

ESP32 DAY5

Bluetooth & WiFi

Let's use the ESP32's Bluetooth capability

- We want to control ESP32's LED via Bluetooth.
- What is Bluetooth?

Interfaces

- Digital Output (DO)
- Digital Input (DI)
- ADC
- ~~DAC~~
- [I²C: DS3231 / AT24C32](#)
- SPI: MAX7219, ST7789

ESP32 IOT WORKSHOP

Control ESP32 LEDs via Bluetooth BLE

An essential guide to Bluetooth Low Energy fundamentals, technical specifications, GATT architecture, and hardware control.

1. Classic vs. BLE Specifications

Comparing raw throughput, latency, power budgets, and channel configurations

2. Deep-Dive Specifications

Detailed breakdown of PHY rates, payload MTU, power states, and range limits

Classic Bluetooth vs. BLE Specifications

Specification	Classic Bluetooth (BR/EDR)	Bluetooth Low Energy (BLE)
PHY Data Speed	1 Mbps (BR) / 2–3 Mbps (EDR)	1 Mbps (LE 1M) / 2 Mbps (LE 2M) / 125–500 kbps (Coded)
Practical Throughput	~1.5 – 2.1 Mbps	~100 – 300 kbps (ESP32 typical) / ~1.4 Mbps max
Power Consumption	High (~1 W, active streaming)	Ultra-Low (0.01 W – 0.1 W, burst cycles)
Connection Latency	~100 ms setup time	< 6 ms fast connection setup
RF Channels	79 channels (1 MHz bandwidth)	40 channels (2 MHz bandwidth: 3 Adv + 37 Data)
Target Workload	Continuous audio & file streams	Sensor updates, state toggles & control payloads

Detailed BLE Technical Specifications

Parameter	BLE 4.2 / 5.0 Standard	ESP32 Hardware Capability
Physical Layer (PHY)	LE 1M, LE 2M, LE Coded (Long Range)	Bluetooth 4.2 BR/EDR & BLE (LE 1M PHY)
Frequency & Spectrum	2.4000 GHz – 2.4835 GHz ISM Band	2.4 GHz Adaptive Frequency Hopping (AFH)
Max ATT Payload (MTU)	23 bytes default (Up to 512 bytes)	Configurable MTU up to 512 bytes via code
Transmit Power (Tx)	-20 dBm to +20 dBm class limit	-12 dBm to +9 dBm (Class 1, 1.5, 2 compliant)
Power Current Draw	Peak < 15 mA (optimized chips)	~130 mA Tx peak / ~95 mA Rx / <10 µA Deep Sleep
Effective Range	10m – 100m+ (Line of Sight)	~15–30m indoor / up to 50m clear line-of-sight

Core Architecture: GAP and GATT



1. GAP Advertising

Dictates connection discovery. Broadcasts packet headers on Channels 37, 38, 39 every 20ms to 10.24s interval.



2. GATT Server

ESP32 maintains a lookup table storing Services, Characteristics, and Descriptors accessible by central devices.



3. GATT Client

Smartphone app (e.g. LightBlue / nRF Connect) initiates connection and sends READ / WRITE / NOTIFY requests.

Services, Characteristics & UUIDs

Structured Data Hierarchy

Services: Containers holding functionality (e.g., "LED Control Service"). Identified by 128-bit UUIDs.

Characteristics: Specific data attributes with properties (READ, WRITE, NOTIFY) containing state bytes.

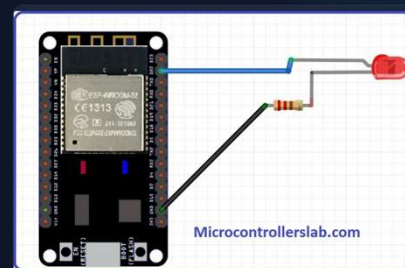
128-bit UUIDs: `4fa1c201-1fb5-459e-8fcc-c5c9c331914b` to distinguish custom hardware endpoints.



GATT Server & ESP32 LED Model

To control an LED attached to an ESP32 GPIO pin:

1. The ESP32 initializes a GATT Server and advertises its name (e.g., `ESP32_LED_Controller`).
2. A smartphone connects as a GATT Client via 2.4GHz RF connection.
3. Writing a single byte payload sets `0x01` (LED ON), while sending sets GPIO LOW (LED `0x00`).
4. Transmission time for a 1-byte write is over BLE `2 ms`.



Key Performance Metrics

2.0

Mbps Max PHY Speed (BLE 5.0)

Optimized Low-Latency Hardware

Latency: Instant response time (< 10 ms) when toggling GPIO pins via smartphone apps.

Connection Intervals: Programmable from 7.5 ms up to 4.0 s to balance latency against battery life.

Burst Payload Efficiency: Transmits small state updates efficiently without maintaining energy-intensive continuous links.

Step-by-Step Lab Workflow

-  **Step 1: Wire Circuit** — Connect LED with 220Ω resistor to ESP32 GPIO 2 and GND.
-  **Step 2: Flash Sketch** — Configure Service UUID, Characteristic UUID, and attach Write Callbacks.
-  **Step 3: Start Advertising** — Set advertising parameters so central clients detect the device.
-  **Step 4: Connect & Command** — Use LightBlue / nRF Connect app to write / to toggle LED 0x01 0x00

```
#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>

// 定義自訂 UUID
#define SERVICE_UUID          "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID   "beb5483e-36e1-4688-b7f5-ea07361b26a8"

// PWM 參數設定
#define PWM_FREQ 5000        // 頻率 5 kHz
#define PWM_RESOLUTION 8     // 解析度 8-bit (值範圍: 0 ~ 255)

class BrightnessCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
        String value = pCharacteristic->getValue();

        if (value.length() > 0) {
            auto brightness = (uint8_t)constrain(value.toInt(), 0, 255);
            ledcWrite(LED_BUILTIN, brightness);
            Serial.printf("LED 亮度更新為: %d / 255\n", brightness);
        }
    }
};
```

Arduino Code

Ready to Build?

Let's flash our first BLE sketch to the ESP32!

ESP32 Hands-on Lab | Bluetooth Low Energy Essentials & Specifications

```
void setup() {
    Serial.begin(115200);
    ledcAttach(LED_BUILTIN, PWM_FREQ, PWM_RESOLUTION);
    ledcWrite(LED_BUILTIN, 0);
    BLEDevice::init("ESP32-LED-Dimmer");
    BLEServer *pServer = BLEDevice::createServer();
    BLEService *pService = pServer->createService(SERVICE_UUID);
    BLECharacteristic *pCharacteristic = pService->createCharacteristic(
        CHARACTERISTIC_UUID,
        BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE
    );
    pCharacteristic->setCallbacks(new BrightnessCallbacks());
    pCharacteristic->setValue("0");
    pService->start();
    BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
    pAdvertising->addServiceUUID(SERVICE_UUID);
    pAdvertising->setScanResponse(true);
    BLEDevice::startAdvertising();
    Serial.println("BLE GATT 伺服器已啟動，等待連線...");
}

void loop() {
    // BLE 任務在背景由 FreeRTOS 處理，主迴圈保持輕量即可
    delay(1000);
}
```

Arduino Code

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>ESP32 BLE</title>
</head>
<body>
  <h2>ESP32 BLE Control</h2>
  <button id="connectBtn">Connect</button>
  <button id="disconnectBtn" disabled>Disconnect</button>
  <br><br>
  <input type="range" id="ledSlider" min="0" max="255" value="0">

  <div id="status">Status: Disconnected</div>
```

HTML code

```
<script>
  async function connect() {
    try {
      statusText.textContent = "Status: Scanning...";

      bleDevice = await navigator.bluetooth.requestDevice({
        filters: [{ services: [SERVICE_UUID] }]
      });

      bleDevice.addEventListener("gattserverdisconnected", onDisconnected);

      statusText.textContent = "Status: Connecting...";
      const server = await bleDevice.gatt.connect();
      const service = await server.getPrimaryService(SERVICE_UUID);
      ledCharacteristic = await service.getCharacteristic(CHARACTERISTIC_UUID);

      statusText.textContent = "Status: Connected to " + bleDevice.name;
      connectBtn.disabled = true;
      disconnectBtn.disabled = false;
      ledSlider.disabled = false;
    } catch (error) {
      statusText.textContent = "Error: " + error.message;
    }
  }
  };
```

HTML code

```
<script>
  const SERVICE_UUID = "4fafc201-1fb5-459e-8fcc-c5c9c331914b";
  const CHARACTERISTIC_UUID = "beb5483e-36e1-4688-b7f5-ea07361b26a8";

  let bleDevice = null;
  let ledCharacteristic = null;

  const connectBtn = document.getElementById("connectBtn");
  const disconnectBtn = document.getElementById("disconnectBtn");
  const ledSlider = document.getElementById("ledSlider");
  const statusText = document.getElementById("status");

  connectBtn.addEventListener("click", connect);
  disconnectBtn.addEventListener("click", () => {
    if (bleDevice && bleDevice.gatt.connected) bleDevice.gatt.disconnect();
  });
  ledSlider.addEventListener("input", () => {
    sendCommand(ledSlider.value);
  });
</script>
```

HTML code

```
function onDisconnected() {
  statusText.textContent = "Status: Disconnected";
  connectBtn.disabled = false;
  disconnectBtn.disabled = true;
  ledSlider.disabled = true;
  ledCharacteristic = null;
}
  async function sendCommand(value) {
    if (!ledCharacteristic) return;
    try {
      const encoder = new TextEncoder();
      await ledCharacteristic.writeValueWithResponse(encoder.encode(value));
    } catch (error) {
      statusText.textContent = "Write error: " + error.message;
    }
  }
</script>
</body>
</html>
```

HTML code

WiFi Station / SoftAP Mode

We want Laptop or Cellphone to connect to the ESP32 via WiFi, and our default browser will be opened automatically to show the webpage to control the LED (or buzzer)



Core Hardware & Architecture

2.4 GHz RF Transceiver, Antenna Interfaces, Data Rates, and Operating Modes

ESP32 Wi-Fi Capability

Hardware Architecture, Operating Modes, Power Consumption, and Comprehensive API
Categorization Reference

Wi-Fi Hardware Specs

- **Standard Support:** IEEE 802.11 b/g/n physical layer compliance.
- **Frequency Range:** Integrated 2.4 GHz RF transceiver module.
- **Data Rates:** Up to 150 Mbps with 802.11n HT40 channel bandwidth.
- **TX Output Power:** Up to +20 dBm max RF power output.
- **Antenna Diversity:** On-board PCB antenna or external U.FL/IPEX.



Wi-Fi Operating Modes



Station Mode (STA)

The ESP32 connects as a wireless client to an Access Point (Router). Essential for cloud logging, MQTT telemetry, HTTP API queries, and web services.



Access Point Mode (SoftAP)

The ESP32 acts as a hotspot allowing laptops or phones to connect directly. Crucial for device setup provisioning, captive portals, and local web configuration.

ESP-NOW Protocol

< 5 ms

Ultra-Low Transmission Latency

Connectionless Direct Peer Communication

ESP-NOW is Espressif's proprietary connectionless protocol that bypasses the traditional Wi-Fi connection handshake process. Packets are sent directly between MAC addresses using 250-byte vendor action frames.

It enables fast peer-to-peer telemetry and battery-operated remote control nodes without requiring a central router.

Security & Encryption Stack



WPA2 / WPA3 Support

Supports modern security protocols including WPA2-Personal, WPA3-Personal, and WPA2-Enterprise (802.1X).



Hardware Crypto Engine

Hardware accelerators offload AES-128/256, SHA-2, RSA, and ECC operations directly from dual CPU cores.



TLS / SSL Layer

Integrated mbedTLS library facilitates secure HTTPS, MQTTS, and Secure WebSockets end-to-end encryption.



Wi-Fi API Categorization

A Structured Breakdown of ESP-IDF / Arduino APIs Grouped by Usage Frequency

常用方法（基本連線、AP 熱點、狀態與 IP 查詢）

方法 (Method)	說明
WiFi.mode(m)	設定工作模式 (WIFI_STA, WIFI_AP, WIFI_AP_STA, WIFI_OFF)
WiFi.getMode()	取得目前 Wi-Fi 模式
WiFi.begin(ssid, pass)	以 Station 模式連線至指定的 AP/路由器
WiFi.disconnect([wifioff, eraseap])	中斷目前的 Wi-Fi 連線
WiFi.status()	取得連線狀態代碼（如 WL_CONNECTED, WL_DISCONNECTED）
WiFi.isConnected()	回傳是否已成功連上路由器 (bool)
WiFi.localIP()	取得 Station 模式下獲配的本機 IPv4 位址
WiFi.macAddress()	取得裝置 Station 的 MAC 位址字串
WiFi.RSSI()	取得目前連線的訊號強度 (dBm)
WiFi.SSID()	取得目前連線的基地台名稱
WiFi.softAP(ssid, [pass, ch, hidden, max_conn])	建立 ESP32 軟體熱點 (SoftAP)
WiFi.softAPIP()	取得 ESP32 熱點自身的 IPv4 位址（預設 192.168.4.1）
WiFi.softAPdisconnect([wifioff])	關閉並停止目前的 SoftAP 熱點

很少用方法

- 進階網路與 IPv6
 - WiFi.enableIPv6() / WiFi.localIPv6()：啟用 IPv6 堆疊及查詢本機 IPv6 位址。
 - WiFi.BSSID() / WiFi.BSSIDstr()：取得當前連線 AP 的 MAC 位址。WiFi.channel()：取得目前通訊使用的 Wi-Fi 頻道編號。
 - WiFi.networkID() / WiFi.broadcastIP()：計算並取得當前網段的網路 ID 與廣播位址。
- 硬體功率與射頻參數
 - WiFi.setTxPower(power) / WiFi.getTxPower()：調整或查詢 RF 發射功率（節能或過熱微調）。
 - WiFi.setDualAntennaConfig(...)：設定雙天線切換模式（需硬體支援 RF 切換器）。
- 企業安全與 WPS 連線
 - WiFi.beginEnterprise(...)：連線至 WPA2/WPA3-Enterprise 企業級認證網路（搭配 802.1x 憑證/帳密）。
 - WiFi.beginWPSConfig()：啟動 WPS (Wi-Fi Protected Setup) 按鈕配網協定。
- 智慧配網 (SmartConfig)
 - WiFi.beginSmartConfig()：啟動 ESP-Touch / SmartConfig 配網監聽模式。
 - WiFi.stopSmartConfig()：停止 SmartConfig 監聽。
 - WiFi.smartConfigDone()：查詢 SmartConfig 是否已取得封包並配置完成。
- 底層操作與 AP 客戶端管理
 - WiFi.softAPgetStationList(list)：讀取已連線客戶端的詳細結構清單（包含個別 MAC 與 RSSI）。
 - WiFi.softAPsetHostname(name)：設定 SoftAP 模式下的主機名稱。
 - WiFi.softAPenableIPv6() / WiFi.softAPIPV6()：啟用並查詢 SoftAP 介面的 IPv6 位址。
 - WiFi.useStaticBuffers(flag)：強制使用靜態記憶體緩衝區以抑制動態記憶體碎片化。
 - WiFi.persistent(bool)：設定是否將 Wi-Fi 認證資訊寫入 NVS 快閃記憶體（設為 false 可延長 Flash 壽命）。

偶爾用方法

方法 (Method)	說明
WiFi.config(ip, gateway, subnet, [dns1, dns2])	設定 Station 固定靜態 IP、閘道與 DNS
WiFi.softAPConfig(ip, gateway, subnet)	設定 SoftAP 的網段與 IP 位址
WiFi.scanNetworks([async, show_hidden, passive, max_ms_per_ch])	掃描周圍 Wi-Fi 熱點（回傳找到的網路數量）
WiFi.scanComplete()	取得非同步掃描狀態或結果數量
WiFi.scanDelete()	清除記憶體中的掃描結果快取
WiFi.SSID(i) / WiFi.RSSI(i) / WiFi.encryptionType(i)	取得掃描清單中索引值 i 的 SSID、訊號與加密型態
WiFi.BSSID(i) / WiFi.channel(i)	取得掃描結果中索引值 i 的 BSSID 與運作頻道
WiFi.onEvent(cb, [event])	註冊 Wi-Fi 事件回呼函式（如 IP 獲取、斷線重連監聽）
WiFi.removeEvent(id)	移除指定的事件監聽監聽器
WiFi.gatewayIP() / WiFi.subnetMask()	取得本機所屬網段的預設閘道與子網路遮罩
WiFi.dnsIP([n])	取得第 n 組 DNS 伺服器位址
WiFi.setHostname(name) / WiFi.getHostname()	設定/取得 DHCP Client 的主機名稱
WiFi.reconnect()	強制以先前設定重新連線
WiFi.setAutoReconnect(auto)	設定斷線後底層是否自動嘗試重連
WiFi.softAPgetStationNum()	取得目前連線至本機 SoftAP 的客戶端數量
WiFi.softAPmacAddress()	取得 ESP32 SoftAP 的硬體 MAC 位址
WiFi.setSleep(mode) / WiFi.getSleep()	設定/查詢 Wi-Fi 節電模式 (WIFI_PS_NONE, WIFI_PS_MIN_MODEM 等)



Transmission Power & Energy

Configurable RF Output Power Levels, Associated Current Consumption, and Sleep Modes

TX Power & Current Consumption

Setting Constant / dBm	Parameter Value	RF Power (mW)	Peak Current (mA)	Recommended Target Use Case
WIFI_POWER_19_5dBm (19.5 dBm)	78 (or 80 = 20 dBm)	~90 – 100 mW	220 – 240 mA	Maximum range through heavy walls & long distance.
WIFI_POWER_17dBm (17.0 dBm)	68	~50 mW	180 – 200 mA	Standard default high-performance Wi-Fi coverage.
WIFI_POWER_15dBm (15.0 dBm)	60	~32 mW	150 – 165 mA	Balanced residential Wi-Fi with reduced thermal output.
WIFI_POWER_13dBm (13.0 dBm)	52	~20 mW	130 – 145 mA	Short-to-medium range indoor sensor nodes.
WIFI_POWER_11dBm (11.0 dBm)	44	~12.5 mW	115 – 125 mA	Energy-conscious connected IoT desktop peripherals.
WIFI_POWER_8_5dBm (8.5 dBm)	34	~7 mW	100 – 110 mA	In-room battery sensors or desk-top controllers.
WIFI_POWER_MINUS_1dBm (-1.0 dBm)	-4 (or 8 = 2 dBm)	~0.8 mW	80 – 85 mA	Ultra-short proximity links, test fixtures & low heat.

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

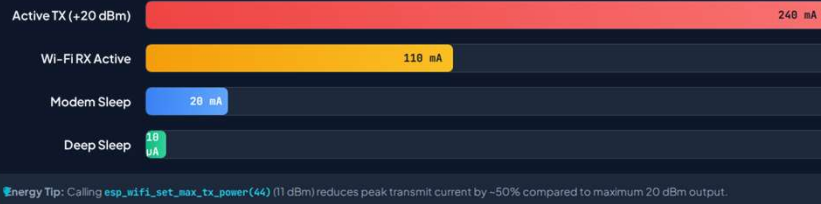
const char *ssid = "ESP32_LED_Controller";
const char *password = "12345678"; // 至少 8 碼

const int PWM_FREQ = 5000;
const int PWM_RESOLUTION = 8;

const byte DNS_PORT = 53;
IPAddress apiP(192, 168, 4, 1);
DNSServer dnsServer;
WebServer server(80);

int currentBrightness = 0;
```

Wi-Fi Power Saving Modes



```
// 使用 C++ Raw String 存放網頁內容，方便直接修改
const char INDEX_HTML[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ESP32 LED 亮度控制</title>
</head>
<body>
  <div class="card">
    <h2>ESP32 LED 調光器</h2>
    <div class="val-box"><span id="brightnessVal">0</span></div>
    <input type="range" min="0" max="255" value="0" class="slider" id="ledSlider"
      oninput="updateBrightness(this.value)">
    <div class="quick-btns">
      <button onclick="setQuickVal(0)">關閉 (0)</button>
      <button onclick="setQuickVal(128)">半亮 (128)</button>
      <button onclick="setQuickVal(255)">全亮 (255)</button>
    </div>
  </div>
</body>
)"
```

```

<script>
const slider = document.getElementById("ledSlider");
const valText = document.getElementById("brightnessVal");
// 初次載入獲取目前數值
fetch("/getBrightness")
  .then(res => res.text())
  .then(val => {
    slider.value = val;
    valText.innerText = val;
  });
function updateBrightness(val) {
  valText.innerText = val;
  // 發送 GET 請求更新亮度 · 不刷新頁面
  fetch("/setBrightness?value=" + val);
}
function setQuickVal(val) {
  slider.value = val;
  updateBrightness(val);
}
</script>
</body>
</html>
)rawliteral";

```

```

void setup() {
  Serial.begin(115200);
  ledcAttach(LED_BUILTIN, PWM_FREQ, PWM_RESOLUTION);
  setPwmDuty(0);
  WiFi.mode(WIFI_AP);
  WiFi.softAPConfig(apIP, apIP, IPAddress(255, 255, 255, 0));
  WiFi.softAP(ssid, password);
  Serial.print("AP IP: ");
  Serial.println(WiFi.softAPIP());
  dnsServer.start(DNS_PORT, "*", apIP);
  server.on("/", HTTP_GET, handleRoot);
  server.on("/setBrightness", HTTP_GET, handleSetBrightness);
  server.on("/getBrightness", HTTP_GET, handleGetBrightness);
  server.on("/generate_204", HTTP_GET, handleRoot); // Android
  server.on("/hotspot-detect.html", HTTP_GET, handleRoot); // Apple iOS / macOS
  server.on("/canonical.html", HTTP_GET, handleRoot);
  server.onNotFound(handleNotFound);
  server.begin();
  Serial.println("Web Server 與 Captive Portal 已就緒");
}
void loop() {
  dnsServer.processNextRequest();
  server.handleClient();
}

```

```

void setPwmDuty(int duty) {
  currentBrightness = constrain(duty, 0, 255);
  ledcWrite(LED_BUILTIN, currentBrightness);
}
void handleRoot() {
  server.send(200, "text/html", INDEX_HTML);
}
void handleSetBrightness() {
  if (server.hasArg("value")) {
    setPwmDuty(server.arg("value").toInt());
    server.send(200, "text/plain", "OK");
  } else {
    server.send(400, "text/plain", "Bad Request");
  }
}
void handleGetBrightness() {
  server.send(200, "text/plain", String(currentBrightness));
}
void handleNotFound() {
  server.sendHeader("Location", String("http://") + apIP.toString() + "/");
  server.send(302, "text/plain", "");
}

```

Homework

- Hopefully with the help from AI tools, you will be able to use ST7789 to display something on 1.69" TFT color screen. (recall yesterday's homework on the next page)
- Today's homework is to use WiFi to connect to your ESP32 and display a webpage that has a canvas to draw something with your mouse, and that something will appear on your 1.69" TFT screen (note it is a colored display, so you should be able to draw colorful things and display colorful things on your TFT screen.)

Homework – use you ST7789 display module to:

- Write a function `dot(int x, int y, uint16_t color)` to draw a pixel.
- Write a function `fill(int x1, int y1, int x2, int y2, uint16_t color)` to draw a solid block.
- Write a function `line(int x1, int y1, int x2, int y2, uint16_t color)` to draw a line
- Write a function `circle(int x, int y, uint16_t color)` to draw a circle.
- Write a function `inverse(bool onoff)` to display normal or inverse display.
- Write functions `flipHorizontally(onoff)` and `flipVertically(onoff)` to flip the display horizontally and vertically (using hardware commands, not rearrange pixels in your code!).
- // Write a function `text(int x, int y, string text, uint16_t color);` to draw a string