***INTERNET OF THINGS LAB RECORD***

***Subject code : BTCS-AMDS-009T***

|  |  |
| --- | --- |
| ***Name:*** | ***Sreerama monisha*** |
| ***Registration Number:*** | ***FET-BAML-2022-2026-004*** |
| ***Course:*** | ***Btech CSE AIML*** |
| ***Semester:*** | ***5th*** |
| ***Faculty:*** | ***Mr. Biswajeeban Mishra &  Mr. Pritam Nanda*** |

|  |  |
| --- | --- |
| ***Remarks*** |  |
| ***Signature*** |  |

******

***SRI SRI UNIVERSITY***

***Bidyadharpur, Cuttack, Odisha.***

***Index***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Sl. No.*** | ***Date*** | ***Experiment/Case Study*** | ***Page No.*** | ***Remark*** |
| ***1*** | ***9-8-2024*** | ***Blinking of Internal LED on Arduino board*** |  |  |
| ***2*** | ***9-8-2024*** | ***Blinking an External LED*** |  |  |
| ***3*** | ***9-8-2024*** | ***Read Humidity and Temperature value using DHT22*** |  |  |
| ***4*** | ***9-8-2024*** | ***A case study on  a communication Protocol: MQTT*** |  |  |
| ***5*** | ***9-8-2024*** | ***Building a web app using Node-Red to fetch DHT sensor data and display it on the web app dashboard*** |  |  |
| ***6*** | ***9-8-2024*** | ***Working with button, LED Interfacing Ultrasonic Sensor*** |  |  |
| ***7*** | ***9-8-2024*** | ***Use of Breadboard*** |  |  |
| ***8*** |  |  |  |  |
| ***9*** |  |  |  |  |
| ***10*** |  |  |  |  |

***Experiment No.:1 Date: 9-8-24***

**Aim of the Experiment**  
To program an Arduino to make the internal LED blink on and off with a delay of 5 seconds between each state, demonstrating basic programming concepts and control over digital outputs.

**Requirements**

1. Arduino board (with built-in LED on pin 13)
2. USB cable for connecting Arduino to a computer
3. Arduino IDE for writing and uploading code

Description

The internal LED on an Arduino, typically connected to pin 13, serves as a convenient tool for experimenting with fundamental programming concepts. This exercise will create a program that makes the internal LED blink on and off with a 5 seconds delay between each state.

// Define the pin number for the internal LED on most Arduino boards

int ledPin = 13;

void setup() {

// Set the internal LED pin to function as an output

pinMode(ledPin, OUTPUT);

}

void loop() {

// Turn the LED on by setting the voltage to HIGH

digitalWrite(ledPin, HIGH);

// Pause for 5 seconds

delay(1000);

// Turn the LED off by setting the voltage to LOW

digitalWrite(ledPin, LOW);

// Pause for 5 seconds

delay(5000);

}

Process Explanation

1. **Variable Declaration:** We declare an integer variable ledPin and set it to 13, which corresponds to the internal LED pin on most Arduino boards.
2. **Setup Function:**
   * The setup() function initializes settings for the Arduino. Here, we use pinMode(ledPin, OUTPUT); to configure the LED pin as an output, allowing it to control the connected LED.
3. **Loop Function:**
   * The loop() function contains the main code to be repeated continuously. It performs the following actions:
     + **Turn On LED:** Use digitalWrite(ledPin, HIGH); to supply voltage to the LED pin, turning the LED on.
     + **Pause:** delay(5000); pauses the program for 5 second, keeping the LED on.
     + **Turn Off LED:** Use digitalWrite(ledPin, LOW); to cut off the voltage to the LED pin, turning the LED off.
     + **Pause Again:** Another delay(5000); keeps the LED off for 5 second.

Conclusion

This simple experiment illustrates the use of the digitalWrite() and delay() functions in Arduino programming to control the internal LED. It's for learning how to manipulate outputs and understanding the basic structure of an Arduino sketch.



***Experiment No.:2***

Aim of the Experiment  
To control an external LED connected to an Arduino by making it blink on and off at regular intervals, demonstrating how to control external devices with Arduino.

Requirements

1. Arduino board
2. External LED
3. Resistor (220Ω or 330Ω recommended for current limiting)
4. Jumper wires
5. USB cable for connecting Arduino to a computer
6. Arduino IDE for writing and uploading code

In this experiment, we'll connect an external LED to the Arduino and write a program to make the LED blink on and off at regular intervals. This process illustrates how we can control external devices using the Arduino.

Circuit Setup:

1. **Connect the Anode**: The longer leg of the LED, called the anode, should be connected to pin 8 on the Arduino.
2. **Connect the Cathode**: The shorter leg of the LED, known as the cathode, should be connected to one of the ground (GND) pins on the Arduino. This completes the circuit by allowing current to flow through the LED.

// Define which pin will control the LED

int ledPin = 8;

void setup() {

// Set the LED pin as an output pin

pinMode(ledPin, OUTPUT);

}

void loop() {

// Turn the LED on by sending a HIGH signal

digitalWrite(ledPin, HIGH);

// Wait for 1 second to keep it on

delay(1000);

// Turn the LED off by sending a LOW signal

digitalWrite(ledPin, LOW);

// Wait for 1 second to keep it off

delay(1000);

}

* **Setup Function**: This part of the code runs once when you start the Arduino. It sets pin 8 as an output, which allows it to send signals to turn the LED on or off.
* **Loop Function**: This part repeats continuously. It turns the LED on for one second (using HIGH), pauses for a second (delay(1000)), turns the LED off (using LOW), and pauses again. This cycle creates the blinking effect.

Conclusion:

This experiment illustrates how easy it is to control external components with an Arduino. By making the LED blink, you get practical experience with basic circuit building and programming. This lays the groundwork for more complex projects involving various electronic components. 

***Experiment No.:3***

Aim of the Experiment  
To measure and display temperature and humidity values using a DHT22 sensor connected to an Arduino, with results shown in the Arduino IDE's serial monitor.

Requirements

1. Arduino board
2. DHT22 sensor
3. Jumper wires
4. Breadboard (optional)
5. USB cable for connecting Arduino to a computer
6. Arduino IDE for writing and uploading code

**Description:**  
In this experiment, we use a DHT22 sensor to measure temperature and humidity, with the results shown in the Arduino IDE's serial monitor.

**Circuit Setup:**

1. Connect the sensor's **VCC** pin to the **5V** pin on the Arduino.
2. Connect the **GND** pin to the Arduino's ground.
3. Connect the **data** pin to **pin 2** on the Arduino.

**Code:**

#include "DHT.h"

// Pin to which the DHT22 sensor is connected

#define DHTPIN 2

// Define the type of DHT sensor

#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

void setup() {

// Start the serial monitor at 9600 baud rate

Serial.begin(9600);

// Initialize the DHT sensor

dht.begin();

}

void loop() {

// Wait a few seconds between measurements

delay(2000);

// Read humidity

float humidity = dht.readHumidity();

// Read temperature in Celsius

float temperature = dht.readTemperature();

// Check if any reads failed and exit early (to try again)

if (isnan(humidity) || isnan(temperature)) {

Serial.println("Failed to read from DHT sensor!");

return;

}

// Print the results to the Serial Monitor

Serial.print("Humidity: ");

Serial.print(humidity);

Serial.print(" %\t");

Serial.print("Temperature: ");

Serial.print(temperature);

Serial.println(" °C");

**Conclusion:**

In this experiment, we successfully interfaced the DHT22 sensor with the Arduino to measure humidity and temperature.The Arduino board reads these values from the sensor and displays them on the serial monitor. This is useful for projects that need to monitor the environment, like weather stations or home automation systems stations or home automation 

***Experiment No.:4***

**Aim of the Experiment**  
To set up an MQTT (Message Queuing Telemetry Transport) broker on a local machine to enable communication between IoT devices and applications for data transfer and monitoring.

**Requirements**

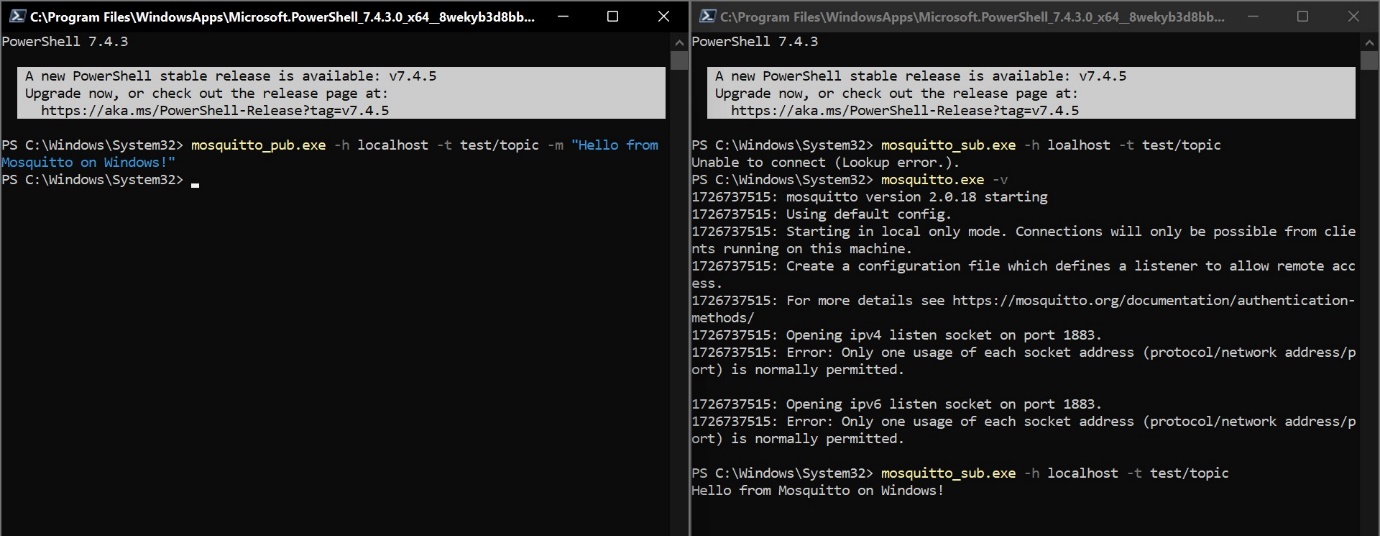
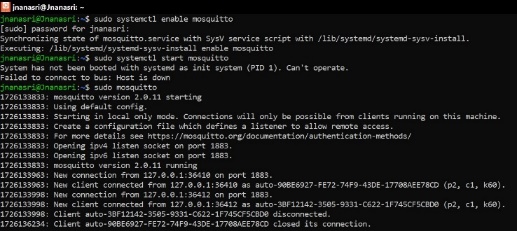
1. Computer with internet access
2. MQTT broker software (e.g., Mosquitto)
3. Terminal or command prompt access
4. MQTT client software (e.g., MQTT.fx or MQTT Explorer)

**Description**  
MQTT is a lightweight protocol ideal for IoT applications. In this experiment, we’ll install and configure an MQTT broker on a local machine, allowing clients to publish messages and subscribe to topics for efficient, real-time data exchange.

**Procedure**

1. **Install MQTT Broker**:
   * Download and install an MQTT broker like Mosquitto on your local machine.
   * Verify the installation by running mosquitto -v in the terminal to see if the broker starts.
2. **Configure the Broker**:
   * Edit the Mosquitto configuration file (usually mosquitto.conf) to set up parameters like port (default is 1883) and allow anonymous connections (optional).
3. **Start the Broker**:
   * Run the broker in the terminal using mosquitto or configure it as a service to start automatically.
4. **Install an MQTT Client**:
   * Download an MQTT client tool (e.g., MQTT.fx or MQTT Explorer) to test message publishing and subscribing.
5. **Publish and Subscribe**:
   * Open the MQTT client, connect to localhost at port 1883, and subscribe to a test topic (e.g., test/topic).
   * Publish a message to the same topic and observe it in the client to confirm successful communication.

**Observation**  
After setting up and running the MQTT broker, clients can successfully connect, publish, and subscribe to topics, allowing real-time message exchanges on the local machine.

**Conclusion**  
This experiment demonstrates how to set up and configure an MQTT broker locally, enabling basic message exchanges between connected devices. The setup lays a foundation for IoT applications, allowing efficient and reliable data transmission across devices and applications. 

***Experiment No.:5***

**Aim of the Experiment**

To visualize real-time temperature and humidity data from a DHT11 sensor on a Node-RED dashboard.

**Requirements**

1. Node.js
2. NodeRED

**Procedure**

**Install Node.js :**

- Installed NodeJS from Official Eclipse Page [ https://nodejs.org/en/download/package-manager ].

- Added node.js to the System Environment Variables PATH [ ' C:\Users\Lokesh Patra\AppData\Roaming\npm ' ], which allows us to use npm commands directly in the Command Prompt or, Terminal.

**Installing & Initialising NodeRED**

- Open Node.js > npm install node-red-dashboard

- [PostInstallation] > Elevated CMD: node-red

> In Client Application, browsed localhost:1880 [ Accessing NodeRED ]

- Inside the NodeRED window, a flow was created w/ the nodes as:

> SERIAL-IN ( Arduino Uno R3 Board )

> DEBUGGER

> DHT FUNCTION

> 2 GAUGES ( Humidity & Temperature )

- Serial In Node: Configured it to read from the correct serial port where my Arduino is connected (e.g., COM11) > Set the baud rate to 9600.

- Configure the DHT Function as:

var m = msg.payload.split(',');

if (m.length === 2) {

var H = { payload: parseFloat(m[0]) };

var T = { payload: parseFloat(m[1]) };

return [H, T];

} else {

return null;

}

- Adjusting Gauge Nodes:

> Humidity:

- Title as "Humidity".

- Value format as `{{value}}%`.

- Minimum value to 0 and the maximum to 100.

> Temperatue:

- Title as ' Temperature '

- Value format as {{value}}°C.

- Ensure that Humidity & Temperature are in the same group.

**Deployment:**

- Uploaded DHT11 /22 Sketch to the Arduino Board through its IDE:

#include <DHT.h>

#define DHTPIN 3

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

void setup() {

Serial.begin(9600);

dht.begin();

}

void loop() {

float H = dht.readHumidity();

float T = dht.readTemperature();

if (isnan(H) || isnan(T)) {

Serial.println("Failed to read from DHT sensor!");

} else {

Serial.println(String(H) + "," + String(T));

}

delay(2000);

}

- After uploading this sketch, close the IDE.

- Deploy the flow in NodeRED.

- Check the Dashboard in the upper-right corner, for the Humidity and Temperature Gauge.

**Observation**

The Node-RED dashboard displays real-time temperature and humidity readings from the DHT11 sensor. As the sensor measures changes in the environment, the dashboard updates accordingly.

**Conclusion**

This experiment successfully demonstrates how to integrate a DHT11 sensor with an Arduino, use MQTT for data transmission, and visualize that data on a Node-RED dashboard. It highlights the effectiveness of combining IoT technologies for real-time monitoring applications.

***Experiment No.:6***

**Aim of the Experiment**

To demonstrate the interaction between a push button, an LED, and an HC-SR04 ultrasonic sensor using Arduino.

**Requirements**

* Arduino UNO R3
* HC-SR04 ultrasonic distance sensor
* Push button
* LED
* Resistor (220Ω for the LED)
* Jumper wires
* Breadboard (optional)

**Procedure:**

**1.Button: Held or, Released?**

Step#1: Connect 2 jumper wires *diagonally* to the Button.

Step#2: One wire to Digital Pin [ *here, 2* ], and the other wire to GND.

Step#3: After configuring the Arduino UNO R3, this sketch is to be uploaded and executed in the IDE:

Step#4: Create a new sketch and type the following code:

**Code:**

const int buttonPin = 2; // Pin where the button is connected

void setup() {

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

pinMode(buttonPin, INPUT\_PULLUP); // Set the button pin as input with internal pull-up resistor

}

void loop() {

int buttonState = digitalRead(buttonPin); // Read the state of the button

if (buttonState == LOW) { // Check if the button is pressed

Serial.println("Button Held!"); // Print message to Serial Monitor

}

else {

Serial.println("Button Released!"); // Print message to Serial Monitor

}

delay(500); // Add a small delay to debounce the button

}

Output: Button Held! ; Button Released

**2.UltraSonic [ HC-SR04 ]: Measures Distance**

Step#1: Connect 4 jumper wires to the UltraSonic Sensor as [VCC: 5v], [Trigger: 9], [Echo: 8], & **GND**.

Step#2: After configuring the **HC-SR04** w/ **UNO R3**, this sketch is to be uploaded and executed in the IDE:

#define PIN\_TRIG 9 // Define the pin for the trigger

#define PIN\_ECHO 8 // Define the pin for the echo

void setup() {

Serial.begin(9600); // Initialize serial communication at 9600 baud

pinMode(PIN\_TRIG, OUTPUT); // Set the trigger pin as output

pinMode(PIN\_ECHO, INPUT); // Set the echo pin as input

}

void loop() {

// Start a new measurement:

digitalWrite(PIN\_TRIG, HIGH); // Set the trigger pin high

delayMicroseconds(10); // Wait for 10 microseconds

digitalWrite(PIN\_TRIG, LOW); // Set the trigger pin low

// Read the result:

int duration = pulseIn(PIN\_ECHO, HIGH); // Read the duration of the pulse from the echo pin

// Calculate distance in centimeters:

Serial.print("Distance in CM: ");

Serial.println(duration / 58); // Print the distance in centimeters

// Calculate distance in inches:

Serial.print("Distance in inches: ");

Serial.println(duration / 148); // Print the distance in inches

delay(1000); // Wait for 1 second before taking the next measurement

}

**Output:**Distance in CM: 121 ; Distance in inches: 47

**3.HC-SR04 + PushButton**

Step#3: Now, a button can be connected to Digital 2 and diagonally to **GND**.

Step#4: After configuring the Button & the sensor, this sketch is to be uploaded and executed in the IDE:

#define PIN\_TRIG 9 // Define the pin for the trigger

#define PIN\_ECHO 8 // Define the pin for the echo

#define buttonPin 2 // Define the pin for the button

void setup() {

Serial.begin(9600); // Initialize serial communication at 9600 baud

pinMode(PIN\_TRIG, OUTPUT); // Set the trigger pin as output

pinMode(PIN\_ECHO, INPUT); // Set the echo pin as input

pinMode(buttonPin, INPUT\_PULLUP); // Set the button pin as input with internal pull-up resistor

}

void loop() {

int buttonState = digitalRead(buttonPin); // Read the state of the button

if (buttonState == LOW) { // Check if the button is pressed

// Start a new measurement:

digitalWrite(PIN\_TRIG, HIGH); // Set the trigger pin high

delayMicroseconds(10); // Wait for 10 microseconds

digitalWrite(PIN\_TRIG, LOW); // Set the trigger pin low

// Read the result:

int duration = pulseIn(PIN\_ECHO, HIGH); // Read the duration of the pulse from the echo pin

// Calculate distance in centimeters:

Serial.print("Distance in CM: ");

Serial.println(duration / 58); // Print the distance in centimeters

// Calculate distance in inches:

Serial.print("Distance in Inches: ");

Serial.println(duration / 148); // Print the distance in inches

} else {

Serial.println("Button Released!"); // Print message to Serial Monitor

}

delay(500); // Add a small delay to debounce the button

}

**Output:**Button Released! ... ; Distance in CM: 24 ; Distance in Inches: 9

**4.Combining PushButton + UltraSonic Sensor + External LED + DHT Sensor**

Step#1: Get 9 Jumper wires [ M2F ], and connect them as:

* HC-SR04:
  + Trigger [ *Digital 9* ]
  + Echo [ *Digital 8* ]
  + VCC [ *5v* ]
  + GND
* DHT 11:
  + VCC [ *3.3v* ]
  + Data [ *Digital 4* ]
  + GND
* Button:
  + Corner 1 [ *Digital 2* ]
  + Diagonal Corner 4 [ GND ]
* External LED:
  + Anode [ +ve, *Digital 13* ]
  + Cathode [ -ve, GND]

Step#2: After configuring the entire Arduino Uno R3 with proper wire connections, the following could be executed in the IDE:

#include <DHT.h>

#define DHTPIN 4 // Pin where the DHT sensor is connected

#define DHTTYPE DHT11 // Define the type of DHT sensor (DHT11 in this case)

#define PIN\_TRIG 9 // Define the pin for the ultrasonic sensor trigger

#define PIN\_ECHO 8 // Define the pin for the ultrasonic sensor echo

#define BUTTON\_PIN 2 // Define the pin for the button

#define LED\_PIN 13 // Define the pin for the LED

DHT dht(DHTPIN, DHTTYPE);

void setup() {

Serial.begin(9600); // Initialize serial communication at 9600 baud

dht.begin(); // Initialize the DHT sensor

pinMode(PIN\_TRIG, OUTPUT); // Set the ultrasonic sensor trigger pin as output

pinMode(PIN\_ECHO, INPUT); // Set the ultrasonic sensor echo pin as input

pinMode(BUTTON\_PIN, INPUT\_PULLUP); // Set the button pin as input with internal pull-up resistor

pinMode(LED\_PIN, OUTPUT); // Set the LED pin as output

}

void loop() {

int buttonState = digitalRead(BUTTON\_PIN); // Read the state of the button

if (buttonState == LOW) { // Check if the button is pressed

// Ultrasonic Sensor Measurement:

digitalWrite(PIN\_TRIG, HIGH); // Set the trigger pin high

delayMicroseconds(10); // Wait for 10 microseconds

digitalWrite(PIN\_TRIG, LOW); // Set the trigger pin low

int duration = pulseIn(PIN\_ECHO, HIGH); // Read the duration of the pulse from the echo pin

float distanceCm = duration / 58.0; // Calculate the distance in centimeters

Serial.print("Distance in CM: ");

Serial.println(distanceCm);

if (distanceCm < 20) { // Example condition: if distance is less than 20 cm

digitalWrite(LED\_PIN, HIGH); // Turn the LED on

} else {

digitalWrite(LED\_PIN, LOW); // Turn the LED off

}

// DHT Sensor Measurement:

float humidity = dht.readHumidity();

float temperature = dht.readTemperature();

if (isnan(humidity) || isnan(temperature)) {

Serial.println("Failed to read from DHT sensor!");

} else {

Serial.print("Humidity: ");

Serial.print(humidity);

Serial.print(" %\t");

Serial.print("Temperature: ");

Serial.print(temperature);

Serial.println(" \*C");

}

} else {

Serial.println("Button Released!");

}

delay(1500); // Delay for 1.5 seconds before the next loop

}

**Output:**Button Released!; Distance in CM: 18.45 -- Humidity: 79.00 % -- Temperature: 30.80 \*C [ LED TURNED ON ] ; Distance in CM: 24.28 -- Humidity: 65.00 % -- Temperature: 30.80 \*C [ LED TURNED OFF ] ; Button Released!

**Conclusion:** Through this experiment we understood the concept of working with button, LED and Ultrasonic sensor.

***Experiment 7:***

**Aim of the Experiment**

To set up the ESP32 DevKit v1 board in Arduino IDE, configure it to scan for nearby Wi-Fi networks, and blink the onboard LED.

**Requirements**

* Arduino IDE
* USB cable for ESP32 connection
* ESP32 DevKit v1 (NodeMCU)

**Procedure:**

**Setting Up ESPressif32**

Step#1: Inside Arduino IDE, Navigate to Files > Preferences > Additional Boards Manager URL > <https://dl.espressif.com/dl/package_esp32_index.json> < *paste this and click OK*

Step#2: Then head into BOARDS MANAGER > and Install esp32 by ESPressif Systems.

Step#3: Now, for establishing the connection, we would need to configure our mainframe w/ the CP210x USB-to-UART Bridge Virtual COM Port (VCP) Driver avaliable at [CP210x USBtoUART Driver](https://www.silabs.com/documents/public/software/CP210x_Windows_Drivers.zip).

Step#4: After a superfluous reboot, the mainframe is ready to be used w/ an ESPressif32, as in the Arduino IDE, we first select the correct COM port (*here, COM12*), and ESP32 Dev Module as the board.

* NOTE: While executing a sketch, the Board reuires to be in DOWNLOAD MODE /BOOT MODE, so for, the BOOT button is to be pressed while uploading the code onto the board, exactly post Connecting... for 3-4 seconds.
  + A simple way around for this redundancy, would be:
    - Holding the BOOT button (GPIO0), the EN (Enable Pin) button is pressed for a second.
    - Post releasing the EN button, finally, the BOOT button is let go.
  + *This would keep the Espressif32 in the bootloader mode, so it could be equipped w/ any further sketch executions.*

Step#5: To blink the internal LED, the following is to be executed in the IDE:

**Code:**

// Define the LED pin

#define LED\_PIN 2 // The onboard LED is connected to GPIO 2 on most ESP32 boards

void setup() {

// Initialize the LED pin as an output

pinMode(LED\_PIN, OUTPUT);

}

void loop() {

// Turn the LED on (HIGH is the voltage level)

digitalWrite(LED\_PIN, HIGH);

// Wait for a second

delay(1000);

// Turn the LED off by making the voltage LOW

digitalWrite(LED\_PIN, LOW);

// Wait for a second

delay(1000);

}

**Output**: Internal LED (blue) blinks @ 1s duration

**Configuring the ESPressif to SCAN Wireless-Fidelity Signals!**

Step#6: The ESP32 can be used to scan nearby Wi-Fi signals using the following sketch:

#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");

Serial.println("Nr | SSID | RSSI | CH | Encryption");

for (int i = 0; i < n; ++i) {

// Print SSID and RSSI for each network found

Serial.printf("%2d", i + 1);

Serial.print(" | ");

Serial.printf("%-32.32s", WiFi.SSID(i).c\_str());

Serial.print(" | ");

Serial.printf("%4ld", WiFi.RSSI(i));

Serial.print(" | ");

Serial.printf("%2ld", WiFi.channel(i));

Serial.print(" | ");

switch (WiFi.encryptionType(i)) {

case WIFI\_AUTH\_OPEN: Serial.print("open"); break;

case WIFI\_AUTH\_WEP: Serial.print("WEP"); break;

case WIFI\_AUTH\_WPA\_PSK: Serial.print("WPA"); break;

case WIFI\_AUTH\_WPA2\_PSK: Serial.print("WPA2"); break;

case WIFI\_AUTH\_WPA\_WPA2\_PSK: Serial.print("WPA+WPA2"); break;

case WIFI\_AUTH\_WPA2\_ENTERPRISE: Serial.print("WPA2-EAP"); break;

case WIFI\_AUTH\_WPA3\_PSK: Serial.print("WPA3"); break;

case WIFI\_AUTH\_WPA2\_WPA3\_PSK: Serial.print("WPA2+WPA3"); break;

case WIFI\_AUTH\_WAPI\_PSK: Serial.print("WAPI"); break;

default: Serial.print("unknown");

}

Serial.println();

delay(10);

}

}

Serial.println("");

// Delete the scan result to free memory for code below.

WiFi.scanDelete();

// Wait a bit before scanning again.

delay(5000);

}

**Output: *n Networks Found! ; Light's Space Stone ; Airtel77***

**Conclusion:** Through this experiment we understood how to work with ESP32 and blink the internal LED ,Configuring ESPressif to scan wireless signals.