture detected
soil mopisture: 1886
Moisture detected
soil mopisture: 1887
Moisture detected
soil mopisture: 4095
soil mopisture: 4095
soil mopisture: 4095
soil mopisture: 4095
soil mopisture: 1038
Moisture detected
soil mopisture: 1106
Moisture detected
```

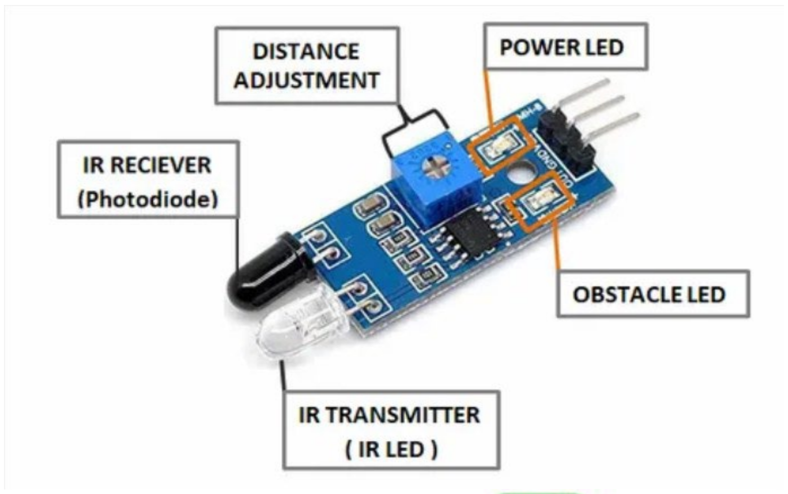
Work Sheet

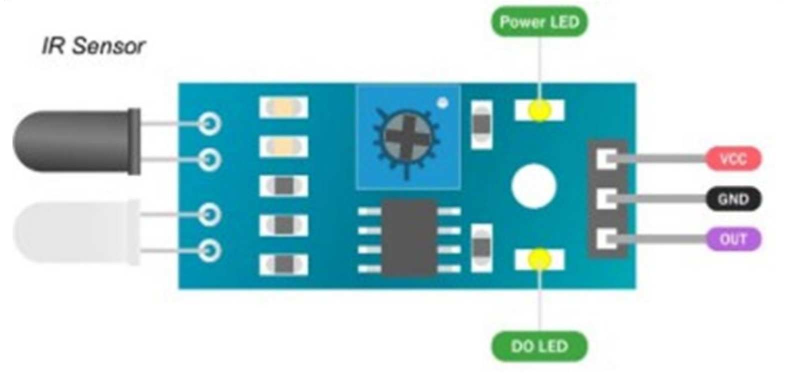
Signature of Staff In-charge with date

EXPERIMENT NO. 5

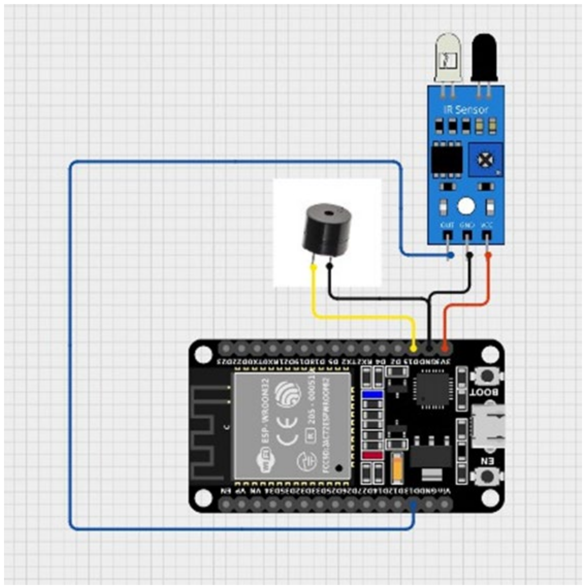
<u>Aim:</u> Motion detection using IR sensor and turn on buzzer upon detection.

IR sensor:





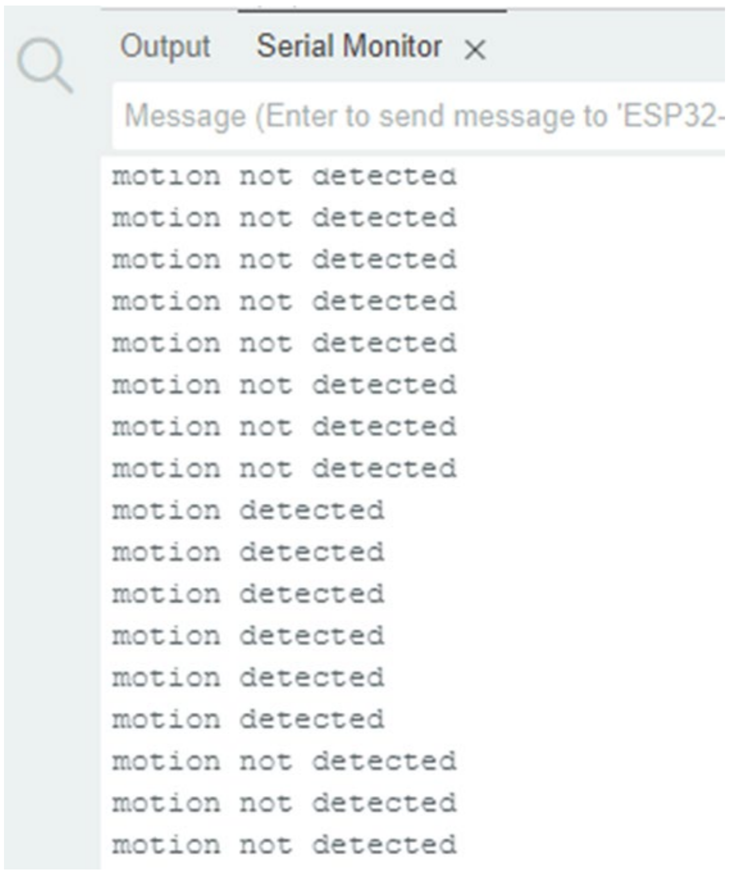
Circuit diagram:



Code:

```
#define IR_PIN 13
void setup() {
Serial.begin(9600);
pinMode(IR_PIN, INPUT);
pinMode(15, OUTPUT);
}
void loop() {
if (digitalRead(IR_PIN) == LOW)
{
Serial.println("Motion detected");
digitalWrite(15,HIGH);
}
else{
Serial.println("Motion not detected");
digitalWrite(15,LOW);
}
delay(200);
}
```

Output on serial monitor:



```
motion not detected
motion not detected
motion not detected
motion not detected
motion not detected
motion not detected
motion not detected
motion not detected
motion detected
motion detected
motion detected
motion detected
motion detected
motion detected
motion not detected
motion not detected
motion not detected
```

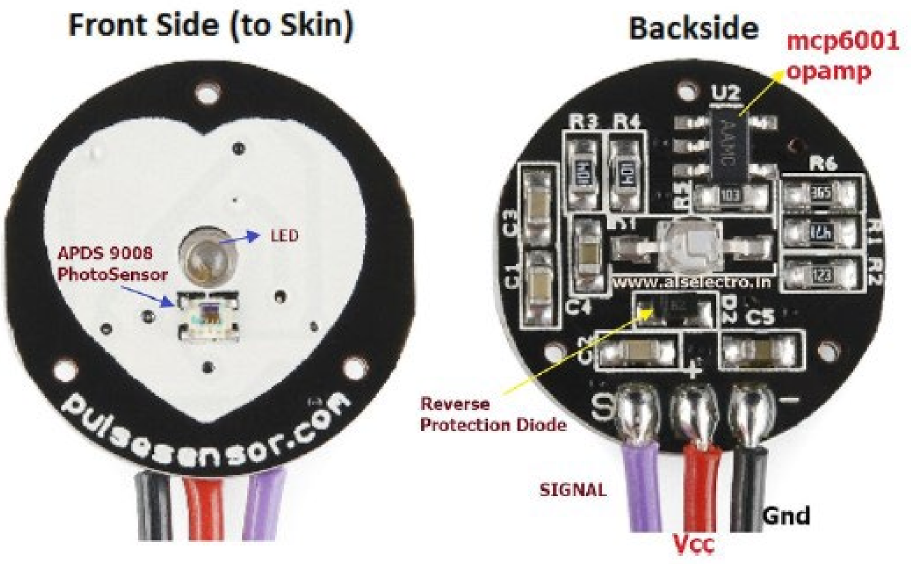
Work Sheet

Signature of Staff In-charge with date

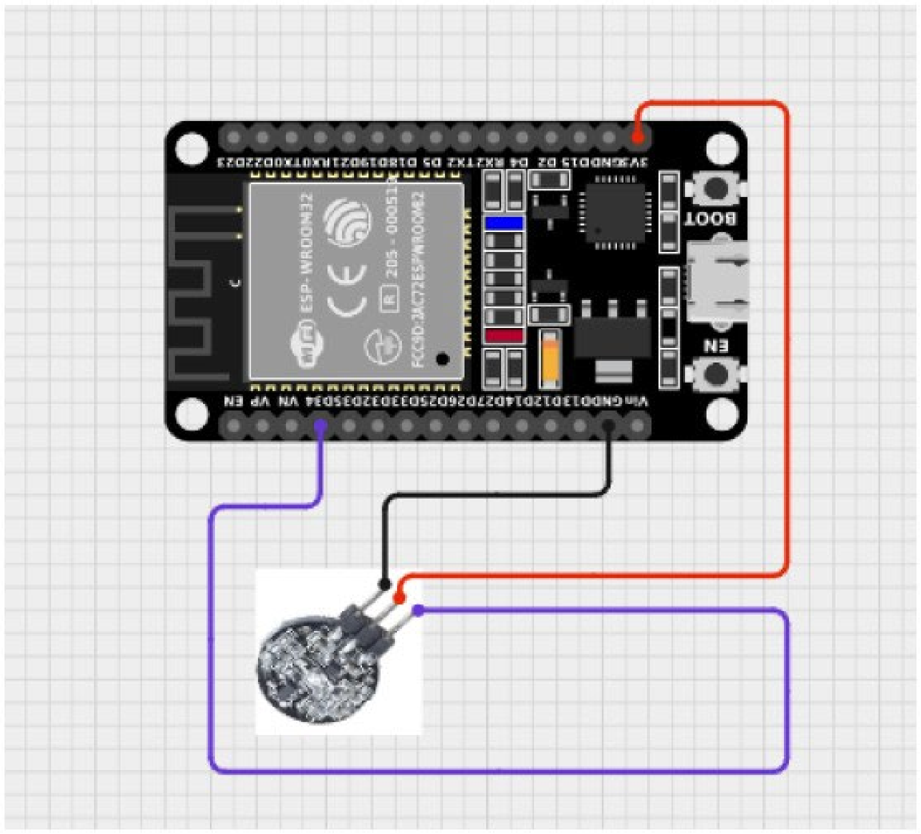
EXPERIMENT NO. 6

Aim: Measure BPM value using pulse sensor and display on serial monitor.

Pulse sensor:



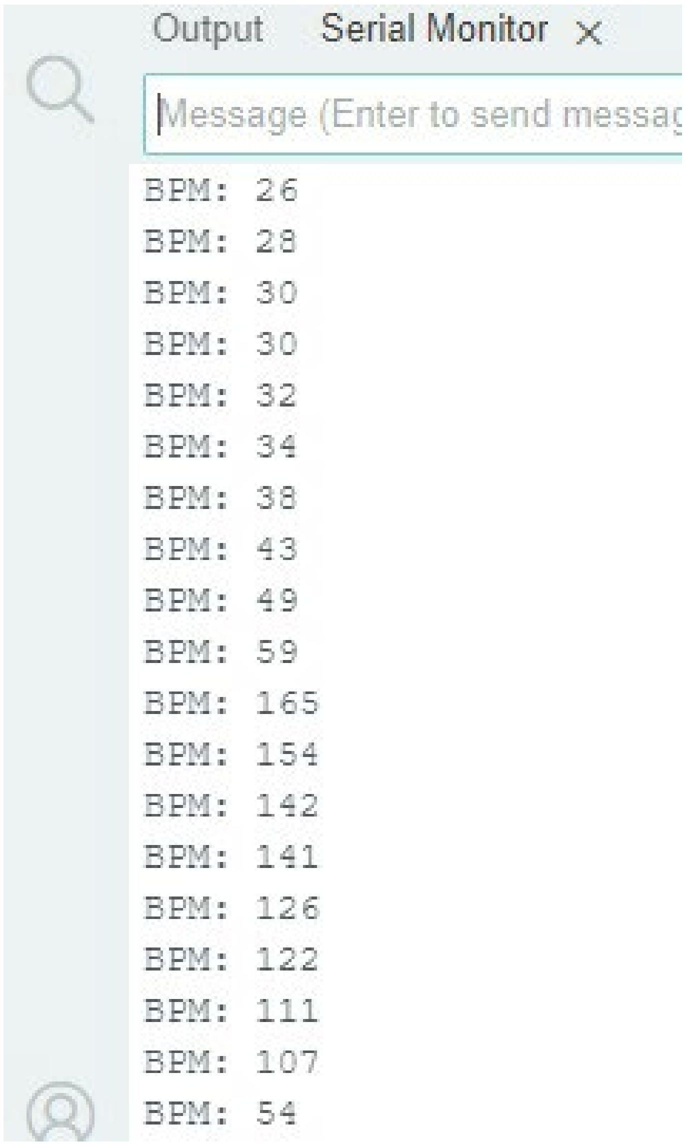
Circuit diagram:



Code:

```
#include <PulseSensorPlayground.h>
#define PULSE_SENSOR_PIN 34 // Analog pin connected to pulse sensor
#define THRESHOLD 550     // Adjust based on your sensor reading

PulseSensorPlayground pulseSensor; // Create PulseSensor object

void setup()
{
Serial.begin(115200);// Configure the pulse sensor
pulseSensor.analogInput(PULSE_SENSOR_PIN);
pulseSensor.setThreshold(THRESHOLD);

if (pulseSensor.begin())
{
Serial.println("Pulse sensor initialized!");
}
else
{
Serial.println("Failed to initialize pulse sensor.");
}
}

void loop()
{
int myBPM = pulseSensor.getBeatsPerMinute(); // Get BPM value
if (pulseSensor.sawStartOfBeat())
{
// If a new beat is detected Serial.print("BPM: ");
Serial.println(myBPM);
}
delay(20); // Small delay to stabilize readings
}
```

Result



```
BPM: 26
BPM: 28
BPM: 30
BPM: 30
BPM: 32
BPM: 34
BPM: 38
BPM: 43
BPM: 49
BPM: 59
BPM: 165
BPM: 154
BPM: 142
BPM: 141
BPM: 126
BPM: 122
BPM: 111
BPM: 107
BPM: 54
```

Work Sheet

Signature of Staff In-charge with date

EXPERIMENT NO. 7

Aim: Configure/Set your ESP32 development board as an access point.

Description: 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.

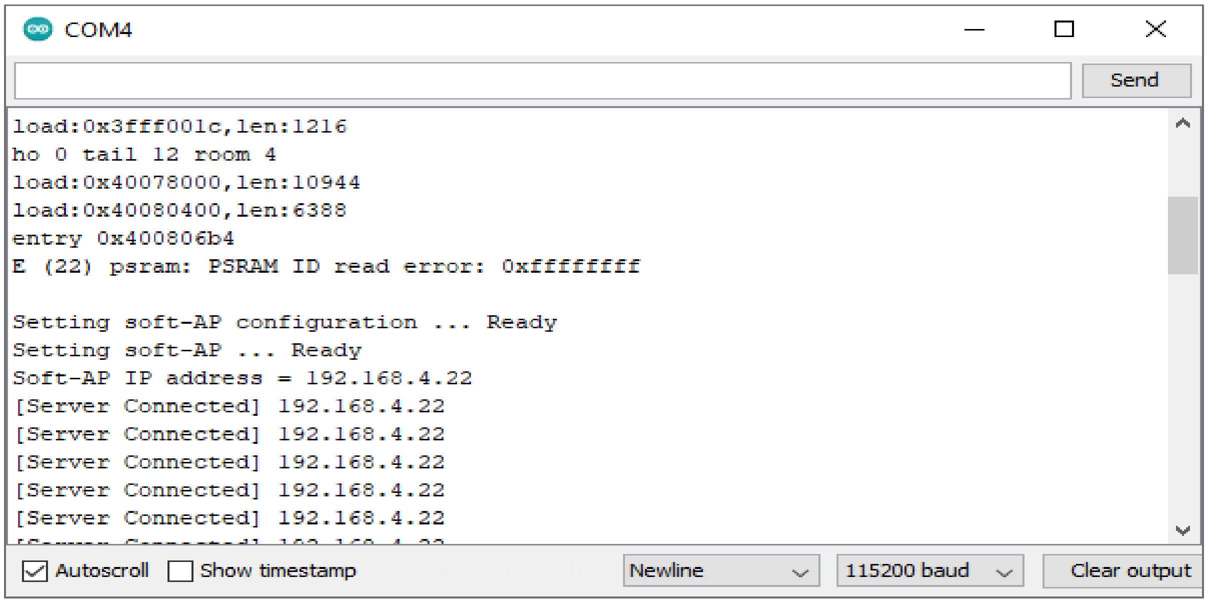
Procedure:

1. Plug the ESP32 development board to your PC
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. Define a SSID name and a password to access the ESP32 development board
4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a "Done uploading" message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

Code:

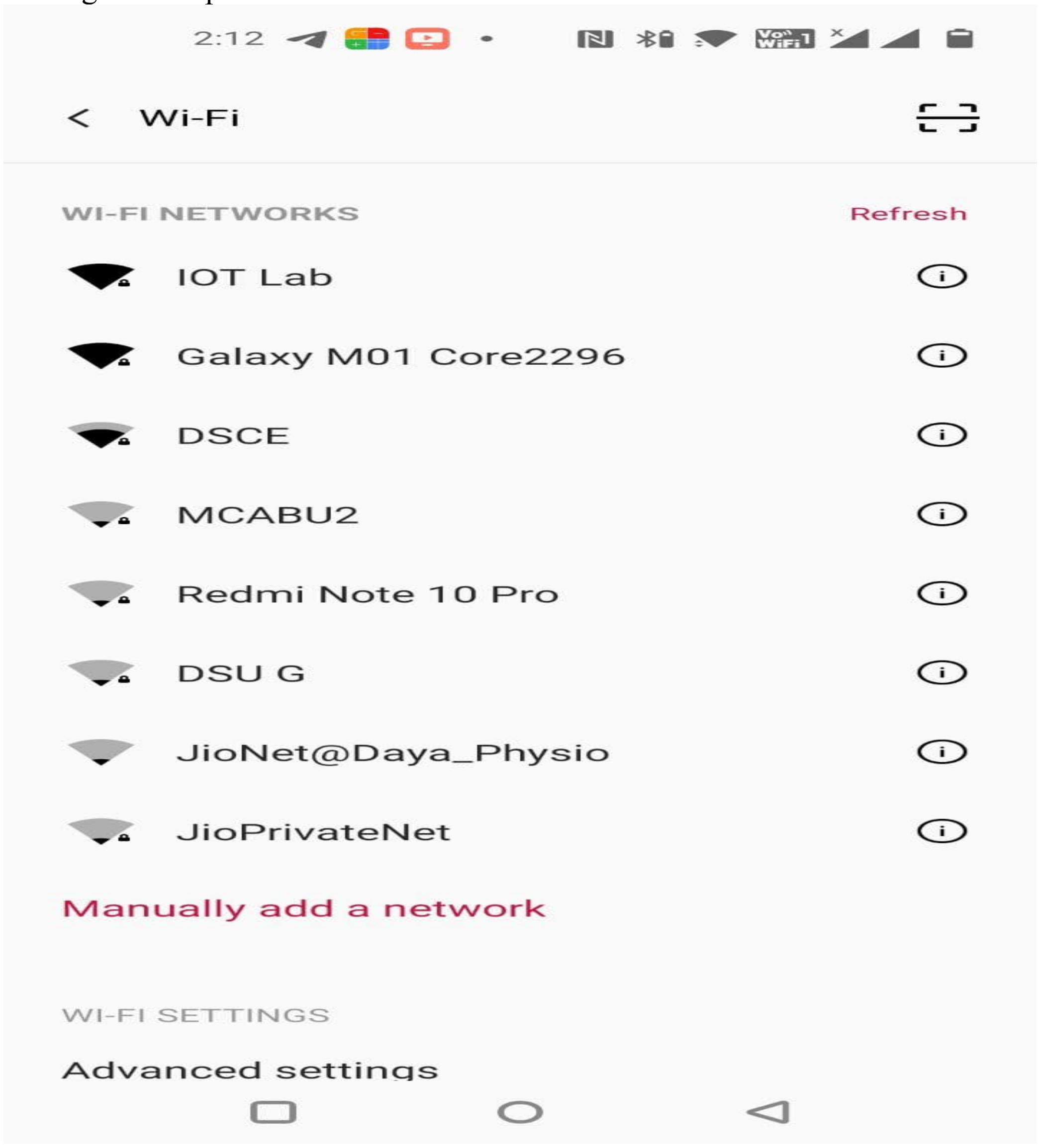
```
#include <WiFi.h>
const char *ssid = "SSID_Name_your_Choice";
const char *password = " Your_Choice(min 6 character) ";
IPAddress local_IP(192,168,4,22);
IPAddress gateway(192,168,4,9);
IPAddress subnet(255,255,255,0);
void setup()
{
 Serial.begin(115200);
 Serial.println();
 Serial.print("Setting soft-AP configuration ... ");
 Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");
 Serial.print("Setting soft-AP ... ");
 Serial.println(WiFi.softAP(ssid,password) ? "Ready" : "Failed!");
 //WiFi.softAP(ssid);
 //WiFi.softAP(ssid, password, channel, ssdi_hidden, max_connection)

 Serial.print("Soft-AP IP address = ");
 Serial.println(WiFi.softAPIP());
}
void loop() {
 Serial.print("[Server Connected] ");
 Serial.println(WiFi.softAPIP());
 delay(500);
}
```

Result:



```
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
E (22) psram: PSRAM ID read error: 0xffffffff

Setting soft-AP configuration ... Ready
Setting soft-AP ... Ready
Soft-AP IP address = 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
[Server Connected] 192.168.4.22
```

Through Smart phone:



2:12

< Wi-Fi

WI-FI NETWORKS Refresh

IOT Lab

Galaxy M01 Core2296

DSCE

MCABU2

Redmi Note 10 Pro

DSU G

JioNet@Daya_Physio

JioPrivateNet

Manually add a network

WI-FI SETTINGS

Advanced settings

Work Sheet

Signature of Staff In-charge with date

EXPERIMENT NO. 8

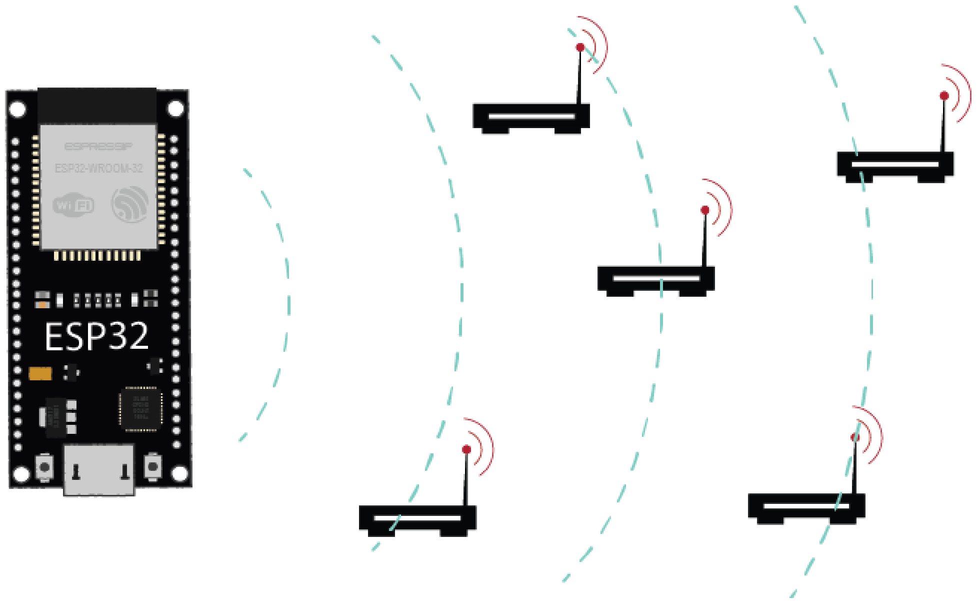
Aim: Scan Wi-Fi networks and get Wi-Fi strength using ESP32 development board.

ESP32 Wi-Fi Modes:
The ESP32 board can act as Wi-Fi Station, Access Point or both. To set the Wi-Fi mode, use WiFi.mode() and set the desired mode as argument:

WiFi.mode(WIFI_STA)	**station mode:** the ESP32 connects to an access point
WiFi.mode(WIFI_AP)	**access point mode:** stations can connect to the ESP32
WiFi.mode(WIFI_STA_AP)	**access point and a station** connected to another access point

Apparatus:

Sl No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	Micro USB cable	1

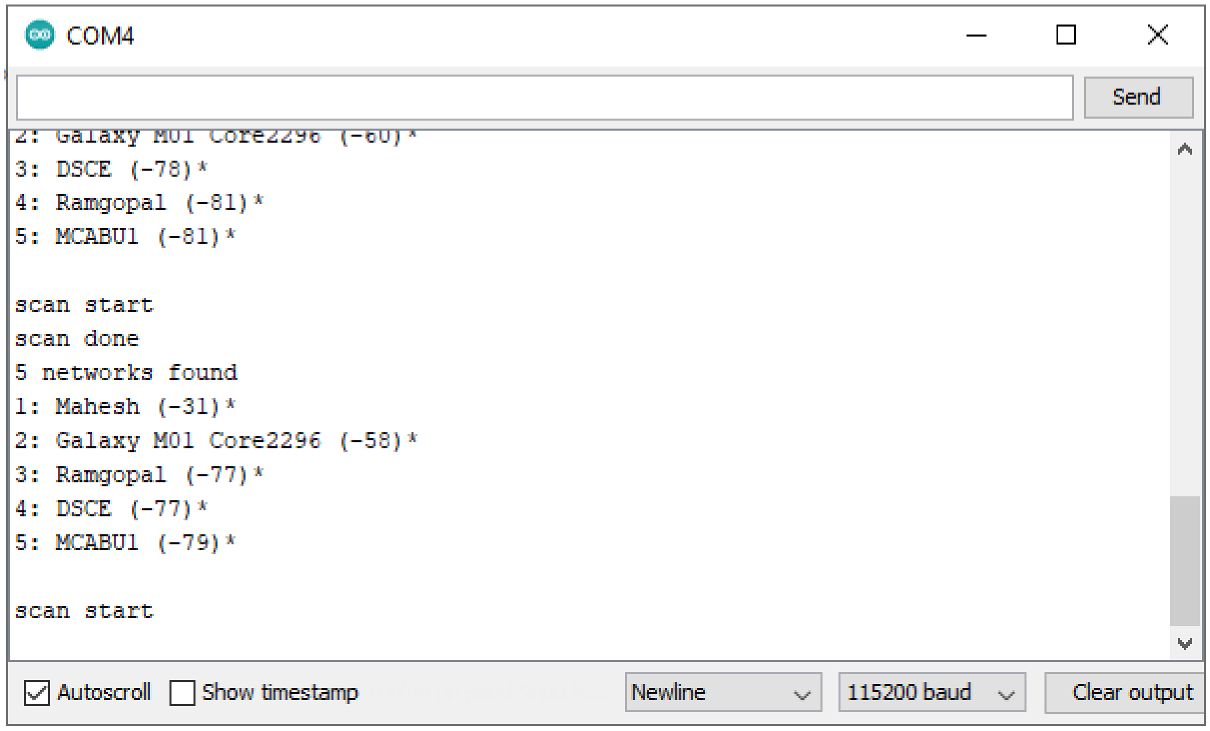


Procedure:
1. Plug the ESP32 development board to your PC
2. Make the connection as per the circuit diagram
3. Open the Arduino IDE in computer and write the program. Save the new sketch in your working directory
   Note: **Make sure you have the right board and COM port selected in your Arduino IDE settings**.
4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a "Done uploading" message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

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);
}
```

Result:


COM4

Send

```
2: Galaxy M01 Core2296 (-60)*
3: DSCE (-78)*
4: Ramgopal (-81)*
5: MCABU1 (-81)*

scan start
scan done
5 networks found
1: Mahesh (-31)*
2: Galaxy M01 Core2296 (-58)*
3: Ramgopal (-77)*
4: DSCE (-77)*
5: MCABU1 (-79)*

scan start
```

☑ Autoscroll ☐ Show timestamp Newline 115200 baud Clear output

Work Sheet

Signature of Staff In-charge with date

EXPERIMENT NO. 9

Aim: Establish connection between IoT node and access point and display of local IP and gateway IP.

Description: 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.

Smartphone generates the WIFI network, acts like an Access Point and is assigned a Gateway IP

ESP32 acts as IOT Node, connects to a WIFI network and is assigned a Local IP Address

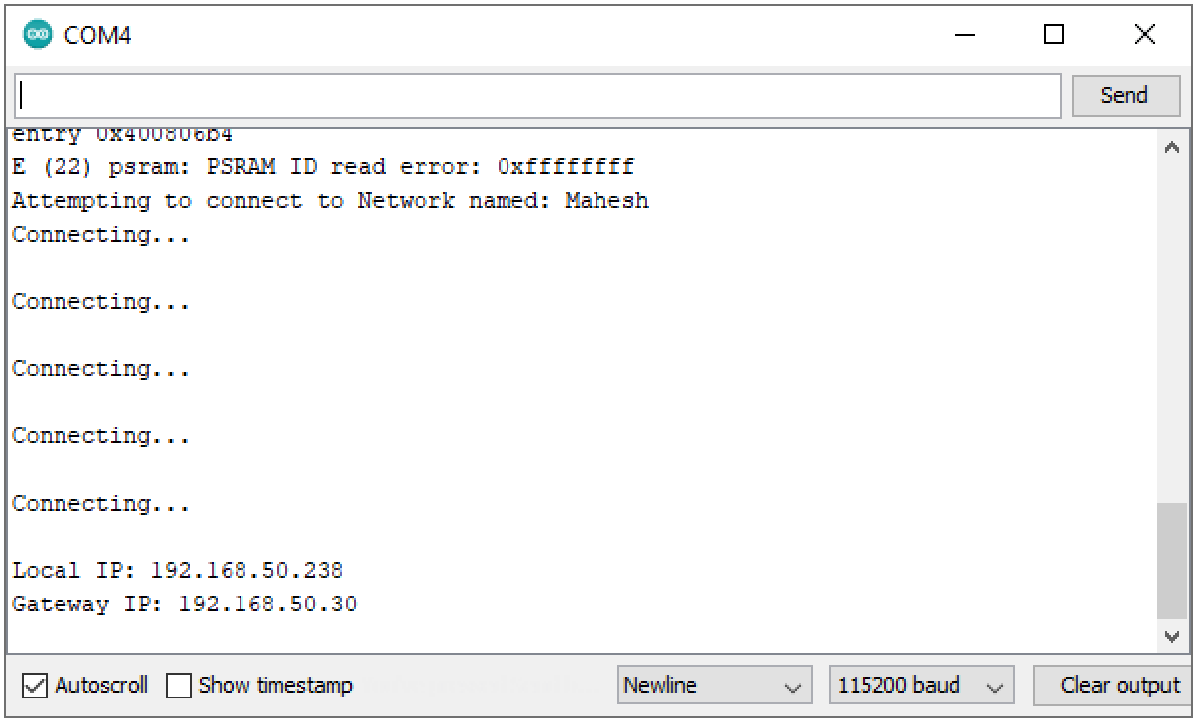
Procedure:

1. Plug the ESP32 development board to your PC
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. Define a SSID name and a password to access the ESP32 development board
4. Compile the program and upload it to the ESP32 Development Board. If everything went as expected, you should see a “Done uploading” message. (You need to hold the ESP32 on-board Boot button while uploading).
5. After uploading the code, open the Serial Monitor at a baud rate of 115200.

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:
}
```

Result:



```
entry 0x400806b4
E (22) psram: PSRAM ID read error: 0xffffffff
Attempting to connect to Network named: Mahesh
Connecting...

Connecting...

Connecting...

Connecting...

Connecting...

Local IP: 192.168.50.238
Gateway IP: 192.168.50.30
```

Work Sheet

Signature of Staff In-charge with date

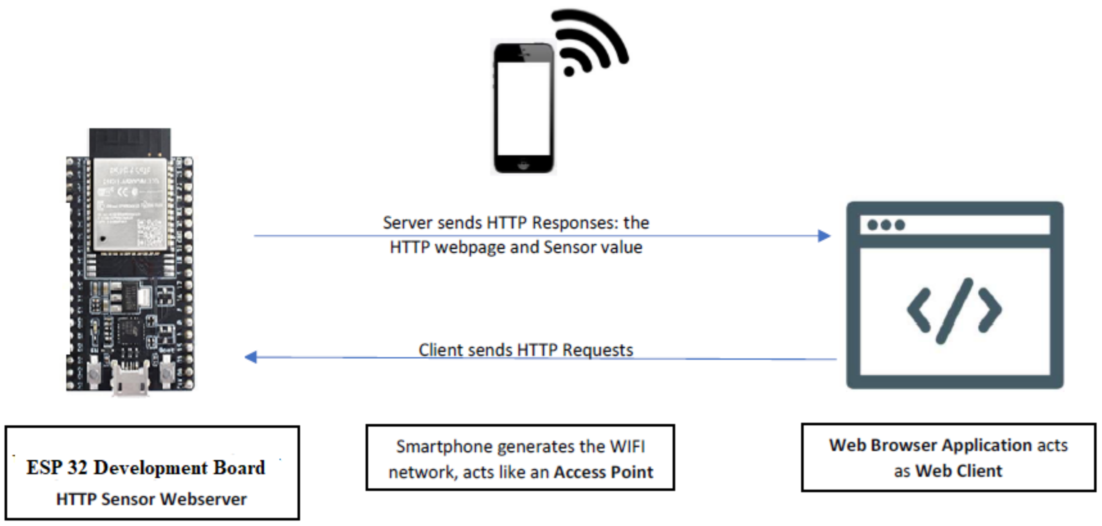
EXPERIMENT NO. 10

<u>Aim:</u> Design and implementation of HTTP based IoT web server to control the status of LED.

Apparatus:

Sl 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

Description: An HTTP based webserver provides access to data to an HTTP client such as web browser. In IoT applications, the IoT Nodes are connected to sensors and actuators. The sensor data can be accessed using a web browser and also the actuators can be controlled from the web browser. This facilitates a wide variety of applications in IoT domain. The underlying protocol used for this communication between a Client and a Server is HTTP. The goal of this lab is to understand the configuration and working principle of an IoT web server. Using a browser, such as google chrome, we will send some HTTP based commands to the web server. Based on the type of command, the IoT Node web server will toggle the LEDs.

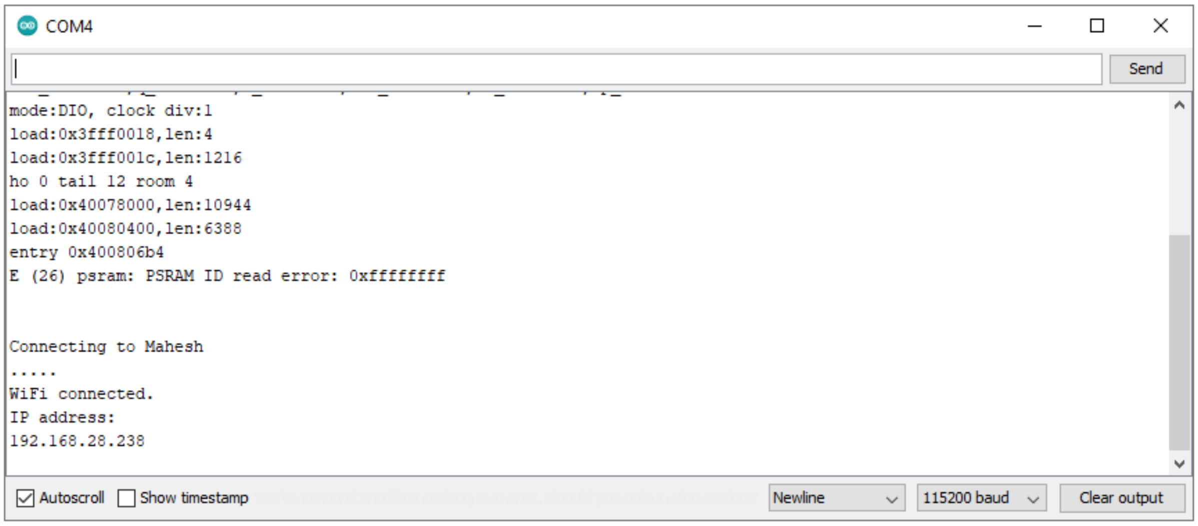

Server sends HTTP Responses: the HTTP webpage and Sensor value
Client sends HTTP Requests
ESP 32 Development Board
HTTP Sensor Webserver
Smartphone generates the WIFI network, acts like an Access Point
Web Browser Application acts as Web Client

Code:

```
#include <WiFi.h>

const char* ssid     = "Mahesh";
const char* password = "012345678";

WiFiServer server(80);

void setup()
{
   Serial.begin(115200);
   pinMode(2, OUTPUT);      // set the LED pin mode

   delay(10);

   // We start by connecting to a WiFi network

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

   WiFi.begin(ssid, password);

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

   Serial.println("");
   Serial.println("WiFi connected.");
   Serial.println("IP address: ");
   Serial.println(WiFi.localIP());

   server.begin();

}

int value = 0;

void loop(){
 WiFiClient client = server.available();   // listen for incoming clients

  if (client) {                             // if you get a client,
   Serial.println("New Client.");           // print a message out the serial port
   String currentLine = "";                // make a String to hold incoming data from the client
   while (client.connected()) {            // loop while the client's connected
```

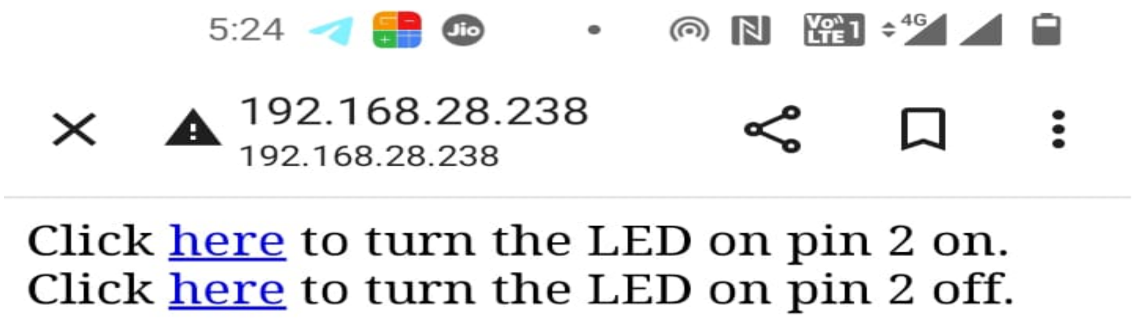
```
        if (client.available()) {             // if there's bytes to read from the client,
          char c = client.read();             // read a byte, then
          Serial.write(c);                    // print it out the serial monitor
          if (c == '\n') {                    // if the byte is a newline character

            // if the current line is blank, you got two newline characters in a row.
            // that's the end of the client HTTP request, so send a response:
            if (currentLine.length() == 0) {
              // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
              // and a content-type so the client knows what's coming, then a blank line:
              client.println("HTTP/1.1 200 OK");
              client.println("Content-type:text/html");
              client.println();

              // the content of the HTTP response follows the header:
              client.print("Click <a href=\"/H\">here</a> to turn the LED on pin 2 on.<br>");
              client.print("Click <a href=\"/L\">here</a> to turn the LED on pin 2 off.<br>");

              // The HTTP response ends with another blank line:
              client.println();
              // break out of the while loop:
              break;
            } else {    // if you got a newline, then clear currentLine:
              currentLine = "";
            }
          } else if (c != '\r') {  // if you got anything else but a carriage return character,
            currentLine += c;      // add it to the end of the currentLine
          }

          // Check to see if the client request was "GET /H" or "GET /L":
          if (currentLine.endsWith("GET /H")) {
            digitalWrite(2, HIGH);               // GET /H turns the LED on
          }
          if (currentLine.endsWith("GET /L")) {
            digitalWrite(2, LOW);                // GET /L turns the LED off
          }
        }
      }
      // close the connection:
      client.stop();
      Serial.println("Client Disconnected.");
    }
  }
```

On serial monitor:



```
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
E (26) psram: PSRAM ID read error: 0xffffffff


Connecting to Mahesh
.....
WiFi connected.
IP address:
192.168.28.238
```

On Web browser/Web Client:



Click here to turn the LED on pin 2 on.
Click here to turn the LED on pin 2 off.

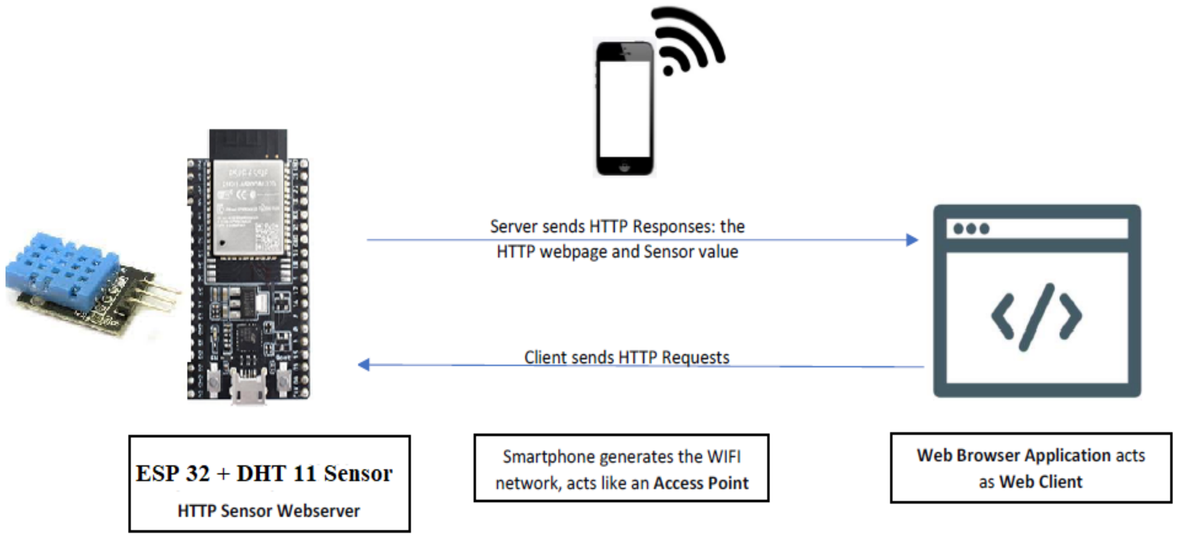
Work Sheet

Signature of Staff In-charge with date

EXPERIMENT NO. 11

<u>Aim:</u> Design and implementation of HTTP based IoT web server to display sensor value.

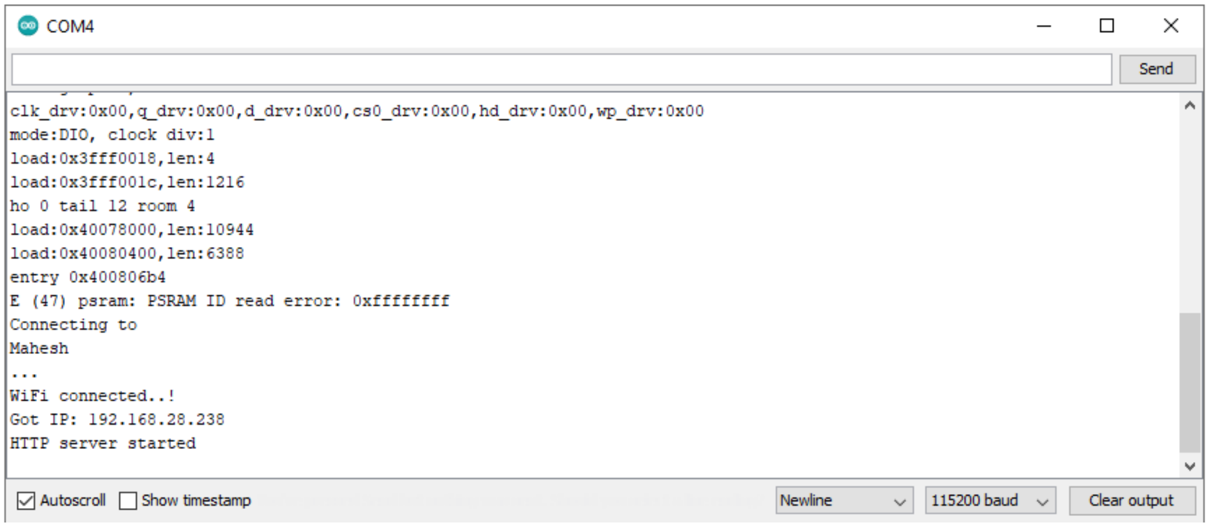
Description: The goal of this experiment is to integrate the HTTP webserver with Sensor. As we know, most IoT Nodes are equipped with sensors and the IoT systems should provide these sensor data to users. In this experiment, we used DHT11 Temperature and Humidity sensor connected to an IoT Node/ ESP32 development board. The IoT Node is configured as a Webserver and the sensor data can be accessed via a web browser.


Server sends HTTP Responses: the HTTP webpage and Sensor value
Client sends HTTP Requests
ESP 32 + DHT 11 Sensor
HTTP Sensor Webserver
Smartphone generates the WIFI network, acts like an Access Point
Web Browser Application acts as Web Client

Code:

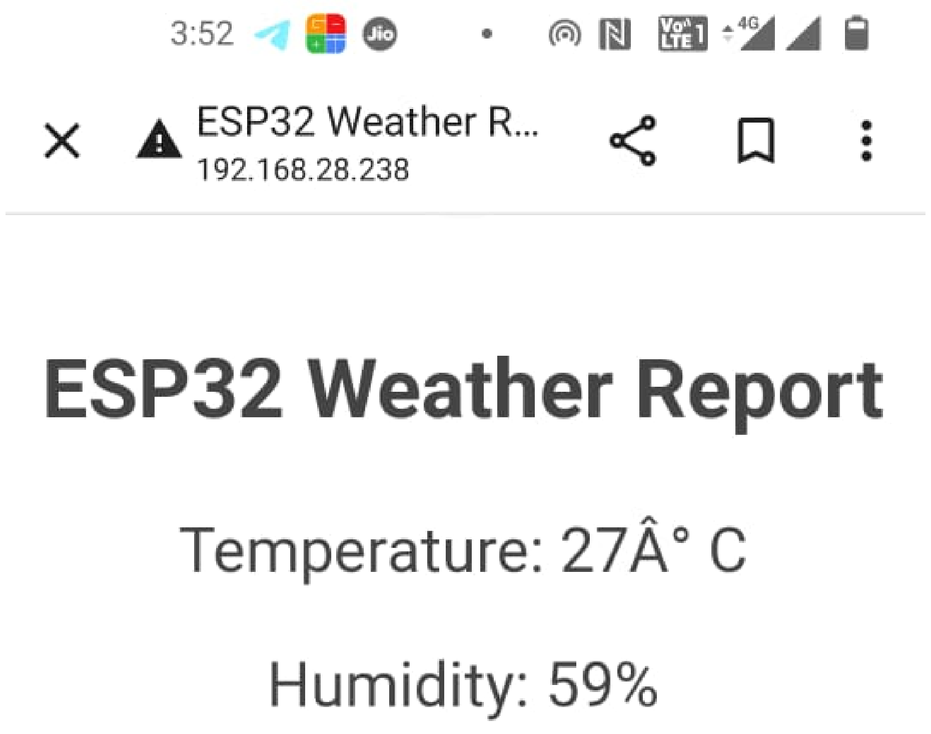
```
#include <WiFi.h>
#include <WebServer.h>
#include "DHT.h"

// Uncomment one of the lines below for whatever DHT sensor type you're using!
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT21   // DHT 21 (AM2301)
//#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321

/*Put your SSID & Password*/
const char* ssid = " REPLACE_WITH_YOUR_SSID ";      // Enter SSID here
const char* password = " REPLACE_WITH_YOUR_PASSWORD";      //Enter Password here

WebServer server(80);

// DHT Sensor
uint8_t DHTPin = 4;

// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);

float Temperature;
float Humidity;

void setup() {
  Serial.begin(115200);
  delay(100);

  pinMode(DHTPin, INPUT);

  dht.begin();

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

  //connect to your local wi-fi network
  WiFi.begin(ssid, password);

  //check wi-fi is connected to wi-fi network
  while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected..!");
  Serial.print("Got IP: ");  Serial.println(WiFi.localIP());

  server.on("/", handle_OnConnect);
  server.onNotFound(handle_NotFound);

  server.begin();
  Serial.println("HTTP server started");

}
void loop() {

  server.handleClient();

}

void handle_OnConnect() {

 Temperature = dht.readTemperature(); // Gets the values of the temperature
  Humidity = dht.readHumidity(); // Gets the values of the humidity
  server.send(200, "text/html", SendHTML(Temperature,Humidity));
}

void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}

String SendHTML(float Temperaturestat,float Humiditystat){
  String ptr = "<!DOCTYPE html> <html>\n";
  ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-
scalable=no\">\n";
  ptr +="<title>ESP32 Weather Report</title>\n";
  ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align:
center;}\n";
  ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";
  ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";
  ptr +="</style>\n";
  ptr +="</head>\n";
  ptr +="<body>\n";
  ptr +="<div id=\"webpage\">\n";
  ptr +="<h1>ESP32 Weather Report</h1>\n";
   ptr +="<p>Temperature: ";
  ptr +=(int)Temperaturestat;
  ptr +="\xC2\xB0 C</p>";
  ptr +="<p>Humidity: ";
  ptr +=(int)Humiditystat;
  ptr +="%</p>";
  ptr +="</div>\n";
  ptr +="</body>\n";
  ptr +="</html>\n";
  return ptr;
}
```

Result:



```
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
E (47) psram: PSRAM ID read error: 0xffffffff
Connecting to
Mahesh
...
WiFi connected..!
Got IP: 192.168.28.238
HTTP server started
```

On Web browser/Web Client:



ESP32 Weather Report

Temperature: 27Â° C

Humidity: 59%

Work Sheet

Signature of Staff In-charge with date

EXPERIMENT NO. 12

Aim: Design and implementation of MQTT based controlling and monitoring using Ubidots MQTT server.

Theory:
MQTT protocol

MQTT stands for Message Queuing Telemetry Transport. MQTT is a machine-to-machine internet of things connectivity protocol. It is an extremely lightweight and publish-subscribe messaging transport protocol. This protocol is useful for the connection with the remote location where the bandwidth is a premium. These characteristics make it useful in various situations, including constant environment such as for communication machine to machine and internet of things contexts. It is a publish and subscribe system where we can publish and receive the messages as a client. It makes it easy for communication between multiple devices. It is a simple messaging protocol designed for the constrained devices and with low bandwidth, so it's a perfect solution for the internet of things applications. Some of the features of an MQTT are given below:

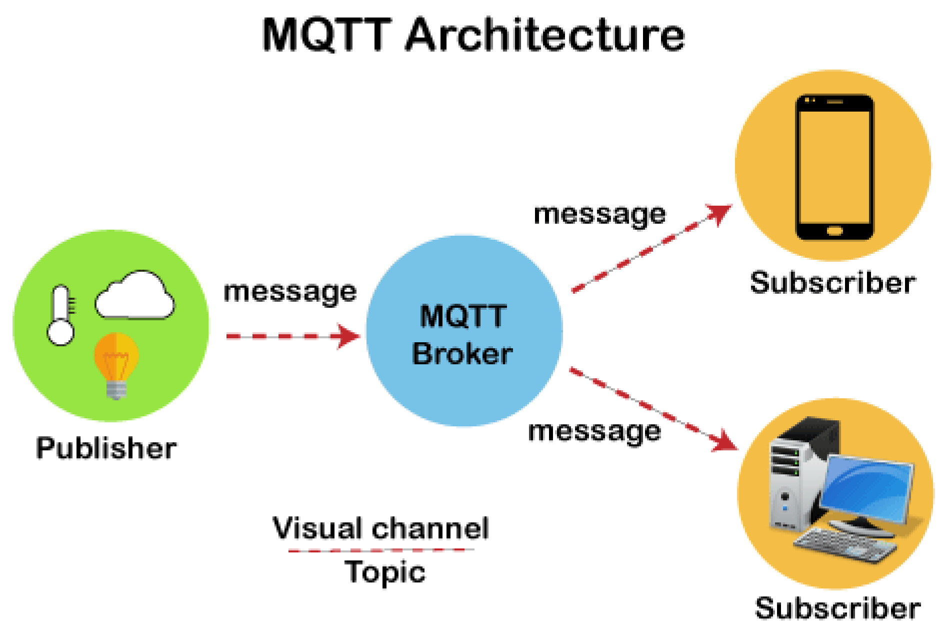
- It is a machine-to-machine protocol, i.e., it provides communication between the devices.
- It is designed as a simple and lightweight messaging protocol that uses a publish/subscribe system to exchange the information between the client and the server.
- It does not require that both the client and the server establish a connection at the same time.
- It provides faster data transmission, like how WhatsApp/messenger provides a faster delivery. It's a real-time messaging protocol.
- It allows the clients to subscribe to the narrow selection of topics so that they can receive the information they are looking for.

MQTT Architecture:

To understand the MQTT architecture, we first look at the components of the MQTT.

- Message
- Client
- Server or Broker
- TOPIC

- **Message:** Message: The message is the data that is carried out by the protocol across the network for the application. When the message is transmitted over the network, then the message contains the following parameters: Payload data, Quality of Service (QoS), Collection of Properties, Topic Name
- **Client:** In MQTT, the subscriber and publisher are the two roles of a client. The clients subscribe to the topics to publish and receive messages. In MQTT, the client performs two operations:
    1. **Publish:** When the client sends the data to the server, then we call this operation as a publish.
    2. **Subscribe:** When the client receives the data from the server, then we call this operation a subscription.

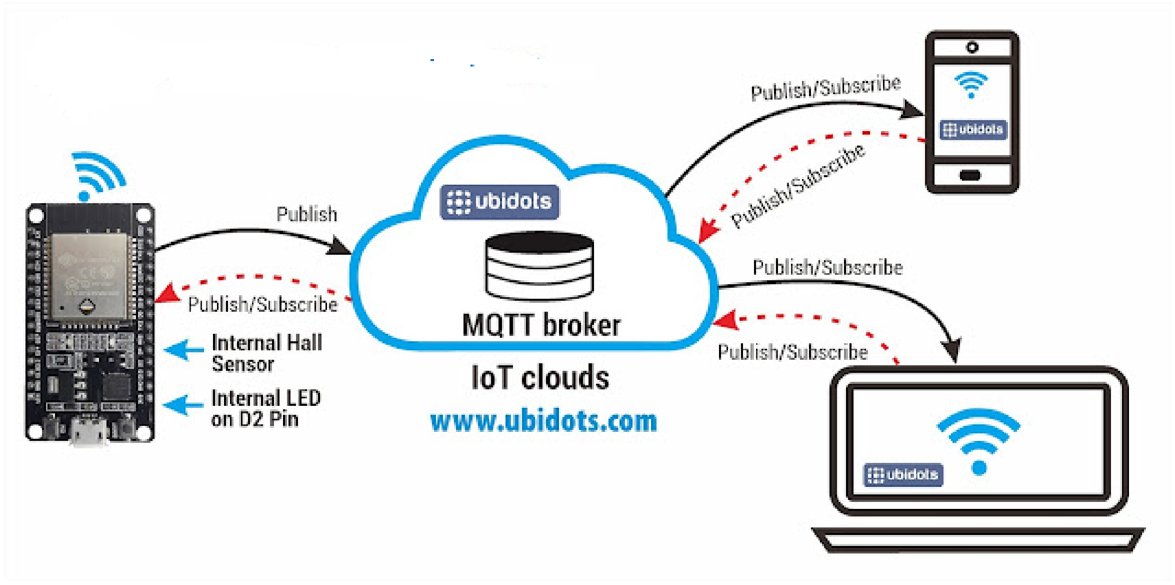


For more details: https://www.javatpoint.com/mqtt-protocol

Design and Implementation of MQTT based Controlling and Monitoring Using Ubidots MQTT Server.

Apparatus:

Sl No	Name of the Equipment	Quantity
1	ESP32 Development Board	1
2	LED	1
3	Jumper wires	2
4	Micro USB cable	1



Create Account on Ubidots Website:

1. **https://www.robotics-university.com/2019/05/internet-of-things-monitoring-sensor-and-controlling-led-on-ubidots-dashboard-using-mqtt-protocol.html**
2. **https://ubidots.com/**
3. **https://help.ubidots.com/en/articles/854333-ubidots-basics-devices-variables-dashboards-and-alerts**

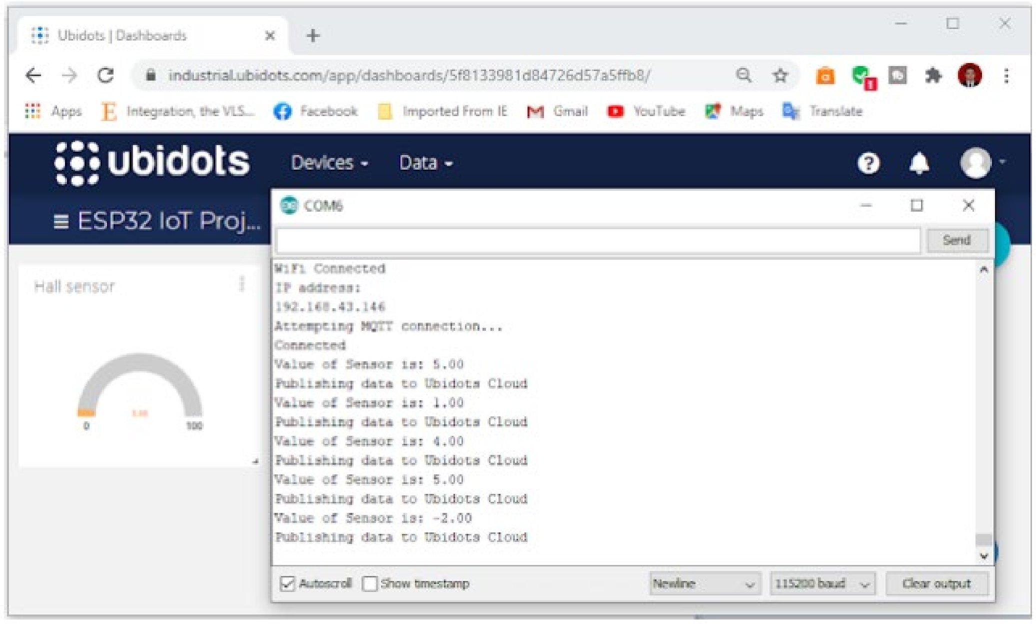
Code:

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

#define WIFISSID " Replace_With_Your_Ssid " // Put your WifiSSID here
#define PASSWORD " Replace_With_Your_Passward" // Put your wifi password here
#define TOKEN "YOUR_TOKEN" // Put your Ubidots' TOKEN
#define MQTT_CLIENT_NAME "ESP32" // MQTT client Name, please enter your own 8-12
alphanumeric character ASCII string;
                                    //it should be a random and unique ascii string and different from all
other devices

/****************************************
 * Define Constants
 ****************************************/
#define VARIABLE_LABEL "Variable Name 1" // Assing the variable label
#define VARIABLE_LABEL_SUBSCRIBE " Variable Name 2" // Assing the variable label
#define DEVICE_LABEL "Device Name" // Assig the device label

#define led 26 // Set the GPIO26 as LED
```

```
char mqttBroker[]  = "things.ubidots.com";
char payload[100];
char topic[150];
char topicSubscribe[100];
// Space to store values to send
char str_sensor[10];

/****************************************
 * Auxiliar Functions
 ****************************************/
WiFiClient ubidots;
PubSubClient client(ubidots);

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.println("Attempting MQTT connection...");

    // Attemp to connect
    if (client.connect(MQTT_CLIENT_NAME, TOKEN, "")) {
      Serial.println("Connected");
      client.subscribe(topicSubscribe);
    } else {
      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 2 seconds");
      // Wait 2 seconds before retrying
      delay(2000);
    }
  }
}
void callback(char* topic, byte* payload, unsigned int length) {
  char p[length + 1];
  memcpy(p, payload, length);
  p[length] = NULL;
  String message(p);
  if (message == "0.0") {
    digitalWrite(led, LOW);
  } else {
    digitalWrite(led, HIGH);
  }

  Serial.write(payload, length);
  Serial.println();
}
```

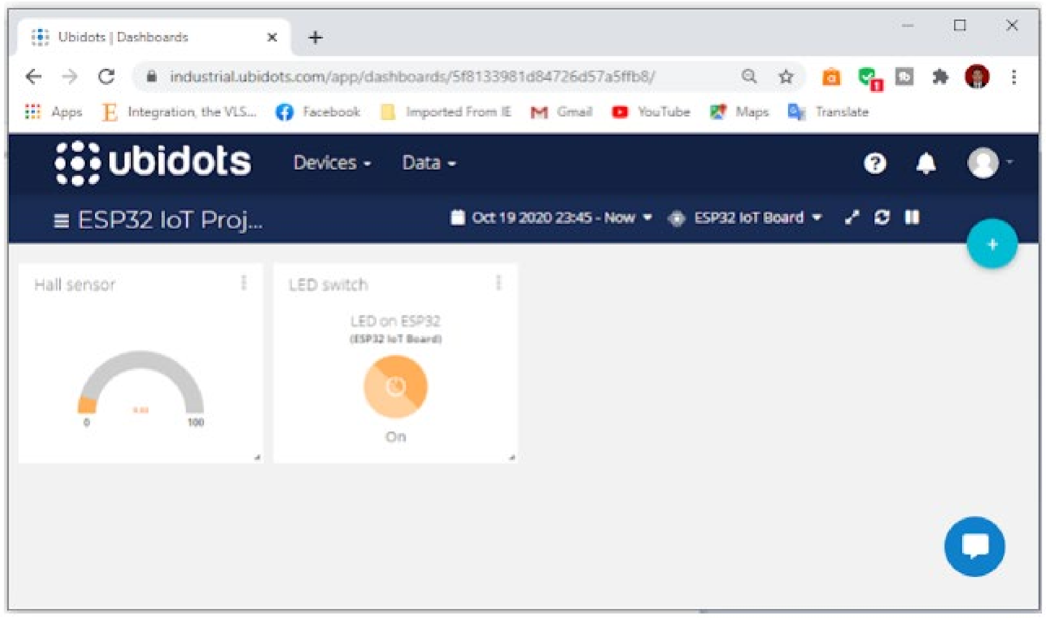
```
/****************************************
 * Main Functions
 ****************************************/
void setup() {
  Serial.begin(115200);
  WiFi.begin(WIFISSID, PASSWORD);
  // Assign the pin as INPUT
  pinMode(led, OUTPUT);

  Serial.println();
  Serial.print("Wait for WiFi...");

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

  Serial.println("");
  Serial.println("WiFi Connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  client.setServer(mqttBroker, 1883);
  client.setCallback(callback);
  sprintf(topicSubscribe,"/v1.6/devices/%s/%s/lv",
DEVICE_LABEL,VARIABLE_LABEL_SUBSCRIBE);

  client.subscribe(topicSubscribe);
}

void loop() {
  if (!client.connected()) {
    client.subscribe(topicSubscribe);
    reconnect();
  }

  sprintf(topic, "%s%s", "/v1.6/devices/", DEVICE_LABEL);
  sprintf(payload, "%s", ""); // Cleans the payload
  sprintf(payload, "{\"%s\":", VARIABLE_LABEL); // Adds the variable label

  float sensor = hallRead();
  Serial.print("Value of Sensor is:- ");Serial.println(sensor);

  /* 4 is mininum width, 2 is precision; float value is copied onto str_sensor*/
  dtostrf(sensor, 4, 2, str_sensor);
  sprintf(payload, "%s {\"value\": %s}}", payload, str_sensor); // Adds the value
  Serial.println("Publishing data to Ubidots Cloud");
  client.publish(topic, payload);
  client.loop();
  delay(1000);
}
```

Result:



```
WiFi Connected
IP address:
192.168.43.146
Attempting MQTT connection...
Connected
Value of Sensor is: 5.00
Publishing data to Ubidots Cloud
Value of Sensor is: 1.00
Publishing data to Ubidots Cloud
Value of Sensor is: 4.00
Publishing data to Ubidots Cloud
Value of Sensor is: 5.00
Publishing data to Ubidots Cloud
Value of Sensor is: -2.00
Publishing data to Ubidots Cloud
```

Realtime monitoring Hall sensor & control LED on Ubidots dashboard:



<u>Work Sheet</u>

Signature of Staff In-charge with date

References:

1. "Developing IoT Projects with ESP32 - Second Edition: Unlock the Full Potential of ESP32 in IoT Development to Create Production-Grade Smart Devices" Vedat Ozan Oner, Packt Publishing, 2023, ISBN: 978-1803237688.
2. "Hands-on ESP32 with Arduino IDE: Unleash the Power of IoT with ESP32 and Build Exciting Projects with This Practical Guide", Asim Zulfiqar, Packt Publishing, 2024, ISBN: 978-1837638031.
3. "ESP32 Formats and Communication Protocols", Neil Cameron, Apress, 2023, ISBN: 978-1484297280.
4. "Embedded Systems mit RISC-V und ESP32-C3", Patrick Ritschel, dpunkt.verlag, 2023, ISBN: 978-3864909270.
5. "ESP32-C3 Wireless Adventure: A Comprehensive Guide to IoT", Espressif Systems, Espressif Systems, 2023.
6. "Simplified Embedded Rust: ESP Standard Library Edition", Omar Hiari, Independently Published, 2024, ISBN: 978-1098136200.

SAMPLE VIVA QUESTIONS

1. What is the function of an LED in the experiment?
2. What does GPIO stand for, and what is its purpose?
3. Why is a resistor connected in series with the LED?
4. What is the significance of the pinMode() function in the program?
5. What is the difference between HIGH and LOW in the context of GPIO pins?
6. How does the ESP32 provide power to the LED?
7. What is the purpose of the delay() function in the code?
8. How can you change the blinking interval of the LED?
9. What would happen if you did not connect a resistor to the LED?
10. What are some real-world applications of controlling an LED with a microcontroller like ESP32?
11. What sensor is commonly used to measure temperature and humidity in this experiment?
12. What is the function of the ESP32 in this experiment?
13. What is the role of the heat index in this experiment?
14. Which library is required to interface the DHT sensor with the ESP32?
15. How is the sensor connected to the ESP32?
16. What is the purpose of the Serial.begin() function in the program?
17. Why do we use a pull-up resistor with the DHT sensor?
18. How does the ESP32 calculate the heat index?
19. What unit is used for temperature and humidity in this experiment?
20. What are some real-world applications of this experiment?
21. What is the principle of operation of an ultrasonic sensor?
22. What are the main pins of the ultrasonic sensor used in this experiment?
23. What is the purpose of the TRIG pin in the ultrasonic sensor?
24. How does the ECHO pin work in the ultrasonic sensor?
25. What formula is used to calculate the distance to the object?

26. What is the speed of sound used in this calculation?
27. Why do we divide the calculated time by 2 in the formula?
28. What is the role of the digitalWrite() and digitalRead() functions in the program?
29. What unit is typically used to display the distance measured by the ultrasonic sensor?
30. What are some real-world applications of ultrasonic distance measurement?
31. What is the principle of a soil moisture sensor?
32. What are the main components of a soil moisture sensor module?
33. What pins are used to connect the soil moisture sensor to the ESP32?
34. What type of signal does the soil moisture sensor output?
35. Why do we use the analogRead() function in this experiment?
36. What is the range of analog values that the ESP32 can read?
37. How does the soil moisture sensor differentiate between wet and dry soil?
38. What is the purpose of the Serial.begin() function in the program?
39. What are the practical applications of measuring soil moisture?
40. What are the limitations of soil moisture sensors?
41. What is the working principle of an IR sensor?
42. What are the main components of an IR sensor module?
43. How does the IR sensor detect motion?
44. What pins are used to connect the IR sensor to an ESP32 or microcontroller?
45. What is the role of the buzzer in this experiment?
46. Which function is used to read the sensor output in the program?
47. What does the HIGH or LOW state from the IR sensor indicate?
48. What function is used to turn the buzzer on or off in the program?
49. What are some real-world applications of IR motion detection?
50. What are the limitations of IR sensors for motion detection?
51. What is the working principle of a pulse sensor?
52. What does BPM stand for, and how is it calculated?
53. What are the key pins of the pulse sensor, and how are they connected to the ESP32?
54. What function is used to read the output of the pulse sensor in the ESP32 program?
55. Why is filtering or signal smoothing often applied to the pulse sensor data?
56. How is the heart rate derived from the pulse sensor data?
57. What is the role of the Serial.begin() function in this experiment?
58. What are the normal BPM ranges for adults?
59. What are some common applications of pulse sensors?
60. What are some challenges in accurately measuring BPM using a pulse sensor?
61. What does it mean to configure an ESP32 as an Access Point (AP)?
62. What is the default IP address assigned to the ESP32 in AP mode?
63. Which library is used to enable Wi-Fi functionality in the ESP32?
64. What function is used to set the ESP32 as an access point?
65. What parameters are required to set up the ESP32 as an AP?
66. How can you make the ESP32 access point open (no password)?
67. What is the maximum number of clients that can connect to the ESP32 in AP mode?
68. How do you check if a client is connected to the ESP32 access point?
69. What are the practical applications of using an ESP32 as an access point?
70. What are the advantages of using an ESP32 as an AP over connecting it to an existing Wi-Fi network?

71. What is the purpose of scanning Wi-Fi networks with an ESP32?
72. Which function is used to initiate a Wi-Fi network scan in ESP32?
73. What does RSSI stand for, and what does it indicate?
74. What is the typical range of RSSI values for Wi-Fi signals?
75. How can you display the list of available networks and their strengths on the serial monitor?
76. What does the term SSID stand for, and what is its significance?
77. How can you determine whether a Wi-Fi network is secured or open?
78. What is the purpose of the WiFi.mode() function in this experiment?
79. What are some real-world applications of Wi-Fi network scanning?
80. What factors can affect the RSSI value of a Wi-Fi network?